FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Analýza zvuku pomocí rekurentních neuronových sítí
Student:	Marie Dresslerová
Vedoucí:	Ing. Michaela Vondráková
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je navrhnout řešení pro průběžné sledování stavu zařízení pomocí analýzy zvuku. Toto sledování má umožnit včasný zásah odborníka, díky kterému se zařízení včas opraví/vymění a tím se zamezí větším škodám, nebo ztrátě dat při nečekané poruše. Ke sledování stavu zařízení bude použit zvukový záznam vybraného zařízení v provozu a rekurentní neuronová síť.

1. Seznamte se a nastudujte knihovnu TensorFlow pro strojové učení, knihovny pro zpracování zvuku a typy rekurentních neuronových sítí (RNN), následně vyberte vhodný typ knihovny pro zpracování zvuku a typ RNN pro analýzu zvuku nahraného toku dat.
2. Navrhněte řešení pro predikci budoucího stavu vybraného zařízení s pomocí nahraných zvukových stop.
3. Vybrané řešení implementujte v jazyce Python za použití knihovny TensorFlow.
4. Vytvořte literární rešerži zabývající se popsaným problémem a postupem řešení, které bylo zvoleno.
5. Zhodnoťte dosažené výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 31. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Analýza zvuku pomocí Rekurentní neuronové sítě

Marie Dresslerová

Katedra teoretické informatiky
Vedoucí práce: Ing. Michaela Vondráková

14. května 2018

Poděkování

Ráda bych poděkovala své vedoucí za její ochotu s jakou se mnou práci řešila, a kolegovi Jurajovi Kundrikovi za jeho konzultace. Též je třeba zmínit mé kolegyně, které mi pomáhaly s přípravou dat pro neuronovou síť a v neposlední řadě je třeba poděkovat mým rodičům, kteří mi obětovali celou místnost na měření po dobu osmi měsíců.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Dresslerová, Marie. *Analýza zvuku pomocí Rekurentní neuronové sítě*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá sběrem dat a následnou analýzou zvuku pevných disků a dále návrhem rekurentní neuronové sítě pro jejich hlubší analýzu, která pak dokáže předvídat budoucí stav stroje. Je zde popsáno jak lze data získat, jak je zpracovat pro potřeby této neuronové sítě a zároveň jak takovou síť navrhnout s pomocí knihovny TensorFlow v jazyce Python.

Klíčová slova Rekurentní neuronová síť, TensorFlow, LibROSA, Deep learning, LSTM, Analýza zvuku

Abstract

This thesis has taken up the task of the hard drive's sound data collection and analysis, with subsequent design of a recurrent neural network for a more detailed data analysis. Such network aims to be able to predict future state of a machine. In this paper it is described how to gather data, how to process those data for the neural network needs and how to design such a network with the TensroFlow library in the Python language.

Keywords Recurrent neural network, TensorFlow, LibROSA, Deep learning, LSTM, Sound analysis

Obsah

Seznam obrázků

Úvod

Stroje nám už dlouhou dobu usnadňují život tím, že po nás převzaly velkou část naší práce. Nicméně ani stroj není bezchybný a občas u něj dojde k poruše. V takovém případě je dobré odhalit poruchu ještě dříve, než nastane, ať už z důvodu předcházení ztráty dat, nebo zbytečnému zdržení při výkonu činnosti.

Firma Optisolutions s.r.o., pro kterou pracuji, se zabývá, mimo jiné, právě touto problematikou, a to je důvod mé volby tohoto tématu. Ačkoli se v Optisolutions zajímáme o širokou škálu strojů a zařízení, tato práce se bude vztahovat pouze na pevné disky, a to z důvodu jejich cenové dostupnosti a z toho vyplývajícího snadného sběru dat. [4]

Výstupem této práce bude predikce selhání pevného disku v dostatečném časovém předstihu, aby bylo možné jej zálohovat a bez rizika odstavit. Vzhledem k nasbíraným datům bude selhání disku představovat povětšinou jeho zaseknutí se, což nemusí ještě nutně vést k selhání, ale i přesto je to nežádoucí stav. Mezi další výstupy bakalářské práce, kromě predikce, patří také sledování aktuálního stavu, v jakém se pevný disk nachází.

V této práci se budu zabývat jak sběrem zvukových nahrávek pevných disků, tak samotným návrhem rekurentní neuronové sítě (RNN) pro jejich analýzu a následnou predikci budoucího stavu zařízení.

Kapitola 1

Cíl práce

Cílem práce je navrhnout řešení pro průběžné sledování stavu zařízení pomocí analýzy zvuku. Toto sledování má umožnit včasný zásah odborníka, díky kterému se zařízení včas opraví, či vymění, a tím se zamezí větším škodám, nebo ztrátě dat při nečekané poruše. Ke sledování stavu zařízení bude použit zvukový záznam vybraného zařízení v provozu a rekurentní neuronová síť.

Kapitola 2

Rešerše

Tato kapitola se zaměřuje na objasnění pojmů a systémů, které s touto prací souvisí. V první části zmíním již existující řešení daného problému, poté vysvětlím princip neuronových sítí, dále se zaměřím na zvuk a také objasním, co je to vlastně TensorFlow.

2.1 Existující řešení predikce stavu zařízení

Analýza zvuku je poměrně běžná záležitost. Existuje nepřeberné množství aplikací, které jsou schopné zvuk rozpoznat. Existují i systémy, které se zabývají přímo mým problémem. Například 3DSignals používá umělou inteligenci k predikci poruchy zařízení na základě jeho zvuku. [5]

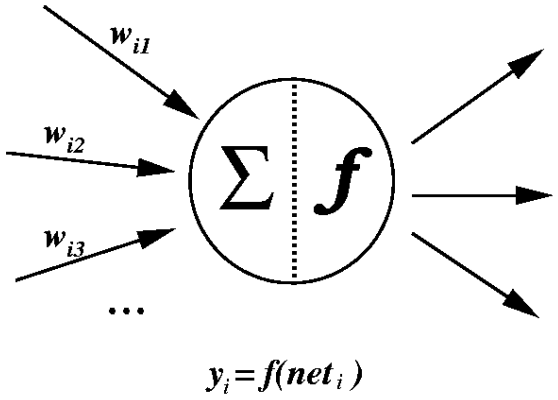
Nedávno se objevila i česká firma, která řeší můj problém podobně jako 3DSignals, a tou je Neuron Soundware. [6]

Tato řešení však nejsou zdarma a jejich integrace s našimi firemními systémy nemusí být jednoduchá, už jen kvůli podpoře, kterou bychom našim zákazníkům nemohli poskytnout přímo.

2.2 Umělá neuronová síť

Umělé neuronové sítě dnes mají velmi široké využití pro svou univerzálnost, kterou podědily po své předloze, tedy po biologické nervové soustavě. Schopnost našeho mozku se učit je dána právě onou nervovou soustavou, především neurony, které ji tvoří. Umělou neuronovou síť tvoří neurony, které se pokoušejí chování svých biologických předloh napodobit.

Biologický neuron se, mimo jiné, skládá z několika dendridů a právě jednoho axonu. Dendridy slouží k přijímání signálů, axon k jeho odeslání. Neuron příchozí signály sečte a pokud jejich hodnota překročí jistou hranici, vytvoří napětí, které se šíří axonem k dalším neuronům.



Obrázek 2.1: Model umělého neuronu, $net_i = \sum_{j=1}^{N} \omega_{ij} y_j$ [1]

Umělý neuron funguje na stejném principu. I on přijímá libovolný počet signálů a odesílá právě jeden. Vstupy neuronu mají svou *synaptickou váhu*, což je číselné ohodnocení naznačující, jestli signál přijatý tímto vstupem přispěje ke správnému výsledku. Právě váha vstupu je to, co se v průběhu učení mění. Příchozí signály jsou násobeny hodnotou váhy vstupu, kterým přišly. K součtu těchto násobků je poté přičtena ještě hodnota vychýlení a to je následně posláno do *aktivační funkce*, jejíž výstup je výsledným výstupem neuronu.

Tento model neuronu se nazývá McCulloch-Pittsův a lze formálně popsat takto:

$$Y = S(\sum_{i=1}^{N} \omega_i x_i + \Theta) \tag{2.1}$$

Kde:

- x_i jsou vstupy (signály) neuronu
- ω jsou synaptické váhy
- Θ je hodnota vychýlení, nebo-li práh
- S(x) je aktivační funkce neuronu
- Y je výstup neuronu

Umělé neuronové sítě se mohou učit dvěma způsoby. Prvním je tzv. učení se s učitelem, druhé učení se bez učitele.

2.2.1 Učení s učitelem

Toto učení je celkem běžné. Jde o učení se zpětnou vazbou, což lze opět ukázat na našem mozku. Pokud v zimě olíznu železnou tyč, přimrzne mi k ní jazyk

a já tak dostanu zpětnou vazbu. V tom okamžiku probíhá jistý vzruch mezi neurony v mém mozku a dochází tak k poučení o tom, že lízat v zimě železnou tyč není dobrý nápad.

Při učení s učitelem umělá neuronová síť, tak jako můj mozek, došla k nějakému výsledku, ke kterému následně dostane zpětnou vazbu, jestli je správný, nebo špatný. Toto se pak opakuje na dostatečném množství dat, kterým se říká trénovací. Velmi zjednodušeně nechávám umělou neuronovou síť olizovat různé předměty při různých teplotách, abych ji naučila, že lízat kov v zimě je špatné, ale při běžné teplotě to ničemu nevadí.

Takové učení obvykle vyžaduje velké množství dat. Ty je potřeba rozdělit na data trénovací a testovací. Jak bylo zmíněno výše, trénovací data slouží k učení sítě, zatímco na datech testovacích zkoušíme, jak dobře síť naučená je.

