VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Využití metod umělé inteligence v řídicích systémech s programovatelnými automaty

Use of Artificial Intelligence Methods in Control Systems with Programmable Controllers

Martin Benek

Bakalářská práce

Vedoucí práce: prof. Ing. Jiří Koziorek, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/080616
vsboee220269FC

Zadání bakalářské práce

Student: **Martin Benek**

Studijní program: B0714A150001 Řídicí a informační systémy

Téma: Využití metod umělé inteligence v řídicích systémech s programovatelnými automaty
Use of Artificial Intelligence Methods in Control Systems with Programmable Controllers

Jazyk vypracování: čeština

Zásady pro vypracování:

Téma se bude zabývat možnostmi využití modulu Simatic NPU v řídicích aplikacích. Testování bude probíhat ve Smart Factory, FEI, VŠB-TU Ostrava.

1. Analýza možností využití metod umělé inteligence v řídicích systémech s programovatelnými automaty.
2. Rozbor typických úloh, pro které lze využít přístupy umělé inteligence v průmyslových aplikacích.
3. Seznámení se s modulem Simatic NPU.
4. Návrh experimentu ověřujícího využití metod umělé inteligence v rámci Simatic NPU.
5. Realizace experimentu v laboratoři a zhodnocení dosažených výsledků.
6. Závěr.

Seznam doporučené odborné literatury:

[1] BERGER, Hans. *Automating with SIMATIC.* 5th edition. Erlangen, Germany: Publicis Publishing, 2013, 284 p. ISBN 978-3895783876.
[2] Technická dokumentace k použitým systémům a SW Siemens.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **prof. Ing. Jiří Koziorek, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního programu: prof. Ing. Petr Bilík, Ph.D.

V IS EDISON zadáno: 16.11.2022 07:10:36

Abstrakt

Bakalářská práce se zabývá možnostmi využití modulu pro umělou inteligenci (AI) SIMATIC S7-1500 TM NPU od firmy Siemens. AI je konkrétně využita ve formě konvoluční neuronové sítě (CNN) ke klasifikaci obrázků pořízených kamerou. V teoretické části je seznámení se základními principy umělé inteligence a popisem CNN, které se používají pro zpracování obrazu. Je zde rovněž rozbor oblastí, kde se AI dá využít společně s programovatelnými automaty (PLC). Nakonec je popsán modul TM-NPU, který se využívá jako výpočetní jednotka pro neuronové sítě. Praktická část se věnuje vytvoření klasifikačního modelu pro klasifikaci typu dřevěných puků a jeho nasazení do modelu TM-NPU. Dále se zabývá vytvořením programu pro PLC, který načítá klasifikační data z modulu a dále s nimi pracuje. Cílem práce je úspěšná demonstrace úlohy, využívající metod umělé inteligence (klasifikace) pomocí modulu TM-NPU.

Klíčová slova

umělá inteligence, Tensorflow, Keras, klasifikace obrázků, automatizace, průmysl 4.0, PLC, TM-NPU

Abstract

This Bachelor thesis describes usage of module for artificial inteligence (AI) SIMATIC S7-1500 TM NPU by Siemens. In particular, AI is used in form of convolutional neural network (CNN) for classification of images taken by camera. In theoretical part there is a familiarization with basic principles of AI and description of CNNs which are used for image processing. Next, there is an analysis on common usages of AI with PLC. Finally, the module TM-NPU which is used as computing capacity for neural networks is described. Practical part is devoted to creation of classification model and it's deployement to TM-NPU. Next, it deals with creation of a PLC program, which process classification data received from the module and perform certain actions based on that data. The aim of this thesis is to successfully demonstrate task, which exploits AI methods (classification) on TM-NPU.

Keywords

artificial intelligence, Tensorflow, Keras, image classification, automatization, industry 4.0, PLC, TM-NPU

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu prof. Ing. Jiřímu Koziorkovi, Ph.D. za poskytnuté konzultace a rady při řešení této práce. Dále bych rád poděkoval panu Ing. Martinu Mikolajkovi, Ph.D. za pomoc a konzultace při programování PLC. Rovněž bych chtěl poděkovat panu Ing. Radku Hofírkovi a panu Christophu Litschauerovi, MSc za poskytnuté materiály a cenné rady k modulu TM-NPU. V neposlední řadě bych chtěl poděkovat panu Ing. Jakubovi Beránkovi za pomoc s teoretickou i praktickou částí týkající se neuronových sítí.

Obsah

Seznam použitých zkratek a symbolů

CNN	–	Convolutional neural network
PLC	–	Programmable logic controller
HMI	–	Human-machine interface
AI	–	Artificial intelligence
GPU	–	Graphics processing unit
TPU	–	Tensor processing unit
CPU	–	Central processing unit
NPU	–	Neural processing unit
VPU	–	Vision processing unit
RGB	–	Red, Green, Blue (barevný model)
RL	–	Reinforcement learning
VF	–	Value function
ReLU	–	Rectifier linear unit
DT	–	Decision tree
SVM	–	Support vector machines
K-NN	–	K-nearest neighbours
RUL	–	Remaining useful life
RMS	–	Efektivní hodnota
MTTF	–	Mean time to failure
NNPLC	–	Neural network and programmable logic controller
FPL	–	Fuzzy Programming Language
SIFT	–	Scale-invariant feature transform
SURF	–	Speeded up robust features
CCD	–	charge coupled device
CMOS	–	Complementary Metal-Oxide Semicon- ductor
R-CNN	–	Region-Based Convolutional Neural Network
YOLO	–	You only look once
SSD	–	Single shot detector

CUDA	–	Compute Unified Device Architecture
cuDNN	–	CUDA deep neural network
FTP	–	File transfer protocol
IP	–	Internet protocol
USB	–	Universal serial bus
ONNX	–	Open Neural Network Exchange
HSP	–	Hardware support package
ST	–	Structured text

Seznam obrázků

Seznam výpisů zdrojového kódu

Kapitola 1

Úvod do problematiky

V posledních letech, kdy se často mluví o čtvrté průmyslové revoluci, je často zmiňována právě umělá inteligence. Zatímco v ostatních průmyslových revolucích byla usnadňována fyzická práce, kterou vykonával člověk, v této revoluci se již hovoří o usnadňování práce duševní. Umělá inteligence hraje v tomto nahrazování významnou roli. Její základní funkcí je dojít k optimálnímu rozhodnutí na základě vstupních dat a předchozích zkušeností. Uvedeme-li jako příklad duševní práce vizuální kontrolu výrobků, můžeme zmínit několik výhod, které má umělá inteligence oproti člověku.
Člověk je ovlivněn faktory jako je únava, emoce, kvalita zraku a vlivy prostředí. Stroj žádný z těchto faktorů neovlivňuje, navíc jeho rychlost rozhodování je mnohonásobně vyšší než u člověka. Stroj se navíc při stejných vstupních datech rozhodne vždy stejně.

Model pro umělou inteligenci TM-NPU představuje tzv. „AI on Edge“ technologii. Její základní charakteristikou je, že výpočty pro umělou inteligenci jsou prováděny v blízkosti polohy, kde se získávají vstupní data. V našem případě to znamená, že obraz z kamery je zpracováván přímo v modulu, který je na tuto kameru napojen a zároveň komunikuje s PLC. (Bez použití AI on Edge by se výpočty musely provádět např. na vzdáleném dedikovaném serveru).

K rozpoznávání obrázků nebo videa se velmi často používají předtrénované modely, například modely patřící do rodiny Mobilenet, SSD či YOLO. Tyto modely jsou vlastně neuronové sítě, které jsou již optimalizovány pro detekci objektů či klasifikaci obrázků. Na uživateli je, aby modely upravil podle svých potřeb (např. „dotrénoval“ sítě pomocí vlastních obrázků). Jazyk, který se nejčastěji používá pro tvorbu a úpravu modelů je Python.

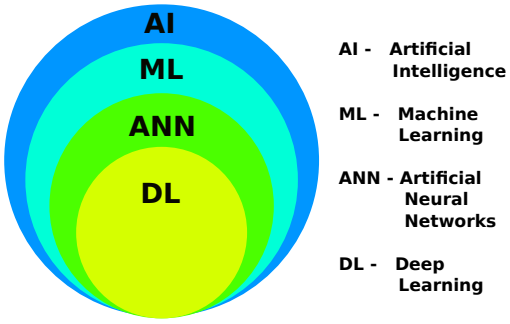
Trénování modelů neuronové sítě může obecně probíhat za pomocí procesoru (CPU), grafické karty (GPU) nebo specializovaného čipu TPU, který je určen konkrétně pro trénování neuronových sítí. Porovnáme-li CPU a GPU, výrazně rychlejší je GPU, jelikož umožňuje paralelní výpočty. TPU je určeno pro trénink s velkým množstvím dat a ve většině případů je rychlejší než GPU. Podle společnosti Google, která tuto jednotku vyvinula, může být 15 až 30 krát rychlejší než standartní GPU [1].

Kapitola 2

Úvod do umělé inteligence

Pojem umělé inteligence (AI) je lidstvu znám zhruba již od 50. let 20. století, avšak teprve proces digitalizace v posledních desetiletích ji umožnil skutečný rozkvět. Digitalizace totiž exponenciálně zvýšila množství dat, které člověk produkuje a právě velké množství dat je jedním z klíčových prvků úspěšného trénování AI. Dnes najdeme nějakou formu umělé inteligence ve většině technologických firem. Jako příklad těch známějších můžeme uvést Google a jeho vyhledávač (search engine), Facebook a jeho cílené reklamy nebo YouTube, které využívá umělou inteligenci k doporučování nejvíce relevantních videí. Mezi další častá využití AI patří detekce objektů, zpracování přirozeného jazyka, expertní systémy, analýza statistických dat aj.

Umělá inteligence je velmi obšírný obor, proto ji dělíme do několika hlavních kategorií. Patří tam **strojové učení** (machine learning), **umělé neuronové sítě** (artificial neural networks) a **hluboké učení** (deep learning). Vztahy mezi těmito kategoriemi dobře znázorňuje obrázek 2.1. Z hlediska nutnosti poskytnutí očekávaných výsledků do systému můžeme výše uvedené kategorie dále dělit na učení s učitelem (supervized learning), bez učitele (unsupervized learning) a zpětnovazebné učení (reinforcement learning).



Obrázek 2.1: Vztahy mezi kategoriemi AI

2.1 Strojové učení

Hlavní rozdíl mezi metodami hlubokého a strojového učení je především v tom, jakým způsobem se systém učí a jaký je typ vstupních dat [2]. Strojové učení obecně funguje na principu hledání korelace mezi vstupními daty. Jednoduchým příkladem pro nalezení přibližné korelace mezi daty může být lineární regrese. Systém se v tomto případě učí přímo ze vstupních dat tak, že určí parametry regresní přímky (podle známého matematického vzorce). Když má pak z nových dat vytvořit predikci, jednoduše tato data dosadí do parametrů regresní přímky a určí funkční hodnotu-predikci. V praxi se používají metody regrese polynomy vyšších řádů. V závislosti na požadovaném výstupu můžeme použít tyto techniky strojového učení: regresní, klasifikační a clusteringové techniky [3].

2.1.1 Supervized learning

Učení s učitelem již podle názvu napovídá, že systému poskytneme kromě vstupních dat i učitele ve formě očekávaných výstupních dat (labeled data). Princip učení s učitelem lze popsat pomocí jednoduchého matematického zápisu. Jsou-li naše labeled data ve tvaru $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$, kde y je očekávaná výstupní hodnota, která je vždy funkcí vstupní hodnoty x, tedy $y = f(x)$, pak je cílem určit co nejpřesnější aproximaci této funkce $f(x)$.

Odchylku mezi aproximací $(\hat{y})$ a očekávanou hodnotou (y) udává účelová funkce (loss function). Existuje několik typů loss function, mezi nejznámější patří *absolute value loss* $L(y, \hat{y}) = |y - \hat{y}|$, *squared error loss* $L(y, \hat{y}) = (y - \hat{y})^2$. Pro systémy, kde výstupní hodnota nabývá pouze dvou stavů (1 a 0) se používá loss function, která je vždy rovna 0 nebo 1 podle toho, jestli se shoduje odhadnutá s očekávanou hodnotou $(y = \hat{y})$.

Může-li výstupní veličina y nabývat pouze hodnot z určité konečné množiny, jedná se o problém **klasifikace**. V druhém případě, kdy y může nabývat nekonečného množství číselných hodnot, jedná se o problém **regrese**. [4]

Nejčastěji zmiňovaným příkladem supervized learning jsou **neuronové sítě**, ty budou popsány v podkapitole hlubokého učení (2.2)

2.1.1.1 Lineární regrese

Jedná se o jeden z nejjednodušších algoritmů učení s učitelem. Vztah mezi vektorem vstupních a výstupních dat se snažíme popsat rovnicí přímky, tedy $y(x) = ax + b$. Cílem lineární regrese je určení parametrů přímky a a b například pomocí metody nejmenších čtverců tak, aby vzdálenost přímky od každého bodu byla co nejmenší. Součet čtverců pro danou aproximaci se spočítá pomocí následujícího vzorce

$$S(a,b) = \sum_{i=1}^{n} [f(x_i) - y_i]^2 = \sum_{i=1}^{n} [ax_i + b - y_i]^2 \tag{2.1}$$

kde:
$S(a,b)$ = součet čtverců (chybová funkce)
$[x_i, y_i]$ = souřadnice bodů vstupních dat
n = počet bodů vstupních dat

Chybová funkce je vlastně kvadratickou funkcí dvou proměnných a a b, její minimum tedy najdeme, položíme-li její parciální derivace rovno 0 (víme že se jedná o kladnou kvadratickou funkci, její parciální derivace bude tedy nulová pouze v minimu).

$$\begin{aligned} 0 &= \frac{\partial S(a,b)}{\partial a} = 2\sum_{i=1}^{n}(ax_i + b - y_i)x_i \\ 0 &= \frac{\partial S(a,b)}{\partial b} = 2\sum_{i=1}^{n}(ax_i + b - y_i) \end{aligned} \tag{2.2}$$

Po dalších úpravách získáme vztah pro výpočet parametrů a a b

$$\begin{aligned} a &= \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \\ b &= \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2} \end{aligned} \tag{2.3}$$

Dosazením souřadnic jednotlivých vstupních bodů do těchto rovnic získáme nejlepší lineární aproximaci vstupních dat. [4]

2.1.1.2 K-nejbližších sousedů

Jedná se o klasifikační algoritmus, který provádí klasifikaci podle toho, jak jsou klasifikovány jeho nejbližší datové body. Je-li vektor vstupních dat $x = (x_1, x_2)$ pak odpovídající datový bod má souřadnice $[x_1, x_2]$. Jako příklad můžeme uvést klasifikaci černobílých obrázků, kde každý datový bod odpovídá jednomu obrázku. Souřadnice tohoto datového bodu pak odpovídají číselným hodnotám jednotlivých pixelů (např. hodnoty 0-255). Počet souřadnic datového bodu se tedy bude rovnat počtu pixelů v obrázku. Vzdálenost mezi jednotlivými body se určuje jako euklidovská vzdálenost bodů.

Tedy

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.4}$$

kde:

d = vzdálenost mezi body

x_i, y_i = body v prostoru se souřadnicemi $(x_1, x_2...x_n), (y_1, y_2...y_n)$

n - rozměr vektorového prostoru

Pokud bychom chtěli klasifikovat barevné obrázky, můžeme použít stejný vzorec avšak s drobnými úpravami, jelikož pixel již nebude skalární veličina, ale vektor o třech složkách (pracujeme-li s barevným modelem RGB).

$$d = \sqrt{\sum_{i=1}^{n}(x_i^r - y_i^r)^2 + (x_i^g - y_i^g)^2 + (x_i^b - y_i^b)^2} \tag{2.5}$$

Pro určení K nejbližších sousedních datových bodů je třeba spočítat vzdálenost od každého jednoho datového bodu a určit K bodů s nejmenší vzdáleností. Parametr K je volen uživatelem, zpravidla podle počtu vstupních dat. U těchto K nejbližších bodů se zjišťuje do jaké kategorie jsou zařazeny. Klasifikace požadovaného datového bodu se pak určí podle nejčastěji se vyskytující kategorie. Jelikož může dojít k situaci, kdy počet výskytů dvou různých kategorií, je stejný je vhodné volit parametr K tak, aby se jednalo o liché číslo.

Výhodou této metody je, že systém není třeba nijak trénovat, pouze mu poskytnout data již klasifikovaných datových bodů (označené obrázky). Nevýhoda je poměrně velká výpočetní náročnost, způsobena nutností počítat vzdálenost od všech datových bodů. Metoda tedy není příliš vhodná pro velké datové sady. [5]

Mezi další významné zástupce učení s učitelem patří **metoda podpůrných vektorů** (support vector machines), **logistická regrese**, **naivní Bayesovy klasifikátory** (naive Bayes) a jiné.

2.1.2 Unsupervized learning

Unsupervized learning neboli učení bez učitele je typ umělé inteligence, která analyzuje a shlukuje neoznačená data [6]. Systému tedy neposkytujeme žádnou sadu očekávaných hodnot, pouze vstupní hodnoty, mezi nimiž se pak hledají skryté vzory (korelace). Často používanou metodou pro shlukování dat je shlukování metodou nejbližších středů (k-means clustering) [7].

