Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Úvod do TensorFlow a jeho aplikace v indoor lokalizaci

Bakalářská práce

Autor: Daniel, Tesař
Studijní obor: Aplikovaná informatika (AI3-p)

Vedoucí práce: Ing. Pavel Kříž, Ph.D

Hradec Králové srpen 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 13. 8. 19 Daniel Tesař

Poděkování:

Děkuji Ing. Pavlovi Křížovi, Ph. D. za jeho odborné rady a dobré vedení při zpravování mojí bakalářské práce.

Anotace

Předmětem bakalářské práce je vysvětlit, co to je umělá neuronová síť, co to je TensorFlow a k čemu se používá. Následně vytvořit neuronovou síť k indoor lokalizaci v budově UHK FIM. Teoretická část je věnována k popisu neuronů a neuronových sítí jak biologických, umělých a jejich použití. Dále zde bude popsána knihovna od Google TensorFlow a nadstavba Keras. Praktická část bude porovnávat různé návrhy umělých neuronových sítí a jejich výsledky. V praktické části se bude pracovat s rádiovými fingerprinty bezdrátových zařízení v budově UHK FIM. Lokalizace pak bude probíhat pomocí naměření sil signálů těchto bezdrátových zařízení. Nakonec se podařilo navrhnout neuronovou síť, která dokázala lokalizovat polohu s odchylkou lehce přes jeden metr.

Klíčová slova

Neuron, neuronová síť, TensorFlow, Keras, lokalizace

Annotation

Title: Introduction to TensorFlow and its application in indoor localization

The topic of this bachelor thesis is to explain what is an artificial neural network, what is TensorFlow and what is it used for. Then create a neural network for indoor localization in the UHK FIM building. The theoretical part is devoted to the description of neurons and neural networks both biological, artificial and their use. Next will be described library from Google TensorFlow and extension Keras. The practical part will compare different designs of artificial neural networks and their results. The practical part will work with radio fingerprints of wireless devices in the UHK FIM building. Localization will then take place by measuring the signal strenght of these wireless devices. Finally, we managed to design a neural network that was able to locate a position with deviation slightly over one meter.

Key words

Neuron, neural network, Tensor Flow, Keras, localization

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM UKÁZEK KÓDU

SEZNAM GRAFŮ

1. ÚVOD

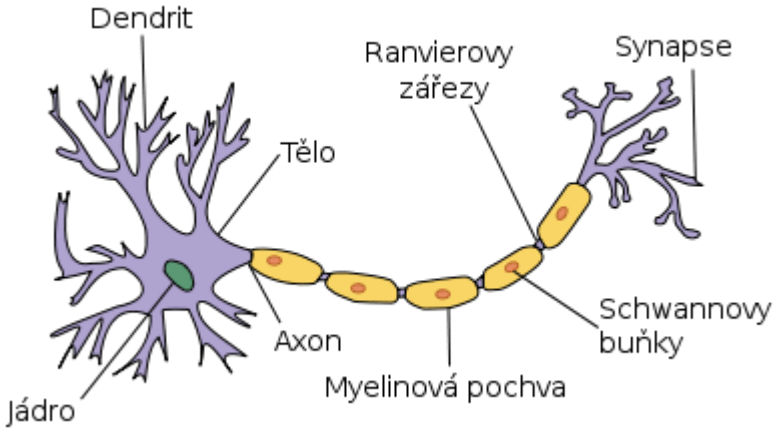
Cílem práce je seznámit se s fungováním umělých neuronových sítí a jejich podobností s fungováním biologické neurnové sítě. Běžní uživatelé totiž používají neuronové sítě na denní bázi a ani si to neuvědomují. A to například při hledání věcí na Google, při vyhledávání dálkových spojů, zobrazování reklam na internetu nebo při použití různých překladačů. Jako další bude seznámení s knihovnou TensorFlow od Google a její nadstavbou Keras, která je jednou z nejpoužívanějších knihoven k tvorbě neuronových sítích a je používána ve spoustě produktech, které lidi běžně používají. V této části by se mělo rozebrat, k čemu se neuronové sítě od TensorFlow používají a jak TensorFlow funguje. Také jak se budují neuronové sítě pomocí nadstavby Keras. Jaké existují typy modelů, typy optimalizátorů, typy metrik, typy aktivačních funkcí a co je to ztrátová funkce. Práce by také měla popisovat strategie výběru parametrů neuronové sítě.

V poslední řadě se bude práce zabývat navržením co nejlepšího modelu umělé neuronové sítě, který bude mít za úkol indoor lokalizaci v budově FIM UHK. Indoor lokalizací se v práci zabývá, protože například uživatelé GPS (Globální Polohový Systém) nemohou GPS používat k lokalizaci uvnitř budov. Celkově se zatím přesnost indoor lokalizace nemůže rovnat s přesností outdoor lokalizaci. V této části bude nejprve popsáno, jak byla získána data k učení, validaci a testování neuronové sítě. Jako další zde bude testováno několik variant modelu neuronové sítě. Budou zde porovnávány jejich výsledky a na základě těchto výsledků bude vytvořen co nejlepší model pro indoor lokalizaci.

2. BIOLOGICKÝ NEURON A NEURONOVÁ SÍŤ

Neuron je základní stavební prvek lidského mozku. Lidský mozek je tvořen šedou a bílou mozkovou kůrou. Šedá kúra je tvořena neurony a bílá kůra je tvořena nervovými vlákny. Neuron je schopen přijmout, vést, zpracovat a odpovědět na elektrické signály. Mozek člověka obsahuje okolo 15 až 25 miliard neuronů. Skladba neuronu je znázorněna na obrázku číslo 1. [1]

2.1.Bilogický neuron



Obrázek 1 - Skladba Neuronu [1]

Neuron je svojí skladbou podoben stromu. První a nejdůležitější součástí neuronu je buněčné jádro, které na základě vstupních signálů od předešlých neuronů a jiných částí těla generuje výstup dle aktivační funkce neuronu. Dendrit je pak nervové zakončení, které do neuronu přivádí tyto vstupní signály. Výstupní signál je z neuronu odváděn přes axon do synaptických zakončení, na kterých probíhá předání informace napojenému dendritu dalšího neuronu. Nervové vzruchy jsou v mozku, a tedy i mezi neurony přenášeny jako elektrické impulzy. [2]

2.2. Biologická neuronová síť

Neuron je přizpůsoben pro přenos signálů tak, že kromě vlastního těla, tzv. sómatu,má i vstupní a výstupní přenosové kanály: dendrity a axon. Z axonu obvykle odbočuje řada větví, tzv. terminálů, zakončených blánou, která se převážně stýká s dendrity jiných neuronů. K přenosu informace pak slouží synapse. Míra synaptické propustnosti je nositelem všech význačných informací během celého života organismu. Šíření informace je umožněno tím, že sóma i axon jsou obaleny membránou, která má schopnost za jistých

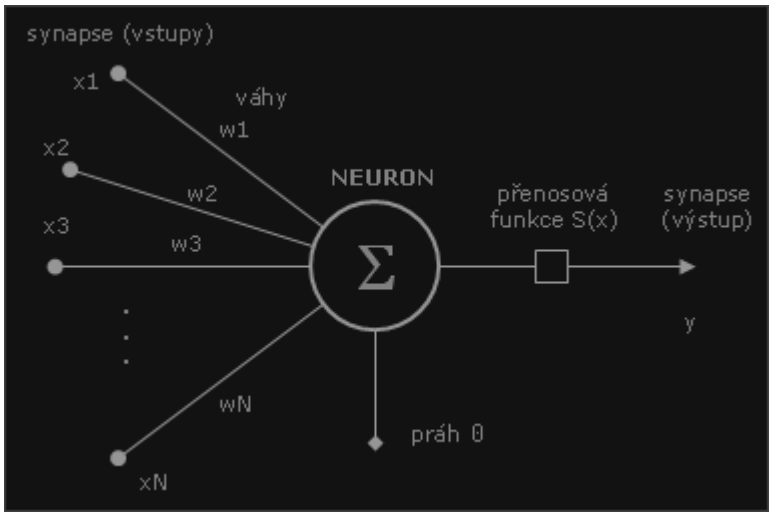
okolností generovat elektrické impulsy. Tyto impulsy jsou z axonu přenášeny na dendrity jiných neuronů synaptickými branami, které svojí propustností určují intenzitu podráždění dalších neuronů. Takto podrážděné neurony při dosažení určité hraniční meze, tzv. prahu, samy generují impuls a zajišťují tak šíření příslušné informace. Po každém průchodu signálu se synaptická propustnost mění, což je předpokladem paměťové schopnosti neuronů. Také propojení neuronů prodělává během života organismu svůj vývoj, v průběhu učení se vytváří nové paměťové stopy nebo při zapomínání se synaptické spoje přerušují. [2]

3. UMĚLÝ NEURON A UMĚLÁ NEURONOVÁ SÍŤ

Základem umělé neuronové sítě je neuron, který získáme přenesením biologického neuronu a jeho zjednodušení do matematického prostředí. Jeho struktura je podobná biologickému neuronu a je znázorněna na obrázku 2.

3.1. Umělý neuron a jeho okolí

Tělo neuronu je nahrazeno vstupními signály-vstupy. Synapse a dendrity jsou jako u biologického neuronu spojení s dalšími neurony. Synaptická propustnost je zde nahrazena váhou.[3]



Obrázek 2 - Neuron a jeho okolí [3]

Umělý neuron se skládá z několika částí:

- Synapse: Synapse je spojení mezi jednotlivými neurony. Počet synapsí určuje velikost neuronové sítě. Pokud síť obsahuje větší počet synapsí, může pracovat s větším množstvím dat.
- Váha: Každá synapse je reprezentována nějakou váhou. Váha násobí nesenou informaci v synapsi. Čím větší, tím větší má daná hodnota cenu. V podstatě se jedná o paměť hodnoty.
- Bias: Bias je systematická chyba, která vede ke zkreslení výsledků. Kromě první vrstvy má každý neuron vstup s hodnotou Bias.
- Práh: Práh je hodnota, při které se aktivuje výstup neuronu.
- Přenosové funkce: Přenosová funkce upravuje hodnoty na výstupu. Je několik typů přenosových funkcí:
    - Skoková funkce
    - Sigmoidální funkce
    - Hyperbolická funkce
    - Radiální báze
    - A další
- Vstup: Každý vstup je prezentován jako jedna synapse. Vstupů může být do neuronu více než jeden.
- Výstup: Výstup je také jako vstup prezentován synapsí. Výstup může být v neuronu jenom jeden. Každý výstup je zpracován přenosovou funkcí. Její výsledná hodnota je synapsí poskytnutá dál do neuronové sítě. [3][4]

3.2. Umělá neuronová síť

Vzhledem k tomu, že se biologická neuronová síť nedá vytvořit, programátoři se od vzniků prvních počítačů snaží vytvořit umělou neuronovou síť. Ta má za úkol napodobovat fungování lidského mozku. Umělé neuronové sítě mají širokou škálu využití. Dají se využít například na rozpoznávání obrazů, předpovídání určitých událostí, chování určitých veličin, automatické parkování nebo překládání textu.

Neuron není sám o sobě schopen dělat moc složité funkce. Síla neuronové sítě roste až propojením neuronů mezi sebou ve větších strukturách. Neurony jsou uspořádány do vrstev, které jsou mezi sebou propojeny.[3][4]

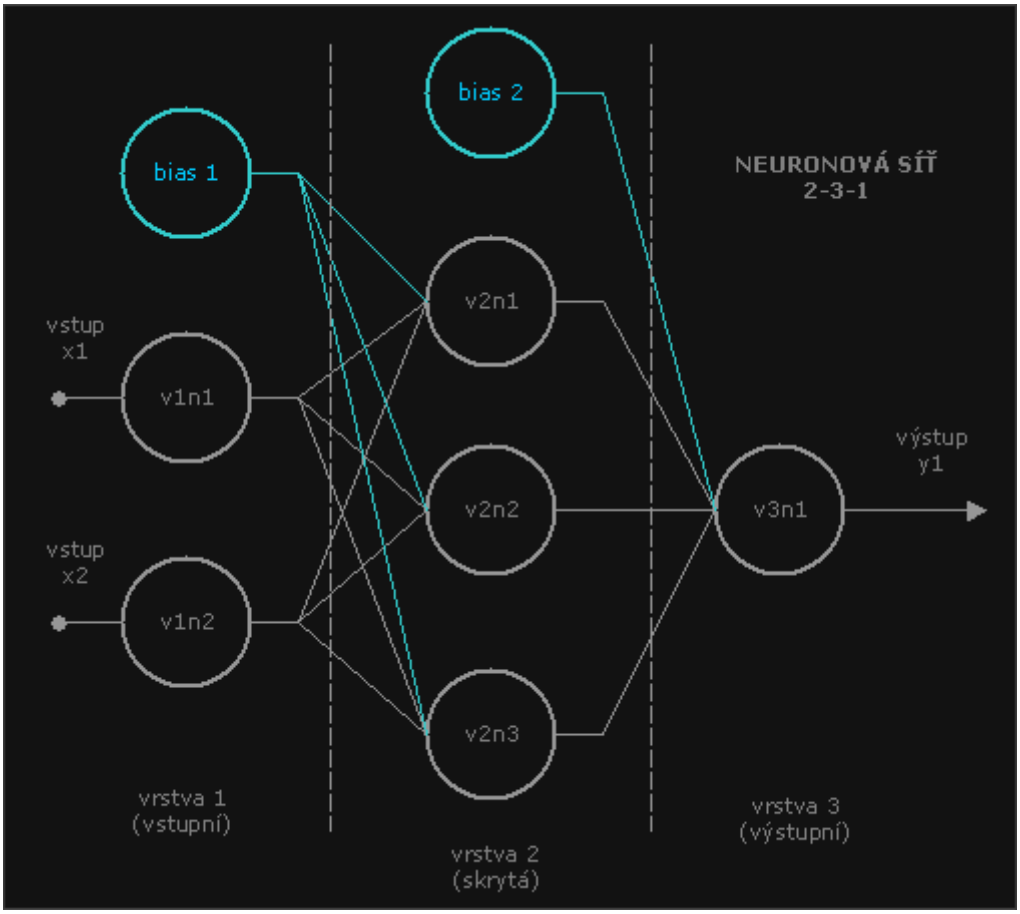
3.2.1. Skladba neuronové sítě a její fungování

Neuronová síť se jako celek rozděluje do třech částí:

- Vstupní vrstva: Vstupní vrstva je první vrstva v neuronové síti. Obsahuje vstupní synapse, přes které vstupují data určená ke zpracování do sítě.
- Skrytá vrstva: Jedná se minimálně jednu vrstvu, která se nachází mezi vstupní a výstupní vrstvou. Vstupy a výstupy skryté vrstvy jsou v neuronové síti skryté.
- Výstupní vrstva: Výstupní vrstva je poslední vrstva v neuronové síti. Její výstupy jsou poté zpracované jako konečné z celého modelu neuronové sítě.

Jednoduchá neuronová síť má vstupní vrstvu, kam vstup vstupuje. Má také minimálně jednu skrytou vrstvu. Poté je zde poslední vrstva, která se nazývá výstupní. Z té získáváme na výstupu výsledky. Poté se v každé vrstvě kromě vstupní objevuje jednotka Bias. Všechny tyto vrstvy a jejich neurony jsou propojeny synapsemi, které mají různé váhy, podle toho, jak moc jsou informace nesené v této synapsi důležité. Taková neuronová síť je znázorněna na obrázku číslo 3.

Způsob, jakým vnášíme inteligenci do této neuronové sítě, je přiřadit hodnoty všem těmto vahám. A to je to, co trénuje neuronová síť. Neuronová síť, přesněji řečeno její učící algoritmus, má za úkol nastavit správné hodnoty vahám synapsí za účelem dosažení požadovaných hodnot. Hodnoty vah synapsí v jedná vrstvě se dají vyjádřit jako pole. Z hlediska matematiky se vyjadřují jako matice.



Obrázek 3 - Jednoduchá neuronová síť 2-3-1 [3]

Vzhledem k tomu, že síť může obsahovat stovky skrytých vrstev a každá tisíce neuronů, tak je jasné, že právě díky složitosti spojení dokáže neuronová síť najít složitá řešení. Pozorovatel nebude nikdy schopen určit, proč něco u nějakého konkrétního příkladu dopadlo nebo proč se váhy nastavily tak, jak se nastavily. Pokud bude vytvořena síť a bude vyučena dvakrát stejnými trénovacími daty, výsledky se mohou lišit. Ze získané neuronové sítě nikdy nebudeme schopni získat interpretaci, proč to tak u konkrétního pozorování dopadlo. Jedná se tedy o metodu typu blackbox. Blackbox znamená, že nejsme schopni jednoduše vyjádřit a zjistit výsledky nebo závislosti mezi různými proměnnými. Tato vlastnost tedy nevadí, pokud se nechce určit, jakým způsobem se k výsledku došlo, ale je předmětem našeho zájmu jen výsledek. [3][4]

3.2.2. Učení umělé neuronové sítě

Před použitím neuronové sítě je potřeba provést učení. Při učení dojde ke změně parametrů neuronové sítě. Tyto paramtery jsou váhy jednotlivých vstupů a prahy

neuronů. Existují dva typy učení. Učení s učitelem a učení bez učitele. Každý styl učení má své výhody a nevýhody a také mohou být více použitelné v jiných typech úloh.

3.2.2.1. Učení s učitelem

Na začátku dostane umělá neuronová síť set trénovacích dat, na kterých jsou zobrazené správné odpovědi na danou problematiku. Trénovací set obsahuje nějaká vstupní data a dále obsahují také hodnoty požadovaného výstupu. Výstup může být spojitá hodnota (regrese) anebo může předpovídat označení třídy vstupního objektu (klasifikace).

