Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Strojové učení s využitím metody transfer learning

Diplomová práce

Autor: Bc. Jan Štol
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Karel Mls Ph.D.

Hradec Králové listopad 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Hradci Králové dne 14.11.2019 Bc. Jan Štol

Poděkování:
Děkuji vedoucímu diplomové práce Ing. Karlu Mlsovi, Ph.D. za metodické vedení práce a odbornou konzultaci při její tvorbě.

Anotace

Tato práce se zabývá tématy strojového učení, umělých neuronových sítí a metodou tzv. přeneseného učení (transfer learning). Jsou zde uvedeny základní modely strojového učení, popsány vybrané architektury umělých neuronových sítí, aktivační funkce nebo také použití přeneseného učení. V praktické části této práce byla využita metoda přeneseného učení a augmentace dat pro natrénování modelu umělé neuronové sítě, který umí klasifikovat české mince. V rámci této práce byla vytvořena malá datová sada obrázků českých mincí, na které byl model trénován.

Annotation

Title: Machine learning with transfer learning method

This diploma thesis focuses on machine learning, artificial neural networks and transfer learning topics. There are mentioned basic machine learning models, selected architectures of artificial neural networks, activation functions or use of transfer learning. In the practical part of this thesis, transfer learning and data augmentation were used to train a model for Czech coins classification. A small dataset of Czech coin images was created which the model was trained on.

Obsah

Seznam obrázků

Seznam grafů

Seznam tabulek

Seznam ukázek kódu

Seznam použitých zkratek a pojmů

AE Autoencoder (autoenkodér).
AI Artificial intelligence (umělá inteligence).
ANN Artificial neural networks (umělé neuronové sítě).
API Application Programming Interface (aplikační programovací rozhraní).

CNN Convolutional neural networks (konvoluční neuronové sítě).
CPU Central processing unit (procesor).

DFFNN Deep feedforward neural networks (hluboké dopředné neuronové sítě).
DL Deep learning (hluboké učení).

FFNN Feedforward neural networks (dopředné neuronové sítě.

GAN Generative adversarial networks (generativní kompetitivní neuronové sítě).
GPU Graphics processing unit (grafická karta).

LSTM Long short-term memory neural networks (sítě s dlouhou krátkodobou pamětí).

ML Machine learning (strojové učení).

NLP Natural language processing (zpracování přirozeného jazyka).

RNN Recurrent neural networks (rekurentní neuronové sítě).

TL Transfer learning (přenesené učení).

1 Úvod

Témata jako strojové učení nebo umělá inteligence se stávají stále více populární. To naznačují například trendy ve vyhledávání [1], ale i velké technologické společnosti (jako je např. Google) přisuzují strojovému učení velký význam [2] a využívají ho ve svých službách.

Strojové učení se stává dostupnější díky zdarma dostupným nástrojům, kurzům jako jsou například zdarma dostupné kurzy od Fast.ai[1] a stále dostupnějšímu výpočetnímu výkonu. Strojové učení se tak rozšiřuje do různých oborů a oblastní lidského života, přináší nové možnosti a dá se tedy očekávat, že poptávka po expertech nebo alespoň lidech se zkušenostmi v této oblasti se bude dále zvyšovat.

Tato práce popisuje možnosti strojové učení a ukazuje použití některých nástrojů a postupů v praxi na malé datové sadě.

Kapitola **2 Strojové učení** vysvětluje některé základní pojmy, uvádí základní typy a modely strojového učení, popisuje možnosti využití a uvádí některé základní nástroje.

Kapitola **3 Umělé neuronové sítě** se zaměřuje na jeden z nejpoužívanějších modelů strojového učení – umělé neuronové sítě. Popisuje neuronové sítě od základních prvků, uvádí stručnou historii, popisuje vybrané architektury umělých neuronových sítí, představuje některé aktivační funkce a vysvětluje princip učení neuronových sítí pomocí zpětného šíření chyby.

Kapitola **4 Transfer learning** se věnuje tzv. přenesenému učení. Tato kapitola uvádí formální definici přeneseného učení, možnosti použití přenosu učení a zmiňuje některé před-trénované modely.

Kapitola **5 Klasifikace českých mincí** se zaměřuje na aplikaci přenosu učení spolu s dalšími technikami v praxi. V této kapitole jsou popsány nástroje a základní postupy, které byly použity při tvorbě klasifikačního modelu českých mincí. Je zde popsána také datová sada, která byla vytvořena v rámci této práce. Výsledky nakonec shrnuje kapitola **6 Shrnutí výsledků**.

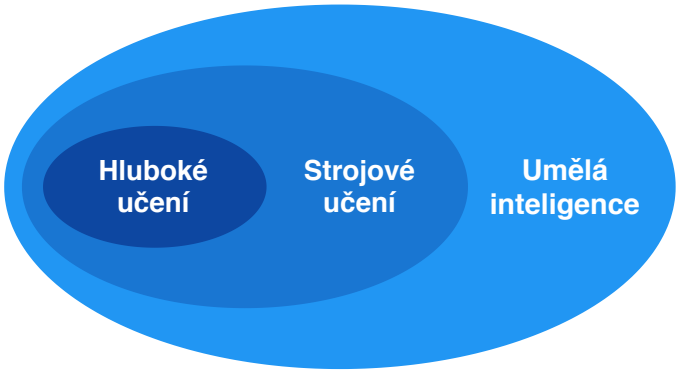
[1] `https://www.fast.ai`

2 Strojové učení

Tato kapitola popisuje některé pojmy související a někdy také zaměňované se strojovým učením, základní typy strojového učení, jeho využití, a některé vybrané modely.

2.1 Strojové učení, hluboké učení, umělá inteligence

S pojmem **strojové učení** souvisejí také pojmy jako **neuronové sítě**, **hluboké učení** nebo **umělá inteligence** a někdy se tyto pojmy také různě zaměňují. Vztah mezi strojovým učením, umělou inteligencí a hlubokým učením je znázorněn na následujícím obrázku.



Obrázek 2.1: Znázornění vztahu umělé inteligence, strojového učení a hlubokého učení (zdroj: podle [3, s. 2])

Jednotlivé pojmy jsou pak dále stručně popsány. Neuronové sítě jsou podrobněji rozebrány samostatně v kapitole 3 Umělé neuronové sítě.

2.1.1 Umělá inteligence

Umělá inteligence (artificial intelligence, zkratka **AI**) je široký obor informatiky, který je často chápán jako snaha vytvořit obecnou umělou inteligenci, která by byla co nejvíce podobná té lidské. Měla by tedy být schopná se například učit, racionálně uvažovat nebo umět řešit různé problémy.

Poprvé byl termín umělá inteligence oficiálně použit v roce 1956, když byl tento

obor zakládán. Použil ho jeden ze zakladatelů – John McCarthy. [4, s.17] McCarthy definuje umělou inteligenci následovně: „*Je to věda a technika tvorby inteligentních strojů, zejména inteligentních počítačových programů.*“[1] [5, s. 2] (překlad: autor)

2.1.2 Strojové učení

Strojové učení (machine learning, zkratka **ML**) je podmnožinou AI nebo také způsob, jak umělé inteligence dosahovat. Strojové učení lze jednoduše charakterizovat jako vytváření programů, které dokáží automaticky upravovat svůj vnitřní stav a chování na základě dat, která dostávají.

Konkrétnější popis uvedl Artur Samuel, který je považován za průkopníka v oblasti ML a AI [6]. Samuel se snažil naučit počítač hrát deskovou hru dáma (checkers) a právě na tomto problému popsal podstatu strojového učení. Uvedl, že během krátké doby se může počítač naučit hrát hru dáma lépe než člověk, který ho k tomu naprogramoval, a to pouze se znalostí základních pravidel a některých tahů. Tyto poznatky lze pak také využít i při řešení jiných úloh. [7]

Mitchell [8, s. 2] pak uvádí formální definici: „*Počítačový program se učí ze zkušenosti E s ohledem na nějakou třídu úloh T a měření výkonu P, jestliže se jeho výkon v úlohách v T, měřený pomocí P, zlepšuje se zkušeností E.*“[2] (překlad: autor)

Dále pokračuje s konkrétním příkladem, který uvádí právě na hře dáma. Počítačový program může zlepšit svůj výkon (měřený jako schopnost vyhrát hru dáma) pomocí zkušenosti získané hraní hry proti sobě samému. Neboli T, P a E z dříve uvedené definice jsou pro tento příklad následující [8, s. 2-3]:

- **T** je hraní hry dáma
- **P** je procento vyhraných her
- **E** je hraní jednotlivých her

2.1.3 Hluboké učení

Hluboké učení (deep learning, zkratka **DL**) se dá považovat za podmnožinu strojového učení. Název „hluboké učení“ vyplývá z využití tzv. hlubokých neuronových sítí (viz 3.3 Architektura neuronových sítí). Díky DL lze řešit složitější úlohy (např. rozpoznávání zvuku a obrazu, zpracování přirozeného jazyka) a také dosahovat lepších výsledků. Kvůli vyšší komplexnosti ale také vyžaduje vyšší výpočetní výkon.

[1] It is the science and engineering of making intelligent machines, especially intelligent computer programs.

[2] A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

2.2 Typy strojového učení

Algoritmy strojového učení se obecně dělí na tři základní typy:

- **Učení s učitelem** (supervised learning)
- **Učení bez učitele** (unsupervised learning)
- **Zpětnovazební učení** (reinforcement learning)

Někdy se také uvádí ještě čtvrtý typ — kombinace učení s učitelem a bez učitele (semi-supervised learning).

2.2.1 Učení s učitelem

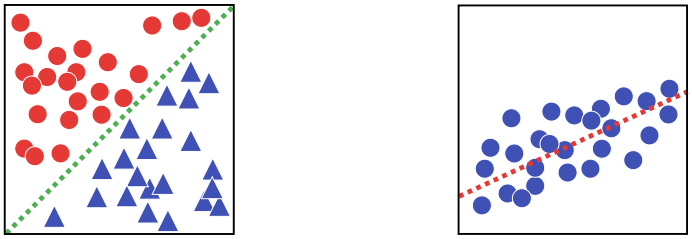
Při učení s učitelem je vytvářen prediktivní model, který dokáže na základě vstupních dat předpovědět výsledek. Pro učení takového modelu využívá učící algoritmus množiny tzv. **označených vzorů** (labeled dataset), kde každý prvek z této množiny obsahuje vstupní data a očekávaný výstup. V praxi je tato množina ještě dále rozdělena na dvě nebo tři samostatné množiny.

První z nich je **trénovací množina** (training dataset), která se využívá pro samotné učení modelu. Další je **validační množina** (validation dataset), která slouží pro kontrolu průběhu učení. Pokud by úspěšnost predikce pro trénovací množinu stoupala a pro ověřovací množinu klesala, znamenalo by to, že si model jen „pamatuje“ trénovací data a dochází k tzv. přeučení (viz 2.4 Přeučení a nedoučení). Nakonec **testovací množina** (test dataset, holdout dataset) slouží pro vyjádření přesnosti modelu po skončení jeho učení. Tyto množiny by měly být disjunktní, aby bylo možné korektně hodnotit průběh učení modelu a jeho výslednou přesnost.

Samotné učení probíhá tak, že model vytvoří na základě vstupů predikci, které je porovnána s očekávaným výsledkem. V závislosti na velikosti rozdílu (chyby) je pak model upraven tak, aby podával přesnější předpovědi, tzn. aby minimalizoval svoji chybu. Tento postup je opakován až dokud celková chyba neklesne pod určitou hranici.

Učení s učitelem řeší dva typy úloh [9]:

- **Klasifikace** (classification) — předpověď kategorie (třídy) na základě vstupních dat; výstupem je **kategorie**, např. červená barva.
- **Regrese** (regression) — odhad hodnoty na základě znalosti vstupu; výstupem je reálné **číslo**, které může vyjadřovat např. cenu.



(a) Klasifikace **(b)** Regrese

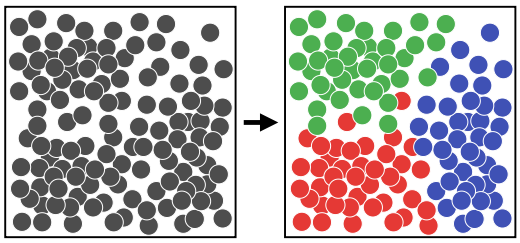
Obrázek 2.2: Grafické znázornění klasifikace a regrese (zdroj: podle [10])

2.2.2 Učení bez učitele

Učení bez učitele je založeno na využití **neoznačených** (surových) **dat**. Na rozdíl od učení s učitelem se tedy model musí naučit strukturu a vztahy mezi jednotlivými prvky bez další pomoci v podobě kategorizování nebo popisu těchto dat. Pomocí učení bez učitele lze tak dobře odhalovat skryté vzory a struktury v těchto surových datech. [11]

Učení bez učitele řeší typicky úlohy [9]:

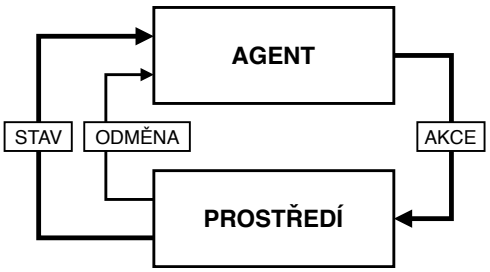
- **Shlukování** (clustering) — objevování seskupení v datech, rozřazování do skupin podle podobných vlastností; např. seskupení zákazníků podle nákupních zvyklostí
- **Asociace** (association) — odhalování skrytých vztahů a pravidel v datech; například lidé, kteří si kupují X mají tendenci si koupit také Y.



Obrázek 2.3: Grafické znázornění shlukování (zdroj: autor)

2.2.3 Zpětnovazební učení

Zpětnovazební učení využívá dvou základních prvků, kterými jsou **agent** a **prostředí** (environment).
Agent je umístěn do nějakého prostředí a pozoruje jeho **stav** (state), který může ovlivnit provedením **akce** (action). Vykonáním akce změní agent stav prostředí a dostane **odměnu** (reward). Na základě této zpětné vazby si agent aktualizuje své znalosti, které používá při provádění akci. Cílem agenta je maximalizovat svoji odměnu a tím dosahovat požadovaného výsledku.



Obrázek 2.4: Schéma zpětnovazebního učení (zdroj: převzato z [12])

2.3 Modely

Strojové učení zahrnuje vytvoření modelu, který je nutné nejprve učit (trénovat), aby poté mohl samostatně řešit určité úlohy. Několik základních kategorií modelů je uvedeno dále.

2.3.1 Umělé neuronové sítě

Umělé neuronové sítě jsou v současnosti nejpopulárnějším a nejpoužívanějším modelem v oblasti strojového učení. Neuronovým sítím je věnována samostatná kapitola **3 Umělé neuronové sítě**.

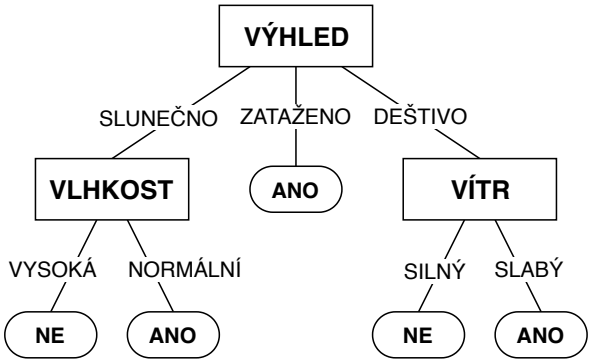
2.3.2 Rozhodovací stromy

Rozhodovací stromy (decision trees) jsou jedním z nejstarších a nejjednodušších modelů ML.

Tento model je reprezentován souvislým grafem, který neobsahuje kružnici neboli stromem. Základem stromu je kořenový vrchol, který je dále pomocí hran spojen s dalšími vrcholy s ty se mohou dále větvit stejným způsobem. Jednotlivé vrcholy stromu představují rozhodování na základě jedné určité vlastnosti, vystupující hrany odpovídají možnostem daného rozhodnutí. Listy stromu nakonec reprezentují konečné výsledky rozhodování.

