FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Odhad dožitého věku pomocí metod strojového učení
Student:	Alexandr Czerný
Vedoucí:	Ing. Michal Štepanovský, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem BP je vytvořit pomocí metod strojového učení data-mining model, který umožní odhadnout věk jedince na základě 3D skenu význačné plochy pánevní kosti. Pokyny pro vypracování:

1. Seznamte se s formátem STL, ve kterém jsou uložena vstupní data.
2. Vytvořte regresní nebo klasifikační model, pomocí kterého lze odhadnout věk jedince.
3. Navržený model implementujte.
4. Analyzujte výsledky / schopnost odhadu věku jedince.

Jednotlivé kroky a obsah práce konzultujte s vedoucím BP.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 18. října 2021 v Praze.

Bakalářská práce

ODHAD DOŽITÉHO VĚKU POMOCÍ METOD STROJOVÉHO UČENÍ

Alexandr Czerný

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Michal Štepanovský Ph.D.
5. ledna 2023

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych především poděkovat svému vedoucímu bakalářské práce Ing. Michalu Štěpanovskému, Ph.D., za jeho odborné vedení a cenné rady při zpracování této práce. Dále bych chtěl poděkovat své rodině, přátelům a přítelkyni za jejich podporu a motivaci během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 5. ledna 2023 .....

Abstrakt

Odhad dožitého věku je jedním ze základních úkolů při vytváření biologického profilu ve forenzní antropologii. Pro odhad se využívají nejrůznější kosterní pozůstatky. Acetabulum aktuálně stojí v popředí výzkumného zájmu. Tato práce se zabývá odhadem dožitého věku ze snímků acetabula pomocí konvolučních neuronových sítí. Kromě toho je kladen důraz na předzpracování vstupních dat a na optimalizaci hyperparametrů zvoleného modelu. Výsledkem práce je regresní model strojového učení pro odhad dožitého věku ze snímků acetabula.

Klíčová slova odhad dožitého věku, acetabulum, strojové učení, předzpracování dat, regrese, klasifikace, konvoluční neuronová síť, TensorFlow

Abstract

Age at death estimation is one of the main tasks when creating biological profile in forensic anthropology. Skeletal remains are used for the estimation. Acetabulum is currently at the forefront of research interest. This thesis focuses on age at death estimation from acetabulum images using convolutional neural networks. In addition, emphasis is placed on input data preprocessing and hyperparameters tuning of selected model. The result of this thesis is a regression machine learning model for age at death estimation using acetabulum scans.

Keywords age-at-death estimation, acetabulum, machine learning, data preprocessing, regression, classification, convolutional neural network, TensorFlow

Seznam zkratek

FA	Forenzní antropologie
ML	Strojové učení
ANN	Umělá neuronová síť
AI	Umělá inteligence
ReLU	Oříznutá lineární funkce
BCE	Binární relativní entropie
CCE	Kategorická relativní entropie
MAE	Střední absolutní chyba
RMSE	Odmocnina střední kvadratické chyby
CNN	Konvoluční neuronová síť
CLAHE	Kontrastem limitovaná adaptivní ekvalizace histogramu
CANNY	Cannyho hranový detektor

Úvod

Vytvoření biologického profilu z nalezených neznámých lidských kosterních pozůstatků představuje stěžejní úkol forenzní antropologie (FA). Kromě odhadu dožitého věku mezi hlavní prvky biologického profilu patří určení pohlaví, výšky a původu. Všechny tyto charakteristiky mohou následně hrát klíčovou roli v procesu identifikace daného jedince, jelikož společně pomáhají omezit okruh možných shod.

Odhad dožitého věku se opírá o pečlivou analýzu kosterních a zubních struktur, u kterých vlivem stárnutí dochází k morfologickým změnám. Ačkoliv jde o oblast, ve které probíhal v uplynulých desetiletích intenzivní výzkum, problém odhadu dožitého věku především u dospělých jedinců stále nabízí mnoho prostoru pro zlepšení, protože univerzální metoda s uspokojivou mírou přesnosti ani dnes neexistuje. Tradiční metody odhadu dožitého věku jsou ve většině případů složité, časově náročné a vyžadují přítomnost zkušeného forenzního antropologa, který věk odhaduje na základě vizuálního hodnocení vybraných indikátorů, což nevyhnutelně vede k subjektivitě. Nepřesnost při odhadu věku je způsobena především tím, že změny, které pozorujeme na vybraných kosterních oblastech, se značně liší napříč celou populací.[1, 2, 3]

Pro odhad dožitého věku se nejčastěji využívají žebra, klíční kosti, křížová kost, kloubní plochy kyčelní kosti, pubická symfýza, nebo opotřebení zubů. V popředí výzkumného zájmu nyní stojí také acetabulum neboli jamka kyčelního kloubu. [2, 3] Ne vždy mají forenzní antropologové možnost pracovat s celou kostrou v ideálním stavu, proto je nezbytné mít širokou paletu metod, které umožňují odhad věku z různých kosterních pozůstatků a zároveň hledat nové způsoby pro odhad dožitého věku z dalších kosterních pozůstatků.

V posledních letech dramaticky roste využití strojového učení (angl. Machine Learning, ML), které tak již proniká téměř do všech oblastí, antropologickou analýzu nevyjímaje. Čím dál více odborných prací zabývající se odhadem dožitého věku z kosterních pozůstatků má za cíl vytvoření přesnější a automatizované metody založené na ML modelech. [3] Hlavním důvodem je vysoká úspěšnost těchto modelů při řešení problémů v jiných oborech spolu s jejich snadnou použitelností díky populárním knihovnám jako jsou Scikit-learn, Tensorflow, nebo PyTorch.

Cílem této práce je prozkoumat možnosti využití konvolučních neuronových sítí pro odhad dožitého věku ze snímků acetabula. Postupně tedy projdeme teorii potřebnou pro vytvoření vlastního ML modelu, dále představíme problémy, které se nejčastěji vyskytují při práci s neuronovými sítěmi a nastíníme jejich možné řešení. V další části představíme metody pro předzpracování dat, které následně využijeme při vytváření vlastního modelu. A konečně v praktické části vytvoříme regresní model pro odhad dožitého věku ze snímků acetabula, představíme dosažené výsledky a popíšeme možný prostor pro další vylepšení.

Kapitola 1
Teoretická část

Tato kapitola poskytuje teoretický základ pro pozdější práci. Začíná stručným úvodem do strojového učení (angl. Machine Learning, ML) a pokračuje představením konceptu umělých neuronových sítí (angl. Artifical Neural Network, ANN), které jsou následně využity v praktické části. Dále jsou definovány metody pro měření úspěšnosti těchto modelů. V další části je představena architektura konvolučních neuronových sítí a vrstvy, z kterých jsou tyto sítě tvořeny. Poté se zaměříme na problémy, které při učení neuronových sítí vznikají a způsoby jejich řešení. V poslední sekci jsou představeny metody pro předzpracování vstupních dat.

1.1 Strojové učení

Strojové učení je jedna z disciplín umělé inteligence (angl. Artifical Intelligence, AI), která se zaměřuje na konstrukci počítačových systémů, jež mají schopnost se automatizovaně učit a zlepšovat na základě poskytnutých informací a zkušeností. Cílem takto vytvořených počítačových systému je řešení úkolů a problémů, na které nebyly explictině naprogramoványě. Zkušenosti jsou typicky reprezentované pomocí dat, dle jejichž typu lze strojové učení rozdělit na dvě kategorie [4, 5]:

- **Učení s učitelem (angl. Supervised learning)**: Poskytnutá datová sada obsahuje jak vstupní hodnoty tak hodnoty vysvětlované proměnné (angl. target variable). Při učení se ML model snaží najít funkci, která by na základě vstupních dat nejlépe predikovala hodnotu vysvětlované proměnné. [4, 5] Klasickým příkladem takového problému může být odhad porodní váhy novorozenců, kde vstupní data jsou příčný průměr hlavy a obvod břicha získané pomocí ultrazvuku a hmotnost při narození je právě vysvětlovaná proměnná. Zde jsou tím učitelem porodní váhy dříve narozených dětí.
- **Učení bez učitele (angl. Unsupervised learning)**: V datové sadě se žádná vysvětlovaná proměnná nenachází. Cílem je data pochopit a pokusit se v nich najít skrytou strukturu. Nejčastějším příkladem učení bez učitele je shlukování (angl. clustering). [4, 5]

V naší práci využijeme pouze učení s učitelem. To lze dále rozdělit podle typu vysvětlované proměnné na dva druhy problému:

- **Klasifikace (angl. Classification)**: V případě, že vysvětlovaná proměnná nabývá jen několika málo hodnot, tedy lze ji považovat za diskrétní, hovoříme o problému klasifikace. [4] Příkladem klasifikačního problému může být spamový filtr, který hodnotí každý příchozí email a rozhoduje, zda ho má označit jako spam, či nikoliv. Nabývá-li vysvětlovaná proměnná pouze dvou hodnot, hovoříme o binární klasifikaci.

- **Regrese (angl. Regression)**: Naopak, nabývá-li vysvětlovaná proměnná tolika hodnot, že ji lze považovat za spojitou, hovoříme o regresním problému. [4] Na odhad ceny nemovitosti můžeme nahlížet jako na regresní problém.

1.2 Umělá neuronová síť

ANN je matematický model inspirovaný biologickými systémy. Motivací pro jeho vytvoření byla snaha napodobit fungování lidského mozku, především nervového systému, který pomocí desítek miliard vzájemně propojených neuronů je schopen paralelně zpracovávat informace. Na rozdíl od lidského mozku představuje neuron v kontextu ANN pouze výpočetní jednotku, kterou lze považovat za základní stavební kámen ANN. Typicky neuron přijímá několik reálných vstupů a produkuje jeden reálný výstup, jak lze vidět na obrázku 1.1. Samotný neuron nemá kapacitu pro řešení složitějších problémů, proto vznikla potřeba neurony vzájemně propojovat (propojení neuronů je analogií se synapsemi) a vytvářet tak výkonnější ANN. [6]

1.2.1 Model perceptronu

Perceptron je nejjednodušším modelem ANN, který lze použít pro binární klasifikaci a obsahuje jediný neuron. Vnitřní potenciál tohoto neuronu získáme součtem všech jeho vstupů pronásobených příslušnými vahami a interceptu (angl. bias). Vzorec pro výpočet vnitřního potenciálu neuronu je tedy: [7, 6]

$$\xi = \omega_0 + \sum_{i=1}^{n} \omega_i x_i = \omega_0 + \boldsymbol{\omega}^{\boldsymbol{T}} \boldsymbol{x}, \tag{1.1}$$

kde $\boldsymbol{x} = (x_1, ..., x_n)^T$ jsou vstupy neuronu, $\boldsymbol{\omega} = (\omega_1, ..., \omega_n)^T$ jsou váhy jednotlivých vstupů, ω_0 je intercept a n je počet vstupů. Výstup neuronu následně získáme aplikací nelineární aktivační funkce na jeho vnitřní potenciál:

$$\hat{Y} = f(\xi) = f(\omega_0 + \omega^T x), \tag{1.2}$$

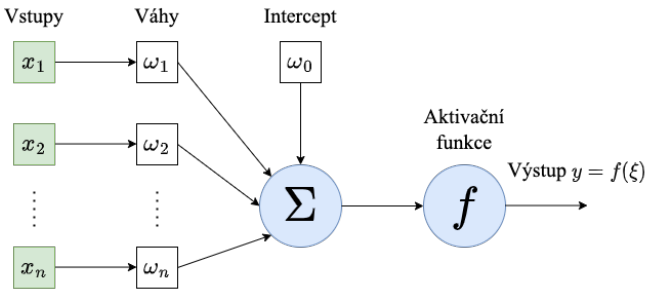
kde f je aktivační funkce. V případě perceptronu se jako aktivační funkce využívá skoková funkce, která je definována následovně:

$$f(\xi) = \begin{cases} 1 & \text{když } \xi \geq 0, \\ 0 & \text{jinak.} \end{cases} \tag{1.3}$$

Samotné perceptrony nejsou v praxi moc využitelné, protože dokáží reprezentovat pouze lineární funkce. Prakticky tak perceptron zvládne reprezentovat logické operace AND nebo OR, ale nikoliv operaci XOR, která není lineárně separovatelná.

1.2.2 Vícevrstvá neuronová síť

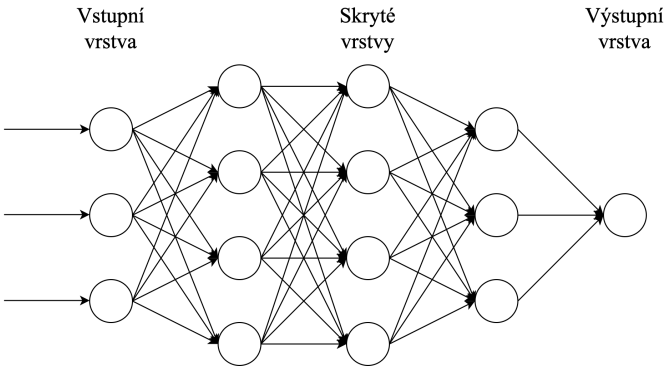
Pro překonání výše zmíněných problémů byl představen koncept vícevrstvé neuronové sítě (angl. Multi-Layer Perceptron, nebo Feed-Forward ANN), která je základním rozšířením samotného perceptronu. Jak již vyplývá z názvu tohoto modelu, neurony jsou zde uspořádany do vrstev. Ve vícevrstvé neuronové síti je vždy neuron propojen se všemi neurony ze sousedních vrstev a naopak neurony v rámci jedné vrstvy nijak propojené nejsou. Díky tomu je výstup neuronu přesně určen jeho vstupy z předchozí vrstvy a jejich váhami. Architektura vícevrstvé neuronové sítě tak vypadá následovně: [6]



Obrázek 1.1 Umělý neuron, obrázek inspirován [8].

- **Vstupní vrstva (angl. Input layer)**: Je složena z neuronů, kterým je přiřazena hodnota na základě vstupních dat celé sítě. Počet neuronů v této vrstvě je tak roven počtu proměnných vstupní datové sady. Technicky se vlastně ani nejedná o neurony v pravém slova smyslu, jelikož v této vrstvě neprovádějí žádný výpočet, nemají váhy ani aktivační funkci, pouze každý neuron předá svou hodnotu všem neuronům v další vrstvě.
- **Libovolné množství skrytých vrstev (angl. Hidden layer)**: Všechny vrstvy nacházející se mezi vstupní a výstupní vrstvou nazýváme skryté vrstvy. Neurony na svém vstupu přijímají výstupní hodnoty neuronů z předchozí vrstvy. Každá skrytá vrstva může mít libovolný počet neuronů.
- **Výstupní vrstva (angl. Output layer)**: Výstup neuronů z této vrstvy již není posílán nikam dále, ale je interpretován jako výsledek celého modelu. Podle typu řešeného problému se liší počet neuronů v této vrstvě. V případě klasifikačního problému je počet neuronů obvykle roven počtu klasifikovaných tříd, tedy v případě klasifikace do deseti tříd bude výstupem deset hodnot, které lze interpretovat jako jistotu příslušnosti k odpovídající třídě. Jestliže řešíme regresní problém, je výstupní vrstva tvořena pouze jedním neuronem, jehož výstup je přímo interpretován jako výsledek predikce.

Na obrázku 1.2 je znázorněna struktura jednoduché vícevrstvé neuronové sítě pro obecnou regresní úlohu. Vstupní vrstva obsahuje tři neurony, následují tři skryté vrstvy, dvě mají po čtyřech neuronech a třetí má tři neurony. Jelikož tato síť řeší regresní úlohu, je zakončena výstupní vrstvou s pouze jedním neuronem.



Obrázek 1.2 Příklad vícevrstvé neuronové sítě. Obrázek inspirován [9].