Úloha algoritmu je předpovědět výstupní hodnotu funkce pro každý vstup. Algoritmus nejdříve zpracuje trénovací příklady a poté teprve předpoví výstupní hodnotu. Tu porovnáme s požadovaným výsledkem a určíme chybu. Poté spočítáme nutnou korekci a upravíme hodnoty vah či prahů, abychom snížili hodnotu chyby. Toto opakujeme až do dosažení minimální chyby. [5]

3.2.2.2. Učení bez učitele

Podobně jako u učení s učitelem dostaneme nějaký set trénovacích dat. Na tomto setu trénovacích dat není určeno, která data jsou „správná“. Je to tedy jenom set nějakých dat, o kterých nevíme, k čemu jsou a co s nimi dělat. Nemáme tedy žádný známý výstup. V tom případě se neuronová síť bude snažit najít nějakou strukturu v těchto datech. Bude se snažit rozdělit data do clusterů (clusterizace). Výstup budou nějaké skupiny dat, o kterých nebude neuronová síť vědět nic přesnějšího. Data pouze rozdělí do skupin, kde by si ta daná data měla být něčím podobná. [6]

3.2.3. Hrozba přeučení sítě

Pokud bude síť obsahovat malý počet neuronů, tak může být schopnost vystihnou a popsat závislosti v trénovacích datech slabá. To znamená, že neuronová síť nebude mít na výstupu správné výsledky. Na druhou stranu, pokud bude síť obsahovat velký počet neuronů, tak síť sice nebude mít problém najít závislosti na trénovacích datech, ale její schopnost práce na datech testovacích bude špatná. Síť se totiž naučí pracovat pouze s trénovacími daty, ale výsledek na nových datech bude horší nebo úplně špatný.

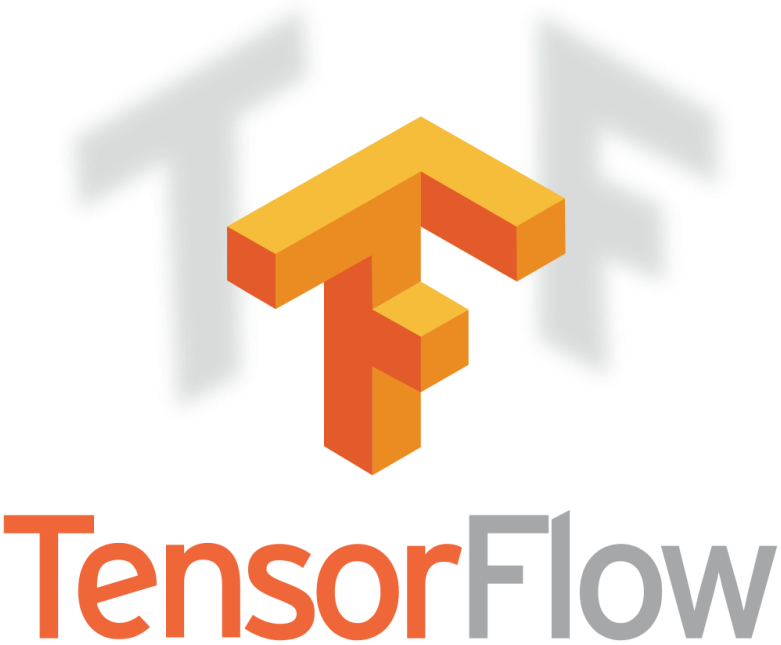
Tento jev se nazývá přeučení sítě, anglicky overfitting. K přeučení ale nemusí docházet jen kvůli špatnému rozložení sítě. K přeučení může docházet pokud:

- Síť obsahuje hodně vstupních parametrů a málo pozorování

- Síť při učení iteruje vícekrát, než by měla - velký počet epoch (opakování učení)
- Síť obsahuje velký počet neuronů
- Špatné rozložení dat (učící, validační, testovací)

4. ÚVOD TENSORFLOW

TensorFlow je otevřená softwarová knihovna pro strojové učení. TensorFlow byl vyvinut společností Google, aby splnil své potřeby systémů schopných vytvářet a vzdělávat neuronové sítě. Aby zjistily a rozluštily vzory a korelace, analogické učení a uvažování, které používají lidé. V současné době se používá v produktech Google. Často nahrazuje roli jeho closed-source předchůdce, DistBelief. TensorFlow byl původně vyvinut týmem Google Brain pro interní použití Google. Poté byl ale vydán 9. února 2015 pod licencí open source Apache 2.0.



Obrázek 4 - Ikona TensorFlow [7]

TensorFlow je druhá generace strojového učení. Architektura umožňuje pracovat na více CPU a GPU. Prostředí používá rozhraní v jazyce Python. Může používat volitelné doplňky CUDA pro výpočty na grafické kartě. Je k použití na platformách Windows, Linux, macOS. Dá se použít také na mobilních platformách Android a iOS.

Výpočty TensorFlow jsou vyjádřeny jako stavové grafy datových toků. Název TensorFlow vychází z vícerozměrných polí, takzvaných tenzorech, na kterých neuronové sítě provádí operace. Pro tyto výpočty si Google sám vyrobil svůj vlastní procesor TPU (Tensor Processing Unit). [7]

4.1. Fungování TensorFlow

Výpočty v TensorFlow jsou definovány jako graf. Tento graf je tvořen operacemi. Jedná se tedy o vytvoření jakési mapy, která definuje nějakou sérii operací a jak jdou po sobě. Tento graf se nazývá DataFlow a je rozdělen na tři části, stejně jako neuronová síť:

1. Vstupní
    - Data zde vstupují do grafu
2. Skrytá
    - Tato vrstva je zodpovědná za výpočty
3. Výstupní
    - Data zde vystupují z grafu

Tento graf je základní stavební strukturou. Graf v sobě obsahuje dvě základní výpočetní jednotky.

1. Uzel – Node
    - Představuje nějakou matematickou operaci, neuron. V TensorFlow známou jako aktivační funkci. Výběr této aktivační funkce má velký dopad na chování celé sítě.
2. Okraj – Edge
    - Představuje vícerozměrné pole. V TensorFlow známé jako tenzor.

TensorFlow pracuje v jazyce Python. Uzly a Okraje jsou objekty Pythonu. Matematické operace se však neprovádí v Pythonu. Výpočty TensorFlow se provádí v C++ a Python jen směřuje, kam mají výpočty pokračovat a co s nimi dělat. Python v tomto případě funguje jako takové pojítko. [8]

4.2 Výhody TensorFlow

Největší výhodou, kterou TensorFlow poskytuje je abstrakce. Místo toho, aby vývojář musel implementovat různé algoritmy a další věci, se může rovnou zaměřit na celkovou logiku aplikace. TensorFlow se stará o detaily v pozadí a vývojář používá již hotové objekty/metody.

Další výhodou je TensorBoard. Je to vizualizační nástroj, který zobrazuje samotný graf TensorFlow. TensorBoard tedy ukazuje, jak síť vypadá a kudy tyto data prochází. Jak se data, která grafem prochází mění a vyvíjí. Výhodou samotného TensorBoard je že se může jeho výstup prohlížet v běžném internetovém prohlížeči. [7] [9]

4.3 Instalace TensorFlow

K potřebám tohoto projektu byl použit systém Windows. Systém Windows by měl být Windows 7 nebo vyšší. V dalším kroku je potřeba nainstalovat Python. Python by měl být vyšší než verze 3.4. Python ve verzi vyšší než 3.4 přichází s pip3 package manager, který je nezbytný k instalaci TensorFlow na Windows.

Obrázek 5 - Ikona Python

Po stažení a instalaci je dobré zkontrolovat, jestli byla opravdu nainstalována správná verze Pythonu, a to pomocí příkazu: python --version, který se zadá do příkazové řádky nebo do Windows PowerShell.

Dalším krokem je instalování samotného TensorFlow díky příkazu: pip install tensorflow. Tento příkaz bude opět zadáván buď do příkazové řádky nebo do Windows PowerShell a povede ke stažení poslední stabilní verze TensorFlow pro práci s procesorem. [10]

Zde jsou uvedeny další důležité příkazy k instalaci TensorFlow na systém Windows:

- pip install --upgrade tensorflow
    - slouží k aktualizaci stabilní verze TensorFlow pro práci s procesorem (CPU)
- pip install tensorflow-gpu
    - slouží ke stažení poslední stabilní verze TensorFlow pro práci s grafickou kartou (GPU)
- pip install --upgrade tensorflow-gpu
    - slouží k aktualizaci stabilní verze TensorFlow pro práci s grafickou kartou (GPU)

4.4. Využití TensorFlow

TensorFlow využívá spousta velkých firem kvůli různým účelům. Airbnb používá TensorFlow za účelem klasifikace obrázků a zlepšení zákazníkovy zkušenosti. PayPal je schopen díky strojovému učení od Google rozpoznat podvodníky a jejich plány. Twitter je schopen díky TensorFlow zařídit, aby uživateli neunikl důležitý tweet, pokud sleduje stovky lidí. V poslední řadě je využívá sám Google v jeho vlastních produktech jako je Gmail, vyhledávání na Google nebo překladač. TensorFlow nabízí spoustu možností, jak jej využít. Také jej může využívat kdokoliv. [11]

4.4.1. Text-Based Aplikace

TensorFlow se používá nejvíce v text-based aplikacích. To jsou aplikace, které jsou založeny na textu a jejich analýze. Tato analýza se dá použít na detekci hrozeb například na sociálních médiích nebo na detekci podvodů u pojištění nebo jiných finančních záležitostí.

Největší využití ale najdeme v detekci jazyka a jeho následného překladu. Google překladač používá TensorFlow k překladu více než 100 jazyků. Dokáže převádět vybraný text z jednoho jazyka do druhého. Nová verze překladače dokáže v některých případech rozpoznat sarkasmus nebo nějaký řečnický obrat, který se poté snaží nepřeložit doslovně, ale tak, jak by ho přeložil člověk.

Společnost Google také zjistila, že u kratších textů může text sumarizovat podle techniky nazývané sekvenční učení. To se dá například využít k vytváření nadpisů pro novinové články a podobně. [12] [13]

4.4.2. Rozpoznávání hlasu/zvuku

Další známé použití TensorFlow jsou aplikace založené na zvuku. Neuronové sítě jsou schopny rozluštit zvukové signály, pokud jim je zvuk podáván tak, jak potřebují. Využití:

- Rozpoznání hlasu
    - Používá se v bezpečnosti. Používá se při vstupu do budovy.
- Vyhledávání hlasem
    - Nejčastěji používáno v telekomunikaci. Google vyhledávání hlasem používá například při používání vyhledávačem.
- Detekce vad
    - Nejčastěji se používá v automobilovém a leteckém průmyslu.
- Hlasem aktivovaní pomocníci
    - Jsou široce rozšíření ve smartphonech a počítačích. Těmto pomocníkům se můžou nadiktovat hlasem akce, který poté mobil provede. Ať jde o psaní SMS zpráv, emailů nebo přidání upozornění do kalendáře. Nejznámějšími pomocníky jsou Siri od Apple, Google Now pro Android a Cortana pro Windows a Windows Phone.
- Aplikace řeč-text
    - Tyto aplikace nejdřív zanalyzují zvukový soubor, u kterého poznají, o který jazyk se jedná a potom ho přepisují z mluveného slova do textu. Tyto aplikace mohou být použity k určení pouhých úryvků zvuku nebo také ve větších zvukových souborech. Mohou například vytvářet titulky k filmům.

Rozpoznávání zvuku může být taky použito v Customer Relationship Management systémech. Kde mohou algoritmy za použití TensorFlow odkazovat zákazníky po sdělení jejich problému k potřebným informacím rychleji než lidé. [12] [13]

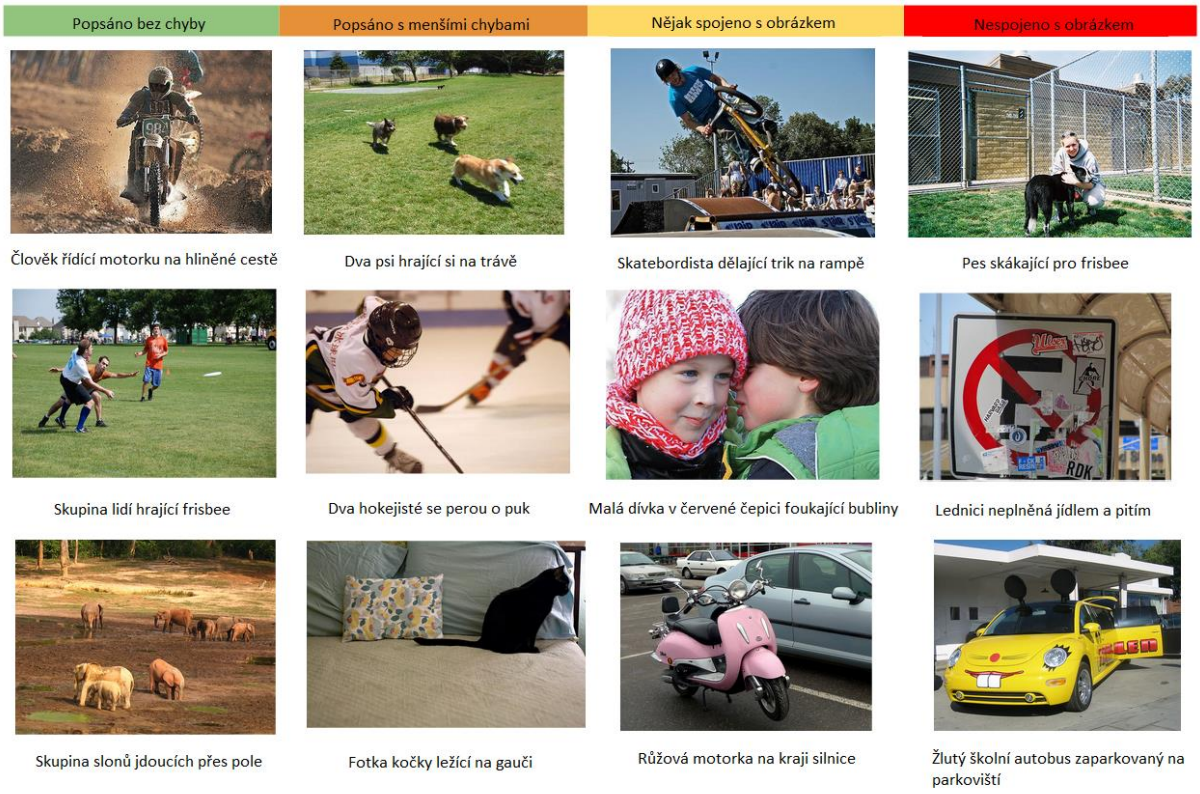
4.4.3. Rozpoznávání obrazu

Rozpoznávání obrazu je zaměřeno nejen na rozpoznání lidí a objektů v obrazech a jejich identifikace, ale také na pochopení jejich kontextu. To znamená, co ten objekt dělá nebo proč tam je. Nejčastěji se využívá na sociálních sítích nebo bezpečnostních zařízeních.

Algoritmy TensorFlow identifikují a zařazují objekty v rámci větších obrázků. Toto se velmi často používá ve strojírenských aplikacích k rozpoznání tvarů za účelem modelování (3D prostorová konstrukce z 2D obrázků). Dalším příkladem použití je také označování lidí na fotografiích na sociálních sítích (Facebook Deep Face).

Rozpoznávání obrazu se začíná rozšiřovat i v oblasti zdravotnictví, kde algoritmy TensorFlow mohou zpracovávat více informací a zaznamenávat více vzorků. Počítače jsou nyní schopny prověřit skeny a zaznamenat více nemocí než lidé.

Níže na obrázku číslo 6 je znázorněno, jak program od Google dokáže rozeznávat určité obrázky. V některých případech je vidět, že program nedokáže jednoznačně určit, o jaký obrázek se jedná. Je to proto, že stroj si nedokáže obrázek rozdělit na menší části a oddělit je od sebe. Prostě si prohlédne obrázek pixel po pixelu a poté hledá podobnou shodu u nějakých obrázků, které již předtím určil nebo je vyhledává v již správě určených obrázcích. [12] [13]



Obrázek 6 - Rozpoznávání obrázků od společnosti Google [12]

4.4.4. Časové řady

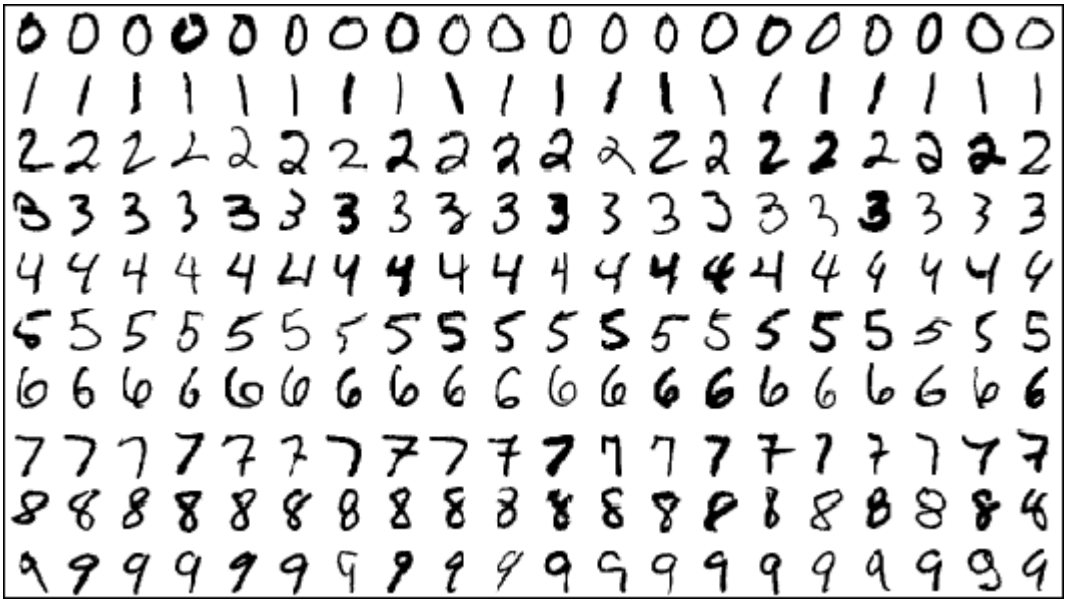
TensorFlow dokáže analyzovat časové řady s cílem získat smysluplné statistiky. TensorFlow dokáže na základě získaných informací předpovídat nějaká časová období. Například, jak se budou vyvíjet akcie, jak se bude měnit kurz měn a podobně.

Nejběžnějším použitím je takzvané Doporučení, které používají společnosti jako Facebook, Google Amazon a mnoho dalších. Jde o to, že tyto společnosti sledují a analyzují naše činnosti a porovnávají je s dalšími miliony jiných zákazníků, aby zjistily, co by si mohl chtít zákazník koupit, přečíst nebo sledovat v televizi.

