TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta strojní

Návrh AI modelu pro automatickou detekci vadných výrobků pomocí metod hlubokého učení

Bakalářská práce

Studijní program: B2301 Strojní inženýrství
Studijní obor: Strojní inženýrství

Autor práce: **Petr Šíma**
Vedoucí práce: Ing. Iaroslav Kovalenko, Ph.D.
Katedra výrobních systémů a automatizace

Konzultant práce: Ing. Maryna Garan, Ph.D.
Katedra výrobních systémů a automatizace

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta strojní

Zadání bakalářské práce

Návrh AI modelu pro automatickou detekci vadných výrobků pomocí metod hlubokého učení

Jméno a příjmení: **Petr Šíma**
Osobní číslo: S18000111
Studijní program: B2301 Strojní inženýrství
Studijní obor: Strojní inženýrství
Zadávající katedra: Katedra výrobních systémů a automatizace
Akademický rok: **2021/2022**

Zásady pro vypracování:

Cílem bakalářské práce je návrh modelu umělé inteligence, který bude schopen posuzovat vadnost výrobku na základě analýzy jeho fotografie.

1. Rešerše aktuálního stavu problematiky.
2. Návrh struktury modelu.
3. Učení modelu pro zvolený typ výrobku.
4. Ladění modelu pro zvýšení jeho přesnosti.
5. Zhodnocení a závěr.

Rozsah grafických prací:	dle potřeby
Rozsah pracovní zprávy:	cca 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] ANSARI, Shamshad. *Building computer vision applications using artificial neural networks: with step-by-step examples in OpenCV and TensorFLow with Python*. [New York]: Apress, [2020]. For professionals by professionals. ISBN 978-1-4842-5886-6.
[2] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, TensorFlow*. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora. ISBN 978-80-247-3100-1.
[3] HOWARD, Jeremy a Sylvain GUGGER. *Deep learning for coders with fastai and PyTorch: AI applications without a PhD*. Sebastopol, California: O'Reilly Media, [2020]. ISBN 978-1-4920-4552-6.

Vedoucí práce:	Ing. Iaroslav Kovalenko, Ph.D. Katedra výrobních systémů a automatizace
Konzultant práce:	Ing. Maryna Garan, Ph.D. Katedra výrobních systémů a automatizace
Datum zadání práce:	15. listopadu 2021
Předpokládaný termín odevzdání:	15. května 2023

L.S.

prof. Dr. Ing. Petr Lenfeld
děkan

Ing. Petr Zelený, Ph.D.
vedoucí katedry

V Liberci dne 15. listopadu 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

16. května 2022 Petr Šíma

Návrh AI modelu pro automatickou detekci vadných výrobků pomocí metod hlubokého učení

Anotace

Hlavní cíl této práce spočívá v návrhu modelu umělé inteligence, schopného vyhodnotit vadnost výrobku na základě analýzy jeho fotografie. Teoretická část se proto zabývá problematikou automatizace kontroly kvality a stručně popisuje současný stav poznání v této oblasti. Dále vymezuje základní pojmy z oblasti strojového vidění a umělé inteligence, skrze které se dostává k popisu strojového a hlubokého učení. Následně se zabývá popisem neuronových sítí, procesem jejich učení a typy uspořádání sítí použitých v praktické části práce. Úvodem praktické části je stručný popis využitého softwaru a hardwaru. Následuje popis zvolených vstupních dat, způsob jejich úpravy a optimalizace pro proces trénování a testování. Dále je uveden první vytvořený model, který byl podnětem a dal za vznik dalším modelům v rámci průběžného vyhodnocování a zvyšování přesnosti detekce vad. Závěrem práce je celkové porovnání navržených modelů a zhodnocení dosažených výsledků.

Klíčová slova: umělá inteligence, strojové vidění, hluboké učení, klasifikace obrazu, konvoluční neuronové sítě, detekce anomalií, autoenkodér

Design of AI model for the automatic identification of defective products using deep learning

Annotation

The main objective of this work is to design an artificial intelligence model capable of evaluating the defectiveness of a product based on the analysis of its photograph. Therefore, the theoretical part initially deals with the issue of automation of quality control and briefly describes the current state of knowledge in this field. It then defines the basic concepts of machine vision and artificial intelligence, through which this part of the thesis arrives at a description of the machine and deep learning. Then there is a description of neural networks, the learning process, and the types of network structures used in this work. The practical part starts with a brief description of the software and hardware used. The following section is devoted to describing the selected input data and how it is modified and optimized for the training and testing process. Subsequently, the first developed model was presented, which was the impetus and gave rise to other models within the framework of continuous evaluation and improvement of defect detection accuracy. The thesis concludes with an overall comparison of the proposed models and assessment of the reached results.

Keywords: artificial intelligence, machine vision, deep learning, image classification, convolutional neural networks, anomaly detection, autoencoder

Poděkování

Rád bych poděkoval panu Ing. Iaroslavu Kovalenkovi Ph.D. a paní Ing. Maryně Garanové Ph.D. za poskytnuté konzultace a velice cenné rady při řešení této práce.

Obsah

Seznam zkratek a symbolů

Seznam zkratek

2D	Two Dimensional
Adam	Adaptive moment estimation
AI	Artificial Inteligence
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DAGM	German Association for Pattern Recognition
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
INRIA	Inria Aerial Image Labeling Dataset
MSE	Mean Squared Error
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
TN	True Negative
TP	True Positive
VGG	Visual Geometry Group
vRAM	video RAM

Seznam symbolů

α	Rychlost učení (viz rovnice 2.5.3 na stránce 26)
$\hat{Y}$	Hodnota predikovaná modelem (viz rovnice 2.5.1 na stránce 25)
σ	Aktivační funkce (viz rovnice 2.4.1 na stránce 21)
θ	Váha neuronu (viz rovnice 2.4.1 na stránce 21)
b	Práh neuronu (viz rovnice 2.4.1 na stránce 21)
J	Ztrátová funkce (viz rovnice 2.5.1 na stránce 25)
K	Počet kategorií (viz rovnice 2.4.4 na stránce 23)
P	Skutečné rozdělení pravděpodobnosti (viz rovnice 2.5.2 na stránce 25)
Q	Predikované rozdělení pravděpodobnosti (viz rovnice 2.5.2 na stránce 25)
x	Vstupní hodnota neuronu (viz rovnice 2.4.1 na stránce 21)
Y	Skutečná hodnota (viz rovnice 2.5.1 na stránce 25)
y	Výstupní hodnota neuronu (viz rovnice 2.4.1 na stránce 21)
z	Součet vynásobených vstupních hodnot neuronu (viz rovnice 2.4.1 na stránce 21)

1 Úvod

V průběhu posledních dekád je kladen stále větší důraz na zvýšení efektivity výroby. Většina odvětví průmyslu je proto podrobena úpravám ve smyslu automatizace výrobního cyklu. Problém nastává u jednoho z klíčových bodů před expedicí výrobku zákazníkovi, a to sice u kontroly kvality. Automatizace v tomto segmentu bývá často z technického i ekonomického hlediska velmi náročná. Díky velké rozmanitosti možných vad, je taková kontrola často prováděna lidmi, což negativně ovlivňuje objektivitu hodnocení a výsledky kontroly tak nejsou konzistentní.

S touto problematikou se díky vývoji technologie dokáží vypořádat techniky umělé inteligence, konkrétně odvětví strojového učení. Přestože byly základy těchto oborů položeny již v padesátých letech minulého století, své technické využití nachází až v dnešní době. Součástí strojového učení je hluboké učení, které využívá neuronových sítí. Speciálním typem jsou pak konvoluční neuronové sítě, které slaví velký úspěch v oblasti rozpoznávání a klasifikace obrazu. Nejen že technické aplikace těchto metod podávají konzistentní výsledky s vysokou přesností ale nevyžadují velkých investic a proces kontroly je z hlediska snadného nasazení na jiný typ výrobku flexibilní.

Metodám umělé inteligence a jejich využití se věnuje tato práce. Cílem je vytvoření modelu schopného automatické detekce vad výrobku na základě analýzy jeho fotografie. K tomu bylo využito programovacího jazyku Python s pomocí softwarových knihoven Keras a TensorFlow. První vytvořený model, byl natrénován na zvolených vstupních datech, vyhodnocen z hlediska přesnosti a stal se výchozím při tvorbě dalších modelů v procesu ladění. To spočívalo v úpravě samotné struktury modelu ale i způsobu předzpracování vstupních dat. Zhodnocení všech vytvořených modelů se nachází v závěru práce kde nechybí ani způsob možné implementace ve výrobě.

2 Teoretická část

Teoretická část se zabývá současným stavem poznání, na který navazuje popis strojového vidění. Představuje pojem umělé inteligence, zejména pak strojové učení. Dále popisuje hluboké učení s důrazem na neuronové sítě a jejich speciální typy, které byly využity v rámci praktické části této práce.

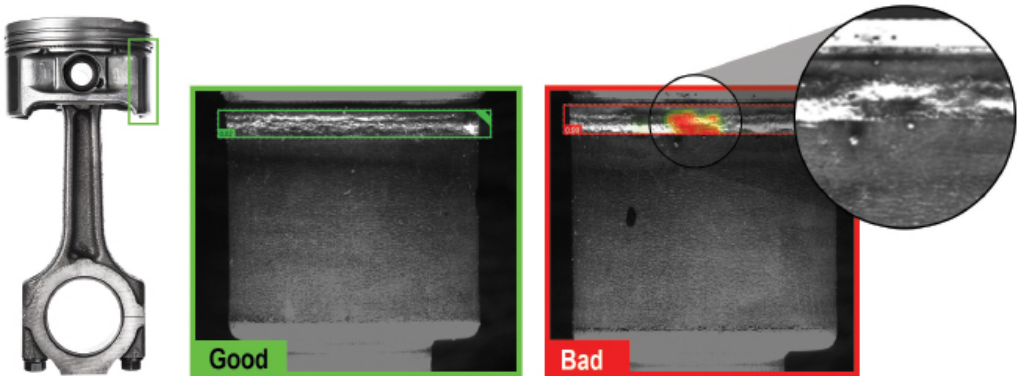
2.1 Současný stav poznání

V době globalizace trhů a rostoucí konkurence nelze z dlouhodobého hlediska uspět s nekvalitním produktem. Nejen že kvalita patří mezi základní požadavky zákazníků, je také rozhodujícím předpokladem úspěšnosti organizace. V mnoha provozech je kontrola kvality výrobků řešena lidskou silou. Na této pozici by měl být člověk zodpovědný a pečlivý. Pokud tomu tak není a dochází k chybám, výrobní společnosti hrozí sankce spojené s reklamacemi nebo poškození jejich dobrého jména. Jedním z problémů způsobených lidským faktorem je objektivita hodnocení. Je třeba rychle a precizně zkontrolovat každý výrobek. Po celou směnu dokáže jeden pracovník konzistentně hodnotit a měřit výrobky jen stěží a v případě kdy se do práce vrhne celá skupina kontrolorů je konzistence výsledků ještě nižší [1, 2].

Řešením problémů spojených s lidským faktorem jsou plně automatizované systémy kontroly kvality. Tyto systémy s přesně definovanými kritérii přináší konzistentní hodnocení výrobků a procesů do kterých lze takovou kontrolu zapojit je nespočet. Vždy by se však mělo jednat o individuálně navržený systém vyhovující požadavkům výrobní společnosti. Právě tyto systémy pomáhají strojírenským firmám uspět v konkurenčním světě zvýšením kvality produkce, celkovým zefektivněním a modernizací provozu. To vede k výraznému šetření firemních financí [3, 4].

Pro případy automatizace kontroly rozměrů jsou systémy počítačového (strojového) vidění s naprogramovanými a jasně danými pravidly stále preferovanou a cenově výhodnou volbou. Pokud se však jedná o složitější úlohu, například kontrolu poškození povrchu, kde škrábance a jiná možná poškození nejsou zcela jednoznačná, byla by potřebná pravidla pro vyhodnocení příliš složitá. Právě pro tyto případy je velice výhodné využití metod hlubokého učení [5].

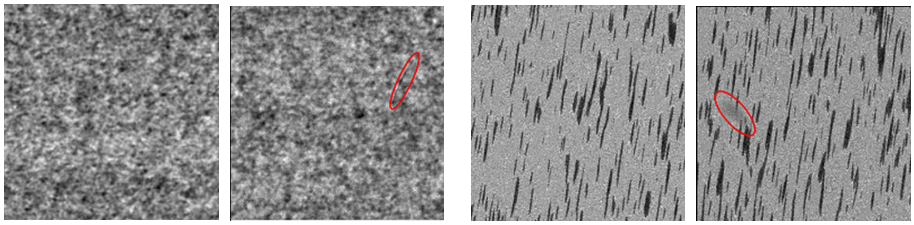
Metody hlubokého učení v posledních letech dosahují oproti klasickým přístupům strojového učení lepších výsledků a to zejména v oblastech zpracování obrazu, počítačového vidění, rozpoznávání řeči, robotiky a řízení procesů. Jedná se právě o oblast zpracování obrazu, ve které tyto metody lze využít ke klasifikaci nebo detekci anomálií. Právě tyto schopnosti nachází své využití v průmyslu. Na následujícím obrázku můžete vidět kontrolu pístové hlavy, kde je metodou detekce anomálií kontrolován svar, jehož kvalita výrazně ovlivňuje výsledný výkon motoru [6].



Obrázek 2.1: Úloha s využitím hlubokého učení [6]

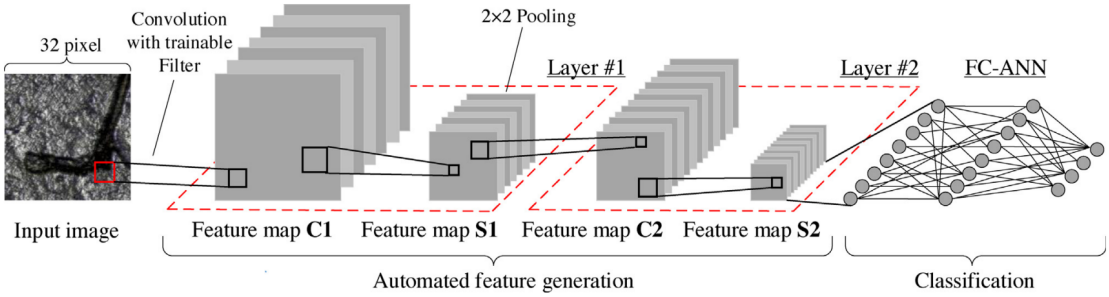
V rámci této práce by bylo vhodné poukázat na model, který proslavil metody hlubokého učení. Jedná se o model AlexNet z roku 2012, který navrhl Alex Krizhevsky pro soutěž ILSVRC-2010 (ImageNet Large Scale Visual Recognition Challenge). V této soutěži se jednalo o klasifikaci 1,2 milionů obrázků do 1000 kategorií. Tento model je velmi hlubokou konvoluční neuronovou sítí, která obsahuje 60 milionů parametrů a 650 tisíc neuronů. Skládá se z 5-ti konvolučních vrstev z nichž jsou některé následovány sdružovací vrstvou MaxPooling. Tyto vrstvy jsou následně přivedeny na plně propojenou síť neuronů plnící rozhodovací funkci. Výstupem je tak vrstva z 1000 neuronů a aktivační funkcí Softmax. Tento model v soutěži ILSVRC-2012, dosáhl správnosti 84% a vyhrál [7].

Kontrolou kvality v oblasti průmyslu se zaobírá odborný článek, jehož autory jsou D. Weimer, B. Scholz-Reiter a M. Shpitalni. V svém jádru pojednává o novém paradigmatu strojového učení, konkrétně o hlubokém učení v podobě modelů konvolučních neuronových sítí. Zkoumá jejich různé konfigurace a dopad na výslednou přesnost detekce vad. Vstupními daty pro učení a testování je soubor DAGM-2007 (Deutsche Arbeitsgemeinschaft für Mustererkennung), který se skládá z šesti datových souborů. Každý z nich obsahuje 1000 obrázků textury bez defektu a 150 obrázků s označenou vadou o rozměrech 512×512 pixelů (obrázek 2.2). Jedná se o uměle vytvořené obrázky inspirované průmyslovým zpracováním obrazu, které jsou si vzájemně velmi podobné. Každý soubor dat je však generován jiným modelem textury a defektu [8].



Obrázek 2.2: Ukázka snímků ze souboru DAGM [9]

Pro klasifikaci snímků do dvanácti kategorií (6 s vadou a 6 bez vady) jsou v této práci využity modely, které obsahují tři vrstvy extrakce map příznaků a dvě vrstvy klasifikátoru. V případě extrakce map příznaků ze vstupního obrázku je využito konvolučních vrstev s rozměrem filtru 3×3. Každá z nich je následována sdružovací vrstvou Max-Pooling, která snižuje rozměry vytvořených map. Ty jsou následně přivedeny do klasifikátoru, který je reprezentován standardní plně propojenou sítí neuronů. Ta se skládá ze dvou skrytých vrstev, kde každá z nich obsahuje 1024 neuronů. Poslední část v podobě výstupní vrstvy obsahuje 12 neuronů s aktivační funkcí Softmax. Takto strukturovanou síť lze vidět na obrázku 2.3. Struktura dosáhla při testování průměrné správnosti 99,2% a ve srovnání s ostatními přístupy byla nejvyšší [8].

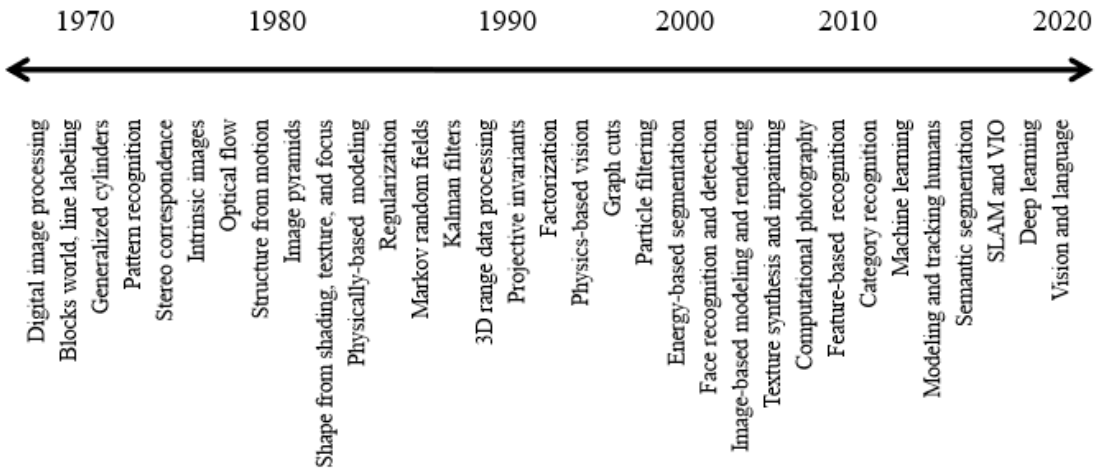


Obrázek 2.3: Model konvoluční neuronvé sítě [8]

Příkladem využití umělé inteligence v průmyslu je firma Audi, která systém založený na metodách hlubokého učení implementovala ve své lisovně v Ingolstadtu. Kontrola vylisovaných plechů probíhá přímo ve výrobní lince pomocí zabudovaných kamer. Snímky z kamer jsou následně analyzovány tak aby systém odhalil i ty nejmenší praskliny. Systém, k jehož naučení bylo využito milionů obrázků pořízených v různých závodech koncernu Volkswagen, dosahuje velké přesnosti [10].

2.2 Strojové vidění

Historie počítačového vidění sahá do 70. let 20. století, kdy bylo cílem vybavit roboty inteligentním chováním a napodobit tak lidskou inteligenci ve smyslu vizuálního vnímání. V té době se řada odborníků domnívala, že řešení problémů s vizuálním vstupem bude snadným úkolem na cestě ke složitějšímu, jako je uvažování a plánování. To, co odlišovalo počítačové vidění od digitálního zpracování obrazu, byla touha obnovit z obrazu trojrozměrnou strukturu a využít ji k plnému porozumění snímané scény. První pokusy zahrnovaly extrakci hran a vytvoření topologické struktury 2D čar. Stručnou časovou osu témat výzkumu v oblasti počítačového vidění lze vidět na obrázku 2.4 [11].



Obrázek 2.4: Časová osa témat výzkumu v oblasti počítačového vidění [11]

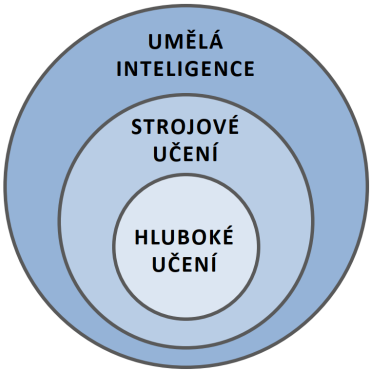
Pod pojem počítačového vidění spadají obecně systémy, které pracují automaticky na základě informací získaných zpracováním obrazu. Za strojové vidění je v současnosti považováno právě využití počítačového vidění v průmyslové automatizaci. Charakteristickou vlastností je vazba na výrobní proces a orientace na typické úlohy spojené s jeho řízením. Mezi tyto úlohy patří hlavně vizuální inspekce předem daných parametrů. Způsob, jakým strojové vidění přistupuje k problému je podobný člověku. Stejně jako lidské oko tak i kamera zachytí obraz zkoumaného objektu. Ve většině případů se jedná o trojrozměrný objekt, který je ozařován tak, aby odražené záření vytvořilo snímacímu prvku kamery jasový dvojrozměrný obraz. Důležité je aby takto získaný obraz obsahoval informace které jsou k jeho vyhodnocení nepostradatelné [12].

