VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE KVÁDRŮ-KRABIC V OBRAZE

DETECTION OF BOXES IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATEJ SOROKA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Soroka Matej**
Program: Informační technologie
Název: **Detekce kvádrů-krabic v obraze**
Detection of Boxes in Image
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou počítačového vidění na bázi strojového učení a konvolučních neuronových sítí.
2. Pořiďte (a průběžně zvětšujte a zkvalitňujte) datovou sadu pro vývoj systému pro detekci krabic (v širokém smyslu) v obraze.
3. Experimentujte s vhodnými algoritmy strojového učení pro detekci krabic v obraze. Pro učení a vyhodnocování použijte shromážděnou datovou sadu.
4. Navrhněte a implementujte demonstrační aplikaci využívající detekce krabic v obraze.
5. Iterativně vyhodnocujte a vylepšujte vytvořené řešení.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Pro udělení zápočtu za první semestr je požadováno:

- body 1.-2., značné rozpracování bodů 3. a 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cieľom tejto práce je experimentovať a vyhodnotis rôznymi prístupmi počítačového videnia s cieľom automatickej detekcie krabíc-kvádrov v obraze, za týmto cieľom boli v riešení použité prístupy založené na neurónových sieťach. V práci boli prevedené experimenty s klasifikáciou pomocou vlastnej dátovej sady, klasifikáciou s využitím vlastnej konvolučnej neurónovej siete, detekciou pomocou pouvného okna, detektora YOLO a v poslednej časti návrh vylepšenia s použitím sietí U-net a MirrorNet.

Abstract

The aim of this work is to experiment and evaluate different approaches of computer vision with the aim of automatic detection of boxes-blocks in the image, for this purpose, approaches based on neural networks were used in the solution. Experiments were performed with classification using our own data set, classification using our own convolutional neural network, detection using a window, YOLO detector and in the last part a proposal for improvement using U-net and MirrorNet networks.

Kľúčové slová

Tensorflow, Detekcia, Segmentácia, Keras, YOLO, U-Net, MirrorNet

Keywords

Tensorflow, Detection, Segmentation, Keras, YOLO, U-Net, MirrorNet

Citácia

SOROKA, Matej. *Detekce kvádrů-krabic v obraze*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Detekce kvádrů-krabic v obraze

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof.Ing. Adama Herouta, Ph.D., uviedol som všetky literárne prameňe, publikácie a ďalšie zdroje z ktorých som čerpal.

.....
Matej Soroka
10. mája 2021

Poďakovanie

Rád by som poďakoval svojmu vedúcemu tejto práce, pánovi prof. Ing. Adamovi Heroutovi, Ph.D., za jeho odborné vedenie, rady, prístup, podporu a trpezlivosť počas tvorby tejto práce. Taktiež by som chcel poďakovať svojej rodine, priateľke Maťke, blízkym priateľom a kolegom Adamovi a Molitannovi za vzájomnú podporu pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je vytvoriť detektor papierových krabíc pomocou umelej inteligencie založenej na počítačovom videní. Takýto detektor má veľké využitie najmä v logistike, kde sa papierové krabice používajú najviac.

Prerekvizity pre vytvorenie detektora krabíc začínajú zostrojením dátovej sady, keďže dátová sada musí čo najviac reprezentovať použitie, nie je dostupné využiť obrazové dáta napríklad z priemyselných kamier, preto je na mieste vytvoriť vlastnú dátovú sadu papierových krabíc z pohľadu podobného priemyselnej kamery, teda zvrchu. Po zostrojení dátovej sady je potrebné preskúmať možnosti a prístupy detekcie vlastného objektu, experimentovať s algoritmami počítačového videnia, vyhodnotiť výsledky prístupov a najmä poznávať problematiku umelej inteligencie a strojového učenia pre detekciu krabíc v obraze a iteratívne sa blížiť k výsledku a navrhnúť čo najefektívnejšie riešenie využívajúce technológie poslednej doby.

Kapitola 2

Problematika detekcie a segmentácie obrázkov

V tejto kapitole je súhrn teoretických znalostí k problematike, ktorá nás vedie k minimálnemu porozumeniu techník a princípov k tomu, aby sme vedeli čo potrebujeme k detekcií objektov ovládať.

2.1 Prístupy pri detekcii objektov v obraze

Pri detekcií objektov v obraze môžeme využiť rôzne techniky, všetko záleží od zložitosti objektu, ktorý potrebujeme detekovať a klasifikovať. Pre jednoduché útvary, napríklad úsečky, môžeme využiť Houghovu transformáciu. No je veľmi nepravdepodobné, že dekecia, ktorú potrebuejeme vykonávať je tak jednoduchá, že si vystačíme len s detekciou kriviek. Preto sa pri zložitejších objektoch využíva strojové učenie. Pri detekcií objektov sa najčastejšie používajú konvolučné neuronové siete.

Posledné roky sme boli svedkami významného pokroku v detekcii objektov pomocou hlbokých konvolučných sietí. Najmodernejšie metódy detekcie objektov sa väčšinou riadia paradigmou založenou na regióne. Vzhľadom na riedku množinu návrhov oblastí sa klasifikácia objektov a regresia ohraničovacieho poľa vykonáva pre každý návrh jednotlivo.

Na odstránenie duplicitných detekcií sa potom použije heuristický krok ručného spracovania, maximálne potlačenie (NMS). V komunite vízií je už roky dobre známe, že kontextové informácie alebo vzťah medzi objektmi pomáhajú rozpoznávaniu objektov. Počas éry hlbokého učenia nedochádza k významnému pokroku vo využívaní objektových vzťahov na detekčné učenie.

Väčšina metód sa stále zameriava na rozpoznávanie objektov. Jedným z dôvodov je, že vzťah medzi objektmi je ťažké modelovať. Objekty sú na ľubovoľných pozíciách obrázkov, v rôznych mierkach, v rôznych kategóriách a ich počet sa môže na rôznych obrázkoch líšiť. Moderné metódy založené na konvolučných neurónových sieťach majú väčšinou jednoduchú pravidelnú sieťovú štruktúru. Nie je jasné, ako sa vyrovnať s nezrovnalosťami v existujúcich metódach.

2.2 Segmentácia, klasifikácia a detekcia

Segmentácia obrazu je úlohou vyhľadávania skupiny pixelov, ktoré patria k sebe. V štatistikách je tento problém známy ako clustrová analýza a je to široko študovaná oblasť so

stovkami rôznych algoritmov. V počítačovom videní segmentácia obrazu je jedným z najstarších a najštudovanejších problémov. Skoré techniky zvyknú používať štiepenie alebo zlučovanie regiónov, ktoré zodpovedajú súčasným a aglomerátivným algoritmom v clustrovacej literatúre. Novšie algoritmy často optimalizujú niektoré globálne kritériá, ako sú napríklad konzistencia v rámci regiónu a dĺžka medziregionálnych hraníc alebo odlišnosť. [17]

Klasifikácia je proces zaradenia objektu do kategórie, ktorá nám je známa, lebo sme ju už videli a máme o nej dostatočné množstvo informácií, teda poznáme ju.

Detekcia, je proces kedy hľadáme a nájdeme v zornom poli obraz, napríklad pomocou segmentácie a klasifikujeme jeho kategóriu a zároveň určíme aj jeho relatívnu polohu v priestore.

2.2.1 Sémantická segmentácia

Dnešné segmentačné modely sú veľmi presné, keď sú trénované na veľkých správne anotovaných dátových sadách, no problém v segmentačných modeloch, je náročnosť anotácií, kedy musíme pracne vyznačovať v obraze n-uholníkové výrezy a nie len rozdeľovať objekty, ale ich aj klasifikovať. [4]

Obr. 2.1: Ukážka segmentácie záberu z ulice, prevzaté z [4]

2.3 Umelá inteligencia

V dnešnej dobe, odvetvie umelej inteligencie je nedeliteľnou súčasťou moderného života. Inteligencia je slovo odvodené z latinského slova intelligentia, ktoré znamená schopnosť rozlišovať, poznávať a chápať. Inteligencia bola dlhú dobu prisudzovaná len ľuďom, no v poslednej dobe je študovaná u zvierat a dokonca aj rastlín. Pojem umelá inteligencia je označovaná inteligencia, neživých subjektov, teda výpočetných systémov a príslušných programov. Klasická umelá inteligencia spočíva v riešení úloh, učení, klasifikácií a rozpoznávaní ktoré sa aplikujú v rozličných oblastiach, napríklad v spracovaní prirodzeného jazyka, alebo v počítačovom videní, ktoré je jedno z mnoha využití umelej inteligencie teda implementácie zraku spojeného so spoznávaním objektov, ktorému som sa venoval v tejto práci.

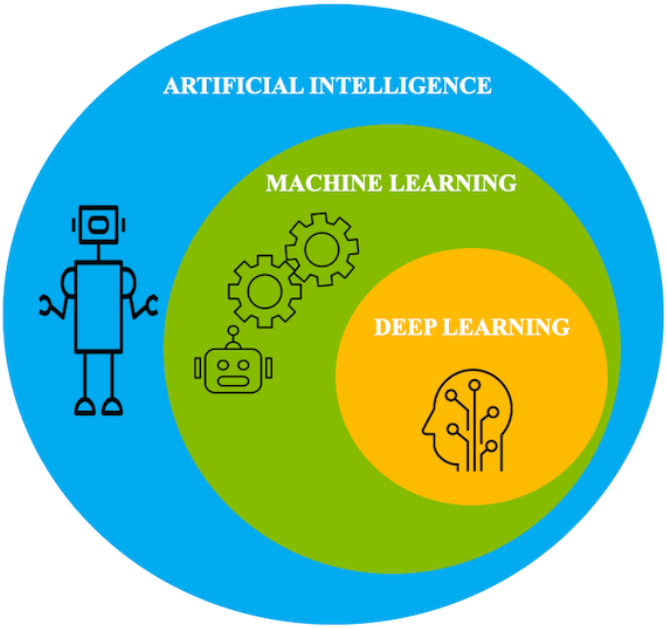
2.3.1 Strojové učenie

Strojové učenie môžeme chápať ako podskupinu umelej inteligencie, ktorá sa zameriava špecificky na počítačové systémy, ktoré majú schopnosť učiť sa. Stroj získava poznatky na základe dát, s ktorými pracuje. V súčasnosti je strojové učenie zapojené do veľkého kolobehu každodenných situácií, napríklad personalizovaná reklama na sociálnych sieťach, ktorá využíva dáta z nášho používania obľúbených kategórií, alebo nám prehrávač hudby doporučuje hudbu na základe toho, akú hudbu počúvame. Strojové učenie vytvára pravidla na základe dát, takže v prípade dát zo sociálnych sietí, ak si prezeráme a odoberáme rôzne zdroje, povedzme o kvetoch, marketingový model to vyhodnotí a pri výbere reklamy zostaví sadu pravidiel tak, aby výstupom modelu boli reklamy, ktoré sa zaoberajú konkrétnou kategóriou, teda o kvetoch. Proces učenia sa pri strojovom učení funguje tak, že stroju určíme vstupné a výstupné dáta, stroj na základe dát trénuje pravidlá, aby keď na vstup privedieme objekt, zo vstupných dát, tak aby výstupom stroja, presnejšie modelu bol výstup prislúchajúci k vstupu.

2.3.2 Hlboké učenie

Hlboké učenie umožňuje automatické učenie viacerých úrovní reprezentácií podkladovej distribúcie údajov, ktoré sa majú modelovať. Algoritmy hlbokého učenia preukázali vynikajúci výkon pri učení a klasifikácii v oblastiach, ako je napríklad prenosové učenie, reč a rozpoznávanie rukou písaných znakov. Hlboké učenie je o automatickom učení viacerých úrovní reprezentácií podkladovej distribúcie údajov, ktoré sa majú modelovať [11]. Inými slovami, algoritmus hlbokého učenia automaticky extrahuje funkcie na nízkej a vysokej úrovni potrebné na klasifikáciu.

Pod pojmami vysoká úroveň sa rozumie funkcia, ktorá hierarchicky závisí od ďalších funkcií. Napríklad v súvislosti s počítačovým videním to znamená, že algoritmus hlbokého učenia sa naučí svoje vlastné nízkoúrovňové reprezentácie zo surového obrazu (napríklad detektor hrany, Gaborove filtre atď.), Potom vytvorí reprezentácie, ktoré závisia od týchto reprezentácie na nízkej úrovni (napríklad lineárne alebo nelineárne kombinácie týchto reprezentácií na nízkej úrovni) a postupne opakujte rovnaký postup pre vyššie úrovne.



Obr. 2.2: Množinový diagram štruktúry výrazov umelej inteligencie. Prevzaté z IBM Developer knowledgebase [1]

2.3.3 Neurónové siete

Neurónové siete sú inšpirované poznatkami o neurónoch a nervových sieťach živých organizmov. Ich hlavný význam je v schopnosti učiť sa, zovšeobecňovať a riešiť nelineárne úlohy. Ich využitie je pre klasifikáciu a regresiu.

2.3.4 Biologický neurón

Biologický neurón sa skladá z tela, výbežku axónu a niekoľkých vlásočnicových výbežkov dendritov. Dendrity sú spojené so synapsami axónom, ktorý slúži na prenos signálu zneurónu. Dendrity slúžia na prijímanie signálov a axón na šírenie vlastného signálu do ostatných neurónov, pričom sila prenášaných signálov závisí na sile danej synapse. Na základe biologického neurónu boli vytvorené umelé modely neurónov.