1.2.3 Aktivační funkce

Aktivační funkce (angl. Activation Function) určuje výstup neuronu. Pro učení neuronové sítě (procesu učení je věnována samostatná část textu) obyvkle požadujeme, aby byla jako funkce parametrů téměř všude diferencovatelná, čehož docílíme volbou vhodných aktivačních funkcí, na rozdíl od skokové aktivační funkce zmíněné u modelu perceptronu. Další vlastnost, kterou by aktivační funkce měla splňovat je nelinearita (ačkoliv může být i lineární). Je dokázáno, že ANN s využitím nelinární aktivační funkce je schopna aproximovat jakoukoliv spojitou funkci. Obecně lze aktivaci neuronu popsat rovnicí z předpisu (1.4)

$$f(\xi) = f\left(\sum_{i=1}^{n} \omega_i x_i + \omega_0\right), \tag{1.4}$$

kde x_i jsou jednotlivé vstupy, ω_i jsou váhy jednotlivých vstupů, ω_0 je intercept a $f(\xi)$ je konečný výstup neuronu. Existuje mnoho různých funkcí, z nichž je každá vhodná pro jiný typ problému. Navíc rozlišujeme aktivační funkce pro neurony ve skryté a výstupní vrstvě. V následující části si ukážeme nejpoužívanější aktivační funkce, podle typu vrstvy, ve které se neuron nachází. [10]

1.2.3.1 Příklady aktivačních funkcí neuronu ve skryté vrstvě

- **Oříznutá lineární funkce (angl. Rectified Linear Unit, ReLU)**: ReLU je jednoduchá aktivační funkce daná předpisem (1.5). Jde o nejčastěji využívanou aktivační funkci pro neurony ve skrytých vrstvách. Její hlavní výhodou je rychlost učení, která je až šestkrát rychlejší, než u hyperbolického tangentu. Dalším bonusem je nízká výpočetní náročnost, protože není nutné počítat žádné exponenty. Mírnou komplikaci představuje neexistence derivace, když $\xi = 0$, proto se většinou dodefinuje, že $f'(0) = 0$. Za určitých okolností navíc může nastat situace, kdy v důsledku například příliš velkého gradientu se váhy neuronu změní takovým způsobem, že jeho aktivace již není možná. Dochází-li k této situaci, je vhodné použít jinou aktivační funkci. [10]

$$f(\xi) = \max(0, \xi) = \begin{cases} \xi & \text{když } \xi \geq 0, \\ 0 & \text{jinak.} \end{cases} \tag{1.5}$$

- **Leaky ReLU**: Jedna z variant funkce ReLU, která pomáhá řešit problém „umírajících“ neuronů. Vidíme, že předpis (1.6) je velmi podobný předpisu funkce ReLU, rozdíl spočívá v tom, že pro hodnoty $\xi < 0$ má funkce velmi malý sklon daný konstantním koeficientem, nejčastěji roven 0,01. Díky tomu nejsou záporné hodnoty potlačeny a můžeme pro ně určit gradient. [10]

$$f(\xi) = \begin{cases} \xi & \text{když } \xi \geq 0, \\ 0,01\xi & \text{jinak.} \end{cases} \tag{1.6}$$

- **Hyperbolický tangent (angl. Hyperbolic tangent)**: Hyperbolivkcý tangent má obor hodnot $(-1, 1)$ a nabývá tak i záporných hodnot, což je při učení ANN výhodou. Využívá se především pro klasifikační problémy. [10]

$$f(\xi) = \tanh(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}} \tag{1.7}$$

1.2.3.2 Příklady aktivačních funkcí neuronu ve výstupní vrstvě

- **Lineární funkce (identita)**: Aktivační funkce dána předpisem (1.8) je využitelná pouze jako aktivační funkce neuronu ve výstupní vrstvě při řešení regresního úkolu, jelikož jejím výstupem jsou reálná čísla. [10]

$$f(\xi) = \xi \tag{1.8}$$

- **Sigmoida (angl. Sigmoid)**: Jelikož má sigmoida, jež je dána předpisem (1.9), obor hodnot $H(f) = (0, 1)$, tak se využívá především pro binární klasifikaci. Výstupní vrstva ANN řešící takový problém má pouze jeden neuron, jehož výstup interpretujeme jako pravděpodobnost příslušnosti ke třídě 1. Je-li $f(\xi) > 0,5$, predikujeme příslušnost ke třídě 1, jinak ke třídě 0. Dnes se tato funkce již příliš nepoužívá, především kvůli problému mizejícího gradientu (angl. vanishing gradient), proto se s ní setkáme nejčastěji ve studijních materiálech. [10]

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \tag{1.9}$$

- **Softmax**: Pro klasifikaci více tříd využijeme funkci Softmax s předpisem (1.10). Nechť máme $n > 2$ tříd, mezi které chceme klasifikovat, tedy ve výstupní vrstvě budeme mít n neuronů. $\xi = (\xi_1, ..., \xi_n)$ ovšem v tomto případě není vnitřní potenciál jednoho neuronu, ale vektor vnitřních potenciálů n neuronů ve výstupní vrstvě. $f_i(\xi)$ pak je aktivační funkce i-tého neuronu. Její výsledek interpretujeme jako pravděpodobnost příslušnosti k i-té třídě. [10]

$$f_i(\xi) = \frac{e^{\xi_i}}{e^{\xi_1} + ... + e^{\xi_c}} \tag{1.10}$$

1.2.4 Ztrátová funkce

Další důležitou funkcí, je ztrátová funkce (angl. Loss Function), kterou používáme pro měření úspěšnosti predikcí ANN na poskytnutých datech. Ztrátová funkce měří rozdíl mezi predikcemi ANN a skutečnými daty a obvykle je naším cílem tento rozdíl minimalizovat. Díky použití ztrátové funkce jsme následně schopni vyhodnotit vytvořený model a posoudit jeho použitelnost při řešení skutečného problému. Ztrátové funkce se opět liší v závislosti na typu řešeného problému, proto zmíníme konkrétní příklady funkcí pro klasifikační i regresní úlohy. [11, 12]

1.2.4.1 Příklady ztrátových funkcí pro klasifikační úlohy

- **Binární relativní entropie (angl. Binary Cross-Entropy, BCE)**: Nejčastější ztrátová funkce pro problém binární klasifikace, je dána předpisem (1.11), kde p_0 resp. p_1 je odhad pravděpodobnosti, že vstupní vektor přísluší k dané třídě. [12]

$$\text{BCE} = -p_0 \log p_0 - p_1 \log p_1 = -p_0 \log p_0 - (1 - p_0) \log (1 - p_0) \tag{1.11}$$

- **Kategorická relativní entropie (angl. Categorical Cross-Entropy, CCE)**: Pro nebinární klasifikaci s n různými třídami můžeme použít zobecněnou verzi binární entropie s předpisem (1.12). [11, 12]

$$\text{CCE} = -\sum_{i=1}^{n} p_i \log p_i \tag{1.12}$$

Existují i další ztrátové funkce pro klasifikační úlohy jako například Gini index, ale ty pro tuto práci nejsou důležité, proto je zde nebudeme více rozebírat.

1.2.4.2 Příklady ztrátových funkcí pro regresní úlohy

- **Střední absolutní chyba (angl. Mean Absolute Error, MAE)**: Lze ji spočítat jako střední hodnotu absolutních hodnot rozdílů mezi predikcí modelu a skutečnou hodnotu. I zde platí, že čím je odhad přesnější, tím je MAE menší. [12]

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y - \hat{y}| \tag{1.13}$$

- **Odmocnina střední kvadratické chyby (angl. Root Mean Squared Error, RMSE)**: Další populární funkcí je RMSE s předpisem (1.14). První rozdíl oproti MAE spočívá v tom, že zde sčítáme čtverce rozdílu mezi predikcí modelu a skutečnou hodnotou, kdežto u MAE jsme sčítaly absolutní hodnoty. Druhým rozdílem je, střední hodnotu rozdílů následně odmocníme. Ačkoliv v praxi se setkáváme i s alternativou, kdy odmocnění neprovádíme a mluvíme pouze o střední kvadratické chybě (angl. Mean Squared Error, MSE). [11, 12]

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2} \tag{1.14}$$

O žádné ztrátové funkci nelze apriori tvrdit, že je univerzálně nejlepší, proto často dochází k jejich kombinování pro komplexnější měření.

1.2.5 Učení neuronových sítí

Cílem jakéhokoli učícího algoritmu není nic jiného, než pro danou množinu vstupních dat najít takovou funkci, která tuto množinu co nejlépe namapuje na odpovídající výstupní data. Jak jsme již naznačili v předchozí sekci, učení je iterativní proces, během kterého je naším cílem minimalizovat chybu predikce, kterou měříme pomocí ztrátové funkce na trénovací množině dat. Samotnému rozdělení dat se budeme věnovat později. [13, 14]

1.2.5.1 Algoritmus zpětného šíření chyby

Rok 1986 znamenal pro rozvoj neuronových sítí velký průlom. Byl totiž poprvé představen učící algoritmus zpětného šíření chyby (angl. Back-Propagation). Abychom ovšem mohli pro učení použít algoritmus zpětného šíření chyby, požadujeme od neuronové sítě, aby byla diferencovatelná. Toho dosáhneme použitím vhodné aktivační funkce, například ReLU a naopak se vyhneme skokové aktivační funkci, kterou jsme představili u modelu perceptronu.

Algoritmus zpětného šíření chyby využívá pro minimalizaci chyby predikce techniku gradientního sestupu, kdy v každém kroku algoritmus spočítá hodnotu gradientu ztrátové funkce. Hlavní funkcí gradientu je, že určuje směr maximálního růstu funkce v daném bodě (při splnění všech podmínek) a jelikož je naším cílem minimalizovat chybu predikce měřenou ztrátovou funkcí, algoritmus se vydá v opačném směru gradientu. Celý učící algoritmus tak lze shrnout do tří fází, které se iterativně opakují dokud nejsou splněna kritéria pro zastavení:

- Inicializace všech vah na malá náhodná čísla
- Na trénovací množině dat spočteme průměrnou chybu predikce
- Spočteme gradient ztrátové funkce
- Provedeme přepočet vah podle předpisu (1.15), kde w^i je vektor všech vah v i-tém kroku, α je parametr rychlosti učení a $\frac{\partial L}{\partial w^i}$ je gradient ztrátové funkce.

$$w^{i+1} = w^i - \alpha\frac{\partial L}{\partial w^i} \tag{1.15}$$

Parametr rychlosti učení (angl. learning rate) je jedním z hyperparametrů modelu neurnové sítě. Určuje velikost kroku při přepočtu vah, jinými slovy kontroluje, jak rychle je model schopný se pžizpůsobit poskytnutým trénovacím datům. Nejčastěji jde o malé kladné reálné číslo mezi 0 a 1. Čím větší rychlost učení je, tím roste riziko, že model uvízne brzy v nějakém suboptimálním řešení (tj. gradientní sestup nalezne pouze lokální minimum). Na druhou stranu, je-li rychlost učení příliš nízká, může se celý proces předčasně zastavit. Hledání optimální hodnoty tohoto parametru je velmi důležitý a komplexní úkol, kterému se budeme věnovat v další části tohoto textu. [13, 14]

Jak jsme již naznačili, při použtití gradientního sestupu existuje riziko, že proces učení uvázne v lokálním minimu a nebo v sedlovém bodě, což je ještě horší varianta. Naštěstí se v praxi ukazuje, že tato rizika jsou velmi malá, jelikož lokální minima i sedlové body se v daném prostoru objevují velmi řídce a dokonce je funkční hodnota často blízká globálnímu minimu.

Zajímavostí je, že ačkoliv byl tento algoritmus objeven již v polovině 80. let minulého století, skutečný rozvoj nastal až po roce 2010, díky mnohem vyšší dostupnosti levného výpočetního výkonu.

Před příchodem algoritmu zpětného šíření chyby se nejčastěji používaly různé optimalizační algoritmy jako například Genetický algoritmus, které se ovšem ukázaly býti slepou vývojovou cestou a proto jim zde nevěnujeme více pozornosti.

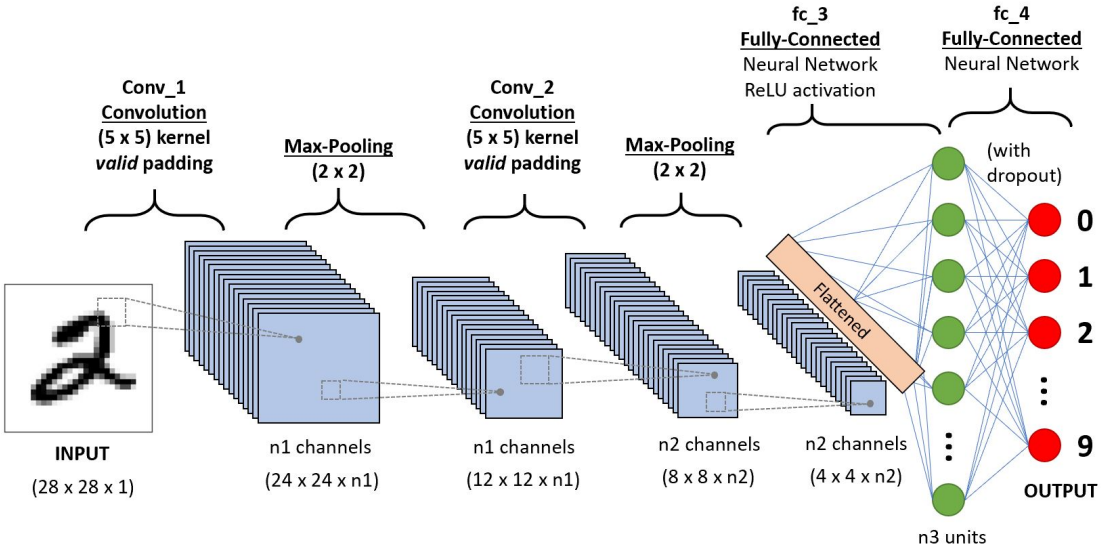
1.3 Konvoluční neuronová síť

Přestože počítače jsou dnes schopné řešit nesmírně složité úlohy, není tomu tak dávno, kdy ještě nedokázaly rozpoznat na obrázku jednoduché objekty. Proto LeCun [15] v roce 1995 přišel s architekturou konvoluční neuronové sítě (angl. Convolutional Neural Network, CNN). Zásadním přínosem této architektury je, že ze vstupních dat je schopná extrahovat složitější vzorce a z nich si sama vytvářet příznaky. Další velmi důležitou výhodou je počet parametrů, který je daleko nižší než u běžné neuronové sítě, což umožňuje rychlejší trénování těchto modelů a tedy i řešení problémů, které bychom s běžně dostupným výpočetním výkonem nebyly pomocí ANN schopni řešit. Kromě enormního výpočetního výkonu bychom navíc pro trénování takové ANN potřebovali velmi rozsáhlou datovou sadu, což může představovat další významný problém. Díky těmto vlastnostem CNN nalézají obrovské uplatnění při řešení problémů z oblasti zpracování obrazu, videa, zvuku a strojového vidění. [15, 16]

Na obrázku 1.3 si můžete prohlédnout znázornění jednoduché CNN, která je složena z několika typů vrstev, kterým se budeme následně podrobněji věnovat. Z obrázku vidíme, že na vstupu této sítě jsou černobílé obrázky o rozměrech 28×28 pixelů na kterých jsou ručně psané čísla a úkolem této CNN je tato čísla správně detekovat. Dále vidíme, že CNN má dvě konvoluční, dvě slučovací a dvě plně propojené vrstvy. Za zmínku stojí, že poslední výstupní vrstva má přesně deset neuronů, jelikož jde o problém klasifikace deseti tříd. Tyto obrázky pocházejí z tzv. MNIST datasetu, který se často využívá pro úvod do problematiky hlubokého učení a strojového vidění.

