VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VYUŽITÍ STROJOVÉHO UČENÍ PRO KONTROLU KVALITY V PRŮMYSLOVÝCH APLIKACÍCH

USING MACHINE LEARNING FOR QUALITY CONTROL IN INDUSTRIAL APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Viktor Gaško

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Roman Parák

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Viktor Gaško**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Roman Parák**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Využití strojového učení pro kontrolu kvality v průmyslových aplikacích

Stručná charakteristika problematiky úkolu:

Práce bude zahrnovat rešerši knihoven pro strojové učení – hluboké učení a jejich využití v oblasti kontroly kvality v průmyslových aplikacích.
Predmětem práce bude návrh řídícího programu pro zpracování obrazových dat z kamery (např. pomocí raspberry pi). Následovat bude imeplementace návrhu, učení a verifikace strojového učení pro kamerovou kontrolu kvality výrobku. Závěr práce bude věnován implementaci návrhu a ověření funkčnosti vytvořeného řešení.
Práce předpokládá aktivní přístup studenta a nutnost práce v laboratoři.

Cíle bakalářské práce:

Seznamte se s problematikou kontroly kvality v průmyslových aplikacích. Zpracujte prehled aktuálniho stavu v dané oblasti.
Proveďte rešerši knihoven pro strojové učeni – hluboké učení a jejich využití v oblasti kontroly kvality.
Navrhněte řídící program pro zpracování obrazových dat z kamery.
Implementujte návrh, učení, verifikaci strojového učení pro kamerovou kontrolu kvality výrobku.
Ověřte funkčnost vytvořeného řešení.

Seznam doporučené literatury:

WEIGEL, Van B. Deep learning for a digital age: technology's untapped potential to enrich higher education. San Francisco: Jossey-Bass, c2002. Jossey-Bass higher and adult education series. ISBN 0-7879-5613-9.

VATSA, Mayank, Richa SINGH a Angshul MAJUMDAR. Deep learning in biometrics. Boca Raton: CRC Press, 2018.

OpenCV: Open Computer Vision library.https://opencv.org/

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Cieľom tejto bakalárskej práce je zoznámiť sa s problematikou kontroly kvality v priemyselných aplikáciách so zameraním na hlboké učenie. K tomuto a podobným problémom bolo vytvorených niekoľko knižníc, ktoré majú za úlohu uľahčiť jeho riešenie. Hlavnou úlohou je vytvorenie programu na kontrolu kvality za pomoci programovacieho jazyka Python a frameworku Tensorflow. Tento program bude pozostávať z troch neurónových sietí, pričom jedna zistí približnú polohu súčiastky, druhá jej farbu a tretia skontroluje správnosť jej výroby.

ABSTRACT

Goal of this bachelor´s thesis is to get acquainted with issue of quality control in industrial applications with focus on deep learning. For this and similar issues was created several libraries which have a purpose of simplifying these issues. Main task is to create program for quality control with help of programming language Python and framework Tensorflow. This program will be comprised of three neural network, from which one will identify the approximate position of the part, second its color, and third will check the correctness of its production.

KLÚČOVÉ SLOVÁ

neurónová sieť, konvolučná neurónová sieť, hlboké učenie, tenzor, dátový súbor, klasifikácia objektu

KEYWORDS

neural network, convolutional neural network, deep learning, tensor, dataset, object classification

BIBLIOGRAFICKÁ CITÁCIA

GAŠKO, V. *Využití strojového učení pro kontrolu kvality v průmyslových aplikacích,* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2019. Vedoucí bakalářské práce Ing. Roman Parák.

POĎAKOVANIE

Rád by som poďakoval vedúcemu práce Ing. Romanovi Parákovi za ochotu, priateľský prístup a možnosť pravidelných konzultácií aj mimo pracovného týždňa.

ČESTNÉ PREHLÁSENIE

Prehlasuj, že táto práca je mojím pôvodným dielom, spracoval som ju samostatne pod vedením Ing. Romana Paráka a s použitím literatúry uvedenej v zozname literatúry.

V Brně dne 23. 5. 2019 .....

Viktor Gaško

OBSAH

1 ÚVOD

Kontrola kvality je už od nepamäti významnou súčasťou výroby produktu, pričom jej najčastejším spôsobom je vizuálna kontrola. Pri neustále rastúcej rýchlosti výroby a automatizácii priemyslu sa posledné desaťročia upúšťa od kontroly ľudským okom, a dnes už je tento proces v takmer každej výrobnej prevádzke a odvetví automatizovaný. Za týmto účelom existuje mnoho 2D/3D senzorov a kamier, ktoré nám slúžia na nasnímanie výrobku. V prvej časti práce sa teda zameriame na 3D metódy rozpoznávania objektu a jeho kontrolu, základné typy 2D kamier a hardware, ktorý je alebo mohol byť použitý v praktickej časti zadania.

Keďže chceme aby kontrola bola automatizovaná, potrebujeme nejakú metódu na spracovanie obrazu a určenie funkčnosti kontrolovaného výrobku. Týmto algoritmom bude hlboké učenie, založené na architektúre neurónových sietí. Bližšie sa teda zameriame na základné princípy činnosti neurónových sietí od jedného neurónu až po konvolučné siete. Tieto siete sú dnes rozšírené vo väčšine odvetví spracovania dát. Vďaka ich stále rastúcej obľube je dostupných aj mnoho knižníc a frameworkov, ktoré sú spojením funkcií a príkazov programovacích jazykov a zjednodušujú proces tvorby týchto sietí.

Cieľom tejto práce je navrhnúť sieť na spracovanie vizuálnych dát a overiť jej funkčnosť pri kontrole správnosti výroby súčiastky. K jej vytvoreniu použijeme programovací jazyk python a jeden z frameworkov zmienených v krátkej rešerši.

Závere z hrnieme dosiahnuté výsledky, pričom dôraz bude kladený hlavne na presnosť pri určovaní stavu súčiastky. Zrekapitulujme výsledky počínania siete a to, či by ju bolo možné implementovať v praxi.

2 METÓDY KONTROLY KVALITY

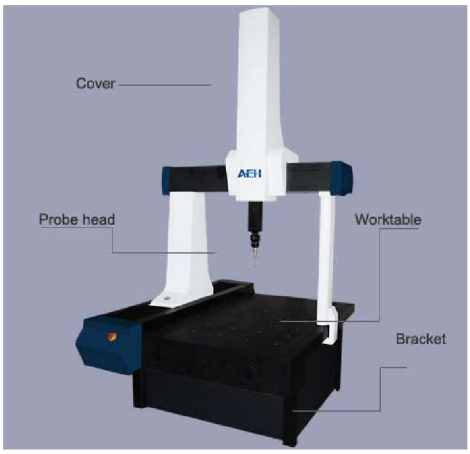
Neodmysliteľnou súčasťou akéhokoľvek priemyselného odvetvia je kontrola kvality. Môže mať viac podôb ako kontrola funkčnosti výrobku alebo kontrola zloženia materiálu, z ktorého je výrobok vyrobený. Jej najčastejšou podobou však bude vizuálna kontrola. Tou sa dá zistiť, či má výrobok správne rozmery a tiež, či na ňom nie sú nejaké vady. Dá sa vykonať pomocou 3D a 2D zariadení.

2.1 3D metódy kontroly kvality

Pri 3D kontrole sa využíva proces nazývaný 3D skenovanie. Ide vlastne o analýzu objektu, ktorej zmyslom je zistiť jeho tvar, alebo iné vlastnosti ako farbu. Zo získaných dát je možné vytvoriť 3D model. Skenery pri kontrole kvality porovnajú získané dáta s preddefinovaným modelom a na základe ich podobnosti, resp. rozdielu, vyhodnocujú stav a funkčnosť vyrobenej súčiastky. Skenovanie sa delí na viacero metód a tými sú: Kontraktné metódy, bezkontaktné aktívne a bezkontaktné pasívne metódy. Bezkontaktné aktívne metódy sú zrejme najrozšírenejšími v oblasti skenovania. Na určenie tvaru telesa využívajú rôzne druhy svetla, alebo iného žiarenia, ktoré tieto skenery sami vyžarujú. Ďalej sa môžu deliť na ďalšie kategórie, z ktorých si predstavíme dobu letu, trianguláciu, štruktúrované svetlo a počítačovú tomografiu. Pri pasívnych metódach sa zameriame na z stereoskopiu a fotometriu.

2.1.1 Kontaktné skenovanie

Pri tomto skenovaní sa mapuje povrch výrobku umiesteného na rovnej podložke pomocou senzoru umiesteného na konci kĺbového ramena, konci nehybného ramena pohybujúceho sa na koľajnicovom systéme, alebo kombinácii oboch zmienených. Typickým príkladom kontaktného skeneru je Coordinate Measuring Machine (CMM). Najbežnejším prevedením je mostový typ, ktorý ma 6 stupňov voľnosti – 3 zaisťuje polohovanie ramena a 3 rotujúca sonda na konci ramena.



Obr. 1: CMM skener

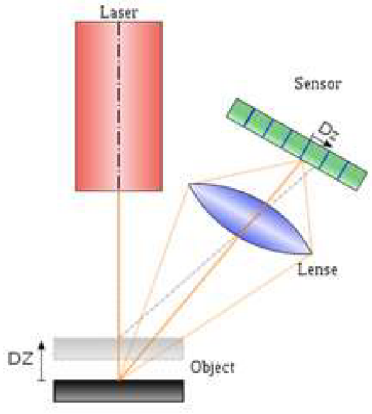
Výhodou CMM a ostatných kontaktných skenerov je vysoká presnosť, ktorá dnes môže byť dosiahnutá len vďaka kontaktu senzoru so súčiastkou. To môže byť zároveň aj nevýhodou, pretože dotyk môže poškodiť súčiastku. Hlavným problémom je veľmi nízka rýchlosť, takže tento typ skenerov nájde uplatnenie len pri kontrole, pri ktorej je kladený vysoký dôraz na presnosť. [1]

2.1.2 Doba letu

Skener ktorý zisťuje tvar objektu pomocou doby letu svetla je v najjednoduchšej podobe zariadenie, ktoré sa skladá zo zdroju svetla, ktorým je laser alebo LED dióda, optického systému na zachytenie svetla, a senzoru ktorý toto svetlo prijíma. Objekt je v pravidelných intervaloch ožarovaný svetlom ktoré má istú vlnovú dĺžku a je poslané zo zdroju svetla. Pomocou rýchlosti svetla, fázového posunu a frekvencie je možné spočítať vzdialenosť objektu. Po nasvietení z viacerých uhlov získame tvar celého objektu. Nevýhodou je nižšia presnosť ktorá je spôsobená kombináciou vysokej rýchlosti svetla a teda nepresným meraním doby jeho letu, a zlým rozlíšením tof kamier (time-of-flight - kamery so svetelným senzorom), ktoré sú často limitované na 320x240 pixelov, a teda presnejšie skenovanie vyžaduje väčšie množstvo polôh a umiestnenie bližšie k objektu, alebo spojenie zistenej hĺbky s vysokým rozlíšením klasickej kamery. Kvôli nepresnosti spôsobenej dobou letu svetla sa uplatňuje hlavne na skenovanie väčších objektov ako sú väčšie časti lodí, lietadiel. [2][3]

2.1.3 Triangulácia

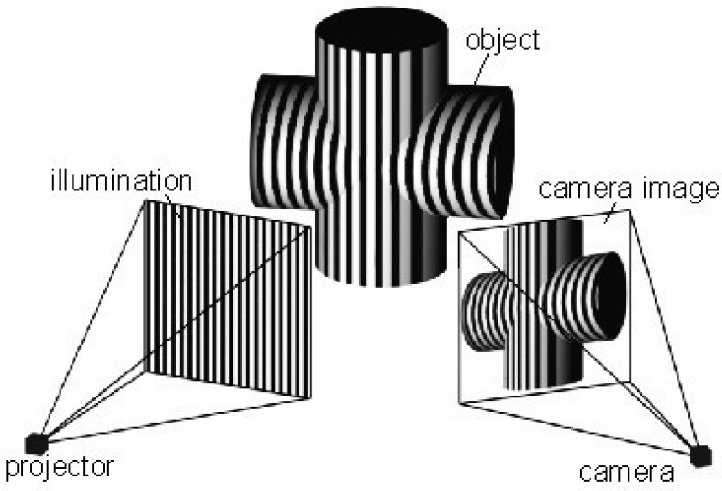
Je skenovacia metóda, pri ktorej kamera spracováva laserové svetlo odrazené od skenovaného objektu. Ide teda o systém tvorený zdrojom laserového svetla a spomínanou kamerou a skenovaným objektom. Vzdialenosť povrchu objektu sa zisťuje podľa uhlu ktorým sa odráža svetlo do kamery. Táto vzdialenosť je spočítateľná pomocou trigonometrie ak poznáme vzdialenosť medzi kamerou a zdrojom svetla. Táto metóda je oproti doby letu pomalšia, keďže zachytáva omnoho menšiu oblasť. Tento nedostatok sa dá odstrániť použitím líniového laseru. Výhodou, aj pri bodovom lasery, je vysoká presnosť. [4][5]



Obr. 2: Triangulácia[6]

2.1.4 Štruktúrované svetlo

Štruktúrované svetlo je najpopulárnejšou skenovacou technikou v odvetví priemyselnej kontroly kvality. Jej základy sú v metóde triangulácie. Skener zachytáva deformáciu svetelného vzoru premietaného na teleso, a podľa toho zisťuje jeho tvar. Svetelný vzor môže mať viacero tvarov ako vertikálne čiary, horizontálne čiary, tvar mriežky alebo samostatné body. Pri vzore vytvorenom umelým bielim alebo žltým svetlom môže nastať šum z okolitého svetla, takže je vhodné toto svetlo potlačiť. Ďalšiu možnosťou je použiť modré alebo infračervené svetlo. Premietané svetlo môže byť pri rôznych typoch tohto skenovania zachytené jednou alebo dvoma kamerami/senzormi. Hlavnou výhodou tejto metódy je jej rýchlosť – niektoré skenery sú schopné naskenovať pohybujúce sa objekty v reálnom čase. [5][7]



Obr. 3: Štruktúrované svetlo[8]

2.1.5 Počítačová tomografia

Počítačová tomografia (CT – computer tomography) je hlavným zástupcom časti bezkontaktných aktívnych metód, nazývanej volumetrické metódy. Pri nich sa okrem povrchu skenuje aj vnútro výrobku. Aj keď sa počítačová tomografia uplatňuje hlavne v medicíne, je použiteľná aj pri skenovaní objektov vyrobených v priemysle. Je možné ju použiť na kontrolu odliatkov alebo elektroniky umiestnenej vo vnútri výrobku. 3D model kontrolovanej súčiastky je vytvorený z jednotlivých 2D Röntgenových snímok prierezov. [5]

2.1.6 Bezkontaktné pasívne metódy

Oproti aktívnym metódam je jediný rozdiel v tom, že pasívne metódy využívajú svetlo z okolia a nemajú vlastný svetelný zdroj. Toto svetlo nemusí byť len viditeľné, môže sa pracovať napríklad aj s infračerveným. Hlavnými predstaviteľmi sú metódy stereoskopie, fotometrie a metóda siluety.

- **Stereoskopia -** metóda založená na rovnakom princípe ako ľudské videnie. Využívajú sa teda 2 kamery ktoré sú umiestnené neďaleko od seba a ich zorné polia sa z časti prekrývajú. Analýzou malých rozdielov medzi jednotlivými

obrazmi je možné zmerať vzdialenosť objektu od kamier, a teda vytvoriť jeho povrch. Keďže poznáme vzdialenosť medzi jednotlivými kamerami, je vzdialenosť, a teda tvar povrchu, počítaný pomocou trigonometrie.

- **Fotometria -** využívaná sa len jedna kamera, ktorá vyfotí objekt pri rôznych svetelných podmienkach. Tento snímkový model je potom invertovaný, vďaka čomu sa získajú informácie o tvare povrchu na každom pixely kamerového senzoru.

Metóde siluety sme si neprestavili, pretože nedokáže rozpoznať tvar vnútrajšku nádoby, čo je výrazným nedostatkom. Tieto metódy sú v porovnaní s aktívnymi metódami menej používané. Výhodou je však ich nízka cena, pretože vo väčšine prípadov skener tvorí iba jedna až dve digitálne kamery. [5] [7]

2.2 2D kontrola kvality a digitálne kamery

Tam kde nie je potrebná, vo valnej väčšine prípadov výrazne drahšia 3D kontrola, nastupujú 2D kontrola. Základom tejto kontroly je kamera, pozostávajúca zo svetelného senzoru a objektívu, a svetelný zdroj pre prípad nedostačujúceho osvetlenia. Už v minulej kapitole sme si mohli všimnúť, že takmer všetky typy 3D skenerov využívajú jednu alebo viac digitálnych kamier. Pri 2D kontrole sa dá samozrejme taktiež očakávať využitie týchto kamier. Digitálne kamery v priemysle môžeme rozdeliť na viacero kategórií.

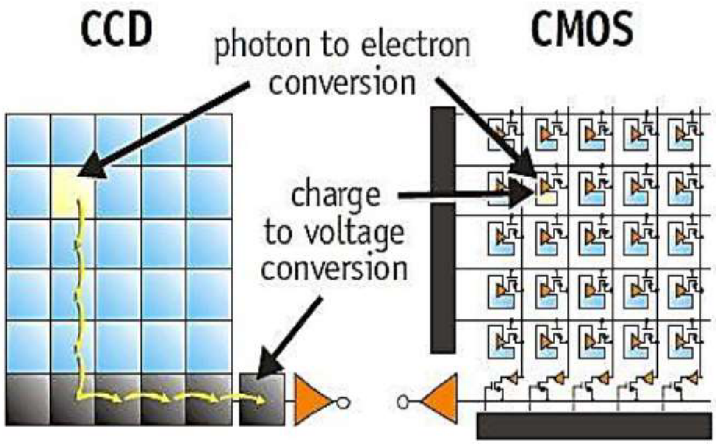
Jedno z delení môže byť podľa schopnosti vyhodnotenia obrazu na klasické a chytré. Zatiaľ čo pri klasických kamerách bude musieť byť obraz privedený do iného zariadenia kde sa spracuje, pri chytrých je hardware a software potrebný na túto operáciu priamo v kamere.