Obraz je zpracováván specializovaným softwarem, který analyzuje a vyhodnocuje potřebné charakteristiky pro následné rozhodování a provedení akčního zásahu ve výrobním cyklu jako například vyřazení vadného výrobku. Tyto systémy strojového vidění, s rozhodováním na základě pravidel, jsou spolehlivé v případě konzistentních a shodně vyrobených dílů. K vyhodnocení využívají postupné filtrování a rozhodo-

vací algoritmy. Takto může systém vysoce přesně kontrolovat stovky až tisíce dílů za minutu a je tak ve srovnání s lidskou kontrolou cenově úspornější. Při řešení úlohy rozhodování je pro analýzu obrazu třeba určit a naprogramovat pevně daná a jednoznačná pravidla. V praxi je strojové vidění s vyhodnocením založeným na pravidlech ideální pro navádění, identifikaci čárových kódů a dalších označení, měření rozměrů nebo detekci přítomnosti dílů. Strojové vidění založené na pravidlech je výhodné jsou-li k dispozici množiny potřebných proměnných. Problém však nastává v situacích, kdy tyto hodnoty nejsou tak jednoznačné a naprogramování potřebných pravidel je příliš složité. Příkladem takové úlohy je vyhledávání škrábanců na obrazovkách elektronických zařízení. Takové poškození může mít různou velikost, vzhled a umístění. S ohledem na různorodost je to právě hluboké učení, které tato pravidla dokáže nalézt a určovat tak rozdíly mezi vyhovujícím a nevyhovujícím dílem [5].

2.3 Umělá inteligence

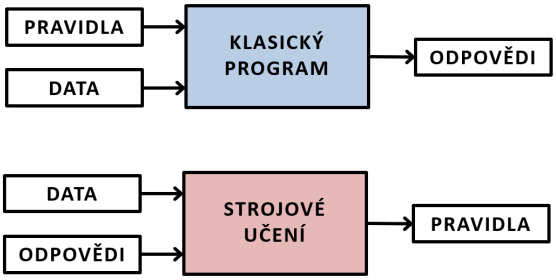
Pojem umělé inteligence (AI) se zrodil již v 50. letech minulého století kdy se hrstka průkopníku začala zabývat otázkou, zda je možné přimět počítače „přemýšlet". Tento obor lze definovat jako snahu o automatizaci úkolů běžně vykonávaných lidmi. Jedná se o rozsáhlý obor, který zahrnuje strojové a hluboké učení (obrázek 2.5) ale také mnoho přístupů, které nezahrnují žádný proces učení. Jako například rané šachové programy schopné oponovat soupeři. Tyto programy zahrnovaly pouze velké množství pevně daných pravidel a nesplňují tak definici strojového učení. Po dlouhou dobu odborníci věřili v možnost dosažení inteligence počítače na úrovni člověka pomocí velkého množství ručně definovaných pravidel. Tento přístup je známý pod pojmem symbolické AI a byl dominantním až do 80. let 20. století před nasazením expertních systémů. Symbolická AI obstála v oblastech řešení dobře definovaných logických problémů, jako je právě hraní šachů. Při řešení komplexních a nejasných úloh jako je rozpoznání řeči nebo klasifikace obrazu se však nalezení takových pravidel stává neřešitelným problémem a namísto symbolické AI tak vznikl přístup strojového učení [13].



Obrázek 2.5: Obor umělé inteligence [13]

2.3.1 Strojové učení

Strojové učení vychází z otázky, je-li počítač schopen překonat námi definované řešení úkolu a přijít se svým vlastním řešením. Dokázal by se automaticky naučit pravidla z pohledu na data, namísto toho, aby tato pravidla vytvářeli programátoři? Tato myšlenka vede k novému pohledu na klasické programování, kde je přístup symbolické AI založen na vkládání pevně daných pravidel společně s daty, ze kterých chceme získat odpovědi. Oproti tomu, je přístup strojového učení odlišný právě tím že do takového systému jsou vkládána data a jim příslušící odpovědi. Výstupem jsou pak právě pravidla pro zpracování těchto dat. (obrázek 2.6). Ty lze následně využít pro zpracování nových dat a získat tak originální odpovědi [13].



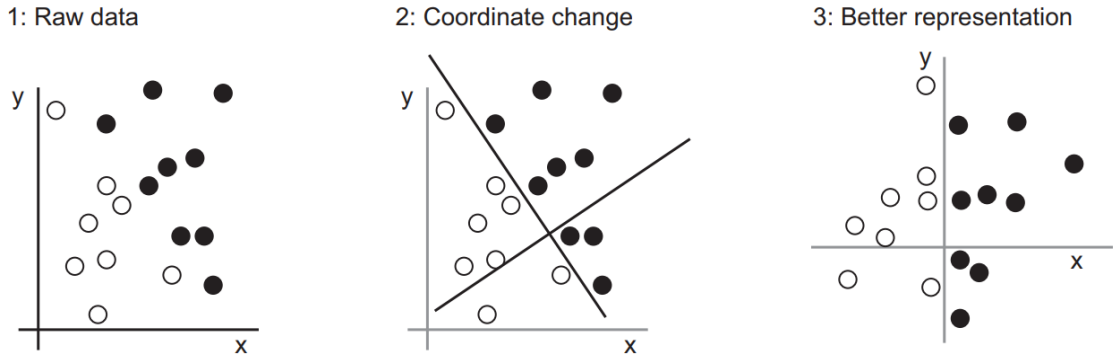
Obrázek 2.6: Přístup strojového učení [13]

Systém strojového učení tedy není explicitně naprogramován ale spíše natrénován. K tomu využívá velkého množství příkladů dat, relevantních pro daný úkol. V těchto datech nachází statistickou strukturu v podobě pravidel, kterou lze využít při automatizovaném zpracování dat nových. Díky tomu je schopen získat originální odpovědi. Takovým příkladem by bylo automatizované přiřazení označení snímkům z dovolené. Již označené snímky by sloužili jako vstup a systém by se z takto označených dat učil statistická pravidla pro přiřazení označení dalším snímkům [13].

I přes fakt že se strojové učení popularizovalo teprve v 90. letech, stalo se rychle oblíbeným a úspěšným podoborem umělé inteligence. Tento trend se odvíjel od dostupnosti výkonnějšího hardwaru a větších souborů dat. Tento podobor úzce souvisí s matematickou statistikou ale v několika ohledech se od ní liší. Tendencí je práce s velkými a komplexními daty. Například soubor milionů obrázků z nichž se každý skládá z desítek tisíc pixelů. Pro takové případy by byl přístup klasické statistické analýzy nepraktický [13].

Aby mohlo takové strojové učení být provedeno ve smyslu získání pravidel, je zapotřebí vstupních dat jako například záznamu řeči nebo pořízených snímků a k nim přiřazené očekávané výstupy ve formě přepisu nebo označení snímku. V neposlední řadě je zapotřebí metrika, kterou lze hodnotit to, zda algoritmus odvádí dobrou práci. Ta slouží jako zpětnovazební signál a upravuje způsob jakým systém pracuje. Tímto způsobem model strojového učení transformuje vstupní data na smysluplné výstupy. Jinými slovy pro dostupná vstupní data hledá vhodnou reprezentaci, která přibližuje očekávané výsledky [13].

Zjednodušeným představitelem této myšlenky je úloha nalezení vhodné reprezentace bodů dvou barev v dvourozměrném prostoru. Cílem je nalezení jednodušších pravidel pro určení kde se nachází body jedné a druhé barvy. Jak lze vidět na obrázku 2.7 jsou pravidla pro rozhodování barvy zjednodušena pomocí transformace jejich reprezentace. Černé body náleží hodnotám x>0 a bílé x<0. Technicky vzato je strojové učení hledáním vhodných reprezentací pro vstupní data v rámci definovaného prostoru možností pomocí zpětné vazby [13].



Obrázek 2.7: Změna reprezentace [13]

2.3.2 Techniky strojového učení

Způsobů jakými přistupují algoritmy strojového učení k řešení problémů je několik. Z hlediska dostupnosti struktury vstupních dat rozpoznáváme dvě základní techniky [14].

- **Učení s učitelem** je přístup využívaný v případě že vstupní data mají vhodnou strukturu a jsou příslušně označena. Pro případ klasifikace se jedná například o soubor fotografií zvířat s přiděleným označením jednotlivých druhů. Takto označená data slouží modelu jako učitel v procesu učení. Pomocí takových dat je model upravován takovým způsobem aby jeho označení na výstupu odpovídalo vstupním datům podle známe struktury. Takto naučený model lze aplikovat na nová neoznačená data, kterým je na základě předchozí zkušenosti schopen přidělovat odpovídající označení [15].
- **Učení bez učitele** oproti učení s učitelem nevyžaduje žádnou strukturu vstupních dat ani označení. Takový model se u neznámých vstupních dat snaží právě tuto strukturu rozdělení dat najít [16].

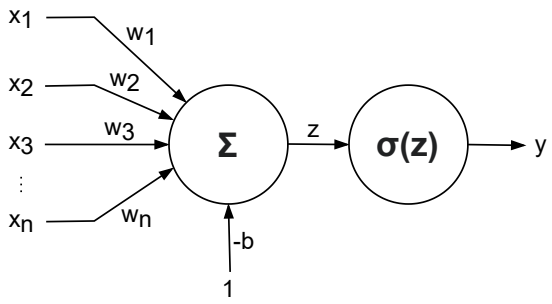
2.3.3 Hluboké učení

Hluboké učení je specifickým podoborem strojového učení. Jedná se o pohled na učení se reprezentace z dat, který klade důraz na učení po sobě jdoucích vrstev stále užitečnějších reprezentací. Hloubka v hlubokém učení je reprezentována počtem těchto vrstev reprezentací modelu. Moderní modely hlubokého učení obsahují desítky nebo dokonce stovky těchto reprezentací, které se učí automaticky v závislosti na trénovacích datech. Jiné přístupy strojového učení se zabývají učením jedné nebo dvou vrstev, někdy se takové učení nazývá mělkým učením. Při hlubokém učení se tyto vrstvené reprezentace dat učí pomocí modelů nazývaných neuronové sítě [13].

2.4 Neuronové sítě

Základní stavební jednotkou neuronových sítí je umělý neuron. První matematický model umělého neuronu, který je dodnes používaný vytvořili roku 1944 Warren McCullough a Walter Pitts [17].

Takový model umělého neuronu lze vidět na obrázku 2.8. Jeho vstupem je vektor hodnot o n prvcích označených $(x_1, x_2, ..., x_n)$, ty nabývají reálných hodnot. Každé takové hodnotě je přidělena příslušná váha θ_i, která je také vektorem hodnot a je parametrem, který prochází úpravou v procesu učení. Zvláštním typem váhy je bias b neboli práh, který umožňuje neuronu modifikovat výstupy nezávisle na jeho vstupech. Z praktického hlediska je práh uvažován za jednu z vah rozšířením vektoru vah a vstupních hodnot o nultou pozici. Poté je vstup této nulté váhy $x_0 = 1$ a jeho příslušná váha má hodnotu $\theta_0 = -b$. Vstupní hodnoty a práh jsou vynásobeny příslušnými vahami a sečteny. Tato hodnota z je transformována pomocí aktivační (přenosové) funkce σ, která je obecně nelineární. Příklady takových funkcí jsou podrobně uvedeny v následující kapitole. Výstupem neuronu je hodnota y, která se v závislosti na použité aktivační funkci nachází v rozmezí $< -1, 1 >$ nebo $< 0, 1 >$ [18, 19].

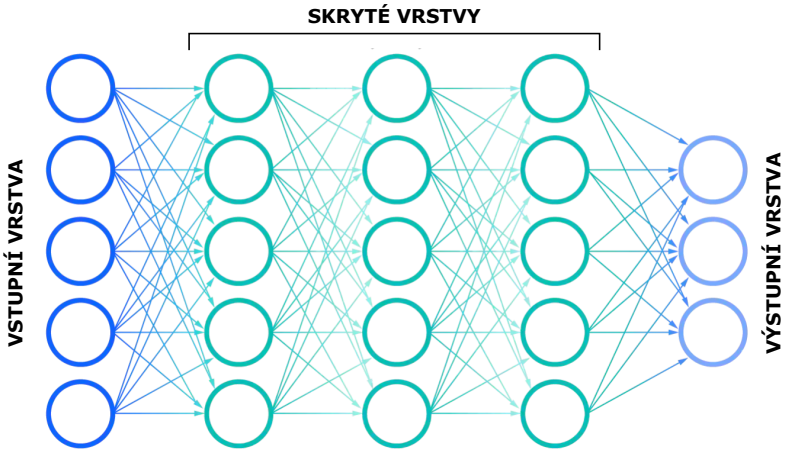


Obrázek 2.8: Model umělého neuronu

Matematický popis umělého neuronu je následující:

$$y = \sigma(z) = \sigma(\sum_{i=0}^{n} x_i \cdot \theta_i) \tag{2.4.1}$$

Takto popsaný neuron nevykonává příliš složitou funkci. Hlubokého učení proto využívají vzájemně propojené neurony uspořádané do sítí. Obecně platí, že taková neuronová síť má vstupní a výstupní vrstvu neuronů a mezi nimi několik skrytých vrstev (obrázek 2.9). Vstupní vrstva přijímá vstupní hodnoty a předává je další vrstvě. V této vrstvě nedochází k žádným výpočtům, jedná se pouze o vrstvu schopnou přijímat vstupní hodnoty. V případě skrytých vrstev se jedná o všechny vrstvy mezi vstupní a výstupní vrstvou. Tyto vrstvy jsou zodpovědné za nelineární transformaci vstupů. Parametry těchto vrstev jako váhy a prahové hodnoty vedené přes aktivační funkci každého neuronu jsou získávány v procesu učení. Každá z těchto vrstev je navržena tak aby produkovala výstupy specifické pro zamýšlený výsledek. Výstupní vrstva je poslední vrstvou sítě a zodpovídá za podobu výstupních hodnot, nebo vektoru hodnot odpovídajících formátu pro daný problém. Právě propojení a schopnost adaptace vah na základě trénovacích dat má široké využití při analýze dat [20, 21].



Obrázek 2.9: Neuronová síť [21]

Pod architekturou neuronových sítí si lze představit způsoby seskupení a spojení neuronů popřípadě jejich vrstev. Většinu těchto způsobů uspořádání však lze rozdělit do dvou následujících kategorií:

- **Dopředné sítě** mají neurony uspořádané do vrstev, kde výstupy každé vrstvy jsou propojeny vždy pouze s následující vrstvou. Jedná se o sítě bez paměti ve smyslu nezávislosti výstupu na předchozím stavu.
- **Rekurentní sítě** neboli zpětnovazební sítě mají oproti dopředným sítím schopnost uchovávat informace o vstupních datech a zohlednit je při vstupu dalších dat. Tyto sítě jsou preferovány v případě zpracování sekvenčních dat jako například časových řad, textu a řeči. Dokáží tvořit hlubší pochopení těchto sekvencí a jejich kontextu z předchozích stavů.

Z hlediska aplikace těchto sítí je celá řada úloh kde neuronové sítě nachází své využití:

- **Klasifikace** se zabývá zařazováním vstupních dat do tříd na základě naučených vlastností na trénovacích datech.
- **Detekce anomálií** využívá právě schopnosti neuronových sítí rozpoznávat určité vzory v datech. Takové sítě mohou být využity k rozpoznání vzorů které se nějakým způsobem liší od těch na kterých byla síť naučena.

2.4.1 Aktivační funkce

Jak již bylo uvedeno aktivační funkce transformuje součet součinů vstupních hodnot a příslušných vah. Při volbě takové funkce je rozhodující účel této transformace. V praxi se jedná o nelineární aktivační funkce, které transformují vstupní hodnotu nelineárním způsobem což zvyšuje funkční kapacitu neuronové sítě. Pro učení sítě pomocí algoritmů zpětné propagace musí být tato funkce také diferencovatelná [22]. Hlavními představiteli aktivačních funkcí používaných při hlubokém učení jsou:

- **ReLU** (Rectified Linear Unit) neboli usměrněná lineární funkce (obrázek 2.10a). Jedná se o nejrozšířenější aktivační funkci v oblasti hlubokého učení od jejího navržení v roce 2010. Úspěch této funkce stojí zejména za rychlostí výpočtů. Výhodou této funkce je že do skrytých vrstev vnáší řídkost jelikož výstupem jsou hodnoty mezi 0 a maximem. Limitujícím je snadnější přeučení, způsobené jevem „umírajícího gradientu". Příčinou je že při dopředném šíření hodnot, některé neurony nejsou aktivovány a nereagují na jakoukoli vstupní hodnotu. Výstupem je nulová hodnota jejíž gradient je také nula a v důsledku toho nedojde k úpravě patřičných vah sítě. Řešením se stala funkce Leaky ReLU (obrázek 2.10b), která je i pro hodnoty menší než nula mírně rostoucí.

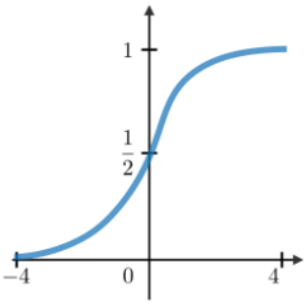
$$\sigma(z) = max(0, z) = \begin{cases} z & \text{pro } z \geq 0 \\ 0 & \text{pro } z < 0 \end{cases} \tag{2.4.2}$$

(a) ReLU[23] (b) Leaky ReLu [23]

Obrázek 2.10

- **Sigmoida** je logistickou funkcí se specifickými parametry. Tato funkce se v případě dopředných sítí pro hluboké učení nachází ve výstupních vrstvách a slouží k vyhodnocení pravděpodobnosti daného jevu. Jedná se o hladkou křivku a její první derivace je rostoucí v celém rozsahu reálných hodnot vstupů, které díky svému předpisu převádí právě do intervalu mezi 0 a 1.

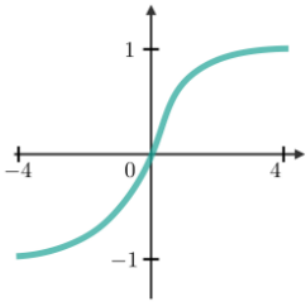
$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4.3}$$



Obrázek 2.11: Sigmoida [23]

- **Hyperbolický tangens**, jehož obor hodnot leží v intervalu od -1 do 1, je obdobou funkce Sigmoid. Je však výkonnější při trénování sítí. Jeho vlastností však je, že aby hodnota gradientu byla rovna 1 musí jeho vstupní hodnota být rovna nule, což opět způsobuje tvorbu "umírajícího gradientu". Funkce nachází své využití hlavně v rekurentních sítích.

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.4.4}$$



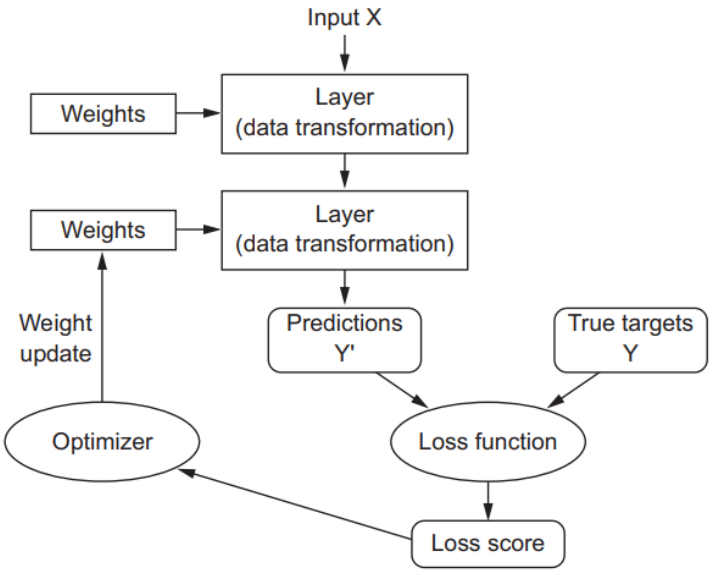
Obrázek 2.12: Hyperbolický tangens [23]

- **Softmax** je funkcí jejíž výstupní hodnoty leží mezi 0 a 1, přičemž jejich součet je roven 1. Tato funkce je často užívaná v případech klasifikace více kategorií K s tím že cílová kategorie má vždy nejvyšší hodnotu pravděpodobnosti.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{pro} \quad i = 1, \ldots, K \tag{2.4.5}$$

2.5 Proces učení neuronové sítě

Hluboké učení spočívá v hledání vhodných reprezentací dat. Tyto reprezentace jsou produktem transformací jednotlivých vrstev sítě. To, jakým způsobem tyto vrstvy transformují vstupní data je dáno jejími vahami, které jsou parametry sítě. V tomto smyslu znamená učení nalezení vhodných parametrů všech vrstev, tak aby síť správně transformovala vzorová vstupní data a přiřazovala jim příslušné výsledky. Právě schopnost se učit, tedy pamatovat si kombinace vah, které vedly k požadovanému výstupu je zkušeností kterou taková síť využívá při odhadu nových výstupů. Jedná se o generalizaci, která spočívá v tom že taková síť dokáže správně zareagovat i na vstupy které nebyly využity pro její učení. Při učení jsou pozorovány rozdíly mezi predikcemi výstupů sítě a skutečnými hodnotami. K tomu slouží ztrátová funkce, která v závislosti na tomto rozdílu vypočítá ztrátové skóre, které pak slouží ve zpětné vazbě, kterou vykonává optimalizátor. Ten je představitelem algoritmu zpětné propagace a udává patřičný způsob toho, jak budou jednotlivé parametry sítě upraveny právě ve smyslu snížení ztrátového skóre. Na úplném začátku takového učení jsou všem vahám přiděleny náhodné hodnoty, které vstup náhodně transformují a výsledek sítě se od skutečnosti velice liší. To v důsledku vede k velmi vysokému ztrátovému skóre. S každým dalším zpracovaným vstupem se však snižuje a to díky úpravě parametrů sítě optimalizačním algoritmem. Opakováním této trénovací smyčky na všech dostupných vstupních datech lze dosáhnout minima ztrátového skóre a tím i přesných predikcí specifických výstupů. Průchod celým souborem dat je nazýván iterací (epochou), jejichž počet je třeba definovat [13, 24].



