ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

DIPLOMOVÁ PRÁCE

2018

JAKUB
KARAFFA

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Karaffa** Jméno: **Jakub** Osobní číslo: **424046**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídicí techniky**

Studijní program: **Strojní inženýrství**

Studijní obor: **Přístrojová a řídicí technika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Neuronové sítě pro rozpoznávání objektů v obrazu ze 3-D skeneru

Název diplomové práce anglicky:

Neural Networks for Object Recognition in Images from 3-D Scanner

Pokyny pro vypracování:

1) Proveďte základní rešerši principů a algoritmů používaných pro rozpoznávání objektů v obrazech včetně NS.
2) Proveďte základní rešerši principů a algoritmů neuronových sítí (NS), které se používají také v hlubokých NS.
3) Uveďte princip a přehled posledního vývoje hlubokých neuronových sítí (NS)
4) Navrhněte a naprogramujte vlastní, zjednodušenou, hlubokou NS a demonstrujte na jednoduché případové studii.
4) Pro specifický účel dat ze 3-D skeneru, navrhněte vlastní hlubokou neuronovou síť, kterou realizujte (vlastním kódem, nebo ve vhodné platformě, např. Tensor Flow)
5) Proveďte experimentální studii realizované sítě na datech ze skeneru a řádně zdokumentujte.

Seznam doporučené literatury:

[1] NIELSEN, Michael A. Neural Networks and Deep Learning [online]. 2015 [vid. 2017-04-03] Dostupné z: http://neuralnetworksanddeeplearning.com
[2] Understanding Hinton's Capsule Networks, https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **18.04.2018** Termín odevzdání diplomové práce: **15.06.2018**

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

18.4.2018
Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych především poděkoval vedoucímu mé diplomové práce Doc. Ing. Ivu Bukovskému, Ph.D. za veškerou pomoc při psaní této práce. Dále bych rád poděkoval mé rodině, která mě podpořila v těžkých chvílích.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s tím, že její výsledky mohou být dále použity podle uvážení vedoucího diplomové práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její spoluautor.

V Praze dne 14.6. 2018 .....

Anotace

V rámci této práce je provedena rešerše algoritmů pro zpracování obrazu včetně hlubokých neuronových sítí, za účelem zpracování fotografií ze 3D skeneru. V teoretické části práce jsou uvedeny některé algoritmy využívané hlubokými neuronovými sítěmi. Jsou popsány vlastnosti speciálního druhu hlubokých neuronových sítí, konvolučních neuronových sítí. Konvoluční neuronové sítě jsou následně využity k rozpoznání obrazu. V rámci práce je vytvořeno několik modelů, všechny jsou vytvořeny v programovém prostředí Python za pomoci knihovny Tensorflow. Na závěr je provedeno porovnáni úspěšnosti těchto modelů.

Abstract

The main aim of this thesis is to provide research of algorithms for picture processing, including deep neural networks, for the purposes of processing the photographs from a 3D scanner. In the theoretical part, several algorithms used in deep neural networks are described. The properties of convolutional neural network, which is a special type of deep neural network, are explained. Convolutional neural networks are then used for image recognition. Several models were developed for the purposes of this thesis. All of them were created in the Python programming environment with the use of Tensorflow library. The conclusion compares the success of these models.

Obsah

Kapitola 1

Úvod

Cílem této práce je prozkoumání možností využití hlubokých neuronových sítí pro zpracování fotografií ze 3D skeneru. Výstupem 3D skeneru bývá tzv. mračno bodů a fotografie. Dle fotografií budou rozpoznány základní stavařské prvky, jako jsou například dveře či okna. V dnešní době probíhá zpracování mračen bodu neautomaticky. Jednotlivé mračna bodů je potřeba mezi sebou propojit, což je proces, u kterého se sice mnoho softwarů snaží o automatizaci, ale většinou neúspěšně. Po získání složeného mračna bodů je zapotřebí data očistit, během skenování vzniká díky různým vlivům šum, datové stíny apod. rušivé elementy, které ve výsledném zpracování být nemají. K odstranění datových chyb jsou implementovány filtry, které jsou schopny některé základní vady podchytit. Filtry jsou ovšem většinou spíše kontraproduktivní, filtry často i s chybnými daty mažou i data korektní. Po očištění mračna bodů nastává výsledné převedení na výkresovou dokumentaci. Tato část začíná importováním mračna do CADu[1], kde je mračno rozděleno dle výškových vrstev a ručně obkreslováno. Celý tento proces snad bude časem možné automatizovat. Tato práce má za cíl ukázku základního rozřazení fotografií do několika kategorií a tím pádem přípravu pro další navazující aplikace spojené s touto problematikou.

1.1 Základní informace o 3D skenerech

Laserových 3D skenerů je velká řada, liší se převážně podle účelu použití. Je zřejmé, že skener pro kontrolu přesností ve strojírenském průmyslu bude mít rozdílné vlastnosti než skener určený pro skenování bytových objektů. Primárním výstupem všech skenerů je mračno bodů, které se podle skeneru liší převážně v rozteči bodů. Většina skenerů je taktéž schopna vyexportovat panoramatické fotky, kterými se budu v této práci zabývat. Pro tuto diplomovou práci použiji data ze skeneru Faro

[1] software pro počítačem podporované kreslení

Focus 3D 330, který je primárně určen k snímání rozsáhlejších objektů. Na obrázku níže lze vidět panoramatickou fotku z tohoto skeneru.

Obrázek 1.1: Fotografie ze skeneru

Laserové skenery nejdříve při snímání polohy bodů zaznamenávají i jejich odrazivost. Takto je získán prvotní náhled ve stupni šedi. Tento náhled je následně propojen s barevnou fotografií a každému bodu je přiřazena barevná hodnota, dle barvy pixelu ve kterém se nachází. U některých softwarů určených pro zpracování mračen bodů, lze náhledy ve stupni šedi obarvit i z fotek pořízených externím fotoaparátem. Celkově se laserové skenery stávají čím dál tím populárnější, z mého pohledu však nastává problém při zpracování dat. Po několika letech práce s těmito skenery si dovolím tvrdit, že hardware je napřed před softwarovým řešením. Neuronové sítě by mohly přinést velké usnadnění při zpracování těchto dat a do budoucna celý proces zpracování automatizovat.

Kapitola 2

Metody rozpoznávání objektů

V této kapitole jsou popsány základní používané algoritmy pro zpracování obrazu. Algoritmus na rozpoznávání obrazu lze rozložit do několika kroků:

- Předzpracování obrazu
- Segmentace
- Nalezení/popis objektů
- Klasifikace

2.1 Předzpracování obrazu

V rámci předzpracování jsou provedeny takové operace, aby byl odstraněn šum, rušení a podobné rušivé vlivy, které mohly vzniknout například ze špatných světelných podmínek [1] [2]. Nejpoužívanější operace pro odstranění těchto vlivů jsou: úprava jasu, úprava kontrastu, převedení na stupně šedi, úprava histogramu, zprůměrování hodnot z několika snímků, zaostření obrazu, zmenšení, otočení či posun.

2.2 Segmentace

Během segmentace je odlišen samostatný objekt, či jeho hrany od pozadí či výplně.[2] Tímto způsobem lze separovat samostatný obraz objektu a zmenšit tak objem vstupních dat. Mezi používané metody pro segmentaci obrazu patří např. detekce hran, prahování, barvení.

2.2.1 Prahování

Postup prahování [1] spočívá ve využití odrazivosti nebo pohltivosti povrchů v pozadí zkoumaného objektu, která je porovnána s jasovou konstantou (práh). Pro objekt s rozdílnou odrazivostí od pozadí, lze díky této metodě dosáhnou jednoduchého očištění. Pokud by se ovšem hodnoty odrazivosti pro pozadí i objekt prolínaly, mohlo by při prahování dojít k odmazání části objektu či nedokonalému odstranění pozadí.

2.2.2 Detekce hran

Při procesu detekce hran se využívá popis hrany jako náhlou a výraznou změnu jasu sousedních pixelů. K detekci těchto změn se používají metody založené na derivacích obrazové funkce, která popisuje vlastnosti jednotlivých pixelů.

2.2.3 Barvení

Během barvení je procházen obraz po řádcích a každému nenulovému elementu je přiřazena hodnota podle sousedních elementů. Sousední elementy jsou určeny dle masky barvení.

i-1,j-1	i,j-1	i+1.j-1
i-1,j	i,j	i+1,j
i-1,j+1	i,j+1	i+1,j+1

Tabulka 2.1: Příklad masky pro barvení

2.3 Nalezení objektů

Pro nalezení objektu jsou využity tvarové charakteristiky, momenty objektu, ramena objektu atp. Cílem metody je získání takového výsledku, který bude odolný vůči posunutí, rotaci a změně velikosti - bude invariantní. Výsledkem těchto operací může být např. tabulka nalezených objektů s jejich popisem jako plocha, obvod.

2.4 Klasifikace

Tato závěrečná operace má za cíl roztřídit nalezené objekty do skupin. V každé skupině se objevují objekty se společnými rysy, které byly důležité při nacházení objektů. Jednou z možných operací sloužících ke klasifikaci objektů jsou neuronové sítě.

Kapitola 3

Základy neuronových sítí

Neuronová síť je propojení několika neuronů, které i sami o sobě mohou být silnou klasifikační jednotkou. Neuronové sítě se nevyužívají pouze pro rozpoznávání obrazu, ale mají využití v široké škále odvětví. Jejich rozmach je také z části způsoben rychlým hardwarovým pokrokem. Neuronové sítě jsou inspirovány lidskou sítí neuronů, která je zodpovědná za lidské chování.

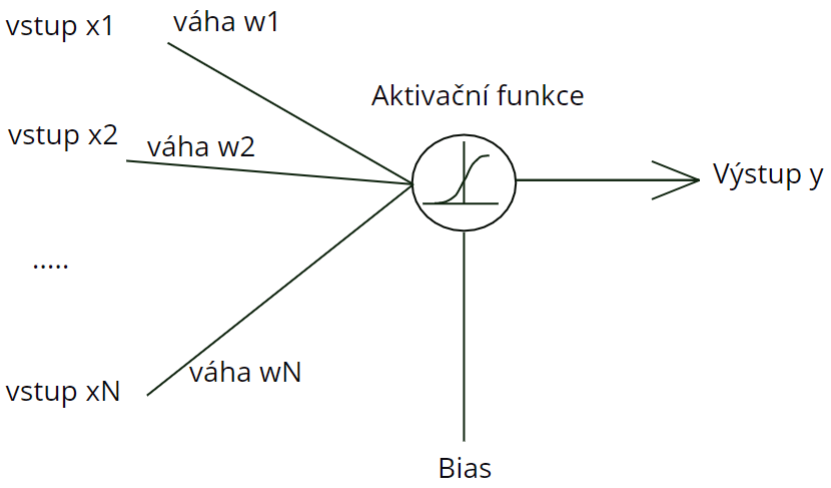
3.1 Historie neuronových sítí

Vznik neuronových sítí je datován k roku 1943, ve kterém pan Warren McCulloch s Walterem Pittsem ve své práci vytvořili jednoduchý matematický model neuronu, což je základní část nervové soustavy [3] [4]. Následně v roce 1949 napsal Donald Hebb knihu „The Organization of Behaviour“, ve které navrhl a popsal učící pravidlo pro synapse neuronů. Pravidlo bylo založeno na zjištění, že synaptické spojení mezi dvěma aktivními neurony se posiluje. Dalším velkým pokrokem byl rok 1957, kdy Frank Resenblatt vynalezl tzv. perceptron, který je zobecněním předcházejících modelů. Pro tento model dokonce navrhl učící algoritmus, u kterého následně matematicky dokázal jeho konvergenci. Na základě toho pokroku byl během let 1957–1958 sestrojen první neuropočítač, který sloužil k rozpoznávání znaků. Problém nastal v roce 1969, kdy pánové Minsky a Papert doložili, že jednovrstvá síť není schopna vyřešit XOR problémy a chtěli tak převést finanční obnos investovaný do výzkumu neuronových sítí do jiného odvětví umělé inteligence. Problém sice šlo vyřešit pomocí vícevrstvé neuronové sítě, ale v té době nebyl znám učící algoritmus pro tento typ sítě. Zvrat nastal až v roce 1974, kdy pan Werbos objevil algoritmus učení vícevrstvých neuronových sítí. Tento algoritmus je dodnes používaný a známý pod názvem backpropagation (zpětné šíření chyby). Tento objev byl znovu nezávisle na sobě objeven v roce 1985 a to pány Rumelhartem, Hintonem, Wiliamsem a Perkerem. V roce 2003 byl pan M.M. Guptou

vývoj neuronových jednotek vyšších řádů. Tyto neuronové sítě se dnes označují jako HONNU (Higher Order Nonlinear Neural Units) a mezi jejich nejznámější zástupce patří QNU (quadratic neural units – vnitřní funkce 2. řádu) nebo CNU (cubic neural units – vnitřní funkce 3. řádu).

3.2 Základy neuronových sítí

Každá neuronová síť se skládá z jednotlivých neuronů. Nejznámějším a nejpoužívanější model neuronu je percepton [3]. Tento model neuronu, je stejně jako ostatní modely založen na matematickém popisu biologického neuronu. Model perceptonu je na obrázku níže.