Ďalšie rozdelenie môže byť podľa prevedenie senzoru na 1D a 2D. Pri 1D senzore je snímaná iba jedna línia a kontrola môže prebiehať tak, že pod kamerou je umiestnená linka po ktorej cestuje výrobok. Takže aj z jeden línie sa dá pri prechode celého výrobku zostaviť jeho tvar. 2D senzor je klasickejšia možnosť, vyskytujúca sa vo väčšine kamier. Bližšie sa teda zameriame na 2D senzory. Prestavíme si kremíkové CCD a CMOS senzory, z ktorých jeden dnes nájdeme takmer v každej digitálnej kamere.

2.2.1 CDD a CMOS senzor

Ako je možné vidieť na obr. 4, obi dva senzory sú zložené z jednotlivých pixelov uložených vedľa seba. V obi dvoch prípadoch sa náboj, z ktorého nakoniec získame obraz, tvorí tak, že pri dopade svetla (fotónov) na pixel je toto svetlo premenené na istý počet elektrónov. Tento počet odpovedá intenzite svetla.

Pri CCD (Charged Coupled Device) senzoroch sa náboje získané v jednotlivých pixeloch postupne zvádzajú na kraj senzoru k jeho výstupu. Tu sú vedené do zosilňovaču a prevedené na napätie. Toto napätie je potom zmerané a je možné zrekonštruovať zachytenú scénu. CCD senzory oproti CMOS senzorom dokážu vytvoriť kvalitnejší obraz, ale kvôli spôsobu zvádzania náboja sú energeticky náročnejšie a pomalšie.

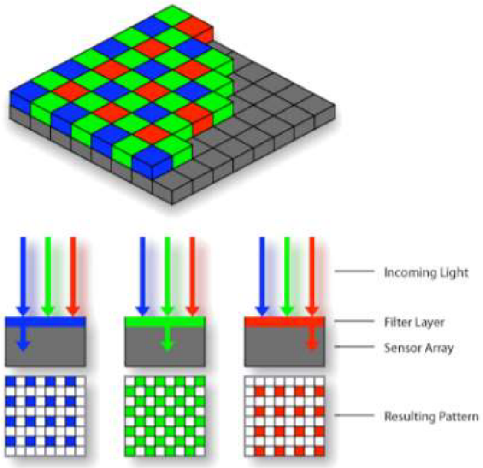


Obr. 4: CCD a CMOS senzor

Pri CMOS (Complementary metal–oxide–semiconductor) je získaný náboj je prevedený na napätie a zosilnení priamo v pixeli. Toto napätie je potom vedené na výstup senzoru a ďalej je opäť možné zrekonštruovať obraz. Ako bolo spomenuté CCD senzory dokážu vytvoriť o niečo kvalitnejší obraz, ale tento rozdiel sa postupne stráca. Dôležitá je hlavne rýchlosť a energetická úspora, a preto stále viac kamier a fotoaparátov využíva CMOS senzory. Vďaka ich rýchlosti je s nimi možné natáčať vysokorýchlostné videá. CMOS senzor využíva aj Raspberry Pi kamera. [9][10]

2.2.2 Bayerov filter

Senzory ktoré sme si teraz predstavili by nám neboli schopné rekonštruovať farebný obraz bez použitia istého filtra. Najpoužívanejším farebným filtrov je Bayerov filter.



Obr. 5: Bayerov filter[11]

Aby mohol byť obraz farebný je každý pixel zakrytý jednou z troch farieb. Filter prepúšťa takú farbu svetla, akú má on sám, takže modré svetlo prejde cez modrý filter atď. Základná veľkosť filtru je 2x2 pixelov a v tejto časti sú dva pixeli zakryté zelenou častou filtru, jeden modrou a jeden červenou. Vzor sa ďalej opakuje. Bayerov filter má niekoľko modifikácií akými je napríklad náhrada jedného zeleného pixelu belasým alebo bielim.[8]

2.3 Spoločnosti zaoberajúce sa kontrolou kvality

Výrobou skenerov a kamier na kontrolu kvality sa zaoberajú aj veľké spoločnosti ako Sony, Panasonic, Intel atď. My si predstavíme menšie spoločnosti Photoneo a Sanezoo.

2.3.1 Photoneo

Je firmou založenou na Slovensku 24. 8. 2013. Zameriava sa hlavne na výrobu 3D skenerov, z ktorými sú spojené aj ďalšie produkty ako zariadenia na bin picking alebo mobilný transportný robot. Od svojho vzniku sa rozšírili do iných štátov ako Nemecko a Čína. Firma má patentovanú technológiu paralelného štruktúrovaného svetla (Paralel structured light) vďaka ktorej vyhrali prestížne ocenenie Vision award 2018, ktoré sa udeľuje za inovácie v odvetví strojového videnia. Táto technológia založená na CMOS senzoroch umožňuje paralelne zachytiť a spracovať niekoľko snímok naraz, a vďaka čomu je možné skenovať objekt pri vysokých rýchlostiach až do 40 m/s. Keďže je táto technológia patentovaná, nachádza sa teda len v skeneroch tejto firmy.[12]

Obr. 6: Photoneo MotionCam 3D

2.3.2 Sanezoo

Je pomerne novou startupovou Brnenskou spoločnosťou zaoberajúcou sa kontrolou kvality pomocou chytrých 2D kamier. V ich systémoch pre strojové videnie sa využívajú rôzne algoritmy umelej inteligencie, strojového a hlbokého učenia. Tieto algoritmy potom pracujú na embedded (zamerané na istú úlohu) počítačoch, ktoré umožňujú veľmi rýchle spracovanie obrazu. Software je naprogramovaný tak, aby mali kamery schopnosť rýchlo sa adaptovať na nové požiadavky vo výrobe. [13]

3 HARDWARE SÚVISIACI S PRÁCOU

V tejto krátkej kapitole si prestavíme počítač Raspberry Pi a produkty rady Nvida Jetson.

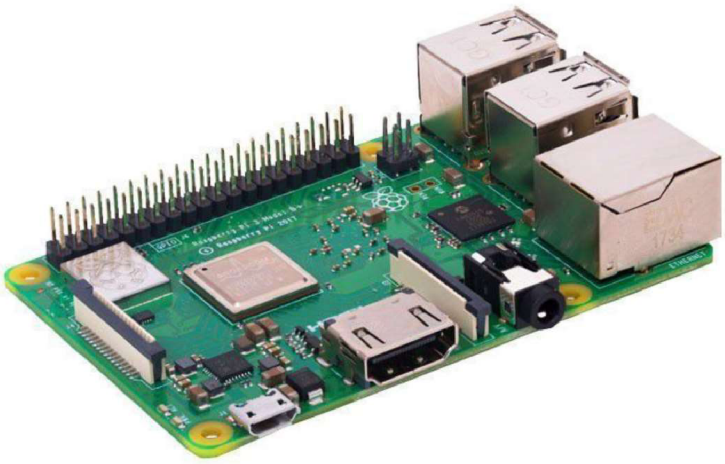
3.1 Raspberry Pi

Ide o malý jedno doskový počítať vyvinutý Raspberry Pi Foundation vo Veľkej Británii za účelom učenia základov programovania. Počas histórie projektu bolo vytvorených niekoľko modelov, pričom jeden z najnovších a ten, ktorý používame v tejto práci je model Raspberry Pi 3 B+. Jeho technické špecifikácie sú nasledovné:

- Procesor: - Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
- Pamäť: - 1GB LPDDR2 SDRAM
- Konektivita: - 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac bezdrôtový LAN, Bluetooth 4.2, BLE
  - Gigabit Ethernet cez USB 2.0 (max. throughput 300Mbps)
  - 4 × USB 2.0 porty

Okrem týchto konektorov sa tu nachádzajú 2 MIPI konektory, DSI pre displej a CSI pre kameru, HDMI konektor a 4-kontaktný TRRS konektor. Ďalej doska podporuje aj micro SD karty. Napájanie je možné cez micro USB konektor, na ktorý je možné pripojiť jednosmerné napätie o veľkosti 5 V.

Dosky pracujú pomocou operačného systému Raspbian založeného na Unixovom operačnom systéme Debian. [14]

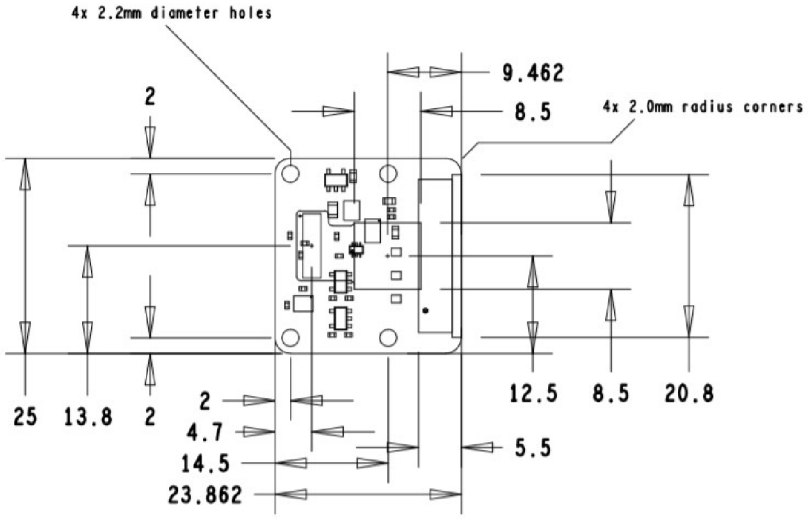


Obr. 7: Raspberry Pi model 3 B+

3.1.1 Kamera Raspberry Pi

Kamerou ktorú využívame aj v tejto práci je 8-megapixelový kamerový modul v2 vydaný v roku 2016. Oproti predchádzajúcej verzií má o 3 megapixeli vyššie rozlíšenie. Je k dispozícii v dvoch verziách a to so senzorom viditeľného svetla a infračerveného svetla.

Použitým senzorom je Sony IMX219 s rozlíšením 3280 x 2464 pixelov. Video je možné natáčať v módoch 1080p30, 720p60 a 480p60/90. Pre prípady horšieho osvetlenia kamera podporuje 2x2 binning, čo je vlastne spojenie 4 pixelov na 1 a tým zvýšená citlivosť kamery na úkor polovičného rozlíšenia, a teda 4-krát menšieho obrazu. Kamera je k doske pripojovaná pomocou flex káblu.



Obr. 8: Rozmery kamery Raspberry pi v2

Obr. 9: Raspberry pi v2 kamera pre viditeľné svetlo

Kameru je možné programovať pomocou jazyku python a knižnice picamera určenej na prácu s ňou. Ďalšou možnosťou je používanie pomocou príkazového riadku Linuxu. [14]

3.1.2 Rozdielné verzie Raspberry Pi

Okrem triedy B je dostupná aj trieda A, ktorá je menšia, a preto má vo verzii 3+ o niečo horšie parametre. Rozdielom je polovičná pamäť RAM (512 mB), jeden USB port namiesto štyroch a žiadny ethernet port. Stále sú dostupné aj niektoré predchádzajúce

modely triedy A a B ako model 3 B, 2 B, 1 B+ a 1 A+. Poslednými dostupnými modelmi sú Raspberry Pi Zero a Zero W, ktoré sú low-cost verziou Raspberry. [14]

3.2 Nvidia Jetson

Je rada zariadení určených špeciálne pre prácu s neurónovými sieťami. Na týchto zariadeniach je možné paralelne pracovať s viacerými sieťami naraz. Silnou stránkou týchto zariadení je ich spotreba energie, ktorá sa pohybuje od 5 w pre najslabšie zariadenie, po 20 W pre najsilnejšie.

Pre nás najzaujímavejším produktom je Nvidia Jetson Nano Developer Kit. Ide o najlacnejší produkt tejto série, s cenou okolo 99 dolárov, so schopnosťou spracovať živý obraz na niekoľkých populárnych modeloch sietí. Plusom je aj spomínaná najnižšia spotreba 5 W. Zaradenie disponuje asi dobrou konektivitou zahrnujúcou niekoľko USB portov, HDMI, ethernet atď. Do zariadenia je možné vložiť aj micro SD kartu so snímkami a sieťou, s ktorými je možné po nabootovaní vývojárskeho balíčka pracovať.

Obr. 10: Jetson Nano Developer Kit

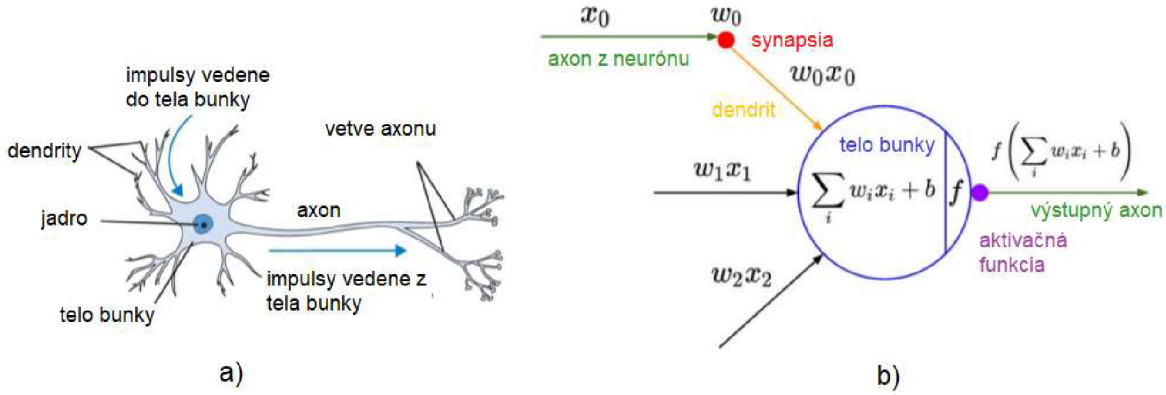
Okrem tohto zariadenia je na oficiálnej stránke dostupných ďalších sedem produktov s cenami pohybujúcimi sa do 1300 dolárov. Tieto zariadenia môžu byť použité napríklad na autonómne riadenie a podobné aplikácie. [15]

4 NEURÓNOVÁ SIEŤ

Hlboké učenie je častou strojového učenia založenou na sieťach pozostávajúcich z vrstiev vzájomne pospájaných jednotiek – neurónov. V tejto kapitole sa pozrieme na základy neurónových sieti a konvolučných neurónových sietí, ktoré sú ich modifikáciou obzvlášť vhodnou na klasifikáciu snímok.

4.1 Neurón

Neurón je základným stavebným kameňom neurónovej siete. Princíp jeho činnosti je často prirovnávaný k činnosti biologického neurónu. V prípade biologického neurónu je cez dendrity v podobe impulzu privedená informácia, ktorá je v bunke spracovaná, a potom poslaná na axon – výstup neurónu.



Obr. 11: a) biologický neurón, b) umelý neurón [16][17]

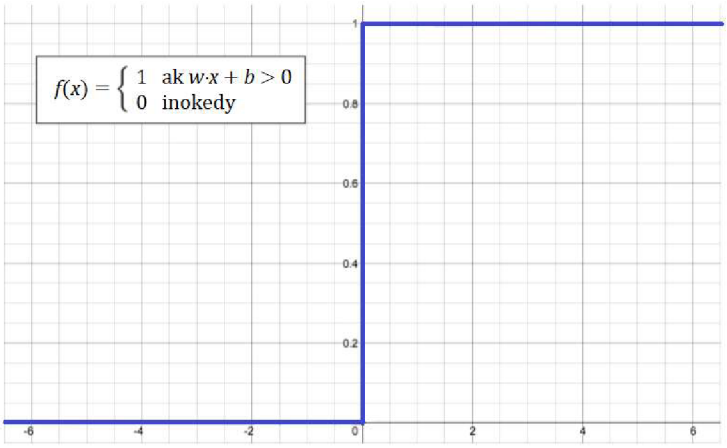
Zatiaľ však nie je úplne známe, čo sa deje v tele neurónu, teda ako je informácia spracovaná a uložená. Naproti tomu princíp činnosti umelého neurónu je vcelku jednoduchý. Na každú z n synapsii (spoj 2 neurónov) prichádza signál buď z axonov neurónov predchádzajúcej vrstvy, alebo priamo zo vstupných dát. Na synapsii je signál vynásobený váhou w ktorá je odlišná pre každý dendrit – vstup. V bunke je potom hodnota z každého vstupu sčítaná, a k tejto sume môže byť pričítaný bias b. Tým sa dostávame na výstup, kde je sa táto hodnota stáva neznámou v aktivačnej funkcii. Konečný tvar pre výstupu pre j-tý neurón je teda: [18][19]

$$y_j = f\left(\sum_i w_{ji} \cdot x_i + b\right) \tag{3.1}$$

4.2 Aktivačná funkcia

Táto funkcia určuje kedy neurón pošle signál a kedy nie. Môže byť lineárna alebo nelineárna. Nelineárne funkcie musia byť diferencovateľné, inak by nemohli fungovať

behom procesu učenia. Prvou aktivačnou funkciou bola skoková funkcia, ktorú využíval tzv. perceptron, ktorý bol navrhnutý Frankom Rosenblattom v roku 1957. Ide o najjednoduchší model neurónu, na výstupe ktorého je skoková funkcia. Tá naberá podľa toho či je súčet sumácie vstupov a biasu záporný alebo kladný buď hodnotu 0 alebo 1. [18][19]



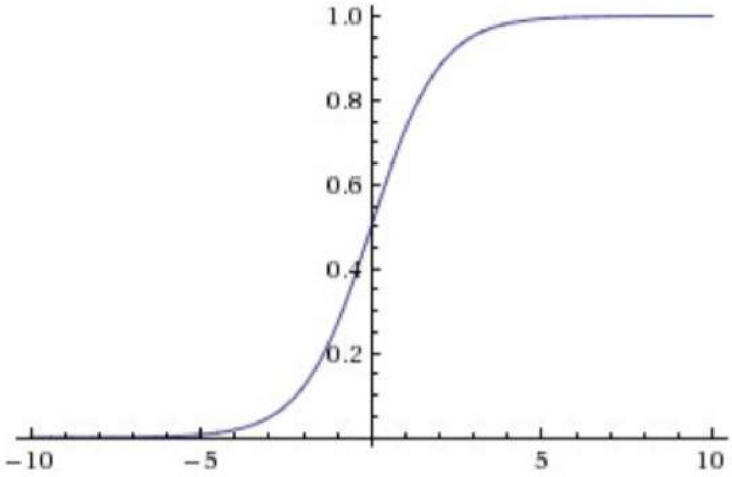
Obr. 12: Skoková funkcia percepotronu

4.2.1 Sigmoid

Je to nelineárna funkcia, ktorej výsledok sa pohybuje medzi hodnotou 0 a 1. Má tvar:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Pri hodnotách x v okolí nuly môžeme pozorovať jej rýchly rast, a teda aj veľkú deriváciu. To je vhodné pri spätnom šírení, ktoré túto deriváciu využíva na učenie siete. Pri vyšších hodnotách x však nastáva problém, keďže gradient je takmer nulový a učenie môže výrazne spomaliť. [18][19]