2.1.2.1 K-means clustering

Jedná se o algoritmus, který rozděluje vstupní data (datové body) do K různých skupin (clusterů). Datové body a jejich souřadnice jsou zde chápány stejně jako v kapitole 2.1.1.2. V prvním kroku jsou náhodně zvoleny souřadnice tzv. středových bodů, jejichž počet je stejný jako parametr K. Tyto středové body představují počáteční body jednotlivých clusterů, ke kterým se v dalších krocích budou přiřazovat datové body. Každý datový bod je následně přiřazen k nejbližšímu středovému bodu (clusteru). Nejbližší středový bod se počítá euklidovskou vzdáleností bodů, viz 2.4 (Není to ale pravidlem, lze použít i jinou než euklidovskou vzdálenost). V následujících krocích se mění poloha středového bodu clusteru tak, aby jeho souřadnice byly aritmetickým průměrem souřadnic bodů, které do daného clusteru patří. Grafické znázornění tohoto algoritmu je na obrázku 2.2 Vztah pro výpočet nové polohy středového bodu je následující

$$\bar{x}_k = \frac{1}{n_k} \sum_{i \in C_k} x_i \tag{2.6}$$

kde:
$\bar{x_k}$ = nová poloha středového bodu
C_k = označení konkrétního clusteru
n_k = počet bodů v daném clusteru
x = bod v clusteru

Po změně polohy středových bodů podle výše uvedeného vzorce je nutno znovu přepočítat vzdálenosti jednotlivých bodů a najít pro ně nové nejbližší clustery. Proces přemisťování středových bodů se opakuje do té doby, dokud dochází k jejich pohybu. Tuto podmínku pro běh algoritmu můžeme matematicky zapsat následovně [7]

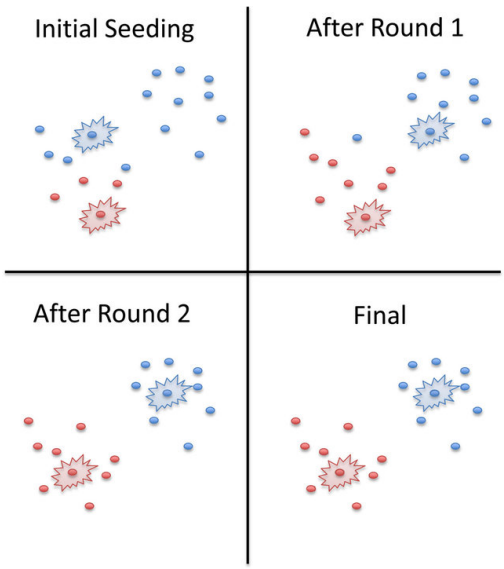
$$\arg\max_{C} \sum_{i=1}^{n} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{2.7}$$

kde:
n = počet datových bodů
C_i = označení i-tého clusteru
x = datový bod

μ_i - střed clusteru C_i

Protože je tento algoritmus velmi citlivý na počáteční polohu středových bodů, je nutné jej provést vícekrát s různými počátečními polohami středových bodů a porovnat výsledky každého běhu. Výsledky se většinou porovnávají pomocí variace příslušnosti datových bodů (jsou-li datové body rozmístěny rovnoměrně do jednotlivých clusterů). [8]

Jelikož ve většině případů nevíme dopředu do kolika clusterů mají být data rozdělena, tedy neznáme parametr K, existuje několik metod, které nám tento výběr usnadní. Nejsnadnější možností je použít "rule of thumb"a parametr K určit jako $K = \sqrt{\frac{n}{2}}$, kde n je počet datových bodů. O něco komplexnější metodou k určení parametru K je tzv. "elbow method". Jedná se o grafickou metodu, kdy se hledá zlomový bod (neboli elbow) v průběhu funkce, která popisuje závislost mezi počtem zvolených clusterů K a velikostí účelové funkce (která se počítá jako squared error loss). [7]

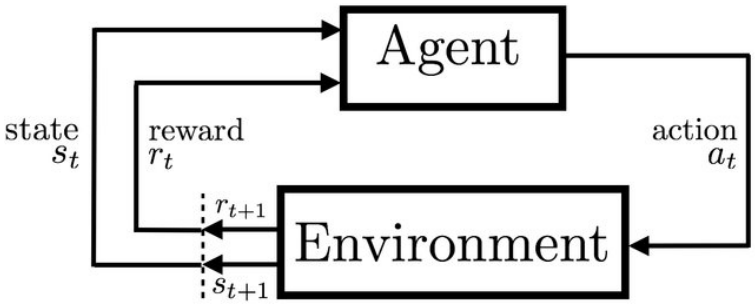


Obrázek 2.2: Grafické znázornění K-means algoritmu [9]

2.1.3 Reinforced learning

Jedná se o techniku strojového učení využívající principu odměn a trestů k dosažení požadovaného cíle. Prostředkem této techniky je tzv. agent, který interaguje s okolním prostředím. Cílem agenta je provádět takové akce, které mu z dlouhodobého hlediska přinesou maximální hodnotu odměny. Na rozdíl od učení s učitelem, agent zde nezná dopředu akce, které by měl provést aby dospěl k požadovanému cíli. Agent se musí tyto akce naučit metodou pokus-omyl.

Základními elementy reinforced learning (RL) jsou: *agent*, *prostředí* (environment), *odměna* (reward), *politika akcí* (policy) a *hodnotová funkce* (value function)[10]. **Agent** svými akcemi působí na prostředí a tím ovlivňuje jeho stav. Při každé akci agenta, je mu prostředím poskytnuta **odměna**, jejíž velikost závisí na aktuálním stavu prostředí a na akci kterou agent vykonal. Odměna tedy definuje, které akce agenta jsou "dobré"a "špatné". Agent může velikost odměny ovlivnit přímo, pomocí své akce, nebo nepřímo kdy svou akcí změní stav prostředí. Grafické znázornění interakce mezi agentem a prostředím je na obrázku 2.3.



Obrázek 2.3: Interakce mezi agentem a prostředím [11]

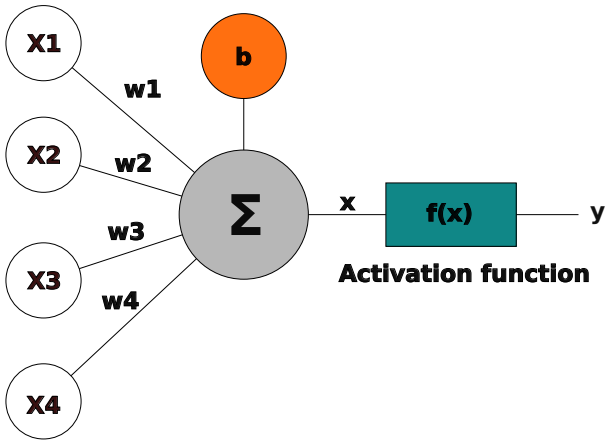
Hodnotová funkce (VF) na rozdíl od odměny udává co je pro agenta dobré z dlouhodobého hlediska. Váže se na jednotlivé stavy prostředí a udává jak velkou odměnu může agent očekávat v budoucnosti. VF určuje vhodnost daného stavu tím, že předpovídá stavy které mohou po aktuálním stavu následovat a velikost odměn, které jsou v těchto stavech dostupné. **Politika akcí** definuje, jakou akci by měl agent vykonat při daném stavu prostředí. Politika se na základě velikosti získané odměny mění. Je-li získaná odměna na danou akci malá, politika se může změnit tak, aby při stejné situaci v budoucnosti vykonal agent jinou akci. Politika tedy vlastně reprezentuje předchozí zkušenosti agenta.

Jednou z hlavních výzev RL je nalezení kompromisu mezi objevováním (exploration) a využíváním (exploitation). Pro získání co největší odměny musí agent provádět akce, které už v minulosti vyzkoušel a o kterých ví, že mu zajistí velkou odměnu. Avšak aby tyto akce objevil, musí vyzkoušet i jiné akce, které ještě neprovedl. Pro řešení tohoto dilematu existuje množství strategií, které bývají souhrnně označovány jako strategie problému vícerukého bandity (Multi-armed bandit). [10]

Pro popis problémů RL se velmi často používají **Markovy rozhodovací procesy**.[11]

2.2 Hluboké učení

Hluboké učení napodobuje činnost lidského mozku, kdy se používá síť neuronů, které jsou navzájem různými způsoby propojeny. Neurony jsou shlukovány do vrstev. Existují 3 druhy vrstev - vstupní, skryté (hidden layer) a výstupní. Vstupní vrstva slouží ke vkládání dat do sítě, výstupní nám poskytuje informace o výsledku (tedy o predikci). Ve skryté vrstvě dochází k nelineární transformaci vstupních dat pomocí weights, biases a aktivačních funkcí [12]. Weight je skutečně "váha"propojení mezi neurony, touto hodnotou se násobí signál z prvního neuronu než je poslán do druhého. Hodnota weight je tedy přiřazena každému spoji mezi neurony. Hodnota bias je přiřazena každému neuronu, tato hodnota se přičítá k signálu z předešlého neuronu. Bias vlastně udává citlivost neuronu na vstupní signál. Aktivační funkce zajišťuje výše zmíněnou nelineární transformaci pomocí různých typů nelineárních funkcí. Jako vstupní parametr pro tyto funkce slouží signál, na který již byl aplikován bias a weight. Jejím výstupem je "síla"aktivace daného neuronu. Na rozdíl od neuronů v mozku, kterými jsou tyto sítě inspirovány, mohou neurony v umělých neuronových sítích nabývat různé intenzity jejich aktivace. Biologické neurony mohou nabývat pouze stavu aktivovaný či neaktivovaný. Mezi aktivační funkce patří například rectified linear unit (ReLU), sigmoid, SoftMax [13]. Na Obrázku 2.4 můžeme vidět grafickou reprezentaci základních prvků neuronové sítě - neurony, weights, biases a aktivační funkci.



Obrázek 2.4: Názorné schéma parametrů v neuronové síti

Po přivedení dat na vstup neuronové sítě je započat proces dopředné propagace (forward propagation). Při tomto procesu jsou vstupní data předávána z jedné vrstvy neuronů do další, mezi každou vrstvou jsou transformována hodnotami parametrů weights a biases a přivedena na vstup aktivačních funkcí. Výstup aktivačních funkcí, udávající míru aktivace neuronu je poté předán do další vrstvy. Učení v tomto případě probíhá úpravou parametrů weight a bias tak, aby po průchodu vstupních dat celou sítí měla výsledná predikce co nejmenší odchylku od očekávané hodnoty. Nejznámějším algoritmem pro učení neuronových sítí je zpětná propagace (backpropagation).

Pojmem hluboké učení se rozumí jakákoli neuronová síť, která má více skrytých vrstev (slovo hluboké odkazuje na hloubku skryté vrstvy)[14]. Je zřejmé, že čím více má neuronová síť vrstev, tím více parametrů je potřeba upravit což znamená větší výpočetní náročnost. V této bakalářské práci bude použita právě metoda hlubokého učení.

Motivace k používání hlubokého učení a umělé inteligence jako takové je zřejmá již z jejího názvu. Jedná se o mimiku inteligence, tedy něčeho co je vlastní pouze živým organismům počítačem. Toto napodobování inteligence spočívá především ve schopnosti činit optimální rozhodnutí na základě **předchozích zkušeností** a vstupních dat. U umělé inteligence se často zmiňuje jako výhoda nepotřeba explicitního programování, což je důsledkem právě této schopnosti učení ze zkušeností. Konkrétněji to znamená, že nemusíme přímo zadávat parametry modelu AI, ale tyto parametry se pomocí procesu zpětné propagace (popsáno v následující kapitole) samy upravují.

2.2.1 Zpětná propagace

Zpětná propagace je proces díky kterému se neuronové sítě učí. Učení probíhá formou úpravy parametrů sítě (tedy weights a biases). Aby bylo možné parametry upravovat, je nutné znát odchylku predikované hodnoty od očekávané. Ta se počítá například pomocí metody zbytkového součtu čtverců (sum of squared residuals) nebo pomocí složitějších metod, jako je křížová entropie (cross-entropy). Tato odchylka se často označuje jako účelová funkce (*cost function*, též *loss function*). Cílem zpětné propagace je tedy úprava parametrů sítě tak, aby hodnota účelové funkce byla minimální.

Jako první je třeba provést dopřednou propagaci (forward propagation), abychom získali predikované hodnoty. Dopředná propagace je proces, kdy jsou vstupní data přivedena na vstup neuronové sítě a následně putují dál do skryté vrstvy (hidden layer), kde jsou transformovány pomocí weights, biases a aktivačních funkcí. Ze skryté vrstvy jsou data dále propagována do vrstvy výstupní, kde již získáme predikované hodnoty. Při první dopředné propagaci jsou hodnoty parametrů zvoleny náhodně, protože ještě neproběhl proces zpětné propagace, který by parametry upravil. [15]

Pomocí odhadnutých dat můžeme spočítat hodnotu účelové funkce například pomocí metody zbytkového součtu čtverců.

$$C = (y - \hat{y})^2 \tag{2.8}$$

kde
y = očekávaná hodnota
$\hat{y}$ = odhadnutá hodnota

K výpočtu minima účelové funkce se často používá algoritmus zvaný *gradientní sestup (gradient descent).* Gradientní sestup pracuje s parciálními derivacemi účelové funkce podle jednotlivých parametrů neuronové sítě. Derivace podle parametru nám říká, jak moc je účelová funkce náchylná na změnu daného parametru. Pro vyjádření parciálních derivací podle parametrů ze všech neuronových vrstev se používá tzv. řetízkové pravidlo (chain rule), které udává jak počítat derivace složených

funkcí. Každá vrstva neuronů totiž představuje další funkci k derivování.[15] Příkladem může být výpočet gradientu podle parametru jednoho z weight parametrů:

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} \tag{2.9}$$

kde:
C = cost function
w_{jk}^{l} = parametr weight v l-té vrstvě spojující j-tý neuron a k-tý neuron v další vrstvě
z_{j}^{l} = výstupní hodnota j-tého neuronu v l-té vrstvě

2.2.2 Transfer learning

Při používání neuronových sítí pro řešení různých problémů se poměrně často stává, že problémy si jsou navzájem podobné. V takovém případě by bylo neefektivní vytvářet vždy novou neuronovou síť k řešení daného problému. **Transfer learning** je metoda, která nám umožňuje tuto situaci řešit přenesením parametrů *weights* z již vytrénované sítě do nové. Přenesením parametrů zajistíme, že nová síť bude umět rozpoznávat stejné rysy (features) jako předtrénovaná síť. Aby tato metoda dobře fungovala, musí být rysy z původní sítě obecné a použitelné i v novém problému [16].

Transfer learning je zvlášť užitečný, když nemáme k dispozici dostatek dat pro dostatečně přesný trénink sítě.

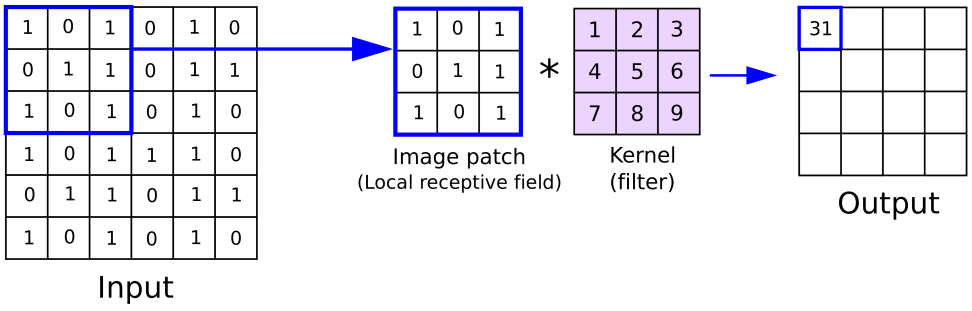
V této práci je transfer learning využit k vytvoření modelu pro klasifikaci puků. Jakou zdrojový model je využit MobilenetV2 a MobilenetV3Small. Tyto modely jsou specificky určeny k detekci rysů v obrázcích a jsou schopny klasifikovat do jednoho tisíce různých kategorií. Ze zdrojového modelu tedy získáme přehled o rysech, které se na obrázku nacházejí (například se může jednat o základní tvary, hrany, zaoblení, přechody barev atd.). Cílový model je pak možno získat trénováním zdrojové sítě na nových obrázcích. V mém případě má cílový model ještě navíc několik vrstev pro úpravu výstupu. Podrobnější informace o architektuře cílového modelu budou uvedeny v kapitole 7.1.1

Kapitola 3

Konvoluční neuronové sítě

Tento typ neuronových sítí je nejčastěji využíván pro zpracování obrazu. Oproti klasickým neuronovým sítím obsahuje několik vrstev neuronů navíc. Jedná se o konvoluční, a pooling vrstvu. U klasifikace či detekce z obrázků se tento typ sítě vyznačuje tím, že v hlubokých vrstvách neuronů se postupně identifikují abstraktní rysy zkoumaného objektu [3]. Souhrnně se tyto vrstvy pro identifikaci rysů nazývají *feature extractors* [17].

V běžné neuronové síti bychom pro klasifikaci obrázku použili vstupní vrstvu neuronů o velikosti $M \cdot N$ neuronů. (Kde M a N jsou rozměry obrázku). Pro obrázek o velikosti 320x320 pixelů bychom tedy potřebovali 102400 vstupních neuronů a ještě větší množství weights pro propojení s další vrstvou. Aby se zredukoval počet vstupních neuronů a dalších parametrů, nahlíží se u CNN na obrázek postupně, po částech. Na každou část se aplikuje filtr, který slouží právě k detekci abstraktních rysů. Filtr je matice pixelů (např. o rozměru 3x3 pixely), jejichž hodnoty jsou nejprve náhodné, následně jsou upraveny pomocí zpětné propagace tak, aby odpovídaly tvaru některého z abstraktních rysů [18].



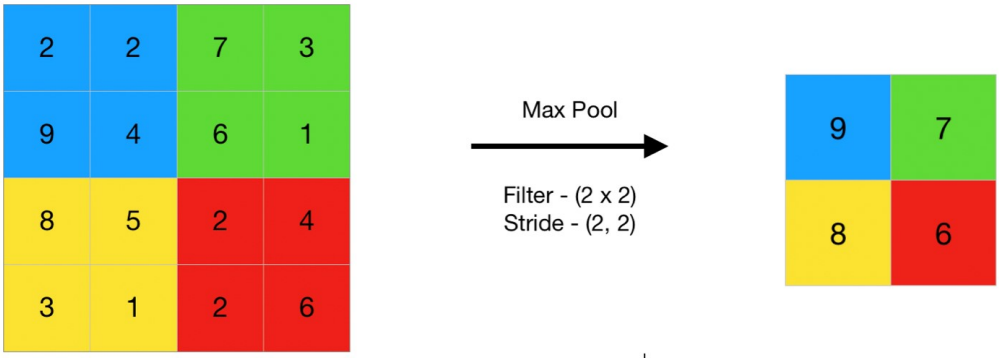
Obrázek 3.1: Proces filtrování [19]

Filtrem se překryje část obrázku a spočítá se skalární součin hodnot překrývajících se pixelů. Výsledek této kalkulace se zapíše do další matice, zvané *feature map* a pokračuje se s překrýváním dalších částí obrázku až se získá skalární součet ze všech částí [18]. Feature map tedy představuje matici, která udává jak moc se daná část obrázku podobá abstraktnímu rysu. Prováděním skalárního součinu části obrázku a filtru provádíme operaci zvanou konvoluce (viz. kapitola 3.1), odtud název konvoluční neuronové sítě. V síti může být více filtrů, aby bylo možné detekovat různé typy rysů [3]. Proces filtrování je znázorněn na obrázku 3.1. Velikost výsledné feature map se dá určit podle vztahu 3.1,

$$O = 1 + \frac{N - F}{S} \tag{3.1}$$

kde **N** je rozměr obrázku (předpokládá se čtvercový tvar), **F** je rozměr filtru a **S** (stride) je velikost kroku, kterým se filtr přesouvá po obrázku [3].

Feature map je následně přivedena na vstup aktivační funkce, nejčastěji ReLU [3]. Výstup z aktivační funkce je dále předán do pooling vrstvy. Tato vrstva provede další zmenšení vstupních dat (tedy feature map matice), aby se zredukoval počet parametrů, které se budou dále do sítě předávat a aby se celkově snížila výpočetní náročnost [20]. Pooling vrstva zároveň sumarizuje výskyt rysů ve feature map. Díky sumarizaci je výsledný model méně náchylný na změny pozice rysů na obrázku [21][20]. Existuje celá řada metod poolingu avšak nejznámější jsou max pooling a average pooling [21]. Max pooling vybere z oblasti, kterou překrývá filtr pouze tu nejvyšší hodnotu, kdežto average pooling spočítá aritmetický průměr všech hodnot v dané oblasti. Výsledná hodnota je pak zapsána do nové feature map. Proces max poolingu je zobrazen na obrázku 3.2.

