

Demonstrační projekt – Lego EV3 robot ovládaný gesty

Michal Šichman



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Michal Šichman**
Osobní číslo: **A22033**
Studijní program: **B0613A140020 Softwarové inženýrství**
Forma studia: **Prezenční**
Téma práce: **Demonstrační projekt – Lego EV3 robot ovládaný gesty**
Téma práce anglicky: **Demonstration project – A Lego EV3 hand gesture controlled robot**

Zásady pro vypracování

1. Popište problematiku počítačového vidění a detekce gest.
2. Popište možnosti ovládání řídicí jednotky Lego EV3.
3. Vyberte vhodnou metodu pro detekci gest a implementujte ji.
4. Sestavte robota řízeného jednotkou EV3 s možností pohybu.
5. Vytvořte rozhraní pro ovládání robota pomocí gest.
6. Řádně zdokumentujte výsledný software.

Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. DAVIES, E. R. Computer vision: principles, algorithms, applications, learning. Fifth edition. London: Academic Press, 2018. ISBN 9780128095751. Dostupné také z: <https://proxy.k.utb.cz/login?url=https://app.knovel.com/hotlink/toc/id:kpCVPAALE3/computer-vision-principles?kpromoter=marc>.
2. BISHOP, Christopher M. Pattern recognition and machine learning. Information science and statistics. New York: Springer, c2006. ISBN 0387310738.
3. MATTHES, Eric. Python crash course: a hands-on, project-based introduction to programming. 2nd edition. San Francisco: No Starch Press, [2019]. ISBN 978-1-59327-928-8.
4. VALK, Laurens. The LEGO Mindstorms EV3 discovery book: a beginner's guide to building and programming robots. San Francisco: No Starch Press, [2014]. ISBN 9781593275327.
5. PAJANKAR, Ashwin a JOSHI, Aditya. Hands-on machine learning with Python: implement neural network solutions with Scikit-learn and PyTorch. [Berkeley]: Apress, [2022]. Dostupné z: <https://doi.org/9781484279212>.

Vedoucí bakalářské práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....

podpis autora

ABSTRAKT

Cieľom bakalárskej práce je vytvoriť projekt pre laboratórium umelej inteligencie. Tento projekt bude slúžiť ako demonštrácia počítačového videnia a detekcie gest. Výsledný software bude schopný z kamerových snímok detekovať a následne vyhodnotiť predvolené gestá a pomocou naprogramovaného rozhrania komunikovať s LEGO EV3 robotom, ktorý na základe identifikovaných gest vykoná príslušné akcie. Návrh aplikácie je koncipovaný s ohľadom na škálovateľnosť, čo umožňuje jednoduchú úpravu súboru detekovateľných gest, a teda flexibilné prispôsobenie riešenia pre iné úlohy. Práca zároveň približuje princípy a fungovanie počítačového videnia.

Kľúčové slová: počítačové videnie, rozpoznávanie gest, ovládanie robota, LEGO EV3

ABSTRACT

The aim of this bachelor's thesis is to create a project for an artificial intelligence laboratory. This project will serve as a demonstration of computer vision and gesture detection. The resulting software will be capable of detecting and evaluating predefined gestures from camera images and, through a programmed interface, communicating with a LEGO EV3 robot, which will perform corresponding actions based on the identified gestures. The application is designed with scalability in mind, allowing for easy modification of the set of detectable gestures, thus enabling flexible adaptation of the solution for other tasks. The thesis also explains the principles and functioning of computer vision.

Keywords: computer vision, gesture recognition, robot control, LEGO EV3

POĎAKOVANIE

Rád by som vyjadril úprimné poďakovanie svojej rodine za neustálu podporu, trpezlivosť a povzbudenie počas celého štúdia. Ich podpora bola pre mňa neoceniteľným zdrojom motivácie.

Zároveň by som chcel poďakovať Ing. Adamovi Viktorinovi, Ph.D. za odborné vedenie a venovaný čas pri písaní tejto práce. Jeho prístup a ochota mi výrazne pomohli pri jej vypracovaní.

Prehlasujem, že pri tvorbe tejto práce som použil nástroj generatívneho modelu AI ChatGPT (chat.openai.com) za účelom asistencie pri formálnych úpravách textu vrátane opravy gramatiky a zlepšenia štylistiky, ako aj na dokumentáciu kódu. Po použití tohto nástroja som vykonal kontrolu obsahu a preberám zaň plnú zodpovednosť.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČASŤ	12
1 POČÍTAČOVÉ VIDENIE.....	13
1.1 DEEP LEARNING	13
1.2 CONVOLUTIONAL NEURAL NETWORKS	13
1.2.1 Convolutional layer	13
1.2.2 Pooling layer	20
1.2.3 Flatten layer.....	21
1.2.4 Fully-connected layer.....	21
1.3 MODEL Y POČÍTAČOVÉHO VIDENIA	22
1.4 ÚLOHY MODELOV POČÍTAČOVÉHO VIDENIA	23
1.5 TRÉNING CV MODELOV.....	24
1.6 TRANSFER LEARNING	26
1.7 POUŽITIE POČÍTAČOVÉHO VIDENIA	27
2 DETEKCIA GEST	28
2.1 MEDIAPIPE HANDS	29
2.2 MODEL ROZPOZNANIA GEST S POUŽITÍM TENSORFLOW	30
3 OVLÁDANIE JEDNOTKY LEGO EV3	31
3.1 PROGRAMOVANIE ROBOTA	32
3.2 OPERAČNÝ SYSTÉM EV3DEV	33
II PRAKTICKÁ ČASŤ	34
4 PRÍPRAVA ROBOTA.....	35
5 ARCHITEKTÚRA A TRÉNING MODELU NA ROZPOZNANIE GEST	37
5.1 ZBER A PRÍPRAVA DÁT NA TRÉNOVANIE	37
5.2 ARCHITEKTÚRA MODELU	38
5.3 TRÉNING MODELU	39
6 PRINCÍP FUNGOVANIA KOMUNIKÁCIE S ROBOTOM A PRENOS SÚBOROV	44
6.1 PRIPOJENIE A KOMUNIKÁCIA S ROBOTOM.....	44
6.2 PRENOS SÚBOROV	45
7 NÁVRH A IMPLEMENTÁCIA APLIKÁCIE	47
7.1 SIEŤOVÁ KOMUNIKÁCIA POMOCOU TCP SOCKETOV.....	47
7.2 PROGRAM RIADIACEHO ZARIADENIA	49
7.3 PROGRAM ROBOTA	51
ZÁVER	53

ZOZNAM POUŽITEJ LITERATÚRY	55
ZOZNAM OBRÁZKOV	60
ZOZNAM TABULIEK.....	61
ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK	62

ÚVOD

V posledných desaťročiach zaznamenala oblasť počítačového videnia výrazný pokrok, pričom jednou z jej kľúčových aplikácií je detekcia a rozpoznávanie gest. Táto technológia nachádza uplatnenie v rôznych oblastiach, ako sú bezdotykové používateľské rozhrania, asistenčné technológie pre zdravotne znevýhodnené osoby, herný priemysel či inteligentné systémy riadenia. Možnosť interakcie s počítačmi alebo inými digitálnymi systémami pomocou pohybov rúk predstavuje významný míľnik vo vývoji interakcie medzi človekom a počítačom.

Cieľom tejto práce je vytvoriť software na rozpoznávanie gest a implementovať rozhranie pre komunikáciu s robotom LEGO EV3. Výsledný software je vytvorený v programovacom jazyku Python s využitím technológií OpenCV, MediaPipe a TensorFlow.

Práca je rozdelená na dve časti – teoretickú a praktickú. Teoretická časť oboznamuje čitateľa s princípmi fungovania moderných metód počítačového videnia a detekcie gest. Praktická časť opisuje postup pri implementácii softwaru na komunikáciu a detekciu gest.

I TEORETICKÁ ČASŤ

1 POČÍTAČOVÉ VIDENIE

Počítačové videnie (angl. *Computer Vision*, skratene CV) je odbor umelej inteligencie (angl. *Artificial Intelligence*), ktorý umožňuje strojom interpretovať vizuálne informácie získané z obrázkov a videí, čím im poskytuje schopnosť „vidieť“ a porozumieť vizuálnemu obsahu. Úlohou CV je identifikovať, analyzovať, klasifikovať a efektívne vyhodnocovať objekty a vzory s cieľom replikovať ľudské schopnosti videnia. Počítačové videnie vychádza z princípov ľudského zraku, ktorý je vyvinutý na rozpoznávanie tvarov, vzorov a objektov. Na dosiahnutie tejto pokročilej úrovne spracovania obrazu sa v CV využívajú sofistikované metódy deep learningu, najmä algoritmy známe ako konvolučné neurónové siete (angl. *Convolutional Neural Networks*), ktoré umožňujú efektívne spracovanie a analýzu vizuálnych dát.[1]

1.1 Deep Learning

Deep learning je podmnožina strojového učenia (angl. *Machine Learning*), ktorá využíva umelé neurónové siete s viacerými vrstvami na analýzu a učenie sa z dát. Počet skrytých vrstiev v neurónovej sieti sa označuje ako jej „hlbka“. Táto technika učenia je mimoriadne efektívna pri rozpoznávaní vzorcov vo veľkých objemoch dát.[2]

1.2 Convolutional Neural Networks

Skratene označované ako ConvNets alebo CNNs sú špeciálny typ hlbokých neurónových sietí, ktoré sú mimoriadne efektívne pri analýze obrázkov a videí. Na identifikáciu vzorov v obraze sa využíva najmä maticová operácia známa ako konvolúcia. CNNs sa skladajú z troch hlavných typov vrstiev, a to sú konvolučná vrstva (angl. *Convolutional Layer*), združovacia vrstva (angl. *Pooling layer*) a plne prepojená vrstva (angl. *Fully-Connected Layer*).[3]

1.2.1 Convolutional layer

Konvolučná vrstva je prvou vrstvou v CNN a zároveň predstavuje hlavnú časť, v ktorej prebieha väčšina výpočtov. Po nej nasledujú ďalšie konvolučné alebo združovacie vrstvy a na konci siete je plne prepojená vrstva. Obraz je prevedený na číselné hodnoty, čo umožňuje neurónovej sieti efektívne interpretovať vzory.[3]

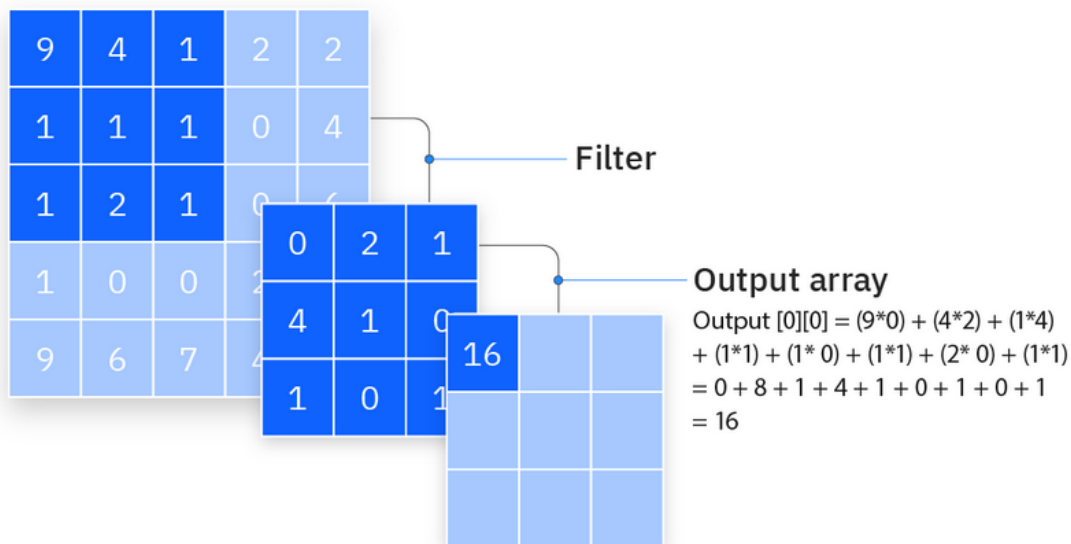
Každá konvolučná vrstva pozostáva z troch kľúčových komponentov:

1. Vstupné údaje – ide o 2D alebo 3D obraz reprezentovaný ako matica hodnôt pixelov, prípadne výstupné údaje z predchádzajúcej vrstvy siete.

2. Filter – matica s hodnotami váh, známa aj ako kernel. Každá vrstva siete môže obsahovať viacero filtrov, pričom každý má vlastné váhy.
3. Výstupné údaje – tzv. feature map, ktorá vzniká ako výsledok aplikácie filtra na vstupné údaje počas procesu konvolúcie.[3]

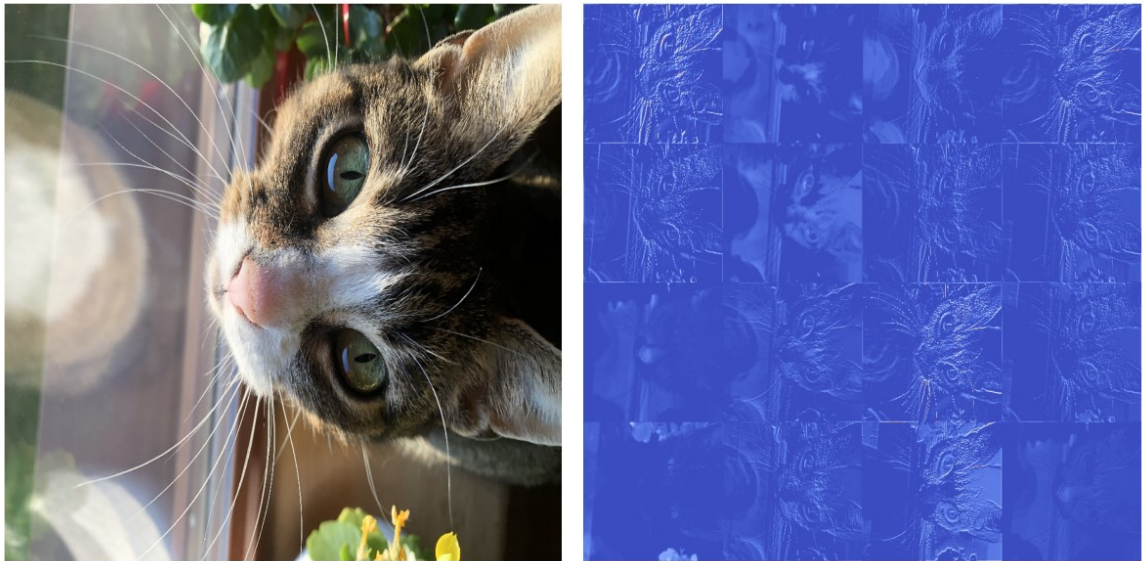
Proces konvolúcie spočíva v aplikáciách filtrov na vstupné dáta (obraz). Postupným posúvaním filtra sa na každom mieste vypočíta bodový súčin filtra s príslušnou oblasťou vstupu (Obr. 1). Výsledkom tejto operácie je súbor hodnôt, ktoré tvoria feature map. Tento proces sa opakuje pre každý filter, t. j. počet výstupných feature maps sa rovná počtu filtrov aplikovaných na vstupné dáta.[4]

Input image



Obr. 1 – Proces konvolúcie[3]

V konvolučnej vrstve môže byť paralelne aplikovaných viacero filtrov. Úlohou filtrov je extrahovať unikátne vlastnosti zo vstupných dát. Každý filter je zameraný na detekciu iného prvku alebo vzoru, napríklad okraje, hrany, textúry, tvary alebo časti objektov (Obr. 2).