Obrázek 3.1: Model neuronu

Do modelu neuronu může vstupovat 1 až N neuronů, podle složitosti úlohy. Vstupem mohou být podněty z vnějšího okolí nebo výstupy z jiných neuronů (u vícevrstvé sítě). Každý vstup je pozměněn o váhu daného vstupu. Percepton také obsahuje prahovou hodnotu (bias), který určuje hranici, za kterou je neuron nabuzen a indukuje výstup podle přenosové funkce σ [5]. Pro lineární neuron bude matematický popis modelu vypadat takto:

$$y = \sigma(b + w_1x_1 + w_2x_2 + ... + w_Nx_N) \tag{3.1}$$

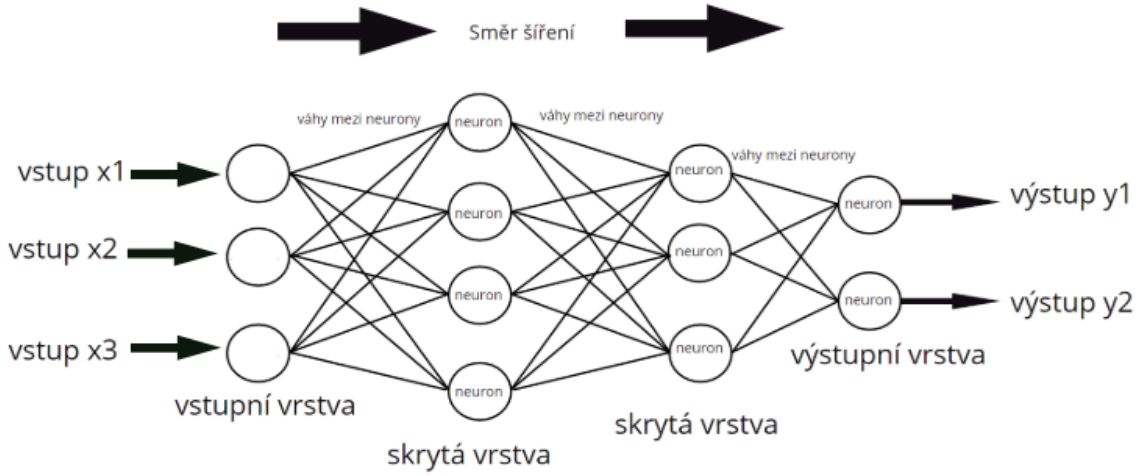
Pro zjednodušení je často uvažován bias jako nultý vstup do neuronu, a tak

předchozí formulace bude vypadat následovně:

$$y = \sigma(\sum_{i=0}^{n} w_i x_i) \tag{3.2}$$

3.3 Vícevrstvé neuronové sítě

Vícevrstvá neuronová síť obsahuje formální neurony, které jsou vzájemně propojeny a to tak, že výstup jednoho neuronu je vstupem dalšího v jiné vrstvě [3]. Výstup z jednoho neuronu může sloužit jako vstup více neuronům, tato různá propojení jsou určena architekturou neuronové sítě. Z hlediska architektury jsou sítě rozlišeny na vstupní, výstupní a skryté vrstvy. Na obrázku 3.2 je vícevrstvá neuronová síť s jednou vstupní, dvěma skrytými a jednou výstupní vrstvou.



Obrázek 3.2: Vícevrstvá neuronová síť 3-4-3-2

3.4 Architektura neuronových sítí

Zatím byl v této práci zmíněn hlavně nejrozšířenější topologický typ sítě a tím je dopředná neuronová síť [4]. Existují i další architektury neuronových sítí, jako například zpětnovazební síť nebo kompletně propojená síť.

3.4.1 Dopředná síť

U tohoto typu sítě jsou neurony řazeny do vrstev, přičemž každý neuron je řazen samostatně, díky čemu je celá síť robustní [5] [4].Tyto vlastnosti jsou využity při konstrukci neuropočítačů, což jsou počítače založených na neuronových sítích. Spojení mezi dvěma neurony smí u dopředné sítě směřovat pouze k neuronu v další vrstvě. U těchto sítí jsme schopni rozpoznat aktivní a adaptivní fázi. V aktivní fázi neuronová síť produkuje na základě vstupů výstupy, přičemž parametry sítě zůstávají konstantní. Během adaptační fáze síť trénovacím algoritmem aktualizuje své vnitřní parametry.

3.4.2 Zpětnovazební síť

Zpětnovazební typ sítě umožňuje stejné propojení, jak v dopředné síti, ale přidává možnost zpětné vazby, nepřímé zpětné vazby a příčné zpáteční vazby. Propojení zpětné vazby je propojení neuronu se sebou samým [5]. Tímto propojením získávají neurony možnost, aby se samy omezovaly či oslabovaly a tím se dostali na úroveň své aktivační hranice. Při nepřímé zpětné vazbě jsou neurony propojeny směrem ke vstupní vrstvě. Příčná zpáteční vazba je propojení dvou neuronů ve stejné vrstvě. Takovéto spojení často zabraňuje ostatním neuronům z vrstvy v aktivaci, jelikož posiluje právě sebe sama.

3.4.3 Kompletně propojená síť

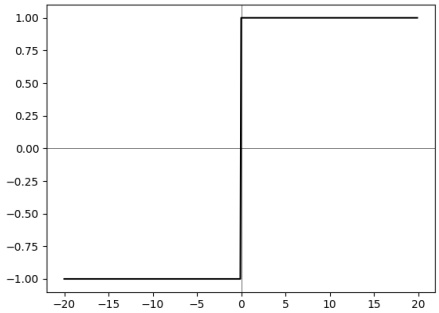
Kompletně propojená neuronová síť dovoluje jakákoliv propojení mezi všemi neurony až na propojení přímé zpětné vazby [5]. Spojení musí být symetrická. V tomto typu sítě může každý neuron udržovat spojení s každým dalším neuronem. Pro tento typ architektury by bylo těžké rozlišit jednotlivé vrstvy, místo toho jsou často jednotlivé neurony uspořádány do řádků a sloupců.

3.5 Aktivační funkce

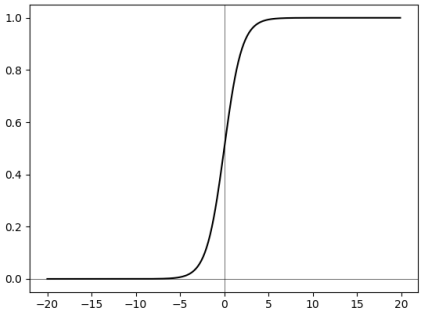
Hlavním cílem aktivačních funkcí je převod vstupního signálu neuronu na výstupní [3]. Bývá zvykem používat jednu a tu samou aktivační funkci pro celou vrstvu. Obecně aktivační funkce může být jakákoliv diferencovatelná funkce. Pro zpracování obrazu je obvykle využívána ReLu aktivační funkce. V následující tabulce 3.3 je přehled grafů některých používaných aktivačních funkcí.

Předpis funkce ReLu:

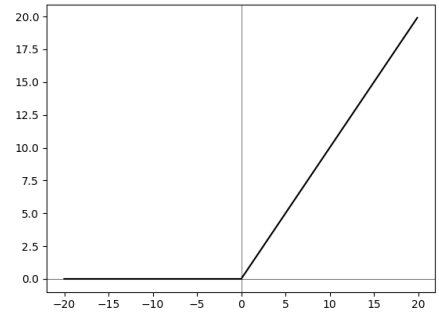
$$\sigma(x) \quad = \quad max(0, x) \tag{3.3}$$



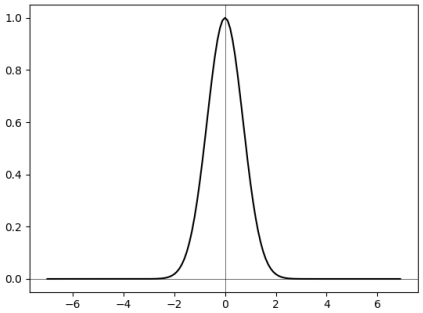
(a) signum



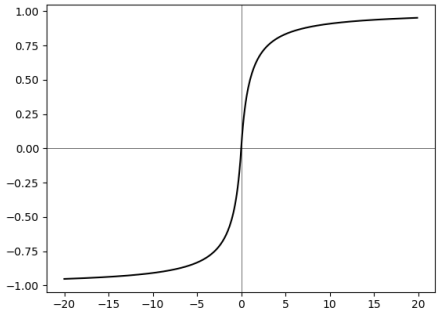
(b) sigmoida



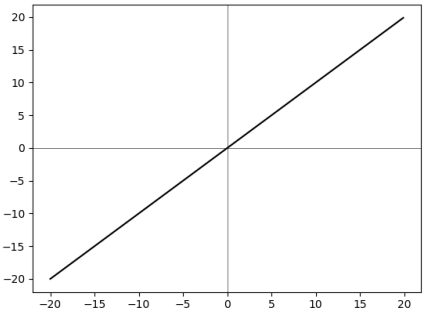
(c) ReLu



(d) Gaussova



(e) softsign



(f) lineární

Obrázek 3.3: Příklady aktivačních funkcí

Kapitola 4

Metody učení neuronové sítě

Učení neuronové sítě neboli adaptivní dynamika má za úkol parametry neuronové sítě nastavit tak, aby neurony byly schopny provádět správně požadovanou operaci [3]. Adaptované parametry neuronů jsou jak váhy, tak i biasy. Adaptaci těchto parametrů obstarávají učící algoritmy. Učení neuronové sítě probíhá opakovaně buď podle počtu učících cyklů nebo do určité přesnosti sítě. Cílem je dosáhnout takové nastavení sítě, aby transformace vstupu na výstup byla dostatečně obecná, a to i pro další neznámé příklady, které nesloužily k učení neuronů. Učení tvoří zásadní výhodu neuronových sítí, a to možnost nalezení řešení pro analyticky obtížné nebo dokonce i neřešitelné úlohy. Vše, co je potřeba je dostatečně velká vstupní množina prvků, sloužící k učení sítě. Za nevýhodu se dá považovat fakt, že výsledné výpočty jsou ukryty ve struktuře sítě a nelze je použít například k vysvětlení řešení úlohy. Je potřeba mít také na vědomí, že ačkoliv jsou neuronové sítě silným výpočtovým algoritmem, tak nezaručují vždy nalezení správně obecné transformace. Dle metody učení rozlišujeme dvě základní kategorie, a to učení s učitelem a učení bez učitele.

4.1 Učení bez učitele

Pro tuto metodu je zásadní, že učící algoritmus nemá k dispozici data k ověření správnosti výstupů neuronové sítě [5]. Metoda pracuje na principu shlukování, kdy ve vstupních datech kategorizuje podobné elementy. Na základě těchto kategorií se snaží data roztřídit do skupin, ale to vše bez autonomní možnosti ověření správnosti. Počet hledaných skupin závisí na parametrech učení, buď může být předem definován nebo vypočten algoritmem.

4.2 Učení s učitelem

Pokud se neuronová síť má učit s učitelem, tak to znamená, že vstupem do sítě je množina dvojic (vstupů a jim odpovídajícím správným výstupů) [5]. Učení tak probíhá na základě porovnání výstupu (odhadu) z neuronové sítě s požadovaným výstupem. Množina těchto dvojic představuje známou část chování sítě. Tato množina je většinou rozdělena na dvě části, trénovací a testovací. Množina pak slouží jak k naučení sítě, tak i k ověření správnosti její funkce. Poměr mezi počtem prvků v trénovací a testovací není pevně stanoven, ale obvykle se pohybuje v poměru 8:2. Samotné trénování pak probíhá iterativně, přičemž algoritmus postupně vkládá do neuronové sítě jednotlivé prvky trénovací množiny a na základě odchylky výstupu ze sítě od požadovaného výstupu aktualizuje váhy neuronů. Interval, ve které dojde k předložení všech prvků z testovací množiny se nazývá epocha. Dle složitosti problému a neuronové sítě se počet epoch trénování může pohybovat od jednotek až do tisíců (obecně je tento počet omezen hardwarovými a časovými možnostmi). Úspěšnost učení neuronové sítě získáme použitím testovací množiny. Dostatečná přesnost při zpracování dat z testovací množiny může sloužit k zastavení algoritmu učení. Obecný postup pro natrénování sítě s učitelem je:

1. Předzpracování vstupních dat
2. Rozdělení vstupních dat na trénovací a testovací
3. Vytvoření architektury sítě
4. Inicializace sítě - obvykle počáteční váhy jsou náhodná čísla
5. Započetí trénovací epochy
6. Předložení vzorku dat sítě
7. Vyhodnocení chyby a aktualizace vah
8. Znovu krok 6. dokud nejsou vyčerpána všechna vstupní data
9. Znovu na krok 5. dokud není dosažena dostatečná přesnost nebo určitý počet proběhnutých epoch
10. Vyhodnocení úspěšnosti neuronové sítě na testovací množině
11. Pokud je úspěšnost nedostatečná, tak můžeme změnit architekturu sítě, přidat počet epoch nebo získat více trénovacích dat a celý postup opakovat

4.3 Chybová funkce

Chybová funkce neboli „loss function“ slouží k matematickému vyjádření chyby neuronové sítě. Obvyklé jsou funkce Mean squared error a Cross entropy [6]. Optimalizační algoritmy se snaží o minimalizace této funkce, proto například nepřipadá možnost lineární vyjádření závislosti mezi výstupem z neuronové sítě a správným výsledkem. Cílem optimalizačních algoritmů algoritmů je pozměnit váhy a biasy v neuronové síti tak, aby hodnota chybové funkce byla co nejmenší.