Rozhodovací stromy třídí vstupní objekty na základě jejich vlastností, tzn. provádějí klasifikaci. Listy stromu reprezentují kategorie, do kterých může daný objekt patřit. [8, s. 52-53]

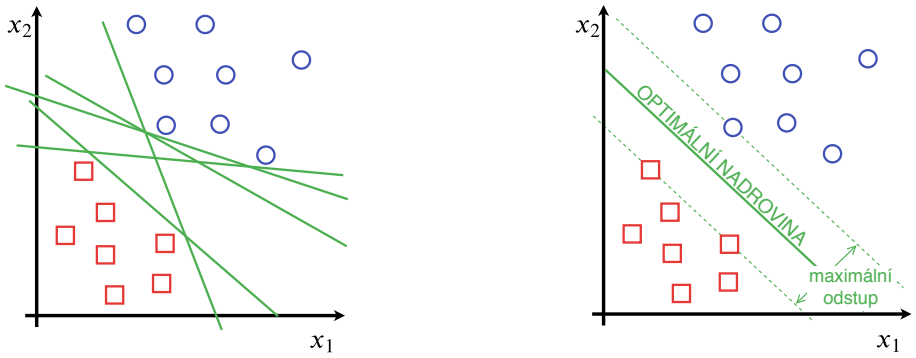
Na obrázku 2.5 je uveden příklad rozhodovacího stromu, který klasifikuje sobotní ráno jako vhodné nebo nevhodné pro hraní tenisu. Například pokud bude slunečno, horko, vysoká vlhkost a silný vítr, bude odpověď „NE“.[8, s. 53]



Obrázek 2.5: Rozhodovací strom pro koncept *hrát tenis* (zdroj: převzato z [8, s. 53], překlad: autor)

2.3.3 Podpůrné vektorové stroje (SVM)

Metoda podpůrných vektorů, známá jako **podpůrné vektorové stroje** (support vector machines, SVM), může být využita pro úlohy regrese i klasifikace, ale využívána je hlavně pro klasifikaci. Populární je zejména pro svoji přesnost a zároveň menší výpočetní zátěž. [13]



(a) Různé nadroviny rozdělující body **(b)** Optimální nadrovina rozdělující body

Obrázek 2.6: Grafické znázornění SVM, rozdělení bodů v prostoru nadrovinou (zdroj: podle [13])

Při klasifikaci je nutné najít **nadrovinu** (hyperplane) v N-dimenzionálním prostoru, která jasně rozděluje datové body. Takových nadrovin existuje mnoho a cílem SVM je najít takovou nadrovinu, která je nejvíce vzdálená od bodů obou tříd (znázorněno na obrázku 2.6). Tato vzdálenost se označuje jako **odstup** (margin) a utváří kolem nadroviny tzv. **hraniční pásmo**. Maximalizace odstupu zajistí vyšší jistotu při klasifikaci budoucích (nových) bodů. Nadrovina je popsána body, které leží v její blízkosti (na okraji hraničního pásma). Vektory těchto bodů se nazývají podpůrné vektory (support vectors) a pomocí nich lze maximalizovat odstup. [13]

2.3.4 Bayesovské sítě

Bayesovské sítě (bayesian networks) představují pravděpodobnostní model využívající bayesovské rozhodování a grafovou reprezentaci modelující pravděpodobnostní vztahy mezi jednotlivými jevy. Bayesovská síť je orientovaný acyklický graf ve kterém každý vrchol představuje jednu náhodnou veličinu a orientované hrany reprezentují pravděpodobnostní závislost mezi veličinami. To znamená, že pokud v grafu existuje hrana A, B spojující náhodné veličiny A a B, pak to značí podmíněnou pravděpodobnost $P(B|A)$. [14]

2.4 Přeučení a nedoučení

Při učení s učitelem (viz 2.2.1 Učení s učitelem) nastává obecně jeden ze tří stavů, které udávají sílu daného modelu a napovídají, jak bude naučený model přesný a použitelný:

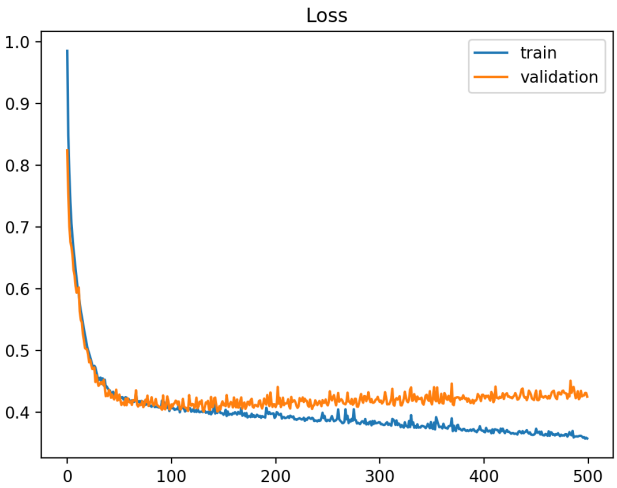
- **Ideální stav** (good fit)
- **Přeučení** (overfitting) – příliš silný model
- **Nedoučení** (underfitting) – příliš slabý model

Během procesu učení je v každém kroku hodnocena výkonnost modelu na trénovacích a validačních datech. Z hodnocení na trénovacích datech tak vzniká tréninková učící křivka, která vyjadřuje, jak dobře se model učí. Z hodnocení na validačních datech vzniká validační učící křivka popisující, jak dobře model generalizuje pro nová data. Tyto **učící křivky** (learning curves) ukazují, jak se vyvíjí výkon modelu v čase a pomocí toho lze relativně snadno diagnostikovat stav daného modelu. [15]

2.4.1 Přeučení

Stav tzv. **přeučení** (známý pod označením overfitting) nastává, pokud se model naučí detaily z množiny trénovacích dat tak, že to negativně ovlivňuje jeho výkon na odlišných datech. To znamená, že se model dobře naučí trénovací data, ale už nedokáže generalizovat pro jiná data. [16]

Přeučení je na grafu učících křivek snadno detekovatelné. Obecně graf ukazuje přeučení, pokud křivky trénovací i validační chyby klesají, ale od určitého bodu začne křivka validační chyby stoupat jako je to znázorněno na obrázku 2.7. [15]
Příčinou přeučení bývá buď příliš malá množina trénovacích dat nebo příliš velká složitost modelu (v případě neuronových sítí velký počet neuronů ve skrytých vrstvách sítě, velký počet spojení, a tedy i velký počet vah). Řešením je proto zvýšit počet trénovacích dat nebo zkusit snížit složitost modelu.



Obrázek 2.7: Příklad křivek učení, které ukazují přeučení modelu (zdroj: [15])

Existují různé metody, které pomáhají přeučení řešit jako například [17]:

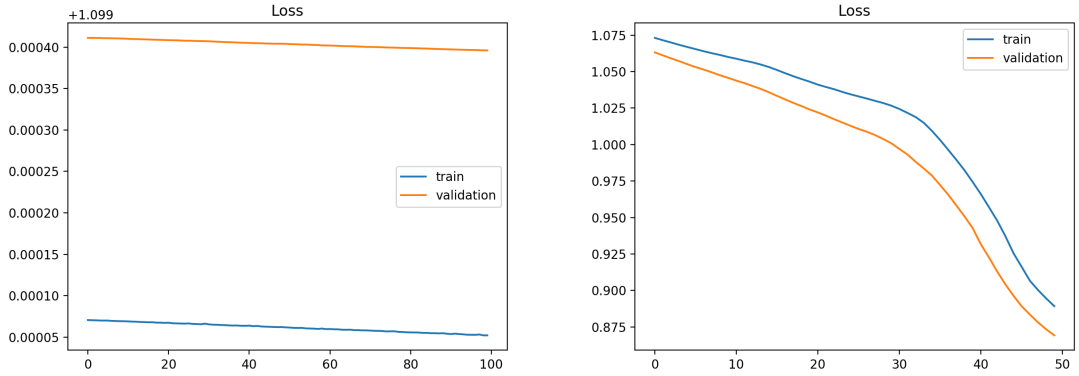
- dropout — náhodné „vypnutí“ (vynechání) některých neuronů (v modelech neuronových sítí)
- omezení vah — omezení hodnot vah tak, aby byly v určitém rozsahu (v modelech neuronových sítí)
- šum (noise) — přidání náhodného šumu, což prakticky znamená zvětšení trénovací množiny dat
- brzké zastavení (early stopping) -– zastavení učení dříve, než se začne výkon snižovat

2.4.2 Nedoučení

Nedoučení (označované jako underfitting) je problém, kdy se model nedokázal naučit z trénovacích dat a ani nedokáže generalizovat pro data nová. [16]

Graf učících křivek ukazuje nedoučení, pokud trénovací chyba zůstává přibližně stejná po celou dobu učení nebo se stále snižuje až dokud nedojde k ukončení učení. Pokud trénovací chyba zůstává přibližně stejná, značí to nedostatečnou kapacitu (složitost) modelu (obrázek 2.8a). Pokud se trénovací chyba stále snižuje, znamená to, že model je schopný se dále učit, ale učení bylo předčasně zastaveno (obrázek 2.8b). [15]

Nedoučení lze snadno řešit zvýšením složitosti modelu (v případě neuronových sítí přidáním více vrstev a/nebo přidáním neuronů do skrytých vrstev sítě), výběrem jiného modelu nebo jen dalším učením. [17]

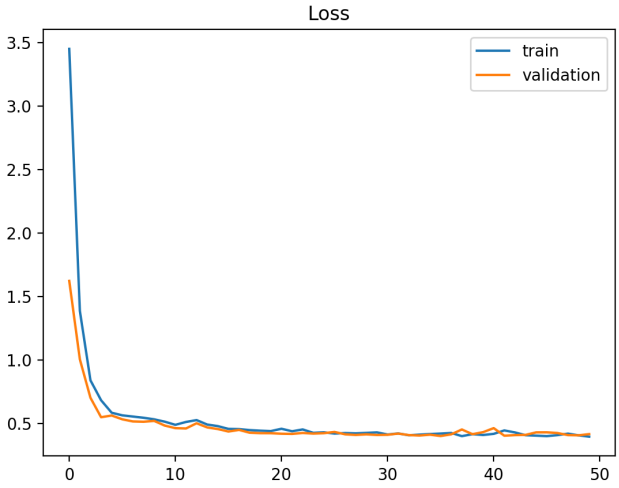


(a) Model nemá dostatečnou kapacitu **(b)** Model potřebuje další trénování

Obrázek 2.8: Příklady učících křivek, které ukazují nedoučení modelu (zdroj: [15])

2.4.3 Ideální stav

V ideálním případě by měla trénovací i validační chyba klesat do určitého stabilního bodu a měl by mezi nimi být pouze malý rozdíl jako to ukazuje obrázek 2.9 na následující straně. [15]



Obrázek 2.9: Příklad učících křivek, které značí ideální stav (good fit) (zdroj: [15])

2.5 Využití strojového učení

Strojové učení nachází využití v širokém spektru oborů od informatiky, přes bankovnictví, marketing nebo např. průmysl až po zdravotnictví.

Populární jsou dnes např. virtuální (hlasoví) **asistenti** jako je Google Assistant, Siri, Alexa nebo Cortana. Pomocí těchto asistentů lze provádět některé jednoduché

úkony hlasem — nastavit budík nebo upomínku, odeslat e-mail, ovládat domácnost (světla, vytápění, spotřebiče)[3], vyhledávat na internetu nebo spouštět jiné aplikace. V roce 2018 společnost Google představila technologii Google Duplex, která dokáže sama vyřizovat telefonní hovory — např. rezervaci místa v restauraci [18]. Ani jeden z těchto asistentů ale (zatím) nepodporuje český jazyk, což limituje jejich využití pro české uživatele.

Běžná jsou dnes také různá **doporučení obsahu** (zprávy, hudba, videa), produktů nebo reklamy. To znamená, že například podle dříve zhlédnutých videích je možné uživateli doporučit další videa, která by ho mohla zajímat. Dále je možné jmenovat **napovídání slov** při psaní textu, stále dokonalejší **automatické překládání mezi jazyky**, **filtrování nevyžádaných e-mailů** (spamu), **odhalování počítačových virů** (malware) atd.

Strojové učení využívají také **autonomní vozidla**, finanční instituce pro **odhalování podvodů**, **zdravotnictví** například při analýze snímků nebo odhalování různých onemocnění, nebo i firmy při hledání minerálů a zlata[4]. Možnosti využití ML jsou zkrátka velké a s rozvojem této oblasti lze očekávat jejich další rozšíření.

I přes tyto všechny možnosti je však nutné poznamenat, že strojové učení v nedokáže v současnosti řešit všechny problémy, není úplně bezchybné a také se s ním pojí některá úskalí jako je ochrana osobních údajů nebo etika.

Jedním případem kdy, strojové učení selhalo je, když autonomní vozidlo srazilo chodce. I přesto, že ho vozidlo pomocí senzorů detekovalo, tak nijak nezareagovalo a chodec tuto srážku nakonec nepřežil. [19]

Jiným příkladem je izraelský startup Faception, který tvrdí, že dokáže pomocí ML z fotografie předpovědět, jestli je daný člověk např. terorista, jaké má IQ apod. Toto oznámení vyvolalo určitou vlnu pochybností a znepokojení zejména kvůli tomu, jestli je vůbec možné takto zjistit osobnost člověka jen podle fotografie, jaký dopad to má na soukromí a také kvůli možným častým tzv. falešně pozitivním předpovědím. [20]

A nakonec je ještě třeba zmínit také tzv. **deepfake**[5], což je technika manipulace a falšování fotografií, videa, hlasu nebo zpráv (ve zpravodajství). Lze tak například falšovat projevy, zaměňovat lidem obličeje a měnit jejich výraz, syntetizovat hlas apod. Odhalování deepfake by mělo pomoci opět strojové učení, např. společnost Adobe pracuje na odhalování manipulace s obličejem na fotografiích[21].

[3] to pochopitelně vyžaduje hardware, který je pro toto přizpůsoben
[4] `https://goldspot.ca`
[5] složeno ze slov *deep learning* a *fake*

2.6 Software, knihovny a nástroje

Důležitou součástí ML je software, knihovny, frameworky a různé nástroje, které umožňují strojové učení realizovat.

2.6.1 Programovací jazyky a vývojová prostředí

V oblasti strojového učení jsou nejpopulárnějšími programovacími jazyky zejména **Python** a **C/C++**. Python se v ML často využívá jako vysokoúrovňové rozhraní pro tvorbu modelů. Samotné výpočty a kritické části (z hlediska výkonu) pak obstarává kód napsaný v C/C++. Lze se ale setkat i s mnoha jinými jazyky, příkladem je Javascript nebo Swift.

Pro vývoj je možné použít téměř jakýkoli editor či vývojové prostředí (např. PyCharm), dále je nutné mít nainstalované potřebné knihovny. Pro samotné trénování modelů lze využít procesor (CPU), ale pro velké a složitější modely (např. konvoluční neuronové sítě) je vhodnější využít dedikovanou grafickou kartu (GPU), která dokáže poskytnout vyšší výkon, a tedy urychlit proces trénování modelu. Momentálně jsou podporovány pouze grafické karty nVidia.

Oblíbeným prostředím nejen pro potřeby ML je Jupyter Notebok[6]. Na tomto prostředí je postavená služba **Colaboratory** (Colab)[7] od společnosti Google.

Jedná se o interaktivní prostředí, které je dostupné zdarma z internetového prohlížeče. Colab poskytuje virtuální počítač (runtime) s CPU, GPU (nVidia Tesla K80) nebo TPU (Tensor processing unit)[8], 12 GB RAM a místem na disku (liší se podle konfigurace CPU/GPU/TPU). Samotné notebooky (soubory .ipynb) jsou uloženy na úložišti Google Drive. Colaboratory nevyžaduje žádnou konfiguraci, takže je vhodné zejména pro začátečníky, pro testování a prototypování, ale lze použít i pro méně náročné projekty. V Colaboratory lze vykonávat různé příkazy jako v Linuxovém prostředí, tzn. v případě potřeby instalovat balíčky, programy a knihovny, kopírovat soubory apod. Soubory jde také stahovat i nahrávat zpět do Colaboratory a existují zde např. funkce pro práci s úložištěm Google Drive.

Nevýhodou je, že Colab není určen pro dlouho běžící úlohy (udávaný limit je 12 hodin). Při odpojení nebo restartu virtuálního počítače (např. při zavření okna prohlížeče) také dochází ke smazání všech dat uložených na disku tohoto počítače. Pro zachování je tak nutné požadovaná data například stáhnout nebo nahrát na úložiště Google Drive.

[6] `https://jupyter.org/`
[7] `https://colab.research.google.com`
[8] Čip navržený speciálně pro potřeby strojového učení společností Google

Pro potřeby vyššího výkonu a méně omezujících podmínek si lze virtuální počítače pronajmout. Příkladem je Google Cloud, Amazon Web Services (AWS) nebo Paperspace[9].

2.6.2 Knihovny a nástroje

Pro realizaci strojového učení, tvorbu a učení modelů existují různé knihovny a nástroje. Některé z těch nejpoužívanějších jsou uvedeny dále.

Keras

Keras je open-source knihovna, která umožňuje vytvářet umělé neuronové sítě a dále s nimi pracovat. Nabízí vysokoúrovňové rozhraní (API) díky kterému je vývoj a prototypování snadné a rychlé. Keras lze také používat spolu s jinými knihovnami, viz dále. [22].

TensorFlow

TensorFlow je pravděpodobně nejznámější knihovnou pro strojové učení. Původně byla tato knihovna vytvořena společností Google pro interní použití, později byl zdrojový kód zveřejněn a nyní se jedná o open-source projekt, do kterého může přispívat i komunita.

V září 2019 byla vydána stabilní verze TensorFlow 2.0, která řeší především snadnou použitelnost prostřednictvím integrace Keras API. [23]

TensorFlow zahrnuje podporu i pro internetové prohlížeče (TensorFlow.js), mobilní a IoT zařízení (TensorFlow Lite). Dále nabízí nástroje jako je TensorBoard pro vizualizaci a ladění nebo TensorFlow Hub, který nabízí snadný přistup k velkému množství existujících modelů.

PyTorch

PyTorch je open-source knihovna, která se zaměřuje na akcelerované výpočty s tenzory pomocí grafické karty a na hluboké neuronové sítě. [24] Jedná se o implementaci knihovny Torch v programovacím jazyce Python, odsud název PyTorch. [25]

[9] `https://www.paperspace.com`

3 Umělé neuronové sítě

Umělé neuronové sítě (artificial neural networks, zkratka **ANN** nebo také jen NN) je výpočetní model, který má široké uplatnění v celé oblasti AI a v posledních letech je to jeden z hlavních modelů využívaný ve strojovém učení.

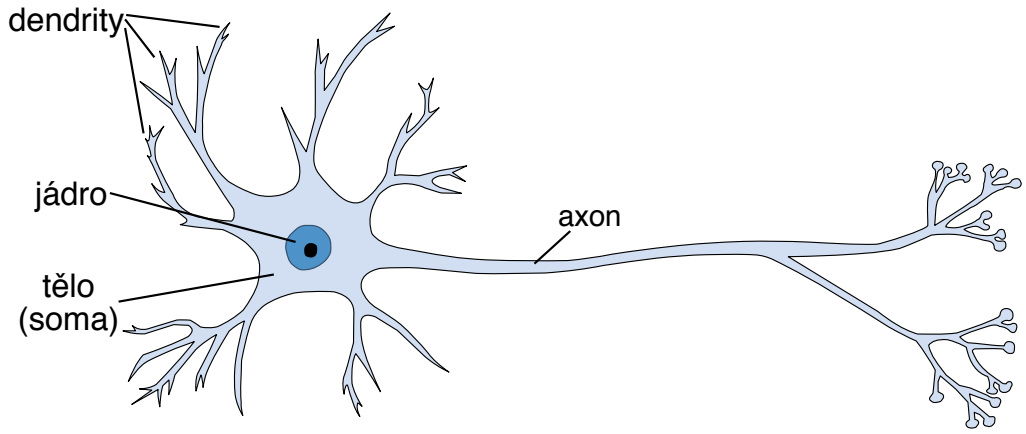
Někdy se ANN označují také jako učící algoritmy, které modelují vztah mezi vstupem a výstupem. [26]

3.1 Neuron

Základním prvkem neuronové sítě je nervová buňka neboli **neuron**.

3.1.1 Biologický neuron

Předlohou pro ANN je složitý systém biologických neuronových sítí jako je lidský mozek a umělé neurony jsou tedy také inspirovány biologickým neuronem, který je znázorněn na obrázku 3.1.