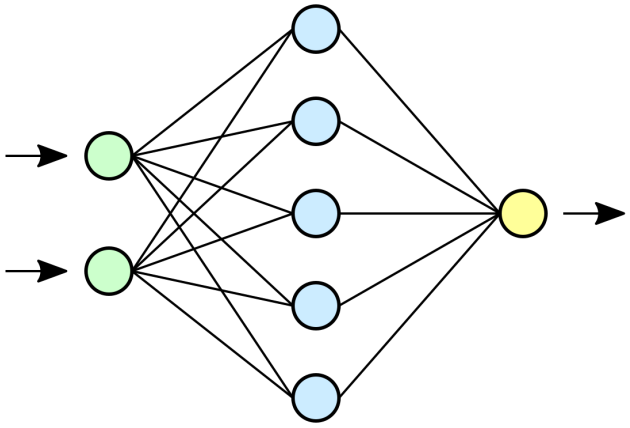
2.3.5 Umelý neurón

Umelý neurón simuluje silu synapsií numerickými váhami tak, že signál sa danou váhou násobí a to určuje jeho silu. Ideou je, že tieto váhy sú predmetom učenia a určujú vplyv neurónov medzi sebou.Váhy môžu byť rovnako ako u biologického neurónu exhcitačné (kladné) alebo inhibičné(záporné). Prijaté signály sa v základnom modeli v neuróne sčítajú a výsledok je vstupom tzv. aktivačnej funkcie, ktorá rozhodne, či neurón vyšle signál a s akou silou. Inými slovami, neurón vypočíta skalárny súčin prijatých signálov a aplikuje naň nelineárnu funkciu. Rôzne typy aktivačných funkcií sú uvedené v kapitole 2.5.

2.3.6 Štruktúra neurónovej siete

Typicky je neurónová sieť štruktúrovaná do vrstiev. Jednotlivé vrstvy môžu byť rôzneho typu, základným typom neurónovej vrstvy je takzvane plne prepojená vrstva. Každý neurón vrstvy je prepojený so všetkými neurónmi predchádzajúcej vrstvy a neuróny v jednej vrstvenie sú navzájom prepojené. Rozlišujeme medzi jednovrstvovými neurónovými sieťami a viacvrstvovými.

Vrstva neurónov, ktorá nie je pripojená priamo na vstup a jej výstup nie je výstupom siete sa nazýva skrytá vrstva. Neurónové siete s aspoň jednou skrytou vrstvou sa nazývajú hlboké neurónové siete. Jednovrstvová architektúra obsahuje vstupnú vrstvu a jednu vrstvu neurónov, ktorá je súčasne aj výstupnou vrstvou. Viacvrstvové siete obsahujú minimálne jednu vrstvu, ktorá nie je ani vstupnou a ani vrstvou, výstupy sú výstupmi celej siete.



Obr. 2.3: Ukážka jednovrstvovej neurónovej siete, kedy vrstva so zelenými blokmi reprezentuje vstupnú vrstvu, vrstva s modrými blokmi reprezentuje skrytú vrstvu neurónovej siete a posledný žltý blok reprezentuje výstupnú vrstvu neurónovej siete. Sieť má teda dva vstupy, ktorým podľa vstupu náleží jeden výstup.

2.4 Konvolučné neurónové siete

Rozdiel medzi konvolučnými neurónovými sieťami a neurónovými sieťami je ten, že konvolučné neurónové siete obsahujú aspoň jednu konvolučnú vrstvu. Hlavné využitie konvolučných neurónových sietí v dnešnej dobe je spracovanie obrazu a spracovanie reči.

Jednoduchá konvolučná sieť je sekvencia vrstiev kde každá vrstva konvolučnej siete transformuje hodnotu aktivácií do nasledujúcich aktivácií pomocou diferencovateľných funkcií. Pre budovanie konvolučných sietí používame tri typy vrstiev, konvolučnú vrstvu, pooling vrstvu a plne prepojenú konvolučnú vrstvu.

2.4.1 Konvolúcia

Konvolúcia je lineárna matematická operácia definovaná nasledujúcou rovnicou

$$x(t) \circledast h(t) = y(t)$$

Znak '*' označuje operáciu konvolúcie. Vstupom konvolúcie je typicky viacrozmerné pole hodnôt, ktoré sa nazýva tenzor. Jadro konvolúcie má zvyčajne rovnakú dimenziu ako je dimenzia vstupu, teda pri spracovaní obrazu to typicky bývajú dve alebo tri dimenzie (čiernobiely alebo RGB obrázok). Výsledok sa ozačuje ako aktivačná mapa (activation map alebo feature map). Mnohé knižnice pre neuronové siete však implementujú operáciu konvolúcie ako vzájomnú koreláciu (cross-correlation), t.j. jadro konvolúcie sa pri násobení neotáča (nerobí sa operácia kernel flipping). Táto varianta je možná, pretože to nemá vplyv na algoritmus učenia siete.

2.4.2 Konvolučná vrstva

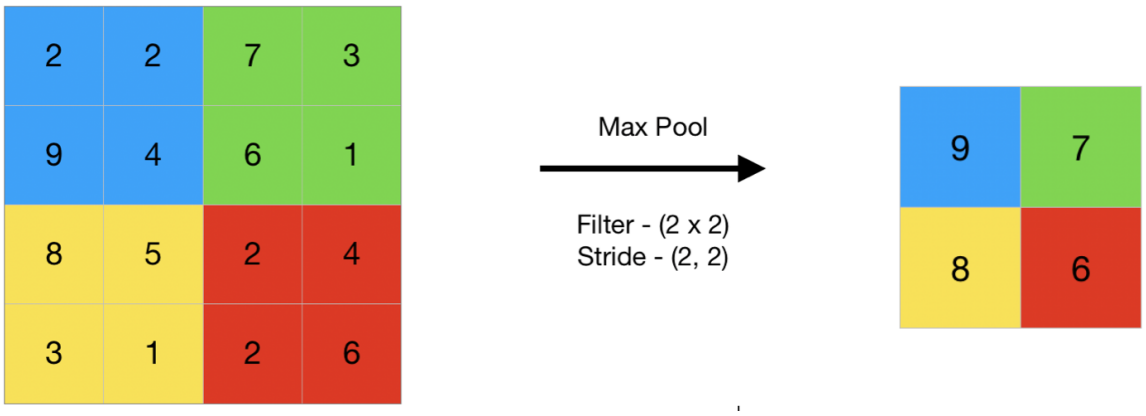
Konvolučná vrstva je základným stavebným kameňom konvolučných sietí, ktorá spracováva najviac počítačovej práce. Parametre konvolučnej vrstvy pozostávajú z filtrov, ktoré majú výhodu v tom, že ich môžeme učiť. Každý filter je priestorovo maličký, no rozširuje sa na celú hĺbku vstupného objemu. Napríklad, typický filter na prvej vrstve konvolučnej siete môže mať veľkosť 5x5x3.

2.4.3 Plne prepojená konvolučná vrstva

Plne prepojená vrstva je podobná spôsobu, akým sú neuróny usporiadané v tradičnej neurónovej sieti. Preto je každý uzol v úplne prepojenej vrstve priamo spojený s každým uzlom v predchádzajúcej aj v nasledujúcej vrstve. [3] Hlavnou nevýhodou plne prepojenej vrstvy je to, že obsahuje veľa parametrov, ktoré si na trénovacích dátach vyžadujú zložité výpočty. Preto sa snažíme eliminovať počet uzlov a pripojení. Odstránené uzly a pripojenia je možné substituovať použitím techniky vypadnutia (anglicky Dropout).

2.4.4 Pooling vrstva

Je bežné pravidelne vkladať pooling vrstvu medzi konvolučné vrstvy v architektúre konvolučných sietí. Funkcia pooling vrstvy je postupne zmenšovať priestorovú veľkosť reprezentácie, aby sa znížilo množstvo parametrov a výpočtov v sieti. Pooling vrstva pracuje nezávisle na každom hĺbkovom reze vstupu a mení svoju veľkosť priestorovo pomocou operácie max. Najbežnejšou formou je zhromažďovacia vrstva s filtrami o veľkosti 2x2 aplikovaná s krokom 2 prevzorkovania každého hĺbkového rezu na vstupe o 2 pozdĺž šírky aj výšky, pričom sa vyradí 75 percent aktivácií. Každá operácia MAX by v tomto prípade brala max. Viac ako 4 čísla (malá oblasť 2x2 v nejakom hĺbkovom reze). Dimenzia hĺbky zostáva nezmenená.



Obr. 2.4: Ukážka max poolingu s filtrom 2×2 a krokom 2, 2.

2.5 Aktivačné funkcie

Táto časť zdôrazňuje rôzne typy aktivačných funkcií a ich vývoj v priebehu rokov. Výskum aktivačných funkcií v architektúrach hlbokého učenia, ktoré sa používajú v rôznych aplikáciách, boli doteraz hlavným výskumným poľom. Najnovšie výsledky výskumu sú načrtnuté nasledovne. Je potrebné kategoricky konštatovať, že tento súhrn AF nie je uvedený v chronologickom poradí, ale je usporiadaný s hlavnými funkciami a nasledujú ich vylepšenia. Celkový súhrn aktivačných funkcií je dôkladne spísaný v literatúre [14].

Aktivačné funkcie slúžia na vykonávanie rôznych výpočtov medzi skrytými vrstvami a výstupnými vrstvami ľubovoľnej danej architektúry hlbokého učenia.

2.5.1 Sigmoid

Sigmoidová aktivačná funkcia sa v niektorej literatúre niekedy označuje ako logistická funkcia alebo squashingová funkcia. Výsledky výskumu funkcie Sigmoid vytvorili tri varianty funkcie aktivácie sigmoidu, ktoré sa používajú v aplikáciách pre strojové učenie. Sigmoid je nelineárna aktivačná funkcia používaná väčšinou v dopredných neurónových sieťach. Je to obmedzená diferencovateľná funkcia definovaná pre vstupné hodnoty, s pozitívnymi deriváciami a s určitým stupňom plynulosti.

2.5.2 Hyperbolický tangens

Hyperbolický tangens je ďalším typom aktivačnej funkcie používaným v strojovom učení a má niektoré varianty používané v aplikáciách strojového učenia. Hyperbolický tangens je funkcia známa ako tanh funkcia, je plynulejšia funkcia s nulovým stredom, ktorej rozsah leží medzi -1 až 1. Funkcia sa stala preferovanou funkciou v porovnaní s sigmoidnou funkciou v tom, že poskytuje lepší tréningový výkon pre viacvrstvové neurónové siete.

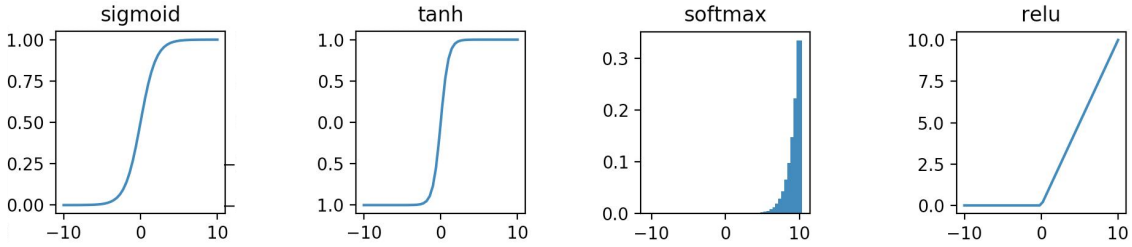
2.5.3 Softmax

Funkcia Softmax je ďalším typom aktivačnej funkcie používanej pri neurálnych výpočtoch. Používa sa na výpočet rozdelenia pravdepodobnosti z vektora reálnych čísel. Funkcia Softmax produkuje výstup, ktorý je v rozsahu hodnôt od 0 do 1, pričom súčet pravdepodobností

je rovný 1. Funkcia Softmax sa používa v modeloch viacerých tried, kde vracia pravdepodobnosti každej triedy, pričom cieľová trieda má najvyššia pravdepodobnosť. Hlavný rozdiel medzi Sigmoid a Softmax AF je v tom, že Sigmoid používa v binárnej klasifikácii, zatiaľ čo Softmax používa viacrozmerné úlohy klasifikácie.

2.5.4 ReLU

Funkciu aktivácie usmernenej lineárnej jednotky (ReLU) navrhli Nair a Hinton 2010 a od tej doby je doteraz najpoužívanejšou aktivačnou funkciou pre aplikácie hlbokého učenia s najnovšími výsledkami. ReLU je aktivačná funkcia s rýchlejším učením, ktorá sa ukázala ako najúspešnejšia a najbežnejšie používaná funkcia. Ponúka lepší výkon a generalizáciu pri hlbokom učení v porovnaní s aktivačnými funkciami Sigmoid a Tanh. ReLU predstavuje takmer lineárnu funkciu a preto zachováva vlastnosti lineárnych modelov, ktoré uľahčujú ich optimalizáciu pomocou metód gradientu a klesania.



