VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NEURONOVÁ SÍŤ PRO PŘENOS STYLU

NEURAL NETWORK FOR STYLE TRANSFER

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — FILIP KADLEC

AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ HŮLKA

SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Filip Kadlec**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Tomáš Hůlka**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Neuronová síť pro přenos stylu

Stručná charakteristika problematiky úkolu:

Interpretace sémantického obsahu obrazu v jiném stylu je obtížnou úlohou. Zvláště kvůli nedostatku prostředků, které by umožňovaly jasné odlišení obsahu fotografie či malby od stylu autora. Tento problém lze vyřešit pomocí vhodně navržené konvoluční neuronové sítě, která je schopna obsah a styl separovat. Styl předlohy je poté možno zkombinovat s jinými fotografiemi, které tak mohou získat podobu např. známých uměleckých děl.

Cíle bakalářské práce:

Cílem této práce je stručná rešerše problematiky následovaná návrhem a implementací neuronové sítě pro přenos stylu.

Seznam doporučené literatury:

GATYS, L.A., ECKER, A.S., BETHGE, M.: A neural algorithm of artistic style (2015). CoRR abs/1508.06576, [cit. 17.9.2018]. Dostupné z: http://arxiv.org/abs/1508.06576.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt
V této bakalářské práci je popsáno strojové učení, typy umělých neuronových sítí a pochody v neuronových sítích, jako dopředné zpracování dat či trénování a učení samotných sítí. Také se zde srovnávají a popisují prostředky pro implementaci neuronových sítí. Praktická část je věnována problematice přenosu uměleckého stylu pomocí konvoluční neuronové sítě.

Abstract
In this bachelor's thesis we describe machine learning, types of artificial neural networks and internal processes of neural networks, such as feedforward data processing and training neural networks. We are also pursuing comparison and description of libraries (such as TensorFlow and Keras), which are suitable for neural networks implementation. In the practical part of thesis, we are dealing with problem called artistic style transfer with convolutional neural network.

Klíčová slova
strojové učení, hluboké učení, umělé neuronové sítě, přenos uměleckého stylu, TensorFlow, Keras

Keywords
machine learning, deep learning, artificial neural networks, artistic style transfer, TensorFlow, Keras

KADLEC, F.*Neuronová síť pro přenos stylu*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2019. 54 s. Vedoucí bakalářské práce Ing. Tomáš Hůlka.

Prohlašuji, že jsem bakalářskou práci na téma „Neuronová síť pro přenos stylu“ vypracoval samostatně a s použitím uvedené literatury a pramenů.

. . . . . .

Filip Kadlec

20. května 2019

Mé poděkování patří Ing. Tomáši Hůlkovi za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Filip Kadlec

Obsah

Úvod

Umělá inteligence, strojové učení, neuronové sítě. Často opakovaná slovní spojení posledních let. Dnes už to ovšem nejsou jen výmysly budoucnosti a obory v akademických kruzích. Většina lidí se s nimi setkává každý den.

Myšlenka neuronových sítí není nijak nová, už ve 40. letech 20. století se rodily první koncepty umělých neuronových sítí inspirované lidským mozkem. Rozkvět a použití těchto sítí však nastaly až v posledních letech. Důsledek tohoto technického pokroku můžeme pozorovat například na mobilních telefonech, které dnes umí rozpoznat obličeje či otisky prstů, na internetových překladačích, v autech, která disponují samořídícím systémem (self-driving cars). Neuronové sítě lze taktéž aplikovat na kategorizaci galaxií či zdravotních fotografií (např. rentgenových), předpověď počasí nebo predikci pohybu hodnot tržních cen na akciovém trhu. Možnosti využití jsou nesčetné, krása ovšem tkví v tom, že typy neuronových sítí, jak je známe dnes, pracují stejně při aplikaci na jakýkoliv problém, který je jim předložen.

Práce provede čtenáře od nejobecnější tematiky postupně hlouběji danou problematikou potřebnou pro pochopení a realizaci cíle práce. Nejprve tedy bude představena oblast umělé inteligence, strojové učení. Od strojového učení se práce přesune k základnímu popisu neuronových sítí. Následovat bude podrobnější popis vnitřních pochodů neuronových sítí, rozdělení sítí a popis jejích druhů s tím, že největší pozornost bude věnována konvolučním neuronovým sítím, jelikož tento typ je použit při řešení úkolu přenosu uměleckého stylu.

Cílem této práce je tedy podat nezasvěcenému čtenáři teoretickou část problematiky srozumitelně a pokud možno poutavě a následně navrhnout a implementovat software pro přenos uměleckého stylu pomocí neuronové sítě za využití knihoven Keras a TensorFlow. Úkolem tohoto programu je převzít z uměleckého díla specifický umělecký styl, kterým dílo disponuje, a aplikovat ho na jiný obrázek či fotografii.

1 Strojové učení

Strojové učení, neboli *machine learning* (ML), je oblast umělé inteligence. Zaměřuje se na vývoj počítačových programů, algoritmů, pomocí kterých je systém schopen se učit a v učené schopnosti se zdokonalovat. Hlavním cílem je umožnit systému učit se automaticky, bez zásahů člověka. Základním předpokladem strojového učení je sestavení těchto algoritmů, které přijímají vstupní data a následné predikce výstupních dat pomocí statistické a pravděpodobnostní analýzy. Algoritmus strojového učení vytvoří trénováním daného úkolu na vzorku dat (trénovací data) matematický model, který je následně užíván pro predikci či rozhodnutí na základě dat vstupujících do tohoto modelu.[1]

1.1 Strojové učení vs hluboké učení

Pojmy strojové učení (machine learning) a hluboké učení (*deep learning*, DL) bývají často zaměňovány. Jejich význam je ale odlišný. Deep learning je podoblast ML a užívá se u neuronových sítí, které jsou vícevrstvé. Neuronovým sítím a jejich vlastnostem bude pozornost věnována v kapitole 2.

Machine learning se standardně hodí k vyhodnocování dat, které jsou v řádech až tisíců datových jednotek, výstupem bývá numerická hodnota značící například klasifikaci, či úspěšnost vyhodnocení. Užívá různé algoritmy pro vytvoření modelu/modelové funkce, která následně predikuje výsledek ze vstupních dat. Lze také říci, že při ML je trénování pod kontrolou. Tím není myšleno, že se do učení zasahuje, ale cíleně se vytváří algoritmy na určitá data a ví se, jaký bude mít učení průběh. ML lze použít například pro: klasifikaci, regresi, nebo shlukovou analýzu dat.[1]

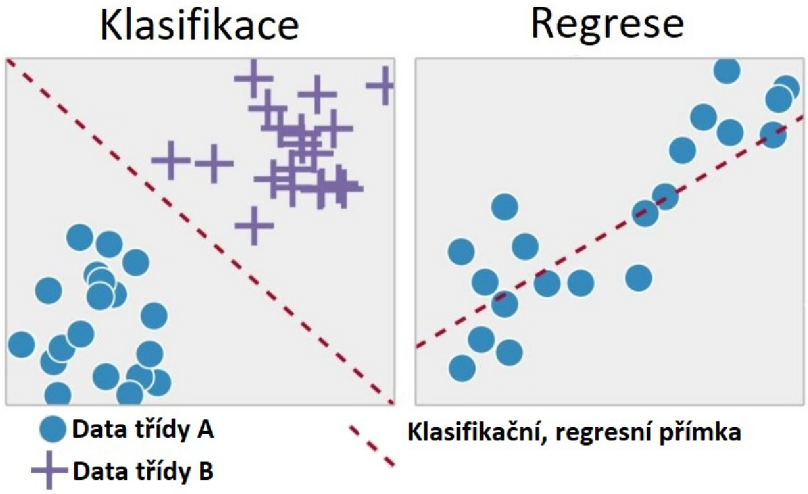
Deep learning se hodí k vyhodnocování obrovských datasetů v řádech milionů datových jednotek. Výstupem také může být numerická hodnota značící klasifikaci či úspěšnost, ale jelikož vstupem mohou být také zvuky, vizuální objekty, texty, zkrátka cokoliv, na co jsme schopni hlubokou neuronovou síť natrénovat, výstupy jsou velice rozmanité (více o neuronových sítích v kapitole 2). DL užívá již zmíněné neuronové sítě, kterými proudí data skrz všechny její vrstvy. Algoritmy uvnitř neuronové sítě jsou víceméně samořízené a sama síť rozpoznává a ukládá si různé charakteristiky, rysy, vlastnosti datasetů. Lze tedy říct, že vhled do sítě a jejích algoritmů není k dispozici a opodstatnění všech vnitřních pochodů sítě není zcela známo. Úlohy, jenž DL řeší, mohou být například: doporučení videí, písniček na YouTube, rozpoznávání objektů, předpověď počasí, či právě přenos uměleckého stylu.[2]

1.2 Strojové učení - rozdělení

ML lze rozdělit na *supervised* ML (učení s učitelem), *unsupervised* ML (učení bez učitele) a *semi-supervised* ML (kombinace učení s učitelem a bez učitele).

1.2.1 Učení s učitelem

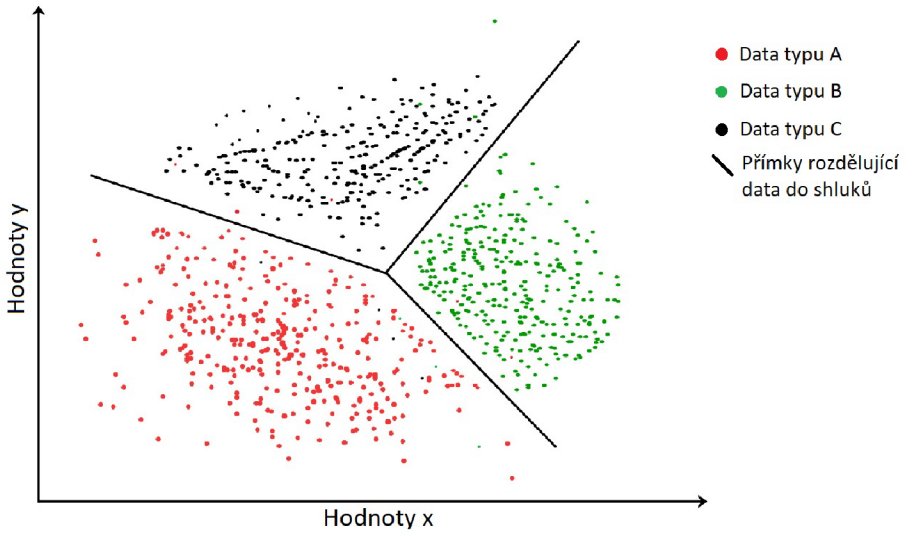
Úkolem algoritmu je v tomto případě vytvořit funkci, která se aproximuje tak dobře, aby byla schopna správně predikovat nová vstupní data (například je správně klasifikovat). Při tomto učení jsou data, či příklady, které budou dejme tomu klasifikovány, označena (rozdělena do tříd). Díky tomu, že jsou označena, je možno při učení algoritmu říci, zda kategorizoval vzorek správně, popřípadě jak moc špatně. Na základě této informace se tento algoritmus „poučí“ a změní své parametry. Tento proces se opakuje tak dlouho, než je funkce dostatečně aproximována a algoritmus klasifikuje, predikuje nová data s určitou (pokud možno co nejvyšší) úspěšností. Jednoduché problémy užívající tento typ učení jsou například klasifikace či lineární regrese.[3, 4]



Obrázek 1.1: Příklady učení s učitelem. Převzato a upraveno z [4]

1.2.2 Učení bez učitele

Při tomto učení data nejsou označena, tudíž se algoritmu nic nesděluje. Tento typ učení se užívá, když není třeba data klasifikovat a není hledán správný výstup, spíše je zde snaha sestrojit funkci, která je schopna popsat skrytou strukturu či distribuci dat. Algoritmus tedy nevyhodnocuje správný výstup, ale data seskupí do různých skupin podle charakteristik, kterými se určité datapointy vyznačují. Toto učení se užívá například při shlukové analýze. Příklad shlukové analýzy je na obr. 1.2.[3, 4]



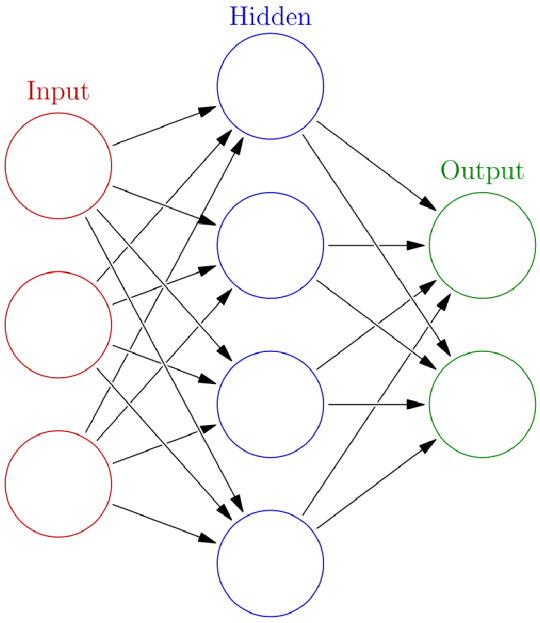
Obrázek 1.2: Příklad učení bez dozoru. Převzato a upraveno z [21]

1.2.3 Kombinace učení s učitelem a bez učitele

Jak název napovídá, jedná se o kombinaci učení pod dohledem a učení bez dohledu. Dalo by se říci, že tento způsob učení je učení pod dohledem, kde je možno se vyhnout velkému množství označených příkladů. Tedy některá data označena jsou, většina ne. Například lze pomocí učení pod dohledem klasifikovat označená data a pomocí učení bez dohledu najít nový druh (třídu) dat, který spojují určité charakteristické znaky a označit je. Následně je možno znovu trénovat s nově označenými daty, atd. Takto natrénovaný model lze používat k predikci, podobně jako model natrénovaný učením pod dohledem.[3]

2 Neuronové sítě

Neuronové sítě jsou velkou podkapitolou ML. Nejlepší představu o neuronových sítích lze získat pomocí obrázku.



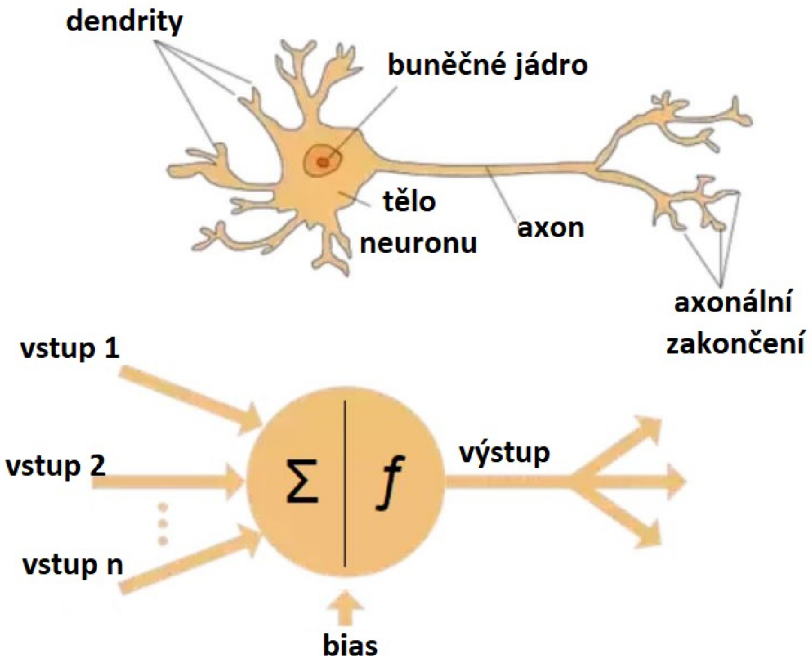
Obrázek 2.1: Zcela propojená neuronová síť [22]

Neuronové sítě (v rámci umělé inteligence) jsou matematické, výpočetní modely, které utváří samostatnou oblast machine learningu. Jsou v podstatě strukturami, které umí souběžně (paralelně) zpracovávat určité množství dat. Jsou utvořeny z tzv. neuronů, jejichž předlohou je biologický neuron (viz podkapitola 2.1). Jednotlivé neurony jsou navzájem propojeny a předávají si jeden druhému informace, které každý neuron mění a zpracuje pomocí aktivačních funkcí, kterými jsou neurony charakterizovány. Neurony mají libovolný počet vstupů, ale vždy jeden výstup.[5]

Neurony standardně tvoří vrstvy (viz obr. 2.1). Každá neuronová síť má vstupní, výstupní vrstvu a skryté vrstvy (hidden layers). Jak názvy napovídají, vstupní vrstva přijímá data a je první, která s daty pracuje. Výstupní vrstva přijme data z poslední skryté vrstvy a vyhodnotí výsledek (výsledky). Proces učení se odehrává právě ve skrytých vrstvách. Tento proces se detailněji popíše v podkapitole 2.2.

2.1 Srovnání umělého neuronu s biologickým neuronem

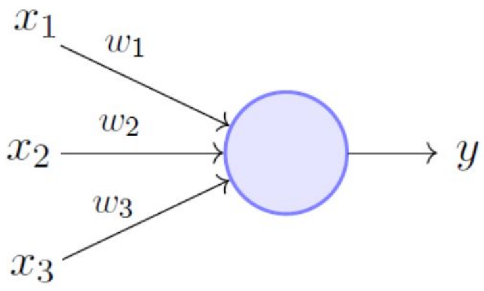
Jak již bylo zmíněno, biologický neuron byl předlohou umělého neuronu.