Obrázek 3.1: Znázornění biologického neuronu (zdroj: podle [27])

Biologické neurony jsou specializované buňky určené především pro přenos a zpracování vzruchů (signálů). Základem neuronu je jeho vlastní tělo označované také jako **soma**. Z těla pak vycházejí výběžky – tzv. **dendrity**, které slouží jako vstupní

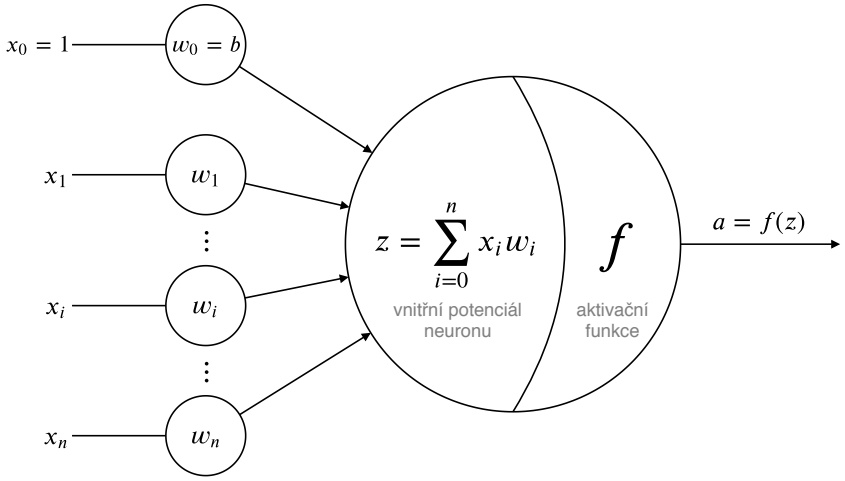
přenosové kanály; a jeden **axon**, který představuje výstupní přenosový kanál. Axon pak bývá dále rozvětvený, a díky tomu může být připojený k dendritům jiných neuronů. K samotnému přenosu signálu slouží mezineuronová rozhraní neboli **synapse**. Synapse lze rozdělit na ty, které mají **excitační** (umožňují šíření vzruchu) nebo **inhibiční** funkci (tlumí šíření vzruchu). [28]

3.1.2 Umělý neuron

Biologické neurony jsou dnes ve skutečnosti vnímány spíše jako malé neuronové sítě než jako prostá „jednotka“ v neuronové síti. [26] Umělé neurony (a obecně umělé neuronové sítě) tedy využívají jen základní a zjednodušené principy, které jsou ale pro dané potřeby dostatečné.

Obecný matematický model neuronu je znázorněn na obrázku 3.2. Neuron je tvořen n **vstupy** $(x_1, x_2, ..., x_n)$, ke kterým jsou přiřazeny **váhy** $(w_1, w_2, ..., w_n)$. Váhy určují významnost jednotlivých vstupů, vyšší hodnota váhy znamená vyšší význam daného vstupu.

Váhy mohou nabývat i záporných hodnot a tím napodobovat inhibiční funkci synapsí u biologického neuronu. [28]



Obrázek 3.2: Model umělého neuronu (zdroj: podle [27])

Kromě uvedených vstupů a vah má neuron ještě zvláštní vstup x_0 s konstantní hodnou 1 a váhu w_0 označovanou také jako b. Tato váha se nazývá **bias**. Někdy se místo biasu uvádí **threshold** (práh) θ. Vztah mezi biasem b a prahem θ lze jednoduše vyjádřit jako $b = -\theta$ (viz dále).

$$z = \sum_{i=0}^{n} x_i w_i = b + \sum_{i=1}^{n} x_i w_i \tag{3.1}$$

Váženým součtem vstupů (rovnice 3.1) je vypočten tzv. **vnitřní potenciál neuronu** z. Vstupy neuronu lze také vyjádřit jako vektor $x = (x_0, x_1, ..., x_n)$ $= (1, x_1, x_2, ..., x_n)$, váhy jako vektor $w = (w_0, w_1, ..., w_n) = (b, w_1, w_2, ..., w_n)$. Pak je možné vnitřní potenciál neuronu formulovat také jako skalární součin vektorů x a w, tedy $z = x \cdot w$.

Výstupní hodnota (aktivace) neuronu a je dána **aktivační funkcí** jako $a = f(z)$. Jednoduchým příkladem takové funkce je skoková funkce (předpis 3.2), která nabývá hodnot 0 a 1 a tím i odpovídá představě biologického neuronu (buď je aktivní nebo neaktivní).

$$f(z) = \begin{cases} 0 & \text{pokud } z < 0 \\ 1 & \text{pokud } z \geq 0 \end{cases} \tag{3.2}$$

V případě použití prahu θ má skoková funkce předpis 3.3. Neboli výstupní hodnota neuronu a (v tomto případě 0 nebo 1) závisí na tom, jestli hodnota vnitřního potenciálu neuronu z dosahuje prahové hodnoty θ.

$$f(z) = \begin{cases} 0 & \text{pokud } z < \theta \\ 1 & \text{pokud } z \geq \theta \end{cases}, \text{ kde } z = \sum_{i=1}^{n} x_i w_i \tag{3.3}$$

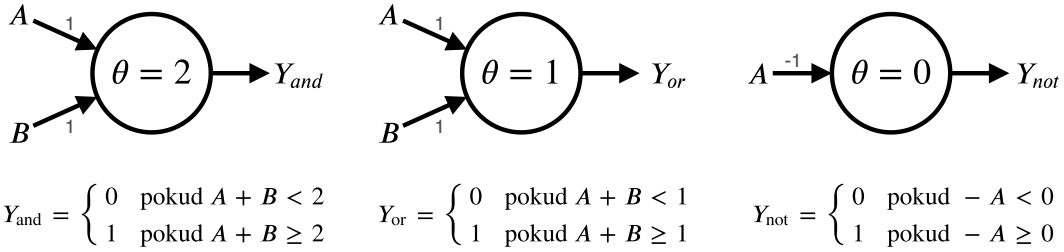
Odsud lze jednoduchou úpravou nerovnice získat vztah $z - \theta < 0$ a z toho tedy už výše uvedený vztah $b = -\theta$.

Existuje mnoho funkcí, které se využívají jako aktivační funkce neuronů. Aktivačním funkcím je věnována samostatná část 3.5 Aktivační funkce.

3.2 Stručná historie neuronových sítí

Historie umělých neuronových sítí sahá až do roku **1943**, kdy **Warren McCulloch** a **Walter Pitts** navrhli první matematický model neuronu, který bývá označován jako **MCP neuron** (McCulloch-Pitts neuron). MCP neuron pracuje pouze s binárními hodnotami (reprezentovanými jako 0 nebo 1) na vstupech i na výstupu. Excitační vstupy jsou ohodnoceny jednou fixní vahou (+1). Aktivní inhibiční vstup (vyjádřen váhou -1) zamezuje aktivaci neuronu bez ohledu na ostatní vstupy. Práh θ je stanoven fixní hodnotou, aktivace neuronu je realizována pomocí skokové aktivační funkce. Aby mohl být daný neuron aktivován (hodnota 1 na výstupu), musí platit, že vážená suma vstupů dosahuje alespoň hodnoty prahu a zároveň nesmí být aktivní inhibiční vstup. Pomocí MCP neuronu lze snadno vyjádřit některé logické funkce (AND, OR, NOT, NOR a NAND), ale už ne funkce XOR nebo XNOR. Omezující

je také možnost použití pouze binární hodnoty pro vstupy a výstup a také fixní (a pro všechny vstupy stejné) váhy. [29, 30]



$$Y_{\text{and}} = \begin{cases} 0 & \text{pokud } A + B < 2 \\ 1 & \text{pokud } A + B \geq 2 \end{cases} \qquad Y_{\text{or}} = \begin{cases} 0 & \text{pokud } A + B < 1 \\ 1 & \text{pokud } A + B \geq 1 \end{cases} \qquad Y_{\text{not}} = \begin{cases} 0 & \text{pokud } -A < 0 \\ 1 & \text{pokud } -A \geq 0 \end{cases}$$

A	B	Y_{and}	Y_{or}
0	**0**	0	0
0	**1**	0	1
1	**0**	0	1
1	**1**	1	1

A	Y_{not}
0	1
1	0

Obrázek 3.3: Reprezentace logických funkcí AND, OR, NOT pomocí MCP neuronu (zdroj: autor)

V roce **1949** vydal psycholog **Donald Hebb** knihu *The Organization of Behavior*. V této knize vyjádřil myšlenku o posilování vazby mezi neurony, pokud jsou dva blízké neurony aktivní ve stejný čas. To také označil jako jednu ze základních funkcí, které jsou nezbytné pro učení a paměť. Tato teorie byla poté aplikována na umělé neurony v podobě využití vážených vstupů (samostatná váha pro každý vstup). [30]

Později v roce **1958** popsal **Frank Rosenblatt** vylepšený model neuronu, který nazval **perceptron**. Na rozdíl od MCP neuronu tedy využíval vážených vstupů, neomezoval se pouze na binární vstupy a také pro něj existoval učící algoritmus. Perceptron se dá chápat jako obecný model neuronu, který je popsán v části 3.1 Neuron. Někdy také bývá označován jako učící algoritmus pro binární klasifikátor.

Ani perceptron však nedokázal řešit všechny funkce. V roce **1969** navíc **Marvin Minsky** a **Seymour Papert** popsali limity perceptronu. Ukázali, že perceptron stále nedokáže vyřešit funkce XOR a XNOR a dokáže tedy řešit pouze tzv. lineárně separabilní problémy. [30] Ve skutečnosti je tento problém řešitelný pomocí více vrstev perceptronů, ale v té době neexistoval učící algoritmus pro vícevrstvou síť. [28]

Výzkum neuronových sítí se v následujících letech zpomalil, toto období je známé jako „AI winter“.

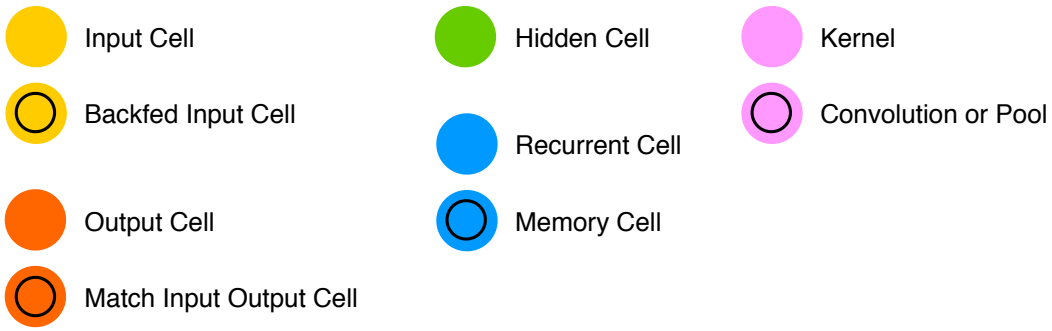
Již v roce 1971 vyvinul Paul Werbos algoritmus zpětného šíření chyby, známý jako **backpropagation** training algorithm, který v roce 1974 publikoval ve své doktorské práci. Ale známým se tento algoritmus stal až v roce **1986**, kdy **D. Rumelhart**, **G. Hinton** a **R. Williams** publikovali článek *Learning Representations by back-propagating errors*. [31]

Algoritmus zpětného šíření chyby je základem při učení neuronových sítí až dodnes. Učení neuronových sítí (včetně algoritmu backpropagation) je věnována část 3.4 Učení neuronových sítí.

3.3 Architektura neuronových sítí

Modely neuronových sítí lze reprezentovat jako vážené orientované grafy, kde vrcholy představují neurony a orientované hrany značí propojení těchto neuronů (výstup jednoho neuronu je vstupem jiných neuronů). Ohodnocením hran jsou vyjádřeny váhy jednotlivých vstupů.

Podle uspořádání tohoto grafu, tedy podle uspořádání neuronů, jejich funkce a funkce celé sítě se dají ANN rozdělit na různé architektury. Existuje mnoho takových architektur, další průběžně vznikají a nelze je tedy obsáhnout úplně všechny.

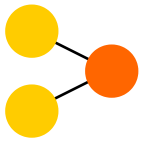


Obrázek 3.4: Význam symbolů použitých na grafických znázorněních architektur neuronových sítí (zdroj: [32])

Dále jsou uvedeny některé vybrané architektury neuronových sítí. U každé z nich je na diagramu pro ilustraci zobrazen její zjednodušený model. Význam symbolů na těchto diagramech popisuje legenda na obrázku 3.4.

3.3.1 Perceptron

Perceptron byl již zmíněn v části 3.2 Stručná historie neuronových sítí jako model umělého neuronu nebo učící algoritmus pro binární klasifikátor. V kontextu architektury neuronových sítí bývá označován také jako nejjednodušší typ neuronových sítí, ale v podstatě se stále jedná o jeden neuron.



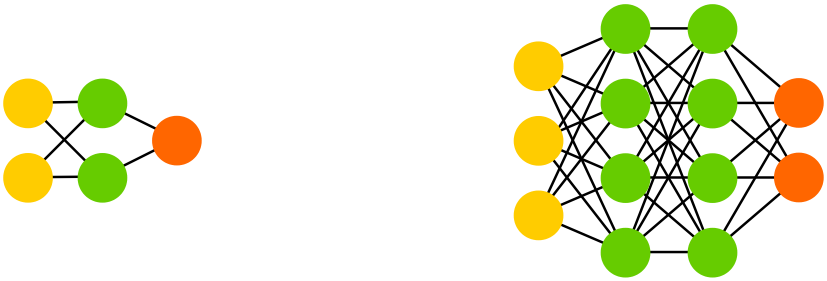
Obrázek 3.5: Perceptron (zdroj: [32])

Pro perceptron jako architekturu sítě je přesnější označení spíše singlelayer perceptron (SLP), tedy síť s jednou vrstvou perceptronů. Sítě s více vrstvami perceptronů (neuronů) se označují jako multiplayer perceptron (MLP), používanější je

spíše označení feedforward neural network (dopředná neuronová síť) — viz 3.3.2 Dopředné neuronové sítě.

3.3.2 Dopředné neuronové sítě

Dopředné neuronové sítě (feedforward neural networks, zkratka **FFNN**) jsou jedny z nejjednodušších neuronových sítí. Jednotlivé neurony jsou obvykle uspořádány to tzv. vrstev. Jednoduchá FFNN je znázorněna na obrázku 3.6a.



(a) Dopředná neuronová sít **(b)** Hluboká dopředná neuronová sít

Obrázek 3.6: Dopředné neuronové sítě (zdroj: [32])

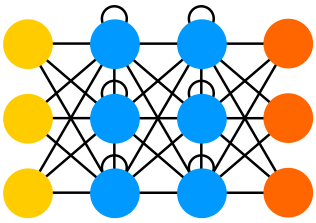
Neurony (jejich výstupy) z jedné **vrstvy** jsou spojeny s neurony (jejich vstupy) ve vrstvě následující. Dvě sousední vrstvy jsou obvykle **úplně spojené**, tedy každý neuron z jedné vrstvy je propojen s každým neuronem v následující vrstvě. Ale v jedné vrstvě spojení mezi neurony neexistují. Tyto vrstvy se dělí na **vstupní** (input layer), **skryté** (hidden layer) a **výstupní** (output layer). Stejným způsobem se označují neurony v těchto vrstvách. Tok informací ve FFNN je jednosměrný, od vstupních přes skryté až do výstupních neuronů.

Dopředné neuronové sítě, které mají více skrytých vrstev se nazývají jako **hluboké dopředné neuronové sítě** (deep feedforward neural networks, **DFFNN**), znázorněné na obrázku 3.6b na předcházející straně. Jako hluboké dopředné neuronové sítě se běžně označují FFNN, které mají více než jednu skrytou vrstvu, v praxi se jedná o sítě spíše s minimálně desítkami skrytých vrstev.

Obvyklé využití FFNN a DFFNN je při tzv. učení s učitelem (viz 2.2.1 Učení s učitelem). Často bývají také součástí jiných architektur neuronových sítí, např. konvolučních neuronových sítí (viz 3.3.4 Konvoluční neuronové sítě). [32]

3.3.3 Rekurentní neuronové sítě

Rekurentní neuronové sítě (recurrent neural networks, **RNN**) jsou podobné dopředným neuronovým sítím, ale na rozdíl od nich mají vnitřní stav (paměť). Skryté neurony nedostávají informace pouze z předchozí vrstvy, ale pomocí rekurentního spojení mají navíc informaci o svém vlastním výstupu v předchozím kroku. [32]

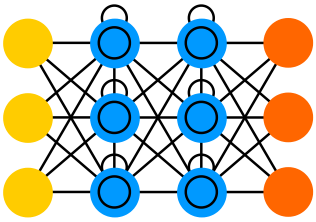


Obrázek 3.7: Rekurentní neuronová síť (zdroj: [32])

Díky této jednoduché paměti jsou RNN využitelné zejména pro sekvenční data, jako jsou například texty. Konkrétně se využívají např. pro generování textu, předpovídání slov (automatické dokončování) strojový překlad nebo předpovídání časových řad. [33]

Problémem RNN je tzv. mizející nebo explodující gradient (viz 3.4.2 Mizející a explodující gradient), kdy dochází ke ztrátě informace v čase. [32]

Jednou z variant RNN, která řeší problém s gradientem jsou sítě **Long short-term memory**, zkratka **LSTM** (do češtiny se překládají jako sítě s dlouhou krátkodobou pamětí).



Obrázek 3.8: LSTM – Long short-term memory network (zdroj: podle [32])

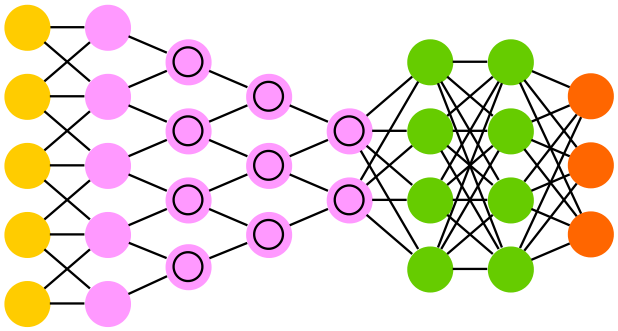
LSTM jsou inspirovány spíše elektrickými obvody, každý (skrytý) neuron má explicitní paměť a tři brány (gates) — vstupní, výstupní a zapomínací (forget). Vstup určuje, kolik informací zůstane uloženo v paměti a výstup určuje kolik informací dostanou neurony v další vrstvě. Zapomínání pak může být vhodné například při zpracování textu — síť při zpracování věty pravděpodobně nebude potřebovat kontext předchozí věty, a tak může tuto informaci „zapomenout“. [32]

3.3.4 Konvoluční neuronové sítě

Konvoluční neuronové sítě (convolutional neural networks, zkratka **CNN** nebo také ConvNet) jsou využívány především při zpracování obrazu. Pro zpracování obrazu je možné použít i klasické dopředné neuronové sítě, ale problémem je velikost obrázků. Jestliže by měl obrázek rozlišení například pouhých 100x100 pixelů, musela by mít taková síť 10 000 vstupů. U barevných obrázků má navíc každý pixel 3 barevné složky (RGB — červená, zelená, modrá), celkem by šlo tedy o 30 000 vstupních

hodnot. [32, 34] Pro barevný obrázek 500x500 pixelů by to bylo 750 000 vstupních hodnot. Zkrátka „klasické“ neuronové sítě nejsou pro takové zpracování obrazu škálovatelné.

