VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2022 Nikola Sarančuk

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE AUTOENKODÉRU PRO ZPRACOVÁNÍ OBRAZOVÝCH DAT

AUTOENCODER IMPLEMENTATION FOR IMAGE ANALYSIS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Nikola Sarančuk

AUTHOR

VEDOUCÍ PRÁCE — Ing. Karel Horák, Ph.D.

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Nikola Sarančuk ***ID:*** 221134

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Implementace autoenkodéru pro zpracování obrazových dat

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je implementace algoritmu detekce anomálií v průmyslové inspekci do stávající SW aplikace. Úkolem je implementovat vybraný algoritmus na bázi NN (konkrétně typu autoenkodér) a testovat jej na demonstrativní úloze.

1. Seznamte se s pojmy unární, binární a multi-class klasifikace.
2. Dle pokynů vedoucího sestavte anotovaný dataset vybraného výrobku čítající 1000+ realizací v průmyslovém standardu.
3. Implementujte vybraný algoritmus NN typu autoenkodér v jazyce Python nebo C dle domluvy s vedoucím a proveďte ověření na sestaveném datasetu.
4. Výsledky ověření vyhodnoťte dle metodiky matice záměn a odvozených parametrů.

DOPORUČENÁ LITERATURA:

1. One-class classification: taxonomy of study and review of techniques. https://doi.org/10.1017/S026988891300043X.
2. One-class classification - Concept-learning in the absence of counter-examples. http://homepage.tudelft.nl/n9d04/thesis.pdf.
3. Anomaly Detection with Autoencoders Made Easy: https://towardsdatascience.com/anomaly-detection-with-autoencoder.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 23.5.2022

Vedoucí práce: Ing. Karel Horák, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Práce se věnuje rešerši problému detekce anomálií v průmyslové inspekci. V práci je popsána umělá neuronová síť a její jednotlivé části. Práce obsahuje kapitolu, kde jsou srovnány unární, binární a multi-class klasifikátory. Dále je v práci vysvětlena architektura konvolučních neuronových sítí a architektura sítí typu autoenkodér. Poté je v práci popsán vytvořený anotovaný dataset. Nakonec je v práci popsána implementace konvolučního autoenkodéru a zhodnocena kvalita klasifikace.

Klíčová slova

klasifikace, unární klasifikátor, binární klasifikátor, multi-class klasifikátor, konvoluční neuronová síť, autoenkodér.

Abstract

The paper is devoted to the research of the problem of anomaly detection in industrial inspection. The paper describes the artificial neural network and its parts. The thesis contains a chapter where unary, binary and multi-class classifiers are compared. The thesis then explaines architecture of convolutional neural networks and autoencoder neural networks.. Then the paper describes the annotated dataset created. Finally, the paper describes the implementation of the convolutional autoencoder and evaluates the classification quality.

Keywords

classification, unary classificator, binary classificator, multi-class classificator, convolutional neural network, autoencoder

Bibliografická citace

SARANČUK, Nikola. Implementace autoenkodéru pro zpracování obrazových dat. Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/142236. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Ing. Karel Horák, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Nikola Sarančuk
VUT ID studenta:	221134
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Implementace autoenkodéru pro zpracování obrazových dat

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Tato práce se věnuje rešerši problému detekce anomálií v průmyslové inspekci. Problém detekce anomálie je, že přesně nevíme, jak tato anomálie vypadá. Nemáme tudíž k dispozici příklady chyb či defektů, které bychom mohli přímo detekovat. Naopak ale máme k dispozici spoustu objektů, které žádné chyby nemají. Abychom pochopili tento problém, seznámíme se s pojmy unární, binární a multi-class klasifikace.

Pro vytvoření vhodného klasifikátoru pro detekci anomálií se zaměřujeme na umělou neuronovou síť typu autoenkodér, která je díky své architektuře schopna anomálie detekovat. Abychom mohli pochopit fungování této sítě, je důležité si nejdříve obecně definovat co to jsou neuronové sítě a z čeho skládají. Také si definujeme pojmy používané pro popis neuronových sítí a způsoby vyhodnocení kvality klasifikace.

První kapitola se zabývá klasifikátory a jejich typy. Také jsou zde definovány způsoby posouzení kvality klasifikátoru.

Druhá kapitola se věnuje popisu neuronových sítí, jejich částí a definicí pojmů používaných pro popis těchto sítí.

Třetí kapitola vysvětluje princip fungování konvoluční neuronové sítě a popisuje jednotlivé vrstvy, které se v těchto sítích vyskytují.

Ve čtvrté kapitole je popsána umělá neuronová síť typu autoenkodér a její použití pro detekci anomálií.

V páté kapitole je popsán vytvořený anotovaný dataset. V následující šesté kapitole je popsána implementace konvolučního autoenkodéru.

V poslední kapitole je zhodnocen implementovaný klasifikátor podle metody matice záměn a z ní vycházejících metrik.

1. KLASIFIKACE

Klasifikace je řazení nových vzorků do vhodné výstupní klasifikační množiny, do které tyto vzorky s největší pravděpodobností patří. Podle počtu těchto výstupních množin můžeme klasifikaci rozdělit na tři skupiny a to unární, binární a multi-class klasifikaci. Vzorky jsou klasifikovány na základě klasifikačního modelu neboli klasifikační funkce [1]. Obecně lze klasifikaci popsat jako dělení klasifikovaného prostoru pomocí klasifikační funkce.

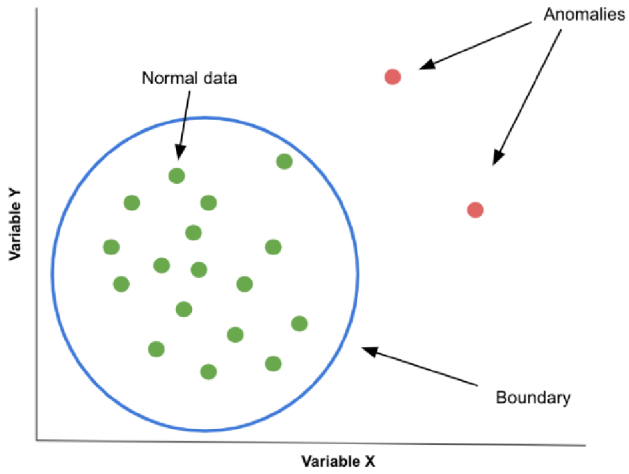
Klasifikátor je funkce, která dle vstupního feature vektoru (sloupcový vektor popisující klasifikované objekty) a vektoru vah vrací zařazení do určité skupiny. Cílem strojového učení je na základě trénovacích dat vytvořit klasifikátor, který je schopný co nejlépe objekty z trénovací sady roztřídit [2].

1.1 Unární klasifikátor

Unární klasifikátor se snaží určit hranici, v jejíž prostoru se nachází co nejvíce objektů patřících do cílové třídy, ale zároveň musí být minimalizována šance, že budou do tohoto prostoru přijmuty i objekty které do třídy nepatří. Jedná se o metodu učení bez učitele, jelikož trénovací data nejsou nějak označena.

Objekty, které nezapadají do modelu tvořeného klasifikátorem se označují jako anomálie nebo odlehlé objekty. Na rozdíl od binární a multi-class klasifikace, které předpokládají že máme k dispozici tréninková data rozdělená víceméně rovnoměrně do všech tříd, máme v unární klasifikaci pro učení k dispozici převážně příklady z cílové třídy. Naopak anomálie nezapadající do této třídy se v trénovací množině vyskytují menšinově nebo dokonce vůbec. To je dáno tím že data z cílové třídy lze získat snadno, ale data z negativní třídy můžou být neznámá nebo těžko získatelná. Ostatní klasifikátory by z důvodu nevyrovnanosti nebo úplné absence dat z určitých tříd nefungovali příliš dobře.

To znamená že v těchto klasifikačních úlohách jsme schopni určit pouze jednu stranu klasifikační hranice. Kvůli tomu je obtížné rozhodnout o tom, jak těsně by tato hranice měla obkličovat data z cílové třídy, tak aby nedocházelo k přijetí odlehlých objektů. Rovněž je složité vybrat správné rysy, které spolehlivě definují objekty z cílové třídy [2].



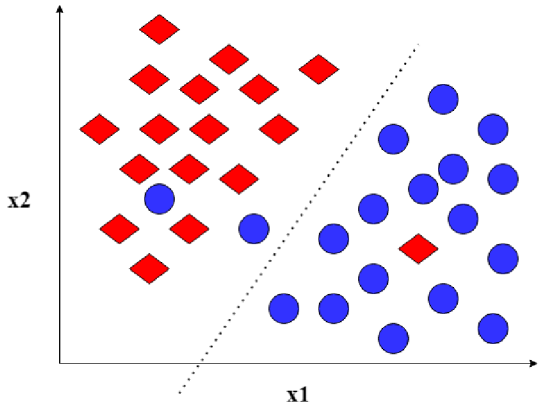
Obrázek 1.1 Ukázka unárního klasifikátoru určujícího cílové objekty a anomálie [3]

1.2 Binární klasifikátor

V binární klasifikaci máme k dispozici popsaná data ze dvou tříd. Model je tedy možné definovat z dvou stran. Tyto modely jsou trénovány na datech, obsahující objekty označené binární hodnotou 1 nebo 0, které představují dvě třídy. Zde už se jedná o učení s učitelem.

Celkově může u binární klasifikace dojít ke čtyřem kombinacím skutečné třídy do které objekt patří, a třídy do které tento objekt klasifikátor zařadil. Kromě skutečných pozitivů (správně klasifikované objekty z pozitivní třídy) a skutečných negativů (správně klasifikované objekty z negativní třídy) se mohou při klasifikaci vyskytnout i falešné pozitivy a falešné negativy. Jedná se o objekty, které byli špatně zařazeny buď do pozitivní nebo negativní třídy.

Pokud tyto kategorie poskládáme do matice tak, že ve sloupcích budou skutečné hodnoty a na řádcích hodnoty klasifikátoru, dostaneme tzv. matici záměn. Na základě této matice poté můžeme určit poměry, které se používají k vyhodnocení kvality klasifikace [4].

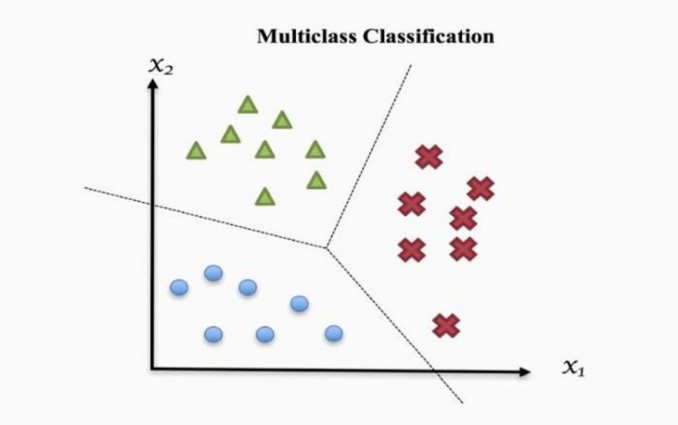


Obrázek 1.2 Ukázka binární klasifikace [5]

Na obrázku 1.2 můžeme vidět ukázku binární klasifikace. Klasifikační funkce znázorněna tečkovanou čarou rozděluje klasifikační prostor na dvě části. Osy x a y reprezentují jednotlivé řádky feature vektoru. Pokud bychom červeným kosočtvercům přiřadili binární hodnotu 1 a modrým kruhům hodnotu 0, je patrné, že kruhy v levé části představují falešné pozitivy a kosočtverec v pravé části naopak falešný negativ.

1.3 Multi-Class klasifikátor

Posledním klasifikátorem je multi-class klasifikátor. Oproti binárnímu klasifikátoru zde není omezen počet tříd. To znamená že objekty jsou zařazovány do N-tříd [4].



Obrázek 1.3 Ukázka multi-class klasifikace – klasifikační prostor rozdělený klasifikátorem [6]

1.4 Metody vyhodnocení kvality klasifikace

Pro vyhodnocení kvality klasifikace jsou často využívány poměry vycházející z matice záměn. Matici záměn vytvoříme tak, že poskládáme do sloupců matice skutečné hodnoty a do řádků hodnoty dané klasifikátorem. Na obrázku č.1.4 je zobrazena matice záměn pro binární klasifikátor [7].