Obr. 2 – Feature maps po aplikácii 16 filtrov

Feature maps v konvolučných neurónových sieťach sa pri prechode jednotlivými vrstvami stávajú čoraz zložitejšími a abstraktnejšími. Nižšie vrstvy rozpoznávajú základné prvky, ako sú hrany a rohy, zatiaľ čo vyššie vrstvy sa učia identifikovať komplexnejšie vzory, napríklad časti objektov alebo celé objekty (Obr. 3).[5]

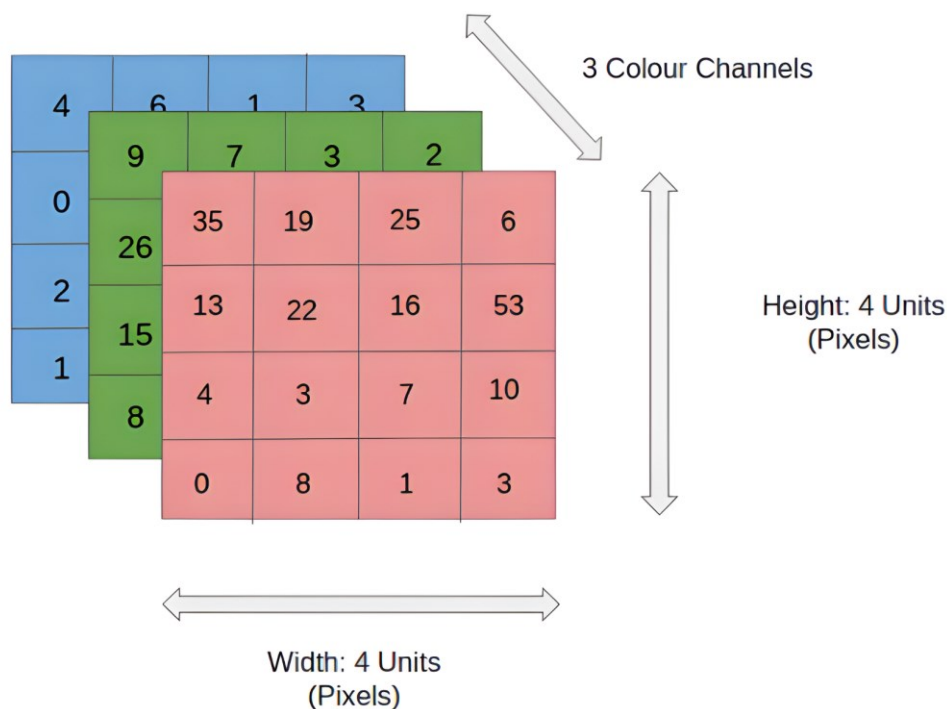


Obr. 3 – Detekcia častí objektu až po celý objekt[3]

Rozmery filtrov

Veľkosť filtrov (výška a šírka) sú hyperparametre siete – vid' konvolučné hyperparametre. Pri konvolúcii je potrebný aj tretí rozmer, označovaný ako kanál c (channel). Počet kanálov,

alebo tiež hĺbka filtra, sa rovná počtu dimenzií vstupnej matice. Napríklad pri RGB obrázku vstup obsahuje 3 kanály, z ktorých každý zodpovedá jednej farebnej zložke (Obr. 4).

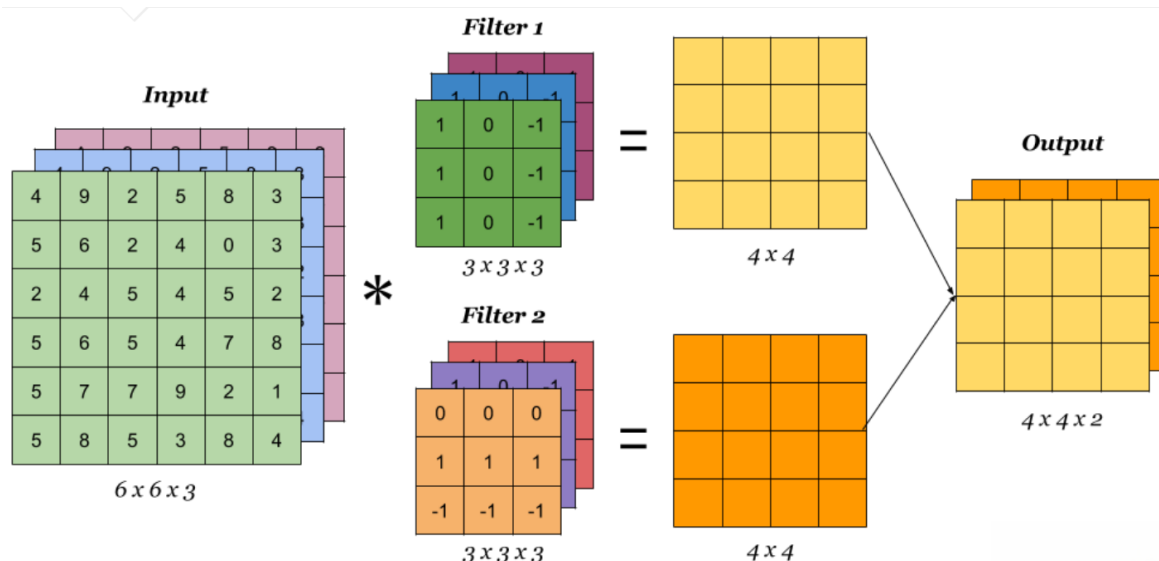


Obr. 4 – RGB kanály obrázku[6]

Proces konvolúcie v konvolučnej neurónovej sieti začína tým, že prvá konvolučná vrstva prijíma vstupný obraz vo forme 3D matice s rozmermi $w \times h \times c$, kde w a h predstavujú šírku a výšku obrazu a c počet farebných kanálov (RGB).

Filter v tejto vrstve má rozmery $f \times f \times c$, kde f označuje veľkosť filtra a c počet jeho kanálov – tento počet musí zodpovedať počtu kanálov vstupného obrazu. Každý kanál filtra sa aplikuje (konvoluje) samostatne na zodpovedajúci kanál vstupu (Obr. 5).[7]

Výsledné hodnoty z jednotlivých kanálov sa následne sčítajú, čím vznikne jedna časť výslednej feature map. Tento postup sa opakuje aj v ďalších konvolučných vrstvách, pričom vstupom už nie je obraz, ale feature maps z predchádzajúcej vrstvy. V takomto prípade platí, že počet kanálov každého filtra musí zodpovedať počtu vstupujúcich feature maps.[7]



Obr. 5 – Proces konvolúcie pri RGB obrázkoch[7]

Tab. 1 – Príklad dimenzií filtrov jednotlivých konvolučných vrstiev

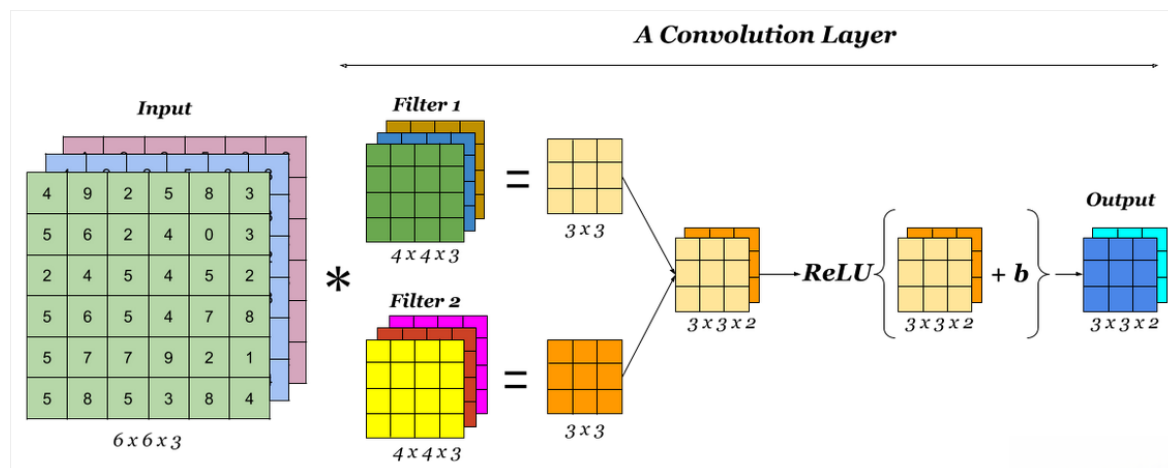
Konvolúcia	Výška filtra	Šírka filtra	Počet kanálov (Vstupujúcich feature maps)	Počet filtrov
1	3	3	3	64
2	3	3	64	128
3	3	3	128	256
4	3	3	256	512

Aplikácia aktivačnej funkcie

Hodnoty feature maps možno chápať ako neuróny v klasickej neurónovej sieti, pričom váhy týchto neurónov sú reprezentované váhami filtra. Tieto neuróny sú priestorovo prepojené s časťou vstupných údajov, nad ktorými bol vykonaný proces konvolúcie. Váhy každého neurónu zodpovedajú váham aplikovaného filtra.[8]

Po vykonaní konvolúcie (t. j. aplikácii filtrov) sa pripočíta bias filtra a následne sa aplikuje aktivačná funkcia (Obr. 6), ktorá zavádza do siete nelinearitu. Najčastejšie používanou aktivačnou funkciou je ReLU (Rectified Linear Unit), definovaná nasledovne: $f(x) = \max(0, x)$. Aktivačná funkcia rozhoduje, či bude daný neurón „aktivovaný“ alebo nie, a tým umožňuje modelu učiť sa komplexné, nelineárne vzory.[9]

Nelinearita znamená, že vzťah medzi vstupom a výstupom nie je priamo úmerný – malé zmeny vo vstupe môžu viesť k výrazným zmenám vo výstupe. Táto vlastnosť umožňuje sieti zachytávať skryté súvislosti a zložité vzory, ktoré by lineárny model nedokázal rozpoznať. Po aplikácii aktivačnej funkcie sa výstupy niekedy označujú ako activation maps.[10]



Obr. 6 – Pripočítanie biasu a aplikácia nelineárnej funkcie ReLU[7]

Úprava parametrov filtrov

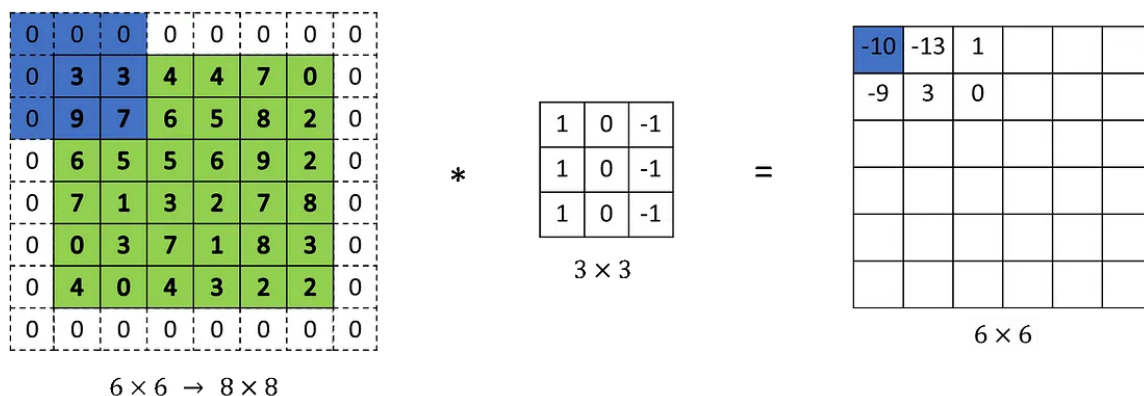
Váhy a biasy filtrov sa počas trénovania siete upravujú pomocou algoritmov backpropagation a gradient descent. Úlohou backpropagation je vypočítať gradienty, teda určiť, o koľko a ktorým smerom sa majú parametre upraviť, aby sa minimalizovala chyba. Následne gradient descent využíva tieto gradienty na aktualizáciu váh a biasov, čím sa postupne zlepšuje výkon (presnosť) siete. V praxi sa často používajú optimalizované verzie gradientného zostupu, ako napríklad SGD (Stochastic Gradient Descent) alebo Adam (Adaptive Moment Estimation).[11]

Konvolučné hyperparametre

Konvolučné hyperparametre slúžia na konfiguráciu a riadenie správania konvolučných vrstiev. Tieto parametre sa definujú ešte pred začiatkom trénovania siete a výrazne ovplyvňujú jej architektúru aj výsledný výkon. Tieto hyperparametre je potrebné nastaviť tak, aby výstupné rozmery jednotlivých vrstiev bolo možné efektívne spracovať v nasledujúcich vrstvách siete.[12]

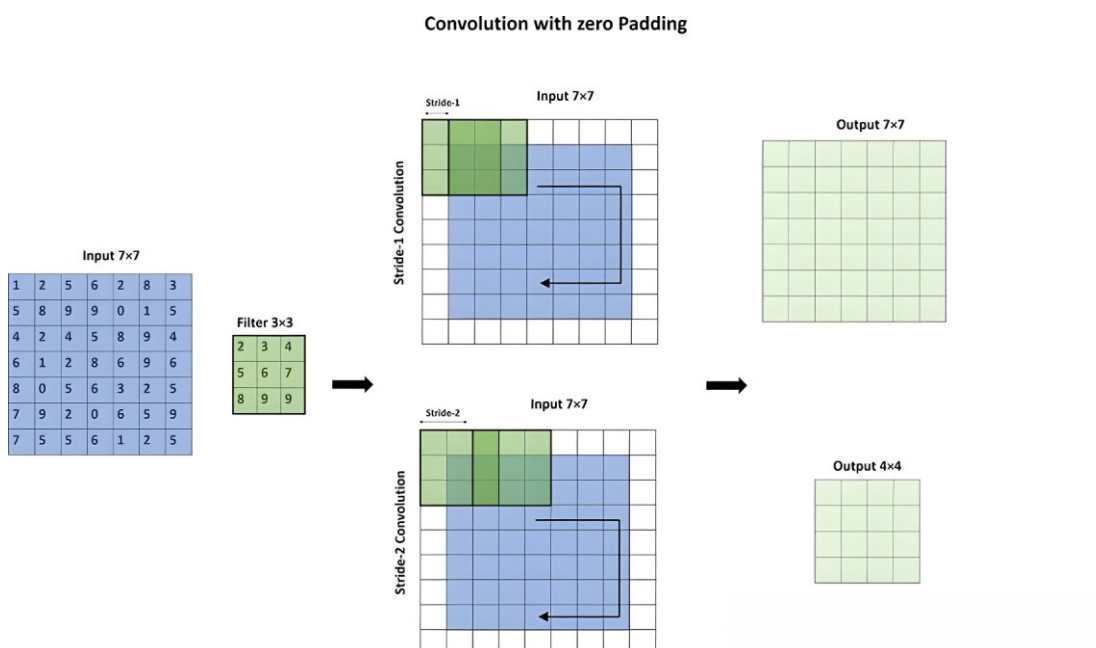
Medzi hlavné hyperparametre patria:

- Počet a veľkosť filtrov – najčastejšie matice o veľkosti 3×3 , 5×5 alebo 7×7 . Táto veľkosť určuje recepčné pole danej konvolučnej vrstvy. Väčší počet filtrov môže zachytiť zložitejšie vlastnosti a vzory, no zároveň zvyšuje výpočtovú náročnosť siete.
- Padding – slúži k zmene a kontrole priestorových rozmerov feature maps na požadovanú veľkosť a pomáha predchádzať strate informácií na okrajoch vstupu. To je docielené pridaním núl po okrajoch pred začatím procesu konvolúcie (Obr. 7).