1.3.1 Konvoluční vrstva

Konvoluční vrstva (angl. Convolutional layer) je základním stavebním kamenem architektury CNN. Jak je již z názvu patrné, tak tato vrstva má co do činění s operací konvoluce. Konvoluce se provádí mezi jádrem (angl. kernel), někdy se místo jádra používá označení filtr a vstupním obrázkem, přičemž výstupem této operace je nový obrázek. Konvoluční vrstva tedy transformuje vstupní obrázek pomocí operace konvoluce za účelem extrakce informací a příznaků a po provedení konvoluce je tento nově vzniklý obrázek následně transformován vhodně zvolenou aktivační funkcí, např. ReLU. Tento proces se většinou několikrát opakuje a CNN tak obsahuje hned několik konvolučních vrstev, přičemž platí, že každá další konvoluční vrstva extrahuje komplexnější informace a příznaky, což v konečném důsledku umožňuje, aby se CNN dokázala naučit



Obrázek 1.3 Příklad jednoduché CNN, složené ze dvou konvolučních vrstev, dvou slučovacích vrstev a dvou plně propojených vrstev, která řeší problém rozpoznávání ručně psaných číslic. Obrázek z [17].

rozpoznávat vzorce, tvary a objekty. [16]

1.3.1.1 Konvoluce

Konvoluce je matematická lineární operace, která se velmi často používá v algoritmech při zpracování dvourozměrného obrazu. S výhodou zde využijeme reprezentaci obrázku pomocí matice jeho pixelů. Jádrem konvoluce rozumíme matici, která nejčastěji bývá čtvercová a měla by mít liché rozměry. Volba liché velikost jádra má tu výhodu, že zachovává dimenzionalitu a při použití techniky zarovnání nám stačí doplnit stejný počet řádků nahoře i dole a stejný počet sloupců vlevo i vpravo. Typicky se tak volí rozměry 3×3, 5×5, nebo 7×7.

Nechť tedy máme matici vstupního obrázku X a jádro konvoluce F. Výsledná matice Y (v anglické literatuře se tato matice často označuje jako Feature Map) je pak dána předpisem (1.16) a hodnota jejího prvku $y_{i,j}$ je dána předpisem (1.17), kde $f_{m,n}$ je prvek jádra a $x_{i+m,j+n}$ je prvek vstupní matice na daných souřadnicích. Tento předpis platí pro konvoluci s jednotkovým krokem a bez žádného zarovnání, což jsou parametry, kterým jsou věnovány samostatné části textu. V tomto základním případě má výstupní matice rozměry $(n_h - f + 1, n_w - f + 1, 1)$, kde h je výška vstupního obrázku, w je šířka vstupního obrázku a f je velikost jádra. Zároveň má výstupní obrázek pouze jeden kanál, protože jsem prováděli konvoluci s jediným filtrem. Rozměry výstupního obrázku jsou typicky menší, což se ovšem dá ovlivnit právě volbou dalších parametrů, jako je zarovnání.

$$Y = X * F \tag{1.16}$$

$$y_{i,j} = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} f_{m,n} x_{i+m,j+n} \tag{1.17}$$

Operaci konvoluce lze obecně popsat tak, že postupně posouváme jádro po vstupním obrázku a v každém kroku spočítáme hodnotu nového pixelu tak, že vynásobíme příslušné pixely ze

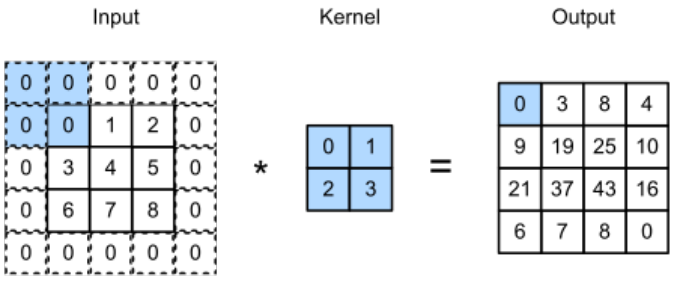
vstupního obrázku s odpovídajícími koeficienty v jádře a všechny tyto hodnoty sečteme. Následně jádro posuneme o zadaný krok a postup opakujeme.

1.3.1.2 Zarovnání

Zarovnání (angl. Padding) je technika používaná k tomu, aby se zachovala velikost výstupního obrázku z konvoluční vrstvy. Jako nevýhoda se může projevit fakt, že krajní pixely se do konvoluce nedostávají tak často, jako pixely v prostřední části obrázku a to by v konečném důsledku mohlo způsobit ztrátu cenné informace.

Řešením těchto problémů je ke krajům matice vstupního obrázku přidat dodatečné řádky a sloupce vyplněné nulami. Počet takto přidaných řádků resp. sloupců určují parametry zarovnání p_h resp. p_w. Je-li tedy například $p_h = 1$ a $p_w = 1$, znamená to, že k původnímu obrázku přidáme řádek a sloupec vyplněný nulami z každé strany. Rozměry vstupního obrázku se tedy změní na $(n_h + 2p_h) \times (n_w + 2p_w)$ a rozměry výstupního obrázku po provedení operace konvoluce budou $(n_h + p_h - f + 1) \times (n_w + p_w - f + 1)$.

Je-li zarovnání $p_h = p_w = 0$, setkáme se v angličtině s označením valid convolution. Je-li zarovnání $p_h = p_w = \frac{f-1}{2}$, tedy vstupní i výstupní obrázek má stejné rozměry, setkáme se v angličtině s označením same convolution. Na obrázku 1.4 vidíme příklad jednoduché konvoluce se zarovnáním, kde jsme k původnímu obrázku o rozměrech 3×3 přidali prázdný řádek a sloupec z každé strany. Výsledkem této konvoluce je obrázek s rozměry dány předpisem uvedeným výše $(3 + 1 - 2 + 1) \times (3 + 1 - 2 + 1) = 4 \times 4$. [18]



Obrázek 1.4 Příklad konvoluce se zarovnáním s parametry $p_h = p_w = 1$. Obrázek z [18].

1.3.1.3 Krok

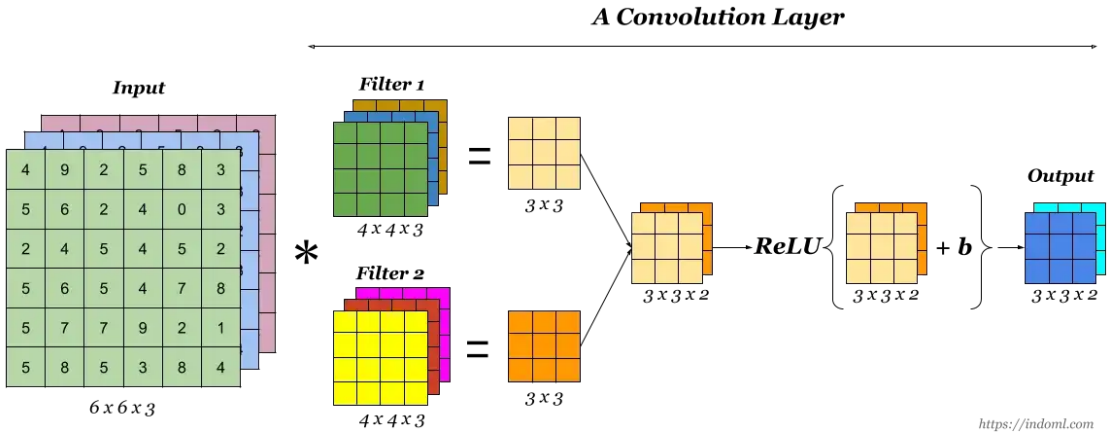
Krok (angl. Stride) je dalším důležitým hyperparametrem CNN, který udává počet pixelů, o něž se posune jádro během operace konvoluce v horizontálním a vertikálním směru. I tento parametr ovlivňuje výsledné rozměry výstupního obrázku z operace konvoluce. Označíme-li krok v horizontálním směru jako s_w a ve vertikálním směru jako s_h, potom rozměry výsledného obrázku jsou dány předpisem (1.18).

$$\left(\frac{n_h - f + p_h + s_h}{s_h}\right) \times \left(\frac{n_w - f + p_w + s_w}{s_w}\right) \tag{1.18}$$

Důležitým poznatkem je, že je-li krok malý, tak se jádro během konvoluce aplikuje na mnoho vzájemně se překrývajících oblastí a rozměry výstupního obrázku budou menší. Naopak je-li krok velký, pak se jádro během konvoluce aplikuje na méně vzájemně se překrývajících regionů a rozměry výstupního obrázku jsou větší.

Na závěr na obrázku 1.5 je znázorněn detail konvoluční vrstvy CNN. Na vstupu je barevný obrázek o rozměrech 6×6 pixelů. Během operace konvoluce na tento obrázek aplikujeme postupně dvě jádra o rozměrech 4×4 a výsledkem této operace tak jsou dva obrázky o rozměrech

3×3 (jelikož jsme nevyužili žádné zarovnání a krok byl 1 v obou směrech). Poslední krok spočívá v aplikaci aktivační funkce ReLU na výsledné obrázky. [18]



Obrázek 1.5 Příklad konvoluční vrstvy CNN. Obrázek z [19]

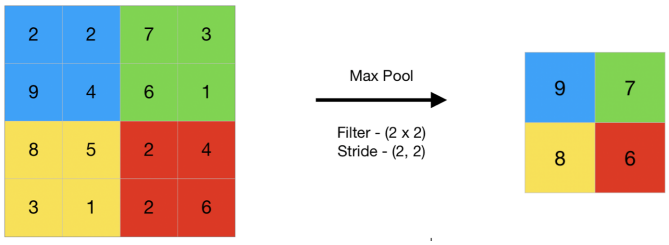
1.3.2 Slučovací vrstva

Slučovací vrstva (angl. Pooling layer) se využívá především pro zmenšení rozměrů výstupních obrázků. Primární motivací pro jejich využití je snížení počtu parametrů sítě, což má velmi pozitivní efekt na schopnost CNN odolat přeučení a kromě toho jejich využití vede ke zrychlení trénovacího procesu. Při aplikaci slučovací vrstvy na obrázek je naším cílem, aby ve výsledném obrázku zůstaly především příznaky nesoucí v sobě požadovanou informaci. Slučovací vrstvy se zatím v tomto ohledu velmi osvědčily, ačkoliv nikdo zatím nedokázal přesně popsat, čím to je dáno. Slučovací vrstvy nemají žádné parametry, které by bylo potřeba trénovat.

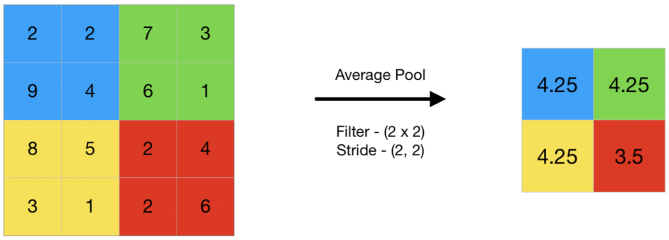
Slučování je velmi podobné konvoluci, také je zde jádro reprezentované maticí o rozměrech $f \times f$, které posouváme po vstupním obrázku vždy o daný krok (s_h, s_w). Rozdíl spočívá v tom, že v každém kroku místo aplikace operátoru konvoluce, spočítáme výsledný prvek v závislosti na typu slučování. Těch existuje několik, ale v praxi se setkáme především s následujícími dvěma. Na obrázku 1.6 vidíme příklad tzv. Max poolingu, který funguje tak, že v každém kroku se z oblasti vstupní matice ohraničené jádrem vybere prvek s maximální hodnotou, kterou pošle na výstup. Druhý typ vidíme na obrázku 1.7, jde o tzv. Average pooling, jenž výsledný prvek spočítá jako průměr prvků ohraničené oblasti matice a opět ho pošle na výstup. Average pooling častěji nalezneme pouze ve velmi hlubokých neuronových sítích.

Na rozdíl od konvoluce je slučování prováděno na každý kanál vstupního obrázku zvlášť, a tím pádem je počet kanálů u výstupního obrázku zachován. Rozměry výstupního obrázku lze spočítat podle předpisu (1.19), kde $n_h \times n_w$ je velikost vstupního obrázku, f je velikost jádra, (s_h, s_w) je velikost kroku a c je počet kanálů vstupního obrázku. [18]

$$\left(\frac{n_h - f}{s_h} + 1\right) \times \left(\frac{n_w - f}{s_w} + 1\right) \times c \tag{1.19}$$



Obrázek 1.6 Příklad Max poolingu, kde je jádro o rozměrech 2×2 aplikováno na matici o rozměrech 4×4 s krokem 2. Obrázek z [20].



Obrázek 1.7 Příklad Average poolingu, kde je jádro o rozměrech 2×2 aplikováno na matici o rozměrech 4×4 s krokem 2. Obrázek z [20].

1.3.3 Plně propojená vrstva

Plně propojené vrstvy (angl. Fully Connected Layer, často nazývána také Dense Layer) v architektuře CNN následují po konvolučních a slučovacích vrstvách. Než se ovšem výstup ze slučovací vrstvy dostane do plně propojené vrstvy, je potřeba transformovat data do formátu, s kterým jsou plně propojené vrstvy schopny pracovat. Proto se často využívá ještě tzv. Flatten layer, která je mezi poslední slučovací vrstvou a první plně propojenou vrstvou a která výstupní data ze slučovací vrstvy převádí do vhodného formátu. Plně propojená vrstva funguje stejně jako skrytá vrstva v běžné ANN a slouží k vytvoření konečné predikce CNN.

1.4 Výzvy při učení neuronových sítí

V této sekci představíme nejčastější problémy kterým čelíme při vytváření ML modelů a co je způsobuje. Dále představíme techniky, pomocí kterých lze těmto problémům do určité míry efektivně čelit.

1.4.1 Přeučení

Pravděpodobně nejčastějším problémem při trénování jakéhokoli ML modelu je přeučení (angl. Overfitting). Tento jev nastává ve chvíli, kdy se model přespříliš přizpůsobí datům, na kterých byl natrénován. Výsledkem pak je, že model generuje skvělé výsledky na trénovacích datech, ale na nových datech, s kterými se model dosud neměl možnost seznámit, jsou výsledky nedostatečně dobré. [18]

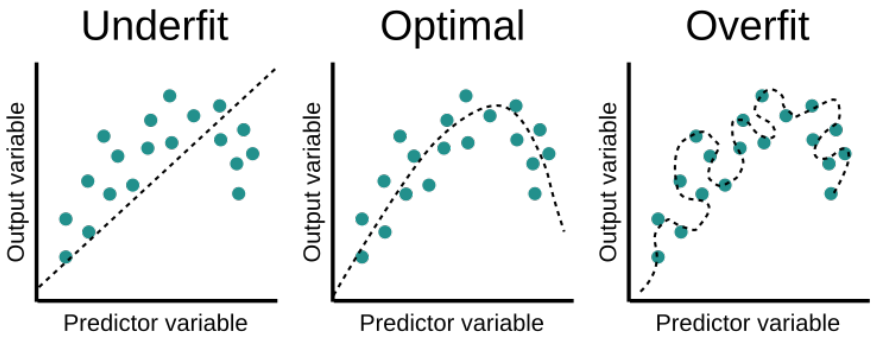
Tento problém může být způsoben celou řadou faktorů, proto vyjmenujeme ty nejběžnější.

- Komplexnost modelu: Při použití modelu s velkým množstvím vnitřních parametrů je pravděpodobnost přeučení vyšší, protože častěji dochází k tomu, že se model přizpůsobí šumu a náhodným výkyvům v trénovacích datech. K přeučení může docházet také v případě výběru složitého modelu k řešení vcelku jednoduchého problému.