Obrázek 2.13: Proces učení [13]

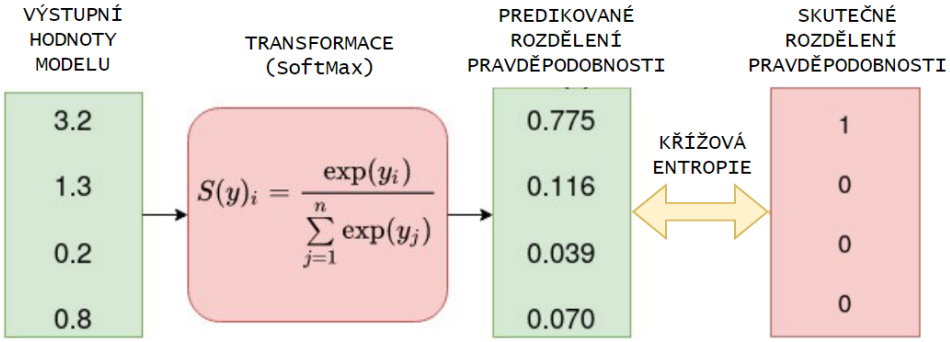
2.5.1 Ztrátová funkce

Pro vyhodnocení schopností modelu je zásadní způsob, jakým je měřena jeho výkonnost. Takové měření v neuronových sítích probíhá pomocí ztrátové funkce $J(\theta)$, která se odvíjí od typu úlohy kterou má daný model vykonávat ale vždy se musí jednat o spojitou diferencovatelnou funkci. Volba správné ztrátové funkce je zásadní pro proces učení a tím i výslednou přesnost modelu [25].

- **Mean Squared Error**, neboli střední kvadratická chyba je ztrátovou funkcí využívanou v případě regresních typů úloh. Tato funkce počítá průměr kvadrátu rozdílů mezi modelem predikovanými ($\hat{Y}_i$) a skutečnými (Y_i) hodnotami, kde predikované hodnoty jsou funkcí vstupních hodnot x_i a příslušných vah θ_i. Výsledek funkce je vždy kladný a v ideálním případě by byl roven nule [26]. Předpis této funkce je následující:

$$J(\theta) = MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.5.1}$$

- **Křížová entropie** je běžně používanou ztrátovou funkcí v případech klasifikace více kategorií K. Princip této funkce spočívá v porovnání predikovaného výstupního rozdělení pravděpodobností Q s jeho skutečným rozdělením P, které je dáno trénovacími daty. Pro určitou kategorii je tato pravděpodobnost rovna 1 a pro ostatní 0. Výstupní hodnoty sítě jsou opět funkcí vstupních hodnot a vah modelu $\hat{Y}_i = f(x_i|\theta)$. Výstupní vrstva modelu tak musí obsahovat aktivační funkci Softmax pro vyhodnocení pravděpodobnosti každé kategorie. Následující obrázek zachycuje tento princip při klasifikaci jedné ze 4 kategorií [27].



Obrázek 2.14: Porovnání rozdělení pravděpodobností [28]

Předpis této funkce pro k kategorií je následující:

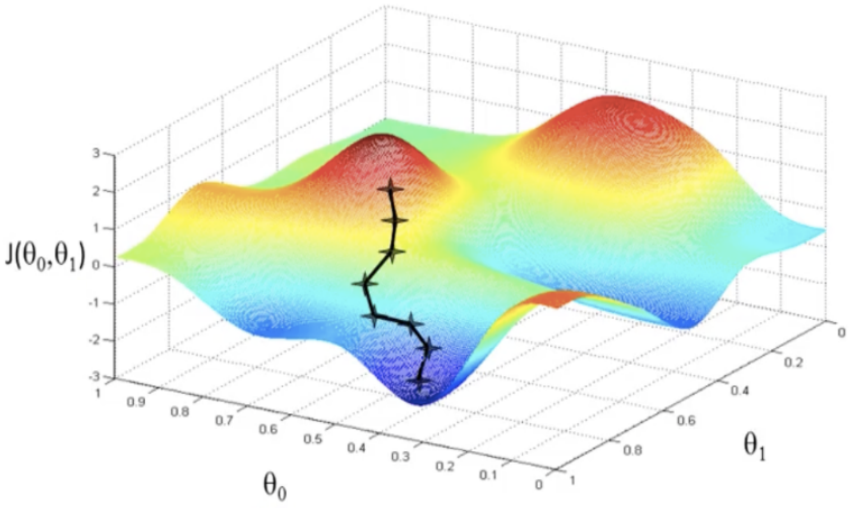
$$J(\theta) = CrossEntropy(P, Q) = -\sum_{i=1}^{k} P(Y_i) \cdot logQ(\hat{Y}_i) \tag{2.5.2}$$

2.5.2 Optimalizace parametrů

Jedná se o optimalizační algoritmus, který v průběhu učení hledá kombinaci jednotlivých vah, tak aby minimalizoval hodnotu ztrátového skóre. Počet parametrů, které je v procesu učení zapotřebí nalézt se odvíjí od architektury sítě. Může se jednat o tisíce či miliony parametrů. K samotnému výpočtu hodnot ztrátového skóre je využito dopředného šíření informací neuronovou sítí s uložením použitých vah, které podléhají úpravě po každé iteraci optimalizačního výpočtu pomocí algoritmu zpětné propagace. Způsobů nalezení těchto optimálních parametrů sítě je několik. Hlavním představitelé těchto metod jsou však založeny na gradientních metodách [29].

- **Gradientní sestup** je nejrozšířenější a velice populární metodou při optimalizaci parametrů neuronových sítí. K nalezení minima ztrátového skóre využívá gradientu ztrátové funkce $\nabla J(\theta)$. Produktem této operace je vektor odpovídající směru nejstrmějšího růstu funkce pro dané parametry. K nalezení minima je zapotřebí upravovat tyto parametry opakovaně právě směrem opačným k tomuto vektoru dokud není nalezeno minimum ztrátové funkce (pokud možno globální). Tuto metodu lze znázornit pro případ optimalizace sítě o dvou parametrech θ_0 a θ_1, kde je ztrátové skóre znázorněno plochou (obrázek 2.15). Je zde také vyznačena křivka s body, které znázorňují jednotlivé výchozí hodnoty pro další optimalizaci. Velikost tohoto kroku se odvíjí od rychlosti učení α [30]. Matematický popis výpočtu aktuální váhy θ_{t+1} je následující:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J(\theta_t) \tag{2.5.3}$$



Obrázek 2.15: Metoda gradientního sestupu [31]

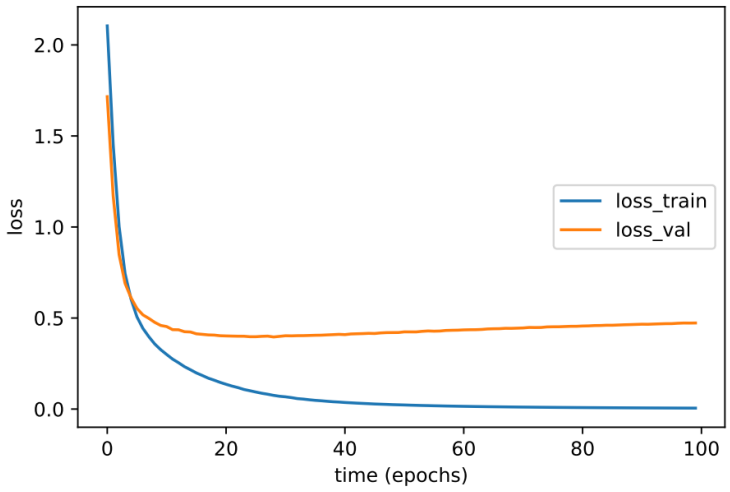
- **Adam** je zkrácené označení pro Adaptive Moment Estimation. Oproti metodám, které k určení směru poklesu využívají pouze stávajícího gradientu, dochází při výpočtech pomocí tohoto algoritmu k adaptaci kroku učení a jeho usměrnění v závislosti na předchozích hodnotách. Díky tomu je Adam nejen rychlejší ale dokáže překonat i problém zaseknutí v lokálním minimu. Právě to z něj dělá je nejpoužívanější a nejvýkonnější algoritmus při učení neuronových sítí [32].

2.5.3 Přeučení

V procesu učení může dojít k jevu takzvaného přeučení. To znamená že model velmi dobře pracuje s trénovacími daty ale naprosto selhává při pohledu na data nová. Vzory které se v těchto trénovacích datech model naučil rozpoznávat nejsou dostatečně zobecněné pro testovací data a tím klesá i výsledná výkonnost naučeného modelu. K prevenci výskytu tohoto jevu přispívá rozdělení dostupných vstupních dat na:

- **Trénovací data** na kterých se daný model učí optimálním parametrům pro získání co nejlepších výsledků.
- **Validační data**, která slouží v procesu učení právě jako nová data na kterých je sledována schopnost generalizace modelu po každém průchodu trénovacími daty (iteraci).
- **Testovací data** slouží ke konečnému vyhodnocení modelu a v procesu učení nejsou využita.

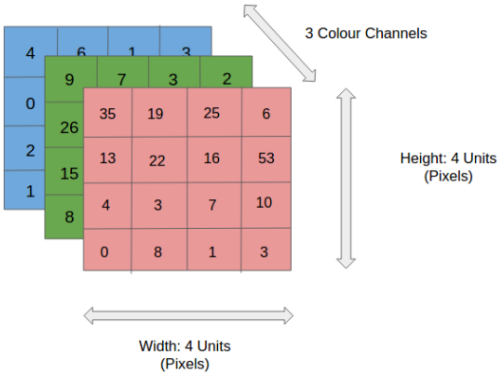
Přeučení díky tomuto rozdělení lze sledovat již v průběhu učení pomocí ztrátového skóre na trénovacích a validačních datech při každé iteraci.



Obrázek 2.16: Přeučení modelu

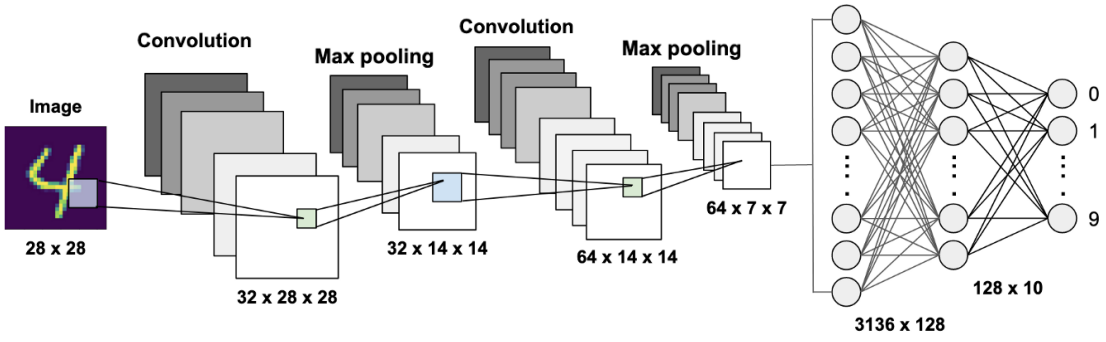
2.6 Konvoluční neuronové sítě

Speciálním typem sítí, které jsou často využívány v počítačovém vidění a oblasti rozpoznání obrazu jsou právě konvoluční neuronové sítě (CNN). Obraz je ve formě vícerozměrného tenzoru, který obsahuje hodnoty jednotlivých pixelů. Jeho rozměry je nejen výška a šířka ale také hloubka, která se odvíjí od počtu barevných kanálů. Například pro RGB (Červená, Zelená, Modrá) je hloubka rovna třem.



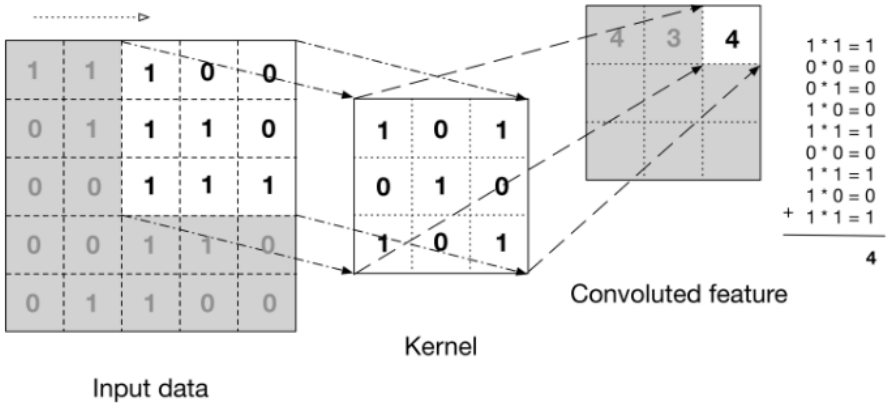
Obrázek 2.17: Tenzor obrazových dat [33]

Pro práci s těmito daty by bylo použití klasických neuronových sítí velmi náročné z hlediska výpočetního výkonu. Jediný neuron v první vrstvě napojený na každou hodnotu barevného obrazu o rozměrech 1024×1024 pixelů by měl přes 3 miliony parametrů. Architektura těchto sítí je proto založena na použití konvolučních a sdružovacích vrstev, které dokáží pracovat s menším počtem parametrů. Tyto vrstvy jsou využity k získání stále většího počtu užitečnějších a jednodušších map příznaků pomocí filtrů. Poslední sada takto získaných map je převedena zplošťovací vrstvou na vektor hodnot, který je vstupem plně propojené sítě neuronů, která pomocí vlastních naučených parametru plní rozhodovací funkci. Takové uskupení vrstev pro klasifikaci ručně psaných čísel lze vidět na následujícím obrázku [34, 35].



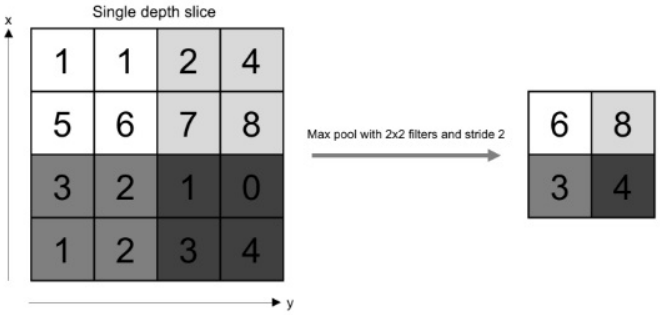
Obrázek 2.18: Konvoluční neuronová síť [36]

- **Konvoluční vrstva** oproti vrstvě neuronů nehledá ve vstupních datech globální vzory. Její princip spočívá právě v hledání lokálních vzorů v obraze. K tomu využívá operaci konvoluce. Ta spočívá v aplikaci konvolučního filtru o předem dané velikosti, který s určitým krokem prochází celým snímkem. Při tomto průchodu násobí hodnoty pixelů vstupního snímku svými vlastními hodnotami, které následně sčítá. Výsledkem je tak příznaková mapa s takto vypočítanými hodnotami. Právě hodnoty těchto filtrů jsou hledanými parametry v procesu učení.



Obrázek 2.19: Princip konvoluční vrstvy [37]

- **Sdružovací vrstvy** využívají filtrů, které oproti těm v konvolučních vrstvách nemají vnitřní parametry ale vykonávají určitou funkci. Využívají se k redukci rozměru map příznaků, kdy filtr prochází touto celou mapou a v daných oblastech o jeho velikostí hledá například maxima hodnot. Taková vrstva se označuje jako MaxPoolingová.

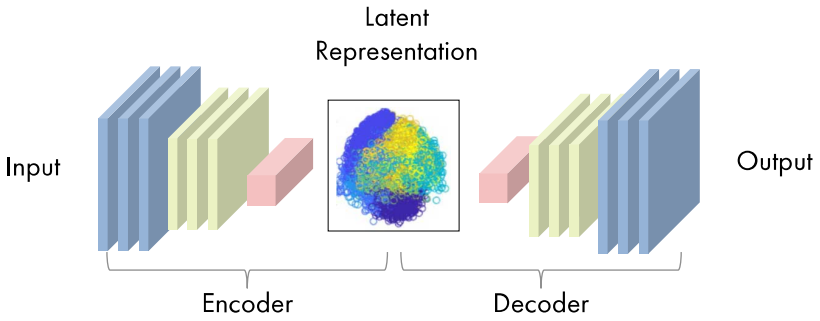


Obrázek 2.20: Sdružovací vrstva MaxPooling [38]

2.7 Autoenkodéry

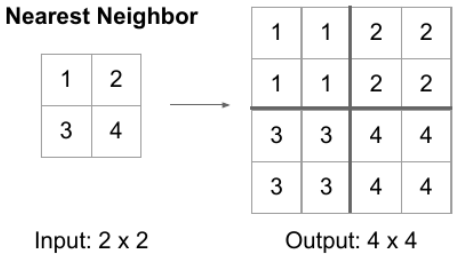
Autoenkodéry jsou specifickým typem dopředných sítí. Jejich schopností je transformovat vstupní snímek v části enkodéru pomocí konvolučních a sdružovacích vrstev a získat tak větší počet menších map příznaků. Ty jsou následně pomocí dekodéru transformovány zpět do původního rozměru. Ke zvětšení rozměru příznakových map využívá vrstev UpSampling. Zmenšené mapy příznaků mezi enkodérem a dekodérem představují latentní reprezentaci, která je zkomprimovanou charakteristikou původního snímku. Při procesu učení jsou parametry upravovány pomocí ztrátové funkce, která sleduje rozdíl mezi vstupním a výstupním snímkem [39].

Samotná latentní reprezentace je produktem komprese snímku a nese velice důležité informace potřebné k jeho zpětné dekompresi do původního formátu. I z těchto dat lze získat velice důležité parametry popisující vstupní snímek. O způsobech hodnocení těchto reprezentací pojednávají odborné články [40] a [41].



Obrázek 2.21: Autoenkodér [42]

- **UpSamplingová vrstva** zajišťuje zvýšení rozměru příznakové mapy. Podobně jako u sdružovácích vrstev tak i tato vrstva nemá žádné parametry k učení. Její princip spočívá v namnožení hodnot v rámci definovaného filtru. Na obrázku 2.22 pak lze vidět vrstvu, která k vyplnění rozšířeného prostoru výstupu v rámci filtru o rozměru 2×2 využívá právě vstupní hodnotu v dané lokalitě [43].

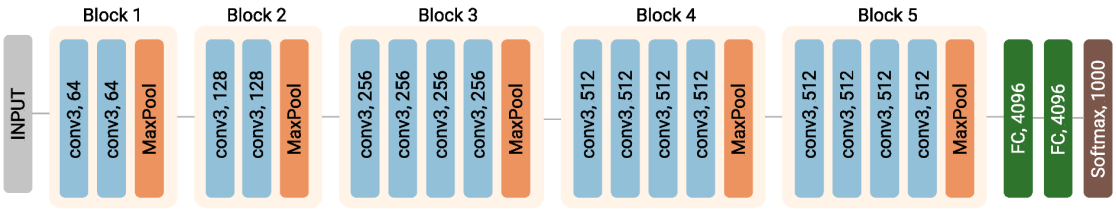


Obrázek 2.22: UpSamlingová vrstva [43]

2.8 Přenos učení

Vytvoření vlastní neuronové sítě může být velice náročné ať z hlediska hledání vhodné struktury, doby učení nebo nedostatku vstupních dat. Tento problém řeší právě metoda přenosu učení, jejímž cílem je zlepšení výkonnosti cílové úlohy využitím znalostí získaných při řešení úlohy jiné. Těmito znalostmi se zamýšlí jednotlivé váhy získané při učení zdrojového modelu na jeho vlastních vstupních datech [44].

V rámci praktické části práce byl z hlediska dostupnosti využit model VGG (Visual Geometry Group), který byl obdobně jako model *AlexNet* vytvořen pro soutěž ILSRVC. Jeho autory jsou A. Zisserman a K. Simonyan. V rámci jejich výzkumu bylo vytvořeno několik konfigurací odvíjejících se od hloubky ve smyslu počtu vrstev, které byly trénovány a hodnoceny na snímcích ImageNet s vysokou správností predikcí. Tu modelu prokázal i v rámci jiných vstupních dat (např. INRIA [45]). V rámci této práce byla využita konfigurace VGG-19 jež je architekturou, která využívá celkem 19 skrytých vrstev [46, 47].



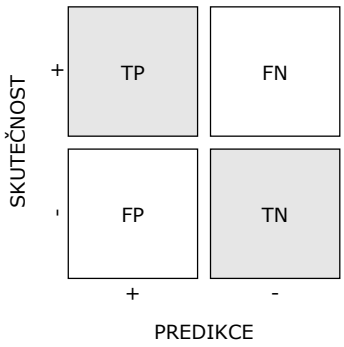
Obrázek 2.23: Struktura modelu VGG-19 [48]

Jedná se o dopřednou síť, jejímž vstupem jsou snímky o rozměru 224×224 pixelů. Tyto snímky vstupují do skrytých vrstev v podobě několika bloků využívajících konvolučních vrstev, kterých je celkem 16. Aktivační funkcí je ReLU a využívají filtrů o velikosti od 1×1 po 3×3 pixely. Tyto bloky jsou redukovány vrstvami MaxPooling. Poslední z těchto bloků je přiveden na rozhodovací část modelu složenou ze tří plně propojených vrstev. Poslední z těchto vrstev odpovídá datům ImageNet a zaměření soutěže a obsahuje proto 1000 neuronů, jejichž počet odpovídá počtu klasifikovaných kategorií. Aktivační funkcí je Softmax kvůli výstupu ve formě pravděpodobnosti.

2.9 Způsoby vyhodnocení

K vyhodnocení modelu dochází již v průběhu učení ale pouze na validačních datech a to z důvodu prevence přeučení na trénovacích datech. K ověření jeho výsledné schopnosti aplikovat získané znalosti na nových datech, však dochází po dokončení procesu učení na testovacích datech. U klasifikačních úloh se pro toto závěrečné vyhodnocení nabízí sledování několika metrik které vychází z matice záměn (obrázek 2.24). U binární klasifikace její hodnoty vychází ze čtyř možných podob výsledných predikcí modelu.