Obraz je reprezentován jako matice, kde jednotlivé hodnoty představují (intenzitu) barvy pixelů. Pro černobílý obraz je tato matice jedna, hodnoty jsou v rozmezí 0 až 255[1] (tzn. černá až bílá). Pro barevné obrazy jsou matice celkem tři, pro každou složku barvy jedna. Opět s hodnotami 0 až 255, které signalizují intenzitu dané barvy.

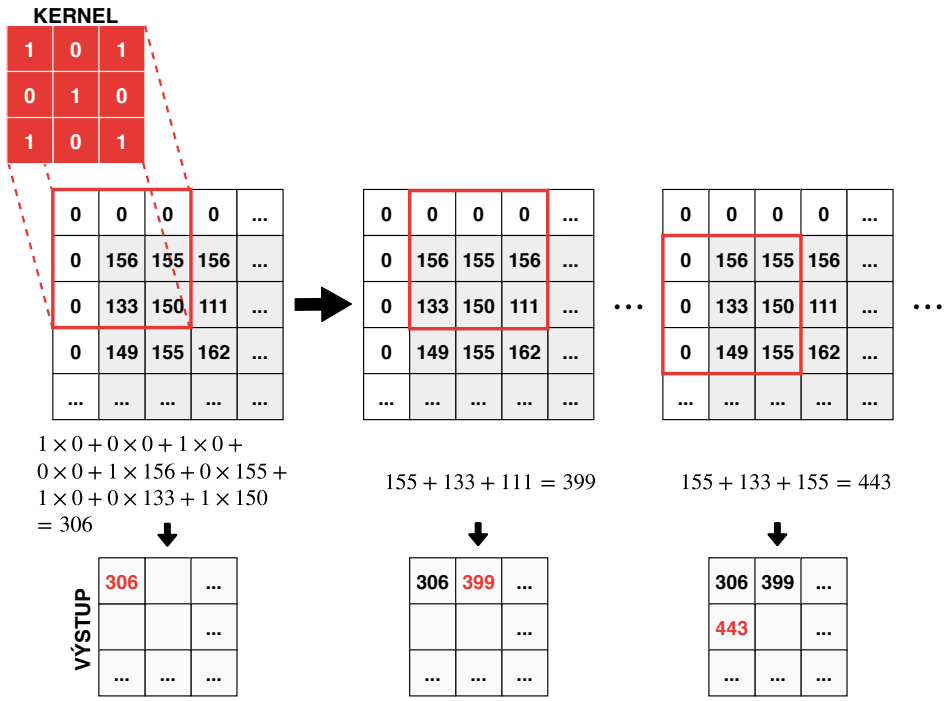


Obrázek 3.9: Konvoluční neuronová síť (zdroj: [32])

Název CNN vychází z operace, která je zde využívána — **konvoluce**. CNN postupně čte obraz po malých částech. Velikost těchto částí závisí na rozměru další matice (např. 3x3) známé jako **kernel** (konvoluční jádro nebo také filtr). Filtr se takto posouvá a probíhá násobení jeho hodnot s hodnotami, které leží pod ním. Součet těchto násobků pak tvoří hodnotu v matici na výstupu konvoluční vrstvy. Tato operace se opakuje, dokud filtr neprojde celou šířku obrazu a poté pokračuje od začátku dalšího řádku. Takto postupně dojde k přečtení celého obrazu. Fungování konvoluce je naznačeno na obrázku 3.10.

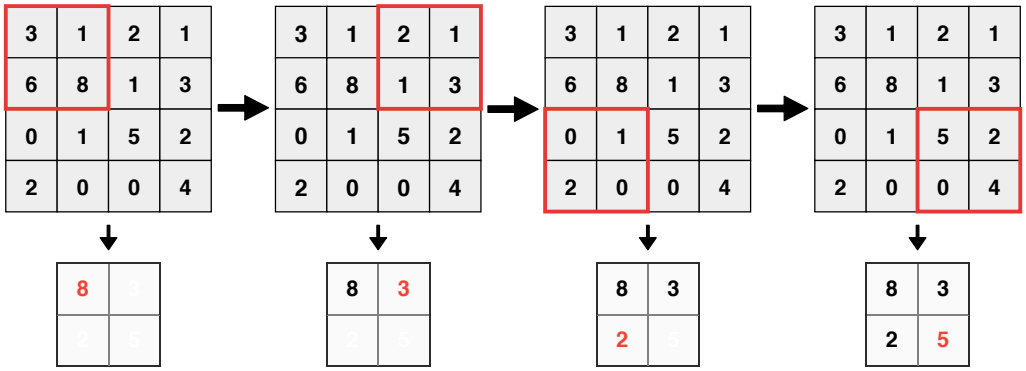
Délka posunu se označuje jako **stride** (krok). Typicky se používá krok délky 1, takže dochází k pomyslnému překrývání filtru při posunu. Aby při procházení nedocházelo ke ztrátě informací, tak se okraje obrázku ošetřují přidáním pixelů např. hodnoty 0. To se nazývá **padding** (odsazení), známé také jako zero padding právě kvůli použití nuly. Výstupem konvoluční vrstvy je potom „obrázek“ (matice), který je stejně velký jako vstup (pokud bylo použito odsazení) a na něj se dá aplikovat další konvoluce. Je důležité, aby CNN měly několik konvolučních vrstev, protože jednotlivé vrstvy umožňují zachytit důležité vlastnosti obrázku – od jednoduchých rysů jako jsou hrany nebo barvy (v prvních vrstvách) přes textury a vzory v následujících vrstvách až po komplexní objekty v posledních vrstvách. [34, 35]

[1] nebo lze použít také hodnoty od 0 do 1 nebo také od -1 do 1



Obrázek 3.10: Princip fungování konvoluce (zdroj: autor podle [35])

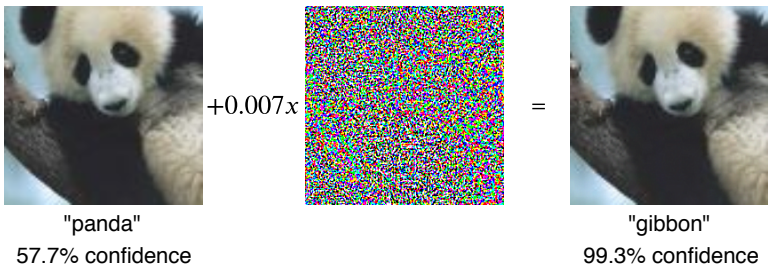
Další důležitou součástí CNN je **pooling** (sdružování), nejčastěji se používá tzv. **max-pooling**. Tato vrstva se aplikuje podobně jako konvoluce na všechny části obrázku (nebo výstupu z konvoluce) a z každé části (např. 2x2) vrací maximum. V tomto případě se ale překrývání nepoužívá, takže dochází ke snížení rozlišení výsledného obrazu, a tedy i k redukci celkového objemu dat. Metoda max-pooling je naznačena na obrázku 3.11.



Obrázek 3.11: Max-pooling (zdroj: autor podle [35])

Kromě snížení nároků na výkon potřebného pro zpracování takového množství obrazových dat dochází také k extrakci dominantních rysů. Postupným skládáním konvolučních a pooling vrstev lze získat dostatečně malou reprezentaci původního obrazu, na kterou lze už použít FFNN. [34, 35] Kombinace konvolučních sítí spolu s dopřednými sítěmi se někdy označuje jako hluboké konvoluční neuronové sítě (deep convolutional neural networks, DCNN). [32]

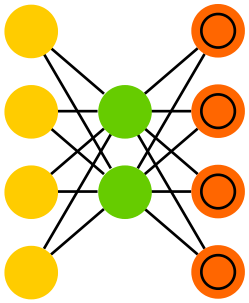
Problémem CNN je, že jsou zranitelné pomocí tzv. matoucích vzorů. Obrázek může být drobně upraven tak, že člověk nepozná rozdíl, ale neuronová síť ho může například špatně klasifikovat. Nebo je síť schopná klasifikovat obrázek s přesností přes 99 %, ale pro člověka je nerozpoznatelný. Tento problém se netýká nutně jen CNN, ale na nich se dá lehce interpretovat (viz obrázek 3.12). Velmi nebezpečné to může být při nasazení neuronových sítí v reálném světě (např. řízení autonomních vozidel). [34, 36]



Obrázek 3.12: Ukázka chybné klasifikace obrázku díky drobné (pro člověka nerozeznatelné) změně (zdroj: převzato z [37, s. 3])

3.3.5 Autoenkodéry

Autoenkodér (autoencoder, **AE**) je jednoduchá (typicky dvouvrstvá) neuronová síť. Základní myšlenkou je automatické kódování vstupů a následně jejich zpětné dekódování na výstupy s co nejmenší chybou. Díky tomu, že je ve skrytých vrstvách menší počet neuronů, než je počet vstupů, nemůže síť pouze jednoduše překopírovat vstup na výstup, ale musí generalizovat. Výhodou autoenkodérů je, že jsou jednoduché (mají malý počet vrstev) a je tedy relativně jednoduché je natrénovat. [32, 38]



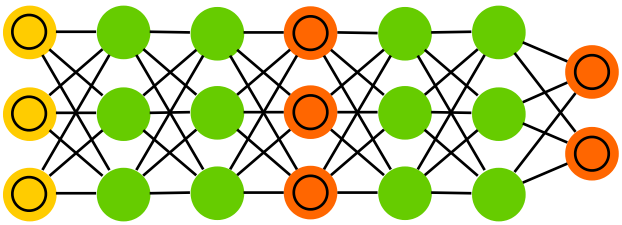
Obrázek 3.13: Autoenkodér (zdroj: [32])

Autoenkodéry se využívají pro odstranění šumu (např. z obrázků, audia), detekci anomálií nebo redukci dimenzionality.

3.3.6 GAN

Generative adversarial networks, označované zkratkou **GAN** (česky se někdy uvádí název generativní kompetitivní neuronové sítě), se skládají ze dvou částí (sítí). Jedna část má za úkol generovat obsah (např. obraz, text), ta se nazývá **generátor**. Druhá část, nazývaná **diskriminátor**, se snaží obsah od generátoru hodnotit. Diskriminátor dostane od generátoru buď trénovací data nebo generovaný obsah a snaží se

odhadnout, ze které množiny data pocházejí. Obě části se navzájem trénují tak, že generátor se snaží být méně předvídatelný (maximalizovat chybu diskriminátoru) a diskriminátor snaží lépe rozeznávat reálná data od generovaných (minimalizovat svoji chybu). Po ukončení trénování se obvykle používá pouze generátor. [34, 32]



Obrázek 3.14: Generative adversarial networks (zdroj: [32])

Pomocí GAN byl vytvořen například generátor lidských obličejů[2], existují různé generátory obrázků, generátory textů (básně, programovací jazyky) apod. GAN lze využít například také pro tzv. **přenos stylu obrazu**[3], kdy může být obyčejná fotografii přetransformována např. do stylu malíře van Gogha.

3.4 Učení neuronových sítí

Znalosti jsou v neuronových sítích uloženy ve vahách (a biasech). Učení (trénování) neuronové sítě znamená postupnou úpravu těchto parametrů tak, aby síť podávala co nejpřesnější výsledky.

Aby bylo možné při procesu učení neuronové sítě upravovat její parametry (váhy a biasy), musí mít tato síť nejdříve nějaké počáteční parametry nastavené. Ty se obvykle nastavují náhodně na nízké hodnoty. Neuronové síti se poté předloží vstupní data, v závislosti na parametrech a celkovém nastavení sítě dojde k aktivaci některých neuronů, konečným výstupem je pak předpověď. Tato **předpověď** se porovná s **očekávaným výsledkem**, který by měla síť poskytnout. Rozdíl mezi předpovědí a očekávaným výsledkem se označuje jako **chyba** (error) a hlavním cílem je minimalizovat tuto chybu. To znamená upravit parametry sítě tak, aby byla výsledná chyba co nejnižší. Postupně je takto celý proces opakován, ideálně dokud neklesne chyba pod nějakou požadovanou mez. Tato metoda učení neuronových sítí je v současnosti nejvíce využívaná, jedná se o tzv. učení s učitelem. Více o způsobech učení je uvedeno v části 2.2 Typy strojového učení.

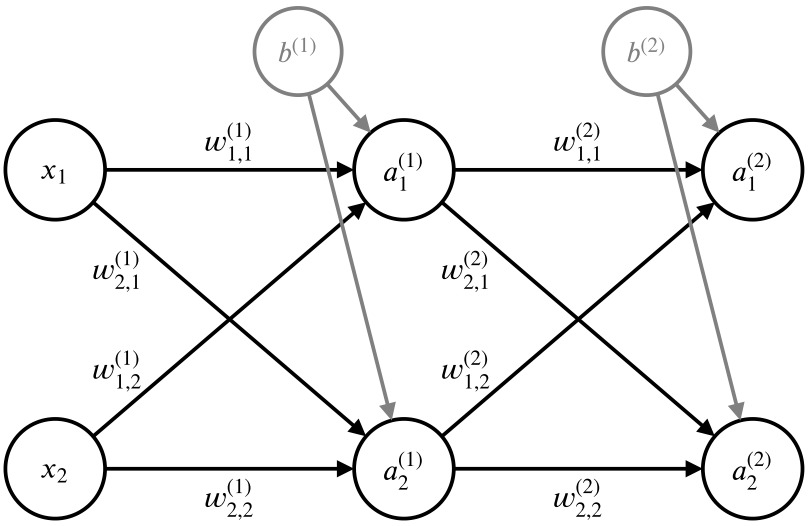
[2] `https://thispersondoesnotexist.com`
[3] `https://demos.algorithmia.com/deep-style`

3.4.1 Zpětné šíření chyby

Pokud je neuronová síť složená pouze z jednoho nebo jedné vrstvy neuronů (vstupy sítě jsou přímo připojeny k výstupním neuronům), není tak složité takovou síť učit neboli upravovat jednotlivé parametry a minimalizovat její chybu. Takové sítě ovšem nejsou dostačující, protože nedokáží řešit komplexnější problémy. Moderní neuronové sítě se v naprosté většině skládají z několika vrstev neuronů, často se o nich hovoří jako o hlubokých neuronových sítích (viz 3.3 Architektura neuronových sítí). Konečný výstup takové sítě je ovlivněn nejen vstupy z předchozí vrstvy, ale i všemi dalšími předchozími vrstvami a jejich neurony.

Hluboké sítě prakticky nebylo možné učit, a to až do objevení algoritmu **backpropagation** (algoritmu zpětného šíření chyby), který je dodnes stále využíván.

Pro lepší ilustraci je na obrázku 3.15 znázorněna jednoduchá neuronová síť, která je využita v následujícím jednoduchém příkladu. Uvedený příklad má za cíl poskytnout pouze základní pohled na algoritmus backpropagation a principy jeho fungování.



Obrázek 3.15: Jednoduchá neuronová síť se dvěma vrstvami pro popis principu fungování algoritmu backpropagation (zdroj: autor)

Prvním krokem učení je inicializace parametrů sítě — vah w a biasů b, obvykle náhodně s nízkými hodnotami. Spodní index u vah značí, mezi kterými dvěma neurony existuje propojení s touto vahou a horní index označuje číslo vrstvy. Například váha $w^{(2)}_{2,1}$ patří ke spojení mezi výstupem prvního neuronu v první vrstvě a vstupem druhého neuronu v druhé vrstvě. Bias existuje pro každou vrstvu pouze jeden, proto má jen horní index s označením vrstvy.

Následně jsou síti předložena data na vstupy x_1 a x_2 a při dopředném průchodu

sítí jsou počítány jednotlivé aktivace neuronů a. Například $a_1^{(2)} = f(z_1^{(2)})$, kde f je aktivační funkce a $z_1^{(2)} = a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + b^{(2)}$ je vnitřní potenciál prvního neuronu ve druhé vrstvě.

Po dopředném průchodu jsou na výstupech sítě, kterými jsou výstupy poslední vrstvy $a_1^{(2)}$ a $a_2^{(2)}$, dostupné předpovědi. Porovnáním předpovědí $a_1^{(2)}$ a $a_2^{(2)}$ s očekávanými hodnotami t_1 a t_2 je vyjádřena chyba (error, cost) $E = g(a, t)$, kde $g(a, t)$ je tzv. **chybová funkce** (loss function, cost function, error function)[4].

Existují různé chybové funkce jako je například střední kvadratická chyba (mean squared error, MSE) nebo křížová entropie (cross-entropy), v tomto případě je pro jednoduchost využita právě MSE, tzn. $g(a, t) = \frac{1}{2}(t - a)^2$ (použití $\frac{1}{2}$ později lehce zjednoduší a zpřehlední výpočet). Chyba pro první výstup je $E_{a_1^{(2)}} = \frac{1}{2}(t_1 - a_1^{(2)})^2$, stejným způsobem je vyjádřena i chyba $E_{a_2^{(2)}}$ pro druhý výstup. Celková chyba E je pak tedy:

$$E = E_{a_1^{(2)}} + E_{a_2^{(2)}} = \frac{1}{2}(t_1 - a_1^{(2)})^2 + \frac{1}{2}(t_2 - a_2^{(2)})^2 \tag{3.4}$$

Dalším krokem je minimalizace této chyby neboli nalezení minima celkové chybové funkce E pomocí úprav parametrů sítě. K tomu se využívá optimalizační metoda nazývaná jako **gradientní sestup** (gradient descent). Například vliv váhy $w_{1,1}^{(2)}$ na celkovou chybu E lze vyjádřit jako $\frac{\partial E}{\partial w_{1,1}^{(2)}}$ neboli parciální derivací funkce E vzhledem k váze $w_{1,1}^{(2)}$, někdy se toto nazývá také jako gradient vzhledem k váze $w_{1,1}^{(2)}$. Obdobně pro vliv biasu $b^{(2)}$ platí $\frac{\partial E}{\partial b^{(2)}}$.[5]

Cílem je minimalizovat chybu, to znamená „pohyb v opačném směru gradientu“ – odsud tedy název gradientní sestup (znázorněno na obrázku 3.16 na následující straně). Toto lze vyjádřit jako $\Delta w_{1,1}^{(2)} \propto -\frac{\partial E}{\partial w_{1,1}^{(2)}}$ ($\Delta w_{1,1}^{(2)}$ značí změnu váhy $w_{1,1}^{(2)}$). Přesněji je tato změna dána jako $\Delta w_{1,1}^{(2)} = -\eta \frac{\partial E}{\partial w_{1,1}^{(2)}}$, kde η je tzv. míra učení (**learning rate**), která udává rychlost změny váhy.