Obrázek 2.2: Biologický a umělý neuron. Převzato a upraveno z [23]

Lze si všimnout podobnosti nejen co se vzhledu týče, ale také podobnosti funkční. Biologický neuron přijímá signál pomocí dendritů. Buněčným jádrem signál zpracovává a následně axonem vede výstupní data.

Umělý neuron přijímá signály vstupními kanály. Stejně jako u neuronu biologického může být vstupních signálů více. Následné zpracování dat probíhá pomocí aktivační funkce a z neuronu vychází jeden výstupní signál. Těmito informacemi byla popsána stavba nejjednoduššího modelu neuronové (dopředné) sítě *perceptron*. Perceptron je neuronová síť o jednom neuronu.[6, 7]



Obrázek 2.3: Ilustrační obrázek perceptronu [8]

Na obr. 2.3 lze popsat, jak perceptron funguje a může tak být zjednodušeně demonstrováno, jak funguje neuronová síť. Zobrazené znaky značené x představují vstupní signály. Zobrazené znaky w jsou váhy konexí (konexí je zde myšleno pomyslné spojení mezi dvěma neurony, nebo vstupem a neuronem). Tím je myšleno, že čím větší je hodnota váhy u dané konexe, tím větší má daný vstup význam při vyhodnocování výsledku. Například váha, která má na konexi hodnotu blízko nule, moc výsledek neovlivní a naopak. Data pronásobená vahami vstupují na obrázku do modrého kruhu, který by zde mohl představovat aktivační (přenosovou) funkci. Aktivační funkce jsou různé a definují výstup z neuronů. Díky nim je možno třeba korigovat výstup do určitého intervalu (např. od 0 po 1). Také právě díky těmto funkcím jsou sítě schopny se učit. Více o aktivačních funkcích a vahách a práci s nimi v podkapitole 2.3. Posledním prvkem je zde výstup z perceptronu označen písmenem y.[6, 8]

Perceptron v jednoduché formě funguje jako binární klasifikátor. Binární klasifikátor je funkce, která umí rozhodnout, zda vstup, reprezentován vektorem čísel, patří do určité třídy. S tím, co bylo do této chvíle popsáno, si lze perceptron definovat jako funkci:

$$f(x) = wx + b, \tag{2.1}$$

kde w je vektor vah, x je vstup (vektor čísel) a b je *bias* (bias posouvá rozhodovací hranici od počátku (od nuly), přičemž nezáleží na žádné vstupní hodnotě. Zde i u složitějších neuronových sítí se před trénováním nastaví na náhodnou hodnotu, stejně jako váhy, a v procesu trénování se mění, než dokonverguje k užitečné hodnotě při řešení problému, např. klasifikace). [8]

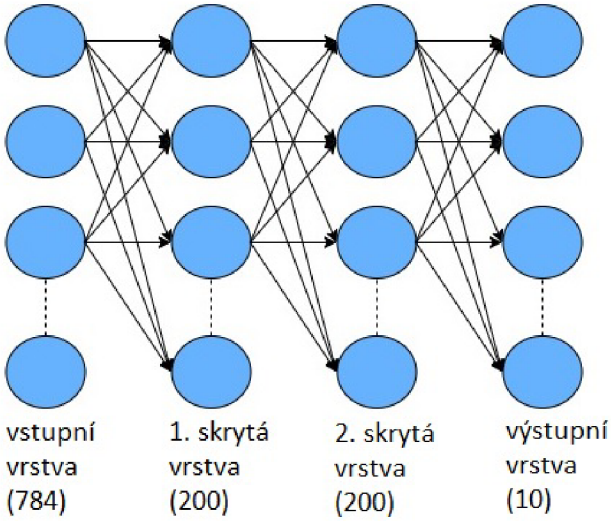
Tato rovnice klasifikuje vstup jako pravdivý, patřící do dané třídy $f(x) = 1$, pokud $wx + b > 0$, jinak $f(x) = 0$.

2.2 Pochody uvnitř neuronových sítí

2.2.1 Dopředné proudění dat

Fungování a prvky neuronových sítí je možno nejlépe vysvětlit na jednom z jednodušších typů neuronových sítí, tj. *fully connected neural network* (zcela propojená neuronová síť, neboli multilayer perceptron, FCNN). Tyto procesy totiž lze aplikovat i na složitější druhy neuronových sítí, avšak tam je potřeba vstřebat více informací, lepší tedy bude, když nyní budou separovaně vysvětleny pochody v sítích a specifickým prvkům pro daný typ sítí bude věnována pozornost ve vlastní kapitole.

A jelikož principy sítí fungují v zásadě pořád stejně, je možné si je dovolit demonstrovat na poměrně základním problému, a to klasifikace. Konkrétně klasifikace ručně psaných čísel. Je užita tato síť (obr. 2.4):



Obrázek 2.4: Příklad zcela propojené neuronové sítě. Převzato a upraveno z [24]

A tento typ vstupů:

Obrázek 2.5: Vzorek vstupů do sítě (z databáze MNIST) [25]

Síť je tvořena vstupní vrstvou o 784 neuronech (to znamená, že do sítě proudí vstup složen z 784 prvků). Tento vstup je černobílý obrázek o 784 pixelech (28x28, viz obr. 2.5). Každý vstupní neuron tedy nyní drží hodnotu jednoho pixelu. Dále síť tvoří dvě skryté vrstvy (hidden layers) o 200 neuronech. To, že je zde právě 200 neuronů je možnost volby. Dále je zde 10 neuronů ve výstupní vrstvě právě proto, že je rozlišovaných (klasifikovaných) 10 čísel (0-9).

Ke kompletnímu popisu sítě na obr. 2.4 už jen chybí popsat samotné neurony. Jednotlivé neurony si lze představit jako funkce, které vstupní signál pomocí své aktivační funkce zpracují a vypočítanou hodnotu posílají jako výstup dál do sítě do vstupů dalších neuronů. Existuje několik aktivačních funkcí. Jejich rozdělení a bližšímu popisu bude věnována podkapitola 2.3. Zde je zvolena jako aktivační funkci sigmoida, která má tvar: $\sigma(x) = \frac{1}{1+e^{-x}}$. Tato funkce do sítě jednak zavede nelinearitu a jednak transformuje výstupy z neuronů při zachování poměru do intervalu (0-1).

Nyní už jsou známy detaily potřebné pro vysvětlení toku dat skrz danou síť. Jak si lze všimnout na obr. 2.4, všechny neurony jsou propojeny se všemi a pracují vesměs stejně, jak bylo vysvětleno v kapitole 2.1, tedy ze vstupní vrstvy jdou do každého neuronu první skryté vrstvy informace ze všech vstupních neuronů. Všechny tyto informace jsou vynásobeny vahami jednotlivých konexí. Následně neurony v první vrstvě data zpracují

aktivační funkcí (vybraná sigmoida) a vyšlou informace dál, kde následně slouží jako vstupní informace do druhé skryté vrstvy a operace se opakuje.

Například operace jakéhokoliv neuronu v první skryté vrstvě by se dala popsat matematickou rovnicí:

$$a_x^2 = \sigma(w_{1,x}^1 a_1^1 + w_{2,x}^1 a_2^1 + w_{3,x}^1 a_3^1 + ... + w_{n,x}^1 a_n^1 + b), \tag{2.2}$$

kde a_x^2 je jakýkoliv neuron v druhé vrstvě (tedy první skrytá), $w_{i,x}^1$ jsou váhy (horní index - z které vrstvy jsou vedeny, spodní index - odkud je konexe vedena (1,x znamená konexe vedena z prvního neuronu v první vrstvě do našeho hledaného neuronu v druhé vrstvě)), a_i^1 jsou výstupní hodnoty z neuronů z předchozí vrstvy a b je tzv. bias (bias slouží k posunutí aktivační funkce od počátku. Je to parametr, který se neuronová síť může učit, stejně jako váhy).[2] Vzhledem k tomu, že díky rovnici č. 2.2 je možno popsat výpočet jednoho neuronu a jsou známy rozměry všech vrstev, lze matematicky popsat i výpočet dat celé vrstvy pomocí matic.

$$\mathbf{a^{(2)}} = \sigma(\mathbf{W^{(1)}a^{(1)}} + \mathbf{b}) \tag{2.3}$$

Tedy:

$$\mathbf{a^{(2)}} = \sigma\left(\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} & \dots & w_{1,n}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} & \dots & w_{2,n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(1)} & w_{m,2}^{(1)} & m_{m,3}^{(1)} & \dots & w_{m,n}^{(1)} \end{bmatrix} \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\right) \tag{2.4}$$

2.2.2 Učení a trénování neuronových sítí

Jelikož v tomto případě jsou výsledky známy a vstupní data jsou označena, jedná se o učení s učitelem (supervised learning). Před vnořením do detailů, lze proces učení shrnout do pár vět.

Síť vyhodnotí[1] první černobílý obrázek s ručně psaným číslem. Vzhledem k tomu, že váhy i bias mají momentálně náhodné hodnoty (běžně se před procesem učení zvolí hodnoty vah náhodně a úkolem při učení je dokonvergovat k těm správným pro daný úkon), je nepravděpodobné, že výsledek bude správný. Po vyhodnocení výsledku řekneme síti, jak moc špatně výsledek určila. Podle této informace změní své parametry (váhy a bias). Tento proces se opakuje do té doby, dokud parametry neuronové sítě nedokonvergují k hodnotám, se kterými bude správně klasifikovat vstupní data.[2]

Z předchozí podkapitoly je již známo, jak dostat ze vstupních dat výsledek. Nyní bude pozornost věnována *ztrátě* (cost). Právě díky ní lze síti sdělit, „jak moc špatně“ byl výsledek vyhodnocen. Ztrátových funkcí je několik, například ta následující se nazývá

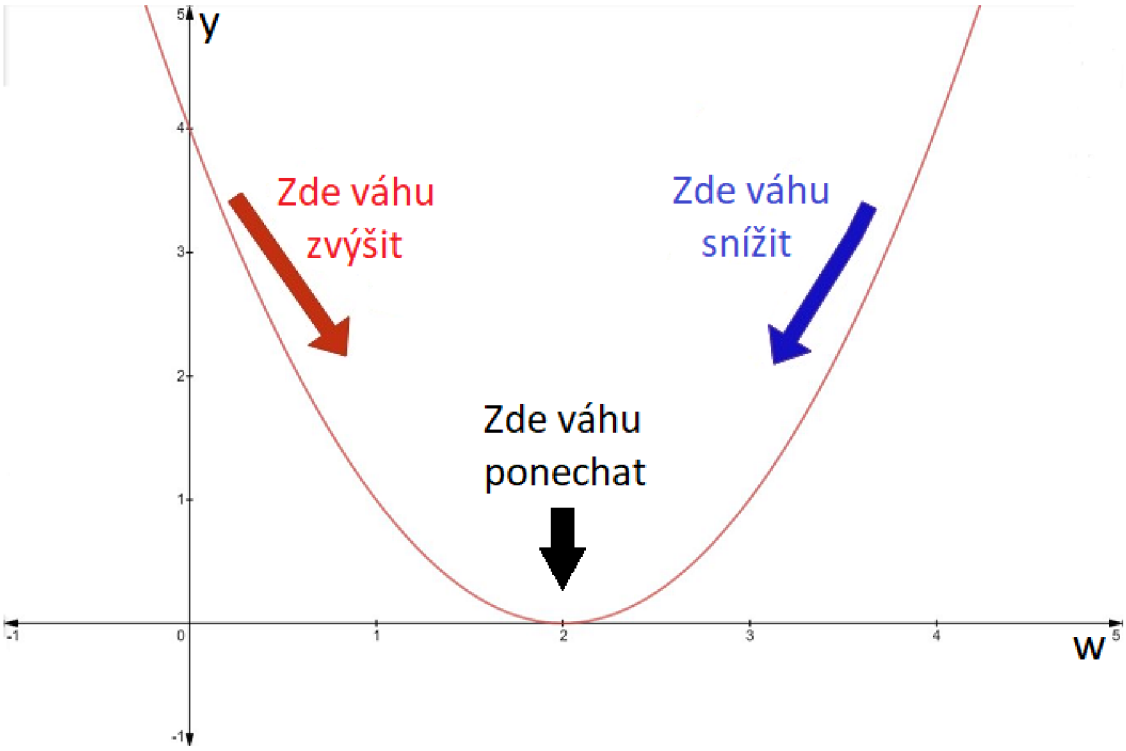
[1] Vyhodnocením je myšlena situace, kdy výstupní vrstva dá výsledek u každého výstupního neuronu, které představují hledaná čísla 0-9. Každý neuron nám dá jiný číselný výsledek v intervalu 0-1. Čím vyšší číslo je u daného neuronu (tedy čísla 0-9), tím je podle neuronové sítě pravděpodobnější, že je to výsledek správný.

střední kvadratická chyba (en: mean squared error). Rovnice pro celkovou ztrátu sítě tedy může vypadat následovně:

$$Cost = \frac{1}{2N}\sum_{i=1}^{N}(Real\_values - Wanted\_values)^2, \tag{2.5}$$

kde N je počet výstupních neuronů. Chtěné hodnoty jsou samozřejmě 1 u čísla, které je správně a 0 u všech ostatních. Nyní je známo jak moc je třeba měnit hodnoty výsledků a jednotlivých výstupních neuronů. Bohužel, k neuronům jako takovým přístup není, jako parametry lze ovládat pouze váhy a bias. Tedy například konexe vedené zpět z čísla, které by mělo být vyhodnoceno jako správné by se měli měnit tím způsobem, že se budou zvyšovat, pokud spojují vybrané číslo s neurony mající čísla kladná a zmenšovat s neurony s čísly zápornými. U ostatních naopak.[9]

Jak ale váhy měnit, jak moc a jakým směrem (kladným či záporným), pokud výpočty probíhají hlouběji a hlouběji zpět sítí a jak to síti sdělit? Odpověď zní: derivacemi ztrátové funkce. Derivace ve své podstatě určuje rychlost změny derivované funkce. Jelikož úkolem je minimalizovat ztrátu a ztrátovou funkci, je v podstatě hledáno minimum této ztrátové funkce.[2, 9] Lze si zde ilustrativně vykreslit ztrátovou funkci do grafu závislosti hodnoty ztrátové funkce na vahách.



Obrázek 2.6: Graf závislosti hodnot ztrát. funkce na hodnotě váhy. [10]

Vzhledem k tomu, že je v síti, se kterou operujeme, necelých 200 000 parametrů, tak nejen vykreslení, ale pouhá představa takového počtu dimenzí je problematická. Pro představu bude dostačující pracovat s dvěma dimenzemi, kde je hledání minima netěžkou záležitostí.

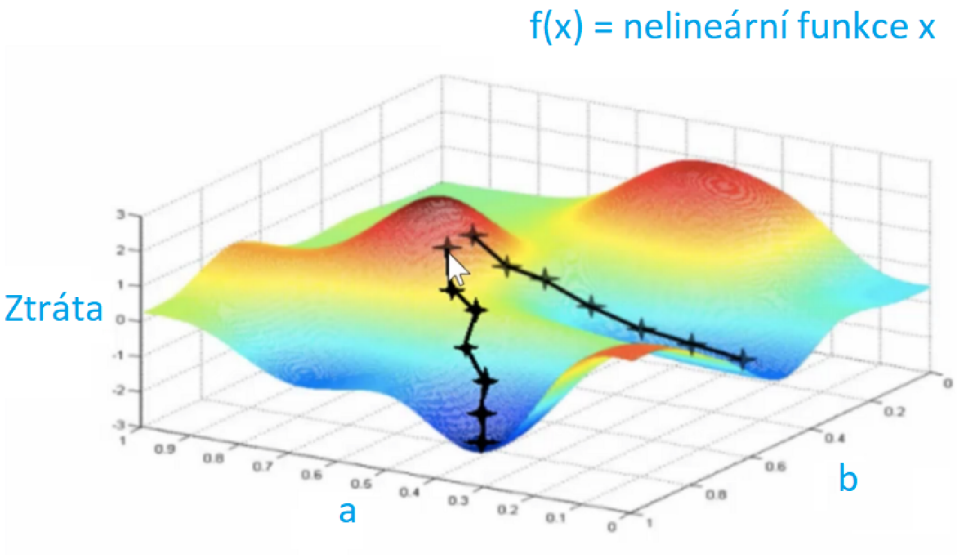
Pokud tedy v tomto konkrétním případě dle obr. 2.6 má váha hodnotu 2 ($w = 2$), tak jsme dosáhli minima ztrátové funkce (derivace je rovna nule) a hodnoty výsledků se neliší od hodnot chtěných.

Pokud $w < 2$ (derivace funkce je záporná), tak při zvětšení hodnoty váhy se hodnota ztrátové funkce sníží.