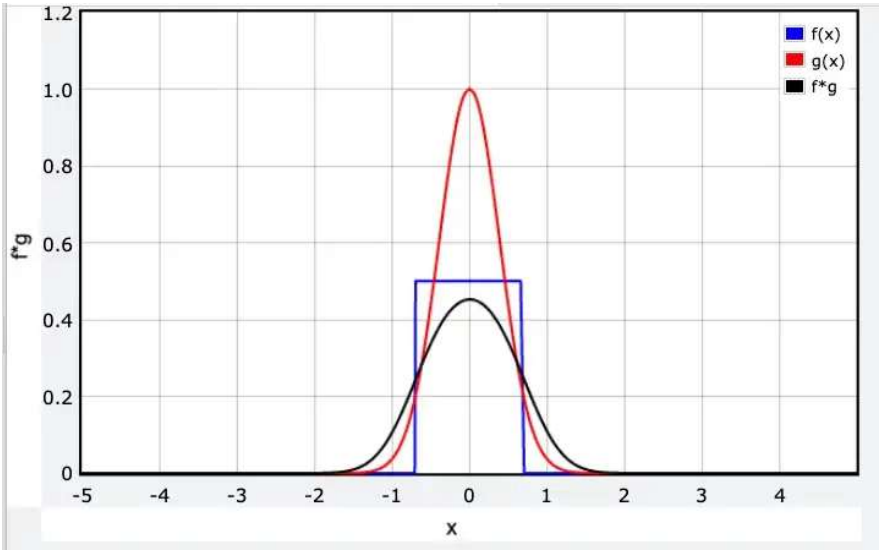


Obrázek 3.2: Proces max-poolingu [20]

Předposlední vrstvou v CNN je plně propojená vrstva. Tato vrstva se běžně vyskytuje i u klasických neuronových sítí. Jak vypovídá její název všechny neurony z předchozí vrstvy jsou zde propojeny se všemi neurony z vrstvy následující. Právě v této části CNN vzniká nejvíce propojení mezi neurony a tedy i nejvíce parametrů, které se síť musí učit [3]. Kvůli této části sítě se v předchozích vrstvách redukovalo množství dat pomocí již zmíněných metod. Poslední vrstvou v síti je samozřejmě vrstva výstupní.

3.1 Konvoluce

Konvoluce je matematická operace nad dvěma funkcemi. Často je znázorňována graficky pomocí překryvu funkcí. Máme-li funkce $f(x)$ a $g(x)$, pak konvoluce těchto dvou funkcí vyjadřuje, do jaké míry funkce $g(x)$ překrývá funkci $f(x)$ v závislosti na posunu funkce $g(x)$ [22]. Funkce, se kterou je prováděn posun se nazývá *konvoluční jádro*. Konvoluce je zobrazena na obrázku 3.3. Červená funkce $g(x)$ je konvoluční jádro a je posouvána přes modrou funkci $f(x)$. Černá funkce, která je výsledkem konvoluce, vyjadřuje velikost společné plochy funkcí v závislosti na posunu konvolučního jádra. Na obrázku je zachycen moment, kdy je konvoluční jádro posunuto tak, aby společná plocha funkcí byla maximální.



Obrázek 3.3: Konvoluce [23]

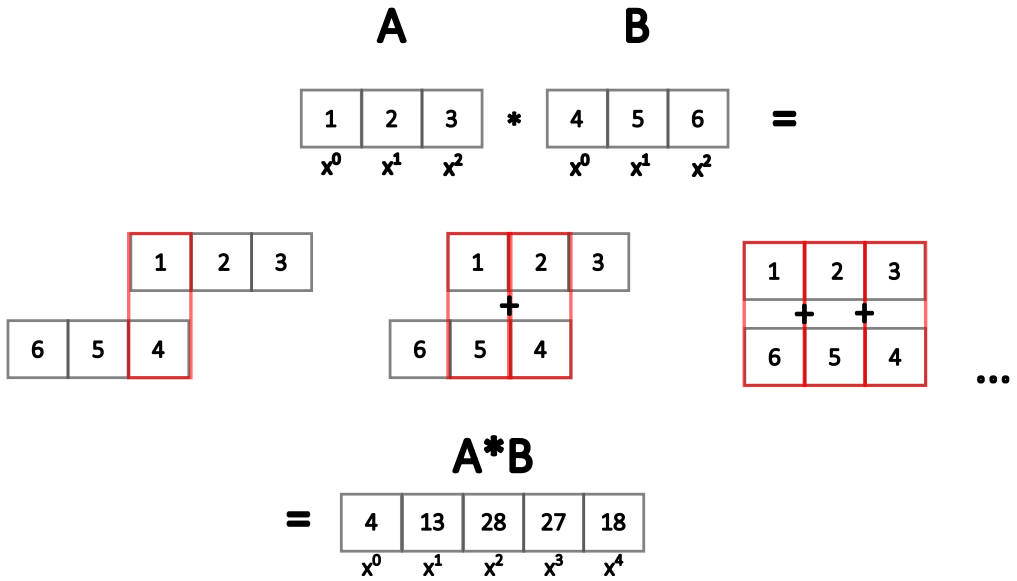
Matematicky je operace konvoluce definována tzv. konvolučním integrálem -

$$f(t) * g(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{3.2}$$

Jelikož je v tomto případě konvoluce funkce času (t), posun konvolučního $g(t)$ jádra zajistíme změnou parametru t.

Konvoluci je možné také definovat jako operaci nad polemi čísel. V takovém případě máme dvě pole čísel, jejichž jednotlivé prvky mezi sebou násobíme a následně sčítáme podle určitého pravidla. Toto pravidlo lze zjednodušeně popsat následovně: konvoluční jádro (jedno z polí) převrátíme a zarovnáme jeho poslední prvek s prvním prvkem druhého pole. Nyní je nad sebou zarovnána jedna dvojice prvků, tyto prvky vynásobíme. Výsledek násobení zapíšeme na začátek nového, výsledného pole. V následujících krocích je konvoluční jádro vždy posunuto o jeden prvek doprava. Prvky, které jsou nad sebou zarovnány vždy vynásobíme a sečteme. Výsledek přidáme do výsledného pole. Pro lepší pochopení je výše popsané pravidlo vizualizováno na obrázku 3.4. Na obrázku jsou zobrazeny první tři pozice konvolučního jádra, jeho posouvání však pokračuje až do bodu, kdy se překrývá pouze první prvek konvolučního jádra a poslední prvek druhého pole. Nejznámější příkladem použití konvoluce nad polemi hodnot je při násobení dvou polynomů. Provedeme-li konvoluci s koeficienty jednotlivých členů polynomu, výsledkem bude pole koeficientů odpovídající výslednému polynomu. [24]

$$(1 + 2x + 3x^2)(4 + 5x + 6x^2) = 4 + 13x + 28x^2 + 27x^3 + 18x^4 \tag{3.3}$$



Obrázek 3.4: Konvoluce s polemi čísel

3.1.1 Použití konvoluce ve zpracování obrazu

Ve zpracování obrazu je konvoluce používána k úpravě obrázků nebo k detekci rysů v něm obsažených. Zde je lepší konvoluci chápat právě jako operaci nad poli čísel (popsáno v předchozí kapitole). Konvolučnímu jádru se zde říká *filtr* a je reprezentováno dvourozměrným číselným polem. Filtr má typicky rozměr 3x3, ale používají se i filtry o rozměrech 2x2, 4x4 nebo 5x5 [25]. Filtrem se postupně překrývají části obrázku a hodnoty překrývajících se prvků se násobí a sčítají. Konvolucí filtru a obrázku můžeme docílit mnoha efektů jako jsou: vyhlazování, doostřování, detekce hran a rysů [26]. Jaký výsledný efekt nám konvoluce přinese, ovlivníme změnou hodnot ve filtru.

Jelikož jsou obrázky dvourozměrné a jejich pixely mají jasně definovanou velikost, nahradíme vzorec pro konvoluci jeho diskrétní verzí s dvěma operátory součtu:

$$(f * g)(x, y) = \sum_{\tau_1=0}^{i_a-1} \sum_{\tau_2=0}^{j_a-1} f(\tau_1, \tau_2) g(i - \tau_1, j - \tau_2) \tag{3.4}$$

Mezi známé filtry patří například tyto:

Detekce vertikálních hran

$$g(x, y) = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \tag{3.5}$$

Tento filtr je sestaven tak, aby při překrytí části obrázku, kde se vyskytuje pouze jedna barva byl výsledek konvoluce nulový. V částech obrázku, kde dochází ke změně barev (výskyt hrany) při pohybu zleva doprava, je výsledek konvoluce nenulové číslo. Jeho velikost závisí na odlišnosti barev pixelů, které filtr překrývá.

Detekce horizontálních hran

$$g(x, y) = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & -2 & -1 \end{pmatrix} \tag{3.6}$$

Stejný princip jako u předchozího filtru, pouze je prováděn pohyb shora dolů.

Rozostření

$$g(x,y) = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \tag{3.7}$$

Jelikož jsou u tohoto filtru všechny prvky stejné, výsledkem konvoluce je průměrná hodnota všech prvků, které zrovna filtr překrýval, vynásobená hodnotou filtru (v tomto případě $\frac{1}{9}$). Tím je vytvořen efekt rozostření obrázku. Tento efekt je zobrazen na obrázku 3.5

Obrázek 3.5: Rozostření obrázku pomocí konvoluce [27]

Kapitola 4

Analýza možností využití metod umělé inteligence v řídicích systémech

K rozvoji umělé inteligence dochází samozřejmě i v oblasti průmyslu. Cílem jejího využití je především zvýšení produktivity a efektivity výroby při zachování optimální výše nákladů [28] a omezení či úplné odstranění nutnosti zásahu člověka. V případech, kdy není možné nebo žádoucí odstranění lidského faktoru, může AI sloužit k usnadnění jeho práce (např. ovládání zařízení pomocí hlasu či gest, použití kolaborativních robotů). Zásadní způsob, jakým AI zvyšuje efektivitu výrobních procesů je analýza velkého množství dat, které tyto procesy generují. Významným příkladem aplikace, která využívá analýzu různých procesních dat je **prediktivní údržba, detekce abnormalit a chybových stavů**. Dalším příkladem užití AI v průmyslu, kterého bude využito i v této práci je vizuální inspekce nebo obecně **strojové vidění**.

4.1 Prediktivní údržba

Údržbu chápeme jako prostředek pro zajištění spolehlivosti a provozuschopnosti zařízení. Jinými slovy se jedná o proces, který odstraňuje následky opotřebení a tím prodlužuje životnost zařízení a zároveň zabezpečuje spolehlivost a bezpečnost provozu. V dnešní době je údržba nedílnou součástí téměř každého výrobního procesu. V některých případech je údržba brána jako rovnocenný proces k samotné výrobě (systém totálně produktivní údržby). [29] Podle vztahu ke skutečnému stavu zařízení můžeme údržbu dělit do těchto tří skupin -

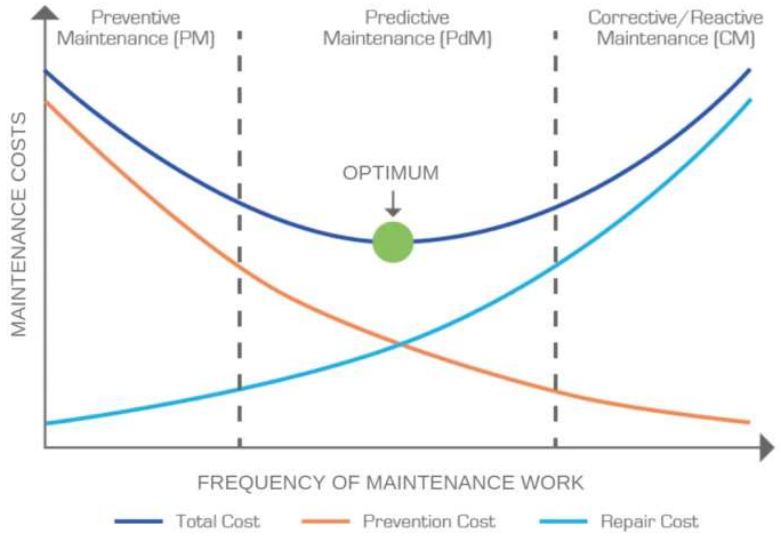
- Údržba po poruše
- Preventivní údržba
- Prediktivní údržba

Grafické znázornění nákladovosti jednotlivých typů údržby je zobrazeno na obrázku 4.1

Údržba po poruše je nejméně efektivní a nejnákladnější typ údržby. Proces údržby je zahájen až když dojde k selhání zařízení. Nedochází zde sice k vynaložení žádných nákladů dokud se zařízení neporouchá, ale náklady při poruše jsou velice vysoké. Do nákladů se promítají faktory jako nutnost mít k dispozici všechny možné náhradní součástky nebo rovnou celé náhradní stroje, dlouhý čas nutný k provedení údržby, cena pracovní síly při práci přes čas. Podniků, které tento způsob údržby využívají není dnes mnoho a i ty, které takto fungují provádí alespoň základní preventivní údržbu formou např. mazání pohyblivých součástí.[30]

Preventivní údržba - zde se údržba provádí pravidelně v předem naplánovaných časových intervalech. Tyto časové intervaly se často určují podle středního času mezi poruchami (MTTF - mean time to failure)[14], který udává dobu provozu do poruchy [31]. Nevýhodou tohoto způsobu údržby je, že se provádí nezávisle na skutečném stavu zařízení, tudíž může být mnohdy zbytečná.

Prediktivní údržba Jedná se o nejefektivnější způsob údržby, jelikož závisí na skutečném stavu zařízení. Stav zařízení je vyhodnocován v reálném čase pomocí různých nedestruktivních technik diagnostiky jako je monitoring vibrací, termografie, tribologie nebo vizuální inspekce [30]. Prediktivní údržbu můžeme dále rozdělit na "data-driven"a "model-based"[32]. Model-based přístup vyžaduje porozumění fyzikálních dějů u zařízení a jejich analytický popis. Data-driven přístup, kde se může využívat právě umělé inteligence vychází z historických dat naměřených na zařízení. Z naměřených dat je pak vytvořen model chování systému při různých podmínkách. V závislosti na typu dostupných procesních dat můžeme data-driven přístup dále dělit na supervised a unsupervised. Supervized můžeme použít tehdy, je-li součásti procesních dat i informace o tom, kdy došlo u zařízení k nějakému selhání či jiné degradaci jeho stavu. Existují-li pouze procesní data bez údajů selhání použijeme unsupervised přístup. Pokud je to možné, bývá upřednostňován supervised přístup.[32]



Obrázek 4.1: Nákladovost různých typů údržby [33]

Nasazení data-driven prediktivní údržby přináší celou řadu výhod, ale i překážek, které je třeba překonat. Mezi hlavní výhody můžeme zařadit zvýšení produktivity výroby, snížení množství neplánovaných odstávek, efektivní plánování údržby (aby k ní došlo když to stroj opravdu potřebuje) a rovněž efektivnější využívání finančních a lidských zdrojů. Podstatnou překážkou či výzvou je nutnost sběru velkého množství dat. Data mohou pocházet z různých senzorů napříč celým výrobním zařízením a mít různé fyzikální rozměry. K vytvoření dobrého prediktivního modelu je samozřejmě zapotřebí aby sesbíraná data byla přesná. Kromě sběru dostatečného množství dat je rovněž zapotřebí tyto data předzpracovat a vybrat pouze ty, které jsou pro danou aplikaci skutečně užitečná. V neposlední řadě je potřeba zvolit vhodný typ algoritmu strojového učení nebo v případě hlubokého učení a neuronových sítí optimálně zvolit hloubku a strukturu sítě. [34] Jako příklad používaných algoritmů strojového učení pro prediktivní analýzu můžeme uvést decision tree (DT), support vector machines (SVM) nebo K-nearest neighbors (K-NN). Trénování prediktivního modelu bývá výpočetně náročné, proto je většinou prováděno na dedikovaných serverech (např. v cloudu). Použití již vytrénovaného modelu je možné i na edge zařízeních, tedy například na modulu TM-NPU.

4.1.1 Přístupy prediktivní údržby

Z hlediska konkrétního cíle prediktivní údržby můžeme rozlišovat tři základní přístupy. Jedná se o detekci chybových stavů, predikce selhání zařízení a odhad zbývající životnosti.

Přístup detekce chybových stavů má za úkol pouze zjistit, zda-li je zařízení ve stavu poruchy či nesprávného fungování. Jedná se tedy o binární klasifikační problém, protože jsou rozeznávány pouze dva stavy "funkční"a "nefunkční". Při detekci chybového stavu je vhodné provést okamžitou údržbu, aby se předešlo úplnému selhání přístroje. U tohoto přístupu mohou být použity metody jak supervized tak unsupervized learning. Při užití supervized learning je u každého datového bodu specifikováno, jaký stav zařízení označuje (funkční, nefunkční). Příkladem vhodného typu algoritmu pro tuto situaci je Logistická regrese. U unsupervized learning, kde není nutné mít označené datové body je chybový stav detekován jako anomálie ve fungování zařízení. Zde je vhodným algoritmem například K-means clustering. V některých případech není dokonce ani nutné mít data z období, kdy k chybovému stavu došlo. Příkladem může být metoda NNPLC popsána v následující kapitole.

Predikce selhání zařízení je rovněž binární klasifikační problém, jehož cílem je předpovědět, zda-li v blízké budoucnosti dojde k selhání zařízení. Tento přístup je vhodný, jsou-li k dispozici historická data o selhání zařízení. Je vhodné aby předpověď o selhání zařízení byla dostupná v dostatečném časovém předstihu aby bylo možné naplánovat údržbu. Dostatečný časový předstih je možno zajistit vhodnou volbou tzv. *monitoring window* (mw), tedy doby pro kterou se předpovídá, zda-li v ní dojde k selhání. Volba *mw* může probíhat například tak, že vytvoříme několik modelů pro predikování, každý s různou délkou *mw*. Po vyhodnocení přesností modelů pak zvolíme *mw* nejpřesnějšího modelu. Data pro tento přístup se většinou skládají ze dvou částí: data ze sledování stavu zařízení (condition monitoring) a záznamy o selhání zařízení.

Odhad zbývající životnosti zařízení (RUL) je přístup, který na rozdíl od předchozích dvou vyžaduje regresní model (predikce je kontinuální hodnota). Tento přístup je vhodný například v situacích, kdy máme mnoho stejných, či podobných přístrojů, které byly uvedeny do provozu v přibližně stejný čas. U těchto přístrojů se dá předpokládat že k i poruše způsobené opotřebením dojde v přibližně stejnou dobu. Využitím RUL můžeme naplánovat postupnou údržbu všech těchto zařízení v dostatečném předstihu. Nevýhodou tohoto přístupu je, že může být použit pouze na přístroje, jejichž postupná degradace stavu je dobře zaznamenatelná. Typickým příkladem jsou přístroje vykonávající točivý pohyb, například elektromotor. Jeho degradace se dá určit pomocí vibrodiagnostiky. Některé přístroje nevykazují žádné známky degradace dokud u nich nedojde k poruše. V takovém případě je možné metodu RUL použít až když dojde k této poruše dojde. Po poruše pak dochází k postupné degradaci zařízení, kterou je možno detekovat.[35]

4.2 NNPLC

Jedná se o poměrně nový a zatím ne příliš rozšířený způsob použití neuronových sítí spolu s PLC. Autorem metody je *Bhargav Joshi*, níže uvedené informace jsou čerpány z jeho diplomové práce [36]. Základní myšlenka spočívá v paralelním využití PLC a neuronové sítě pro zpracování vstupů a následné porovnání výstupů. Na PLC i na neuronovou síť se dá pohlížet jako na systém, který z určitých vstupních dat vygeneruje data výstupní. Výstupy obou systému je pak možno porovnat a spočítat odchylku.