Testovací data jsou zpravidla jiná, než trénovací, což je vcelku logické, neboť se dá očekávat, že na trénovacích datech, které by jí byly předloženy k testování, si síť povede dobře a vrátí relativně přesné výsledky. Jinými slovy nám takový test vůbec nic neřekne o tom, jak dobře síť natrénovaná je.

Dělení dat se dá řešit všelijak. Můžeme data rozdělit na několik dílů, kdy v prvním kroku síť natrénujeme na první části a na zbytku budeme testovat. Ve druhém kroku ji budeme trénovat na druhé části a na zbytku testovat a tak budeme pokračovat, dokud síť nebude natrénovaná podle našich potřeb.

Trénování sítě s sebou nese ještě jeden problém, a to ten, aby se nepřetrénovala. Přetrénovaná síť postrádá svou univerzálnost. Vrací přesné výsledky na specifických datech, ale vůbec nezvládá data nová. Aby se tomuto stavu předešlo, je potřeba volit správná trénovací data, která zahrnou pokud možno všechny možné situace. Trénovací data tedy musí být dostatečně velká na to, aby obsahovala co nejvíce situací, ale zároveň se to s nimi nesmí přehnat, aby síť nebyla až příliš specializovaná.

2.2.2 Učení bez učitele

Učení bez učitele je přesný opak předchozí metody, kde není žádná zpětná vazba. Tato metoda obvykle využívá evoluční algoritmy, což lze velmi zjednodušeně přirovnat ke klasické evoluci, kdy dochází ke křížení jedinců, mutacím a podobně. V případě neuronových sítí je tento přístup označován za neuroevoluci a v principu funguje tak, že existuje populace neuronových sítí, které se učí na stejnou věc, načež dochází ke křížení mezi sítěmi, které se chovají nejlépe. S každým křížením vzniká nová generace, přičemž v poslední generaci by měly být nejlépe natrénované neuronové sítě.

2.2.3 Vícevrstvá neuronová síť

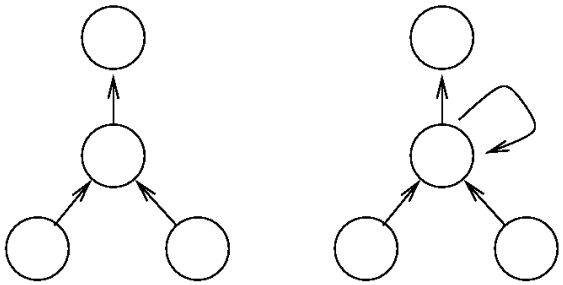
Vrstvy jsou jedním z nejběžnějších a nejefektivnějších způsobů propojení neuronů. Vrstva není nic jiného, než množina neuronů, které přijímají signály

a posílají je dál. Zároveň však představuje nějakou úroveň. Obvyklé propojení vrstev vypadá tak, že výstup každého neuronu z nižší vrstvy tvoří vstup každého neuronu vyšší vrstvy. Tomu se říká *úplné propojení neuronů*. Mámeli vícevrstvou neuronovou síť, pak počáteční vstupní hodnoty dostává vždy nejnižší vrstva, které se proto říká *vstupní* a naopak konečný výsledek vrací vrstva nejvyšší, *výstupní*.[7]

2.2.4 Rekurentní neuronová síť

Obvykle bývají umělé neuronové sítě dopředné, to znamená, že signály proudí napříč vrstvami jedním směrem vždy od nejnižší vrstvy k nejvyšší, viz obrázek 2.2. Rekurentní sítě toto omezení nemají a díky tomu jsou schopné uchovat informaci. Dá se zde mluvit o paměti, ale je nutno dodat, že se jedná o paměť velmi krátkodobou. Další vadou na kráse RNN je jejich obtížné trénování. [2]

Problém paměti a trénování RNN se dá vyřešit pomocí již existujících architektur. Velmi oblíbenou a používanou je Long Short-Term Memory (LSTM), naopak relativně novou architekturou je Echo State Networks (ESN).

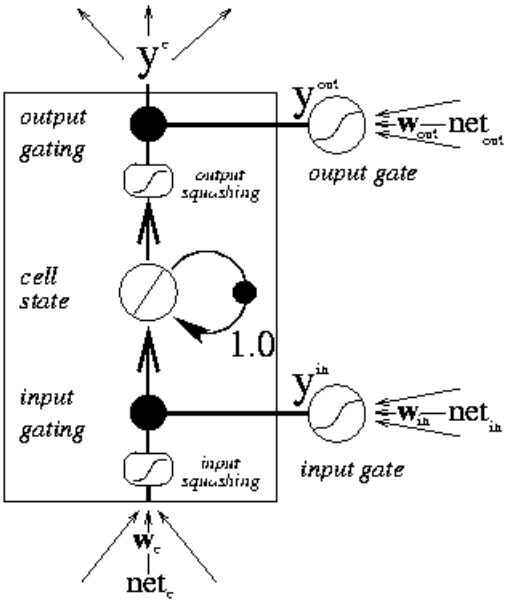


Obrázek 2.2: Ukázka dopředné sítě (vlevo) a rekurentní (vpravo) [2]

2.2.5 Long Short-Term Memory

Long Short-Term Memory (LSTM) síť se dá natrénovat vcelku efektivně a zároveň řeší i problém s krátkodobou pamětí. Tuto síť tvoří *LSTM paměťové bloky*, jejichž prací je právě uchovávání informace po libovolný časový úsek.

Každý paměťový blok obsahuje vstupní a výstupní bránu, vedle toho však má také klasický vstup a výstup, jako tomu bylo u neuronu. Informace do bloku tedy proudí jak vstupem bloku, tak přes vstupní i výstupní bránu. Ač to tak podle názvu nevypadá, výstupní brána neslouží k předávání dat, ale k přijímání výstupních dat jiných bloků. Výstup bloku je, stejně jako u neuronu, právě jeden.



Obrázek 2.3: Blok LSTM sítě [3]

Další důležitou složkou bloku jsou *paměťové buňky*. Na obrázku 2.3 je paměťový blok s jednou paměťovou buňkou. Tou je jednoduchý neuron, který má sám na sebe rekurentní vazbu s váhou nastavenou na 1. Díky tomu je buňka schopná udržet svůj aktuální stav v případě, že neobdrží žádný vstup. [3]

2.2.6 Echo State Networks

Jak již bylo zmíněno, tento návrh sítě je relativně nový. Jeho velkým pozitivem je rychlost, jakou jsou v rámci sítě prováděny výpočty. To je dáno především řídkým propojením mezi neurony, což znamená, že neurony nejsou mezi vrstvami propojené každý s každým, jak je tomu zvykem u ostatních sítí. [8]

2.3 Zvukové formáty

V dnešní době lze nalézt několik zvukových formátů, jejichž kvalita je určena frekvencí, v jaké zvuk ukládají, a ve volbě komprese.

Hned ze začátku je třeba říct, že frekvence uložené nahrávky nijak nesouvisí se zvukovou frekvencí. Je potřeba si uvědomit, že zvuk je spojitý signál, což nelze moc dobře uložit a je tedy potřeba z tohoto signálu udělat signál

diskrétní. Frekvence nahrávky nám říká, jak jemné škálování bylo použito. Platí, že čím je frekvence vyšší, tím je škálování jemnější a zvuk je kvalitnější.

Diskrétním hodnotám zvuku říkáme rámy (angl. *frames*). Při frekvenci 8 000 Hz, což je jedna z nejnižších, můžeme říct, že 8 rámů popisuje jednu tisícinu vteřiny. Více kvalitnější záznam s frekvencí 44 100 Hz popisuje stejný časový úsek 441 rámy. I zde je tedy vidět, jakým způsobem frekvence ovlivňuje kvalitu nahrávky.

Nabízí se otázka, k čemu vlastně jsou nahrávky s nízkou frekvencí. Předchozí příklad nám ovšem dává jasnou odpověď. Čím je vyšší frekvence, tím je větší množství rámů, které je potřeba uložit. Jinými slovy kvalita nahrávky stojí místo na disku, a je tedy otázkou, kdy se to vyplatí a kdy je to zbytečné plýtvání.

Někdy je ovšem velmi žádoucí mít kvalitní nahrávku, která ale zabírá méně místa. To jsou požadavky, které se kladou především na nahrávky hudební. Existují služby, umožňující poslech nepřeberného množství hudby, která dokonce může být stažena na zařízení uživatele a přesto nezabírá mnoho místa, jakkoli je kvalitní.

Toto umožňuje právě komprese, která může být buď ztrátová, nebo bezeztrátová.

Ztrátová komprese používá algoritmy, díky kterým dojde ke snížení množství uložených dat, ale nelze je už zpětně obnovit do původního stavu. Dochází tedy ke ztrátám, které mají negativní dopad na kvalitu zvuku.

Bezeztrátová komprese je naopak ta, která umožňuje mít kvalitní nahrávky s menší velikostí dat. Algoritmy, které využívá, mají inverzní varianty, které dokáží data obnovit do původního stavu beze ztrát.

Znovu tedy vyvstává otázka, k čemu dělat nahrávky s nízkou frekvencí, když existuje bezeztrátová komprese. Tato odpověď už nemusí být tak přesvědčivá, ale jsou případy, kdy komprese jen znepříjemňuje práci přidáváním výpočtů, které je potřeba provést navíc. Aneb někdy zkrátka stačí 8 rámů na tisícinu vteřiny.

Zvukové nahrávky mohou být uloženy v několika formátech. Liší se ve způsobu ukládání dat, tedy i ve výsledné kvalitě a velikosti souboru.