Pokud $w > 2$ (derivace funkce je kladná), tak snížení hodnoty ztrátové funkce lze dosáhnout zmenšením hodnoty váhy.[10]

A přesně to je žádané, najít minimum ztrátové funkce s ohledem na všechny váhy v systému. Při učení je samozřejmě chtěné minimalizovat ztrátu co nejrychleji, takže je zde snaha jít po „nejstrmější cestě“ směrem k minimu. Jinak řečeno, je zde snaha jít ve směru proti gradientu ztrátové funkce (gradient descent).[2]

Ovšem u neuronových sítí není problém mít počet parametrů v rámci milionů a vstupní data také nemusí mít pouze 784 datapointů jako v tomto konkrétním případě. To je spousta kalkulace, která musí být provedena během jedné iterace. Tudíž se snažíme situaci vyřešit tím, že není počítána ztráta, její funkce a následná úprava vah s každým vzorkem dat, ale ztráta je spočítána pro větší počet vzorků trénovacích dat (typicky 32, 64) a až poté se upravují váhy. Výpočty v průběhu tudíž nejsou tak přesné, ale jsou mnohem rychlejší a požadovaný výsledek (najití minima ztrátové funkce) se dostaví mnohem dříve. Tento proces se nazývá *stochastic gradient descent* (SGD).[2]



Obrázek 2.7: Stochastic gradient descent. Převzato a upraveno z [26]

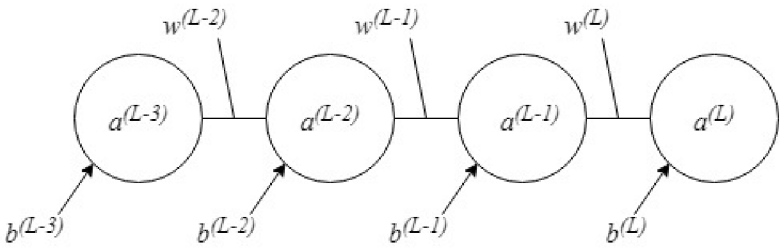
Dále stojí za zmínění něco podrobnějšího ohledně učení a úprav vah, jako takových. Na předešlém příkladu byla jedna váha, kterou jsme upravovali (obr. 2.6). Neuronové sítě však obvykle nejsou o jednom neuronu, či jedné vrstvě, ale většinou bývají několikavrstvé. Výpočet a úprava hodnot vah poté začne být komplikovanější. Trénování tak lze provádět pomocí algoritmu *backpropagation*.

Backpropagation je algoritmus učení pod dohledem pro trénování neuronových sítí. Jak název napovídá, proces se pohybuje sítí směrem od konce (výstupu) ke vstupu. Tento algoritmus dělá víceméně to, co bylo doteď popsáno ohledně učení, snaží se upravit pa-

rametry sítě tak, aby správně plnila funkci pomocí ztrátové funkce a jejího snižování po gradientu. Pro detaily ohledně zmíněných věcí se stačí vrátit v této kapitole zpět. Dalším předmětem výkladu bude matematický základ schovaný za backpropagation.[2]

Jak je již známo, všechny parametry se mění ve směru nejrychlejšího klesání ztrátové funkce. Tedy stačí parciálně zderivovat ztrátovou funkci podle daného parametru, jehož změna je předmětem zájmu ($\frac{\partial C}{\partial w}$). Kde C je ztrátová funkce (cost function) a w je určitá váha.[2, 10, 11]

Nejlépe lze tento problém vysvětlit na jednoduchém příkladu. V příkladu je za úkol spočítat změnu konkrétní váhy na tomto modelu proti směru gradientu (obr. 2.8). Jedná se o jedno vlákno neuronů vytržené z neuronové sítě. Neuron nejvíce napravo je součástí výstupní vrstvy, tudíž říká, jak síť vyhodnotila výsledek tímto neuronem.



Obrázek 2.8: Vlákno neuronů vytržené ze zcela propojené neuronové sítě

- $a^{(L)}$ je zde výstupní hodnota neuronu, L, L-1 je pouze indexové značení (Last, Last-1)
- y je požadovaný (chtěný) výstup
- $w^{(L)}$ je hodnota poslední váhy
- $b^{(L)}$ je hodnota posledního biasu

Také je možné napsat rovnici pro ztrátovou funkci, jen s jiným značením (dle obrázku):

$$C = \frac{1}{2N} \sum_{i=1}^{N} (a^L - y)^2 \tag{2.6}$$

Rovněž je možné napsat rovnici pro výpočet hodnoty posledního neuronu, která též byla zmiňována jak v této, tak v maticové formě:

$$a^L = \sigma(w^L a^{L-1} + b^L), \tag{2.7}$$

kde se vnitřek závorky (dalo by se pojmenovat jako vážená suma) označí neznámou, což se bude hodit pro následné jednodušší zacházení při výpočtech:

$$a^L = \sigma(z^L) \tag{2.8}$$

Nyní je úkolem zjistit parciální derivaci ztrátové funkce podle váhy, která je předmětem zájmu. To se dá také interpretovat jako míra citlivosti ztrátové funkce na změnu váhy.

Jinak řečeno, když se změní váha o nějakou hodnotu, jak velký vliv to bude mít na ztrátovou funkci.[2, 10, 11]

Se vztahem, který je hledán, jako takovým $\frac{\partial C}{\partial w}$ nelze moc pracovat, jelikož C je sice funkcí parametrů, podle kterých potenciálně můžeme derivovat, to ale nemáme explicitně zadáno. Řešením tohoto problému je tzv. řetězcové pravidlo (neboli *chain rule*). Zmíněné pravidlo tkví v tom, že lze užít vzorec pro složenou derivaci do dvou na sobě závislých funkcí. Princip spočívá v tom, že funkci, která se derivuje, lze nahradit jiným výrazem, který lze snáze derivovat.[2, 11] Matematický popis je možná stravitelnější, než ten slovní:

$$F(x) = f(g(x)), \tag{2.9}$$

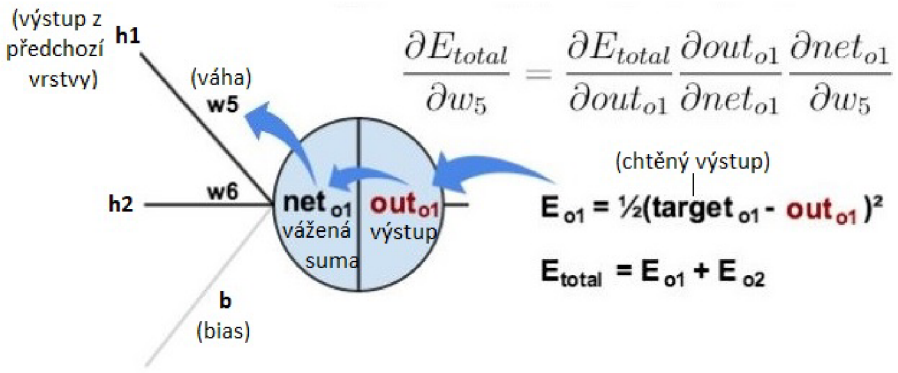
pak:

$$\frac{dF}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{2.10}$$

Toto lze aplikovat na řešený problém:

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{2.11}$$

Opět, co tento vzorec říká jde nejlépe znázornit na obrázku.
(Pozn.: $\partial E_{total} = \partial C$, $\partial w_5 = \partial w$, $\partial out_{o1} = \partial a^{(L)}$, $\partial net_{o1} = \partial z^{(L)}$)



Obrázek 2.9: Grafická ukázka pochodů v rovnici č. 2.11. Převzato a upraveno z [11]

Toto pravidlo lze aplikovat na všechny neurony s tím, že vstupní a výstupní data proudící směrem od konce sítě ke vstupní vrstvě na sobě jsou závislá stejně, jako když data proudila dopředně. Rozdíl je ten, že při dopředném průchodu dat sítí se užívá funkcí neuronů pouze k výpočtu, při backpropagaci do neuronu data vstoupí jako funkce, která musí být podle řetězcového pravidla zderivována (pro zjištění největšího spádu) a následně tím lze získat hodnotu změny váhy.

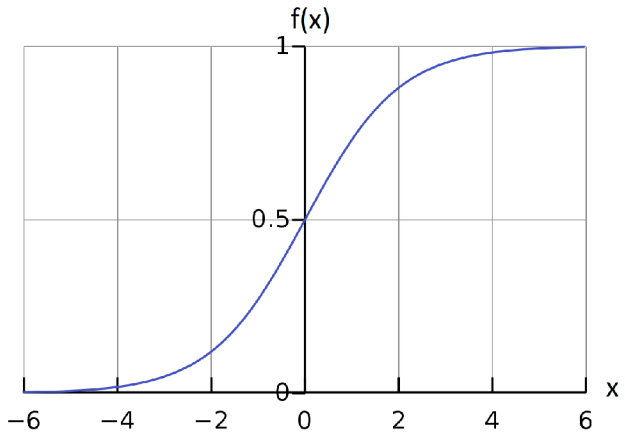
Po získání této hodnoty už se jen odečte/přičte k váze hodnota, která byla zjištěna. Poté se algoritmus opakuje s ohledem na pravidla zmíněná v této kapitole.[2, 11]

2.3 Aktivační funkce

Aktivační funkce jsou nedílnou součástí neuronových sítí. Definují jednotlivé neurony. Lze říct, že neurony nejsou nic jiného, než právě tyto funkce. Každá aktivační funkce vezme číslo (vstup, při vícero vstupech jejich sumu) a provede matematickou operaci, která bude zvolena.

Také je nesmírně důležité zavést do neuronové sítě nelinearity, jelikož síť pouze s lineárními funkcemi na neuronech je ekvivalentní lineární neuronové síti *bez skrytých vrstev*. Z čehož plyne, že k žádnému učení vah, ani při prohlubování sítě, nedojde.
V praxi se lze běžně setkat s několika aktivačními funkcemi.[2]
Sigmoida má tvar:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$



Obrázek 2.10: Funkce sigmoida. Převzato a upraveno z [27]

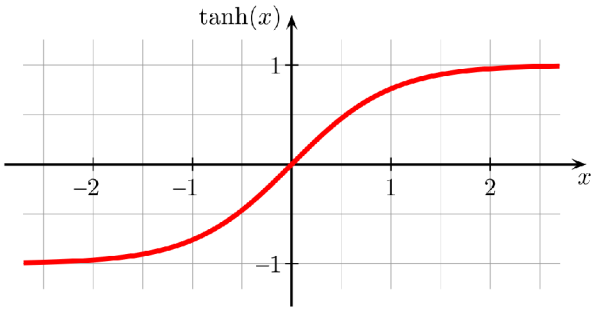
Tato nelinearita, jak je zobrazeno na obr. 2.10, zhustí hodnoty do intervalu 0-1. Lze si zde všimnout, že hodnoty vzdálené od nuly (neboli hodnoty blížící se k $\pm\infty$) mají výstupní hodnoty 0 nebo 1.

Sigmoida se dnes jako aktivační funkce neuronových sítí používá jen zřídka. V této práci v předchozích příkladech byla použita pro její jednoduchost a názornost, avšak v praxi má několik nedostatků. Problémem je právě to, čeho si lze všimnout na obr. 2.10, že funkce nabývá hodnot 0 a 1 poměrně blízko počátku. Tedy, pokud se hodnota výstupu funkce na neuronech blíží nule, tak konexe jdoucí z tohoto neuronu „zabije“ (budou dále předávat minimální, žádný signál). Podobný problém nastává při saturaci neuronu (neuron nabývá hodnoty téměř jedna). V tento moment síť vyhodnotí hodnotu za zcela správnou a síť se sotva naučí měnit své parametry.[2, 12]

Dalším problémem je, že výstupy ze sigmoidy nejsou nulově středěné. To zde znamená, že data jdoucí z neuronu (sigmoidy) jsou vždy větší rovna nule. To může zapříčinit to, že gradient při backpropagaci bude zcela kladný či záporný (na všech konexích), což situaci při hledání gradientu ztěžuje (při vykreslení se průběh hledání minima klikatí a čas značně prodlužuje). Tento problém však není tak významný, jako saturace/zabíjení neuronů.[2, 12]

Hyperbolický tangens má klasicky tvar:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.13}$$



Obrázek 2.11: Funkce tanh [28]

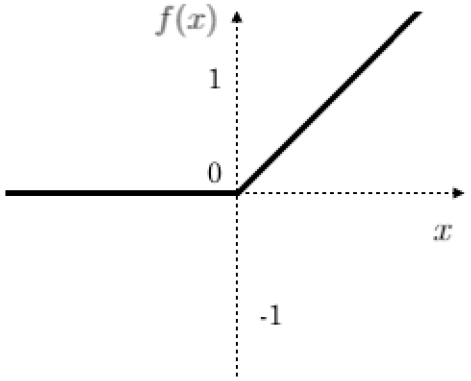
Jak lze zaznamenat, tanh(x) není nic jiného než sigmoida v jiném měřítku, nulou středěná a pro nižší výpočetní náročnost můžeme použít tvar:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.14}$$

Hyperbolický tangens výstupní hodnoty vmáčkne do intervalu [-1;1]. Také si lze všimnout, že je středěný nulou, takže jeden problém oproti sigmoidě ubývá, avšak ten větší zůstal. A to saturace a zabíjení neuronů. V praxi je hyperbolický tangens upřednostňován oproti sigmoidě téměř vždy. [2, 12]

ReLU (Rectifier, Rectified Linear Unit, překladem zkorigovaná lineární jednotka) se oproti předešlým funkcím používá již mnohem více. Má tvar:

$$f(x) = max(0,x) \tag{2.15}$$

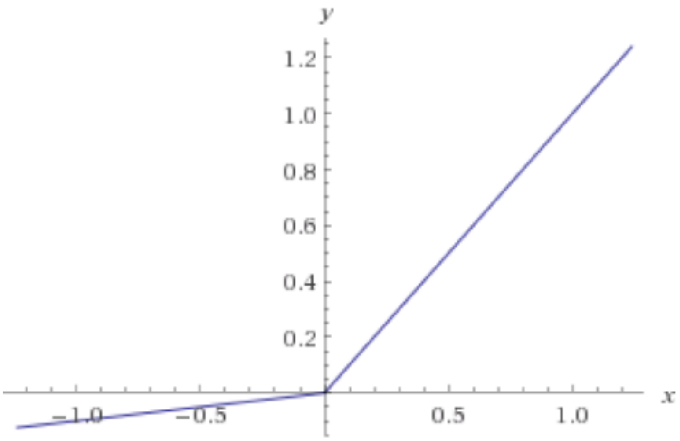


Obrázek 2.12: Funkce ReLU [29]

Slovně řečeno, vše, co má hodnotu 0 a méně, se změní na nulu (či jí zůstane) a vše, co je větší než nula, si svou hodnotu ponechá. Funkce má, jak může být patrné i z obrázku či předpisu funkce, jisté výhody a nevýhody. Například je možné s jistotou říci, že tato funkce je mnohem méně výpočetně náročná oproti předešlým funkcím, kde se vyskytovalo několik exponenciál, sčítání, odčítání a zlomků. Zde se pouze vynulují hodnoty menší než nula. Také bylo experimentálně zjištěno, že ReLU značně zrychluje proces hledání minima ztrátové funkce (pomocí stochastic gradient descent). Ve srovnání s hyperbolickým tangens či sigmoidou je velkým krokem dopředu to, že nám v kladné části neurony nesaturují.

Největší nevýhodou ReLU je nulování (zabíjení) neuronů, které bylo popsáno výše. Totiž všechny hodnoty menší než nula vynulují neuron a všechny následné konexe navazující na něj.[2, 12] Tento problém se řeší funkcí s názvem *Leaky ReLU*.

Leaky ReLU je funkce, která je od nuly stejná, jako ReLU, avšak přímce vedoucí zleva do nuly je dán mírný sklon (například $0.1x$) a tím se snažíme zbavit efektu „zabíjení“ neuronů.



Obrázek 2.13: Funkce Leaky ReLU [30]

Předpis takovéto funkce bychom mohli napsat do tvaru:

$$f(x) = max(ax,x), \tag{2.16}$$

kde a je hodnota námi zvolená podle toho, jaký sklon chceme přímce udělit.

Zde je místo pouze pro polemizování, zda se hledaný efekt dostavil, jelikož některé odborné práce říkají, že ano, některé to negují.

2.4 Předzpracování dat a správná velikost neuronové sítě

Pro ideální fungování neuronové sítě je potřeba navrhnout správnou síť a zároveň připravit data na vstup do sítě. Problematice přípravy dat a vhodným sítím se bude věnovat právě tato kapitola.

2.4.1 Správná velikost neuronové sítě a overfitting

S tím, co bylo doposud vysvětleno už může být jasné, že při návrhu neuronové sítě jde krom o správnost výsledku také o rychlost trénování. Tuto rychlost lze měnit na základě správné volby aktivačních funkcí, optimizátorů pro klesání podle gradientu a podobně. Rychlost trénování však jde ovlivnit i mnohem intuitivnější věcí, velikostí neuronové sítě. Logicky, pokud se bavíme o neuronové síti s dvaceti vrstvami o stovkách až tisících neuronech, tudíž v ní bude miliony a miliony parametrů, bude jedna iterace trénování trvat mnohem déle, než u sítě s dvěma vrstvami o dvaceti neuronech.

Toto je poměrně důležitá otázka před implementací dané sítě, jelikož pokud je zájem detekovat nejmenší skryté sémantické vzorce v datech, pak je chtěno, aby byla síť pokud možno co nejhlubší, avšak před námi vstane otázka, zda na to je k dispozici výpočetní výkon. [2]

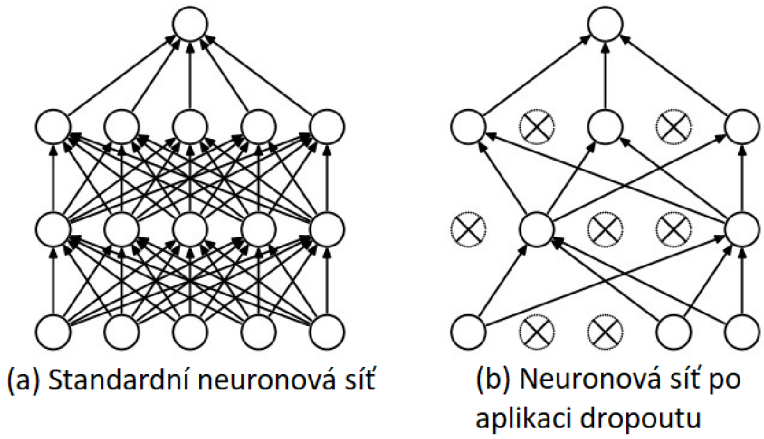
Dejme tomu, že máme k dispozici neomezený výpočetní výkon a síť lze mít hlubokou jak jen je možné. Na povrch vyvstane další problém. Tento problém se nazývá *overfitting*.