Obr. 7 – Pridanie núl po okrajoch[14]

- Stride – tento hyperparameter určuje, o koľko sa filter posunie počas konvolúcie po vstupných dátach (Obr. 8). Najčastejšie používanou hodnotou je stride veľkosti 1.[13, 14]



Obr. 8 – Použitie hyperparametru Stride[15]

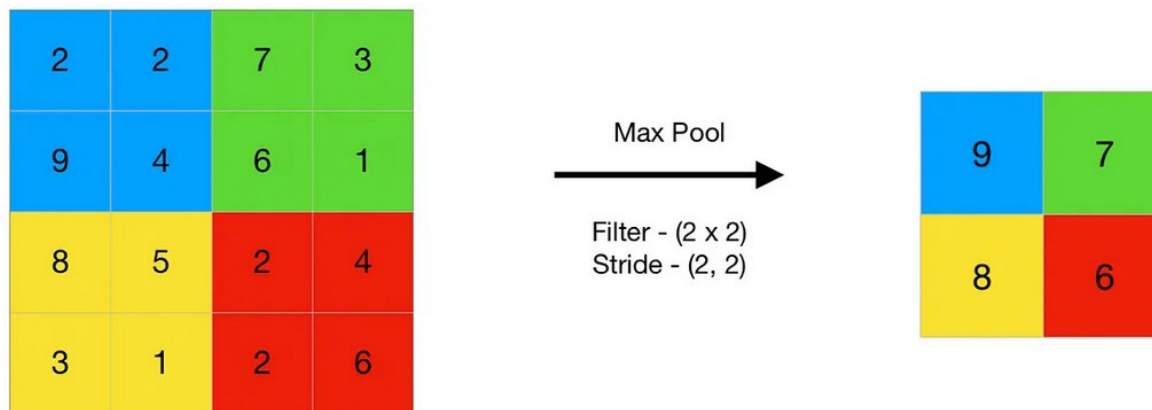
Dimenzie feature maps konvolučnej vrstvy

Veľkosť je udávaná vzorcom: $O = \frac{I+2P-K}{S} + 1$. Tento vzorec platí za predpokladu, že mapa má štvorcový tvar. Ak nie, výpočet sa aplikuje osobitne na výšku a šírku.[15]

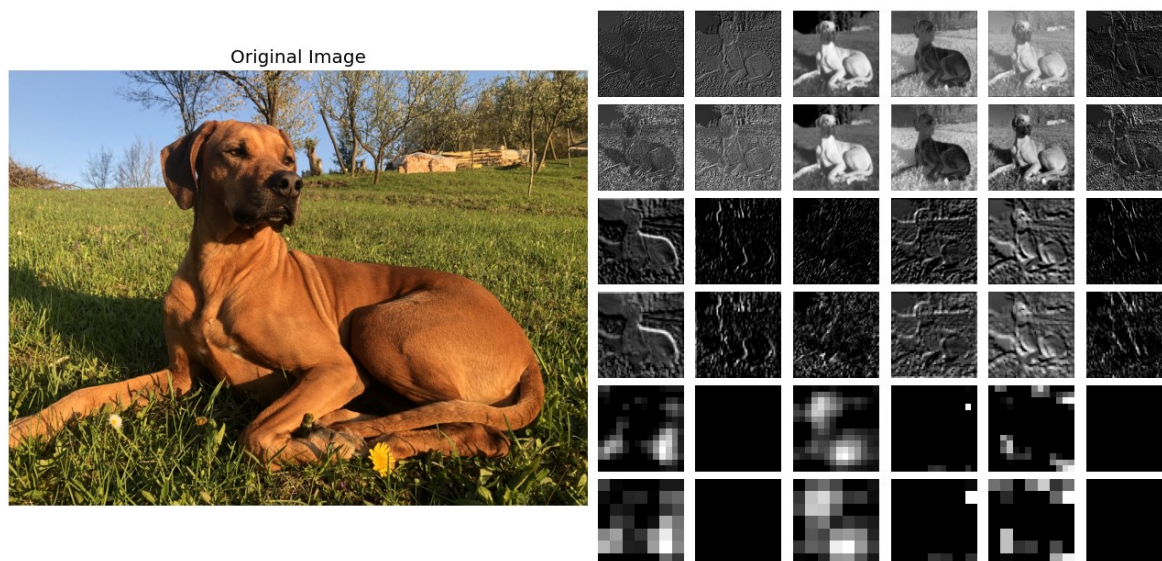
- I – Input
- P – Padding
- K – Kernel (Filter)
- S – Stride

1.2.2 Pooling layer

V tejto vrstve nastáva redukcia dimenzionality tým, že podobne ako v konvolučnej vrstve je aplikovaný na vstupné dáta (feature maps) filter, ktorý však nemá žiadne váhy, preto táto vrstva nepodlieha tréningu. Filter v tejto vrstve vykoná nad dátami agregáciu funkciu. Existujú dva hlavné typy funkcií: *Max pooling* – vyberie najväčšiu hodnotu z mapy pokrytej filtrom (Obr. 9) a *Average pooling* spriemeruje hodnoty pokryté filtrom. Vďaka tejto vrstve sa výrazne zníži výpočtová a pamäťová náročnosť v ďalších vrstvách, tým že sa znížia rozmery feature maps (ich počet však zostáva nezmenený). Rozmery sú počítané nasledovne: $O = \frac{I-K}{S} + 1$. [17]



Obr. 9 – Použitie funkcie Max pool[17]



Obr. 10 – Feature maps pred aplikáciou a po aplikácii max pooling

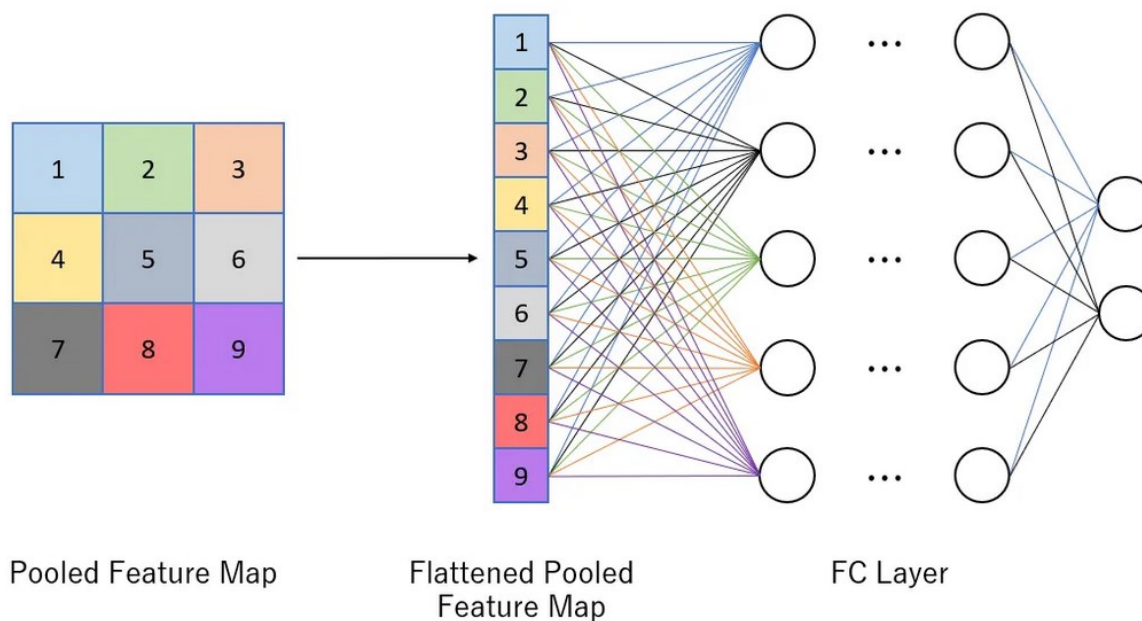
Obrázok 10 ilustruje prvých 6 výstupov niektorých konvolučných vrstiev (tieto vrstvy nenasledujú priamo za sebou) a následnú aplikáciu funkcie max pooling. Posledný riadok znázorňuje feature maps pred vstupom do flatten layer.

1.2.3 Flatten layer

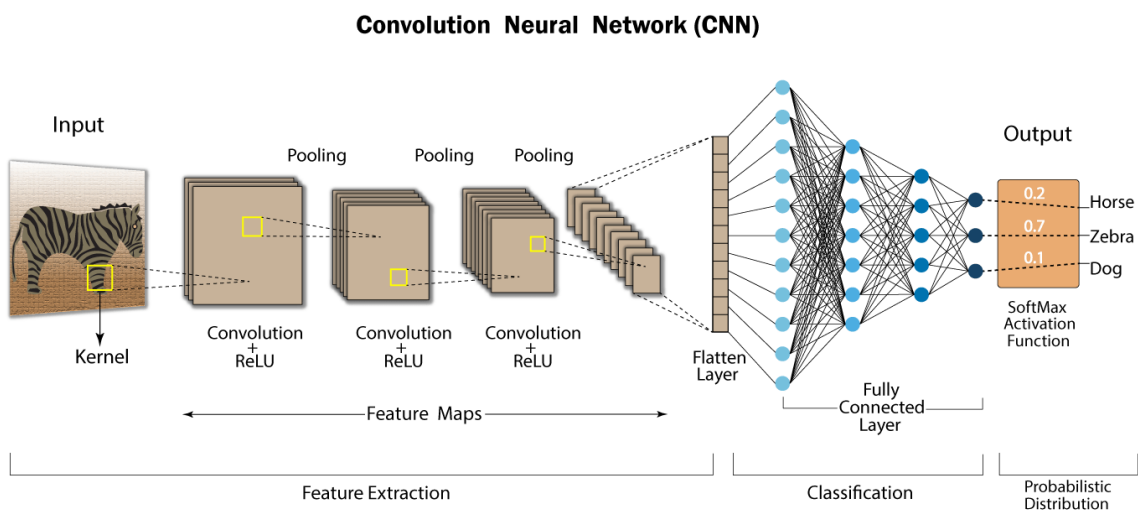
Výstupné feature maps z poslednej konvolučnej alebo pooling vrstvy sa vo flatten vrstve transformujú jednorozmerný vektor (pole čísel) (Obr. 11). Tento vektor slúži ako vstup pre poslednú vrstvu ktorá následne vykoná spracovanie a klasifikáciu vstupných údajov.[12]

1.2.4 Fully-connected layer

Posledná vrstva konvolučnej neurónovej siete niekedy označovaná ako Dense layer. Ide o neurónovú sieť, ktorá obsahuje viacero vrstiev skladajúcich sa z neurónov. Ako názov napovedá každý neurón z prechádzajúcej vrstvy je spojený s neurónom v nasledujúcej vrstve pomocou učiteľnej váhy (Obr. 11, 12). V poslednej vrstve sa vykonáva finálna klasifikácia na základe vlastností extrahovaných z prechádzajúcich FC vrstiev s využitím aktivačnej funkcie softmax na vhodnú klasifikáciu vstupov a vytvorenie pravdepodobnosti od 0 do 1 pre výsledné triedy (Obr. 12). Táto vrstva je trénovaná spolu s konvolučnou vrstvou, kedy sa nastavuje hodnota váh a biasov neurónov pomocou už spomínaných techník gradient descent a backpropagation.[18]



Obr. 11 – Dáta z Flatten layer vstupujúce do Fully-connected layer[13]



Obr. 12 – Konvolučná neurónová sieť[18]

1.3 Modely počítačového videnia

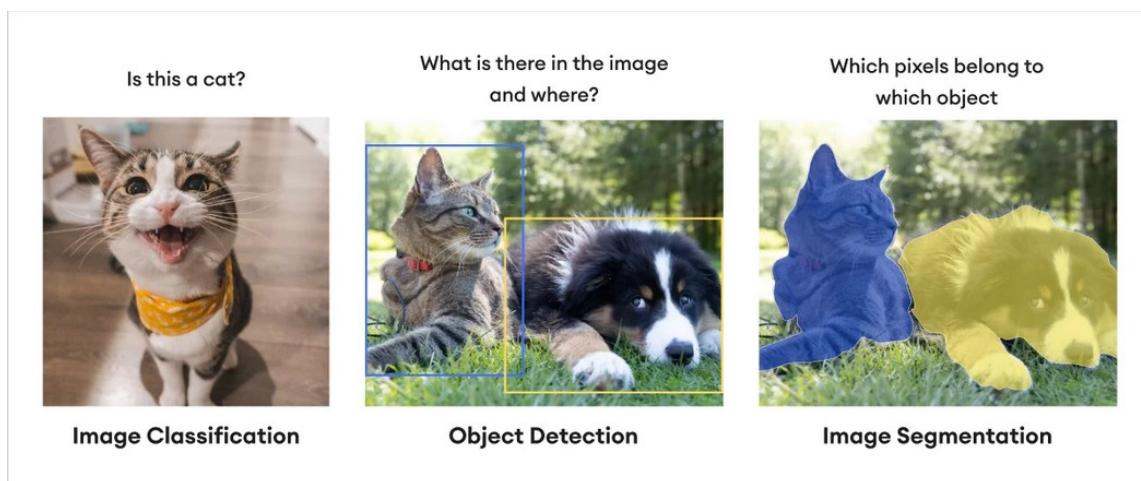
Modely používajú konvolučnú neurónovú sieť, ktorá bola trébovaná na vykonávanie konkrétnej úlohy. Táto sieť sa správne naučila vzory z trébových údajov a dokáže sa zovšeobecniť na nové vstupné dáta, vďaka čomu dokáže správne vyhodnotiť nové, predtým nevidené údaje.

1.4 Úlohy modelov počítačového videnia

Existuje viacero typov modelov počítačového videnia, pričom každý vyžaduje odlišný typ tréningu a tréningových dát. Výber vhodného CNN modelu pre počítačové videnie závisí od konkrétnej úlohy.

Medzi hlavné úlohy modelov počítačového videnia patria:

- Technika klasifikácie obrazu – skúma obraz a priradzuje mu štítok (label) podľa jeho obsahu (Obr. 13). Klasifikačný model kategorizuje obrázky do skupín.
- Detekcia objektov – modely detekcie objektov prehľadávajú obrázky alebo videá a identifikujú konkrétne objekty, ktoré následne označia (Obr. 13). Táto technológia sa používa napríklad vo výrobe na kontrolu kvality, kde je takýmto spôsobom možné v reálnom čase detekovať chyby.
- Segmentácia obrazu – na rozdiel od klasifikácie rozdeľuje segmentácia obraz na jednotlivé časti, čím oddeľuje objekty od ich pozadia (Obr. 13). To je obzvlášť cenné v medicínskom zobrazovaní, napríklad pri izolovaní nádorov na röntgenových snímkach alebo MRI pre presnejšiu diagnostiku.



Obr. 13 – Klasifikácia, detekcia a segmentácia objektov[20]

- Sledovanie objektov – po detekcii objektu dokážu modely sledovať pohyb objektu. To je nevyhnutné pre autonómne vozidlá, kde systém musí sledovať pohyb chodcov, vozidiel a prekážok v dynamickom prostredí. Sledovanie objektov sa tiež často využíva v športovej analytike a kamerových bezpečnostných systémoch.

- Optické rozpoznávanie znakov (angl. *Optical character recognition*, skrátene OCR) – prevádza text z obrázkov do strojovo čitateľného formátu, čo umožňuje automatizované spracovanie dokumentov. Využíva sa v bankovníctve na digitalizáciu šekov alebo v lekárstve na spracovanie dokumentov a záznamov pacientov.
- Odhad polohy tela (angl. *Pose estimation*) – detekcia a sledovanie pohybov častí ľudského tela na obrázkoch alebo videách (Obr. 14).[19, 20]



Obr. 14 – Detekcia ľudského tela[21]

Tab. 2 – Príklad modelov pre vykonávanie konkrétnej úlohy

Úloha	Klasifikácia obrazu	Detekcia objektov	Segmentácia obrazu	Sledovanie objektov	OCR	Pose estimation
Model	ResNet, VGG, MobileNet	YOLO, Faster R-CNN	U-Net, Mask R-CNN	SORT, DeepSORT	Tesseract, EasyOCR	OpenPose, MediaPipe

1.5 Tréning CV modelov

Tréning modelov počítačového videnia je veľmi výpočtovo náročný a komplexný proces. Tento proces vyžaduje starostlivé plánovanie, kvalitné údaje a neustále ladenie na dosiahnutie optimálnych výsledkov. Dôležitým aspektom je zabezpečiť rozmanitosť tréningových dát, aby modely fungovali spoľahlivo v rôznych podmienkach. To pomáha predchádzať zaujatosti a zlepšuje celkovú robustnosť modelu.

Zber a príprava údajov

Získavanie rozsiahlych a rôznorodých obrázkov alebo videí, ktoré sú relevantné pre konkrétnu úlohu. Tieto údaje musia byť starostlivo vybrané, aby sa zabezpečila reprezentatívnosť rôznych scenárov a podmienok, ako napríklad rôzne uhly záberov, rôzne svetelné podmienky a iné variácie. V prípade, tréovania modelu na rozpoznávanie psov by bolo potrebné „nakrmiť“ model tisíckami až miliónmi obrázkov psov, kde by bolo potrebné brať v úvahu základné charakteristiky, ako plemeno psa, jeho veľkosť, farbu, atď. Vo všeobecnosti platí, čím viac kvalitných dát je modelu poskytnutých, tým presnejší model bude.[19, 22]

Anotácia údajov

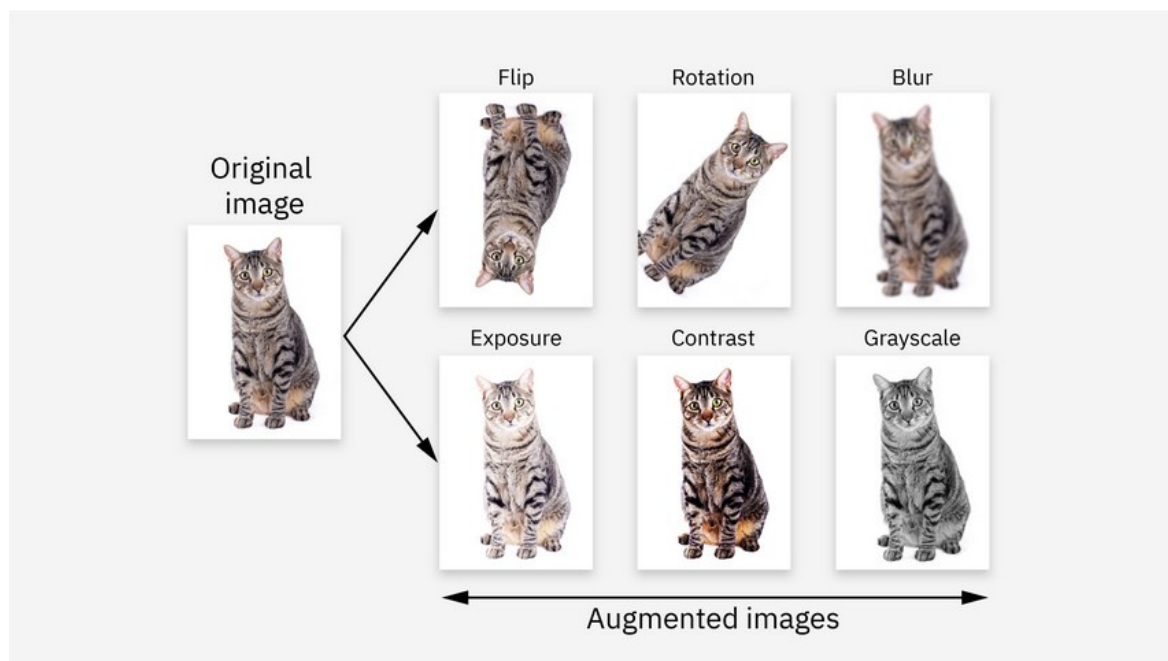
Označovanie obrázkov alebo videí, aby model vedel, čo sa na nich nachádza. Tento proces môže zahŕňať označovanie objektov, hraníc, kľúčových bodov alebo iných relevantných informácií. Správne označenie je kľúčové pre efektívne učenie s učiteľom (angl. *supervised learning*).[23]