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

Obrázek 1.4 Metoda matice záměn [8]

Matice se skládá z:

- Skutečných pozitivů (True Positive TP) – objekty, které klasifikátor správně klasifikoval jako objekty patřící do pozitivní třídy
- Skutečných negativů (True Negative TN) – objekty, které klasifikátor správně klasifikoval jako objekty patřící do negativní třídy
- Falešných pozitivů (False Positive FP) – objekty, které klasifikátor špatně klasifikoval jako objekty patřící do pozitivní třídy
- Falešných negativů (False Negative FN) – objekty, které klasifikátor špatně klasifikoval jako objekty patřící do negativní třídy

Pro určení kvality a porovnání klasifikace poté z těchto hodnot můžeme určit:

- Přesnost (Accuracy) – Udává, jak často je klasifikátor schopen správně klasifikovat objekt. Ideálně by se tato hodnota měla rovnat jedné, samozřejmě prakticky tato hodnota bude nižší.

$$Přesnost = \frac{TP + TN}{TP + TN + FP + FN}$$

- Chybovost (Error Rate) – Udává, jak často klasifikátor špatně klasifikuje objekt. Tato míra se dá také určit jako 1 – Přesnost.

$$Chybovost = \frac{FP + FN}{TP + TN + FP + FN}$$

- Senzitivita (Recall) – Udává, jak často klasifikátor správně určí objekt, který patří do pozitivní třídy.

$$Senzitivita = \frac{TP}{TP + FN}$$

- Specificita (Specificity) – Udává, jak často klasifikátor správně určí objekt, který patří do negativní třídy.

$$Specificita = \frac{TN}{TN + FP}$$

- Preciznost (Precision) – Udává, jak často je klasifikace správná, pokud klasifikátor klasifikuje že objekt patří do pozitivní třídy.

$$Preciznost = \frac{TP}{TP + FP}$$

- F-míra (F-Measure) – Jedná se o vážený průměr senzitivity a preciznosti.

$$F = \frac{Preciznost \cdot Senzitiva}{Preciznost + Senzitivita}$$

[7]

2. NEURONOVÁ SÍŤ

Úkolem neuronových sítí je obecně aproximace nějaké funkce *f*. V této kapitole se seznámíme s částmi, ze kterých se neuronové sítě skládají.

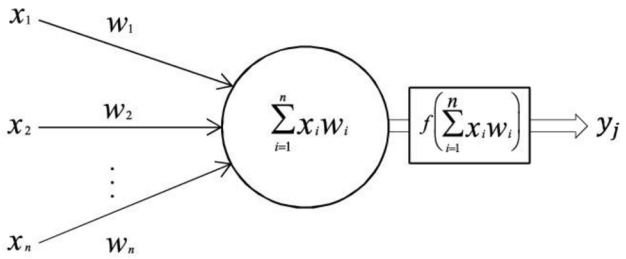
Původně byli umělé neuronové sítě inspirovány mozkem a nervovou soustavou živých organismů [9]. Například perceptronová síť vznikla jakožto model neuronů na sítnici lidského oka, pomocí kterých jsme schopni rozeznávat vzory [10]. V dnešní době už se ale neuronovými sítěmi nesnažíme kopírovat fungování mozku. Jedná se už spíše o složité matematické modely, jejichž cílem je co nejlépe aproximovat určitou matematickou funkci.

Umělou neuronovou síť si lze představit jako acyklický orientovaný graf, popisující, jakým způsobem kombinujeme různé funkce. Uzly tohoto grafu nazýváme neurony. Neurony, které stojí spolu na stejné úrovni dohromady tvoří vrstvu neuronové sítě. Počet těchto vrstev určuje hloubku sítě. Pokud se neuronová síť skládá z více než tří vrstev (mezi tyto vrstvy nepočítáme vstupní a výstupní vrstvu), můžeme ji nazvat hlubokou neuronovou sítí [9].

2.1 Neuron

Nejelementárnějším stavebním blokem, ze kterého se skládají veškeré neuronové sítě, je neuron.

Každý neuron může mít více vstupů, ale má vždy pouze jeden výstup. Stejně jako u jeho biologického protějšku, je neuron k ostatním neuronům v síti připojen přes synaptické váhy. Tyto váhy jsou v době učení neuronové sítě upravovány. Každý vstup má k sobě přiřazenou právě jednu váhu **w**. Vstupem může být také práh neuronu **b**. Jedná se v podstatě také o váhu, která však není přiřazena ke vstupu **x** a proto se k váženým vstupům pouze přičítá. Neuron poté spočítá součet vážených vstupů a prahu. Tím získá tzv. skrytý potenciál **z** a pomocí aktivační funkce **f(z)** vypočítá výstupní hodnotu **y**. Konečná rovnice pro výstup neuronu je tedy $\mathbf{y = f(x_i \cdot w_i + b)}$. Vstupem do neuronu mohou být přímo vstupy které přivádíme do sítě, práh neuronu nebo výstupy jiných neuronů, záleží, v které vrstvě se neuron nachází [9].



Obrázek 2.1 Model umělého neuronu

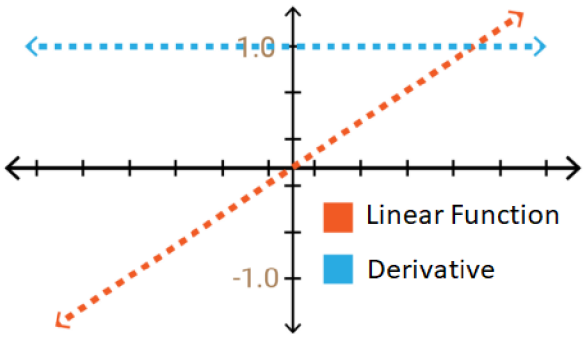
2.1.1 Lineární aktivační funkce

Aktivační funkce můžeme rozdělit na funkce lineární a nelineární.

Lineární aktivační funkce má ovšem dvě velké nevýhody. Pokud v neuronech použijeme lineární funkce, nezáleží pak na tom, kolik má síť vrstev, ale vždy se bude chovat jako by měla

vrstvu pouze jednu. Síť nebude schopna vytvořit klasifikátor pro složitější vstupní data, jelikož se bude jednat o lineární regresní model. To je dáno tím, že lineární kombinací dvou lineárních funkcí dostaneme opět jenom lineární funkci.

Druhou nevýhodou je derivace této funkce. Derivací lineární funkce totiž dostaneme konstantu, která není vůbec vázána na náš vstup do funkce. To znamená že při učení dané sítě nejsme schopni využít zpětného šíření chyby. Algoritmus zpětného šíření chyby bude popsán dále v práci [11].



Obrázek 2.2 Lineární funkce a její derivace [11]

2.1.2 Nelineární aktivační funkce

Kvůli těmto nevýhodám se v dnešních moderních neuronových sítích v drtivé většině využívají funkce nelineární.

Nelinearita funkce nám pomůže se všemi nedostatky lineárních funkcí. Kombinací nelineárních funkcí můžeme dosáhnout aproximace libovolně složité funkce. Díky tomu jsme schopni vytvořit přesné mapování pro komplexní vstupní data. Nelineární funkce lze také derivovat bez toho abychom při derivaci přišli o vztah ke vstupu **z**. To znamená že můžeme využít zpětného šíření chyby a zefektivnit tak proces učení.

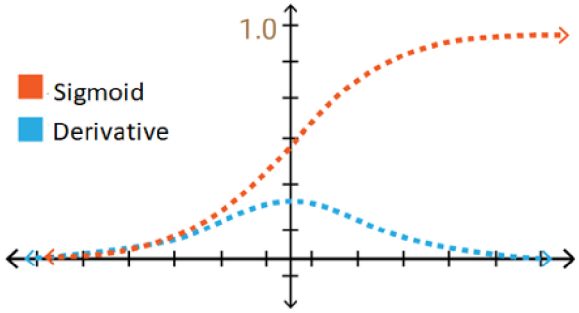
Důležitou vlastností aktivační funkce je její výpočetní náročnost, protože počítač s touto funkcí bude muset počítat pro každý neuron v síti.

Funkce používané pro aktivaci neuronů v moderních sítích jsou například sigmoida, hyperbolický tangens, ReLU nebo Leaky-ReLU [11].

Sigmoida

Výhodou využití této funkce jsou normalizované hodnoty v rozmezí 0 a 1 na výstupu. Tyto hodnoty můžeme považovat za pravděpodobnosti.

Naopak nevýhodou je zmenšující se spád této funkce u jejích koncových hodnot. To znamená, že výstupní hodnota y se téměř nemění v reakci na změny proměnné x. Derivace se proto v těchto hodnotách blíží nule, což se projeví jako pomalé učení sítě. Tento jev se nazývá mizející gradient. Funkce také není vystředěná okolo nuly [11].

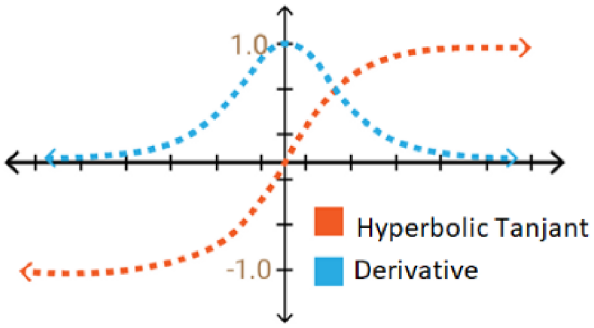


Obrázek 2.3 Sigmoida a její derivace [11]

Hyperbolický tangens

Tato funkce je velice podobná funkci sigmoid, avšak její obor hodnot je od -1 do 1.

To má za důsledek vetší strmost její derivace, což umožňuje rychlejší učení neuronových sítí díky většímu rozsahu hodnot. Nevýhoda mizejícího gradientu u této funkce přetrvává [11].

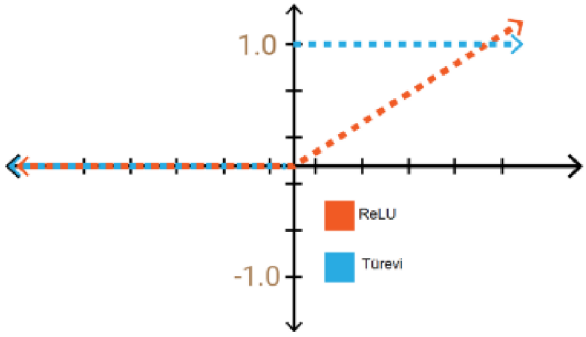


Obrázek 2.4 Hyperbolický tangens a jeho derivace [11]

ReLU (Rectified Linear Unit)

U této aktivační funkce má část na kladné ose stejné vlastnosti jako funkce lineární, ale hodnoty na záporné ose jsou mapovány nulou. Její obor hodnot je tedy od 0 do nekonečna.

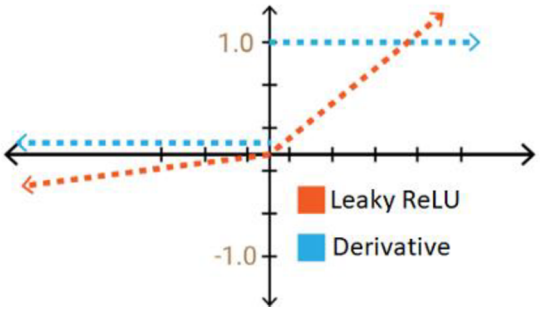
To způsobuje tzv. umírání neuronů. Kvůli tomu se sice snižuje schopnost modelu se správně naučit ze vstupních dat, ale výrazně se zlepšuje výpočetní náročnost. Proto je tato aktivační funkce v současné době jedna z nejpoužívanějších u neuronových sítích s mnoha vrstvami [11].



Obrázek 2.5 ReLU a její derivace [11]

Leaky ReLU

Jedná se o funkci, která je snaží vyřešit nedostatek umírajících neuronů v záporné části funkce ReLU. Toho je dosaženo tím, že záporné hodnoty budeme násobit hodnotou 0,01. Díky tomu se zvýší obor hodnot této funkce na mínus nekonečno až plus nekonečno [11].



Obrázek 2.6 Leaky-ReLU a její derivace [11]

2.2 Učení neuronové sítě

Učení neuronové sítě je nastavování vah a prahu neuronu. Tyto váhy slouží k určení důležitosti dané vstupní proměnné. Vyšší váha tedy přispívá k výstupu více. Na začátku jsou váhy zvoleny náhodně. Učením sítě upravujeme váhy tak abychom minimalizovali chybu sítě. V současné době je nejpopulárnějším algoritmem pro učení neuronových sítí algoritmus zpětného šíření chyby. Podle toho, zda máme k dispozici k tréninkovým datům i požadované výstupy, můžeme učení rozdělit na učení s učitelem a učení bez učitele [12].

2.2.1 Učení s učitelem

Učení s učitelem (ang. Supervised learning) je způsob učení u kterého máme v trénovací sadě k dispozici označená data. Označení těchto dat slouží modelu k tomu, aby mohl měřit chybu předpověděných výsledků a na základě této chyby se učit [12].