Chyba E není přímou funkcí[6] váhy $w_{1,1}^{(2)}$, proto je třeba $\frac{\partial E}{\partial w_{1,1}^{(2)}}$ přepsat jako derivaci složené funkce[7]:

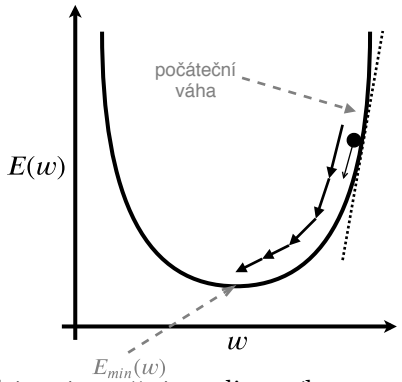
$$\frac{\partial E}{\partial w_{1,1}^{(2)}} = \frac{\partial E}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(2)}} \tag{3.5}$$

[4]Označení *loss* bývá někdy používáno jako chyba pro jeden průchod a *cost* pak jako chyba pro celý tréninkový dataset (nebo pro jeho určitou část — dávku).

[5]Úprava biasu dále není v tomto příkladu uváděna, princip výpočtu těchto změn je stejný jako u vah.

[6]Jedná se o složenou funkci

[7]anglicky se toto označuje jako *chain rule*



Obrázek 3.16: Grafické znázornění gradientního sestupu (zdroj: autor podle [39])

Pro jednotlivé složky platí:

$$\frac{\partial E}{\partial a_1^{(2)}} = \frac{\partial(\frac{1}{2}(t_1 - a_1^{(2)})^2)}{\partial a_1^{(2)}} = (t_1 - a_1^{(2)})(-1) = -(t_1 - a_1^{(2)}) \tag{3.6}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = f'(z_1^{(2)}) \tag{3.7}$$

$$\frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(2)}} = \frac{\partial(a_1^{(1)} w_{1,1}^{(2)} + a_2^{(1)} w_{1,2}^{(2)} + b^{(2)})}{\partial w_{1,1}^{2)}} = a_1^{(1)} \tag{3.8}$$

a tedy

$$\frac{\partial E}{w_{1,1}^{(2)}} = -(t_1 - a_1^{(2)}) f'(z_1^{(2)}) a_1^{(1)} \tag{3.9}$$

Odsud lze vyjádřit $\delta_{a_1^{(2)}} = -(t_1 - a_1^{(2)}) f'(z_1^{(2)})$.

Poté je výpočet pro všechny ostatní váhy ve druhé (poslední) vrstvě následující:

$$\frac{\partial E}{w_{1,1}^{(2)}} = \delta_{a_1^{(2)}} a_1^{(1)} \qquad \frac{\partial E}{w_{2,1}^{(2)}} = \delta_{a_2^{(2)}} a_1^{(1)} \tag{3.10}$$

$$\frac{\partial E}{w_{1,2}^{(2)}} = \delta_{a_1^{(2)}} a_2^{(1)} \qquad \frac{\partial E}{w_{2,2}^{(2)}} = \delta_{a_2^{(2)}} a_2^{(1)} \tag{3.11}$$

a tedy např. změna váhy $w_{1,1}^{(2)}$ je pak $\Delta w_{1,1}^{(2)} = -\eta \delta_{a_1^{(2)}} a_1^{(1)}$.

Dále je třeba pokračovat vahami v dalších (skrytých) vrstvách, v tomto případě vahami v první vrstvě. Pro $w_{1,1}^{(1)}$ platí

$$\frac{\partial E}{w_{1,1}^{(1)}} = \frac{\partial E}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial w_{1,1}^{(1)}} \tag{3.12}$$

Výstupy neuronů skryté vrstvy ovlivňují oba neurony ve výstupní vrstvě (tedy i jejich chybu), to znamená:

$$\frac{\partial E}{\partial a_1^{(1)}} = \frac{\partial E_{a_1^{(2)}}}{\partial a_1^{(1)}} + \frac{\partial E_{a_2^{(2)}}}{\partial a_1^{(1)}} \tag{3.13}$$

kde

$$\frac{\partial E_{a_1^{(2)}}}{\partial a_1^{(1)}} = \frac{\partial E}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} = \frac{\partial E}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} w_{1,1}^{(2)} \tag{3.14}$$

$$\frac{\partial E_{a_2^{(2)}}}{\partial a_2^{(1)}} = \frac{\partial E}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial a_2^{(1)}} = \frac{\partial E_2}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} w_{2,1}^{(2)} \tag{3.15}$$

Dále analogicky jako předtím (ve druhé vrstvě)

$$\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} = f'(z_1^{(1)}) \tag{3.16}$$

$$\frac{\partial z_1^{(1)}}{\partial w_{1,1}^{(1)}} = \frac{\partial (x_1 w_{1,1}^{(1)} + x_2 w_{1,2}^{(1)} + b^{(1)})}{\partial w_{1,1}^{2)}} = x_1 \tag{3.17}$$

Dosazením vznikne zápis

$$\frac{\partial E}{w_{1,1}^{(1)}} = \left(\frac{\partial E}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} w_{1,1}^{(2)} + \frac{\partial E}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} w_{2,1}^{(2)} \right) f'(z_1^{(1)}) x_1 \tag{3.18}$$

a z předchozích výpočtů lze pak dále dosadit a získat

$$\frac{\partial E}{w_{1,1}^{(1)}} = (\delta_{a_1^{(2)}} w_{1,1}^{(2)} + \delta_{a_2^{(2)}} w_{2,1}^{(2)}) f'(z_1^{(1)}) x_1 \tag{3.19}$$

Nakonec lze opět vyjádřit δ, např. $\delta_{a_1^{(1)}} = (\delta_{a_1^{(2)}} w_{1,1}^{(2)} + \delta_{a_2^{(2)}} w_{2,1}^{(2)}) f'(z_1^{(1)})$. Výpočet pro váhy v první (skryté) vrstvě pak následující:

$$\frac{\partial E}{w_{1,1}^{(1)}} = \delta_{a_1^{(1)}} x_1 \qquad \frac{\partial E}{w_{2,1}^{(1)}} = \delta_{a_2^{(1)}} x_1 \tag{3.20}$$

$$\frac{\partial E}{w_{1,2}^{(1)}} = \delta_{a_1^{(1)}} x_2 \qquad \frac{\partial E}{w_{2,2}^{(1)}} = \delta_{a_2^{(1)}} x_2 \tag{3.21}$$

Změna váhy $w_{1,1}^{(1)}$ je poté dána jako $\Delta w_{1,1}^{(1)} = -\eta \delta_{a_1^{(1)}} x_1$ a obdobně to platí i pro ostatní váhy v první vrstvě.

Pokud by byla síť hlubší, pokračovalo by zpětné šíření chyby dále stejně i pro další skryté vrstvy. Prakticky dochází jen k využívání předchozích výpočtů a jejich „řetězení“.

Po zpětném průchodu a přenastavení vah následuje opět dopředný průchod a vše se opakuje většinou až dokud neklesne chyba pod určitou hranici nebo dokud není proveden určitý počet průchodů (tzv. epoch).

3.4.2 Mizející a explodující gradient

Mizející gradient (**vanishing gradient**) je nestabilní chování, které se vyskytuje při učení hlubokých a také v rekurentních neuronových sítí (viz. 3.3 Architektura neuronových sítí), kdy není možné šířit informace o gradientu z vrstev u výstupu zpět k vrstvám u vstupu (postupným násobením nízkých hodnot na intervalu $\langle 0, 1 \rangle$ se gradient stále zmenšuje a přibližuje nule). Sítě tak předčasně konvergují ke špatnému řešení nebo je prakticky nemožné je učit. Tento problém se řeší většinou používáním různých aktivačních funkcí (viz 3.5 Aktivační funkce), před-trénováním vah apod. [40, 41]

Opačným problémem je druhé nestabilní chování, tzv. explodující gradient (**exploding gradient**), kdy se hodnoty gradientu při učení postupně zvyšují, to znamená velké změny parametrů sítě, a tedy nestabilní učení. Tento problém se často projevuje tak, že rychle dochází k velkým změnám vah během učení (váhy se mohou dostat na hodnotu NaN) případně chyba při trénování se může dostat až na hodnotu NaN. Pro ošetření explodujícího gradientu existují metody jako např. gradient clipping, což prakticky znamená omezení velikosti gradientu. Nebo může stačit jen zmenšit dávky (batch size) pro trénování. [42]

3.5 Aktivační funkce

Aktivační funkce[8] (activation function) definuje jaký bude výstup neuronu na základě jeho vstupů.

Jednou ze základních vlastností, kterou by měla aktivační funkce mít je **nelinearita**. Lineární funkce jsou tzv. uzavřené na skládání, to znamená, že výstupem složených lineárních funkcí bude opět jen lineární funkce. Naopak je dokázáno, že síť s využitím nelineární aktivační funkce dokáže aproximovat jakoukoli spojitou funkci. [43, 44]

Aktivační funkce by zároveň měla být také **spojitě diferencovatelná**. Důvodem je učící algoritmus zpětného šíření chyby (viz 3.4.1 Zpětné šíření chyby). Nicméně v

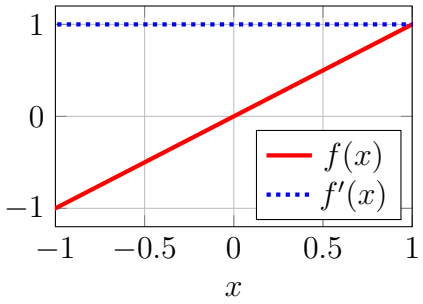
[8] někdy se označuje také jako přenosová funkce

praxi se využívají i funkce, které tuto podmínku přesně nesplňují (viz 3.5.4 ReLU).

Existuje celá řada aktivačních funkcí, některé známé nebo často využívané jsou uvedeny dále.

3.5.1 Lineární funkce (identita)

Jak již bylo uvedeno, lineární funkce, v tomto případě přesněji **identita** $f(x) = x$ s oborem hodnot $H(f) : y \in (-\infty, \infty)$, není vhodná jako aktivační funkce. Její derivace je konstantní ($f'(x) = 1$), takže není možné použít algoritmus backpropagation pro učení. Také je uzavřena na skládání, skládáním této funkce tedy nakonec vznikne opět jen lineární funkce.



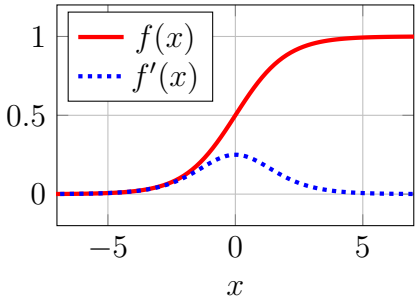
Graf 3.1: Linerární funkce (identita)

V praxi je tato funkce využitelná pouze ve výstupní vrstvě neuronů při řešení úloh regrese (výstupní hodnoty jsou reálná čísla).

3.5.2 Sigmoida

Sigmoida je tradiční aktivační funkce, její předpis a derivace jsou dány předpisem 3.22. Obor hodnot je $H(f) : y \in (0, 1)$. Její průběh je zobrazen na grafu 3.2.

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x)(1 - f(x)) \tag{3.22}$$



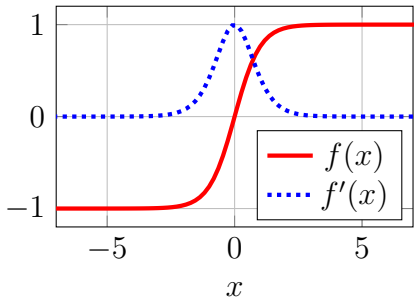
Graf 3.2: Sigmoida

Dříve byla tato funkce obecně doporučována a často využívána, dnes se již téměř nepoužívá zejména kvůli problému mizejícího gradientu (**vanishing gradient**, viz 3.4.2 Mizející a explodující gradient) a obecně existenci jiných (lepších) funkcí. Setkat se s ní lze spíše už jen ve výukových materiálech a příkladech.

3.5.3 Hyperbolický tangens

Funkce hyperbolický tangens, označovaná jako tanh, je dána předpisem 3.23, derivace je $f'(x) = tanh'(x) = 1 - tanh(x)^2$ a obor hodnot je $H(f) : y \in (-1, 1)$.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.23}$$



Graf 3.3: Hyperbolický tangens (tanh)

Tato funkce je podobná funkci sigmoida, rozdíl je v oboru hodnot, které jsou zde v intervalu $(-1, 1)$. Takže dokáže lépe reprezentovat negativní hodnoty, pro které bude mít aktivace také zápornou hodnotu. Také má silnější gradient (derivace dosahuje vyšších hodnot), ale stejně jako u sigmoidy je zde problém mizejícího gradientu.

3.5.4 ReLU

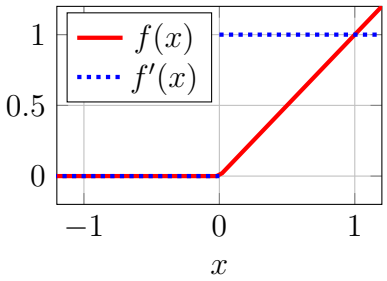
ReLU je zkratka pro Rectified Linear Unit (usměrněná linearní funkce). Tato jednoduchá funkce je dána předpisem 3.24, její derivace předpisem 3.25 (na následující straně). Obor hodnot je $H(f) : y \in \langle 0, \infty)$.

$$f(x) = \begin{cases} 0 & \text{pro } x \leq 0 \\ x & \text{pro } x > 0 \end{cases}, \text{ nebo také } f(x) = max(0, x) \tag{3.24}$$

$$f'(x) = \begin{cases} 0 & \text{pro } x \leq 0 \\ 1 & \text{pro } x > 0 \end{cases} \tag{3.25}$$

V současnosti je ReLU obecně doporučovanou výchozí volbou při výběru aktivační funkce (pro skryté neurony). [27] [45, s. 171]

Při porovnání ReLU s tanh na identické síti bylo pozorováno, že s ReLU dochází k 6násobnému zrychlení učení. [47, s. 3] Výhodou ReLU může být také nižší výpočetní náročnost na rozdíl od sigmoidy nebo funkce tanh, kde se počítá s exponenty. [27]



Graf 3.4: Funkce ReLU

Nevýhodou funkce ReLU je, že není spojitě diferencovatelná – neexistuje zde derivace když $x = 0$. V praxi je toto řešeno předpokladem, že $f'(0) = 0$. [46]

Větším problémem je tzv. „dying ReLU“, tedy že neurony s aktivační funkcí ReLU mohou „zemřít“. To znamená, že například příliš velký gradient může způsobit takovou změnu váhy, že se neuron už nebude schopen nikdy aktivovat. [27].

Pro předejití tomuto problému vznikají různé varianty ReLU. Pokud v ANN dochází k „umírání“ neuronů, je vhodné zvolit nějakou z variant této funkce.

Leaky ReLU

Jednou variantou, která pomáhá řešit problém „dying ReLU“ je funkce **leaky ReLU**. Ta se liší v tom, že pro že pro hodnoty $x < 0$ má funkce konstantní hodnotou daný malý sklon (např. 0.01). [27]

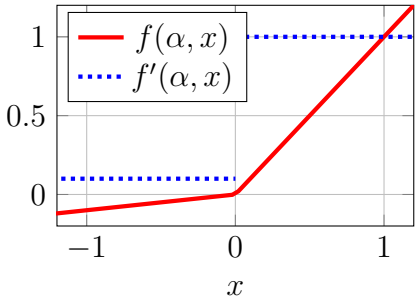
PReLU

Obecná verze, ze které leaky ReLU vychází, je funkce **PReLU** (parametric ReLU). Je dána předpisem 3.26, derivace této funkce předpisem 3.27. Obor hodnot je pak $H(f) : y \in (-\infty, \infty)$, pokud platí, že $\alpha > 0$.

$$f(\alpha, x) = \begin{cases} \alpha x & \text{pro } x < 0 \\ x & \text{pro } x \geq 0 \end{cases}, \text{ nebo také } f(x) = max(\alpha x, x) \tag{3.26}$$

$$f'(\alpha, x) = \begin{cases} \alpha & \text{pro } x < 0 \\ 1 & \text{pro } x \geq 0 \end{cases} \tag{3.27}$$

Místo konstantní hodnoty pro určení sklonu tedy využívá parametr α, který je možné učit (učením upravovat). Použitím PReLU místo ReLU a učením tohoto parametru bylo pozorováno mírné zvýšení přesnosti při klasifikaci obrázků bez téměř žádného zvýšení výpočetní zátěže. [48]



Graf 3.5: Funkce PReLU s parametrem $\alpha = 0.1$

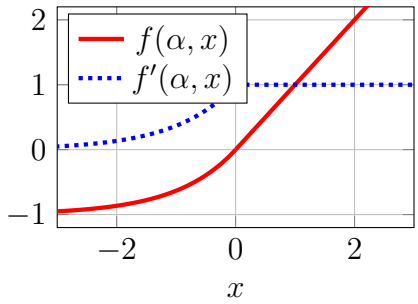
3.5.5 ELU

Exponential linear unit (ELU) je funkce podobná ReLU (PReLU). Je definovaná předpisem 3.28, její derivace vyjádřena na předpisu 3.29 na následující straně. Obor hodnot je $H(f) : y \in (-\alpha, \infty)$.

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{pro } x \leq 0 \\ x & \text{pro } x > 0 \end{cases} \tag{3.28}$$

$$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{pro } x \leq 0 \\ 1 & \text{pro } x > 0 \end{cases} \tag{3.29}$$

Liší se v hodnotách pro $x \leq 0$, kde má hladký průběh. Parametr α je stejně jako u PReLU možné učit.



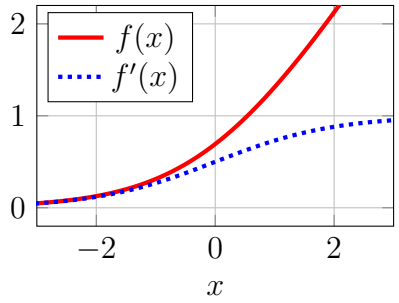
Graf 3.6: Funkce ELU s parametrem $\alpha = 1.0$

Také bylo ukázáno, že ELU dokáže v některých případech urychlit učení a poskytnout vyšší přesnost při klasifikaci než ReLU. [49]

3.5.6 Softplus

Funkce softplus bývá někdy nazývána také jako SmoothRelu, protože se v podstatě jedná o hladkou aproximaci práčě funkce ReLU. Její definice a derivace jsou uvedeny na předpisu 3.30, obor hodnot je $H(f) : y \in (0, \infty)$. Je možné vidět, že derivací této funkce je sigmoida.

$$f(x) = \ln(1 + e^x) \qquad f'(x) = \frac{1}{1 + e^{-x}} \tag{3.30}$$



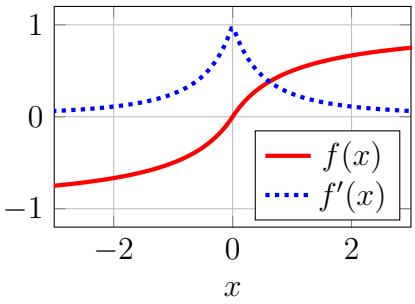
Graf 3.7: Softplus

3.5.7 Softsign

Funkce softsign s definicí a derivací uvedenými na předpisu 3.31 a oborem hodnot $H(f) : y \in (-1, 1)$ je podobná funkci hyperbolický tangens.

$$f(x) = \frac{x}{1 + |x|} \qquad f'(x) = \frac{1}{(1 + |x|)^2} \tag{3.31}$$

Na rozdíl od funkce tanh se ale softsign přibližuje pomaleji ke svým asymptotám. [50, s. 251]



Graf 3.8: Softsign

3.5.8 Softmax

Softmax je funkce (předpis 3.32), která převede reálné hodnoty na pravděpodobnosti. Přesněji převede vektor n reálných čísel $(x_1, x_2, ..., x_n)$ na (vektor) rozdělení pravděpodobnosti, které obsahuje n hodnot. [51]

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \tag{3.32}$$

Softmax se využívá při klasifikaci, kde jednotlivé pravděpodobnosti vyjadřují jistotu správné odpovědi.