Další využití Časových řad je především v oblasti financí, účetnictví, bezpečnosti nebo detekci rizik. [12] [13]

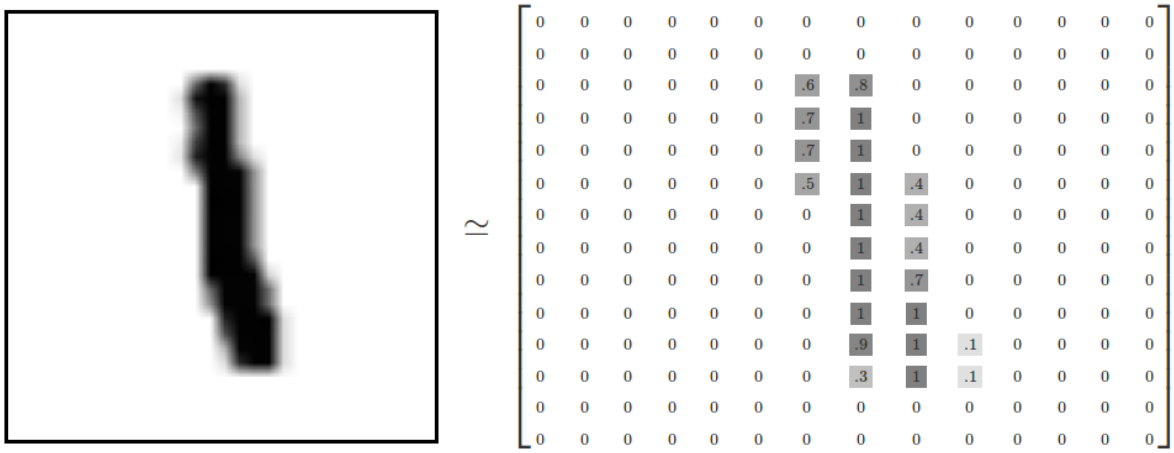
4.4.5. Rozpoznávání ručně psaných číslic

Jde o velmi základní projekt, kde se utvoří velmi jednoduchá neuronová síť, která má za úkol rozpoznávat ručně psané číslice. K tomu je použitá databáze MNIST, která obsahuje 70 000 ručně psaných číslic o velikosti 28x28 pixelu. Číslice mohou vypadat tak jako na obrázku číslo 7.



Obrázek 7 - Ukázka číslic z databáze MNIST [14]

Aplikace si rozdělí data na 55 000 trénovacích dat, 10 000 testovacích dat a 5 000 validačních dat. Rozdělení dat na tyto tři skupiny je velmi důležité. Pokud by například bylo málo testovacích dat, úspěšnost by mohla být příliš malá. Dalším krokem bude, že si data, na kterých trénuje, rozdělí do dvourozměrného pole 28x28, které obsahuje čísla 0 až 1. Obrázek může být reprezentován podobně jako číslo 1 na obrázku číslo 8. Číslice 0 je pro prázdné místo pixelu a číslice 1 je pro černé místo pixelu.



Obrázek 8 - Číslo jedna vyjádřené na dvourozměrném poli [15]

U těchto trénovacích dat je i označení, o jaké číslo se jedná a poté porovnávají s validačními daty. Po tréninku na těchto datech, si neuronová síť začne na vstup vybírat náhodné číslice z testovacích dat a snaží se říct, o které číslo se jedná. Neodpoví

jednoznačně, ale v procentech. Ve výsledku je schopno říct o jaké číslo se jedná s určitostí od 90 procent do 92 procent. Zbylá procenta patří číslům, která se budou mít podobné rozložení číslic na dvourozměrném poli. [14] [15]

4.4.6. Fast Style Transfer

Projekt Fast Style Transfer je velmi složitý a propracovaný projekt na GitHubu TensorFlow, který dokáže přenést styly různých malířů na obrázky. TensorFlow se tedy naučí používat techniky slavných malířů a aplikovat je na různé fotografie, viz obrázky číslo 9 a 10. Strojové učení si tedy prohlédlo malířův obraz pixel po pixelu a vytvořilo si nějaký model, který se pak aplikoval na fotografii. [16]

Obrázek 9 - Edvard Munch, Výkřík, přeneseno na fotografii [16]

Obrázek 10 - Leonid Afremov, Rain Princess, přenesno na fotografii [16]

5. NADSTAVBA KERAS

Hlavním autorem je inženýr François Chollet ze společnosti Google. Keras je open-source API napsané v Pythonu. Keras je schopný fungovat nad TensorFlow, CNTK nebo Theano. Používá se k experimentování s modely neuronových sítí. Keras není přímo knihovnou, jedná se spíš o nadstavbu/interface k přehlednějšímu, rychlejšímu a intuitivnějšímu návrhu neuronové sítě.

Obrázek 11 - Ikona Keras [18]

Google se rozhodl Keras podpořit v knihovně TensorFlow. Keras od té doby slouží jako interface. Keras jako interface přináší tyto výhody: [17] [18]

- Je uživatelsky příjemný. Představuje jednoduchý a přehledný interface. Poskytuje jednoduše pochopitelnou zpětnou vazbu. Ať už se jedná o chybové hlášky nebo vysvětlení různých metod a tříd.
- Skládání modelů je velmi jednoduché, protože samotný model se vytváří v blocích.
- Je velmi modulární. Uživatel může tvořit nové vrstvy modelu, loss funkce, optimalizátory a další.

5.1. Instalace

Vzhledem k tomu, že TensorFlow podporuje Keras ve své knihovně již od roku 2017, není potřeba po instalaci TensorFlow instalovat Keras zvlášť.

5.2. Modely

V Kerasu se dá pracovat s dvěma typy modelů. Tyto modely se nazývají Sekvenční a Model class API. Tyto modely obsahují tyto společné metody a atributy:

- Model.layers – list vrstev které model obsahuje

- Model.inputs – list vstupních tensorů modelu
- Model.outputs – list výstupních tensorů modelu
- Model.summary() – vypsání reprezentace podoby modelu
- Model.get_config() – vrátí slovník obsahující nastavení modelu. Toto nastavení se poté dá načíst.
- Model.get_weights() – vrátí list matic NumPy se všemi vahami v modelu
- Model.set_weights() – nastaví váhy modelu z listu matic NumPy. Pole by měla být stejného rozměru jako které dostaneme pomocí metody get_weights().
- Model.to_json() – vrátí reprezentaci podoby modelu jako JSON řetězec. Neobsahuje váhy.
- Model.to_yaml() – vrátí reprezentaci podoby modelu jako YAML řetězec. Nebosahuje váhy.
- Model.save_weights() – uloží váhy modelu jako HDF5 soubor
- Model.load_weights() – načte váhy modelu z HDF5 souboru. Pokud se načítají váhy z jiné reprezentace architektury modelu, ale některé vrstvy se podobají, použije se parametr metody by_name=True. Tento parametr zajistí načtení vah pro vrstvy, které jsou pojmenované stejně jak v modelu, do kterého chceme váhy načíst, tak v modelu, ze kterého tyto váhy načítáme. [19]

5.2.1. Sekvenční model

Sekvenční model je výborný pro většinu modelů strojového učení, přesto má však nějaká omezení. Omezení jsou:

- Pokud má mít model více vstupních zdrojů
- Pokud má mít model více výstupních destinací
- Model neumí znovu používat nějaké vrstvy

Sekvenční model je lineární soubor vrstev. Při vytváření Sekvenčního modelu je možno vrstvy přidat tak, že se do konstruktoru přidá list vrstev, nebo se přidají pomocí metody add(): [20]

Přidání vrstev pomocí přidání listu do kontruktoru:

```
1. import keras as keras
2. model = keras.models.Sequential([
3.     keras.layers.Dense(32, input_shape=(784,)),
4.     keras.activations.relu,
5.     keras.layers.Dense(10),
6.     keras.activation.relu.softmax,
7. ])
```

Ukázka kódu 1 - Sekvenční model, přidání vrstev pomocí listu, převzato z [20]

Přidání vrstev pomocí metody add():

```
1. import keras as keras
2. model = keras.models.Sequential()
3. model.add(keras.layers.Dense(86, activation='relu', input_shape=[86],
   name="vstupni_vsrtva"))
4. model.add(keras.layers.Dense(172, activation='relu',
   name="skryta_vsrtva1"))
```

Ukázka kódu 2 - Sekvenční model, přidání vrstev metodou add(), převzato z [20]

5.2.1.1. Vstup do modelu

Na začátku tvorby modelu musí být u první vrstvy nastaven tvar dat, které bude model očekávat. Proto se u sekvenčního modelu nastavuje v první vrstvě, která se nazývá vrstva vstupní, nastavuje vstupní tvar. Tento vstupní tvar se dá nastavit několika způsoby:

- Přidání argumentu input_shape do první vrstvy, čímž se definuje, jak tvar těchto vstupních dat bude vypadat. V podstatě se jedná o definici tvaru matice. Tento argument je nejčastěji používaný.

```
model.add(Dense(32, input_shape=[784,]))
```

- Některé vrstvy, které jsou 2D, podporují argument input_dim k definici tvaru vstupních dat. Některé 3D vrstvy podporují argumenty input_dim a input_length.

```
model.add(Dense(32,input_dim=784))
```

Tyto dva zápisy výše jsou naprosto totožné. U prvního zápisu je pouze použit argument input_shape a druhého input_dim. Oba zápisy definují jako vstup jeden rozměr a velikosti 784. [20]

5.2.1.2. Kompilace

Kompilací se nazývá proces, při kterém se nastavuje proces učení. Tento proces se vyvolá jednoduchou metodou compile(). Tato metoda by mohla vypadat nějak takto:

```
1. model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
   metrics=['accuracy'])
```

Tato metoda obsahuje tři parametry, které definují proces učení: [20]

- Optimizer – požadovaný parametr. Výběr optimalizátoru. Je možno využít již existujících optimalizátorů nebo vytvořit vlastní.
- Výběr loss funkce – požadovaný parametr. Výběr loss funkce. Pomocí loss funkce se vybírá cíl, který se model bude snažit minimalizovat.
- Metrics – nepožadovaný parametr. List metrik. Pokud se bude model zabývat klasifikací, bude parametr metrik nastaven na metrics=[‚accuracy‘]. Dále může být metrika vybrána jakákoliv již existující metrika nebo je možné se metriku vytvořit.

5.2.1.3. Trénink

Model vytvořený pomocí Keras je trénován na trénovacích datech, která jsou matice NumPy. Trénink modelu se zavolá pomocí metody fit(). Tato metoda by mohla vypadat nějak takto:

```
1. model.fit(data, labels, epochs=10, batch_size=32)
```

```
1.  model.fit(data, labels, batch_size=None, epochs=10, verbose=1,
   callbacks=None, validation_split=0.0, validation_data=None,
   shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0,
   steps_per_epoch=None, validation_steps=None, validation_freq=1)
```

Jak je v kódu výše ukázáno, metoda fit() obsahuje mnoho parametrů, kterými se dá proces učení upravit podle svých představ. Je také důležité vzít v potaz, že některé problémy a složitější učení může chtít si tyto parametry upravit podle svých potřeb k dosažení co nejlepších výsledků. Parametry, které jsou vždy požadované jsou tyto: [20]

- X – Matice NumPy s daty určenými k trénování. Může to být i list matic NumPy.
- Y – Matice NumPy s cílovými (label) daty. Může to být i list matic NumPy.
- Epochs – Je to celé číslo, které označuje počet opakování tréninku na poskytnutých datech X a Y.

5.2.2. Model class API

Na rozdíl od Sekvenčního Modelu, se zde musí vytvořit samotná vstupní vrstva třídy Input a definovat její vstup.

```
input = Input(shape=(2,))
```

Poté se všechny vrstvy spojují po dvojicích. Takže se u druhé vrstvy definuje vrstva, které je vstupem do této vrstvy.

```
output = Dense(2)(input)
```

Po vytvoření všech vrstev modelu a jejich propojení je potřeba definovat model jako celek. K tomu slouží třída Model, kde se jen specifikuje, která vrstva modelu je vstupní a která vrstva modelu bude výstupní.

```
model = Model(inputs=input, outputs=output)
```

V případě více vstupů nebo více výstupů, zadávají se argumenty třídy Model jako list vrstev. [21]

```
model = Model(inputs=[input1, input2], outputs=[output1, output2, output3])
```

5.2.3. Vlastní model – Model subclassing

Mimo těch dvou modelů se dá v Keras od verze 2.2.0 také vytvořit plně přizpůsobitelný vlastní model pomocí třídy Model a implementací vlastní metody call.

Zde je uveden příklad, jak by taková podtřída mohla vypadat: [19]

```
import keras

class SimpleMLP(keras.Model):

    def __init__(self, use_bn=False, use_dp=False, num_classes=10):
        super(SimpleMLP, self).__init__(name='mlp')
        self.use_bn = use_bn
        self.use_dp = use_dp
        self.num_classes = num_classes

        self.dense1 = keras.layers.Dense(32, activation='relu')
        self.dense2 = keras.layers.Dense(num_classes, activation='softmax')
        if self.use_dp:
            self.dp = keras.layers.Dropout(0.5)
        if self.use_bn:
            self.bn = keras.layers.BatchNormalization(axis=-1)
```

```
    def call(self, inputs):
        x = self.dense1(inputs)
        if self.use_dp:
            x = self.dp(x)
        if self.use_bn:
            x = self.bn(x)
        return self.dense2(x)

model = SimpleMLP()
model.compile(...)
model.fit(...)
```

Ukázka kódu 3 - Model subclassing, převzato z [19]

5.3. Optimalizátory

Pokaždé když v neuronové síti proběhne jeden cyklus učení, síť vygeneruje data, která predikuje a porovná je se správnými daty. Tyto data mezi sebou mají vždy nějaký rozdíl a neuronová síť musí s tímto rozdílem nějak naložit a podle něj upravit váhy v neuronové síti tak, aby k vedli ke zlepšení predikce a tím pádem k menšímu rozdílu mezi předpovězenými daty a daty správnými. O toto zlepšení se v Keras starají optimalizační algoritmy – optimalizátory.

Keras podporuje spoustu algoritmů jako je například SGD, RMSProp, Adam, Nadam a další. Optimalizátor se dá do modelu předat jako textový řetězec, poté použije výchozí parametry optimalizátoru, nebo jako objekt, kde dají parametry optimalizátoru nastavit.

V současné době se používají hlavně dva optimalizátory. A to Adam a jeho vylepšení Nadam. Tyto optimalizátory se používají, protože jsou nejlepší, tedy nejrychlejší. [22] [23]

5.4. Ztrátová funkce

Ztrátová funkce je metoda, která hodnotí, jak moc úspěšný je model neuronové sítě při své práci a poté se používá k trénování neuronové sítě. Ztrátová funkce vypočítává rozdíl mezí predikovanými daty a daty správnými.

Ztrátová funkce se volí podle toho, jestli neuronová síť řeší problémy regrese nebo klasifikace. Optimalizátor se snaží hodnotu ztrátové funkce minimalizovat na minimum. Ideální hodnota je 0, což by znamenalo, že neuronová síť funguje naprosto přesně. Nikdy se nemůže stát, že by hodnota klesla pod 0. Díky hodnotě ztrátové funkce je možné poznat, zda úprava neuronové sítě vedla ke zlepšení, tedy má nižší hodnotu ztrátové funkce, nebo ke zhoršení, tedy má hodnotu ztrátové funkce vyšší. [32][33][24]

5.5. Metriky

Metrika je funkce k posouzení výkonosti modelu neuronové sítě. Metrická funkce je podobná ztrátové funkci. Jediný rozdíl je, že výsledky metriky nejsou použity při tréninku modelu. Ztrátová funkce může být použita jako funkce metrická.

Metrika se nastavuje v metodě compile(). Metriky jsou zaznamenávány na konci každé epochy v tréninkové sadě. Pokud jsou poskytnuta validační data, tak se metrika vypočítává také pro ně. [24]

Metriky pro regresi jsou:

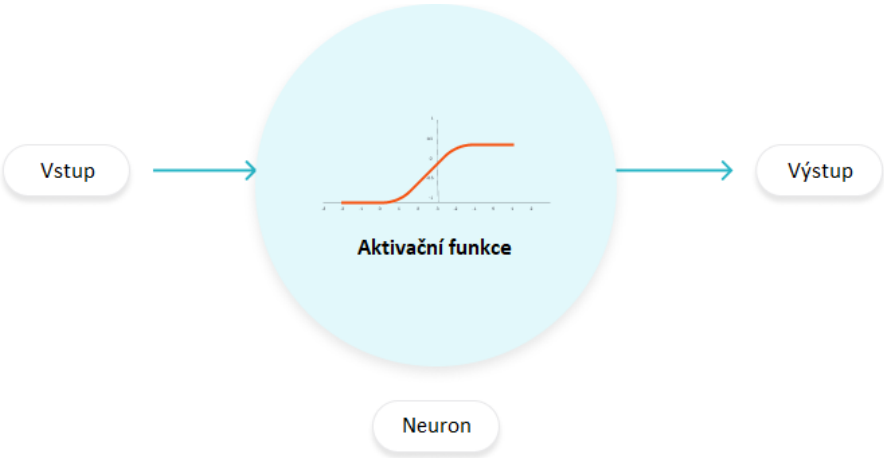
- Mean Squared Error
- Mean Absolute Error
- Mean Absolute Percentage Error
- Cosine Proximity

Metriky pro klasifikaci jsou:

- Binary Accuracy
- Categorical Accuracy
- Sparse Categorical Accuracy
- Top k Categorical Accuracy
- Sparse Top k Categorical Accuracy

5.6. Aktivační funkce

Aktivační funkce v neuronových sítích určuje, co udělá neuron s daty, které obdrží na svém vstupu. Na tyto data co do něj vstoupí použije aktivační funkci a pošle je na svém výstupu dál do neuronové sítě.



Obrázek 12 - Aktivační funkce, převzato z [25]

Aktivační funkce je matematická rovnice a stará se o to, jak moc daný neuron bude na základě vstupních dat aktivní. Tyto funkce jsou připojeny ke každému neuronu v síti a určují, zda má být funkce aktivována(vypálena) nebo ne. Ve většině případů by se to tedy dalo rozdělit, že aktivita neuronu se pohybuje mezi 1 až 0. Kde 1 znamená, že neuron je zcela aktivní a 0, že neuron je zcela neaktivní. Někdy se toto rozmezí mění.