Mean squared error (MSE)

$$C(w,b) \quad = \quad \frac{1}{n}\sum_{x}\|y(x)-d(x)\|^2 \tag{4.1}$$

Cross entropy

$$C(w,b) \quad = \quad -\frac{1}{n}\sum_{x}(y\ ln(d)+(1-y)ln(1-d)) \tag{4.2}$$

přičemž n je celkový počet trénovacích vstupů, d je vektor výstupů z neuronové sítě při vstupu x, y je správný výstup odpovídající vstupu x.

4.4 Backpropagation

Tento algoritmus je jeden z nejpoužívanějších při učení neuronových sítí, dokonce dle [3] je využit přibližně v 80% aplikacích. Backpropagation lze rozdělit na 3 fáze, které se cyklicky opakují, dokud nejsou splněny ukončovací podmínky. Zmíněné tři fáze jsou:

1. Dopředné šíření vstupních dat
2. zpětné šíření chyby
3. aktualizace vah a biasů

V první fázi probíhá převod vstupu neuronové sítě na výstup. Nejdříve obdrží každý neuron ze vstupní vrstvy signál a zprostředkuje přenos tohoto signálu ke všem neuronům následující vrstvy. V následující vrstvě každý neuron zpracuje obdržený signál a pošle výsledek do následující vrstvy. Tento cyklus se opakuje, dokud nedojde signál do výstupní vrstvy. Tímto způsobem získáme odezvu neuronové sítě (d_k) na vstupní signál.
V druhé fázi se pomocí chybové funkce zjistí celková chyba neuronové sítě a z ní pak chyby pro neurony v jednotlivých vrstvách. Na základě chyby vzniklé u všech neuronových spojení se ve třetí fázi aktualizují váhy a biasy.

4.4.1 Ukázka algoritmu backpropagation

Odvození backpropagation je realizováno na vícevrstvé neuronové síti, u které je uvažován bias jako nultý vstup [7] [8]. Chybu lze tedy napsat dle 4.1 zapsat jako:

$$C(w) \quad = \quad \frac{1}{2}(y-d)^2 \tag{4.3}$$

Výstup z j-tého neuronu lze vyjádřit pomocí rovnice 4.4, ve které je $vstup_j$ vstup do neuronu j, σ aktivační funkce, d_k výstup z neuronu z předcházející vrstvy a w_{kj} je váha spojení neuronu k a j

$$d_j \quad = \sigma(vstup_j) = \sigma(\textstyle\sum_{k=1}^{n} w_{kj} d_k) \tag{4.4}$$

Parciální derivace chyby v závislosti na vahách bude:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial d_j} \frac{\partial d_j}{\partial vstup_j} \frac{\partial vstup_j}{\partial w_i j} \tag{4.5}$$

$$\frac{\partial vstup_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k=1}^{n} w_{kj} d_k = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i \tag{4.6}$$

Pokud bude splněna podmínka, že aktivační funkce (pro tento případ použita sigmoida) bude derivovatelná, tak také platí:

$$\frac{\partial d_j}{\partial vstup_j} = \frac{\partial}{\partial vstup_j} \sigma(vstup_j) = \sigma(vstup_j)(1 - \sigma(vstup_j)) \tag{4.7}$$

Výpočet parciální drtivace $\frac{\partial C}{\partial d_j}$ pro případ, že d_j je odhad z výstupní vrstvy.

$$\frac{\partial C}{\partial d} = \frac{\partial}{\partial d} \frac{1}{2} (y-d)^2 = d - y \tag{4.8}$$

Pokud by ovšem d_j byl výstupem z neuronu v jiné než výstupní vrstvě, dospělo by se ke složitější rovnice. Chybovou funkci lze tedy vyjádřit v závislosti na vstupech všech neuronů L = 1, 2, ..., n, které obdržel signál neuronu j.

$$\frac{\partial C(d_j)}{\partial d_j} = \frac{\partial C(vstup_1, vstup_2, ..., vstup_n)}{\partial d_j} \tag{4.9}$$

$$\frac{\partial C}{\partial d_j} = \sum_{l \in L} \left(\frac{\partial C}{\partial vstup_l} \frac{\partial vstup_l}{\partial d_j}\right) = \sum_{l \in L} \left(\frac{\partial C}{\partial d_l} \frac{\partial d_l}{\partial vstup_l} w_{jl}\right) \tag{4.10}$$

Spojením těchto rovnic vznikne.

$$\frac{\partial C}{\partial w_{ij}} = \delta_j d_i \tag{4.11}$$

přičemž

$$\delta_j d_i = \frac{\partial C}{\partial d_j}\frac{\partial d_j}{\partial vystup_j} = \begin{cases} (d_j - y_j)d_j(1-d_j) \\ (\sum_{l\in L} w_{jl}\delta_l)d_j(1-d_j) \end{cases} \tag{4.12}$$

A díky tomu lze konečně vyjádřit požadovanou změnu Δw_{ij}

$$\Delta w_{ij} = -\eta\frac{\partial C}{\partial w_{ij}} = -\eta\delta_j d_i \tag{4.13}$$

kde η je koeficient učení (learning rate) a celý výraz je přenásoben -1 kvůli tomu, že $\frac{\partial C}{\partial w_{ij}}$ vyjadřuje gradient chybové funkce a cílem je jít proti gradientu, tak aby chybová funkce byla co nejmenší.

4.5 Optimizéry

Optimizer je algoritmus, který se snaží o minimalizaci chybové funkce a tím zpřesnění samotného výstupu [9]. Minimalizování chybové funkce optimizery realizují změnami vah a biasů. Před použitím optimalizačních algoritmů je potřeba definovat následující:

1. **Chybovou funkci**
2. **Poskytnout počáteční odhad parametrů sítě - většinou se volí náhodné hodnoty nebo 0**
3. **Stanovení konvergenční podmínky**(často už součástí optimizeru)
4. **Vybrání a definování optimizeru**

4.5.1 Adam

Adam je jeden z nejpoužívanějších optimizérů pro neuronové sítě. Tento algoritmus byl následně využit při zpracování praktické části této práce. Pro použití tohoto optimizéru je potřeba nejdříve definovat vstupní parametry:

1. α - velikost kroku
2. $\beta_1, \beta_2 \in [0, 1)$ - exponenciálně klesající koeficienty pro odhady momentů

3. $f(\theta)$ - stochasticky objektivní funkce s parametry θ. (chybová funkce)
4. $\theta_\mathbf{0}$ - počáteční odhad parametrů

Následně je potřeba inicializovat některé proměnné:

1. $\mathbf{m_0} \leftarrow \mathbf{0}$ - inicializace prvního momentového vektoru (1.m.v)
2. $\mathbf{v_0} \leftarrow \mathbf{0}$ - inicializace druhého momentového vektoru (2.m.v)
3. $t \leftarrow 0$ - inicializace kroku

Optimalizace funguje dle následující algoritmu:

while podmínka konvergence **do**:
$t \leftarrow t + 1$
$\mathbf{g_t} \leftarrow \nabla_\theta f_t(\theta_{t-1})$ - získání gradientu
$\mathbf{m_t} \leftarrow \beta_1 \cdot \mathbf{m_{t-1}} + (1 - \beta_1) \cdot \mathbf{g_t}$ - aktualizace odhadu 1.m.v
$\mathbf{v_t} \leftarrow \beta_2 \cdot \mathbf{v_{t-1}} + (1 - \beta_2) \cdot \mathbf{g_t}^2$ - aktualizace odhadu 2.m.v
$\mathbf{\hat{m}_t} \leftarrow \frac{\mathbf{m_t}}{1-\beta_1^t}$ - vypočítání upraveného odhadu 1.m.v
$\mathbf{\hat{v}_t} \leftarrow \frac{\mathbf{v_t}}{1-\beta_2^t}$ - vypočítání upraveného odhadu 2.m.v
$\theta_\mathbf{t} \leftarrow \theta_\mathbf{t-1} - \frac{\alpha \cdot \mathbf{\hat{m}_t}}{\sqrt{\mathbf{\hat{v}_t}} + \epsilon}$ - aktualizace parametrů
end while
return $\theta_\mathbf{t}$[9]

4.5.2 RMSprop

Tato metoda je založena na výpočtu exponenciálně klesajícího průměru gradientu [10].

$\mathbf{E}\|\mathbf{g^2}\|_\mathbf{t} \leftarrow 0.9E\|g^2\|_{t-1} + 0.1g_t^2$ -získání průměru z předchozích gradientů

$\theta_\mathbf{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{E\|g^2\|_t + \epsilon}} g_t$ -aktualizace parametrů

4.5.3 Adagrad

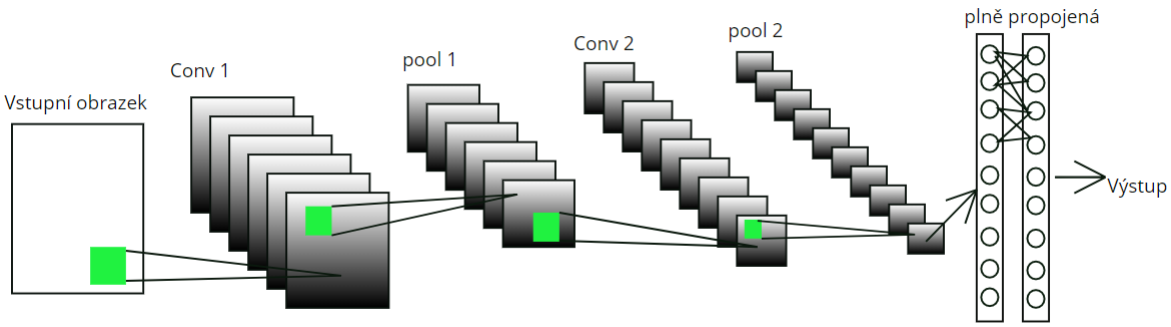
Metoda Adagrad je velmi podobná RMSprop, liší se ve výpočtu průměru gradientu, přičemž u metody RMSprop se počítá přes exponenciálně klesající průměr, zde jako suma gradientů [10].

$\theta_\mathbf{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$ -aktualizace parametrů, přičemž G_t je suma druhých mocnin gradientů až do času t

Kapitola 5

Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN - Convolutional Neural Network) jsou vícevrstvé neuronové sítě určené především k zpracování obrazu[11] [12]. Jejich výhodou nad „obyčejnými“ vícevrstvými NS je robustnost vůči deformaci vstupního vektoru (obrazu), jako je například posunutí či rotace. Tuto výhodu CNN získávají díky metodě sdílení vah, kde je jedna váha sdílena více neurony. Deformovaný obraz, tak bude aktivovat sice odlišné neurony, ale se stejnými vahami jako původní nezdeformovaný obraz. CNN se skládá ze vstupní, konvoluční, sdružovací(pooling nebo sub-sampling layer) a jako poslední je plně propojená vícevrstvá síť. Je obvyklé, že CNN má několik konvolučních a sdružovacích vrstev, výstup z konvoluční vrstvy je pak vstupem do sdružovací vrstvy, ze které signál přechází do další konvoluční vrstvy atd..



Obrázek 5.1: Konvoluční neuronová síť

5.1 Konvoluční vrstvy

Zatímco první konvoluční vrstva detekuje jednodušší vzorce (jako například hrany), čím hlouběji postupujeme, tím CNN detekuje složitější a složitější vzorce [11] [12]. Druhá konvoluční vrstva je schopná odhalit například rohy nebo kruhy, až se dostane k tomu, že je schopná detekovat celé objekty a s vysokou přesností rozlišit velmi složité vzorce. Konvoluční vrstva je tvořena z několika příznakových map a aktivačních funkcí[1].

5.1.1 Příznaková mapa

Příznaková mapa je filtr s maskou o rozměru $n \times n$, která se posouvá po obraze o určitý počet pixelů s (parametr masky - stride) [11] [12]. Na příklad si lze představit obrázek o velikosti 20x20, příznakovou mapu 5x5 a stride = 1. Oblast masky, tedy 25 pixelů bude vstupem do jednoho neuronu, ten bude mít tedy i s biasem 26 vstupů. Maska bude aplikována na celý obrázek a získá se tak 256 (16 x 16) neuronů, které budou mít stejné váhy i bias (metoda sdílení vah). Získáno je 26 vah (25 vah + bias jako nultá váha) a 256 neuronů. Právě díky takto posunutým neuronům je CNN odolná vůči deformaci vstupního obrazu. Výsledkem použití jedné masky na obraz je tzv. aktivační mapa (pro tento případ je to tedy 256 neuronů). Jedna takto vzniklá aktivační mapa je schopna detekovat například horizontální čáru. Obvykle je však cílem detekce více než jednoho příznaku, proto je využito vícero masek, dostane se tak sada příznakových map. Uvnitř každé mapy jsou sdíleny neurony, avšak mezi sebou příznakové mapy váhy nesdílí.

5.2 Sdružovací vrstva

Cílem sdružovací vrstvy je redukovat velikost aktivačních map z konvoluční vrstvy [11] [12]. Vstupem je tedy aktivační mapa z konvoluční vrstvy a výstupem je zmenšená aktivační mapa. Zmenšení probíhá za použití čtvercové masky, která je na vstupní aplikační vrstvu aplikovaná tak, aby se jednotlivé oblasti nepřekrývaly. To znamená, že pro masku m x m musí být stride $s = m$. Pokud bude například zvolena maska 2x2 bude aktivační mapa zmenšena na polovinu jak vertikálně, tak horizontálně. Většinou jsou tyto vrstvy naprogramovány na jednoduché operace jako například nejpoužívanější maximální hodnota z masky.