Overfitting znamená, že se neuronová síť až příliš uzpůsobí trénovacím datům a v momentě, kdy po síti bude požadováno vyhodnocení dat jiných než trénovacích, tak zcela selže i přes to, že na trénovacích měla skvělé výsledky. Tento problém si je možno představit tak, že síť má určitou kapacitu na „zapamatování“ dat (síť v podstatě nedělá nic jiného, než že si pamatuje všechny zjištěné rysy v datech). Problém nastane v momentě, kdy se tyto informace téměř nemění. To se může stát, je-li vůči velikosti sítě málo trénovacích dat. Tedy síť si většinu těchto dat separátně „zapamatuje“[2].

Jak proti takovémuto problému bojovat? Buď jde zmenšit hloubku sítě tak, aby byla dostatečně malá a její parametry se dostatečně měnily (učily se). Dále je možno zvýšit množství trénovacích dat. Zde samozřejmě záleží na tom, zda jsou tato data přístupná, nebo zda je možno je nějakým způsobem opatřit. [2]

Také se užívají různé regulační techniky. Jednou z nejznámějších a nejužívanějších je *dropout*. Dropout je vcelku jednoduchá a účinná metoda.

[2]Tento problém jde připodobnit ke studentům matematiky. Tzv. overfitting (co se lidí týče, pouze obrazně řečeno) se u studenta objeví, pokud se člověk naučí jednotlivé postupy příkladů ze sešitu od učitele bez jakéhokoliv porozumění. Kdežto student, který si příkladů projde víc, neučí se jen postupy, ale problému komplexně porozumí, pak nemá problém vyřešit příklady, které předtím sice nikdy neviděl, ale vztahují se k naučenému problému



Obrázek 2.14: Dropout. Převzato a upraveno z [31]

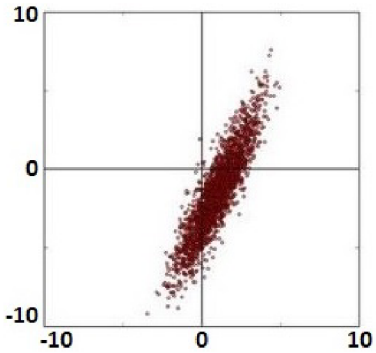
Při každé iteraci se vybere procento celkového počtu neuronů (dle naší volby, např. 30%) a vynulují se („zabijí“). V tento moment trénuje síť pouze ty konexe mezi neurony, které nebyly vynulovány. Po jednom průchodu se zabitým neuronům a vahám vrátí jejich předešlé hodnoty a znovu se náhodně zvolí procento neuronů, které se vynulují. Proces se opakuje po celou dobu učení.

Tímto způsobem je možné se efektivně zbavovat overfittingu. Tyto regulační techniky se používají víceméně při každém trénování (ať už za konkrétním účelem, či preventivně). [2]

2.4.2 Předzpracování dat

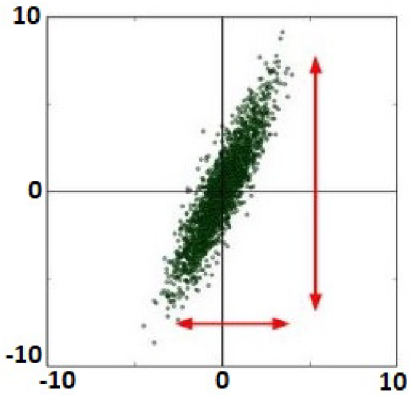
Jak již bylo několikrát zmíněno, data by před vstupem do sítě měla projít tzv. předzpracováním (anglicky *data preprocessing*). Bylo by výpočetně náročné, když by, dejme tomu každý obrázek, měl jiné rozměry, jiný počet pixelů a podobně. A vzhledem k tomu, že dnes můžeme dát jako vstupní data do sítě i jiné věci, např. celé texty knih, zvukové záznamy apod., tak je na místě, aby se data před vstupem do sítě znormalizovala.

Na obr. 2.15 jsou znázorněny hodnoty našeho pomyslného datasetu, jež se chystáme zpracovat.



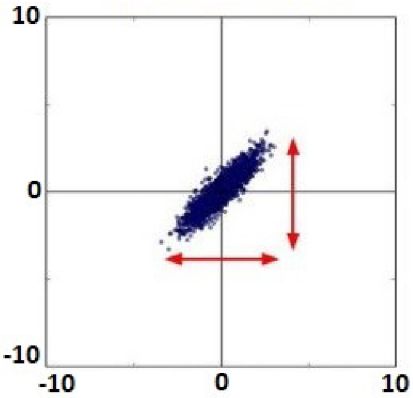
Obrázek 2.15: Původní data [31]

Jedním z druhů předběžného zpracování dat je tzv. *mean subtraction* (odečtení průměru). Tato běžná metoda zpracování dat je zcela jednoduchá. Vezme se celý dataset, spočítá se průměrná hodnota všech datapointů a ta se odečte od každého prvku. Tato metoda může být graficky znázorněna jako vystředění hodnot okolo počátku. [2]



Obrázek 2.16: Vystředěná data [31]

Další běžně užívanou metodou předběžného zpracování dat je *normalizace* dat. Normalizací je zde myšlena normalizace dimenzí dat tak, aby byli přibližně stejné. Ve 2D obrázku (obr. 2.17) si to lze snadno zobrazit. Této normalizace jde docílit například tím, že se každá dimenze vydělí její směrodatnou odchylkou (data by měla být nulou středěna). Data po vycentrování a normalizaci vypadají následovně. [2]



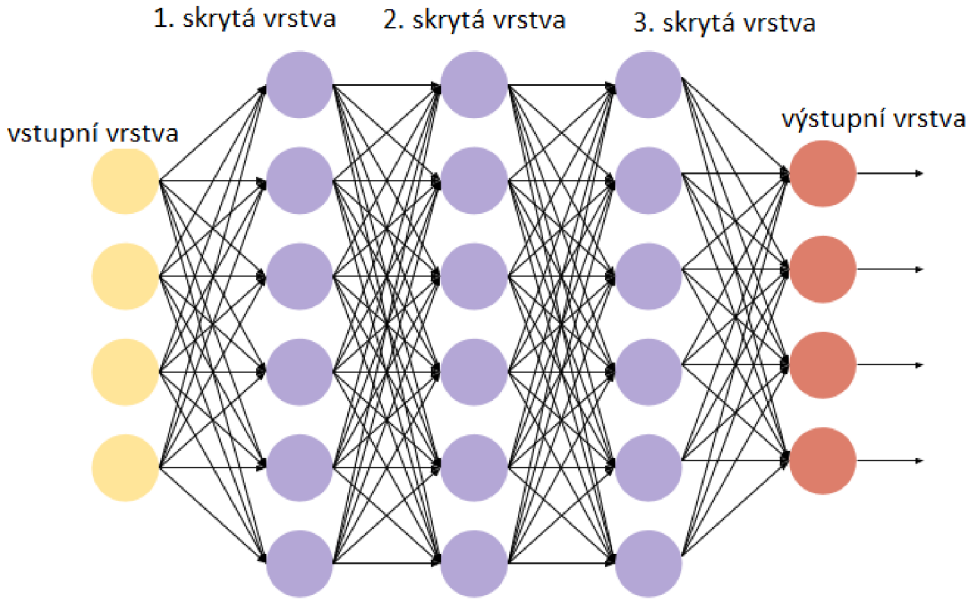
Obrázek 2.17: Vystředěná a znormalizovaná data [31]

2.5 Typy neuronových sítí

Když by bylo žádoucí vyjmenovat a popsat všechny typy neuronových sítí, ať už dle architektury či funkcionality, obsahově by to stačilo na samostatnou práci. Tato práce představí 3 nejrozšířenější, v praxi nejpoužívanější typy.

2.5.1 Zcela propojená neuronová síť

Zcela propojená neuronová síť (anglicky fully connected neural network, feed forward neural network, multilayer perceptron) je jedním ze základních konceptů hlubokých neuronových sítí.



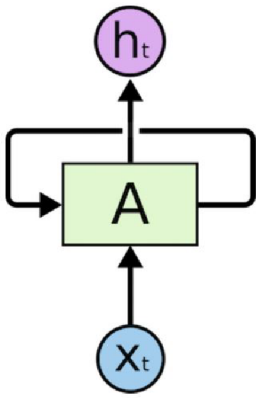
Obrázek 2.18: Zcela propojená neuronová síť. Převzato a upraveno z [32]

Tento typ sítí bude popsán stručně, jelikož pochody uvnitř sítě (směr proudění dat, trénování, backpropagation, stochastic gradient descent, loss function atd.) se zabývá celá podkapitola 2.2.

Jak si lze všimnout na obrázku 2.18, struktura se skládá ze vstupní vrstvy, n skrytých vrstev a výstupní vrstvy. Všechny neurony jsou mezi vrstvami propojeny se všemi. Tedy vstupní vrstva představuje data vstupující do neuronové sítě. Ve skrytých vrstvách jsou neurony představovány aktivačními funkcemi a probíhá zde veškeré učení. Výstupní vrstva vyhodnocuje výsledek (ať už pravda/nepravda, či pravděpodobnost s jakou síť data klasifikovala). Síť se zpravidla učí (trénuje) algoritmem zvaným backpropagation (viz podkapitola 2.2). Data zde proudí dopředně, tedy proudí směrem od vstupní vrstvy k výstupní vrstvě.

2.5.2 Rekurentní neuronová síť

Nejlépe lze základní model rekurentní neuronové sítě (anglicky *recurrent neural network*, RNN) představit na ilustračním obrázku.



Obrázek 2.19: Rekurentní neuronová síť [33]

kde x_t jsou vstupní data, A je samotná neuronová síť a h_t je výstup, upravené porozumění dat, skrytý stav (anglicky *hidden state*). U základních verzí RNN je skrytý stav roven výstupu.

Jak tedy RNN fungují? Síť přijme data, zpracuje je a vypočítá výstup, ten lze porovnat s chtěným výstupem (pomocí ztrátové funkce). Podle ztráty se upraví váhy a zároveň síť upraví své parametry. Data však neproudí do dalších vrstev, jako u dopředných sítí, nýbrž výstupy ze sítě do ní znovu vstupují (jak je znázorněno na obr. 2.19). Operace se znovu opakují. Skrytý stav lze popsat následující rovnicí:

$$h_t = (h_{t-1}; x_t) \tag{2.17}$$

Slovně řečeno, aktuální stav sítě vychází z dat předešlého stavu funkce (h_{t-1}) a aktuálních vstupních dat (x_t). To vše zpracováno matematickou funkcí, kterou lze definovat samotnou síť (např. aktivační funkcí, viz kapitola 2.3). [13, 14]

Pro konkrétní případ, kdy je neuronová síť definována pouze funkcí hyperbolickým tangens, by mohl výpočet h_t vypadat následovně:

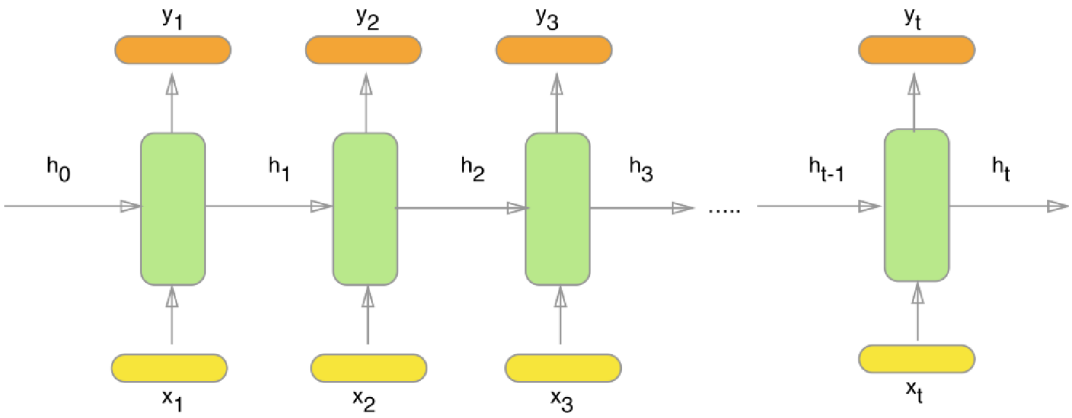
$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2.18}$$

Tedy aktuální stav je roven váze konexe W_{hh}, kterou mění síť sebe sama, vynásobena předchozím stavem h_{t-1}, sečteno s vahou W_{xh} vedoucí ze vstupu do sítě, vynásobené samotným vstupem x_t. To vše zpracováno funkcí, v tomto případě hyperbolickým tangens.

RNN má oproti spoustě typů neuronových sítí přednosti, kterých naplno využívá. Například nemá pevně danou velikost vstupu a zároveň není pevně daný počet matematických operací (jako u FCNN či CNN jsou operace pevně dané počtem vrstev a počtem neuronů ve vrstvách). Zároveň si ukládá (memoruje) části předešlých vstupů a využívá je k přesnějšímu vyhodnocení výsledku. Tím, co zde bylo sděleno, je mířeno na fakt, že se RNN hodí ke zpracování sekvenčních dat (i časově sekvenčních), což by u jiných typů neuronových sítí nebylo možné. Tedy lze jako vstup brát texty, zvuky, obrázky atd. v časovém

sledu a vpouštět je do sítě po sekvencích (bližší popis fungování sítě si představíme ještě v této kapitole). Díky těmto vlastnostem se RNN hodí například na rozpoznávání řeči, jazykové modelování, překlad textů. Jsme schopni „popsat obrázek“ (vstup je obrázek, výstup psaný text, který nám vstup slovně popíše), lze predikovat výsledky, či napovídat další slova či písmena v psaném textu. Tento výčet samozřejmě není konečný. [2, 13, 14]

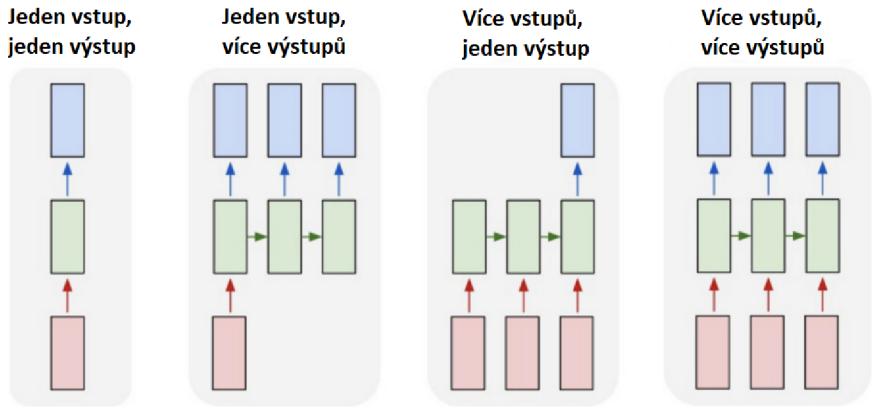
Fungování RNN se může na první pohled zdát složité, či chaotické. Ale když se výpočetní graf (computational graph) vykreslí tímto způsobem, tak se situace více objasní:



Obrázek 2.20: „Rozbalená“ rekurentní neuronová síť [34]

kde x_i jsou vstupy, h_i jsou funkce (skryté stavy) neuronové sítě a y_i jsou výstupy. Tedy když se každý stav sítě vykreslí jako jednotlivé výpočetní složky, je možné se v tom mnohem lépe orientovat (obr. 2.19 a 2.20 jsou ekvivalentní).

Na obr. 2.21 jsou uvedeny typy architektur RNN. Každá z nich se hodí na jiný úkon, avšak princip sítě zůstává stejný, pouze se mění počet vstupů do sítě a počet výstupů z ní.