Nejdříve jsou shromážděna vstupní a výstupní data z PLC při optimálním běhu. Optimálním se rozumí takový stav, kdy vstupy do PLC nejsou ovlivněny žádnou poruchou např. porucha edge zařízení (senzory), porucha v komunikaci mezi edge zařízením a PLC. Je potřeba shromáždit takové množství dat, aby zahrnovala všechny situace, které mohou za optimálního běhu nastat. Tyto data jsou pak použita pro trénování neuronové sítě. Zde je zásadní výhoda oproti metodám prediktivní údržby. Ty vyžadují označená data zahrnující jak optimální, tak poruchový stav, aby se síť mohla naučit mezi těmito stavy rozeznávat. U metody NNPLC vystačí pouze data z optimálního běhu. Má to samozřejmě i svou nevýhodu a to takovou, že při výskytu nějaké chyby ji sice zaregistrujeme, ale nebudeme vědět o jakou konkrétní chybu se jedná (detekujeme pouze odchylku od optimálního stavu).

Použitím pouze dat z optimálního běhu je vlastně využíváno běžně nežádoucího jevu tzv. overfittingu. K tomuto jevu dochází, když AI model dosahuje velmi dobrých výsledků na datech se kterými je trénován (training data), ale již nedokáže dobře pracovat s novými daty (testing data). V tomto případě je model "overfitted"na data z optimálního běhu.

Sběr dat z PLC - správná data z běhu PLC jsou kritická pro fungování této metody, musí být tedy zajištěn jejich korektní sběr. Data můžou pocházet například z externích analogových a digitálních modulů připojených k PLC. Nejdůležitějším aspektem při jejich sběru je správné načasování. Je nutno znát délku jednoho PLC cyklu, tedy za jak dlouho načte PLC hodnoty ze vstupů, zpracuje svůj program a zapíše odpovídající hodnoty na své výstupy. Ideální by bylo sbírat vstupní i výstupní data na konci každého jednoho cyklu. Z různých důvodů ale můžeme chtít sbírat data méně často (např. kvůli omezené paměti vyhrazené pro tato data, snížení výpočetní náročnosti), pak je vhodné provádět sběr v celočíselných násobcích délky jednoho cyklu. Příliš málo častý sběr dat může ale způsobit, že nezaznamenáme některý z možných stavů, které mohou nastat. Pokud se v programu PLC využívají timery, měly by se zaznamenávat i jejich hodnoty, abychom zachytili časovou závislost ve vztahu mezi vstupem a výstupem.

Porovnávání výstupu PLC a NN - Při porovnávání výstupních hodnot z PLC a NN je důležité nezapomínat na to, že NN bude vždy vykazovat určitou míru chybovosti způsobené nedokonalým trénováním a tudíž způsobovat odchylku od výstupu z PLC. Je tedy nutné umět rozlišit mezi odchylkou způsobenou training errorem a odchylkou způsobenou chybovým stavem. Ve většině případů to lze poznat z velikosti dané odchylky, protože odchylka způsobena chybovým stavem bývá podstatně větší než ta způsobena nedokonalým trénováním sítě. Nelze na to ale spoléhat vždy, proto autor navrhuje použití například metody klouzavého průměru efektivní hodnoty (RMS) odchylky. V případě odchylky způsobené nedokonalým trénováním by měla hodnota průměrné RMS odchylky vždy konvergovat k hodnotě blízké training erroru. V případě odchylky způsobené chybovým stavem by měla průměrná chyba růst daleko nad hodnotu training erroru.

Metody nasazení NN - autor práce rozlišuje dvě základní metody nasazení NN. Jedná se o dynamicky a staticky nasazenou. U **dynamicky** nasazené běží neuronová síť paralelně s PLC v reálném čase. Hodnota odchylky mezi NN a PLC je tedy vyhodnocována v reálném čase, a na potenciální detekci chybového stavu lze okamžitě reagovat. Nevýhodou tohoto způsobu je nutnost bezchybně synchronizovat NN a PLC, aby oba systémy generovaly výstupy ve stejný čas.
U **statického** nasazení NN problém synchronizace zaniká. NN je zde po dobu několika PLC cyklů nečinná, pouze se sbírají data ze vstupů a výstupů PLC. Po uplynutí stanovené doby se posbíraná data předají NN k vyhodnocení. NN vyhodnotí odchylku pro každý PLC cyklus ze kterého byla data posbíraná, tyto odchylky jsou poté zprůměrovány a vyhodnoceny. Ještě před statickým nasazením NN do provozu se vyhodnotí odchylka při náhodně zvolených vstupních hodnotách. Tento krok je opakován pro několik různých kombinací vstupních hodnot. Vstupní data jsou sice volena náhodně, měla by ale být zvolena tak, aby výstupní odchylka byla vždy větší než training error. Výsledná hodnota odchylky je spočítána jako průměr odchylek pro všechny zvolené kombinace vstupních dat. Označíme-li tuto výslednou odchylku jako E_p, training error jako E_T a odchylku, kterou získáme po konci každého časového úseku, po který se sbírají data z PLC jako E_a, pak můžeme definovat následující pravidla pro detekci chybových stavů:

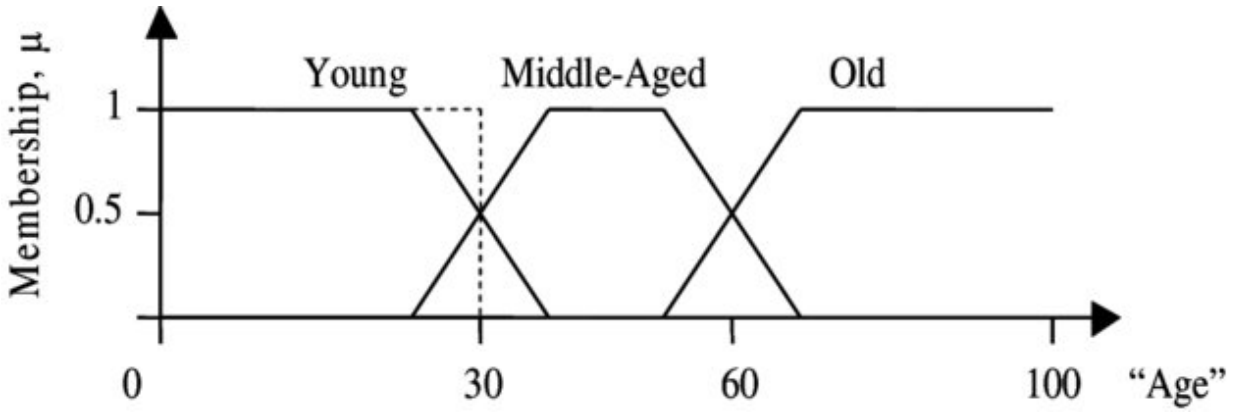
$E_a \leq E_T$ - běžný stav
$E_T \leq E_a \leq E_p$ - vysoká pravděpodobnost odchylky z NN
$E_a \geq E_p$ - vysoká pravděpodobnost abnormality v běhu systému

Tato metoda je vhodná pro obecnou detekci abnormalit v systému. Příkladem může být závada na edge zařízení nebo kybernetický útok na systém za účelem modifikace PLC kódu. Nevýhodou této metody je, že nedokáže detekovat konkrétní chybový stav, pouze že došlo k odchýlení od stavu běžného. Naopak výhoda spočívá v tom, že je zapotřebí pouze dat z optimálního běhu PLC k úspěšnému vytrénování modelu AI.

4.3 Fuzzy řízení

Za součást umělé inteligence je považována i Fuzzy logika. Termín fuzzy označuje věci či děje které jsou nejasné, neurčité, vágní. Výhodou fuzzy logiky je to, že nám umožňuje popsat nepřesné či nejasné informace běžným jazykem. Oproti booleanovské logice lépe vyjadřuje znalosti a uvažování člověka [37]. Na rozdíl od booleanovského systému logiky, kdy proměnná může nabývat pouze dvou stavů *1 (True)* a *0 (False)* ve fuzzy logice může proměnná nabývat více hodnot v intervalu (0,1).

Matematicky řečeno, fuzzy proměnná určuje stupeň příslušnosti prvku k množině, kde stupeň příslušnosti může nabývat hodnot z intervalu (0,1). Stupně příslušnosti k fuzzy množinám můžeme vizualizovat příkladem s věkem a věkovými skupinami (obrázek 4.2)



Obrázek 4.2: Fuzzy množiny [38]

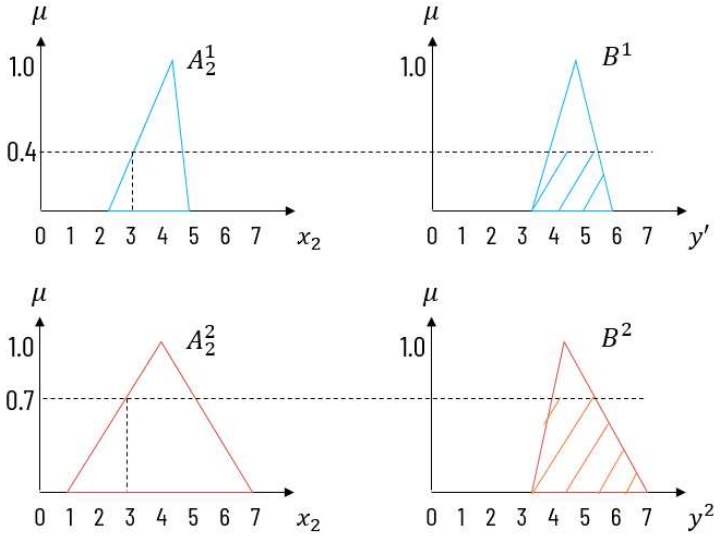
Z grafu je zřejmé, že člověk, který má do 20 let je se stoprocentní pravděpodobností členem množiny *mladý*. Jiný člověk, kterému je 35 let má již jen malý stupeň příslušnosti k množině *mladý*, avšak má vysoký stupeň příslušnosti k množině *ve středním věku*. Na výše uvedeném obrázku je funkce příslušnosti ve tvaru lichoběžníku, ale používají se i jiné tvary jako je trojúhelník či Gaussova křivka.

Fuzzifikace je proces, kdy vstupní data převádíme na hodnoty příslušnosti k jednotlivým fuzzy množinám [39]. Na obrázku 4.2 je zobrazena fuzzifikace vstupních dat (věk) do fuzzy množin *mladý, středního věku a starý*. Fuzzy množiny by měly pokrývat celý interval možných vstupních hodnot aby bylo možné fuzzifikovat jakoukoli hodnotu vstupní veličiny (Pro případ věku by fuzzy množiny měly pokrývat alespoň interval $(0, 120)$). Pro skutečné pokrytí celého intervalu možných vstupních hodnot, by se fuzzy množiny měly překrývat [40].

Vyhodnocení pravidel (Inference) - pro každý fuzzy proces musí existovat množina pravidel, které definují vztahy mezi vstupními a výstupními proměnnými fuzzy procesu. Tato pravidla jsou většinou získávána z předchozích zkušeností či z obecné znalosti daného problému [39]. Navážeme-li na příklad s věkem, můžeme definovat pravidla například pro to, jaký příjem může člověk mít v daném věku. (Uvedená pravidla jsou velmi zjednodušená a nemusí odpovídat skutečnosti)

1. Pokud je člověk příliš mladý, má nulový nebo nízký příjem
2. Člověk ve středních letech má vysoký příjem
3. Starý člověk má nízký příjem

K vyhodnocení pravidel se nejčastěji používá **Mamdaniho** nebo **Sugenova** inferenční metoda Mamdaniho metoda funguje na principu "ořezání"výstupní funkce. Nejdříve se určí funkční hodnota příslušnostních funkcí při daném vstupu. Funkční hodnota se pak vynese do grafu výstupních funkcí a ve výšce vyneseného bodu se provede řez. Pro lepší představu je proces vynášení funkčních hodnot zobrazen na obrázku 4.3



Obrázek 4.3: Vynesení funkčních hodnot do grafu[41]

Sugenova metoda zjednodušuje výstupní funkce do takového tvaru, že mají hodnotu 1 pouze v jediném bodě. Výstupní funkce jsou tedy ve tvaru kolmice na osu X. I v této metodě vynášíme funkční hodnoty jako u Mamdaniho metody, průsečík však bude pro jednu funkci vždy pouze v jednom bodě.

Pro náš případ můžeme využít Mamdaniho metodu ve zjednodušené textové podobě. Uvedeme-li konkrétní příklad člověka, kterému je 35 let, jeho hodnoty příslušnosti mohou být 0,2 pro množinu *mladý* a 0,8 pro množinu ve *středních letech*. Dosazením do pravidel získáme funkce příslušnosti pro aktivovaná pravidla, tedy že daný člověk má 20% šanci mít nízký příjem a 80% šanci mít příjem vysoký.

Agregace výstupů aktivovaných pravidel - V tomto kroku se sjednocují všechny příslušnosti aktivovaných pravidel (výstupní funkce) do jedné množiny.

Defuzzifikace - zde se převádí sjednocená funkce příslušnosti pravidel na výstupní veličinu. Pro Mamdaniho metodu probíhá defuzzifikace jako výpočet plošného těžiště pro plochu sjednocené výstupní funkce. U Sugenovy metody je výpočet podstatně jednodušší, pouze se spočítá vážený průměr jednotlivých průsečíků výstupních funkcí.[39]

4.4 Strojové vidění

Strojové vidění je technologie umožnující strojům vnímat své okolí pomocí snímání a zpracování obrazu. Tato technologie bývá používána především v oblastech jako je vizuální inspekce, detekce defektů, polohování objektů a zjišťování jejich rozměrů, třídění a sledování objektů [42].

Strojové vidění přistupuje k danému úkolu velmi podobně jako člověk. Stejně jako lidské oko zachytí i kamera obraz zkoumaného předmětu, systém jej vyhodnotí podle předepsaného algoritmu a provede akci na základě výsledku vyhodnocení. Sledovaný objekt, většinou trojrozměrný, je ozařován zdrojem záření. Objekt musí být schopen toto záření odrazit tak, aby odražené záření vytvořilo na snímacím prvku senzoru jasový dvojrozměrný obraz. Důležité je, aby v tomto obraze byla obsažena informace, kterou je zapotřebí o sledovaném objektu znát [43].

Pro zpracování zachyceného obrazu se používají různé přístupy. Mezi nejznámější patří použití metod umělé inteligence, které umožňuje například klasifikaci obrázků, detekci či segmentaci objektů. Mimo umělou inteligenci lze použít tradiční algoritmy pro zpracování obrazu jako je SIFT či SURF algoritmus. Tyto algoritmy lze použít např. pro filtraci či segmentaci obrazu.

4.4.1 Princip snímání obrazu

Obecný princip zachycení obrazu spočívá v použití senzorů detekujících světlo. Tyto senzory jsou schopné generovat elektrický náboj úměrný intenzitě dopadajícího světla. Nejznámější zástupci těchto senzorů jsou CCD (charge coupled device) a CMOS (Complementary Metal-Oxide Semiconductor) senzory. Oba tyto senzory používají mřížku velkého množství fotocitlivých buněk (pixelů), na které dopadá světlo. Liší se především způsobem, jakým je vygenerovaný elektrický náboj uložen do paměti. CCD senzor "posouvá"jednotlivé náboje po mřížce do výstupního zesilovače, kde je náboj převeden na napětí. Funkcionalitou se to podobá posuvnému registru. [44]. Senzory CMOS zesilují náboj z každého pixelu zvlášť pomocí tranzistorů a přímo zapisují do paměti. CCD senzory jsou náročnější na výrobu a tedy dražší. Jsou ale méně náchylné k šumu, poskytují tedy kvalitnější výstupní obraz. CMOS senzory jsou levnější na výrobu a také energeticky úspornější. Neposkytují však tak kvalitní obraz jako CCD.[45] K zachycení barvy světla se před mřížku fotocitlivých senzorů umisťují filtry, které propouští jenom světlo o určité vlnové délce (barvě). U monochromatických kamer se používají tři různé mřížky filtrů pro jednotlivé barvy - červenou, zelenou a modrou. Pro získání výsledného obrazu je tedy nutné pořídit tři snímky, každý s jiným filtrem. Tyto snímky se pak sečtou do výsledného, vysoce kvalitního obrazu. U barevných kamer je použita pouze jedna mřížka filtrů, jednotlivé filtry v mřížce ale propouštějí různé barvy světla. [46] Umístění jednotlivých filtrů může být provedeno např. pomocí Bayerovy masky. Výsledné hodnoty pixelů jsou pak dopočítány pomocí speciálních algoritmů.

4.4.2 Metody Deep learning ve strojovém vidění

Klasifikace - cílem klasifikace je přiřazení správného označení (labelu) vstupnímu obrazu. Může-li být obrázku přiřazen právě jeden label, jedná se o single-label klasifikaci. Tato metoda je používána nejčastěji. Pokud obrázek může patřit do více kategorií současně (mít více labelů), jedná se o multi-label klasifikaci. Tento způsob klasifikace bývá používán například ve zdravotnictví, kdy lze z jednoho rentgenového snímku odhalit více zdravotních komplikací [47]. Další rozdělení klasifikačních metod je na více třídovou (multi-class) a binární klasifikaci. U binární jsou obrázky klasifikovány pouze do dvou tříd, u multi-class mohou být klasifikovány do 3 a více tříd.

Aby mohla NN pracovat s labely obrázků, je nutné je nejdříve převést do číselné podoby. K tomu se dá využít jedné ze dvou metod - integer a one-hot kódování. Integer kódování spočívá v převedení všech labelů na celá čísla. Je vhodné pro situace, kdy je možno labely seřadit podle nějakého vztahu. Příkladem můžou být labely "malý", "střední", "velký". One-hot se používá pro labely, mezi nimiž žádný podobný vztah není. Kódování je provedeno formou převodu na binární vektor (obsahuje pouze 0 a 1) o rozměru $1 \text{x} N$, kde N je počet tříd. U single-label klasifikace obsahuje tento vektor vždy právě jednu jedničku. V případě multi-label, může obsahovat až N jedniček. Pozice jedničky ve vektoru udává, která třída je v obrázku přítomna. Právě pro multi-label klasifikaci je kódování one-hot ideální.

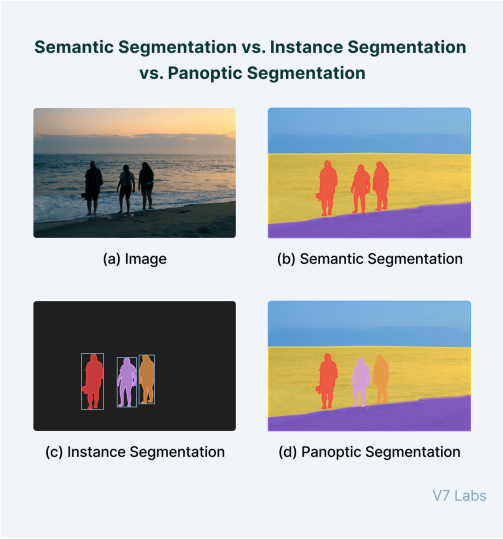
Ke klasifikaci lze využít metod supervized i unsupervized učení. Unsupervized učení je vhodné v případě, kdy máme k dispozici velké množství obrázků k tréninku a vytvoření labelu pro každý z nich by bylo časově náročné. Obrázky jsou pak rozděleny do několika kategorií na základě jejich společných rysů. Častěji používané jsou metody supervized učení, například metoda podpůrných vektorů, metoda k-nejbližších sousedů nebo neuronové sítě. Mezi významné architektury CNN pro klasifikaci patří **Mobilenet a VGG16**.