2.3.1 Formát MP3

Pro hudbu je nejčastější formát MP3, který už v sobě obsahuje ztrátovou kompresi, čímž zajišťuje malou velikost výsledného souboru. Nejvyšší využití má v hudbě, kde i přes ztrátovou kompresi zachovává poměrně vysokou kvalitu zvuku. U ostatních typů zvuku je už ale výsledek horší.

2.3.2 Formát WAV

Formát WAV sám o sobě žádnou kompresi nepoužívá, což ovšem neznamená, že ji nemůže mít. Záleží na autorovi nahrávky, jak zvuk do WAVu uloží.

Nejčastějším formátem dat ve WAV je *lineární pulzně kódovaná modulace*, která je bezeztrátová a výpočetně nenáročná. To jsou vlastnosti, které jsou oblíbené pro práci se zvukem.

2.4 Knihovny Pythonu pro práci se zvuky

Dojde-li na práci se zvukem v jazyce Python, jsou dvě možnosti, jakou knihovnu si zvolit. Tyto dvě možnosti jsou vybrány na základě dokumentace obou knihoven a podpory komunity.

2.4.1 Python/wave

Python/wave patří do standardní knihovny Pythonu. Jak už název napovídá, tato knihovna je určena výhradně pro nahrávky formátu WAV. Omezuje se na základní práci, jako je čtení a ukládání souboru, získání informací o nahrávce, což znamená velikost frekvence, počet rámů a samozřejmě i rámy samotné. Jedná se tedy o základní nástroj pro práci se surovými daty. [9]

2.4.2 LibROSA

LibROSA už je skutečným nástrojem pro analýzu zvuku. Podporuje několik formátů, mezi nimiž je MP3 i WAV, a obsahuje nepřeberné množství funkcí pro práci s nimi.

Mezi velké výhody LibROSY patří tvorba grafických znázornění zvuku, jako je třeba spektrogram, jednoduché získání specifických údajů o zvuku, jako je jeho hlasitost, nebo výška. Pokud to nahrávka umožňuje, LibROSA dokáže identifikovat i jednotlivé tóny.

Co je ovšem také důležité zmínit je implementace *Krátkodobé Fourierovy tranformace (STFT)*, která provádí časově-frekvenční analýzu a díky které mohu dostat reprezentaci vybraného časového úseku jediným číslem.[10]

2.5 TensorFlow

TensorFlow je knihovna pro Python, která slouží k provádění složitých numerických výpočtů. Tyto výpočty je navíc schopná provádět jak na procesoru (CPU), tak na grafické kartě (GPU), jejíž výpočetní výkon je znatelně vyšší.

Hlavní rozdíl mezi výpočtem v TensorFlow a výpočtem v klasickém Pythonu je ten, že v Pythonu, nebo ve většině programovacích jazyků, se výraz $x = a + 1$ vyhodnotí okamžitě. V TensorFlow ovšem nejprve popíšeme náš

početní problém jako abstraktní model a výpočet zahájíme až tehdy, když je zcela popsán.

K popisu modelu slouží *tensory*. Ty si lze představit jako vektory. Při popisu modelu vytváříme *výpočetní graf*, kde uzly představují tensory a hrany vztahy mezi nimi.

Díky tomu, že má TensorFlow při výpočtu kompletní přehled o tom, co vlastně počítám, dokáže mé výpočty optimalizovat a tím problém vyřešit efektivně.

Všechny tyto vlastnosti dělají z TensorFlow mocný nástroj pro strojové učení. [11]

Kapitola 3

Analýza a návrh

V této kapitole ještě blíže představím naši firmu a poté popíši způsob, jakým jsem se rozhodla řešit daný problém, a zdůvodním zde má rozhodnutí ohledně volby technologií.

3.1 Společnost Optisolutions

Bakalářská práce vznikla pro potřeby společnosti Optisolutions s.r.o., která se specializuje na praktické využití umělé inteligence v reálných provozech, a to především ve strojním průmyslu. Neuronové sítě jsou zde tedy zcela běžnou záležitostí, i když práce se zvukem je u nás novinkou.

Naše stávající sítě pracují s obrázky, kde není potřeba časový kontext, jako u zvuku. Jejich výsledky se ale mohou využít různými způsoby, ať už jde o spuštění alarmu, nalezení středu hlavně, či určení akce pro robotickou ruku, Karla.

Dokážeme rozpoznat rozbité zařízení, či „zmetky“ na základě jeho vzhledu, ovšem poznat jej podle zvuku, nebo tento stav jen předpovědět, to je něco, co nám chybí a kvůli čemu vznikla tato práce.

3.2 Možnosti řešení

Můj problém se dá rozložit do čtyř částí, z nichž každá se dá řešit několika způsoby. První částí je získávání dat z pevných disků, druhá jejich formát, poté jejich zpracování a nakonec návrh samotné sítě.

3.2.1 Sběr dat

Při získávání dat existují ve výsledku jen dvě možnosti. Najít databanku, která obsahuje potřebná data, nebo si je vytvořit.

3.2.1.1 Převzatá data

Na internetu lze najít velké množství nahrávek pevných disků od různých výrobců se všemi možnými typy poškození. Nevýhoda těchto dat je v tom, že bývají různě dlouhá a především nepopisují stav disku v časovém kontextu. Nedají mi informaci o tom, jak se disk choval před tímto chybným stavem, a tím pádem jsou pro mě vcelku bezcenná.

3.2.1.2 Vlastní data

Pokud budu nahrávat pevné disky sama, mohu si zajistit stejnou délku a formát nahrávky. Především ale budu mít na sebe navazující data popisující chování disku v průběhu jeho života.

Nevýhodou tohoto řešení je ovšem to, že nikdy nevím, kdy se disk dostane do chybového stavu, či jestli se do něj někdy dostane. Čekání na tuto událost tak může trvat příliš dlouhou dobu.

Vzhledem k tomu, že je pro mne časový kontext nezbytně důležitý, vytváření vlastních dat se tedy stalo jedinou možnou volbou i s vědomím toho, že docílit chybového stavu nebude jednoduché.

3.2.2 Formát zvuku

Když jde o zvuk, mám možnost vybírat z několika zvukových formátů, z čehož o dvou jsem se už stačila zmínit. Vedle WAV a MP3 se lze setkat s několika dalšími formáty, jako jsou například WMA (Windows Media Audio), nebo OGG. Jelikož ale, stejně jako MP3, využívají ztrátovou kompresi, je pro mě zbytečné se jimi dál zabývat.

3.2.2.1 MP3

Jak již bylo několikrát zmíněno výše, MP3 používá ztrátovou kompresi, díky které zabírá menší místo na disku. Tato komprese funguje dobře pro hudební nahrávky, ale u mluveného slova už mívá problém.

Přestože zvuk pevného disku není mluvené slovo, není to ani hudba a tedy nemám jistotu, že ztráta z komprese nebude příliš znatelná.

3.2.2.2 WAV

WAV sám o sobě žádnou kompresi nemá, což ovšem znamená, že bude zabírat více místa. Nicméně způsob, jakým je ukládán, je natolik jednoduchý, že v případě, kdy by bylo potřeba nad ním provádět nějaké složitější operace, by nemusel nastat žádný větší problém.

Pokud navíc nebudu lpět na vysoké frekvenci nahrávky, nemusí být ani velikost onoho souboru zase tak obrovská.

3.2.3 Zpracování zvuku

Pokud mám nahrávky ve formátu WAV, mám dvě možnosti, jakou knihovnu v Pythonu zvolím pro práci se zvukem. Python/wave, nebo LibROSA.

3.2.3.1 Python/wave

Tuto knihovnu už jsem dříve představila, proto jen ve stručnosti zopakuji, že se jedná o standardní knihovnu Pythonu, která slouží k základním operacím, jako je čtení a zápis.

Jistě by šlo využít tuto knihovnu a poté za pomocí numerické knihovny NumPy zpracovat data tak, jak budu potřebovat. Otázkou je, jestli to nejde snáz.

3.2.3.2 LibROSA

LibROSA je komplexní knihovna pro Python, která kromě čtení a zápisu souboru dokáže data zpracovat různými způsoby. Pro mě to znamená, že zahrnuje vše, co umí Python/wave, a zároveň implementuje funkce a algoritmy, které bych později mohla chtít využít.

Z tohoto důvodu jsem si ji vybrala pro práci se zvukovými nahrávkami, neboť svým širokým záběrem mi může výrazně usnadnit práci.

3.2.4 Nástroj pro návrh neuronové sítě

Ze zadání práce pro mě jasně vyplývá, že pro návrh sítě musím použít TensorFlow. To však ale neznamená, že nemohu použít knihovnu, která TensorFlow používá. Jednou takovou je knihovna Keras, kterou již využíváme pro projekty ve firmě.

3.2.4.1 TensorFlow

TensorFlow zcela jednoznačně umožňuje vytvořit mnou navrženou síť, je ovšem otázkou, jestli by takto „ručně“ vytvořená síť, vzhledem k mým zkušenostem, byla stejně efektivní, jako ta, kterou by vytvořil Keras.

Co je ovšem jisté, že tento ruční návrh by byl rozhodně komplikovanější a náročnější, než návrh v Kerasu.

3.2.4.2 Keras

Keras je API pro tvorbu neuronových sítí v Pythonu. Může běžet hned na třech výpočetních knihovnách, a to TensorFlow, CNTK, nebo Theano. Pro mě je podstatné, že může využívat právě TensorFlow.

Návrh sítě v Kerasu funguje jako skládání jednotlivých vrstev dohromady. Nejprve se definuje model, který určuje, jakým způsobem se vrstvy

budou skládat za sebe, poté do modelu přidáváme jednotlivé vrstvy tak, jak potřebujeme.