[1] Pro CNN se nejčastěji používá ReLu, případně leaking ReLu

5.3 Tensorflow

Tensorflow (tf) je open source softwarová knihovna využívající se při mnoha procesech, ale převážné je využíván pro strojové učení [13]. Tensorflow využívá metodu data flow graph (graf datového toku) a podporuje výpočty při použití GPU, CPU, CPUm (CPU pro mobilní zařízení), a další. Architektura tf je navržena tak, aby zvládala větší objemy dat a složitější operace. Tensorflow je využíván v mnoha různých oblastech strojového učení jako například rozpoznávaní řeči, počítačové vidění, robotika, vyhledávání a zpracování jazyka. V této práci byl použit tf v prostředí python. Tensorflow byl vybrán z důvodu podpory grafických karet NVIDIA.

5.3.1 Ukázka tensorflow

Jako ukázka hluboké neuronové sítě byl vybrán program k redukci šumu. Program je realizován v prostředí Python. Využity byly knihovny matplotlib, tensorflow a numpy. Redukce šumu je realizována díky hluboké neuronové síti složené z encoderu a decoderu. Vstupem do hluboké neuronové sítě je zašuměný průběh funkce sin. Vstupy a výstupy datového grafu tensorflow jsou realizovány pomocí příkazů placeholder.

```
X = tf.placeholder("float", [None, num_input])
```

Tento příkaz definuje pole vstupní proměnné, datový typ je určen jako float a rozměr pole je definován jako [None, počet vstupů]. Váhy a biasy lze vytvořit pomocí příkazu tf.Variable

```
'encoder_w1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
```

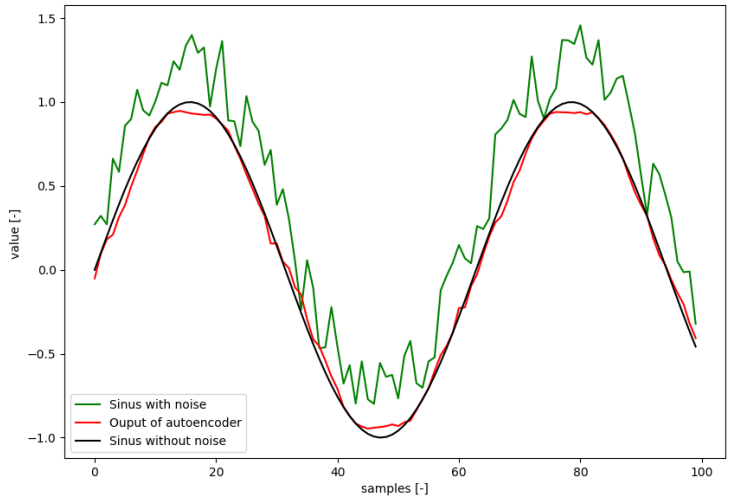
Přičemž ukázaný příkaz pro vytvoření váhy w1 tvoří matici [počet vstupů, počet neuronů v první skryté vrstvě] náhodných čísel. Za chybovou funkci bylo vybráno MSE a jako optimizer byl zvolen Adam.

```
loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

Před samotným učením jsou inicializovány všechny proměnné.A následné učení proběhlo v několika krocích.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for i in range(1, num_steps+1):
    _, l, summary = sess.run([optimizer, loss, merged_summary_op],
    feed_dict={X: MatX,Y: MatY})
```

Na obrázku 5.2 je vidět výsledné porovnání vstupních zašuměných dat (zelená), průběh funkce sin bez šumu (černá) a výstup z neuronové sítě (červená). Vytvořený program je v příloze.



Obrázek 5.2: Odšumění vstupních dat (zelená) za pomoci hluboké neuronové sítě. Výstupem hluboké neuronové sítě jsou data bez šumu (červená)

5.4 Tensorboard

Složité výpočty, kvůli kterým je využívána knihovna tensorflow, často trvají dlouho dobu[2]. Pokud by bylo potřeba sledovat průběh učení, je několik možností:

1. vypisování loss a accuracy do příkazového řádku
2. tensorboard
3. kombinace obou dvou předchozích možností

Většinou je využita kombinace obou dvou možností, tensorboard je však zcela zásadní pro optimalizaci složitějších NS, protože plní dvě hlavní funkce [14]:

1. Zobrazuje loss, accuracy a další skalární veličiny v podstatě v reálném čase
2. Umožňuje vykreslit výpočetní graf

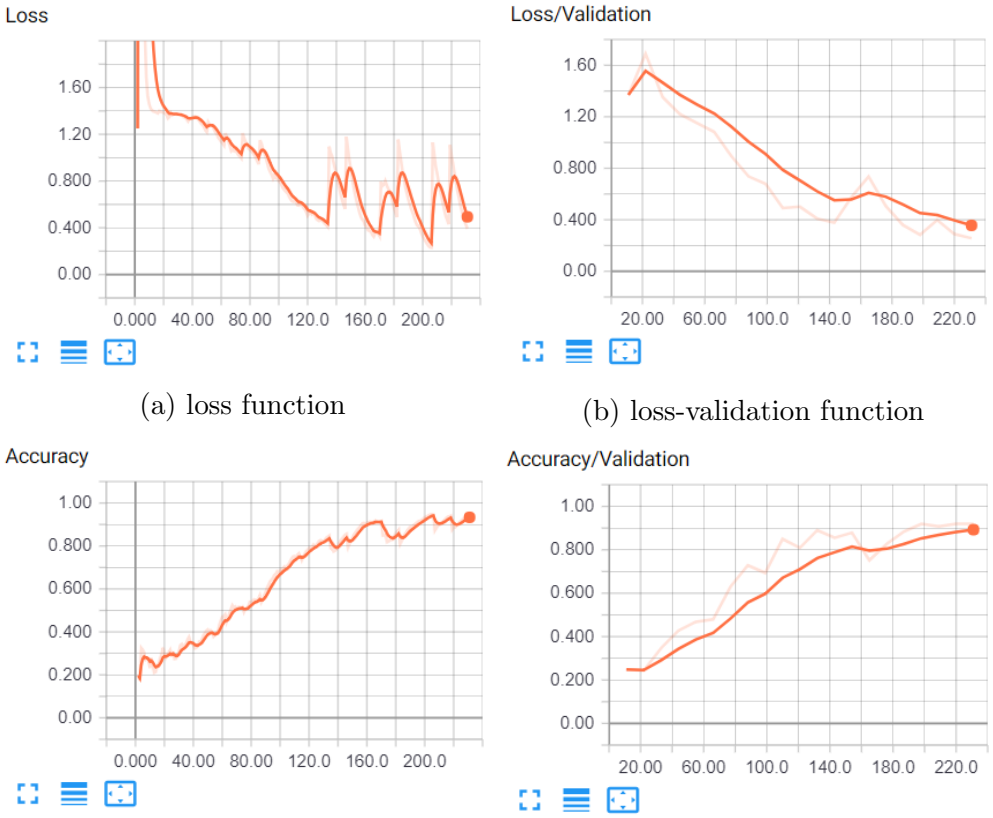
[2]Například natrénovat jednoduchou konvoluční neuronovou síť zabere na průměrném počítači řádově desítky minut.

Tensorboard běží paralelně s trénováním NS na lokálním serveru. Trénovaný model má obvykle definovaný adresář, kam průběžně ukládá informace učení. Jednoduchým příkazem lze spustit lokální server s daným adresářem a na něm sledovat průběh učení:

```
tensorboard --logdir="cesta_k_modelu"
```

5.4.1 Vykreslování skalárních funkcí

Při průběhu učení je většinou nejzajímavější loss function a accuracy, což umožňuje například zjistit, zda se model vůbec učí[3].



(a) loss function

(b) loss-validation function

(c) accuracy

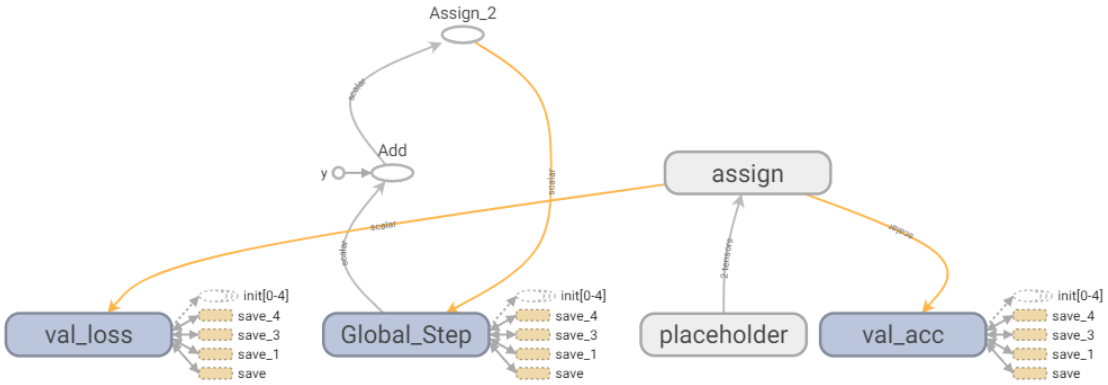
(d) accuracy-validation

Obrázek 5.3: Zobrazování skalárních veličin v tensorboard

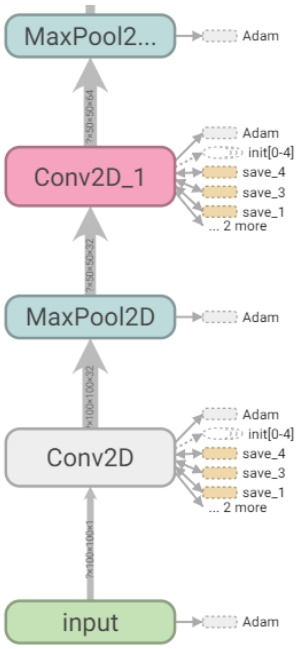
[3]Pokud loss function neklesne za první 3 epochy učení, pravděpodobně nemáme správný model.

5.4.2 Zobrazení grafu

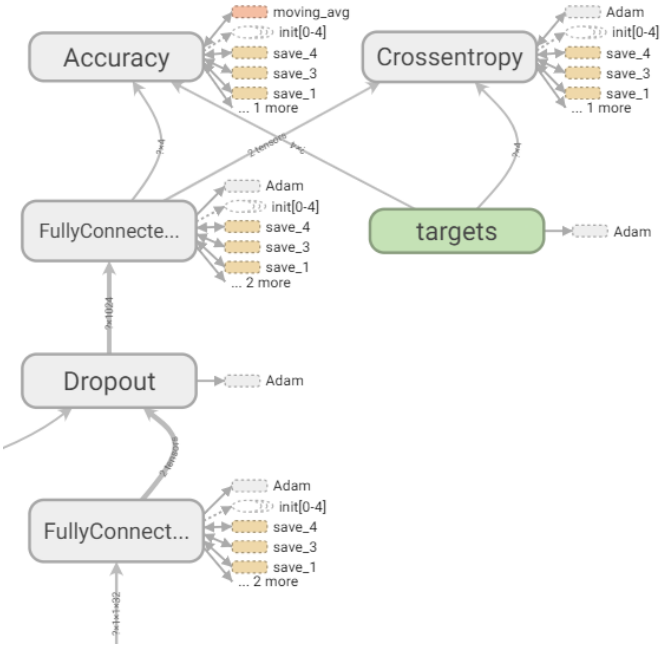
Pokud je konstruován graf v tensorflow, často jsou vytvořeny velmi složité struktury, tensorboard umí i takto složitý graf vizualizovat. Jedná se o výbornou pomůcku při optimalizaci a odstranění chyb či zlepšování modelu.



(a) loss function a global step



(b) Vrstvy v NN

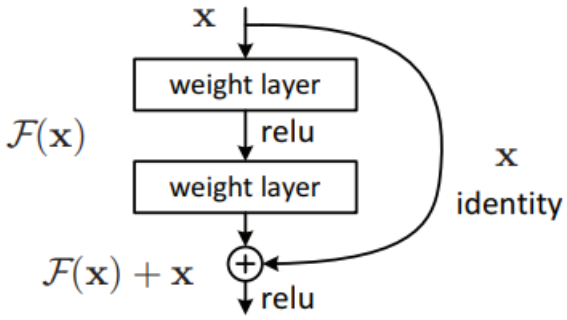


(c) Graf trénování

Obrázek 5.4: Části grafu v tensorboard

5.5 Residual network - ResNet

Hluboké konvoluční neuronové sítě vedly k sérii průlomů v rozpoznávání obrazu i v mnohých jiných rozpoznávacích problémech. Trendem se stalo jít co nejhlouběji v neuronové síti (mít co nejvíce skrytých vrstev), tak aby NS byla schopna řešit složitější úkoly a také aby se výsledná přesnost NS zlepšila [15]. Ale se s zvětšující se hloubkou NS se také zvětšuje problém učit síť, nastává také problém s přesností, která se sice nejdříve začne zlepšovat, ale časem se zhorší. [4]. ResNet se snaží vyřešit oba tyto problémy. Obecně se CNN při řešení problému snaží rozpoznat několik funkcí na konci každé vrstvy. U Residual učení, se už síť nesnaží naučit nové funkce, ale doučit se zbytkové funkce. K tomu využívá ResNet přeskakování vrstev (vstup do vrstvy n je napojen do vrstvy $n + x$). Pro Residuálního učení se předpokládá, že pokud by několik vrstev bylo schopno asymptoticky aproximovat funkci $H(x)$, přičemž x je vstup do první z nich, tak by tyto nelineární vrstvy byly schopny asymptoticky aproximovat i reziduální funkci $F(x) = H(x) - x$ (samozřejmě pokud $H(x)$ a x mají stejnou velikost). Na obrázku 5.5 je vidět ukázka reziduálního bloku.