- **Skutečně pozitivní predikce (TP)** kdy model predikoval kladný výstup, který ve skutečnosti je kladný
- **Skutečně negativní predikce (TN)** kdy model predikoval záporný výstup, který ve skutečnosti je záporný
- **Falešně pozitivní predikce (FP)** kdy model predikoval kladný výstup, který je ve skutečnosti záporný
- **Falešně negativní predikce (FN)** kdy model predikoval záporný výstup, který je ve skutečnosti kladný



Obrázek 2.24: Matice záměn

- **Správnost** je definována jako podíl všech správných predikcí (TP a TN), ku jejich celkovému počtu. Tato metrika je však zavádějící pro případy nevyvážených dat.

$$\text{Správnost} = \frac{TP + TN}{TP + TN + FP + FN} \cdot 100\ [\%] \tag{2.9.1}$$

- **Přesnost** je využívána zejména v situacích kdy je velice nechtěným jevem falešně pozitivní predikce. Oroti správnosti bere ohled pouze na kladné případy a její definice je proto následující:

$$\text{Přesnost} = \frac{TP}{TP + FP} \cdot 100\ [\%] \tag{2.9.2}$$

- **Úplnost** je měřítkem toho, kolik kladných případů z celkového počtu bylo modelem skutečně odhaleno.

$$\text{Úplnost} = \frac{TP}{TP + FN} \cdot 100\ [\%] \tag{2.9.3}$$

- **F1-skóre** přikládá stejnou váhu přesnosti a úplnosti a dělá z něj metriku která bere v potaz oba nechtěné případy a to sice falešně pozitivní a falešně negativní predikce.

$$\text{F1-skóre} = 2 \cdot \frac{\text{Přesnost} \cdot \text{Úplnost}}{\text{Přesnost} + \text{Úplnost}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}\ [-] \tag{2.9.4}$$

3 Praktická část

V této části práce je nejdříve uveden použitý hardware a software pro realizaci principů a technik představených v teoretické části práce. Následuje popis zvolených vstupních dat a jejich úpravy, což dalo za vznik výchozímu modelu konvoluční neuronové sítě, u kterého nechybí způsob předzpracování vstupních dat, kompilace a konfigurace. Díky těmto nastavením byl model podroben procesu učení a následnému vyhodnocení, které vedlo k vytvoření dalších modelů v procesu ladění a zvyšování přesnosti. Veškeré zdrojové kódy popsané v této části práce lze nalézt v příloze A.

3.1 Použitý hardware a software

Pro účely této práce byl použit počítač s 64-bitovým operačním systémem Windows, disponující následujícím hardwarem:

- **CPU** : Intel Core i7-10850H 2,7 GHz
- **RAM** : 16 GB (3200 MHz)
- **GPU** : NVIDIA Quadro T2000 (4 GB vRAM)

Modely hlubokého učení jsou v samém základu shlukem matematických operací. K snadné implementaci těchto principů a metod při návrhu, učení a vyhodnocení modelů v této práci byly na základě doporučené literatury vybrány softwarové knihovny Keras a TensorFlow. Dále byla využita knihovna OpenCV, která poskytuje velké množství funkcí pro práci s obrazem. Z hlediska použití těchto knihoven se výchozím programovacím jazykem této práce stal Python. Tvorba programů pak probíhala pomocí open-source distribuce Miniconda, která umožňuje instalaci knihoven a spuštění vývojového prostředí Spyder a internetové platformy Jupyter Notebook.

- **Keras a TensorFlow** jsou knihovny zaměřené na samotnou implementaci metod hlubokého učení díky kterým jsou vstupní snímky reprezentovány ve formě tenzorů. Ty jsou mnohavrstvými tabulkovými strukturami, které lze zpracovávat v rámci neuronových sítí. V případě knihoven Tensorflow je pak na výběr mezi verzí využívající procesoru a verzí, která využívá grafické karty. Verze pro GPU byla využita v případě této práce.
- **OpenCV** je velice obsáhlou knihovnou, jejíž funkce nachází své hlavní využití v oblasti počítačového vidění. V této práci byly využity pro úpravu snímků vstupního souboru dat.

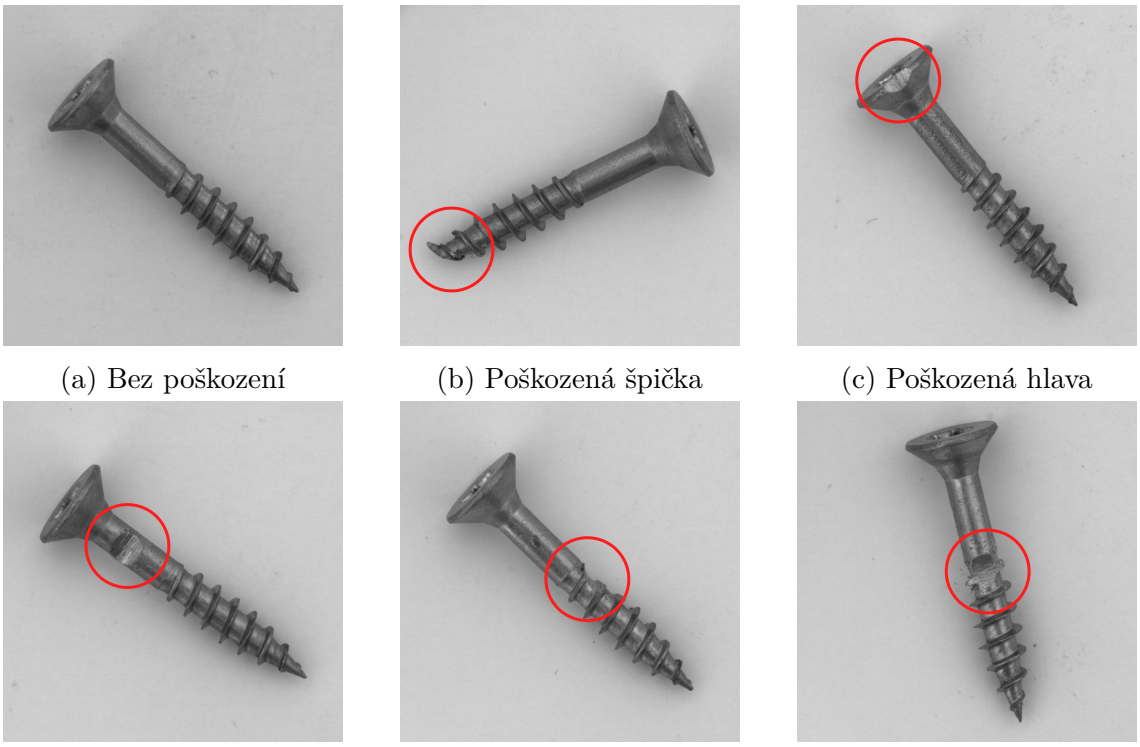
3.2 Vstupní data

Pro použití metod hlubokého učení je zapotřebí zvolit vhodná vstupní data. Pro účely této práce byla požadavkem patřičná vazba na strojní výrobu a rozmanitost možných poškození. Z toho důvodu byl jako zdroj vstupních dat vybrán datový soubor MVTec anomaly detection dataset [49].

Soubor je určen pro porovnání metod detekce anomálií se zaměřením na průmyslovou kontrolu a je volně stažitelný ze stránek společnosti. Obsahuje přes 5000 obrázků ve vysokém rozlišení rozdělených do patnácti různých kategorií objektů a textur. Každá z nich má strukturovaná data v podobě podsložek, které jsou členěny podle toho jestli se jedná o snímky objektů s poškozením nebo bez. Díky této struktuře bylo možno použít techniku strojového učení s učitelem.

Z hlediska zaměření práce byla z tohoto souboru vybrána kategorie vrutů a to kvůli faktu že těchto strojních součástí dokáže výrobní linka produkovat i stovky za minutu. Právě nadměrná produkce a rozměry těchto součástí kladou velké nároky na zručnost a pozorovací schopnosti lidského kontrolora. Jedná se tak o ideální případ, který by mohl podléhat automatizaci s použitím metod hlubokého učení.

Vybraný soubor vstupních dat obsahuje snímky v rozlišení 1024×1024 pixelů. Konkrétně se jedná o 361 snímků výrobků bez vady a 119 snímků zmetků rozdělených do pěti kategorií, které jsou včetně vyznačením jejich specifického poškození představeny na sadě obrázků 3.1.



(a) Bez poškození (b) Poškozená špička (c) Poškozená hlava

(d) Poškozený krček (e) Poškozený závit (bok) (f) Poškozený závit (shora)

Obrázek 3.1: Příklady snímků v jednotlivých kategoriích [49].

3.2.1 Úprava původní struktury

Ze stránek společnosti byl stažen soubor vstupních dat ve formátu *.zip*, který obsahuje složky *train*, *test* a *ground_truth*. Nutno podotknout, že tento soubor dat je určen pro porovnání velice výkonných a složitých modelů detekce anomálií v přesné lokalitě snímku. K vyhodnocení této schopnosti slouží právě složka *ground_truth*, která obsahuje masky odpovídající oblastem poškození ze složky test s přesností na pixely. Tato práce se však omezuje pouze na posouzení toho, zda se jedná o výrobek bez vady nebo o výrobek s vadou, případně jakou. Složka *ground_truth* tak není v práci využita.

Struktura staženého souboru vstupních dat je následující:

```
Originální datový soubor
├── ground_truth ..... 119 snímků
│   ├── manipulated_front ..... 24 snímků
│   ├── scratch_head ..... 24 snímků
│   ├── scratch_neck ..... 25 snímků
│   ├── thread_side ..... 23 snímků
│   └── thread_top ..... 23 snímků
├── test ..... 160 snímků
│   ├── good. ..... 41 snímků
│   ├── manipulated_front ..... 24 snímků
│   ├── scratch_head ..... 24 snímků
│   ├── scratch_neck ..... 25 snímků
│   ├── thread_side ..... 23 snímků
│   └── thread_top ..... 23 snímků
└── train ..... 320 snímků
    └── good ..... 320 snímků
```

Z důvodů uvedených v kapitole 2.5.3 je pro trénování, průběžnou validaci a závěrečné testování modelu třeba rozdělit původní soubory snímků do tří složek. K tomuto rozdělení vznikl program, který lze nalézt v příloze A.1. Přesněji řečeno se jedná o ručně definovanou funkci, která kopíruje veškeré snímky z původních složek *train* a *test* do jedné složky. Při tomto kopírování je zachována původní struktura v podobě podsložek jednotlivých kategorií od *good* po *thread_top*. Dále dochází k vytvoření tří nových prázdných složek *train*, *valid* a *test*, které obsahují původní strukturu kategorií ve formě podsložek. K jejich vyplnění dochází při postupném kopírování snímků ze spojené složky v závislosti na požadovaném rozdělení. Toto rozdělení ve formě seznamu tří čísel je stejně jako cesta k původnímu souboru vstupním parametrem funkce.

V rámci této práce bylo pomocí této funkce vytvořeno několik souborů s různým rozdělením dat. Pro účely této práce a možnost porovnání všech modelů bylo zvoleno rozdělení, kde trénovací data obsahují 50%, validační 30% a testovací 20% snímků původního souboru a to z důvodu dostatečného počtu snímků ve validačním a testovacím souboru.

Pro toto rozdělení je zde uvedeno patřičné zadání vstupních parametrů a vyvolání definované funkce:

```
org_ds_path = r"..\screw_orig"
distribution = [50, 30, 20] #[train, valid, test]
ds_distribution(org_ds_path,distribution)
```

Výpis kódu 3.1: Funkce rozdělení

Parametry *org_ds_path* je cesta k původnímu souboru a *distribution* požadované rozdělení ve tvaru seznamu $[50, 30, 20]$. Toto rozdělení se nevztahuje na celou složku snímků ale pracuje v rámci jejích podsložek a je tak pouze přibližné, jelikož musí pracovat s celými čísly. Struktura souboru vytvořeného pomocí takto nadefinovaných parametrů je následující:

```
Upravený datový soubor
├── train ..... 238 snímků (~ 50%)
│   ├── good ..... 180 snímků
│   ├── manipulated_front ..... 12 snímků
│   ├── scratch_head ..... 12 snímků
│   ├── scratch_neck ..... 12 snímků
│   ├── thread_side ..... 11 snímků
│   └── thread_top ..... 11 snímků
├── valid ..... 141 snímků (~ 30%)
│   ├── good ..... 108 snímků
│   ├── manipulated_front ..... 7 snímků
│   ├── scratch_head ..... 7 snímků
│   ├── scratch_neck ..... 7 snímků
│   ├── thread_side ..... 6 snímků
│   └── thread_top ..... 6 snímků
└── test ..... 101 snímků (~ 20%)
    ├── good ..... 73 snímků
    ├── manipulated_front ..... 5 snímků
    ├── scratch_head ..... 5 snímků
    ├── scratch_neck ..... 6 snímků
    ├── thread_side ..... 6 snímků
    └── thread_top ..... 6 snímků
```

3.2.2 Normalizace dat

Pro učení je z hlediska rychlosti rozhodující s jakým rozsahem čísel bude model pracovat. Jelikož jsou data načítána ve formátu *RGB*, jsou hodnoty jednotlivých pixelů snímku v rozsahu (0,255) a to není ideální. K urychlení procesu učení se tedy využívá normalizace těchto hodnot.

V této práci je toho dosaženo použitím speciální vrstvy *Rescaling* z knihovny TensorFlow. Ta dělí hodnoty jednotlivých pixelů v dávkách snímků jejich maximem 255 a převádí je tak do rozsahu od 0 do 1. Tato vrstva je v rámci bezejmenné *lambda* funkce aplikována metodou *.map* na předzpracovaná data.

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
valid_ds = valid_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds  =  test_ds.map(lambda x, y: (normalization_layer(x), y))
```

Výpis kódu 3.2: Normalizace dat

Další úprava předzpracovaných vstupních dat spočívala v optimálním využití procesoru. Toho je dosaženo předběžným načítáním vstupních dat během doby, kdy je jejich dávka zpracovávána modelem. Díky tomu je optimálně využit procesor a dochází k úspoře času při jejich užívání. O velikosti dávky snímků, které mají být připraveny je rozhodováno automaticky pomocí funkce $tf.data.experimental.AUTOTUNE$.

```
train_ds = (train_ds.cache().prefetch(buffer_size=tf.data.experimental.
    AUTOTUNE))
```

Výpis kódu 3.3: Optimalzace načítání dávek snímků

3.2.3 Umělé rozšíření dat

Umělé rozšíření se využívá v případech, kdy datový soubor obsahuje nedostatečné množství snímků potřebných k učení modelu. Jedná se o úpravu originálních a jedinečných snímků, pro kterou byl v případě této práce vytvořen speciální sekvenční model o jedné vrstvě. Rozšíření aplikuje přímo a pouze v procesu učení modelu v rámci dávek snímků. Použitou úpravou je překlopení každého snímků okolo vertikální osy, což vede k zdvojnásobení jejich počtu.

```
data_augmentation =tf.keras.Sequential([preprocessing.RandomFlip('
    horizontal'])
```

Výpis kódu 3.4: Umělé rozšíření dat

3.2.4 Dodatečná úprava snímků

V rámci zvyšování přesnosti došlo k úpravě samotných snímků. Jak lze vidět na obrázku 3.1, vruty jsou na fotografiích natočeny různým směrem což není ideální jelikož vytvořená síť nemá dostatek snímků se stejně natočenými vruty a různým poškozením. Například pro kategorii s poškozeným krčkem (*scratch_neck*) je v trénovacím souboru dat pouze 12 snímků.

Pro jejich natočení byl vytvořen program, který lze nalézt v příloze A.2. Ten obsahuje funkci, která využívá především knihovny OpenCV. Na výpisu z kódu 3.1 lze vidět část vytvořené funkce, ve které dochází k natáčení jednotlivých snímků. To probíhalo jejich porovnáním s předlohou, kterou byla první fotografie v celém souboru. Odpovídajícího natočení bylo dosaženo natáčením jednotlivých snímků s krokem 1° v rozsahu 0 až 360°. V každém kroku byl natočený snímek porovnán s předlohou pomocí funkce *matchTemplate*. Hodnota příslušící jeho odlišnosti byla zapsána do seznamu hodnot s odpovídajícím úhlem natočení. Po postupném pootočení snímku

v plném rozsahu byla vybrána minimální hodnota odlišnosti a snímek byl pootočen o odpovídající úhel. Místa snímku, která by po natočení zůstala prázdná byla vyplněna pomocí jeho krajních pixelů.

```
for mask_phi in range(0,360):
    mask_rot_mtx = cv.getRotationMatrix2D(center=m_center,angle=
    mask_phi,scale=1)
    rot_mask_copy = cv.warpAffine(src=cv_nrot_mask_copy,
                            M=mask_rot_mtx,
                            dsize=(m_w, m_h),
                            borderMode=cv.BORDER_REPLICATE)
    rot_mask_match = cv.matchTemplate(cv_mask_0,rot_mask_copy,cv.
    TM_SQDIFF_NORMED)
    min_mask_val, max_val, min_loc, max_loc = cv.minMaxLoc(
    rot_mask_match)
    mask_Value.append(min_mask_val)
    mask_Phi.append(mask_phi)

mask_Phi_index = mask_Value.index(min(mask_Value))
nrot_mask_cv = cv.imread(org_mask_path)
f_rot_mtx = cv.getRotationMatrix2D(center=m_center, angle=mask_Phi[
    mask_Phi_index], scale=1)
f_rot_mask = cv.warpAffine(src=nrot_mask_cv,
                           M=f_rot_mtx,
                           dsize=(m_w, m_h),
                           borderMode=cv.BORDER_REPLICATE)
f_img = cv.imwrite(org_mask_path,f_rot_mask)
os.remove(org_mask_copy_path)
```

Výpis kódu 3.5: Natáčení vrutů jedním směrem

Načtení, otočení a nahrazení všech snímků z datového souboru trvalo zhruba 30 minut. Výsledek však nebyl dokonalý a některé snímky musely být otočeny ručně o 180°. Ukázku výsledné úpravy lze vidět na sadě obrázků 3.2.

(a) Poškozená špička (b) Poškozený krček (c) Poškozený závit

Obrázek 3.2: Příklady natočených snímků

3.3 Výchozí model

Tato kapitola se zabývá návrhem výchozího modelu, vytvořeného na základě poznatků z teoretické části práce. Ten díky struktuře vstupních dat využívá strojového učení s učitelem. Je zde uveden způsob předzpracování vstupních dat a proces učení navrženého modelu. Výsledkem se stal naučený model, který byl podroben testování a konečnému vyhodnocení. Celý program, který byl v rámcitohoto modelu vytvořen lze v plné podobě nalézt v příloze A.3.

Způsob předzpracování vstupních dat:

Vstupními daty se stal již zmíněný rozdělený soubor dat, se snímky šesti kategorií. Pro proces učení modelu bylo zapotřebí z těchto dat vytvořit datovou strukturu, se kterou dokáže pracovat. K jejímu vytvoření bylo využito funkce *image_dataset_from_directory* z knihovny Keras. Trénovací, validační i testovací soubory dat byly upraveny následovně.

```
tf.keras.utils.image_dataset_from_directory(directory,
                                            labels = 'inferred',
                                            label_mode = 'categorical',
                                            color_mode = 'rgb',
                                            image_size = (256, 256),
                                            batch_size = 32,
                                            shuffle = True
                                            seed = 3)
```

Výpis kódu 3.6: Předzpracování dat

- **directory :** Odkazuje na umístění souboru podle toho jestli se jedná o trénovací, validační nebo testovací soubor.
- **labels :** Z hlediska přítomné struktury je nastaveno na $'inferred'$, což přiděluje snímkům označení podle složky, ve které se nachází.
- **color_mode :** Zajišťuje transformaci snímku ve smyslu změny barevných kanálů. Pro případ vybraných vstupních dat byla nastavena transformace do tří kanálů, a to sice $'rgb'$.
- **batch_size :** Neboli velikost dávek, která určuje kolik snímků projde modelem než dojde k optimalizaci jeho parametrů. S ohledem na dostupný hardware byl počet snímků v jedné dávce pro tento model nastaven na 32.
- **image_size :** Transformuje rozměr vstupního snímku. Ten byl opět s ohledem na hardware snížen na 256×256 pixelů. Ačkoli je tento rozměr menší než původní, je dostačujícím k rozpoznání vad.
- **Shuffle:** Byl aktivován hodnotou $True$ a původní abecední uspořádání podle označení snímků bylo promícháno aby nedocházelo k optimalizaci vah na stejné kategorii snímků víckrát než jednou což podporuje výslednou schopnost generalizace.

- **Seed:** je hodnotou, která zajišťuje stejné promíchání a to kvůli možnosti porovnání mezi zvolenými parametry při definici procesu učení.

Výsledkem se stala proměnná typu $tf.data.Dataset$. Ta je reprezentována tenzorem ve tvaru (obrázky, označení), kde jsou obrázky ve tvaru (velikost dávky, šířka, výška, počet kanálů) a označení ve tvaru (velikost dávky, příslušná kategorie). Po této úpravě byla aplikována normalizace hodnot pixelů a úprava potřebná k optimálnímu využití procesoru tak jak je uvedeno v kapitole 3.2.

Struktura modelu:

První vytvořenou strukturou se stal sekvenční model, tedy model jehož vrstvy na sebe postupně navazují a nemají žádná další spojení. Skládá se ze tří spojených částí specifických vrstev. Sestavení spočívalo ve vytvoření prázdného modelu $tf.keras.Sequential()$, do které byly přidávány vrstvy ve specifickém pořadí pomocí metody $model.add$.

První a tudíž i vstupní část modelu obsahuje již nadefinované umělé rozšíření dat v podobě implementace modelu $data\_augmentation$ popsaného v kapitole 3.2.

Druhá část modelu využívá konvolučních vrstev $Conv2D$, u kterých byl postupně zvyšován počet produkovaných map příznaků z důvodu získání dostatečného počtu reprezentací. Dalšími definovanými parametry byl rozměr filtru a aktivační funkce. Aktivační funkcí všech těchto vrstev se stala funkce $ReLU$. Jedná se o tři vrstvy, kde po každé z nich byl redukován rozměr produkovaných map pomocí sdružovací vrstvy $MaxPooling2D$. Poslední vrstvou této části je vrstva $Flatten$, která z map příznaků ve formě vícerozměrného tenzoru tvoří vektor hodnot.

Třetí a poslední část modelu plní rozhodovací funkci a jejím vstupem je získaný vektor hodnot. Jedná se o plně propojenou síť reprezentovanou vrstvami $Dense$. Tyto vrstvy byly definovány počtem neuronů a aktivační funkcí. Celkem se jedná o tři vrstvy kde byla u prvních dvou vrstev použita aktivační funkce $ReLU$. U třetí, výstupní vrstvy se jednalo o aktivační funkcí $Softmax$. Ta zajišťuje výsledek ve formě pravděpodobnosti aktivace jednoho ze šesti neuronů, jejichž počet byl nastaven tak, aby odpovídal klasifikovaným kategoriím.

Nastavené počty map příznaků a neuronů, použité aktivační funkce a rozměry filtrů můžete společně s počtem parametrů a výstupními rozměry jednotlivých vrstev vidět v tabulce 3.1. Výstupní rozměr pro konvoluční a maxpoolingové vrstvy má tvar trojrozměrného tenzoru ve formátu (výška, šířka, počet map příznaků). U plně propojených vrstev se jedná o počty neuronů.

Tabulka 3.1: Struktura výchozího modelu

Typ vrstvy	Výstupní rozměr	Aktivační funkce	Rozměr filtru	Počet parametrů
Augmentation	(256,256,3)	-	-	0
Conv2D	(252,252,8)	ReLU	(5,5)	608
MaxPooling2D	(126,126,8)	-	(2,2)	0
Conv2D	(124,124,64)	ReLU	(3,3)	4672
MaxPooling2D	(62,62,64)	-	(2,2)	0
Conv2D	(61,61,128)	ReLU	(2,2)	32896
MaxPooling2D	(30,30,128)	-	(2,2)	0
Flatten	115200	-	-	0
Dense	64	ReLU	-	7372864
Dense	32	ReLU	-	2080
Dense	6	Softmax	-	198
Počet parametrů k učení:				7413318
Celkový počet parametrů:				7413318

Učení modelu:

Prvním potřebným krokem k realizaci učení modelu byla jeho konfigurace a kompilace, v rámci které bylo zapotřebí definovat optimalizační algoritmus, typ ztrátové funkce a sledovanou metriku v procesu učení.

V knihovně TensorFlow je na výběr z devíti optimalzačních algoritmů v podobě tříd funkce *tf.keras.optimizers*. V případě tohoto modelu se jím po několika experimentech a porovnání výsledných hodnot ztrátového skóre stal algoritmus Adam. Vnitřní nastavení byla ponechána na výchozích hodnotách kromě kroku učení, který byl nastaven na hodnotu 1×10^{-4}.