2.2.2 Učení bez učitele

Učení bez učitele (ang. Unsupervised learning) nepotřebuje ke svému učení označená data. To znamená že jsou schopny hledat skryté vzory v datech, bez nutnosti lidského zásahu při prvotním označení dat. Tato vlastnost je velice výhodná pro velké soubory dat, kde by jejich označování vyžadovalo obrovské lidské zdroje [12].

2.2.3 Algoritmus zpětného šíření chyby

Je to iterativní gradientní algoritmus pro minimalizaci chyby sítě. Chyba sítě je definována jako rozdíl mezi výstupem sítě a požadovanými výstupními hodnotami. Kvůli tomu, že pro učení potřebujeme požadované hodnoty, považujeme algoritmus zpětného šíření chyby za učení s učitelem.

Algoritmus funguje následujícím způsobem. Nejdříve síť provede výpočet výstupu pro daný tréninkový vzor. Podle toho se určí chyba sítě vzhledem k aktuálnímu tréninkovému vzoru. Pokud budeme upravovat váhy po přeložení jednotlivých vzorů, jedná se o tvz. online učení. U offline učení nejdříve projdeme celou tréninkovou množinu a až poté upravujeme váhy. Postupně tedy určujeme chybu pro všechny vzory v tréninkové množině. Součtem chyb u jednotlivých vzorů získáme celkovou chybu.

Následně dochází na adaptaci vah sítě. Jelikož se jedná o gradientní algoritmus, jsou ve výpočtu využívány parciální derivace. Proto je tedy velice důležité, aby derivace aktivační funkce neuronu nebyla konstanta. Celý tento proces opakujeme tak dlouho, dokud není celková chyba neuronové sítě menší než námi požadovaná chyba nebo dokud nedojde k překročení maximálního počtu iterací [9].

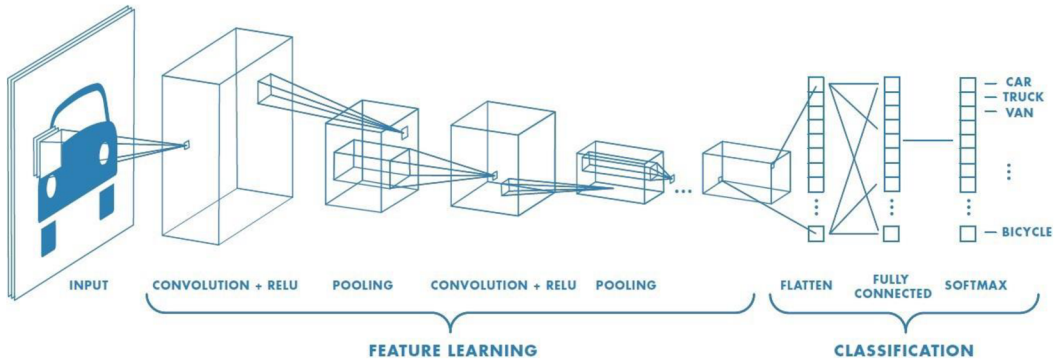
3. KONVOLUČNÍ NEURONOVÉ SÍTĚ

Jedná se o třídu umělých neuronových sítí, které aspoň v jedné ze svých vrstev využívají konvoluci. Jelikož obrázky jsou reprezentovány jako velké matice s několika vrstvami, je potřeba je převést do formy, která je méně výpočetně náročná ale nedojde při ní ke ztrátě důležitých informací, které potřebujeme pro správnou klasifikaci. K získaní charakteristických rysů z obrazu slouží konvoluční vrstvy.

Obecně tyto rysy můžeme získat použitím velkého počtu malých filtrů, takzvaných kernelů. V začátečních konvolučních vrstvách můžeme ze vstupního obrazu extrahovat jednoduché rysy, například hrany. Postupným přidáváním těchto vrstev je neuronová síť schopna z obrazu rozpoznávat i složitější struktury, což je důležité právě pro klasifikaci z obrazu.

Ke snížení velikosti výstupu konvolučních vrstev se používají pooling neboli sdružovací vrstvy. Dále může obsahovat slučovací vrstvu. Tato vrstva slučuje různé větvě v síti. Konvoluční sítě jsou často větvené, každá větev poté slouží k detekci různých rysů v obraze. Finální čtvercová mapa je pak zarovnána do sloupcového vektoru.

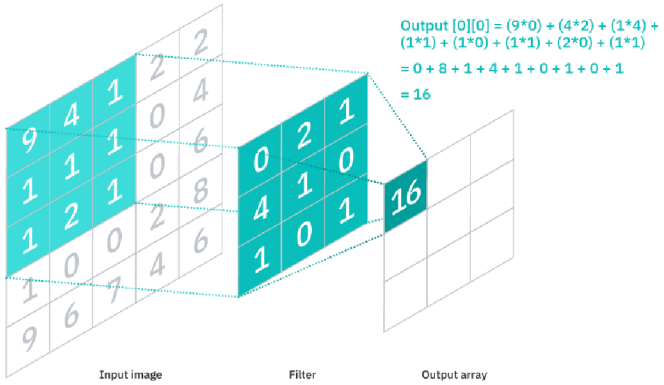
Dále konvoluční neuronová síť obsahuje plně propojenou vrstvu a softmax vrstvu. Plně propojená vrstva slouží k finální klasifikaci. Poslední vrstva softmax má za úkol výstupní hodnoty převést na normalizované hodnoty v rozmezí 0-1, které lze interpretovat jako pravděpodobnosti [13].



Obrázek 3.1 Návaznost vrstev v konvoluční neuronové síti [13]

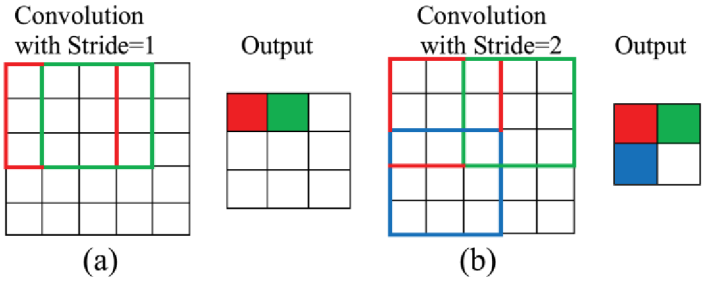
3.1 Konvoluční vrstva

V konvoluční vrstvě je na vstupní obrázek aplikován kernel. Kernel je filtr, čtvercová matice. Kernel se přesouvá přes vstupní obraz a postupně vykonává maticové násobení. Tato hodnota se poté zapisuje do buněk výstupní mapy [13].



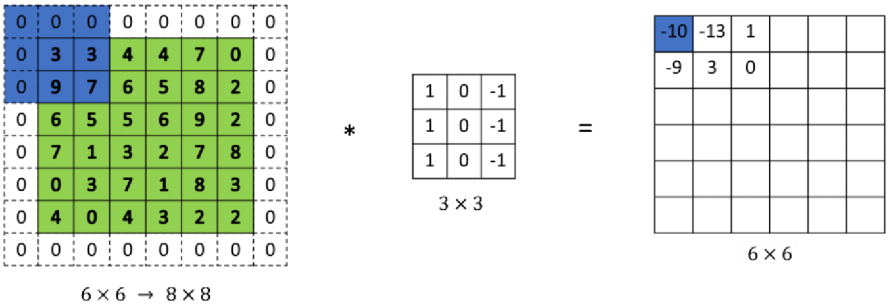
Obrázek 3.2 Kernel a jeho aplikace na vstupní obrázek [14]

Kernel se posouvá o hodnotu kroku, která se zde nazývá stride. Pokud je obrázek tvořen více rozměry (např. RGB) kernel má vždy stejnou hloubku. To znamená že kernel aplikujeme na každou vrstvu vstupního obrázku, a jednotlivé hodnoty z těchto vrstev sčítáme. K těmto hodnotám navíc přičítáme bias hodnotu. Výsledkem této operace je pak už pouze jednovrstvá aktivační mapa. Velikostí kernelu a parametru stride dokážeme ovlivňovat velikost výstupní mapy. Výsledná mapa bude mít velikost (N - F) / stride + 1, kde N je rozměr vstupní mapy a F je rozměr kernelu [13].



Obrázek 3.3 Znázornění velikosti kroku a velikosti výstupní výstupní mapy [15]

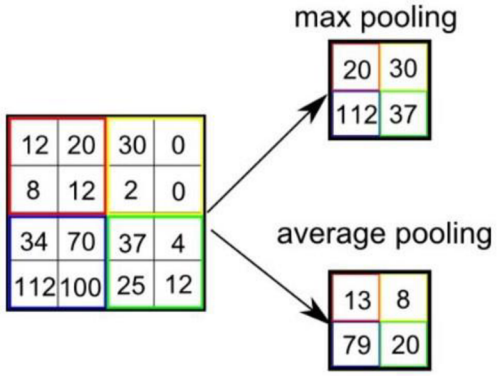
Pokud nechceme, aby se výsledná mapa zmenšila, můžeme vstupní mapu rozšířit o výplň. Výplň je tvořena nulami na kraji vstupní mapy. Můžeme také použít několik různých filtrů na vstupní mapu. Výstupní mapa poté může mít tolik vrstev, kolik filtrů jsme paralelně použili [13].



Obrázek 3.4 Výplň vstupní mapy [16]

3.2 Sdružovací vrstva

V této vrstvě dochází k redukci rozměru mapy použitím filtru, podobně jako u konvoluční vrstvy. Na rozdíl od kernelu ale tento filtr nemá žádné váhy. Pooling vrstvy existují typu average nebo max. Average pooling počítá průměrnou hodnotu z oblasti překryté filtrem. Max pooling vybere maximální hodnotu z oblasti. V této vrstvě sice dochází ke ztrátě informací, zároveň se ale snižuje komplexita a riziko přeučení a zlepšuje efektivita NN [13].



Obrázek 3.5 Max pooling a average pooling vrstva [13]

3.3 Up Sampling vrstva

Up Sampling vrstva je opakem vrstvy sdružovací. Dochází zde ke zvětšování rozměru mapy. Možností výpočtu hodnot pro nové pixely je více, v této práci je využívána up sampling vrstva, která pouze zkopíruje hodnotu na nové pixely ve svém okolí. Toto okolí je definováno velikostí matice. Například matice o velikosti 2 x 2 zdvojnásobí rozměr mapy. [17]

3.4 Plně propojená vrstva

Další vrstvou v konvolučních sítích je plně propojená vrstva. Tato vrstva slouží k finální klasifikaci.

Klasifikace je prováděna získáním nelineární kombinace vlastností objektů, které jsme získali jako výstup z konvoluční vrstvy. Každý neuron v této vrstvě je spojen se všemi neurony ve vrstvě předchozí [13].

3.5 Softmax vrstva

Nakonec se v konvoluční sítí používá vrstva neuronů s aktivační funkcí softmax. Tato vrstva je schopná normalizovat výstupní hodnoty z plně propojené vrstvy do rozmezí 0 až 1 a tím umožňuje určit pravděpodobnost s jakou patří vstupní objekt do určité třídy [13].

4. AUTOENKODÉRY

Autoenkodér je zvláštní typ umělé neuronové sítě, která je trénována na to, aby co nejlépe rekonstruovala vstupní data na svém 00výstupu. Aby nedocházelo k pouhému kopírování dat ze vstupu na výstup, obsahuje autoenkodér zúžení, kde dochází ke zmenšení rozměru. Autoenkodéry se používají pro zmenšení rozměru dat, pro redukci šumu v datech, pro extrakci rysů přítomných v datech, pro doporučovací systémy a pro detekci anomálií. Vzhledem k tomu, že nepotřebuje popsaná data řadí se autoenkodér do kategorie učení bez učitele. Někdy se také uvádí že se jedná o tzv. semi-supervised učení, jelikož sice nemáme k dispozici popsaná data, ale jako učitele využíváme data vstupní.

Autoenkodér se skládá ze dvou částí, a to enkodéru a dekodéru. Jak můžeme vidět na obr. č.4.1, enkodér postupně zmenšuje rozměr vstupních dat. Místo s nejmenším rozměrem dat nazýváme kód. Data v tomto zkomprimovaném stavu můžeme reprezentovat jako bod v latentním prostoru. Dekodér zase postupně zvětšuje tento rozměr zpět na původní. Dekodér má stejný počet skrytých vrstev a neuronů jako enkodér. Zmenšováním rozměrů v kódovacím procesu dochází ke ztrátě informací. Naopak dekodér se poté redukovaná data snaží na výstupu zrekonstruovat tak, aby se co nejvíce podobala datům na vstupu. Porovnáním rekonstruovaného výstupu s původními vstupními daty získáme rekonstrukční chybu.