Data preprocessing

Ide o úpravu obrázkov a videí tak, aby boli vhodné na spracovanie modelmi strojového učenia. Tento krok zvyšuje presnosť, rýchlosť a robustnosť modelov, pretože eliminuje šum, normalizuje dáta a zabezpečuje konzistentnosť. Väčšina modelov je tréovaná na fixnej veľkosti obrázkov.[22]

Dátová augmentácia

Dátová augmentácia je kľúčovou technikou pri tréovaní modelov počítačového videnia, pretože pomáha zlepšiť generalizáciu a zvyšuje rozmanitosť tréovaných údajov bez potreby ďalších vzoriek z reálneho sveta. Ide o proces úpravy tréovaných údajov prostredníctvom náhodných transformácií, ako je prevrátenie, preklad, orezanie, otočenie, náhodné vymazanie (Obr. 15) alebo úprava RGB kanálov obrázku. Vytvorenie šumu náhodným pridaním čiernych, bielych a farebných pixelov. Kopírovanie označených častí obrázku a ich vloženie do iného obrázku, čo pomáha pri detekovaní v rôznych pozadiach alebo geografických oblastiach.[24]



Obr. 15 – Dátová augmentácia[24]

Vyhodnotenie a ladenie modelov počítačového videnia

Po tréningu modelu je dôležité vyhodnotiť jeho výkon a optimalizovať ho tak, aby poskytoval čo najlepšie výsledky. Tento proces zahŕňa meranie presnosti, identifikáciu problémov a úpravu hyperparametrov. Overfitting je jav, kedy model vykazuje vysokú presnosť na tréningových dátach, ale zlyháva pri generalizácii na neznámych alebo nevidených dátach. Inými slovami, model sa „naučil naspamäť“ špecifické detaily tréningových dát namiesto toho, aby sa naučil všeobecné vzory. Najlepším riešením na zníženie overfittingu je získať viac tréningových dát a prípadná úprava architektúry siete. Opakom overfittingu je underfitting, ku ktorému dochádza vtedy, keď je ešte priestor na zlepšenie z tréningových údajov. Môže k nemu dôjsť z viacerých dôvodov: ak model nie je dostatočne výkonný, je príliš regularizovaný alebo jednoducho nebol trénovaný dostatočne dlho. To znamená, že sieť sa nenaučila relevantné vzory z tréningových dát.[25]

1.6 Transfer learning

Technika strojového učenia, pri ktorej sa vopred natrénovaný model ako napríklad VGG opätovne použije na novú, ale súvisiacu úlohu. Namiesto toho, aby bol model trénovaný od začiatku, znalosti sú „prenesené“ z existujúceho modelu a následne sa doladia pre konkrétnu potrebu. Táto technika umožňuje rýchlejšie a efektívnejšie trénovanie, a to aj s obmedzeným množstvom dát.[26]

1.7 Použitie počítačového videnia

Počítačové videnie umožňuje spoločnostiam v rôznych odvetviach znižovať cenu prevádzkových nákladov a automatizovať výrobu. V oblasti organizácie obsahu dokáže počítačové videnie identifikovať osoby alebo objekty na fotografiách a podľa toho ich kategorizovať, čo sa bežne používa v aplikáciách na ukladanie fotografií a na sociálnych médiách. Optické rozpoznávanie znakov zlepšuje extrakciu textu, vďaka čomu je možné lepšie vyhľadávať veľké množstvo informácií a automatizovať spracovanie dokumentov. Mestá využívajú počítačové videnie na riadenie dopravy, rozpoznávanie evidenčných čísiel áut a optimalizáciu svetelných križovatiek s cieľom znížiť dopravné zápchy. V poľnohospodárstve sa satelitné a dronové snímky analyzujú na monitorovanie plodín, zisťovanie burín a hodnotenie nedostatku živín, čo pomáha optimalizovať výnosy. Automobilový priemysel využíva počítačové videnie v autonómnych vozidlách, kde detekcia a sledovanie objektov v reálnom čase pomáhajú autonómnym vozidlám bezpečne navigovať. V zdravotníctve analýza obrazu pomáha lekárom rýchlejšie a presnejšie diagnostikovať choroby a monitorovať pacientov. Vo výrobe zabezpečuje údržbu strojov, kontrolu kvality výrobkov a efektívne procesy balenia. Technológia rozpoznávania tváří je ďalšou kľúčovou aplikáciou, ktorá sa nachádza uplatnenie pri overovaní totožnosti a bezpečnostných účeloch. Počítačové videnie so svojimi širokými možnosťami využitia v rôznych odvetviach prináša revolúciu vo fungovaní podnikov čím zlepšuje fungovanie, efektivitu a bezpečnosť.[27, 28, 29]

2 DETEKCIA GEST

Rozpoznávanie gest umožňuje počítačovým systémom interpretovať a reagovať na pohyby častí ľudského tela, najčastejšie rúk, tváre alebo hlavy. Táto forma interakcie medzi človekom a počítačom otvára možnosti ovládania zariadení bez potreby fyzického kontaktu, čo je potrebné najmä v situáciách, kde je dôležitá bezdotyková komunikácia, napríklad z hľadiska hygieny, bezpečnosti alebo komfortu.

Gestá môžu byť zachytávané pomocou rôznych zariadení – od bežných kamier cez hĺbkové senzory až po špecializované nositeľné zariadenia, ako sú inteligentné rukavice. Získané údaje sa následne spracúvajú pomocou algoritmov počítačového videnia a strojového učenia s cieľom identifikovať konkrétne vzory a priradiť im zodpovedajúci význam.[30]

Statické a dynamické gestá

Statické gestá zahŕňajú fixné polohy rúk alebo tela, ktoré vyjadrujú špecifické význam bez pohybu v čase. Rozpoznávanie týchto gest sa opiera o analýzu priestorových informácií z jednej snímky obrazu. Bežne sa používa pri tlmočení posunkovej reči a umožňuje prekladať špecifické tvary rúk na zodpovedajúce písmená alebo slová.

Dynamické gestá zahŕňajú pohyby v čase, napríklad mávanie alebo ukazovanie. Rozpoznávanie týchto gest si vyžaduje analýzu časových sekvencií obrazov na zachytenie informácií o pohybe. Využívajú sa vo virtuálnej realite a v hrách, kde interpretácia pohybových sekvencií zlepšuje interakciu.[31]

Výzvy detekcie gest

Detekcia gest čelí viacerým výzvam, medzi ktoré patria variabilita a dynamickosť prostredia, kde zmeny v osvetlení, šum na pozadí, rôzne uhly pohľadu, tieň a odlesky môžu ovplyvniť presnosť rozpoznávania. Rôznorodosť používateľov, ako sú odlišné veľkosti rúk a individuálne spôsoby vykonávania gest, čo taktiež môže viesť k nepresnostiam. Keďže mnohé aplikácie vyžadujú rozpoznávanie gest v reálnom čase, je potrebné efektívne spracovanie a vyhodnotenie údajov. Navyše, ak sú niektoré gestá podobné, môže dôjsť k ich prekryvaniu, čo sťažuje ich rozpoznanie.

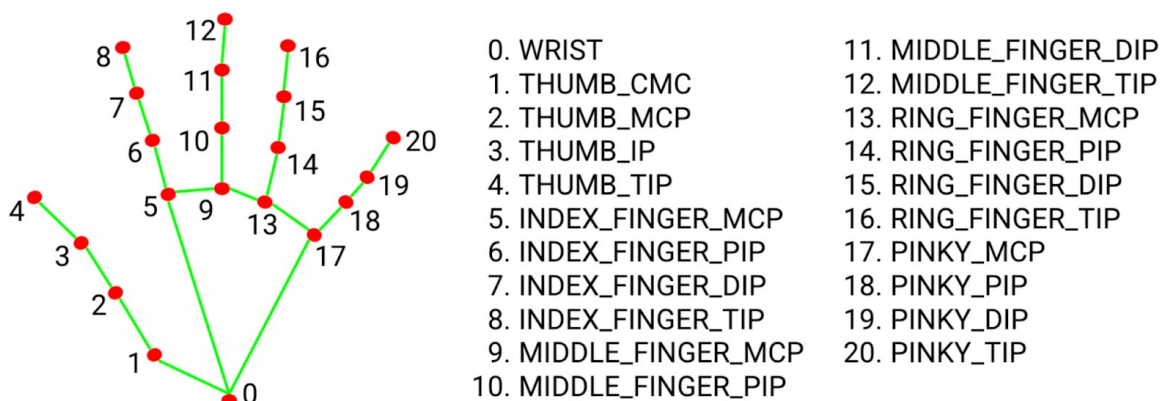
2.1 MediaPipe Hands

MediaPipe je open-source framework vyvinutý spoločnosťou Google, ktorý obsahuje súbor optimalizovaných knižníc a modelov. Jedným z týchto modelov je *MediaPipe Hands*, ktorý umožňuje presné detekovanie rúk z obrázkov, videí alebo real-time zdrojov.

Model *MediaPipe Hands* využíva machine learning pipeline pozostávajúcu z viacerých po sebe nadväzujúcich modelov, ktoré spolupracujú na detekcii a sledovaní rúk v reálnom čase. Pipeline je optimalizovaná pre rýchlosť a presnosť aj na zariadeniach s nižším výkonom (napr. mobilné zariadenia alebo embedded systémy).

Prvým krokom je aplikácia *Palm Detection Modelu* na detekciu dlane v celom vstupnom obraze. Výstupom tohto modelu je bounding box, ktorý definuje polohu a orientáciu dlane v obraze.

Na základe detekovaného bounding boxu sa vykoná výrez príslušnej oblasti z pôvodného obrazu, ktorý je následne poskytnutý ďalšiemu modelu – *Hand Landmark Model*. Tento model lokalizuje 21 trojrozmerných bodov (3D landmarks) na ruke vrátane kĺbov a koncov prstov (Obr. 16). Okrem súradníc bodov x a y model odhaduje aj hĺbkovú hodnotu každého bodu s označením z a určuje či ide o pravú alebo ľavú ruku.



Obr. 16 – Lokalizácia 21 bodov ruky[32]

Po inicializácii detekcie je pipeline schopná pokračovať v sledovaní rúk bez potreby opakovanej detekcie v každom snímku. Sledovanie znižuje výpočtové nároky a zvyšuje plynulosť. Ak sa sledovanie preruší napr. ruka opustí obraz, model sa vráti k detekcii a celý proces sa opakuje.

Táto architektúra umožňuje *MediaPipe Hands* dosahovať robustný a efektívny odhad polohy ruky s vysokou presnosťou a nízkou latenciou aj v náročnejších podmienkach, čo ho

robí ideálnym pre aplikácie, ako je rozpoznávanie gest v reálnom čase. V praktickej časti je model použitý na získanie bodov, ktoré slúžila ako tréningové a vstupné dáta pre model rozpoznania gest. Robot bude ovládaný pomocou statických gest oboch rúk.[32, 33]

2.2 Model rozpoznania gest s použitím TensorFlow

Na trénovanie modelu klasifikácie gest je použitá open-source knižnica *TensorFlow*, ktorá umožňuje efektívne vytvárať, trénovať a používať modely strojového učenia.

TensorFlow obsahuje deep learning API Keras, ktorá poskytuje jednoduchý spôsob definovania architektúry neurónových sietí. Keras výrazne zjednodušuje prácu s vrstvami modelu, výberom optimalizačných algoritmov, stratových funkcií a metód trénovania.[34]

3 OVLÁDANIE JEDNOTKY LEGO EV3

Ide o tretiu generáciu programovateľnej robotickej stavebnice zo série LEGO Mindstorms (Obr. 17). EV3 pozostáva z hlavnej výpočtovej jednotky tzv. kocky a ponúka rozšírené možnosti interakcie so svojím okolím vďaka širokej škále senzorov – vrátane senzorov dotyku, farby, ultrazvuku či infračerveného žiarenia. Tieto komponenty umožňujú vytváranie dynamických a interaktívnych robotických systémov, ktoré môžu reagovať na vonkajšie podnety. V rámci tejto práce je EV3 využitý ako zariadenie, ktoré vykonáva príkazy na základe detekovaných gest.[35]



Obr. 17 – Stavebnica LEGO Mindstorms EV3[35]

EV3 kocka predstavuje centrálnu programovateľnú jednotku, ktorá umožňuje pripojenie motorov a senzorov. Je napájaná 6 AA batériami alebo dobíjacím lítium-iónovým akumulátorom. Motory sa pripájajú do výstupných portov s označením *A* až *D*, zatiaľ čo senzory sa zapájajú do vstupných portov označených číslami *1* až *4* (Obr. 18). Kocka taktiež obsahuje USB a mini USB port. O fungovanie systému sa stará procesor ARM9 s frekvenciou 300 MHz. Interná pamäť zahŕňa 16 MB flash pamäte a 64 MB operačnej pamäte RAM. Úložná kapacita je rozšíriteľná pomocou microSD karty s kapacitou do 32 GB. Jednotka podporuje Bluetooth a Wi-Fi, avšak pri použití Wi-Fi je potrebné použiť dodatočný USB Wi-Fi adaptér tzv. dongle.[36]

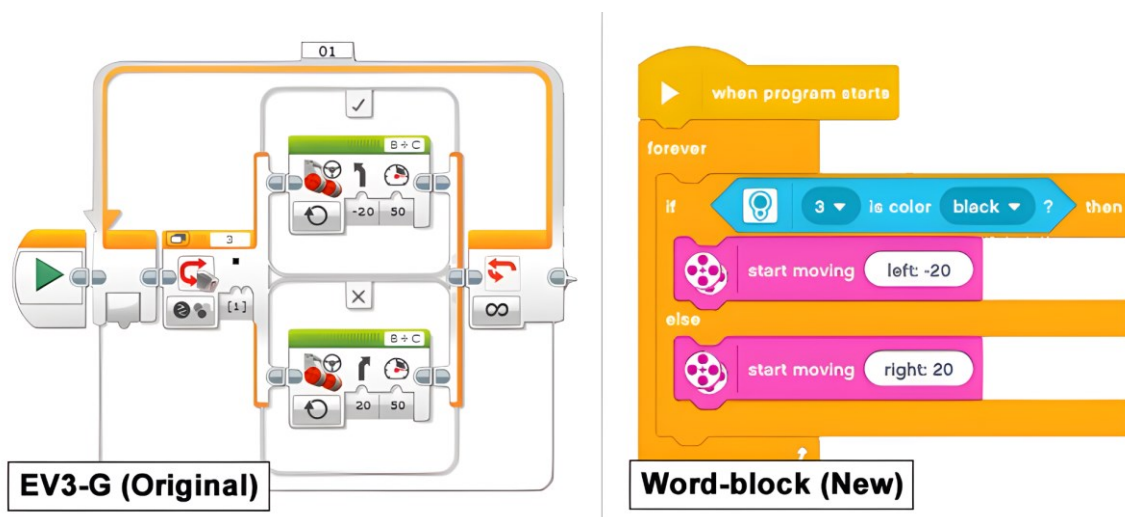


Obr. 18 – Porty kocky EV3[36]

3.1 Programovanie robota

Najjednoduchším spôsobom, ako začať s programovaním robota LEGO EV3 je využitie vývojového prostredia *EV3 Classroom*. Ide o oficiálne programovacie prostredie vyvinuté spoločnosťou LEGO pre robotickú stavebnicu LEGO Mindstorms EV3, ktoré nahrádza starší software *LEGO EV3-G*.

EV3 Classroom využíva rozhranie založené na jazyku Scratch. To umožňuje blokové programovanie prostredníctvom vizuálneho systému typu drag-and-drop (Obr. 19). Toto riešenie je vhodné najmä pre začiatočníkov a študentov, pretože umožňuje intuitívne vytvárať programy bez potreby znalosti klasických programovacích jazykov.[37]



Obr. 19 – EV3 Classroom[37]

Využitie programovacích jazykov

Pre možnosť ovládania robota pomocou tradičných programovacích jazykov je nutné použiť knižnice, tie však LEGO firmware nepodporuje. Tieto knižnice poskytujú väčšiu flexibilitu a ďalšie rozširujúce funkcie programovania v porovnaní s oficiálnym programovacím softwarom LEGO.