U každé vrstvy lze nastavit aktivační funkci nebo počet výstupů, což znamená počet neuronů, nebo u LSTM vrstev počet paměťových bloků. Tím pádem jsem schopná s Kerasem jednoduše navrhnout neuronovou síť přesně podle mých představ.

Další výhodou Kerasu je skutečnost, že naše dosavadní sítě ve firmě byly navrhované právě v něm, což jsou cenné zkušenosti, které mohou být aplikované při řešení mého problému.

Po dohodě s mou vedoucí práce jsem se rozhodla pro využití Kerasu, neboť pro vypracování zadání čistě v TensorFlow nebyl dostatečný časový prostor.

3.2.5 Architektura RNN

Už jsem zmiňovala dvě architektury RNN, které by připadaly v úvahu, a to Long Short-Term Memory a Echo State Networks.

3.2.5.1 Long Short-Term Memory

LSTM patří mezi nejpoužívanější architekturu RNN. Snadno se učí a Keras ji implementuje jak v podobě vrstvy (sítě), tak i jako jednotlivý blok. Navíc je osvědčeným řešením pro problémy vyžadující po neuronové síti paměť, což je něco, co moje síť nezbytně potřebuje, protože chci-li něco předvídat, pak si musím nutně pamatovat to, co předcházelo.

3.2.5.2 Echo State Networks

EST je výpočetně méně náročná než LSTM a hodí se dokonce i pro predikci časových řad, ale zase pro ni neexistuje přímá implementace v Kerasu, což mi značně ztěžuje práci.

Bylo by určitě zajímavé vyzkoušet, jak by si tato síť vedla na mém problému, ale vzhledem k omezeným časovým možnostem jsem se rozhodla zůstat u osvědčené a implementované LSTM sítě.

3.3 Predikce a Klasifikace

V této práci řeším vlastně dva problémy. Potřebuji predikovat chování pevného disku a toto chování potřebuji klasifikovat, aneb označit jej za dobré, nebo špatné.

Řešit predikci a klasifikaci jako jeden problém by sice nejspíš šlo, ale bylo by to velice náročné a chtělo by to více dat a více času, než pro tuto práci mohu mít.

Samozřejmě, pokud bych dokázala určit, kdy se teprve schyluje k poruše, stačila by mi k řešení mého problému pouze kasifikační síť. Tento stav ovšem

neznám, neboť když pevný disk začne vydávat špatné zvuky, které jsem schopná rozpoznat, už je většinou něco špatně. Přestože u pevného disku ještě v tomto momentě mám obvykle dost času na provedení zálohy, u jiného zařízení tomu tak být nemusí. Z tohoto důvodu jsem zvolila řešení se dvěmi neuronovými sítěmi, kdy první predikuje chování a druhá jej klasifikuje.

Toto řešení je velice jednoduché, respektive nejtěžší na něm je pouze správné navržení sítí a především správně zpracovaná a rozvržená data. Poté je více než snadné natrénovat první síť, aby predikovala krátký časový úsek, a druhou síť, aby tento úsek klasifikovala.

3.4 Predikovaný stav

Než se začnou data pro neuronové sítě zpracovávat, je potřeba si říct, jaký stav chci predikovat, aneb co je chyba a co ne.

Toto rozhodnutí se odvíjí od dat, která mám k dispozici. Jak jsem zmínila výše, přimět disk, aby se porouchal bez toho, aniž bych do něj bušila kladivem, není vůbec jednoduché. Dá se předpokládat, že v praxi budu mít dostatečné množství dat k tomu, abych mohla neuronové sítě naučit na skutečnou poruchu. Pro mé potřeby ale stačí, aby byla schopná poznat libovolný specifický budoucí stav, ke kterému disk může dojít.

Tento stav je tedy potřeba volit na základě získaných dat, aneb podle toho, čeho se nám se zařízením povedlo docílit.

Kapitola 4

Realizace

Celou tuto práci lze rozdělit do několika fází. První fází je nahrávání disků, druhou zpracování těchto nahrávek a třetí je návrh neuronových sítí. V této kapitole vysvětlím, jakým způsobem jsem vše zpracovala.

4.1 Nahrávání disků

Prvním úkolem bylo získání co největšího počtu starých pevných disků. Takové disky mají větší sklony k poruchovosti, což byla vlastnost pro mě velice žádoucí. Kritéria pro měřící disky byla následující.

- **Stáří** - čím starší, tím lepší.
- **Funkčnost** - disk se musel roztočit a musel být schopný se připojit k zařízení, aby bylo možné jej zatěžovat. Disk, který se už buď netočil, nebo nepřipojil, byl „mrtvý“ a pro mou práci už neměl žádný význam.
- **Rozhraní** - abych byla schopná jej rozběhnout u mě doma, musel mít rozhraní, které jsem byla schopná připojit, což znamenalo buď SATA, nebo IDE

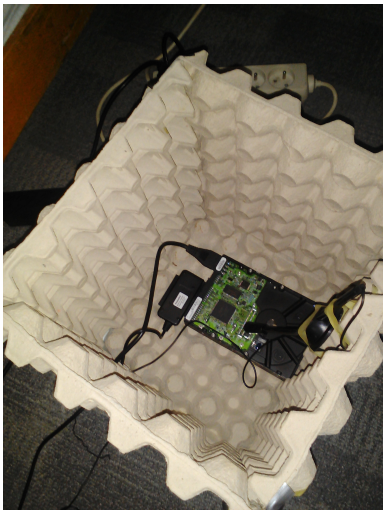
K jejich nahrávání jsem využila svůj stolní počítač a starší notebook, což mi umožnilo nahrávat dva disky zároveň. Na obou zařízeních byl systém Fedora, který mi umožnil jednoduše spouštět zátěžové programy a měřící skript co nejefektivněji.

Disky byly připojeny k zařízení externě, vždy jeden disk na jedno zařízení. Každé zařízení mohlo nahrávat jen jeden disk současně, protože k němu lze připojit pouze jeden mikrofon. Externí zapojení zajistilo, že nahrávka nebyla rušena zvuky systémového disku a zároveň to umožnilo disk „uzavřít“ do krabic z papírových plat na vajíčka.

Tyto krabice měly za úkol tlumit okolní zvuk a zajistit tak kvalitnější nahrávky, viz obrázek 4.1. Výhoda krabic byla i v tom, že zvuk disku zesilovaly.

Kromě disku byl v krabici připevněn i mikrofon, který byl připojen k měřícímu zařízení. Na tom běžel skript, který pouštěl program Alsa pro nahrávání 30 s zvukových záznamů. Záznamy byly ukládány ve formátu WAV bez komprese s frekvencí 8 000 Hz.

Tato frekvence byla zvolena z důvodu nižší velikosti výsledné nahrávky. Jak vysvětlím později, snížená kvalita zvuku daná touto frekvencí mi v tomto případě vůbec nevadila.



Obrázek 4.1: Jedna z měřících krabic z plat na vajíčka obsahující mikrofon a pevný disk

Dál bylo potřeba vymyslet způsob, jak disku pomoci k jeho poškození. Tato pomoc nesměla být mechanická, neboť by byla slyšet v nahrávce. Proto jsem musela přijít na to, jak disk poškodit skrze jisté operace.

Už z principu fungování pevných disků, kde máme pohyblivou hlavici a točící se plotýnky, je jasné, že pevným diskům nesvědčí náhodné přístupy, neboť to znamená více pohybu, než u přístupu k souvislým datům. Takovým generátorem náhodných přístupů může být například jeden z nejlepších řadících algoritumů, *QuickSort*.

Rozhodla jsem se tedy vytvořit program, který implementuje QuickSort trochu jinak, než jsme zvyklí. Zatímco běžně řadící algoritmy řadí data v operační paměti, můj QuickSort řadil data v souborech.

První část programu vygenerovala 300 000 souborů. V každém souboru bylo náhodné vygenerované číslo několikrát prokopírováno, aby byl soubor trochu větší. Soubory byly pojmenovány 0-299 999.txt. Toto číslo představovalo index každého souboru, čímž pádem jsem měla pole náhodných celých čísel o velikosti 300 000. Takové pole zabíralo na disku zhruba 10 GB, což je velikost, která se nevejde ani do operační paměti mých dvou zařízení, natož

do nějaké cache, což výrazně zvyšuje pravděpodobnost náhodného přístupu na disk.

Druhou částí programu už byl samotný QuickSort, který s každým svým průchodem přistupoval a zapisoval do náhodného souboru, dokud nebylo pole setříděné. Když se tak stalo, spustilo se další generování náhodných dat, a program tak běžel v nekonečné smyčce.

Abych diskům ještě trochu pomohla s jejich odchodem, svítila jsem na ně infralampou, čímž jsem je velmi rychle dostala mimo jejich komfortní teplotu. Nahřívání nesmělo trvat moc dlouho, neboť infralampa dosahuje vysokých teplot a roztavení elektroniky nebyl přímo ten typ poškození, o jaký jsem se pokoušela. Proto souběžně se zahříváním probíhala kontrola teploty disku a při dosažení 80 °C bylo zahřívání ukončeno. V případě disků, které teplotu nedovedly měřit, byl použit skromný odhad.

Tuto vysokou teplotu si disky byly schopny udržet po dlouhou dobu, neboť byly neustále vystaveny vysoké zátěži v přehřáté místnosti a s tepelně izolační vrstvou kolem krabic.

Chybový stav, kterého jsem tímto způsobem docílila, byl zaseknutí hlavice, což ještě nemusí nutně znamenat definitivní poškození disku, ale je to stav, který mohu zkusit předpovědět. Navíc, co mohu ze svých měření potvrdit, je to, že s opakovaným zahlcováním disku se disk vrací do funkčního stavu čím dál hůř, až se dostane do stavu, kdy se sám vypne. Tento stav už přináší otazník, neboť nemám nijak zaručeno, že se ještě někdy zapne.