- Příliš mnoho iterací učení: Po určitém počtu iterací učení modelu na trénovacích datech může opět docházet k přeučení a je tak vhodné používat nějakou optimalizační techniku, která učení včas zastaví.
- Nedostatečný rozsah trénovacích dat: Model který je natrénován na malé datové sadě může být více náchylný k přeuční, jelikož se model opět přizpůsobí šumu trénovacích dat.
- Trénovací data obsahují irelevantní informace: Pokud trénovací data obsahují irelevantní informace, model se může naučit vzorce, které nejsou použitelné pro vytváření správných predikcí na nových datech.

Na obrázku 1.8 v jeho pravé části vidíme ilustraci přeučeného modelu, kdy pozorujeme, že model se bezchybně přizpůsobil datům, na kterých byl natrénován. Oproti tomu v prostřední části vidíme optimálně naučený model, který sice na trénovacích datech nedosahuje tak dobrých výsledků, ale mnohem lépe si poradí s novými daty.

Abychom zabránili přeučení je zapotřebí pozorně vybrat správný model pro řešení daného problému, pro samotné učení připravit dostatečně kvalitní data a také správně vyhodnotit výsledky modelu na datech na kterých nebyl model trénován. V případě, že i tak problém přeučení nastává je vhodné zkusit využít další techniky, které tento problém umí řešit a kterým se budeme věnovat později.



Obrázek 1.8 Vizualizace přeučeného modelu, nedostatečně naučeného modelu a optimálně naučeného modelu. [21]

1.4.2 Nedostatečné učení

Nedostatečné učení (angl. Underfitting) se objevuje v situacích, kdy zvolený model není schopen dostatečně zachytit a přizpůsobit se trénovacím datům. Tento problém je nejčastěji způsoben zvolením příliš jednoduchého modelu, který nestačí pro komplexitu daného problému. Částečně jsme tento problém naznačili v sekci, která se věnovala modelu perceptronu, kdy jsme zmínili, že dokáží reprezentovat pouze lineární funkce a jakékoli složitější problémy jsou nad jejich možnosti. Další příčinnou může opět být nedostatek kvalitních trénovacích dat, kvůli čemuž model nemá dostatek relevantních informací pro natrénování všech parametrů modelu. [18]

Na obrázku 1.8 v jeho levé části vidíme ilustraci nedostatečně naučeného modelu, který není schopen zachytit strukturu trénovacích dat.

1.4.3 Mizející gradienty

Problém mizejících gradientů (angl. Vanishing Gradients) se týká především hlubokých neuronových sítí. Nastává během učení, když je propagovaný gradient příliš nízký a tím pádem

počáteční vrstvy, které jsou během algortimu zpětného šíření chyby upravovány až na konci, mohou zůstat beze změny. To činí učení velmi obtížné, jelikož nedochází ke správnému upravování parametrů sítě a důsledkem tohoto problému pak je nízká přesnost neuronové sítě.

Mizející gradient je poměrně často způsoben použitím hyperbolického tangentu nebo sigmoidy jako aktivační funkce. Tyto funkce totiž pro velké vstupy vracejí velmi nízké hodnoty z rozmezí (0, 1). Problém pak nastává během algoritmu zpětného šíření chyby, který počítá derivace po vrstách sítě od té poslední až k té počáteční. Tedy pro získání derivace počáteční vrstvy je podle řetízkového pravidla nutné vynásobit derivace všech vrstev od poslední vrstvy k první a gradient v takovém případě v průběhu propagace exponenciálně klesá. V extrémním případě může být gradient dokonce nula a učení se tím zastaví.

Přímočarým řešením se tak nabízí změna aktivační funkce za jinou, která velké vstupní hodnoty nemapuje na hodnoty tak malé a nezpůsobuje tak propagaci extrémně malého gradientu, jako je například ReLU nebo Leaky ReLU. Dalším částečným řešením bývá vhodná inicializace vah neuronové sítě.

Analogií k mizejícím gradientům je tzv. Exploding Gradients (nemá vhodný český překlad), což je vlastně opačný problém. Problém nastává, když jsou jednotlivé gradienty příliš velké, jelikož to opět způsobuje problémy s učením. Důvody vzniku jsou obdobné jako u mizejících gradientů, tedy například inicializace vah na příliš velké hodnoty, nebo kvůli umělému zesilování gradientů v průběhu algoritmu učení, kdy jsou gradienty propagovány směrem k počáteční vrstvě. Pro řešení se dají využít stejné metody jako pro mizející gradienty, nebo také technika ořezávání gradientů (angl. Gradient Clipping), která gradientům zabraňuje, aby přerostly určitou hranici. V případě jejího překročení jsou přenastaveny na nějakou zvolenou maximální mez. Oříznutí je většinou provedeno po spočtení gradientu v průběhu algoritmu zpětného šíření chyby. [18, 22]

1.4.3.1 Inicializace vah

Existuje mnoho způsobů jak inicializovat váhy neuronové sítě, a výběr správné metody není vždy jednoznačný, proto se obecně doporučuje v průběhu vytváření modelu vyzkoušet vícero technik a změřit úspěšnost dané techniky a následně vybrat tu, která daný problém řeší nejlépe.

- Náhodná inicializace (angl. Random Initialization): Tato metoda spočívá v inicializaci vah na náhodná čísla. Často se setkáme s inicializací vah jako hodnot normálního rozdělení se střední hodnotou rovnou 0 a standardní odchylkou $0,01$. Tato varianta v kombinaci se sigmoidou jako aktivační funkcí může způsobovat problém mizejícího gradientu. Inicializují-li se váhy jako náhodná velká čísla, aktivační funkce je namapuje blízko 1, kde se gradient mění velmi pomalu a učení tak může trvat příliš dlouho. Naopak inicializují-li se váhy jako velmi malá náhodná čísla, nastane stejný problém. [23]

- Heova inicializace (angl. He Initialization): Tuto metodu představil Kaiming He a je navržena specificky pro použití s ReLU aktivační funkcí nebo nějakou její variantou. Funguje tak, že váhy jsou nastaveny tak, aby rozptyl výstupu každého neuronu byl roven dvojnásobku rozptylu jeho vstupů. Toho se docílí opět inicializováním vah jako hodnot normálního rozdělení ovšem se střední hodnotou 0 a standardní odchylkou $\sqrt{\frac{2}{n}}$, kde n je počet vstupů do daného neuronu a všechny intercepty jsou nastaveny na 0. Motivací pro vznik této metody je právě eliminace problému mizejícího gradientu, jelikož i ReLU aktivační funkce může sama o sobě produkovat hodnoty blízko nule, což opět může vés k velmi pomalému učení neuronové sítě. Použitém této inicializace bychom měli dosáhnout většího rozptylu výstupů aktivační funkce ReLU a tím tak docílit rychlejšího a stabilnějšího učení. [23]

- Glorotova inicializace (také někdy Xavierova, angl. Glorot initialization): Funguje na stejném principu jako Heova inicializace, ale je určena pro použití v kombinaci s hyperbolickým tangentem jako aktivační funkcí. Rozdíl spočívá také v tom, že při Glorotově inicializaci nasta-

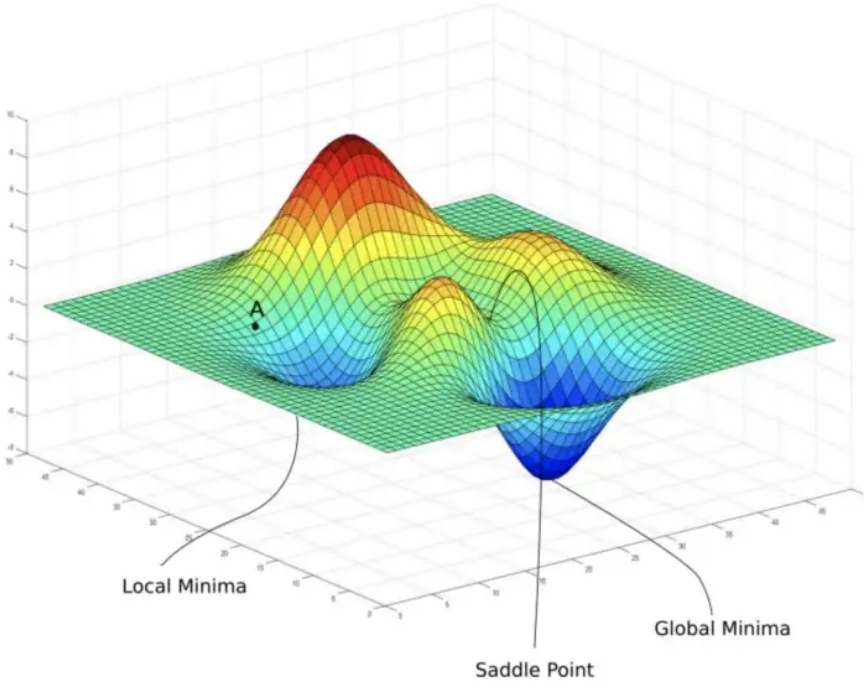
vujeme váhy jako prvky normálního rozdělení se střední hodnotou 0 a standardní odchylkou $\sqrt{\frac{1}{n}}$, kde n je počet vstupů daného neuronu. [23]

1.4.4 Problém lokálního minima

Jak jsme naznačili již v sekci věnované učení neuronových sítí, tak tento proces není nic jiného než minimalizace nějaké ztrátové funkce a při minimalizaci se snažíme dojít do globálního minima dané funkce.

Problém lokálního minima (angl. local minima problem) odkazuje na situaci, kdy se učící algoritmus neuronové sítě zasekne v lokálním minimu ztrátové funkce a nalezne tak pouze suboptimální řešení daného problému. Gradientní sestup totiž spoléhá na informace z daného okolí o kterých doufá, že ho dovedou do globálního minima. Algoritmus ovšem v žádné chvíli nemá informace o kompletní ploše a proto ve své základní verzi jednoduše doklesá do nějakého bodu o kterém prohlásí, že je lokální minimum a skončí. Algoritmus tak neumí najít nějaké lokální minimum, zapamatovat si ho a začít opět stoupat a hledat další lokální minimum, které by pak mohl s tím původním porovnat.

Nejnovější výzkum ovšem potvrzuje, že nalezení lokálního minima může mít v konečném důsledku pozitivní efekt, jelikož neurnové sítě, které jsou naučené na hledání globálního minima bývají přeučené a nedosahují tak dobrých výsledků na nových datech. Naopak se ukazuje, že větším problémem pro učení neuronových sítí mohou být sedlové body (angl. Saddle Points). V případě sedlového bodu je totiž gradient velmi nízký, kvůli čemuž je pro algoritmus složité rozhodnout, jakým směrem se dále vydat. Naštěstí se ukazuje, že s rostoucí dimenzionalitou prostoru se počet sedlových bodů exponenciálně snižuje. [18] Na obrázku 1.9 vidíme příklad funkce a označení příslušných bodů.



Obrázek 1.9 Vizualizace komplexní funkce a naznačení přitomnosti lokálního a globálního minima a sedlového bodu. Obrázek z [24].

1.4.5 Nevyváženost dat

Posledním problémem, kterému se budeme věnovat je nevyváženost dat (angl. Data Imbalance), tedy situace kdy množina dat pro učení neuronové sítě není rovnoměrně vyvážená. V případě klasifikačního problému si lze tento problém představit tak, že datová sada obsahuje pro některé třídy výrazně více vzorků než pro jiné. Například při problému odhalování podvodných bankovních transakcí budou výrazně převládat validní transakce nad těmi podvodnými. To pak může činit problém během učení neuronové sítě, jelikož ta se může přizpůsobit pouze datům z majoritně zastoupené třídy a dosahovat tak slabých výsledků při predikcích na datech z minoritně zastoupené třídy.

Proto existuje několik technik, které nám pomáhají se s tímto problémem vypořádat. Obecně můžeme tyto techniky rozdělit podle toho jestli upravují samotná data, nebo učící algoritmus. Techniky upravující data jsou v podstatě dvě, jedna vytváří nové datové vzorky (angl. Oversampling) z existujících dat minoritně zastoupené třídy a ta druhá odstraňuje datové vzorky (angl. Undersampling) z existujících dat majoritně zastoupené třídy. Oversampling využívá algoritmus SMOTE (z angl. Synthetic Minority Oversampling Technique), který funguje tak, že náhodně vybere prvek z minoritně zastoupené třídy dat a nalezne k jeho nejbližších sousedů (také z minoritně zastoupené třídy dat) z kterých vytvoříme nový prvek této třídy a dosáhneme tak vyrovnání velikosti obou tříd. Opakem tohoto přístupu pak je odstranění některých prvků majoritně zastoupené třídy. Prvky, jež chceme odstranit většinou vybíráme buďto náhodně, což je nejjednodušší a nejrychlejší přístup, nebo pomocí nějaké heuristické funkce. Hlavní nevýhodou tohoto přístupu je potenciální ztráta informací z dat, které se rozhodneme odstranit a to může vést k horším výsledkům na nových datech. Oba tyto přístupy mají své výhody i nevýhody, s kterými je dobré počítat a raději vyzkoušet, který z nich se pro danou situaci hodí lépe.

Algoritmický přístup pro vyvážení dat je založen na použití vážené ztrátové funkce, která penalizuje vzorky z majoritně zastoupené třídy a tím tak snižuje jejich váhu.

1.5 Zvýšení přesnosti neuronových sítí

V této sekci si představíme další technicky pro zvýšení přesnosti neuronových sítí a eliminaci některých problémů zmíněných v sekci předchozí. Dále si ukážeme jak správně připravit data pro učení.

1.5.1 Regularizace

Významnou technikou, kterou se snažíme zabránit přeučení neuronové sítě je nějaká forma regularizace (angl. Regularization) modelu. Princip fungování této techniky je založený na penalizaci vah neuronové sítě během jejího učení. Cílem je snížit komplexitu modelu, abychom zabránili jeho nežádoucímu přízpůsobení se pouze trénovacím datům a zlepšili tak jeho predikce na datech, na kterých nebyl trénovaný. V praxi se nejčastěji setkáme s L2 regularizací, Dropout regularizací a Včasným zastavením učení. [18, 25]

1.5.1.1 L2 regularizace

L2 regularizace, nebo také hřebenová regrese (angl. L2 Regularization, Ridge Regression) je nejčastějším typem regularizace. Tato technika rozšiřuje klasickou ztrátovou funkci o tzv. penalizační člen, který je přímo úměrný kvadrátu normy vektoru vah ω. Rozšířenou ztrátovou funkci $\hat{L}$ tak můžeme přepsat jako:

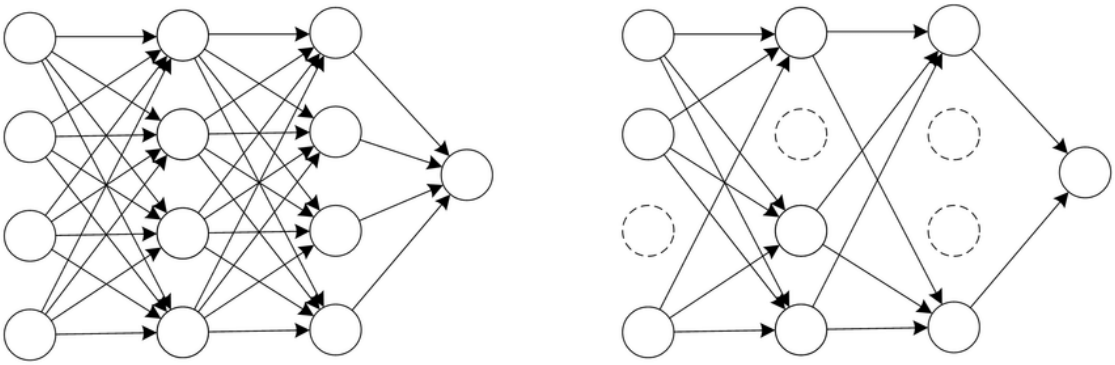
$$\hat{L}(\omega) = L(\omega) + \lambda||\omega||_2^2 \tag{1.20}$$

kde L je původní ztrátová funkce neuronové sítě a λ je nový hyperparametr modelu, který se někdy označuje jako stupeň regularizace a určuje jak moc je náše model regularizován. Vidíme, že pro $\lambda > 0$ se cílí na takové vektory ω, jež mají co nejmenší složky. Správná volba tohoto parametru není jednoduchá, jelikož s jeho rostoucí hodnotou klesá komplexita modelu, tím pádem se zbavíme přeučení modelu ovšem hrozí nám ztráta přesnosti. Naopak je-li hodnota tohoto parametru nízká, nemusíme se zbavit přeučení modelu. Proto se během ladění hyperparametrů snažíme nalézt správný balanc mezi přesností a komplexitou modelu. [18, 25]

V praxi se někdy setkáme také s L1 regularizací, která funguje na stejném principu, ovšem penalizační člen je zaveden odlišně.