Učení je prováděno minimalizací této rekonstrukční chyby. Neuronová síť je tak nucena k tomu, aby ve svém kódu měla co nejlepší komprimovanou reprezentaci vstupních dat a dokázala tak ignorovat nepodstatná data, např. šum. Tato redukovaná reprezentace dat je poté validována tím, že se redukovaná data znovu dekódují do původního obrazu. Idealním výstupem autoenkodéru jsou tedy data naprosto shodná se vstupními daty. Proto je možné při učení autoenkodéru využít algoritmus zpětného šíření chyby, jelikož můžeme požadovaný výstup nastavit stejný jako vstup.

Pro kód poté platí

$$z = g_\phi(x), \tag{4.1}$$

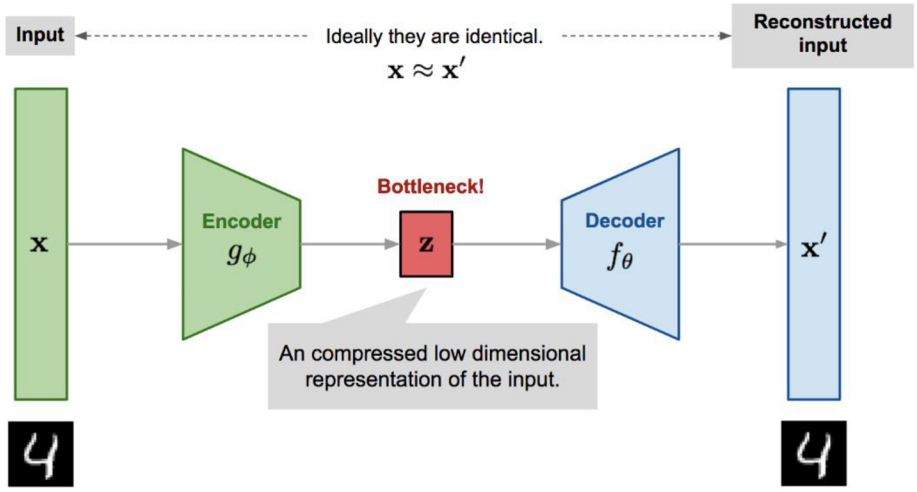
kde z je redukovaná reprezentace vstupních dat, x je vstup do neuronové sítě, g je funkce enkodéru a Φ jsou učené parametry (váhy). Rekonstruovaný výstup sítě lze poté zapsat jako

$$x' = f_\theta\left(g_\phi(x)\right), \tag{4.2}$$

kde f je funkce dekodéru a θ opět parametry. Rekonstrukční chybu L určíme pomocí střední kvadratické chyby (MSE – Mean Squared Error) jako

$$L_{AE}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}\left[x^{(i)} - f_\theta\left(g_\phi\left(x^{(i)}\right)\right)\right]^2, \tag{4.3}$$

kde n je počet vzorů [18].



Obrázek 4.1 Neuronová síť typu autoenkodér [18]

4.1 Řídký autoenkodér

U řídkého autoenkodéru je zúžení ve středu nahrazeno tzv. řídkostí. Řídkost je parametr, který určuje podmínku pro průměrnou aktivaci neuronu ve skryté vrstvě. Toto omezení zajistí, že i autoenkodér s velkým počtem neuronů ve skryté vrstvě nebude data ze vstupu pouze kopírovat, ale naučí se důležité korelace mezi rysy v datech. Průměrnou aktivaci neuronu určíme vztahem:

$$\hat{\rho}_j = \frac{1}{n}\sum_{\mathrm{i}=1}^{n}\left[a_j^{(l)}(x^{(i)})\right], \tag{4.4}$$

kde m je počet vzorů v trénovací množině a $a_j^{(l)}\left(x^{(i)}\right)$ je aktivace j-tého neuronu ve skryté vrstvě l pro vzorek $x^{(i)}$ z trénovací množiny. Parametr řídkosti poté nastavíme na hodnotu blízkou hodnotě, kdy je neuron neaktivní. Tato hodnota záleží na aktivační funkci neuronu, budeme uvažovat aktivační funkci sigmoida, tedy hodnota, kdy je neuron neaktivní je 0. Parametr řídkosti ρ můžeme tedy nastavit např. na 0,005. Aby síť byla schopná této podmínce vyhovět, všechny aktivační hodnoty neuronů ve skryté vrstvě budou muset být blízké nule. Dosáhneme toho přidáním penalizační funkce podle vztahu

$$\sum_{\mathrm{j}=1}^{s_l} \rho \log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}, \tag{4.5}$$

s_2 je zde počet neuronů ve skryté vrstvě. Penalizační funkce bude rovna nule, pokud $\hat{\rho}_j = \rho$. Ztrátová funkce poté bude

$$L_{SAE}(\theta) = \mathrm{J}(\theta) + \beta\sum_{l=1}^{L}\sum_{j=1}^{s_l}\rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}, \tag{4.6}$$

kde parametr β určuje váhu penalizace řídkosti [18].

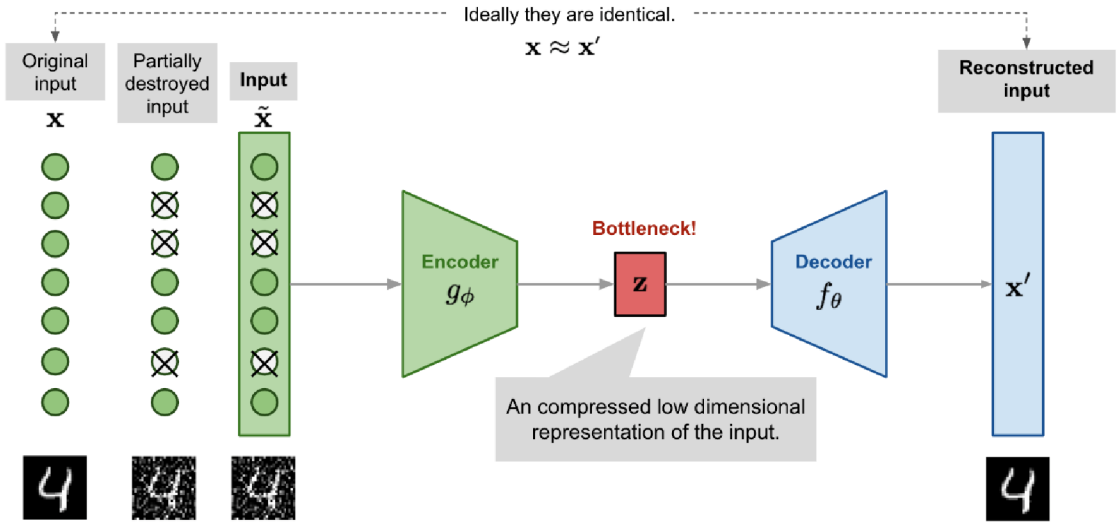
4.2 Denoising autoenkodér

Další možností, jak zamezit tomu, aby nedocházelo ke kopírování dat ze vstupu na výstup je data před přivedením na vstup sítě nejdříve zašumět. Ze vstupních dat vytvoříme poškozená data tak, že náhodné vstupy v síti vynulujeme. Poté provádíme učení stejně jako u klasického autoenkodéru. Z výstupu sítě a vstupu před poškozením vypočítáme rekonstrukční chybu, kterou se při učení snažíme minimalizovat. Díky tomu je autoenkodér schopen se naučit rekonstruovat poškozená data. Ztrátovou funkci můžeme napsat jako

$$\tilde{x}^{(i)} \sim \mathrm{M}_D(\tilde{x}^{(i)}|x^{(i)}) \tag{4.7}$$

$$L_{DAE}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}\left[x^{(i)} - f_\theta\left(g_\phi\left(\tilde{x}^{(i)}\right)\right)\right]^2 \tag{4.8}$$

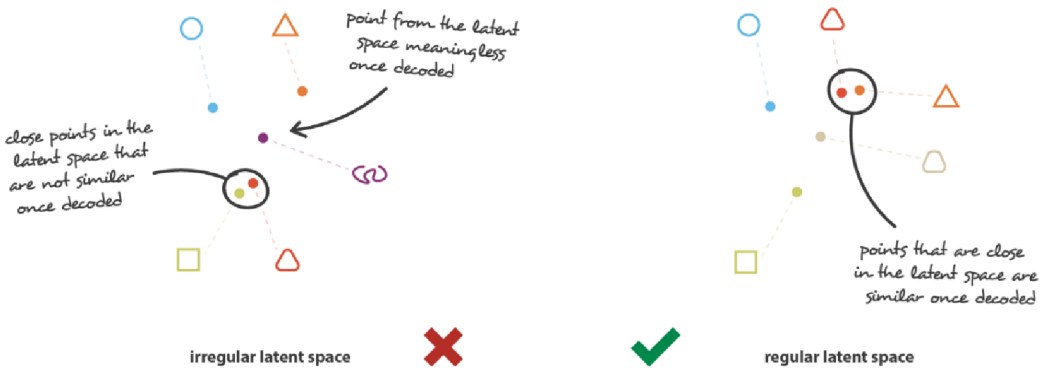
kde M_D definuje mapování vstupních dat na vstupní zašuměná data [18].



Obrázek 4.2 Princip fungování denoising autoenkodéru [18]

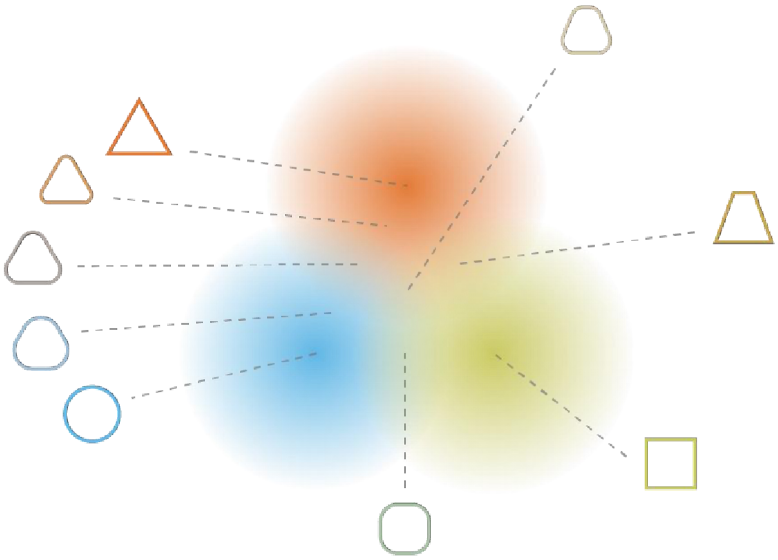
4.3 Variační autoenkodér

Variační autoenkodéry jsou autoenkodéry uzpůsobené pro generování dat. Jelikož máme po natrénování sítě k dispozici enkodér a dekodér, mohli bychom pomocí dekodéru generovat nová data tím, že bychom na vstup přivedli data z latentního prostoru. Data v latentním prostoru ale nejsou vhodně uspořádána, jelikož při trénování jsme na uspořádání v latentním prostoru vůbec nehleděli.



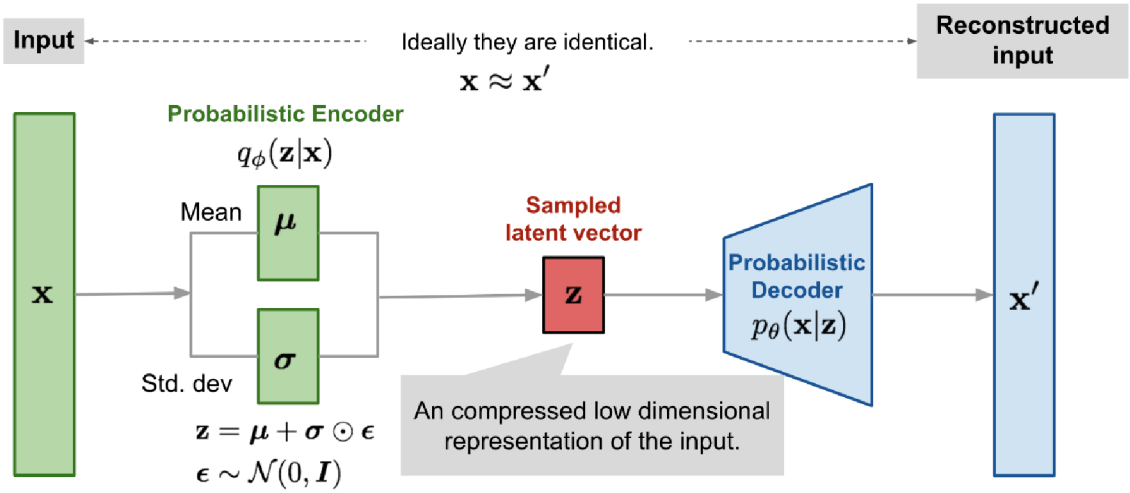
Obrázek 4.3 Rozdíl mezi pravidelným a nepravidelným rozložením v latentním prostoru [19]

Na obrázku 4.3 je vidět rozdíl mezi pravidelným a nepravidelným rozložením bodů v latentním prostoru. Pokud chceme využívat dekodér pro generování nových dat, musíme zajistit, aby dva body blízko sebe v latentním prostoru po dekódování vraceli podobná data, a aby všechny body po dekódování vraceli data které dávají smysl [19].