4.2 Zpracování nahrávek

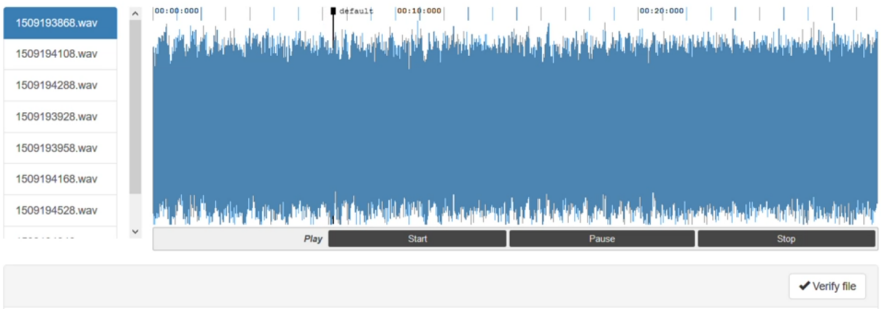
Tato činnost měla dvě fáze. První bylo označení chybových stavů, druhou samotné zpracování zvuku pro neuronovou síť.

4.2.1 Označení chybových stavů

Pro tuto činnost jsem rozšířila naši už stávající firemní aplikaci zvanou *Neurotylda*, kde moje kolegyně už dříve označovaly obrázky pro naše neuronové sítě. Jedná se o webovou aplikaci, kam se nahrávají soubory do datasetů a následně jsou v nich ručně označovány chybové stavy.

Mé rozšíření muselo zachovat všechny funkce, které byly možné u obrázků, což znamená možnost nahrání jednoho, či více souborů, jejich mazání, validaci, označení a úpravu označení.

Problém zvuků oproti obrázkům byl v tom, že zvuk se musí nějak vizualizovat. K tomu mi posloužila JavaScriptová knihovna Waves.js, díky které bylo možné v aplikaci zvuk přehrát a vizualizovat. Vizualizaci je možné přiblížit i oddálit. Pravým kliknutím je možné přidat označení chybového úseku, které je možné jak posouvat, tak zvětšovat, či zmenšovat. [12]



Obrázek 4.2: Ukázka vizualizace zvukové nahrávky v Neurotyldě

Každá nově nahraná nahrávka měla stav *Nevalidní*. Pokud by byl dataset s touto nahrávkou stažen a předhozen neuronové síti, nahrávka by byla ignorována, neboť buď ještě nebyla zpracovaná, nebo čekala na smazání.

To, jestli je nahrávka určena ke smazání, určoval samozřejmě její obsah. Občas se během nahrávání stalo, že se nahrávací zařízení zaseklo a nahrávka tím pádem byla „kousavá“. V praxi se to sice může stát, ale můj čas byl příliš omezený na to, abych učení neuronové sítě přizpůsobovala poškozeným datům, proto bylo třeba takové nahrávky smazat.

Dalším příkladem špatné nahrávky byl ruch okolí, který občas prošel skrze všechny izolační vrstvy. Takovým ruchem byl třeba štěkot psa za dveřmi, což je poměrně dost intenzivní zvuk na to, aby prošel skrze plata vajíček. Ze stejného důvodu jako u sekaných nahrávek jsem se rozhodla i tyto vyřadit.

Aplikace pro všechny případy skladuje všechny soubory, včetně těch smazaných. Pokud jsem se rozhodla stáhnout dataset, stáhla jsem si všechny tyto soubory spolu s jedním csv, který obsahoval všechny uložené informace, tedy původní název souboru, jestli je smazaný, jestli je validní a samozřejmě intervaly chybových úseků.

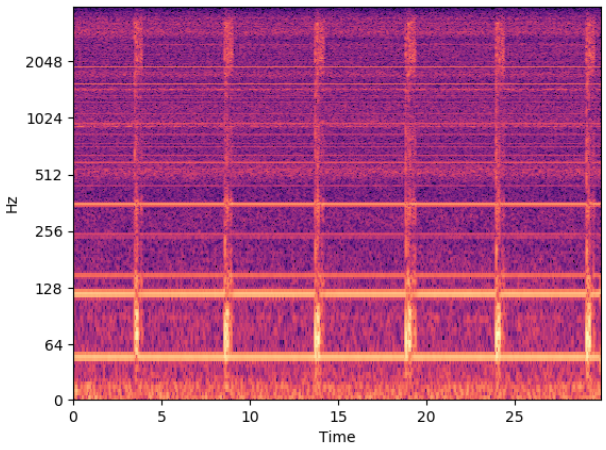
Chybovým stavem se v této práci rozumí zaseknutí hlavice. To vznikalo z důvodu zahlcení disku při čtení a zápisu. To se projevovalo jakýmsi „cvakáním“, jak se hlavice marně pokoušela dostat k místu určení. Nejedná se ovšem o běžné cvakání rozbitých disků, které je okamžitě patrné. Tento zvuk je sám o sobě zcela v pořádku, a tedy to, že jde o chybu, se pozná až jeho opakováním. Proto se za chybu považoval alespoň čtyřsekundový časový úsek, ze kterého bylo zřejmé, že se disk skutečně zaseknul a nevykonává žádnou produktivní činnost.

4.2.2 Zpracování zvuku

Dataset považuji za připravený v momentě, kdy jeho popisující csv nese název audio[index].csv, kde index označuje jeho pořadí v časovém kontextu, a ve složce audio jsou všechny validní nahrávky. Pokud jsou tyto podmínky splněny, je možné začít analyzovat data.

K analýze nahrávek jsem použila knihovnu LibROSA. Pomocí této knihovny jsem vždy načetla soubor a data z něj jsem nechala zpracovat Krátkodobou Fouriorovou transformací. Zde bylo potřeba se zamyslet, s jakou citlivostí transformaci nastavit, aneb jakou šířku budou mít jednotlivé STFT sloupce.

Následující spektrogramy reprezentují vždy jeden konkrétní soubor z trénovacích dat, který popisuje chybné chování disku. To lze spatřit v podobě světlých svislých čar, které představují zmíněné „cvaknutí“.

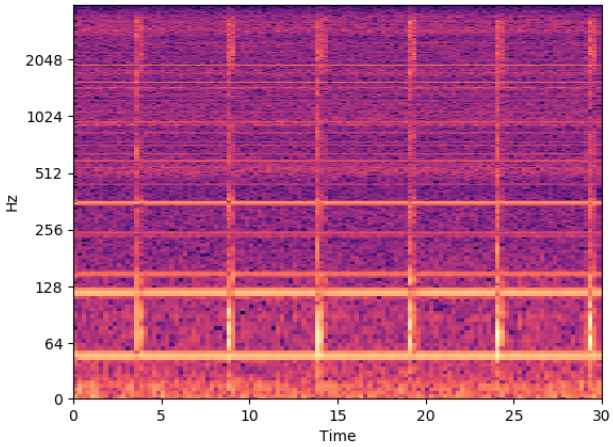


Obrázek 4.3: Spektrogram po zpracování STFT s šířkou sloupce 8 rámů (1 ms)

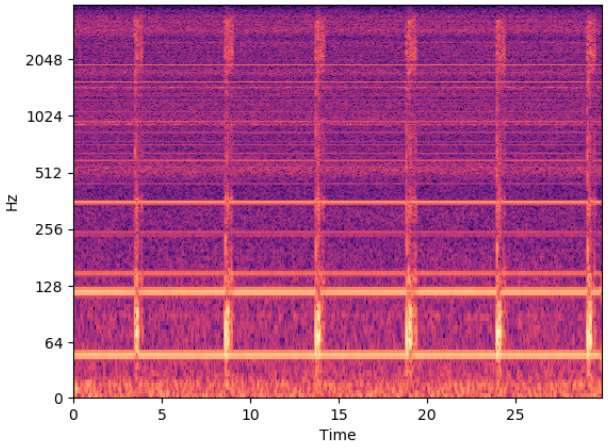
Čím menší šířka, tím přesnější data, viz obrázek 4.3, respektive tím větší množství dat, neboť pomocí STFT mohu data seskupit do většího časového úseku, na němž se neuronová síť může lépe učit. To znamená, že pokud budu mít úzké sloupce, budu mít větší množinu dat, což neuronové síti bude trvat déle na zpracování.

Varianta příliš širokých sloupců ovšem také není dobrá, neboť tím ztrácíme přesnost, respektive zobecňujeme data takovým způsobem, že z nich už nelze nic poznat, viz obrázek 4.4.

Najít vhodný kompromis znamenalo udělat pár pokusů, kdy nejrozumějšího výsledku v poměru kvalita/velikost jsem dosáhla s šířkou sloupce 80 rámů, viz obrázek 4.5. Jak je z obrázků vidět, mezi šířkou 8 rámů a 80 rámů není v kvalitě patrný skoro žádný rozdíl, aneb skutečně se jedná o dva rozdílné spektrogramy.



Obrázek 4.4: Ukázka dat po zpracování STFT s šířkou sloupce 2000 rámů (250 ms)



Obrázek 4.5: Ukázka dat po zpracování STFT s šířkou sloupce 80 rámů (10 ms)

Zrychlení zpracování těchto dat neuronovou sítí je však mnohem výraznější, což jen mluví ve prospěch šířky 80 rámů. A jak jsem dříve avizovala, proč mi nižší frekvence souboru nevadí, pak je toto má odpověď. Jelikož seskupuji data do menších částí, přičemž 10 ms není zas tak málo, pak mě nízká frekvence skutečně nemusí moc trápit, ba naopak.