3.2 Operačný systém ev3dev

Umožňuje spustiť na EV3 plnohodnotný operačný systém Linux, ktorý poskytuje pokročilé možnosti programovania nad rámec predvoleného (default) firmwaru LEGO. Vďaka tomu bude možné s robotom komunikovať použitím sieťových protokolov. Súčasťou operačného systému je tzv. low-level driver framework, ktorý poskytuje štruktúrovaný spôsob ovládania senzorov, motorov a tlačidiel pomocou driverov. Tento driver framework následne používajú knižnice, aktuálne sú knižnice dostupné v jazykoch: Python, MicroPython, Java, Go, C, C++ a Rust.[38]

II PRAKTICKÁ ČASŤ

4 PRÍPRAVA ROBOTA

Lego robot bude mať jednoduchý dizajn, ktorý zaručí bezproblémový prístup k vstupným a výstupným portom a konektoru na nabíjanie batérie. Ak bude robot využívať batérie typu AA je potrebné zabezpečiť jednoduché rozobratie a prístup k spodnej časti kocky v prípade nutnosti výmeny batérií.

Hardware robota

Na pohyb bude robot používať dva veľké servomotory. Tieto motory budú pripojené pomocou vodičov RJ12 ku kocke na výstupných portoch (Obr. 20) napr. *B* a *C*.

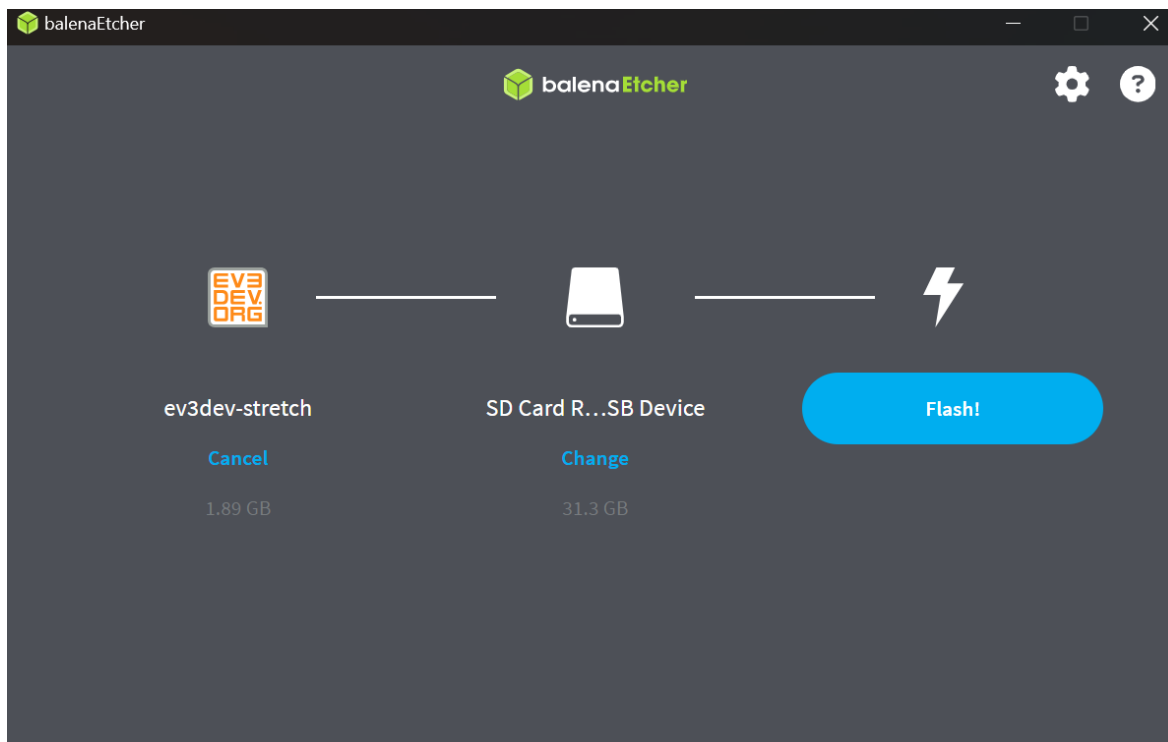


Obr. 20 – LEGO robot s možnosťou pohybu

Software robota

Robot bude používať operačný systém *ev3dev*. K nahratiu OS je potrebná čítačka microSD kariet spolu s microSD kartou, ktorej kapacita by mala byť minimálne 4GB a maximálne 32GB tohto typu karty sa hovorí microSDHC. Po stiahnutí image súboru operačného systému dostupného na stránkach *ev3dev* je tento súbor nahratý na microSD kartu pomocou open-source softwaru *balenaEtcher* (Obr. 21). Po úspešnom nahratí OS na kartu je karta

vložená do microSD slotu kocky. Systém *ev3dev* beží priamo z microSD karty a nijakým spôsobom nemení pôvodný firmware EV3. Na návrat do pôvodného prostredia stačí systém vypnúť a vybrať microSD kartu.[38]



Obr. 21 – Nahratie ev3dev na microSD kartu pomocou *balenaEtcher*

5 ARCHITEKTÚRA A TRÉNING MODELU NA ROZPOZNANIE GEST

Nasledujúca kapitola sa popisuje zber a prípravu dát a techniky použité pri tréningu modelu rozpoznania gest.

5.1 Zber a príprava dát na tréovanie

Model bude tréovaný na rozpoznanie troch základných gest – ☞ (palec hore), ☞ (dlaň) a ☞ (päť). Výsledné gesto bude kombináciou gest oboch rúk, napríklad gesto ☞ ☞ je interpretované ako pokyn pre pohyb vpred. Program *extract_landmarks.py* slúži na extrakciu 21 referenčných bodov ruky. Na získavanie a spracovanie obrazu je použitá knižnica *OpenCV* (Open Source Computer Vision Library), následne *MediaPipe Hands* lokalizuje body v získanom obraze. Počas behu programu stlačením klávesy 0 – 9 pri vykonaní požadovaného gesta sú v danom momente extrahované body ruky a priradí sa im trieda (label) zodpovedajúca stlačenej klávese. Pri pridávaní nových gest je nutné sa uistiť, že daná trieda nezodpovedá inému gestu. Každé gesto je potrebné zaznamenať pre obe ruky, v rôznych vzdialenostiach a uhloch opätovným stláčaním rovnakej klávesy 0 – 9, aby sa zabezpečila robustnosť tréningových dát. Extrahované body sú následne normalizované a uložené do CSV súboru, ktorý slúži ako dataset. Každý riadok datasetu predstavuje jedno gesto. Normalizácia prebieha nasledovne: medzi všetkými bodmi sa nájdu tie dva, ktorých x-ové súradnice sú od seba najviac vzdialené. Tieto dve hodnoty – x_{min} a x_{max} sa použijú ako základ pre normalizáciu. Súradnica x_n každého bodu sa upraví podľa vzorca na normalizáciu – $x_{normalized} = \frac{x_n - x_{min}}{x_{max} - x_{min}}$. Tento postup sa potom rovnakým spôsobom zopakuje aj pre y-ové súradnice.

5.2 Architektúra modelu

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	2,752
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 3)	99

Total params: 4,931 (19.26 KB)

Trainable params: 4,931 (19.26 KB)

Non-trainable params: 0 (0.00 B)

Obr. 22 – Architektúra modelu

Výsledný model (neurónová sieť) (Obr. 22) používa Sekvenčný model, ktorý je súčasťou TensorFlow Keras API. Sekvenčný model je najjednoduchším typom modelu, ktorý Keras API ponúka. Vrstvy tohto modelu za sebou lineárne nadväzujú (jedna vrstva za druhou). Používanie sekvenčného modelu v Keras API má však určité obmedzenia. Toto rozhranie neumožňuje vytvárať zložitejšie architektúry, ako sú siete s viacerými vstupmi alebo výstupmi, či také, v ktorých vrstvy nielen nasledujú za sebou, ale si zároveň medzi sebou zdieľajú informácie o extrahovaných vlastnostiach. Pre úlohu klasifikácie gest pomocou bodov detekovaných rúk je sekvenčný model vhodnou voľbou.[39]

Táto neurónová sieť funguje veľmi podobne ako Fully-connected (Dense) layer v konvolučnej neurónovej sieti – neuróny v nasledujúcej vrstve sú pomocou učiteľných váh spojené s každým neurónom v predchádzajúcej vrstve. Prvou vrstvou modelu je vstupná vrstva, ktorá prijíma vstupný vektor s hodnotami získaných bodov ruky, ide o 42 hodnôt (21 bodov každý obsahujúci súradnice x a y). Po tejto vrstve nasledujú tri za sebou idúce vrstvy typu Dense. Prvá vrstva má 64 neurónov a druhá 32 neurónov obe vrstvy používajú aktivačnú funkciu ReLU. Tretia Dense vrstva slúži ako klasifikačná a počet neurónov je rovný počtu detekovateľných gest. Táto vrstva používa aktivačnú funkciu softmax pre

vytvorenie finálnej pravdepodobnosti. Celá sieť má 4931 trénovateľných parametrov (váh a biasov).

Dropout vrstvy sú aktívne len počas tréningu. Ide o techniku, pri ktorej sa náhodne ignorujú niektoré neuróny, aby sa znížila ich závislosť na sebe. Pri ďalšej predikcii sa náhodne vyberie iná kombinácia neurónov, ktoré budú ignorované. Tým sa model učí vytvárať viacero nezávislých reprezentácií a nebude sa spoliehať len na konkrétne neuróny. Výsledkom je, že model sa stáva odolnejší voči overfittingu a lepšie sa generalizuje na nové dáta. Obe dropout vrstvy majú nastavený dropout rate na 0,3 to znamená, že 30% neurónov bude dočasne ignorovaných.

5.3 Tréning modelu

Model sa učí z CSV súradníc získaných gest. Pred začatím tréningu je tento súbor dát náhodne premiešaný a potom rozdelený na tréningové a validačné dáta v pomere 80:20. Pri tréningu model nájde vzťahy medzi vzdialenosťami jednotlivých bodov gesta a priradenou triedou. Keďže dáta sú označené ide o učenie s učiteľom. Vo fáze trénovania modelu je potrebné opakovane vypočítať funkciu straty (angl. *loss function*), ktorá meria, ako veľmi sa predpovede modelu líšia od očakávaných označení – inými slovami, ako zle model funguje. Na výpočet veľkosti straty model používa funkciu *SparseCategoricalCrossentropy*, ktorá je vhodná pre klasifikačné úlohy s dvomi a viac triedami. Táto funkcia očakáva, že triedy budú reprezentované ako celé čísla.[40]

Pri tréningu modelu sa používajú dva hyperparametre – *epoch* a *batch size*. Jedna epocha predstavuje jeden kompletný prechod celým tréningovým súborom (datasetom) počas tréningu. Po každej epoche je model validovaný na 20% validačných dátach, ktoré neboli použité pri samotnom učení. Cieľom validácie je overiť ako správne dokáže model predikovať aj na nových, doposiaľ nevidených vstupoch. *Batch size* určuje, koľko tréningových príkladov alebo vzoriek (predstavuje 1 riadok datasetu) bude klasifikovaných predtým, než sa vykoná výpočet veľkosti priemernej hodnoty straty v batchi (niekedy označovaná angl. *cost*) a aktualizácia parametrov modelu pomocou optimalizačného algoritmu.[41]

Ak napríklad tréningový súbor obsahuje 1000 vzoriek, pričom *batch size* je 100 a počet *epoch* je 10, model v jednej epoche vykoná 10 aktualizácií váh, čiže za celý tréning tak spolu vykoná 100 aktualizácií parametrov.

```

Epoch 1/128
24/24 — 2s 23ms/step - accuracy: 0.3605 - loss: 1.2316 - val_accuracy: 0.6335 - val_loss: 0.9312
Epoch 2/128
24/24 — 0s 4ms/step - accuracy: 0.5426 - loss: 0.9685 - val_accuracy: 0.7958 - val_loss: 0.8268
Epoch 3/128
24/24 — 0s 5ms/step - accuracy: 0.6211 - loss: 0.8927 - val_accuracy: 0.8429 - val_loss: 0.7474
Epoch 4/128
24/24 — 0s 5ms/step - accuracy: 0.7098 - loss: 0.7831 - val_accuracy: 0.8482 - val_loss: 0.6835
Epoch 5/128
24/24 — 0s 5ms/step - accuracy: 0.7300 - loss: 0.7518 - val_accuracy: 0.8639 - val_loss: 0.6198
Epoch 6/128
24/24 — 0s 5ms/step - accuracy: 0.7354 - loss: 0.6988 - val_accuracy: 0.8639 - val_loss: 0.5623
Epoch 7/128
24/24 — 0s 5ms/step - accuracy: 0.7610 - loss: 0.6455 - val_accuracy: 0.8691 - val_loss: 0.5112
Epoch 8/128
24/24 — 0s 4ms/step - accuracy: 0.7809 - loss: 0.6150 - val_accuracy: 0.8901 - val_loss: 0.4663
Epoch 9/128
24/24 — 0s 5ms/step - accuracy: 0.8425 - loss: 0.5366 - val_accuracy: 0.9005 - val_loss: 0.4197
Epoch 10/128
24/24 — 0s 5ms/step - accuracy: 0.8436 - loss: 0.5228 - val_accuracy: 0.9005 - val_loss: 0.3813
Epoch 11/128
24/24 — 0s 5ms/step - accuracy: 0.8425 - loss: 0.4617 - val_accuracy: 0.9005 - val_loss: 0.3457
Epoch 12/128
24/24 — 0s 5ms/step - accuracy: 0.8464 - loss: 0.4551 - val_accuracy: 0.9267 - val_loss: 0.3150
Epoch 13/128
...
Epoch 127/128
24/24 — 0s 5ms/step - accuracy: 0.9975 - loss: 0.0223 - val_accuracy: 1.0000 - val_loss: 0.0023
Epoch 128/128
24/24 — 0s 4ms/step - accuracy: 0.9974 - loss: 0.0178 - val_accuracy: 1.0000 - val_loss: 0.0023

```

Obr. 23 – Výpis počas tréningu modelu

Jednotlivé metriky pre epochu na obrázku znamenajú:

- *accuracy* – Presnosť na tréningových dátach; ukazuje, ako správne model klasifikoval všetky tréningové vzorky.
- *loss* – Priemerná chyba (strata) na tréningových dátach; nízka hodnota znamená, že predikcie modelu boli blízke skutočným hodnotám.
- *val_accuracy* – Presnosť na validačných dátach; hovorí ako správne predpovedal model všetky validačné vzorky.
- *val_loss* – Priemerná strata na validačných dátach; nízka hodnota znamená, že model si zachoval vysokú kvalitu správnych predikcií aj na dátach, ktoré počas tréningu nevidel.

Na obrázku (Obr. 23) finálna epocha 128 zobrazuje hodnoty *accuracy* – 99,74%, *loss* – 0,0178, *val_accuracy* – 100% a *val_loss* – 0,0023. Priemerná strata (*loss*) je veľmi nízka a presnosť (*accuracy*) vysoká znamená to, že model sa správne naučil vzory v dátach.

Keďže model správne predikoval všetky validačné dáta napovedá to tomu, že neprichádza k overfittingu. Aj keď je *val_accuracy* je 100% *val_loss* nikdy nebude rovná nule, pretože strata okrem chybovosti predikcií taktiež hovorí o tom ako si model v danej predikcii verí. Inými slovami ak model pri tréningu určil predikciu výslednej triedy s istotou

60% bude mať táto predikcia väčšiu hodnotu straty ako predikcia s 80% istotou. Nulová strata by znamenala, že model bol vždy stopercentne istý a zároveň vždy správny, čo je takmer nemožné. V prípade, že model predikuje náhodne a nesprávne „model nevie čo robí“ je hodnota straty najväčšia ako je možné vidieť pri predikciách v prvých epochách (Obr. 23), preto veľkosť straty určuje ako zle alebo dobre model funguje.

K natrénovaniu troch gest, ktoré vie model rozpoznať, bolo použité približne 750 tréningových a 200 validačných vzoriek. Avšak pri pridávaní nových gest a teda aj ďalších tréningových vzoriek môže nastať situácia, že model prestane správne predikovať nové gestá alebo celková presnosť začne klesať. V takom prípade je vhodné experimentovať s architektúrou a hyperparametrami siete, aby sa model dokázal lepšie prispôbiť zvýšenej zložitosti úlohy. Zmena architektúry môže znamenať napríklad pridanie väčšieho počtu vrstiev alebo zväčšenie počtu neurónov v jednotlivých vrstvách. Zmena hyperparametrov môže zahŕňať použitie iného optimalizačného algoritmu zvýšenie počtu epoch atď.