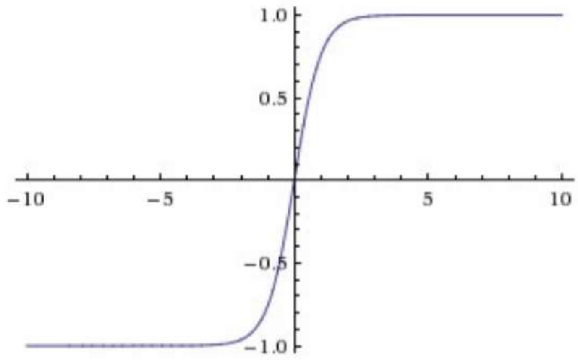


Obr. 13: Funkcia sigmoid

4.2.2 Tanh

Tvar tejto funkcie je podobný tvaru sigmoidu, s tým rozdielom, že naberá hodnoty od -1 do 1. Vo väčšine prípadov kde sa používal sigmoid ho táto funkcia nahradila, pretože je o niečo strmší, a tým pádom by mal byť pri učení efektívnejší. Stále je tu však problém malého gradientu v oblastiach ďalej od nuly. [18][19]

$$Tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.3}$$



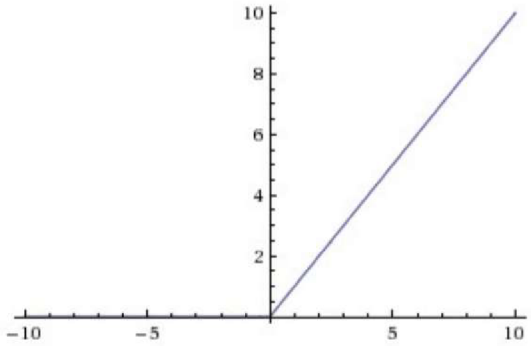
Obr. 14: Funkcia Tanh

4.2.3 ReLU

Rectified linear unit (usmerňujúca lineárna jednotka) je dnes zrejme najpoužívanejšia aktivačná funkcia. Má veľmi jednoduchý tvar:

$$y = \max(0, x) \tag{3.4}$$

Vďaka tomu že je lineárna, nenastáva problém malého gradientu ako v prípade sigmoidu a Tanh. Ďalšou výhodou je to, že je nenáročná na výpočet. Jej jediným negatívom je strata záporných hodnôt x. Má mnoho modifikácií ako napríklad Leaky ReLU a Parametric ReLU. Tieto funkcie riešia aj spomenutý nedostatok, keďže pri zápornom x naberajú hodnôt $0{,}01 \cdot x$ resp. $a \cdot x$. [18][1]



Obr. 15: Funkcia ReLU

4.2.4 Softmax

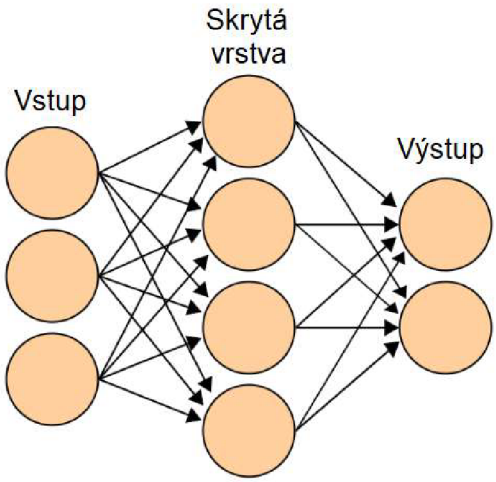
Softmax počíta rozdelenie pravdepodobnosti z vektoru reálnych čísel. Ide vlastne o rozdelenie čísel o veľkosti 0 až 1 medzi jednotlivé neuróny vo vrstve v ktorej je táto funkcia použitá. Suma týchto pravdepodobností sa rovná 1. Jej tvar je:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3.5}$$

Používa sa vo viac triednych sieťach, kde na výstupe vráti pravdepodobnosť každej triedy, pričom vybraná trieda má najväčšiu hodnotu. Softmax sa teda používa vo väčšine neurónových sieťach vo výstupných vrstvách. Hlavným rozdielom medzi sigmoidom a softmaxom je ten, že sigmoid slúži len na binárnu klasifikáciu, zatiaľ čo softmax sa používa pri viacrozmernej klasifikácií. [18][19]

4.3 Model neurónovej siete

Neurónová sieť vzniká spojením prepojením vstupu a výstupu aspoň jednou tzv. skrytou vrstvou neurónov, pričom výstup každého neurónu je poslaný na vstup každého neurónu v ďalšej vrstve (v zložitejších sieťach to nemusí vždy platiť). Všetky vrstvy takto vzájomne prepojených neurónov (vrátane vstupu a výstupu) sa označujú fully-connected, teda plne prepojené vrstvy.



Obr. 16: Jednoduchá sieť s jednou skrytou vrstvou

V dnešnej dobe sa používajú viacvrstvé neurónové siete ktoré môžu mať aj stovky skrytých vrstiev. Takéto siete sú nazývané hlboké neurónové siete. V každej vrstve sa môže nachádzať rozdielny počet neurónov a rôzne aktivačné funkcie v závislosti na voľbe užívateľa. [19]

4.3.1 Parametre a hyperparametre siete

Hodnoty ktoré určujú ako sa bude sieť správať sa delia na parametre a hyperparametre. Rozdiel medzi nimi je v tom, že hyperparametre sú volené užívateľom. Ide vlastne o veličiny ktoré ovplyvňujú topológiu siete a to ako sa bude učiť. Medzi hyperparametre ktoré musí mat zadefinované každá sieť počet vrstiev siete, počet neurónov vo vrstve, aktivačné funkcie, počet iterácií pri učení, rýchlosť učenia α. Ďalšími môžu byť tie, ktoré nie sú použité v každom modely, ako percento zníženia počtu neurónov pri aplikácií Dropoutu – techniky na náhodné odstraňovanie zvoleného percenta neurónov vo vybranej vrstve. Táto technika zabraňuje javu nazývanému overfitting. Ide o príliš. presnú úpravu váh, pri ktorej sieť bude účinná len na súbore dát na ktorom sa učila.

Parametrami siete sú hodnoty váhy a biasu, ktoré sa menia pri učení a teda ich hodnoty nie sú nastaviteľné užívateľom.

4.4 Učenie neurónových sietí

Veľká väčšina neurónových sieti využíva k učeniu algoritmus nazývaný backpropagation – spätné šírenie. Algoritmus pracuje z chybou na výstupe, podľa ktorej sa upravujú parametre siete. To znamená, že je potrebné označiť vstupné dáta, aby algoritmus vedel k akým výsledkom sa má dopracovať. Parametre siete sú upravované tak, aby chyba bola čo najmenšia. Pre vysvetlenie princípu algoritmu použijeme dva vzťahy, jeden z nich bude pre dobrú názornosť pri derivovaní vzťah funkcie sigmoid. Tieto vzťahy teda sú:

$$y_j(w,x) = \sum_i w_{ji} \cdot x_i \tag{3.6}$$

$$\sigma_j(x) = \frac{1}{1 + e^{y(w,x)}} \tag{3.7}$$

Cieľom trénovania je získať pri danom vstupe požadovaný výstup. Za týmto cieľom definuje chybovú funkciu pre výstup každého neurónu. Táto funkcia sa nazýva error alebo cost function a má tvar (existujú aj iné druhy tejto funkcie):

$$E_j(w,x,d) = (\sigma_j(w,x) - d_j)^2 \tag{3.8}$$

Kde *d* je chyba na výstupe neurónu. Funkcia je na druhú, pretože chceme aby jej výsledky boli pozitívne a taktiež je veľká chyba zväčšená a malá zmenšená. Tým pádom má odstránenie veľkých chýb pre algoritmus väčší význam a učenie je efektívnejšie. Ďalej budeme pracovať s chybou celej siete:

$$E(w,x,d) = \sum_j (\sigma_j(w,x) - d_j)^2 \tag{3.9}$$

Teraz môžeme upraviť váhy pomocou metódy gradientového zostupu:

$$\Delta w_{ij} = -\alpha \cdot \frac{\partial E}{\partial w_{ij}} \tag{3.10}$$

Teda prírastok váhy sa rovná zápornej hodnote rýchlosti učenia α vynásobenej závislosťou predchádzajúcej váhy a chyby siete. Vzťah je násobený rýchlosťou učenia u ktorej sa používajú hodnoty 0 až 1 (často napríklad $1 \cdot 10^{-2}$ a podobne), aby sa sieť neupravila len presne pre dané vstupy a u ostatných by bola prakticky nepoužiteľná. Pre funkčnosť algoritmu potrebujeme teda len deriváciu chyby siete vzhľadom k danej váhe. Najprv teda vypočítame, ako moc závisí chyba siete na výstupe neurónu deriváciou (3.8) podľa výstupu:

$$\frac{\partial E}{\sigma_j} = 2 \cdot (\sigma_j - d_j) \tag{3.11}$$

Teraz vypočítame, ako závisí výstup na aktivačnej funkcií (ktorá závisí na váhe). Použijeme skĺbenie vzťahu (3.6) a (3.7):

$$\frac{\partial \sigma_j}{\partial w_{ij}} = \frac{\partial \sigma_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = \sigma_j \cdot (1 - \sigma_j) \cdot x_i \tag{3.12}$$

Z (3.11) a (3.12) vyplýva:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \sigma_j} \cdot \frac{\partial \sigma_j}{\partial w_{ij}} = 2 \cdot (\sigma_j - d_j) \cdot \sigma_j \cdot (1 - \sigma_j) \cdot x_i \tag{3.13}$$

A spojením (3.10) a (3.13) získame pre každú váhu

$$\Delta w_{ij} = -2 \cdot \alpha \cdot (\sigma_j - d_j) \cdot \sigma_j \cdot (1 - \sigma_j) \cdot x_i \tag{3.14}$$

Vzťah (3.14) by sme boli schopný uplatniť v sieti s dvoma vrstvami. Pre sieť s tromi vrstvami musíme spraviť pár zmien. Pokiaľ chceme zmeniť hodnotu váh v_{ik} predchádzajúcej vrstvy, musíme najprv počítať závislosť chyby nie na váhach, ale na vstupe predchádzajúcej vrstvy. Pri tejto operácií by stačilo zameniť x_i s w_{ji} v (3.12), (3.13) a (3.14). My však potrebujeme zistiť aj ako závisí chyba na zmene váhy v_{ik}. Takže:

$$\Delta w_{ik} = -\alpha \cdot \frac{\partial E}{\partial w_{ik}} \cdot \frac{\partial x_i}{\partial w_{ik}} \tag{3.14}$$

Kde:

$$\frac{\partial E}{\partial w_{ik}} = 2 \cdot (\sigma_j - d_j) \cdot \sigma_j \cdot (1 - \sigma_j) \cdot w_{ji} \tag{3.15}$$

A pri predpoklade že existujú vstupy u_k do neurónu s váhou v_{ik} (zo vzťahu (3.12)):

$$\frac{\partial x_i}{\partial v_{ik}} = x_i \cdot (1 - x_i) \cdot v_{ik} \tag{3.16}$$

Pri pridaní ďalšej vrstvy postupujeme rovnako, a teda spočítame ako chyba závisí na vstupe a váhach prvej vrstvy.

Nevýhodou spätného šírenia je, že pri gradientovom zostupe nemusí byť nájdené absolútne minimum ale iba lokálne. Z tohto stavu sa tento algoritmus nedostane a teda sieť nikdy nebude tak efektívna ako by mohla a bolo by potrebné ďalšie pretrénovanie ktoré tiež nezaručuje zlepšenie. Dnes existujú algoritmy ako AdaGrad, RMSProp a ich veľmi efektívna kombinácia Adam ktoré využívajú gradientu vypočítaného spätným šírením a tým odstraňujú tento problém a zlepšujú efektivitu učenia. [20]

4.5 Konvolúcia

Konvolučné neurónové siete (CNN) sú siete pre prácu s dátami ktoré majú mriežkovú topológiu. Ide napríklad o časové dáta ktoré sa dajú previesť do 1-D mriežky, alebo snímky reprezentované 2-D mriežkou. Názov dostali podľa lineárnej operácie nazývanej konvolúcia. Vysvetlenie konvolúcie je nasledovné: Senzorom sa snažíme zistiť polohu nejakého bodu *x(t)* v čase *t*. Signál senzoru však môže byť rušivý a preto nie vždy získame presnú polohu. Z toho je pre nás lepšie spriemerovať niekoľko meraní. Kvôli neustále sa meniacej polohe bodu sú pre nás dôležitejšie novšie merania, teda chceme aby na ne bola kladená väčšia váha. Preto použijeme funkciu *w(a)* kde *a* je čas, kedy bolo meranie spravené. Po aplikácií tejto funkcie na všetky merania získame funkciu:

$$s(t) = \int x(a)w(t-a)da \tag{3.18}$$

Táto operácia sa nazýva konvolúcia. Obvykle je zapísaná v tvare:

$$s(t) = (x * w)(t) \tag{3.19}$$

V terminológií konvolučných sietí je *x* vstupom a funkcia *w* sa nazýva kernel. Výstup sa obvykle nazýva mapa rysov - feature map. Pri práci s dátami však nie je možné získať dáta v ľubovoľnom čase, ale len v istom časom intervale. Takže budeme predpokladať že *x* a *w* sú definované iba na časovom indexe *t*, čím získame tzv. diskrétnu konvolúciu:

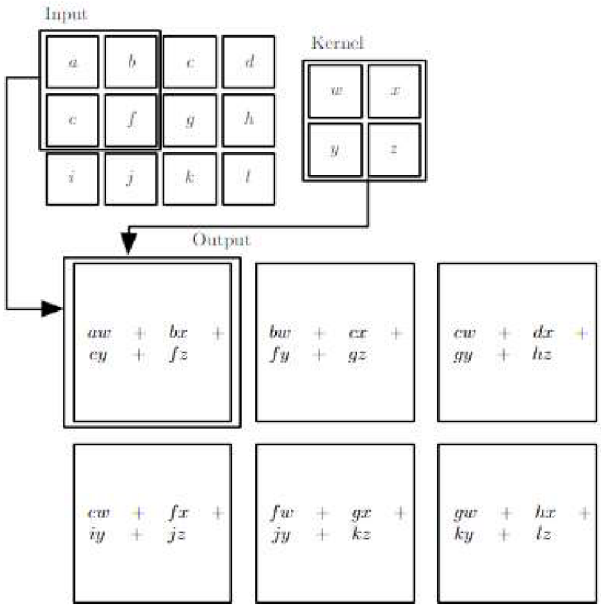
$$s(t) = (x * w)(t) \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.20}$$

V aplikáciách strojového učenia sú vstupy aj kernely viacdimenzionálne matice nazývané tensory. Konvolúciu však často používame na viac ôs. Pre príklad vstup *x* nahradíme dvojrozmerným vstupom *I*, a taktiež na tento rozmer zmeníme aj kernel čím získame *K*:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.21}$$

Keďže je konvolúcia komutatívne, môžeme prevrátiť poradie vstupu a kernelu. Táto kombinácia je jednoduchšia na implementáciu z dôvodu menšej odchýlky v rozsahu *m* a *n*. Namiesto zmenenia poradia sa dnes používaná funkcia nazývaná cross-correlation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.22}$$

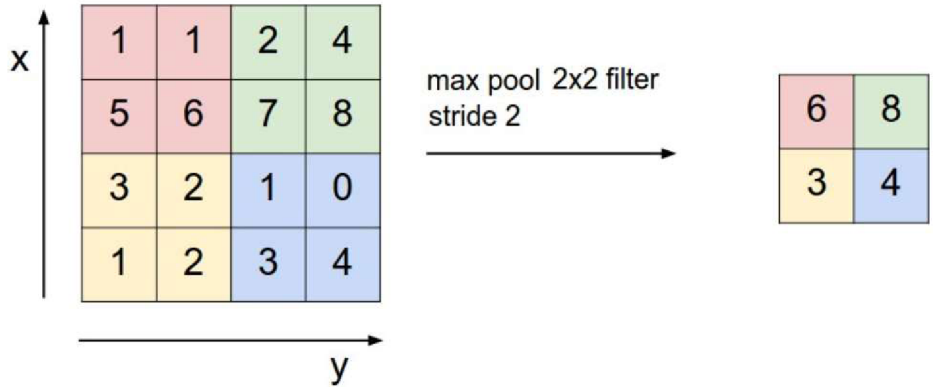


Obr. 17: 2-D príklad konvolúcie

Ako je možné vidieť váhy v kernely sú použité na celý vstup. V každej konvolučnej vrstve môže byť použitých *n* kernelov ktorých váhy sa počas učenia upravujú aby zachytili rôzne hrany alebo farby a tým umožnili sieti určiť správny výstup. Výstupom kernelu je lineárnu aktivácia, ktorá ďalej musí prejsť cez aktivačnú funkciu. Nakoniec je výstup tejto funkcie vo väčšine prípadov prevedený cez združovaciu vrstvu - pooling layer. [21]

4.6 Združovacia vrstva

Združovacia vrstva – pooling layer, je vlastne filtrom ktorý slúži na zredukovanie veľkosti vstupu – výstupu z aktivačnej funkcie, a tým zjednodušenie a zrýchlenie výpočtu. Pri poolingu sa podľa istého kritéria spája časť výstupov. Pri tejto operácií sa strácajú dáta ale sieť väčšinou nepotrebuje na pixel presnú polohu hľadaného objektu na snímke, ale skôr základné črty. Dnes sa používajú dva druhy poolingu a to max pooling a average pooling. Prvý zmienený je možné vidieť na obr. 18. Z oblasti na ktorú bol uplatnený vyberá maximálnu hodnotu. Average pooling pracuje na rovnakom princípe, akurát namiesto maxima vyberá priemer z danej oblasti. [21]



Obr. 18: Max pooling[22]

4.7 CNN

CNN je definovaná ako sieť v ktorej je aspoň jedna konvolučná vrstva. To znamená že okrem tejto vrstvy sú v nej aj klasické vrstvy, nazývané fully-connected, ktoré sú umiestnené pri výstupe, vďaka čomu nedostávame dvojrozmerný obrazový výstup ktorý by sme dostali z konvolúcie. Konvolučné vrstvy majú oproti klasickým naviac niekoľko hyperparametrov.