Obrázek 2.21: Typy architektur RNN. Převzato a upraveno z [13]

Za zmínění také stojí proces učení. RNNs stejně jako konvoluční, nebo zcela propojené sítě využívají algoritmus backpropagation. Jediný rozdíl je ten, že se musí učit každý stav našeho modelu (jak je vidno na obr. 2.20). Tedy se spočítá ztrátová funkce pro každý

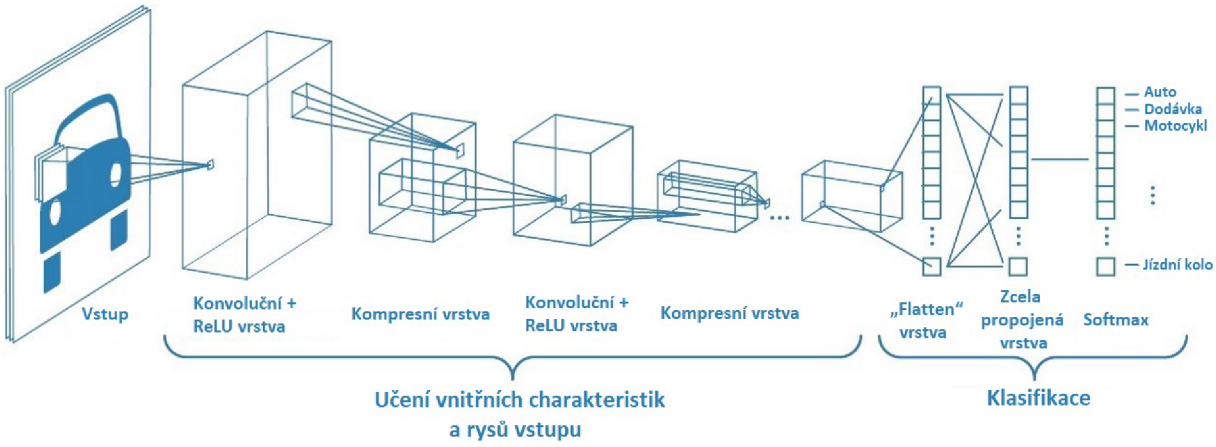
výstup $a_1, a_2, ...a_t$ (na obr. 2.20 značeno y) , následně se provede backpropagation každým stavem $h_1, h_2, ...h_t$ až na počátek (každý stav sítě má své váhy, tedy i vlastní gradient proti kterému je snaha váhy snižovat). Jednoduše řečeno, dělá se backpropagation od aktuálního stavu skrz všechny stavy až k počátečnímu. [2, 14]

Zde už je nastíněn problém, na který lze v praxi narazit. Takovýto výpočetní proces může být časově náročný. Proto se při trénování ve skutečnosti trénuje „po částech“. Tím je myšleno, že se provede x výpočetních operací, provede se backpropagation a pokračuje se v posílání vstupů do sítě a po dalších x operacích se přejde na proces učení atd. (podobně, jako se v praxi používá stochastic gradient descent, tak zde se také váhy neupravují s každým vzorkem, ale nechá se sítí projít více dat a úpravy probíhají až po projití desítek vzorků těchto dat). [13]

Jedním z největších problémů je „ztráta paměti“ rekurentní neuronové sítě. Tím je myšleno, že na síť už nemá takový vliv proces vykonaný před tisíci operacemi jako ten, který se právě vykonal. Proti tomuto problému se bojuje například sítěmi s LSTM (long short-term memory), kdy se do těla funkce přidává další pomyslná větev, kam se po vyhodnocení ukládají data, která síť dle určitých priorit přefiltruje a uloží).

Závěrem podkapitoly se hodí zmínit, že i rekurentní neuronové sítě bývají vícevrstvé. Počet vrstev je ale značně nižší než u jiných typů sítí, vzhledem k výpočetní náročnosti už jako takové. [13]

2.5.3 Konvoluční neuronová síť



Obrázek 2.22: Schéma konvoluční neuronové sítě. Převzato a upraveno z [15]

Už jen dle obr. 2.22 může být zřejmé, že konvoluční neuronová síť (anglicky convolutional neural network, CNN) se podobá zcela propojené síti (FCNN) více, než rekurentní neuronová síť (RNN). A ve skutečnosti také funguje do určité míry podobně jako FCNN. Data zde proudí dopředně, čili od vstupu k výstupu, také se zde používá ztrátová funkce, a trénuje se pomocí algoritmu backpropagation a rovněž se užívá stochastic gradient descent. A přesto se tato síť v mnoha ohledech liší a má značně lepší výsledky než FCNN.

Asi největší odlišností a zároveň základním prvkem CNN jsou jiné vrstvy, než které byly v této práci doposud představeny:

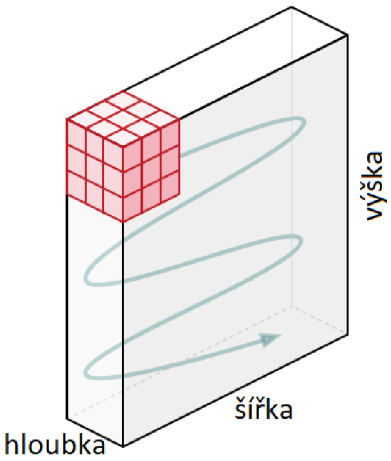
2.5.3.1 Konvoluční vrstva

Konvoluční vrstva (anglicky Convolutional layer) pracuje diametrálně odlišně od vrstvy z FCNN. Vstupem není vektor hodnot všech pixelů. Místo toho se zde operuje s filtry (kernely), skrz které vždy prochází části vstupu (např. obrázku). Jinak řečeno, tento filtr se pohybuje po ploše (objemu) celého obrázku. Na každé pozici filtru vůči obrázku se pronásobí hodnoty pixelů obrázku a filtru a to vytvoří dílčí výstupy z konvoluční vrstvy. Jeden výstup a je tedy možno napsat jako:

$$a = w^T x + b, \tag{2.19}$$

kde w jsou vlastnosti filtru (číselné hodnoty), které v kontextu konvoluční vrstvy lze vnímat jako váhy, jelikož právě tyto parametry se pomocí backpropagation upravují. Označení T zde znamená pouze to, že je tento vektor transponován, aby pronásobení se vstupními daty proběhlo korektně. Proměnná x jsou zde hodnoty vstupních dat a b je bias. Výsledkem této operace je jedno číslo, jelikož v operaci probíhá skalární součin a přičtení biasu. [2]

Pohyb filtru je znázorněn na obr. 2.23.



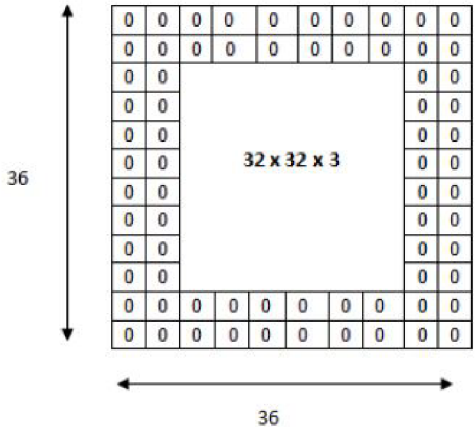
Obrázek 2.23: Pohyb filtru (kernelu) skrz vstup do konvoluční vrstvy. Převzato z [15]

Jak si lze povšimnout, vstup (obrázek) je na obr. 2.23 zobrazen trojrozměrně. Šířka a výška jsou jednoduše rozměry vstupu (v pixelech) a hloubkou je zde myšlen počet barev (RGB), tedy 3. Filtr má vždy stejnou hloubku jako vstup.

Filtr může mít rozdílné rozměry a pohybuje se jím různě velkými posuvy. Čím menší filtr je, tím více informací a prvků (vlastností) síť z dat rozpozná, ovšem nesmí se zapomenout na výpočetní výkon. Typická volba rozměrů filtru může být např. 3x3 či 5x5. Podobné pravidlo zde platí o posuvu. Čím menší posuv, tím více informací bude detekováno, avšak tím více operací se musí vykonat. Také nemá smysl používat větší posuv, než je rozměr filtru, „přeskakovali“ by se pixely a některá data ze vstupu by se vůbec nebrala v potaz, takže výsledky by nemusely správně vypovídat o vlastnostech vstupních dat na základě chybějících informací.[2, 15]

Je samozřejmě možné použít na jedné vrstvě několik filtrů. Opět, čím více filtrů, tím více vlastností se síť naučí, ale stojí to výpočetní výkon. Vždy je důležité brát tento aspekt v potaz.

Například, když se vezme jeden filtr o rozměrech 5x5x3 a aplikuje se na obrázek o rozměrech 32x32x3, celkovým výstupem bude tzv. *aktivační mapa*, která bude mít rozměry 28x28x1. Z tohoto příkladu je patrné, že výstupy se nám znatelně zmenšují, což je jev v některých případech chtěný, v některých naopak. Totiž menší rozměry zpracovaných dat se rovnají rychlejší zpracování sítí (méně početních operací), avšak ztrácíme zde určité množství vlastností ze vstupu. Tento problém se řeší metodou *zero padding* (volný překlad: obalení nulami).[2]



Obrázek 2.24: Zero padding [35]

Při tomto procesu se vstup doslova obalí prázdnými pixely (o hodnotě nula), tedy z pohledu vlastností se nic nezmění, jelikož síť s nimi nebude nijak pracovat, avšak při procesu to pomůže v tom, že se výstup rapidně nezmenší. Další velkou výhodou zero paddingu je fakt, že podporuje schopnost lépe číst data z okrajů obrázku. Při představě, jak filtr prochází obrázkem, je patrné, že na krajních pozicích čte informace znatelně méněkrát než uprostřed. Díky zero paddingu však i okrajem projde vícekrát, jelikož prochází i pozicemi, kde jsou prázdné krajní pixely. [2]

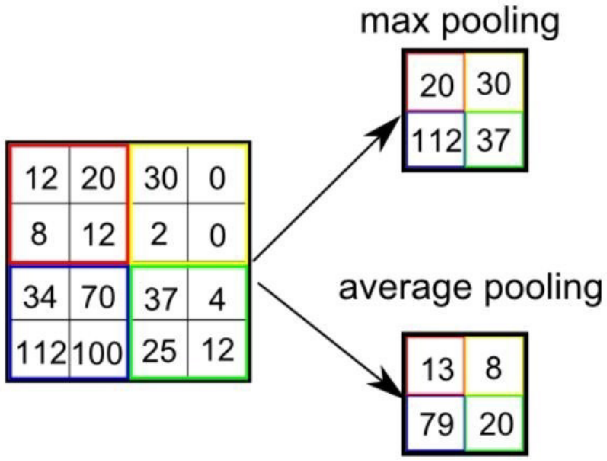
2.5.3.2 ReLU vrstva

Doteď se všechna pozornost upírala na konvoluční vrstvu. Jak také bylo zjištěno, neprobíhá v ní prakticky nic jiného než násobení a sčítání. Čili samé lineární operace. Jak bylo zmíněno v informacích o FCNN z podkapitoly 2.2, tak neuronová síť se bez nelinearit nemá šanci jakkoliv učit. Je tedy naprosto běžné, že za každou konvoluční vrstvou je vrstva s nelinearitou, například ReLU. ReLU má oproti jiným aktivačním funkcím spoustu výhod (i nevýhod), všechny důležité jsou už popsány v podkapitole 2.3. [2]

2.5.3.3 Kompresní vrstva

Tato vrstva (anglicky *pooling layer*) dělá částečně to, proti čemu se používal zero padding, tj. zmenšuje objem dat, který proudí sítí. Důvod je jednoduchý, jak bylo zmíněno, čím méně dat proudí sítí, tím rychleji vše probíhá a tím lépe se data spravují. Opět, je nesmírně důležité tento proces optimalizovat, aby se neztrácely důležité informace z původního obrázku. Na druhou stranu, pokud bude používáno několik vrstev s velkým množstvím

filtrů, pak je tato komprese nezbytná, jelikož s každým filtrem sice vznikne menší aktivační mapa, než je vstup do vrstvy, ale těchto map bude několik. Je tedy důležité, jak již bylo několikrát zmíněno, najít balanc mezi schopností sítě učit se a výpočetním výkonem. [2] Mělo by se také zmínit, jak tato komprese funguje. Například pomocí procesů *max pooling* či *average pooling*. Tento proces je principiálně jednoduchý. Vezme se opět pomyslný filtr a projede se s ním celý vstup do této vrstvy. Nyní se filtrem však nebudou pronásobovat všechny hodnoty na všech pozicích jako u konvoluční vrstvy, ale na každé pozici se vybere největší hodnota z oblasti vstupu, kterou tento filtr překrývá (nebo průměr hodnot, záleží zda se užívá average či max pooling). [2, 15]
Z obrázku to bude opět jednoduše pochopitelné:



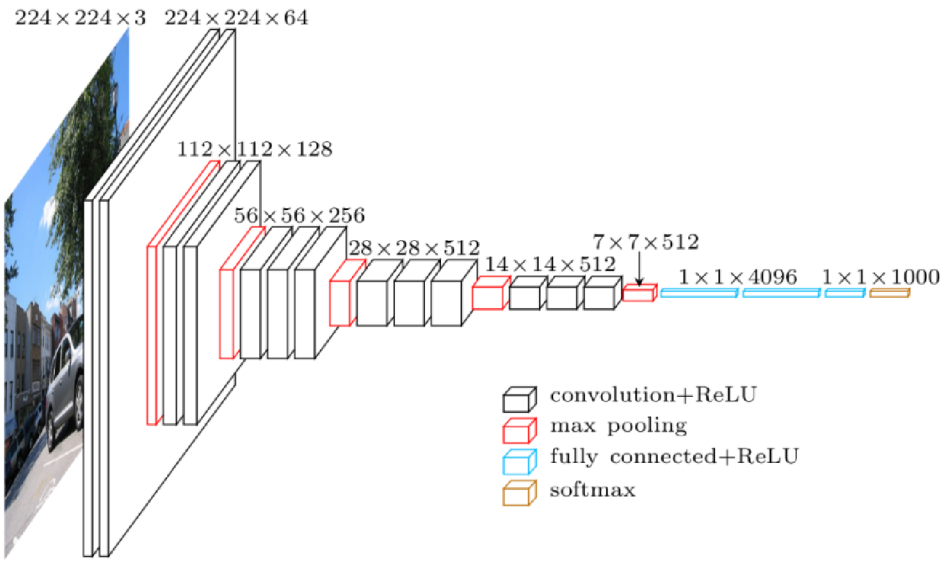
Obrázek 2.25: Max pooling, average pooling [15]

2.5.3.4 Zcela propojená vrstva

Jak už název napovídá, tato vrstva byla částečně představena. Pracuje totiž stejným způsobem, jako pracují vrstvy ve zcela propojené neuronové síti. Čili všechny neurony jsou propojeny se všemi vstupy a každý z nich vytvoří jeden výstup. Tyto výstupy poté z každého neuronu v této vrstvě míří do neuronů další vrstvy, či výstupní vrstvy, která po zpracování vstupů z předchozích neuronů již udává chtěný výsledek. Bližší informace o pochodech ve zcela propojené vrstvě se nachází v podkapitole 2.2. [2, 15]

Patří také říct, že před každou zcela propojenou vrstvou se nachází *flatten layer*. Ta má pouze jednu funkci. Data, která do teď měla dva až tři rozměry transformuje do jednoho vektoru, který je následující zcela propojená vrstva schopna vzít jako vstup.

Teď, když jsou popsány všechny možné druhy vrstev v konvolučních neuronových sítích, by se mělo zmínit typické uspořádání těchto vrstev, jelikož nyní to bude dávat lepší smysl, než když by se tato problematika představila na začátku kapitoly. Většinou se kombinují vrstvy v následujícím pořadí - např. jednou, dvakrát či třikrát dvojice konvoluční vrstvy následované ReLU vrstvou. Po této kombinaci standardně narazíme na kompresní vrstvu (pooling layer). Zmíněné pořadí vrstev se několikrát opakuje, než data narazí před koncem sítě na tzv. flatten layer následovanou fully connected layer. [2] Příklad takovéto architektury CNN je zobrazen na obr. 2.26:



Obrázek 2.26: Příklad architektury CNN (model VGG16) [36]

Konvoluční sítě mají výborné výsledky nejen v klasifikaci obrázků do různých tříd. Je možno s nimi obrázky nejen třídit, ale i hledat v rámci samotného obrázku několik objektů (detekce objektů), jsou schopny nejen tyto objekty nalézt, ale vytyčit obrys těchto objektů (segmentace objektů). Také je možné klasifikovat nejen obrázky, ale i celé texty či videa (například podle kontextu, zvuků, chování, prostředí atd.). Také lze dělat tzv. „image captioning“, což je proces, kde vstupem je obrázek, výstupem věta popisující obrázek.

Tyto možnosti jsou samozřejmě aplikovány na reálné problémy. Například je snaha implementovat sítě do samořídících se aut (zde může být využití například čtení značek rozpoznávání překážek, přechodů, chodců apod.). Dále lze pomocí těchto sítí rozpoznávat obličeje či otisky prstů (viz mobilní telefony). Také je s nimi možné kategorizovat celé galaxie, či zdravotní fotografie (rentgenové snímky, lze kategorizovat snímky dle nemocí) apod. [2]

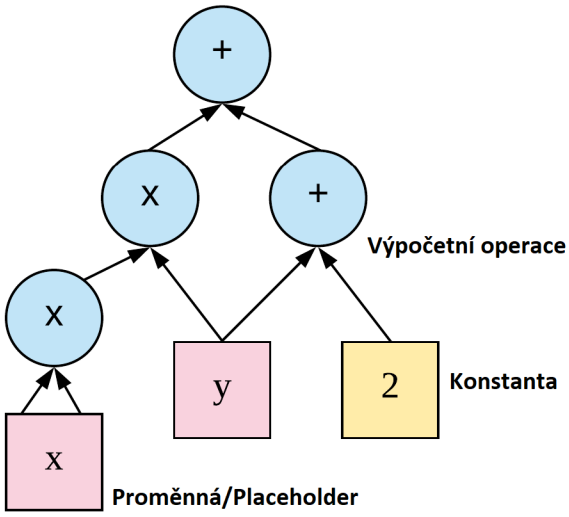
3 Nástroje pro hluboké učení a neuronové sítě

Sestavení programu, který bude obsahovat funkční výpočtový model konvoluční neuronové sítě, by bylo v nízkoúrovňovém programovacím jazyce obtížné nejen pro laika, ale i pro zasvěceného. Naštěstí pro širokou veřejnost bylo vytvořeno už několik vysokoúrovňových softwarových knihoven, které vytváření, trénování, nebo jen používání modelů (nejen) konvolučních neuronových sítí značně usnadňují. V této kapitole budou představeny knihovny, které byly použity při vytvoření praktické úlohy této bakalářské práce, přenosu uměleckého stylu pomocí konvoluční neuronové sítě.