Pro jednoduchou demonstraci fungování následuje příklad podle [51]:
Pro čísla -1, 0, 3 a 5 platí, že

$$\sum_{j=0}^{n} e^{x_j} = e^{-1} + e^{0} + e^{3} + e^{5} = 169.87$$

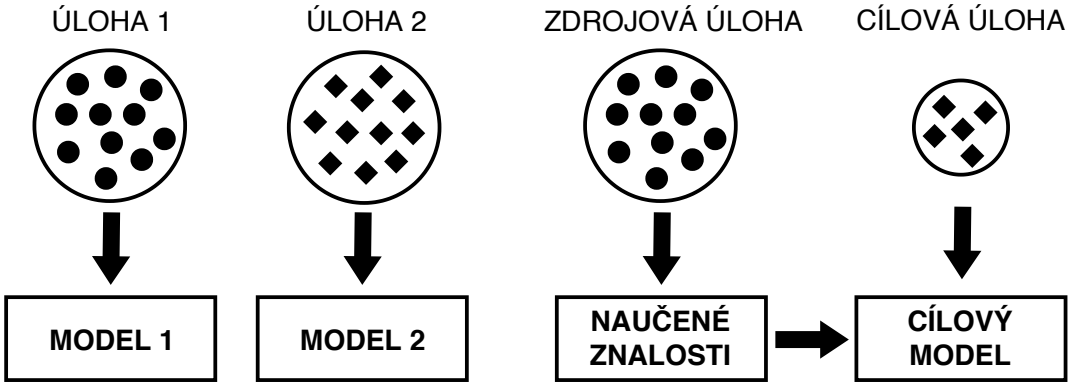
To znamená:

$f(-1) = \frac{e^{-1}}{169.87} = \frac{0.37}{169.87} = 0.002$
$f(0) = 0.006$
$f(3) = 0.118$
$f(5) = 0.874$

Součet všech pravděpodobností je 1. Čím větší hodnota x, tím vyšší pravděpodobnost.

4 Transfer learning

Transfer learning (přenesené učení nebo přenos učení, zkratka **TL**) je metoda strojového učení, která dokáže zlepšit učení nové (cílové) úlohy pomocí přenosu znalostí z jiné (příbuzné) a již naučené (zdrojové) úlohy. Lidé využívají přenosu znalostí z jedné úlohy na jinou (novou) přirozeně a čím více nová úloha souvisí s tou předešlou, tím snazší je se tuto novou úlohu naučit. [52] Příkladem může být učení se cizích jazyků. Při učení se nového jazyka člověk přirozeně využívá základních znalostí, které se naučil již dříve. Má povědomí o existenci nějaké abecedy, pravidel gramatiky apod. – nemusí se tedy učit vše od začátku.

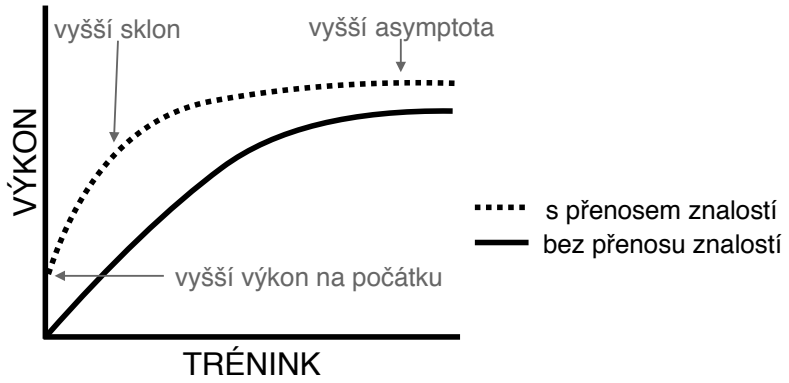


(a) Tradiční způsob strojové učení **(b)** Transfer learning

Obrázek 4.1: Srovnání ML a TL (zdroj: autor podle [53, s. 1346])

Existují tři základní míry, které mohou ukazovat na zlepšení učení (znázorněno na obrázku 4.2 na následující straně). Vždy jde o porovnání učení s využitím přenosu znalostí a učení tzv. „od nuly“ [52]:

- dosažitelná počáteční úroveň výkonu při učení cílového úkolu
- čas, který je potřeba pro naučení se cílového úkolu
- dosažitelná konečná úroveň výkonu při učení cílovém úkolu



Obrázek 4.2: Tři způsoby, jak může přenos znalostí zlepšit učení (zdroj: [52, s. 2], překlad: autor)

4.1 Formální definice

(tato část je zpracována podle [53, s. 1346-1347])

Nejprve je třeba definovat dva základní koncepty – **doménu** (domain) a **úlohu** (task). Doména $\mathcal{D} = \{\mathcal{X}, P(X)\}$ se skládá z prostoru vstupů (feature space) $\mathcal{X}$ a marginálního rozdělení pravděpodobnosti $P(X)$, kde $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Úloha $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ skládá ze dvou částí, kde $\mathcal{Y}$ je prostor výstupů (label space) a $f(\cdot)$ je prediktivní funkce, kterou možné naučit z trénovacích dat obsahující dvojice $x_i \in X$ a $y_i \in \mathcal{Y}$. Tato funkce může být využita pro předpověď výsledku $f(x)$ pro novou hodnotu x a může být zapsána také jako $P(y|x)$.

Například pro klasifikaci textových dokumentů je $\mathcal{X}$ prostorem všech dokumentů, x_i je i-tý vektor termínů[1] odpovídající nějakému dokumentu a X je trénovací vzorek dokumentů. Pro binární klasifikaci je $\mathcal{Y}$ množina výstupních hodnot 0,1 (true, false), y_i je 0 nebo 1 (true nebo false).

Pokud je dána zdrojová doména $\mathcal{D}_S$ a k ní zdrojová úloha $\mathcal{T}_S$, a také cílová doména $\mathcal{D}_T$ a k ní cílová úloha $\mathcal{T}_T$, pak je cílem TL zlepšit učení cílové prediktivní funkce $f_T(\cdot)$ v $\mathcal{D}_T$ pomocí znalostí v $\mathcal{D}_S$ a $\mathcal{T}_S$, kde $\mathcal{D}_S \neq \mathcal{D}_T$ nebo $\mathcal{T}_S \neq \mathcal{T}_T$.

Pokud pro domény platí, že $\mathcal{D}_S \neq \mathcal{D}_T$, tak to znamená že $\mathcal{X}_S \neq \mathcal{X}_T$ nebo $P_S(X) \neq P_T(X)$. Na příkladu klasifikace textových dokumentů to znamená, že jsou dokumenty ve zdrojové a cílové sadě například napsané v odlišných jazycích nebo diskutují jiná témata.

Podobně platí pro úlohy, že pokud $\mathcal{T}_S \neq \mathcal{T}_T$, tak $\mathcal{Y}_S \neq \mathcal{Y}_T$ nebo $P(Y_S|X_S) \neq P(Y_T|X_T)$. Při klasifikaci dokumentů to znamená, že mají dokumenty ve zdrojové a cílové doméně jiné kategorie, například zdrojová doména a cílová doména používají rozdílné kategorie, nebo jsou kategorie mezi doménami nevyvážené.

[1] term vector – obsah dokumentů je reprezentován např. jako vektor všech slov

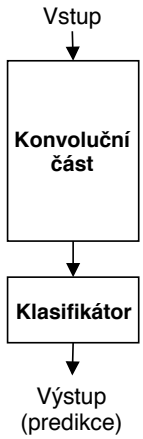
4.2 Použití přenosu učení

Jednou z nejpopulárnějších oblastí strojového učení, kde se v současnosti využívá TL, jsou hluboké neuronové sítě, přesněji hluboké konvoluční neuronové sítě.

Trénování velkých konvolučních neuronových sítí může trvat i s využitím nejvýkonnějších grafických karet několik dnů nebo dokonce týdnů, a navíc je potřeba dostatečně velká množina trénovacích dat. Sestavení velké množiny trénovacích dat je nejen náročně, ale někdy i nemožné. Často totiž bývá k dispozici jen malé množství obrazových dat využitelných pro trénování – příkladem mohou být medicínské snímky, fotografie vzácných živočichů apod.

Oba tyto problémy dokáže řešit právě TL použitím před-trénovaného modelu. Pokud je takový model trénovaný na velké a dostatečně obecné datové sadě, může pak dobře posloužit jako obecný vizuální model pro TL [54].

Jak již bylo zmíněno v části 3.3.4 Konvoluční neuronové sítě, CNN určené pro klasifikaci obrázků se skládají ze dvou částí – **konvoluční části** (konvolučního základu) a **klasifikační části** (klasifikátoru) – znázorněno na obrázku 4.3. [55]



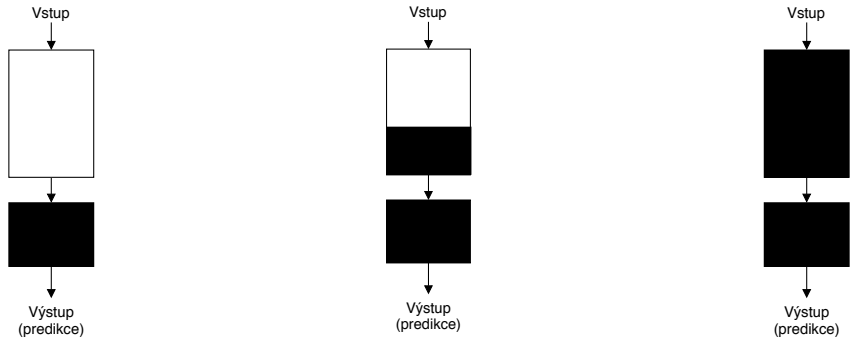
Obrázek 4.3: Zjednodušená reprezentace složení CNN pro klasifikaci obrazu (zdroj: [55])

Základním krokem pro aplikování TL na klasifikaci obrázků je volba modelu a nahrazení klasifikační části vlastním klasifikátorem. Díky tomu bude nový model klasifikovat podle požadovaných tříd, a ne tříd původního modelu.

Poté následuje volba jedné ze tří strategií pro trénování a ladění modelu [55]:

- **Trénování pouze klasifikační části sítě** – je základní strategie (schématicky znázorněna na obrázku 4.4a na následující straně), kdy dochází k trénování pouze klasifikační části sítě, vrstvy v konvoluční částí jsou tzv. „zmrazené“ (frozen) a nedochází v nich tedy k změnám vah (učení). Tato strategie je vhodné zejména pokud je k dispozici pouze malá datová sada a/nebo pokud daný před-trénovaný model řeší klasifikaci podobného typu obrázků. Často se tento způsob využití označuje také jako *feature extraction*.
- **Trénování některých vrstvev konvoluční části sítě** – zde dochází i k trénování některých vrstev v konvoluční části sítě. Je tedy třeba dobře zvolit a případně testovat, které vrstvy v konvoluční části je vhodné ještě trénovat. Schématicky je tato strategie znázorněna na obrázku 4.4b na následující straně. Vhodná je v případech, kdy je k dispozici malá a odlišná datová sada od datové sady před-trénovaného modelu nebo velká a podobná datová sada.

- **Trénování celé sítě** – tato strategie (znázorněna na obrázku 4.4c na následující straně) se někdy označuje také jako inicializace vah (weights initialization). Dochází zde k trénování celé sítě, váhy před-trénované konvoluční části slouží právě jako počáteční bod. Aby nedocházelo k velkým změnám v konvoluční části sítě, a tedy ztrátě předchozích znalostí, používá se pro učení této části nízká míra učení. Tato strategie je vhodná, pokud je datová sada odlišná od datové sady před-trénovaného modelu.



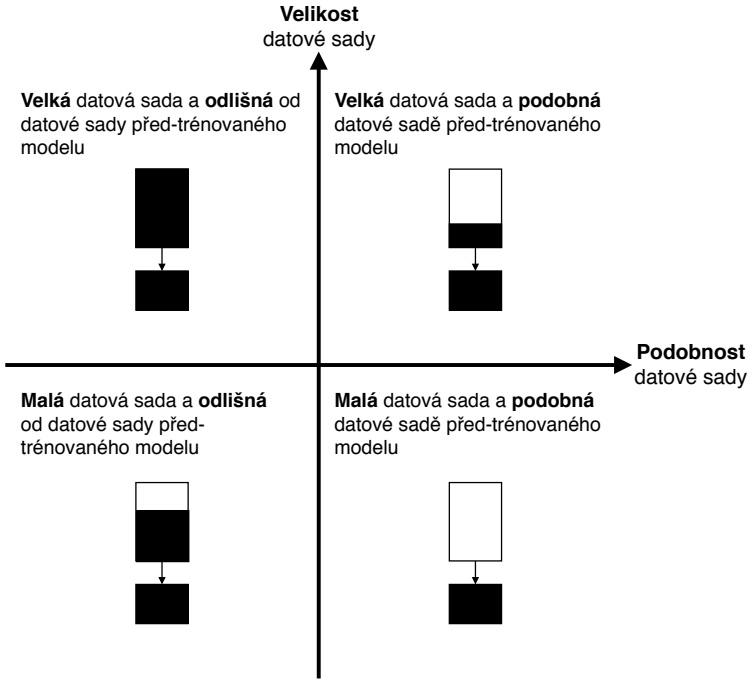
(a) Trénování pouze klasifikátoru

(b) Částečné trénování konvoluční části

(c) Trénování obou částí sítě (celé sítě)

Obrázek 4.4: Strategie využití TL při klasifikaci obrazu. Černá barva znázorňuje trénované části sítě, bíla barva ukazuje „zmrazené“ části sítě (zdroj: podle [55])

Volbu strategie podle velikosti datové sady a její podobnosti s datovou sadou před-trénovaného modelu shrnuje následující obrázek.



Obrázek 4.5: Matice zobrazující výběr volby strategie podle velikosti a podobnosti datové sady (zdroj: podle [55])

Využití TL pochopitelně není limitováno pouze na konvoluční neuronové sítě. Další oblastí, kde se ještě TL běžně využívá je například zpracování přirozeného jazyka (NLP).

4.3 Před-trénované modely

Základním předpokladem pro TL je existence před-trénovaných modelů, které dosahují vysoké přesnosti ve zdrojové úloze. Naštěstí existuje mnoho „state of the art" před-trénovaných modelů, na jejichž vývoji se podílí celé týmy expertů.[56]

Každý model (nebo skupina modelů) má určité využití, některé mohou sloužit pro klasifikaci obrázků, jiné pro detekci objektů v obrázku nebo například pro převod textu na řeč, analýzu sentimentu atd. Tyto modely bývají snadno dostupné přímo pomocí nástrojů jako je Keras.

Mezi nejznámější modely (skupiny modelů) pro zpracování obrazu (počítačové vidění) patří například:

- **AlexNet** – `https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`
- **VGGNet** – `https://arxiv.org/abs/1409.1556`
- **ResNet** – `https://arxiv.org/abs/1512.03385`
- **GoogLeNet** (Inception V3) – `https://arxiv.org/abs/1512.00567`
- **MobileNet** – `https://arxiv.org/abs/1704.04861`
- **EfficientNet** – `https://arxiv.org/abs/1905.11946`

Tyto modely byly trénovány na obrovských datových sadách, které obsahují miliony již označených (roztříděných) obrázků. Jednou z nejznámějších velkých datových sad je ImageNet[2]. Po dobu několika let byla každý rok pořádána soutěž Large Scale Visual Recognition Challenge (ILSVRC), která zahrnovala klasifikaci obrázků a detekci objektů na obrázcích z datové sady ImageNet. V roce 2012 tuto soutěž (v klasifikaci obrázků) vyhrála právě síť AlexNet a to odstartovalo následný vzrůst zájmu o konvoluční neuronové sítě.

V oblasti NLP jsou pak známé modely:

- **Word2Vec** – `https://arxiv.org/abs/1310.4546`
- **BERT** – `https://arxiv.org/abs/1810.04805`
- **ULMFiT** – `https://arxiv.org/abs/1801.06146`
- **GloVe** – `https://nlp.stanford.edu/projects/glove`

[2] `http://www.image-net.org/`

5 Klasifikace českých mincí

Tato kapitola ukazuje využití před-trénovaného modelu a augmentace datové sady na vytvoření modelu pro klasifikaci (standardních) českých mincí. Pro tuto úlohu byla vytvořena datová sada, která je v této kapitole také popsána.

5.1 Použité nástroje

Pro vývoj a trénování modelu bylo využito prostředí **Colaboratory** a knihovna **TensorFlow 2.0** (viz 2.6 Software, knihovny a nástroje). I přes všechny limity uvedené v kapitole 2.6 bylo prostředí Colaboratory dostačující pro tuto úlohu, navíc umožňuje následně celý kód komukoliv snadno sdílet a spouštět. TensorFlow je prakticky standardem v oblasti ML, je dobře zdokumentován a nabízí snadný převod modelu do jiných formátů (například pro web).

Datová sada je sdílená na Kaggle.com (viz 5.2 Datová sada). Pro snadné rozdělení datové sady byla využita knihovna **split-folders**[1]. Pro augmentaci datové sady byla použita knihovna **imgaug**[2], která nabízí pokročilé možnosti konfigurace a lze snadno použít s knihovnou TensorFlow (viz 5.3.1 Augmentace datové sady).

5.2 Datová sada

V rámci této práce byla vytvořena datová sada českých mincí, která je veřejně dostupná na **`https://www.kaggle.com/janstol/czech-coins`**.