1.5.1.2 Dropout

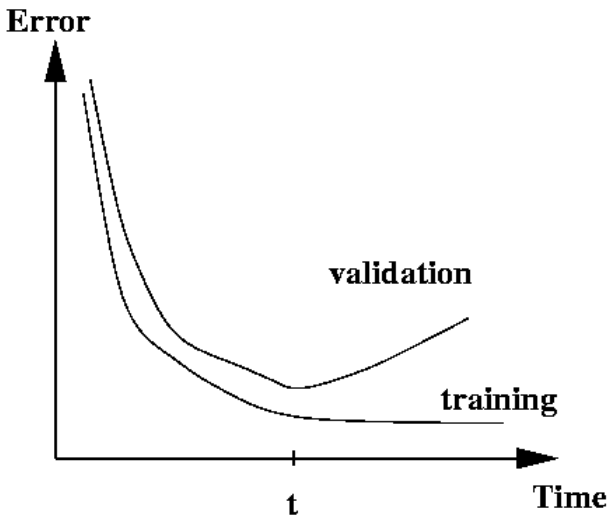
Další velmi populární a často používanou regularizační technikou je Dropout regularizace. Ta je založena na poměrně jednoduchém principu, který lze ve zkratce popsat tak, že během trénování neuronové sítě je libovolný neuron s pravděpodobností p ze sítě dočasně odstraněn. Na obrázku 1.10 vidíme znázornění jednoduché neuronové sítě, kde byly čtyři neurony vypnuty. Snadno nahlédneme, že po aplikaci Dropout regularizace je neuronová síť daleko jednodušší a díky tomu odolnější proti přeučení. Za zmínku stojí, že neurony jsou takto vypínány pouze během trénování modelu, samotné predikce vytváří kompletní síť a že tato technika je aplikována na jednotlivé vrstvy, nikoliv na neuronovou síť jako celek. [18, 25]



Obrázek 1.10 Vizualizace standardní neuronové sítě a neuronové sítě po aplikaci Dropout regularizace. Obrázek z [26].

1.5.1.3 Včasné zastavení učení

Technika včasného zastavení učení (angl. Early Stopping) je další způsob jak zabránit přeučení neuronové sítě. Během učení totiž průběžně vyhodnocuje úspěšnost modelu na validační množině (kterou blíže vysvětlíme později) dat a zastaví ho ve chvíli, kdy se úspěšnost na trénovací množině dál zvyšuje a na validační množině se buďto nezlepšuje, nebo dokonce snižuje. Dojde-li k situaci, kdy model nadále zlepšuje svou přesnost na trénovacích datech, ale na těch ostatních se již dále nezlepšuje jde o typickou ukázku přeučení modelu. Ačkoliv jde o velmi jednoduchou techniku, tak je velmi efektivní a v praxi nesmírně často využívaná. [18, 25] Na obrázku 1.11 vidíme průběh ztrátové funkce neuronové sítě během učení. Vidíme, že až do bodu t predikční chyba klesá na trénovací i validační množině dat, ovšem v tomto bodě nastává zlom a zatímco chyba na trénovacích datech klesá i nadále, tak oproti tomu na validačních datech se chyba již začíná zhoršovat.



Obrázek 1.11 Znázornění průběhu ztrátové funkce neuronové sítě na trénovací resp. validační množině dat během učení. Obrázek z [27].

1.5.2 Dávková normalizace

Dalším problémem, který může během učení neuronové sítě vyvstat je změna rozdělení hodnot vstupů skrytých vrstev neuronové sítě. V anglické literatuře se tento problém často označuje jako tzv. Internal Covariate Shift. Dávková normalizace (angl. Batch Normalization) je technika, která pomáhá tento problém efektivně řešit tím, že stabilizuje celou síť a sníží její citlivost vůči inicializaci vah a regularizaci.

Funguje tak, že každá dávka dat, která je použita v dané iteraci učení neuronové sítě, je standardizována tj. data mají střední hodnotu rovnou nule a standardní odchylku rovnou jedné a odpovídají tak standardnímu normálnímu rozdělení.

Ačkoliv bylo původní motivací pro použití této technika vyřešení problému Internal Covariant Shift, postupně se ukazuje, že tato metoda příznivým způsobem vyhlazuje ztrátovou funkci a její gradienty jsou tak prediktivnější. Díky tomu lze při učení neuronové sítě použít větší rozsah parametru rychlosti učení a celý proces tak urychlit. [18]

1.5.3 Stochastický gradientní sestup

Existují dva hlavní důvody, které spolu úzce souvisí, proč se základní algoritmus gradientního sestupu v praxi v podstatě vůbec nepoužívá. Prvním důvodem je, že výpočet všech derivací pro celou datovou sadu může trvat i dost dlouho. Druhým důvodem je velikost potřebné paměti potřebné pro tyto výpočty, jelikož ty se typicky provádí přímo v paměti. S rostoucím objemem dat tak úměrně roste doba potřebná pro trénování neuronové sítě a velikost potřebné paměti (a s ní spojené dodatečné náklady).

Stochastický gradientní sestup (angl. Stochastic Gradient Descent) je pravděpodobnostní aproximace gradientního sestupu. V každém kroku totiž algoritmus spočítá gradient pouze pro

náhodně vybranou část dat, což vede k signifikantnímu zrychlení. Teoreticky může gradient počítat pouze z jednoho náhodně vybraného datového bodu, v takovém případě může být problém, že změny vah mají typicky vyšší rozptyl, proto se ve většině variant algoritmu vybírá n datových bodů.

V praxi se navíc chceme vyhnout manuálnímu ladění parametru rychlosti učení, proto použijeme nějakou vylepšenou verzi tohoto algoritmu jako např. Adam (z angl. Adaptive Movement Estimation), AdaGrad nebo RMSProp, které pro výběr tohoto parametru využívají gradienty spočítané v předchozích iteracích. [18]

1.5.4 Rozdělení dat

Do této chvíle jsme přesnost našeho modelu měřili pouze během jeho trénování. Chceme-li ovšem získat objektivní míru přesnosti modelu na nových datech, nemůžeme jeho přesnost měřit pouze na datech na kterých byl trénován, což jsme ostatně naznačili již v sekci věnované přeučení. Intuitivně by nás tak mohlo data rozdělit na trénovací a testovací množinu, kde trénovací množinu využijeme pro učení modelu a následně jeho chybu změříme na testovacích datech, se kterými během učení nepřišel do styku. Ačkoliv takto dostaneme střízlivější odhad přesnosti, stále tento způsoben není dostatečný. Problém totiž nastává při ladění (angl. tuning) hyperparametrů, jelikož v této situaci potřebujeme měnit jejich hodnoty a vždy model s danými hyperparametry natrénovat a následně změřit jeho přesnost. Kdybychom totiž postupovali tak, že jako nejlepší model vybereme ten s nejlepší přesností změřenou na testovací množině, mohlo by se jednat o příliš optimistický odhad, protože model by byl vybrán na základě těchto dat a byl by jim tak přizpůsoben a porušili bychom tím základní pravidlo, že model nesmí během tréninku přijít s testovacími daty do styku.

Jako řešení se tedy nabízí rozdělit data na tři podmnožiny: trénovací (angl. training set), validační (angl. validation set) a testovací (angl. testing set). Neexistuje žádný univerzální poměr pro dělení dat, nicméně s následujícím postupem se setkáme poměrně často. Testovací množinu vytvoříme tak, že náhodně vybereme 20 % dat z původní množiny. Ze zbylých dat vytvoříme validační množinu tak, že z nich opět náhodně vybereme 20 % dat a zbytek prohlásíme za trénovací množinu. Je důležité data vybírat náhodně, jelikož nevíme, zda v pořadí dat není nějaké skryté pravidlo, které by mohlo ovlivnit naše výsledky. Abychom se nemuseli hledáním optimálního poměru dělení dat zabývat, byly vytvořeny i důmyslnější strategie jako např. křížová validace. [28]

1.5.4.1 Křížová validace

Jednou z nevýhod výše uvedeného postupu je dramatické snížení počtu dat použitelných pro učení modelu a také, že výsledný model může být závislý na konkrétní náhodné volbě při konstrukci trénovací a validační množiny dat. Řešením těchto problémů je tzv. křížová validace (angl. Cross Validation).

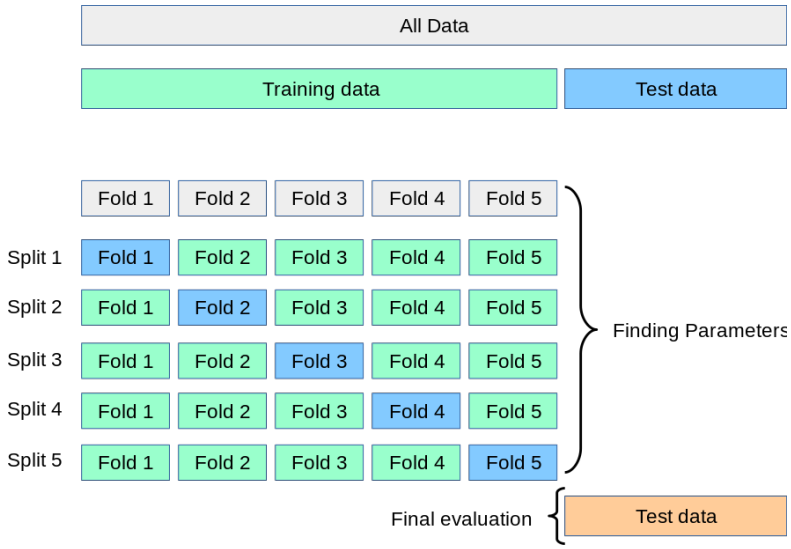
Před použitím této metody z dat opět vyčleníme testovací množinu, kterou využijeme až v samotném závěru pro spočítání výsledné přesnosti modelu, ale validační množinu již potřebovat nebudeme. V základní verzi křížové validace, rozdělíme zbytek dat na k menších částí, kde k je alespoň 2 a nejvýše rovno počtu bodů v trénovací množině dat, a následně pro každou tuto část dat opakujeme následující kroky:

- Model je natrénován na zbývajících $k - 1$ částech, které označíme jako trénovací data.
- Změříme přesnost natrénovaného modelu na dané části dat, kterou označíme jako validační množinu a přesnost si uložíme.

Z takto změřených přesností následně spočteme průměr a ten prohlásíme za přesnost modelu na trénovacích datech. Toto celé můžeme opakovat pro různé kombinace hodnot hyperparametrů

a vybrat model s nejlepší přesností. Na závěr změříme přesnost tohoto modelu na od začátku oddělené testovací množině dat.

Křížová validace může být výpočetně velmi náročná, proto základní strategie s klasickou validační množinou také nachází své uplatnění. Na obrázku 1.12 je znázornění křížové validace, kde $k = 5$. Vidíme, že nejprve data rozdělíme na trénovací a testovací, která následně rozdělíme na pět částí. Poté provedeme samotnou křížovou validaci a změříme výslednou přesnost modelu na testovací množině dat. [28]



Obrázek 1.12 Vizualizace křížové validace. Obrázek z [28].

1.6 Předzpracování dat

Předzpracování dat (angl. Data Preprocessing) je nedílnou součástí přípravy jakéhokoliv modelu strojového učení. Cílem předzpracování je data připravit do co nejvhodnější podoby pro následné trénování modelu. Předzpracování tak má zásadní vliv na jeho celkovou přesnost, proto by mu měla být také věnována řádná péče, jelikož často se lze setkat s nedostatečnou přípravou dat, kterou se pak snažíme dohnat dodatečnými úpravami modelu.

Konkrétní použité techniky se vždy v první řadě odvíjí od problému, jež chceme řešit. Protože v naší práci jsou vstupní data tvořená obrázky, zaměříme se v této sekci na představení základních technik pro zpracování obrazu.

1.6.1 Augmentace dat

Další častou nepříjemností se kterou se musíme zabývat při vytváření modelů strojového učení je nedostatek kvalitních dat. Důvodů proč je nemáme může být celá řada, např. cena jejich získání, nedostupnost atd. Pomocí augmentace (angl. Data Augmentation) jsme schopni uměle generovat dodatečná trénovací data. Cílem je vytvořit taková data se kterými se daný model může ve skutečnosti opravdu setkat. Ty jsou vytvořeny aplikací nejrůznějších transformací na již existující data. Mezi ty nejčastěji využívané patří:

- Horizontální a vertikální posun (angl. Horizontal and Vertical Shift): Posun obrázku v daném směru způsobí posun všech pixelů a tím pádem určitá část obrazu bude oříznuta a naopak vznikne region, jehož pixely budeme muset dodefinovat. Posunem obrázku se nezmění jeho

rozměry. Zároveň existuje mnoho variant jak dodefinovat nové pixely, můžeme například zvoli nějakou pevně danou konstantu, nebo se často opakuje nejbližší pixel od toho oříznutého.

- Horizontální a vertikální překlopení (angl. Horizontal and Vertical Flip): Překlopení obrázku znamená obrácení pořadí řádků resp. sloupců v případě vertikálního resp. horizontálního překlopení.
- Rotace (angl. Rotation): Rotace obrázku otočí obrázek o náhodný počet stupňů ve směru hodinových ručiček. Stejně jako v případě posunutí i rotací může dojít ke ztrátě některých pixelů a naopak ke vzniku regionů, kde je potřeba pixely dodefinovat.
- Změna jasu (angl. Brightness): Obrázek můžeme buďto ztmavit nebo zesvětlit. Cílem této augmentace je natrénovat model na datech, které vznikly v různých světelných podmínkách.
- Přiblížení/Oddálení (angl. Zoom In/Out): Přiblížení obrázku provedeme tak, že obrázek nejdříve uměle zvětšíme a následně ořízneme tak, aby objekt na obrázku byl větší a aby byly zachovány původní rozměry obrázku. Oddálení naopak provedeme tak, že obrázek zmenšíme a následně k němu přidáme dodatečné pixely, abychom opět zachovali původní rozměry obrázku.
- Přidání šumu (angl. Add Noise): Přidáním šumu obvykle cheme docílit toho, aby se model dokázal lépe přizpůsobit skutečným datům, která často nějaký šum obsahují. Existuje celá řada typů šumu, které můžeme do obrázku vkládat, za zmínku stojí např. Gaussovský šum (angl. Gaussian Noise) nebo tzv. Sůl a pepř (angl. Salt and Pepper Noise). Šum přidáváme po jednotlivých pixelech, jak vidíme z předpisu (1.21), kde $s(x, y)$ je původní hodnota pixelu na souřadnicíh x, y, $n(x, y)$ je je šum, který k danému pixelu přidáváme a $w(x, y)$ je výsledná hodnota pixelu po aplikaci šumu. Jak už samotný název napovída, Gaussovský šum vychází z Gaussova rozdělení, tedy hodnoty šumu, které přidáváme k pixelům jsou prvky tohoto rozdělení. Čím vyšší rozptyl toto rozdělení ma, tím více šumu bude k obrázku přidáno a čím vyšší je střední hodnota rozdělení, tím světlejší je výsledný obrázek. Na obrázku 1.13 vidíme efekt přidání tohoto šumu na obrázek.