```
optimizer =tf.keras.optimizers.Adam(learning_rate = 1e-4)
```

Výpis kódu 3.7: Nastavení optimalizačního algoritmu

Ztrátová funkce a hodnocená metrika je definována v rámci metody *model.compile*, která vytvořený model konfiguruje pro učení. V rámci predikce jedné z šesti kategorií aktivační funkcí Softmax, se ztrátovou funkcí stala křížová entropie, která je blíže popsána v kapitole 2.5.1. Společně se ztrátovým skóre se v rámci učení stala sledovanou metrikou také průběžná správnost predikcí. Toto nastavení se nachází ve výpisu kódu 3.8.

```
model1.compile(optimizer = optimizer,
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])
```

Výpis kódu 3.8: Konfigurace výchozího modelu

Aby v procesu učení modelu nedocházelo k přeučení a degradaci optimalizovaných vah, bylo zapotřebí definovat funkci předběžného zastavení. Sledovaným parametrem se vzhledem k definici přeučení (kapitola 2.5.3) stalo ztrátové skóre na validačních datech. Vzhledem k tomu že k přeučení dochází při nárůstu této hodnoty, byl parametr *mode* nastaven na $'min'$ a indikuje tak stav, kdy už nedochází k jejímu poklesu. Nejnižší akceptovaná změna ztrátového skóre, uvažována za pokrok vůči předchozí iteraci byla nastavena na hodnotu $0,001$. Pro případ zániku přeučení byla nastavena prodleva 50 iterací. Aby model neuložil nevhodné parametry, které byly v rámci této prodlevy získány, bylo nastaveno také ukládání nejlepších parametrů.

```
early_stopping = callbacks.EarlyStopping(monitor='val_loss',
                                         min_delta=0.001,
                                         patience=50,
                                         verbose=1,
                                         mode='min',
                                         restore_best_weights=True)
```

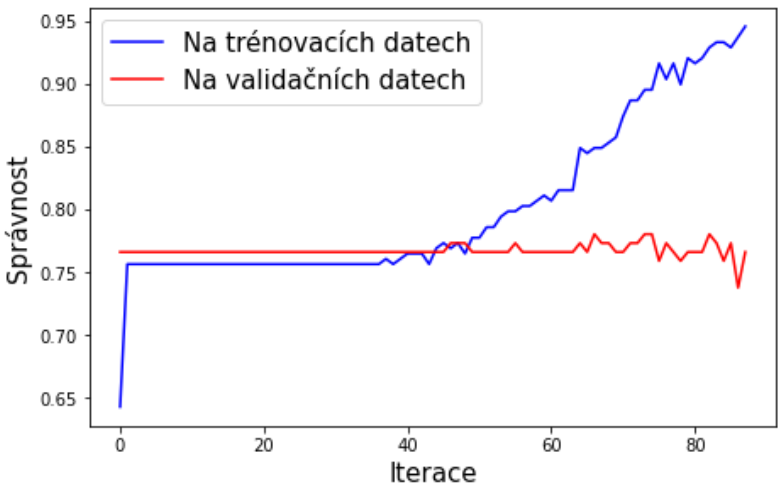
Výpis kódu 3.9: Funkce předběžného zastavení

Vlastní učení takto nakonfigurovaného modelu proběhlo v rámci metody $model.fit$. Mimo toho na kterých datech má docházet k učení a validaci získaných parametrů zde bylo zapotřebí nadefinovat počet iterací. Vzhledem k předchozím pokusům bylo nastaveno 300 iterací se zapojením funkce předběžného zastavení.

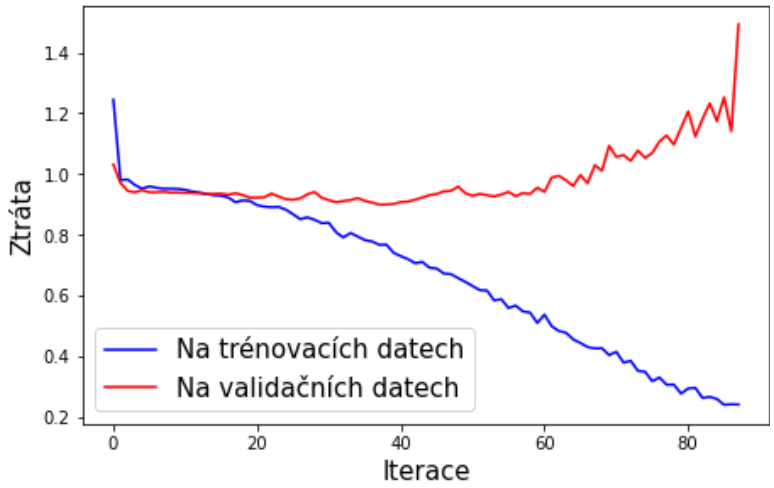
```
history = model1.fit(train_ds, validation_data = valid_ds,
                     epochs=300,
                     verbose = 1,
                     callbacks=early_stopping)
```

Výpis kódu 3.10: Učení modelu

Vývoj správnosti a ztrátového skóre při učení a validaci na příslušných datech lze nejlépe vidět na obrázku 3.3 a 3.4. Z těchto grafů je patrné, že došlo k přeučení a nebylo tak dokončeno všech 300 iterací. Funkce předběžného zastavení zaznamenala nárůst ztrátového skóre na validačních datech a uložila parametry z 38. iterace. Je zde také vidět ztráta schopnosti generalizace, která se projevila nárůstem správnosti a nalezením nejoptimálnějších parametrů pouze pro trénovací data.



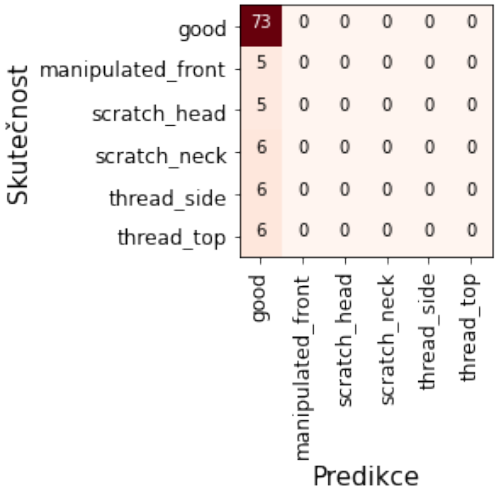
Obrázek 3.3: Vývoj správnosti v průběhu učení prvního modelu



Obrázek 3.4: Vývoj ztráty v průběhu učení prvního modelu

Vyhodnocení modelu:

V rámci tvorby tohoto modelu docházelo k různým úpravám hodnot a parametrů. Při jeho testování metodou *model.evaluate* na odpovídajícím souboru dat dosahovalo popsané nastavení nejlepších výsledků ztrátového skóre 1.0741 a správnosti 72, 27%. K vizualizaci jeho výsledné schopnosti klasifikovat vady slouží matice záměn, kterou lze vidět na obrázku 3.5.



Obrázek 3.5: Matice záměn získaná pomocí výchozího modelu

Jak si lze všimnout, model přiřadil všem 101 snímkům z testovacího souboru označení *good*, tedy bez vady. Výsledná správnost 72% tedy odpovídá zastoupení takto označených snímků v testovacím souboru dat. Výsledná přesnost, úplnost a F1-skóre pro každou kategorii se nachází v tabulce 3.2 společně s jejich váženým průměrem.

Tabulka 3.2: Vyhodnocení prvního modelu

Kategorie	**Přesnost** [%]	**Úplnost** [%]	**F1-skóre** [-]	**Počet snímků**
Bez vady	72	1	0,84	73
Poškozená špička	0	0	0	5
Poškozená hlava	0	0	0	5
Poškozený krček	0	0	0	6
Poškozený závit (bok)	0	0	0	6
Poškozený závit (shora)	0	0	0	6
Vážený průměr	52	72	0,61	

3.4 Druhý model

Tento model vznikl na základě dosažených výsledků prvního modelu. Ty nebyly ideální, jelikož se modelu nepodařilo klasifikovat ani jednu z kategorií poškození. Následující model přináší vylepšení v samotné struktuře, která využívá přenosu učení. Pro učení a vyhodnocení bylo použito stejně rozdělených vstupních dat, tentokrát však ve dvou verzích a to se snímky náhodně a stejně natočených vrutů. Program definující tento druhý model se nachází v příloze A.4.

Struktura modelu:

Vylepšení oproti výchozímu modelu spočívá v použití metody přenosu učení, jelikož výsledek poukazuje na nedostatečný počet vstupních dat v trénovacím souboru i přes jeho umělé rozšíření. Ve struktuře bylo využito přenosu znalostí získaných při učení na jiné úloze, v podobě již natrénovaného modelu. Tím se díky výborné schopnosti klasifikovat snímky stal model VGG-19. Jeho struktura je blíže popsána v kapitole 2.8. Model byl implementován do následující vytvořené struktury pod názvem *base_model* definovaného pomocí funkce *tf.keras.aplications*. Ta umožňuje potřebné úpravy pro jeho nasazení na jiný typ úlohy.

```
base_model = tf.keras.applications.vgg19.VGG19(include_top=False,
                                               weights='imagenet')

base_model.trainable = False
```

Výpis kódu 3.11: Načtení předtrénovaného modelu

Nejprve byla nastaven parametr *include_top*, reprezentující původní vrstvy plnící roli klasifikace na hodnotu *False*. To zajišťuje možnost nasazení vlastních plně propojených vrstev, jelikož původní soubor dat slouží ke klasifikaci 1000 kategorií. Dále byl nastaven parametr *weights* tak, aby byly použity již natrénované váhy sítě, které reprezentují přenášené znalosti. Dále bylo znemožněno jeho opětovné učení pomocí metody *.trainable* nastavené na *False*, právě proto aby byly využity již získané parametry pro tvorbu reprezentací.

Výsledná struktura je sekvenčním modelem, kde je nejprve zapojena vrstva umělého rozšíření v podobě vrstvy *data_augmentation*. Na tuto vrstvu navazuje zmíněný model VGG-19, který pomocí již natrénovaných parametrů produkuje mapy příznaků, tentokrát však pro snímky vrutů. Výstupní mapy příznaků musely být převedeny pomocí vrstvy *Flatten* na vektor hodnot, který je vstupem rozhodovací části v podobě plně propojených vrstev. Ty byly navrženy pro klasifikaci snímků vrutů do šesti kategorií. Jedná se o vrstvy *Dense* s aktivační funkcí *ReLU*. Výjimkou je poslední vrstva, ve které byla použita aktivační funkce Softmax pro získání pravděpodobností jednotlivých kategorií. Výsledný návrh struktury je znázorněn v tabulce 3.3.

Tabulka 3.3: Struktura druhého modelu

Typ vrstvy	Výstupní rozměr	Aktivační funkce	Počet parametrů
Augmentation	(224,224,3)	-	0
VGG19	(7,7,512)	-	20024384
Flatten	25088	-	0
Dense	64	ReLU	1605696
Dense	16	ReLU	1040
Dense	6	Softmax	102
Počet parametrů k učení:			1606838
Celkový počet parametrů:			21631222

Způsob předzpracování vstupních dat:

Vstupními daty se staly vytvořené soubory náhodně a stejně natočených vrutů. Pro možnost porovnání s předchozím modelem bylo použito stejného rozdělení trénovacích, validačních a testovacích dat. Vzhledem k využití přenosu učení ve formě implementace modelu VGG-19, musela být nastavena transformace rozměru snímků tak, aby jejich rozměr odpovídal původním snímkům na kterých byl trénován. Jedná se tedy o rozměr 224×224 pixelů. Jelikož předtrénovaný model v poslední vrstvě využívá 512 map příznaků, musel být z důvodu nedostatku paměti RAM snížen také počet snímků v jedné dávce na 16. Nastavení těchto parametrů proběhlo v rámci funkce *image_dataset_from_directory*, společně s promícháním a převodem do požadovaného barevného formátu.

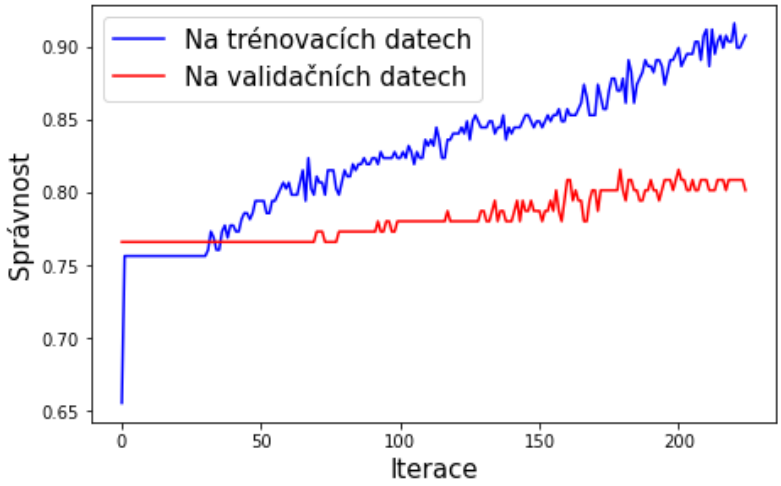
```
tf.keras.utils.image_dataset_from_directory(directory,
                                            labels = 'inferred',
                                            label_mode = 'categorical',
                                            color_mode = 'rgb',
                                            image_size = (224, 224),
                                            batch_size = 16,
                                            shuffle = True
                                            seed = 3)
```

Výpis kódu 3.12: Předzpracování dat

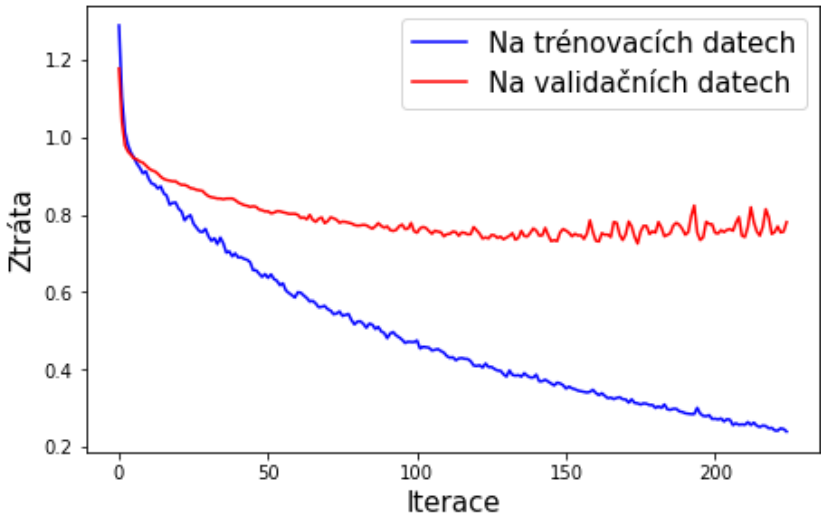
Učení modelu:

Samotná konfigurace a kompilace proběhla stejným způsobem jako u prvního modelu. Byl použit optimalizační algoritmus *Adam* se stejným krokem učení. Hlavním rozdílem a výhodou oproti výchozímu modelu je, že došlo k využití již naučených parametrů, kterých je přes 20 milionů. Při učení tohoto modelu tak docházelo k hledání optimálních vah pouze pro rozhodovací část v podobě plně propojených vrstev, což proces učení urychlilo. Nastaveno bylo 600 iterací a k indikaci přeučení byla zapojena funkce předběžného zastavení.

V prvním případě se tedy jednalo o učení modelu pomocí náhodně natočených vrutů s využitím vrstvy umělého rozšíření. Vývoj správnosti a ztrátového skóre v průběhu učení je uveden na obrázcích 3.6 a 3.7. Z těchto grafů lze vidět, že ztrátové skóre hodnocené na validačních datech dosáhlo po 100. iteraci hodnot nižších než $0,8$. Po 150. iteraci se však společně se zvedající přesností na trénovacích datech začalo uchylovat k vyšším hodnotám a došlo tak k přeučení. Funkce předběžného zastavení zachytila nejmenší hodnoty ztrátového skoře na validačních datech ve 175. iteraci a uložila patřičné váhy.

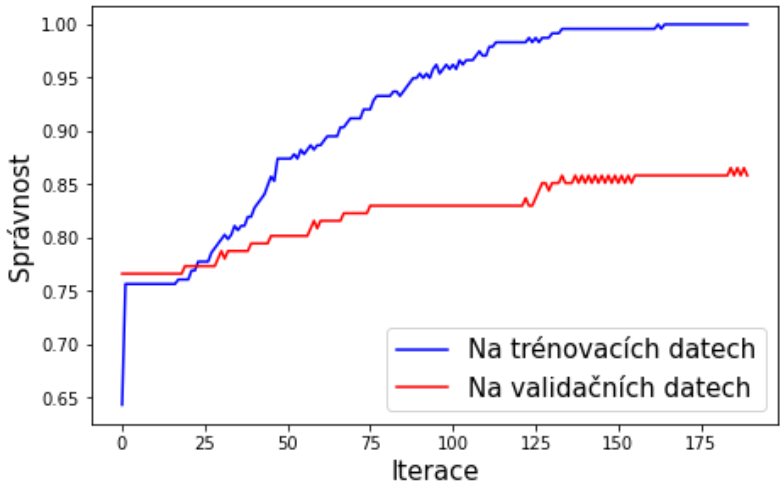


Obrázek 3.6: Vývoj správnosti v průběhu učení druhého modelu

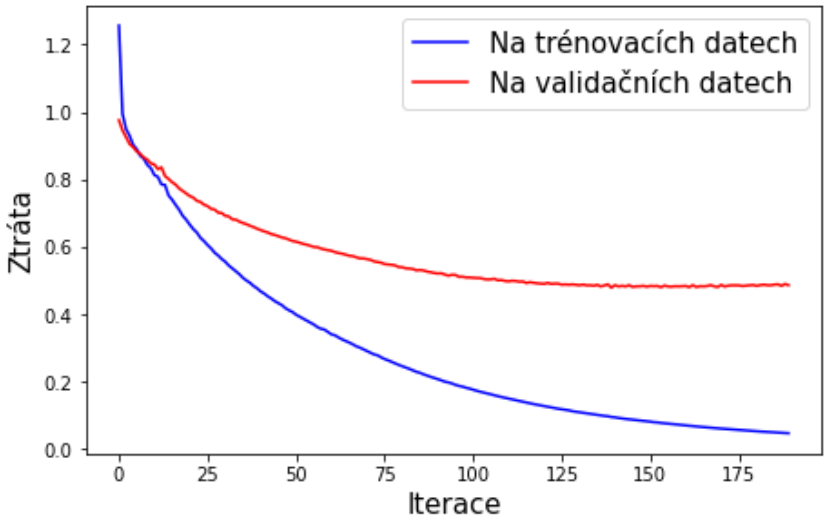


Obrázek 3.7: Vývoj ztráty v průběhu učení druhého modelu

V druhém případě, tedy při využití snímků stejně natočených vrutů bylo využito stejné konfigurace a kompilace s tím rozdílem, že byla vynechána vrstva umělého rozšíření snímků. Překlopení podél vertikální osy by totiž vedlo k narušení stejného natočení. Průběh učení s použitím těchto dat lze vidět na obrázku 3.8 a 3.9. Ztráta na validačních datech se uchýlila k hodnotě $0,5$. Funkce předběžného zastavení zachytila její minimum ve 140. iteraci a uložila patřičné váhy. Od této iterace se však tato validační ztráta začala velice mírně zvedat, což je známkou přeučení. To potvrzuje i nárůst správnosti u trénovacích dat oproti té na validačních datech.



Obrázek 3.8: Vývoj správnosti při učení na stejně natočených vrutech



Obrázek 3.9: Vývoj ztráty při učení na stejně natočených vrutech

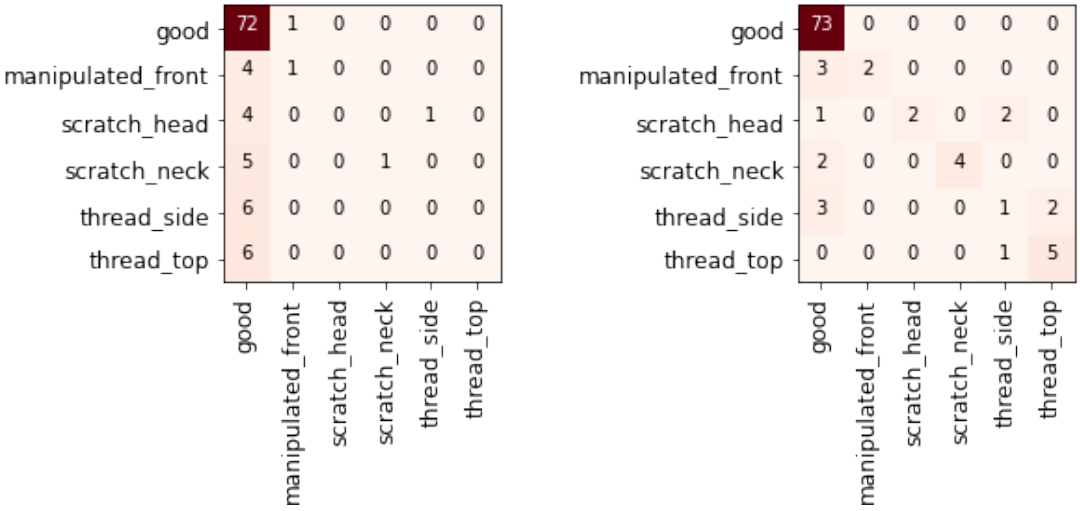
Vyhodnocení modelu:

Výše popsaná nastavení vedla k nejlepším hodnotám správnosti a ztrátového skóre, které byly získány při jeho testování na odpovídajícím souboru dat. Souhrn dosažený hodnot pro obě verze vstupních dat se nachází v tabulce 3.4.

Tabulka 3.4: Testování druhého modelu

Použitá data	**Správnost** [%]	**Ztrátové skóre** [-]
Náhodně natočené vruty s rozšířením	73,26	0,9733
Stejně natočené vruty bez rozšíření	85,14	0,4171

Při použití stejně natočených vrutů se jedná o pokrok jak ve výsledné správnosti tak v dosažení nižší hodnoty ztrátového skóre. Tento pokrok je znatelný i při zhodnocení pomocí matic záměn v obrázku 3.10 pro oba případy použitých dat.



(a) Pro náhodně natočené vruty (b) Pro stejně natočené vruty

Obrázek 3.10: Matice záměn druhého modelu

Model trénovaný na náhodně natočených vrutech správně nerozpoznal ani jeden ze snímků s poškozením na závitu a poškozením hlavy vrutu. V rozpoznání těchto vad se mnohem lépe dařilo modelu s použitím stejně natočených vrutů. Výsledné správnosti, přesnosti a dosažená F1-skóre pro každou z kategorií můžete vidět pro náhodně natočené vruty v tabulce 3.5 a pro stejně natočené vruty v tabulce 3.6.

Tabulka 3.5: Vyhodnocení modelu pro náhodně natočené vruty

Kategorie	**Přesnost** [%]	**Úplnost** [%]	**F1-skóre** [-]	**Počet snímků**
Bez vady	74	99	0,85	73
Poškozená špička	50	20	0,29	5
Poškozená hlava	0	0	0	5
Poškozený krček	100	17	0,29	6
Poškozený závit (bok)	0	0	0	6
Poškozený závit (shora)	0	0	0	6
Vážený průměr	62	73	0,64	

Tabulka 3.6: Vyhodnocení modelu pro stejně natočené vruty

Kategorie	**Přesnost** [%]	**Úplnost** [%]	**F1-skóre** [-]	**Počet snímků**
Bez vady	89	99	0,94	73
Poškozená špička	100	40	0,57	5
Poškozená hlava	100	40	0,57	5
Poškozený krček	100	67	0,8	6
Poškozený závit (bok)	25	17	0,2	6
Poškozený závit (shora)	71	83	0,77	6
Vážený průměr	86	86	0,84	

3.5 Autoenkodér

Posledním vytvořeným modelem v rámci této práce je model autoenkodéru. Ten založen na jiných principech než předchozí modely a jeho schopnost klasifikace se omezuje pouze na rozpoznání toho zda se jedná o vrut s vadou nebo bez. Celý program se nachází v příloze A.5.

Předzpracování vstupních dat:

Pro učení tohoto modelu je využito pouze snímků bez vady. Těmi se tak staly snímky z původní složky *good*, která obsahuje celkem 320 snímků náhodně natočených vrutů. Dalšími vstupními daty se staly snímky ze složky *test*. Ty byly rozděleny do dvou souborů. Prvním se stal soubor představující vzorek dat získaný pomocí lidské kontroly a slouží k získání rozhodujících hodnot. Druhý soubor pak slouží k závěrečnému vyhodnocení a testování.