Tato datová sada obsahuje 6 tříd (pro 1, 2, 5, 10, 20 a 50 Kč), jednotlivé obrázky jsou rozděleny do odpovídajících samostatných adresářů (01czk, 02czk, ... 50czk). Každá třída obsahuje přibližně 300 obrázků mincí o velikosti 640x640 px. Mince byly zachyceny pomocí fotoaparátu v mobilním telefonu, každá mince je zachycena z obou stran, na různých podkladech, z různých úhlů a za různých světelných podmínek.

[1] `https://github.com/jfilter/split-folders`
[2] `https://github.com/aleju/imgaug`

V datové sadě nejsou zahrnuty mince: 10 Kč – vzor 2000[3], 20 Kč – vzor 2000[4] a také 20 Kč – vzory 2018[5] a 2019[6].

5.3 Vývoj a trénování modelu

Při vývoji a trénování modelu bylo využito technik TL (viz 4 Transfer learning) a také augmentace datové sady (viz dále).

V případě klasifikace mincí dává smysl využít výsledný model v mobilní nebo webové aplikaci, obecně na zařízeních s nižším výkonem, omezenou výdrží akumulátoru apod.

S ohledem na tuto skutečnost byl jako základ pro vývoj nového modelu zvolen model **mobilenet_v2_1.0_224**. Jedná se o základní verzi modelu ze skupiny **MobileNetV2**, který byl trénován na datové sadě ImageNet a pracuje se vstupními obrázky o velikosti 224x224 px. Modely MobileNet(V2) jsou optimalizovány pro mobilní zařízení – mají nižší počet parametrů, dosahují nižší výsledné velikosti, ale také nižší přesnosti. Porovnání několika vybraných modelů je uvedeno v tabulce 5.1

Tabulka 5.1: Porovnání některých před-trénovaných modelů. Top-1 a Top-5 ukazuje přesnost modelu na validační datové sadě ImageNet. (zdroj: [57])

Model	Velikost	Top-1	Top-5	Počet parametrů
VGG 19	549 MB	0.713	0.900	143 667 240
ResNet152V2	232 MB	0.780	0.942	60 380 648
Xception	88 MB	0.790	0.945	22 910 480
InceptionV3	92 MB	0.779	0.937	23 851 784
MobileNetV2	14 MB	0.713	0.901	3 538 984
DenseNet201	80 MB	0.773	0.936	20 242 984
NASNetMobile	23 MB	0.744	0.919	5 326 716

Všechny tyto modely jsou snadno dostupné pomocí Keras API, které je součástí TensorFlow 2.x, viz 5.3.2 Sestavení modelu.

Kompletní zdrojový kód je dostupný v souboru `czk_coins_classification.ipynb` v elektronické příloze k této práci. Pro práci s tímto souborem je nutné ho nahrát do prostředí Colaboratory (vyžaduje Google účet). Některé důležité části jsou více popsány dále.

[3] `https://www.cnb.cz/cs/bankovky-a-mince/mince/10-kc-vzor-2000`
[4] `https://www.cnb.cz/cs/bankovky-a-mince/mince/20-kc-vzor-2000`
[5] `https://www.cnb.cz/cs/bankovky-a-mince/mince/20-kc-vzor-2018-i`
[6] `https://www.cnb.cz/cs/bankovky-a-mince/mince/20-kc-vzor-2019-i`

5.3.1 Augmentace datové sady

Augmentace neboli rozšíření datové sady je způsob, jak uměle zvětšit existující datovou sadu a předcházet tak například problému přeučení (viz 2.4 Přeučení a nedoučení). Obecně se jedná o přidání náhodného šumu k existujícím datům čímž prakticky vzniknou data nová.

V případě obrázků se používá rotace, posunutí, převrácení, přiblížení/oddálení, změnu jasu a kontrastu, přidání efektů jako je rozmazání nebo šum atd. Je ale nutné volit použité úpravy s ohledem na danou datovou sadu.

Pro generování upravených obrázků existuje v TensorFlow 2.x třída **ImageDataGenerator** (tf.keras.preprocessing.image.ImageDataGenerator). Tato třída nabízí ale pouze základní transformace jako je překlopení, rotace, posunutí apod. Chybí zde pokročilejší úpravy, například možnost přidat šum nebo rozmazání. Zároveň je možné pomocí parametru `preprocessing_function` předat funkci, která bude úpravy provádět. Díky tomu je snadné použít knihovnu imgaug, která nabízí pokročilou konfiguraci augmentace obrázků.

```
1  import imgaug.augmenters as iaa
2  import tensorflow as tf
3
4  aug = iaa.Sequential([
5      iaa.Fliplr(0.5),
6      iaa.Affine(scale=(0.9, 1.5), rotate=(-180, 180)),
7      iaa.OneOf([
8        iaa.Dropout(p=(0.01, 0.05)),
9        iaa.GammaContrast((0.8, 1.2)),
10     ]),
11 ], random_order=True)
12
13 datagen = tf.keras.preprocessing.image.ImageDataGenerator(
14     preprocessing_function=aug.augment_image
15 )
16
17 train_generator = datagen.flow_from_directory(
18     "/path/to/train-data", target_size=(224,224),
19     batch_size=64
20 )
```

Ukázka kódu 5.1: Příklad použití knihovny imgaug

Ukázka kódu 5.1 ilustruje použití knihovny imgaug pro nastavení augmentace. U poloviny obrázků bude provedeno horizontální překlopení, každý obrázek bude rotován

v rozsahu -180° až 180°, dojde k změně měřítka mezi 90 a 150 % původní velikosti a také bude použita jedna z transformací Dropout nebo GammaContrast. Tyto úpravy jsou díky parametru `random_order` jsou prováděny v náhodném pořadí.

Vyvolání úpravy obrázku se provádí zavoláním metody `augment_image(img)`. Toto volání lze předat přímo jako argument třídě ImageDataGenerator (řádek 17), která bude tuto metodu volat automaticky během trénování.

Uvedený kód je zatím v podstatě jen konfigurace augmentace a generování. Nakonec je tedy třeba ještě zavolat metodu `flow_from_directory()` (řádek 19). Pomocí parametrů se předá cesta k adresáři s trénovací datovou sadou, velikost obrázků a velikost jedné dávky. Také je možné specifikovat např. jestli má dojít k zamíchání dat, jaký použít seed apod. Tato metoda pak vrací iterátor/generátor, který vrací dávku obrázků a k nim odpovídající označení třídy.

Obrázek 5.1: Ukázka augmentace obrazových dat. V prvním řádku jsou originály, druhý řádek ukazuje jejich vygenerovanou úpravu. (zdroj: autor)

Generování obrázků pomocí třídy ImageDataGenerator probíhá za běhu, upravené obrázky není třeba ukládat. Obdobně je vytvořen generátor validačních dat, u kterých se ale obecně augmentace nepoužívá.

5.3.2 Sestavení modelu

Jak již bylo výše uvedeno, pomocí Keras API v TensorFlow 2.x jsou snadno dostupné některé modely, včetně modelu MobileNetV2, který je použit v této práci. Jeho stažení je ukazuje následující kód.

```
1 base_model=tf.keras.applications.MobileNetV2(
2     input_shape=(224, 224),
3     alpha=1.0,
4     include_top=False,
5     weights='imagenet'
6 )
```

Ukázka kódu 5.2: Stažení modelu pomocí Keras API v TensorFlow 2.x

Parametr `include_top=False` udává, že bude stažen model bez vrchní vrstvy (klasifikátoru) a je tedy nutné dodat vlastní klasifikátor, například jako je to uvedeno v ukázce kódu 5.3.

```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(6, activation='softmax')])
```

Ukázka kódu 5.3: Sestavení modelu

Zde je použita například i vrstva Dropout, která v tomto případě náhodně deaktivuje 20 % neuronů. To by mělo pomoci předcházet problému s přeučením, jak bylo zmíněno v 2.4 Přeučení a nedoučení). Poslední vrstva obsahuje 6 výstupů pro 6 tříd mincí a aktivační funkci softmax, díky které jsou na výstupech pravděpodobnosti, které vyjadřují jistotu správné predikce (viz 3.5.8 Softmax).

K předtrénovanému modelu (`base_model`) je také důležité zmínit možnost ladění pomocí tzv. odmrazování/zmrazování vrstev – viz 4.2 Použití přenosu učení. Pomocí `base_model.trainable = False` lze všechny vrstvy před-trénovaného modelu zmrazit, to znamená že trénovány budou pouze vrstvy klasifikátoru. Přes jednotlivé vrstvy modelu lze také iterovat a nastavovat tak jednotlivé vrstvy. Ukázka kódu 5.4 uvádí příklad, kdy dojde k zmrazení pouze prvních 10 vrstev (základního) modelu.

```
base_model.trainable = True
for layer in base_model.layers[:10]:
    layer.trainable = False
```

Ukázka kódu 5.4: Zmrazování vrstev modelu

Nakonec je třeba celý model pomocí metody `compile()` zkompilovat (viz ukázka kódu 5.5). Parametr `optimizer` specifikuje optimalizátor a míru učení, `loss` ztrátovou funkci a `metrics` metriky podle kterých se hodnotí model v průběhu trénování a testování.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)
```

Ukázka kódu 5.5: Kompilace modelu

5.3.3 Trénování a ladění modelu

Když je model zkompilovaný, je možné zahájit trénování. V tomto případě, kdy byl použit generátor se trénování spouští zavoláním metody `fit_generator()`.

```
model.fit_generator(
    train_generator,
    validation_data=val_generator,
    epochs=10,
)
```

Ukázka kódu 5.6: Zahájení trenování modelu

Tato metoda také nabízí různé parametry, pomocí kterých lze modifikovat chování při trénování modelu. Nutné je předat generátor trénovacích dat (popsáno v 5.3.1 Augmentace datové sady), žádoucí je také generátor validačních dat (předaný pomocí parametru `validation_data`) a počet epoch (`epochs`).

Užitečný je parametr `callbacks`, kam je možné předat tzv. callbacky. K jejich vykonání dochází v určitých bodech při trénování (například po konci každé epochy). TensorFlow nabízí několik existujících callbacků – například TensorBoard callback umožňuje vizualizovat průběh trénování v nástroji TensorBoard, EarlyStopping umožňuje předčasně ukončit trénování, pokud již nedochází k zlepšování apod. Existuje zde také možnost vytvářet vlastní callbacky, jeden takový je použit i v této práci – ConfusionMatrixCallback pro logování matice záměn.

Pomocí parametru `initial_epoch` lze specifikovat od které epochy má začít trénování. To je užitečné při pokračování v trénování z určitého (předchozího) bodu.

V této práci došlo k využití výše uvedených možností a byl implementován vlastní proces trénování a ladění modelu. Celý proces se nastavuje pomocí jednoduché konfigurace uvedené v ukázce kódu 5.7

```
FINE_TUNING_SETUP = [
  {"epochs": 15, "lr": 1e-3, "fine_tune_from": -1},
  {"epochs": 100, "lr": 1e-4, "fine_tune_from": 10},
]
```

Ukázka kódu 5.7: Konfigurace pro trénování a ladění modelu

Jedná se o klasické pole (seznam), kde jednotlivé prvky tvoří slovníky. Každý slovník představuje nastavení pro určitý krok. Klíč `epochs` značí počet epoch pro daný krok, `lr` udává míru učení pro daný krok a `fine_tune_from` nastavuje které vrstvy v před-trénovaném modelu budou zmrazené, kde -1 znamená zmrazení všech vrstev v tomto modelu.

Výše uvedený příklad znamená, že by nejdříve došlo k trénování pouze klasifikátoru po dobu 15 epoch, poté by by bylo zmrazeno jen prvních 10 vrstev základního

modelu a trénování by pokračovalo dalších 100 epoch.

Tuto konfiguraci zpracovává kód uvedený v ukázce kódu 5.8. Některé části kódu jsou záměrně vynechány pro zachování stručnosti. Kompletní zdrojový kód je dostupný v elektronické příloze k této práci, jak již bylo uvedeno dříve.

```
epochs = 0
for i, setup in enumerate(FINE_TUNING_SETUP):
    initial_epoch = epochs
    epochs = epochs + setup["epochs"]
    learning_rate = setup["lr"]
    fine_tune_from = setup["fine_tune_from"]

    if fine_tune_from == -1:
        base_model.trainable = False
    else:
        base_model.trainable = True
        for layer in base_model.layers[:fine_tune_from]:
            layer.trainable = False

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate),
        # loss, metrics
    )

    if fine_tune_from < 10:
        callbacks.append(early_stopping_callback)

    model.fit_generator(
        train_generator,
        validation_data=val_generator,
        epochs=epochs, initial_epoch=initial_epoch,
        callbacks=callbacks
    )
```

Ukázka kódu 5.8: Zpracování konfigurace a trénování modelu

Jedná se o jednoduchý `for` cyklus, pomocí kterého je konfigurace postupně procházena. V každém kroku dojde k nastavení vrstev v základním modelu, celý model se znovu zkompiluje (s případnou novou mírou učení) a spustí se trénování.

Ve výše uvedeném případě ještě za určité podmínky dochází k přidání callbacku pro předčasné zastavení učení, pokud již nedochází k dostatečným změnám.

Po kompletním dokončení trénování je možné model jednoduše uložit:

```
1 model.save("path-to-dir/saved_model", save_format="tf")
```

Ukázka kódu 5.9: Uložení modelu do formátu SavedModel

Takto uložený model lze pak zpětně nahrát a vrátit se například k trénování nebo ho převést do jiného formátu (např. pro mobilní aplikaci – TensorFlow Lite nebo web – TensorFlow.js).

6 Shrnutí výsledků

V rámci této práce byla vytvořena malá datová sada obsahující fotografie českých mincí (viz 5.2 Datová sada). S využitím některých, v této práci dříve uvedených, postupů byl na této datové sadě natrénován model umělé neuronové sítě, který klasifikuje české mince.

Byly vyzkoušeny různé konfigurace, tři vybrané jsou uvedeny v elektronické příloze v souboru `czk_coins_classification.ipynb`, dále jsou k této práci přiloženy logy a uložené modely. Jedna (nejlepší) konfigurace je dále popsána.

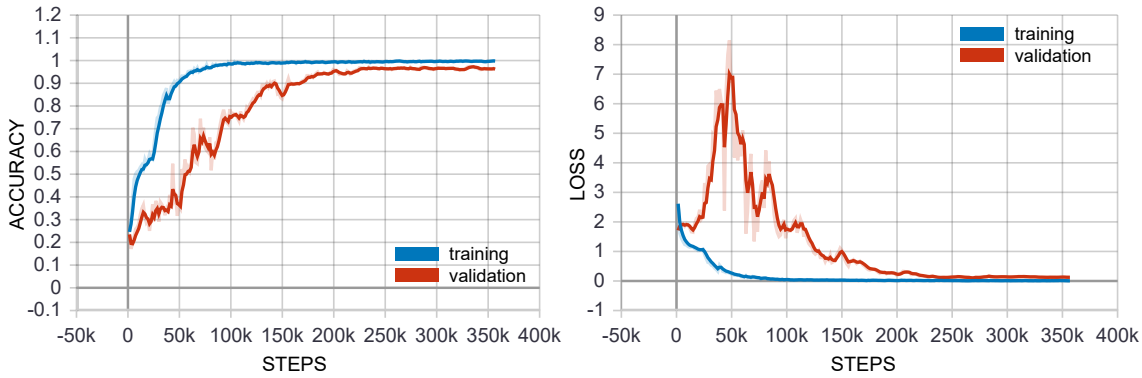
S konfigurací, která je označená jako `czk01` (tabulka 6.1) bylo dosaženo **96%** přesnosti modelu. Je ale nutné zdůraznit, že tato přesnost byla dosažena na (malé) **validační datové sadě**. U posledních dvou kroků této konfigurace bylo také použito předčasné zastavení s argumenty `monitor='val_loss', min_delta=1e-2` a `patience=20`. V předposledním kroku konfigurace bylo trénování automaticky přerušeno u 93. epochy (celkem 198. z 205), v posledním kroku u 40. epochy (celkem 245. z 305). Je předpokládáno, že použitý před-trénovaný model MobileNetV2 nebyl trénován na obrázcích mincí a tak byly trénovány i hlubší vrstvy tohoto modelu.

Tabulka 6.1: Konfigurace trénování modelu – `czk01`

epochs	learning rate	fine_tune_from
15	1e-3 (0.001)	-1
10	3e-4 (0.0003)	100
10	5e-5 (0.00005)	60
20	3e-4 (0.0003)	30
50	7e-5 (0.00007)	10
100	3e-5 (0.00003)	3
100	2e-5 (0.00002)	0

Na učících křivkách (obrázky 6.1a, 6.1b) je možné pozorovat průběh učení. Vzhledem k tomu, že po rozdělení datové sady zbylo v trénovací množině přibližně 1500 obrázků a velikost dávky (batch size) byla nastavena na 100, tak bylo pro každou epochu provedeno $1500/100 = 15$ kroků (neplést s kroky v konfiguraci). To také znamená, že jedna epocha odpovídá 1500 krokům (steps) na těchto grafech učících křivek.

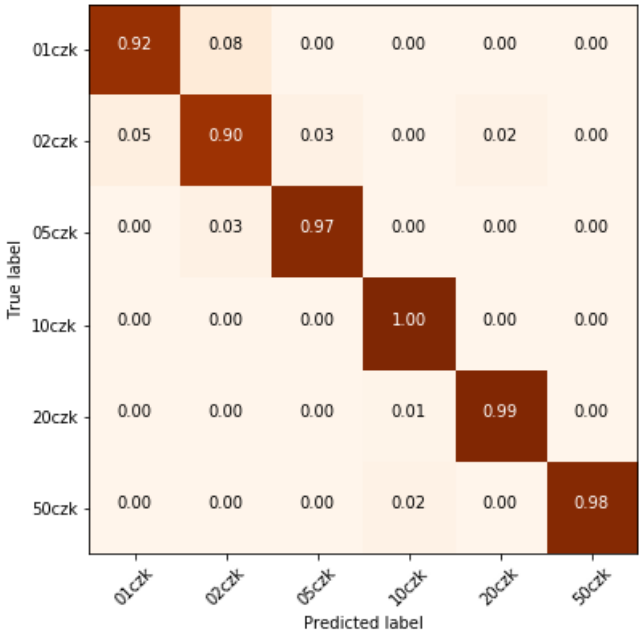
Je zde vidět, že pro parametr `patience` byla zvolena pravděpodobně příliš vysoká hodnota a trénování tak zbytečně pokračovalo dále i když už prakticky nedocházelo k dalšímu zlepšování ztráty. Navíc by už nemusel být prováděn ani poslední krok v konfiguraci (na obrázku přibližně krok 300k).



(a) Trénovací a validační přesnost **(b)** Trénovací a validační ztráta

Obrázek 6.1: Učící křivky pro učení s konfigurací `czk01` (zdroj: autor)

Na dalším obrázku (6.2) je matice záměn. Na ní je vidět, že k záměnám (nesprávné klasifikaci) docházelo nejvíce mezi korunami a dvoukorunami. To vzhledem k stejné barvě a podobnosti těchto mincí z lícní strany není úplně překvapivé.



Obrázek 6.2: Matice záměn – natrénovaný model (zdroj: autor)

Nakonec byl také proveden malý test klasifikace na obrázcích mimo datovou sadu (viz obrázek 6.3). Model byl schopen správně klasifikovat i verze 10 Kč a 20 Kč, které nebyly v datové sadě.

Obrázek 6.3: Test klasifikace na obrázcích mimo datovou sadu. Pod jednotlivými obrázky jsou uvedeny pravděpodobnosti s názvy tříd. (zdroj: viz níže)

Zdroje obrázků:

- `https://www.publicdomainpictures.net/en/view-image.php?image=27676&picture=czech-money`
- `https://pixabay.com/photos/coins-crown-czech-macro-2420840`
- `https://www.leftovercurrency.com/exchange/czech-koruna/czech-koruna-coins/10-czech-koruna-coin-commemorative`
- `https://www.leftovercurrency.com/exchange/czech-koruna/czech-koruna-coins/20-czech-koruna-coin-commemorative`

7 Závěr

Práce se zabývala strojovým učením, umělými neuronovými sítěmi, metodami přeneseného učení a jejich aplikací v praxi – zejména pro klasifikaci obrazových dat.

V rámci této práce byla vytvořena malá datová sada českých mincí, která je veřejně dostupná na kaggle.com. Na této datové sadě byl následně natrénován model pro klasifikaci českých mincí. Bylo ukázáno, jak je možné využít technik přenosu učení (transfer learning) a vzhledem k velikosti datové sady byly použity metody pro její augmentaci.

Celý kód pro trénování modelu je dostupný jako tzv. notebook, který je třeba nahrát do interaktivního prostředí Colaboratory. Toto prostředí je snadno dostupné prostřednictvím internetového prohlížeče, nabízí výkonný hardware a je dostupné zdarma (nutný je pouze účet Google). Tato práce ukazuje, že i přes určité limity je v tomto prostředí možné natrénovat relativně jednoduchý klasifikační model. Tento notebook by měl být použitelný i pro klasifikaci jiných obrázků, bylo by nutné provést změny ve stahování datové sady, konfiguraci apod.

Existuje několik způsobů, jak by šlo na tuto práci navázat nebo ji zlepšit. Naprosto zřejmé je vylepšení datové sady přidáním dalších fotografií mincí, zejména těch chybějících. Vylepšit by šel také kód pro trénování, například umožnit podrobnější konfiguraci. Vhodná by byla také mobilní nebo webová aplikace (využívající natrénovaný model), na kterou již nezbyl v rámci této práce čas.

V oblasti strojového učení dochází k rychlému vývoji, proto je možné, že některé části této práce budou časem neaktuální (to se týká zejména softwarových knihoven a zdrojového kódu), budou existovat lepší modely, nové metody apod.

Seznam použité literatury

[1] Machine learning - Explore. In: *Google Trends* [online]. [cit. 2019-07-06]. Dostupné z: https://trends.google.com/trends/explore?date=2010-01-01%202019-07-01&q=%2Fm%2F01hyh_

[2] LEE, Timothy B. Why Google believes machine learning is its future. In: *Ars Technica* [online]. WIRED Media Group, c2019, 5/10/2019 [cit. 2019-07-06]. Dostupné z: https://arstechnica.com/gadgets/2019/05/googles-machine-learning-strategy-hardware-software-and-lots-of-data/

[3] Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI. HOLZINGER, Andreas et al. (Eds.). *Machine Learning and Knowledge Extraction* [online]. Cham: Springer International Publishing, 2018, 2018-08-24, s. 1-8 [cit. 2019-07-01]. Lecture Notes in Computer Science. ISBN 978-3-319-99739-1. Dostupné z: http://link.springer.com/10.1007/978-3-319-99740-7_1

[4] RUSSELL, Stuart J. a Peter NORVIG. *Artificial intelligence: a modern approach*. Third edition. Upper Saddle River, New Jersey 07458: Prentice Hall, c2010. ISBN 978-0-13-604259-4.

[5] MCCARTHY, John. What Is Artificial Intelligence?. In: *Professor John McCarthy* [online]. Stanford University, 2007 Nov 12 [cit. 2019-07-01]. Dostupné z:http://jmc.stanford.edu/articles/whatisai/whatisai.pdf

[6] MCCARTHY, John a Ed FEIGENBAUM. In Memoriam Arthur Samuel: Pioneer in Machine Learning. *AI Magazine* [online]. 1990, Fall 1990, **11**(3), 10-11 [cit. 2019-07-07]. ISSN 0738-4602. Dostupné z: http://www.aaai.org/ojs/index.php/aimagazine/article/view/840/758

[7] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* [online]. 2000, **44**(1.2), 206-226 [cit. 2019-07-01]. DOI: 10.1147/rd.441.0206. ISSN 0018-8646.

[8] MITCHELL, Tom M. *Machine Learning*. New York: McGraw-Hill, c1997. ISBN 00-704-2807-7.

[9] BROWNLEE, Jason. Supervised and Unsupervised Machine Learning Algorithms. In: *Machine Learning Mastery* [online]. c2019, March 16, 2016 [cit. 2019-06-01]. Dostupné z: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms

[10] SONI, Devin. Supervised vs. Unsupervised Learning. In: *Towards Data Science* [online]. Mar 22, 2018 [cit. 2019-06-01]. Dostupné z: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[11] What is Unsupervised Learning?. In: *Deep AI* [online]. [cit. 2019-06-01]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/unsupervised-learning

[12] DUTTA, Praneet, Chris RAWLES a Yujin TANG. Deep reinforcement learning on GCP: using hyperparameter tuning and Cloud ML Engine to best OpenAI Gym games. In: *Google Cloud Blog* [online]. December 10, 2018 [cit. 2019-06-06]. Dostupné z: https://cloud.google.com/blog/products/ai-machine-learning/deep-reinforcement-learning-on-gcp-using-hyperparameters-and-cloud-ml-engine-to-best-openai-gym-games

[13] GANDHI, Rohith. Support Vector Machine — Introduction to Machine Learning Algorithms. In: *Towards Data Science* [online]. Jun 7, 2018 [cit. 2019-06-01]. Dostupné z: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[14] SONI, Devin. Introduction to Bayesian Networks. In: *Towards Data Science* [online]. [cit. 2019-06-02]. Dostupné z: https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e

[15] BROWNLEE, Jason. How to use Learning Curves to Diagnose Machine Learning Model Performance. In: *Machine Learning Mastery* [online]. c2019, February 27, 2019 [cit. 2019-07-11]. Dostupné z: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance

[16] BROWNLEE, Jason. Overfitting and Underfitting With Machine Learning Algorithms. In: *Machine Learning Mastery* [online]. c2019, March 21, 2016 [cit. 2019-07-11]. Dostupné z: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms

[17] BROWNLEE, Jason. How to Avoid Overfitting in Deep Learning Neural Networks. In: *Machine Learning Mastery* [online]. c2019, December 17, 2018 [cit. 2019-07-11]. Dostupné z: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error

[18] LEVIATHAN, Yaniv a Yossi MATIAS. Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone. In: *Google AI Blog* [online]. May 8, 2018 [cit. 2019-08-08]. Dostupné z: https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html

[19] GIBBS, Samuel. Uber's self-driving car saw the pedestrian but didn't swerve – report. In: *The Guardian* [online]. Guardian News, c2019, Tue 8 May 2018 [cit. 2019-08-19]. Dostupné z: https://www.theguardian.com/technology/2018/may/08/ubers-self-driving-car-saw-the-pedestrian-but-didnt-swerve-report

[20] LUBIN, Gus. 'Facial-profiling' could be dangerously inaccurate and biased, experts warn. In: *Business Insider* [online]. Insider, c2019, Oct. 12, 2016 [cit. 2019-08-08]. Dostupné z: https://www.businessinsider.com/does-faception-work-2016-10

[21] ADOBE COMMUNICATIONS TEAM. Adobe Research and UC Berkeley: Detecting Facial Manipulations in Adobe Photoshop. In: *Adobe Blog* [online]. Adobe, c2018, 06-14-2019 [cit. 2019-08-10]. Dostupné z: https://theblog.adobe.com/adobe-research-and-uc-berkeley-detecting-facial-manipulations-in-adobe-photoshop

[22] CHOLLET, Francois and others. Home. *Keras Documentation* [online]. 2015 [cit. 2019-10-01]. Dostupné z: https://keras.io

[23] GOLDIEGADDE. Release TensorFlow 2.0.0. *Tensorflow GitHub* [online]. c2019, Sep 30, 2019 [cit. 2019-10-02]. Dostupné z: https://github.com/tensorflow/tensorflow/releases/tag/v2.0.0

[24] PyTorch. *Github* [online]. c2019 [cit. 2019-10-02]. Dostupné z: https://github.com/pytorch/pytorch

[25] YEGULALP, Serdar. Facebook brings GPU-powered machine learning to Python. In: *InfoWorld* [online]. IDG Communications, c2019, Jan 19, 2017 [cit. 2019-10-03]. Dostupné z: https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html

[26] Artificial Neural Network. *NVIDIA Developer* [online]. c2019 [cit. 2019-07-02]. Dostupné z: https://developer.nvidia.com/discover/artificial-neural-network

[27] Neural Networks Part 1: Setting up the Architecture. *CS231n Convolutional Neural Networks for Visual Recognition* [online]. Stanford University [cit. 2019-07-02]. Dostupné z: https://cs231n.github.io/neural-networks-1

[28] VOLNÁ, Eva. *Neuronové sítě 1* [online]. Druhé vydání. Ostrava: Ostravská univerzita v Ostravě, 2008 [cit. 2019-07-01]. Dostupné z: http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf

[29] MCCULLOCH, Warren S. a Walter PITTS. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*. 1943, 5(4), 115-133. DOI: 10.1007/BF02478259. ISSN 0007-4985. Dostupné také z: http://link.springer.com/10.1007/BF02478259

[30] WALLIS, Charles. History of the Perceptron [online]. [cit. 2019-06-20]. Dostupné z: https://web.csulb.edu/~cwallis/artificialn/History.htm

[31] WIDROW, Bernard a Michael A. LEHR. 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE*. 1990, 78(No. 9), 1415-1442. DOI: 10.1109/5.58323. ISSN 00189219. Dostupné také z: http://www-isl.stanford.edu/~widrow/papers/j199030years.pdf

[32] VAN VEEN, Fjodor a Stefan LEIJNEN. The Neural Network Zoo. In: *The Asimov Institute* [online]. c2019, September 14, 2016 [cit. 2019-07-01]. Dostupné z: http://www.asimovinstitute.org/neural-network-zoo

[33] PILÁT, Martin. Neuronové sítě - RBF sítě a rekurentní sítě. In: *Martin Pilát* [online]. 2. dubna 2019 [cit. 2019-07-07]. Dostupné z: https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/neuronove-site-rbf-site-rekurentni-site

[34] PILÁT, Martin. Neuronové sítě - konvoluční sítě a zpracování obrazu. In: *Martin Pilát* [online]. 10. dubna 2019 [cit. 2019-07-07]. Dostupné z: https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/neuronove-site-konvolucni-site-zpracovani-obrazu

[35] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. In: *Towards Data Science: Sharing concepts, ideas, and codes* [online]. Dec 15, 2018 [cit. 2019-07-07]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[36] NGUYEN, Anh, Jason YOSINSKI a Jeff CLUNE. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2015, 2015, , 427-436 [cit. 2019-07-08]. DOI: 10.1109/CVPR.2015.7298640. ISBN 978-1-4673-6964-0. Dostupné z: http://www.evolvingai.org/files/DNNsEasilyFooled_cvpr15.pdf

[37] GOODFELLOW, Ian J., Jonathon SHLENS a Christian SZEGEDY. Explaining and Harnessing Adversarial Examples. *ArXiv:1412.6572v3 [stat.ML]* [online]. 20 Mar 2015 [cit. 2019-07-03]. Dostupné z: https://arxiv.org/abs/1412.6572v3

[38] MATERNA, Jiří. Deep Learning: budoucnost strojového učení?. In: *Blog Seznam.cz* [online]. Seznam.cz, 2019, 9. 1. 2013 [cit. 2019-07-01]. Dostupné z: https://blog.seznam.cz/2013/01/deep-learning-budoucnost-strojoveho-uceni

[39] RASCHKA, Sebastian. Single-Layer Neural Networks and Gradient Descent. In: *Dr. Sebastian Raschka* [online]. c2013-2019, Mar 24, 2015 [cit. 2019-07-07]. Dostupné z: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

[40] BROWNLEE, Jason. How to Fix the Vanishing Gradients Problem Using the ReLU. In: *Machine Learning Mastery* [online]. c2019, January 11, 2019 [cit. 2019-07-12]. Dostupné z: https://machinelearningmastery.com/

how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function

[41] KATHURIA, Ayoosh. Intro to optimization in deep learning: Vanishing gradients and choosing the right activation function. In: *Paperspace Blog* [online]. c2019, 9 July 2018 [cit. 2019-08-01]. Dostupné z: https://blog.paperspace.com/vanishing-gradients-activation-function

[42] BROWNLEE, Jason. A Gentle Introduction to Exploding Gradients in Neural Networks. In: *Machine Learning Mastery* [online]. c2019, December 18, 2017 [cit. 2019-07-12]. Dostupné z: https://machinelearningmastery.com/exploding-gradients-in-neural-networks

[43] MHASKAR, H.N. a C.A. MICCHELLI. How to Choose an Activation Function. COWAN, J.D., G. TESAURO a J. ALSPECTOR, eds. *Advances in neural information processing systems 6* [online]. Morgan-Kaufmann, 1994, s. 319-326 [cit. 2019-07-01]. ISBN 1-55860-274-7. Dostupné z: https://papers.nips.cc/book/advances-in-neural-information-processing-systems-6-1993

[44] NIELSEN, Michael A. A visual proof that neural nets can compute any function. Neural Networks and Deep Learning [online]. Determination Press, 2015 [cit. 2019-07-01]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap4.html

[45] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: MIT Press, 2016. ISBN 978-0262035613. Dostupné také z: http://www.deeplearningbook.org

[46] BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU). In: *Machine Learning Mastery* [online]. c2019, January 9, 2019 [cit. 2019-06-10]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks

[47] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. ImageNet Classification with Deep ConvolutionalNeural Networks. PEREIRA, F., C. J. C. BURGES, L. BOTTOU a K. Q. WEINBERGER, eds. *Advances in Neural Information Processing Systems 25* [online]. Curran Associates, 2012, s. 1097-1105 [cit. 2019-07-01]. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[48] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv:1502.01852v1 [cs.CV]* [online]. 6 Feb 2015 [cit. 2019-07-01]. Dostupné z: https://arxiv.org/abs/1502.01852v1

[49] CLEVERT, Djork-Arné, Thomas UNTERTHINER a Sepp HOCHREITER. Fast and Accurate Deep Network Learning by Exponential Linear Units

(ELUs). *ArXiv:1511.07289v5 [cs.LG]* [online]. 22 Feb 2016 [cit. 2019-07-02]. Dostupné z: https://arxiv.org/abs/1511.07289v5

[50] GLOROT, Xavier a Yoshua BENGIO. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* [online]. PMLR, 2010, (9), 249-256 [cit. 2019-07-02]. Dostupné z: http://proceedings.mlr.press/v9/glorot10a.html

[51] ZHOU, Victor. A Simple Explanation of the Softmax Function. In: *Victor Zhou* [online]. July 22, 2019 [cit. 2019-07-28]. Dostupné z: https://victorzhou.com/blog/softmax

[52] TORREY, Lisa a Jude SHAVLIK. Transfer Learning. *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques.* 2009. Hershey, PA: Information Science Reference, c2010, s. 242-264. ISBN 9781605667669. Dostupné také z: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

[53] PAN, Sinno Jialin a Qiang YANG. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* [online]. 2010, 22(10), 1345-1359 [cit. 2019-06-06]. DOI: 10.1109/TKDE.2009.191. ISSN 1041-4347. Dostupné z: http://ieeexplore.ieee.org/document/5288526

[54] Transfer Learning Using Pretrained ConvNets. In: *TensorFlow* [online]. [cit. 2019-08-15]. Dostupné z: https://www.tensorflow.org/tutorials/images/transfer_learning

[55] MARCELINO, Pedro. Transfer learning from pre-trained models. In: *Towards Data Science* [online]. Oct 23, 2018 [cit. 2019-08-20]. Dostupné z: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

[56] SARKAR, Dipanjan. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. In: *Towards Data Science: Sharing concepts, ideas, and codes* [online]. Nov 14, 2018 [cit. 2019-06-01]. Dostupné z: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[57] FRACOIS, Chollet and others. Applications. *Keras Documentation* [online]. 2015 [cit. 2019-11-10]. Dostupné z: https://keras.io/applications/#applications

Obsah elektronické přílohy

Součástí této práce je také elektronická příloha. Jedná se o ZIP archiv, jehož struktura je následující:

- soubor **czk_coins_classification.ipynb** – obsahuje zdrojový kód pro trénování modelu; pro práci s ním je ho třeba nahrát do prostředí Colaboratory.
- adresáře **czk01**, **czk02** a **czk03** – odpovídají jednotlivým trénovacím konfiguracím; obsahují uložené modely a výpis z konzole při trénování.
- adresář **log** – obsahuje logy z trénování modelu podle výše uvedených konfigurací; pro prohlížení těchto logů je třeba je otevřít v nástroji TensorBoard.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Jan Štol**

Studium: I1600273

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Strojové učení s využitím metody transfer learning**

Název diplomové práce AJ: Machine learning with transfer learning method

Cíl, metody, literatura, předpoklady:

Cíl práce: Navrhnout a ověřit pokročilé algoritmy strojového učení pro malé soubory trénovacích dat. Osnova: 1. Úvod, rešerše tématu 2. Cíl práce a způsob řešení 3. Strojové učení 4. Návrh a ověření učících algoritmů 5. Závěr a shrnutí výsledků

GOODFELLOW, Ian, Yoshua BENGIO a Aaron COUR-VILLE. Deep learning. Cambridge, Massachusetts: MIT Press,2016. ISBN 978-0262035613 TORREY, Lisa; SHAVLIK, Jude. Transfer learning. In: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques. IGI Global, 2010. p. 242-264. PAN, Sinno Jialin; YANG, Qiang. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 2009, 22.10: 1345-1359.

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Karel Mls, Ph.D.

Oponent: Mgr. et Mgr. Rafael Doležal, Ph.D.

Datum zadání závěrečné práce: 21.10.2014