Obr. 2.5: Prehľad priebehov aktivačných funkcií

Kapitola 3

Návrh riešenia detekcie krabíc

Cieľom tejto bakalárskej práce je experimentovať s rôznymi prístupmi pre detekciu a rozpoznávanie krabíc v obraze, táto kapitola popisuje návrh implementácie a prostriedky, pomocou ktorých je tento cieľ dosiahnutý.

Keďže okolo nás sa nachádza veľa objektov v tvare kvádra, rôzne druhy nábytku, elektronika, či samotné steny miestností, problém pri detekcií objektov ako je krabica je ten, že samotný objekt je nedostatočne výrazný, čo vedie k false-positive detekciám.Preto som sa rozhodol, že vytvorím vlastnú dátovú sadu a následne som vytvoril prvý prototyp, ktorý fungoval na dotrénovaní sieti EfficientDet, ktorú som rozšíril o nový objekt, krabicu.

Následne som experimentoval s rôznymi prístupmi, vlastný model som následne nahradil prístupom YOLO, ktorý fungoval spoľahlivo, no tento detektor bol náročný na výpočetné zdroje a viedol k false-positive deketciám kvôli nedostatočnej dátovej sade.

Následne som sa rozhodol, že krabicu rozdelím na podobjekty, kedže krabica je kváder, môžeme využiť vlastnosti geometrického objektu na to, aby sme pridali možnosť validácie objektu. Rozdelenie do podčastí zvýši počet klasifikovaných objektov, no zníži počet false-positive detekcií.

Výhoda rozdelenie spočíva aj v tom, že vieme zvýšiť presnosť detekcie pomocou pridania postprocessingu. Rozhodol som sa pracovať s touto myšlienkou, vytvoril som novú dátovú sadu obsahujúcu výrezy podčastí krabice a použil som znovu prístup YOLO, no zistil som, že YOLO na to nie je prispôsobené. Tak som hľadal nové riešenia.

Vytvoril som si vlastný konvolučný model pre klasifikáciu podobjektov krabice, čo nestačilo, keďže som potreboval pridať detekciu. Preto som sa rozhodol využiť metódu posuvného okna. Metóda na vlasnej sieti funguje spoľahlivo, ale rýchlosť detekcie nebola vôbec dostatočná.

Následne som sa oboznámil s problematikou plne konvolučných sietí a segmentácie, rozhodol som sa, že využijem prístup UNET.

3.1 Rozdelenie krabice na podobjekty

Pri prvom experimente s EfficientDet som narazil na problémy s detekciou krabíc ako jednotným objektom. Miera false-positive detekcií bola natoľko vysoká, že po konzultácií s vedúcim práce sme sa rozhodli rozdeliť krabicu na podobjekty.

Miera false positive detekcií by sa dala zredukovať mohutnosťou dátovej sady, no rozhodli sme sa pre experimentálny prístup rozdelenia, kedy som krabicu rozdelil na rôzne počty podobjektov vysvetlené v kapitole s experimentami5.2, v tejto kapitole je vysvetlené

rozdelenie, ktoré je podľa experimentov optimálne čo sa týka vizuálneho rozdelenia rohov, kedy vizuálny rozdiel medzi podobjektami je najnižší, dosiahnutie najväčšieho rozdielu medzi podobjektami nám dovoľuje vhodne využiť augmentáciu dátovej sady.

3.1.1 Vonkajšia hrana

Hrana krabice, ktorá pri pohľade pozorovateľa je rozdelená na 2 časti, prvá polovica je krabica a druhá je pozadie. Pri vyrezávaní malých obrázkov dáva zmysel vyrezať dvojnásobok výrezu otáčať aby sme zachytili hranu z každého uhla.

Obr. 3.1: Vonkajšie hrany ohraničujú celý objekt krabice.

3.1.2 Vnútorná hrana

Hrana, ktorá pri pohľade pozorovateľa obsahuje 3 časti, svetlejšiu stranu krabice, viditeľnú hranu a tmavšiu stranu krabice, ktorá je hranou predeľovaná. Pri vyrezávaní malých obrázkov dáva zmysel vyrezať dvojnásobok výrezu otáčať aby sme zachytili hranu z každého uhla.

Obr. 3.2: Vnútorné hrany sú všetky hrany okrem vonkajších hrán, ktoré spojuje vnútorný roh.

3.1.3 Vnútorný roh s tromi hranami

V prípade pohľadu zhora sa jedná o najbližší roh k pozorovateľovi, okolie rohu je krabica bez pozadia a spája tri vnútorné hrany.

Obr. 3.3: Bod reprezentujúci vnútorný roh.

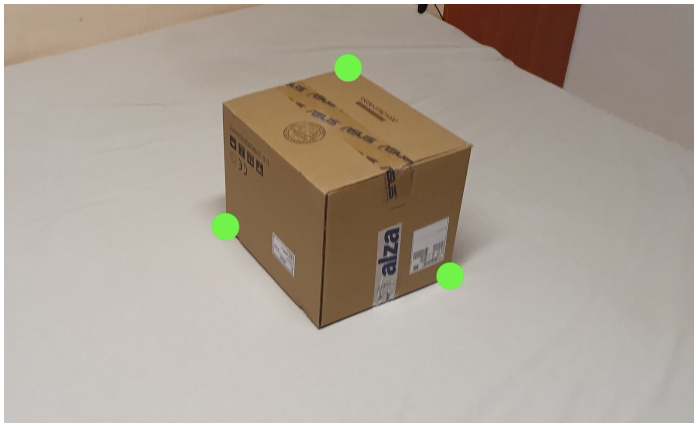
3.1.4 Vonkajší roh s tromi hranami

Roh, ktorý sa nachádza na prednej strane krabice z pohľadu pozorovateľa, do ktorého sa spájajú práve 2 vonkajšie hrany a jedna vnútorná hrana. Tieto rohy su oproti sebe po uhlopriečke čelnej strany krabice a ich spoločný sused po vnútornej hrane je vnútorný roh s tromi hranami.

Obr. 3.4: Body reprezentujúce vonkajšie rohy s troma hranami.

3.1.5 Vonkajší roh s dvoma hranami

Roh, ktorý sa pri pohľade zhora nachádza na spodnej časti krabice na opačnej strane ako vnútorný roh s tromi hranami, spájajú sa v ňom dve vonkajšie hrany.



Obr. 3.5: Body reprezentujúce vonkajšie rohy s dvoma hranami.

3.1.6 Pozadie

Pozadie je taktiež dôležité pri odlíšení krabice od pozadia. Pozadie je dôležité pre oddelenie objektu krabice od ostatných objektov v obraze preto, pretože pri detekcii, ktorá má prebiehať pre každý pixel obrazu, tak v prípade, že sme na pixeli, ktorý nepatrí podobjektu krabice, tak ho klasifikujeme ako pozadie, teda pozadie je všetko okrem podčasti krabice.

3.2 Prístupy založené na posuvnom okne

Pri probléme klasifikácie a zároveň detekcie objektu v obraze, je metóda posuvného okna. Táto metóda funguje tak, že nad obrázkom je vytvorené pomyselné okno o veľkosti menšej ako obrázok samotný. Okno sa následne posúva cez celý vstupný obraz.

Okno začína v ľavom hornom rohu vstupného obrázka. V okne, ktoré je výrezom obrázka sa prebehne klasifikácia, výsledok klasifikácia sa uloží a okno sa následne posunutie smerom doprava. Vzdialenosť posunutia okna je možné meniť, čím sa detekcia zrýchli, no zníži sa presnosť. Rovnako to platí aj pre veľkosť posuvného okna, v prípade že zvolíme nevhodnú veľkosť posuvného okna, detekcia bude nepresná, preto je dôležité zvoliť veľkosť okna tak, aby veľkosť posuvného okna bol približne tak veľký, ako veľkosť detekovaných objektov vo vstupnom obraze.

Táto metóda bola použitá už v roku 2012, kedy bola technika sliding window použitá vo výskume membrán pre elektronické mikroskopy [8]

3.3 Detektor YOLO

You Only Look Once real-time deteknčný algoritmus, ktorý je jeden z najefektivnejších detektorov objektov, ktorý prináša veľa najinovatívnejších nápadov vychádzajúcich z komunity výukumníkov počítačovného videnia.

YOLO prišlo na scénu počítačového videnia so seminárnym dokumentom Josepha Redmona z roku 2015. Okamžite si získal veľkú pozornosť kolegov z oblasti počítačového videnia.

Detekcia objektov je jeden z častých problémov počítačového videnia, kedy odpovedáme na dve otázky, čo a kde je na obrázku. Samotná klasifikácia je menej komplexná, keďže sa objekty nie len rozpoznávajú, ale aj lokalizujú v obrázku.

YOLO používa odlíšny prístup, využíva konvolučné neurónové siete pre detekciu v reálnom čase. Algoritmus aplikuje samotnú neurónovú sieť na celý obrázok a následne obrázok rozdeľuje do regiónov a predpovedá bounding boxy[1].

YOLO sa stalo populárne, lebo dosahuje vysokeú presnosť a zároveň možnosť bežať v reálnom čase. Ako názov vyplýva, obrázok je nutné

Alogirmus je populárny, pretože dosahuje vysokú presnosť a zároveň dokáže bežať v reálnom čase. Algoritmus sa na obraz „pozerá iba raz“ v tom zmysle, že na predpovedanie vyžaduje iba jeden priechod šírenia dopredu neurónovou sieťou. Po maximálnom potlačení (čím sa zabezpečí, že algoritmus detekcie objektov zistí každý objekt iba raz), potom na výstup odošle rozpoznané objekty spolu s ohraničujúcimi políčkami.

3.4 Nástroje

Pre riešenie akéhokoľvek problému pri strojovom učení, je potrebné vytvoriť model, model je nutné implementovať v programovacom jazyku, v tejto kapitole je zoznam nástrojov, ktoré som pre riešenie detekcie krabíc použil ja.

3.4.1 Python

Python je jedným z najpopulárnejších programovacích jazykov pre dátové vedy a vďaka svojej vývojárskej a open source komunite bolo vyvinuté veľké množstvo užitočných knižníc pre vedecké výpočty a strojové učenie [15].

Python je interpretovaný jazyk, ktorý vám môže ušetriť značný čas počas vývoja programu, pretože nie je potrebná žiadna kompilácia a linting. Prekladač je možné používať interaktívne, čo uľahčuje experimentovanie s funkciami jazyka, písanie programov na vyhodenie alebo testovanie funkcií počas vývoja programu zdola nahor. Je to tiež praktická stolová kalkulačka, uviedol Guido van Rossum, tvorca jazyka v tutoriály pre Python [19].

Alternatívy: Matlab, JavaScript, R-lang, Java

3.4.2 Jupyter Notebook

Jupyter notebook je open-source webová aplikácia, ktorá nám umožňuje vytvárať a zdieľať dokumenty, ktoré obsahujú zdrojové kódy, rovnice, či vizualicácie vo forme grafov. Jupyter notebook je nástroj široko používaný v data science komunite, keďže notebook je možné používať aj na numerické simulácie, štatistické modelovanie či machine learning.

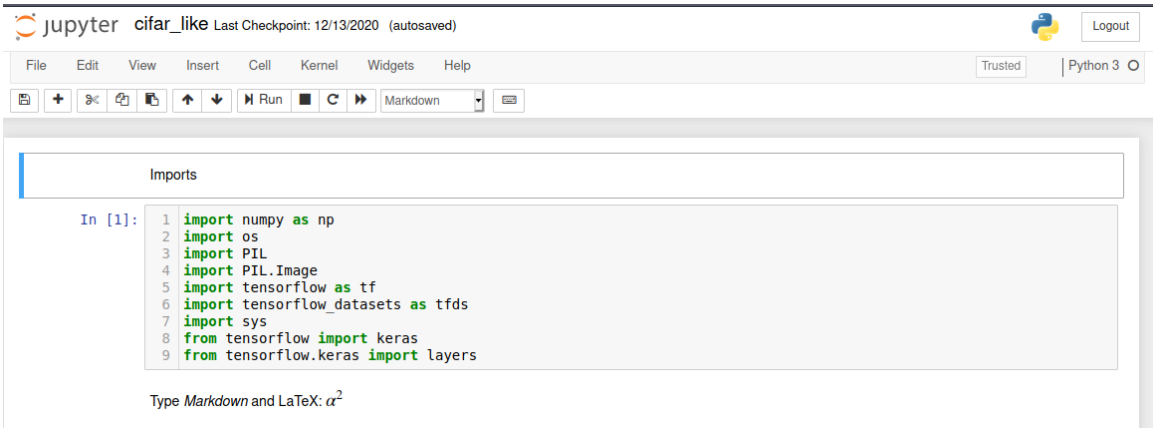
Spôsoby inštalácii je mnoho, najjednoduchší spôsob inštalácie je pomocou balíčkovacieho systému Anaconda, nástroj taktiež zameraný pre data science komunitu. Ja som zvolil inštaláciu pomocou Package Installer for Python (pip).

Po inštalácií sa notebook spúšťa pomocou terminálu jenoduchým príkazom `jupyter notebook`. Príkaz spustí webový jupyter server, ktorý nám poskytne webovú aplikáciu na lokálnom zariadení na príslušnom porte, ktorý je voľný. Predvolený port je `8888`.

Po spustení sa nám otvorí nová karta v prehliadači, kde sa nám otvorí prehľadávač súborov v zložke, v ktorej sme ho v termináli spustili, následne si vytvoríme nový notebook v menu, alebo si otvoríme už existujúci súbor. Python notebooky majú predponu `.ipynb`.