```
Nový datový soubor
├── Trénovací soubor dat ..... 320 snímků
│   └── good
├── Označený vzorek dat .....66 snímků
│   ├── good .....20 snímků
│   └── anomaly ..... 46 snímků
└── Testovací soubor dat ..... 94 snímků
    ├── good .....21 snímků
    └── anomaly ..... 73 snímků
```

Normalizace hodnot pixelů jednotlivých snímků proběhla v rámci funkce *ImageDataGenerator* z knihovny *Keras* nastavením jejího parametru *rescale* na hodnotu 1./255, jelikož jsou vstupní snímky načítány ve formátu *RGB*. Samotné načtení pak proběhlo pomocí metody *.flow_from_directory*, která oproti předchozí metodě nevytváří proměnnou formátu *tf.data.Dataset* ale generuje dávky snímků přímo v průběhu učení z jejich umístění ve tvaru (velikost_dávky, rozměr, počet kanálů). Nastavenými parametry se stal požadovaný rozměr, velikost dávky, promíchání s parametrem a informace o tom, že se jedná o snímky jedné třídy.

```
1 normalize = ImageDataGenerator(rescale=1./255)
2 train_ds = normalize.flow_from_directory(train_dir,
3                                          target_size = (256,256),
4                                          batch_size = 16,
5                                          shuffle=True, seed=3,
6                                          class_mode = 'input')
```

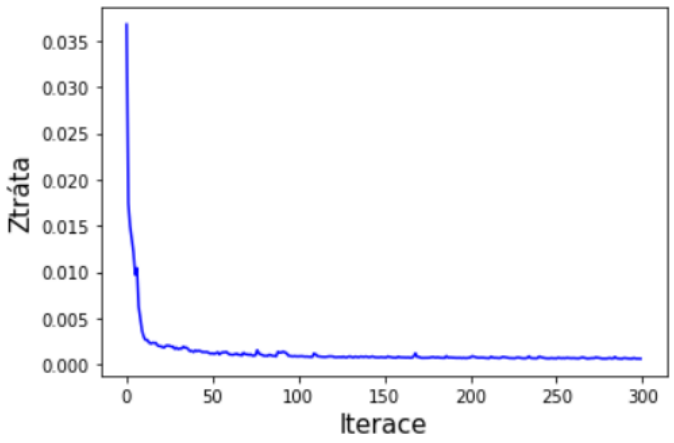
Výpis kódu 3.13: Předzpracování dat

Struktura modelu:

Navržená struktura je sekvenčním model s postupně propojenými vrstvami. Její první částí je enkodér. Ten podobně jako konvoluční neuronové sítě předchozích modelů využívá vrstev $Conv2D$ a aktivační funkce $ReLU$, pro zvýšení počtu map příznaků a následné $MaxPooling2D$ vrstvy k snížení jejich rozměru. Tímto způsobem vstupní snímek komprimuje do rozměru 8×8 pixelů a reprezentuje ho pomocí 512-ti map příznaků. Ty nesou důležité informace, potřebné pro zpětnou rekonstrukci snímku. Ta probíhá v druhé části modelu, kterou je dekodér. Ten reprezentaci dekomprimuje tak aby jeho výstup byl stejný jako vstup enkodéru, tedy snímek o rozměrech 256×256 pixelů se třemi barevnými kanály. K zvýšení rozměru, tentokrát snižujícího se počtu příznakových map využívá vrstev $UpSampling2D$. Poslední konvoluční vrstva dekodéru využívá aktivační funkce Sigmoid, a to z toho důvodu aby výstupní a vstupní snímky měli stejný rozsah hodnot pixelů, tedy od 0 do 1.

Učení modelu:

Tento model oproti předchozím nemá rozhodovací funkci ale snaží se co nejlépe komprimovat a dekomprimovat vstupní snímky. Při jeho učení tak docházelo k porovnání jednotlivých hodnot pixelů vstupního a výstupního snímku. Toho bylo dosaženo nastavením konfigurace a kompilace modelu, kde byla ztrátovou funkcí střední kvadratická chyba. Ta v procesu učení sloužila jako ztrátové skóre pro optimalizační algoritmus, kterým se stal algoritmus $Adam$. Ten byl ponechán ve výchozím nastavení, tedy s krokem učení 1×10^{-3}. Učení modelu proběhlo pomocí metody $model.fit$, kde bylo nastaveno 300 iterací. Vstupními daty v tomto případě byly pouze snímky bez vady. To jak se v průběhu těchto iterací vyvíjelo ztrátové skóre lze vidět na obrázku 3.11. Ztrátové skóre dosáhlo v poslední iteraci hodnoty 6.0549×10^{-4}.

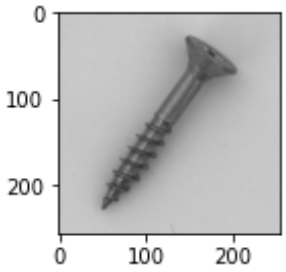


Obrázek 3.11: Ztrátové skóre v průběhu učení autoenkodéru

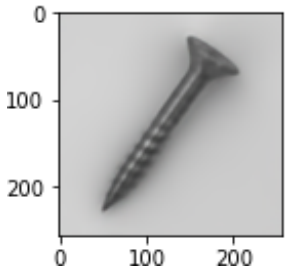
K vizualizaci toho jak takto naučený model dokáže vstupní snímky rekonstruovat, slouží sada obrázků 3.12, kde se vlevo nachází vstupní snímek a vpravo jeho zrekonstruovaná verze po průchodu naučeným autoenkodérem.

Tabulka 3.7: Struktura autoenkodéru

Typ vrstvy	**Výstupní rozměr**	**Aktivační funkce**	**Rozměr filtru**	**Počet parametrů**
		Enkodér		
Conv2D	(256,256,32)	ReLU	(3,3)	896
MaxPooling2D	(128,128,32)	-	(2,2)	0
Conv2D	(128,128,64)	ReLU	(3,3)	18496
MaxPooling2D	(64,64,64)	-	(2,2)	0
Conv2D	(64,64,128)	ReLU	(3,3)	73856
MaxPooling2D	(32,32,128)	-	(2,2)	0
Conv2D	(32,32,256)	ReLU	(3,3)	295168
MaxPooling2D	(16,16,256)	-	(2,2)	0
Conv2D	(16,16,512)	ReLU	(3,3)	1180160
MaxPooling2D	(8,8,512)	-	(2,2)	0
		Dekodér		
Conv2D	(8,8,512)	ReLU	(3,3)	2359808
Upsampling2D	(16,16,512)	-	(2,2)	0
Conv2D	(16,16,256)	ReLU	(3,3)	1179904
Upsampling2D	(32,32,256)	-	(2,2)	0
Conv2D	(32,32,128)	ReLU	(3,3)	295040
Upsampling2D	(64,64,128)	-	(2,2)	0
Conv2D	(64,64,64)	ReLU	(3,3)	73792
Upsampling2D	(128,128,64)	-	(2,2)	0
Conv2D	(128,128,32)	ReLU	(3,3)	18464
Upsampling2D	(256,256,32)	-	(2,2)	0
Conv2D	(256,256,3)	Sigmoid	(3,3)	867
Počet parametrů k učení:				5496451
Celkový počet parametrů:				5496451



(a) Vstupní snímek

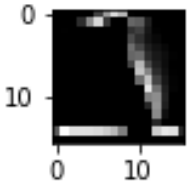


(b) Zrekonstruovaný snímek

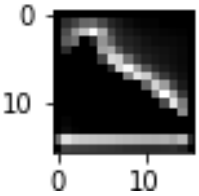
Obrázek 3.12: Rekonstrukce pomocí autoenkodéru

Způsob klasifikace snímků:

Autoenkodér dokáže vzhledem k snímkům na kterých byl trénován dobře rekonstruovat snímky bez vady. Jinak tomu však je při rekonstrukci snímků s vadou. K sledování rozdílů bylo využito dvou parametrů. Prvním z nich se stala rekonstrukční chyba mezi snímkem který do autoenkodéru vstupuje a tím, který z něj vystupuje. Tu zastupuje samotná střední kvadratická chyba. Druhým parametrem se staly hodnoty jádrových odhadů hustoty. Pro jejich získání bylo využito pouze části enkodéru bez poslední $MaxPooling2D$ vrstvy. Jedná se tak o zkomprimované reprezentace snímku o rozměrech 16×16 pixelů, kterých je celkem 512. Na obrázku 3.13 pak lze vidět jednu z reprezentací této části pro snímek vrutu s vadou a bez.



(a) Vrut bez vady



(b) Vrut s vadou

Obrázek 3.13: Ukázka příznakových map

Vstupními hodnotami jádrového odhadu hustoty jsou tedy hodnoty pixelů těchto komprimovaných reprezentací. Samotný jádrový odhad je definován pomocí funkce $sklearn.neighbors.KernelDensity$ z knihovny $Scikit-learn$, ve které je parametrem použité gaussovské jádro s patřičnou šířkou $band\_with = 0{,}2$. Nejprve je definován odhad hustoty na trénovacích datech použitím metody $.fit$ a parametru $encoded\_images\_vector$, který odpovídá hodnotám zkomprimované reprezentace snímků v trénovacím souboru dat. Vůči tomuto odhadu byl sledován rozdíl pomocí metody $.score\_samples$.