Obrázek 5.5: ResNet blok [15]

[4]Nejedná se o syndrom přeučení, při kterém se NS zaměří na detaily z trénovací množiny a tím ztrácí obecnost řešení.

Kapitola 6

Návrh CNN pro rozpoznávaní obrazu

Tato CNN byla navržena za účelem rozpoznávání obrazu. Cílem sítě je, aby byla schopna roztřídit vstupní obrazy do tří kategorií. Jako kategorie byly vybrány dveře, okna a schody. Neuronová síť byla navržena v prostředí Python a to za použití několika knihoven, avšak pro NS byla nejdůležitější knihovna Tensorflow. Pro testování bylo vytvořeno několik CNN, které byly následně porovnány a z nich vybrána jedna s nejlepší přesností. Data použitá pro učení těchto modelů jsou získána z 360° fotografií pořízených skenerem a také z databází MCIndoor20000 [16] a ImageNet [17]

6.1 Data preprocessing

Data ze skeneru byla získána v prostředí softwaru FARO Scene, což je program určený ke zpracování dat ze skenerů. Jelikož cílem této práce je detekce obrazu, a ne rozpoznávaní objektu, tak byly využity pouze části fotografií. Z fotografie tak byly exportovány pouze obrázky se zmíněnými kategoriemi. Ukázka obrazu použitého při validaci CNN je vidět na obrázku 6.1.

Obrázek 6.1: Obraz ze skeneru

MCIndoor20000 je veřejná databáze získaná z kliniky Marshfield. Databáze je plně roztříděná a její použití je zdarma pro výzkumné účely. Data set MCIndoor20000 se skládá z více než 20 000 digitálních obrázků, které jsou rozřazeny do tří kategorií, dveře, schody a nemocniční značky. Aby byla databáze rozsáhlejší, tak jsou obrazy rotovány, zrcadleny, zašuměny atp. Databáze je dostupná na adrese https://github.com/bircatmcri/MCIndoor20000.

Obrázek 6.2: Obraz z databáze MCIndoor20000

ImageNet je velká databáze obrázků, která je vytvořená k využití při tvorbě NS pro rozpoznávání objektů. Tato databáze obsahuje přes 14 milionů obrázků, které jsou rozřazeny do 27 kategorií, které se následně ještě dělí do subkategorií. Pro učení vytvořených modelů byly použito kolem 700 obrázků pro každou kategorii a pro testování bylo použito celkem 90 obrázků. Jelikož jsou data takto různorodá, obrazy nemají jednotný formát, tak je potřeba nejdříve tyto fotky unifikovat.

Hlavní kategorie	podkategorie	průměrně obrázků na podkategorii	Celkem obrázků
Obojživelník	94	591	56K
Zvíře	3822	732	2799K
Spotřebič	51	1164	59K
Pták	856	949	812K
Krytina	946	819	774K
Přístroj	2385	675	1610K
Tkanina	262	690	181K
Ryba	566	494	280K
Kytka	462	735	339K
Jídlo	1495	670	1001K
Ovoce	309	607	188K
Houba	303	453	137K
Nábytek	187	1043	195K
Geologie	151	838	127K
Bezobratlý	728	573	417K
Savec	1138	821	934K
Hudební nástroj	157	891	140K
Rostlina	1666	600	999K
Plaz	268	707	190K
sport	166	1207	200K
Budova	1239	763	946K
Nástroj	316	551	174K
Strom	993	568	564K
Nádoba	86	912	78K
Zelenina	176	764	135K
Vozidlo	481	778	374K
Osoba	2035	468	952K

Tabulka 6.1: ImageNet databáze, data z roku 2010 [17]

6.1.1 Unifikace dat

V prostředí Python byla vytvořena funkce, která jednotlivé fotky upravila a rozřadila je do jednotlivých kategorií. Použitý průběh zpracování:

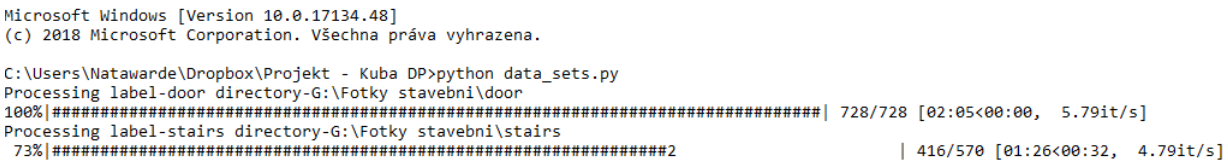
1. Rozřazení fotek do jednotlivých složek dle kategorie (velké usnadnění díky kategoriím z použitých databází)
2. Načtení složky a první obrázek v této složce
3. Úprava obrázku - úprava velikosti obrázku a převedení na stupně šedi
4. Uložení zpracovaného obrázku do trénovacího pole s popisem dle aktuální složky
5. Dle kroku 3. a 4. zpracování všech obrázků ve složce a následný přechod na další složku 1.
6. Proházení pořadí trénovacích dat
7. Uložení trénovacích dat

(a) Před zpracováním

(b) Po zpracování

Obrázek 6.3: Porovnání obrazu před a po zpracování

Testovací množina byla upravena stejně jako trénovací.



Obrázek 6.4: Průběh předzpracování dat

6.2 Vytvoření modelu

Pro testování bylo navrženo několik modelů, avšak všechny vychází ze stejného základu - CNN. U všech modelů byla použita aktivační funkce ReLu, optimizer Adam, a chybová funkce typu categorical cross entropy. K naučení a testování modelů byla použita nadstavba Tensorflow a to TFLearn. U jednotlivých modelů byl změněn počet vrstev, learning rate, počet trénovacích epoch a také rozměr obrázků.

6.2.1 Realizace modelů v prostředí Python

Při vytváření programu v prostřed Python(py) je nejprve potřeba naimportovat správné knihovny:

```
import numpy as np
import matplotlib.image as mpimg
import PIL import Image
import tqdm import tqdm
import random import shuffle
```

NumPy je jedna ze základních knihoven pro pracování s Pythonem. Umožňuje využívat n rozměrné pole a má různé další analytické funkce.

Knihovna Matplotlib slouží obecně k zobrazování grafů. Její součást matplotlib.image slouží k zobrazení obrazů převedených do maticového tvaru.

PIL slouží k zpracování obrázků, umožňuje funkce jako je .reshape (změna velikosti obrazu), či .convert(převedení obrazu například na stupně šedi).

Visuální knihovna tqdm slouží k přehlednějšímu zobrazení průběhu programu.

Příkaz shuffle z knihovny random je použit k promíchání pořadí jednotlivých obrazů v připraveném data setu.

```
Classification = ['window','door','stairs']
ImgSize = 120
lr = 0.01
```

V této části kódu jsou popsány jednotlivé kategorie, je definována velikost obrázku ($120x120$) a specifikována learning rate.

```
ModelName ='classificator-{}-{}.model'.format(lr,ImgSize)
traindata = np.load(pathtraindata.npy)
import tflearn
from tflearn.layers.conv import conv2d, maxpool2d
from tflearn.layers.core import inputdata, dropout, fullyconnected
from tflearn.layers.estimator import regression
```

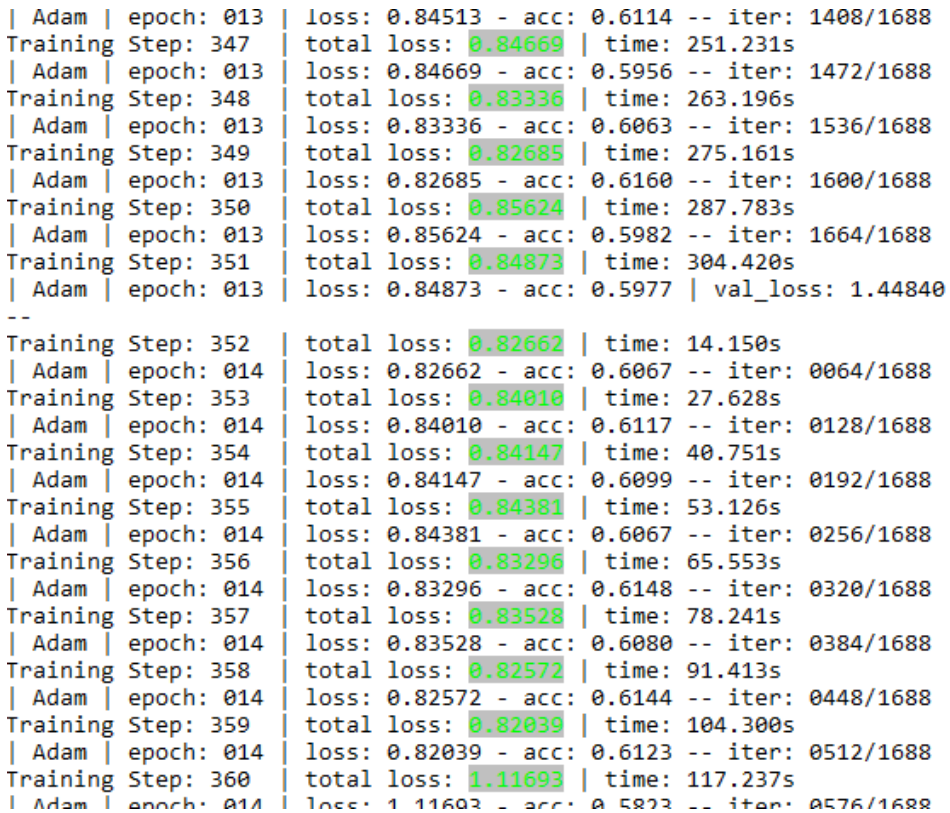
Nejdříve proběhlo pojmenování modelu dle learning rate a ImgSize. V dalším řádku kódu se načítají trénovací data, přičemž path značí cestu k nim. Následně jsou importovány zbylé knihovny sloužící k vytvoření a naučení modelu v knihovně tflearn.

```
convnet = inputdata(shape=[None, ImgSize, ImgSize, 1], name="input\)

convnet = conv2d(convnet,32,6,activation='relu')
convnet = maxpool2d(convnet,6)

...

convnet = fullyconnected(convnet,512,activation='relu')
convnet = dropout(convnet,0.75)

convnet = fullyconnected(convnet,len(TrainDirDict),activation='softmax')
convnet = regression(convnet,optimizer='adam',learningrate=lr,
loss='categoricalcrossentropy',name='relutargets')
model = tflearn.DNN(convnet,tensorboard_dir='log')
```

Tato část kódu definuje architekturu neuronové sítě. Příkaz *input_data* lze interpretovat jako vytvoření vstupní vrstvy. Příkazy *conv_2d* a *max_pool_2d* vytváří konvoluční vrstvu a následný pooling. Obdobně by se dalo vytvořit neomezené množství vrstev. Následně jsou vytvořeny plně propojené vrstvy (fully_connected). Poslední plně propojená vrstva je brána jako výstupní, proto je také tvořena ze 3 neuronů a její aktivační funkce je softmax. Funkcí *regression* definujeme parametry sítě jako je optimizer, learning rate a chybovou funkci (loss) a *name* pro pozorování. Jako poslední *tflearn.DNN* slouží pro uložení modelu do logu pro zobrazení v Tensorboard.

```
train = train_data[:-300]
test = train_data[-300:]

X = np.array([i[0] for i in train]).reshape(-1,ImgSize,ImgSize,1)
Y = [i[1] for i in train]

test_x = np.array([i[0] for i in test]).reshape(-1,ImgSize,ImgSize,1)
test_y = [i[1] for i in test]

model.fit({'input': X}, {'targets': Y}, n_epoch=20, validation_set=
({'input': test_x}, {'targets': test_y}), snapshot_step=500,
show_metric=True, run_id=ModelName)

model.save(ModelName)
```

Nejdříve jsou data rozdělena do dvou skupin *test* a *train*. Následně jsou data extrahovaná na vstupy a label. Trénovací data jsou uložena do proměnné X, přičemž jsou ještě konvertována na formát pro funkci $model.fit$. Odpovídající label k datům X je označen proměnnou Y. Obdobně jsou zpracována testovací data pro $model.fit$. Na závěr jsou ve funkci $model.fit$ definovány vstupy, výstupy a tréninková množina (díky níž počítá tato funkce přesnost během učení), zobrazovací periodu vzorků a id modelu. Na závěr je model uložen. Průběh učení jednoho z modelů je vidět na obrázku 6.5