Detekce objektů - na rozdíl od klasifikace se detekce objektů nezabývá obrázkem jako celkem, ale objekty v něm obsažených. Kromě zjištění přítomnosti daného objektu v obrázku je provedena také lokalizace tohoto objektu, výstupem modelu je tedy kromě informace o přítomnosti objektu i souřadnice obdélníku (bounding boxu), v němž se daný objekt nachází. Labeling obrázků zde probíhá označením žádaných objektů pomocí bounding boxů. Tyto bounding boxy musí být následně převedeny na souřadnice, aby mohly být použity jako vstup pro NN. Pro detekci objektů se ve většině případů používají metody supervized učení. Často používané jsou modely hlubokého učení patřící do rodiny **R-CNN** nebo **YOLO**.
R-CNN neboli *Region-Based Convolutional Neural Network* je typ konvoluční neuronové sítě, na jejíž vstup jsou přivedeny souřadnice různých oblastí z obrázku. Těchto oblastí je asi 2000 a jsou vybrány např. pomocí selective search algoritmu. CNN z těchto oblastí vytvoří feature map. Dále je feature map zpracována například pomocí metody podpůrných vektorů, která klasifikuje jaký objekt se v dané oblasti nachází. Rychlost této metody je v řádech desítek vteřin na snímek [48].

Tento model je tedy velmi pomalý a dnes se používá především jeho vylepšená verze Faster R-CNN. Tato verze nerozděluje vstupní obrázek do oblastí, na vstup je přiveden celý obrázek. Rozdělení do oblastí je provedeno až na feature map. Dalším vylepšením této verze je nahrazení algoritmu pro výběr oblastí (jako je selective search) další neuronovou sítí. Rychlost této metody je v řádech stovek milisekund na snímek [48].[49]

YOLO neboli *You only look once* je velmi rychlá metoda detekce objektů a je tedy často používaná pro detekci v reálném čase. Obrázek je zde rozdělen do mřížky o rozměrech NxN. Každá buňka v mřížce je pak zodpovědná za detekci a lokalizaci objektu v ní obsaženého. Pro každou buňku se odhadne poloha bounding boxu a určí pravděpodobnost s jakou se v daném bounding boxu objekt nachází. V dalším kroku se ponechají pouze bounding boxy s nejvyšším confidence score. Výhodou této metody je, že využívá pouze jednu CNN a tudíž je velmi rychlá. Její rychlost může být 45 až 150 detekcí za sekundu [50]. Nejnovějším členem rodiny *YOLO* je *YOLOv7*.[51]

Segmentace je vlastně vylepšená verze detekce objektů, dokáže totiž přesně lokalizovat pozici hledaného objektu na obrázku pomocí masky. Segmentaci dělíme do dvou základních typů - *sémantická a instanční* segmentace. Sémantická segmentace přiřazuje label každému pixelu obrázku. Výsledkem je tedy maska, která zobrazuje hledané objekty, ale nerozlišuje mezi nimi. Pokud by tedy bylo na obrázku více výskytů hledaného objektu, jejich masky by byly označeny stejnou barvou. Instanční segmentace rovněž přiřazuje label každému pixelu, zároveň ale rozlišuje mezi instancemi jednotlivých objektů. Instanční segmentace se však soustředí pouze na segmentaci hledaných objektů, neumí segmentovat např. pozadí. V případě více výskytů objektu na obrázku, bude mít maska každého výskytu jinou barvu. Někdy se rovněž hovoří o panoptické segmentaci, která kombinuje vlastnosti obou předchozích. [52]
Pro lepší představu jsou jednotlivé typy segmentace zobrazeny na obrázku 4.4.



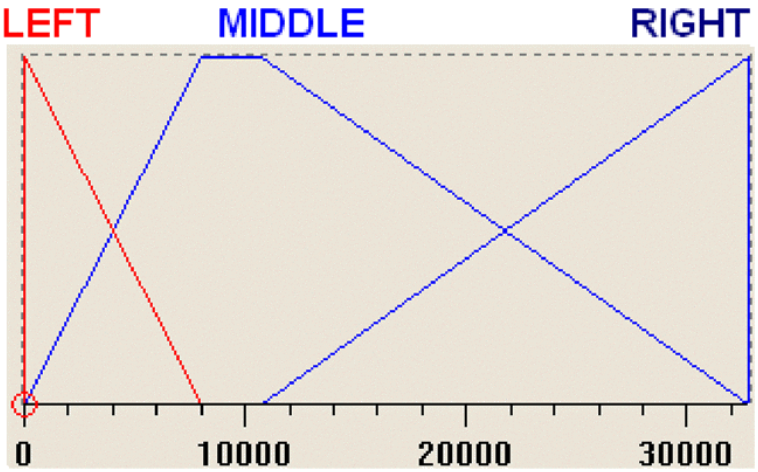
Obrázek 4.4: Různé typy segmentace [53]

Kapitola 5

Rozbor typických úloh s využitím umělé inteligence

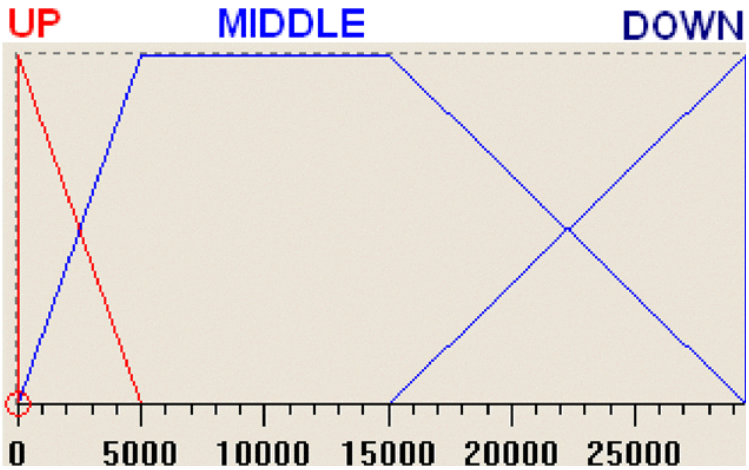
5.1 Příklad fuzzy řízení s PLC

V tomto příkladu bude popsáno fuzzy řízení rychlosti pohybu robotického ramena. Zdroj informací je [54]. Vstupem do PLC jsou dva analogové signály v rozsahu 0-10 V, které reprezentují vychýlení joysticku ve dvou osách. Změnu napětí při vychýlení zajišťují dva potenciometry. Joystick je pomocí na něm umístěného přepínače schopen ovládat dva různé klouby ramena. Výstupem jsou rovněž dva analogové signály v rozsahu 0-10 V, tyto signály udávají rychlost pohybu v jednotlivých osách. Je dáno celkem 6 funkcí příslušnosti, 3 pro pohyb v každé ose. Funkce příslušnosti pro pohyb v ose X je zobrazena na obrázku 5.1



Obrázek 5.1: Funkce příslušnosti pro pohyb v ose X

Funkce příslušnosti pro pohyb v ose Y je na obrázku 5.2. U obou obrázků znázorňujících funkce příslušnosti reprezentují hodnoty na ose X vstupní napětí do PLC (0-10 V) převedené pomocí A/D převodníku na reálná čísla v rozsahu $(0, 32500)$.



Obrázek 5.2: Funkce příslušnosti pro pohyb v ose Y

Množina pravidel popisující vztah mezi vstupem a výstupem je následující

1. Pokud je Vstup1 *VLEVO* pak je Výstup1 *RYCHLE*
2. Pokud je Vstup1 *UPROSTŘED* pak je Výstup1 *POMALU*
3. Pokud je Vstup1 *VPRAVO* pak je Výstup1 *RYCHLE*
4. Pokud je Vstup2 *NAHORU* pak je Výstup2 *RYCHLE*
5. Pokud je Vstup2 *UPROSTŘED* pak je Výstup2 *POMALU*
6. Pokud je Vstup2 *DOLŮ* pak je Výstup2 *RYCHLE*

Úlohy fuzzy řízení jsou v PLC vykonávány pomocí nástroje Fuzzy Control ++ vyvinutého firmou Siemens. V tomto nástroji se pracuje s tzv. Fuzzy Programming Language (FPL), který umožňuje snadnou definici fuzzy množin, pravidel, funkcí příslušnosti, atd.

Řízení fuzzy je v tomto příkladu použito pouze pro řízení rychlosti ramena. Ostatní důležité operace funkce jako uchopování, řízení otáček motoru atd. jsou prováděny za pomocí klasického řízení s PLC. Řízení rychlosti fuzzy systémem je zde prováděno bez zpětné vazby o skutečné rychlosti. Ta se může měnit v závislosti např. na hmotnosti zátěže, kterou rameno drží. Toto může být vyřešeno pomocí regulace, ke které je opět možno využít fuzzy řízení v podobě PI fuzzy regulátoru.

5.2 Příklad využití strojového vidění s TM-NPU

V tomto příkladu bude seznámení s možností využití strojového vidění pro univerzální úchop robotického ramena. V příkladu je využíván modul TM-NPU, který je použit i v praktické části této práce. Zdroj informací je [55].

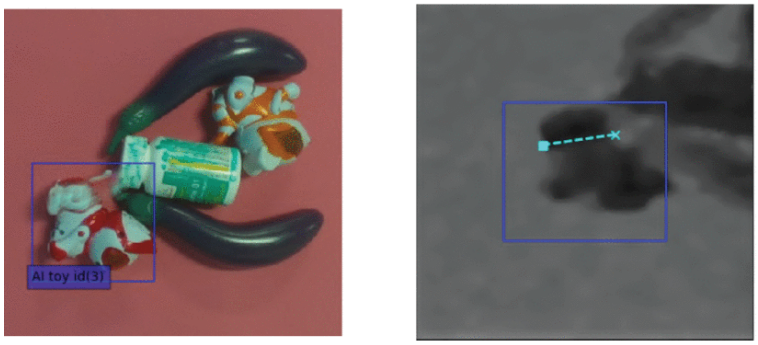
Univerzální úchop znamená, že se dokáže automaticky přizpůsobit detekovanému objektu a jeho rozměrům. Jeho univerzálnost rovněž spočívá ve schopnosti odhadnout vhodný úchop bez ohledu na polohu, natočení a vzdálenost požadovaného objektu (za předpokladu že je v dosahu ramena a snímací kamery). Návrh vhodné univerzálního úchopu závisí především na typu koncového efektoru (vakuové přísavky, dvoučelistní uchopovače, chapadla a.j.), typu robota (jednoramenný, víceramenný) a rozpoložení uchopovaných objektů (zda-li je objekt osamocen nebo je obklopen jinými objekty).

V prezentovaném příkladu je úloha následující: systému je prezentována nádoba s několika objekty. Uživatel si pomocí HMI (human-machine interface) vybere objekt, který má být uchopen a přemístěn do druhé nádoby. V případě že požadovaný objekt se v první nádobě již nenachází, uživatel je upozorněn. Pohybové vybavení pro úlohu sestává s jednoramenného robota KUKA KR 3 Agilus se šesti stupni volnosti a dvoučelistního uchopovače Robotiq 2F-85. Řízení robotického ramena je prováděno pomocí PLC S7-1516 a k němu připojeného modulu TM-NPU, který slouží pro zpracování obrazu z kamery pomocí CNN. Použitá kamera je Intel RealSense D435, která dokáže zachytit i hloubku obrazu. Inference v modulu TM-NPU je vyvolána přes PLC, které tak učiní na základě požadavku uživatele vloženého skrze HMI. Výsledkem inference je typ detekovaného objektu a souřadnice pixelů, kde by měl být proveden úchop. Na PLC je aby na základě získaných souřadnic nastavilo šířku úchopu a řídilo pohyb ramena.

Obrázek 5.3: Objekty v první nádobě

Modul TM-NPU je využíván k dvěma hlavním úlohám: **detekci objektů** a **výpočet úchopu**. Jako architektura CNN pro detekci objektů je použita Mobilenet SSD, která je vhodná pro edge zařízení. Na vstup do této CNN je přiveden barevný obrázek z kamery (bez informace o hloubce). Výstupem jsou souřadnice bounding boxů označující jednotlivé objekty a pravděpodobnost s jakou se jedná o konkrétní objekt. Samotné detekci předchází preprocessing vstupních barevných obrázků, který zahrnuje kroky jako škálování, ořezávání a konverzi kódování barev (například do RGB nebo BGR kódování). Síť byla trénována na 200 různých obrázcích zachycujících objekty, které se vyskytují ve finální aplikaci.

Pro výpočet úchopu je použitá speciální neuronová síť **FC-GQ-CNN**, která patří do rodiny **Dex-Net** vyvíjené univerzitou Berkeley v Kalifornii. Na vstup této sítě je přivedena část hloubkového snímku z kamery (depth frame). Tato část je vyříznuta z původního depth frame na základě souřadnic bounding boxů z předchozího kroku detekce objektů. Neuronová síť vyhodnotí miliony různých úchopů se čtyřmi stupni volnosti a vybere ten nejlepší. Na rozdíl od CNN použité pro detekci objektů, tato neuronová síť je velmi dobrá v generalizaci a ve výsledku dokáže předpovědět vhodný úchop i pro objekty, na kterých nebyla trénována.



(a) Color frame with bounding box.

(b) Depth frame with computed grasp.

Obrázek 5.4: Výsledek detekce objektu a výpočtu úchopu

Tento systém byl představen na veletrhu v Hannoveru v roce 2019. V rámci této akce dokázal běžet 5 dní bez přerušení s průměrnou výkonností 1500 úchopů na den. Pro málo zaplněnou krabici s objekty dosahovala úspěšnost úchopu 90-95%. Při větším množství objektů v krabici je pro systém těžší odhadnout správný úchop (zejména pokud je požadovaný objekt z části překryt jinými objekty). Celý proces od zpracování pokynu od uživatele přes výpočet vhodného úchopu až do počátku pohybu ramena zabral méně než jednu vteřinu. Celkový výkon PLC a TM-NPU přitom nepřesáhl 10 W. Tyto výsledky dělají ze systému ideální nástroj pro industriální aplikace v reálném čase.

Kapitola 6

Modul TM-NPU

Modul TM-NPU vyvinutý firmou Siemens, umožňuje zpracování dat pomocí neuronových sítí. Obzvlášť vhodný je pro práci s konvolučními neuronovými sítěmi (CNN). Jedná se o tzv. edge zařízení, tedy je umístěno blízko zdroje dat - technologického procesu. Stejně jako běžně používané I/O moduly je modul TM-NPU připojen k PLC pomocí tzv. backplane bus.

Modul je vybaven procesorovou VPU jednotkou *Intel Movidius Myriad X Vision*, která umožňuje účinné využití neuronových sítí. VPU procesor je speciální typ procesoru, který je určen k urychlení úloh strojového vidění [56].

6.1 Periferie

K modulu mohou být připojeny externí podporované senzory pomocí USB 3.1 nebo RJ-45 rozhraní. Modul umožňuje použití vždy pouze jednoho portu v jednom čase. V současné době jsou jedinými podporovanými senzory kamery značky Intel a Basler. Konkrétně se jedná o tyto modely:

- Intel RealSense D435 (tento model je využíván i v této práci)
- Intel RealSense D415
- Basler acA1300-60gc
- Basler acA1920-25gc

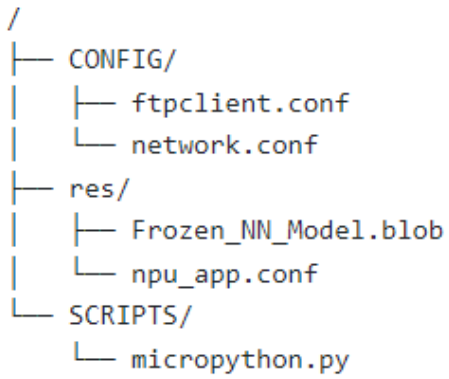
V tiskové zprávě společnosti se objevuje i zmínka o možnosti využití mikrofonu jako periferie [57], nicméně technická dokumentace modulu takovou možnost nezmiňuje.

6.2 Paměťová SD karta

Jako paměť slouží modulu paměťová SD karta, která se běžně používá i pro interní paměť u PLC. Jedná se o modely s article number *6ES7954-8LPxx-0AA0*. Doporučená velikost pro TM-NPU je

2GB. Na kartu se nahrává vytrénovaný model neuronové sítě, soubor se síťovou konfigurací, Micropython skript a soubor s konfigurací NN modelu. **Micropython** je úspornou a efektivní implementací jazyka Python 3, která obsahuje pouze malý počet vybraných knihoven a je optimalizována pro běh na mikrokontrolerech [58]. Ve skriptu jsou také definovány vstupy a výstupy PLC, které budou použity pro komunikaci s modulem. Veškerá funkcionalita modulu je zajištěna právě pomocí Micropython skriptu. [59]

Modul umožňuje přenos souborů pomocí FTP serveru. K využití této funkcionality je potřeba na SD kartu nahrát konfigurační soubor FTP klienta (soubor *ftpclient.conf*). Soubor se síťovou konfigurací (soubor *network.conf*) slouží k nastavení IP adresy modulu, aby bylo možné komunikovat s kamerou (je-li kamera připojena pomocí Ethernetu). Modul vyžaduje tento soubor i v případě, že je použita USB kamera. Z manuálu však není jasné k čemu je v takovémto případě soubor využíván. Struktura potřebných složek a souborů je zobrazena na obrázku 6.1 (soubor *ftpclient.conf* není povinný)



Obrázek 6.1: Struktura adresářů SD karty

Výše uvedenou adresářovou strukturu lze na SD kartě vytvořit ručně nebo se může využít aplikace AI Model Deployer, která tuto strukturu vygeneruje za nás.

6.3 Model neuronové sítě

Vytrénovaný model neuronové sítě musí být na kartu uložen ve formátu .blob. Tento je obecně využíván k reprezentaci binárních dat. Pro konverzi do tohoto formátu lze použít speciální software od firmy Siemens - **AI Model Deployer**. Tento software umožňuje konverzi modelů pro klasifikaci obrázků a detekci objektů. Jsou podporovány modely následujících frameworků: *TensorFlow1,2, Keras, ONNX*. Nejčastěji jsou používány modely s architekturou *MobileNet*. Maximální velikost již překonvertovaného modelu by neměla přesáhnout 50 Mb.

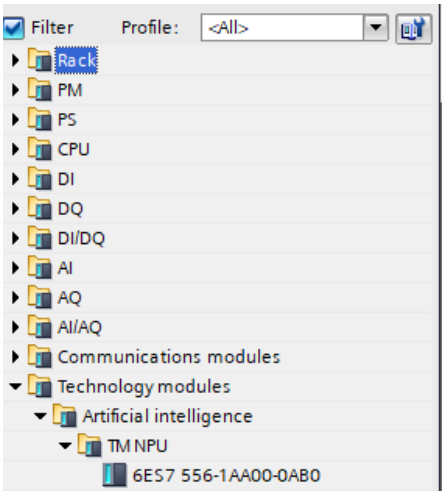
AI Model Deployer z velké části využívá na pozadí software od firmy Intel - **OpenVINO**. Tento software slouží k implementaci modelů umělé inteligence na zařízení s CPU, GPU a VPU značky Intel. Pokud by tedy nebyl k dispozici AI Model Deployer, lze použít i tento nástroj, který je zdarma (má však nevýhodu v tom, že nemá grafické rozhraní).