[1] Bounding box je obdĺžnik, ktorý ohraničuje obbjekt, následne tým špecifikuje pozíciu, klasifikuje objekt triedou a vyhodnotí mieru istoty klasifikácie.

Alternatívy: PyCharm, Apache Zeppelin



Obr. 3.6: Jupyter notebook je výborný nástroj pre vzdialenú prácu, kedy pomocou bezpečného `ssh` spojenia sa vieme pripojiť na výpočetné zariadenie a využívať tak jeho výkon ak má zariadenie spustený Jupyter server.

3.4.3 Tensorflow

TensorFlow je end-to-end open-source platforma pre strojové učenie [2]. TensorFlow poskytuje aplikačné rozhranie pre budovanie modelov, ich trénovanie a validáciu.

Systém bol pôvodne vyvynutý výskumníkmi a inžiniermi pracujúcimi v Google Brain tíme patriacej pod Google Machine Intelligence Research organizácií aby spracovávali strojové učenie a rozvíjali štúdiá v hlbokom neuronovom učení. TensorFlow je dostatočne obecný na to, aby pokryl väčšinu problémov týkajúcich sa strojového učenia.

Systém poskytuje API vrstvu implementovanú v jayzku Python a C++. Taktiež TensorFlow provides stable Python and C++ APIs, taktiež poskytuje spätne kompatibilné API pre jazyky ako napríklad JavaScript, Java, Go.

3.4.4 Google Colaboratory

Google Colaboratory, skrátene Colab nám umožňute písať a spúšťať kód v jazyku Python v prehliadači s výhodou toho, že máme k dispozícií výpočetný výkon grafických kariet v Google Cloude pre trénovanie, ktorý je zadarmo. Taktiež veľká výhoda je to, že nemusíme vykonávať konfiguráciu cloudu, inštalácie Python. Google Collaboratory taktiež pri vytvorení projektu obsahuje nainštalovaný TensorFlow.

Prostredie Google Collaboratory je odvodené od Jupyter Notebooku, ktorého funkcie plne využíva a rozširuje, veľká výhoda Google Collaboratory je v zdieľaní projektov a možnosti spoločnej práce na projekte.

Alternatívy: Azure Notebooks, Amazon Sagemaker, IBM DataPlatform Notebooks

3.4.5 Keras

Keras je aplikačná vrstva, ktorá využíva praktiky pre redukciu kognitívnej záťaže pri práci so strojovým učením. Poskytuje jednoduché API, ktoré minimalizuje počet ľudských akcií

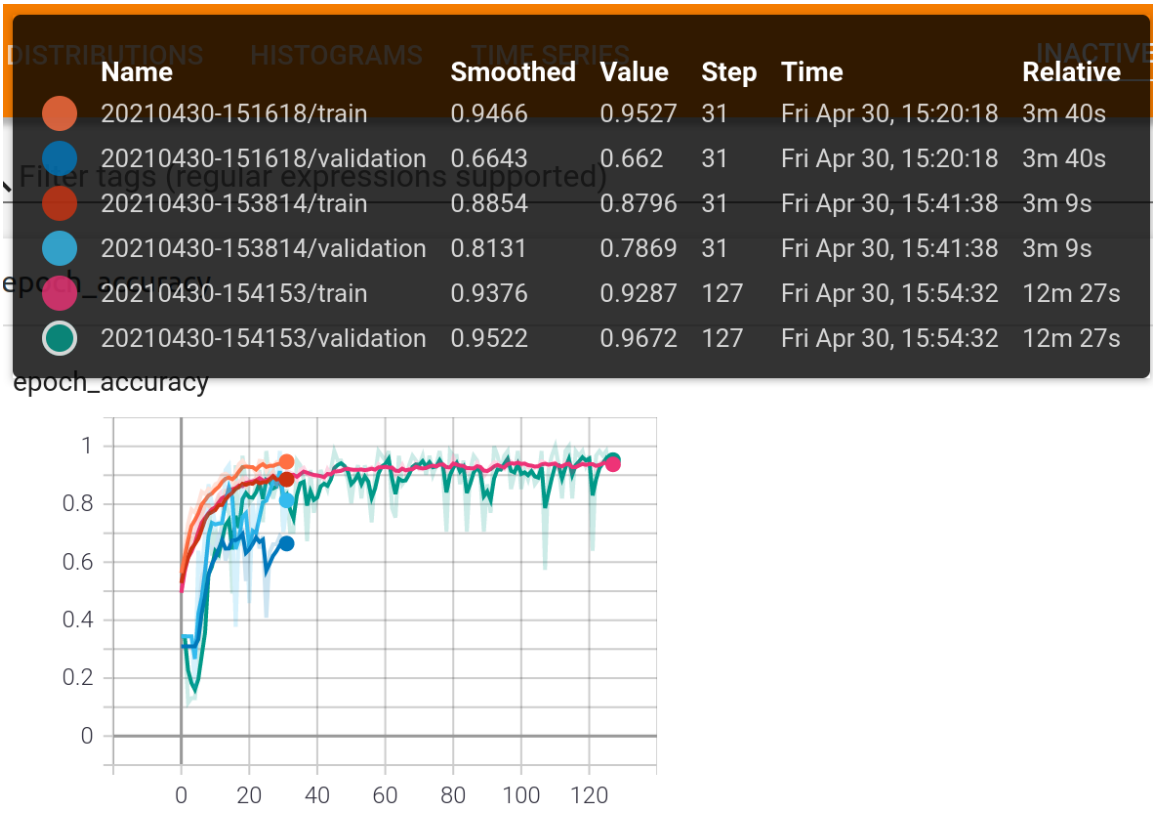
potrebných na bežné prípady použitia. [7] Keras je framework postavený nad TensorFlow, veľká výhoda Keras je relativne čistý výstup a jednoduchšie hľadanie chýb pri procese vytvárania a trénovania modelu.

Keras je kompaktná a ľahko naučiteľná knižnica na vysokej úrovni pre Python na hlboké učenie, ktorá môže bežať nad TensorFlow. Umožňuje vývojárom sústrediť sa na hlavné koncepty hlbokého učenia, ako je vytváranie vrstiev pre neurónové siete, a zároveň sa starať o drsné detaily tenzorov, ich tvary a matematické detaily. TensorFlow musí byť pre Keras koncovým serverom. Keras je možné používať pre aplikácie hlbokého učenia bez interakcie s relatívne zložitým TensorFlow [12]. Existujú dva hlavné druhy rámca: sekvenčné API a funkčné API. Sekvenčné API je založené na myšlienke postupnosti vrstiev; toto je najbežnejšie použitie systému Keras a najľahšia časť systému Keras. Sekvenčný model možno považovať za lineárny sled vrstiev.

Alternatívy: TFLearn, DeepPy, Torch

3.4.6 TensorBoard

TensorBoard poskytuje vizualizáciu a nástroje potrebné pre prácu a experimentovanie so strojovým učením, poskytuje vizualiácie a metriky napríklad pri presnosti trénovania dátovej sady, obsahuje možnosť vizualiácie modelu, histogramov váh, biasov a iných dynamických tenzorov. Tensorboard sa spúšťa pri štarte trénovania a je dostupný od začatia prvej trénovacej epochy.



Obr. 3.7: Veľká výhoda tensorboardu je v zobrazovaní historických trénovacích dát, na obrázku sú zobrazené trénovacie hodnoty pre rôzne trénovacie iterácie.

3.4.7 OpenCV

OpenCV je open-source knižnica pre počítačové videnie [5]. Knižnica je napísana v jazyku C a C++ a beží pod operačným systémom Linux, Windows a MacOS.

Knižnica OpenCV je aktívne vyvýjaná na rozhraniach pre Python, Ruby, Matlab a ďalších jazykoch.

OpenCV bol navrhnutý pre výpočtovú efektívnosť a so silným zameraním na aplikácie bežiace v reálnom čase. OpenCV je napísaný v optimalizovanom jazyku C a môže využívať výhody viacjadrových procesorov. Ak chcete ďalšiu automatickú optimalizáciu na architektúrach Intel, môžete si kúpiť knižnice Intel Integrated Performance Primitive (IPP), ktoré pozostávajú z nízkoúrovňových optimalizovaných rutín v mnohých rôznych algoritmických oblastiach. OpenCV automaticky používa príslušnú knižnicu IPP za behu, ak je nainštalovaná.

Jedným z cieľov OpenCV je poskytnúť ľahko použiteľnú infraštruktúru počítačového videnia ktorá pomáha ľuďom rýchlo vytvárať pomerne sofistikované vizuálne aplikácie. OpenCV Knižnica obsahuje viac ako 500 funkcií, ktoré zahŕňajú mnoho oblastí videnia vrátane tovární kontrola produktu, lekárske zobrazovanie, bezpečnosť, užívateľské rozhranie, kalibrácia kamery, stereo vízia a robotika. Pretože počítačové videnie a strojové učenie idú často ruka v ruke OpenCV tiež obsahuje úplnú univerzálnu knižnicu strojového učenia (MLL).

Táto subknižnica je zameraná na štatistické rozpoznávanie a klastrovanie. MLL je veľmi užitočné pre úlohy spojené s víziou, ktoré sú jadrom poslania OpenCV, ale je všeobecne natoľko všeobecné, aby sa dali použiť pri akýchkoľvek problémoch so strojovým učením.

Kapitola 4

Dátové sady, príprava dát a trénovanie modelu

Pri rozdelení krabice do podčastí bolo potrebné predpripraviť dátovú sadu, podľa ktorej sa bude model učiť. Keďže verejne dostupné dátové sady boli nevhodné, rozhodol som sa zostrojiť vlastnú dátovú sadu. Dátová sada pozostáva z 300 fotiek rôznych krabíc. Pričom z každej fotky boli vyextrahované výrezy podobjektov krabice. Pre trénovanie detektorov krabíc bola zostrojená dátová sada vo viacerých iteráciách v závislosti na počte podobjektov a typu prístupu k problému detekcie krabíc.

4.1 Dátové sady

V tejto podkapitole sú popísané dátové sady, ktoré boli použité pre experimenty s detekciou krabíc.

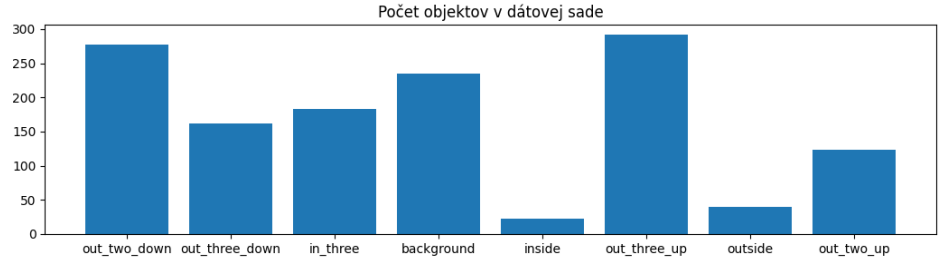
4.1.1 Vlastná dátová sada

Vlastná dátová sada obsahuje fotografie vo vysokom rozlíšení, z ktorých boli vytvorené výrezy o veľkosti 32x32, veľkú chybu som spravil pri vyrezávaní, kedy som si presné body výrezu neuložil z počiatočnej fotky, ale uložil som si len výrezy. Dátová sada je uložená v zložke `raw-data`.

Obr. 4.1: Ukažka dátovej sady s najjednoduchším pozadím, zložitejším s rozvlneným pozadím a pozadím so zmenou farebného spektra

4.1.2 Dátová sada pre klasifikáciu ôsmych podobjektov

Dátová sada sa nachádza v zložke `ds8`.
Dátová sada obsahuje krabicu rozdelenú na osem častí popísanú. Táto dátová sada je pre experiment, kedy som z krabice vybral dve druhy hrán, vnútornú a vonkajšiu. K hranám, som pridal rohy krabice, ako prvý som vybral roh, ktorý je viditeľný takmer vždy pri pohľade na krabicu, jediný moment, kedy tento roh nie je vidieť, je keď na krabicu pozeráme priamo. Je to roh, v ktorom sa spájajú tri vnútorné hrany. Následne som rohy rozdelil ešte do dvoch kategórií, rohy, ktorý sú na vrchu alebo na spodku krabice. Následne polohu rozšíril o rozdiel či roh spájajú z pohľadu dve strany, alebo strany tri. Posledný objekt je pozadie.