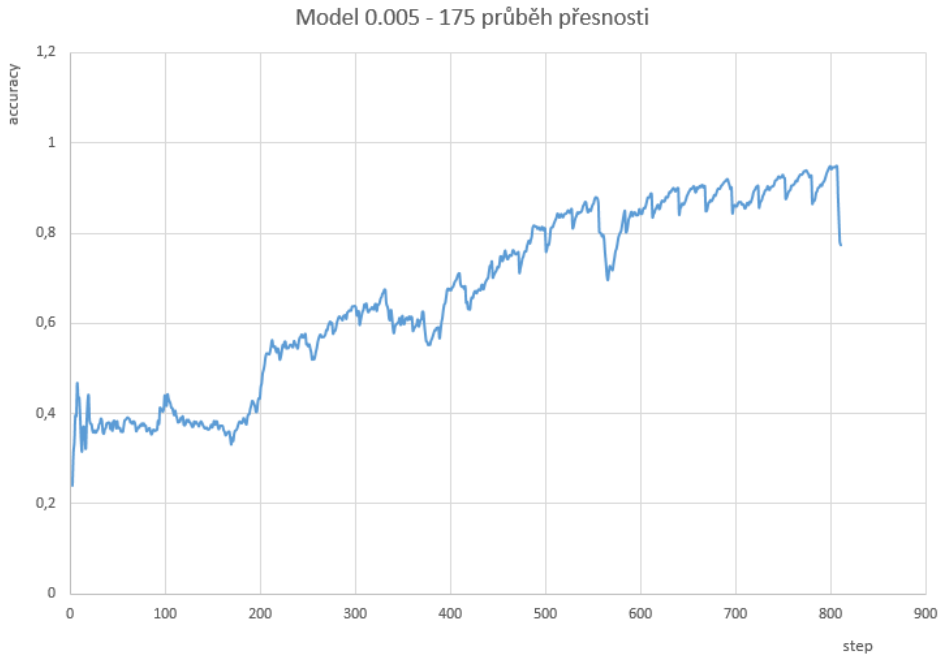
Obrázek 6.5: Průběh trénování NS

6.2.2 Model-0.005-175

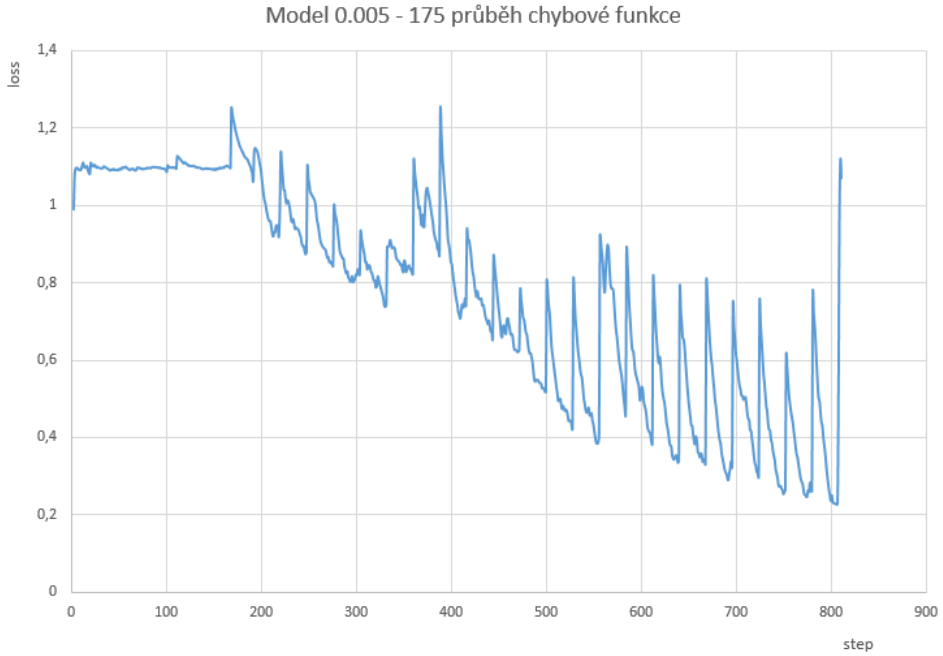
Parametry modelu: learning rate = 0.005 ImageSize = 175 epoch = 30

Konvoluční vrstva	32 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	1024 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	32 filtrů	stride = 6	aktivační funkce = ReLu
Max pooling			
Plně propojená vrstva	512 neuronů	aktivační funkce = ReLu	
Plně propojená vrstva	3 neurony	aktivační funkce = softmax	

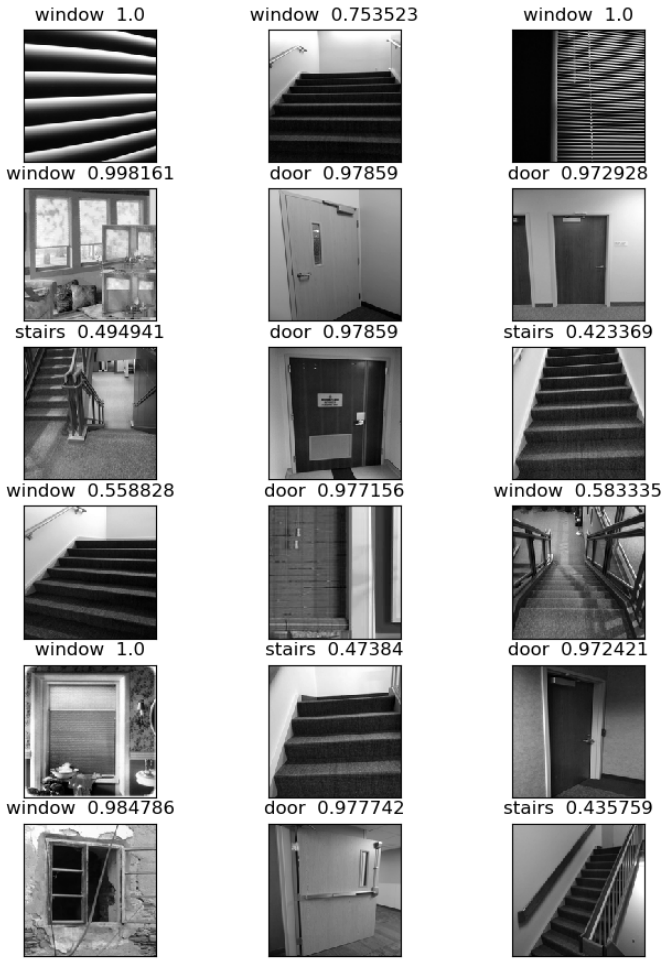
Tabulka 6.2: Architektura modelu-0.005-175



Obrázek 6.6: Průběh přesnosti modelu-0.005-175 během učení



Obrázek 6.7: Průběh chybové funkce během učení u modelu-0.005-175



Obrázek 6.8: Výsledné rozpoznání obrázků modelu-0.005-175

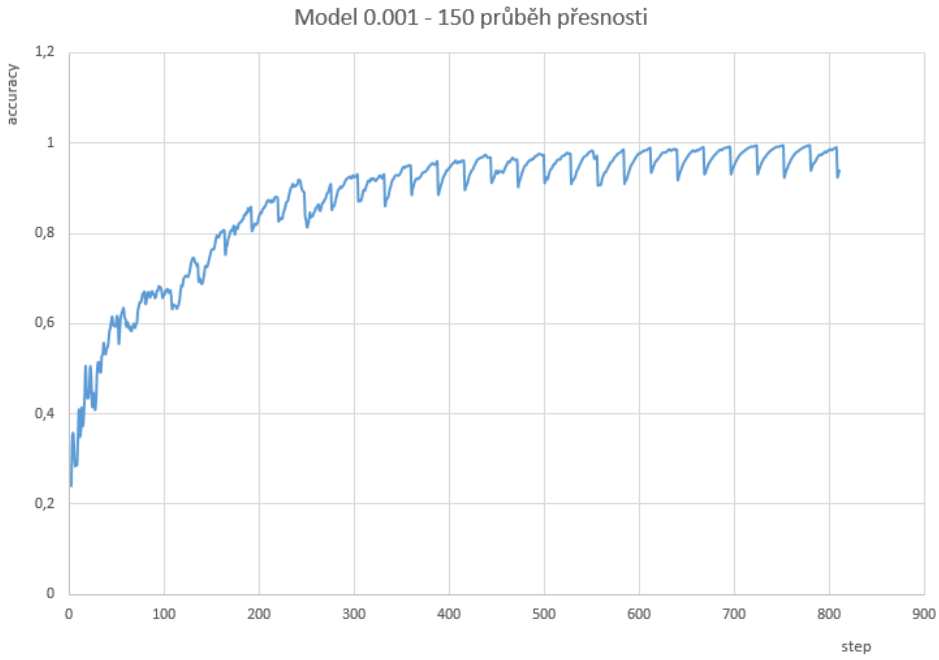
Tento model je vyhodnocen jako jeden z úspěšnějších. Model dosáhl po 30 epochách přesnosti kolem 80%, z grafu 6.6 lze vyčíst, že přesnost byla ještě o něco vyšší, ale ke konci učení došlo k náhlému propadu. Z obrázku 6.8 je patrné, že model má problém s rozpoznáním oken a schodů.

6.2.3 Model-0.001-150

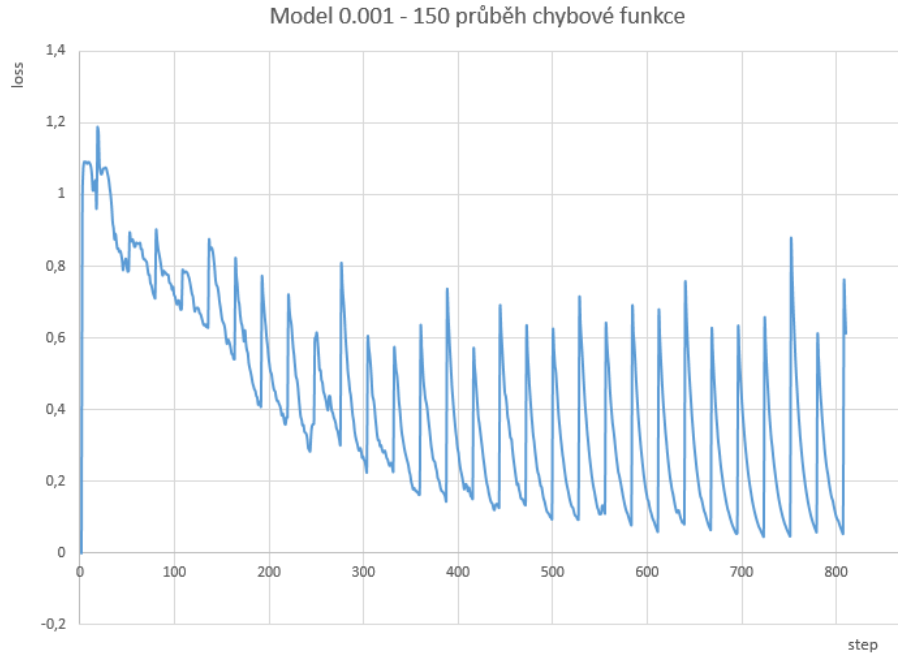
parametry modelu: learning rate = 0.001 ImageSize = 150 epoch = 40

Konvoluční vrstva	32 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	32 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Plně propojená vrstva	1024 neuronů	aktivační funkce = ReLu	
Plně propojená vrstva	3 neurony	aktivační funkce = softmax	

Tabulka 6.3: Architektura modelu-0.001-250



Obrázek 6.9: Průběh přesnosti modelu-0.001-150 během učení



Obrázek 6.10: Průběh loss funkce u modelu-0.001-150

Obrázek 6.11: Výsledné rozpoznání obrázků u modelu-0.001-150

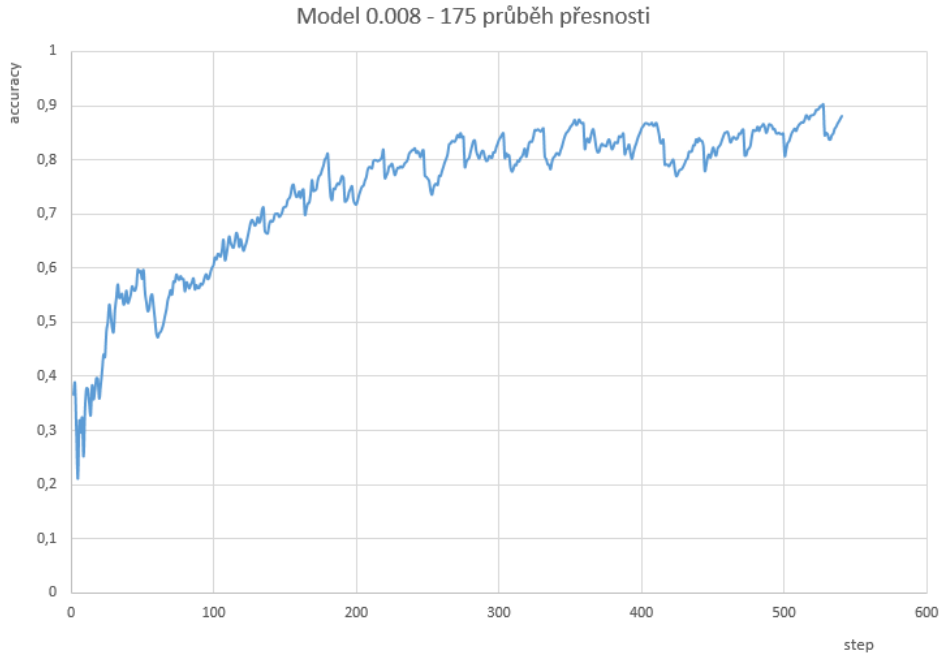
Tento model dosáhl nejlepších výsledků v porovnání s ostatními vytvořenými modely. Přesnost modelu dosáhla hodnoty cca 90%. K trénování bylo poskytnuto 40 epoch a learning rate 0.001. Na obrázku 6.11 je vidět, že model odhadl správně 14 z 15 obrázků.