Pokiaľ ide o počet nových vzoriek alebo tried (gest), pri ktorých by sa mala architektúra zmeniť, neexistuje jednoznačné pravidlo. Všeobecne však platí, že ak počet gest významne narastie a model už nedokáže udržať požadovanú presnosť, je čas zvážiť zložitejšiu architektúru. Rovnako, ak objem dát narastie niekoľkonásobne, jednoduchý model môže byť nedostatočný na zachytenie nových vzorov.

Aktualizácia parametrov pomocou Stochastic Gradient Descent (SGD)

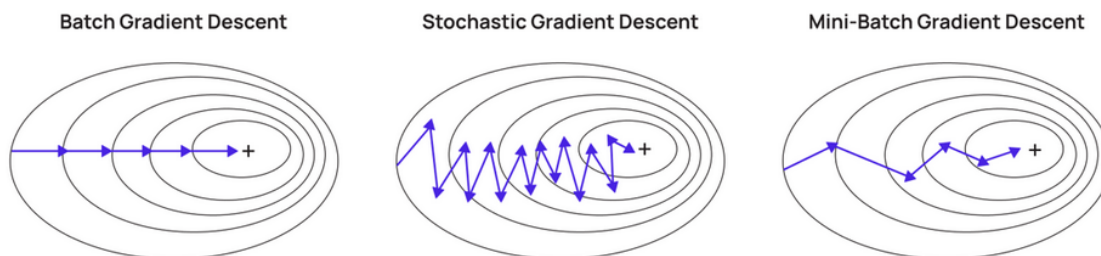
Po vyhodnotení straty sa pomocou algoritmu backpropagation vypočítajú priemerné gradienty všetkých vzoriek v aktuálnej batchi, ktoré určujú smer a mieru úprav váh a biasov neurónovej siete. Veľkosť týchto gradientov závisí od hodnoty straty – čím je strata väčšia, tým výraznejšie budú aj úpravy parametrov modelu. Aby sa chyba modelu minimalizovala použije vypočítané gradienty optimalizačný algoritmus Stochastic Gradient Descent (SGD) na aktualizáciu parametrov siete. Tento algoritmus patrí medzi varianty algoritmu gradient descent. Cieľom optimalizačného procesu je priblížiť sa k tzv. globálnemu minimu – teda stavu, pri ktorom je hodnota straty čo najnižšia, vďaka čomu model dosahuje najlepšiu možnú presnosť. Tento proces, pri ktorom sa model postupným tréňovaním zlepšuje a približuje k optimálnemu riešeniu sa nazýva konvergencia (Obr. 24).[42]

Klasický (batch) gradient descent aktualizuje parametre až po spracovaní celého tréningového súboru (jednej epochy). Na vykonanie jednej aktualizácie je potrebné vypočítať gradienty pre celý dataset a brať v úvahu každý tréningový príklad. Algoritmus je

síce presný ale veľmi pomalý (Obr. 24) pri hľadaní globálneho minima. Okrem toho môže byť nepraktický pre väčšie súbory údajov, ktoré sa nezmestia do operačnej pamäte.[42, 43]

Na druhej strane SGD vykonáva aktualizáciu parametrov po každom tréningovom príklade x a jeho príslušnej triede y . SGD býva zvyčajne oveľa rýchlejší ako batch gradient descent. Aktualizácia parametrov na základe iba jedného príkladu však môže viesť k vysokej variabilite (kolísaniu), čo môže spôsobiť, že strata bude počas tréningovania výrazne kolísať (Obr. 24).[42, 43]

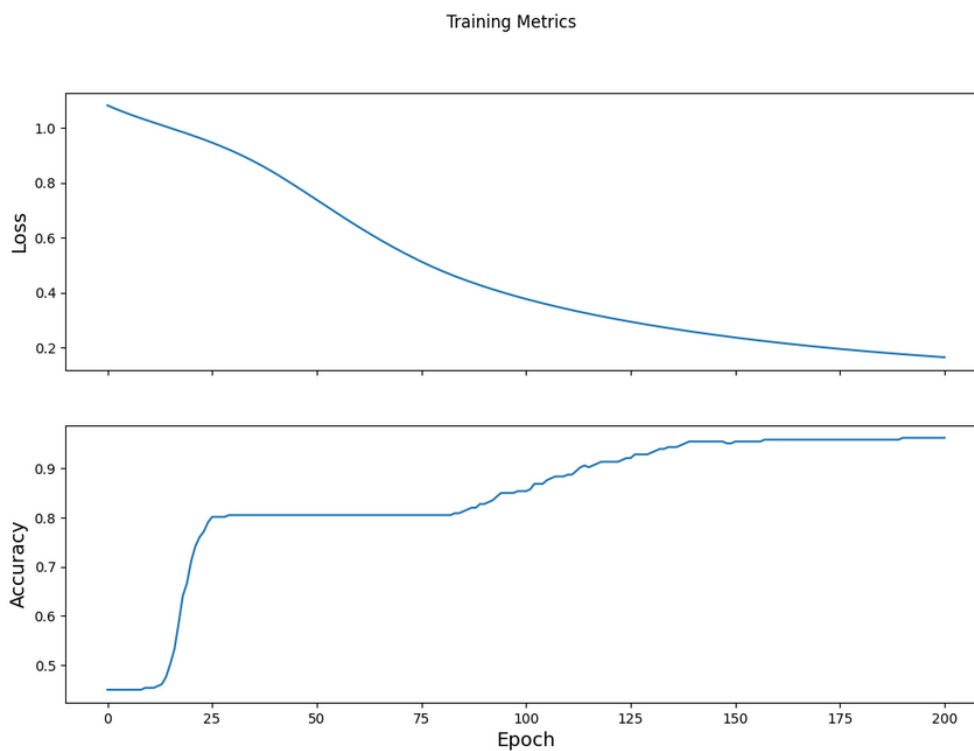
V praxi sa však najčastejšie používa kompromis medzi týmito dvoma prístupmi – Mini-Batch Gradient Descent, ktorý aktualizuje váhy po každom mini-batchi obsahujúcom N tréningových príkladov. Tréningová vzorka (batch), zvyčajne obsahuje medzi 16 a 256 tréningových príkladov v závislosti od aplikácie a tréningového hardvéru. Tento prístup kombinuje výhody oboch predchádzajúcich metód – zvyšuje stabilitu učenia oproti SGD, čo môže viesť k stabilnejšej konvergencii (Obr. 24) a zároveň znižuje výpočtovú náročnosť oproti batch gradient descent. Pretože model používa batch size o veľkosti 32 ide technicky o Mini-Batch Gradient Descent, aj keď sa optimalizačný algoritmus stále označuje ako SGD.[42, 43]



Obr. 24 – Optimalizačné algoritmy pri konvergencii[42]

Iteratívnym výpočtom priemernej straty a gradientov pre každý batch model postupne upravuje svoje parametre pomocou optimalizačného algoritmu počas tréningu, a tým minimalizuje chybovosť (Obr. 25). Dlhší tréning automaticky nezaručuje lepší model. Po určitom bode môže dôjsť k overfittingu a model stratí schopnosť generalizácie na neznáme dáta. V prípade potreby rozšírenia súboru detekovateľných gest alebo nepresnej klasifikácie modelom je potrebné rozšíriť dataset o nové tréningové dáta a zopakovať proces tréningu. Model je následne konvertovaný do formátu TFLite a uložený. TFLite je optimalizovaný na použitie predovšetkým pri mobilných a embedded zariadeniach. Pri real-time rozpoznávaní gest je však dôležité minimalizovať dobu opakovanej predikcie, teda zabezpečiť, aby bol

model čo najvýkonnejší a práve to formát TFLite umožňuje. Princíp fungovania TFLite modelu je podrobnejšie vysvetlený v časti venovanej jeho použitiu.[40, 46]



Obr. 25 – Vzťah medzi presnosťou a chybovosťou počas tréningu[40]

6 PRINCÍP FUNGOVANIA KOMUNIKÁCIE S ROBOTOM A PRENOS SÚBOROV

Nasledujúca kapitola popisuje spôsob fungovania bezdrôtovej komunikácie medzi robotom a riadiacim zariadením prostredníctvom technológie Bluetooth. Vysvetľuje postup párovania zariadení, pridelenie IP adries, ako aj využitie sieťových protokolov TCP (Transmission Control Protocol) a UDP (User Datagram Protocol) na efektívny prenos dát, a tým umožňuje jednoduché a flexibilné spojenie v reálnom čase. Okrem samotnej komunikácie je opísaný aj proces prenosu súborov, konkrétne Python skriptu, ktorý bude robotom vykonávaný.

6.1 Pripojenie a komunikácia s robotom

Pre pripojenie robota k inému zariadeniu, ktoré bude vybavené kamerou na snímanie gest je najskôr potrebné na robotovi zapnúť Bluetooth. Postup je nasledovný: v hlavnom menu kocky v sekcii Wireless and Networks → Bluetooth → Powered. Úspešné zapnutie signalizuje ikona Bluetooth v pravom hornom rohu displeja.

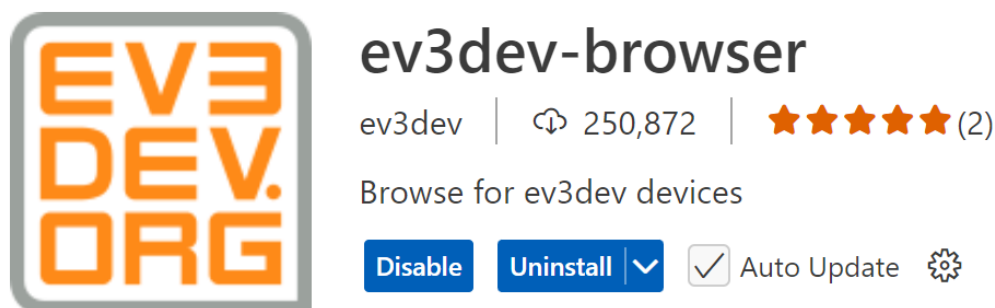
Následne je v sekcii Bluetooth potrebné zvoliť možnosť Start Scan. Po krátkej chvíli by sa malo v sekcii Devices objaviť zariadenie, s ktorým sa má robot spárovať. Je potrebné potvrdiť spárovanie na oboch stranách – na zariadení aj na robotovi. Po úspešnom spárovaní sa na displeji robota v ľavom hornom rohu zobrazí IP adresa, ktorá bude následne použitá na komunikáciu v nasledujúcej kapitole.

V tomto prípade ide spojenie typu Bluetooth Network Encapsulation Protocol (BNEP), ktoré je súčasťou profilu Personal Area Networking (PAN). Tento profil umožňuje, aby sa Bluetooth správal ako bežné sieťové rozhranie (podobne ako Ethernet alebo Wi-Fi), čo umožňuje pridelovanie IP adries zariadeniam v rámci Bluetooth siete.[44]

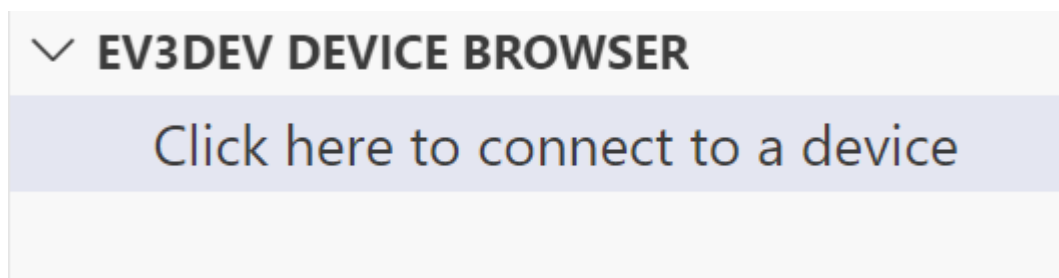
Vďaka tomu je možné medzi počítačom a robotom vytvoriť sieťové spojenie, cez ktoré sa dá komunikovať v reálnom čase prostredníctvom IP paketov pomocou protokolov TCP alebo UDP. Takéto riešenie je výhodné z hľadiska jednoduchosti implementácie rozhrania na bezdrôtovú komunikáciu a umožňuje efektívne využitie existujúcich sieťových protokolov.

6.2 Prenos súborov

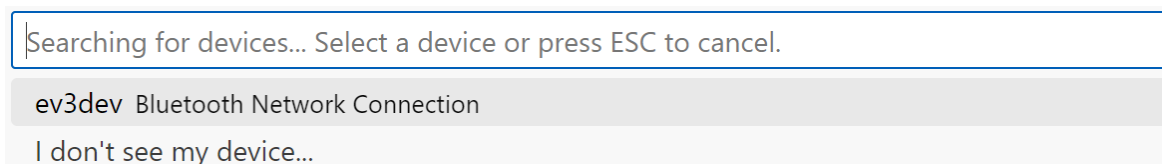
Na prenos súborov, konkrétne Python skriptu, ktorý bude robot vykonávať, je použité rozšírenie *ev3dev-browser* v dostupné v IDE Visual Studio Code (Obr. 26). Po nainštalovaní rozšírenia a úspešnom pripojení robota k riadiacemu zariadeniu cez Bluetooth je možné pomocou tohto rozšírenia vyhľadať robota, pripojiť sa k nemu a následne preniesť súbory z aktuálneho priečinka projektu. Obrázky 27 – 29 zobrazujú postup pripojenia a prenosu súborov. Priečinko so skriptom je dostupný v hlavnom menu robota v sekcii File Browser. Skript je spustiteľný priamo z robota alebo je možné preniesť a zároveň aj spustiť stlačením klávesy *F5*. Ako alternatíva sa dá použiť prenos pomocou SSH.



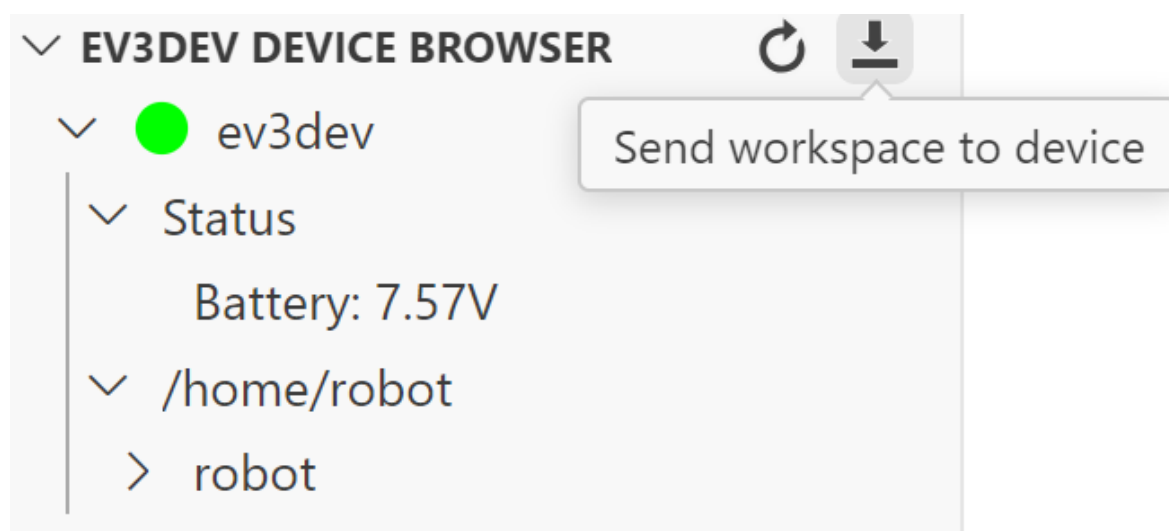
Obr. 26 – Rozšírenie ev3dev-browser



Obr. 27 – Pokus o pripojenie k robotovi



Obr. 28 – Vyhľadanie robota



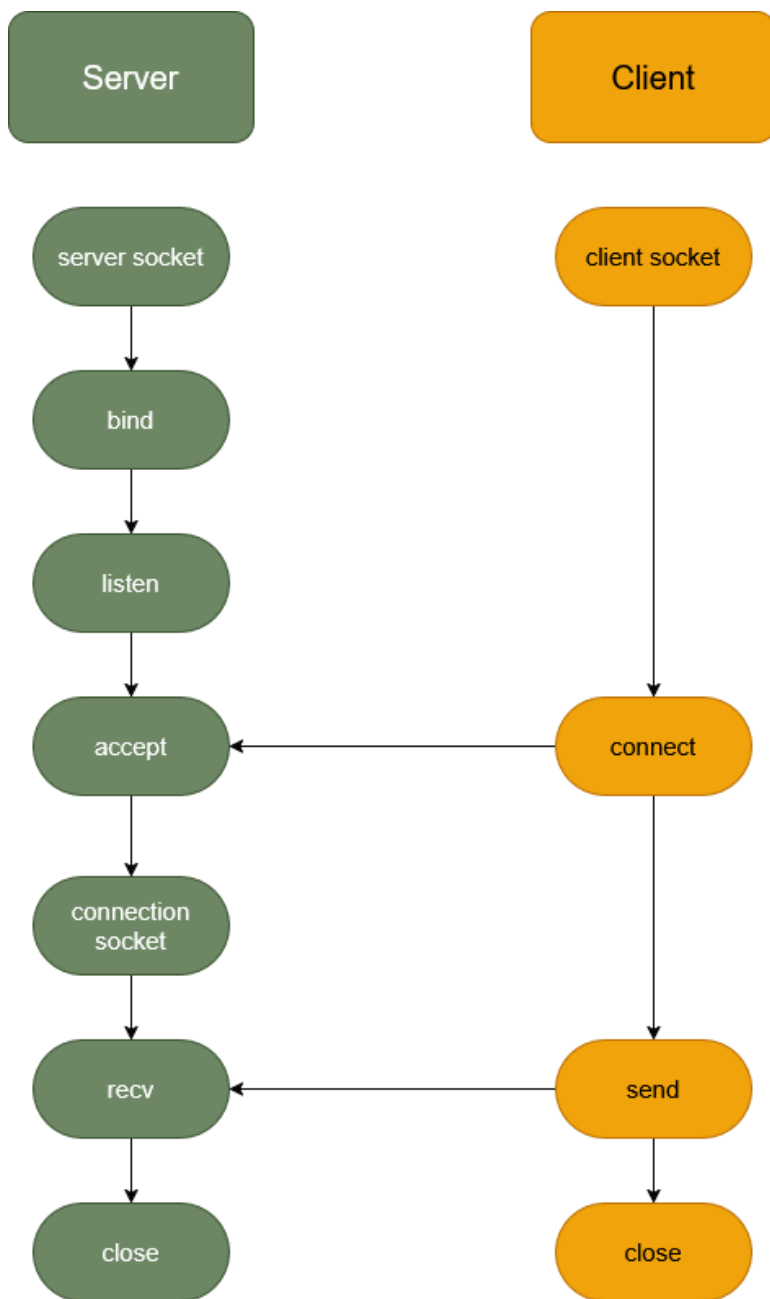
Obr. 29 – Úspešné pripojenie k robotovi

7 NÁVRH A IMPLEMENTÁCIA APLIKÁCIE

Systém pozostáva z dvoch samostatných programov – jeden bude vykonávaný na strane robota a druhý na strane zariadenia s kamerou. Oba programy používajú programovací jazyk Python.