Obrázek 4.4 Latentní prostor s pravidelným rozložením [19]

Abychom dosáhli pravidelného rozložení latentního prostoru, bude výstupem enkodéru místo vektoru hodnot rozložení pravděpodobností. Toho můžeme dosáhnout zavedením pravděpodobnostního enkodéru a dekodéru [18].



Obrázek 4.5 Princip variačního autoenkodéru [18]

4.4 Autoenkodér pro detekci anomálií

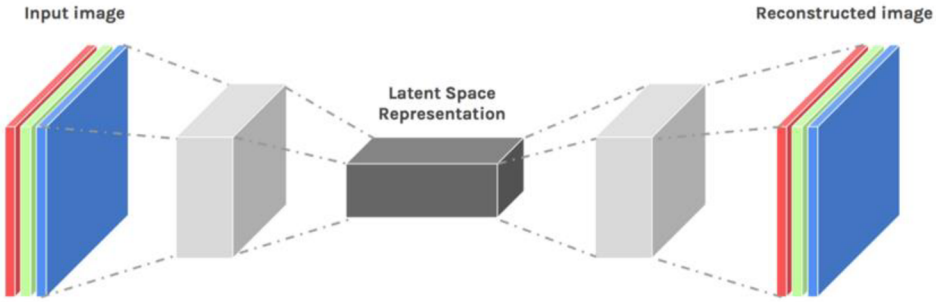
Autoenkodér lze využít pro detekci anomálií. Jelikož je autoendokér učený na redukci a následnou regeneraci dat pouze jedné třídy, regenerovaný model se pro data s anomáliemi bude hodně odlišovat [18].

Učení probíhá následujícím způsobem. Nejdříve náhodně inicializujeme parametry **θ** a **Φ**. Pro první vzor z datasetu vypočítáme rekonstruovaný výstup **x´**. Vypočítáme ztrátovou funkci **L** jako rozdíl vstupního vektoru **x** a výstupního vektoru **x´**. Pomocí algoritmu backpropagation zpětně vypočteme a upravíme postupně všechny váhy v síti. Tento postup zopakujeme postupně pro všechny vzory v datasetu. Opakovaně předkládáme celý dataset dokud není rekonstrukční chyba nižší než požadujeme.

Detekci anomálií provádíme výpočtem ztrátové funkce a porovnáním její hodnoty s předem nastaveným prahem. Práh můžeme určit tak, že si zobrazíme do grafu výsledky rekonstrukční chyby pro testovací data vzhledem ke vzorům [20].

4.5 Konvoluční autoenkodér

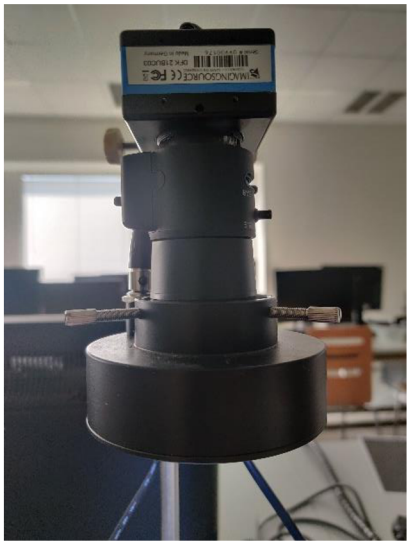
Jelikož se autoenkodéry často používají pro obrazová data, je vhodné zde využít princip konvoluce. Nahradíme plně propojené vrstvy autoenkodéru konvolučními vrstvami. Část enkodéru bude obsahovat konvoluční a pooling vrstvy a bude tedy docházet ke zmenšení rozměru ale ke zvětšení počtu kanálů. Abychom na výstupu dostali data, která mají stejný rozměr jako data vstupní, musí dekodér obsahovat vrstvy, které realizují transponovanou konvoluci nebo upsampling vrstvy se stejnými parametry jako mají vrstvy v enkodéru [21].



Obrázek 4.6 Konvoluční autoenkodér [21]

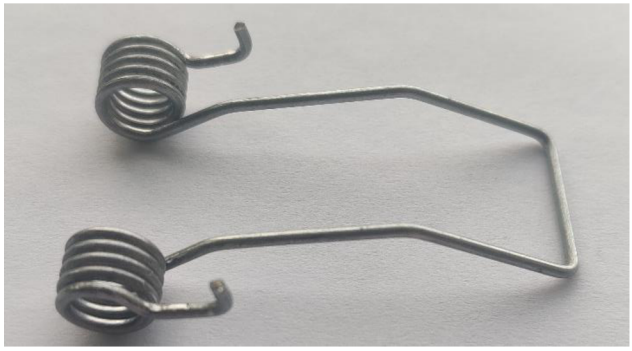
5. VYTVÁŘENÍ ANOTOVANÉHO DATASETU

Vytváření anotovaného datasetu probíhalo v laboratoři počítačového vidění v místnosti SE2.149. Pro získávání snímků byla využita průmyslová kamera DFK 21BUC03 (výrobní číslo 09900176). Kamera fotila v rozlišení 744x480 pixelů ve formátu RGB. Ke kameře bylo dále připevněno kruhové osvětlení, které bylo při snímání nastaveno na plný výkon. Kamera se nacházela 34 centimetrů nad snímaným objektem. Jako pozadí byl použit bílý kancelářský papír A4.



Obrázek 5.1 Použitá průmyslová kamera

Jako výrobek pro vytváření datasetu byla zvolena úchytná pružina pro podhledová světla. Vytvořený dataset čítá celkem 1 200 obrázků výrobku, z toho 900 vzorků jsou normální data (pozitivní třída) a 300 vzorků obsahuje anomálie (negativní třída).



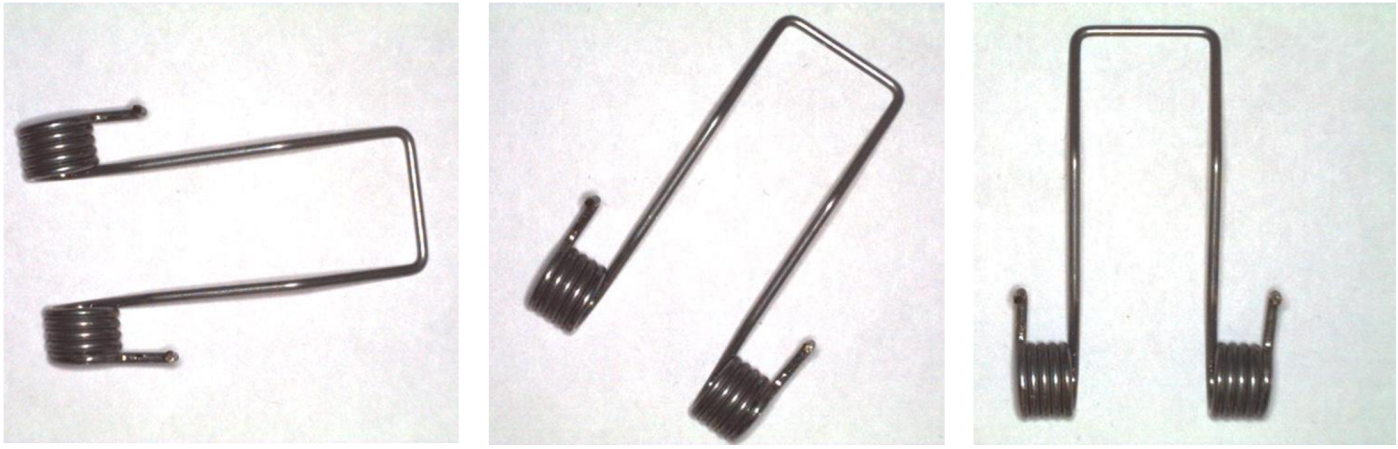
Obrázek 5.2 Úchytná pružina pro podhledová světla

Dataset byl rozdělen na trénovací množinu čítající 550 vzorků z pozitivní třídy, validační množinu obsahující 50 vzorků z pozitivní třídy a testovací množinu, která obsahuje 300 vzorků z pozitivní a 300 vzorků z negativní třídy.

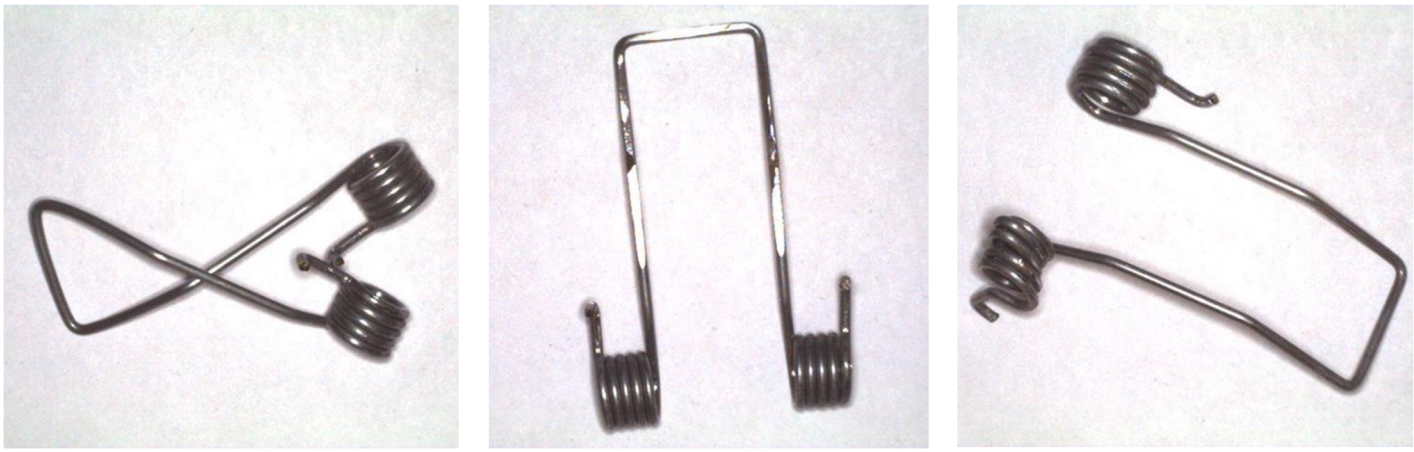
Vzory v negativním datasetu představují různé typy a velikosti mechanického nebo povrchového poškození. Typické příklady vzorů z obou tříd jsou zobrazeny na obrázcích 5.3 a 5.4. Obrázky byli pro potřebu neuronové sítě zmenšovány na rozlišení 224 x 224 pixelů. K tomuto zmenšení dochází přímo v programu pro trénování modelu.

Dataset je rozdělen do složek pro trénovací, validační a testovací data. Testovací složka obsahuje podsložky pro pozitivní a negativní třídu, trénovací množina obsahuje pouze podsložku pro pozitivní třídu. Tento způsob rozdělení datasetu byl zvolen kvůli implementaci v knihovně Keras, která pro jednoduchou správu datasetu takové dělení vyžaduje. Celý dataset je obsažen v příloze C k bakalářské práci.

Vzory s anomáliemi obsahují různě velké mechanické poškození. Některé vzory jsou různě ohnuté, některým chybí část nebo celá pružina. U některých vzorů je naopak pružina delší. Na obrázku 5.4 vpravo můžeme vidět vzor kde jsou obě pružiny deformované. Uprostřed se nachází vzor pružiny, která byla zbroušena, a proto je zde vidět velký odlesk. Dataset také obsahoval vzory které měli delší či kratší nohy vedoucí k pružinám. Dále některé vzory mají roztaženou pružinu. Dalším poškozením byl připájený cín na různých částech pružiny. Vzory obsahují také několik do sebe zamotaných pružin. Nakonec jsou v datasetu vzory, které některé tyto prvky kombinují. Jeden vzor neobsahuje pružinu vůbec.



Obrázek 5.3 Příklad vzorů z pozitivní třídy



Obrázek 5.4 Příklad vzorů z negativní třídy

6. IMPLEMENTACE KONVOLUČNÍHO AUTOENKODÉRU

Implementace autoenkodéru byla vytvářena pomocí programovacího jazyku Python verze 3.10.0 s knihovnami Keras ve verzi 2.8.0 a TensorFlow ve verzi 2.0. Mezi dalšími použitými pomocnými knihovnami byli NumPy, SkLearn a PIL. Obecně tyto knihovny umožňují jednoduchou implementaci a správu neuronových sítí.