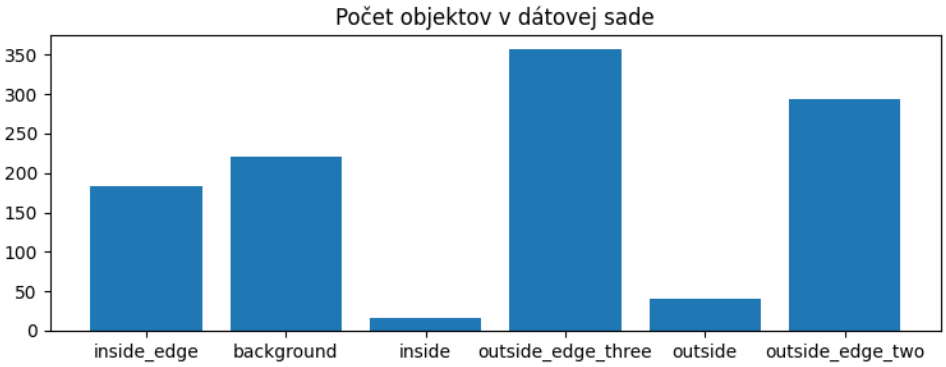


Obr. 4.2: Graf počtu objektov v dátovej sade pre osem podčastí

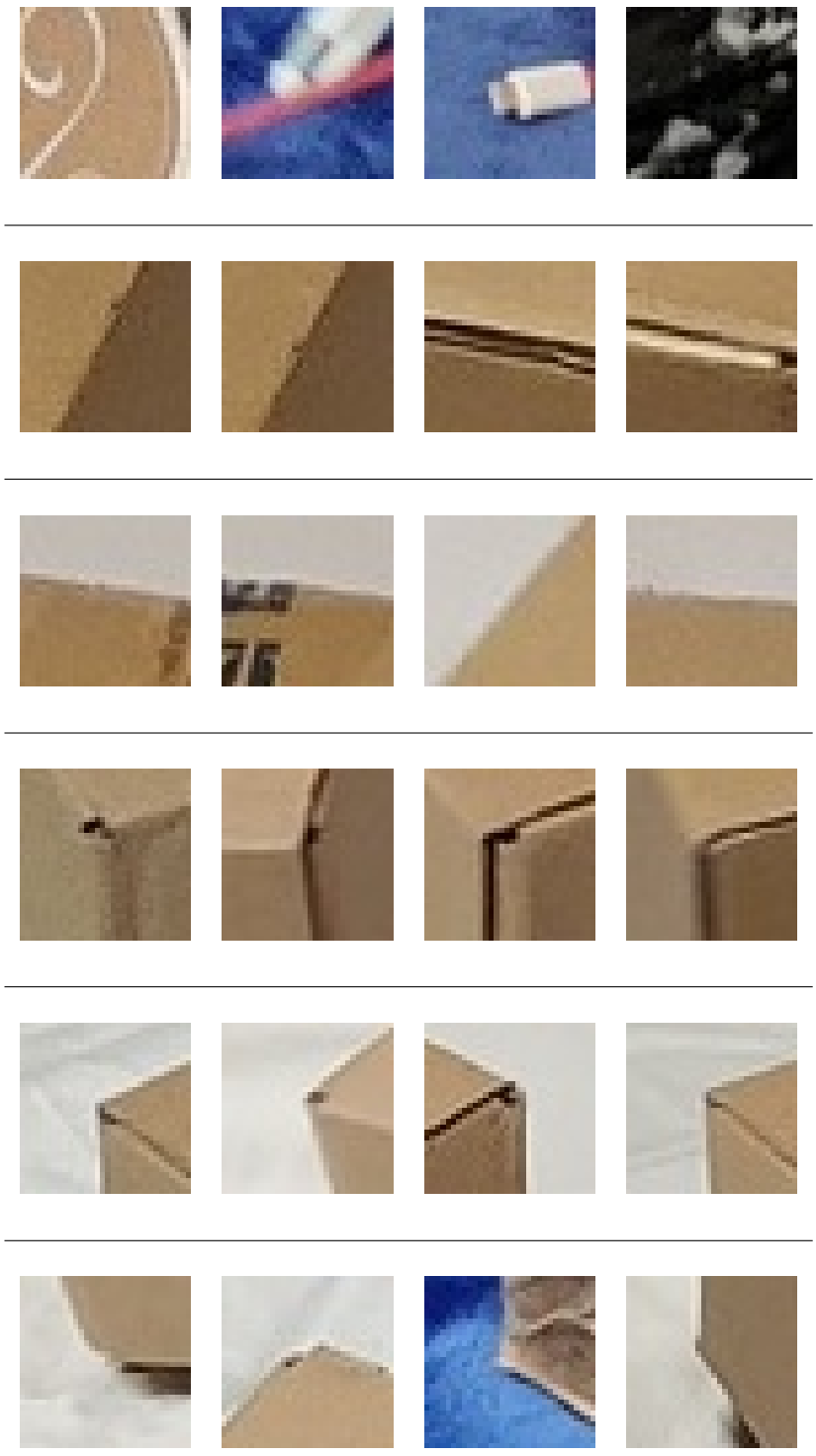
4.1.3 Dátová sada pre klasifikáciu šiestich podobjektov

Dátová sada sa nachádza v zložke `ds6`.

Dátová sada obsahuje krabicu rozdelenú na šesť častí popísanú popísanú v kapitole **??**. Táto dátová sada je pre experiment, kedy som z krabice vybral dve druhy hrán, vnútornú a vonkajšiu. K hranám, som pridal rohy krabice, ako prvý som vybral roh, ktorý je viditeľný takmer vždy pri pohľade na krabicu, jediný moment, kedy tento roh nie je vidieť, je keď na krabicu pozeráme priamo. Je to roh, v ktorom sa spájajú tri vnútorné hrany. Následne som rohy rozdelil ešte do dvoch kategórií, roh, ktorý spája z pohľadu dve strany a roh, ktorý spája strany tri. Posledný objekt je pozadie.



Obr. 4.3: Graf počtu objektov v dátovej sade pre šesť podčastí



Obr. 4.4: Na obrázku sú vyzobrazené ukážky prvkov dátovej sady vo veľkosti 32x32, každý riadok obsahuje podobjekt. Prvý riadok reprezentuje prvky pozadia, druhý riadok obsahuje vnútornú hranu, tretí riadok obsahuje vonkajšiu hranu, štvrtý riadok obsahuje prvky vnútorného rohu, piaty a šiesty riadok obsahujú prvky rohov s troma a dvoma hranami. Podrobný popis podobjektov sa nachádza v kapitole 3.1

Vstupné parametre programu		
Popis	Prepínač	Dátový typ
Cesta k vstupnej zložke s obrázkami	-i	string
Label objektu pre anotáciu	-l	string

Tabuľka 4.1: Prehľad vstupných parametrov

4.1.4 YOLO dátová sada

Dátová sada sa nachádza v zložke `ds-yolo`.

V dátovej sade pre YOLO, detektor využíva špeciálnu dátovú sadu, ku každému obrázku náleží `.txt` súbor s rovnakým názvom ako má obrázok v rovnakej zložke. Každý `.txt` súbor obsahuje anotácie pre obrázkový súbor, anotačný súbor obsahuje zoznam objektov a k nim prislúchajúce súradnice, kde sa objekt nachádza. Formát pre YOLO je v tvare:

```
trieda-objektu x-súradnica y-súradnica šírka výška
```

Kedy trieda objektu, je číslo označujúce kľúč poľa objektov, x-súradnica je súradnica stredu anotácie na osi x, y-súradnica je súradnica stredu anotácie na osi y. Šírka a výška reprezentuje veľkosť šírky a výšky objektu v obrázku, zmena je v tom, že hodnoty v x a y nie sú aboslútne, ale relatívne.

Obsah anotačného `.txt` súboru následne môže obsahovať aj viac anotácií, a to tak, že každý jeden objekt v obrázku reprezentuje jeden riadok textového súboru s prislúchajúcim číslom objektu, ktorému patrí.

4.2 Príprava dát

V tejto podkapitole sú ukážky a vysvetlenia častí implementácie pre prípravu dát z dátovej sady.

4.2.1 Program pre anotácie dát

Program je uložený pod názvom `annotate.py`.

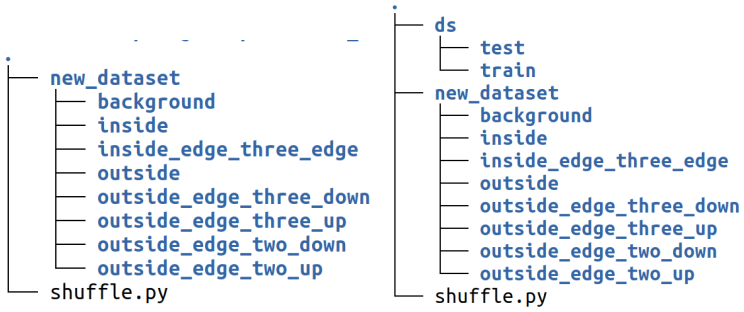
Pre anotáciu dát som implementoval program, ktorý mi vyhovoval v rámci rýchlosti. Program, ktorý som implementoval, číta z predom určenej zložky obrázky, obrázok zobrazí, čaká na n-klikov do obrázka, ktoré obsahujú predom určenú kategóriu zadanú ako parameter programu. Po stlačení klávesy enter sa vytvorí súbor vo formáte JSON s rovnakým názvom ako má obrázok s pridanou príponou .json. Obsah súboru je upravený o nové názvy kategórií a ich polohou v obrázku. Program obsahuje prograss bar na štandardnom výstupe. Progress bar sa aktualizuje po každej anotácií súboru a zobrazuje hodnotu v percentách, ktorá znázorňuje aký pomer dátovej sady máme anotovaný.

4.2.2 Program pre rozdelenie testovacej a trénovacej sady

Program je uložený pod názvom `shuffle.py`.

Vstupné parametre programu		
Popis	Prepínač	Dátový typ
Cesta k vstupnej zložke s dátami	-i	string
Cesta k výstupnej zložke	-o	string
Pomer medzi testovaciou a trénovaciou sadou	-r	float

Tabuľka 4.2: Prehľad vstupných parametrov



Obr. 4.5: Stromová štruktúra pred a po vygenerovaní dátovej sady

Pre rozdelenie dátovej sady som zostrojil program, ktorý načíta zložku obsahujúcu zložky s kategóriami kde každá zložka s kategóriou obsahuje obrázky dátovej sady. Program načíta postupne každú zložku kategórie a obrázky v zložke rozdelí v pomere, ktorý si určíme a roztriedi ich do novej zložky, ktorá obsahuje dve podzložky `train` a `test`, ktoré obsahujú zložky s kategóriami objektov a tie zložky obsahujú roztriedené obrázky.

4.2.3 Program pre detekciu pomocou sliding-window

Program je uložený pod názvom `detector.py`.

Program príjma ako vstup model a obrázok, ktorý chceme spracovať, výsledok programu je obrázok, ktorý je pixel po pixely označený farbou, ktorá patrí ku klasifikovanému objektu.

Obrázok sa načíta pomocou knižnice Pillow a jeho následná úprava po klasifikácii pomocou knižnice cv2.

Vstupné parametre programu		
Popis	Prepínač	Dátový typ
Cesta k vstupnému obrázku	-i	string
Cesta k modelu pre detekciu	-m	string
Cesta k výstupnému obrázku	-o	string
Hranica pre detekciu objektu	-t	float

Tabuľka 4.3: Prehľad vstupných parametrov

4.3 Trénovanie modelu

V tejto podkapitole sú ukážky a vysvetlenia častí implementácie pre trénovanie modelu. Celý obsah implementácie vysvetleného kódu je súčasť Jupyter Notebooku.

Neurónová sieť musí byť nakonfigurovaná tak, aby produkovala aplikáciu sady vstupov (buď „priamo“, alebo prostredníctvom relaxačného procesu) požadovanú sadu výstupov. Existujú rôzne metódy nastavenia so silnými stránkami. Jedným zo spôsobov je nastavenie váh explicitne pomocou a priori vedomosti. Ďalším spôsobom je „trénovať“ neurónovú sieť tým, že ju kŕmi výučbovými vzormi a nechať sieť meniť svoju váhu podľa nejakého pravidla učenia. [9]

4.3.1 Paradigma učenia

Učebné situácie môžeme rozdeliť do dvoch odlišných druhov:

- Učenie pod dohľadom alebo asociačné učenie, pri ktorom sa sieť trénuje poskytovaním vstupu a zodpovedajúcimi výstupnými vzormi. Tieto páry vstupov a výstupov môže poskytnúť externý učiteľ alebo systém obsahujúci sieť (s vlastným dohľadom).
- Učenie bez dohľadu alebo samoorganizácia, v ktorej je (výstupná) jednotka trénovaná, aby reagovala do zhlukov vzoru vo vstupe. V tejto paradigme sa predpokladá, že systém dokáže pokryť štatisticky významné prvky vstupnej populácie. Na rozdiel od kontrolovaného učenia paradigma neexistuje a priori súbor kategórií, do ktorých sa majú vzory zatriediť. Systém si musí skôr vytvoriť vlastnú reprezentáciu vstupných stimulov.

4.3.2 Epocha

Počet epoch je hyperparameter, ktorý definuje, koľkokrát bude algoritmus učenia pracovať s celým súborom trénovacej dátovej sady.

Jedna epocha znamená, že každá vzorka v trénovacej dátovej sade mala príležitosť aktualizovať parametre interného modelu. Epocha sa skladá z jednej alebo viacerých dávok (anglicky batches).

O epoche môžete uvažovať ako o cykle for nad počtom epoch, kde každá iterácia postupuje cez množinu trénovacej dátovej sady. V rámci tohto cyklu je ďalši vnorený cyklus, ktorý iteruje cez každú dávku vzoriek (anglicky batches), kde jedna dávka má „veľkosť dávky“ (anglicky batch size) ktorý reprezentuje počet vzoriek.

Počet epoch je tradične veľký, často stovky alebo tisíce, čo umožňuje výučbový algoritmus bežať, kým nie je chyba z modelu dostatočne minimalizovaná. [6]

4.3.3 Augmentácia

Pojem sa týka metód na konštrukciu iteratívnej optimalizácie alebo algoritmov vzorkovania zavedením nepozorovaných údajov alebo latentných premenných [18].

Augmentácia je výborný spôsob, ako môžeme obmedziť prispôsobenie na modeloch, kde zvyšujeme množstvo trénovacích údajov pomocou informácií, ktoré už máme v našich trénovacích dátach. Pole rozširovania údajov nie je nové a na konkrétne problémy sa v skutočnosti uplatnili rôzne techniky rozširovania údajov. Hlavné techniky spadajú do kategórie ohýbania dát, čo je prístup, ktorý sa snaží priamo rozšíriť vstupné údaje do údajového priestoru modelu. Veľmi všeobecnou a akceptovanou súčasnou praktikou pri rozširovaní obrazových

údajov je vykonávanie geometrických a farebných augmentácií, ako je zrkadlenie obrazu, otáčanie, orezávanie a zmena farebnej palety obrazu [20]. Napríklad ak zoberieme obrázok, na ktorom je objekt, ktorého detekcia nezávisí od natočenia objektu, tak môžeme pridať augmentáciu otočenia a vytvoríme tým N nových prvkov dátovej sady, kedy N je hodnota počtu otočených vzoriek.

4.3.4 Metóda pre výber dátovej sady a augmentácie

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    f"{dataset_directory}/train",
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    f"{dataset_directory}/test",
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)

data_augmentation = keras.Sequential([
  layers.experimental.preprocessing.RandomRotation((X, Y)),
])
```

4.3.5 Metóda pre vytvorenie modelu

```
def make_model(input_shape, num_classes):
  inputs = keras.Input(shape=input_shape)
  x = data_augmentation(inputs)

  x = layers.experimental.preprocessing.Rescaling(1. / 255)(x)
  x = layers.Conv2D(32, 3, strides=2, padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Activation('relu')(x)

  x = layers.Conv2D(64, 3, padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Activation('relu')(x)

  previous_block_activation = x

  for size in [128, 256, 512, 728]:
    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(size, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(size, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding='same')(x)

    residual = layers.Conv2D(
      size, 1, strides=2, padding='same')(previous_block_activation)
    x = layers.add([x, residual])
    previous_block_activation = x

  x = layers.SeparableConv2D(1024, 3, padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Activation('relu')(x)

  x = layers.GlobalAveragePooling2D()(x)
  if num_classes == 2:
    activation = 'sigmoid'
    units = 1
  else:
    activation = 'softmax'
    units = num_classes

  x = layers.Dropout(0.5)(x)
  outputs = layers.Dense(units, activation=activation)(x)
  return keras.Model(inputs, outputs)
```

Ukážka vytvárania siete s dynamickým vstupom počtu klasifikovaných objektov a statickou veľkosťou vstupu obrázka. Metóda v prvej vrstve nastaví vstup pre dátovú sadu, následne nad dátovou sadou vykoná augmentáciu a začne vytvárať plne prepojené konvolučné vrstvy. V poslednej časti metóda podľa počtu klasifikovaných kategórií určí typ aktivačnej funkcie a nakoniec model vyvorí objekt `keras.Model`, ktorá pri inicializácií príjme vstup a výstup.

4.3.6 Metóda pre trénovanie modelu

```
epochs = 32
model = make_model(input_shape=image_size + (3,), num_classes=N)

callbacks = [
  keras.callbacks.ModelCheckpoint('model.h5'),
]

model.compile(optimizer=keras.optimizers.Adam(1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()


model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

Po vytvorení modelu a načítaní dátovej sady spustíme trénovanie neurónovej siete nad dátovou sadou, framework keras pre všetky kroky obsajhuje metódy, ktoré potrebujeme. Pred spustením trénovania nastavíme callback pre vytváranie chechkpointu pre model, aby nám ukladal po každej iterácií dočasný model. Následne vyberieme počet epoch, následne vyberieme metódu pre kategorickú stratu v metóde `compile`, ktorá nám model skompiluje. Trénovanie sa začne po spustení metódy `fit`, kde zadáme dátové sady, počet epoch, callbacky a začne sa trénovanie.

4.3.7 Metóda pre rýchlu validáciu modelu po trénovaní

```
import numpy as np

for i~in range(0, N):
  img = keras.preprocessing.image.load_img(f'{i}.jpg',
                                           target_size=image_size)

  img_array = keras.preprocessing.image.img_to_array(img)
  img_array = tf.expand_dims(img_array, 0)
  predictions = model.predict(img_array, batch_size=128)
  print(np.argmax(predictions[0]))
```

Po natrénovaní modelu potrebuejme jednoducho zvalidovať model, model zvalidujem tým, že na vstup program príjme N obrázkov, z ktorých každý reprezentuje podčasť kategórie, ak je výstup sekvencia čísel od 0..N, zistíme, že model je správne nastavený.

V prípade, že sekvencia je nelineárna a hodnoty sa odlišujú, vieme určiť problémy ako nevadidná dátova sada alebo vysokú podobnosť podobjektov, kedy by bolo vhodné dátovú sadu zredukovať.

Program v každej iterácií využíva model kedy na vstup vyberie obrázok s číslom detekovaného podobjektu a porovná ho s výsledkom predikcie modelu.

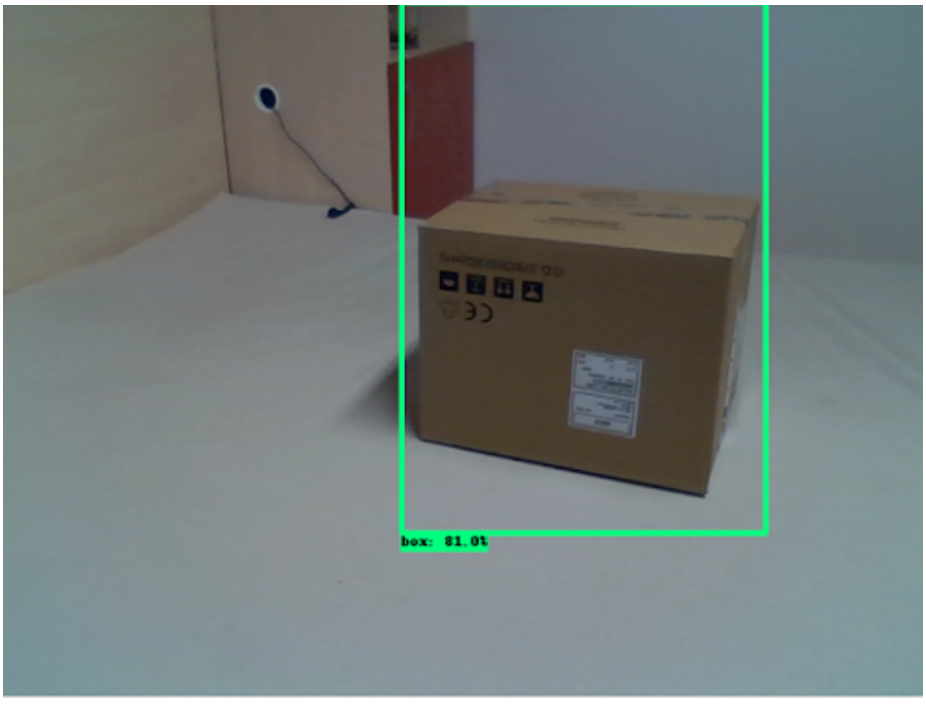
Kapitola 5

Experimenty a implementácie

5.1 EfficientDet

Prvý experiment prebehol tak, aby som si overil, či je vôbec možné krabice ako objekty detekovať nad vytvorenou dátovou sadou. Experiment bol úspešný, čím som si overil svoju prvú iteráciu detekcie a dátovej sady.

Experiment spočíval vo vytvorení 50 fotiek krabíc a ich anotácii. Následne som vybral predtrénovaný model EfficientDet a dotrénoval som ho tak, aby bol model schopný detektovať krabice. Výber modelu prebiehal čisto náhodne a pred výberom som nevykonával žiadne porovnanie medzi predtrénovanými modelmi.

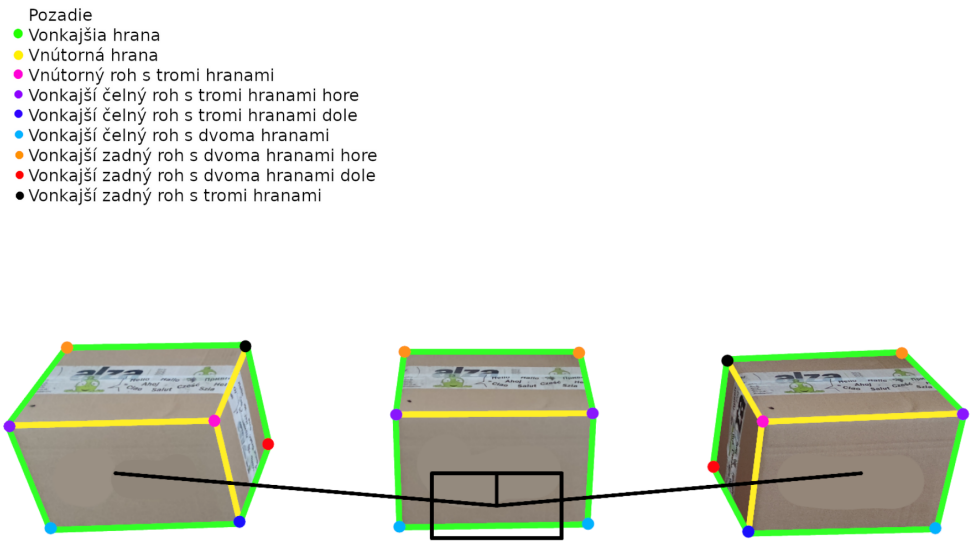


Obr. 5.1: Experiment pomocou dotrénovania predtrénovaného modelu EfficientDet, na fotke s jednoduchým pozadím mala detekcia krabice presnosť približne 80 percent

5.2 Vlastný konvolučný model a sliding window

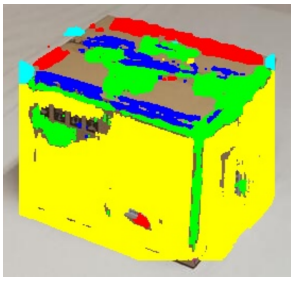
Experiment spočíval v zostrojení dátovej sady obsahujúcej výrezy 32x32 pixelov, pričom boli výrezy krabice rozdelené do troch iterácii.

V prvej iterácií som používal 10 podobjektov krabice kedy v prípade statického pozorovateľa môžu nastať prípady, ked sa nám uhol medzi pozíciou pozorovateľa a objektom po prechode cez nulový bod (krabica je priamo pred ohniskom pozorovateľa) zmení ostrosť uhla, čo nám zapríčiní preklopenie typov rohov. Toto preklápanie je možné eliminovať tak, že pozorovateľ bude zaznamenávať len v uhle <0, 90), (90, 180> kedy sa eliminujú preklopenia medzi intervalmi, kedy pri prechode od 89-91 "stav" objektu bude iný v každom uhle.



Obr. 5.2: Ukážka 10 podobjektov krabice a preklopenie perspektívy z pohľadu statického pozorovateľa

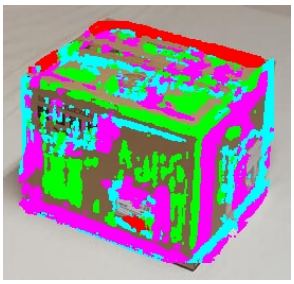
V druhej iterácií, aby som zamedzil problému s preklopením perspektívy, ubral som z desiatich podobjektov dva objekty tým, že som zlúčil rohy na horné a dolné rohy s počtom hrán, ktoré roh spája. Takže napríklad horný roh s dvoma hranami.



Obr. 5.3: Ukážka detekcie 8 podobjektov krabice a preklopenie perspektívy z pohľadu statického pozorovateľa

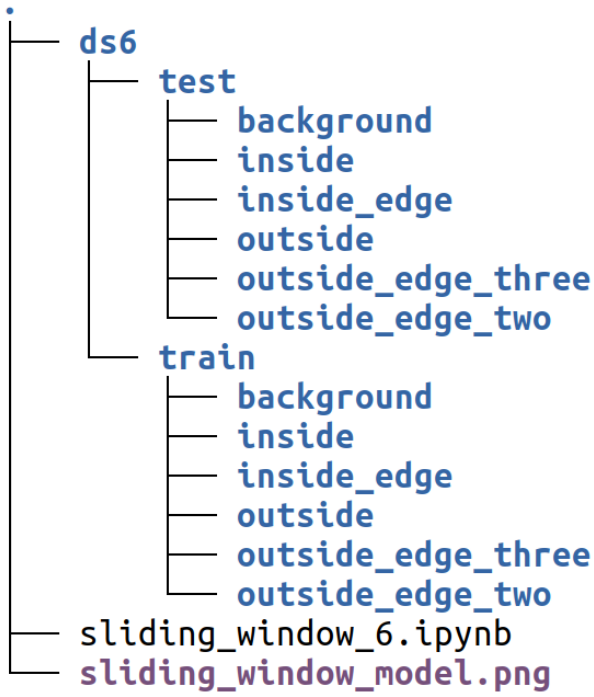
V tretej iterácií experimentu som krabicu rozdelil do šiestich podobjektov tak, že som rohy krabicke bral podľa dvoch typov, rohy s dvoma stranami a rohy s troma stranami. Veľ-

kou výhodou bolo to, že som mohol využiť augmentáciu dátovej sady, využil som náhodne otočenie v intervale 0-180 stupňov.



Obr. 5.4: Ukážka detekcie 6 podobjektov krabice

Implementácia pre dva druhy dátových sád pre osem a šesť podobjektov krabice sa nachádzajú v zložkách sliding-window-n, kde n značí počet podobjektov.



Obr. 5.5: Príklad stromovej štruktúry zložky pre šesť podobjektov, každá zložka obsahuje dátovú sadu rozdelenú na trénovacie a testovacie dáta, jupyter notebook a obrázok sumáru modelu

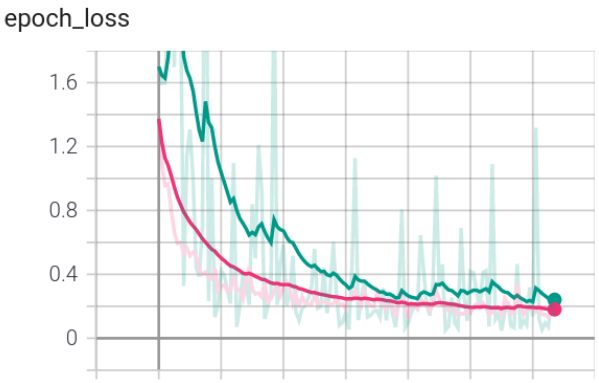
Detekcia pomocou posuvného okna je spoľahlivá, no veľmi náročná na výpočetný výkon, je náročná tak, že na zariadení len na CPU som sa nedostal na real-time detekciu. Výpočetná náročnosť je tak vysoká kvôli tomu, že v každej iterácií posuvného okná dochádza ku klasifikácií, čiže každú iteráciu prebiehajú konvolučné výpočty. Zrýchliť detekciu som sa pokúsil pomocou zväčšenia posunu posuvného okna.

Alternatívne možno použiť miestne optimalizačné metódy namiesto globálnych, a to tak, že najskôr identifikujete nádejné oblasti v obraze a odtiaľ maximalizujete postup gra-

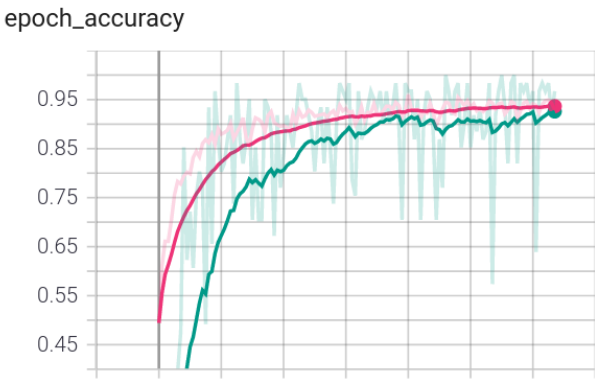
dientu. Zredukované techniky vyhľadávania obetujú robustnosť lokalizácie na dosiahnutie prijateľnej rýchlosti [10]. Ich implicitným predpokladom je, že kvalitatívna funkcia je plynulá a pomaly sa mení. To môže viesť k nesprávnym odhadom alebo dokonca k úplným zhodám s umiestneniami objektov, najmä ak má maximum funkcie klasickej triedy podobu ostrého vrcholu v priestore parametrov.

Trénovanie modelu a počty prvkov v dátovej sade pri kompilácií modelu.

```
Found 788 files belonging to 6 classes.
Using 631 files for training.
Found 306 files belonging to 6 classes.
Using 61 files for validation.
```



Obr. 5.6: Epoch loss graf po dotrénovaní 128 epoch, ružový graf zobrazuje stratu pre trénovaciu sadu a zelený stratu pre sadu testovaciu. Konečná hodnota straty pre testovanie je 0,1438 a pre trénovanie 0,1769

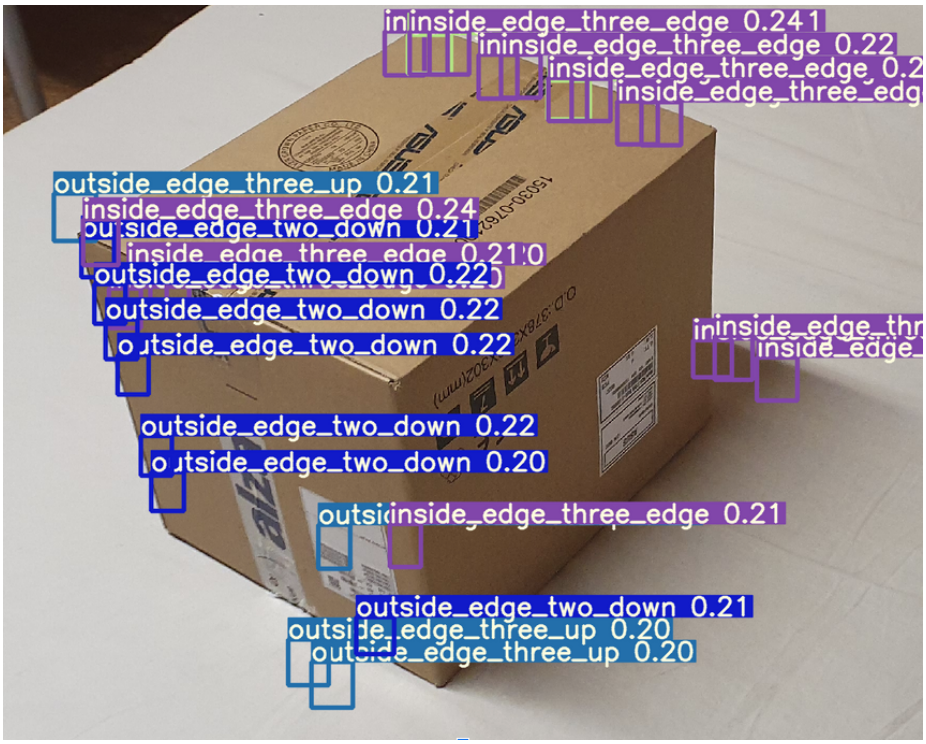


Obr. 5.7: Graf presnosti modelu na trénovacej dátoej sade po dotrénovaní 128 epoch, výsledná hodnota prenosti je 96,2% pre trénovaciu sadu a 92,8% pre testovaciu sadu

5.3 YOLO

Experiment spočíval v zostrojení dátovej sady pre známy detektor YOLO, kedy som anotoval dáta experimentom, kedy som vytvoril 32x32 obrázky, ktorých veľkosť bounding boxu anotácie bol rovnaký ako šírka a výška samotného obrázka.

Experiment som vykonával na počítači bez grafickej karty, preto rýchlosť detekcie bola veľmi podpriemerná. Taktiež YOLO nie je implementované pre tento typ anotácií, preto by bolo vhodné zopakovať experiment s anotáciami, ktoré sú súčasťou fotografie alebo obrázka.



Obr. 5.8: Experiment pomocou dotrénovania predtrénovaného modelu EfficientDet, na fotke s jednoduchým pozadím mala detekcia krabice presnosť približne 80 percent

Kapitola 6

Vylepšenia

V tejto kapitole sú uvedené prístupy k detekcií, ktoré som po naštudovaní zhodnotil, že sú vhodné pre riešenie problému detekcie nie len krabíc, ale akéhokoľvek objektu, ktorý sa skladá z logicky rozmiestnených podobjektov.

6.1 UNET

U-net je architektúra konvolučných sietí pre rýchlu a precíznu segmentáciu obrázkov, architektúra bola vydaná v roku 2015 kedy prekonala najpopulárnejšiu metódu posuvného okna na ISBI výzve pre segmentáciu neurónových štruktúr v elektrónových miktroskopických hromád.

Architektúra U-net na tejto výzve vyhrala Grand Challenge pre počítačovo automatizované detekcie kazov v Bitewing Radiography na ISBI 2015 a taktiež architektúra vyhrala výzvu Cell Tracking Challenge na ISBI 2015 na dvoch najväčších výzvach v kategórií prenosov svetelných mikroskopových kategórií s veľkým prehľadom oproti konkurenčným implementáciám.

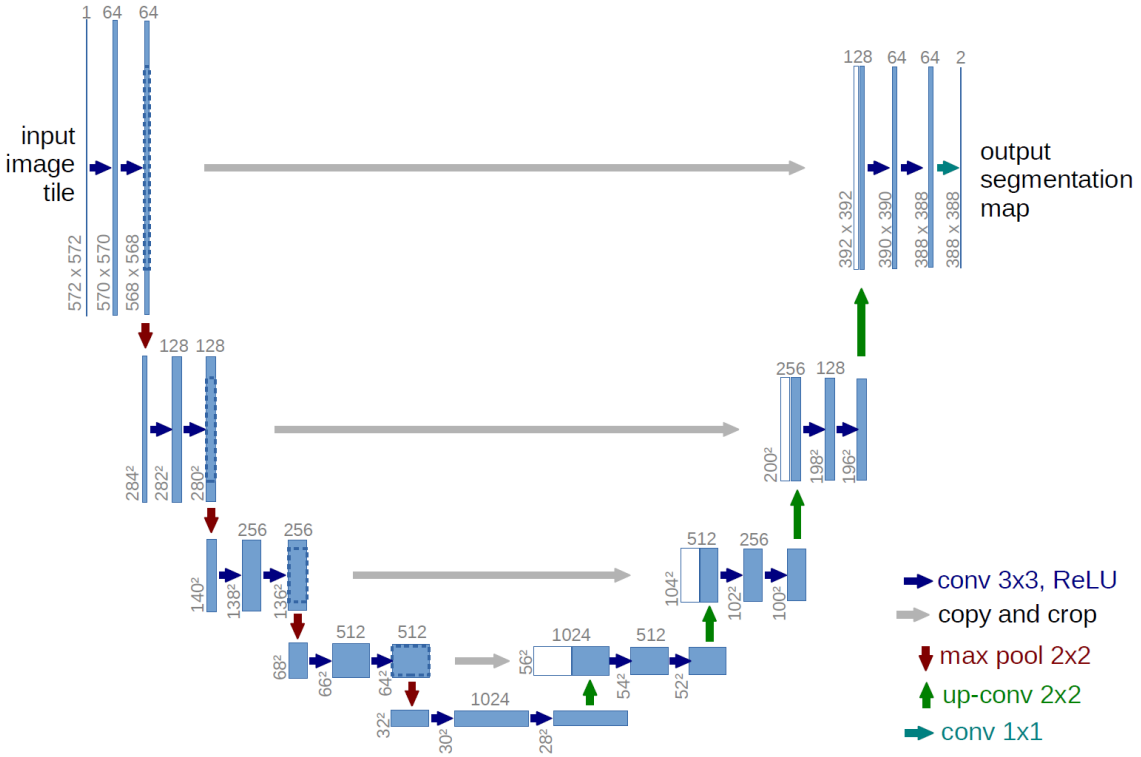
Typické využitie konvolučných sieti je na úlohy klasifikácia, kde výstup z obrázka je jedna trieda. No existujú vizuálne problémy, špeciálne v biomedicínskych vizualizáciách a ich processingu, kde výstup z obrázka obsahuje nie len klasifikáciu, ale aj lokalizáciu daného objektu v obraze. Teda až na takej úrovni, kedy sa klasifikácia vykonáva v každom pixeli obrazu. [8]

UNET využíva architektúru plne prepojených konvolučných sietí, ktorá pozostáva z kontrakčnej cesty (ľavá strana) a expanzívnej cesty (pravá strana). Cesta kontrakcie využíva typickú architektúru konvolučnej siete. Pozostáva z opakovanej aplikácie dvoch konvolúcií 3x3, po ktorej nasleduje usmernená lineárna jednotka (ReLU) a operácia združovania 2x2 max s krokom 2 pre prevzorkovanie. V každom kroku prevzorkovania zdvojnásobíme počet výkonných strojov.

Každý krok v expanzívnej ceste pozostáva z prevzorkovania mapy funkcií, po ktorom nasleduje konvolúcia 2x2, ktorá znižuje počet kanálov funkcií na polovicu, zreťazenie s príslušne orezanou mapou funkcií zo zmluvnej cesty a dva konvolucie 3x3 nalsedovanej ReLU. Orezanie je potrebné z dôvodu straty hraničných pixelov pri každej konvolúcii. Na konečnej vrstve sa použije konvolúcia 1x1 na mapovanie každého 64-komponentného vektora funkcií na požadovaný počet tried. Sieť má celkovo 23 konvolučných vrstiev. Ak chcete umožniť plynulé obkladanie výstupnej segmentačnej mapy, je dôležité zvoliť veľkosť vstupnej dlaž-

dice tak, aby sa všetky pooling operácie 2x2 použili na vrstvu s párnymi x veľkosť a veľkosť y. [16]

Výsledkom U-net siete N heatmap[1], kedy N je počet klasifikovaných tried modelu. Veľkou výhodou tohoto prístupu je ten, že nad heatmapami môžeme využívať logické spojitosti medzi objektami, napíklad ak vieme, že vnútorný roh3.1.3 je spájaný práve troma vnútornými hranami, tak podľa heatmap vieme túto skutočnosť validovať.



Obr. 6.1: Na príklade s 32x32 pixelov, každý modrý blok reprezentuje multi-kanálovú feature mapu. Počet kanálov je označený nad každým blokom. x-z-veľkosť je uvedená na spodnom ľavom okraji bloku. Biele bloky označujú kópie feature máp. Šípky označujú operácie uvedené v legende. [16]

6.2 MirrorNet

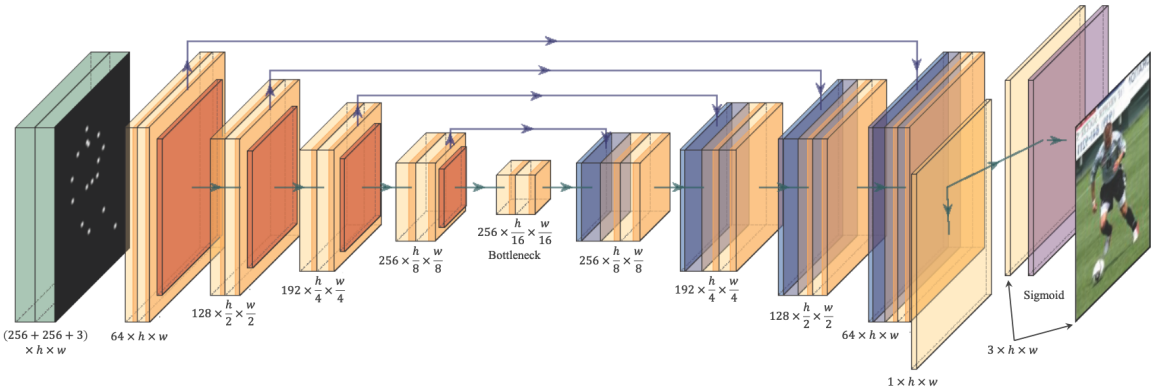
Tento prístup je inšpirovaný kognitívnymi poznatkami o systéme zrkadlovo-neurónového systému človeka a navrhol hlboko bajesovský rámec s názvom MirrorNet pre odhad 2D pozícií z ľudských obrazov.

Kľúčovou myšlienkou je spoločne trénovať generatívne modely obrazov a póz rovnako ako modely rozpoznávania vzhľadov, scén a pozitívov úplne štatistickým spôsobom. Z technického hľadiska sú dvojúrovňové zrkadlové systémy (VAE) trénované spoločne s hierarchickým autoencoding prístupom (obraz → postoj → primitívny → postoj → obraz), takže sa berie do úvahy vierohodnosť aj vernosť póz. Vďaka povahe plne generatívneho modelovania

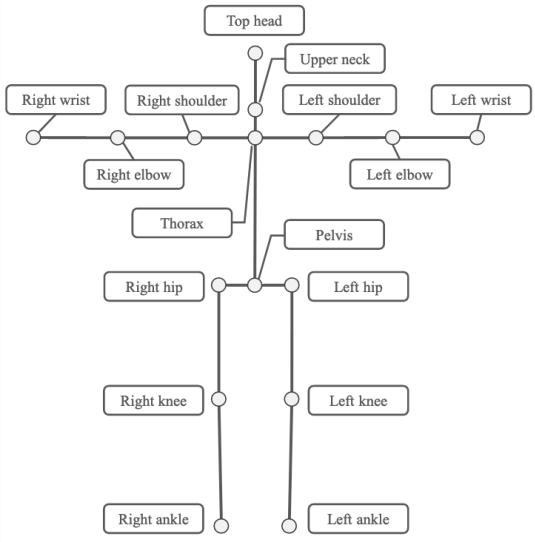
[1] V tomto prípade je heatmapa mapou, kde každý jeden prvok mapy obsahuje typ klasifikovaného objektu a jeho mieru pravdepodobnosti

je MirrorNet prvou architektúrou pre odhad polohy, ktorú je možné teoreticky trénovať z anotovaných obrázkov bez dozoru, keď sa zavedú vhodné indukčné predsudky. [13]

Tento prístup je veľmi podobný s prístupom navrhnutými v tejto práci, no je náročný na anotácie a vytvorenie modelu krabice.



Obr. 6.2: Ukážka Mirrornet siete pre detekciu postavy hráča futbalu [16]



Obr. 6.3: Na obrázku je znázornená postava ľudského tela rozdelená do šestnástich podčastí, prepojených kĺbami, podobný prístup aký bol použitý v tejto práci 5.2

Kapitola 7

Záver

Cieľom tejto práce bolo preskúmať, experimentovať a poznávať prístupy pre detekciu papierových krabíc pomocou algoritmov počítačového videnia. Podľa výsledkov experimentovania bol tento cieľ čiastočne splnený s výhradou pracovať v reálnom čase. V prvej iterácií bol zostrojený model, ktorý pracoval s predtrénovanou sieťou EfficientDet, ktorá bola obohatená o nový objekt, papierovú krabicu. Tento experiment dopadol úspešne, no mal svoje nedostatky vo forme veľkého počtu false-positive detekcií, tento prístup dotrénovania neurónových sietí je efektívny v prípade, že sa snažíme klasifikovať objekt, ktorý je výraznejší ako papierová krabica, ktorá sa v rámci počítačového videnia jednoducho zamení sa objekty v bežnom styku so svetom, napríklad nábytok ako police, skrine či chladničky, ktoré majú jedno spoločné, sú v tvare kvádra. Ako koncept s overením na minimálnom počte vzoriek v dátovej sade dotrénovanie siete bolo dostatočné, no v prípade detekcie krabice ako celku, je nutné zostrojiť masívnu dátovu sadu papierových krabíc, ideálne s využitím reálny záznamov napríklad z logistických centier.

Preto po konzultácií s vedúcim práce som sa rozhodol, že vytvorím experiment s dekompozíciou krabice na podobjekty, kedy som sa snažil ku krabici pristúpiť viac ako k matematickému objektu kvádra, a nie ako ku objektu krabice, experimentoval som s rôznymi počtami podčastí, najprv som rozdelil krabicu do desiatich podčastí a postupne som redukoval počet z desať na osem, následne z osem na šesť, a v poslednej iterácií zo šesť na päť, kedy som zistil, že redukcia na päť podobjektov je už priveľa. Ideálny počet podobjektov, kedy je možné najefektívnejšie využiť augmentáciu dátovej sady je šesť, ako som popísal v kapitole 3.1.

Experimenty s dekompozíciou objektu na podobjekty sú časovo náročné, keďže pri každej zmene počtu alebo zmene významu objektu je potrebné ohýbať dátovú sadu, preto pre experimenty, ako som vytváral ja, je vhodné mať správne podchytenú dátovú sadu tak, aby v prípade zmeny počtu podobjektov vygenerovanie novej dátovej sady nebolo časovo náročné. V tomto som ja spravil veľkú chybu, kedy pri anotovaní dátovej sady, som vytvoril len výrezy o veľkosti 32x32 z pôvodných fotografií v plnom rozlíšení a neuložil som si pozície podobjektov vo fotografiách. Preto, by som na začiatku postupoval tak, že by som si vytvoril anotácie vo forme čo najpresnejších bodov pomyselného stredu podobjektu, uložil ich napríklad vo formáte JSON so správne sémantickym názvom, kedy by bolo možné nie len že dynamicky generovať počty podobjektov, ale aj určiť veľkosť daného výrezu.

Po vytvorení dátových sád som zostrojil vlastnú konvolučnú sieť, pre klasifikáciu podobjektov krabíc, kedy som najviac prenikol do problematicky zostrojenia plne prepojených konvolučných sietí. Túto sieť som používal pre klasifikáciu, no k detekcii mi chýbala lokalizácia. Keďže som vedel klasifikovať statický obraz, prirodzene som začal experiment

s prístupom posuvného okna, no narazil som na problém s veľmi veľkou dobou detekcie obrazu, kedy som sa nepriblížil k detekcii v reálnom čase, experimentoval som s rôznymi veľkosťami vstupného obrazu, či rôznymi veľkosťami posunu okna. Previedol som aj experiment, kedy som pouvné okno náhodne umiestnil do obrazu a v prípade, že som detekoval podobjekt, odrazil som sa od daného miesta a začal hľadať v okolí daného podobjektu. Tento experiment som si interne nazval Monte Carlo, no neprišlo mi, že sa detekcia približuje reálnemu času, keďže v každej iterácií posuvného okna sa vykonávalo veľké množstvo výpočetne náročných konvolúcií.

Preto som sa sústredil na iné prístupy, prístup siete MirrorNet a U-net, ktoré benefitujú z toho, že vytvárajú konvolučnú sieť, ktorá pozostáva z dvoch častí, stláčajúcej, kedy vstupný obraz je stlačený do čo najmenšieho vektora s navyšším počtom kanálov, ktorý je závislý na počte podobjektov. Následne druhá časť siete, expanzná, rozťahuje vektor a rozťahuje ho dovtedy, kým sa veľkosť výstupu nerovná veľkosti vstupného obrázka, no len šírky a dĺžky, na rozdiel od vstupu, kedy pri farebnom obrázku je počet kanálov 3 (RGB), no tretí kanál výstupu je práve počet klasifikovaných podobjektov. Teda výstupom týchto sietí sú heatmapy, pre každý podobjekt. Náročnosť použitia MirrorNet a U-net sa líši, v prípade siete MirrorNet je vhodné použiť 3D model detekovaného objektu, no pri prístupe U-net je postačujúce správne anotovať dátovú sadu, kedy anotovaný podobjekt vo fotografií je rozložený na pravdepodobnostné rozloženie pomocou gaussovej krivky tak, aby sme vedeli určiť správnu koreláciu medzi anotáciou a výslednou heatmapou.

Ku koncu práce by som rád poznamenal, že táto práca bola pre môj osobný rozvoj veľkým prínosom, poskytla mi vynikajúci vstup do problematiky nie len počítačového videnia, ale aj strojového učenia, ktorý chcem rozvýjať. Aj keď práca nie je z môjho pohľadu úspešná, verím, že prispela k poznaniu nie len pre mňa, ale aj k priblíženiu vyriešenia problému detekcie papierových krabíc v obraze.

Literatúra

[1] *An introduction to deep learning*. Dostupné z: https://developer.ibm.com/technologies/artificial-intelligence/articles/an-introduction-to-deep-learning/.

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* [online]. Dostupné z: https://www.tensorflow.org/.

[3] Albawi, S., Mohammed, T. A. a Al Zawi, S. Understanding of a convolutional neural network. In: IEEE. *2017 International Conference on Engineering and Technology (ICET)*. 2017.

[4] Birodkar, V., Lu, Z., Li, S., Rathod, V. a Huang, J. The surprising impact of mask-head architecture on novel class segmentation. 2021.

[5] Bradski, A. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. 1. ed. O'Reilly Media, 2008. ISBN 0-596-51613-4. Gary Bradski and Adrian Kaehler.

[6] Brownlee, J. What is the Difference Between a Batch and an Epoch in a Neural Network? *Deep Learning; Machine Learning Mastery: Vermont, VIC, Australia*. 2018.

[7] Chollet, F. *About Keras* [online]. [cit. 2021]. Dostupné z: https://www.keras.io/about.

[8] Ciresan, D., Giusti, A., Gambardella, L. a Schmidhuber, J. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. Curran Associates, Inc. 2012, zv. 25. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf.

[9] Kröse, B., Krose, B., Smagt, P. van der a Smagt, P. *An introduction to Neural Networks*. 1993.

[10] Lampert, C., Blaschko, M. a Hofmann, T. *Beyond Sliding Windows: Object Localization by Efficient Subwindow Search*. Los Alamitos, CA, USA: IEEE Computer Society, jún 2008. Best paper award.

[11] Lauzon, F. Q. An introduction to deep learning. 2012, s. 1438–1439. DOI: 10.1109/ISSPA.2012.6310529.

[12] Moolayil, J., Moolayil, J. a John, S. *Learn Keras for Deep Neural Networks*. Springer, 2019.

[13] Nakatsuka, T., Yoshii, K., Koyama, Y., Fukayama, S., Goto, M. et al. MirrorNet: A Deep Bayesian Approach to Reflective 2D Pose Estimation from Human Images. 2020.

[14] Nwankpa, C., Ijomah, W., Gachagan, A. a Marshall, S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *CoRR*. 2018, abs/1811.03378. Dostupné z: http://arxiv.org/abs/1811.03378.

[15] Raschka, S. a Mirjalili, V. Python Machine Learning: Machine Learning and Deep Learning with Python. *Scikit-Learn, and TensorFlow. Second edition ed.* 2017.

[16] Ronneberger, O., Fischer, P. a Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. Dostupné z: https://arxiv.org/pdf/2004.03811.pdf.

[17] Szeliski, R. *Computer vision algorithms and applications*. London; New York: Springer, 2011. ISBN 9781848829343 1848829345 9781848829350 1848829353. Dostupné z: http://dx.doi.org/10.1007/978-1-84882-935-0.

[18] Van Dyk, D. A. a Meng, X.-L. The art of data augmentation. *Journal of Computational and Graphical Statistics*. Taylor & Francis. 2001.

[19] Van Rossum, G. et al. Python Programming Language. In: *USENIX annual technical conference*. 2007, sv. 41, s. 36.

[20] Wang, J., Perez, L. et al. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit.* 2017, zv. 11.