7.1 Siet'ová komunikácia pomocou TCP socketov

Po úspešnom spárovaní zariadení prostredníctvom technológie Bluetooth je možné začať komunikáciu medzi zariadeniami. Táto komunikácia bude prebiehať pomocou protokolu TCP, ktorý zabezpečuje spoľahlivý prenos dát. V rámci tejto architektúry robot vystupuje ako TCP server, ku ktorému sa klient (riadiace zariadenie) pripája pomocou IPv4 adresy robota. Každý TCP paket odoslaný z klienta obsahuje príkaz, ktorý reprezentuje detekované gesto a rýchlosť, akou má robot tento príkaz vykonať. Na implementáciu tejto komunikácie sú použité sockety, ktoré sú súčasťou Python *socket* modulu. Sockety pre server a klienta sa inicializujú nasledovne: `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`. Pri vytváraní socketov je potrebné špecifikovať rodinu adries, ktoré bude socket používať v tomto prípade *AF_INET* označuje IPv4 adresy, teda adresy využívajúce *Internet Protocol* verzie 4. *SOCK_STREAM* určuje typ protokolu používaného na prenos – TCP. Na nasledujúcom obrázku (Obr. 30) je znázornená sekvencia volaní socket API, ktorá ilustruje celý proces od vytvorenia socketu až po prijímanie a odosielanie správ medzi zariadeniami.[45]



Obr. 30 – Sekvencia volaní socket API

Metódy *bind()*, *listen()* a *accept()* sú špecifické pre serverový socket. Metóda *bind((HOST, PORT))* slúži na priradenie socketu ku konkrétnemu sieťovému rozhraniu a portu. HOST je nastavený ako prázdny reťazec (""), čo zodpovedá IPv4 adrese 0.0.0.0 – server socket s touto adresou počúva na všetkých prístupných sieťových rozhraniach, a teda aj na tom, ktoré bolo vytvorené pomocou Bluetooth PAN. Port predstavuje číslo, na ktorom server očakáva pripojenia od klienta. Port je predvolene nastavený na 12345. Metóda *listen()* umožní serveru prijímať pripojenia – socket sa tým stáva „počúvajúcim“. Metóda *accept()* pozastaví vykonávanie programu robota a čaká na prichádzajúce spojenie. Keď sa klient

pripojí pomocou *connect((HOST, PORT))* (HOST IPv4 adresa robota, PORT – 12345) metóda vráti nový socket, ktorý reprezentuje toto konkrétne spojenie. Práve tento socket sa následne používa na prijatie príkazov od klienta. Je odlišný od pôvodného „počúvajúceho“ socketu, ktorý by naďalej mohol prijímať nové spojenia. Po nadviazaní spojenia medzi klientom a serverom sa na výmenu údajov používajú metódy *send()* a *recv()*, pričom pomocou *send()* klient odosiela príkazy na vykonanie a server socket reprezentujúci toto spojenie ich prijíma s použitím *recv()*. V prípade, že príde príkaz na ukončenie komunikácie použitie blokov *with* s každým socketom zabezpečuje, že sa sockety automaticky uzavru bez nutnosti volania *close()*. [45]

7.2 Program riadiaceho zariadenia

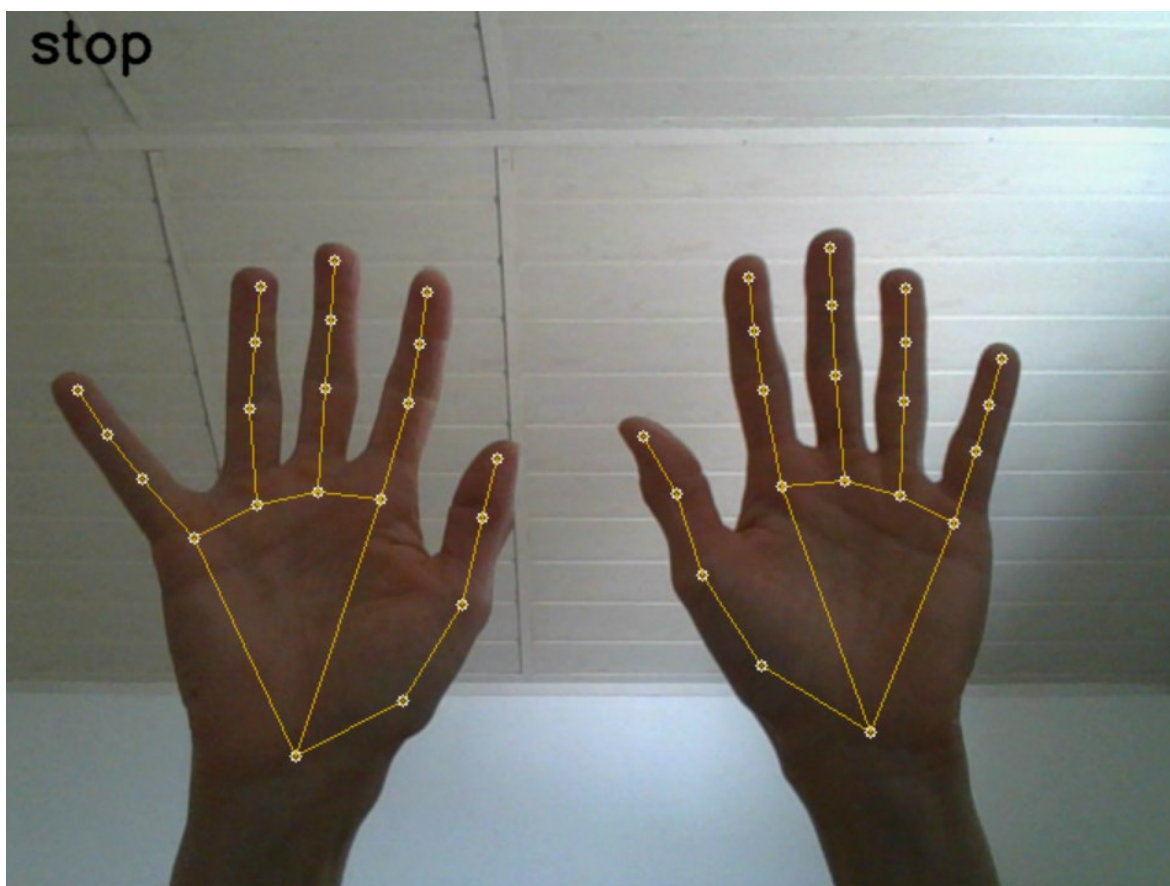
Program na strane klienta začína overením poskytnutej IPv4 adresy a portu (v prípade poskytnutia nového portu), následne sa pokúsi nadviazať spojenie s robotom. Po úspešnom pripojení je možné ovládať robota pomocou gest zachytených kamerou. Na získanie a spracovanie obrazu je opäť použitá knižnica *OpenCV*. *MediaPipe Hands* detekuje ruky v obraze a pomocou metódy *process(image)* extrahuje body rúk. Extrahované body sú normalizované a vyhodnotené natrénovaným modelom detekcie gest. [32]

Použitie modelu detekcie gest

Keďže model je vo formáte TFLite na fungovanie je potrebný *TensorFlow Lite Interpreter*, ktorý je schopný načítať a spustiť modely v tomto formáte. Pri inicializácii interpreteru je nutné poskytnúť ako parameter cestu k modelu. Interpreter potom zavolá funkciu *allocate_tensors()*, aby alokoval pamäť a inicializoval všetky vstupné, výstupné a prechodné tensor, ktoré bude model používať pri predikciách. Tensor je viacrozmerné pole, ktoré sa používa na výpočet a obsahuje údaje, ako sú vstupy/výstupy modelu, váhy, biasy, medzivýsledky (aktivácie medzi vrstvami). Po alokácii interpreter načíta podrobnosti o vstupnom a výstupnom tensore, aby zistil očakávaný tvar a dátový typ vstupných a výstupných dát s použitím metód *get_input_details()* a *get_output_details()*. Údaje sa pretransformujú a upravujú tak, aby zodpovedali vstupným požiadavkám modelu, načítajú sa do vstupného tensora *set_tensor(input_tensor, input_data)* a spustí sa predikcia pomocou metódy *invoke()*. Výsledok sa potom extrahuje z výstupného tensora – *get_tensor(output_tensor)*, čím sa dokončí proces inferencie. Vďaka tomu, že model je už načítaný do pamäti a interpreter má alokovanú pamäť pre všetky potrebné tensor, je opakovaný výpočet klasifikácie rýchly a efektívny. [46, 47, 48]

Interpretácia detekovaných gest rúk

Model predikuje gesto pre každú ruku zvlášť a vráti triedu s najväčšou pravdepodobnosťou. Výsledný príkaz je tvorený kombináciou detekovaných tried ľavej a pravej ruky. Každá kombinácia gest má priradený konkrétny kód príkazu. Napríklad kombinácia detekovaných gest s triedami 1 (ľavá ruka) a 1 (pravá ruka) predstavuje príkaz „stop“. V prípade, že nie sú v obraze detekované obe ruky je príkaz automaticky interpretovaný ako „stop“.



Obr. 31 – Detekcia príkazu „stop“

Minimalizácia chyby pri detekcii a vyhodnotení príkazu

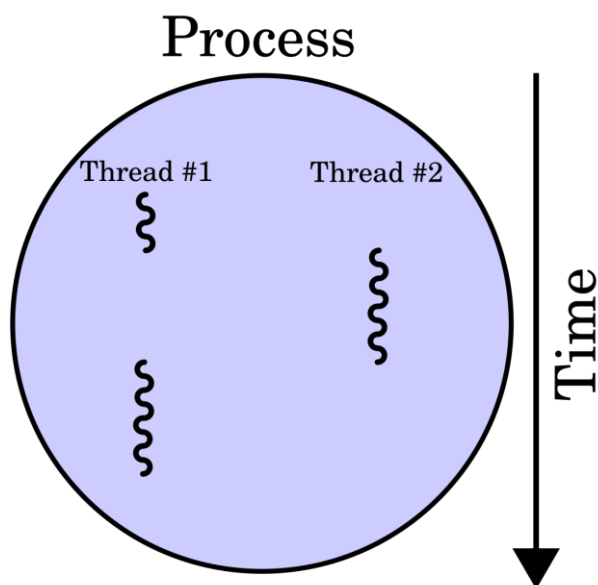
Aby sa predišlo chybnému vyhodnocovaniu gest a následných príkazov pri ich zmene počas pohybu ruky, je použitá dátová štruktúra *deque* (double-ended queue) s maximálnou kapacitou 8, ktorá slúži ako história príkazov. V prípade, že je kapacita plná sú staré príkazy (ľavá strana) odstránene a nové príkazy sú pridané na koniec (pravá strana). Týmto spôsobom sa minimalizujú náhodné zmeny príkazov spôsobené transformáciou rúk pri

zmene gest. Nový príkaz je odoslaný robotovi len vtedy, ak najčastejšie sa opakujúci príkaz v histórii sa nerovná aktuálne vykonávanému príkazu.

7.3 Program robota

K ovládaniu motorov robota je použitá knižnica *ev3dev2*, ktorá je súčasťou operačného systému *ev3dev*. Táto knižnica poskytuje jednoduchý spôsob ovládania motorov pomocou triedy *MoveTank(left_motor_port, right_motor_port)*. Trieda prijíma dva parametre a to sú výstupné porty, ku ktorým sú motory pripojené. Súčasťou tejto triedy sú metódy *on(left_speed, right_speed)* a *stop()* pre ovládanie robota. Metóda *on()* umožňuje pohyb dopredu a dozadu pri aplikácii rovnakej kladnej alebo zápornej rýchlosti na oba motory. Pre pohyb vpravo alebo vľavo je nutné aplikovať rozdielne rýchlosti podľa potreby smeru pohybu.

Keďže ide o sieťovú komunikáciu, ktorá patrí medzi blokujúce I/O operácie, je potrebné zaistiť, aby bol robot schopný súčasne vykonávať aktuálny príkaz a zároveň reagovať na nové príkazy prijaté zo siete. Toto je docielené pomocou vlákien (angl. *threads*), ktoré umožňujú súbežné vykonávanie rôznych častí kódu v rámci jedného procesu. Bežiaci program tak predstavuje proces, ktorý obsahuje dve vlákna (Obr. 32): *NetworkThread* – zodpovedné za prijímanie a spracovanie dát zo sieťového spojenia a *CommandThread* – vykonávajúce príkazy prijaté prostredníctvom komunikácie.



Obr. 32 – Vykonávanie vlákien v rámci procesu[49]

Synchronizácia vlákien

Pri práci s viacerými vláknami v rámci procesu je dôležité zaistiť tzv. thread safety – teda zabezpečiť, aby program fungoval správne aj pri súčasnom vykonávaní kódu viacerými vláknami. Bez správnej synchronizácie môže dôjsť k súbehu (angl. *race condition*) – situácii, keď výsledok programu závisí od nepredvídateľného poradia vykonávania vlákien. Ak napríklad jedno vlákno zapisuje a druhé číta príkaz môže to spôsobiť čiastočnú aktualizáciu hodnoty premennej, a tým pádom nekonzistentnosť. Synchronizácia je preto kľúčová na zaistenie deterministického a spoľahlivého správania procesu s viacerými vláknami. Program využíva synchronizačný mechanizmus *Condition*. Objekt *Condition* zaručuje, že pomocou bloku *with* sekcia kódu, kde sa číta alebo zapisuje do premenných *command* a *speed* sú chránené zámkom, čiže iba jedno vlákno má prístup k týmto premenným v jednom okamihu. *Condition* taktiež poskytuje metódy *wait_for()* a *notify()*. Pomocou týchto metód je možné zabezpečiť, že v prípade príkazu „stop“ sa vlákno *CommandThread* uspí (zablokuje) a čaká na volanie metódy *notify()*. Vďaka tomu nebude nutné neustále kontrolovať zmenu premennej *command*, čím sa šetrí systémové zdroje. Ak však príde nový príkaz metóda *notify()* volaná z *NetworkThread* zobudí (odblokuje) vlákno *CommandThread*, ktoré vyhodnotí podmienku *wait_for(lambda: self._command != "stop")* a pokiaľ príkaz nie je „stop“ vlákno začne s vykonávaním nového príkazu v opačnom prípade je vlákno opäť blokované.[50]

ZÁVER

Práca bola vypracovaná s dôrazom na splnenie stanovených bodov zadania. V teoretickej časti bola podrobne rozpracovaná problematika počítačového videnia a detekcie gest, pričom osobitná pozornosť bola venovaná princípom fungovania konvolučných neurónových sietí – kľúčovej technológii pri analýze a spracovaní obrazových dát. V kapitole venovanej detekcii gest boli predstavené možnosti detekcie a identifikované výzvy, ktoré pri tomto procese môžu nastať.