6.1 Keras

Keras je open-source rozhraní pro programování aplikací strojového učení na platformě TensorFlow v jazyce Python. Keras se skládá ze třech hlavních částí, a to Models API, Layers API a Callbacks API.

Models API se stará o vytváření, správu a trénování modelů neuronových sítí. Další částí je Layers API, která obsahuje jednotlivé vrstvy, jakožto základní stavební blok modelů neuronových sítí a funkce pro práci s nimi.

Poslední Callbacks API umožňuje provádět akce v různých etapách učení. Mezi tyto funkce patří například zastavení učení při dosažení požadované chyby, periodické ukládání verze modelu s aktuálními hodnotami vah či ukládání statistik a logů [22].

6.2 TensorFlow

TensorFlow je open-source platforma pro strojové učení a umělou inteligenci od společnosti Google. Tato platforma umožňuje pro zrychlení výpočtů hardwarovou akceleraci pomocí grafických karet nebo speciálních ASIC Tensor processing unit určených přímo pro akceleraci výpočtů umělé inteligence. Další výhodou je možnost distribuovat výpočty na několik výpočetních zařízeních což může značně urychlit AI výpočty.

Platforma obsahuje kromě funkcí pro trénování umělé inteligence také řadu ztrátových funkcí a metrik pro vyhodnocení kvality modelu.

TensorFlow je možné využít v řadě programovacích jazyků, mezi které patří např. Python, C++, Java nebo Javascript [23].

6.3 Popis programu pro trénování sítě

K trénování modelu slouží přiložený program train.py. Program pro trénování autoenkodéru nejdříve načte dataset. K tomuto dochází pomocí třídy ImageDataGenerator, která je součástí knihovny Keras a umožňuje jednoduché načítání, spravování a předzpracování dat z datasetu [24]. Zde dochází k oříznutí obrázků ze 744x480 pixelů na požadovaný vstupní rozměr 224 x 224 pixelů.

Dále je v programu vytvořen model, tento model je vytvářen jako třída Sequential. V této třídě postupně přidáváme jednotlivé vrstvy neuronové sítě. Vrstvy jsou přidávány metodou add [25]. Keras nabízí velké množství předdefinovaných základních vrstev včetně všech popsaných v kapitole 3. [26]. Je také možné vytvořit si nové vlastní vrstvy, pro potřeby vytvářeného modelu však stačily předdefinované vrstvy.

První použitou vrstvou je 2D konvoluční vrstva. V této vrstvě je definován počet použitých kernelů, jejich rozměr, aktivační funkce a výplň. Využíváme zde aktivační funkce ReLu a jako výplň je nastavená hodnota same, toto zajistí vyplnění okrajů vstupní matice nulami. Spolu s implicitně nastaveným parametrem Stride na hodnotu 1 nedojde v této vrstvě ke zmenšení rozměru matice. Dalším implicitně nastaveným parametrem je využití prahu. Implicitně se práh u této vrstvy využívá [27].

Další vrstvou je 2D Max Pooling vrstva. Parametry jsou v této vrstvě velikost filtru a výplň. Dochází ke zmenšení rozměru dat na polovinu [28].

Na výstupu enkodéru, kde mají data nejmenší rozměr, se nachází dvě plně propojené vrstvy, obě se sto neurony. Plně propojená vrstva se zde nazývá Dense vrstva. Jako parametry jsou nastaveny počet neuronů a aktivační funkce. Implicitně tato vrstva také využívá práh [29].

Jak bylo popsáno v kapitole č.4.2, aby byl zachován rozměr výstupu stejný jako na vstupu, musíme symetricky použít vrstvy se stejnými parametry, avšak Max Pooling vrstva je zde nahrazena 2D transponovanou konvoluční vrstvou.

Po sestavení modelu je potřeba model zkompilovat. Při kompilaci modelu vybíráme optimalizační algoritmus a ztrátovou funkci. Také je zde možné pomocí parametru run_eagerly zapnout sekvenční běh programu pro učení modelu. Tato funkce je vhodná pro debugování nebo pokud nemáme k dispozici výpočetní jednotku s dostatečnou pamětí pro paralelní výpočty. Při použití tohoto módu ovšem dochází k výraznému zpomalení běhu programu. Parametr run_eagerly je implicitně nastaven na hodnotu false [30].

Důležitým parametrem pro trénování je velikost dávky – batch_size. Tento parametr určuje, kolik vzorů je možné naráz paralelně zpracovávat. Tento parametr je ovlivňován počtem trénovatelných parametrů a také velikostí jednotlivých vrstev. Maximalní velikost je poté určena velikostí paměti na výpočetní jednotce.

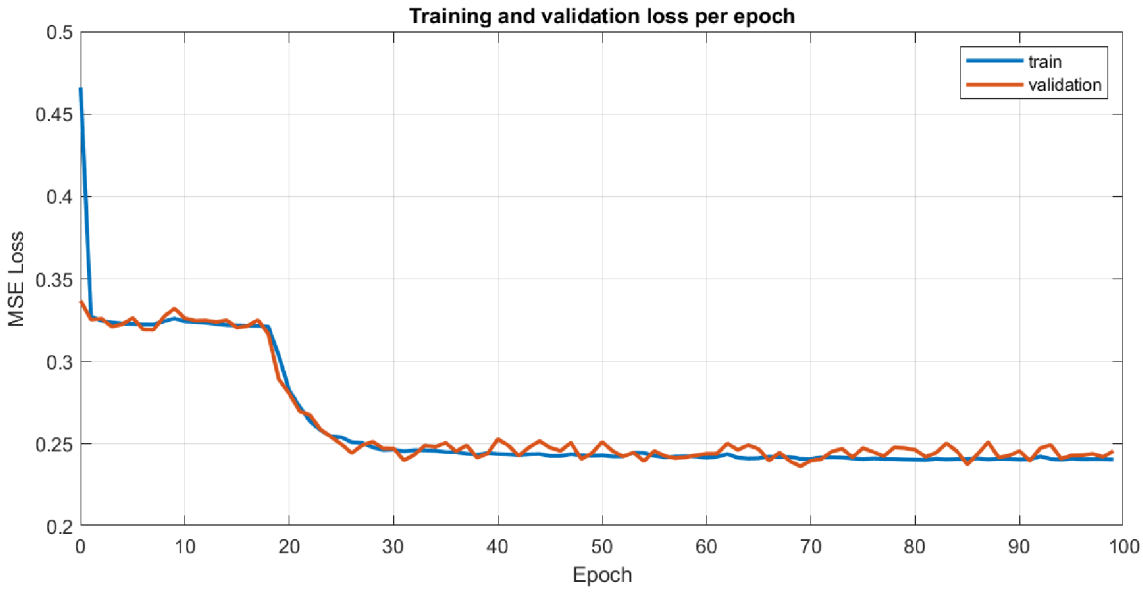
Dále definujeme parametry pro logování, statistiku a předběžné ukončení trénování. Po nastavení těchto pomocných parametrů můžeme spustit trénování pomocí funkce fit_generator. Při volání této funkce předáváme trénovací a validační dataset. Dále počet epoch, po který chceme model trénovat a počet kroků na epochu, který vypočteme pomocí dříve nastaveného parametru batch_size, jako počet vzorů v trénovací množině/batch_size. Nakonec předáme funkci dříve definované pomocné parametry pro logováni atd [30].

V této chvíli začne trénování modelu. Může dojít k chybě běhu programu kvůli vyčerpání paměti na výpočetní jednotce. V takovém případě je nutné snížit parametr batch_size. Pokud i po snížení parametru batch_size na 1 dochází k této chybě, je potřeba zvážit změnu architektury neuronové sítě, snížení rozlišení vstupního obrázku nebo použití výpočetní jednotky s větším množstvím paměti. V nejhorším případě je také možno zapnout při kompilaci modelu mód run_eagerly, jak už ale bylo řečeno v tomto módu bude trénování probíhat velice pomalu.

Po dosažení jednoho z parametrů pro ukončení trénování se zobrazí okno se vstupními a výstupními obrázky z autoenkodéru. Zde můžeme ověřit, jestli rekonstrukce probíhá správně. Po zavření okna s rekonstrukcí ze sítě se program pro trénování sítě ukončí.

6.4 Trénování sítě

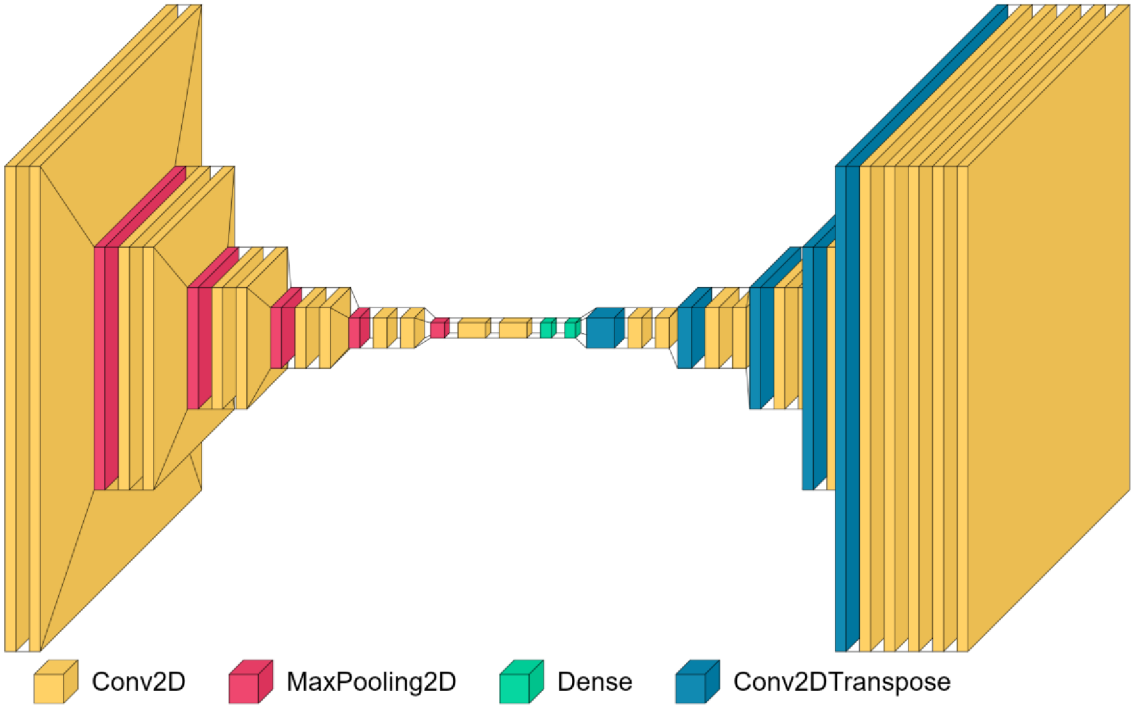
Trénování neuronové sítě probíhalo na grafické kartě NVIDIA GTX 1650 v mobilní verzi. Tato karta nabízí 4 GB paměti a 1024 CUDA jader. Pro trénování modelu byl použit optimalizační algoritmus ADAM a střední kvadratická chyba (MSE – Mean Squared Error). Celkově model obsahoval téměř 10 miliónů trénovatelných parametrů. Použitá grafická karta zvládala maximální velikost dávky 20 vzorků. Model měl nastaven pro trénování maximální počet 600 epoch. K zastavení trénování dojde, pokud proběhne 600 epoch nebo pokud se chyba během třiceti epoch nezmenší či nezvětší alespoň o 0,0005. Trénování se zastavilo po 100 epochách, kdy byla dosažena hranice pro předčasné ukončení trénování. Na obrázku 6.1 je zobrazen vývoj chyby pro trénovací a validační množinu.



Obrázek 6.1 Graf vývoje chyby vzhledem k epochám

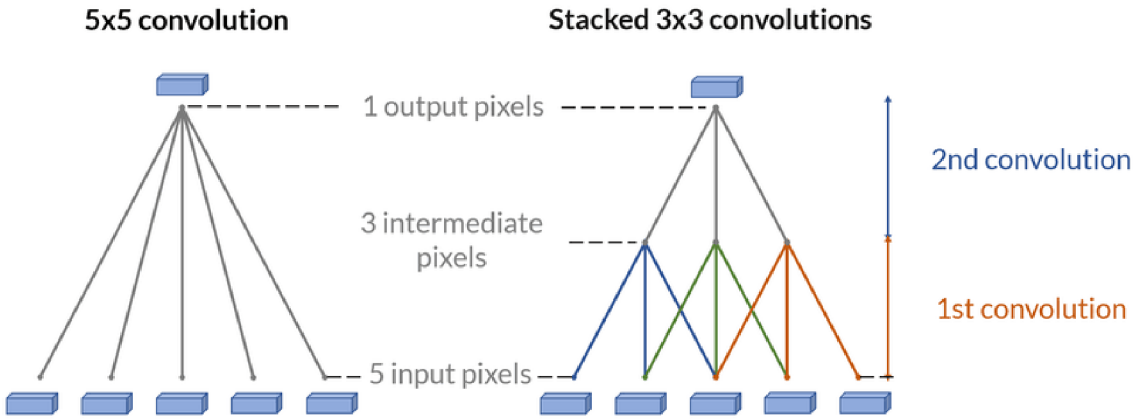
6.5 Popis vybrané architektury

Základní architektura vycházela z konvolučního autoenkodéru pro hledání prasklin v betonu [31]. Na obrázku č. 6.2 je znázorněna 3D vizualizace vytvořeného modelu. Přesný popis modelu včetně vstupních a výstupních rozměrů map je uveden v příloze A.