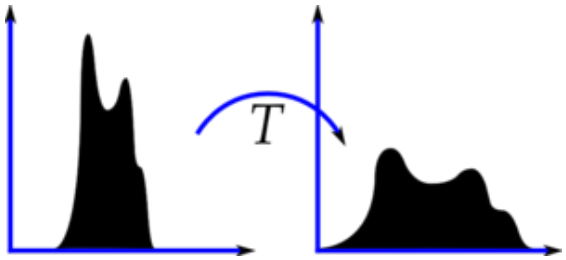
$$w(x, y) = s(x, y) + n(x, y) \tag{1.21}$$

Při výběru konkrétních metod tedy musíme být opatrní a uvažovat v kontextu dané datové sady a problematiky řešeného problému. Moderní hluboké neuronové sítě by navíc měly být schopny z obrázků správně extrahovat příznaky bez ohledu na jejich umístění. Na druhou stranu i tak může augmentace pomoci při učení příznaků, které jsou nezávislé na výše uvedených transformacích a v konečném důsledku tak zlepšit přesnost modelu. Augmentace je typicky aplikována pouze na trénovací množinu dat, což představuje rozdíl ve srovnání se změnou velikosti obrázku, takové změny musí být aplikovány na všechna data. [29]

Obrázek 1.13 Znázornění efektu přidání Gaussovského šumu. Vlevo vidíme původní obrázek a vpravo vidíme obrázek po zašumění. Obrázek z [30].

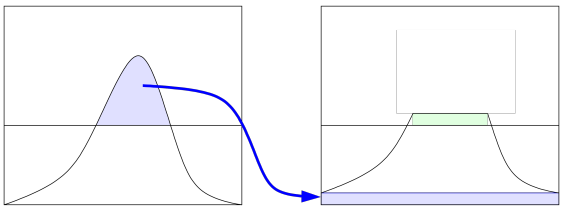
1.6.2 Ekvalizace histogramu

Histogram obrázku ukazuje rozdělení jeho intenzity. V každém jeho sloupci tedy vidíme počet pixelů s danou intenzitou. Cílem ekvalizace histogramu (angl. Histogram Equalisation, HE) je vylepšení kontrastu obrazu, konkrétně lepší rozložení intenzity. Používá se zejména když jsou jednotlivé pixely reprezentovány blízkými hodnotami. Na obrázku 1.14 vidíme příklad histogramu před ekvalizací a po ní. Vidíme, že po ekvalizace jsou pixely lépe rozdělené do všech regionů histogramu. Výhodou této metody je, že není výpočetně náročná a lze ji snadno implementovat. Často se používá právě pro rentgenové snímky kostí, kde s její pomocí lze zřetelněji zobrazit struktury.



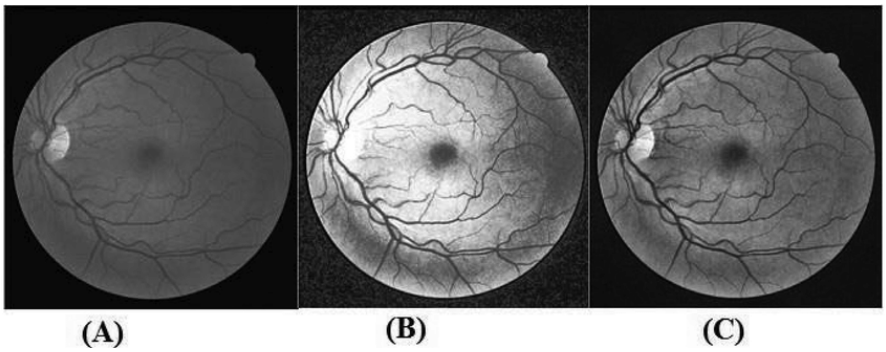
Obrázek 1.14 Příklad histogramu obrázku před ekvalizací a po ní. Obrázek z [31].

Jak vidíme na obrázku 1.16 tak samotná ekvalizace histogramu nemusí být úplně optimální. Zatímco klasické vyrovnání histogramu je globální operace, která pracuje s jedním globálním histogramem, existuje sofistikovanější varianta, Adaptivní ekvalizace histogramu (angl. Adaptive Histogram Equalisation, AHE), která obrázek rozdělí na několik menších bloků (např. 8×8 pixelů) a následně provádí ekvalizaci histogramů těchto jednotlivých bloků. Problém nastává v situacích, kdy se v těchto malých blocích vyskytuje šum, jelikož AHE tento šum zesiluje. Abychom tomuto jevu zabránili, použijeme Kontrastem limitovanou adaptivní ekvalizaci histogramu (angl. Contrast Limited AHE, CLAHE). CLAHE funguje stejně jako AHE, vyžaduje ovšem ještě zadání parametru, který zamezí zvyšování kontrastu v homogenních oblastech. Limitování kontrastu funguje tak, že před aplikací ekvalizace, zkontroluje původní histogram daného bloku, a je-li nějaký sloupec nad tímto zadaným parametrem, algoritmus tyto nadlimitní pixely rovnoměrně rozdělí mezi ostatní sloupce jak můžeme vidět na obrázku 1.15. [32]



Obrázek 1.15 Přerozdělení pixelů v histogramu nad zadaným kontrastním limitem. Obrázek z [32].

Vidíme, že po tomto přerozdělení jsou opět některé sloupce (zeleně označený region) nad tímto limitem, je-li to žádoucí, lze tento postup opakovat do té doby, než jsou všechny sloupce pod zadaným limitem. Na závěr na obrázku 1.16 můžeme porovnat tyto metody a jejich efekt na původní obrázek. Především si všimneme, že díky aplikaci CLAHE se nám podařilo odstranit nežádoucí zesvětlení některých částí obrázku, které se objevilo po obyčejné globální ekvalizaci histogramu. [32]



Obrázek 1.16 Srovnání obyčejné ekvalizace histogramu a CLAHE. Na obrázku A je původní obrázek, na obrázku B je obrázek po globální ekvalizaci histogramu a na obrázku C je obrázek po aplikaci CLAHE. Obrázek z [33].

1.6.3 Detekce hran

Detekce hran (angl. Edge Detection) je další důležitou technikou při zpracování obrazu, pomocí které jsme schopni extrahovat důležité vlastnosti zachycených objektů. Takto získané příznaky pak lze použít pro další úkoly. Další výhodou je, že tato technika pomůže obrázek zjednodušit a zbavit se tak některých informací, které pro nás nemusí být podstatné, což je výhodné při dalším zpracování, protože například učení ML modelu na jednodušších datech může být signifikantně rychlejší.

Hrany v obrázku obecně můžeme popsat jako náhlou změnu intenzity sousedících pixelů. Při detekci hran hledáme právě tyto změny. Existuje hned několik způsobů jak hrany v obraze detekovat, při výběru konkrétní metody hraje vždy roli charakteristika obrázku a požadovaný výsledek. Několik základních technik si proto představíme o něco podrobněji.

1.6.3.1 Sobelův operátor

Sobelův operátor (angl. Sobel Operator) využívá dvě jádra, které se zkonvolují se vstupním obrazem. Dvě jádra jsou použita abychom detekovali náhlé změny jasu v horizontálním a vertikálním směru. Označíme-li vstupní obraz jako A, pak podle předpisů (1.22) resp. (1.23) získáme výsledné obrazy G_x resp. G_y, jež v každém bodě obsahují spočtený odhad derivace (derivace je aproximována pomocí konvoluce s příslušným jádrem). Předpokládáme, že v místě nějaké hrany bude hodnota derivace vysoká. Na stejném principu fungují další metody, které se většinou liší hlavně použitým jádrem (např. Scharrův operátor nebo Prewittův operátor). V některých případech využijeme i větší jádra než 3×3, jelikož mohou být odolnější vůči šumu. Zatímco všechny tyto metody pracovali s první derivací, tak hrany lze hledat i pomocí druhé derivace. Tyto metody většinou pracují s Laplaceovým operátorem a kombinují ho s nějakou metodou vyhlazování pro odstranění šumu.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{1.22}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{1.23}$$

1.6.3.2 Cannyho hranový detektor

Pokročilejší způsob detekce hran je použití Cannyho hranového detektoru (angl. Canny Edge Detector, Canny), který zahrnuje několik fází. V první fázi dochází k odstranění šumu pomocí Gaussova filtru. Vyhlazený obrázek poté zkonvolujeme se Sobelovým jádrem v obou směrech a získáme opět dva obrázky G_x a G_y. Z těchto obrázků získáme velikost a směr gradientu pro každý pixel následnově:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.24}$$

$$\Theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{1.25}$$

Směr gradientu je vždy kolmýk hraně a je zaokrouhlen na jeden ze čtyř úhlů reprezentujících vertikální, horizontální a dva diagonální směry. Po získání velikosti a směru gradientu se algoritmus pokusí odstranit všechny pixely, které nepředstavují hranu. Každý pixel je zkontrolován, jestli je lokálním maximem ve svém okolí ve směru gradientu. Pokud pixel není označen jako lokální maximum je pro další fázi algoritmu tzv. potlačen, tedy nastaven na 0. V posledním kroku je naším cílem ponechat pouze opravdu relevantní hrany. Zvolíme dva prahy T_1 a T_2. Všechny hrany s gradientem větším něž T_1 ponecháme jako hrany a všechny hrany s gradientem nižším než T_2 odstraníme. Hrany jejichž gradient leží mezi těmito prahy klasifikujeme jako relevantní hrany v závislosti na tom, jestli sousedí s pixelem, jehož gradient je nad prahem T_1. Na obrázku 1.17 si můžeme prohlédnout příklad detekce hran pomocí Canny.

Obrázek 1.17 Příklad detekce hran pomocí Canny. Obrázek z [34].

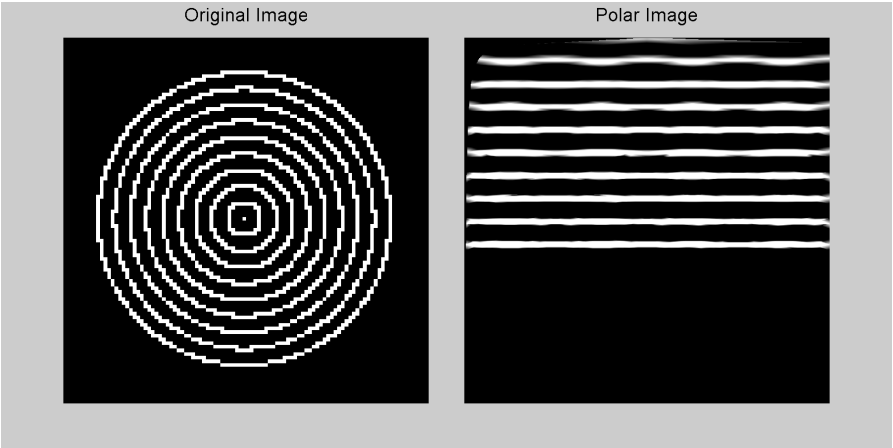
1.6.4 Polární souřadnicový systém

V některých situacích se nám také může hodit přechod ze standardního Kartézského systému souřadnic do Polárního systému souřadnic. Tato soustava souřadnic v rovině totiž může být vhodnější pro analýzu některých vlastností a vzorců v daném obraze. V polární soustavě souřadnic je bod definován dvěma souřadnicemi r, který udává vzdálenost bodu od počátku soustavy souřadnic a φ, jež udává úhel spojnice tohoto bodu a počátku od zvolené osy, které nejčastěji odpovídá osa x kartézského systému. Chceme-li tedy převést obrázek z kartézského systému do polárního, musíme převést jeho jednotlivé pixely a použijeme následující předpisy, kde x, y jsou souřadnice bodu v kartézském systému souřadnic:

$$r = \sqrt{x^2 + y^2} \tag{1.26}$$

$$\varphi = \begin{cases} \text{arctg}(\frac{y}{x}), & \text{je-li } (x > 0) \wedge (y > 0), \\ \text{arctg}(\frac{y}{x}) + \pi, & \text{je-li } (x < 0), \\ \text{arctg}(\frac{y}{x}) + 2\pi, & \text{je-li } (x > 0) \wedge (y < 0). \end{cases} \tag{1.27}$$

Na obrázku 1.18 vidíme transformaci obrázku soustředných kružnic po převodu do polárního systému. Lze si všimnout, že z kružnic se staly přímky. Důležité je také zmínit, že jako počátek soustavy souřadnic jsme označili střed obrázku.



Obrázek 1.18 Srovnání stejného obrázku ve dvou různých souřadnicových systémech. Obrázek z [35].

Kapitola 2
Implementace

Tato kapitola shrnuje prkatickou část naší práce. Nejprve jsou představeny nejdůležitější použité technologie pro implementaci navržených modelů, následně jsou popsány poskytnutá data. Dále představíme architekturu použitých modelů a dosažené výsledky. Na závěr také popíšeme prováděné experimenty.

2.1 Seznam použitých technologií

2.1.1 Python

Pro samotnou implementaci jsme vybrali moderní vysokoúrovňový skryptovací jazyk Python. Ten je široce používaný v mnoha odvětvích, především však nachází uplatnění při analýze dat a vytváření ML modelů. Mezi hlavní přednosti tohoto jazyka patří hlavně jeho jednoduchost a s ní spojená rychlost vývoje. Hlavním důvodem pro výběr tohoto jazyka jsou ovšem již existující nástroje pro zpracování dat a vytváření ML modelů, které nám tak výrazně usnadní práci.

2.1.2 Jupyter notebook

Jupyter notebook je webové interaktivní rozhraní určené pro vývoj a rychlé prototypování. Na rozdíl od klasického textového editoru je toto prostředí rozděleno na jednotlivé buňky, kde každá buňka může obsahovat kód, text, nebo další obsah. Každou buňku můžeme samostatně spustit a její výstup vidíme přímo pod danou buňkou. Toto prostředí je tak velmi populární pro tvorbu a sdílení dokumentů, jež obsahují spustitelný kód. Ačkoliv dříve byl tento nástroj zamýšlen pouze pro použití s programovacími jazyky Julia, Python a R, tak v dnešní době již podporuje přes 40 programovacích jazyků.

2.1.3 TensorFlow a Keras

TensorFlow je volně dostupná knihovna pro vytváření ML modelů a práci s AI. Byla vytvořena vývojáři společnosti Google a poprvé představena v roce 2015. Usnadňuje řešení celé škály problémů z oblastí zpracování přirozeného jazyka, zpracování obrazu a vědeckých výpočtů. Jednou z hlavních vlastností knihovny TensorFlow je schopnost využívat grafickou nebo jinou akceleraci a tím signifikantně zrychlovat učení ML modelů, které potřebují velké množství dat. To umožňuje řešení i velmi komplexních problémů. Jako bonus také slouží velmi kvalitní dokumentace, která slouží jako zdroj potřebných informací pro práci s touto knihovnou.

Nad touto knihovnou pak stojí vysokoúrovňové API[1] Keras pro práci s neuronovými sítěmi, jež bylo napsáno v Pythonu a jehož cílem je usnadnit vytváření a trénování modelů pro hluboké učení. Kromě toho je také vhodné pro samotný vývoj a rychlé prototypování. Keras nabízí předdefinované vrstvy a funkce, které pouze stačí správně poskládat a nakonfigurovat. Není proto náhoda, že Keras je využívaný mezi organizacemi jako CERN nebo NASA.