V dnešní době jsou používány hlavně nelineární aktivační funkce, které pomáhají neuronovým sítím se naučit komplexnější data. Z těchto nelineárních aktivačních funkcí se používá převážně Relu nebo Elu, pokud se jedná o aktivační funkci na skryté vrstvě. Pokud se jedná o výstupní vrstvu neuronové sítě, která se zabývá regresí, tak by se měla použít Lineární aktivační funkce. Pokud je cílem neuronové sítě klasifikace, měla by se na výstupní vrstvě použít aktivační funkce Softmax. [25] [26]

Keras používá tyto aktivační funkce:

- Softmax
- Elu
- Selu
- Softplus
- Softsign
- Relu
- Tanh

- Sigmoid
- Hard_sigmoid
- Exponential
- Linear

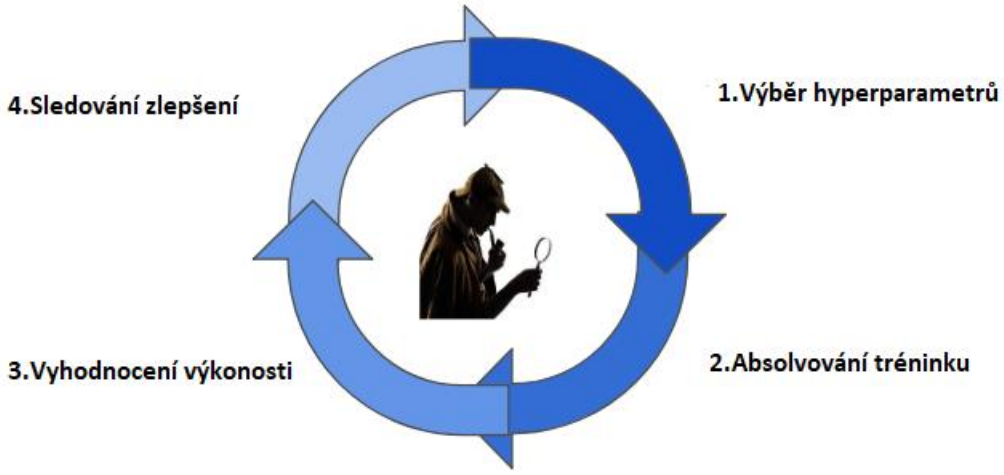
5.7. Výběr hyperparametrů sítě

Model neuronové sítě obsahuje spousty parametrů, které se dají měnit a ovlivňují fungování sítě. Ne všechny tyto parametry přispívají stejný způsobem k procesu učení modelu neuronové sítě. Tyto parametry, které se v síti dají nastavit jsou nazývány hyperparametry.

Mezi tyto hyperparametry tedy spadá learning rate, dropout, počet epoch a podobné proměnné. Ale patří sem zde také komponenty modelu jako je počet vrstev, aktivační funkce vrstvy a podobně. Pokud se tedy jedná o nějakou komplexnější neuronovou síť, je jasné, že hledání těchto parametrů nemusí být úplně triviální. Hledání všech kombinací těchto parametrů může trvat velmi dlouho.

Proces hledání těchto parametrů probíhá ve čtyřech krocích. Prvním krokem je nastavením co nejlepší konfigurace. Druhý krok je trénink podle této konfigurace. Dalším krokem je vyhodnocení úspěšnosti neuronové sítě. V poslední části se podle strategie hledání hyperparametrů vybere nová konfigurace sítě a celý tento proces probíhá znovu.

Hledání může být ukončeno, pokud bude nalezeno optimální řešení, po určité době nebo po utracení určitých prostředků, kterými může být například počet cyklů hledání a podobně.



Obrázek 13 - Průběh hledání hyperparametrů, převzato z [27]

Při hledání hyperparametrů se uplatňují čtyři strategie, které budou níže představeny. [27] [28]

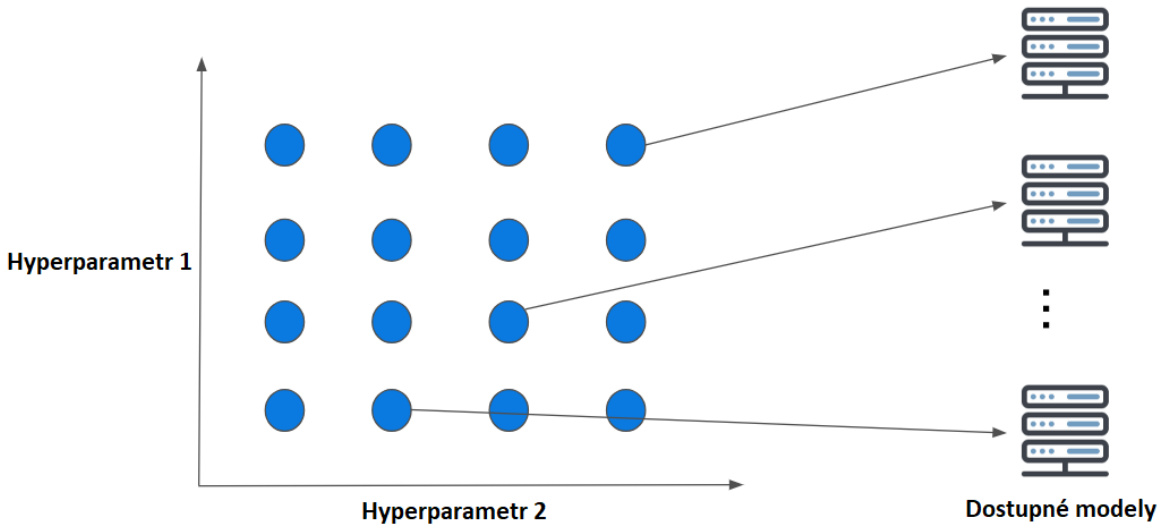
5.7.1. Babysitting

Tento přístup je 100 % manuální. Jde tedy o to, že se všechny různé kombinace konfigurace neuronové sítě vyzkouší manuálně a úspěšnost a výsledky sítě se evidují. Poté, co se všechny kombinace konfigurace sítě vyzkouší, se vybere ta nejúspěšnější konfigurace a je použita.

Pokud se tento přístup aplikuje na nějakou malou neuronovou síť, není problém tento postup použít. Při větší neuronové síti je tento postup velmi nepraktický a zdlouhavý. Jeho výhoda je, že se dají evidovat podrobné výsledky sítě. [27] [28]

5.7.2. Grid Search

Tento přístup vyzkouší každou možnou konfiguraci neuronové sítě a poté vrátí tu nejlepší možnou konfiguraci. Tedy pro každý parametr, pro který má být nalezena vhodná hodnota, je vytvořena nějaká dimenze hodnot, které se budou zkoušet.



Obrázek 14 - Grid Search hledání dvou hyperparametrů, převzato z [27]

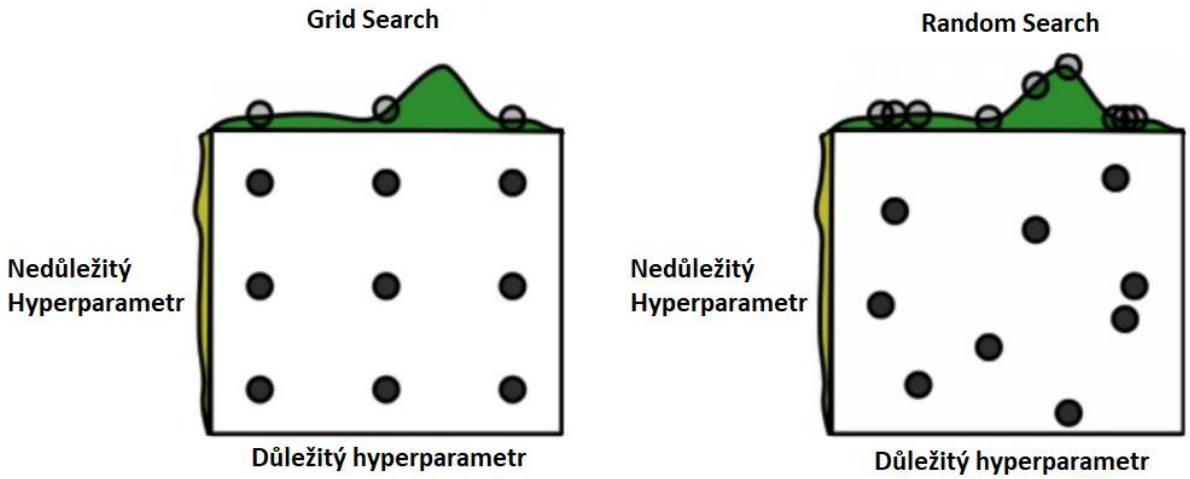
Obrázek číslo 14 ukazuje jednoduchou mřížku, hledání ve dvou dimenzích. Tedy pro dva hyperparametry sítě.

Největším problémem tohoto přístupu je jeho dimezionalita. To znamená, že čím víc parametrů sítě se snažíme v tomto přístupu nastavit, tím větší je pak dimenze těchto hodnot, ze kterých můžeme vybírat. Tím pádem se zvětšuje i počet kombinací konfigurací sítě. Čas hledání těchto konfigurací roste exponenciálně s každým dalším přidáním dimenze. Může se tedy stát, že vyhledání optimální konfigurace zabere velmi mnoho času.

V praxi se tedy tento přístup používá, pokud je dimenze hledaných parametrů menší nebo rovna čtyřem. I přesto, že na konci této strategie je nalezena nejlepší konfigurace této sítě, přesto se více používá Random Search strategie. [27] [28]

5.7.3. Random Search

Tato strategie se liší pouze v prvním bodu hledání konfigurace. Pro daný parametr není vybraná hodnota z dimenze hodnot, ale vybere se rozmezí, ze kterého se poté hodnota parametru náhodně vybírá.



Obrázek 15 - Grid Search vs Random Search, převzato z [27]

Jak je možné si všimnout na obrázku číslo 15, tak pokud je použita strategie Grid Search, tak pro každý parametr byly použity tři stejné proměnné. To se při strategii náhodného výběru s velkou pravděpodobností nestane, protože jsou proměnné pro daný parametr vybírány náhodně. Výhodou této strategie je, že nám pomůže najít nejlepší konfiguraci sítě v méně iteracích než Grid Search. Nemusí to být sice jasně nejlepší konfigurace, ale může se jí alespoň přibližovat. Je to proto, že Grid Search se pohybuje na mřížce, kdežto strategie Random Search může vybrat hodnoty i mimo tuto mřížku. [27] [28]

5.7.4. Bayesovská optimalizace

Problém Grid Search a Random Search je to, že každý nový odhad konfigurace sítě je nezávislý na tom předchozím. Tedy si nevezme nic z předchozí konfigurace a nová konfigurace se neupravuje na základě té předchozí. Což je například při strategii Babysitting základní a jediná pozitivní vlastnost. Protože kdokoliv upravuje konfiguraci sítě, tak vychází z její předchozí podoby a používá její předchozí podobu k vylepšení.

Tento problém řeší Bayesovská optimalizace. Tato strategie vytváří model, který se snaží předpovídat hodnoty prametrů, které se mají nastavit. Při každé iteraci konfigurace si bude model čím dál více jistější, jak by konfigurovatelné atributy měly vypadat.

Tento proces, co se rychlosti týče, je velmi rychlý. Zároveň je možno si uložit a poté prohlédnout historii všech konfigurací modelu neuronové sítě, které proběhly. [27] [28]

6. NÁVRH A IMPLEMENTACE APLIKACE PRO LOKALIZACI

Aplikace pro indoor lokalizaci zajišťuje lokalizaci uvnitř budov, kde není například možno použít GPS (Globální Polohový Systém). Může také sloužit k navigaci v budově, kterou někdo nezná a hledá například určitou kancelář. V případě indoor lokalizace se nedá použít, jako v případě GPS (Globální Polohový Systém), signál přijímaný ze satelitů, který pomůže lokalizovat, kde se kdo nachází. Ať už z důvodu, že signál mnohdy neprojde skrze stěny budovy, tak z důvodu, že by neukazoval pozici tak přesně, aby se dalo například poznat, u jakých dveřích v budově osoba stojí. Proto se v indoor lokalizaci musí používat takzvané radiové fingerprinty, díky čemuž se dá rozpoznat bezdrátové zařízení. V tomto návrhu budou pro aplikaci důležité síly signálů od těchto fingerprintů. Díky těmto silám se pak bude moci lokalizovat poloha uvnitř budovy. Kde síla signálu jednoho zařízení bude reprezentovat, jak daleko se v daném rádiusu od daného bezdrátového zařízení uživatel vyskytuje. Pokud tedy bude aplikace pracovat s více takovýmito zařízeními a jejich signály, tak bude dát alespoň přibližně lokalizovat poloha určité osoby.[34] konec

Bude se jednat o model, který se bude snažit určit pozici uživatele na mapce školy Univerzity Hradec Králové, budova J na třetím podlaží. Jak již bylo řečeno, model bude určovat pozici na základě přijatých sil signálů z různých vysílačů. Tvorba modelu se dá rozdělit na dvě části. První část bude vytvoření dat pro model, které potom bude používat. Takzvaně dataset. Druhá část je vytvoření samotného modelu, který data bude používat a na základě nich se učit.

6.1. Data set

Pro danou problematiku bylo za potřebí vytvořit vlastní dataset za pomoci kterého bude možno model vytrénovat. Data set byl vytvořen rozparsováním souboru couchdump.json. Tento soubor obsahoval záznamy o měřeních. Vzhledem k tomu, že model má zatím fungovat jen pro třetí podlaží a data byla naměřena 26.2. 2016, měli jsme vymezené, která data ze souboru budeme vytahovat. Z dat, která byla naměřena v tento den a tohoto patra, jsme potřebovaly dostat tři množiny dat, které jsou pro model podstatné. První množina

je naměřená síla signálů z BleScanů. Další množina je na je naměřená síla signálů z WifiScanů. Poslední množina obsahovala souřadnice X a Y, které znázorňují pozici na mapce třetího podlaží. K tomu, abychom tato požadovaná data dostali, bylo použito vývojové prostředí Microsoft Visual Studio od Microsoftu. Ve Visual Studiu byl použit NuGet Packet Manager. NuGet je správce balíčků pro vývojovou platformu společnosti Microsoft včetně .NET. Klientské nástroje NuGet poskytují možnost produkovat a konzumovat balíčky. Přes NuGet byl stažen a použit balíček Newtonsoft.Json, který slouží k serializování a deserializování dat ze souborů .json. Pracuje na způsobu reflexe.

6.1.1. Deserializace dat

Pro deserializaci souboru couchdump.json byla vytvořena třída Parse.cs. Do této třídy se přes speciální vložení zkopíroval souboru couchdump.json, který soubor převedl na třídy. Vzhledem ke zkopírování celého souboru do třídy Parse.cs, jsou zde zaneseny i parametry, které o měření nebudou potřeba znát. Proto třídy byly upraveny a byly jim zanechány jen parametry, které budou potřeba.

BleScany byly v souboru couchdump.json zastoupeny pomocí MAC adres. Pro každé jedno měření mohl být některý BleScan zachycen několikrát, a proto mohl pokaždé obsahovat různou hodnotou síly signálu. Proto byla pro více hodnot síly signálu jednoho BleScanu vybrána hodnota mediánu. Pokud nějaký BleScan nebyl pro dané měření zachycen, byla mu přiřazena zástupná hodnota 100. To z toho důvodu, že silnější měřič by mohl zachytit nějaký BleScan, který předtím zachycen nebyl. Model by poté například nemusel fungovat nebo ovlivnit hodnoty vystupující z modelu, v tomto případě X a Y pozice na mapce. Toto samé platilo i pro WiFiScany. Pozice X a Y byly pro každé jedno měření stejné, při měření se nepohybovalo.

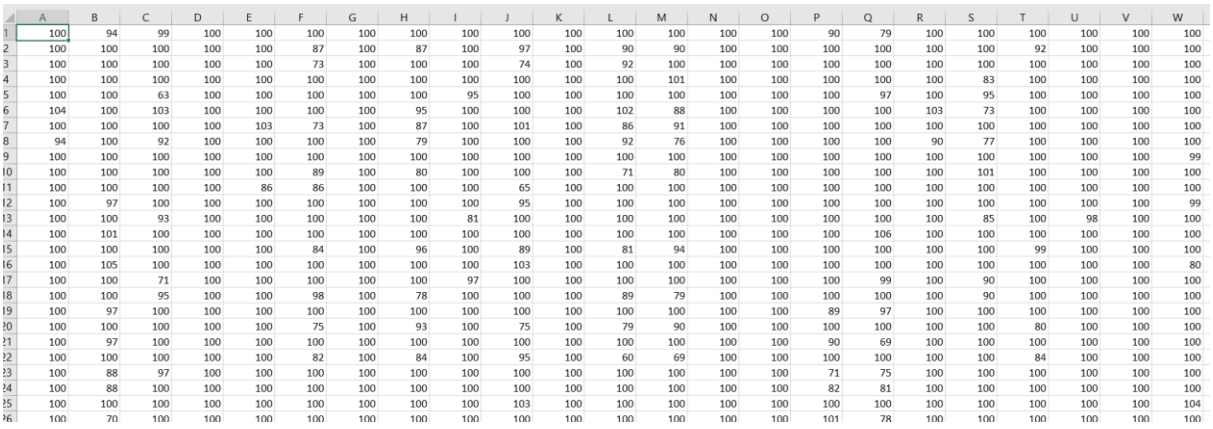
6.1.2. Program

Nejdůležitejším krokem je založení objektu root který je typem Rootobject. Do něj potom přes JsonConvert a metodu DeserializeObject bude načten a deserializován celý soubor couchdump.json. Do objektu root jsou poté uložena data, která jsou vybrána na zakladě našich tříd a názvů jejich paramtetrů. Pokud v souboru najde data, která jsou stejně pojmenována jako názvy parametrů různých tříd, uloží je do sebe. Pokud by žádná taková nenašel, zůstala by prázdná, tedy null.

Poté, co bude daný soubor deserializován, se bude objekt root procházet jako jeden velký list a ukládat data, která budou pro model potřeba. Jak již bylo řečeno, není zapotřebí dat, která nebyla naměřená 26.2.2016 nebo data která nejsou na třetím patře, tedy na J3NP. Pokud tedy tuto podmínku nesplní, toto měření bude z objektu root odstraněno. Poté se uloží MAC adresy všech BleScanů a WiFiScanů, které budou nalezena ve validních měřeních.