6.2.4 Model-0.008-175

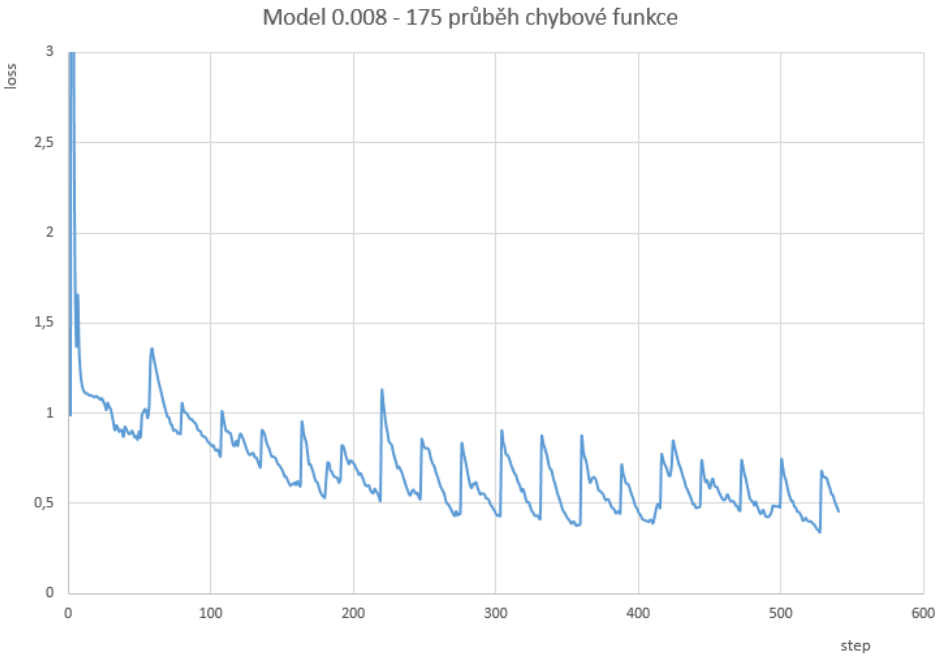
parametry modelu: learning rate = 0.008 ImageSize = 175 epoch = 20

Konvoluční vrstva	32 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	32 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Plně propojená vrstva	1024 neuronů	aktivační funkce = ReLu	
Plně propojená vrstva	3 neurony	aktivační funkce = softmax	

Tabulka 6.4: Architektura modelu-0.008-175



Obrázek 6.12: Průběh přesnosti modelu během učení u modelu-0.008-175



Obrázek 6.13: Průběh loss funkce u modelu-0.008-175

Obrázek 6.14: Výsledné rozpoznání obrázků u modelu-0.008-175

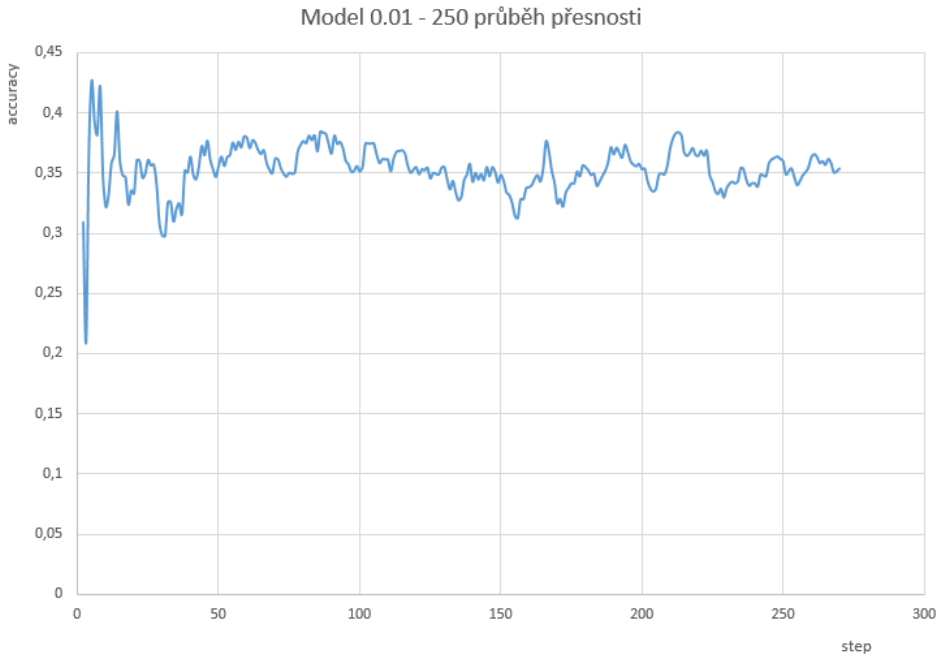
Tento model je svou architekturou velmi podobný modelu-0.001-150, který dosáhl vynikajících výsledků. U tohoto modelu se liší learning rate, která je zvolena na hodnotu 0.008 a velikost obrázku, která je oproti modelu-0.001-150 o 25pixelů na stranu větší. Tyto změny se projevily negativně a model nedosahuje tak dobrých výsledků. Výsledná přesnost je pod 90%. I tak je tento model dostatečně natrénován k detekci většiny obrazů. Model z dotázaných 15 obrázků odhadl tři špatně.

6.2.5 Model-0.01-250

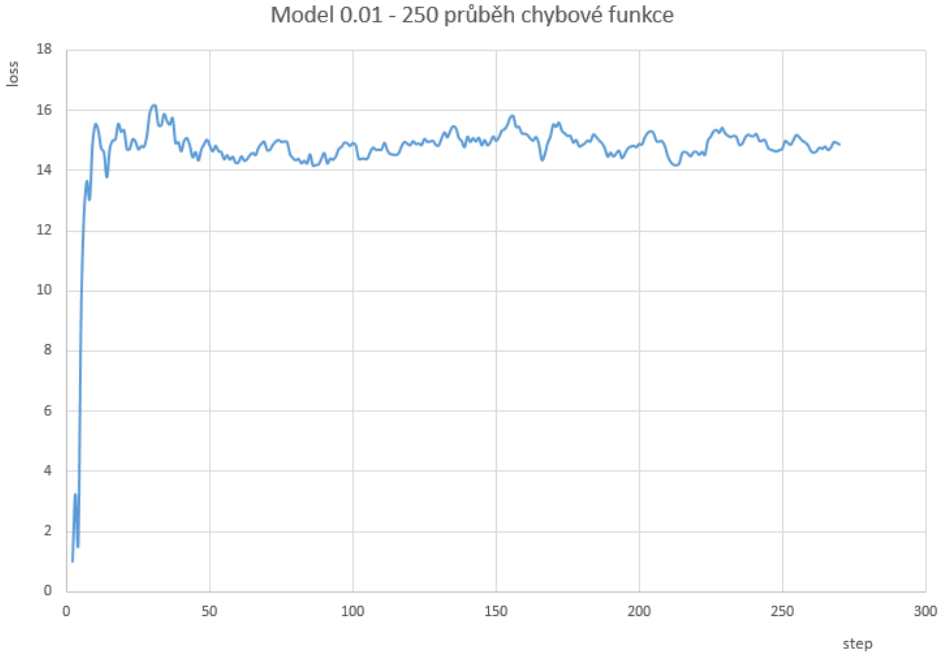
parametry modelu: learning rate = 0.01 ImageSize = 250 epoch = 10

Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	256 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	1024 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	1024 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	512 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	128 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Konvoluční vrstva	64 filtrů	stride = 5	aktivační funkce = ReLu
Max pooling			
Plně propojená vrstva	512 neuronů	aktivační funkce = ReLu	
Plně propojená vrstva	3 neurony	aktivační funkce = softmax	

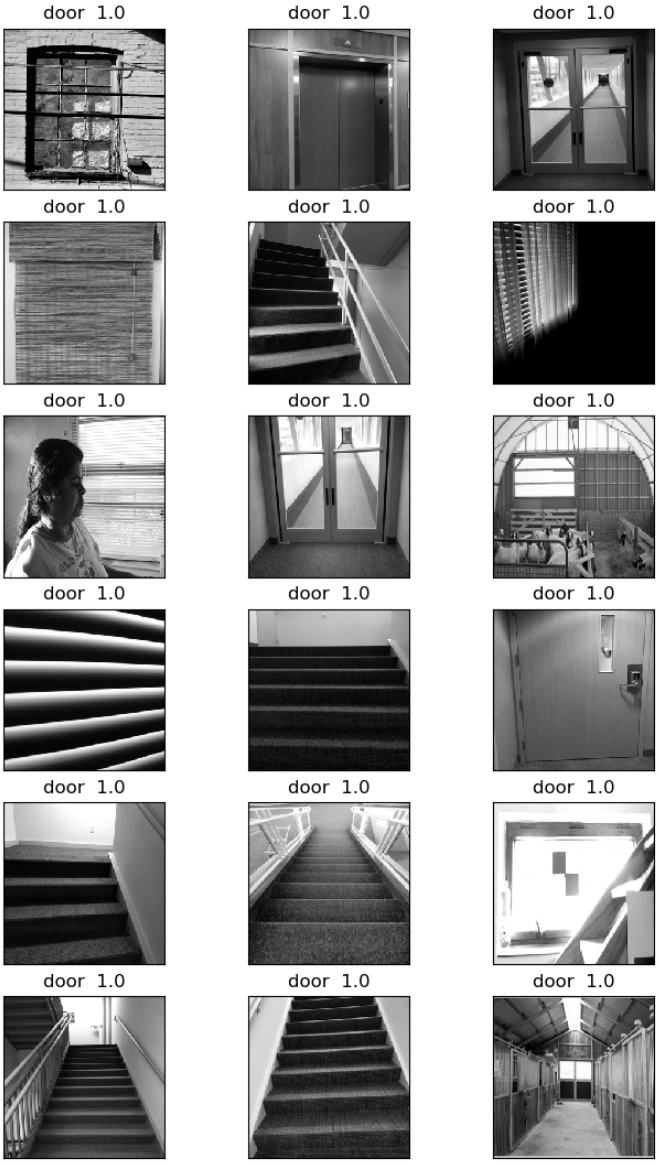
Tabulka 6.5: Architektura modelu-0.01-250



Obrázek 6.15: Průběh přesnosti modelu během učení u modelu-0.01-250



Obrázek 6.16: Průběh loss funkce u modelu-0.01-250

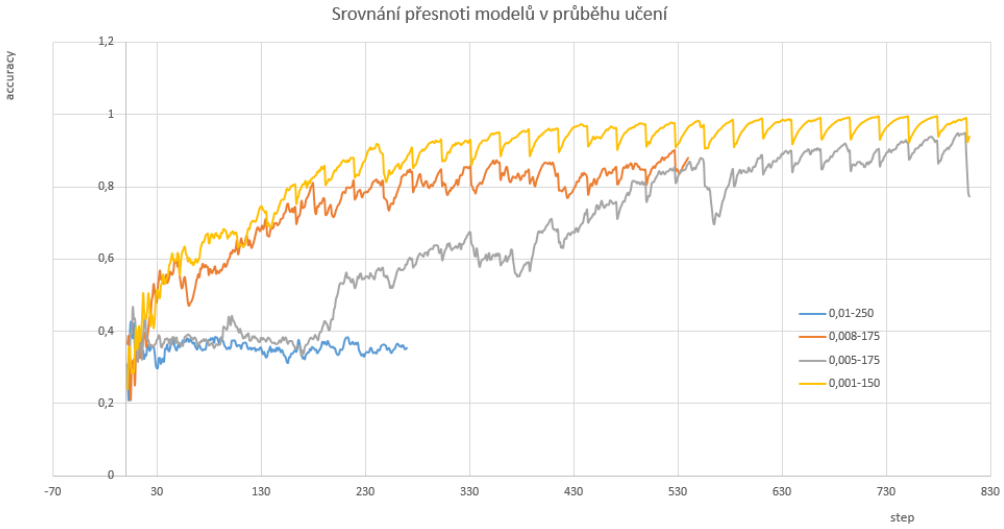


Obrázek 6.17: Výsledné rozpoznání obrázků u modelu-0.01-250

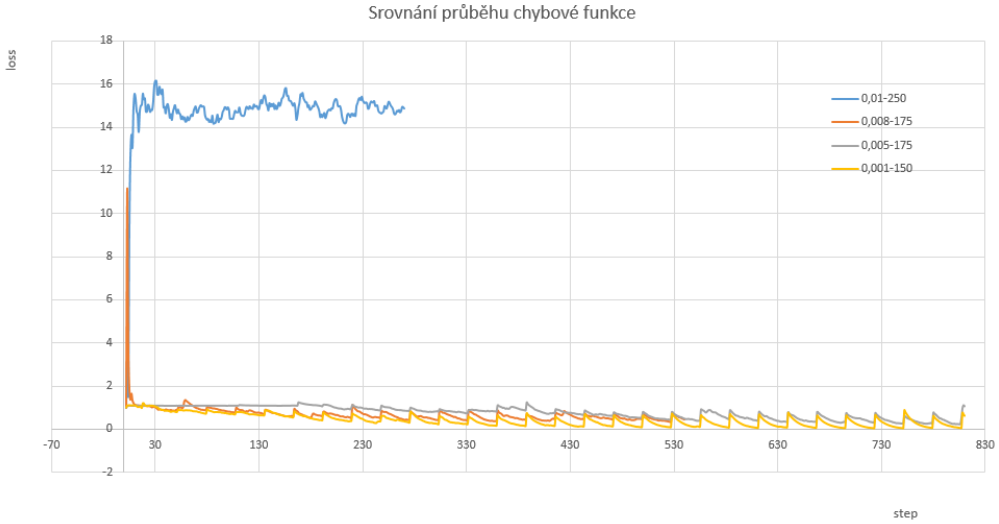
Model s poměrně vysokou learning rate (0.01) a rozlišením obrázků 250x250 se ukázal jako chybný. K učení tohoto modelu bylo poskytnuto pouze 10 epoch, ale z grafů 6.16 a 6.15 je patrné, že model by pravděpodobně větších přesností nedosáhl ani při zvýšeném počtu trénovacích epoch. Celkově je tento model hodnocen jako neúspěšný a nepoužitelný. Všechny dotazované obrázky byly vyhodnoceny jako dveře.