```
Density = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(
    encoded_images_vector)
```

Výpis kódu 3.14: Definice parametru aplikovaného jádra

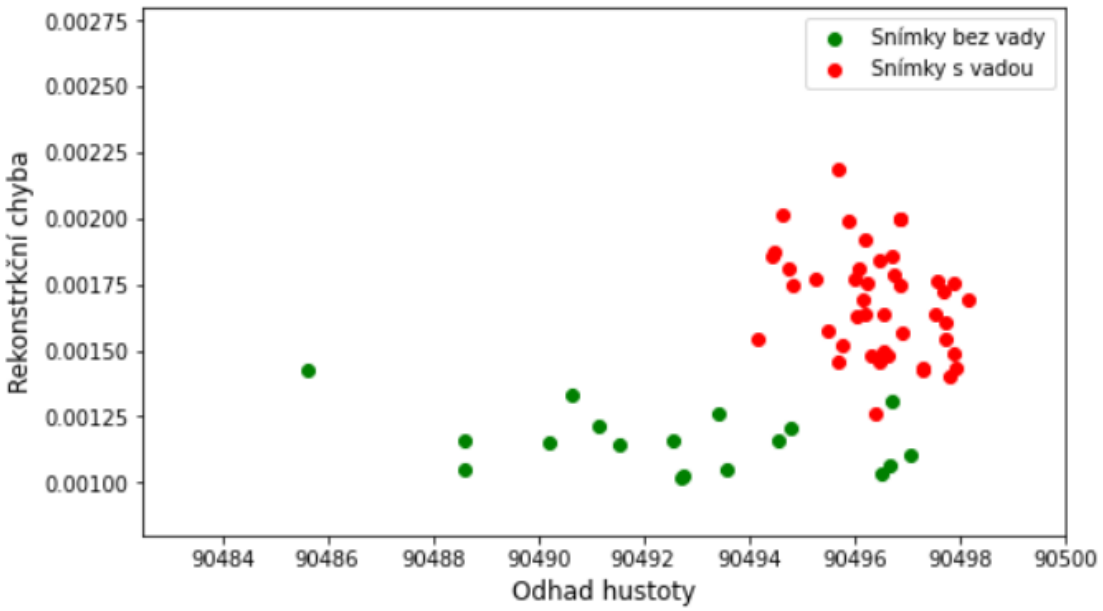
Takto definované hodnotící parametry bylo možno aplikovat na označený vzorek dat obsahující snímky s vadou a bez vady. Pro všechny snímky byla vyhodnocena

hodnota odhadu hustoty a rekonstrukční chyba. Následně byla vypočtena jejich průměrná hodnota, která je společně se směrodatnou odchylkou uvedena v tabulce 3.8. Jak lze vidět, průměrné hodnoty sledovaných parametrů u snímků s vadou a bez vady se od sebe navzájem liší.

Tabulka 3.8: Hodnoty hustoty a rekonstrukční chyby

	Hustota [-]	**Rekonstrukční chyba [-]**
Snímky bez vady	$90492,180(\pm 3,094)$	$115,945(\pm 11,269) \times 10^{-5}$
Snímky s vadou	$90496,436(\pm 1,044)$	$168,297(\pm 19,858) \times 10^{-5}$

Pro lepší znázornění byl vytvořen dvojrozměrný graf, který lze vidět na obrázku 3.14. Na ose y leží hodnoty rekonstrukčních chyb každého snímku a na ose x pak jejich hodnoty odhadů jádrové hustoty.

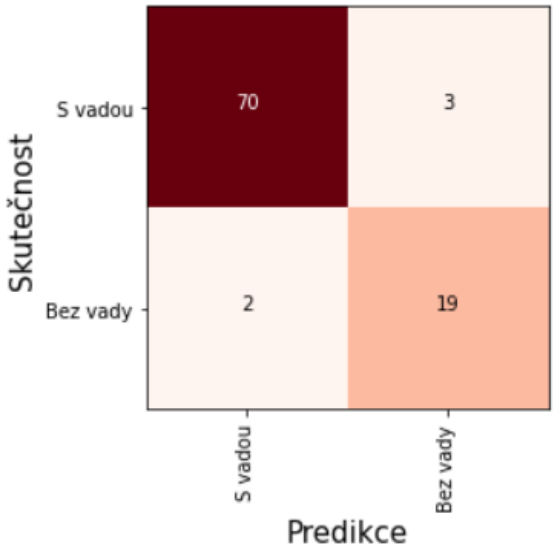


Obrázek 3.14: Odhady hustoty a rekonstrukčních chyb pro jednotlivé snímky

Díky získaným hodnotám a vizuální analýze vytvořeného grafu mohla být stanovena podmínka, která bude v rámci funkce hodnotit jednotlivé snímky s rekonstrukční chybou menší než $0,00148$ a hustotou menší než 90496 jako snímky bez vady.

Vyhodnocení modelu:

Pro konečné vyhodnocení modelu na testovacích datech bylo využito definované funkce s podmínkou využívající naučený autoenkodér. K ověření její schopnosti klasifikovat snímky bylo využito snímků z testovacího souboru dat, tedy 21 snímků bez vady a 73 snímků s vadou. Získaná predikovaná kategorie snímku byla společně s jeho skutečným označením zapisována do seznamu hodnot, díky kterému bylo možné schopnost klasifikace znázornit v matici záměn (obrázek 3.15).



Obrázek 3.15: Matice záměn získaná pomocí autoenkodéru

Díky této matici bylo možno vyhodnotit přesnost, úplnost, F1-skóre a jejich vážené průměry (tabulka 3.9). Výsledná celková správnost predikcí dosahovala 95%.

Tabulka 3.9: Vyhodnocení autoenkodéru

Kategorie	**Přesnost** [%]	**Úplnost** [%]	**F1-skóre** [-]	**Počet snímků**
Bez vady	86	90	0,88	21
S vadou	97	96	0,97	73
Vážený průměr	95	95	0,95	

3.6 Způsob možného nasazení do výroby

Výrobě vrutu předchází mnoho procedur, kdy je polotovar ve formě drátu postupně přetvořen do výsledného tvaru. Poslední část výroby, kterou je tvorba závitu probíhá na válcovacích zařízeních pomocí plochých válcovacích čelistí. Jedná se o dvě čelisti s drážkami v podobě negativního profilu závitu z nichž jedna stojí a druhá koná přímočarý vratný pohyb. Při každém zdvihu je zhotoven jeden vrut.

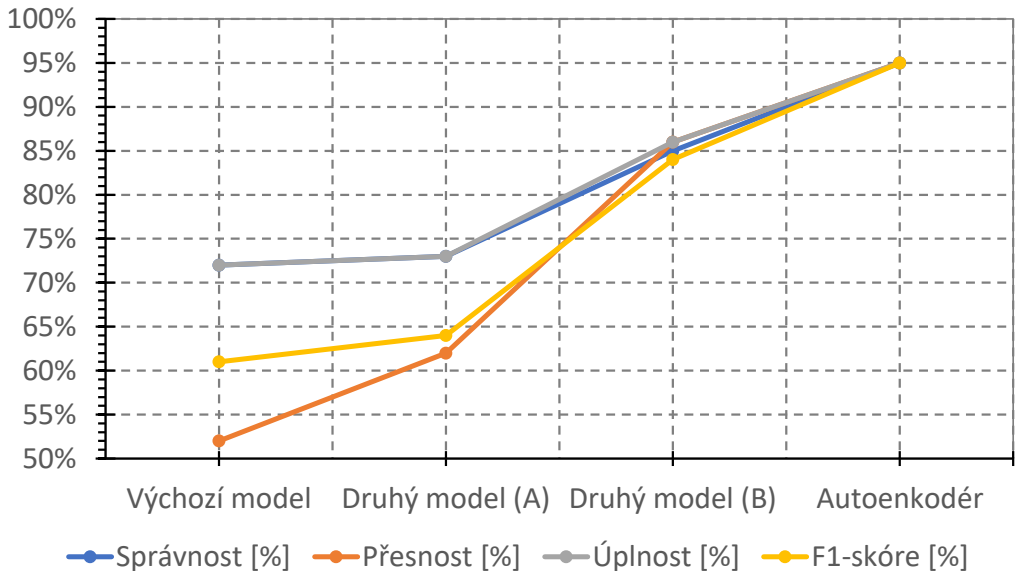
Implementace kontroly vad by mohla probíhat přímo za touto částí výrobní linky, kde by padající vruty vedené lištou mohly být kontrolovány pomocí průmyslové kamery. Jejich třídění by bylo zajištěno pohyblivou přepážkou nebo proudem stlačeného vzduchu. Pro celý proces pořízení snímku, jeho vyhodnocení a provedení akčního zásahu by musel být naprogramován celistvý program, jelikož tato práce se zabývá pouze tvorbou a trénováním modelů. Poslední vytvořený model autoenkodéru by však sloužil v samém jádru tohoto programu k vyhodnocení toho zda se jedná o snímek vrutu s vadou nebo bez.

Model tedy lze využít v rámci propojení průmyslové kamery s počítačem nebo pomocí individuálního zařízení. V případě individuálního zařízení by se jednalo o minipočítač Raspberry Pi s vlastní kamerou a nainstalovanou 64-bitovou distribucí Ubuntu. Pro možnost použití vytvořených modelů by bylo zapotřebí instalace knihovny TensorFlow Lite. Při jeho použití nedochází k přepočtu optimalizovaných vah. Zařízení by tak nebylo vytíženo a stíhalo podávat potřebné informace o přítomnosti vad. Rozhodujícím faktorem by pak byla rychlost klasifikace. Autoenkodér stíhal v rámci definované funkce klasifikovat jeden snímek za 0,2 sekundy, což by pro produkci mělo být dostačující. S procesem kontroly by mohlo souběžně docházet také ke sběru dalších dat a přenosu do individuálního systému. Ten by prováděl dodatečné trénování a optimalizaci modelu, který by mohl být v rámci údržby přenesen do individuálního zařízení.

4 Závěr

Cílem této bakalářské práce bylo navržení modelu umělé inteligence pro automatickou detekci vadných výrobků. V teoretické části proto byla nejprve představena motivace a důvod použití metod hlubokého učení v oblasti kontroly kvality a současný stav poznání. Dále jsou zde představeny základní pojmy z oblasti strojového vidění a umělé inteligence. Podrobněji je zde popsána oblast strojového učení a využívaných technik, na které navazuje přístup hlubokého učení. To ve svém jádru využívá neuronových sítí, které jsou zde popsány včetně procesu jejich učení. Navazující kapitoly se zabývají popisem speciálních typů těchto sítí pro zpracování a klasifikaci obrazu a to sice konvolučními neuronovými sítěmi a autoenkodéry. Dále je zde popsána metoda přenosu učení, společně se strukturou, která byla využita v praktické části práce.

Samotná praktická část se zpočátku zabývá popisem vybraných vstupních dat, kterými se z hlediska zaměření a možnosti nasazení metod popsaných v teoretické části staly snímky vrutů. Je zde popsána úprava původní struktury a způsoby modifikací potřebných pro učení. Z hlediska strukturovanosti souborů, ve kterých se tyto snímky nachází, bylo snahou prvních dvou navržených modelů klasifikovat jednu z šesti definovaných kategorií. Poslední vytvořený model autoenkodéru však využívá jiných technik a omezuje se tak pouze na klasifikaci toho zda se jedná o snímek vrutu s vadou nebo bez. Výchozí strukturou se stal vlastní návrh modelu konvoluční neuronové sítě. Výsledná správnost klasifikace snímků však při jeho testování nepřesáhla hodnotu $72,3\%$, jelikož všem snímkům vrutů přidělil označení „bez vady". To poukázalo na nevyváženost snímků v souboru vstupních dat i přes využití umělého rozšíření. V rámci ladění a zvyšování přesnosti tak vznikl druhý model. Ten přináší vylepšení zejména v samotné struktuře, která využívá techniky přenosu učení. Tento model při učení na stejném souboru snímků jako první model dosáhl výsledné správnosti $73,3\%$. To vedlo k úpravě vstupních dat ve smyslu natočení vrutů ve snímcích do stejné polohy, což vedlo k razantnímu pokroku ve výsledné správnosti, která dosáhla $85,1\%$. Použití tohoto modelu by však při jeho implementaci do výroby znamenalo to, že by vruty musely být natáčeny do stejné polohy také v průběhu jejich kontroly což není příliš praktické. Posledním navrženým modelem je autoenkodér, ten pro hodnocení využívá technik detekce anomálií a jeho výsledná klasifikace dokáže odhalit pouze to zda se jedná o vrut s vadou a nebo bez. Díky využití tohoto modelu však bylo dosaženo správnosti 95%. V rámci testování všech modelu byla hodnocena také přesnost, úplnost a F1-skóre. Tyto hodnoty vyplývají z vytvořených matic záměn a jejich vážené průměry v rámci klasifikovaných kategorií lze vidět v obrázku 4.1.



Obrázek 4.1: Vývoj hodnocených metrik

Je známo, že čím menší je rozdíl mezi metrikami, tím více je model důvěryhodný při jeho nasazení. Z obrázku 4.1 lze vidět že model využívající přenosu učení a stejně natočených vruty (Druhý model (B)) a model autoenkodér v tomto směru dosáhly nejlepších výsledků. Vzhledem k tomu, že F1-skóre autoenkodéru je vyšší, můžeme s ním počítat jako s výsledným modelem pro možnou implementaci do systému kontroly kvality. To je detailněji popsáno v kapitole 3.6.

Možný směr pokračování výzkumu by spočíval ve vylepšení rozhodovací části autoenkodéru. Vylepšení by spočívalo ve využití metod učeni bez učitele, například algoritmu k-means se dvěma shluky. Dalším perspektivním směrem by byla lokalizace přesného místa anomálie, která by umožnila identifikaci typu závady.

Použitá literatura

[1] SPEJCHALOVÁ, Dana. *Management kvality*. VSEM, 2011. ISBN 978-80--86730-68-4. Google-Books-ID: UZO8PwdOlggC.

[2] *Optická kontrola kvality: Rychlá, precizní a v duchu Průmyslu 4.0* [Svět průmyslu] [online]. 2021-12-15 [cit. 2022-04-25]. Dostupné z: https://svetprumyslu.cz/2021/12/15/opticka-kontrola-kvality-rychla-precizni-a-v-duchu-prumyslu-4-0/.

[3] *Počítačové vidění a strojírenství: ideální symbióza | Kinali* [online] [cit. 2022-04-13]. Dostupné z: https://www.kinali.cz/cs/clanky/pocitacove-videni-a-strojirenstvi-idealni-symbioza/.

[4] *Optická kontrola kvality vyniká rychlostí i spolehlivostí | Kinali* [online] [cit. 2022-04-13]. Dostupné z: https://www.kinali.cz/cs/clanky/opticka-kontrola-kvality-vynika-rychlosti-i-spolehlivosti/.

[5] *Hluboké učení + strojové vidění = kontrola kvality nové generace* [Vše o průmyslu] [online] [cit. 2022-05-09]. Dostupné z: https://www.vseoprumyslu.cz/inspirace/firemni-novinky/hluboke-uceni-strojove-videni-kontrola-kvality-nove-generace.html.

[6] *Deep Learning Inspections for Automotive Industry | Cognex* [online] [cit. 2022-06-09]. Dostupné z: https://www.cognex.com/blogs/deep-learning/deep-learning-for-automotive-industry.

[7] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E HINTON. ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems* [online]. Curran Associates, Inc., 2012, sv. 25 [cit. 2022-06-09]. Dostupné z: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[8] *Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection | Elsevier Enhanced Reader* [online] [visited on 2022-06-21]. Available from DOI: 10.1016/j.cirp.2016.04.072.

[9] *DAGM 2007* [online] [cit. 2022-06-21]. Dostupné z: https://conferences.mpi-inf.mpg.de/dagm/2007/prizes.html.

[10] *Audi optimizes quality inspections in the press shop with artificial intelligence* [Audi MediaCenter] [online] [visited on 2022-06-09]. Available from: https://www.audi-mediacenter.com:443/en/press-releases/audi-optimizes-quality-inspections-in-the-press-shop-with-artificial-intelligence-10847.

[11] SZELISKI, Richard. Computer Vision: Algorithms and Applications, 2nd Edition. [N.d.], p. 1232.

[12] HAVLE, Otto. Strojové vidění I: Principy a charakteristiky. [B.r.], s. 4.

[13] CHOLLET, François. *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018. ISBN 978-1-61729-443-3. OCLC: ocn982650571.

[14] RAY, Susmita. A Quick Review of Machine Learning Algorithms. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019, s. 35–39. Dostupné z DOI: 10.1109/COMITCon.2019.8862451.

[15] *Supervised vs Unsupervised Learning - Javatpoint* [www.javatpoint.com] [online] [visited on 2022-05-16]. Available from: https://www.javatpoint.com/difference-between-supervised-and-unsupervised-learning.

[16] KIRAN, B. Ravi, Dilip Mathew THOMAS, and Ranjith PARAKKAL. An Overview of Deep Learning Based Methods for Unsupervised and Semi-Supervised Anomaly Detection in Videos. *Journal of Imaging* [online]. 2018, vol. 4, no. 2, p. 36 [visited on 2022-05-16]. ISSN 2313-433X. Available from DOI: 10.3390/jimaging4020036. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[17] *Explained: Neural networks* [MIT News | Massachusetts Institute of Technology] [online] [visited on 2022-05-11]. Available from: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[18] *A Single Neuron* [online] [visited on 2022-05-16]. Available from: https://kaggle.com/ryanholbrook/a-single-neuron.

[19] *Matematická biologie učebnice: Matematický model a aktivní dynamika neuronu* [online] [cit. 2022-05-17]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--jednotlivy-neuron--matematicky-model-a-aktivni-dynamika-neuronu.

[20] JAIN, A.K., Jianchang MAO a K.M. MOHIUDDIN. Artificial neural networks: a tutorial. *Computer*. 1996, roč. 29, č. 3, s. 31–44. ISSN 1558-0814. Dostupné z DOI: 10.1109/2.485891. Conference Name: Computer.

[21] *What are Neural Networks?* [Online]. 2021-08-03 [visited on 2022-05-11]. Available from: https://www.ibm.com/cloud/learn/neural-networks.

[22] NWANKPA, Chigozie et al. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning* [online]. 2018-11-08 [cit. 2022-06-01]. arXiv:1811.03378. arXiv. Dostupné z arXiv: 1811.03378[cs]. type: article.

[23] NALBORCZYK, Ladislas et al. Can we decode phonetic features in inner speech using surface electromyography? *PLOS ONE* [online]. 2020, vol. 15, no. 5, e0233282 [visited on 2022-06-02]. ISSN 1932-6203. Available from DOI: 10.1371/journal.pone.0233282.

[24] SYDENHAM, P. H. and THORN, Richard (eds.). *Handbook of measuring system design*. Chichester, England: Wiley, 2005. ISBN 978-0-470-02143-9.

[25] WANG, Qi et al. A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science* [online]. 2022, vol. 9, no. 2, pp. 187–212 [visited on 2022-06-07]. ISSN 2198-5804, ISSN 2198-5812. Available from DOI: 10.1007/s40745-020-00253-5.

[26] JUNG, Alexander. *Machine Learning: The Basics* [online]. 2022-01-29 [cit. 2022-06-07]. arXiv:1805.05052. arXiv. Dostupné z arXiv: 1805.05052[cs,stat]. type: article.

[27] BROWNLEE, Jason. *A Gentle Introduction to Cross-Entropy for Machine Learning* [Machine Learning Mastery] [online]. 2019-10-20 [cit. 2022-06-07]. Dostupné z: https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[28] KOECH, Kiprono Elijah. *Cross-Entropy Loss Function* [Medium] [online]. 2021-11-25 [visited on 2022-06-07]. Available from: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[29] SOYDANER, Derya. A Comparison of Optimization Algorithms for Deep Learning. *International Journal of Pattern Recognition and Artificial Intelligence* [online]. 2020, roč. 34, č. 13, s. 2052013 [cit. 2022-06-08]. ISSN 0218-0014, ISSN 1793-6381. Dostupné z DOI: 10.1142/S0218001420520138.

[30] RUDER, Sebastian. *An overview of gradient descent optimization algorithms* [online]. 2017-06-15 [cit. 2022-06-07]. arXiv:1609.04747. arXiv. Dostupné z arXiv: 1609.04747[cs]. type: article.

[31] CERIGO, Daniel Burkhardt. *On Why Gradient Descent is Even Needed* [Medium] [online]. 2018-10-30 [visited on 2022-06-07]. Available from: https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635.

[32] KINGMA, Diederik P. a Jimmy BA. *Adam: A Method for Stochastic Optimization* [online]. 2017-01-29 [cit. 2022-06-06]. arXiv:1412.6980. arXiv. Dostupné z arXiv: 1412.6980[cs]. type: article.

[33] KATHIRKAMAR, Vignesh. *An Introduction to image analysis using OpenCV* [Analytics Vidhya] [online]. 2020-08-31 [visited on 2022-06-08]. Available from: https://medium.com/analytics-vidhya/an-introduction-to-image-analysis-using-opencv-2ce8db47e05c.

[34] O'SHEA, Keiron a Ryan NASH. *An Introduction to Convolutional Neural Networks* [online]. 2015-12-02 [cit. 2022-06-07]. arXiv:1511.08458. arXiv. Dostupné z arXiv: 1511.08458[cs]. type: article.

[35] CIRESAN, Dan Claudiu et al. Convolutional Neural Network Committees for Handwritten Character Classification. In: *2011 International Conference on Document Analysis and Recognition*. 2011, s. 1135–1139. Dostupné z DOI: 10.1109/ICDAR.2011.229. ISSN: 2379-2140.

[36] PATEL, Krut. *MNIST Handwritten Digits Classification using a Convolutional Neural Network (CNN)* [Medium] [online]. 2019-12-01 [visited on 2022-06-09]. Available from: https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9.

[37] *CNN for Deep Learning | Convolutional Neural Networks* [Analytics Vidhya] [online]. 2021-05-01 [visited on 2022-06-08]. Available from: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/.

[38] *Max pooling in Convolutional neural network — MATLAB Number ONE* [online] [cit. 2022-06-08]. Dostupné z: https://matlab1.com/max-pooling-in-convolutional-neural-network/.

[39] EHSANI, Narges, Farrokh AMINIFAR, and Hamed MOHSENIAN-RAD. Convolutional autoencoder anomaly detection and classification based on distribution PMU measurements. *IET Generation, Transmission & Distribution* [online]. [N.d.], vol. n/a [visited on 2022-05-24]. ISSN 1751-8695. Available from DOI: 10.1049/gtd2.12424. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/gtd2.12424.

[40] BEGGEL, Laura, Michael PFEIFFER a Bernd BISCHL. *Robust Anomaly Detection in Images using Adversarial Autoencoders* [online]. arXiv, 2019 [cit. 2022-06-26]. Č. arXiv:1901.06355. Dostupné z arXiv: 1901.06355[cs,stat].

[41] *High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning | Elsevier Enhanced Reader* [online] [visited on 2022-06-26]. Available from DOI: 10.1016/j.patcog.2016.03.028.

[42] *Convolutional Neural Network Architecture | CNN Architecture* [online] [cit. 2022-05-24]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.

[43] MISHRA, Divyanshu. *Transposed Convolution Demystified* [Medium] [online]. 2021-07-25 [visited on 2022-06-21]. Available from: https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba.

[44] TORREY, Lisa and Jude SHAVLIK. *Transfer Learning* [Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques] [online]. 2010 [visited on 2022-06-17]. Available from DOI: 10.4018/978-1-60566-766-9.ch011. ISBN: 9781605667669 Pages: 242-264 Publisher: IGI Global.

[45] *Papers with Code - INRIA Aerial Image Labeling Dataset* [online] [visited on 2022-06-21]. Available from: https://paperswithcode.com/dataset/inria-aerial-image-labeling.

[46] SIMONYAN, Karen a Andrew ZISSERMAN. *Very Deep Convolutional Networks for Large-Scale Image Recognition* [online]. arXiv, 2015 [cit. 2022-06-21]. Č. arXiv:1409.1556. Dostupné z arXiv: 1409.1556[cs]. Number: arXiv:1409.1556.

[47] *VGG Very Deep Convolutional Networks (VGGNet) - What you need to know* [viso.ai] [online]. 2021-10-06 [cit. 2022-06-21]. Dostupné z: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/.

[48] MARIN, Ivana et al. Deep-Feature-Based Approach to Marine Debris Classification. *Applied Sciences* [online]. 2021, vol. 11, no. 12, p. 5644 [visited on 2022-06-21]. ISSN 2076-3417. Available from DOI: 10.3390/app11125644. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[49] BERGMANN, Paul et al. MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. Long Beach, CA, USA: IEEE, 2019, pp. 9584–9592 [visited on 2022-03-30]. ISBN 978-1-72813-293-8. Available from DOI: 10.1109/CVPR.2019.00982.

Přílohy

A Zdrojové kódy

Programy, na které bylo odkazováno v rámci praktické části práce se nachází v následujících přílohách A.1 až A.5.

Veškeré zdrojové kódy, použitá vstupní data a uložené modely jsou volně přístupné v repozitáři na platformě GitHub, pomocí následujícího odkazu:
https://github.com/petrsima/Batchelor-thesis.git.

A.1 Úprava struktury vstupních dat

```
import matplotlib.pyplot as plt
import numpy as np
import os
import shutil
from tqdm.auto import tqdm
import time

def ds_distribution(org_ds_path,distribution):
    os.chdir(r'\Users\HP\Documents\GIT-BP')
    ds_split_names = ['train', 'valid', 'test']
    src_orig = org_ds_path + '\\test'
    categories = os.listdir(src_orig)
    new_ds_path = "Screw_ds(" + str(distribution[0]) + "," + str(
   distribution[1]) + "," + str(distribution[2]) + ")"

    if not os.path.exists(new_ds_path):
        os.makedirs(new_ds_path)
        os.chdir(new_ds_path)
        new_ds_dir=os.getcwd()

        for i in ds_split_names:
            if not os.path.exists(i):
                os.makedirs(i)
                os.chdir(i)
                for j in categories:
                    os.makedirs(j)
            os.chdir(new_ds_dir)

        all_new = new_ds_dir + '\\All_images'
        shutil.copytree(src_orig, all_new)

        train_good_imgs_dir = org_ds_path + '\\train\good'
        train_good_imgs_list = os.listdir(train_good_imgs_dir)
        for n in train_good_imgs_list:
            shutil.copy(train_good_imgs_dir + "\\" + n,
            all_new + "\\good" +"\\0" + n)

        category_list = os.listdir(all_new)
        [ a, b, c] = distribution
        ds_split_names = ['train', 'valid', 'test']
        dst_ = new_ds_dir + "\\"
        dst_train = dst_ + ds_split_names[0]
        dst_valid = dst_ + ds_split_names[1]
        dst_test = dst_ + ds_split_names[2]

        for i in tqdm(category_list):
            cat = all_new + "\\" + i
            img_names = os.listdir(cat)
            num_of_items = len(img_names)
            a_items = int(num_of_items*(distribution[0]/100))
            b_items = int(num_of_items*(distribution[1]/100))
```

```
            c_items = int(num_of_items*(distribution[2]/100))
            for a in img_names[:a_items]:
                shutil.copy(cat + "\\" + a,
                            dst_train + "\\" + i + "\\" + a)
            for b in img_names[a_items:(a_items+b_items)]:
                shutil.copy(cat + "\\" + b,
                            dst_valid + "\\" + i + "\\" + b)
            for c in img_names[(a_items+b_items):]:
                shutil.copy(cat + "\\" + c,
                            dst_test + "\\" + i + "\\" + c)
        time.sleep(0.1)

        print("New dataset folder made! \n \n" + "Dataset directory:\t"
    + new_ds_dir)
        os.chdir(r'\Users\HP\Documents\GIT-BP')
        shutil.rmtree(all_new)
    else:
        os.chdir(r'\Users\HP\Documents\GIT-BP')
        print("Already made!")

org_ds_path = r"C:\Users\HP\Documents\GIT-BP\screw_orig"
distribution = [50, 30, 20]
f = ds_distribution(org_ds_path,distribution)
```

A.2 Natočení vrutů stejným směrem

```
import matplotlib.pyplot as plt
import numpy as np
import os
import shutil
import cv2 as cv
from tqdm.auto import tqdm
import time
import PIL
from PIL import Image
import pathlib
import pandas as pd
import itertools

def ds_prep(ds_path,version):

    os.chdir(ds_path)
    ds_split_names = ['train', 'valid', 'test']
    org_ds_path = r"C:\Users\HP\Documents\GIT-BP\screw_orig"
    categories = os.listdir(r"C:\Users\HP\Documents\GIT-BP\screw_orig\test")
    new_ds_path = "Screw_ds-v" + str(version) + "(oriented)"

    if not os.path.exists(new_ds_path):
        os.makedirs(new_ds_path)
        os.chdir(new_ds_path)
        new_ds_dir=os.getcwd()
        print("New dataset folder made. \n \n" + "Dataset directory:\t" + new_ds_dir + " \t( " + str(len(categories)) + " categories )")

        for i in ds_split_names:
            if not os.path.exists(i):
                os.makedirs(i)
                os.chdir(i)
                for j in categories:
                    os.makedirs(j)
            os.chdir(new_ds_dir)
    else:
        print("Use new version number.")


    src_orig = org_ds_path + '\\test'
    dst_new = new_ds_dir + "\\All_images"
    shutil.copytree(src_orig, dst_new)

    for k in categories:
        mask_origin = dst_new + "\\" + k
        src_img_list = os.listdir(mask_origin)
        mask_src = mask_origin + "\\" + src_img_list[0]
        mask_dst = mask_origin + "\\0_MASK.png"
        os.rename(mask_src,mask_dst)

```