Po předchozím kroku už v objektu root zůstala pouze ta měření, která jsou pro model podstatná. Na tyto měření bude nahlédnuto znovu a budou ukládány adresy a síly signálu BleScanů do Dictionary<string, List<int>> bleScans. Může se stát, že pro jedno měření bude naměřeno více signálů pro jeden BleScan. Jak již bylo řečeno, toto je vyřešeno pomocí mediánu. Pokud pro nějaké měření nebyl nějaký BleScan zachycen, přidá se. Síla signálu bude mít zástupnou hodnotu -100. Poté je potřeba BleScany pro dané měření seřadit, aby pokaždé byli ve stejném pořadí. Úplně stejný proces proběhne i s WiFiScany. Nakonec vše, jak je seřazené, bude zapsáno pomocí třídy FileStream do příslušných souborů.

Soubor data.csv, na obrázku číslo 16, bude obsahovat všechna měření, kterých je celkem 340. Každé jedno měření bude v jednom řádku obsahovat sílu signálu z každého BleScanu a WiFiScanu.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	100	94	99	100	100	100	100	100	100	100	100	100	100	100	100	90	79	100	100	100	100	100	100
2	100	100	100	100	100	87	100	87	100	97	100	90	90	100	100	100	100	100	100	92	100	100	100
3	100	100	100	100	100	73	100	100	100	74	100	92	100	100	100	100	100	100	100	100	100	100	100
4	100	100	100	100	100	100	100	100	100	100	100	100	101	100	100	100	100	100	83	100	100	100	100
5	100	100	63	100	100	100	100	100	95	100	100	100	100	100	100	100	97	100	95	100	100	100	100
6	104	100	103	100	100	100	100	95	100	100	100	102	88	100	100	100	100	103	73	100	100	100	100
7	100	100	100	100	103	73	100	87	100	101	100	86	91	100	100	100	100	100	100	100	100	100	100
8	94	100	92	100	100	100	100	79	100	100	100	92	76	100	100	100	100	90	77	100	100	100	100
9	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99
10	100	100	100	100	100	89	100	80	100	100	100	71	80	100	100	100	100	100	101	100	100	100	100
11	100	100	100	100	86	86	100	100	100	65	100	100	100	100	100	100	100	100	100	100	100	100	100
12	100	97	100	100	100	100	100	100	100	95	100	100	100	100	100	100	100	100	100	100	100	100	99
13	100	100	93	100	100	100	100	100	81	100	100	100	100	100	100	100	100	100	85	100	98	100	100
14	100	101	100	100	100	100	100	100	100	100	100	100	100	100	100	100	106	100	100	100	100	100	100
15	100	100	100	100	100	84	100	96	100	89	100	81	94	100	100	100	100	100	100	99	100	100	100
16	100	105	100	100	100	100	100	100	100	103	100	100	100	100	100	100	100	100	100	100	100	100	80
17	100	100	71	100	100	100	100	100	97	100	100	100	100	100	100	100	99	100	90	100	100	100	100
18	100	100	95	100	100	98	100	78	100	100	100	89	79	100	100	100	100	100	90	100	100	100	100
19	100	97	100	100	100	100	100	100	100	100	100	100	100	100	100	89	97	100	100	100	100	100	100
20	100	100	100	100	100	75	100	93	100	75	100	79	90	100	100	100	100	100	100	80	100	100	100
21	100	97	100	100	100	100	100	100	100	100	100	100	100	100	100	90	69	100	100	100	100	100	100
22	100	100	100	100	100	82	100	84	100	95	100	60	69	100	100	100	100	100	100	84	100	100	100
23	100	88	97	100	100	100	100	100	100	100	100	100	100	100	100	71	75	100	100	100	100	100	100
24	100	88	100	100	100	100	100	100	100	100	100	100	100	100	100	82	81	100	100	100	100	100	100
25	100	100	100	100	100	100	100	100	100	103	100	100	100	100	100	100	100	100	100	100	100	100	104
26	100	70	100	100	100	100	100	100	100	100	100	100	100	100	100	101	78	100	100	100	100	100	100

Obrázek 16 - Ukázka ze souboru data.csv, zdroj vlastní

Soubor labels.csv, na obrázku číslo 17 níže, bude obsahovat souřadnice X a Y pro všechna měření, kterých je také 340. Tyto dva soubory použijeme jako vstupy do modelu, který se vytrénuje pro stanovené účely.

	A	B	C	D	E	F	G	H	I	J
1	450	2300								
2	1750	900								
3	1600	750								
4	1450	2100								
5	750	2250								
6	1450	2150								
7	1750	600								
8	1450	2150								
9	450	600								
10	1750	2000								
11	1300	750								
12	450	750								
13	1100	2450								
14	450	1050								
15	1850	750								
16	200	750								
17	750	2450								
18	1650	2100								
19	450	1600								
20	1900	750								
21	450	2050								
22	1750	1100								
23	450	1650								
24	450	1500								
25	450	700								
26	450	1150								

Obrázek 17 - Ukázka ze souboru labels.csv, zdroj vlastní

6.2. Model

Model bude vytvořen v jazyce Python. Python je vysokoúrovňový skriptovací programovací jazyk. Python je vyvíjen jako open source projekt a je používán pro spoustu platforem (Windows, Linux, macOS a další). Při modelování modelu bylo použito prostředí pythonu ve verzi Pythonu 3.5.

Jako další byly při vytváření použity tyto balíčky:

- NumPy ve verzi 1.16.1
- TensorFlow ve verzi 1.13.1
    - Keras ve verzi 2.2.4
    - TensorBoard ve verzi 1.13.1
- Matplotlip ve verzi 3.0.2

Model bude mít pokaždé vstupní vrstvu o 86 neuronech, protože existuje 86 vysílačů, ze kterých se dá naměřit signál a také bude mít výstupní vrstvu o 2 neuronech, protože chceme jako výstup 2 souřadnice – souřadnici X a souřadnici Y. Skryté vrstvy se budou pro některé konfigurace měnit. To proto, že to, kolik skrytých vrstev bude model obsahovat a kolik budou obsahovat skryté vrstvy neuronů ovlivňuje přímo výstupy sítě. Toto bude tedy předmětem testování. Dále se budou testovat různé učící algoritmy. Při testování se bude postupovatv následujících fázích. Nejdřív budou testovány učící algoritmy, poté model sítě s větším počtem skrytých vrstev. V poslední řadě bude otestováno, jak úspěšný bude model, pokud mu bude poskytnuto více učících dat.

6.2.1. Balíčky

Jako první budou naimportovány pro model tyto důležité balíčky:

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

Balíček TensorFlow obsahuje věci potřebné pro vytvoření modelu. TensorFlow v sobě taktéž obsahuje Keras, který se bude převážně používat a díky němu vytvářet model.

Balíček NumPy je základním balíčkem pro vědecké výpočty s Pythonem. Kromě vědeckého využití může být NumPy také použit jako efektivní multidimenzionální kontejner obecných dat. Toto využití je pro potřeby tvorby našeho modelu nejdůležitější. Jak již bylo řečeno, TensorFlow pracuje maticemi a NumPy a pomůže data ze souborů .csv převést na matici.

Matplotlib je 2D grafická knihovna, která dokáže například zobrazit průběh funkce nebo různé typy grafů. Tato knihovna bude v modelu použita kvůli vynesení různých hodnot do grafů.

V poslední řadě bude potřeba balíček Random, který bude z části zaručovat reprodukovatelnost výsledků neuronové sítě.

6.2.2. Reprodukovatelnost výsledků

Pokud je stejný model neuronové sítě trénován několikrát a pokaždé v jiném čase, tak je velmi pravděpodobné, že model neuronové sítě bude obsahovat jiné výsledky. Ať už se jedná hodnotu ztrátové funkce, hodnotu metrik nebo predikce. Toto se stává proto, že

váhy modelu neuronové sítě se na začátku inicializují náhodně. Vzhledem k tomu, že se bude chtít porovnávat výkonost modelu neuronové sítě při jiných nastaveních, je tento efekt nechtěný. Proto bude potřeba tuto náhodu odstranit. Pokud ale bude vytvářen model neuronové sítě a bude používat hledání parametrů pomocí strategie Random Search, tak se tento postup použít nedá. Vzhledem k tomu, že strategie Random Search vychází z náhodného hledání parametrů a pomocí toho postupu „zamkneme“ náhodu na daných hodnotách, tak by se parametry pro každé učení neuronové sítě volily stejně.

Odstranit náhodu se dá pomocí nastavení semínka (seed) generátoru pseudonáhodných čísel. Vzhledem k tomu, že TensorFlow a Keras používá semínka náhody z různých balíčků, je potřeba ho nastavit na více místech. Pokud by se všechna tyto semínka náhody nenastavily, model neuronové sítě by s velkou pravděpodobností pokaždé obsahoval jiné výsledky.

Jako první je potřeba nastavit semínko náhody v samotném pythonu:

```
os.environ['PYTHONHASHSHEED'] = '0'
```

Toto semínko náhody zaručuje reprodukovatelnost jako takovou. Jeho hodnota musí být pokaždé rovna nule. Jako další je nastavení náhody balíčku NumPy, knihovny Pythonu Random a TensorFlow.

```
np.random.seed(1)
rnd.seed(1)
tf.set_random_seed(1)
```

Hodnota semínka náhody zde může být volen jakýkoliv Integer a čísla se nemusí rovnat. Jde vyloženě o to výběr semínka náhody, které tyto knihovny budou používat.

V poslední řadě je donutit Keras používat pouze jedno vlákno ke svým výpočtům neboli použití více vláken může vést k tomu, že výsledky nebudou reprodukovatelné. Takovýto kód vypadá takto:

```
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
tf.keras.backend.set_session(sess)
```

Všechna tato nastavení vedou k reprodukovatelnosti výsledků a jsou všechna pro reprodukovatelnost zásadní. Žádné z těchto nastavení by se nemělo vynechat, aby

reprodukovatelnost byla zaručena. Tato nastavení by také měla proběhnout na začátku programu. Až poté by měly následovat věci jako tvorba modelu neuronové sítě a její samotný trénink.[30] [31]

Pokud se bude při hledání parametrů používat strategie Random Search, nedá se reprodukovatelnost zaručit, protože se pracuje právě s náhodou. Pokud by se například volil z nějakého rozmezí hyperparametr rychlost učení (learning rate), tak by byla pro každý pokus vybrána stejná hodnota.

7. TESTOVÁNÍ A VÝSLEDKY

Cílem testování bude vyzkoušet různé strategie hledání hyperparametrů neuronové sítě a porovnání jejich výsledků. Cílem také bude nalezení co nejlepší neuronové sítě pro indoor lokalizaci. Všechny výsledky budou obsahovat hodnoty v pixelech, kde se přibližně 60 pixelů rovná jednomu metru.

7.1. Tvorba NN pomocí strategie Babysitting

Jak již byl řečeno, strategie Babysitting je při hledání hyperparametrů neuronové sítě plně manuální. Tento postup hledání hyperparametrů byl ze všech nejrychlejší. Konfigurátor sítě, tedy člověk, který síť konfiguruje, může při hledání nejlepší konfigurace neuronové sítě postupovat podle instinktu a konfiguraci upravovat na základě výsledků předchozích konfigurací. Hyperparametry při hledání konfigurace neuronové sítě byly:

- epochs = 100
- lr = 0.001
- validation_split = 0.2
- test_data_size = 34

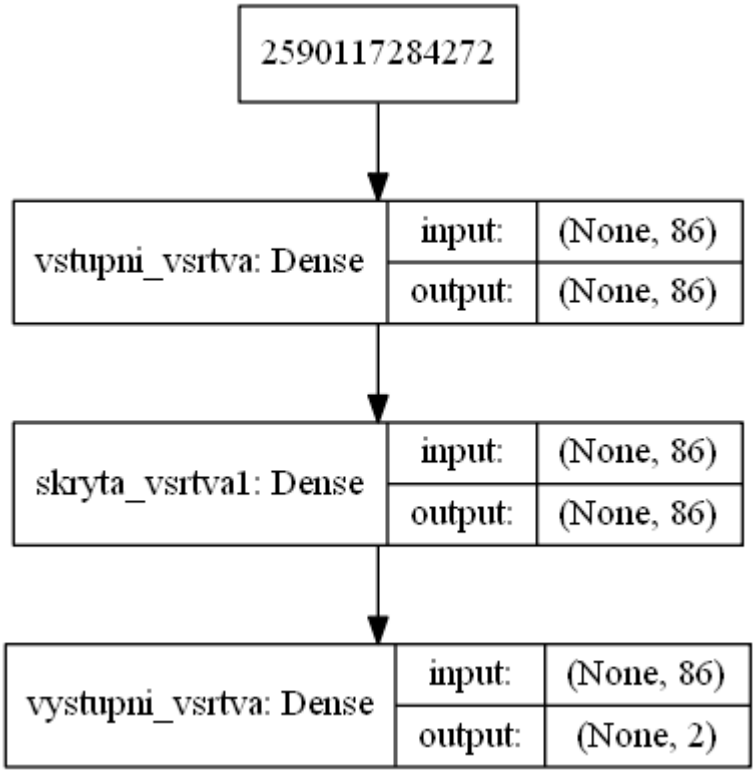
Kde epochs je počet iterací modelu. Lr je rychlost učení učícího algoritmu. Hyperparametr validation_split slouží k rozdělení tréninkových dat v daném poměru na tréninková a validační. Zde je nastaveno 0.2, což znamená že 20 % tréninkových dat bude použito pro validaci. Jako poslední je zde test_data_size. Tento parametr usekne daný počet z tréninkových dat a použije je jako data testovací.

7.1.1. První testovací data – 340 příkladů

Při prvních pokusech hledání nejlepší konfigurace bylo k tréninku použito 340 záznamů jak pro trénink, validaci a testování.

7.1.2. Testování učících algoritmů

Neuronová síť se při testování učících algoritmů skládá celkem ze 3 vrstev. První, vstupní vrstva, obsahuje 86 neuronů, protože pro každé měření existuje 86 sil signálů (z 86 unikátních MAC adres WiFi přístupových bodů a Bluetooth Low Energy Beaconů). Další vrstva obsahuje 86 neuronů a jedná se o skrytou vrstvu. Poslední vrstva je výstupní a obsahuje pouze dva neurony. Jak tato síť vypadá je možné si prohlédnout na obrázku 18. Tyto dva neurony reprezentují výstup X a Y, které určují pozici na mapce. Jak takováto síť vypadala je znázorněno na obrázku níže. Všechny tyto vrstvy, včetně výstupní, měly aktivační funkci Relu. Pouze výstupní vrstva obsahuje lineární aktivační funkci, protože pokud se jedná o neuronovou síť, které řeší problém regrese, tak by měla používat právě lineární aktivační funkci.



Obrázek 18 - Model nerozšířené neuronové sítě pro testování, zdroj vlastní

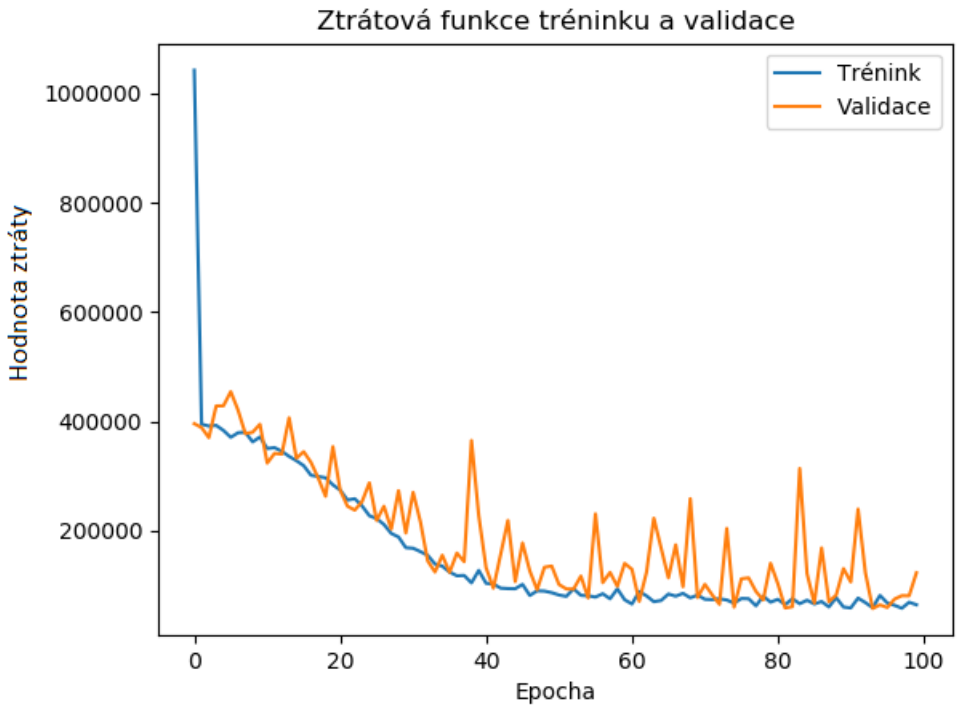
Pro každou predikci výsledků byla vždy použita stejná data vstupních signálů. Vždy se jednalo o stejný počet měření. Tento počet se nastavoval v hyperparametru

test_data_size, pro které se pak predikovala pozice X a Y. V těchto testováních byl tento hyperparametr vždy 34.

7.1.2.1. RMSprop

Jako první byl při testování použit učící algoritmus RMSprop. Tento učící algoritmus je jeden z nejzákladnějších. [23] Při testování nedosahoval nijak dobrých výsledků. Jak je vidět na grafu číslo 1, při tréninku ztrátová funkce nejdřív klesala velmi rychle, což je při tréninku celkem běžný jev. Poté ale měla menší výkyvy a občas se hodnoty ztráty funkce zvedly, což znamená, že algoritmus byl v předchozí iteraci přesnější. Ve výsledku ale ztrátová funkce při tréninku klesala.