V nasledujúcej kapitole boli preskúmané a popísané možnosti ovládania robota LEGO EV3 pomocou základného LEGO softwaru alebo s použitím knižníc. Vzhľadom na to, že natívny firmware robota nepodporuje sieťovú komunikáciu, bol použitý operačný systém ev3dev založený na Linuxe, ktorý túto funkcionality umožňuje. Praktická časť zahŕňa zostavenie robota s možnosťou pohybu a prípravu microSD karty s operačným systémom ev3dev.

Na detekciu rúk v reálnom čase bol zvolený framework MediaPipe Hands, pričom samotné vyhodnocovanie gest zabezpečuje neurónová sieť vytvorená a natrénovaná pomocou knižnice TensorFlow. Toto riešenie vďaka svojej flexibilitě umožňuje jednoducho rozšíriť súbor detekovateľných gest alebo ho prispôbiť vlastným potrebám. Na získavanie a spracovanie obrazových dát bola použitá knižnica OpenCV.

Bezdrôtová komunikácia medzi riadiacim zariadením a robotom bola realizovaná prostredníctvom protokolu Bluetooth Network Encapsulation Protocol (BNEP) a siete Personal Area Network. Tento prístup umožnil využívať protokol TCP na odosielanie príkazov robotovi. Ovládanie motorov robota bolo zabezpečené prostredníctvom Python knižnice ev3dev2, ktorá je súčasťou operačného systému ev3dev.

Výsledný software pozostáva z dvoch častí – jedna časť beží na riadiacom zariadení s kamerou, druhá priamo na robotovi. Tento software bol dôkladne zdokumentovaný – vrátane zdôvodnenia a popisu architektúry, spôsobu komunikácie, princíp detekcie gest a celkového fungovania systému. Práca úspešne demonštruje ovládanie robota pomocou gest rúk v reálnom čase a splňa všetky body zadania bakalárskej práce.

ZOZNAM POUŽITEJ LITERATÚRY

- [1] What is computer vision? IBM [online]. 27. júl 2021 [cit. 2025-02-8]. Dostupné na internete: <https://www.ibm.com/think/topics/computer-vision>
- [2] What is deep learning? IBM [online]. 17. jún 2024 [cit. 2025-02-10]. Dostupné na internete: <https://www.ibm.com/think/topics/deep-learning>
- [3] What are convolutional neural networks? IBM [online]. [cit. 2025-02-11]. Dostupné na internete: <https://www.ibm.com/think/topics/convolutional-neural-networks>
- [4] SKALSKI Piotr. Gentle Dive into Math Behind Convolutional Neural Networks. Medium [online]. 12. apríl 2019 [cit. 2025-02-11]. Dostupné na internete: <https://medium.com/data-science/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9>
- [5] HESARAKI Saba. Feature Map. Medium [online]. 18. október 2023 [cit. 2025-02-15]. Dostupné na internete: <https://medium.com/@saba99/feature-map-35ba7e6c689e>
- [6] Convolution Neural Network for image classification. Clemson University [online]. [cit. 2025-02-15]. Dostupné na internete: https://clemsonciti.github.io/rcde_workshops/python_deep_learning/07-Convolution-Neural-Network.html
- [7] Student Notes: Convolutional Neural Networks (CNN) Introduction. IndoML [online]. [cit. 2025-02-18]. Dostupné na internete: <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>
- [8] Convolutional Neural Networks. Stanford University [online]. [cit. 2025-05-18]. Dostupné na internete: <https://cs231n.github.io/convolutional-networks/>
- [9] WIELE Christian. Why Are Neural Nets Non-linear? Medium [online]. 23. október 2020 [cit. 2025-02-22]. Dostupné na internete: <https://medium.com/swlh/why-are-neural-nets-non-linear-a46756c2d67f>
- [10] KEITA Zoumana. An Introduction to Convolutional Neural Networks (CNNs). Datacamp [online]. 14. november 2023 [cit. 2025-02-25]. Dostupné na internete: <https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns>

- [11] ROY Abhijit. An Introduction to Gradient Descent and Backpropagation. Medium [online]. 14. jún 2020 [cit. 2025-02-25]. Dostupné na internete: <https://medium.com/data-science/an-introduction-to-gradient-descent-and-backpropagation-81648bdb19b2>
- [12] JETHWA Akhil. Understanding Convolutional Hyperparameters. Kaggle [online]. [cit. 2025-03-01]. Dostupné na internete: <https://www.kaggle.com/code/akhiljethwa/understanding-convolutional-hyperparameters>
- [13] ITO ARAMENDIA Alejandro. Convolutional Neural Networks (CNNs): A complete guide. Medium [online]. 1. január 2024 [cit. 2025-05-18]. Dostupné na internete: <https://medium.com/@alejandritoaramendia/convolutional-neural-networks-cnns-a-complete-guide-a803534a1930>
- [14] SENGUPTA Joy. How to decide the hyperparameters in CNN. Medium [online]. 21. máj 2025 [cit. 2025-03-01]. Dostupné na internete: <https://medium.com/@sengupta.joy4u/how-to-decide-the-hyperparameters-in-cnn-bfa37b608046>
- [15] Schematic of 2D CNN for stride 1 and stride 2 with zero padding adding additional layers. ResearchGate [online]. [cit. 2025-03-01]. Dostupné na internete: https://www.researchgate.net/figure/Schematic-of-2D-CNN-for-stride-1-and-stride-2-with-zero-padding-adding-additional-layers_fig13_350386951
- [16] GUAN Di. Pooling Layers. guandi1995.github.io [online]. [cit. 2025-03-08]. Dostupné na internete: <https://guandi1995.github.io/Pooling-Layers/>
- [17] JAIN Abhishek. Pooling and their types in CNN. Medium [online]. 12. február 2024 [cit. 2025-03-08]. Dostupné na internete: <https://medium.com/@abhishekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611>
- [18] Convolutional Neural Network | Deep Learning. Developers Breach [online]. 21. august 2020 [cit. 2025-03-13]. Dostupné na internete: <https://developersbreach.com/convolution-neural-network-deep-learning/>

- [19] MULLANY Lila. Introduction to Computer Vision Model Training. Medium [online]. 9. jún 2020 [cit. 2025-03-13]. Dostupné na internete: <https://medium.com/data-science/introduction-to-computer-vision-model-training-c8d22a9af22b>
- [20] Image segmentation detailed overview. SuperAnnotate [online]. 22. november 2023 [cit. 2025-03-13]. Dostupné na internete: <https://www.superannotate.com/blog/image-segmentation-for-machine-learning>
- [21] Training and Optimizing a 2D Pose Estimation Model with NVIDIA TAO Toolkit, Part 1. NVIDIA [online]. 10. jún 2021 [cit. 2025-03-18]. Dostupné na internete: <https://developer.nvidia.com/blog/training-optimizing-2d-pose-estimation-model-with-tao-toolkit-part-1/>
- [22] DL, Gen. Devin. How to Train Best Models of Deep Learning in Computer Vision. Medium [online]. 24. december 2023 [cit. 2025-03-22]. Dostupné na internete: <https://medium.com/@tubelwj/how-to-train-best-models-of-deep-learning-in-computer-vision-d5bdab10400d>
- [23] What is image annotation? Introduction to image annotation for machine learning. SuperAnnotate [online]. 5. september 2022 [cit. 2025-03-25]. Dostupné na internete: <https://www.superannotate.com/blog/introduction-to-image-annotation>
- [24] What is data augmentation? IBM [online]. 7. máj 2024 [cit. 2025-03-25]. Dostupné na internete: <https://www.ibm.com/think/topics/data-augmentation>
- [25] Overfit and underfit. TensorFlow [online]. [cit. 2025-04-02]. Dostupné na internete: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
- [26] MISHRA Priya. Transfer Learning with Convolutional Neural Networks. Tutorialspoint [online]. 12. júl 2023 [cit. 2025-04-7]. Dostupné na internete: <https://www.tutorialspoint.com/transfer-learning-with-convolutional-neural-networks>
- [27] The Use Cases of Computer Vision. Glair [online]. 14. február 2022 [cit. 2025-04-15]. Dostupné na internete: <https://glair.ai/post/the-uses-case-of-computer-vision>
- [28] Popular applications of computer vision in smart cities. Viso [online]. [cit. 2025-04-15]. Dostupné na internete: <https://viso.ai/applications/computer-vision-in-smart-cities/>
- [29] What is Computer Vision? Hyperscience [online]. [cit. 2025-04-15]. Dostupné na internete: <https://www.hyperscience.ai/resource/computer-vision/>

- [30] Gesture recognition. Wikipedia [online]. [cit. 2025-04-20]. Dostupné na internete: https://en.wikipedia.org/wiki/Gesture_recognition
- [31] HARINI, K. a S. UMA MAHESWARI. A novel static and dynamic hand gesture recognition using self organizing map with deep convolutional neural network. AUTOMATIKA [online]. 29. august 2023 [cit. 2025-04-22]. Dostupné na internete: <https://www.tandfonline.com/doi/full/10.1080/00051144.2023.2251229>
- [32] GOOGLE. MediaPipe Hands. MediaPipe Documentation [online]. [cit. 2025-04-25]. Dostupné na internete: <https://mediapipe.readthedocs.io/en/latest/solutions/hands.html>
- [33] GOOGLE. Gesture recognition guide for Python. Google AI [online]. [cit. 2025-04-26]. Dostupné na internete: https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer/python
- [34] Keras: The high-level API for TensorFlow. TensorFlow [online]. [cit. 2025-04-27]. Dostupné na internete: <https://www.tensorflow.org/guide/keras>
- [35] SANSING Chad. Build Better Robots with LEGO Mindstorms Education EV3. School Library Journal [online]. 24. február 2014 [cit. 2025-04-28]. Dostupné na internete: <https://www.schoollibraryjournal.com/story/build-better-robots-with-lego-mindstorms-education-ev3-slj-review>
- [36] ROBERTSON Adi. Lego's new Mindstorms EV3 robotics platform comes with Android and iOS support. The Verge [online]. 7. január 2013 [cit. 2025-04-28]. Dostupné na internete: <https://www.theverge.com/2013/1/7/3845222/lego-introduces-android-ios-compatible-ev3-mindstorms-robotics>
- [37] FLOT Jesse. Help for the LEGO EV3 Robotics Teacher. Carnegie Mellon University [online]. 19. august 2021 [cit. 2025-04-28]. Dostupné na internete: https://www.cmu.edu/roboticsacademy/news/cmra-news-2021/ev3_teacher_help.html
- [38] EV3DEV. Oficiálna dokumentácia a podpora pre ev3dev. ev3dev.org [online]. [cit. 2025-05-18]. Dostupné na internete: <https://www.ev3dev.org/>
- [39] The Sequential model. Keras [online]. 25. jún 2023 [cit. 2025-04-29]. Dostupné na internete: https://keras.io/guides/sequential_model/
- [40] Custom training: walkthrough. TensorFlow [online]. [cit. 2025-04-29]. Dostupné na internete: https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough

- [41] BROWNLEE Jason. [4-Difference Between a Batch and an Epoch in a Neural Network. Machine Learning Mastery [online]. 15. august 2022 [cit. 2025-04-30]. Dostupné na internete: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
- [42] SIDDELL Kathleen. Gradient Descent Algorithm – Optimization for Machine Learning. alwaysAI [online]. 10. január 2023 [cit. 2025-05-1]. Dostupné na internete: <https://alwaysai.co/blog/what-is-gradient-descent>
- [43] TANNER Gilbert. Gradient Descent. ML Explained [online]. 3. máj 2021 [cit. 2025-05-2]. Dostupné na internete: <https://ml-explained.com/blog/gradient-descent-explained>
- [44] Bluetooth Network Encapsulation Protocol 1.0. Bluetooth [online]. [cit. 2025-05-4]. Dostupné na internete: <https://www.bluetooth.com/specifications/specs/bluetooth-network-encapsulation-protocol-1-0/>
- [45] JENNINGS Nathan. Socket Programming in Python (Guide). Real Python [online]. 7. december 2024 [cit. 2025-05-8]. Dostupné na internete: <https://realpython.com/python-sockets/>
- [46] GOOGLE. TensorFlow Lite inference. Google Source [online]. [cit. 2025-05-11]. Dostupné na internete: <https://android.googlesource.com/platform/external/tensorflow/+/ec63214f098a2bfc87b628219ad0718750d4e930/tensorflow/lite/g3doc/guide/inference.md#Load-and-run-a-model-in-Python>
- [47] tf.lite.Interpreter. TensorFlow [online]. [cit. 2025-05-14]. Dostupné na internete: https://www.tensorflow.org/api_docs/python/tf/lite/Interpreter
- [48] Introduction to Tensor with Tensorflow. GeeksforGeeks [online]. 25. február 2025 [cit. 2025-05-14]. Dostupné na internete: <https://www.geeksforgeeks.org/introduction-tensor-tensorflow/>
- [49] Thread (computing). Wikipedia [online]. [cit. 2025-05-17]. Dostupné na internete: https://en.wikipedia.org/wiki/Thread_%28computing%29
- [50] DIVAKARAN Adarsh. Python Thread Safety: Using a Lock and Other Techniques. Real Python [online]. 23. október 2024 [cit. 2025-05-17]. Dostupné na internete: <https://realpython.com/python-thread-lock/>

ZOZNAM OBRÁZKOV

Obr. 1 – Proces konvolúcie[3]	14
Obr. 2 – Feature maps po aplikácii 16 filtrov	15
Obr. 3 – Detekcia častí objektu až po celý objekt[3]	15
Obr. 4 – RGB kanály obrázku[6]	16
Obr. 5 – Proces konvolúcie pri RGB obrázkoch[7]	17
Obr. 6 – Pripočítanie biasu a aplikácia nelineárnej funkcie ReLU[7]	18
Obr. 7 – Pridanie núl po okrajoch[14]	19
Obr. 8 – Použitie hyperparametru Stride[15]	19
Obr. 9 – Použitie funkcie Max pool[17]	20
Obr. 10 – Feature maps pred aplikáciou a po aplikácii max pooling.....	21
Obr. 11 – Dáta z Flatten layer vstupujúce do Fully-connected layer[13]	22
Obr. 12 – Konvolučná neurónová sieť[18]	22
Obr. 13 – Klasifikácia, detekcia a segmentácia objektov[20].....	23
Obr. 14 – Detekcia ľudského tela[21]	24
Obr. 15 – Dátová augmentácia[24]	26
Obr. 16 – Lokalizácia 21 bodov ruky[32]	29
Obr. 17 – Stavebnica LEGO Mindstorms EV3[35]	31
Obr. 18 – Porty kocky EV3[36]	32
Obr. 19 – EV3 Classroom[37]	32
Obr. 20 – LEGO robot s možnosťou pohybu.....	35
Obr. 21 – Nahratie ev3dev na microSD kartu pomocou <i>balenaEtcher</i>	36
Obr. 22 – Architektúra modelu	38
Obr. 23 – Výpis počas tréningu modelu	40
Obr. 24 – Optimalizačné algoritmy pri konvergencii[42].....	42
Obr. 25 – Vzťah medzi presnosťou a chybovosťou počas tréningu[40].....	43
Obr. 26 – Rozšírenie ev3dev-browser.....	45
Obr. 27 – Pokus o pripojenie k robotovi	45
Obr. 28 – Vyhľadanie robota	45
Obr. 29 – Úspešné pripojenie k robotovi	46
Obr. 30 – Sekvencia volaní socket API	48
Obr. 31 – Detekcia príkazu „stop“	50
Obr. 32 – Vykonávanie vlákien v rámci procesu[49]	51

ZOZNAM TABULIEK

Tab. 1 – Příklad dimenzií filtrov jednotlivých konvolučních vrstev	17
Tab. 2 – Příklad modelov pre vykonávanie konkrétnej úlohy	24

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

2D	Two-Dimensional
3D	Three-Dimensional
Adam	Adaptive Moment Estimation
API	Application Programming Interface
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
CV	Computer Vision
FC	Fully Connected (layer)
IDE	Integrated Development Environment
I/O	Input/Output
IP	Internet Protocol
IPv4	Internet Protocol version 4
MRI	Magnetic Resonance Imaging
OCR	Optical Character Recognition
OpenCV	Open-Source Computer Vision Library
OS	Operating System
RAM	Random Access Memory
ResNet	Residual Network
SGD	Stochastic Gradient Descent
SSH	Secure Shell
TCP	Transmission Control Protocol
TFLite	TensorFlow Lite