6.3 Porovnání modelů

V této kapitole jsou porovnány 4 navržené modely. Všechny modely až na Model-0.01-250 se pohybují s výslednou úspěšností kolem 80-90%, což je relativně dobrý výsledek. Naopak Model-0.01-250 se při učení zasekl na přesnosti kolem 33% a trénovací algoritmus nebyl schopen zlepšení. Na grafech 6.18 a 6.19 lze vidět srovnání průběhu přesnosti a porovnání chybové funkce pro tyto modely.



Obrázek 6.18: Porovnání dosažené přesnosti 4 navržených modelů



Obrázek 6.19: Porovnání průběhů chybových funkcí 4 navržených modelů

Jako model s nejlepšími výsledky lze vybrat Model-0.001-150. Je to vůči ostatním model s nejmenším learning rate a s nejmenším rozlišení obrázku. Při trénování tohoto modelu bylo nastaveno nejvíce epoch, ale už od počátku algoritmu učení se výsledky tohoto modelu jeví jako nejlepší.

6.4 Závěr

Cílem této diplomové práce bylo provést rešerši na metody zpracování obrazu a to včetně metod založených na hlubokých neuronových sítí a tyto aplikovat na rozpoznávání objektů skenované budovy. Obsahem práce měl tedy také být návrh vlastní hluboké neuronové sítě pro rozpoznání obrazu. Při vytváření hluboké NS pro rozpoznání obrazu bylo nejdříve potřeba zvolit vhodné prostředí. K řešení tohoto problému byl vybrán programovací jazyk Python a k realizaci sítě především knihovna Tensorflow.

První překážkou bylo získání dostatku dat k trénování hluboké NS. Bylo použito cca 700 obrázků z každé kategorie, která měla být rozpoznávána. Bylo rozhodnuto, že kategorie budou tři - dveře, okna a schody. Trénovací data byla získána z databází ImageNet, MCIndoor20000 a část byla ručně vyexportována z fotografii pořízených 3D skenerem. Po získání dat bylo nutné data předzpracovat a unifikovat. Obě tyto části byly také realizovány v prostředí Python. Během předzpracování byly obrazy překonvertovány na stejný rozměr a následně všechny převedeny do stupně šedi.

Po získání dostatečného počtu trénovacích dat bylo potřeba navrhnout architekturu hluboké neuronové sítě. Ke zpracování obrazu je nejvíce využíván typ hlubokých neuronových sítí s názvem konvoluční neuronová síť (CNN). Při hledání správné konfigurace CNN bylo vyzkoušeny různé aktivační funkce, chybové funkce, různé počty neuronů, vrstev atp. Po různých testech byly do této práce uvedeny 4 modely.

Každý model má odlišnou architekturu a specifika, jako jsou rozlišení vstupního obrazu, learning rate a počet učících epoch. Modely se naopak shodují v aktivačních funkcí použitých v jednotlivých vrstvách (byla vybrána funkce ReLu, která při experimentech dosahovala nejlepších výsledků). Na závěr celé práce proběhlo vyhodnocení těchto modelů a porovnání jejich výsledné přesnosti. Model-0.01-250 (learning rate = 0.01 a rozlišení obrazu 250x250) byl jako jediný zcela neúspěšný. Přesnost tohoto modelu uvízla kolem hodnoty 33%, detekoval tak všechny obrázky jako dveře (door). Zbylé tři modely dosáhly srovnatelného výsledku s přesností nad 80%. Nejlepší výsledky dosáhl model-0.001-150, který byl schopen dosáhnout výsledné přesnosti cca 90%. Tento model měl vůči ostatním nejnižší learning rate (0.001) a nejmenší rozlišení obrazu (150x150), je tvořen z 9 konvolučních vrstev a dvou plně propojených vrstev (více o tomto modelu tab. 6.3).

Literatura

[1] Jiří HOZMAN. Základní metody předzpracování obrazu. *HOZMAN, J. Základní metody předzpracování obrazu*, 2003.

[2] Jiří Št'astnỳ. *Netradiční metody a algoritmy pro rozpoznávání objekt technologické scény*. VUTIUM, 2006.

[3] Eva Volná. Neuronové sítě 1. *Ostrava: Ostravská univerzita v Ostravě. Vydání: druhé*, 2008.

[4] Jiří ŠÍMA and Roman NERUDA. Teoretické otázky neuronovỳch sítí. praha: Matfyzpress,(1996), 390 s. Technical report, ISBN 80-85863-18-9, 1996.

[5] Roman BISKUP. Možnosti neuronovỳch sítí. *Disertační práce. Praha: Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta*, 2009.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Michael A Nielsen. *Neural networks and deep learning*. Determination Press, 2015.

[8] Yong Li, Yang Fu, Hui Li, and Si-Wen Zhang. The improved training algorithm of back propagation neural network with self-adaptive learning rate. In *Computational Intelligence and Natural Computing, 2009. CINC'09. International Conference on*, volume 1, pages 73–76. IEEE, 2009.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Martin Schmid. Konvoluční neuronové sítě a jejich implementace. *dspace.cuni.cz*, 2011.

[13] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[14] Tensorboard: Visualizing learning — tensorflow.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Fereshteh S Bashiri, Eric LaRose, Peggy Peissig, and Ahmad P Tafti. Mcindoor20000: a fully-labeled image dataset to advance indoor objects detection. *Data in Brief*, 2018.

[17] Princeton University Stanford Vision Lab, Stanford University. Imagenet: an image database organized according to the wordnet hierarchy, 2016, http://image-net.org/index, 2018-03-08.

Seznam obrázků

Seznam tabulek

Kapitola 7

Přílohy

7.1 Vytvoření modelu CNN pro rozpoznání obrazu

```
# imports
import os
import numpy as np
import matplotlib.image as mpimg

from PIL import Image
from tqdm import tqdm
from random import shuffle

# parameters
TrainDirDict = {
    'door':'G:\\Fotky stavebni\\door',
    'window':'G:\\Fotky stavebni\\window',
    'stairs':'G:\\Fotky stavebni\\stairs'
}

#Classification = list(TrainDirDict.keys())
Classification = ['door','stairs','window']
TestDir = 'G:\\Fotky stavebni\\test'
ImgSize = 250
lr = 0.011

# ___________________MODEL_________________
```

```
ModelName = 'classificator-{}-{}.model'.format(lr,ImgSize)

train_data = np.load('C:\\Users\\Natawarde\\Dropbox\\Diplomka\\data_sets
\\train_data.npy')

import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

convnet = input_data(shape=[None, ImgSize, ImgSize, 1], name='input')

convnet = conv_2d(convnet,64,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,128,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,256,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,512,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,1024,5,activation='relu')
convnet = max_pool_2d(convnet,5)


convnet = conv_2d(convnet,1024,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,512,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,256,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = conv_2d(convnet,128,5,activation='relu')
convnet = max_pool_2d(convnet,5)
```

```
convnet = conv_2d(convnet,64,5,activation='relu')
convnet = max_pool_2d(convnet,5)

convnet = fully_connected(convnet,512,activation='relu')
convnet = dropout(convnet,0.8)

convnet = fully_connected(convnet,len(TrainDirDict),activation='softmax')
convnet = regression(convnet,optimizer='adam',learning_rate=lr,
loss='categorical_crossentropy',name='targets')

model = tflearn.DNN(convnet,tensorboard_dir='log')


""" if os.path.exists('{}.meta'.format(ModelName)):
    model.load(ModelName)
    print('model loaded!') """

train = train_data[:-300]
test = train_data[-300:]

X = np.array([i[0] for i in train]).reshape(-1,ImgSize,ImgSize,1)
Y = [i[1] for i in train]

test_x = np.array([i[0] for i in test]).reshape(-1,ImgSize,ImgSize,1)
test_y = [i[1] for i in test]

model.fit({'input': X}, {'targets': Y}, n_epoch=2,
validation_set=({'input': test_x}, {'targets': test_y}),
    snapshot_step=500, show_metric=True, run_id=ModelName)

model.save(ModelName)

import matplotlib.pyplot as plt


test_data = np.load('C:\\Users\\Natawarde\\Dropbox\\Diplomka
\\data_sets\\test_data.npy')

fig=plt.figure()
```

```
for num,data in enumerate(test_data[0:15]):
    img_num = data[1]
    img_data = data[0]

    y = fig.add_subplot(5,3,num+1)
    orig = img_data
    data = img_data.reshape(ImgSize,ImgSize,1)

    model_out = model.predict([data])[0]
    print(model_out)
    max_index = np.argmax(model_out)
    str_label = '{} - {}'.format(Classification[max_index], model_out[max_index])
    y.imshow(orig,cmap='gray')
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
plt.show()
```

7.2 Vytvoření neuronové sítě pro odstranění šumu

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np

from functions import gen

logs_path = '/tmp/tensorflow_logs/example/'

""" Vytvoření vstupních dat"""
dt = 0.1
t = np.arange(0,2000,dt)
y1 = np.sin(t)
y2 = y1+np.random.rand(len(t))/2
y3 = y1+np.random.rand(len(t))/2

""" Datové parametry """
SamplesBack = 100
LearningDataLenght = 1000
```

```
MatX = gen(y2,SamplesBack,LearningDataLenght)
MatY = gen(y1,SamplesBack,LearningDataLenght)
MatTrain = gen(y3,SamplesBack,LearningDataLenght)


""" Parametry učení a zobrazování """
learning_rate = 0.001
num_steps = 1000
display_step = 100

""" Autoencoder architektura """
num_hidden_1 = 42
num_hidden_2 = 18
num_hidden_3 = 8
num_input = SamplesBack

X = tf.placeholder("float", [None, num_input])
Y = tf.placeholder("float", [None, num_input])

weights = {
    'encoder_w1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
    'encoder_w2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2])),
    'encoder_w3': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_3])),
    'decoder_w1': tf.Variable(tf.random_normal([num_hidden_3, num_hidden_2])),
    'decoder_w2': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_1])),
    'decoder_w3': tf.Variable(tf.random_normal([num_hidden_1, num_input])),
}
biases = {
    'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
    'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2])),
    'encoder_b3': tf.Variable(tf.random_normal([num_hidden_3])),
    'decoder_b1': tf.Variable(tf.random_normal([num_hidden_2])),
    'decoder_b2': tf.Variable(tf.random_normal([num_hidden_1])),
    'decoder_b3': tf.Variable(tf.random_normal([num_input])),
}

""" Encoder """
def encoder(x):
    # Encoder sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_w1']),
```

```
                                    biases['encoder_b1']))
    # Encoder  sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_w2']),
                                    biases['encoder_b2']))

    layer_3 = tf.nn.sigmoid(tf.add(tf.matmul(layer_2, weights['encoder_w3']),
                                    biases['encoder_b3']))
    return layer_3


""" Decoder """
def decoder(x):
    # Decoder  sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_w1']),
                                    biases['decoder_b1']))
    # Decoder sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_w2']),
                                    biases['decoder_b2']))

    layer_3 = tf.nn.tanh(tf.add(tf.matmul(layer_2, weights['decoder_w3']),
                                    biases['decoder_b3']))
    return layer_3

""" Model """
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)

# Predikce
y_pred = decoder_op
# Cíle
y_true = Y

# loss a optimizer, MSE
loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)

""" Tensorboard """
with tf.name_scope('Model'):
    # Model
    y_pred = decoder_op = decoder(encoder(X))
```

```
with tf.name_scope('Loss'):
    # cross entropy
    loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))

with tf.name_scope('Adam'):
    # Adam
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)

with tf.name_scope('Accuracy'):
    # Accuracy
    acc = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
    acc = tf.reduce_mean(tf.cast(acc, tf.float32))

# Summary na loss
tf.summary.scalar("loss", loss)
# Summary na accuracy
tf.summary.scalar("accuracy", acc)
# Sloučí
merged_summary_op = tf.summary.merge_all()



""" Training """
# Inicializace proměných
init = tf.global_variables_initializer()

# Začátek trénování
# Start a new TF session
with tf.Session() as sess:

    # Začátek init
    sess.run(init)

    summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

    # Trénování
    for i in range(1, num_steps+1):
        _, l, summary = sess.run([optimizer, loss, merged_summary_op],
```

```
            feed_dict={X: MatX,Y: MatY})
            # Summary
            summary_writer.add_summary(summary, i)

            # Zobrazení během učení
            if i % display_step == 0 or i == 1:
                print('Step %i: Batch Loss: %f' % (i, l))

    # Testing

    g = sess.run(decoder_op, feed_dict={X: MatTrain[:5]})

# Tisk
try:
    for _ in range(1):
        plt.figure(1)
        plt.plot(MatTrain[_],'g',label='Sinus with noise')
        plt.plot(g[_],'r',label='Ouput of autoencoder')
        plt.plot(MatY[_],'k',label='Sinus without noise')
        plt.legend()
        plt.xlabel('samples [-]')
        plt.ylabel('value [-] ')
        plt.show()
except:
    pass
```