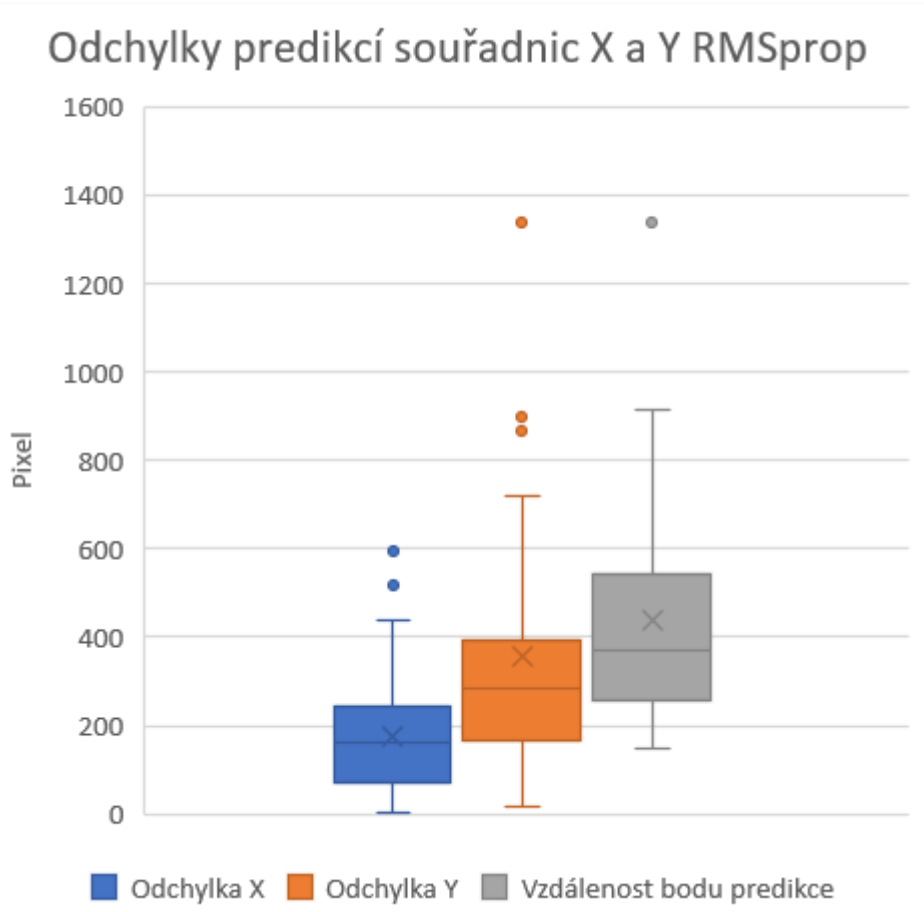
Viditelně větší problém měla neuronová síť při validaci, kde se hodnoty ztrátové funkce chovaly jako na horské dráze. Jednou byla neuronová síť při validaci přesná, ale při další iteraci se její přesnost zvětšila například až dvojnásobně. Znamená to tedy, že si neuronová síť občas „neví rady“ s daty, která ještě neviděla.



Graf 1 - Ztrátová funkce tréninku a validace RMSprop, zdroj vlastní

Pokud se jedná o predikce, tak ztrátová funkce dosáhla hodnoty 267 na testovacích datech. To znamená, že model neuronové sítě se v průměru mýlil v každé predikci na každém X a Y o 267 pixelů. Pokud se jedná a individuální predikce pro X, tak průměrná odchylka byla 176 pixelů a průměrná odchylka pro Y byla 358 pixelů. V tomto případě je vidět, že si neuronová síť nedokáže dobře poradit s predikcí Y koordinace.

Graf číslo 2 ukazuje, že odchylka pro souřadnice X a Y několikrát dosahovala extrémních hodnot. Model neuronové sítě měl také mnohem větší problémy predikovat souřadnici Y. Celkově byla průměrná vzdálenost bodu predikce 439 pixelů (7,316 metrů) a medián 369 pixelů (6,15 metrů).



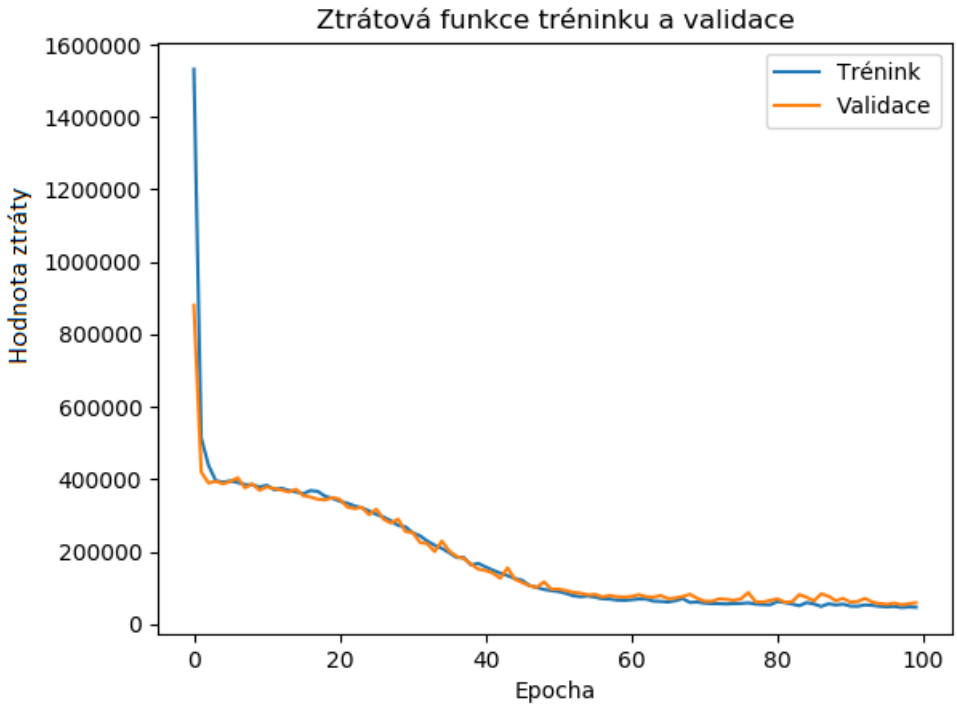
Graf 2 - Odchylka predikcí RMSprop, zdroj vlastní

7.1.2.2. Adam

Učící algoritmus Adam je momentálně jeden z neoblíbenějších a dnes nejpoužívanějších učících algoritmů. [23][29] Je výpočetně efektivní a má malé paměťové nároky. Jak je

zobrazeno na obrázku níž. Na začátku při tréninku ztrátová funkce klesá velmi rychle. Pak se přesnost na tréninkových datech ustálí a začne klesat velmi pomalu.

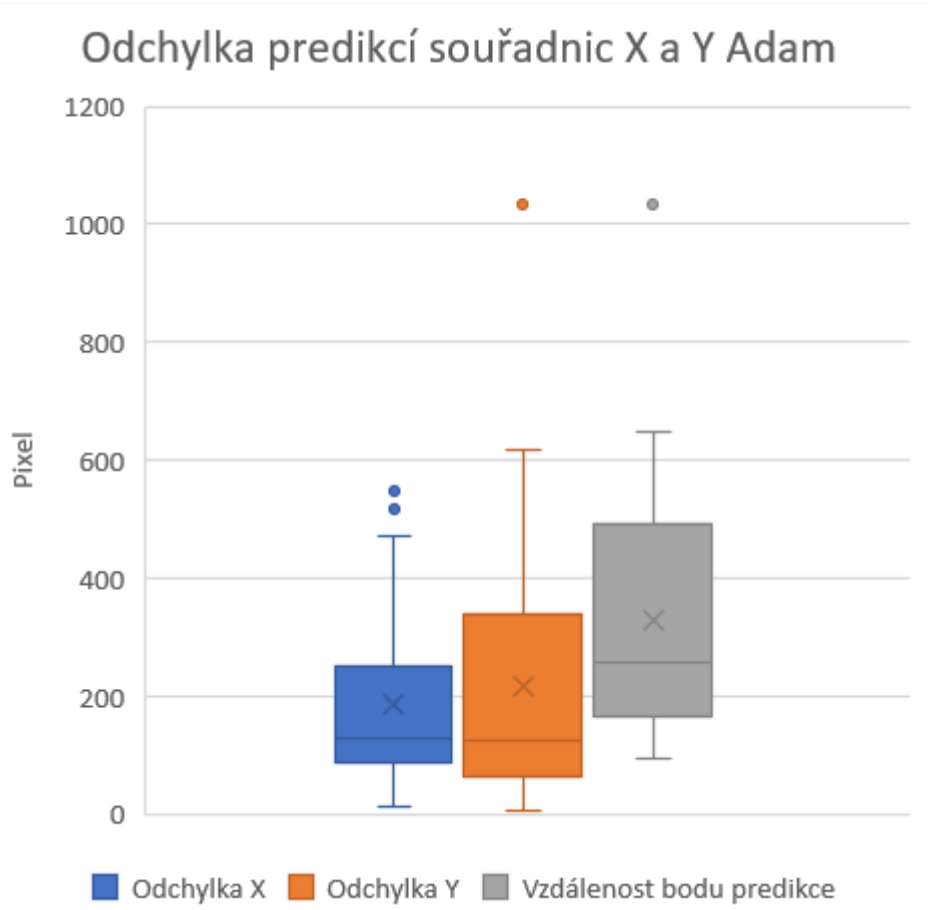
Pokud se jedná o validaci, tak oproti algoritmu RMSprop nedosahuje při validaci v podstatě žádných výkyvů. Její hodnota ztrátové funkce krásně kopíruje hodnotu ztrátové funkce při tréninku, což značí že si neuronová síť dokáže poradit stejně i s daty, která neviděla při tréninku. Toto vše se dá vyčíst s grafu číslo 3.



Graf 3 - Ztrátová funkce tréninku a validace Adam, zdroj vlastní

Při predikování byla hodnota ztráty 201, což znamená průměrnou chybu predikce pro každé měření a pro každé X a Y tohoto měření 201 pixelů. Na grafu číslo 2 je možno vidět, že průměrná odchylka pro X souřadnici byla 186 pixelů a pro Y souřadnici byla 216 pixelů. Zde je vidět oproti učícímu algoritmu RMSprop obrovské zlepšení co se týká predikce Y souřadnice, ale mírné zhoršení predikce X souřadnice. Na něm je vidět, že u predikcí Y souřadnic už odchylka nedosahuje několika extrémních hodnot, ale obsahuje již jen jednu extrémní hodnotu.

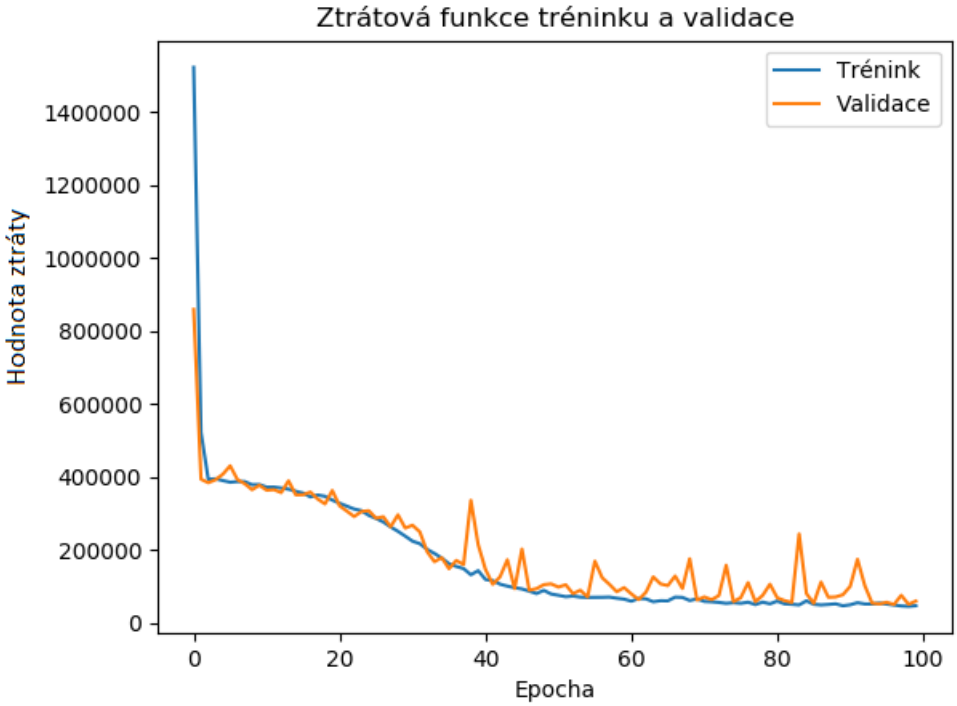
Graf číslo 4 vyobrazuje že průměrná vzdálenost predikce bodu byla 330 pixelů (5,5 metrů), medián byl 257 pixelů (4,283 metrů), což je oproti předchozímu výsledku za použití učícího algoritmu RMSprop značné zlepšení.



Graf 4 - Odchylka predikcí Adam, zdroj vlastní

7.1.2.3. Nadam

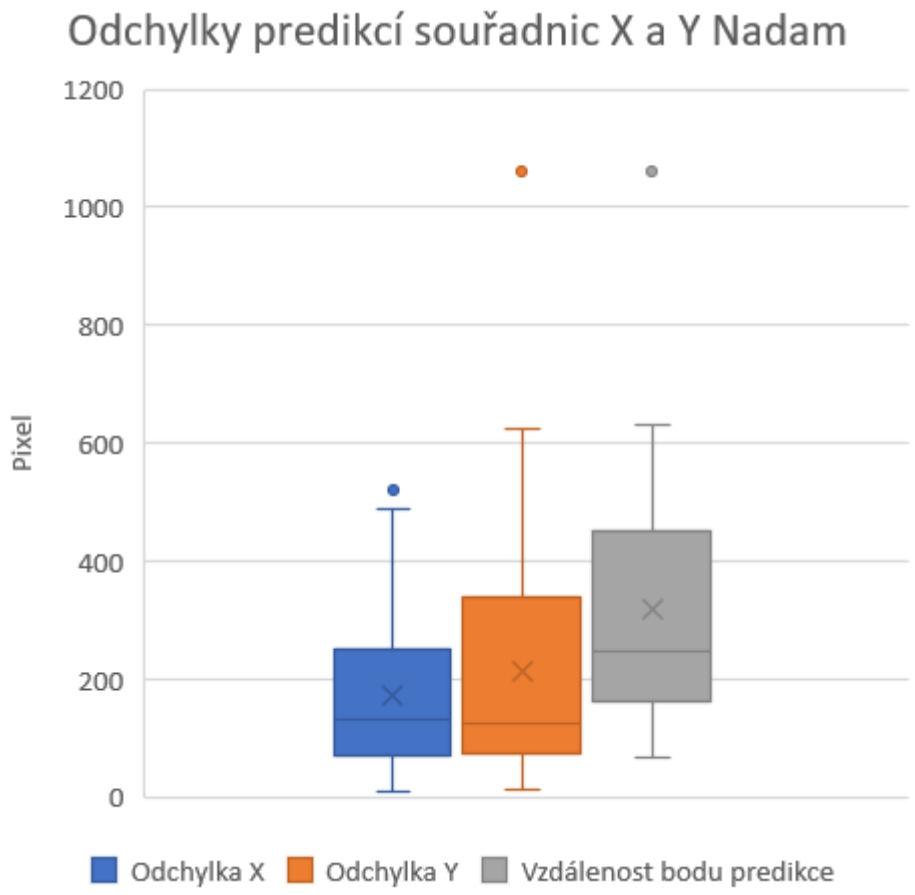
Učící algoritmus Nadam kombinuje učící algoritmy NAG a Adam. [23] Z tohoto pohledu by se mělo zdát, že by tento účící algoritmus měl dosahovat nejlepších výsledků. Pokud se podíváme na graf níž, je vidět, že trénink probíhal velmi podobně jako v případě učícího algoritmu Adam. Při validaci měl Nadam při některých validacích větší hodnotu ztráty.



Graf 5 - Ztrátová funkce tréninku a validace Nadam, zdroj vlastní

Hodnota ztrátové funkce při predikcích byla oproti algoritmu Adam lehce nižší s hodnotou 194. Také se zlepšily obě průměrné odchylky pro jednotlivé souřadnice X a Y. Průměrná odchylka souřadnice pro X byla 174 pixelů a odchylka pro Y byla 215 pixelů. Je zajímavé, že i přes to, že si při validaci učící algoritmus Nadam nevedl tak dobře jako učící algoritmus Adam, tak i přes to má při predikci lepší výsledky.

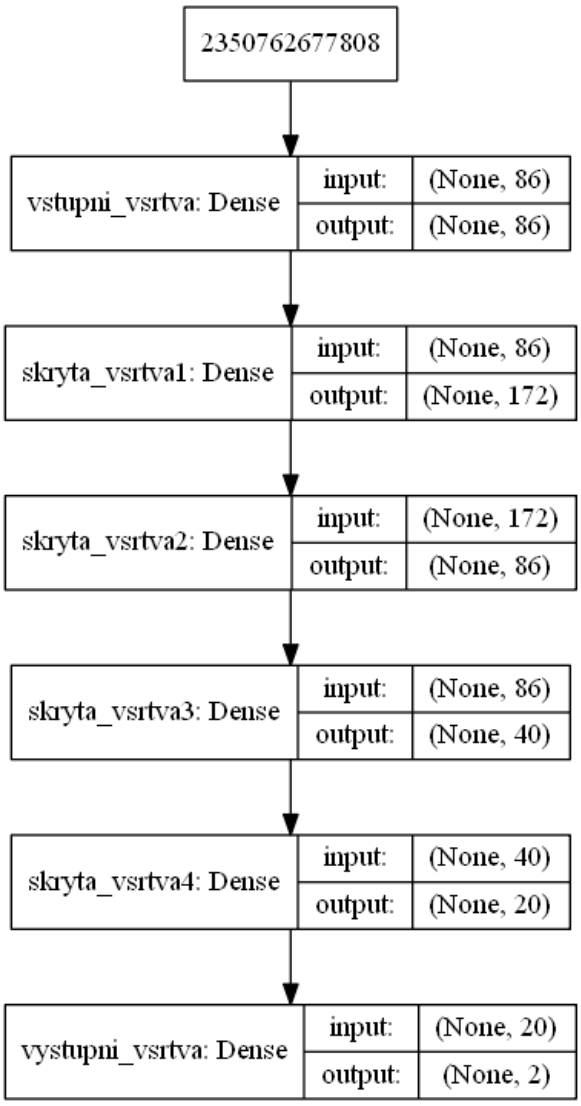
Jak je vidět na grafu odchylek predikcí číslo 6, výsledky jsou velmi podobné jako u učícího algoritmu Adam. Přesto odchylky dosahují velmi mírného zlepšení. Průměrná vzdálenost predikce a medián vzdálenosti predikce dosahují mírného zlepšení. Průměrná vzdálenost bodu predikce je 318 pixelů (5,3 metrů), medián 248 pixelů (4,13 metrů).