Obrázek 6.2 3D vizualizace vytvořeného konvolučního autoenkodéru

Model na vstupu dostává obrázky zmenšené na rozměr 224 x 224 pixelů tvořené třemi kanály RGB. Hodnoty pixelů jsou normalizovány na hodnoty od 0 do 1. Vstup poté prochází dvěma po sobě řazenými konvolučními vrstvami s velikostí kernelu 3 x 3 a hodnotou kroku 1. Použití dvou po sobě jdoucích konvolučních vrstev s velikostí filtru 3 x 3 se chová jako bychom použili jednu vrstvu s velikostí filtru 5 x 5, ale počet trénovatelných parametrů je nižší [32].



Obrázek 6.3 Příklad použití dvou konvolučních vrstev [32]

Po konvolučních vrstvách následuje max pooling vrstva s velikostí filtru 2 x 2 a velikostí kroku 2. Zde dochází ke zmenšení rozměru dat na polovinu.

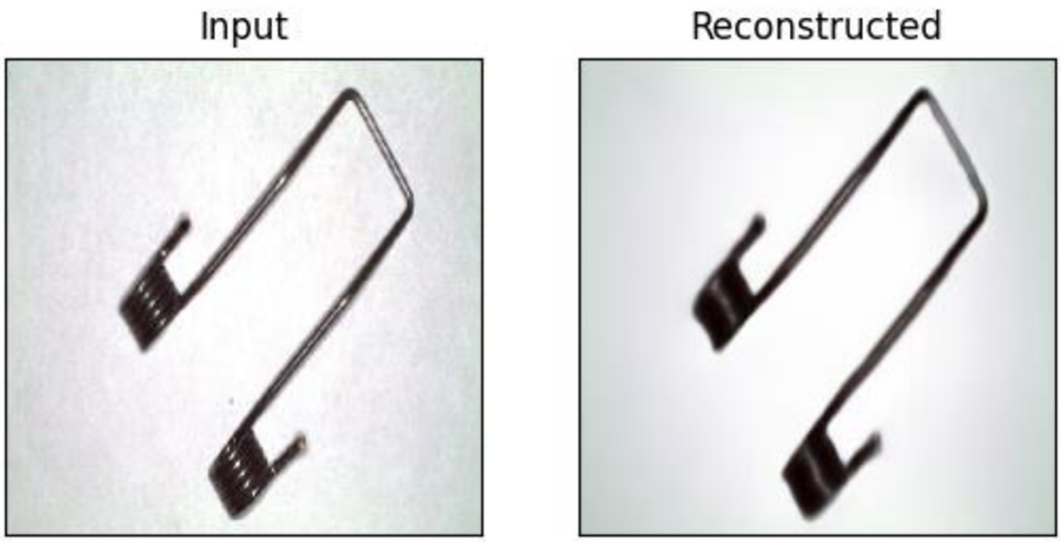
Postupně se data zmenšují až na nejmenší rozměr na výstupu enkodéru, kde mají velikost 7 x 7 pixelů a 100 kanálů. Na obrázku č. 6.4 je zobrazen jeden z těchto kanálů. Uprostřed

autoenkodéru se poté nachází dvě plně propojené vrstvy, každá se sto neurony. Plně propojené vrstvy kombinují jednotlivé extrahované rysy a pomáhají síti zjistit důležité souvislosti mezi těmito rysy.

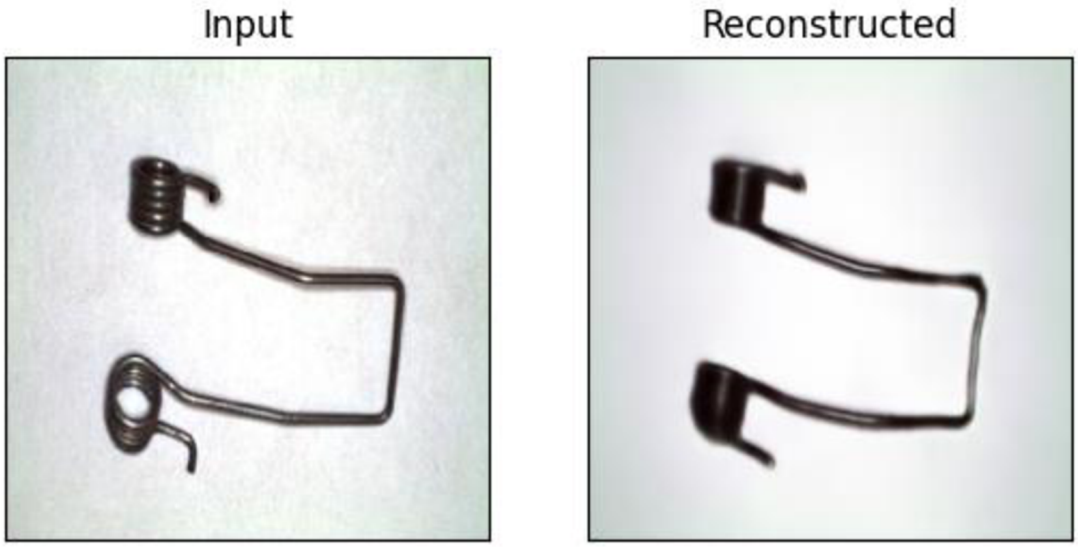
Obrázek 6.4 Vizualizace komprimovaných dat v kódu – jeden kanál

V dekodéru poté dochází k rekonstrukci dat zpět na původní velikost. To je realizováno vrstvami s transponovanou konvolucí s velikostí kernelu 2 x 2 a hodnotou kroku 2. Po těchto vrstvách je vždy výstupní mapa dvakrát větší. V poslední vrstvě je použita aktivační funkce sigmoid, která opět hodnoty jednotlivých pixelů normalizuje do rozmezí hodnot 0 až 1

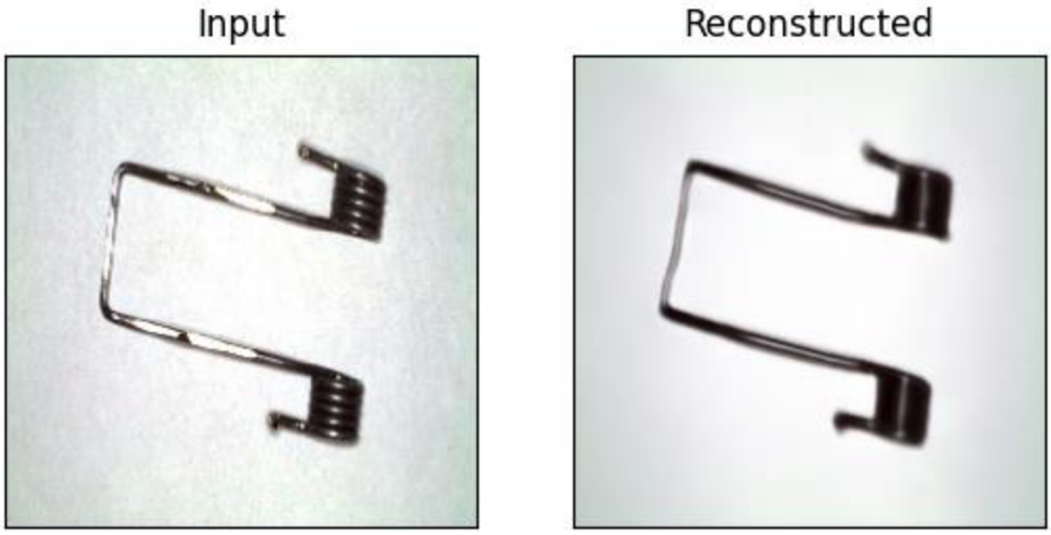
Pokud se vstupní data hodně odlišují od normálních dat, není autoenkodér schopen správně vstup rekonstruovat a dochází k velké chybě. Na obrázku č. 6.5 je možné vidět rekonstrukci pro data z normální třídy. Rekonstrukce se příliš neliší od vstupu, rekonstrukční chyba MSE zde byla 0,0012. Na obrázcích 6.6 a 6.7 je zobrazena rekonstrukce vzorků z negativní třídy. Z porovnání vstupu a výstupu ze sítě je patrné, že síť se snaží obrázek zrekonstruovat stejně jako by se jednalo o normální data, z toho plyne vysoká rekonstrukční chyba. Pro první vzor s anomálií byla chyba 0,0091 a pro druhý 0,0051.



Obrázek 6.5 Rekonstrukce obrázku z pozitivní třídy



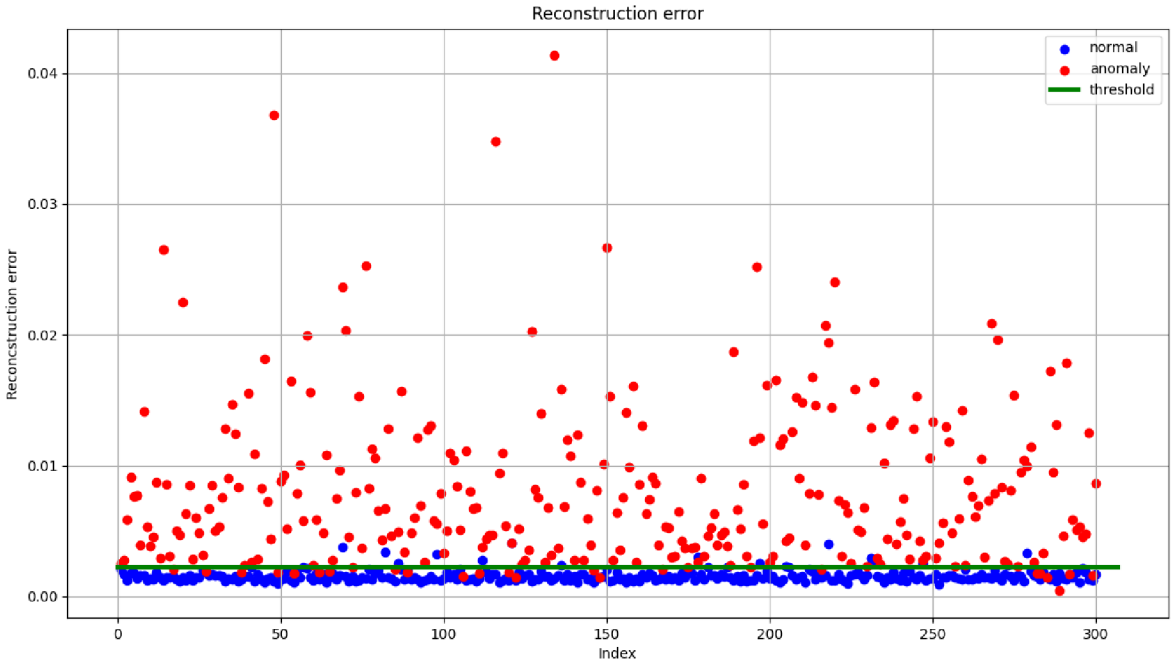
Obrázek 6.6 Rekonstrukce obrázku z negativní třídy – Ohyb pružiny



Obrázek 6.7 Rekonstrukce obrázku z negativní třídy – Zbroušený povrch

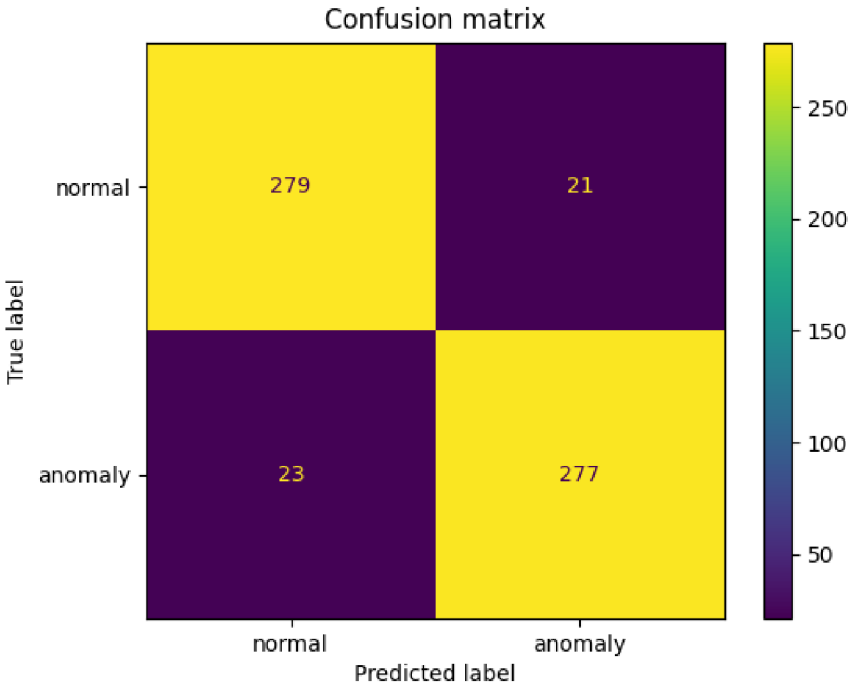
7. VYHODNOCENÍ KVALITY KLASIFIKACE

Pro vyhodnocení klasifikace slouží přiložený program evaluation.py. Pro určení prahu jsme si zobrazili graf, na kterém je možno vidět, jak velkou rekonstrukční chybu mají jednotlivé vzory a do které třídy patří. Pomocí tohoto grafu jsme navrhli práh 0,0022. Vzorky nad tímto prahem označíme za anomálie a vzorky pod ním za normální data. Na základě prahu jsme poté provedli klasifikaci všech vzorů v testovacím datasetu.