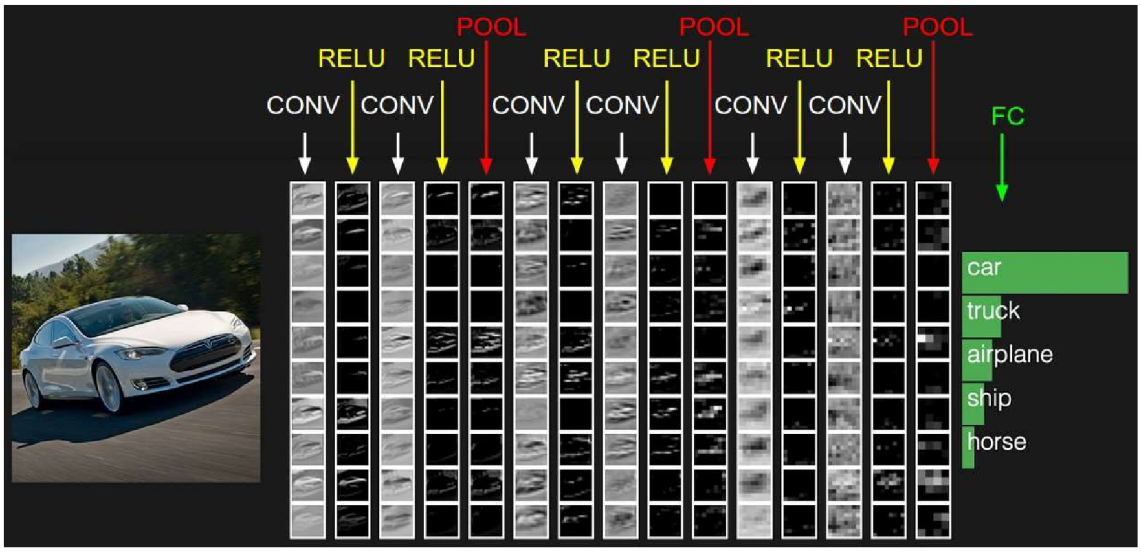
4.7.1 Hyperparametre CNN

Vo fully-connected vrstvách sú jedinými hyperparametrami určujúcimi veľkosť siete počet vrstiev a počet neurónov v nich. V konvolučných a pooling vrstvách sú hyperparametrami rozmer kernelu/filtru, počet filtrov, skok - stride, a orámovanie vstupného snímku/feature map. Pre presnejší popis použijeme obr. 17 a 18. Na prvom spomenutom obr. je možné vidieť že kernel (ďalej filter) má rozmery 2x2. To platí aj pri príklade poolingu. Počet filtrov hovorí koľko filtrov o danom rozmere bude uplatnených na každú časť siete. Sieť si upravuje každý filter na určité črty, pričom prvý filter môže zachytiť horizontálne hrany, druhý vertikálne atď. Vrstva pooling má rovnaký počet filtrov ako predchádzajúca konvolučná vrstva a počet filtrov s ďalšími vrstvami väčšinou rastie. Pri poklese množstva filtrov by sme prišli o dáta a výpočty na predchádzajúcich vrstvách by boli zbytočné. Stride určuje veľkosť skoku. Na obr. 17 je vidieť že vstup b a f bol násobený váhami pri posune doprava dva krát. Z toho vyplýva že zvolený skok

má hodnotu jedna. Na obr. 18 je podľa plôch rozdielnych farieb vidieť že skok je dva. Posledným hyperparametrom je orámovanie snímku ktoré slúži hlavne na upravenie rozmeru výstupu.

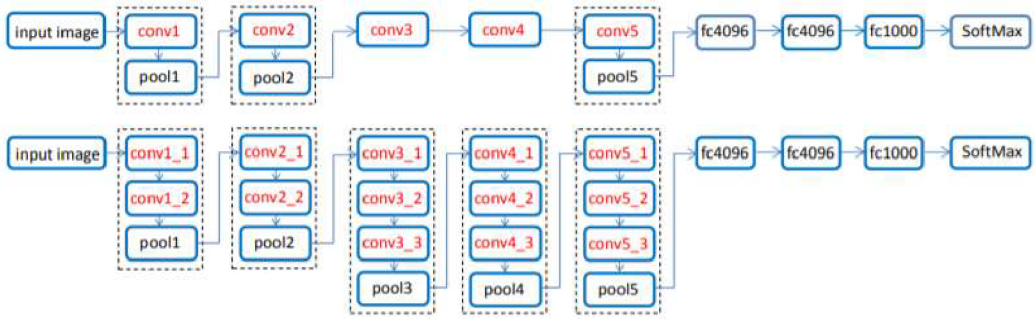
4.7.2 Príklady CNN

Klasická CNN a príklad jej činnosti je dobre viditeľný na obr. 17. Pomocou filtrov sa snímka postupne zjednodušuje až ostáva iba niekoľko pixelov ktoré reprezentujú dôležité črty snímky, podľa ktorých sa sieť rozhoduje.



Obr. 19: Príklad činnosti CNN

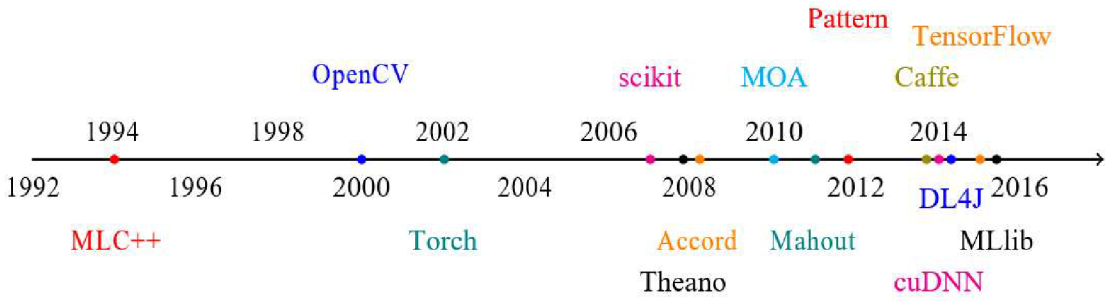
V histórii konvolučných sietí sa nájde mnoho významných sietí ktoré svojím výkonom v danej dobe predbehli konkurenciu. Za účelom porovnania týchto sietí usporiada projekt ImageNet každoročne súťaž ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Výherné architektúry sa často v pôvodnej alebo modifikovanej podobe ďalej používajú na riešenie rôznych problémov v oblasti rozpoznania snímok. Príkladmi týchto sietí môže byť AlexNet, víťaz z roku 2012, alebo VGG, architektúra ktorá v roku 2014 získala druhé miesto ale dnes je stále často používaná za účelom extrahovania rysov. [16][23]



Obr. 20: AlexNet (hore), VGG (dole)

5 FRAMEWORKY PRE HLBOKÉ UČENIE

Moderné algoritmy strojového učenia a umelej inteligencie zmenili prístup k problémom v mnohých oblastiach vedy a techniky. Nárast ich používania môžeme spozorovať hlavne v oblastiach spracovania obrazu, prirodzeného jazyku a rozpoznania reči. Taktiež sú obrovským prínosom pre online marketing a vyhľadávanie. V posledných rokoch sa stalo obzvlášť populárnym hlboké učenie, ktoré je oblasťou strojového učenia. Za účelom prevedenia týchto algoritmov do počítačových programov začalo vznikať mnoho knižníc. Za prvú sa považuje MLC++ pre jazyk C++ vydané v roku 1994. To aj OpenCV, vydané v roku 2000, sú však knižnice hlavne pre strojové učenie. Prvou knižnicou pre hlboké učenie je Torch, vydané v roku 2002. Od tej doby pribudlo mnoho knižníc ako Theano, Caffe, Tensorflow atď. V tejto kapitole sa zameriame na niekoľko najpopulárnejších knižníc pre hlboké učenie. [24]



Obr. 21: Časová os vydania knižníc pre strojové a hlboké učenie

5.1 TensorFlow

TensorFlow API je framework vydaný spoločnosťou Google v novembri 2015. Je to druhá generácia softvéru vytvoreného v rámci projektu Google Brain. Bol vytvorený za účelom implementácie modelov strojového učenia vo veľkej miere. Je napísaný v jazykoch Python, C++ a CUDA (rozhranie vytvorené spoločnosťou Nvidia). Pracuje na širokej škále zariadení od mobilného telefónu, cez jedno CPU/GPU až po veľké výpočtové systémy s tisíckami GPU. V roku 2018 bol spoločnosťou Google spustený cloud TPU (tensor processing unit) na ktorom je možné prenajať si veľké množstvo výpočtového výkonu.

TensorFlow má dnes radu ďalších modelov a rozšírení. Niektorými z nich sú TensorBoard, ktorý slúži k jednoduchšej vizualizácií dát verziu lite upravenú pre mobilné platformy a TensorFlow pre Swift alebo JavaScript.

5.1.1 Princíp činnosti TensorFlowu

V TensorFlowe je výpočet reprezentovaný diagramom, ktorý je zostrojený z niekoľkých uzlov. Tento diagram je potom zostrojený v jednom z podporovaných jazykov. Každý uzol reprezentuje inštanciu operácie. Diagram pracuje s tensormy. Klientske programy

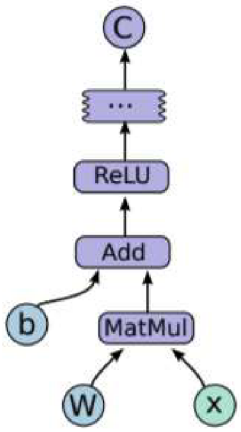
(jazyky) komunikujú s TensorFlowom prostredníctvom tzv. *Session* ktorý uskutoční príkazy definované v grafe a má v sebe uložené výsledné hodnoty. [25][26]

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

Obr. 22: Príklad kódu v Python v2



Obr. 23: Diagram reprezentujúci kód z obr. 4.2

5.2 Theanno

Framework Theanno bol vytvorený na Université de Montréal v rámci projektu Montreal Institute for Learning Algorithms (MILA) v roku 2010. Je to program pre spracovanie matematických výrazov napísaný v jazyku Python, ktorý využíva syntax knižnice Numpy, ktorá slúži na prácu s maticami, a kombinuje ho s optimalizovaným strojovým učením. Funguje na princípe optimalizovania používateľom zadaných výrazov, ich následnom prevedení do jazyka C++/CUDA a spätnom preložení do Pythonu. V dobe svojho vzniku dosahoval pri práci na GPU 6,5-krát až 44-krát vyššiu rýchlosť ako jeho alternatívy. Avšak v roku 2017, po uvedení verzie 1.0 bol jeho vývoj zastavený z dôvodu množstva iných knižníc/frameworkov s podobnou funkcionalitou. [27][28]

5.3 The Microsoft Cognitive Toolkit

The Microsoft Cognitive Toolkit (CNTK) bol vytvorený spoločnosťou Microsoft v roku 2017. Môže byť použitý v jazykoch Python C# a C++ alebo môže byť použitý samostatne pomocou jeho vlastného popisného jazyku s názvom BrainScript. CNTK podporuje 64-bitové verzie Linuxu a Windowsu. Je jednou z prvých knižníc ktorá podporuje formát Open Neural Network Exchange (ONNX). Ten umožňuje presúvanie modelu medzi rôznymi frameworkami. [29]

5.4 Keras

Keras je API jazyku Python, ktorá poskytuje bloky vyššej úrovne pre tvorbu neurónových sietí. To znamená že sám nezvláda operácie nižšej úrovne, ako je práca s tensormi. Z toho vyplýva že potrebuje nejaký framework nižšej úrovne, ktorý tieto operácie zvláda. Dnes sú týmito frameworkami TensorFlow, Theanno a CNTK. Tým kombinuje jednoduchý užívateľský prístup s funkcionalitou spomenutých frameworkov. Vývoj Kerasu je podporovaný hlavne spoločnosťou Google, preto je odporúčané používať ho spolu s knižnicou TensorFlow. [30]

5.5 PyTorch

PyTorch je balík na tvorbu neurónových sietí určený pre jazyk Python. Jeho vývoj zabezpečuje hlavne výskumná skupina AI Facebooku. Podporuje aj jazyky C/C++ a CUDA. Model výpočtu je založený na frameworku Torch. Zobral si z neho napríklad knižnice na prácu s tensormi, čím sa líši od konkurencie ktorá často využíva NumPy. PyTorch je dynamickým frameworkom, čím sa líši od konkurencie ako napríklad TensorFlow a Caffe, ktoré sú statické. V tomto prípade je vybudovaný diagram ktorý dopredu diferencovaný a potom niekoľko krát vykonaný. Pri zmene časti diagramu sa musí trénovanie začať odznova. Výhodou je vyššia rýchlosť pretože diagram môže byť spracovaný paralelne. Pri dynamickom modelovaní diagramu je možné ho zmeniť chovanie siete ľubovoľne bez oneskorenia alebo zmeny času potrebného na dopočítanie výsledku. Toto riešenie je vhodné napríklad pre dáta rozdielnych veľkostí. Spolu s CNTK, Caffe2 a MXNet podporuje ONNX. [31]

5.6 Caffe

Framework Caffe bol vyvinutý tímom Berkeley AI Research. Je napísaný v jazyku C++ a CUDA s rozhraním podporujúcim Python a Matlab. Výhodou je plynulý prechod ak chceme presunúť výpočet z CPU na GPU a vysoká rýchlosť. Počas vývoja dosahoval jedny z najvyšších rýchlostí pri rôznych populárnych modeloch neurónových sietí. [32]

5.7 Caffe2

V roku 2017 oznámil Facebook Caffe2 ktoré má oproti svojmu predchodcovi zlepšenú funkcionalitu a rozšírený podporovaný hardware (napr. podpora mobilných platforiem). Je napísaný v jazyku C++ a Python. Základnou výpočtovou jednotkou v Caffe2 sú tzv. *Operator*, ktoré sa dajú považovať za flexibilnejšiu verziu *vrstiev* v Caffe. Keďže Facebook vlastní okrem Caffe2 aj PyTorch boli tieto dva frameworky v roku 2018 spojené, čo znamená že kód napísaný v Caffe2 môže byť prečítaný pomocou PyTorch. [33][34]

5.8 MXNet

MXNet vzniká pod záštitou The Apache Software Foundation, sponzorovanou projektom Apache Incubator. Používa ho spoločnosťou Amazon. Je napísaný vo veľkom množstvom jazykov, a to: Python, C++, Clojure, Java, Julia, Perl, R a Scala. Pre zjednodušenie vytvorenia a vytrénovania siete je v ňom implementovaná API pre hlboké učenie s názvom Gloun. [35]

5.9 DL4J

Deep learning for Java (DL4J) je napísaný v jazyku Java a je kompatibilný so všetkými Java Virtual Machine (slúži na chod Java kódu) jazykmi ako Scala, Clojure alebo Kotlin. Základné výpočty sú napísané v C, C++ a Cuda. Na prácu s tensormi využíva knižnicu ND4J. Je jediným z pomedzi populárnejších frameworkov ktorý nepodporuje Python, avšak je možné do neho importovať modely napísané v Kerase. Jeho vývoj zaisťuje startup spoločnosť Skymind. [36]

5.10 Chainer

Tento framework bol vytvorený v roku 2015 Japonskou spoločnosťou Preferred Networks v spolupráci s IBM, Intel, Microsoft, Nvidia a AWS. Je napísaný v jazyku Python a pre prácu s tensormi využíva NumPy. Pre prácu na rozhraní CUDA využíva knižnicu CuPy, ktorá je implementáciou NumPy na CUDA. Podobne ako PyTorch, funguje na princípe dynamického modelovania diagramu. [37]

5.11 FastAI

FastAI, vytvorený spoločnosťou fast.ai, nepracuje samostatne, ale podobne ako Keras potrebuje framework nižšej úrovne. V tomto prípade sa jedná o PyTorch. FastAI bolo vytvorené, pretože niektoré modely sú v Kerase a TensorFlowe veľmi ťažko implementovateľné. Tvorcom taktiež vyhovoval dynamický diagram PyTorchu. [38]

5.12 Nvidia cuDNN

Je to knižnica pre hlboké učenie od spoločnosti Nvidia. Je to veľmi významná knižnica, aj napriek tomu že sa s jej pomocou siete často nepíšu, pretože je implementovaná v takmer každom populárnom frameworku, vrátane takých ako TensorFlow, PyTorch, Caffe, Caffe2, Chainer, MXnet atď. Vďaka tento knižnici môžu spomenuté frameworku komunikovať s rozhraním Nvida CUDA. [39]

5.13 Spoplatnený sotfware pre hlboké učenie

Všetky doteraz zmienené knižnice/frameworky sú open-source, teda je ich možné používať bez poplatkov. So spoplatnených za zmienku určite stojí Wolfram Mathematica a Matlab.

5.13.1 Wofram Mathematica

Wolfram Mathematica je vyvíjaný spoločnosťou Wolfram Research, sídlaciou v Champaing, Illinois. Okrem tvorby neurónových sietí zahrňuje aj ostatné odvetvia strojového učenia, geometrie, práce s dátami a ďalšie funkcie. Je napísaný v jazyku Wolfram. Vďaka službe Mathematica Online je možné tento systém využívať v službe Cloud pomocou akéhokoľvek prehliadaču. [40]

5.13.2 Matlab

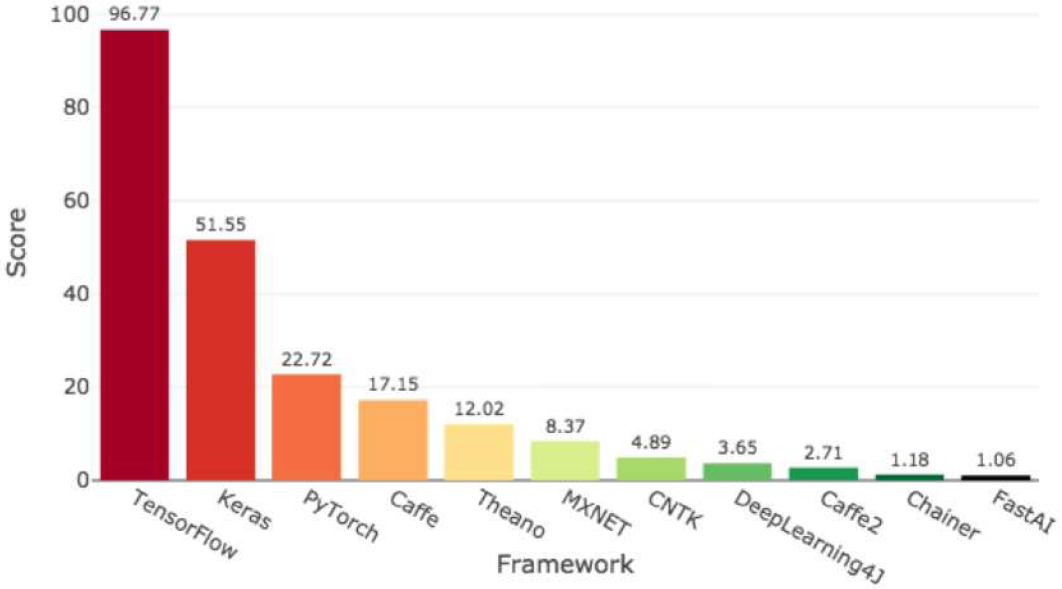
Programovací jazyk Matlab je jazykom vytvoreným spoločnosťou MathWorks určením prevažne pre numerické výpočty. Okrem príkazov na tvorbu neurónových sietí sú v ňom zahrnuté aj užitočné aplikácie na predbežné spracovanie dát a podobne. Priamo spolupracuje s frameworkami Caffe a Keras-TensorFlow a vďaka službe ONNX spolupracuje aj z ostatnými frameworkami ktoré túto službu podporujú. [41]

5.14 Porovnanie frameworkov pre hlboké učenie

V tejto kapitole v rýchlosti porovnáme popularitu a výkon vybraných frameworkov. Rozvoj hlbokého učenia je veľmi rýchly, a to platí aj pre tieto frameworky. Jednotlivé grafy sa preto o pár mesiacov veľmi líšiť. To platí aj o tých zahrnutých v tejto práci, ktoré aj napriek tomu že boli vydané relatívne nedávno, dnes už nemusia platiť.

5.14.1 Popularita

Graf zostavený z údajov dostupných počas roka 2018 sleduje viacero faktorov a to: požiadavky na trhu práce, publikácie na ArXiv, Medium a vydané knihy na Amazon Books, ďalej aktivita vyhľadávania na prehliadači Google, hlasovanie na vedeckej stránke KDnuggets a nakoniec popularita na GitHube. Skombinovaním týchto dát získame graf:



Obr. 24: Popularita frameworkov v roku 2018

Jedná sa skôr o orientačný graf, ale je v ňom jasne viditeľná prevaha ako TensorFlowu, ktorý vyhral takmer každú z jednotlivých kategórií, tak Kerasu, ktorý TensorFlow využíva. Za zmenku stojí, že väčšina týchto frameworkov bola vydaná až v druhej polovici dekády a jedine Theanno malo svoju prvú verziu vydanú pred rokom 2010. [42]

5.14.2 Výkon

Porovnáme časové údaje pri extrahovaní rysov/trénovaní na populárnych sieťových modeloch/datasetoch. Tieto časy boli merané na serverových grafických kartách Tesla K80 a P100. Niektoré spomenuté frameworky sa v tomto porovnaní neobjavia a naopak tu nájdeme také, ktoré doteraz neboli spomenuté.

Pri prvom porovnaní (Tab. 1) sa zameriame na dĺžku trénovanie na CNN sieti architektúry VGG (podobná tej na obr. 18) a CIFAR-10 dateset (50k tréningových sníkom., 10k testovacích snímok, 32x32x3).

Tab. 1: Porovnanie frameworkov – trénovací čas(s)

DL Library	K80/CUDA 8/CuDNN 6	P100/CUDA 8/CuDNN 6
Caffe2	148	54
Chainer	162	69
CNTK	163	53
MXNet(Gluon)	152	57
Keras(CNTK)	194	76
Keras(TF)	241	76
Keras(Theano)	269	93
Tensorflow	173	57
Lasagne(Theano)	253	65
MXNet(Module API)	145	52
PyTorch	169	51
Julia - Knet	159	??
R - Keras(TF)	205	72

Tab. 2: Porovnanie frameworkov – extrahovanie rysov(s)

DL Library	K80/CUDA 8/CuDNN 6	P100/CUDA 8/CuDNN 6
Caffe2	14.1	7.9
Chainer	9.3	2.7
CNTK	8.5	1.6
MXNet(Gluon)		1.7
Keras(CNTK)	21.7	5.9
Keras(TF)	10.2	2.9
Tensorflow	6.5	1.8
MXNet(Module API)	7.7	1.6
PyTorch	7.7	1.9
Julia - Knet	6.3	???
R - MXNet	???	???
R - Keras(TF)	17	7.4

V Tab. 2 vidíme čas potrebný na extrahovanie rysov z 1000 snímok. Pri extrahovaní rysov ide o zapamätanie dôležitých hrán, rohov a ďalších čŕt snímky, podľa

ktorých sieť rozpoznáva triedu snímky. Pri tomto porovnaní boli použité 264x264 snímky a sieť architektúry typu ResNet-50.

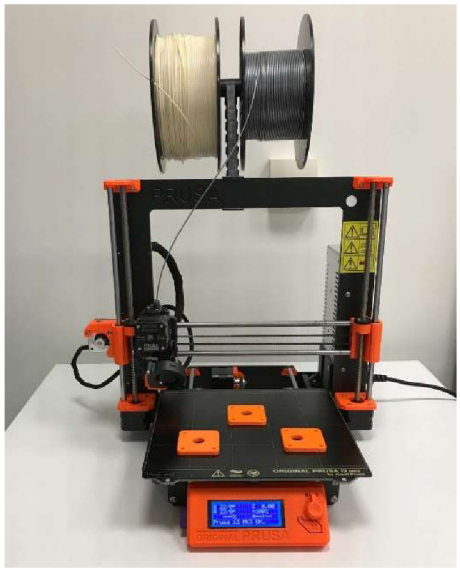
Toto porovnanie je ako v prípade popularity taktiež skôr orientačné, pretože v tejto práci je použitá iná sieť a tréning prebieha na výrazne slabších výpočtových jednotkách, ale je možné vidieť že TensorFlow si medzi ostatnými frameworkmi vedie slušne, a rozhodne patrí medzi rýchlejšie. To, jeho obľúbenosť, a s ňou aj množstvo dostupných zdrojov, z neho robí vhodný framework pre implementáciu v tejto práci. [43]

6 IMPLEMENTÁCIA PROGRAMU

Hlavným cieľom tejto práce je zostrojenie neurónovej siete na určenie správnosti vyrobenej súčiastky. Okrem toho môžeme zistiť ďalšie jej vlastnosti ako farbu, ktorá može znamenať napr. typ súčiastky. V danej kapitole sa teda zameriame na tvorbu tejto siete.

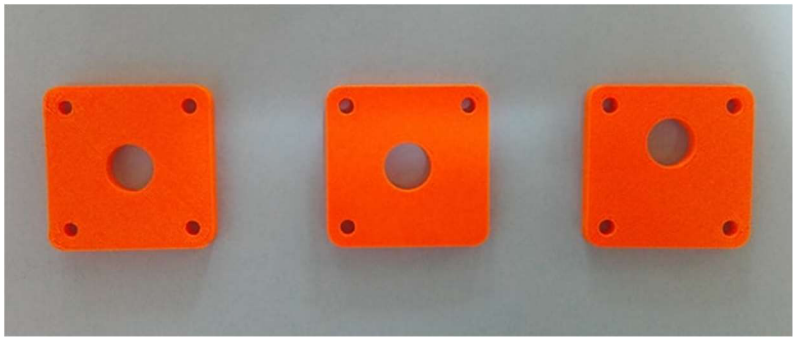
6.1 Kontrolovaná súčiastka

Súčiastka bola vyrobená na 3D tlačiarni Prusa i3 mk3 z PLA materiálu. Ide o štvorec so zaoblenými hranami, rôznym počtom dier v rohoch a jednou väčšou dierov v strede súčiastky, resp. neďaleko od neho. Pôvodne mala dvojnásobné rozmery ale z praktického hľadiska boli nakoniec rozmery zmenšené. Hrana má teda dĺžku 5 cm, malé diery majú priemer 0,5 cm a veľká diera má priemer 1,5 cm.



Obr. 25: Tlačiareň Prusa i3 mk3

Súčiastka má 3 kategórie, pričom prvá je v poriadku (ok), druhej chýba jedna malá diera a teda je opraviteľná (repair), a tretia je nefunkčná pretože má vychýlenú veľkú dieru (bad).



Obr. 26: 3 druhy súčiastky: ok (vľavo), repair (stred), bad (vpravo)

Tieto súčiastky boli vyrobené vo viacerých farbách. Sú nimi oranžová, zelená, čierna, bledohnedá a tmavohnedá.

6.2 Programovací jazyk a knižnice/frameworky

Ako už bolo viac krát spomenuté, sieť bude napísaná v jazyku Python, verzií 3.6.7. Ako vývojové prostredie bude použité základné prostredie Pythonu Python IDLE a prostredie PyCharm vývojovej spoločnosti JetBrains.

K napísaniu programu ďalej využijeme niekoľko knižníc, ktoré nám výrazne uľahčia prácu. Presnejšie ide o:

TensorFlow: V dnešnej dobe najpoužívanejší framework na tvorbu neurónových sietí. Na zaznamenanie loss funkcie počas trénovania a prípadne skontrolovanie parametrov siete využijeme jeho rozšírenie Tensorboard. Využijeme aj jeho modifikáciu TensorFlow-GPU, ktorý spolu s aplikáciou CUDA a cuDNN podporuje prácu na grafických kartách Nvidia.

Numpy: Knižnica pre operácie s maticami. Je súčasťou balíku TensorFlow, ktorý ju využíva pri výpočtoch.

Keras: Nadstavba na TensorFlow (alebo iný framework). Rozhodli sme sa pracovať s TensorFlowom a teda Keras používame len v rámci jedného príkazu – zoradenia označenia súčiastok v matici.

Scikit-learn: Knižnica strojového učenia taktiež použitá len v rámci jedného príkazu, a to shuffle – využívame na premiešanie matíc vstupných údajov, pričom údaje ktoré mali voči sebe rovnaký index pred premiešaním, ho majú voči sebe rovnaký aj po premiešaní.

OpenCV: Veľmi dôležitá knižnica, ktorú využívame hlavne na otvorenie snímok, ale aj mnoho iných úloh ako kreslenie do snímky, jej natočenie, konvertovanie do sivých odtieňov (grayscale), atď.

Random: Knižnica využitá pri práci so snímkami pri tvorbe dátového súboru, napr. náhodná intenzita Gaussovho šumu.

Matplotlib: Použitá za účelom zostrojenia výstupných grafov.

Time: Použitá na zaznamenanie doby potrebnej na vykonanie programu, nepotrebná k funkčnosti kódu.

6.3 Použitý hardware

K trénovaniu siete bolo využívané aj CPU aj GPU. V prípade procesora ide o 6-jadrový Intel Core i7-8750H so základnou frekvenciu 2,20 GHz s možnosťou využitia technológie Intel Turbo Boost, ktorá umožňuje navýšiť frekvenciu na 4,10 GHz. V prípade grafickej karty ide o Nvidia GeForce GTX 1050Ti so štvôr gigabajtovou kapacitou pamäti. Obi dve jednotky sú vyrobené 14 nm procesom.

6.3.1 CUDA a cuDNN

Použili sme verziu CUDA 10.0 a k nej kompatibilnú najnovšiu verziu 7.5.1 cuDNN. Potrebné verzie boli uvedené na oficiálnej stránke TensorFlowu. Je dostupné veľké množstvo verzií týchto programov a veľmi jednoducho sa môže stať že nebudú kompatibilné s Tensorflowom alebo Pythonom. Nainštalovaný balík zaberá cca. 2 GB.

6.4 Riešenie problému

V dnešnej dobe existuje niekoľko algoritmov ktoré by spolu s jedným modelom siete zvládli klasifikáciu a lokalizáciu objektu. Jedna sieť by pravdepodobne dokázala zvládnuť aj klasifikáciu nielen podľa typu súčiastky, ale aj farby. Pri riešení týmto spôsobom je však možné predpokladať potrebu oveľa väčšieho množstva dát, čo by však znamenalo extrémnu časovú náročnosť počas zberu dát, ich spracovania a nakoniec aj trénovania. Preto boli nakoniec použité 3 siete, z ktorých 1 určí približnú polohu súčiastky, s pomocou tejto polohy bude vyrezaná menšia časť snímku ktorej druhá a tretia sieť určí kategóriu a farbu. Takto bude možné porovnať ako si neurónové siete vedú pri jednotlivých problémoch.

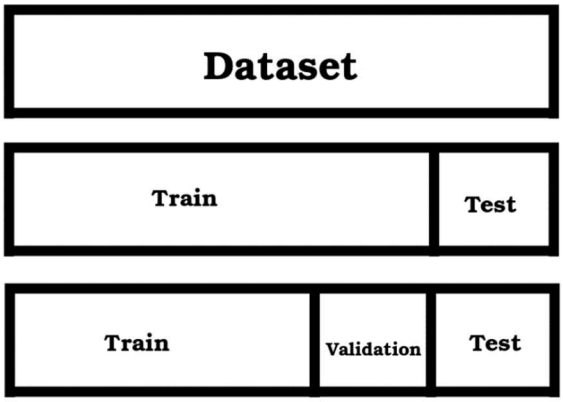
6.5 Dataset

Pri tvorbe datasetu (dátového súboru) boli použité snímky vyfotografované Michalom Huberom, ktorý pracoval na Raspberry Pi za účelom detekcie objektu pomocou strojového videnia. Cieľ jeho práce bol podobný ako cieľ tejto práce, a vďaka tomu bolo možné využívať jeho snímky. Spôsob riešenia problému v daných prácach bol však odlišný. [44]

K úspešnému vytrénovaniu siete je potreba mať dostatočne veľký a správne rozdelení dataset (súbor dát – v našom prípade snímky). Veľkosti jednotlivých datasetov pre jednotlivé siete sú rôzne. Pre trénovanie určenia kategórie objektu bolo z rôznych dôvodov použitých viacero datasetov.

Dataset môže byť rozdelený na train (trénovací) a test (testovací) v pomere približne 9:1. Ešte používanejšou možnosťou rozdelenia je rozdelenie na train, validation (overovací) a test (testovací) v pomere približne 8:1:1. Tieto rozdelenia sa používajú za účelom overenia správnosti učenia, pri ktorom môže nastať že počas trénovania sieť dosahovala skvelé výsledky, tie sa však nedajú na vstupy mimo trénovací súbor, pretože sieť sa naučila zlé parametre alebo sa príliš prispôsobila danému súboru (tzv. overfitting).

Pri trénovaní je dataset ďalej delení na menšie časti tzv. mini batche, ktoré majú v sebe istý počet snímok a postupne všetky prejdú sieťou v jednej epoche (jedno spracovanie celého datasetu, teda všetkých mini batchov). V praxi sa ukázalo že toto ďalšie rozdelenie priaznivo ovplyvňujú výkon siete. Taktiež sa zistené, že je vhodné, ak má mini batch veľkosť rovnajúcu sa n-tej mocnine dvoch. Nami zvolená hodnota je 2^5, teda 32.

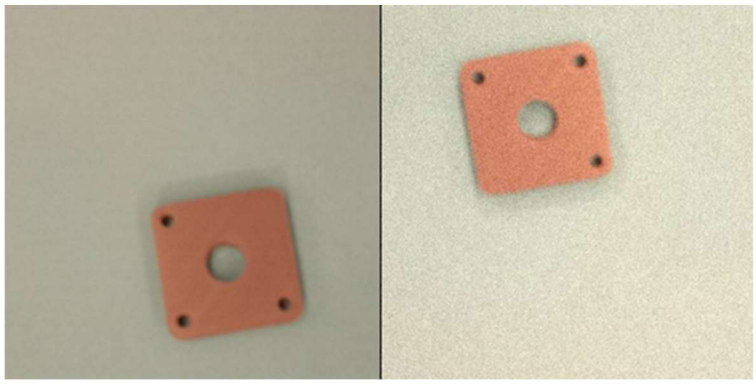


Obr. 27: Rozdelenie datasetu

V našom prípade sme pri overovaní funkčnosti určovania polohy súčiastky zvolili malé množstvo snímok, ktoré bolo potrebné skontrolovať, pretože sme nemohli priamo zistiť úspešnosť ako v prípade určovania kategórie a farby. Pri trénovaní kategórie a farby sme použili overovací súbor ktorý predstavoval 9 až 12% z trénovacieho , a testovací súbor o veľkosti 117 (bez čiernych súčiastok) resp. 147 snímok. Trénovací súbor pre polohu súčiastky má veľkosť 265 snímok. Dataset pre trénovanie farby je rozdelený na trénovaciu časť o veľkosti 1088 snímok a overovaciu časť o veľkosti 133 snímok. Ako už bolo spomenuté datasetov pre kategóriu súčiastky bolo vytvorených viacero, k problému sa dostaneme v kapitole 7.4. [45]

6.5.1 Tvorba datasetu

Bolo rozhodnuté že počet nafotených snímok súčiastky bude celkom nízky, presnejšie medzi 270-290. K dispozícii bol dostatok snímok na inom povrchu, ale dá sa povedať že toto by mohlo modelu skôr uškodiť, vzhľadom na to že mnoho povrchov malo inú textúru a farbu, ktorá by sa na výrobnej linke nikdy nenachádzala. Bolo teda potrebné umelo navýšiť počet súčiastok. K tomu nám pomohlo rotovanie súčiastky zmena jej polohy a aplikácia Gaussovho šumu – štatistického šumu, ekvivalentného tomu, ktorý vzniká napr. pri fotografovaní pomocou niektorých optických senzorov pri zhoršenom osvetlení. Tieto zmeny boli aplikované pomocou jednoduchého programu.



Obr. 28: Rotácia + Gaussov šum

6.6 Zvolené modely sietí

Pre každú jednu sieť sme zvolili trochu inú veľkosť. Pre zistenie polohy súčiastky bola zvolená sieť s troma konvolučnými a piatimi plne prepojenými vrstvami. V prípade rozpoznania farby sme použili model siete s dvoma konvolučnými vrstvami, troma plne prepojenými a malým množstvom filtrov. Model o takejto veľkosti bol zvolený lebo bola predpokladaná jednoduchosť určenia farby. Modely na detekciu a kategóriu objektu sú pomocou vyššie spomenutej knižnice opencv na vstupe do siete prevádzané do odtieňov šedej. RGB hodnoty by efektivitu týchto sietí pravdepodobne nezvýšili v týchto sieťach by nemali význam.

6.6.1 Modely na určenie kategórie objektu

Pri určovaní kategórie objektu bolo vyskúšané veľké množstvo menších sietí s dvoma až troma konvolučnými vrstvami, a tromi až piatimi plne prepojenými vrstvami. Taktiež boli vytvorené dva väčšie modely, ktoré čerpali inšpiráciu z AlexNet a VGG.

Tab. 3: Model imitujúci AlexNet [46]

Veľkosť/operácia	Filter	Hĺbka	Krok	Počet parametrov
240 x 240 x 1				
Conv1 + Relu	12 x 12	64	3	(12*12*1+1)*64=9280
77 x 77 x 64				
Max. pool	2 x 2		2	
38 x 38 x 64				
Conv2 + Relu	5 x 5	96	1	(5*5*64+1)*96=153696
34 x 34 x 96				
Conv3 + Relu	3 x 3	96	1	(3*3*96+1)*96=83040
32 x 32 x 96				
Max. pool	2 x 2		2	
16 x 16 x 96				
Conv4 + Relu	3 x 3	128	1	(3*3*96+1)*128=110720
14 x 14 x 128				
Conv5 + Relu	3 x 3	128	1	(3*3*128+1)*128=147584
12 x 12 x 128				
Max. pool	2 x 2		2	
6 x 6 x 128				
Fc6 + Relu				(6*6*128)*512=2359296
512				
Dropout = 0,5				
Fc7 + Relu				512*32=16384
32				
Fc8 + Softmax				32*3=96
3				
				Celkovo: 2880096

Oproti AlexNet sa tu nájde niekoľko rozdielov, hlavne v počte filtrov a prepojení konvolučných vrstiev. Veľkosti filtrov (okrem prvého ktorý má rozmer 12x12 namiesto 11x11) a počet konvolučných aj plne prepojených vrstiev je však totožný. Taktiež je zhodný počet združovacích vrstiev, pričom 2 z 3 majú rovnakú polohu. Bola teda hlavne znížená veľkosť filtrov a veľkosť plne prepojených vrstiev, a to z dôvodu že AlexNet je určená na klasifikáciu viacerých kategórií a rozmanitejších snímok.

Tab.4: Model imitujúci VGG [47]

Veľkosť/operácia	Filter	Hĺbka	Krok	Počet parametrov
240 x 240 x 1				
Conv1 + Relu	3 x 3	36	1	(3*3*1+1)*36=396
238 x 238 x36				
Max. pool	2 x 2		2	
119 x 119 x36				
Conv2 + Relu	3 x 3	48	1	(3*3*36+1)*48=15600
117 x 117 x 48				
Conv3 + Relu	3 x 3	48	1	(3*3*48+1)*48=20784
115 x 115 x 48				
Max. pool	2 x 2		2	
57 x 57 x 48				
Conv4 + Relu	3 x 3	64	1	(3*3*48+1)*64=27712
55 x 55 x 64				
Conv5 + Relu	3 x 3	64	1	(3*3*64+1)*64=36928
53 x 53 x 64				
Conv6 + Relu	3 x 3	96	1	(3*3*64+1)*96=55392
51 x 51 x 96				
Max. pool	2 x 2		2	
25 x 25 x 64				
Conv7 + Relu	3 x 3	96	1	(3*3*96+1)*96= 83040
23 x 23 x 96				
Conv8 + Relu	3 x 3	128	1	(3*3*96+1)*128=110720
21 x 21 x 128				
Conv9 + Relu	3 x 3	128	1	(3*3*128+1)*128= 147584
19 x 19 x128				
Max. pool	2 x 2		2	
9 x 9 x 128				
Fc8 + Relu				(9*9*128)*512=5308416
512				
Dropout = 0,5				
Fc9 + Relu				512*32=16384
32				
Fc10 + Relu				32*3=96
3				
				Celkovo: 5823052

U sietí typu VGG, presnejšie VGG16 sa pri všetkých konvolúciach používajú 3x3 filtre a využíva sa veľké množstvo týchto vrstiev (VGG16 - 13 konvolučných vrstiev, 3 plne prepojené). Medzi vybranými vrstvami sa nachádzajú združovacie vrstvy, ktorých rozmer je vždy 2 x 2 s krokom s = 2.

Pri tomto modely je sme oproti skutočnej VGG16 použili o 4 konvolučné vrstvy menej. Rozdielom je aj počet filtrov a rozmer plne prepojených vrstiev. Zamerajme sa na vzťah na rozmer vrstvy po prechode filtrom, ktorý má po zanedbaní orámovania ktoré nepoužívame tvar:

$$n_{i+1} = \frac{n_i - f}{s} + 1 \tag{6.1}$$

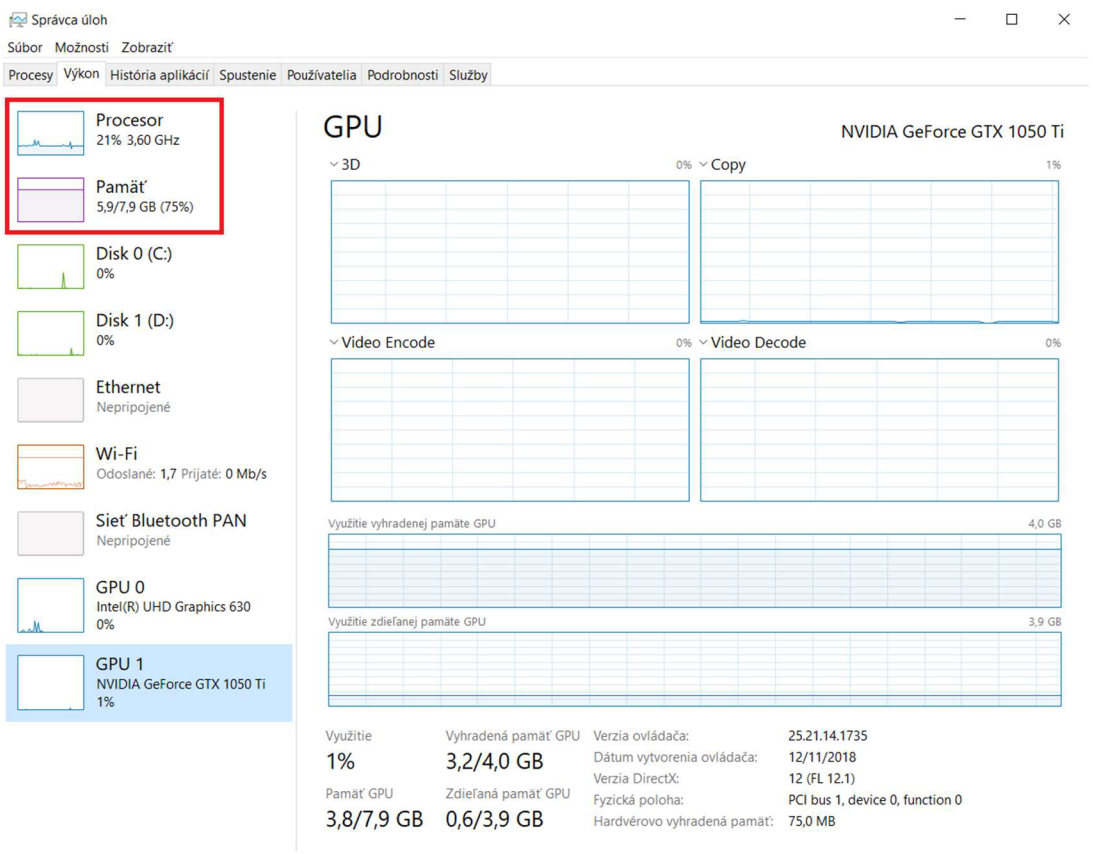
Kde f je rozmer filtru, s krok, n rozmer v danej vrstve. Tento vzťah platí okrem konvolučných vrstiev aj na združovaciu vrstvu, a teda môžeme vidieť že krok 2 delí v prípade troch združovacích vrstiev nepárne číslo, čo môže spôsobiť stratu niektorých informácií. Tento problém bol odstránení pri upravovaní modelu zmenou druhého z rozmerov 3x3 na 2x2. Miernie stúpol počet parametrov siete. Boli vyskúšané aj iné počty konvolučných vrstiev a filtrov. Viac o trénovaní tejto siete v kapitole 7.4.3. [18]

7 VÝSLEDKY

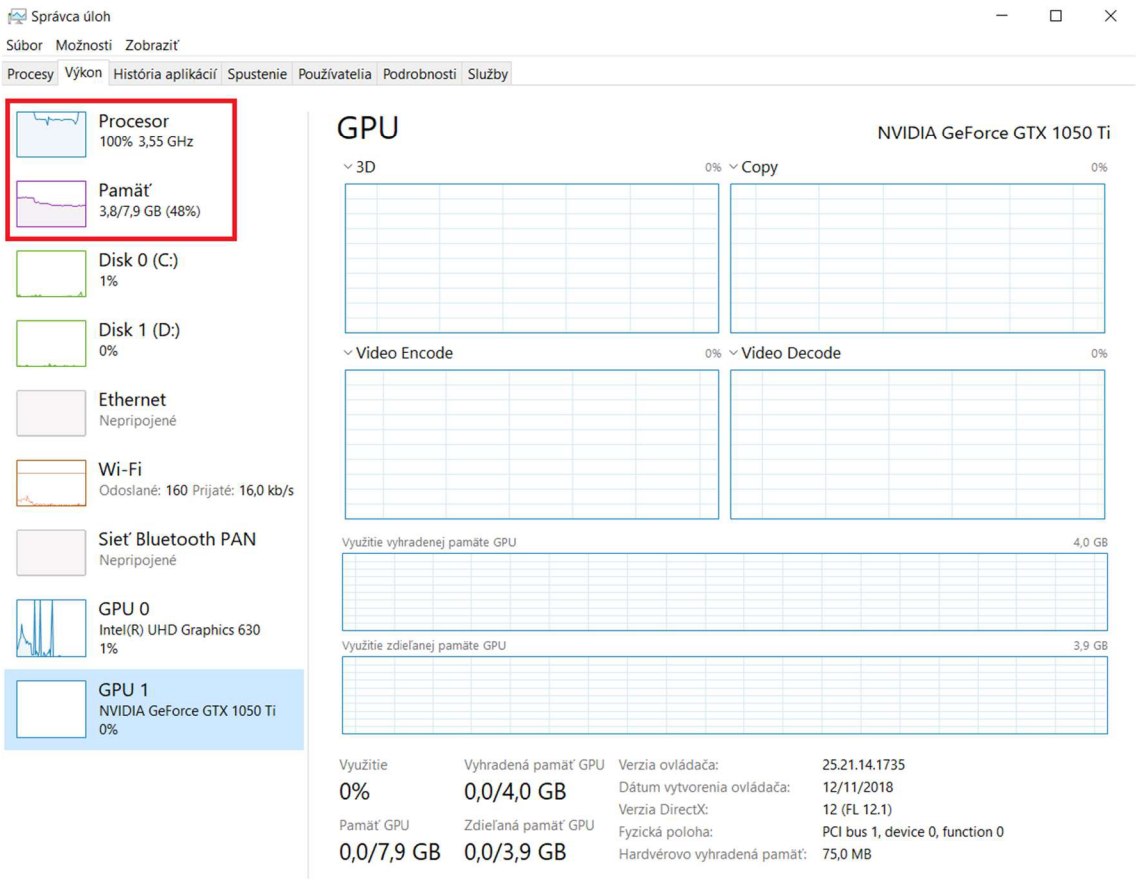
V kapitole sa zameriame na výsledky modelu určenia polohy súčiastky, ďalej farby a nakoniec modelu na kontrolu kategórie súčiastky. Tieto výsledky potom zhodnotíme ako celok.

7.1 Trénovanie na CPU a GPU

Trénovanie siete je veľmi dôležitým hľadiskom pri rozhodovaní či použiť hlbokú sieť alebo iný druh strojového učenia/spracovania dát. Preto je vhodné zamerať sa rozdiel medzi trénovaním na CPU a GPU. GPU má v tomto smere významnú výhodu keďže dáta dokáže spracovať paralelne. CPU je vhodnejšie skôr na menšie výpočty a teda veľké matice budú oveľa efektívnejšie spracované na GPU. Ďalšiu výhodou grafickej karty je že môže svoju pamäť zaplniť frontov operácií ktorá čaká na spracovanie, čo znamená že po vykonaní jednej operácie čaká na vstupe hneď ďalšia. Tým sa ďalej urýchľuje výpočet.

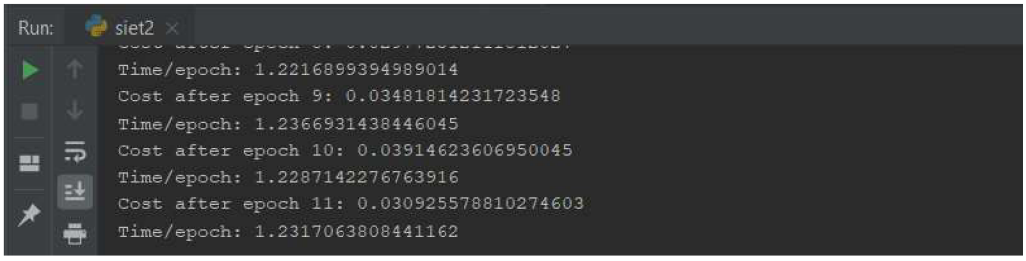


Obr. 29: Trénovanie na GPU



Obr. 30: Trénovanie na CPU

Pri trénovaní na GPU je vidieť zaplnenie vyhradenej pamäte GPU, plnšiu RAM a výrazne nižšie zaťaženie procesora. Je však vidieť nízke zaplnenie GPU, na ktorý pravdepodobne nestíhali dosť rýchlo prichádzať dáta na spracovanie. Rozdiel v efektivite výpočtu je najviac viditeľný pri rýchlosti trénovania upraveného modelu AlexNET.



Obr. 31: Doba trvania jednej epochy pri použití GPU(Time/epoch[s])

```
Cost after epoch 1: 1.0986007404327394
Time/epoch: 20.978593349456787
Cost after epoch 2: 1.0986071252822875
Time/epoch: 24.537800073623657
Cost after epoch 3: 1.098589119911194
Time/epoch: 23.43434166908264
Cost after epoch 4: 1.098574185371399
Time/epoch: 19.772133588790894
Cost after epoch 5: 1.098577799797058
Time/epoch: 24.102554321289062
```

Obr. 32: Doba trvania jednej epochy pri použití CPU(Time/epoch[s])

7.2 Detekcia objektu

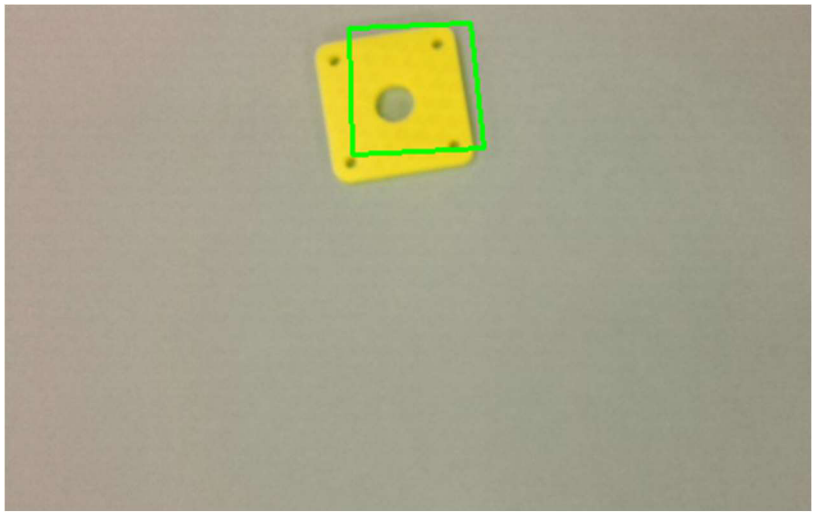
Prvou úlohou bolo zaistiť dostatočne presnú detekciu súčiastky, ktorá by vybrala časť snímky, kde na ktorej sa daná súčiastka nachádza.

7.2.1 Trénovanie modelu na detekciu objektu

Ako parametre, ktoré sa má sieť naučiť boli zvolené rohy súčiastky. Hodnoty v oboch osách boli odčítané z dvoch proti stojných rohov. Keďže sa jedná o štvorcovú súčiastku ďalšie dva rohy sa dali dopočítať, čo výrazne uľahčilo prácu. Pôvodne bola snaha vytrénovať sieť len na súradniciach dvoch rohov, v tomto prípade však výsledky po istej dobe trénovania vyzerali horšie. To je pravdepodobne spôsobené tým že sieť nevedela na ktorý roh sa má zamerať. Detekcia bola trénovaná len na CPU, čo výrazne predĺžilo čas trénovania a to zhruba na 2 dni. Loss funkcia za tú dobu klesla z hodnoty asi 110 000 (4 rohy a súradnice od 1 do 640, z toho vyplýva veľká chyba) na 630. Kvôli aplikácií dropout v jednej vrstve zrejme nemohla klesnúť na nižšie hodnoty a nenastal overfitting. Počas trénovania bola rýchlosť učenia upravovaná v rozmedzí hodnôt $1 \cdot 10^{-3}$ až $1 \cdot 10^{-6}$

7.2.2 Výsledky modelu na detekciu objektu

Cieľom zachytenia rohov súčiastky bola snaha vytvoriť štvorec okolo hrán súčiastky zobrazenej na snímke o rozmeroch 640x480 pixelov. Tento cieľ nebol splnený, keďže viacero rohov je správne zachytených iba vo veľmi ojedinelých prípadoch.



Obr. 33: Detekcia súčiastky

Zachytenie rohov však nie je nevyhnutné ku správnemu chodu siete a táto presnosť je dostatočná pre určenie približnej polohy objektu a vďaka tomu nasledovnému vloženiu menšej časti snímky do zvyšných sietí. Pre tento prístup sme sa rozhodli z dôvodu urýchlenia a zvýšenia siete, keďže pri väčšom obrázku by bol väčší problém určiť kategóriu súčiastky. Okolo stredu nájdeného štvoruholníka sa zoberie štvorec

o hrane 240 a ten je vložený na vstupy ďalších sietí. Pokiaľ je stred blízko hrany je zobraný štvorec predĺžený do ďalších strán aby si zachoval rozmer 240x240. Tu môže nastať situácia, kedy stred štvorca nie je umiestnený dosť presne a nedá sa určiť kategória súčiastky. Pri aplikáciách na snímky na ktorých bola poloha súčiastky len mierne pozmenená táto situácia nastala len veľmi výnimočne, Pri snímkach z väčšou zmenou polohy súčiastky sa však táto chyba už opakovala častejšie. Po preskúmaní tréningového súboru bolo napríklad vidieť, že len málo súčiastok bolo lokalizovaných v ľavom dolnom rohu, čo pri testovaní na snímkach otočených o 180° spôsobilo väčšiu nepresnosť, keďže viacero súčiastok sa ocitlo v týchto miestach.

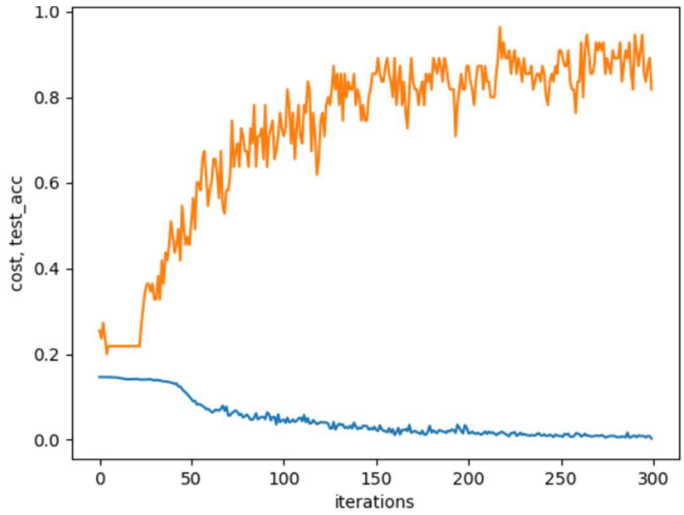
Obr. 34: Nesprávne detekovaná a obrezaná súčiastka

7.3 Rozpoznanie farby

Rozpoznanie farby by mala byť pre hlbokú sieť jednoduchou úlohou (minimálne v porovnaní s určením kategórie súčiastky), keďže sa zisťuje len rozdiel farby jednotlivých súčiastok, čo je oproti rozoznaniu počtu/polohy dier výraznejší rozdiel. Ďalším faktom ktorý zjednodušuje úlohu je to, že až na malé rozdieli intenzity osvetlenia ostáva pozadie rovnaké. V trénovacom súbore je preto použitý Gaussov šum ktorý do istej miery nahradzuje zmenu intenzity svetla.

7.3.1 Trénovanie modelu na určenie farby

Malé siete z ktorými sa pôvodne plánovalo pracovať sa rýchlo ukázali ako veľmi neefektívne. Ich tréning bol extrémne rýchly, a presnosť na overovacom súbore taktiež stúpala k hodnote 100%. Prechod na testovací súbor ukázal, že učenie nebolo správne a parametre ktoré sa sieť naučila boli mimo trénovacieho a overovacieho súboru nepoužiteľne.



Obr. 35: Priebeh učenia, malá sieť na rozpoznanie farby;
modrá – loss, oranžová – presnosť overovací súbor

Preto bola na zistenie farieb použitá sieť podobná AlexNet uplatnená aj pri kontrole kategórie výrobku. Priebeh učenia bol veľmi podobný, a už pri 50 epoche sieť dosahovala 98 % presnosť.

7.3.2 Výsledku modelu na určenie farby

Pri použití AlexNet modelu na testovací súbor boli dosiahnuté výrazne lepšie výsledky v porovnaní z malou sieťou. Presnosť na testovacom súbore dosiahla rovnakých hodnôt ako v prípade toho overovacieho, čo predstavuje okolo 97 % . V obidvoch testovacích súboroch došlo k piatim chybám, z čoho 4 bola bledohnedá súčiastka považovaná za tmavohnedú alebo oranžovú. Pri týchto chybách bola pravdepodobnosť vypočítaná sieťou väčšinou nižšia, z čoho vyplýva, že sieť sa naučila rozoznávať hlavne farbu súčiastky a nezaoberala sa inými vlastnosťami, ako napríklad polohou súčiastky. Bola teda vytrénovaná správne. Správnosť funkcie bola overená aj na snímkach so zmeneným kontrastom.

Obr. 36: Efektivita siete pri zmene kontrastu (stred - originál)

7.4 Rozpoznanie kategórie súčiastky

Pri rozpoznaní kategórie súčiastky sa vyskytlo viacero problémov týkajúcich sa učenia zlých parametrov alebo neklesania loss funkcie – sieť sa nebola schopná učiť. Bližšie si teda priblížime proces učenia a tvorbu siete.

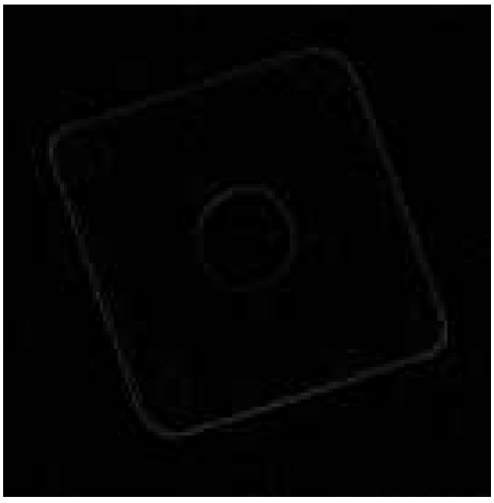
7.4.1 Dataset na rozpoznanie kategórie súčiastky

Ako pri určení farby súčiastky tu bol predpoklad, že menšia sieť bude dostatočná na určenie správnej kategórie. Oproti predtrénovaným modelom dostupným na internete, ktoré sú často kvôli rôznym súťažiam schopné rozpoznať vyšší počet kategórií, je rozpoznanie jedného z troch možných druhov problém, ktorý by mal teoreticky vyžadovať menšiu veľkosť siete. Počet hrán na snímke je vďaka rovnému podkladu taktiež výrazne menší, a preto by sa dalo očakávať že množstvo potrebných filtrov bude menšie. Ako dataset bol teda použitý súbor s 530 obrázkami a boli na neho aplikované rôzne menšie siete. Výsledky na tomto datasete však neboli uspokojivé, aj keď bol vytvorený rovnakým spôsobom ako nasledovné verzie. Siete sa nedokázali naučiť správne hodnoty, a kategóriu zrejme určovali podľa nesprávnych parametrov (ako poloha alebo natočenie súčiastky). Bolo teda vytvorených viacero iných datasetov s rôznym množstvom snímok, v prípade každej použitej siete však nastal problém neklesajúcej loss funkcie. Nakoniec bol teda vytvorený dataset bez čiernych súčiastok.

7.4.2 Nízka viditeľnosť otvorov čiernej súčiastky

Ako už bolo spomenuté, filtre konvolučnej siete sa upravujú tak aby dokázali zachytiť isté parametre vstupného snímku ako hrany natočené pod istým uhlom alebo istú časť farebného spektra. Pre rozpoznanie našich súčiastok je potrebné zamerať sa na diery a zistiť ich počet alebo polohu veľkej diery voči ostatným hranám. To môže komplikované, až neuskutočniteľné pokiaľ sieť tieto diery nie je schopná rozoznať.

Nižšie je možné vidieť graf učenia siete z troma konvolučnými vrstvami a štyrmi plne prepojenými vrstvami.

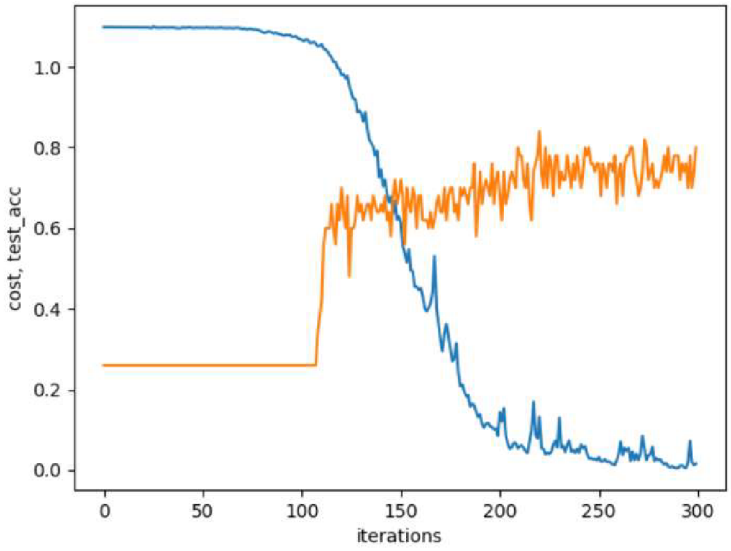


Obr. 37: Laplaceov filter aplikovaný na čiernu súčiastku

Filter konvolučnej vrstvy síce nepracuje na základe derivácií ale dá sa predpokladať že mnohé výstupy budú vyzerať podobne, pretože filter nebol schopný zachytiť ľudskému oku zle viditeľnú hranu. Nakoniec bol zvolený dataset bez čiernych súčiastok pozostávajúci z 904 snímok rozdelený na 800 snímok určených do trénovacieho súboru a 104 snímok do overovacieho.

7.4.3 Trénovanie sietí na určenie kategórie súčiastky

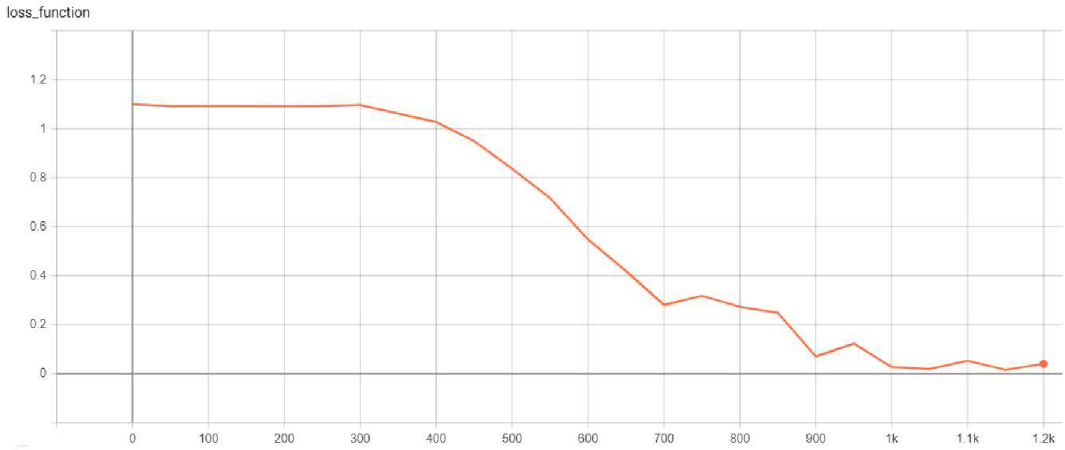
Trénovanie začínalo na menších sieťach, ktoré sa ukázali ako vhodná voľba pri prvých testovaniach, kedy sme používali veľké súčiastky ktoré zaberali väčšiu časť snímky. Zo začiatku sme používali menší dataset, na ktorom sme získali nežiadúce výsledky. Na tomto datasete sa boli siete schopné učiť, ale ako v prípade menších sietí na určenie farby, učenie neprebehlo úspešne a mimo tohto súboru boli siete nepoužiteľné. Chyba bola pravdepodobne vo veľkosti datasetu, keďže ani väčšie siete sa nedokázali naučiť správne parametre.



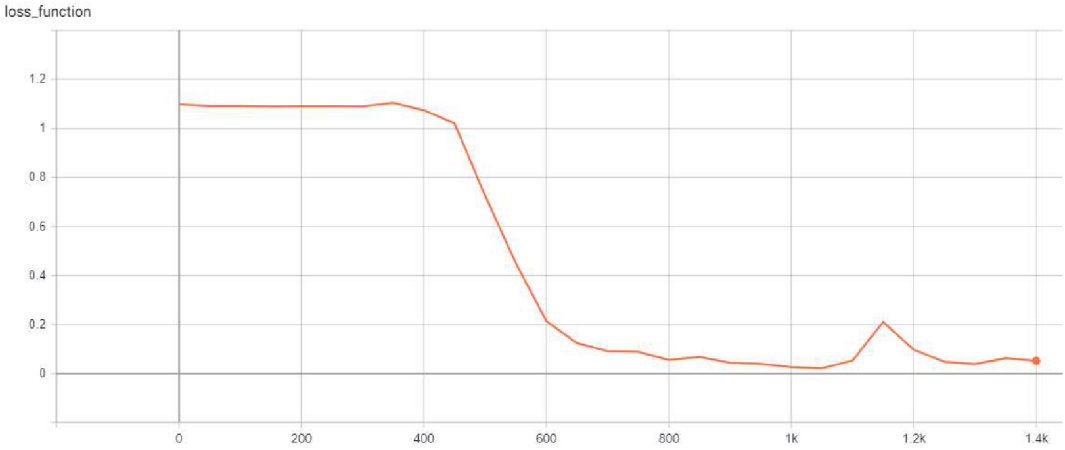
Obr. 38: Graf trénovania malej siete na rozpoznanie kategórie;
modrá – loss, oranžová – presnosť overovací

Priebeh bol aj pri iných, podobne veľkých sieťach, takmer zhodný. Je možné vidieť podobný priebeh ako pri malých sieťach na rozpoznanie farby s rozdielom pomalšieho učenia a o niečo menšej presnosti.

Trénovanie teda začalo prebiehať na datasete bez čiernych súčiastok. V tomto prípade pri použití menších sietí loss funkcia neklesala vôbec. Nakoniec teda boli použité naše upravené implementácie VGG a AlexNet. Pri VGG máme zobrazený graf siete detailne popísanej v kapitole 6.6.1. Trénovali sme aj ostatné zmienené úpravy VGG, ale výsledky boli rovnaké(jeden filter na 2x2) alebo loss neklesal. Pri tomto aj ostatných trénovaniach iných sietí bola snaha pracovať zhruba s rovnakou rýchlosťou učenia pre každú sieť, a meniť ju až pri neúspešnom učení. Pre tieto siete to bolo $1 \cdot 10^{-5}$.



Obr. 39: Priebeh trénovania – AlexNet



Obr. 40: Priebeh trénovania – VGG

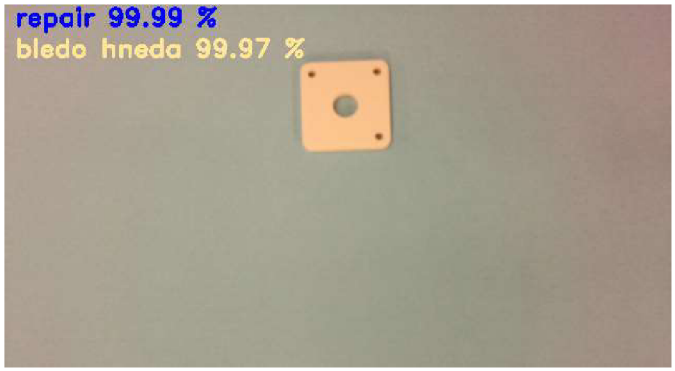
V prípade VGG sa aj napriek klesaniu loss funkcie a drobných úpravám nepodarilo zaistiť aby presnosť na overovacom súbore stúpla nad 0,4. Pri trénovaní AlexNet boli výsledky pozitívnejšie.

7.4.4 Výsledky modelu na určenie kategórie súčiastky

Upravený model siete AlexNet dosiahol počas trénovania pre nás dostatočné výsledky a presnosť pri overovacom súbore cez 75 %. Presnosť pri testovaní sa držala v podobných hodnotách. Pri súbore o veľkosti 117 snímok (bez čiernych) sieť uhádla kategórie 89 zo 117 snímok a teda dosiahla úspešnosť 76 %. Pri teste na 147 siedmich sieť uhádla 102 zo 147 snímok a teda mala presnosť 69 %. Hodnoty zo snímok ktoré majú tieto súbory zhodné sa v obidvoch prípadoch rovnajú, a teda sieť uhádla 13 čiernych snímok z 30. To je presnosť 43 %, čo je celkom blízko jednej tretine ktorá by mala zo siete vystúpiť pokiaľ nie je naučená (3 kategórie a teda 33 % šanca na správny typ). Tým pádom nie je možné rozhodnúť či bola sieť schopná využiť naučené váhy aj na čiernu súčiastku. Pri čiernej súčiastke bol však len jeden presný zásah v kategórii opraviteľných súčiastok a zhruba polovica presných zásahov v ostatných kategóriách a môžeme predsa len predpokladať,

že hlavným problémom v rozpoznaní sú ťažko viditeľné malé otvory. Najväčšia presnosť bola v kategórii „*bad*" ktorá je lepšie rozlíšiteľná. Kategória „*ok*" si tiež viedla dostatočne keďže niektoré čierne súčiastky v nej boli o niečo lepšie osvetlené. Nakoniec treba spomenúť že súčiastky v testovacom súbore sa líšia hlavne v polohe, pričom natočenie ostáva rovnaké, alebo sa líši len veľmi málo. Tento fakt mohol zlepiť výsledky siete.

Ako je vidieť na obr. 41 nižšie, sieť dokáže určiť niektoré súčiastky správne s vysokou presnosťou, ale stále sú tu aj nežiaduce opaky ktoré nastávajú kvôli niektorým nesprávne naučeným parametrom. V prípade hlbokých sietí je najúčinnejšiu obranou voči tomuto problému čo najväčší dataset.