Graf 6 - Odchylka predikcí Nadam, zdroj vlastní

7.1.3. Rozšíření neuronové sítě

Z předchozích výsledků testování je patrné, že si nejlépe a velmi podobně si vedly při predikcích učící algoritmy Nadam a Adam. Jako dalším logickým krokem se jeví rozšíření neuronové sítě o další skryté vrstvy. Tato neuronová síť bude tedy vypadat nějak takto:



Obrázek 19 - Model rozšířené neuronové sítě pro testování, zdroj vlastní

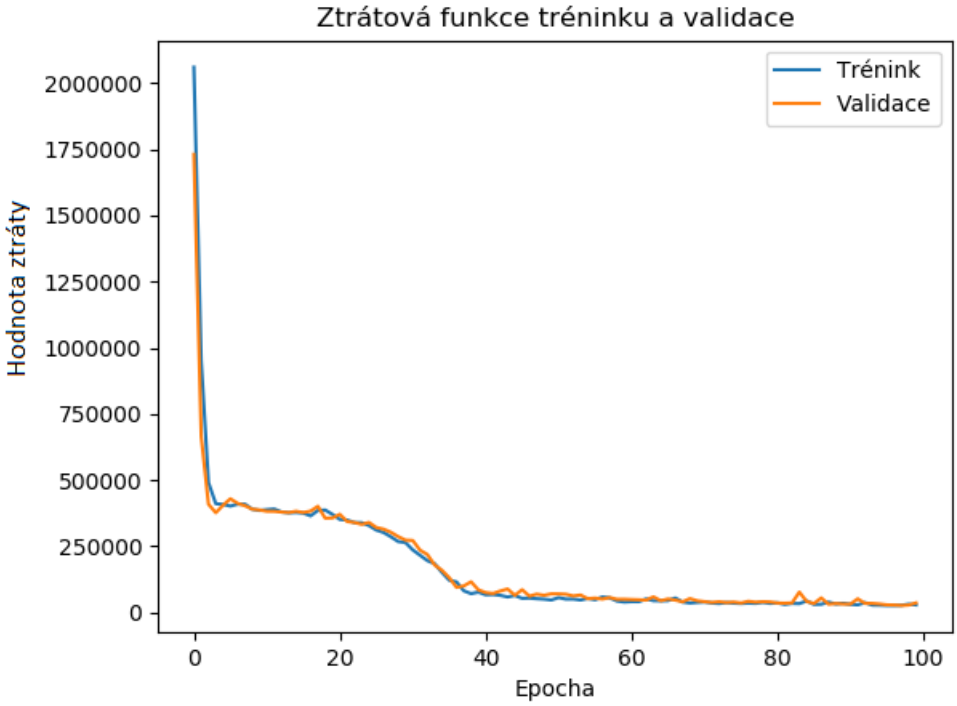
První skrytý vrstva bude obsahovat 172 neuronů. To proto, aby se data rozmělnila. Dalo by se říct, že se ze vstupu 86 sil signálů udělá 172 sil signálů. Tím pádem bude moci neuronová síť nastavit více vah a tím přesněji určit souřadnice X a Y. Poté budou přidány další dvě skryté vrstvy o 86, 40 a 20 neuronech. Tyto vrstvy budou obsahovat méně neuronů, protože na výstupu potřebujeme jen dva výstupy. Tím, že bude v každé další vrstvě méně neuronů by mělo směřovat k tomu, že se budou vybírat jen váhy které jsou důležité a ostatní se „zahodí“. Což znamená, že jim bude přiřazena menší váha. Nakonec v neuronové síti bude znovu výstup o dvou neuronech. Tento výstup zajistí predikce součadnice X a Y.

Pro každou predikci výsledků byla vždy použita stejná data vstupních signálů. Vždy se jednalo o stejný počet měření. Tento počet se nastavoval v hyperparametru

test_data_size, pro které se pak predikovala pozice X a Y. V těchto testováních byl tento hyperparametr vždy 34.

7.1.3.1. Rozšíření neuronové sítě s učícím algoritmem Adam

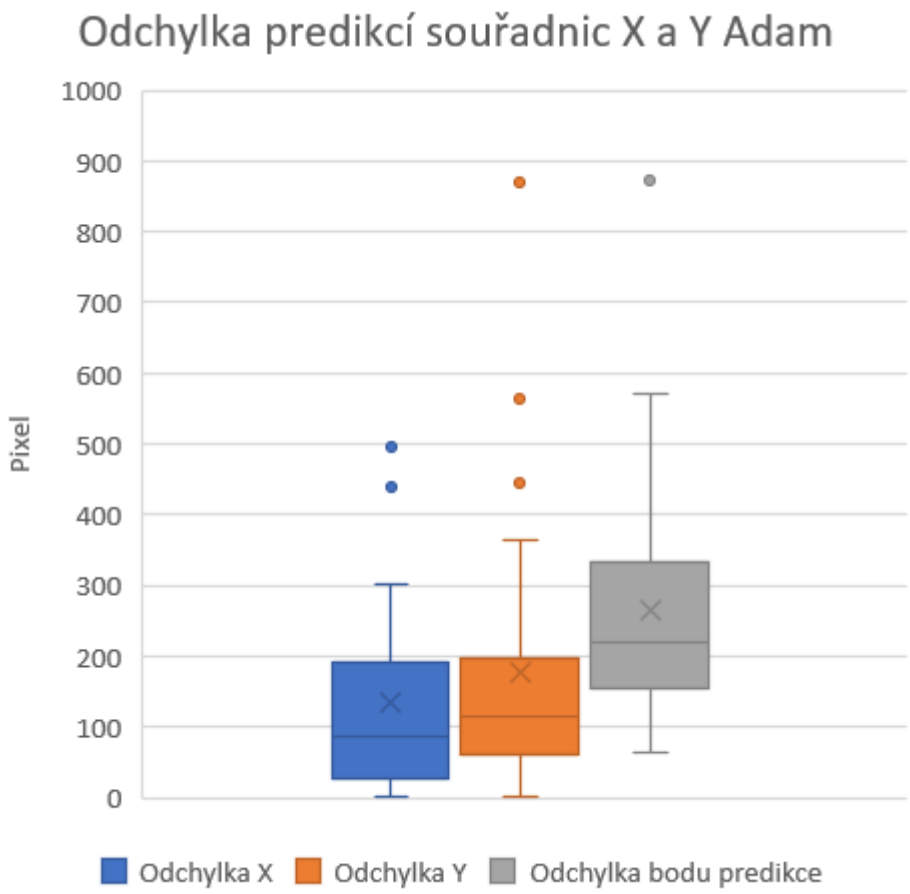
Jak je graficky níž zobrazeno na grafu číslo 7, průběh ztrátové funkce se při tréninku ani validaci nijak neliší od výsledků menší sítě s použitím stejného učícího algoritmu Adam. Hlavním rozdílem je větší přesnost predikce. Hodnota ztrátové funkce při predikci byla 157, což je o více než dvacetiprocentní zlepšení oproti předchozímu testování menší neuronové sítě se stejným učícím algoritmem.



Graf 7 - Ztrátová funkce tréninku a validace Adam, rozšířená NN, zdroj vlastní

Na dalším grafu číslo 8 je vidět, že se průměrná odchylka predikce souřadnice X a také souřadnice Y snížila. Hodnota průměrné odchylky pro souřadnici X byla 135 pixelů, pro souřadnici Y byla 178 pixelů. Dokonce nejmenší odchylka od reálné souřadnice X byla pouhé dva pixely. Nejmenší odchylka od reálné souřadnice Y byla 3 pixely. Přesto že se přesnost neuronové sítě zvedá, přesto tento model nedokáže predikovat souřadnici Y tak dobře jako souřadnici X. I když se vrchní hranice odchylky pro souřadnici Y rapidně snížila, obsahuje stále několik extrémů a spodní hranice odchylky stále není tak dobrá jako u predikce souřadnice X. Vzdálenost bodu predikce měla při tomto testování

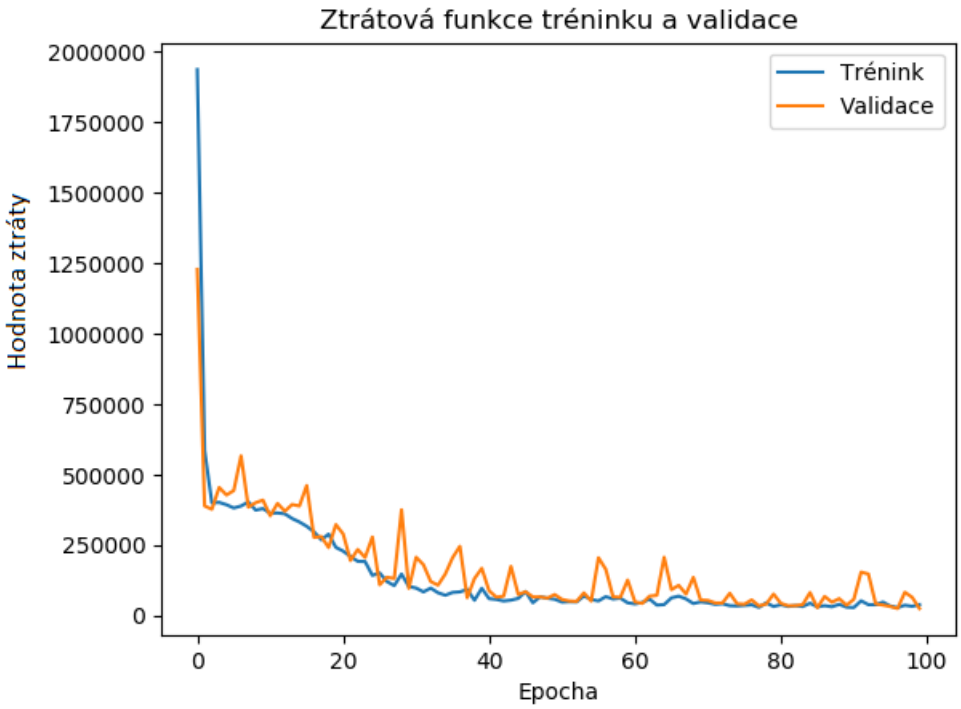
průměr 266 pixelů (4,43 metrů) a median 219 pixelů (3,65 metrů). Zde je tedy také vidět jasné zlepšení oproti předchozím testováním s menšími sítěmi.



Graf 8 - Odchylka predikcí Adam, rozšířená NN, zdroj vlastní

7.1.3.2. Rozšíření neuronové sítě s učícím algoritmem Nadam

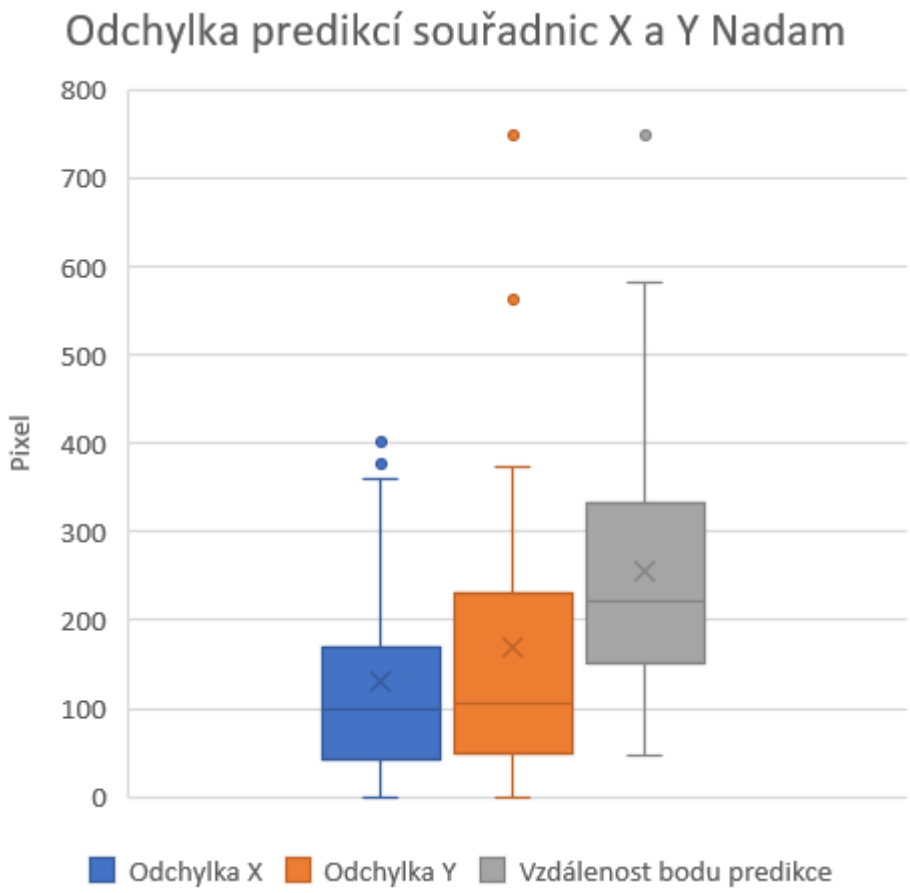
Jak je možné si všimnout v grafu číslo 9, průběh ztrátové funkce je pro validaci a trénink opět velmi podobný jako při použití menší neuronové sítě se stejným učícím algoritmem Nadam.



Graf 9 - Ztrátová funkce tréninku a validace Nadam, rozšířená NN, zdroj vlastní

Učící algoritmus Nadam opět snížil celkovou hodnotu ztrátové funkce při predikci. Hodnota ztrátové funkce byla 149, což není obrovská změna oproti modelu stejné sítě s učícím algoritmem Adam. Také zlepšil průměrné odchylky predikcí pro jednotlivé souřadnice X a Y. Průměrná odchylka se zlepšila na 130 pixelů pro souřadnici X, respektive 169 pixelů pro souřadnici Y. Ubyly také extrémní hodnoty a klesla spodní hranice odchylky pro Y. V poslední řadě se neuronové síti podařilo určit také pro nějakou predikci přesnou souřadnici X i souřadnici Y. Takže odchylka byla 0. Toto vše se dá vypozorovat na grafu níž.

Z tohoto grafu lze také vypozorovat že průměrná vzdálenost bodu predikce je 255 pixelů (4,25 metrů) a medián je 220 pixelů (3,67 metrů). Oproti předchozímu testování se sice průměrná odchylka zlepšila, ale medián se zhoršil o jeden pixel.



Graf 10 - Odchylka predikcí Nadam, rozšířená NN, zdroj vlastní

7.1.4. Umělé vytvoření více učících dat

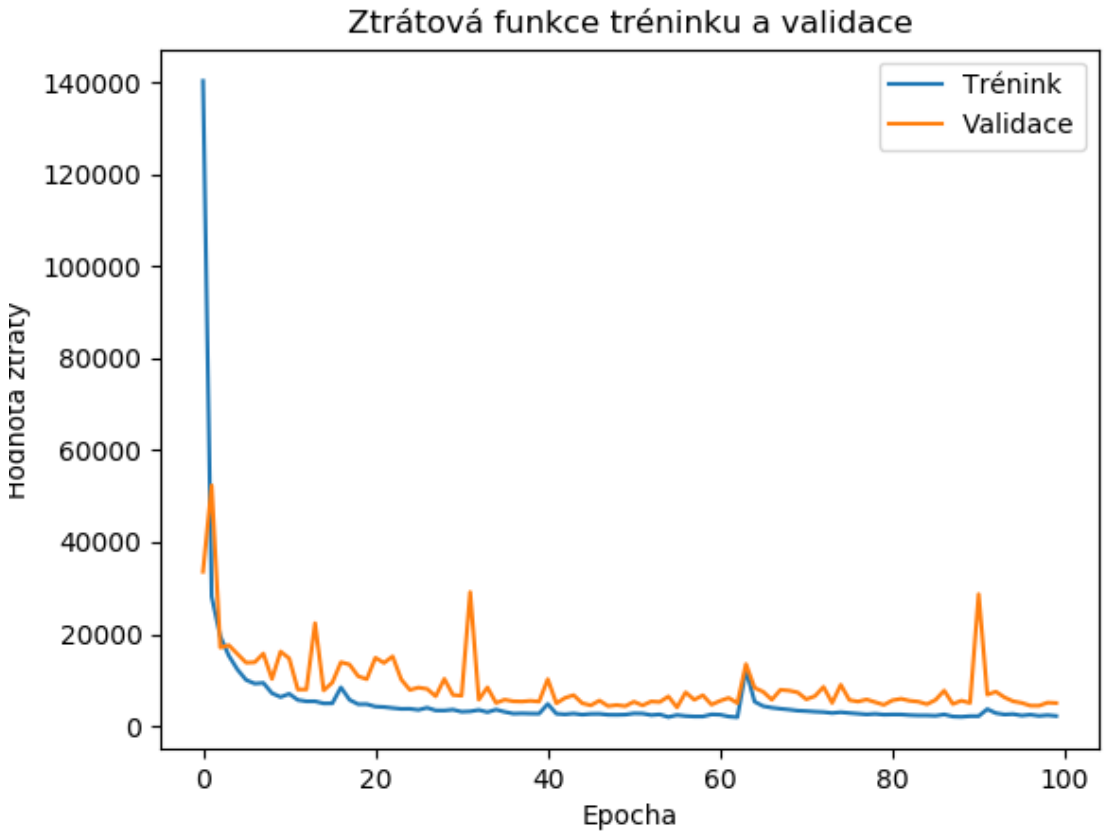
V přechozích výsledcích testování vyšel nejlépe rozšířený model neuronové sítě s učícím algoritmem Nadam. Jako další možností vylepšení sítě je sehnat více různorodých dat k trénování a testování. Vzhledem k tomu, že pro naše účely bylo validních pouze 340 měření, bylo potřeba uměle vytvořit další.