Prohnáním dat STFT moje práce ovšem zdaleka nekončí, neboť tento formát ještě není zcela žádoucí. Funkce stft LibROSY mi totiž vrací pole

frekvencí a jejich sil napříč časem. Já ale potřebovala pole reprezentačních hodnot podle času, což znamená, že jsem vzala absolutní hodnotu průměru všech sil napříč frekvencí pro daný čas.

Takto jsem dostala hodnotu pro každých 10 ms reprezentující daný zvuk, a to už je něco, co se mohlo uložit pro neuronovou síť spolu s informací, jestli je daných 10 ms chybových nebo ne.

Jelikož jsem měla z csv datasetu seznam chybových intervalů a zároveň jsem věděla, v jakém čase se daná hodnota nachází, mohla jsem jí přiřadit chybovou hodnotu 0, pokud v tomto čase chyba nastala a hodnotu 1, pokud je disk v tomto čase v pořádku.

Tato data pro neuronovou síť byla uložena do nového csv nesoucí původní název nahrávky, což je timestamp jejího pořízení. Díky tomu je potřeba každou nahrávku analyzovat pouze jednou, což šetří výpočty při trénování, k čemuž se dostanu později.

4.3 Zpracování dat pro predikující síť

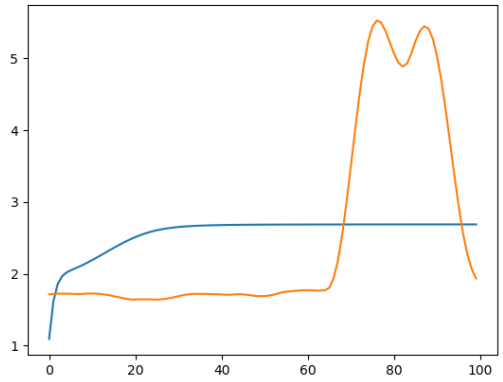
Ačkoli obě sítě pracují se stejnými daty, každá je potřebuje přeci jen trochu jinak. Predikující síť vůbec nezajímá, jestli je daný časový úsek chybový, nebo ne. Zároveň, protože predikce je o něco složitější, než klasifikace, potřebuje větší časový úsek ke zpracování, než klasifikační síť.

Právě zvolení vhodných časových úseků je základem úspěšné predikce. Pokud vezmu krátký úsek, neuronová síť nebude mít dostatečný podklad k pochopení vzorce chování. Naopak ve velkém úseku se může snadno ztratit.

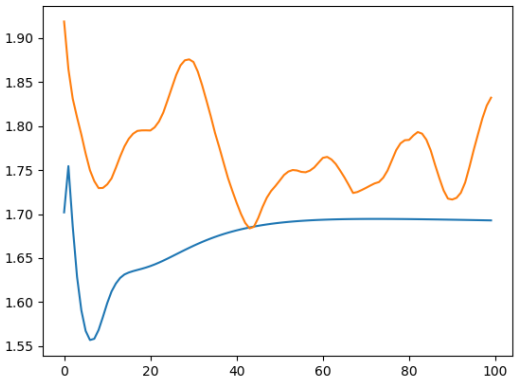
Volba časového úseku pro předpověď se musí odvíjet od délky časového úseku, který chci předvídat. Protože se snažím předvídat zaseknutí disku způsobené jeho zahlcením, může mi stačit predikovat jednu vteřinu, neboť je to čas dostatečný k tomu, aby se mohl spustit nějaký spouštěč, který pozastaví nebo výrazně zredukuje zápisy na disk, čímž mu dá nějaký čas na srovnání se. Zároveň může o celé situaci informovat obsluhu, která může disk později vyměnit.

Jedna vteřina je ještě celkem snadná na naučení, pokud se zvolí správný interval, ze kterého se má predikovat. Toho se docílí odhadem a pokusy, k čemuž už je potřeba mít navrhnutou neuronovou síť. Následující ukázky jsou prováděny na stejném návrhu a liší se pouze časovým úsekem potřebným k predikci. K jejich vygenerování posloužila knihovna Pythonu Matplotlib [13] ve spojení s knihovnou Pandas, která slouží ke zpracování a analýze dat. [14]

Na obrázku 4.6 je znázorněna predikce na základě 15 s. Jak je vidět, tak se síť v patnácti vteřinách zcela ztratila a její predikce je tedy zcela mimo. Tuto křivku si držela takřka při každé predikci zcela nezávisle na vstupu, což je asi nejhorší výsledek, jakého jsem během svých pokusů docílila.

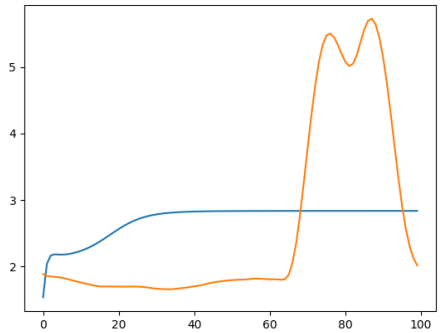


Obrázek 4.6: Predikce na základě 15 s. Modrá je výstup neuronové sítě, oranžová je skutečnost. Osa X znázorňuje časovou osu v milisekundách, osa Y reprezentační hodnotu pro daný čas.



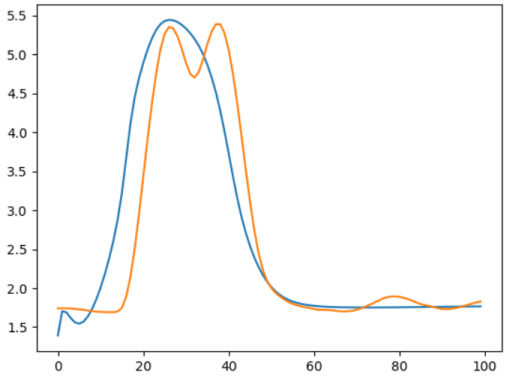
Obrázek 4.7: Predikce na základě 10 s. Modrá je výstup neuronové sítě, oranžová je skutečnost. Osa X znázorňuje časovou osu v milisekundách, osa Y reprezentační hodnotu pro daný čas.

Jeden z dalších pokusů zkoušel predikovat na základě 10 s, viz obrázek 4.7. Tady byl výsledek zajímavější, ačkoli stále ani zdaleka uspokojivý. Síť se na některých případech celkem hezky chytla, místy měla predikce i celkem přesné, ale ve většině případech to bylo stále dost mimo na to, aby se to dalo použít jako správný výsledek.



Obrázek 4.8: Predikce na základě 2 s. Modrá je výstup neuronové sítě, oranžová je skutečnost. Osa X znázorňuje časovou osu v milisekundách, osa Y reprezentační hodnotu pro daný čas.

Příkladem malého časového úseku jsou data se základem k predikci 2 s. Jak je vidět na obrázku 4.8, tato data neuronové síti nestačí k tomu, aby poznala vzor, dle kterého má predikovat další děj. Její výsledek je tedy zcela nevyhovující.



Obrázek 4.9: Predikce na základě 6 s. Modrá je výstup neuronové sítě, oranžová je skutečnost. Osa X znázorňuje časovou osu v milisekundách, osa Y reprezentační hodnotu pro daný čas.

Nejlépe si síť vedla na datech se základem k predikci 6 s. Na obrázku 4.9 je krásně vidět, že její predikce celkem dobře kopíruje skutečné chování disku. Samozřejmě jsou tam jisté odchylky, kterých se v přiděleném čase nezbavím, ale jedná se o solidní základ pro klasifikační síť, což bylo mým záměrem.

4.4 Zpracování dat pro klasifikační síť

Jak již bylo avizováno, klasifikační síť je výrazně jednodušší, ať už mluvíme o datech, nebo o návrhu. Nejprve se zaměřím na data.

Vstup pro klasifikační síť se od té pro predikční liší v tom, že jí jako požadovaný výsledek pro sérii vstupů nedávám reprezentační hodnoty časového úseku, ale jejich klasifikaci, aneb jestli je úsek chybový, nebo ne.

Délka časového úseku klasifikační sítě se odvíjí od minimální délky chyby, kterou chceme klasifikovat. V mém případě pro identifikaci zaseknutí potřebuji alespoň 4 vteřiny. Při trénování tedy potřebuji označit čtyřsekundový úsek, jestli je chybový, nebo ne. Já mám ovšem takovou informaci pouze o 10 ms úsecích.

To se dá vyřešit dvojím způsobem. První, co mě napadlo, bylo zprůměrovat hodnoty v tomto časovém úseku a výsledek zaokrouhlit. To je jistě správné řešení za předpokladu, že síť bude stoprocentně přesná.

Natrénovat síť na takovou přesnost by zabralo hodně času, který jsem neměla, proto jsem zvolila řešení, které nevyžadovalo absolutní přesnost neuronové sítě a zároveň dosahovalo poměrně dost přesných výsledků.

Toto řešení spočívá v tom, že neuronová síť v trénovacích datech nedostává celá čísla, ale čísla reálná v rozmezí 0 až 1. Jinými slovy jsem vzala průměr hodnocení časového úseku, aniž bych jej zaokrouhlila. S takovými daty má síť s lehkým natrénováním nevracela přesné výsledky, ale přesto se držela blízko správných hodnot. Vše, co zbývalo, bylo nastavit práh mezi 0 a 1, který určil jaká hodnota ještě označuje dobrý stav a odkud už je stav chybný.

Volba prahu určuje, jaká chyba sítě mi vadí vic, jestli je to mylné označení chybného stavu, nebo mylné označení stavu dobrého. Při prahu 0,5 říkám, že je pro mě oboje označení stejně důležité. Pokud budu práh přibližovat k nule, budu preferovat mylné označení dobrého stavu, naopak čím blíže bude práh jedničce, tím více budu preferovat mylné označení chyby.