3.1 TensorFlow

TensorFlow (TF) je open-source knihovna vhodná nejen pro machine learning. TF byl vytvořen společností Google. Má v sobě zabudované složité matematické operace a jak už název napovídá, vyznačuje se prací s tenzory (tenzory jsou popsány vzápětí). Poté, co jej začali lidé používat pro vytváření modelů neuronových sítí a práci s nimi, začal se i TF samotný přizpůsobovat k práci s neuronovými sítěmi a stal se dobrým nástrojem pro deep learning.

Každý program napsaný v TF je založen na vytvoření a následném použití tzv. výpočetního grafu (computational graph). Tento graf si lze představit jako síť pospojovanou uzly (každý uzel představuje operaci, od jednoduchého sčítání a násobení po složité funkce). Každá operace může mít jako výstup jedno číslo, vektor, matici, či tenzor. Tenzor si můžeme připodobnit k vícedimenzionální matici, tedy dvoudimenzionální tenzor je matice o rozměrech $n \times n$. [16]



Obrázek 3.1: Vizualizace jednoduchého výpočetního grafu. Převzato a upraveno z [37]

Na obr. 3.1 je znázorněn jednoduchý výpočetní graf se základními matematickými operacemi. Tento graf lze mimochodem přepsat do funkce $f(x,y) = x^2y + y + 2$.
Práce s výpočetními grafy probíhá následovně. Vytvoří se výpočetní graf. Následně se vytvoří a spustí tzv. session (výpočetní relace), čímž se spustí chod grafu, tedy od začátku (od vstupů) proběhnou v návaznosti všechny operace až po konec grafu. Jak je možno si všimnout už z obrázku, výpočetní graf v TF se vyznačuje schopností provádět operace paralelně. Výpočetní graf tedy slouží k vytvoření matematického modelu, sám o sobě nic nepočítá a neprovádí žádné operace. Od toho je zde session. [16]

TF také rozlišuje konstanty a proměnné. Jako v každém jiném programovacím jazyce, konstanty jako takové není možné žádnou operací změnit, kdežto proměnné ano. Nicméně zde figuruje třetí druh objektu, jenž drží určité hodnoty, tzv. placeholder. Placeholder je objekt, proměnná, které není hodnota přiřazena při vytváření grafu, jako u konstant, nýbrž až v momentě, kdy se spouští výpočetní relace. Dalo by se říct, že jsou to vstupní parametry, jež musí uživatel vyplnit při spuštění výpočetního procesu. [16]

Po představení základních elementů fungování knihovny TF lze demonstrovat tyto prvky na jednoduchém a objasňujícím příkladě. Příklad bude inspirován obrázkem 3.1, tedy rovnicí $f(x,y) = x^2y + y + 2$.

```
import tensorflow as tf

# We define constants, placeholders and variables, which will take part
# in the graph.
x = tf.placeholder(tf.float32)  # Placeholder x
y = tf.placeholder(tf.float32)  # Placeholder y
c = tf.constant(2.0)            # Constant c

# Building the graph.
multiplied_x = x*x   # First operation on the left flow
multiplied_xy = multiplied_x*y  # Second operation on the left flow
summed_yc = y + c    # First operation on the right flow
result = multiplied_xy + summed_yc  # Final operation in the graph

# Defining TensorFlow session and running comp. graph through the session.
sess = tf.Session()  # Defining session
print(sess.run(result, feed_dict={x: 7, y: 5}))  # Running the session
                                                 # (= executing graph)
                                                 # Feeding dictionary
                                                 # with placeholders
# Result 42
```

3.2 Keras

Srovnání frameworků Keras a TF lze připodobnit k programovacím jazykům Python a C. Jazyk C je náročný, poměrně nízkoúrovňový jazyk, který má ale ty výhody, že uživatel je schopen napsat vše od nuly a sám. Python na druhou stranu je značně jednodušší, dělá spoustu věcí za uživatele, má více vestavěných funkcí a knihoven, zato je o to pomalejší a pro zkušeného vývojáře ne tak flexibilní. V tomto případě lze přirovnat TF k jazyku C

a Keras k Pythonu. Koneckonců, Keras je v Pythonu napsán, kdežto TF je psán v C++ (naštěstí pro spoustu uživatelů je TF svázán i s pythonem, takže se nemusí užívat jazyk C++). Keras také používá TF jako backend, tedy oba jazyky se dají v jednom programu kombinovat (lze je nejen kombinovat, Keras jako takový je již v TF zabudován a lze ho volat z knihovny příkazem tf.keras, jako každou jinou knihovnu. Je tedy možné Keras používat separovaně, ale i přímo v TF programu).

Keras je pro uživatele (především začátečníka) mnohem stravitelnější a intuitivnější. Je specializovaný na deep learning (tedy neuronové sítě), tudíž disponuje více vestavěnými funkcemi s nimi spojenými. Například zde najdeme předdefinované aktivační funkce, optimizátory SGD, vrstvy neuronových sítí a další příkazy na snazší zpracování obrázků a textu (jelikož s tím neuronové sítě pracují obzvlášť často).[17]

Názorná ukázka (následující program), jež zobrazuje jednoduchost kódu v Kerasu ledacos objasní. Jedná se o model konvoluční neuronové sítě o čtyřech konvolučních vrstvách určen pro klasifikaci obrázků. V případě nejasností ohledně termínů užitých v této části práce je nutno se vrátit do kapitoly 2.

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Activation, MaxPooling2D,
                         Dropout
# First part of the code
# Building the model
model = Sequential()
model.add(Conv2D(64, kernel_size=3, padding='same', activation='relu',
                 input_shape=(28, 28, 3)))
model.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(32, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Second part of the code
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Third part of the code
# Compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model on training data
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)
```

```
# Evaluate scores on test data
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Program je pro přehlednost jeho popisu rozdělen na tři části.

Na první části kódu se nachází, krom importování následně použitých příkazů z knihovny Keras, vytvoření modelu neuronové sítě a následující přidávání jednotlivých vrstev. Jak je znázorněno, implementace je intuitivní, pouze metodou *.add* (čili přidat) připojíme k síti určitou vrstvu jedním řádkem se všemi potřebnými specifikacemi. Například první přidaná vrstva je vrstvou konvoluční o 64 filtrech, tyto filtry mají rozměry 3x3. „Same“ padding se týká tzv. *zero paddingu* (viz kapitola 2), který se nastaví tak, aby výstup z vrstvy měl stejné rozměry, jako vstup do vrstvy. Následně je ve vrstvě už zakomponována nelinearita ReLU (dalo by se říct samotná vrstva) a poslední specifikací při volání *Conv2D* je vstup do vrstvy, který zde musíme definovat vůči rozměrům vstupních obrázků (zde je vstup nastaven pro obrázky o rozměrech 28x28 o třech barvách RGB). Po dvou vrstvách konvolučních (+ ReLU) je zakomponována vždy vrstva kompresní, ve které užíváme *max pooling* s filtry o rozměrech 2x2. Posledním prvkem je zde dropout nastaven na 25%.

Nyní se pozornost přesune k druhé části programu. Jak již bylo několikrát zmíněno, konvoluční neuronové sítě jsou zakončeny zcela propojenými vrstvami (fully connected layers). Tak tomu je i zde, *dense layer* je stejné označení pro zcela propojenou vrstvu. Dense layer standardně provádí klasifikaci. Nachází se zde tedy 2 vrstvy, jedna vrstva o 512 neuronech, druhá o stejném počtu neuronů jako je počet tříd, do kterých kategorizujeme vstupy. *Flatten* pouze změní dimenze proudících dat (na jednorozměrné pole), jelikož, jak bylo zmíněno v kapitole 2, zcela propojené vrstvy vyžadují takto zpracované vstupy. *Softmax* je aktivační funkce, která výstup zpracuje do pravděpodobnostního rozdělení, výstupem nebudou nahodilá čísla a nebude se vybírat to největší, ale softmax to přepočítá proporcionálně do procent. Dál se v 2. části kódu nachází dobře známé věci jako ReLU vrstva či dropout.

V poslední, třetí, části se nachází tři řádky. První řádek kompiluje výše vytvořený model. Specifikují se zde optimizer (používají se efektivnější optimizátory pro sestup proti gradientu), ztrátová funkce a co se měří, zde přesnost klasifikace.

Další příkaz je vše, co je potřebné k natrénování modelu. Jako parametry bere vstupní trénovací data, volitelně i testovací data a počet epoch, tedy kolikrát celý dataset projde sítí.

Pomocí posledního příkazu se nechají projít sítí testovací data (tedy obrázky, které síť ještě nikdy neviděla) a ověří se její schopnost kategorizovat obrázky.

Jak je zobrazeno na ukázkovém kódu, Keras je uživatelsky velice přívětivý. Na dvaceti řádcích se dá nadefinovat model neuronové sítě o čtyřech konvolučních vrstvách, natrénovat ho a ověřit naučené schopnosti na testovacích datech.

4 Přenos uměleckého stylu pomocí konvoluční neuronové sítě

Úkolem této práce je přenést umělecký styl obrázku (nebo typické rysy obrázku, jak bude znázorněno, nemusí se vždy jednat o uměleckou tvorbu) na jiný. Jak si lze všimnout, v uměleckém díle je možné oddělit dvě složky. První je složka obsahová (anglicky content), tato složka vyjadřuje faktické informace o obrázku (na obraze je člověk, příroda, město, obraz vyjadřuje nějakou aktivitu). Druhá složka je stylová (anglicky style), tato složka popisuje uměleckou stránku obrázku (obraz je namalován dlouhými vlnitými tahy, tečkami, obraz je v tmavých barvách). U přenosu uměleckého stylu se tedy využije obsahové složky vzhledem k obrázku, jejž chceme „přemalovat“ a stylové složky z obrázku obsahující umělecký styl, které chceme na již zmíněný obrázek nanést. Dalo by se říci, že tohoto přenosu uměleckého stylu nevědomě využívají i umělci, kteří se snaží zobrazit nějaké skutečnosti v určitém uměleckém stylu (kubismus, expresionismus, impresionismus apod.). Při řešení tohoto úkolu se ovšem nevyužije schopností šikovného umělce, nýbrž konvoluční neuronové sítě. Právě konvoluční neuronové sítě jsou schopny rozpoznat a interpretovat určité rysy a vlastnosti obrázků (tyto sítě lze natrénovat i tak, že budou schopny rozpoznávat obrazy podle uměleckých stylů). Zmíněná schopnost se perfektně hodí pro řešený úkol přenosu uměleckého stylu. V podstatě problém řešíme následovně. Do procesu jdou 2 vstupy. První vstup je obrázek uměleckého stylu a druhý vstup je obrázek, na který se umělecký styl aplikuje. Výstupem je obrázek, který je, dá se říct, mix vstupujících obrázků. Tedy ponechá si obsah měněného obrázku, vizuálně však vypadá jinak, jelikož se na něm projeví umělecký styl (změny barev, či specifických tvarů, tahů apod.).

Obrázek 4.1: Ukázka vstupů a výstupu ze sítě [38][39]

Jak už bylo zmíněno, k řešení tohoto problému bude použita konvoluční neuronová síť. Síť už musí být natrénovaná, jelikož je nezbytné, aby byla schopna sémantický obsah detekovat a z obrázku separovat. Může být natrénovaná například na rozpoznávání obrázků (neboli image recognition).

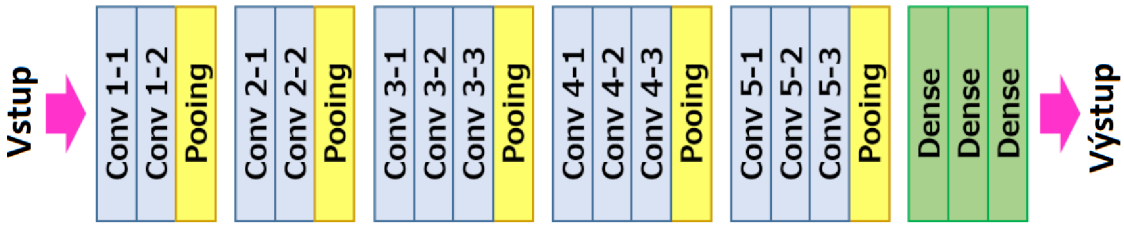
Také určitě stojí za zmínku, že čím hlouběji se v síti data nachází, tak tím menších detailů si síť všímá. Tedy v prvních vrstvách síť detekuje větší celky v obrázku, kdežto ve vrstvách hluboko v síti nejmenší detaily. Tento fakt se zanedlouho využije.

Při přenosu uměleckého stylu lze prohlásit některé vrstvy neuronové sítě za tzv. „style layers“ (vrstvy, které budou optimalizovat výstup z pohledu uměleckého stylu) a „content layers“ (vrstvy, které budou zachovávat obsah původního obrázku). S tím, co bylo zmíněno, je jasné, že content layers je důležité volit z prvních vrstev sítě, jelikož z obsahového hlediska je požadováno celkovou strukturu obrázku ponechat. Style layers se volí jak z vrstev horních, tak z vrstev hluboko v síti, jelikož umělecký styl se projeví jednak z celku, ale také z detailů. Je tedy rozdíl, které style layers z neuronové sítě se využijí, pokud by se využívaly pouze vrstvy hluboko v síti, tak síť nezohlední větší prvky v obrázku a výstup bude více jednolitý, chaotický, kdežto s využitím i vrstev na začátku sítě bude obraz více vyhlazený a přechody mezi objekty hladší. Tyto dva druhy vrstev jsou oddělitelné. To znamená, že je možné s obsahem z původního obrázku a s obsahem z uměleckého stylu manipulovat nezávisle na sobě.[18, 19]

4.1 Použitá neuronová síť

Při řešení úkolu se používá modelu konvoluční neuronové sítě s názvem *VGG16*. Tato síť byla navržena dvojicí K. Simonyan a A. Zisserman z Oxfordské univerzity a dosahuje výborných výsledků při klasifikaci, například na databázi ImageNet (ImageNet je dataset aktuálně obsahující přes 14 milionů obrázků, které jsou roztříděny do 1000 tříd). Síť zde dosahuje úspěšnosti klasifikace až 92,7%. [20]

Struktura VGG16 je zobrazena na obr. 2.26, nebo zde na obr. 4.2.



Obrázek 4.2: VGG16. Převzato a upraveno z [40]

VGG16 má 16 vrstev (počítají se pouze vrstvy, které mají parametry, pomocí kterých se síť učí). Užívá uspořádání 2-3 konvoluční vrstvy (v konv. vrstvách je zde už zahrnuta vrstva ReLU) následované jednou kompresní vrstvou (pooling layer). Síť je zakončena zcela propojenými vrstvami, které klasifikují a produkují výstup ze sítě. Síť také v konvolučních vrstvách používá filtry (kernely) s rozměrem 3x3 a v kompresních vrstvách filtry 2x2.[20] Více informací o jednotlivých vrstvách, filtrech a pochodech v síti je nutno se vrátit do kapitoly 2.

Tento model má samozřejmě i svá negativa. Má obrovský počet parametrů, kterými se učí (přes 138 milionů). Takže natrénovat dostatečně takto hlubokou síť trvá týdny i s dobrým výpočetním výkonem. V tomto případě se ovšem použije již předtrénovaného modelu, ve kterém jsou všechny naučené parametry uloženy.

4.2 Samotný přenos uměleckého stylu

Jak bylo zmíněno na začátku této kapitoly, vstupem zde jsou dva obrázky. Obrázek, který je měněn dle uměleckého stylu a obrázek obsahující umělecký styl. Výstupem je obrázek původní, na kterém je aplikován umělecký styl. Tento obrázek se však musí prvně vytvořit. Na začátku tedy je prázdný vybělený obrázek, který se bude postupně měnit do finální podoby výstupu. Přenos stylu v podstatě probíhá následovně:

1) Vytvoření/ nahrání modelu VGG16
2) Vytvoření ztrátových funkcí
3) Nahrání vstupních obrázků
4) Průchod vstupů neuronovou sítí
5) Výpočet ztrát[1] výstupního obrázku vůči vstupním obrázkům
6) Výpočet gradientu ztrátových funkcí všech parametrů
7) Změna obrázku proti gradientu ztrátové funkce
8) Opakování bodů 4-7, po dosažení požadované kvality obrázku

Zkrátka řečeno, je zde snaha o minimalizaci ztrátové funkce proti gradientu vůči vstupním obrázkům a podle toho se mění výstupní obrázek, který se s každou iterací podobá více oběma vstupním obrázkům.[18]

Bod číslo 5 si zasluhuje větší pozornost. Po tom, co se do sítě vloží vstupní obrázky a síť je zpracuje, přechází se do fáze výpočtů ztrát. Tyto ztráty se ovšem neodečítají z výstupu sítě jako takové (hodnoty pravděpodobnosti klasifikace obrázku do tříd), ale z jednotlivých vrstev. Tedy *content loss* se počítá jako kvadratická chyba mezi hodnotami, které se odečtou na dané vrstvě při projití vstupního obrázku původní fotografie a hodnotami výstupního obrázku.

Style loss je ještě o trochu složitější. Funguje v podstatě stejně, jako *content loss*, avšak při výpočtu ztráty se neberou v úvahu přímo hodnoty odečtené na vrstvách, ale z těchto hodnot se vytvoří Gramova matice (tato matice se získá jednoduchým způsobem, tedy vynásobí se vektor hodnot na vrstvě sebou samým, jen transponovaným. Tedy rovnice pro Gramovu matici vypadá následovně: $\mathbf{G} = \mathbf{x}\mathbf{x}^{\mathbf{T}}$, kde $\mathbf{x}$ je vektor hodnot na dané vrstvě, $\mathbf{x}^{\mathbf{T}}$ je vektor hodnot na dané vrstvě transponován.). Gramova matice se používá z toho důvodu, že díky ní je možné vypozorovat, které prvky se spolu zároveň „aktivují“ (myšleno, že mají velkou hodnotu, tedy mají velký dopad na umělecký styl), a které ne. Takže hodnoty v matici blízké nule tolik umělecký styl neovlivňují a naopak.