Jak již bylo řečeno, každé měření obsahuje 86 sil signálů z BleScanů a WiFiScanů. Každá tato síla signálu je zastoupena střední hodnotou z naměřených sil signálu pro dané měření. Uměle vytvořená měření byla vytvořena tak, že se pro jedno dané měření našel WiFiScan nebo BleScan, který obsahoval nejvíc naměřených sil signálů. Tento počet naměřených sil signálu určoval, kolikrát se dané měření uměle vytvoří. Potom se pro každé uměle vytvořené měření vybere vždy jedna síla signálu z každého WiFiScanu nebo BleScanu. Pokud už pro dané uměle vytvořené měření není žádná další naměřená síla signálu WiFiScanu nebo BleScanu, tak se naměřené síly signálu budou iterovat znovu od

začátku. Tímto se v datech nebudou objevovat zástupné hodnoty 100, což znamená že pro dané měření nebyl signál naměřen.

Aby uměle vytvořená data byla ve správném poměru, je potřeba v tomto poměru už rozdělit data původní. To znamená, že budou nalezena validní měření, kterých bylo pro účely neuronové sítě 340 a poté se rozdělí v poměru 70% trénovacích dat, 20% validačních dat a 10% testovacích dat. Až poté bude na těchto částech použito umělé vytvoření dat. Poté se vytvořil tvlášť soubor .csv pro test, validaci a trénink.

Po umělém vytvoření bude tedy 23 647 trénovacích dat, 6 893 validačních dat a 3 277 testovacích dat. Neuronové síť bude vypadat takto bude stejný jako při testování rozšířené sítě. Jako učící algoritmus bude použit algoritmus Nadam.



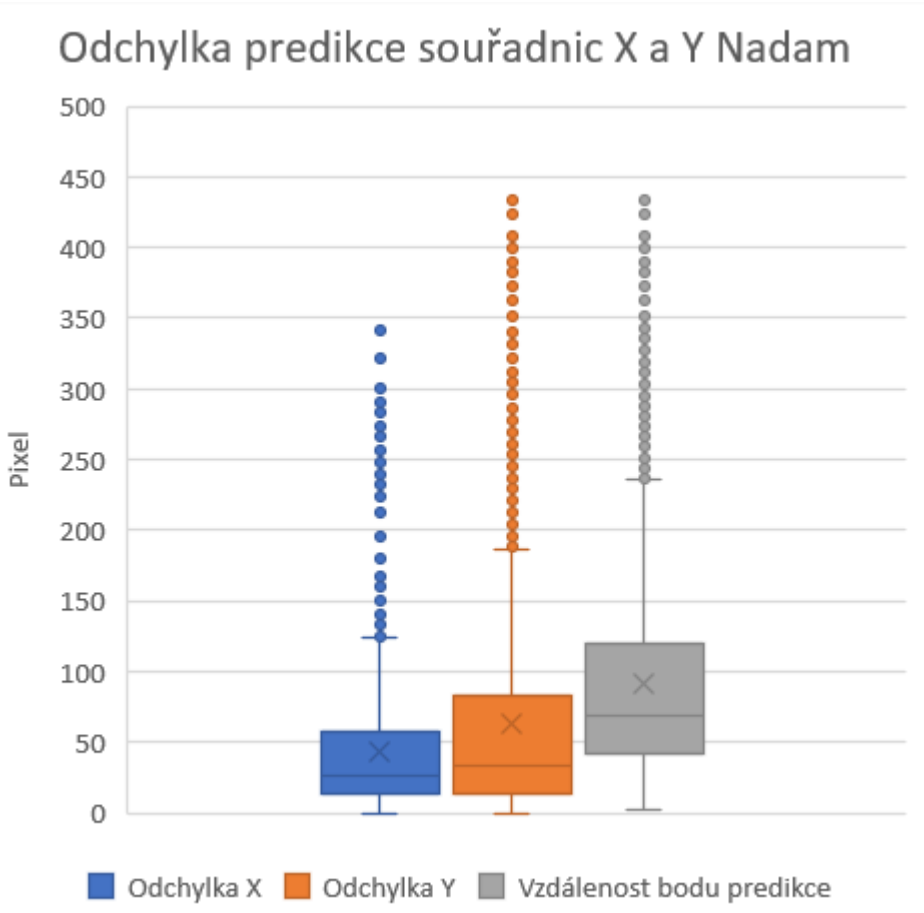
Graf 11 - Ztrátová funkce tréninku a validace Nadam, rozšířená NN, více dat, zdroj vlastní

Jak je vidět na grafu 11 výš, průběh hodnot ztrátové funkce je opět velmi podobný jako při předchozích použitích učícího algoritmu Nadam s modelem menší sítě, i s modelem s rozšířeným - občas obsahuje výkyvy při validaci. Průměrná odchylka při predikcích

byla 53 pixelů. Cože je o 77 pixelů méně než v předchozích testováních s méně daty. Průměrná odchylka při predikcích byla na souřadnici X 44 pixelů, na souřadnici Y 63 pixelů. Jak je možno vidět na grafu níž, přesto že průměrné odchylky pro každou souřadnici vycházejí velice pěkně, přesto existují predikce, kdy se odchylka pro souřadnici Y dostala až k hodnotě 450 pixelů. Také zde existují predikce naprosto přesné, tedy s hodnotou 0, pro obě souřadnice.

V grafu číslo 12 se objevuje mnoho extrémních hodnot. To je hlavně proto, že bylo vytvořeno několik umělých měření, s trošku jinými silami signálů u některých BleScanů nebo WiFiScanů, se stejnými souřadnicemi. Proto i když se například bude většina signálů rovnat, ale pak budou některé síly signálů jen lehce vyšší nebo nižší, tak můžou ovlivnit výsledek predikce.

Pokud se jedná o vzdálenost bodu predikce tak průměr byl 92 pixelů (1,53 metrů), medián 68 pixelů (1,13 metrů). Pokud tyto výsledky porovnáme jakýmkoliv předchozím testováním, tak jsou výsledky mnohem lepší.



Graf 12 - Odchylka predikcí Nadam, rozšířená NN, více dat, zdroj vlastní

7.2. Výsledky

Výsledky vycházejí celkem z šesti testů. V prvních třech testech model vypadal tak jako na obrázku číslo 18. Jednalo se tedy o malý model sítě s jednou skrytou vrstvou. Při těchto testech se porovnávali učící algoritmy RMSprop, Adam a Nadam. Tyto učící algoritmy byly vybrány, protože učící algoritmus RMSprop jeden ze základních učících algoritmu, nejedná se o nějak zvláště používaný učící algoritmus. Učící algoritmus Adam je momentálně jeden z nejvíce používaných učících algoritmů. Jako třetí byl vybrán učící algoritmus Nadam, jedná se o rozšířenou verzi učícího algoritmu Adam. [23][29] Výsledky po těchto třech testováních ukázali, že učící algoritmus RMSprop dosahuje mnohem horších výsledků než učící algoritmy Adam a Nadam, kde výsledky byly velmi podobné.

Při dalších dvou testováních byl model neuronové sítě rozšířen o další 3 skryté vrstvy. Model tedy obsahoval celkem 4 skryté vrstvy. Jak tento model vypadal, je znázorněno na obrázku číslo 19. S tímto modelem už byly testovány jen učící algoritmy Adam a Nadam. Výsledky byly opět velmi podobné, ale výsledky učícího algoritmus Nadam byly lehce lepší. Proto byl tento model neuronové sítě s učím algoritmem Nadam použit i při posledním testování, kde pracovalo s větším objemem učících, validačních a testovacích dat díky umělému vytvoření z dat původních. Díky poskytnutí většímu počtu dat byly výsledky tohoto testování velmi dobré. Ve finále byla průměrná vzdálenost předikovaného bodu od toho původního 92 pixelů, což je v přepočtu na metry 1,53 metrů. Medián vzdálenosti předikovaného bodu byl 68 pixelů – 1,13 metrů.

Výsledky těchto šesti testování jsou zobrazeny v tabulce číslo 1.

60 pixelů = 1 metr	Ztrátová funkce	Průměrná odchylka X	Průměrná odchylka Y	Vzdálenost bodu predikce průměr	Vzdálenost bodu predikce medián
Malá síť, 340 dat, algoritmus RMSprop	267 px	176 px	358 px	439 px = 7,316 metrů	369 px = 6,15 metrů
Malá síť, 340 dat, algoritmus Adam	201 px	186 px	216 px	330 px = 5,5 metrů	257 px = 4,283 metrů
Malá síť, 340 dat, algoritmus Nadam	194 px	174 px	215 px	318 px = 5,3 metrů	248 px = 4,13 metrů
Rozšířená síť, 340 dat, algoritmus Adam	157 px	135 px	178 px	266 px = 4,43 metrů	219 px = 3,65 metrů
Rozšířená síť, 340 dat, algoritmus Nadam	149 px	130 px	169 px	255 px = 4,25 metrů	220 px = 3,67 metrů
Rozšířená síť, umělě vytvořená data, algoritmus Nadam	53 px	44 px	63 px	92 px = 1,53 metrů	68 px = 1,13 metrů

Tabulka 1 - Výsledky testování, zdroj vlastní

8. ZÁVĚR

V rámci bakalářské práce byla nejdřív probrána teorie týkající se biologického neuronu a biologické neuronové sítě a podobnost s umělým neuronem a umělou neuronovou sítí. V další části byla popsána knihovna TensorFlow a její základy a použití. Poté byla vysvětlena knihovna Keras, byly zde popsány její základní prvky důležité k vytvoření neuronové sítě. Popsané základy jako je ztrátová funkce, optimalizátory, metriky a podobně.

V poslední části se testovalo vytvoření neuronové sítě, které měla za úkol lokalizovat pozici na mapce školy na základě vstupu signálů z různých vysílačů do neuronové sítě. Nejprve zde bylo ukázáno, jak byla vybrána data pro model neuronové sítě. Poté zde bylo otestováno několik učících algoritmů, větší rozložení sítě a závislost učení na velikosti učících, validačních a testovacích dat. Z výsledků vyplývá, že neuronová síť dosahovala při posledním testování solidních výsledků, kde se predikce pro každou souřadnici lišila jen velmi málo. Nejlepších výsledků dosahovala poslední testovaná neuronová síť. Jednalo se o rozšířenou neuronovou síť, která trénovala za pomoci uměle vytvořených učících dat. Jako učící algoritmus používala tato neuronová síť algoritmus Nadam. Medián vzdálenosti predikovaného bodu od bodu, který měla neuronová síť určit byl 69 pixelů, což je 1,15 metrů.

Testování by se dala ještě rozšířit o hledání hyperparametrů pomocí strategie Grid Search a Random Search. Pomocí těchto strategií by se s velkou pravěpodobností dal vybudovat model neuronové sítě, který by pracoval lépe a byl přesnější. Bohužel tyto dva procesy jsou velmi výpočetně náročné a vzhledem k tomu, že testování probíhala výhradně na CPU (Centrálné procesorová jednotka), tak by jedno testování mohlo trvat až několik desítek hodin. Pokud by se podařilo nainstalovat TensorFlow které pracuje s GPU (Grafická procesorová jednotka), tyto procesy by byly znatelně rychlejší. [27] [28]

SEZNAM POUŽITE LITERATURY

[1] MEDIAWIKI. Neuron [online]. [cit. 18.7.2019] Dostupné z: https://www.wikiskripta.eu/w/Neuron

[2] Mgr. MEDALOVÁ, Kristína. Neuron a jeho stavba [online]. [cit. 18.7.2019] Dostupné z: https://www.mentem.cz/blog/neuron/

[3] TEAM SHRIMPHOOD.NET, fjura. Neuronové sítě, Vývoj a testování [online]. [cit. 18.7.2019] Dostupné z: http://www.shrimphood.net/neuronove-site-vyvoj-a-testovani.html

[4] JAIN, K. Anil, MAO, Jianchang. Artificial Neural Networks: A Tutorial [online]. [cit. 18.7.2019] Dostupné z: http://metalab.uniten.edu.my/~abdrahim/mitm613/Jain1996_ANN%20-%20A%20Tutorial.pdf

[5] WIKIPEDIA. Supervised Learning [online]. [cit. 18.7.2019] Dostupné z: https://en.wikipedia.org/wiki/Supervised_learning

[6] GRAVES, Alex. Unsupervised Learning [online]. [cit 18.7.2019] Dostupné z: https://deepmind.com/blog/unsupervised-learning/

[7] sem odkaz na TensorFlow

[8] YEGULALP, Sedar. What is TensorFlow? The machine learning library explained [online]. [cit. 18.7.2019] Dostupné z: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[9] TENSORFLOW. TensorBoard: Visualizing Learning [online]. [cit. 18.7.2019] Dostupné z: https://www.tensorflow.org/guide/summaries_and_tensorboard

[10] TENSORFLOW. Install TensorFlow with pip [online]. [cit. 18.7.2019] Dostupné z: https://www.tensorflow.org/install/pip

[11] TENSORFLOW. Why TensorFlow: Case Studies [online]. [cit. 18.7.2019] Dostupné z: https://www.tensorflow.org/about/case-studies/

[12] EXASTAX. TOP FIVE USE CASES OF TENSORFLOW [online]. [cit. 18.7.2019] Dostupné z: https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/

[13] FLORES, Anna. TOP FIVE USE CASES OF TENSORFLOW [online]. [cit. 18.7.2019] Dostupné z: https://www.linkedin.com/pulse/top-five-use-cases-tensorflow-deep-learning-anna-flores

[14] WINDER, Simon A. J. Training neural nets on MNIST digits [online]. [cit. 18.7.2019] Dostupné z: http://simonwinder.com/2015/07/training-neural-nets-on-mnist-digits/

[15] TENSORFLOW. MNIST For ML Biginners [online]. [cit. 3.10.2018] Dostupné z: https://www.tensorflow.org/versions/r1.2/get_started/mnist/beginners

[16] ČÍŽEK, Jakub. Hrátky s „umělou inteligencí“ od Googlu: zkusil jsem vytvořit kopiui sebe samotného [online]. [cit. 18.7.2019] Dostupné z: https://www.zive.cz/clanky/hratky-s-umelou-inteligenci-od-googlu-zkusil-jsem-vytvorit-kopii-sebe-samotneho/sc-3-a-184986/default.aspx

[17] KERAS. Keras: The Python Deep Learning library [online]. [cit. 18.7.2019] Dostupné z: https://keras.io/

[18] WIKIPEDIA. Keras [online]. [cit. 18.7.2019] Dostupné z: https://en.wikipedia.org/wiki/Keras

[19] KERAS. About Keras models [online]. [cit. 18.7.2019] Dostupné z: https://keras.io/models/about-keras-models/

[20] KERAS. Getting started with the Keras Sequential model [online]. [cit. 18.7.2019] Dostupné z: https://keras.io/getting-started/sequential-model-guide/

[21] KERAS. Getting started with the Keras functional API [online]. [cit 18.7.2019] Dostupné z: https://keras.io/getting-started/functional-api-guide/

[22] BILOGUR, Aleksey. Keras optimizers [online]. [cit 18.7.2019] Dostupné z: https://www.kaggle.com/residentmario/keras-optimizers

[23] MACK, David. How to pick the best learning rate for your machine learning project [online]. [cit. 18.7.2019] Dostupné z: https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

[24] BROWNLEE, Jason. How to Choose Loss Functions When Training Deep LEarning Neural Networks [online]. [cit. 18.7.2019] Dostupné z: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[25] MISSINGLINK.AI. 7 Typer of Neural Network Activation Functions: How to Choose? [online]. [cit. 18.7.2019] Dostupné z: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[26] WALIA, Anish Singht. Activation functions and it's types-Which is better? [online]. [cit. 18.7.2019] Dostupné z: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[27] GOZZOLI, Alessio. Practical guide to hyperparameters search for deep learning models [online]. [cit. 18.7.2019] Dostupné z: https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

[28] MIKKO. Hyperparameter Optimatization with Keras [online]. [cit. 18.7.2019] Dostupné z: https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53

[29] KHANDELWAL, Renu. Overview of different Optimizers for neural networks [online]. [cit 18.7.2019] Dostupné z: https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3

[30] KERAS. How can I obtain reproducible results using Keras during developement? [online]. [cit 18.7.2019] Dostupné z: https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development

[31] DEEPLIZARD. Reproducible results with Keras [online]. [cit 18.7.2019] Dostupné z: https://deeplizard.com/learn/video/HcW0DeWRggs

[32] BROWNLEE, Jason. Loss and Loss Functions for Training Deep Learning Neural Networks [online]. [cit 1.8.2019] Dostupné z: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[33] ALGORITHMIA. Introduction to Loss Functions [online]. [cit 1.8.2019] Dostupné z: https://blog.algorithmia.com/introduction-to-loss-functions/

[34] KRIZ, Pavel, MALY, Filip, KOZEL, Tomas. Improving Indoor Localization Using Bluetooth Low Energy Beacons [online]. [cit 1.8.2019] Dostupné z: https://www.hindawi.com/journals/misy/2016/2083094/abs/

PŘÍLOHA Č.1 – OBSAH PŘILOŽENÉHO CDROM

- bp – .doc soubor s bakalářskou prací
- vstupní data – vstupní nezpracovaná data
- výstupní data – výstupní zpracovaná data, tabulky, grafy, výsledky sítí
- zdrojové kódy – zdrojové kódy použitých programů

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Daniel Tesař**
Studium:	I1800686
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Úvod do TensorFlow a jeho aplikace v indoor lokalizaci**
Název bakalářské práce AJ:	Introduction to TensorFlow and its Applications in Indoor Localization

Cíl, metody, literatura, předpoklady:

Cíl: Seznámit se s knihovnou TensorFlow a nadstavbou Keras pro implementaci umělých neuronových sítí. Popsat typické případy užití včetně praktických příkladů. Navrhnout, implementovat o ověřit nasazení v řešení problému rádiové indoor lokalizace. Osnova: 1. Úvod 2. Cíl 3. Biologický neuron a neuronová síť 4. Umělý neuron a umělá neuronová síť 5. Úvod do TensorFlow 6. Nadstavba Keras 7. Návrh a implementace aplikace pro lokalizaci 8. Testování a výsledky 9. Závěr

1. TENSORFLOW, 2017. MNIST For ML Biginners [online]. Dostupné z: https://www.tensorflow.org/versions/r1.2/get_started/mnist/beginners 2. Galushkin, A.: Neural networks theory 3. Ripley, B.: Pattern Recognition and neural Networks

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Pavel Kříž, Ph.D.

Datum zadání závěrečné práce: 14.1.2018