Obrázek 7.1 Určení prahu pro klasifikaci

Pro vyhodnocení kvality sestrojeného klasifikátoru jsme také vytvořili matici záměn, ze které jsme poté vypočítali jednotlivé metriky uvedené v kapitole 1.4. Z matice záměn vyplývá, že model byl schopen správně klasifikovat 279 vzorů z pozitivní třídy a 277 vzorů z třídy negativní. 23 vzorků z třídy negativní klasifikátor špatně označil za data z pozitivní třídy a naopak 21 vzorků z pozitivní třídy označil za negativní třídu.



Obrázek 7.2 Matice záměn

Některé metriky si navzájem odporují. Například budeme-li chtít zvýšit preciznost, můžeme zvýšit hodnotu prahu. Tím ale dojde ke zvýšení počtu falešných negativů a může se snížit senzitivita. Proto záleží, kterou metriku budeme při klasifikaci preferovat.

Tabulka 7.1 Porovnání klasifikačních metrik

Přesnost	Chybovost	Senzitivita	Specificita	Preciznost	F-Míra
0,93	0,07	0,92	0,93	0,93	0,92

Obrázek 7.3 Vzory z negativní třídy s nejmenší rekonstrukční chybou

Na obrázku 7.3 jsou uvedeny tři vzory z negativní třídy, které měly nejmenší rekonstrukční chybu. Obrázek vlevo neobsahuje pružinu vůbec (rekonstrukční chyba = 0,00043), prostřední obrázek obsahuje pružinu s ustřiženými úchyty (rekonstrukční chyba = 0,00149) a pružiny vpravo jsou zbroušené (rekonstrukční chyba = 0,00146). Můžeme vidět, že pro tyto vzory je obtížné poznat anomálii i pro člověka. Pro správnou klasifikaci těchto vzorů bychom nejspíše potřebovali větší rozlišení, hlubší síť a pohled na pružinu z jiného úhlu.

8. ZÁVĚR

V této práci jsem provedl řešerši a popis částí potřebných pro sestrojení umělé neuronové sítě typu autoenkodér. Popsal jsem základní princip fungování neuronových sítí, jejich učení a dělení na základě dostupnosti označených dat. Dále jsem definoval unární, binární a multi-class klasifikátor. Ukázal jsem způsob, jakým lze vyhodnotit kvalitu klasifikace pomocí matice záměn. Vytvořil jsem popis konvolučních neuronových sítí a funkcí jednotlivých vrstev. Popsal jsem princip fungovaní neuronové sítě autoenkodér a její různé varianty.

Další částí práce bylo vytvoření anotovaného datasetu čítajícího celkem 1200 vzorků pro vytvoření unárního klasifikátoru. Unární klasifikátor byl vytvářen jako konvoluční autoenkodér.

Následně byla popsána zvolená architektura a implementace v programovacím jazyce python. Nakonec byl vytvořený klasifikátor zhodnocen metodou matice záměn a z ní vycházejících metrik pro vyhodnocení kvality klasifikátoru. Z celkového počtu 600 vzorků z testovací množiny klasifikátor špatně určil pouze 44 vzorků.

LITERATURA

[1] KHAN, Shehroz S. a Michael G. MADDEN. One-class classification: taxonomy of study and review of techniques. The Knowledge Engineering Review [online]. 2014, 29(3), 345-374 [cit. 2021-12-28]. ISSN 0269-8889. Dostupné z: https://arxiv.org/pdf/1312.0049.pdf doi:10.1017/S026988891300043X

[2] TAX, David. One-class classification: Concept-learning in the absence of counter-examples [online]. Delft, 2001 [cit. 2021-12-28]. ISBN 90-75691-05-X. Dostupné z: http://homepage.tudelft.nl/n9d04/thesis.pdf. Diplomová práce. Delft University of Technology.

[3] MINOGUE, Paul. Anomaly detection – can it help in contact centre management? [online]. [cit. 2021-12-28]. Dostupné z: https://www.edgetier.com/blog/anomaly-detection-can-it-help-in-contact-centre-management/

[4] MISHRA, Utsav. Binary and Multiclass Classification in Machine Learning [online]. 2021 [cit. 2021-12-28]. Dostupné z: https://www.analyticssteps.com/blogs/binary-and-multiclass-classification-machine-learning

[5] BASHAR, Khdr. A Comparison of Classification Models for Imbalanced Datasets [online]. Budapešť, 2021 [cit. 2021-12-28]. ISBN 10.13140/RG.2.2.26879.12966. Dostupné z: https://www.researchgate.net/publication/353288055_A_Comparison_of_Classification_Models_for_Imbalanced_Datasets. Diplomová práce. Eötvös Loránd University.

[6] BAND, Amey. Multi-class Classification — One-vs-All & One-vs-One [online]. 2020 [cit. 2021-12-28]. Dostupné z: https://towardsdatascience.com/multi-class-Dclassification-one-vs-all-one-vs-one-94daed32a87b

[7] Simple guide to confusion matrix terminology [online]. 2014 [cit. 2021-12-28]. Dostupné z: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

[8] The Science of Machine Learning: Confusion matrix [online]. 2020 [cit. 2021-12-28]. Dostupné z: https://www.ml-science.com/confusion-matrix

[9] SAZLI, Murat Hüsnü. A brief review of feed-forward neural networks. *Communications, Faculty Of Science, University of Ankara* [online]. , 11-17 [cit. 2021-12-28]. ISSN 1303-6009. Dostupné z: https://dergipark.org.tr/en/download/article-file/1615327 doi:10.1501/0003168

[10] LEFKOWITZ, Melanie. Professor's perceptron paved the way for AI – 60 years too soon [online]. 2019 [cit. 2021-12-28]. Dostupné z: https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

[11] Activation Functions (Linear/Non-linear) in Deep Learning [online]. 2020 [cit. 2021-12-28]. Dostupné z: https://xzz201920.medium.com/activation-functions-linear-non-linear-in-deep-learning-relu-sigmoid-softmax-swish-leaky-relu-a6333be712ea

[12] ŠNOREK, Miroslav. Neuronové sítě a neuropočítače. Praha: Vydavatelství ČVUT, 2002. ISBN 80-01-02549-7.

[13] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [online]. 2018, Dec 15, 2018 [cit. 2021-12-28]. Dostupné z:

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[14] IBM, Cloud Education. Convolutional Neural Networks [online]. 20 October 2020 [cit. 2021-12-28]. Dostupné z: https://www.ibm.com/cloud/learn/convolutional-neural-networks
[15] YEPEZ, Juan a Seok-Bum KO. Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems [online]. 2020, 20 October 2020, 28(4), 853-863 [cit. 2021-12-28]. ISSN 1063-8210. Dostupné z: https://www.computer.org/csdl/journal/si/2020/04/08957307/1ilQS7GE9MY doi:10.1109/TVLSI.2019.2961602
[16] MATIĆ, Vladimir et al. #004 CNN Padding [online].2018, 1.11.2018 [cit. 2021-12-28]. Dostupné z: https://datahacker.rs/what-is-padding-cnn/
[17] UpSampling2D layer. Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/layers/reshaping_layers/up_sampling2d/
[18] WENG, Lilian. From Autoencoder to Beta-VAE [online]. Aug 12, 2018 [cit. 2022-01-11]. Dostupné z: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html
[19] ROCCA, Joseph. Understanding Variational Autoencoders (VAEs) [online]. Sep 24, 2019 [cit. 2022-05-19]. Dostupné z: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
[20] KHANDELWAL, Renu. Anomaly Detection using Autoencoders [online]. Jan 21, 2021 [cit. 2022-01-11]. Dostupné z: https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea
[21] DESPOIS, Julien. Autoencoders — Deep Learning bits #1 [online]. February 7th 2017 [cit. 2022-01-11]. Dostupné z: https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694
[22] About Keras. Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/about/
[23] TensorFlow. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation [cit. 2022-05-12]. Dostupné z: https://en.wikipedia.org/wiki/TensorFlow
[24] Image data preprocessing. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/preprocessing/image/
[25] The Sequential class. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/models/sequential/
[26] Keras layers API. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/layers/
[27] Conv2D layer. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/layers/convolution_layers/convolution2d/
[28] MaxPooling2D layer. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/layers/pooling_layers/max_pooling2d/
[29] Dense layer. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/layers/core_layers/dense/

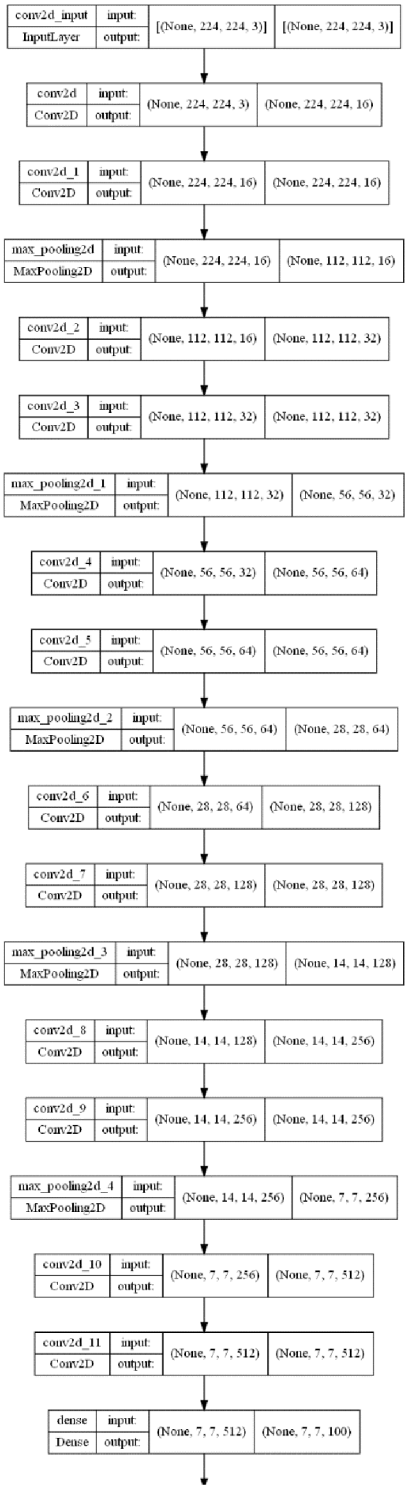
[30] Model training APIs. In: Keras.io [online]. [cit. 2022-05-12]. Dostupné z: https://keras.io/api/models/model_training_apis/
[31] CHOW, J.K., Z. SU, J. WU, P.S. TAN, X. MAO a Y.H. WANG. Anomaly detection of defects on concrete structures with the convolutional autoencoder. Advanced Engineering Informatics [online]. 2020, 45 [cit. 2022-05-12]. ISSN 14740346. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1474034620300744
[32] CHAZAREIX, Arnault. [online]. [cit. 2022-05-12]. Dostupné z: https://www.sicara.fr/blog/2019-10-31-convolutional-layer-convolution-kernel

SEZNAM PŘÍLOH

Přílohy B a C jsou uloženy na přiloženém CD.

Příloha A - Model autoenkodéru

A.1 Model – enkodér



A.2 Model – dekodér