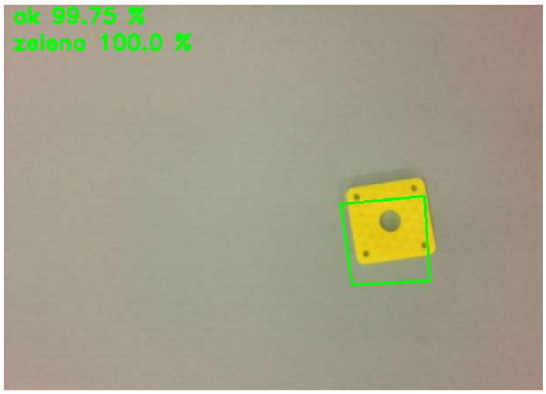
Obr. 41: Presné určenie kategórie

Obr. 42: Veľká istota pri nesprávnom určení a nízka pri správnom

7.5 Spojenie sietí

Modely mali byť skúšané na základnej doske a kamere Raspberry Pi na ktorej boli aj odfotené fotky, ktoré boli použité pri trénovaní. Pri pokuse o spustenie siete na tomto zariadení ale nastalo viacero problémom súvisiacich s nekompatibilitou použitých knižníc a frameworkov. Na Raspbiane (OS dosky Raspberry Pi) sa bohužiaľ nepodarilo naištalovať OpenCV a TensorFlow, ktoré sú nevyhnutné k chodu programu. Tento problém by mohla odstrániť inštalácia OS Ubuntu, ktorá sa používala na starších verziách Raspberry Pi ale nie je kompatibilná s modelom 3 B+.

Spojenie sietí teda nakoniec prebehlo len na osobnom počítači pri kontrole niekoľkých snímok. Z dosiahnutých vsledkov vyplíva, najpresnejšia bola sieť na určenie farby, ktorá pri niekoľkých testoch správne trafila farbu pri každom správnom alebo takmer správnom lokalizovaní súčiastky. Presnosti dalších dvoch sietí už boli horšie, ale pri správnom určení polohy dosahovala na klasifikáciu objektu výsledky podobné tým na testovacích súboroch.

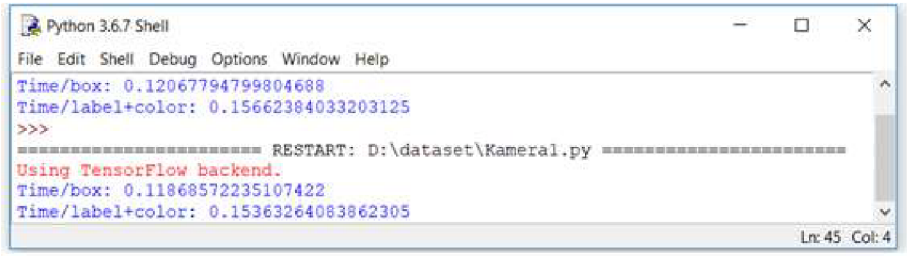


Obr. 43: Správne určenie všetkých troch kategórií

7.6 Zlepšenie výsledkov sietí

Vo všetkých prípadoch by pomohlo zväčšenie datasetu, ktoré je často univerzálnym riešením na väčšinu problémov pri nevyhovujúcich výsledkoch sietí. Je vidieť, že najlepšie výsledky má sieť, s najväčším tréningovým súborom. Pri lokalizácií objektu sme kvôli zdĺhavému trénovaniu na CPU bol použitý iba jeden model siete a teda by bolo vhodné meniť veľkosť siete a iné hyperparametre. Taktiež by stačilo zamerať sa na stred súčiastky namiesto okrajov, pretože orámovanie pre nás nie je príliš dôležitým údajom. Táto úprava by pravdepodobne mohla viesť k urýchleniu siete. Určenie farieb je veľmi presné a jeho výsledky sú na veľmi dobrej úrovni. Pri klasifikácii objektu by okrem zväčšenia datasetu mohlo pomôcť predspracovanie snímky ktoré by mohlo zvýrazniť dôležité hrany a podobne.

Najťažším problémom teda ostáva zlepšenie klasifikácie objektu. Pravdepodobne by sa dalo pracovať aj na zlepšení rýchlosti výpočtu, ale nevieme ako by bola snímka spracovaná na zariadeniach určených špeciálne na prácu s neurónovými sieťami, takže rýchlosť zatiaľ prehlásime za dostatočnú.



Obr. 44: Rýchlosť detekcie objektu, zistenia jeho triedy a farby[s]

8 ZÁVER

Úlohou tejto práce bolo posúdiť dnešný stav automatizovanej kontroly kvality v priemysle, hlbokého učenia, a spojiť tieto odvetvia pri kontrole zadanej súčiastky. Postupne bol teda spracovaný prehľad dnešných možností kontroly kvality, hlbokého učenia so zameraním na konvolučné neurónové siete na spracovanie obrazového vstupu, a frameworkov ktoré tento problém zjednodušujú. Nebude prekvapením že každé z týchto odvetví veľmi rýchlo napreduje a vyvíja sa.

Ku kontrole kvality pomocou strojového učenia bolo nutné vybrať si vhodný programovací jazyk a framework , ktorými sa stali Python a TensorFlow. Obidve varianty sú v dnešnej dobe veľmi populárne a sú užívateľsky prívetivé. Vďaka tejto voľbe bolo programovanie hlbokých sietí pomerne jednoduché. Snímky boli vyfotené pomocou malého počítača a kamery Raspberry Pi. Tieto zariadenia sú rozšírenou učebnou pomôckou a získanie snímok boli viac ako dostatočné. Zo získaných snímok bolo vytvorených niekoľko dátových súborov ktoré sme použili pri učení sietí. Rozhodli sme sa pomocou jednej neurónovej siete určiť polohu súčiastky, a pomocou tejto polohy získať menšiu snímku ktorú môžeme použiť na vstup do ďalších sietí na určenie správnosti výroby súčiastky a jej farby. Vďaka tejto možnosti je algoritmus o niečo rýchlejší a pravdepodobne aj presnejší. Rôzne siete sme trénovali pomocou centrálnej procesorovej jednotky aj grafickej jednotky. Vďaka tomu sme zistiť rozdiel v dĺžke učenia siete. Ako už bolo dopredu jasné trénovanie na grafickej jednotke bolo výrazne rýchlejšie. Počas trénovania sme otestovali viacero rôznych modelov sietí, čo je najťažšia časť práce s neurónovými sieťami, keďže to, či ste zvolil alebo nezvolil správne riešenie viete až pod dlhej dobe trénovania.

Vo výsledku sme získali tri siete z ktorých jedna dokáže určiť farbu súčiastky s veľmi vysokou presnosťou, a dve siete na určenie polohy a klasifikáciu objektu s horšími výsledkami. Pri určení polohy bol pravdepodobne problémom nízky počet snímok na vytrénovanie siete, ktorý bol aj viac ako štvornásobne nižší v porovnaní s ostatnými tréningovými súbormi. Tento problém je ľahko odstrániteľný zväčšením tohto súboru, čo je jednoduchá, ale keďže musíme označovať polohu objektu, časovo náročná úloha. Vhodné by bolo otestovať aj rozdielne modeli s inou veľkosťou. Najväčším problémom však ostáva určenie triedy súčiastky, čo je kvôli malým rozdielom v súčiastkach pre sieť najťažšia úloha.

9 ZOZNAM POUŽITEJ LITERATÚRY

[1] Types of 3D Scanners and 3D Scanning Technologies. EMS [online]. 2018 [cit. 2019-04-28]. Dostupné z: https://www.ems-usa.com/tech-papers/3D%20Scanning%20Technologies%20.pdf

[2] SCHUON, Sebastian, THEOBALT, Christian, DAVIS, James, THRUN, Sebastian. *High-Quality Scanning Using Time-Of-Flight Depth Superresolution*. [online]. 2008 [cit. 2019-04-28]. Dostupné z: https://cs.stanford.edu/people/theobalt/TOF_CV_Superresolution_ final.pdf.

[3] GOKTURK, Burak S., YALCIN, Hakan, BAMJI, Cirus. *A Time-Of-Flight Depth Sensor – System Description, Issues and Solutions*. [online]. 2004 [cit. 2019-04-28]. Dostupné z: https://web.archive.org/web/20070623233559/http://www.canesta.com/assets/pdf/technicalpapers/CVPR_Submission_TOF.pdf

[4] FRANÇA, João Guilherme, GAZZIRO, Mario, IDE, Alessandro, SAITO, José H. *A 3D scanning system based on laser triangulation and variable field of view*. [online]. 2005 [cit. 2019-04-28]. Dostupné z: https://www.researchgate.net/publication/4186151_A_3D_scanning_system_based_on_laser_triangulation_and_variable_field_of_view

[5] MOSTAFA, Ebrahim Abdel-Bary. *3D LASER SCANNERS: HISTORY, APPLICATIONS, AND FUTURE*. [online]. 2011 [cit. 2019-04-28]. Dostupné z: https://www.researchgate.net/publication/267037683_3D_LASER_SCANNERS_HISTORY_APPLICATIONS_AND_FUTURE

[6] ABDELHAFIZ, Ahmed. *Integrating Digital Photogrammetry and Terrestrial Laser Scanning*. [online]. 2009 [cit. 2019-04-29]. Dostupné z: https://www.researchgate.net/publication/278157315_Integrating_Digital_Photogrammetry_and_Terrestrial_Laser_Scanning

[7] CHUA, Chee Kai, WONG, Chee How, YEONG, Wai Yee. *Standards, Quality Control, and Measurement Sciences in 3D Printing and additive manufacturing*. [online]. Elsevier Science, 2017 [cit. 2019-04-29]. ISBN-978-0-12-813489-4. Dostupné z: https://books.google.cz/books?isbn=0128134909

[8] ALEXANDREA P. *3D scanning through structured light projection*. [online]. 2017 [cit. 2019-04-29]. Dostupné z: https://www.3dnatives.com/en/structured-light-projection-3d-scanning/

[9] SINHA, G. R. *Optical Sensor: Photomultipliers and CCD Sensors*. [online]. 2018 [cit. 2019-05-01]. Dostupné z: https://www.researchgate.net/publication/322198310_Optical_Sensor_Photomultipliers_and_CCD_Sensors

[10] HOSNY, Mouaz Mhd Zafer al. *CMOS camera sensors*. [online]. 2017 [cit. 2019-05-01]. Dostupné z: https://www.researchgate.net/publication/320740980_CMOS_camera_sensors

[11] SIORDIA, Oscar S., DIEGO, Isaac Martín de, CONDE, Cristina, CABELLO, Enrique. *Wireless In-vehicle Complaint Driver Environment Recorder*. [online]. 2011 [cit. 2019-05-01]. Dostupné z: https://www.researchgate.net/publication/221050957_Wireless_In-vehicle_Complaint_Driver_Environment_Recorder

[12] Photoneo [online]. [cit. 2019-05-12]. Dostupné z: https://www.photoneo.com/

[13] Sazenoo [online]. [cit. 2019-05-12]. Dostupné z: https://www.sanezoo.com/

[14] Raspberry Pi [online]. [cit. 2019-05-02]. Dostupné z: https://www.raspberrypi.org/

[15] Nvidia Jetson [online]. [cit. 2019-05-18]. Dostupné z: https://developer.nvidia.com/

[16] Convolutional Neural Networks [online]. [cit. 2019-03-23]. Dostupné z: http://cs231n.github.io/convolutional-networks/

[17] MIJWEL, Maad M. *Pattern Recognition and Neural Networks* [online]. 2017 [cit. 2019-03-23]. Dostupné z: https://www.researchgate.net/publication/322632189_ Pattern_Recognition_and_Neural_Networks

[18] NWANKPA, Chigozie, IJOMAH, Winifred, GACHAGAN, Anthony, MARSHALL, Stephen. *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning* . [online]. 2018 [cit. 2019-03-23]. Dostupné z: https://arxiv.org/pdf/1811.03378.pdf

[19] Al-BDOUR, Ghadeer, AL-QURRAN, Raffi, AL-AYYOUB, Mahmoud, SHATNAWI, Ali. *A detailed comparative study of open source deep learning frameworks*. [online]. 2019 [cit. 2019-03-23]. Dostupné z: https://arxiv.org/pdf/1903.00102.pdf

[20] GERSHENSON, Carlos. *Artificial Neural Networks for Beginners*. [online]. 2003 [cit. 2019-05-23]. Dostupné z: https://arxiv.org/ftp/cs/papers/0308/0308031.pdf

[21] GOODFELLOW, Ian, BENGIO, Yoshua, COURVILLE, Aaron. *Deep Learning*. [online]. MIT Press, 2016 [cit. 2019-03-24]. Dostupné z: http://www.deeplearningbook.org/

[22] WEI, Kai. *Convolutional Neural Network max pooling visualization*. [online]. 2018 [cit. 2019-03-24]. Dostupné z: https://kwei7.quora.com/Convolutional-Neural-Network-max-pooling-visualization

[23] YU, Wei, YANG, Kuiyuan, BAI, Yalong, YAO, Hongxun, RUI, Yong. *Visualizing and Comparing Convolutional Neural Networks*. [online]. 2014 [cit. 2019-03-26]. Dostupné z: https://arxiv.org/pdf/1412.6631.pdf

[24] GOLDSBOROUGH, Peter. *A Tour of TensorFlow*. [online]. 2016 [cit. 2019-03-26]. Dostupné z: https://arxiv.org/pdf/1610.01178.pdf

[25] ABADI, Martín, AGARWAL, Ashish, BARHAM, Paul, BREVDO, Eugene, CHEN, Zhifeng, CITRO, Craig, CORRADO, Greg S., DAVIS, Andy, DEAN, Jeffrey, DEVIN, Matthieu, GHEMAWAT, Sanjay, GOODFELLOW, Ian, HARP, Andrew, IRVING, Geoffrey, ISARD, Michael, JOZEFOWICZ, Rafal, JIA, Yangqing, KAISER, Lukasz, KUDLUR, Manjunath, LEVENBERG, Josh, MANÉ, Dan, SCHUSTER, Mike, MONGA, Rajat, MOORE, Sherry, MURRAY, Derek, OLAH, Chris, SHLENS, Jonathon, STEINER, Benoit, SUTSKEVER, Ilya, TALWAR, Kunal, TUCKER, Paul, VANHOUCKE, Vincent, VASUDEVAN, Vijay, VIÉGAS, Fernanda, VINYALS, Oriol, WARDEN, Pete, WATTENBERG, Martin, WICKE, Martin, YU, Yuan, ZHENG, Xiaoqiang. *TensorFlow: Large-scale machine learning on heterogeneous systems*. [online]. 2015. [cit. 2019-03-26]. Dostupné z: http://download.tensorflow.org/paper/whitepaper2015.pdf

[26] Tensorflow [online]. [cit. 2019-03-29]. Dostupné z: https://www.tensorflow.org/

[27] BERGSTRA, James, BREULEUX, Olivier, BASTIEN, Frédéric, LAMBLIN, Pascal, PASCANU, Razvan, DESJARDINS, Guillaume, TURIAN, Joseph, WARDE-FARLEY, David, BENGIO, Yoshua, *Theano: A CPU and GPU Math Compiler in Python*. [online]. 2010 [cit. 2019-03-29]. Dostupné z: http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010 .pdf

[28] LAMBLIN, Pascal. *MILA and the future of Theano*. [online]. 2017 [cit. 2019-03-29]. Dostupné z: https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY

[29] The Microsoft Cognitive Toolkit [online]. [cit. 2019-03-29]. Dostupné z: https://docs.microsoft.com/en-us/cognitive-toolkit/

[30] Keras [online]. [cit. 2019-03-29]. Dostupné z: https://keras.io/

[31] Pytorch[online]. [cit. 2019-03-29]. Dostupné z: https://github.com/pytorch

[32] Caffe [online]. [cit. 2019-03-29]. Dostupné z: http://caffe.berkeleyvision.org/

[33] Caffe2 [online]. [cit. 2019-03-29]. Dostupné z: https://caffe2.ai

[34] Caffe2 [online]. [cit. 2019-03-29]. Dostupné z: https://research.fb.com/downloads/caffe2/

[35] Apache MXNet [online]. [cit. 2019-03-30]. Dostupné z: https://mxnet.apache.org/

[36] Deep Learning for Java [online]. [cit. 2019-03-30]. Dostupné z: https://deeplearning4j.org/

[36] Chainer [online]. [cit. 2019-03-30]. Dostupné z: https://github.com/chainer/chainer

[38] HOWARD, Jeremy. *Introducing Pytorch for fast.ai*. [online]. 2017 [cit. 2019-03-30]. https://www.fast.ai/2017/09/08/introducing-pytorch-for-fastai/

[39] Nvidia cuDNN [online]. [cit. 2019-03-30]. Dostupné z: https://developer.nvidia.com /cudnn

[40] Wolfram mathematica [online]. [cit. 2019-03-30]. Dostupné z: https://www.wolfram.com /mathematica/

[41] Matlab [online]. [cit. 2019-03-30]. Dostupné z: https://uk.mathworks.com/products/matlab .html

[42] HALLE, Jeff. *Deep Learning Framework Power Scores 2018*. [online]. 2018 [cit. 2019-04-02]. Dostupné z: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

[43] KARMANOV, Ilia. *DeepLearningFrameworks*. [online]. 2018 [cit. 2019-04-02]. https://github.com/ilkarman/DeepLearningFrameworks

[44] HUBER, Michal. Detekce objektu pro průmyslového robota s využitím počítačového vidění. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Ing. Roman Parák.

[45] Sizeof Dev and Test Sets. [online]. 25.8.2017 [cit. 2019-05-17]. Dostupné z: https://www.youtube.com/watch?v=_Fe5kKmFieg

[46] KRIZHEVSKY, Alex, SUTSKEVER, Ilya, HINTON, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks*. [online]. 2012 [cit. 2019-05-18]. Dostupné z: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[47] SIMONYAN, Karen, ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. [online]. 2015 [cit. 2019-05-18]. https://arxiv.org /pdf/1409.1556.pdf

10 ZOZNAM OBRÁZKOV A TABULIEK

Zoznam obrázkov

Zoznam tabuliek

11 ZOZNAM PRÍLOH

A: CD-ROM

Tab. 1: Obsah CD

Zložka	Súbor
Bakalárska práca	BP_Gaško_193105.pdf
code	CNN_classification.py CNN_color.py CNN_localization.py model.py rotate_gauss_filter.py
values	bias_box.ckpt.data-00000-of-00001 bias_box.ckpt.index bias_box.ckpt.meta bias_classification.ckpt.data-00000-of-00001 bias_classification.ckpt.index bias_classification.ckpt.meta bias_color.ckpt.data-00000-of-00001 bias_color.ckpt.index bias_color.ckpt.meta weights_box.ckpt.data-00000-of-00001 weights_box.ckpt.index weights_box.ckpt.meta weights_classification.ckpt.data-00000-of-00001 weights_classification.ckpt.index weights_classification.ckpt.meta weights_color.ckpt.data-00000-of-00001 weights_color.ckpt.index weights_color.ckpt.meta
train_class	0.png až 903.png
train_color	0.png až 1220.png
train_localization	0.png až 264.png
test1	0.png až 116.png
test2	0.png až 146.png