Vypočtené ztráty se následně sečtou a tím vyjde výsledná kombinovaná ztráta výstupního obrázku od vstupních obrázků. Následně už jen TensorFlow vypočítá gradient ztrátových funkcí a změní se výstupní obrázek proti směru tohoto gradientu.[19, 18]

[1] Ztráta je vypočtena ztrátovou funkcí, v tomto případě střední kvadratická chyba (mean squared error, čili MSE): $MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$, kde N je počet data pointů, f_i je vypočtená hodnota a y_i je chtěná hodnota jednotlivých hodnot.

4.3 Optimalizace přenosu uměleckého stylu

Po pochopení pochodů v přenosu uměleckého stylu práce teprve začíná. Tento proces je třeba optimalizovat tak, aby na výstupu byl obrázek, který co nejvěrohodněji ponese umělecký styl, jenž je na něj nanášen. Toho lze docílit změnami na různých částech neuronové sítě i procesu.

Lze:

- změnit *výpočet a využití jednoduchého gradientu za lepší metodu výpočtu tohoto gradientu* (za využití optimizérů jako ADAM, SGD, Adagrad apod.).
- měnit *počet iterací přenosu stylu*. Může vypadat, že čím více iterací, tím lépe. To však platí jen do určité míry. Ve chvíli, kdy se doiteruje do momentu, ve kterém se ztrátová funkce ustálí na minimu, tak se s dalšími iteracemi obrázek nemění. Změna počtu iterací je tedy otázka optimalizace výpočetního výkonu s časem, nikoliv přenosu samotného stylu.
- měnit *hodnotu jednoho kroku (step size)*. To je podobné jako při výpočtu gradientu. Čím menší je hodnota kroku, tím je přesnější změna obrázku, ale o to pomalejší. Tato optimalizace jde ruku v ruce s počtem iterací.
- změnit *výběr content/style layers*. Sice je známo, kde je dobré je volit, ale zároveň není k dispozici přímý náhled do vrstev, pochodů, nebo sítí rozluštěných rysů sémantického obsahu uměl. stylu, takže se zde projeví velký prostor pro experimentaci s výběrem těchto vrstev.
- jak bylo zmíněno na začátku kapitoly, lze separovaně a přímo ovlivňovat *content* i *style layers*, takže je možné jednoduše proporcionálně *pronásobit váhy jednotlivých druhů vrstev*. Například, když se pronásobí všechny váhy *style layers* konstantou o hodnotě 20 a váhy *content layers* například jedničkou, tak na výsledném obrázku umělecký styl doslova přebije původní obrázek. Chceme tedy najít určitý balanc, kde se dostatečně uměl. styl nanese na původní obrázek, ale zároveň ho nezdeformujeme příliš.

Následně bude demonstrována změna výstupu v závislosti na změně parametrů. Vstupem jsou stejné obrázky jako na obr. 4.1. Význam parametrů pod obrázky: čísla content a style weights jsou koeficienty, jimiž jsme pronásobili všechny content či style váhy. Iterations je počet iterací. Step size je hodnota jednoho kroku. Content a style layers jsou vrstvy v neuronové síti, které jsme vybrali jako stylové či obsahové vrstvy.

(a) content weights: 1,5; style weights: 10;
iterations: 100; step size: 10;
content layer: 10; style layers: 1-13

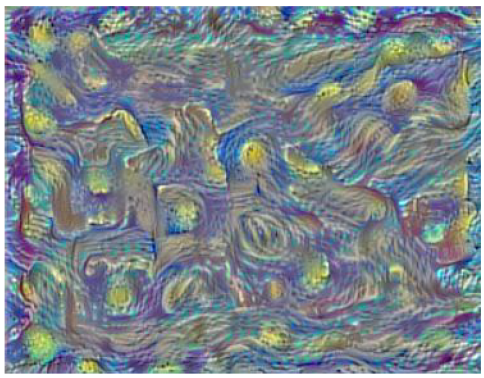
(b) content weights: 1,5; style weights: 10;
iterations: 120; step size: 10;
content layer: 7; style layers: 1-13

Obrázek 4.3: První dvojice obrázků

Na obr. 4.3(a) je patrné, že umělecký styl začíná „přebíjet“ původní obsah obrázku. Zapříčiněno to je především zvolením obsahové vrstvy hluboko v síti (content layer je desátá vrstva ze třinácti, není již schopna uchovat původní rysy objektů).

Obr. 4.3(b) vypadá značně lépe než obr. 4.3(a). Stačilo přitom posunout obsahovou vrstvu na 7. od začátku sítě. Obrázek stále nese nedostatky, ale dalo by se říct, že je nakročeno správným směrem.

(a) content weights: 1,5; style weights: 13;
iterations: 100; step size: 1;
content layer: 4; style layers: 1-13



(b) content weights: 1,5; style weights: 12;
iterations: 100; step size: 3;
content layer: 4; style layers: 5-13

Obrázek 4.4: Druhá dvojice obrázků

Na obr. 4.4(a) byl proveden pokus o optimalizaci, výstup však není uspokojivý. Myšlenka byla posunout obsahovou vrstvu opět výš v síti (content layer: 4) a zároveň navýšit váhy stylových vrstev (style weights: 13). Obrázek však vypadá zplihle a rozmazaně.

Na obr. 4.4(b) je typická ukázka toho, když umělecký styl absolutně přebije původní obsah. Zapříčiněno je to převážně volbou stylových vrstev, které se tentokrát experimentálně zvolily od páté po nejhlubší (třináctou), ale i volbou hodnoty pronásobení vah stylových vrstev, jenž byly oproti původní hodnotě 10 zvednuty na 12. S tím, že byla stylovým vrstvám oproti obsahovým přidělena větší důležitost a že ve stylových vrstvách nefigurují ty, které nesou umělecký styl z „větší perspektivy“ (nesou pouze drobné prvky uměleckého stylu, jak bylo dříve v této kapitole řečeno, na začátku sítě vrstvy pracují

s celými objekty a obrázkem jako celkem a čím hlubšími vrstvami síť s obrázkem pracuje, tím drobnějšími detaily se zabývá), byl po aplikování přenosu stylu původní obsah zcela nabourán.

(a) content weights: 1,5; style weights: 12; iterations: 100; step size: 3; content layer: 2,3,4; style layers: 2-11

(b) content weights: 1,5; style weights: 12; iterations: 100; step size: 3; content layer: 4; style layers: 1-13

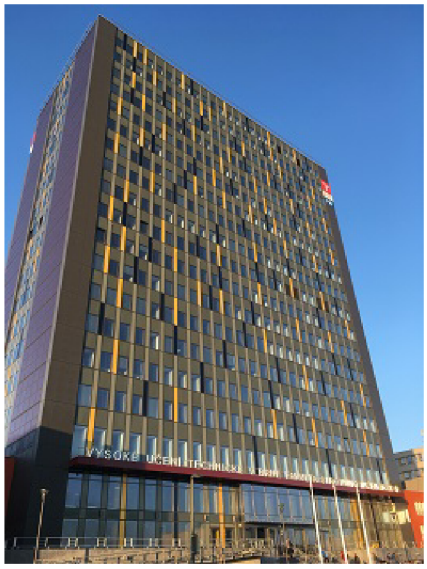
Obrázek 4.5: Třetí dvojice obrázků

Na obrázku 4.5(a) se nachází nevydařený experiment. Myšlenkou zde bylo zvolit více obsahových vrstev (2,3,4) pro dobré zachování původního obsahu. Dále zvolení hodnoty pronásobení stylových vah 12 a vynechání vrstvy číslo jedna ze stylových vrstev pro ignorování největších sémantických obsahů z uměleckého stylu. Tento experiment nevyšel zcela optimálně.

Posledním, nejúspěšnějším, příkladem pokusu o přenos uměleckého stylu z obrazu Starry Night na fotografii Telče byl obrázek č. 4.5(b). U tohoto pokusu byly vráceny hodnoty měnitelných parametrů blízko k těm, u kterých se začínalo. Je také důkazem toho, že sebemenší změna těchto parametrů může zapříčinit znatelnou změnu u výstupu. Tento výsledek přenosu uměleckého stylu věrohodně přenáší styl z původního obrazu: barvy byly adekvátně přeneseny, obloha je vířivá, u světlých mraků byla tendence vytvořit žluté kruhy (představující hvězdy) a domy jsou lehce vlnité, podobně jako městečko v pozadí obrazu. To vše za zachování původního obsahu fotografie.

4.4 Grafické zobrazení průběhu přenosu uměleckého stylu

Tato podkapitola slouží k ukázce průběhu přenosu uměleckého stylu. Prozatím byla řečena spousta informací o tom, jak přenos stylu probíhá a funguje, co ho ovlivňuje, informace o vstupech a výstupech, ale praktický průběh bude ukázán až nyní, po potřebném vstřebání všech informací.
Vstupem do pokusu budou tyto obrázky:



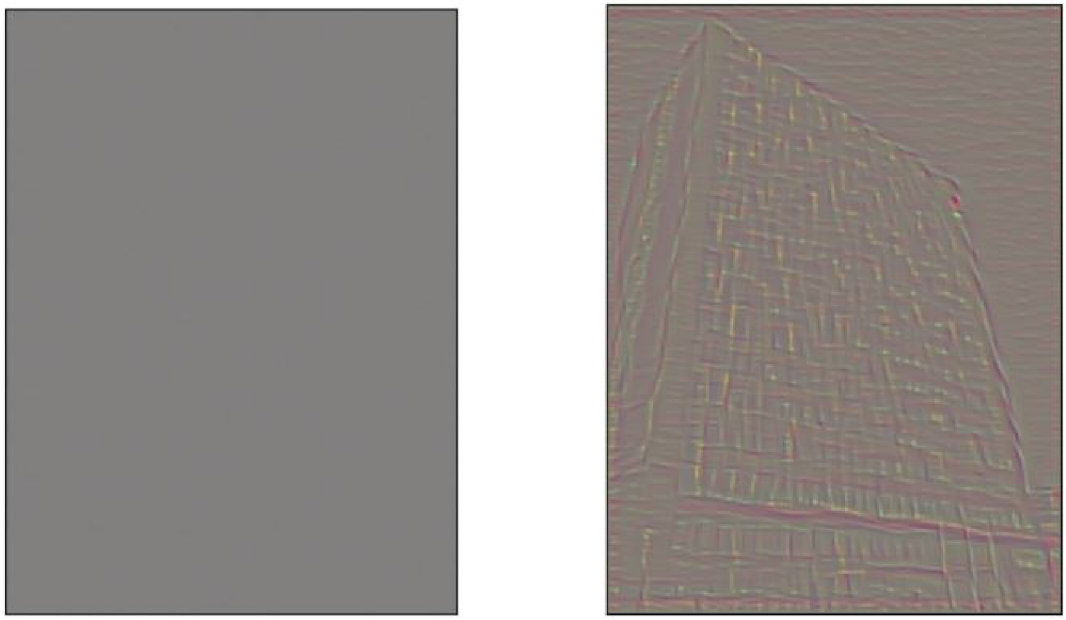
(a) Výšková budova A1, VUT FSI [54]

(b) Edvard Munch, Výkřik [53]

Obrázek 4.6: Vstup do přenosu uměleckého stylu

Nastavení parametrů před pokusem je (pozn.: význam parametrů je stejný, jako v předchozí podkapitole 4.3):

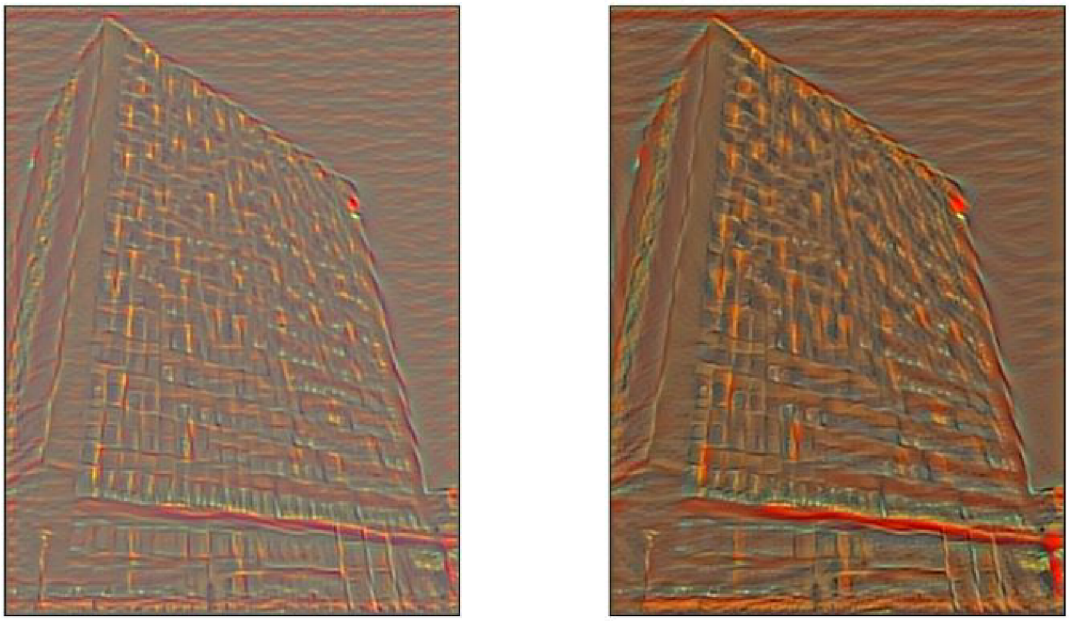
- content weights: 2
- style weights: 10
- iterations: 400
- step size: 1
- content layer: 4
- style layers: 1-13



(a) 0 iterací (b) 20 iterací

Obrázek 4.7: Demonstrace průběžného výsledku č. 1

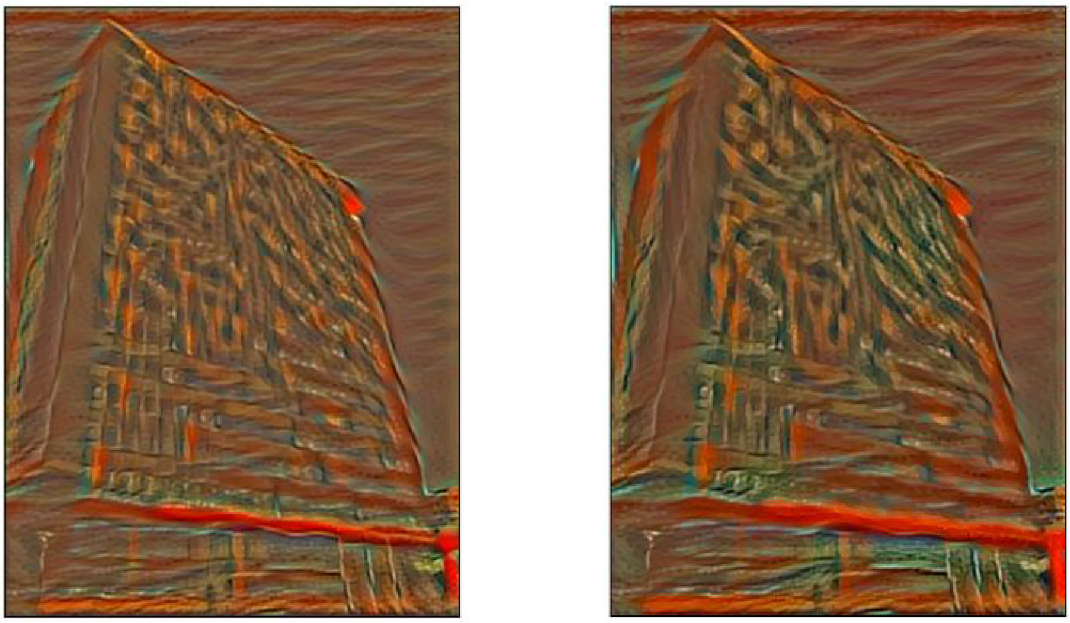
Na prvním obrázku s nula iteracemi je možné vidět, na čem se začíná- vybělený, barevně neutrální obrázek.
Po dvaceti iteracích už jsou jasně vidět obrysy budovy i s barevným rozlišením (u výrazných prvků budovy).



(a) 40 iterací (b) 60 iterací

Obrázek 4.8: Demonstrace průběžného výsledku č. 2

Po čtyřiceti iteracích už je možné vidět barevné i tvarové rozložení finálního obrázku, i když je produkt stále bledý a rozmazaný.
Po šedesáti iteracích již jsou barvy syté, deformace původního obrázku jsou taktéž výraznější a podobají se více tahům z obrazu Výkřik.



(a) 80 iterací (b) 100 iterací

Obrázek 4.9: Demonstrace průběžného výsledku č. 3

Nyní je už patrné, že čím více iterací proběhlo, tím menší změnu vyvolá iterace nadcházející. Výstup pomalu konverguje k finální podobě a na další změnu v obrázku je potřeba mnohem více iterací. Je možné si všimnout, že přenos uměleckého stylu má problém s prvky, které do kompozice fotografie nesedí, či znatelně vyčnívají, jako například logo FSI v pravém horním rohu budovy. Nicméně i s tím je si přenos schopen poradit, když je mu dán dostatečný počet iterací.