```
    train_good_imgs_dir = r"C:\Users\HP\Documents\GIT-BP\screw_orig\train\good"
    train_good_imgs_list = os.listdir(r"C:\Users\HP\Documents\GIT-BP\screw_orig\train\good")
    for n in train_good_imgs_list:
        shutil.copy(train_good_imgs_dir + "\\" + n,
                    dst_new + "\\good" +"\\0" + n)

    for k in tqdm(categories[1:],desc = "Matching all category templates "):
        cv_mask_0 = cv.imread(dst_new + "\\" + categories[0] + "\\0_MASK.png")
        org_mask_path = dst_new + "\\" + k + "\\0_MASK.png"
        org_mask_copy_path = dst_new + "\\" + k + "\\0_MASK(copy).png"
        shutil.copy(org_mask_path,
                    org_mask_copy_path)
        cv_nrot_mask_copy = cv.imread(org_mask_copy_path)
        m_h, m_w = cv_nrot_mask_copy.shape[:2]
        m_center = (m_w/2, m_h/2)
        mask_Value = list()
        mask_Phi = list()

        for mask_phi in range(0,360):
            mask_rot_mtx = cv.getRotationMatrix2D(center=m_center, angle=mask_phi, scale=1)
            rot_mask_copy = cv.warpAffine(src=cv_nrot_mask_copy,
                                          M=mask_rot_mtx,
                                          dsize=(m_w, m_h),
                                          borderMode=cv.BORDER_REPLICATE)
            rot_mask_match = cv.matchTemplate(cv_mask_0,rot_mask_copy, cv.TM_SQDIFF_NORMED)
            min_mask_val, max_val, min_loc, max_loc = cv.minMaxLoc(rot_mask_match)
            mask_Value.append(min_mask_val)
            mask_Phi.append(mask_phi)

        mask_Phi_index = mask_Value.index(min(mask_Value))
        nrot_mask_cv = cv.imread(org_mask_path)
        f_rot_mtx = cv.getRotationMatrix2D(center=m_center, angle=mask_Phi[mask_Phi_index], scale=1)
        f_rot_mask = cv.warpAffine(src=nrot_mask_cv,
                                   M=f_rot_mtx,
                                   dsize=(m_w, m_h),
                                   borderMode=cv.BORDER_REPLICATE)
        f_img = cv.imwrite(org_mask_path,f_rot_mask)
        os.remove(org_mask_copy_path)
    time.sleep(0.1)

    for k in tqdm(categories, desc = "Matching images in each category "):
        nrot_img_path = dst_new + "\\" + k
        nrot_img_dirs = os.listdir(nrot_img_path)
        list_of_nrot_images = nrot_img_dirs[::-1]

```

```
        for i in tqdm(range(1,len(list_of_nrot_images)),desc = f"Matching category : {k} "):
            img_mask = cv.imread(nrot_img_path + "\\0_MASK.png")
            org_img_path = nrot_img_path + "\\" + str(list_of_nrot_images[i])
            org_img_copy_path = nrot_img_path + "\\copy_" + str(list_of_nrot_images[i])
            shutil.copy(org_img_path,
                        org_img_copy_path)
            Value = list()
            Phi = list()
            cv_org_img_copy = cv.imread(org_img_copy_path)
            i_h, i_w = cv_org_img_copy.shape[:2]
            i_center = (i_w/2, i_h/2)

            for phi in range(0,360):
                img_rot_mtx = cv.getRotationMatrix2D(center=i_center, angle=phi, scale=1)
                rot_img_copy = cv.warpAffine(src=cv_org_img_copy,
                                             M=img_rot_mtx,
                                             dsize=(i_w, i_h),
                                             borderMode=cv.BORDER_REPLICATE)
                rot_img_match = cv.matchTemplate(img_mask,rot_img_copy,cv.TM_SQDIFF_NORMED)
                min_val, max_val, min_loc, max_loc = cv.minMaxLoc(rot_img_match)
                Value.append(min_val)
                Phi.append(phi)

            Phi_index = Value.index(min(Value))
            org_img_cv = cv.imread(org_img_path)
            f_rot_mtx = cv.getRotationMatrix2D(center=i_center, angle=Phi[Phi_index], scale=1)
            f_rot_img = cv.warpAffine(src=org_img_cv,
                                      M=f_rot_mtx,
                                      dsize=(i_w, i_h),
                                      borderMode=cv.BORDER_REPLICATE)
            f_img = cv.imwrite(org_img_path,f_rot_img)
            os.remove(org_img_copy_path)
        time.sleep(0.1)

os.chdir(r'\Users\HP\Documents\GIT-BP')

ds_path = r"C:\Users\HP\Documents\GIT-BP"
h = ds_prep(ds_path,1.0)
```

A.3 Výchozí model

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers , callbacks
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers.experimental import preprocessing
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
import pathlib
import pandas as pd
from sklearn.metrics import confusion_matrix , classification_report
import itertools

dataset_dir = r'C:\Users\HP\Documents\GIT-BP\Screw_ds(50,30,20)'
train_dir = dataset_dir + '\\train'
valid_dir = dataset_dir + '\\valid'
test_dir  = dataset_dir + '\\test'

labels='inferred'
label_mode='categorical'
color_mode='rgb'
image_size=(256, 256)
batch_size= 32
shuffle=True
seed=3

train_ds = tf.keras.utils.image_dataset_from_directory(
                train_dir,
                labels=labels,
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
                shuffle=shuffle,
                seed=seed)

valid_ds=tf.keras.utils.image_dataset_from_directory(
                valid_dir,
                labels=labels,
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
                shuffle=shuffle,
                seed=seed)

test_ds=tf.keras.utils.image_dataset_from_directory(
                test_dir,
                labels=labels,
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
```

```
                shuffle=shuffle,
                seed=seed)

class_names = test_ds.class_names
print(class_names)

data_augmentation = tf.keras.Sequential([preprocessing.RandomFlip('
    horizontal')])

AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = (train_ds.cache().prefetch(buffer_size=AUTOTUNE))
test_ds = (test_ds.cache().prefetch(buffer_size=AUTOTUNE))
valid_ds = (valid_ds.cache().prefetch(buffer_size=AUTOTUNE))

normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
valid_ds = valid_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))

def cnn_model():
    model = tf.keras.Sequential()
    model.add(data_augmentation)
    model.add(tf.keras.layers.Conv2D(8, (5,5), activation = 'relu'))

    model.add(tf.keras.layers.MaxPooling2D((2,2)))
    model.add(tf.keras.layers.Conv2D(64, (3,3), activation = 'relu'))

    model.add(tf.keras.layers.MaxPooling2D((2,2)))
    model.add(tf.keras.layers.Conv2D(128, (2,2), activation = 'relu'))

    model.add(tf.keras.layers.MaxPooling2D((2,2)))

    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(32, activation='relu'))
    model.add(tf.keras.layers.Dense(6 , activation='softmax'))
    return model
model1 =  cnn_model()

optimizer =tf.keras.optimizers.Adam(learning_rate=1e-4)

model1.compile(optimizer = optimizer,
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])

early_stopping = callbacks.EarlyStopping(monitor='val_loss',
                                         min_delta=0.001,
                                         patience=50,
                                         verbose=1,
                                         mode='min',
                                         restore_best_weights=True)

history = model1.fit(train_ds,
```

```
                      validation_data = valid_ds,
                      epochs=300,
                      verbose = 1,
                      callbacks=early_stopping)

model1.summary(line_length=120, positions=[.33, .55, .67, 1.])

model_to_save = r"C:\Users\HP\Documents\GIT-BP\models\CNN0"
model1.save(filepath=model_to_save , save_format='h5')

print(history.history.keys())
def plot_accuracy_loss(history):

    fig = plt.figure(figsize=(16,10))

    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'],'b-', label = "Na trénovacích
    datech")
    plt.plot(history.history['val_accuracy'], 'r-', label = "Na
    validacních datech")
    plt.ylabel("Správnost",fontsize=15)
    plt.xlabel("Iterace",fontsize=15)
    plt.legend()

    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'],'b-', label = "Na trénovacích
    datech")
    plt.plot(history.history['val_loss'], 'r-', label = "Na validacních
     datech")
    plt.ylabel("Ztráta",fontsize=15)
    plt.xlabel("Iterace",fontsize=15)

    plt.legend()
    plt.show()

history_frame = pd.DataFrame(history.history)
print(history_frame.head())
plot_accuracy_loss(history)

test_loss, test_acc = model1.evaluate(test_ds)
print(str(test_acc*100) + '%')

image_predict = model1.predict(test_ds)
image_predict = np.argmax(image_predict, axis=1)
image_true = tf.concat([y for x, y in test_ds], axis=0)
image_true = np.argmax(image_true, axis=1)

cm = confusion_matrix(y_true=image_true, y_pred=image_predict)

def plot_confusion_matrix(cm, classes,cmap=plt.cm.Reds):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
    [1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Skutecnost',fontsize=15)
    plt.xlabel('Predikce',fontsize=15)

plot_confusion_matrix(cm=cm, classes=class_names)

print(classification_report(image_true, image_predict, target_names=
    class_names))
```

A.4 Druhý model

```
import matplotlib.pyplot as plt
import numpy as np
from tqdm.auto import tqdm
import time
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers , callbacks
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers.experimental import preprocessing
import pathlib
import pandas as pd
from sklearn.metrics import confusion_matrix , classification_report
import itertools


dataset_path = r"C:\Users\HP\Documents\GIT-BP\Screw_ds(50,30,20)_oriented"

train_path = pathlib.Path(dataset_path + '/train')
valid_path = pathlib.Path(dataset_path + '/valid')
test_path  = pathlib.Path(dataset_path + '/test')

labels='inferred'
label_mode='categorical'
color_mode='rgb'
image_size=(224, 224)
batch_size= 16
shuffle=True
seed=3

train_ds = tf.keras.utils.image_dataset_from_directory(
                train_path,
                labels=labels,
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
                shuffle=shuffle,
                seed=seed)

valid_ds = tf.keras.utils.image_dataset_from_directory(
                valid_path,
                labels=labels,
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
                shuffle=shuffle,
                seed=seed)

test_ds = tf.keras.utils.image_dataset_from_directory(
                test_path,
                labels=labels,
```

```
                label_mode=label_mode,
                color_mode=color_mode,
                image_size=image_size,
                shuffle=shuffle,
                seed=seed)

class_names = test_ds.class_names
print(class_names)

a,b = image_size
input_shape = (a,b,3)

plt.figure(figsize=(15, 15))
for images, labels in train_ds.take(1):
    for i in range(5):
        ax = plt.subplot(5, 5, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[np.argmax(labels[i].numpy())])
        plt.axis("off")

AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = (train_ds.cache().prefetch(buffer_size=AUTOTUNE))
test_ds = (test_ds.cache().prefetch(buffer_size=AUTOTUNE))
valid_ds = (valid_ds.cache().prefetch(buffer_size=AUTOTUNE))

normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
valid_ds = valid_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))

base_model = tf.keras.applications.vgg19.VGG19(include_top=False,
                            weights='imagenet',
                            input_shape= input_shape,
                            classifier_activation='softmax')
base_model.trainable = False

def cnn_model():
    model = tf.keras.Sequential()

    #Pretrained model
    model.add(base_model)

    model.add(tf.keras.layers.Flatten())

    #Clasification head
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(16, activation='relu'))
    model.add(tf.keras.layers.Dense(6, activation='softmax'))

    return model

model1 =  cnn_model()

```

```
optimizer =tf.keras.optimizers.Adam(learning_rate=1e-4)

model1.compile(optimizer = optimizer,
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])

base_model.summary()
model1.summary()

early_stopping = callbacks.EarlyStopping(monitor='val_loss',
                                         min_delta=0.001,
                                         patience=50,
                                         verbose=1,
                                         mode='min',
                                         restore_best_weights=True)

history = model1.fit(train_ds,
                     validation_data = valid_ds,
                     epochs=600,
                     verbose = 1,
                     callbacks=early_stopping)

model_to_save = r"C:\Users\HP\Documents\GIT-BP\models\CNN1"
model1.save(filepath=model_to_save , save_format='h5')

def plot_accuracy_loss(history):

    fig = plt.figure(figsize=(16,10))

    plt.subplot(221)
    plt.plot(history.history['accuracy'],'b-', label = "Na trénovacích
    datech")
    plt.plot(history.history['val_accuracy'], 'r-', label = "Na
    validacních datech")
    plt.ylabel("Správnost",fontsize=15)
    plt.xlabel("Iterace",fontsize=15)
    plt.legend(fontsize=15)

    plt.subplot(222)
    plt.plot(history.history['loss'],'b-', label = "Na trénovacích
    datech")
    plt.plot(history.history['val_loss'], 'r-', label = "Na validacních
     datech")
    plt.ylabel("Ztráta",fontsize=15)
    plt.xlabel("Iterace",fontsize=15)
    plt.legend(fontsize=15)
    plt.show()

history_frame = pd.DataFrame(history.history)

print(history_frame.head())
plot_accuracy_loss(history)

test_loss, test_acc = model1.evaluate(test_ds)
```

```
155 print(str(test_acc*100) + '%')
156
157 image_predict = model1.predict(test_ds)
158 image_predict = np.argmax(image_predict, axis=1)
159
160 image_true = tf.concat([y for x, y in test_ds], axis=0)
161 image_true = np.argmax(image_true, axis=1)
162
163 cm = confusion_matrix(y_true=image_true, y_pred=image_predict)
164
165 def plot_confusion_matrix(cm, classes,cmap=plt.cm.Reds):
166     plt.imshow(cm, interpolation='nearest', cmap=cmap)
167     plt.colorbar()
168     tick_marks = np.arange(len(classes))
169     plt.xticks(tick_marks, classes, rotation=90,fontsize=12)
170     plt.yticks(tick_marks, classes,fontsize=12)
171
172     thresh = cm.max() / 2.
173     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
    [1])):
174         plt.text(j, i, cm[i, j],
175             horizontalalignment="center",
176             color="white" if cm[i, j] > thresh else "black")
177
178     plt.tight_layout()
179     plt.ylabel('Skutecnost',fontsize=15)
180     plt.xlabel('Predikce',fontsize=15)
181
182 plot_confusion_matrix(cm=cm, classes=class_names)
183 print(classification_report(image_true, image_predict, target_names=
    class_names))
```

A.5 Autoenkodér

```
import numpy as np
import matplotlib.pyplot as plt
import random
import glob
from PIL import Image
from tensorflow import keras
from tensorflow.keras import layers , callbacks
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
import tensorflow as tf
import os
from tqdm import tqdm
import cv2
from keras.preprocessing.image import img_to_array
from sklearn import metrics
import itertools
from sklearn.metrics import confusion_matrix , classification_report
from sklearn.neighbors import KernelDensity

train_dir = '../input/screw-anomaly-dataset/screw_dataset/AE_dataset/
    train'
test_dir = '../input/screw-anomaly-dataset/screw_dataset/AE_dataset/
    test'
anomaly_dir = '../input/screw-anomaly-dataset/screw_dataset/AE_dataset/
    anomaly'

batch_size = 16
image_size = (256,256)

normalize = ImageDataGenerator(rescale=1./255)

train_ds = normalize.flow_from_directory(
                train_dir,
                target_size = image_size,
                batch_size = batch_size,
                shuffle=True,
                seed=3,
                class_mode = 'input')

# Encoder
model = Sequential()
model.add(Conv2D(32,(3,3), activation='relu', padding='same',
    input_shape=(256,256,3)))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(128,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(256,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))
```

```
model.add(Conv2D(512,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Decoder
model.add(Conv2D(512,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(256,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(128,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(32,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))

model.add(Conv2D(3,(3,3), activation='sigmoid', padding='same'))

model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['mse'])

model.summary()

history = model.fit(train_ds,
                    batch_size=batch_size,
                    shuffle=True,
                    verbose=1,
                    epochs=300)

model.save("AE.h5")

train_pred = model.predict(train_ds, verbose=1)

train_pred_img = [train_pred[i,...] for i in range(1)]

plt.figure(figsize=(10,2))
plt.suptitle('Generované snímky z trénovacího souboru')
for i in range(1,4):
    ax = plt.subplot(1, 4, i)
    plt.imshow(train_pred[i])
plt.show()

encoder = Sequential()
encoder.add(Conv2D(32,(3,3), activation='relu', padding='same',
            input_shape=( 256, 256, 3),
            weights=model.layers[0].get_weights()) )
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(64,(3,3), activation='relu', padding='same',
            weights=model.layers[2].get_weights()))
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(128,(3,3), activation='relu', padding='same',
            weights=model.layers[4].get_weights()))
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(256,(3,3), activation='relu', padding='same',
```

```
            weights=model.layers[6].get_weights()))
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(512,(3,3), activation='relu', padding='same',
            weights=model.layers[8].get_weights()))
encoder.summary()

normilize = tf.keras.Sequential([layers.Rescaling(1./255)])

# Vzor oznacenych dat
good_data = []
good_files = glob.glob('../input/screw-anomaly-dataset/screw_dataset/
    AE_dataset/test/good/*')
for i in tqdm(good_files[0:19]):
    img = cv2.imread(i,1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(256,256))
    good_data.append(img)

good_data = np.reshape(good_data,
                       (len(good_data), 256, 256, 3))
good_data = normilize.predict(good_data)

anomaly_data = []
anomaly_files = glob.glob('../input/screw-anomaly-dataset/screw_dataset
    /AE_dataset/anomaly/*/*')
for i in tqdm(anomaly_files[0:45]):
    img = cv2.imread(i,1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(256,256))
    anomaly_data.append(img)

anomaly_data = np.reshape(anomaly_data,
                          (len(anomaly_data), 256, 256, 3))
anomaly_data = normilize.predict(anomaly_data)

# Testovaci data
test_good_data = []
test_good_files = glob.glob('../input/screw-anomaly-dataset/
    screw_dataset/AE_dataset/test/good/*')
for i in tqdm(test_good_files[20:42]):
    img = cv2.imread(i,1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(256,256))
    test_good_data.append(img)

test_good_data = np.reshape(test_good_data,
                            (len(test_good_data), 256, 256, 3))
test_good_data = normilize.predict(test_good_data)

test_anomaly_data = []
test_anomaly_files = glob.glob('../input/screw-anomaly-dataset/
    screw_dataset/AE_dataset/anomaly/*/*')
for i in tqdm(test_anomaly_files[46:120]):
    img = cv2.imread(i,1)
```

```
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(256,256))
    test_anomaly_data.append(img)

test_anomaly_data = np.reshape(test_anomaly_data,
                               (len(test_anomaly_data), 256, 256, 3))
test_anomaly_data = normilize.predict(test_anomaly_data)

encoded_images = encoder.predict_generator(train_ds)
encoder_output_shape = encoder.output_shape
out_vector_shape = encoder_output_shape[1]*encoder_output_shape[2]*
    encoder_output_shape[3]
encoded_images_vector = [np.reshape(img, (out_vector_shape)) for img in
     encoded_images]

ref_density = KernelDensity(kernel='gaussian',
                            bandwidth=0.2).fit(encoded_images_vector)

# Funkce porovnaní hustot
def density_and_recon_err(batch_images):
    density_list=[]
    recon_error_list=[]
    for im in tqdm(range(0, batch_images.shape[0]-1)):

        img  = batch_images[im]
        img = img[np.newaxis, :,:,:]
        encoded_img = encoder.predict([[img]],verbose=0)
        encoded_img = [np.reshape(img, (out_vector_shape)) for img in
    encoded_img]
        density = ref_density.score_samples(encoded_img)[0]
        reconstruction = model.predict([[img]],verbose=0)
        reconstruction_error = model.evaluate([reconstruction],[[img]],
     batch_size = 1,verbose=0)[0]
        density_list.append(density)
        recon_error_list.append(reconstruction_error)

    average_density = np.mean(np.array(density_list))
    stdev_density = np.std(np.array(density_list))
    average_recon_error = np.mean(np.array(recon_error_list))
    stdev_recon_error = np.std(np.array(recon_error_list))

    return density_list,average_density,stdev_density,recon_error_list,
    average_recon_error,stdev_recon_error

test_values = density_and_recon_err(good_data)
anomaly_values = density_and_recon_err(anomaly_data)

print("Test Avg.Density = " + str(test_values[1]) + "(" + str(
    test_values[2]) +")" )
print("Test Rec.Error = " + str(test_values[4]) + "(" + str(test_values
    [5]) +")" )
print("Anomaly Avg.Density = " + str(anomaly_values[1]) + "(" + str(
    anomaly_values[2]) +")" )
print("Anomaly Rec.Error = " + str(anomaly_values[4]) + "(" + str(
```

```
    anomaly_values[5]) +")" )

fig, ax = plt.subplots(figsize=(8, 15))
ax.set_aspect(5000)

x1 = test_values[0]
y1 = test_values[3]
x2 = anomaly_values[0]
y2 = anomaly_values[3]

ax.scatter(x1, y1, color="g",label='Snímky bez vady')
ax.scatter(x2, y2, color="r", label= 'Snímky s vadou')

ax.set_xlim(90482.5,90500)
ax.set_ylim(0.0008, 0.0028)
plt.xlabel("Odhad hustoty",fontsize=12)
plt.ylabel("Rekonstrkcní chyba",fontsize=12)
plt.legend()

plt.show()

# Funkce hodnoceni snimku z testovaciho osuboru dat
def predictions(batch_images,type):
    predict = []
    truth = []
    density_score = 90496
    recon_err_score = 0.00148 #0.00148

    for im in tqdm(range(0, batch_images.shape[0])):
        img  = batch_images[im]
        img = img[np.newaxis, :,:,:]

        encoded_img = encoder.predict([[img]],verbose=0)
        encoded_img = [np.reshape(img, (out_vector_shape)) for img in
    encoded_img]
        density = KDE.score_samples(encoded_img)[0]

        recon = model.predict([[img]],verbose=0)
        recon_err = model.evaluate([recon],[[img]], batch_size = 1,
    verbose=0)[0]

        if density < density_score and recon_err < recon_err_score:
            predict.append(['good'])
            truth.append(type)
        else:
            predict.append(['anomaly'])
            truth.append(type)
    prediction=[truth,predict]
    return prediction

anomaly_ev = predictions(test_anomaly_data, ['anomaly'])
test_ev = predictions(test_good_data, ['good'])
total_ev = [anomaly_ev[0] + test_ev[0], anomaly_ev[1] + test_ev[1]]

```

```
249 image_true = total_ev[0]
250 image_predict = total_ev[1]
251 cm = confusion_matrix(y_true=image_true, y_pred=image_predict)
252
253 def plot_confusion_matrix(cm,
254                           classes,
255                           cmap=plt.cm.Reds):
256
257     plt.imshow(cm, interpolation='nearest', cmap=cmap)
258     plt.colorbar()
259     tick_marks = np.arange(len(classes))
260     plt.xticks(tick_marks, classes, rotation=90)
261     plt.yticks(tick_marks, classes)
262
263     thresh = cm.max() / 2.
264     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
    [1])):
265         plt.text(j, i, cm[i, j],
266             horizontalalignment="center",
267             color="white" if cm[i, j] > thresh else "black")
268
269     plt.tight_layout()
270     plt.ylabel('Skutecnost',fontsize=15)
271     plt.xlabel('Predikce',fontsize=15)
272
273 plot_confusion_matrix(cm=cm,classes = ['S vadou','Bez vady'])
274 print(classification_report(image_true, image_predict, target_names=['S
     vadou','Bez vady']))
```