Na kartě může být nahráno několik modelů, v jednom čase může být však aktivní pouze jeden. Výběr modelu je prováděn pomocí konfiguračního souboru *npu_app.conf*. V tomto souboru je mimo jiné specifikován typ použité kamery a její rozlišení. [59]

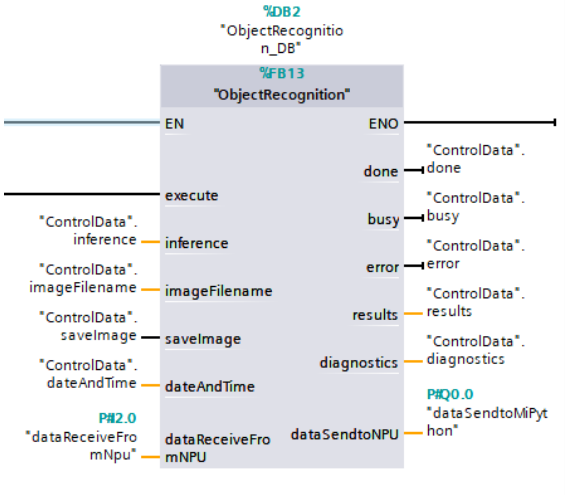
6.4 Práce s modulem v TIA Portalu

Pro práci s modulem bylo nutno doinstalovat podpůrný hardwarový balíček (HSP), zde je použit konkrétně *HSP_V16_0333_002_S71500_TM_NPU_1.2_1.3*. Po nainstalování byl model dostupný v katalogu modulů pod svým article number *6ES7556-1AA00-0AB0* (obrázek 6.2). V mém případě bylo navíc zapotřebí nahrát novou verzi firmwaru na modul, aby svou verzi odpovídal HSP v TIA Portalu. Pro update firmwaru modulu bylo zapotřebí na SD kartě vytvořit složku *FWUPD* a do ní nahrát soubor pro update, nakonec modul restartovat. Po úspěšném update byl na kartě vygenerován log soubor s hlášením o úspěšném provedení update. Nakonec bylo ještě nutné provést update backplanebus konektoru. Ten se prováděl v TIAPortalu, v nástroji online diagnostiky modulu. HSP i veškerý firmware mi byl poskytnut firmou Siemens.

Společnost Siemens poskytla spolu s modulem i aplikační příklad s programem pro PLC. Hlavní části tohoto programu je funkční blok (FB) *ObjectRecognition* (obrázek 6.3), který zajišťuje komunikaci s TM-NPU. Blok je naprogramován v ST (structured text) formátu.



Obrázek 6.2: TM-NPU v katalogu TIA Portalu



Obrázek 6.3: FB ObjectRecognition

Pro pořízení obrázku, spuštění inference a získání klasifikačních dat je nutno přivést náběžnou hranu na vstupní parametr *execute*. V mém případě je tento parametr ovládán digitálním výstupem indukčního senzoru. Parametr *inference* slouží k nastavení typu inference. Parametr může nabývat následujících hodnot: 0 - žádná inference, 1 - klasifikace, 2 - detekce objektů pomocí *YOLO*, 3 - detekce objektů pomocí *Tensorflow*. V této úloze je parametr nastaven na hodnotu 1 - klasifikace. Parametry *imageFilename* a *saveImage* slouží k ukládání pořízených obrázků na SD kartu v modulu(lze využít k shromažďování trénovacích dat), této možnosti není využíváno, *saveImage* je nastaveno na *False*. Všechny tyto parametry jsou před odesláním do modulu "zabaleny"do jednoho objektu *dataSendtoMiPython*, který lze vidět jako parametr v pravé dolní části FB.

Podobný objekt je i *dataReceiveFromNpu*, jenž obsahuje data, která byla z NPU přijata. Jeho obsah je znázorněn na obrázku 6.4.

PLC tags

	Name	Tag table	Data type	Address
1	dataReceiveFromNpu	NPU_data	"typeDataReceiveFromNpuResults"	%I2.0
2	fwBytes		typeFWBites	%I2.0
3	newDataAvailable		Bool	%I4.0
4	active		Bool	%I4.1
5	appStatus		Byte	%IB5
6	results		Array[0..200] of Byte	%I6.0
7	results[0]		Byte	%IB6
8	results[1]		Byte	%IB7
9	results[2]		Byte	%IB8

Obrázek 6.4: Struktura datového typu pro příjem dat z NPU

Nejdůležitější proměnnou ve výše zobrazeném objektu je pole *results*, do kterého se nahrávají klasifikační data. Pole obsahuje celkem 200 prvků, každý odpovídá jedné klasifikační třídě. Lze tedy pracovat s modelem, který klasifikuje až 200 různých tříd. Všechny ostatní proměnné v tomto objektu jsou pouze informační nebo diagnostické. Například proměnná *active* indikuje tzv. lifesign modulu a proměnná *appStatus* obsahuje stavový kód modulu, který pak lze dohledat v manuálu.

První tři proměnné, na pravé straně funkčního bloku slouží k indikaci stavu FB. Proměnná *done* nabývá log. 1 když je inference úspěšně dokončena a PLC přijme platná data. Proměnná *busy* je v log. 1 po celou dobu trvání inference a výměny dat mezi PLC a NPU. Sestupná hrana této proměnné je použita dále v programu, pro sepnutí kontaktu ihned po dokončení inference. Náběžná hrana v proměnné *error* indikuje že zpracování FB neproběhlo v pořádku. Proměnné *results* a *diagnostics* pocházejí z objektu *dataReceiveFromNpu*, který byla popsán v předchozím odstavci.

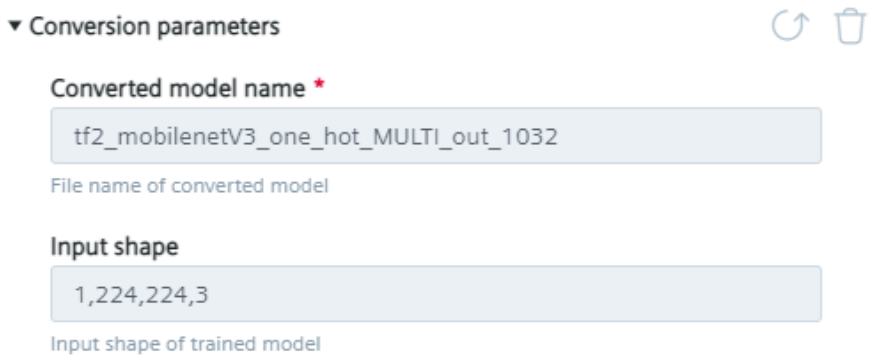
Ve vlastnostech modulu (properties), lze nastavit, který z portů pro kameru bude aktivní. Rovněž tady lze zapnout FTP klient a server.
Pro komunikaci s PLC modul vyžaduje 256 bytů vstupní i výstupní paměti PLC.

6.5 Práce s AI Model Deployer

Jak již bylo zmíněno, modul pro svou činnost vyžaduje model neuronové sítě ve formátu **.blob**. Pro konverzi do tohoto formátu jsem použil dedikovaný software **AI Model Deployer**. Program prozatím umožňuje konvertovat pouze modely pro klasifikaci a detekci objektů.

Software umožňuje konverzi z několika různých formátů modelů, jako například tensorflow frozen (*.pb*) a Keras model (*.h5)*. Jelikož v této práci pracuji s knihovnou Keras (která součástí knihovny Tensorflow), použil jsem formát .h5. Získat model v tomto formátu je velmi snadné, stačí na vytrénovaném keras modelu zavolat metodu **save()** a jako parametr poskytnout cestu k uložení. Nejnovější verze knihovny Tensorflow, která je programem podporována je 2.6. Právě tuto verzi využívám.

Před samotnou konverzí je možno nastavit několik různých parametrů modelu, jako například rozměr vstupních dat, počet vstupních a výstupních neuronů, některé kroky preprocessingu aj. V mém případě bylo zapotřebí vyplnit pouze jeden parametr a to rozměr vstupu. Ostatní parametry byly buď správně předvyplněny (na základě verze NPU modulu, kterou jsem v programu zvolil) nebo byly zanechány prázdné. Tyto prázdné parametry jsou pak totiž doplněny Micropython scriptem při inferenci.



Obrázek 6.5: Příklad parametrů pro konverzi v AI Model Deployer

Konverze obvykle nezabere více než 20 vteřin (pro klasifikační .h5 model). Překonvertovaný model by neměl přesáhnou velikost 50 Mb.

Kromě konverze modelu umožňuje program rovněž vytvořit adresářovou strukturu, kterou pak lze již přímo nahrát na SD kartu modulu. Aby ji bylo možné vytvořit, je nezbytné poskytnout Micropython skript, vybrat konvertovaný model z předchozího kroku a nastavit typ použitého PLC a kamery. Strukturu lze pak stáhnout v zazipovaném formátu, před přesunem na SD kartu je nutné ji rozbalit.

Kapitola 7

Návrh a realizace demonstračního experimentu

V demonstrační úloze je využita umělá inteligence ve formě CNN pro klasifikaci obrázků. Klasifikována je barva, stav a přítomnost dřevěných puků. Informace o struktuře neuronové sítě a procesu trénování jsou v následující kapitole.

V úloze je využito otočného karuselu a barevných puků, které se na něj umisťují. Dále jsou využity nástroje pro simulaci vrtání, ražení a ofuku puků. Tyto nástroje pouze simulují konkrétní činnost, na puku tedy nejsou prováděny žádné skutečné úpravy. Karusel obsahuje celkem 8 výřezů do který se dají vložit puky. Pod každým z těchto výřezů (na spodní straně) je umístěn kovový váleček, který je snímán indukčním senzorem. Senzor je tedy aktivován vždy, když je nad ním část karuselu, kde se nachází výřez pro puk. Pro monitoring polohy puků je v okolí karuselu umístěno několik optických senzorů. Tyto senzory jsou u každého ze tří nástrojů a indikují že puk se zrovna nachází pod tímto přístrojem.

Obrázky puků jsou pořizovány kamerou Intel Realsense D435, která je umístěna nad karuselem. Obrázek je pořízen vždy, když kovový válec pod karuselem aktivuje indukční senzor. Po pořízení obrázku je ihned provedena klasifikace a data o ní jsou poslána do PLC. Protože pořízení obrázku i klasifikace jsou prováděny dost rychle, není potřeba karusel zastavovat.

Klasifikovány jsou následující třídy: barva (může být modrá nebo červená), díra uvnitř puku, označení izolační lepící páskou, zašpinění (v podobě zbytků gumy po gumování), vychýlení puku z výřezu a samotná přítomnost puku. Jeden puk může mít více různých vlastností, které jsou klasifikovány (tedy patřit do více tříd) a proto je využita multi-label klasifikace.

Na základě přítomnosti či nepřítomnosti klasifikovaných tříd se poté rozhoduje, jaké nástroje budou na puk použity.

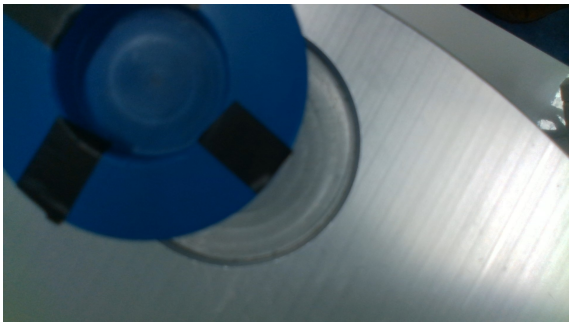
- **Vrtačka** je použita, pokud puk nemá díru
- **Razník** je použit, pokud puk není označen

- **Ofuk** je použit, pokud je puk zašpiněn.

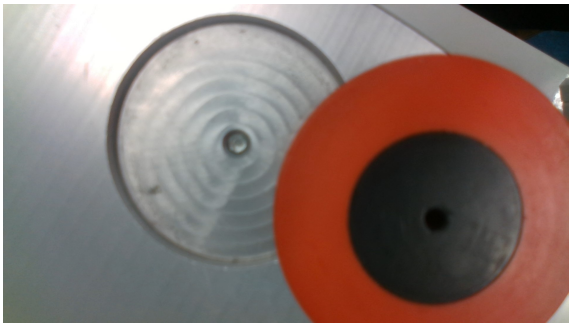
Při detekci vychýlení puku je celý karusel zastaven a uživatel vyzván aby jej umístil do správné polohy. Při příliš velkém vychýlení se může stát, že ne všechny vlastnosti puku budou klasifikovány správně. Je-li detekován prázdný výřez, uživatel je pouze informován, žádná další akce není provedena.

Níže je zobrazeno několik různých stavů puků, které mohou při klasifikaci nastat.

(a) Modrý puk s dírou a značkou



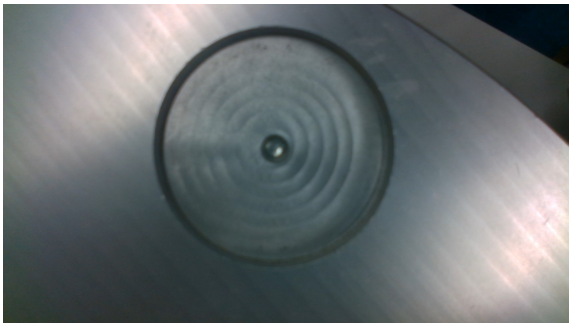
(b) Modrý puk se značkou, vychýlený



(c) Červený puk s dírou, vychýlený

(d) Červený puk se značkou

(e) Červený puk, špinavý



(f) Prázdný výřez

Obrázek 7.1: Ukázka několika různých stavů puků

7.1 Příprava klasifikačního modelu

Aktuální verze modulu podporuje následující frameworky pro tvorbu NN modelů: *Tensorflow 1 a 2, Keras a ONNX*. V této práci jsou využívány frameworky Tensorflow 2.6.0 a Keras 2.6.0. Pro trénink a práci s modelem neuronové sítě je využito virtuálního python prostředí (virtual environment) s verzí pythonu 3.9.12.

V této práci je využito předtrénovaných modelů CNN z rodiny *Mobilenet*, které jsou volně dostupné pro knihovnu Keras. Díky těmto modelům není potřeba navrhovat celou strukturu neuronové sítě, struktura je již navržena speciálně pro zpracování obrazu. Na uživateli pak je, aby případně přidal další vrstvy nerunové sítě a model dotrénoval na konkrétních obrázcích. Dotrénováním se rozumí přizpůsobení parametrů NN (weights, biases) tak, aby správně predikovala třídy obrázků. Avšak i tento krok si lze výrazně usnadnit použitím předtrénovaných parametrů weights. Tyto parametry byly získány trénováním daného modelu na velmi velké databázi obrázků zvané *Imagenet*. Při první dopředné propagaci tak nejsou parametry zvoleny náhodně, což zajišťuje rychlejší zvýšení přesnosti modelu. Využití předtrénovaných parametrů také umožňuje vytrénování přesného modelu na menším počtu obrázků. Databáze Imagenet je mimo jiné často využívána jako standart při porovnávání výkonnosti různých modelů.[60]
Pro demonstrační úlohu byly otestovány dva modely, MobilenetV2 a MobilenetV3Small.

K trénování neuronových sítí je rovněž používán CUDA Toolkit, který umožňuje provádění potřebných výpočtů za pomoci GPU místo CPU. Jeho užití výrazně zvyšuje rychlost trénování. K CUDA Toolkitu je často využívána i knihovna cuDNN, která poskytuje implementaci standardních funkcí pro práci s neuronovými sítěmi, optimalizovaných pro běh na GPU [61]. V této práci je konkrétně využita CUDA 11.2 a knihovna cuDNN 8.1.

7.1.1 Architektura neuronové sítě

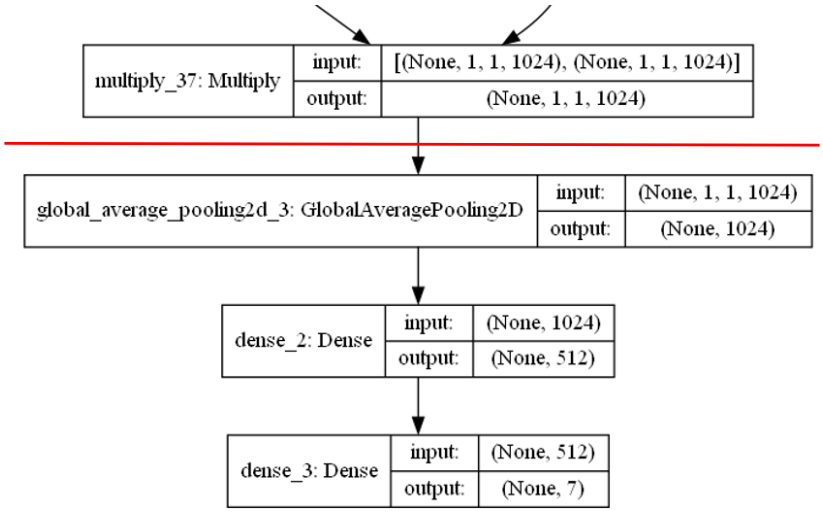
Jak již bylo zmíněno dříve, základ klasifikačního modelu tvoří předtrénovaná neuronová síť typu Mobilenet. Tato síť může být použita bez jakýchkoli úprav k okamžité klasifikaci. Bez vlastního trénování je schopna dobře klasifikovat pouze obrázky, na kterých byla předtrénována, proto ji musíme poskytnout vlastní obrázky a labely k tréninku. Používat takto celou síť, bez jakýchkoliv úprav v její struktuře je možné, avšak ne zcela ideální. Ve výchozím stavu totiž vždy produkuje výstupní vektor o délce 1000 prvků (protože síť byla trénována pro klasifikaci 1000 různých tříd). V mém případě probíhá klasifikace do sedmi různých tříd, ideální by tedy byl výstupní vektor o velikosti sedm. Změnu velikosti výstupu lze provést vynecháním poslední dense vrstvy modelu a její náhradou vlastními vrstvami. Protože se počítá s tím, že uživatel bude chtít podobnou úpravu provést, vynechání této vrstvy se jednoduše provede nastavením parametru *include_top* na hodnotu *False*. Na výpisu 1 je zobrazen postup k vytvoření předtrénovaného modelu MobilenetV3Small.

```
IMG_SHAPE = (224,224,3)
base_model = tf.keras.applications.MobileNetV3Small(input_shape=IMG_SHAPE,
                                                     weights='imagenet',
                                                     include_top=False
                                                     )
```

Výpis 1: Postup vytvoření předtrénovaného MobilenetV3 modelu

U takto upravené sítě si dále může uživatel přidal libovolný počet vlastních vrstev. V mém případě postačilo pouze přidat Global Average Pooling (GAP) vrstvu a 2 dense vrstvy. GAP zde slouží ke zmenšení dimenze výstupu z mobilenetu na 1D. První dense vrstva je určena pro zvýšení počtu parametrů k trénování a druhá slouží jako výstupní vrstva, která již produkuje predikce. Zvýšení počtu parametrů bylo nutné z toho důvodu, že předtrénovaná NN je použita ve "zmraženém"(freezed) módu, což znamená, že její parametry se netrénují. Zmražení se využívá často v případě, že chceme využít toho, co předtrénovaná síť už umí klasifikovat a nějak dále pracovat s jejím výstupem. Rovněž se využívá v případě, že nemáme dostatečný počet vstupních dat, takže by nebylo možné dobře vytrénovat všechny parametry. Jelikož jsem shromáždil poměrně velké množství obrázků k tréninku (přes 4000) model bylo možné přesně vytrénovat i bez zmražení. Protože však v mém případě trvala konvergence modelu bez zmražení vždy delší dobu, raději jsem ho používal.