2.1.4 Další knihovny

Dále pro zpracování a transformaci dat jsme využili populární knihovny NumPy, SciPy, Pandas, Scikit-learn a OpenCV. Pro grafické vizualizace jsme využili především knihovny Matplotlib a Seaborn.

2.2 Datová sada

Vstupní data použitá pro tuto bakalářskou práci byly poskytnuty vedoucím práce. Datová sada obsahuje celkově 843 skenů acetabula. Všechny skeny už byly převedeny ze STL[2] formátu do PNG[3] a jsou ve stupních šedi, tedy obsahují pouze jeden kanál, kde každý pixel nese informaci o intenzitě. Obrázky jsou čtvercové a mají rozměr 521 × 521 pixelů, přičemž každý pixel má 16 bitovou hloubku. Na obrázku 2.1 vidíme příklad acetabula, jež patřilo 61leté ženě. Můžeme si také všimnout jakési vady na levém obrázku v levém horním rohu, jedná se o chyby, které vznikly během procesu skenování a se kterými se vypořádáme během procesu předzpracování dat.

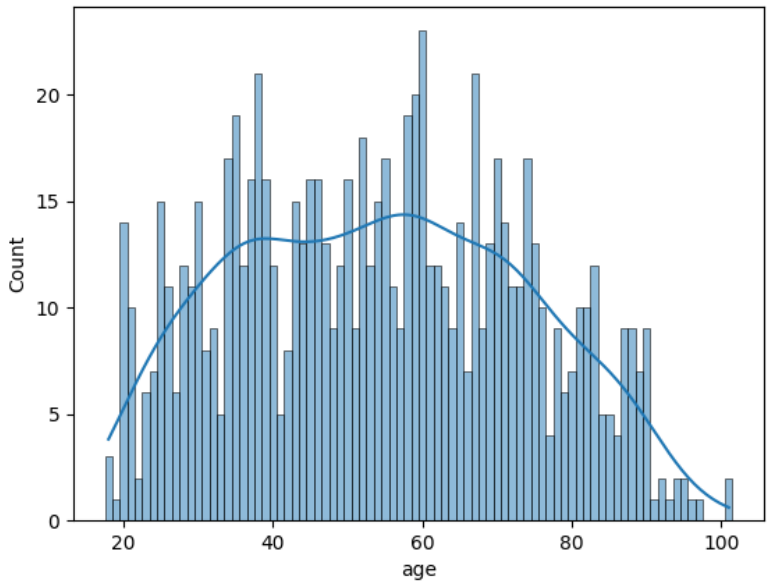
Obrázek 2.1 Příklad skenů acetabula, vlevo vidíme obyčejný sken, zatímco vpravo sken stejného acetabula po převodu do polárních souřadnic.

Dále všechny obrázky obsahují anotaci, která udává věk daného člověka při dožití. Na obrázku 2.2 je graf znázorňující rozdělení věku mezi jednotlivými vzorky dat. Minimální věk v naší datové sadě je 18 let kdežto maximální 101 let, průměrný věk je potom přibližně 54,25 let a standardní odchylka činí asi 19,28 let.

2.3 Přístup k řešení problému

Na problém odhadu věku můžeme nahlížet jako na regresní i klasifikační problém. Jako intuitivnější se nabízí regresní přístup, jelikož věk je spojitá proměnná. Problém ovšem může být s přesností, protože jak jsme naznačili již v úvodu, při odhadu dožitého věku z kosterních pozůstatků odhadujeme věk na základě nějakých degenerativních změn, které se na dané kosti

[1] Z angl. Application Programming Interface.
[2] Z angl. Standard Triangle Language, formát pro práci s 3D objekty.
[3] Z angl. Portable Network Graphics.



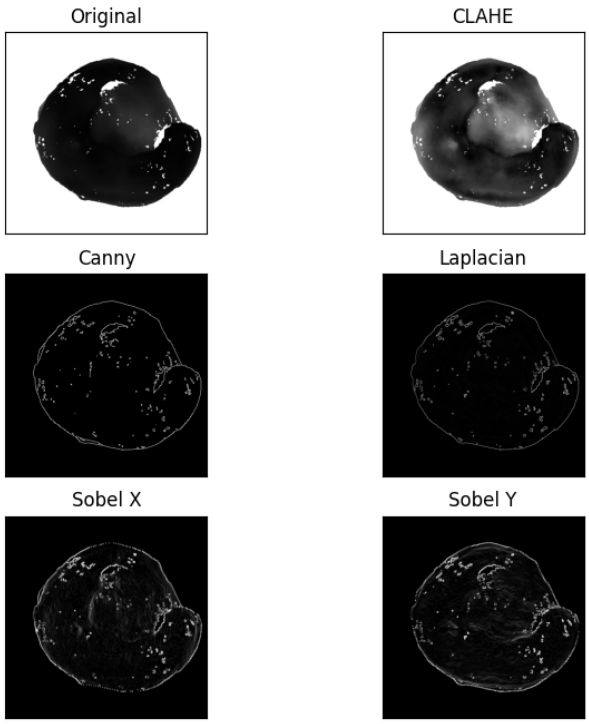
Obrázek 2.2 Rozdělení věku.

typicky vyskytují s rostoucím věkem. Tyto změny se ovšem v populaci nevyskytují rovnoměrně, vždy jsou ovlivněny širokou škálou faktorů od životního stylu jedince až po jeho genetické predispozice. Při sestavování biologického profilu člověka navíc často není nutné znát věk zcela přesně, ale spokojíme se i s intervalem, který nám poskytne další informaci pro vytvoření celkového kontextu o daném člověku. Proto se na problém odhadu věku často nahlíží jako na klasifikační, kdy jsou vytvořeny věkové intervaly, mezi které chceme daného člověka zařadit. Ačkoliv takto můžeme docílit vyšší přesnosti modelu, vždy to bude na úkor rozsahu jednotlivých intervalů. Jinými slovy, budeme-li klasifikovat například mezi tři věkové intervaly, přesnost modelu bude pravděpodobně vyšší ovšem za cenu toho, že daný interval bude mít poměrně široký rozsah, který nám nemusí stačit pro vytvoření kvalitního biologického profilu. Další problém v případě klasifikace navíc je samotný počet těchto věkových intervalů a jejich okraje, tedy to, kde dané intervaly začínají a kde končí. Práce Dudzika [36] navíc naznačuje, že přesný a spolehlivý odhad dožitého věku je možný pouze při klasfikaci do tří velmi širokých věkových skupin (do 30 let, 30 až 60 let a nad 60 let). Proto jsme se rozhodli v této práci věnovat úsilí pouze regresnímu modelu.

2.3.1 Předzpracování dat

Cílem předzpracování dat bylo co nejlépe připravit vstupní obrázky pro následné učení CNN. Ze všeho nejdříve jsme přistoupili k oříznutí obrázků. Jak lze vidět na obrázku 2.1 vlevo, acetablum je ze všech stran obklopeno bílými pixely, jež žádnou informaci nenesou. Proto jsme vytvořili funkci, která obrázky ořízla tak, aby všechny měly stejný výsledný rozměr a aby měly co nejméně prázdných řádků a sloupců. Kromě toho bylo nutné odstranit první řádek obrázku, který mohl obsahovat určitou deformaci vzniklou během skenování acetabula. Z původního rozměru 521 × 521 pixelů se nám podařilo obrázky zmenšit na 467 × 467 pixelů, což může snížit výpočetní náročnost učení modelu. Jelikož měl každý pixel 16 bitovou hloubku, použili jsme normalizaci hodnot, abychom docílili 8 bitové hloubky pixelů. Hodnoty pixelů tedy nově byly v rozsahu 0 až 255. 8 bitová hloubka se totiž hodí pro práci s knihovnou OpenCV, kterou jsme využili

pro další předzpracování. Dále jsme na takto ořízlé obrázky aplikovali CLAHE. Na závěr jsme postupně na takto upravené obrázky aplikovali jednotlivé detektory hran, konkrétně Cannyho hranový detektor, Laplacianův Sobelův a upravené obrázky uložili. Celkově jsme takto získali pět samostatných datových sad, které byly následně využity pro trénink modelů. Analogicky předzpracujeme i snímky acetabula, které byly dříve převedeny do polární soustavy souřadnic. Dohromady tedy máme celkem 10 datových sad, každou po 843 snímcích acetabula.

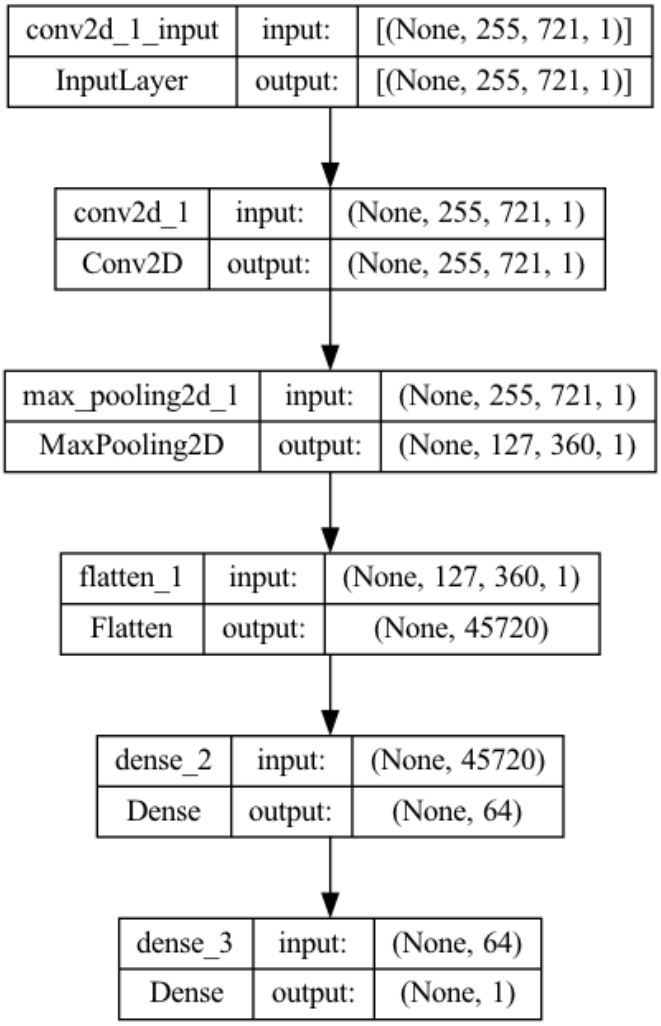


Obrázek 2.3 Předzpracované obrázky acetabula.

2.3.2 Regresní model

Ze všeho nejdříve jsme při konstrukci regresní CNN řešili její architekturu. Bohužel se rychle ukázalo, že s hloubkou dané CNN klesá její schopnost se přizpůsobit naší datové sadě a nedocházelo ke konvergenci. Jako poměrně slibná se ovšem ukázala velmi jednoduchá architektura s jedinou konvoluční vrstvou. V další práci jsme tedy pracovali pouze s touto architekturou, kterou si můžete prohlédnout na obrázku 2.4. Tato architektura není konečná, jelikož jsme dále přidávali například Dropout vrstvy, nicméně tvoří základní stavební blok, který jsme dále rozšiřovali. Vidíme, že tato CNN je přizpůsobena obrázkům acetabula v polární soustavě souřadnic, jelikož ve vstupní vrstvě očekává obrázek o rozměrech 255×721 pixelů.

Pro generování dávek dat jsme využili příhodnou třídu *ImageDataGenerator* z knihovny Keras, která data umí vhodně předzpracovat a navíc poskytuje rozhraní pro augmentaci dat v reálném čase. My jsme se rozhodli využít augmentaci pro zvýšení počtu trénovacích dat, konkrétně jsme využili horizontální a vertikální posun obrázku, otočení obrázku (maximálně o 10



Obrázek 2.4 Architektura naší regresní CNN.

stupňů), přiblížení a oddálení obrázku a také přidání Gaussovského šumu (třída *ImageDataGenerator* sama o sobě přidat šum neumí, ale lze tento způsob augmentace použít přidáním vlastní dodatečné funkce). Záměrně jsme nevyužili možnost augmentace pomocí překlopení obrázku, jelikož snímek acetabula by vždy měl v rámci předzpracování být správně transformován tak, aby kloub byl natočen konzistentně pouze jedním směrem a překlopení zde díky tomu nedává smysl. Tato třída také u všech obrázků po aplikaci všech transformací normalizuje hodnoty jejich pixelů mezi 0 a 1.

Dále jsme vytvořili vlastní *DataFrame* pomocí knihovny Pandas, který měl pouze dva sloupce a sice název obrázku a příslušnou anotaci s věkem, jež odpovídá stáří daného acetabula. Na obrázku 2.5 vidíme náhled tohoto *DataFramu*, který odpovídá výběru tréninkových dat. Lze si všimnout, že sloupec úplně vlevo, který představuje index, není seřazený, což je dáno náhodným výběrem během dělení datové sady. *DataFrame* jsme zkontruovali proto, abychom mohli využít metodu třídy *ImageDataGenerator flow_from_dataframe*, která vrací samotný generátor dat, jež následně předáme metodě zajišťující učení CNN.

	img	age
56	37Co2_ac_dex_M49.png	49
834	2Cr_ac_dex_M54.png	54
109	81Cr_ac_dex_M71.png	71
782	108Co2_ac_sin_M68.png	68
381	ECH_10S_ac_sin_M73.png	73

Obrázek 2.5 Náhled *DataFramu*.

2.3.3 Experimenty

Po získání architektury CNN jsme vyzkoušeli tuto síť natrénovat na jednotlivých datových sadách a prozkoumali jsme dosažené výsledky. CNN byla trénována jak pomocí křížové validace, tak i pomocí klasického rozdělení dat na tři množiny (trénovací, validační a testovací). Hyperparametr rychlosti učení jsme nastavili na 1×10^{-4}. Při této hodnotě se CNN chová stabilně, stále dochází během učení ke konvergenci a rychlost učení je přijatelná. Je-li hodnota vyšší tak nám sice stačí méně trénovacích epoch, ale CNN se začíná chovat nestabilně a ke konvergenci může docházet příliš rychle, což má za následek přeučení modelu.