(a) 150 iterací (b) 400 iterací

Obrázek 4.10: Demonstrace průběžného výsledku č. 4

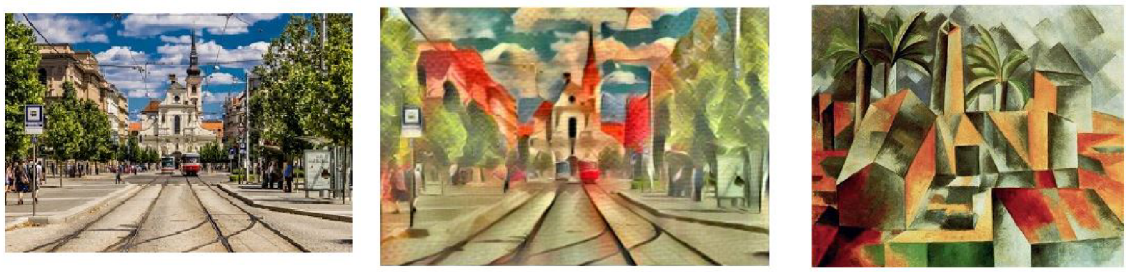
Na poslední dvojici obrázků je vidět, jak málo se výstup změní po překročení určité hranice iterací. I tak je ale zřejmé, že poměrně znatelné změny jsou stále proveditelné, jen je zapotřebí mnohem delší čas k jejich uskutečnění. Po 250 iteracích se změnily tvary v obrázku jen nepatrně a barvy, které byly předtím jen naznačeny jsou nyní zcela syté. Výstup tedy po 400 iteracích vypadá lépe, než po 100, nýbrž práce, která byla na výstupu provedena za prvních 100 iterací je mnohem větší, než za další stovky iterací, které nebyly tak efektivní.

4.5 Ukázka výsledků přenosu uměleckého stylu

V této podkapitole je demonstrováno několik výstupů, které byly vytvořeny pomocí algoritmu přenosu uměleckého stylu.

Obrázek 4.11: Ukázka výstupu č. 1 [41, 42]

Obrázek 4.12: Ukázka výstupu č. 2 [43, 44]



Obrázek 4.13: Ukázka výstupu č. 3 [45, 46]

Obrázek 4.14: Ukázka výstupu č. 4 [48, 47]

Obrázek 4.15: Ukázka výstupu č. 5 [49, 50]

Obrázek 4.16: Ukázka výstupu č. 6 [51]

Obrázek 4.17: Ukázka výstupu č. 7 [52, 53]

5 Závěr

Cílem práce bylo vytvoření programu, který bude umět přenést umělecký styl z uměleckého díla na jiný obrázek či fotografii. První část práce, jež byla teoretického charakteru, čtenáře seznámila s oblastí umělé inteligence, strojovým učením a jeho druhy.

Poté se pozornost přesunula k užší oblasti tohoto oboru, neuronovým sítím. Zde bylo objasněno, z čeho neuronové sítě sestávají, jak zpracovávají data, jaké při tom používají nástroje, nebo jak se vlastně neuronové sítě učí. Byly zde detailně popsány v praxi nejpoužívanější typy sítí se zaměřením na konvoluční neuronové sítě, jelikož tento typ byl použit při řešení praktické části práce. V neposlední řadě byly předvedeny knihovny Keras a TensorFlow, jež byly použity pro sestavení programu přenosu uměleckého stylu.

Druhá, praktičtější, část práce se již zaobírá samotným přenosem uměleckého stylu. Pro řešení úkolu byla vybrána hluboká konvoluční neuronová síť VGG16, jež dosahuje vynikajících výsledků při klasifikaci obrázků z rozsáhlých databází. Parametry této sítě již byly před použitím natrénovány, právě pomocí zmíněné klasifikace. Program pro přenos uměleckého stylu byl sepsán v jazyce Python, konkrétně byly použity knihovny TensorFlow v kombinaci s NumPy a Keras.

Výstupem z programu přenosu uměleckého stylu je obrázek, který nese obsah původní fotografie, či obrázku, na který chceme vybraný styl aplikovat. Zároveň nese i vybraný umělecký styl, tzn. jsou na něj naneseny podobné prvky, jako se nachází na uměleckém díle, tj. specifické tahy štětcem, tvary, barvy a podobně. Cíle práce tedy bylo dosaženo a s pomocí neuronové sítě jsme schopni za pár minut vytvořit obrázek v osobitém uměleckém stylu, který člověk mohl zdokonalovat třeba i desítky let. Ovšem je zde také velký prostor pro vylepšení a rozšíření. Program měl velké problémy při aplikování uměleckého stylu na místa, která zcela nezapadala do kontextu původního obrázku, nebo vyčnívala (viz v páté kapitole náušnice na obr. 4.17 Dívky s perlou, či znak v pravém horním rohu budovy FSI v podkapitole 4.4). Problematická také byla místa, která byla „prázdná“ (např. modrá obloha, jednobarevné pozadí apod.). Tato místa byla pouze zalita uměleckým stylem (viz pozadí obr. 4.11, nebo opět obrázky z podkapitoly 4.4 budovy FSI).

Přenos uměleckého stylu je pozadu v porovnání s komerčními aplikacemi, které využívají podobné technologie. Důvodů může být hned několik. K vylepšení vzhledu výstupních obrázků by mohlo dojít po zakomponování detekce objektů, na které chceme uměl. styl nanést, například pomocí další neuronové sítě. Tím by se zamezilo vyplňování „prázdných“ míst v obrázku slitým uměl. stylem. Dále by zde byla možnost použití hlubší neuronové sítě, což by nám poskytlo větší variabilitu při nastavování parametrů před přenosem stylu. Také by se vzhled mohl zlepšit pomocí výpočetního výkonu. Jak je demonstrováno v podkapitole 4.4, čím více iterací přenosu se použije, tím dokonalejší je nanesení uměl. stylu. Klíčovým faktorem by taktéž mohla být jinak předtrénovaná síť, například by mohla být učena na obrázcích uměleckých děl a ne na databázích, které obsahují fotografie obyčejných objektů. Dalším prvkem, který by mohl výrazně ovlivnit výstup, jsou doprovodné úpravy obrázků, například podle potřeby vyostřit, zahladit hrany, nebo filtry, které pracují s barvami či kontrastem.

Program tedy zdárně plní funkci, která byla cílem práce. A i když nebyly použity dodatečné prvky na zkrášlení výstupu, je pozoruhodné, co je samotná neuronová síť s podpůrným programem schopna vyprodukovat.

6 Literatura

[1] MITCHELL, Tom M. Machine learning. Boston: McGraw-Hill, 1997, xvii, 414 s. ISBN 0-07-042807-7

[2] Stanford Vision and Learning Lab: CS231n: Convolutional Neural Networks for Visual Recognition [online]. Stanford: Li, Johnson, Yeung, 2018 [cit. 2019-03-30]. Dostupné z: http://cs231n.stanford.edu/index.html

[3] BROWNLEE, Jason, 2016. Supervised and Unsupervised Machine Learning Algorithms. In: Machine Learning Mastery [online]. March 16 [cit. 2019-03-30]. Dostupné z: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[4] SONI, Devin, 2018. Supervised vs. Unsupervised Learning. In: Towards Data Science [online]. Mar 22 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[5] WOODFORD, Chris, 2018. Neural networks. In: Explainthatstuff [online]. March 14 [cit. 2019-03-30]. Dostupné z: https://www.explainthatstuff.com/introduction-to-neural-networks.html

[6] NAGYFI, Richárd, 2018. The differences between Artificial and Biological Neural Networks. In: Towards Data Science [online]. Sep 4 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[7] HOLLÄNDER, Branislav, 2018. Natural vs Artificial Neural Networks. In: Becoming Human [online]. Aug 20 [cit. 2019-03-30]. Dostupné z: https://becominghuman.ai/-natural-vs-artificial-neural-networks-9f3be2d45fdb

[8] LAGANDULA, Akshay, 2018. Perceptron: The Artificial Neuron. In: Towards Data Science [online]. Aug 12 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d

[9] MCDONALD, Conor, 2017. Machine learning fundamentals (I): Cost functions and gradient descent. In: Towards Data Science [online]. Nov 27 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220

[10] MOAWAD, Assaad, 2018. Neural networks and back-propagation explained in a simple way. In: Medium [online]. Feb 1 [cit. 2019-03-30]. Dostupné z: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

[11] MAZUR, Matt, 2015. A Step by Step Backpropagation Example. Mattmazur.com [online]. [cit. 2019-03-30]. Dostupné z: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[12] WALIA, Anish, 2017. Activation functions and it's types. In: Towards Data Science [online]. May 29 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[13] DONGES, Niklas, 2018. Recurrent Neural Networks and LSTM. In: Towards Data Science [online]. Feb 26 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

[14] BANERJEE, Suvro, 2018. An Introduction to Recurrent Neural Networks. In: Medium [online]. May 23 [cit. 2019-03-30]. Dostupné z: https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912

[15] SAHA, Sumit, 2018. A Comprehensive Guide to Convolutional Neural Networks. In: Towards Data Science [online]. Dec 15 [cit. 2019-03-30]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[16] TensorFlow [online], [cit. 2019-04-01]. Dostupné z: https://www.tensorflow.org

[17] Keras Documentation [online], [cit. 2019-04-01]. Dostupné z: https://keras.io

[18] GATYS, L.A., ECKER, A.S., BETHGE, M.: A neural algorithm of artistic style (2015). CoRR abs/1508.06576, [cit. 2019-04-25]. Dostupné z: http://arxiv.org/abs/1508.06576.

[19] TensorFlow-Tutorials, Github [online]. [cit. 2019-04-25]. Dostupné z: https://github.com/Hvass-Labs/TensorFlow-Tutorials

[20] SIMONYAN, K., ZISSERMAN, A.: Very Deep Convolutional Networks for Large-Scale Visual Recognition (2014), [cit. 2019-04-25]. Dostupné z: http://www.robots.ox.ac.uk/ vgg/research/very_deep/

[21] Clustering in Machine Learning, In: GeeksforGeeks [online]. [cit. 2019-04-01]. Dostupné z: https://www.geeksforgeeks.org/clustering-in-machine-learning/

[22] Artificial neural network, In: Wikipedia [online]. [cit. 2019-04-01]. Dostupné z: https://en.wikipedia.org/wiki/Artificial_neural_network

[23] Differences between artificial neural network (computer science) and biological neural network, In: Quora [online]. [cit. 2019-04-01]. Dostupné z: https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network

[24] Neural network for MNIST classification, In: Adventures in Machine Learning [online]. [cit. 2019-04-01]. Dostupné z: https://adventuresinmachinelearning.com/pytorch-tutorial-deep-learning/

[25] Image classification, MNIST digits, In: NeuPy [online]. [cit. 2019-04-01]. Dostupné z: http://neupy.com/2016/11/12/mnist_classification.html

[26] Stochastic gradient descent in plain English, In: Medium [online]. [cit. 2019-04-01]. Dostupné z: https://medium.com/@julian.harris/stochastic-gradient-descent-in-plain-english-9e6c10cdba97

[27] Sigmoid function, In: Wikipedia [online]. [cit. 2019-04-01]. Dostupné z: https://en.wikipedia.org/wiki/Sigmoid_function

[28] Hyperbolický tangens, In: Wikipedia [online]. [cit. 2019-04-01]. Dostupné z: https://cs.wikipedia.org/wiki/Hyperbolický_tangens

[29] ReLU, In: Research Gate [online]. [cit. 2019-04-01]. Dostupné z: https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

[30] What is the "dying ReLU" problem in neural networks?, In: Stackexchange [online]. [cit. 2019-04-01]. Dostupné z: https://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks

[31] Data Preprocessing, In: CS231n: Convolutional Neural Networks for Visual Recognition [online]. [cit. 2019-04-01]. Dostupné z: http://cs231n.github.io/neural-networks-2/

[32] Neural Networks & Backpropagation with ND4J, In: Github [online]. [cit. 2019-04-01]. Dostupné z: https://github.com/drewnoff/spark-notebook-ml-labs/tree/master/labs/DLFramework

[33] Understanding The Recurrent Neural Network, In: Medium [online]. [cit. 2019-04-01]. Dostupné z: https://medium.com/mindorks/understanding-the-recurrent-neural-network-44d593f112a2

[34] How Recurrent Neural Networks work, In: Towards Data Science [online]. [cit. 2019-04-01]. Dostupné z: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[35] Convolutional neural networks - What is done first? Padding or convolving?, In: Stackexchange [online]. [cit. 2019-04-01]. Dostupné z: https://stats.stackexchange.com/questions/274601/convolutional-neural-networks-what-is-done-first-padding-or-convolving

[36] Common architectures in convolutional neural networks, In: Jeremy Jordan [online]. [cit. 2019-04-01]. Dostupné z: https://www.jeremyjordan.me/convnet-architectures/

[37] TF Tutorials: 1- Graph and Session, In: Easy TensorFlow [online]. [cit. 2019-04-01]. Dostupné z: https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session

[38] Náměstí Zachariáše z Hradce, In: Město Telč [online]. [cit. 2019-04-25]. Dostupné z: http://www.mesto-telc.cz

[39] The Starry Night, In: Wikipedia [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/The_Starry_Night

[40] VGG16 – Convolutional Network for Classification and Detection, In: Neurohive [online]. 20 November 2018 [cit. 2019-04-25]. Dostupné z: https://neurohive.io/en/-popular-networks/vgg16/

[41] Fire, In: LabelMe [online]. [cit. 2019-04-25]. Dostupné z: http://labelme.csail.mit.edu-/Release3.0/Images/users/rak21/fire/

[42] Tiger Face Close-Up Art Print on Canvas, In: Amazon [online]. [cit. 2019-04-25]. Dostupné z: https://www.amazon.com/Tiger-Close-Up-Canvas-Poster--inches/dp/B01JILEZ0W

[43] Mona Lisa, In: Wikipedia [online]. [cit. 2019-04-25]. Dostupné z: https://cs.wikipedia.org/wiki/Mona_Lisa

[44] Autumn leaves, In: Ssvpscotland [online]. [cit. 2019-04-25]. Dostupné z: https://www.ssvpscotland.com/diocese-of-paisley-autumn-festival-newsletter/autumn-leaves/

[45] Jaro v Brně, In: Lumenn [online]. [cit. 2019-04-25]. Dostupné z: http://lumenn.blog.cz/1803/jaro-v-brne

[46] Pablo Picasso, 1909, Brick Factory at Tortosa, In: Wikipedia [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/-File:Pablo_Picasso,_1909,_Brick_Factory_at_Tortosa,_oil_on_canvas,_50.7_x_60.2_cm,_Hermitage_Museum.jpg

[47] Wild Flowers, In: Alexandra Romano Art [online]. [cit. 2019-04-25]. Dostupné z: https://alexandraromanoart.com/products/wild-flowers-original-colourful--summer-fresh-abstract-expressionism-painting

[48] Steampunk Rotation of the Gears, In: Videoblocks [online]. [cit. 2019-04-25]. Dostupné z: https://www.videoblocks.com/video/steampunk-rotation-of-the--gears-bt05bqcginiskpbg

[49] Appareil photo ancien, In: Takala [online]. [cit. 2019-04-25]. Dostupné z: http://takala.fr/appareil-photo-ancien/

[50] Gold ornaments, In: Deepart [online]. [cit. 2019-04-25]. Dostupné z: https://deepart.io/latest/

[51] Cubism Art Works, In: Pinterest [online]. [cit. 2019-04-25]. Dostupné z: https://cz.pinterest.com/pin/810929476623341683/

[52] Girl with a Pearl Earring, In: Wikipedia [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Girl_with_a_Pearl_Earring

[53] The Scream, In: Wikipedia [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/The_Scream

[54] Budova FSI VUT, In: Kapselshalflanghaarz [online]. [cit. 2019-05-07]. Dostupné z: https://kapselshalflanghaarz.blogspot.com

7 Seznam použitých zkratek a symbolů

zkratky

ML	Machine learning (strojové učení)
DL	Deep learning (hluboké učení)
FCNN	Fully connected neural network (zcela propojená neuronová síť/ mlutilayer perceptron)
SGD	Stochastic gradient descent
RNN	Recurrent neural network (rekurentní neuronová síť)
CNN	Convolutional neural network (konvoluční neuronová síť)
RGB	Barevný model červená-zelená-modrá
TF	Tensorflow

Symboly

w	váhy neuronů, váhy vrstvy neuronové sítě
x	vstup do neuronů, do vrstvy neuronové sítě
b	bias
$\sigma(x)$	sigmoida
a	výstup z neuronu, vrstvy neuronové sítě
C	Cost (celková ztráta ztrátové funkce)
y	požadovaný výstup neuronové sítě
z	vážená suma výstupu neuronu
$tanh(x)$	hyperbolický tangens
h	skrytý stav rekurentní neuronové sítě

8 Seznam obrázků

9 Seznam příloh

A. PDF soubor se zvětšenými obrázky

- Grafické zobrazení průběhu přenosu uměleckého stylu
- Ukázka výstupů

A PDF soubor se zvětšenými obrázky

Vzhledem k tomu, že výstupní obrázky v bakalářské práci nemusejí být zcela čitelné, ať už na papíře, nebo elektronicky v menším rozlišení, je k elektronické verzi práce přiložen soubor s větší verzí těchto obrázků. Přiložený soubor obsahuje obrázky z páté kapitoly, konkrétně ukázku výstupů a zobrazení průběhu přenosu uměleckého stylu.