Struktura posledních vrstev neuronů v modelu je zobrazena na obrázku 7.2



Obrázek 7.2: Poslední 4 vrstvy neuronů modelu

Jelikož celá neuronová síť obsahuje přes 100 vrstev neuronů, na obrázku jsou zobrazeny pouze poslední 4. Červená čára označuje, kde končí Mobilenet část sítě a kde začínají mé vlastní vrstvy. První dense vrstva, která slouží ke zvýšení počtu parametrů obsahuje 512 neuronů, druhá, výstupní obsahuje stejný počet neuronů, jako je počet klasifikovaných tříd (v mém případě 7). Aktivační funkce u výstupní vrstvy je *Sigmoid*. Tato funkce se spolu s multi-label klasifikací používá často, jelikož dokáže poskytnout pravděpodobnost přítomnosti dané třídy bez ohledu na přítomnost ostatních tříd. Při použití freezed modelu MobilenetV3Small a vlastních dense vrstev, které jsou zmíněny výše má síť celkem 528 391 trénovatelných parametrů.

Při spojování mobilenet a vlastních vrstev jsem nejdříve používal *keras.Sequential* API, avšak ze zatím neznámého důvodu měl modul NPU problém při zpracovávání modelu s takto spojenými vrstvami. Místo toho jsem tedy dále využíval tzv. *keras functional API*, které s modulem fungovalo dobře. Spojování vrstev pomocí functional API je zobrazeno níže. (výpis 2)

```
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(512)(x)
classifier = tf.keras.layers.Dense(7,activation="sigmoid")(x)
model=tf.keras.Model(inputs=base_model.input,outputs=classifier)
```

Výpis 2: Použití keras functional API pro spojení vrstev

Kromě modelu MobilenetV3Small byl otestován i model MobilenetV2. U toho bylo navíc zapotřebí přidat vlastní vstupní a normalizační vrstvu, aby vstupní data odpovídala požadavkům modelu (bude vysvětleno v kapitole 7.1.3). Tento model měl větší počet neuronů v poslední vrstvě (opět byl použit parametr $include\_top = False$) a tedy přidáním stejných vrstev jako u předchozího modelu vzniklo více trénovatelných parametrů, celkem 659 463. Zvýšený počet parametrů samozřejmě zna-

menal i delší dobu trénování, oproti modelu MobilenetV3Small se tento při 10 epochách trénoval asi o 50 % déle.

7.1.2 Pořízení a labeling obrázků

Pořizování obrázků probíhalo tak, že jsem velmi pomalu točil karuselem s pukem a u toho s velkou kadencí pořizoval snímky. V případě puku se značkou jsem navíc rotoval i se samotným pukem, aby se síť mohla naučit značku rozpoznávat při libovolném natočení puku. Podobně jsem postupoval i u snímků s vychýleným pukem, kde jsem měnil velikost a směr vychýlení. Tímto způsobem se mi podařilo nashromáždit téměř pět tisíc obrázků. Pro tuto demonstrační úlohu je to podstatně více, než kolik by bylo minimálně zapotřebí, avšak platí, že čím více trénovacích dat tím lépe.

Klasifikační úloha je typu multi-label (na jednom obrázku může být více tříd), pro kódování labelů je tedy využita metoda one-hot. Díky ní lze označit více tříd na jednom obrázku a rovněž naučit model, aby všechny tyto třídy rozeznal. Níže je zobrazeno kódování všech 7 klasifikačních tříd na one-hot vektory (výpis 3). Pro označení více tříd na jednom obrázku stačí odpovídající one-hot vektory sečíst.

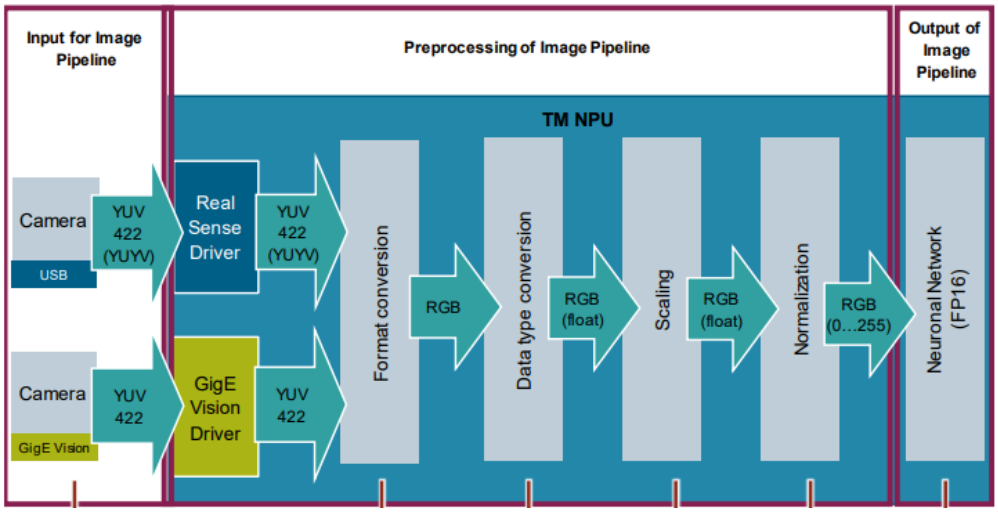
```
labels = {"red":((1,0,0,0,0,0,0)),
          "blue":((0,1,0,0,0,0,0)),
          "hole":((0,0,1,0,0,0,0)),
          "mark":((0,0,0,1,0,0,0)),
          "out":((0,0,0,0,1,0,0)),
          "mess":((0,0,0,0,0,1,0)),
          "empty":((0,0,0,0,0,0,1))
          }

red-hole-mark = labels["red"] + labels["hole"] + labels["mark"]
```

Výpis 3: Kódování použitých labelů

I když se v úloze se vyskytuje jen 7 tříd pro klasifikaci, pořizoval jsem více než 7 druhů snímků puků. Bylo to kvůli tomu, že třídy na obrázku se mohly různě kombinovat (např. značka mohla být na modrém puku, nebo na červeném puku spolu s dírou atd.) a bylo potřeba aby se i model naučil rozpoznávat tyto kombinace. Celkem jsem tak pořízené snímky rozdělil do 24 kategorií. Příkladem mohou být kategorie *blue-hole-mark*, *blue-mark*, *blue-out*, *red-hole-mark-out*, *red-mark-mess* atd.

7.1.3 Preprocessing obrázků

Před samotným trénováním NN na pořízených obrázcích je nutno tyto obrázky upravit (provést preprocessing). Jaké konkrétní úpravy mají být provedeny závisí na typu použité neuronové sítě a na aplikaci. Modul TM-NPU má přesně stanoveny kroky preprocessingu, které provádí pro každý pořízený obrázek (kroky lze vidět na obrázku 7.3). Stejný preprocessing musí být použit i při trénování modelu NN. Pro práci s obrázky je využívána knihovna *OpenCV*.



Obrázek 7.3: Kroky preprocessingu pro TM-NPU

Převod na barevný formát RGB - Tento krok není potřeba explicitně provádět, protože obrázky pořízené kamerou se na počítač ukládají již v tomto formátu. Při načítání obrázků pomocí knihovny openCV je však použit formát BGR, je třeba tedy provést jednoduchou konverzi z BGR do RGB formátu.

Konverze datového typu - při načtení obrázku jsou hodnoty jeho pixelů reprezentovány jako celočíselné hodnoty (integer) v rozsahu 0-255. Většina modelů NN však vyžaduje jako vstup hodnoty s plovoucí desetinnou čárkou (floats). Proto je nutné převést hodnoty pixelů obrázku z datového typu *int* na *float*. Převod na float je rovněž užitečný při normalizaci hodnot pixelů.

Změna velikosti obrázku (scaling) - Scaling obrázku se provádí především ke snížení výpočetní a paměťové náročnosti. Často používaný rozměr po zmenšení je 224x224 pixelů. Tento rozměr je použit i v této práci.

Normalizace hodnot - jedná se o běžně prováděný krok u všech typů neuronových sítí. Zajišťuje převod vstupních hodnot (většinou v rozsahu 0-255) na jiný, vhodnější rozsah. Umožňuje rychlejší zpracování NN, normalizovaná data jsou rovněž vhodnější pro vstup do aktivačních funkcí. Nejčastěji používané rozsahy pro normalizaci jsou $(0; -1)$ a $(-1; 1)$.

Model MobilenetV2 vyžaduje vstupní data v rozsahu (-1,1), což neodpovídá tomu, co modul NPU síti poskytuje (poskytuje data v rozsahu 0-255). Proto bylo zapotřebí u tohoto modelu přidat vlastní vstupní a normalizační vrstvu, která zajistí převod na žádaný rozsah. Model MobilenetV3Small už tuto normalizační vrstvu obsahuje nativně, není tedy potřeba žádných úprav. Preprocessing pro model MobilenetV2 je zobrazen na výpisu 4.

```
img = cv2.imread(IMG_PATH) # Načtení obrázku
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Převod do formátu RGB
img = cv2.resize(img,(224,224)) # Zmenšení obrázku na rozměr 224x224

#Konverze pole obrázků z int na float16
train_images = np.array(train_images, dtype="float16")
#Normalizace pro MobilenetV2
train_images = tf.keras.applications.mobilenet_v2.preprocess_input(train_images)
```

Výpis 4: Kroky preprocessingu pro MobilenetV2

7.1.4 Trénink modelu

Protože je klasifikační úloha použitá v této práci poměrně jednoduchá, neuronová síť dosahovala při tréninku vysoké přesnosti i při malém počtu trénovacích epoch. Jelikož se jedná o multi-class klasifikaci, je vhodné pro vyhodnocování přesnosti modelu používat metriky využívajících hodnot *true/false positive* a *true/false negative* [62]. Klasickou metriku accuracy zde není příliš vhodné používat, protože přesnost je vyhodnocována na základě každého prvku v predikovaném vektoru zvlášť, nikoli na základě vektoru jako celku (to pak může způsobovat zkreslení přesnosti modelu). Pokud by například byl vektor skutečných hodnot $(0, 0, 0, 1, 0, 0, 0)$ a model by predikoval vektor $(1, 0, 0, 0, 0, 0, 0)$ byla by přesnost predikce $5/7 = 71\%$, což vypadá, že model je relativně přesný, i když tomu tak není.

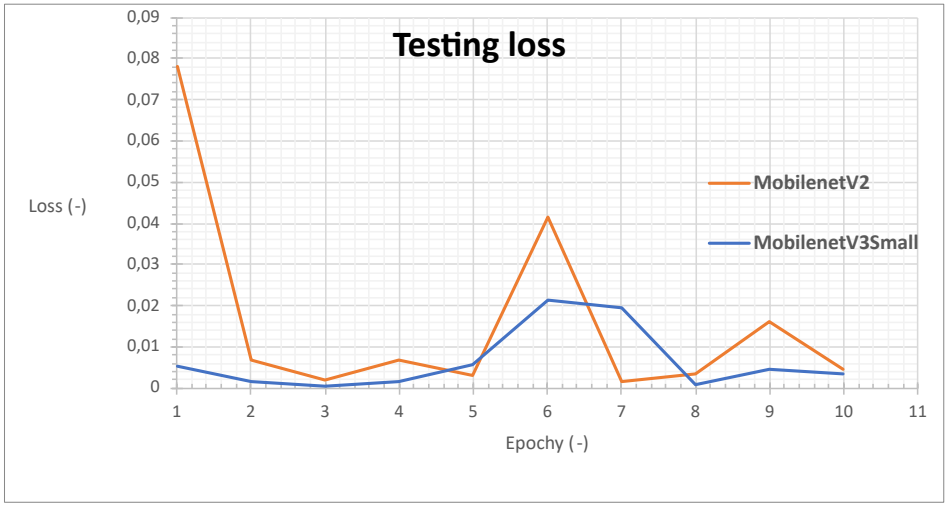
Aby bylo možné vyhodnotit přesnost modelu, je zapotřebí si rozdělit data na trénovací a testovací. Trénovací data jsou použita pro učení sítě, na testovacích datech si model ověřuje svou přesnost. Poměrně často užívaný poměr rozdělení trénovacích a testovacích dat je 80 : 20. Tento poměr jsem použil i v mém případě.

Oba modely MobilenetV2 i MobilenetV3Small si vedly velmi dobře, pro dosažení téměř nulového počtu falešných predikcí stačily jednotky trénovacích epoch. Pro porovnání efektivity tréninků můžeme použít graf průběhu účelové funkce (obrázky 7.4 a 7.5).

Díky jednoduchosti klasifikační úlohy trval trénink vždy pouze pár minut. Tréninků jsem proto provedl desítky, ve většině si model MobilenetV3Small vedl o něco lépe. Níže uvedené grafy jsou tedy pouze orientační, pocházejí z jednoho konkrétního tréninku.

Model MobilenetV2 si při tréninku ve většině případů vedl o něco hůře, hodnoty jeho účelové funkce byly většinou vyšší.

Obrázek 7.4: Průběh účelové funkce pro trénovací data



Obrázek 7.5: Průběh účelové funkce pro testovací data

Jako optimalizační algoritmus je použit ADAM [63], který se mimo jiné často využívá právě pro neuronové sítě určené ke klasifikaci či detekci objektů. Tento algoritmus využívá stochastický gradientní sestup (viz kapitola 2.2.1). Oproti klasickému gradientnímu sestupu se stochastický liší tím, že v jednom kroku nepočítá účelovou funkci ze všech bodů v datasetu, ale pouze z náhodně zvolené skupiny datových bodů, kterým se říká mini-batch. Díky tomu je proces gradientního sestupu podstatně rychlejší. [64]

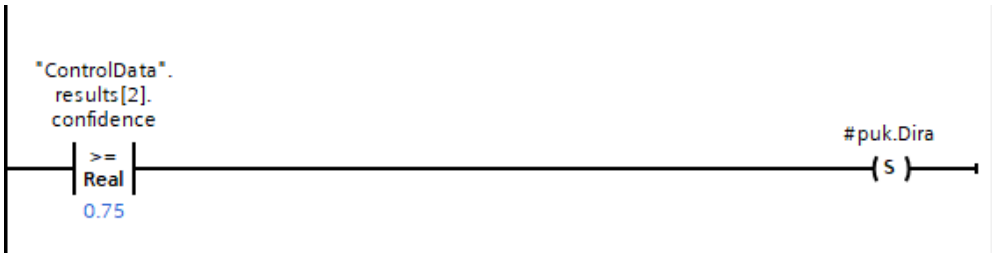
Použitá účelová funkce je zde binární křížová entropie (binary cross-entropy), která je pro multi-label klasifikaci nejběžnější.

7.2 PLC program

Pro řízení demonstrační úlohy je využito PLC S7-1500 1516-3 PN/DP značky Siemens. K němu jsou ještě připojeny celkem 3 moduly, z nichž jeden je samotné TM-NPU. Zbylé dva poskytují digitální vstupy a výstupy pro řízení úlohy. K vizualizaci průběhu úlohy je využito HMI TP700 Comfort, rovněž od Siemensu.

Běh celé úlohy se zahajuje zapnutím pohonu karuselu, pomocí proměnné *Enable*. Když se karusel točí, jeho válečky na spodní straně jsou detekovány indukčním senzorem, což způsobí aktivaci FB *ObjectDetection* a následnou inferenci na modulu. Klasifikační data z inference jsou pak zapsána do pole *results*, nacházející se v datovém bloku (DB) *ControlData* (viz kapitola 6.4). Klasifikační data obsahují pravděpodobnosti výskytu tříd na obrázku. Každý prvek pole tedy představuje pravděpodobnost výskytu dané třídy.

U každého prvku (u každé třídy) se dále zjišťuje, zda-li pravděpodobnost jeho výskytu je dostatečná, konkrétně větší než 75%. Ověřování pravděpodobnosti výskytu pro vlastnost "Dira"lze vidět na obrázku 7.6



Obrázek 7.6: Ověření pravděpodobnosti výskytu vlastnosti puků

Pokud ano je daná třída zapsána do vlastností temporary proměnné *#puk*. Tato proměnná je datového typu *Puk_properties* což je můj vlastní datový typ pro uložení vlastností puku. Jeho struktura je na obrázku 7.7. Vlastnost *Barva* je datového typu USInt, aby do ní bylo možné uložit tři různé hodnoty: 0 - červená, 1 - modrá, 2 - barva nebyla inicializována (je to výchozí hodnota).

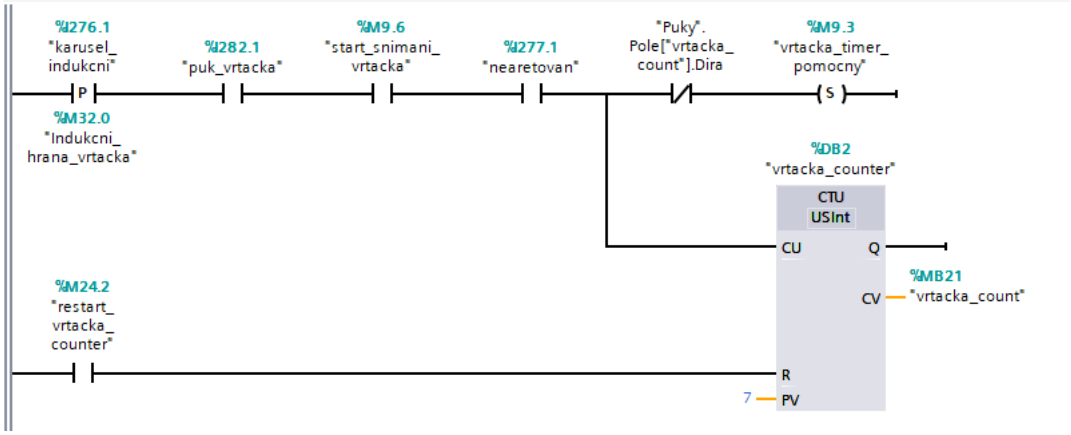
	Temp	
	puk	"Puk_properties"
	Barva	USInt
	Dira	Bool
	Znacka	Bool
	Mimo	Bool
	Spinavy	Bool
	Prazdny	Bool

Obrázek 7.7: Temporary proměnná #puk

Po vyhodnocení pravděpodobnosti výskytu všech tříd je pomocí instrukce MOVE proměnná *#puk* zapsána do pole. Je tak učiněno pouze v případě, že nebyl detekován prázdný výřez. Do pole se zapisuje i v případě detekce vychýlení puku, avšak zde je navíc zastaven karusel. Toto pole slouží k ukládání stavů všech načtených puků. Jeho velikost odpovídá počtu výřezů v karuselu a tedy i maximálnímu možnému počtu puků na něm umístěných.