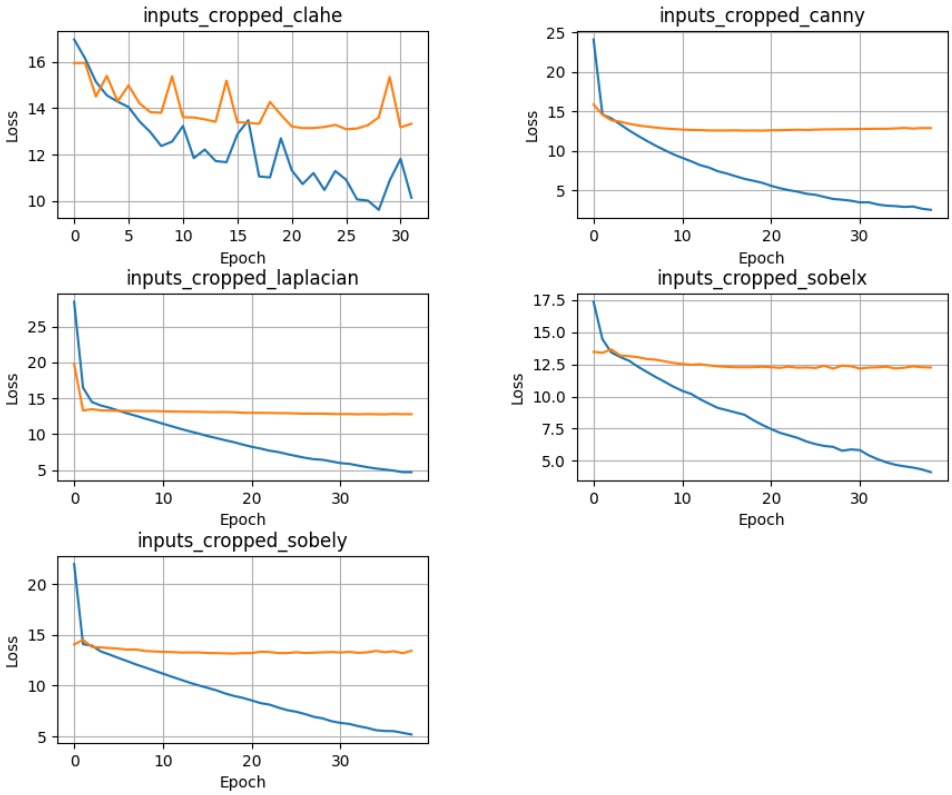
V tabulce 2.1 vidíme srovnání přesnosti modelu na jednotlivých datových sadách. Sloupec *Dataset* označuje název složky s danou datovou sadou a navíc název určuje jakou transformací data prošly. Sloupce *Train Loss*, *Val Loss* a *Test Loss* označují hodnotu ztrátové funkce na dané množině dat (trénovací, validační a testovací). Jako ztrátová funkce byla použita MAE. Z dat vidíme, že model poměrně dobře funguje na datech, které prošly pouze ekvalizací histogramu pomocí CLAHE a které byly tranformovány do polární soustavy souřadnic. Obecně lze říci, že na datech v polární soustavě souřadnic si model vede o něco lépe, než na datech v klasické kartézské soustavě. Zároveň u snímků, na které byl aplikován nějaký detektor hran dochází k výraznějšímu přeučení, nejvíce se to projevuje u Cannyho hranového detektoru. Výjimku tvoří model natrénovaný na datové sade *inputs_cropped_sobelx*, který dosáhl celkově nejlepšího výsledku na testovací množině dat, když jako jediný docílil MAE pod hodnotu 12.

Tabulka 2.1 Přehled dosažené přesnosti modelu na jednotlivých datových sadách

Dataset	Train Loss	Val Loss	Test Loss
inputs_cropped_clahe	10,14	13,33	13,39
inputs_cropped_canny	2,53	12,89	14,94
inputs_cropped_laplacian	4,71	12,79	13,66
inputs_cropped_sobelx	4,11	12,26	13,27
inputs_cropped_sobely	5,19	13,42	11,91
inputs_transformed_cropped_clahe	9,92	11,59	12,70
inputs_transformed_cropped_canny	2,57	13,34	12,87
inputs_transformed_cropped_laplacian	5,73	13,72	14,23
inputs_transformed_cropped_sobelx	4,65	13,47	13,38
inputs_transformed_cropped_sobely	3,61	12,30	13,49

Na obrázku 2.6 vidíme graf srovnávající průběh trénovací a validační ztrátové funkce během tréninku CNN na jednotlivých datových sadách. Z grafu je dobře vidět, že na datech, kde proběhla detekce hran dochází k výraznějšímu přeučení a zároveň je učení stabilnější, jelikož ztrátová funkce má velice nízkou volatilitu.

Dalším experimentem bylo vytvoření tzv. Ensemble modelu. Tento přístup je založený na tom, že natrénujeme několik modelů a jejich výsledné predikce zkombinujeme a spočítáme z nich průměr. Průměr těchto predikcí pak využijeme pro získání celkové přesnosti modelu oproti testo-



Obrázek 2.6 Graf ukazuje srovnání průběhu trénovací a validační ztátové funkce během trénování modelu na jednotlivých datových sadách v kartézské soustavě souřadnic.

vacím datům. V našem případě tedy vytvoříme dva Ensemble modely. První model bude postavený na predikcích modelů natrénovaných na datových sadách v kartézské soustavě souřadnic a ten druhý na predikcích modelů natrénovaných na datových sadách v polární soustavě souřadnic. Srovnání výsledků vidíme v tabule 2.2. Zde si vede poměrně výrazně lépe druhý model, nicméně dosažené výsledky jsou srovnatelné s výsledky samostatných modelů a nejedná se tak o výrazné zlepšení.

Tabulka 2.2 Přehled dosažené přesnosti Ensemble modelů.

Model	Test Loss
Datové sady v kartézské soustavě souřadnic	15,22
Datové sady v polární soustavě souřadnic	13,69

2.3.4 Hyperparametry

Nedílnou součástí vytváření ML modelů je také ladění hyperparametrů (angl. Hyperparameter Tuning). Jedním z hyperparametrů je i parametr rychlosti učení, který jsme vyřešili již v předchozí sekci. Dalším parametrem je aktivační funkce, kde jsme nakonec ponechali funkci

ReLU. Pro inicializaci vah vrstev CNN jsme využili Glorotovu rovnoměrnou inicializaci a jako optimalizační algoritmus byl zvolen algoritmus Adam. Velikost dávky byla ponechána na 32 snímcích, jelikož příliš neovlivňovala přesnost modelů.

2.4 Diskuse

Nejlepší námi navržené modely dosáhly přesnosti MAE 11,91 a 13,69 let. První model byl trénován na datové sadě, která byla předzpracována pomocí aplikace CLAHE a Sobelova operátoru. Učení probíhalo po dobu 40 epoch a výsledky byly změřeny na testovací množině dat. Druhý model byl vytvořen pomocí Ensemble techniky z jednotlivých CNN natrénovaných na všech datových sadách po dobu 20 epoch, které byly v rámci předzpracování převedeny do polární soustavy souřadnic. Z predikcí jednotlivých CNN byl následně spočten průměr a ten byl použit pro vyhodnocení přesnosti výsledného modelu. Další experimenty a výzkum by tyto výsledky mohly ještě zpřesnit.

Ačkoliv existují metody pro odhad dožitého věku s výrazně vyšší přesnosti, mohou tyto výsledky sloužit jako základ pro další výzkum. Největší prostor pro zlepšení vidíme ve využití komplexnějších metod pro ladění hyperparametrů. Pomoci by také mohlo rozšíření datové sady o více vzorků, díky čemuž bychom mohli vytvářet větší a hlubší CNN. Další prostor pro zlepšení tkví v použití jiné architektury CNN, např. provést experimenty s již předtrénovanými neuronovými sítěmi, které knihovna Keras nabízí.

Závěr

Hlavním cílem této práce bylo prozkoumat schopnosti konvolučních neuronových sítí pro odhad dožitého věku ze snímků acetabula a pokusit se takovou konvoluční neuronovou síť implementovat a dosáhnout co nejlepších výsledků.

V první části této práce jsme představili základní teorii potřebnou k pochopení a vytváření umělých neuronových sítí a měření jejich přesnosti. Dále jsme zmínili časté komplikace spojené s učením neuronových sítí a nastínili jsme metody pro jejich eliminaci. Zároveň byly předvedeny metody předzpracování dat pro konvoluční neuronové sítě.

V praktické části jsme provedli předzpracování vstupních dat a vytvořili jsme tak vstupní datové sady pro naše modely. Dále jsme implementovali vlastní řešení pro odhad dožitého věku a provedli celou řadu experimentů s cílem zlepšit přesnost našeho modelu. Nakonec jsme vybrali dva modely s nejvyšší přesností. Výsledky těchto modelů jsou spíše průměrné, nicméně mohou sloužit jako základ pro další výzkum a s pomocí dalších experimentů může být jejich přesnost dále zvýšena. Existuje také prostor pro další úpravu architektury modelu.

Bibliografie

1. UBELAKER, Douglas H; KHOSROWSHAHI, Haley. Estimation of age in forensic anthropology: historical perspective and recent methodological advances. *Forensic sciences research*. 2019, roč. 4, č. 1, s. 1–9. Dostupné z DOI: 10.1080/20961790.2018.1549711.
2. FRANKLIN, Daniel. Forensic age estimation in human skeletal remains: current concepts and future directions. *Legal Medicine*. 2010, roč. 12, č. 1, s. 1–7. Dostupné z DOI: 10.1016/j.legalmed.2009.09.001.
3. KOTĚROVÁ, Anežka; NAVEGA, David; ŠTEPANOVSK, Michal; BUK, Zdeněk; BRŽEK, Jaroslav; CUNHA, Eugénia. Age estimation of adult human remains from hip bones using advanced methods. *Forensic science international*. 2018, roč. 287, s. 163–175. Dostupné z DOI: 10.1016/j.forsciint.2018.03.047.
4. BADILLO, Solveig; BANFAI, Balazs; BIRZELE, Fabian; DAVYDOV, Iakov I; HUTCHINSON, Lucy; KAM-THONG, Tony; SIEBOURG-POLSTER, Juliane; STEIERT, Bernhard; ZHANG, Jitao David. An introduction to machine learning. *Clinical pharmacology & therapeutics*. 2020, roč. 107, č. 4, s. 871–885. Dostupné z DOI: 10.1002/cpt.1796.
5. MAHESH, Batta. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*. 2020, roč. 9, s. 381–386. ISSN 2319-7064. Dostupné také z: https://www.researchgate.net/profile/Batta-Mahesh/publication/344717762_Machine_Learning_Algorithms_-A_Review/links/5f8b2365299bf1b53e2d243a/Machine-Learning-Algorithms-A-Review.pdf?eid=5082902844932096.
6. LE; HO; LEE; JUNG. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. *Water*. 2019, roč. 11, č. 7, s. 1387. ISSN 2073-4441. Dostupné z DOI: 10.3390/w11071387.
7. ABRAHAM, Ajith. Artificial neural networks. *Handbook of measuring system design*. 2005. Dostupné také z: http://wsc10.softcomputing.net/ann_chapter.pdf.
8. DEEPAI. What is Perceptron? In: [online]. 2022 [cit. 2022-12-29]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/perceptron.
9. ABBAS, Osman. Neural Networks in Business Forecasting. *International Journal of Computer (IJC)*. 2005. ISSN 2307-4523. Dostupné také z: https://www.researchgate.net/publication/295907353_Neural_Networks_in_Business_Forecasting.
10. SHARMA, Sagar; SHARMA, Simone; ATHAIYA, Anidhya. Activation functions in neural networks. *towards data science*. 2017, roč. 6, č. 12, s. 310–316. Dostupné také z: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.
11. BROWNLEE, Jason. Loss and Loss Functions for Training Deep Learning Neural Networks. In: [online]. 2019 [cit. 2022-12-30]. Dostupné z: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

12. YATHISH, Vishal. Loss Functions and Their Use In Neural Networks. In: [online]. 2022 [cit. 2022-12-30]. Dostupné z: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9.
13. RUMELHART, David E; DURBIN, Richard; GOLDEN, Richard; CHAUVIN, Yves. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*. 1995, s. 1–34.
14. KOECH, Kiprono Elijah. How Does Back-Propagation Work in Neural Networks? In: [online]. 2022 [cit. 2022-12-30]. Dostupné z: https://towardsdatascience.com/how-does-back-propagation-work-in-neural-networks-with-worked-example-bc59dfb97f48.
15. LECUN, Yann; BENGIO, Yoshua et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*. 1995, roč. 3361, č. 10, s. 1995. Dostupné také z: http://www.iro.umontreal.ca/~lisa/pointeurs/handbook-convo.pdf.
16. O'SHEA, Keiron; NASH, Ryan. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*. 2015. Dostupné také z: https://arxiv.org/pdf/1511.08458.pdf.
17. SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. In: [online]. 2018 [cit. 2022-12-30]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
18. ZHANG, Aston; LIPTON, Zachary C.; LI, Mu; SMOLA, Alexander J. Dive into Deep Learning. *arXiv preprint arXiv:2106.11342*. 2021. Dostupné také z: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html.
19. PARMAR, Ravindra. Demystifying Convolutional Neural Networks. In: [online]. 2018 [cit. 2022-12-30]. Dostupné z: https://towardsdatascience.com/demystifying-convolutional-neural-networks-384785791596.
20. KHOSLA, Savya. CNN — Introduction to Pooling Layer. In: [online]. 2022 [cit. 2023-01-02]. Dostupné z: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/.
21. OLEXSYS. Data/Parameter Ratio, Have you got enough data for your parameters? In: [online]. 2022 [cit. 2023-01-02]. Dostupné z: https://www.olexsys.org/olex2/docs/reference/diagnostics/data-parameter-ratio/.
22. WANG, Chi-Feng. The Vanishing Gradient Problem. In: [online]. 2019 [cit. 2022-12-30]. Dostupné z: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.
23. BROWNLEE, Jason. Weight Initialization for Deep Learning Neural Networks. In: [online]. 2021 [cit. 2022-12-30]. Dostupné z: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/.
24. JAYAWARDANA, Rahul; BANDARANAYAKE, Thusitha. ANALYSIS OF OPTIMIZING NEURAL NETWORKS AND ARTIFICIAL INTELLIGENT MODELS FOR GUIDANCE, CONTROL, AND NAVIGATION SYSTEMS. *International Research Journal Technology and Science*. 2021. ISSN 2582-5208. Dostupné také z: https://www.researchgate.net/figure/Fig-7-Parameters-of-the-gradient-descent-variation-optimization-Gradient-Descent_fig6_350567223.
25. SHUBHAM, Jain. An Overview of Regularization Techniques in Deep Learning. In: [online]. 2018 [cit. 2022-12-30]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.
26. KHALIFA, Amine Ben; FRIGUI, Hichem. Multiple Instance Fuzzy Inference Neural Networks. 2016. Dostupné také z: https://www.researchgate.net/publication/309206911_Multiple_Instance_Fuzzy_Inference_Neural_Networks.

27. SCHRAUDOLPH, Nic. Preventing overfitting, Early stopping. In: [online]. 2022 [cit. 2023-01-01]. Dostupné z: https://cnl.salk.edu/~schraudo/teach/NNcourse/overfitting.html.

28. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, roč. 12, s. 2825–2830. Dostupné také z: https://scikit-learn.org/stable/modules/cross_validation.html.

29. BROWNLEE, Jason. How to Configure Image Data Augmentation in Keras. In: [online]. 2019 [cit. 2022-12-30]. Dostupné z: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/.

30. SONG, Chunlin; SUDIRMAN, Sud; MERABTI, Madjid. A Spatial and Frequency Domain Analysis of the Effect of Removal Attacks on Digital Image Watermarks. 2010. Dostupné také z: https://www.researchgate.net/figure/Effect-of-adding-Gaussian-noise-to-an-image-a-Original-watermarked-image-b-Gaussian_fig2_229008840.

31. OPENCV. Histograms - 2: Histogram Equalization. In: [online]. 2022 [cit. 2023-01-01]. Dostupné z: https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html.

32. WIKIPEDIA. Adaptive histogram equalization. In: [online]. 2022 [cit. 2023-01-01]. Dostupné z: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization.

33. WALY, Mohamed. Detection of Retinal Blood Vessels by using Gabor filter with Entropic threshold. *Majmaah Journal of Health Sciences*. 2016, roč. 4, s. 36–55. Dostupné také z: https://www.researchgate.net/figure/A-Green-Channel-B-after-histogram-equalization-C-after-CLAHE_fig12_320402762.

34. BOGOTOBOGO. Canny Edge Detection. In: [online]. 2022 [cit. 2022-12-13]. Dostupné z: https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Canny_Edge_Detection.php.

35. SETCHELL, Mark. Converting an image from Cartesian to Polar - Limb Darkening. In: [online]. 2018 [cit. 2022-12-13]. Dostupné z: https://stackoverflow.com/questions/51675940/converting-an-image-from-cartesian-to-polar-limb-darkening.

36. DUDZIK, Beatrix; LANGLEY, Natalie R. Estimating age from the pubic symphysis: A new component-based system. *Forensic science international*. 2015, roč. 257, s. 98–105. Dostupné také z: https://www.sciencedirect.com/science/article/pii/S0379073815003254.

Obsah přiloženého média