Protože je lepší mylně upozornit na chybu, než chybu opomenout, je můj práh nastaven na hodnotu 0,6.

4.5 Návrh rekurentní neuronové sítě pro predikci

Při návrhu LSTM RNN řeším počet vrstev, počet bloků uvnitř vrstev, aktivační funkci, ztrátovou funkci a optimalizátor.

Keras už stačil spoustu věcí vyřešit za mě. Implementoval několik možných aktivačních i ztrátových funkcí, včetně optimalizátorů. Mimo to samozřejmě implementuje i samotné sítě, respektive vrstvy.

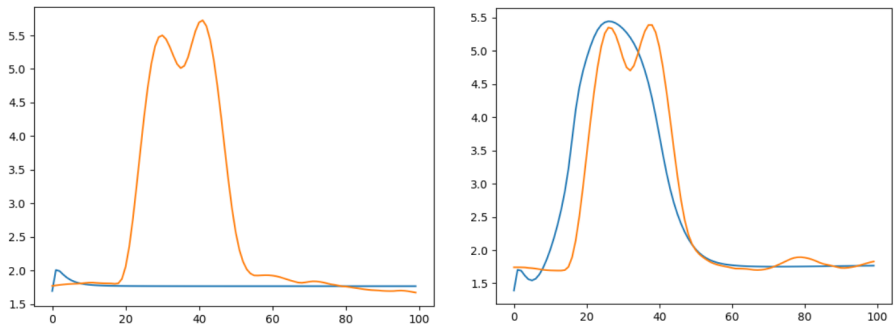
U neuronových sítí je dobré začít jednoduše, bývá to mnohdy lepší, než vymýšlet komplikované sítě s komplikovanými funkcemi. Proto, když řeknu, že mnou navržená predikční síť obsahuje pět vrstev, dá se zapřemýšlet, jestli je to skutečně nezbytně nutné.

Vrstvy v sobě nemají mnoho bloků, není to potřeba. Jsou počtně symetricky rozdělené tak, že první a poslední vrstva mají 32 bloků, druhá a čtvrtá 64 a třetí 128.

Zatímco u sítě s jedním výstupem, jakým je například moje klasifikační vrstva, stačí v Kerasu skládat jednu vrstvu za druhou, v případě této sítě se musí výstup předchozí vrstvy roznásobit do další vrstvy, abychom měli úplné propojení typické pro LSTM.

Zbylé nastavení jsem ponechala typické pro LSTM, což znamená lineární aktivační funkci a RMSprop optimalizátor, který je zas doporučen pro rekurentní neuronové sítě.

Co zbývá, je nastavení ztrátové funkce, která se snaží co nejlépe trestat neuronovou síť za špatné výsledky. V úvahu přicházela vestavěná ztrátová funkce *mean_squared_error (průměrná čtvercová chyba)* a *mean_absolute_error (průměrná absolutní chyba)*.



Obrázek 4.10: Porovnání predikce s mean_absolute_error (vlevo) a s mean_squared_error (vpravo).

Z obrázku 4.10 je patrné, která ztrátová funkce dosahuje lepších výsledků, respektive že funkce mean_absolute_error nedosahuje takřka výsledků žádných. Z tohoto důvodu byla ve výsledné síti použita funkce mean_squared_error.

4.6 Návrh rekurentní neuronové sítě pro klasifikaci

Klasifikace zvuku už je vcelku probádaná oblast, proto existuje mnoho návrhů a jejich implementací.

Klasifikace není tak náročná, jako predikce, proto mi pro dobré výsledky stačí tři malé LSTM vrstvy se 32, 64 a 32 bloky. Jelikož má síť pouze jeden výstup, není potřeba výstupy vrstev nějak roznásobovat.

Nastavení této sítě jsem zkoušela dvěma způsoby.

4.6.1 Aktivační funkce Softmax

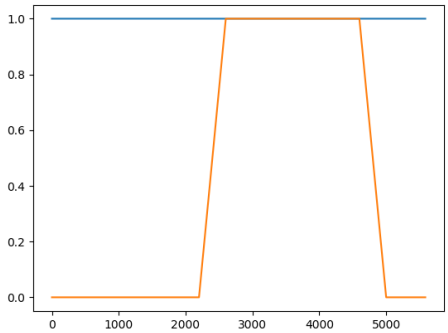
Jako první jsem vyzkoušela aktivační funkci *Softmax*, která se používá pro určení pravděpodobnosti. Jedná se o normalizovanou exponenciální funkci, která seskupí K-dimenzionální vektor z $\sigma(z)$, kde rozsah všech hodnot je mezi 0 až 1. Funkce je dána vzorcem: [15]

$$\sigma : \mathbf{R}^K \to \{z \in \mathbf{R}^K | z_i > 0, \sum_{i=1}^{K} z_i = 1\} \tag{4.1}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; pro \; j = 1, ..., K \tag{4.2}$$

Za ztrátovou funkci jsem zvolila funkci *binary_crossentropy* (binární křížová entropie), což byla s binárními daty vcelku jasná volba.

Optimalizátor Adam mi byl doporučen na základě zkušeností s klasifikačními sítěmi. Jedná se o metodu stochastické optimalizace, která je výpočetně i paměťově nenáročná.

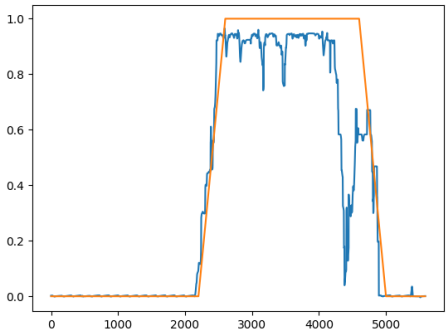


Obrázek 4.11: Klasifikace pomocí LSTM RNN s aktivační funkcí Softmax

Na obrázku 4.11 porovnávám výsledek klasifikace takto nastavené neuronové sítě (modrá funkce) se skutečným hodnocením (oranžová funkce). Jak je patrné, neuronová síť si nevedla vůbec dobře a ve výsledku nevracela nic jiného, než 1.

4.6.2 Nastavení podle predikující neuronové sítě

Vzhledem ke špatným výsledkům předchozího nastavení a výborných výsledků predikční sítě, jsem se rozhodla vyzkoušet její nastavení i pro klasifikaci. To znamená, že jsem použila lineární aktivační funkci, optimalizátor RMSprop a ztrátovou funkci mean_squared_error.



Obrázek 4.12: Klasifikace pomocí LSTM RNN s aktivační funkcí Softmax

Toto nastavení se ukázalo být velmi účinným, jak ukazuje obrázek 4.12, na kterém je vidět, jak klasifikace sítě kopíruje skutečné hodnoty. Odchylky jsou dány poměrně krátkou dobou učení.

4.7 Trénování predikující rekurentní neuronové sítě

Dat, kterých jsem během této práce nasbírala, bylo nepřeberné množství. Proto mi nedělalo nejmenší problém vybírat data trénovací a testovací.

Jak již bylo zmíněno, mé nahrávky trvají více méně 30 s. Neuronová síť ovšem nelpí na délce dat, respektive na jejich množství. Jediné, co musí být pevné, je délka úseku základu pro predikci a délka predikovaného úseku.

Aby měla neuronová síť k dispozici komplexnější data, rozhodla jsem se ji netrénovat po jedné nahrávce, ale rovnou po čtyřech. Tak dostávala 2 minuty dat v kuse.

Recyklace těchto dat poté spočívala v tom, jak se čtyřmi použitými nahrávkami naložím. Při jedné nahrávce je jasné, že ji budu chtít vyhodit, neboť trénovat na stejných datech mi nic nového nepřinese a testovat taková data nedává smysl.

Teď to ovšem byly nahrávky čtyři. Tady bylo potřeba si uvědomit spojitost zvuku. Vyhodit všechny čtyři nahrávky by vytvořilo skok mezi každými dvěma minutami, což jsem nechtěla. Že se trénovací data částečně překrývají, ničemu nevadí, jinými slovy, pokud časový interval 0:00 až 1:00 a poté 0:30 až 1:30,

nemohu to považovat za stejná data, přestože mají půlminutový překryv. Důležitá věc je ta, že druhý interval v sobě nese spojitost mezi časem 1:00 a 1:01, což je něco, o co bych přišla, pokud bych překryv nepoužila.

Z tohoto důvodu byla při každém kroku trénování vyhozena pouze první nahrávka, takže zbylé tři mohly být použity v dalších krocích trénování.

Při každém trénování musí dojít také k testování, abych věděla, jak si síť s daným nastavením vede. Tak, jako jsem vybírala čtyři soubory pro trénování, vybrala jsem i čtyři soubory k testování. Toto číslo bylo zvoleno čistě jen z praktického hlediska, neboť u testování by stačil i jeden soubor, na druhou stranu testování na vícero nahrávkách ničemu neuškodí.

Testované nahrávky není potřeba nějak označovat. Síť si z nich nic neodnesla a mohou tedy být později použity k trénování.

4.8 Trénování klasifikační rekurentní neuronové sítě

Její trénování bylo velice jednoduché, jelikož nepotřebuje tak komplexní data jako predikující síť. Přesto jsem jí předávala trénovací data po dvou souborech, neboť i tato síť potřebuje mít plynulý přechod pro případ, že by chyba nastala mezi soubory.

Samotné trénování se pak už od předchozí sítě nijak nelišilo. Vyřazování souborů fungovalo stejně jako u predikující sítě, kdy samozřejmě obě sítě měly vlastní seznam použitých nahrávek.