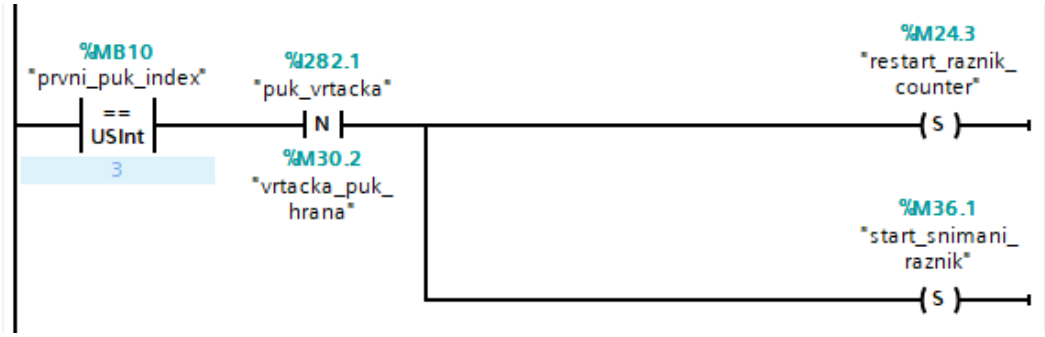
Prvky z pole jsou použity při rozhodování o aktivaci simulačních nástrojů. Každý nástroj má svůj vlastní CTU counter, který je inkrementován vždy, když je pod nástrojem detekován puk. Hodnota těchto counterů (CV value) pak slouží jako index při čtení pole s puky. Pokud tedy například byly detekovány dva puky pod vrtačkou, u dalšího puku se bude číst z pole na indexu 2. Z pole se tedy načtou vlastnosti puku na daném indexu a na základě pravidel uvedených v kapitole 7 se rozhodne o aktivaci nástroje.

Pro indexaci pole s puky je využito nepřímého adresování. Příklad inkrementace counteru a ověření pravidla pro aktivaci nástroje je uvedeno na obrázku 7.8. Ověření pravidla je zde formou negace vlastnosti puku "Dira". Je-li tato podmínka splněna (tedy puk nemá díru) je aktivována pomocná proměnná *vrtacka_pomocna_timer*, která později zahájí sekvenci vrtání.



Obrázek 7.8: Inkrementace counteru a nepřímé adresování pole s puky

Na uvedeném obrázku lze rovněž vidět pomocnou proměnnou *start_snimani_vrtacka*. Ta slouží k tomu, aby se nástroj aktivoval až když pod nástrojem projede první naskenovaný puk. Karusel totiž může být zcela zaplněn puky a není žádoucí aby countery nástrojů reagovaly na nenaskenované puky. Podobná proměnná je použita u všech nástrojů. K aktivaci těchto proměnných dochází na základě hodnoty counteru *indukcni_counter*, který je inkrementován při náběžné hraně z indukčního senzoru. Na obrázku 7.9 je zobrazena aktivace této pomocné proměnné pro razník. Hodnota counteru je uložena v proměnné *prvni_puk_index*. Counter je restartován při dosažení hodnoty 8, což indikuje, že karusel se otočil o celých 360°.



Obrázek 7.9: Aktivace pomocné proměnné pro start snímání nástroje

V případě razníku je start snímání zahájen, když je *prvni_puk_index* roven třem. To odpovídá situaci, kdy je první naskenovaný puk jeden výřez před razníkem. Na výše uvedeném obrázku lze rovněž vidět restartování counteru daného nástroje, což zajišťuje, že úloha bude správně fungovat cyklicky.

Kapitola 8

Závěr

V teoretické části této práce bylo popsáno základní rozdělení a principy metod umělé inteligence. Detailněji pak byla popsána operace konvoluce a z ní vycházející konvoluční neuronové sítě, jenž jsou v této práci využívány. Dále v teoretické části je analýza možností využití umělé inteligence v systémech s PLC. Jsou zde zmíněny metody jako preventivní údržba, strojové vidění či fuzzy řízení. Kromě rozboru těchto metod je uvedeno i několik příkladů s praxe (dynamický úchop čelistí robota, fuzzy řízení robotického ramena). Jako další je v práci popsán modul TM-NPU, který slouží ke zpracovávání neuronových sítí a je hlavním elementem v praktické části práce.

V praktické části byla navrhnuta demonstrační úloha pro klasifikaci stavů dřevěných puků, na základě snímků pořízených kamerou. Klasifikaci zajišťuje konvoluční neuronová síť založena na modelu MobilenetV3Small. Trénink sítě probíhal v prostředí pythonu za využití knihovny Keras. Pro trénink bylo shromážděno přes čtyři tisíce obrázků, zachycujících různé stavy puků.
Vytrénovaná síť byla z pythonu exportována do formátu .h5 a následně převedena do formátu .blob, který je vyžadován modulem NPU. Konverze do tohoto formátu byla provedena pomocí dedikovaného software AI Model Deployer. Model v tomto formátu byl přesunut na SD kartu modulu, kde byl připraven k použití. Pro pořizování obrázků byla k modulu připojena kamera Intel RealSense D435.
Pro výměnu dat mezi PLC a modulem byl využit speciální funkční blok "Object detection", který byl poskytnut společností Siemens jakou součást aplikačního příkladu. Výstupem tohoto bloku je pole pravděpodobností výskytu každé klasifikační třídy na pořízeném obrázku. Na základě těchto pravděpodobností je dále v programu rozhodováno, zda-li bude aktivován některý z nástrojů pro "obrábění"puku.

Literatura

1. SHAHID, Amna; MUSHTAQ, Malaika. A Survey Comparing Specialized Hardware And Evolution In TPUs For Neural Networks. In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*. 2020-11, s. 1–6. Dostupné z DOI: 10.1109/INMIC50486.2020.9318136. ISSN: 2049-3630.
2. *What is Deep Learning?* [online]. 2022-03. [cit. 2022-11-04]. Dostupné z: https://www.ibm.com/cloud/learn/deep-learning.
3. ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017-08, s. 1–6. Dostupné z DOI: 10.1109/ICEngTechnol.2017.8308186.
4. RUSSELL, Stuart J.; NORVIG, Peter; DAVIS, Ernest. *Artificial intelligence: a modern approach*. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0-13-604259-4.
5. LEARNED-MILLER, Erik G. Introduction to supervised learning. *I: Department of Computer Science, University of Massachusetts*. 2014, s. 3.
6. *What is Unsupervised Learning?* [online]. 2022-03. [cit. 2022-11-12]. Dostupné z: https://www.ibm.com/cloud/learn/unsupervised-learning.
7. KODINARIYA, Trupti M; MAKWANA, Prashant R. Review on determining number of Cluster in K-Means Clustering. *International Journal*. 2013, roč. 1, č. 6, s. 90–95.
8. STATQUEST WITH JOSH STARMER. *StatQuest: K-means clustering* [online]. 2018. [cit. 2022-11-12]. Dostupné z: https://www.youtube.com/watch?v=4b5d3muPQmA.
9. PAGE, Justin; LIECHTY, Zach; HUYNH, Mark; UDALL, Joshua. BamBam: Genome sequence analysis tools for biologists. *BMC research notes*. 2014-11, roč. 7, s. 829. Dostupné z DOI: 10.1186/1756-0500-7-829.
10. SUTTON, Richard S; BARTO, Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.
11. BHATT, Shweta. *Reinforcement Learning 101* [online]. 2019-04. [cit. 2022-11-22].

12. *Hidden Layer* [online]. 2019-05. [cit. 2022-11-04]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning.

13. SHARMA, Sagar; SHARMA, Simone; ATHAIYA, Anidhya. Activation functions in neural networks. *towards data science*. 2017, roč. 6, č. 12, s. 310–316.

14. *Difference between a Neural Network and a Deep Learning System* [online]. 2022-02. [cit. 2022-11-04]. Dostupné z: https://www.geeksforgeeks.org/difference-between-a-neural-network-and-a-deep-learning-system/. Section: Difference Between.

15. KOSTADINOV, Simeon. *Understanding Backpropagation Algorithm* [online]. 2019-08. [cit. 2023-02-11]. Dostupné z: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

16. BROWNLEE, Jason. *A Gentle Introduction to Transfer Learning for Deep Learning* [online]. 2017-12. [cit. 2023-03-23]. Dostupné z: https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

17. NEURONOVESITE_CZ. *Klasifikace obrázků pomocí neuronových sítí.* [online]. 2021. [cit. 2022-11-04]. Dostupné z: https://www.youtube.com/watch?v=9U-RQQJmE6E.

18. STATQUEST WITH JOSH STARMER. *Neural Networks Part 8: Image Classification with Convolutional Neural Networks (CNNs)* [online]. 2021. [cit. 2022-11-04]. Dostupné z: https://www.youtube.com/watch?v=HGwBXDKFk9I.

19. REYNOLDS, Anh H. *Anh H. Reynolds* [online]. [B.r.]. [cit. 2023-02-11]. Dostupné z: https://anhreynolds.com/blogs/cnn.html.

20. *CNN | Introduction to Pooling Layer* [online]. 2019-08. [cit. 2022-11-04]. Dostupné z: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/. Section: Machine Learning.

21. SINGH, Pravendra; RAJ, Prem; NAMBOODIRI, Vinay P. EDS pooling layer. *Image and Vision Computing* [online]. 2020-06, roč. 98, s. 103923 [cit. 2022-11-04]. ISSN 02628856. Dostupné z DOI: 10.1016/j.imavis.2020.103923.

22. WEISSTEIN, Eric W. *Convolution* [online]. [B.r.]. [cit. 2022-11-28]. Dostupné z: https://mathworld.wolfram.com/Convolution.html. Publisher: Wolfram Research, Inc.

23. ALLAIN, Rhett. *Calculating the Convolution of Two Functions With Python* [online]. 2020-08. [cit. 2022-11-26]. Dostupné z: https://medium.com/swlh/calculating-the-convolution-of-two-functions-with-python-8944e56f5664.

24. 3BLUE1BROWN. *But what is a convolution?* [B.r.]. Dostupné také z: https://www.youtube.com/watch?v=KuXjwB4LzSA.

25. KIM, Sung; CASPER, Riley. Applications of convolution in image processing with MATLAB. *University of Washington*. 2013, s. 1–20.

26. BRUCHANOV, Martin. *Diskrétní 2D konvoluce*. Praha, 2006. Dostupné také z: https://bruxy.regnet.cz/fel/36ACS/konvoluce.pdf.

27. SINHA, Utkarsh. *Convolutions: Image convolution examples - AI Shack* [online]. [B.r.]. [cit. 2022-11-28]. Dostupné z: https://aishack.in/tutorials/image-convolution-examples/.

28. HATTORI, Hitoshi; SAKAGAMI, Masanori; OGOU, Mikoto; DEBUN, Takuya. *Easy-to-Use AI-enabled Recorders and PLCs*. 2020. Tech. zpr. Yokogawa Technical Report English Edition.

29. HELEBRANT, František. Technická diagnostika a spolehlivost–IV. Provoz a údržba stroj. *VŠB-TU Ostrava*. 2008, roč. 130, s. 978–80.

30. MOBLEY, R Keith. *An introduction to predictive maintenance*. Elsevier, 2002.

31. ČSN IEC, 60050-192. *Střední doba provozu do poruchy MTTF*. 2016-03. Standard. International Electrotechnical Commission.

32. PAOLANTI, Marina; ROMEO, Luca; FELICETTI, Andrea; MANCINI, Adriano; FRONTONI, Emanuele; LONCARSKI, Jelena. Machine Learning approach for Predictive Maintenance in Industry 4.0. In: *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*. 2018, s. 1–6. Dostupné z DOI: 10.1109/MESA.2018.8449150.

33. KANANESAN, Kavina. *Basic Principles of Preventive Maintenance* [online]. 2019-03. [cit. 2023-01-26]. Dostupné z: https://www.nrx.com/principles-preventive-maintenance/. Section: Blog.

34. Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges. [B.r.], roč. 123. ISSN 0166-3615. Dostupné z DOI: https://doi.org/10.1016/j.compind.2020.103298.

35. JAN, Lukány. *Aplikace technik umělé inteligence v prediktivnı údržbě*. 2020. Dipl. pr. České vysoké učenı technické v Praze. Vypočetnı a informačnı centrum.

36. JOSHI, Bhargav. *Application Of Neural Network On PLC-based Automation Systems For Better Fault Tolerance And Error Detection*. 2019. Dis. pr. Auburn University.

37. PÉREZ, Isaías González; GODOY, A. José Calderón; GODOY, Manuel Calderón. Fuzzy controller based on PLC S7-1200: Application to a servomotor. In: *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. 2014, sv. 01, s. 156–163.

38. O'BRIEN, Amy. FLIHI: Fuzzy Logic Implemented HIll-based muscle model. 2022-11.

39. KAZÁK, Josef; MATOUŠEK, Václav. Fuzzy systémy, fuzzy logika. [B.r.], s. 42.

40. UZUN OZSAHIN, Dilber; UZUN, Berna; OZSAHIN, Ilker; MUSTAPHA, Mubarak Taiwo; MUSA, Musa Sani. Chapter 6 - Fuzzy logic in medicine. In: ZGALLAI, Walid (ed.). *Biomedical Signal Processing and Artificial Intelligence in Healthcare*. Academic Press, 2020, s. 153–182. Developments in Biomedical Engineering and Bioelectronics. ISSN 25897527. Dostupné z DOI: https://doi.org/10.1016/B978-0-12-818946-7.00006-8.

41. CODECRUCKS. *Mamdani Fuzzy Inference System - Concept* [online]. 2021-08. [cit. 2022-12-11]. Dostupné z: https://codecrucks.com/mamdani-fuzzy-inference-system-concept/.

42. *What Is Machine Vision?* [online]. [B.r.]. [cit. 2022-11-29]. Dostupné z: https://www.intel.com/content/www/us/en/manufacturing/what-is-machine-vision.html.

43. HAVLE, Otto. *Časopis Automa Strojové vidění I: Principy a charakteristiky* [online]. [B.r.]. [cit. 2022-11-29]. Dostupné z: https://automa.cz/cz/casopis-clanky/strojove-videni-i-principy-a-charakteristiky-2008_01_36550_5518/.

44. *CCD vs. CMOS - srovnání senzorů* [online]. [B.r.]. [cit. 2022-12-07]. Dostupné z: https://www.w-technika.cz/ccd-vs-cmos-srovnani-senzoru/?&force_sid=h7oves5oqhi6kampmgpc2i3cg2.

45. *Key differences between CCD and CMOS imaging sensors* [online]. [B.r.]. [cit. 2022-12-07]. Dostupné z: https://www.flir.eu/support-center/iis/machine-vision/knowledge-base/key-differences-between-ccd-and-cmos-imaging-sensors/.

46. OPT TELESCOPES. *Monochrome vs Color Cameras* [online]. 2021. [cit. 2023-02-28]. Dostupné z: https://www.youtube.com/watch?v=CTlOB7n3VBk.

47. *Image Classification in Machine Learning [Intro + Tutorial]* [online]. [B.r.]. [cit. 2022-12-09]. Dostupné z: https://www.v7labs.com/blog/image-classification-guide,%20https://www.v7labs.com/blog/image-classification-guide.

48. NEURONOVESITE_CZ. *Úvod do detekce objektů v obraze (RCNN, Fast RCNN, Faster RCNN, YOLO)* [online]. 2021. [cit. 2023-01-24]. Dostupné z: https://www.youtube.com/watch?v=gkYeIpeAbcY.

49. GANDHI, Rohith. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms* [online]. 2018. [cit. 2022-12-09]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

50. *YOLO : You Only Look Once - Real Time Object Detection* [online]. 2020-06. [cit. 2022-12-09]. Dostupné z: https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/.

51. HUI, Jonathan. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3* [online]. 2022-09. [cit. 2022-12-09]. Dostupné z: https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

52. PAZDERKA, PRÁCE Bc RADEK. SEGMENTACE OBRAZOVÝCH DAT POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍT. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2018, roč. 40, č. 6, s. 1480–1493.

53. *Panoptic Segmentation: Definition, Datasets & Tutorial [2022]* [online]. [B.r.]. [cit. 2022-12-09]. Dostupné z: `https://www.v7labs.com/blog/panoptic-segmentation-guide,%20https://www.v7labs.com/blog/panoptic-segmentation-guide`.

54. TOMAS, Skulavik; MICHAL, Kopcek; ALENA, Kopeekova. Fuzzy control of robotic arm implemented in PLC. In: *2013 IEEE 9th International Conference on Computational Cybernetics (ICCC)*. 2013, s. 45–49. Dostupné z DOI: `10.1109/ICCCyb.2013.6617628`.

55. SOLOWJOW, Eugen; UGALDE, Ines; SHAHAPURKAR, Yash; APARICIO, Juan; MAHLER, Jeff; SATISH, Vishal; GOLDBERG, Ken; CLAUSSEN, Heiko. Industrial Robot Grasping with Deep Learning using a Programmable Logic Controller (PLC). In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, s. 97–103. Dostupné z DOI: `10.1109/CASE48305.2020.9216902`.

56. COLANER, Seth; PUBLISHED, Matthew Humrick. *A Third Type Of Processor For VR/AR: Movidius' Myriad 2 VPU* [online]. 2016-01. [cit. 2022-11-29]. Dostupné z: `https://www.tomshardware.com/news/movidiud-myriad2-vpu-vision-processing-vr,30850.html`.

57. SIEMENS. *Siemens představuje možnosti umělé inteligence pro PLC Simatic* [online]. [B.r.]. [cit. 2022-11-30]. Dostupné z: `https://www.siemenspress.cz/siemens-predstavuje-moznosti-umele-inteligence-pro-plc-simatic/`.

58. *MicroPython - Python for microcontrollers* [online]. [B.r.]. [cit. 2023-03-10]. Dostupné z: `http://micropython.org/`.

59. *Technology module TM-NPU*. 2021. Dostupné také z: `https://cache.industry.siemens.com/dl/files/877/109765877/att_979771/v1/S71500_tm_npu_manual_en-US_en-US.pdf/`.

60. *Papers with Code - ImageNet Dataset* [online]. [B.r.]. [cit. 2023-03-15]. Dostupné z: `https://paperswithcode.com/dataset/imagenet`.

61. *CUDA Deep Neural Network* [online]. 2014-09. [cit. 2023-01-09]. Dostupné z: `https://developer.nvidia.com/cudnn`.

62. GRANDINI, Margherita; BAGLI, Enrico; VISANI, Giorgio. *Metrics for Multi-Class Classification: an Overview*. 2020. Dostupné z arXiv: `2008.05756 [stat.ML]`.

63. TEAM, Keras. *Keras documentation: Adam* [online]. [B.r.]. [cit. 2023-03-20]. Dostupné z: `https://keras.io/api/optimizers/adam/`.

64. SRINIVASAN, Aishwarya V. *Stochastic Gradient Descent — Clearly Explained !!* [online]. 2019-09. [cit. 2023-03-20]. Dostupné z: `https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31`.

A Seznam příloh

Součástí práce jsou níže uvedené přílohy, které jsou nahrány do systému Edison.

- **Model_training.ipynb** - zdrojový kód pro trénování modelu v prostředí Jupyter notebook
- **Clasification_model.h5** - vytrénovaný klasifikační model
- **TIA_Project.zip** - TIA Portal projekt, zahrnuje kód pro PLC a HMI
- **S71500_tm_npu_manual_en-US_P1.3.pdf** - technická dokumentace k modulu TM-NPU
- **TMNPU_P1.3-ApplicationExampleGettingStarted** - návod pro vytvoření aplikace s TM-NPU