4.9 Výsledné testování

K výslednému testování slouží program, který jako parametr dostane timestamp, nebo-li název souboru s připravenými daty. Pokud takový soubor neexistuje, ale je k dispozici dataset s informacemi o nahrávce z té doby, datový soubor vytvoří a provede test.

Jelikož predikující síť vrací jednu vteřinu a klasifikační jich potřebuje čtyři, je potřeba vždy nejprve postupně načíst 4 vteřiny, než se začnou klasifikovat. To není zas takový problém, neboť v realném případě budu mít vždy k dispozici předchozí 3 vteřiny pomocí kterých budu moct klasifikovat i poslední predikovanou.

V tomto programu je nastaven práh pro rozdělení dat z klasifikační neuronové sítě na chybné stavy a dobré stavy. Program tedy vrátí klasifikaci každé vteřiny, což je buď 0 (chyba) nebo 1 (v pořádku) na základě čeho se může spustit spouštěč, který pozastaví, či přibrzdí zápisy na disk, potažmo ještě přivolá obsluhu.

Kapitola 5

Zhodnocení a doporučení dalšího rozvoje

Tato práce ještě rozhodně není kompletní. Ačkoli soustava dvou neuronových sítí je schopná předvídat stav zařízení tak, aby se stihlo něco udělat, jedna sekunda není příliš mnoho a jistě lze docílit lepšího výsledku.

Pro mě je však i tento výsledek úspěch, neboť se jedná o mé první neuronové sítě, které nevrací špatné výsledky. S trochou více času a s prací s Kerasem hned od začátku věřím, že by sítě mohly dosahovat ještě lepších výsledků na delších odhadech.

5.1 Zhodnocení praktické části

5.1.1 Keras

Volba Kerasu se ukázala být velmi šťastnou. Díky Kerasu jsem byla schopná dohnat čas, který jsem ztratila snahou o návrh sítě čistě v TensorFlow, což bez předešlých zkušeností s neuronovými sítěmi šlo jen velice těžko.

Keras se prokázal jako velice univerzální a funkční API, se kterým je snadné se seznámit a pracovat. Je velice univerzální, přestože neobsahuje implementaci všech typů sítí, jako třeba Echo State Networks. Pro mě však byla zcela dostačující podpora LSTM sítí.

5.1.2 Použitelnost

Celý program je vytvořen jako několik samostatných modulů, které lze spouštět jednotlivě, nebo jako celek z příkazové řádky. Takto lze spouštět test, nebo trénovat samostatně neuronové sítě. Závislosti jednotlivých modulů jsou dány jejich importem.

Díky možnosti skládání modulů je úprava finálního programu velice jednoduchá a dá se provést podle konkrétních potřeb. Má práce není nasazena

na žádný server a nedostává tak plynulý přísun dat. Proto je výsledný program přizpůsoben k analýze obdrženého souboru a slouží v podstatě jako ověření konceptu.

Zároveň je celá práce psaná tak, aby šla spustit jak v Pythonu 3. 6. 3, který je jako výchozí pro mé zařízení, tak v Pythonu 2. 7. x, který je výchozí pro firemní server.

Obě neuronové sítě jsou schopné vyhodnocovat testy rychle, takže ani nehrozí, že by predikovaná sekunda uběhla dřív, než se vypočte samotná predikce.

Přesnost predikce budoucího stavu je okolo 70%. Toto číslo nejvíce sráží predikce chování disku, kde se síť občas nemusí zcela trefit. Samotná klasifikace už je výrazně přesnější, zhruba kolem 90%, někdy i více.

5.2 Co lze dál rozvíjet

5.2.1 Délka predikovaného úseku

Pro predikci delšího úseku je potřeba trochu více času, než jsem měla k dispozici. Ta možnost tu ovšem je a stačí k tomu jen vytrvalého trénování predikční sítě, což je stále jen otázka času.

Během tohoto akademického roku se mi povedlo nahromadit dostatek dat, aby bylo trénování možné, nicméně trénování sítě na discích je něco, co naše firma zcela nevyužije, a tím pádem nevyužije ani tato data.

To však ale neznamená, že samotná síť není využitelná, neboť je jen otázkou dat, na jaké zařízení se natrénuje a je-li schopná predikovat stav disku, pak je schopná predikovat stav čehokoli, co vydává chybové zvuky.

5.2.2 Učení na různé typy zařízení

Pevné disky z hlediska praxe nemají moc využití, pokud jde o predikci jejich poruchy. Zvukový doprovod jejich závady zahrnuji mezi jejich výhody, neboť v mnohým případech jsem schopná podle zvuku předpovědět trvalou poruchu dříve, než ztratím data, a to i bez umělé neuronové sítě.

V této práci jsem se o všem snažila o ověření konceptu, kdy jsem schopná navrhnout síť, která bude poruchu předvídat. Pevné disky byly levným způsobem, jak tento koncept ověřit, přestože praktické využití zde není tak velké.

Pokud tato síť má být využívána smysluplně, je třeba získat data ze zařízení, u kterých má predikce jejich stavu větší smysl. Na těchto datech se poté dají natrénovat neuronové sítě podle návrhu z této práce.

5.2.3 Uživatelské rozhraní

Celá síť by se dala obalit uživatelským rozhraním, ve kterém by mohl být seznam sledovaných zařízení spolu s informací o aktuálním stavu a následně

upozorněním o pravděpodobné chybě.

Z tohoto rozhraní by bylo možné rovnou provádět akce na daném zařízení, jako například přimět zařízení vypnout se, pozastavit se, nebo restartovat se.

Zařízení by byla rozdělena podle typů, podle kterých by se jim přiřadila natrénovaná neuronová síť, neboť každý typ vydává jiné zvuky, podle kterých se síť trénuje. Rozhraní by také umožňovalo nastavit akce, které se mají při upozornění provádět automaticky.

Toto uživatelské rozhraní by ovšem mělo smysl pouze v případě, kdy je neuronová síť schopná predikovat stav s větším časovým předstihem, než je jedna vteřina.

Závěr

Cílem práce bylo navrhnout řešení pro průběžné sledování stavu zařízení pomocí analýzy zvuku. Toto sledování mělo umožnit včasný zásah odborníka, díky kterému se zařízení včas opraví, či vymění, a tím se zamezí větším škodám, nebo ztrátě dat při nečekané poruše. Ke sledování stavu zařízení je použit zvukový záznam vybraného zařízení v provozu a rekurentní neuronová síť.

Navržená síť je schopná zpracovávat zvukové nahrávky sledovaných zařízení a na základě toho predikovat jejich stav jednu vteřinu dopředu. Její rozhraní lze snadno přizpůsobit pro okamžité vyhodnocení, které je pro průběžné sledování nezbytné.

Přesnost vyhodnocení budoucího stavu je okolo 70%. Tento výsledek lze ovlivnit nastavením prahu, podle kterého se stav označí za dobrý, nebo chybný. Jeho nastavení je už ovšem věc osobní preference, aneb jaký typ špatné klasifikace je mi milejší.

Cíl práce byl tedy splněn, ačkoli se nedá mluvit o zcela ideálním stavu a je potřeba dalšího rozvoje, především trénování sítě, aby se takového stavu docílilo.

Literatura

[1] Hochreiter, S.; Schmidhuber, J.: Artificial Neuron Models. 1999, online, accessed 10-May-2018. Dostupné z: http://www.willamette.edu/~gorr/classes/cs449/ann-overview.html

[2] Hochreiter, S.; Schmidhuber, J.: Recurrent Networks. 1999, online, accessed 10-May-2018. Dostupné z: http://www.willamette.edu/~gorr/classes/cs449/rnn1.html

[3] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. 1999, online, accessed 10-May-2018. Dostupné z: http://www.willamette.edu/~gorr/classes/cs449/lstm.html

[4] Optisolutions: Optisolutions s.r.o. [online]. 2018. Dostupné z: http://optisolutions.cz

[5] 3DSignals: 3DSignals [online]. 2018. Dostupné z: https://www.3dsig.com/

[6] Soundware, N.: Neuron Soundware [online]. 2018. Dostupné z: https://www.neuronsw.com/

[7] Vondrák, I.; škola báňská Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky, V.: *Umělá inteligence a neuronové sítě*. VŠB - Technická univerzita Ostrava, 2009, ISBN 9788024819815. Dostupné z: https://books.google.cz/books?id=LcSMQwAACAAJ

[8] Lanc, J.: *Tvorba hudby počítačem*. Diplomová práce, VUT FIT, 2009.

[9] Python Software Foundation: *wave — Read and write WAV files*. 2018. Dostupné z: https://docs.python.org/3/library/wave.html

[10] librosa development team: *LibROSA*. 2017. Dostupné z: http://librosa.github.io/librosa/

[11] TensorFlow: *TensorFlow*. 2018. Dostupné z: https://www.tensorflow.org/

[12] The French National Research Agency: *Waves.js - UI*. 2017. Dostupné z: http://wavesjs.github.io/waves-ui/

[13] The Matplotlib development team: *Matplotlib: Python plotting*. 2018. Dostupné z: https://matplotlib.org/

[14] NumFOCUS: *Python Data Analysis Library*. 2018. Dostupné z: https://pandas.pydata.org/

[15] Lan, H.: The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers. *Towards Data Science*, 11 2017. Dostupné z: https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932

Příloha **A**

Seznam použitých zkratek

RNN Rekurentní neuronová síť (Recurrent neural network)

LSTM Long Short-Term Memory

EST Echo State Networks

STFT Krátkodobá Fourierova transformace (Short-time Fourier transform)

WMA Windows Media Audio

API Application Programming Interface

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `thesis.pdf` ..... text práce ve formátu PDF
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `audio` ..... audio nahrávky HDD
    - `data` ..... zpracovaná data pro RNN
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX