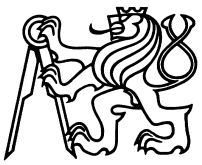
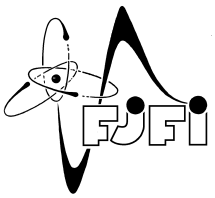


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Rozpoznávání ultrazvukových signálů pomocí konvolučních neuronových sítí

Recognition of Ultrasonic Signals Using Convolutional Neural Networks

Bakalářská práce

Autor: **Martin Kovanda**

Vedoucí práce: **Ing. Milan Chlada, Ph.D.**

Konzultant: **Ing. Josef Krofta, Ph.D.**

Akademický rok: 2019/2020

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Martin Kovanda
Studijní program:	Aplikace přírodních věd
Obor:	Matematické inženýrství
Zaměření:	Aplikované matematicko-stochastické metody
Název práce (česky):	Rozpoznávání ultrazvukových signálů pomocí konvolučních neuronových sítí
Název práce (anglicky):	Recognition of Ultrasonic Signals Using Convolutional Neural Networks

Pokyny pro vypracování:

1. Seznámení s teoretickým modelem konvolučních neuronových sítí (CNN).
2. Výběr vhodné softwarové plaformy a implementace základního modelu CNN pomocí dostupných knihoven.
3. Optimalizace časo-frekvenční interpretace dostupných ultrazvukových dat (spektrogram, spojitá waveletová transformace, apod.).
4. Výpočet tréninkových, validačních a testovacích množin adekvátního rozsahu.
5. Provedení numerických experimentů pro ověření rychlosti učení CNN a úspěšnosti rozpoznávání v závislosti na jejich architektuře a variabilitě analyzovaných dat.

Doporučená literatura:

1. R. Rojas, Neural networks: a systematic introduction. Springer-Verlag Berlin Heidelberg, ISBN 3-540-60505-3, 1996.
2. F. Chollet, Deep Learning with Python. Manning Publications Co., ISBN 9781617294433, 2017.
3. S. Kapur, Computer Vision with Python 3. Packt Publishing Ltd., ISBN 9781788299763, 2017.
4. S. Pal, A. Gulli, Deep Learning with Keras. Packt Publishing Ltd., ISBN 9781787128422, 2017.
5. J. L. Rose, Ultrasonic Waves in Solid Media. Cambridge University Press, 2004.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Milan Chlada, Ph.D.
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Jméno a pracoviště konzultanta:

Ing. Josef Krofta, Ph.D.
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Datum zadání bakalářské práce: 31.10.2019

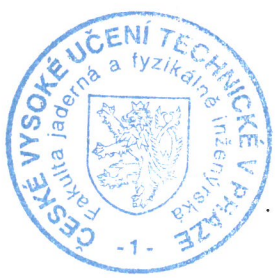
Datum odevzdání bakalářské práce: 7.7.2020

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 23. října 2019

garant oboru

vedoucí katedry



děkan

Poděkování:

Chtěl bych zde poděkovat svému školiteli Ing. Milanu Chladovi, Ph.D., za trpělivost, pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé bakalářské práce.

Čestné prohlášení:

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 7. července 2020 Martin Kovanda

Název práce:

Rozpoznávání ultrazvukových signálů pomocí konvolučních neuronových sítí

Autor: Martin Kovanda

Obor: Matematické inženýrství

Zaměření: Aplikované matematicko-stochastické metody

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Milan Chlada, Ph.D., Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Konzultant: Ing. Josef Krofta, Ph.D., Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Abstrakt: V této práci jsou shrnuty základní architektury neuronových sítí a možnosti použití vrstevnatých a konvolučních neuronových sítí (CNN) na rozpoznávání ultrazvukových signálů. Za tímto účelem jsou porovnávány architektury, které se navzájem liší hodnotami užitých hyper-parametrů. Těmi jsou například celkový počet neuronů v síti spolu s počtem použitých vrstev. Dále je znázorněn přínos použití L2 regularizace, dropoutu a svazkové normalizace za účelem potlačení přeučení sítě. V případě CNN jsou pak tyto regularizace porovnávány zvlášť pro konvoluční i vrstevnatou část sítě. Použití CNN je umožněno zpracováním ultrazvukového signálu pomocí časo-frekvenční transformace. Za tímto účelem byl testován tzv. *HFD spektrogram*, pomocí kterého byly ultrazvukové signály transformovány do tenzorové podoby s využitím různě volených parametrů. Na takto transformovaných signálech byly úspěšně testovány CNN pro identifikaci 11 různých letových režimů vrtulníku.

Klíčová slova: akustická emise, hluboké učení, neuronové sítě, strojové učení

Title:

Recognition of Ultrasonic Signals Using Convolutional Neural Networks.

Author: Martin Kovanda

Abstract: The project summarizes basic architectures of neural networks and possible usage of dense and convolutional neural networks (CNN) for recognition of ultrasonic signals. For this purpose various architectures are compared using different hyper-parameters, e.g. total amount of neurons together with a number of used layers. Furthermore, the

project shows benefits of using L2 regularization, dropout and batch normalization in order to reduce over-fitting. In case of CNN these techniques are compared individually both for convolutional and dense part of a network. Usage of CNN is possible thanks to processing signals using a time-frequency transformation. For this purpose *HFD spectrogram* was tested. This transformation was then used for transforming ultrasonic signals to a tensor form using many diverse parameters. CNN were successfully tested on signals transformed by this transformation for recognition of 11 various flight modes.

Key words: acoustic emission, deep learning, neural networks, machine learning

Obsah

Úvod

V současné době dochází k čím dál většímu rozmachu umělé inteligence, který úzce souvisí s postupným zlepšováním hardwarových možností. Jednou z oblastí, kde dochází k jejímu využití, je i analýza signálu. Díky umělé inteligenci v podobě neuronových sítí totiž není potřeba provádět často složité parametrizace a je umožněno například analyzovat netransformovaný signál přímo. K tomuto účelu jsou již vrstevnaté neuronové sítě využívány. Součástí této práce je zkoumání přínosu použití konvolucních neuronových sítí (CNN), které mají oproti vrstevnatým sítím několik zásadních výhod, jako třeba menší citlivost na posun signálu, ale i možnost využití různých časo-frekvenčních transformací.

Neuronová síť se chová jako transformace, která ze vstupních dat vyextrahuje užitečné informace. Příkladem může být i zařazení do předem definovaných tříd, což mají za úkol tzv. *klasifikátory*. Výstupem klasifikátoru je rozdělení pravděpodobnosti zařazení vstupních dat do jednotlivých tříd. Před tím, než se začaly používat neuronové sítě, bylo velice težké tohoto cíle dosáhnout, zejména pro složitější data. Každý model totiž obsahoval spoustu parametrů, které se musely nastavovat ručně, případně pomocí složitých matematických metod. Takový přístup byl neefektivní a vyžadoval neúměrné množství času. Příchod neuronových sítí tento problém výrazně zjednodušil. Stačí již pouze navrhnout architekturu sítě (strukturu, použité aktivační funkce atd.) a poskytnout jí dostatečný počet dat, podle nichž potom síť sama své parametry cíleně upraví.

Tato práce shrnuje základní poznatky ohledně neuronových sítí, od pojmů, jako je neuron a perceptron, až po CNN a genetický algoritmus. Ve druhé části práce jsou potom tyto poznatky aplikovány při klasifikaci ultrazvukových signálů, přičemž je zkoumána a porovnávána vhodnost různých architektur sítí a variant časo-frekvenčních interpretací signálů, zejména pomocí HFD spektrogramu. Všechny důležité kódy použité při jeho implementaci jsou shrnuty v samostatné kapitole.

Kapitola 1

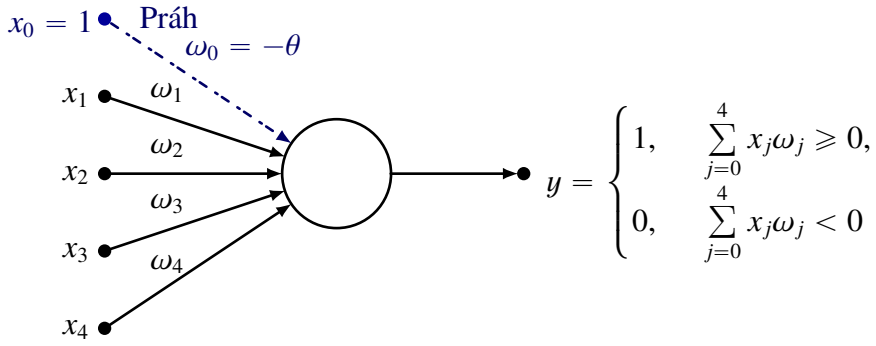
Základní architektury neuronových sítí

Historie vývoje teorie neuronových sítí se datuje už od roku 1943, kdy vědci Warren McCulloch a Walter Pitts popsali způsob, kterým by mohly fungovat neurony v lidském mozku [1]. Na základě toho byly vytvořeny i první návrhy umělých neuronových sítí. První praktické využití však přišlo až roku 1959, kdy vznikl systém *MADALINE* (*Multiple ADAptive LINear Elements*), který odbourával ozvěnu vznikající v telefonech během běžných hovorů. Od té doby rychlost vývoje neuronových sítí úzce souvisí s narůstajícím výkonem výpočetní techniky. Postupem času totiž vznikají větší a výpočetně náročnější problémy, na které jsou aplikovány různé modely neuronových sítí.

1.1 Perceptron a Neuron

1.1.1 Perceptron

Perceptron je model, který ze vstupních dat x_i vypočítá vážený součet přes váhy ω_i a na základě prahové hodnoty θ určí hodnotu binárního výstupu jako 0 nebo 1. Tato prahová hodnota se může chápat jako jednotkový vstup s váhou $-\theta$, který nazýváme *práh* (*bias*) [2]. Vstupní hodnoty perceptronu mohou být jak binární, tak i reálná čísla. Pomocí



Obrázek 1.1: Jednoduché schéma perceptronu se čtyřmi vstupy a prahem, kde θ je prahová hodnota.

perceptronu lze modelovat některé funkce, jako například funkci *OR* [2]. Uveďme příklad pro 2 proměnné. Dosazením do vztahu pro y získáme 4 nerovnice:

$$x_1 = 0,\ x_2 = 0 \ \Rightarrow\ \mathrm{OR}[x_1, x_2] = 0,\quad \sum_{j=0}^{2} x_j \omega_j = \omega_0 < 0, \tag{1.1}$$

$$x_1 = 1,\ x_2 = 0 \ \Rightarrow\ \mathrm{OR}[x_1, x_2] = 1,\quad \sum_{j=0}^{2} x_j \omega_j = \omega_0 + \omega_1 \geqslant 0, \tag{1.2}$$

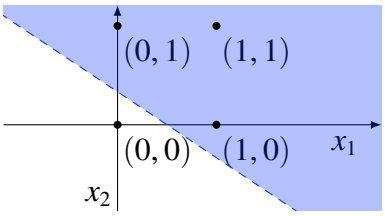
$$x_1 = 0,\ x_2 = 1 \ \Rightarrow\ \mathrm{OR}[x_1, x_2] = 1,\quad \sum_{j=0}^{2} x_j \omega_j = \omega_0 + \omega_2 \geqslant 0, \tag{1.3}$$

$$x_1 = 1,\ x_2 = 1 \ \Rightarrow\ \mathrm{OR}[x_1, x_2] = 1,\quad \sum_{j=0}^{2} x_j \omega_j = \omega_0 + \omega_1 + \omega_2 \geqslant 0. \tag{1.4}$$

Tuto soustavu řeší například $\omega_0 = -1, \omega_1 = 2, \omega_2 = 3$. Po dosazení dostáváme nerovnici

$$2x_1 + 3x_2 - 1 > 0. \tag{1.5}$$

Celkově se tedy perceptron chová jako lineární binární klasifikátor, který generováním



Obrázek 1.2: Vizualizace perceptronu pro modelování funkce OR.

vhodné nadroviny dělí vstupní prostor na dvě části. Pro lineárně neseparabilní data je však tento model nevhodný a musí se použít jiné metody. Například pro obyčejnou XOR funkci už musí být použito několik vrstev perceptronů [3].

1.1.2 Neuron

Neuron se chová podobně, jako perceptron, ale namísto binárního výstupu poskytuje celý interval hodnot (často v intervalu [0,1]) v závislosti na tzv. *aktivační funkci* [4]. Neuron je definován rovnicí

$$y = f\Big(\sum_{j=0}^{n} x_j \omega_j\Big), \tag{1.6}$$

kde f představuje aktivační funkci a zbytek značení zůstává stejné, jako v případě perceptronu. Neuron je tedy zobecněním perceptronu, protože perceptron získáme z neuronu jednoduchým dosazením Heavisideovy funkce Θ za aktivační funkci. Díky tomu, že výstupy neuronů mohou nabývat libovolné hodnoty z intervalu $\mathbb{R}$, mají také větší možnosti využití. To je jeden z důvodů, proč se právě ony využívají nejčastěji v tzv. *vrstevnatých sítích*.

1.2 Vrstevnaté sítě

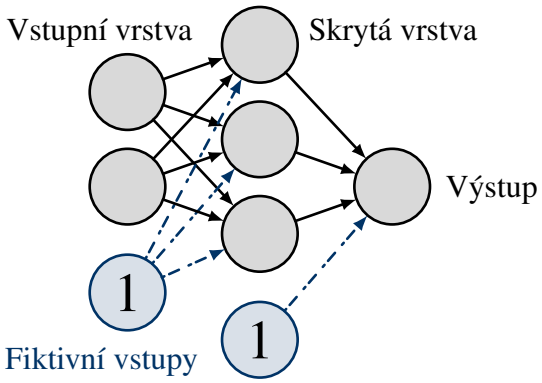
Vrstevnaté sítě jsou matematický model inspirovaný vazbami mezi neurony v lidském mozku [4]. Jedná se však o pouhou analogii, protože mozek ve skutečnosti funguje odlišně. Tyto sítě jsou obecně složeny ze tří typů vrstev, vstupní vrstvy, která pouze distribuuje vstupní data, jedné nebo více skrytých vrstev a nakonec výstupní vrstvy. Neurony z každé vrstvy jsou synapsemi propojeny se všemi neurony z předchozí i následující vrstvy. Neurony z různých vrstev také mohou mít různé aktivační funkce. Například pro výstupní vrstvu klasifikátoru se často využívá tzv. *sigmoida*, protože platí, že

$$f(x) = \frac{1}{1+e^{-x}} : \mathbb{R} \to [0,1]. \tag{1.7}$$

Obecně však není ve výstupní vrstvě potřeba použít žádnou aktivační funkci ($f(x) = x$), například pokud může výstup nabývat libovolné hodnoty (předpovídání cen nemovitostí apod. [5]). Na vícetřídovou klasifikaci se pak vyplatí použít tzv. funkci *softmax* [6], která výstupní hodnoty přeškáluje tak, aby vektor výstupu udával rozdělení pravděpodobnosti zařazení do jednotlivých tříd.

Vrstevnaté sítě již fungují dobře i na komplikovanějších problémech. V současné době však neexistuje žádná metoda, která by určila ideální počet skrytých vrstev a počet neuronů v jednotlivých vrstvách. Výstupní vrstva je potom vytvořena tak, aby poskytovala výsledky ve vhodném tvaru. Například pro binární klasifikaci stačí použít pouhý jeden neuron s aktivační funkcí sigmoid. Pro vícetřídovou klasifikaci je možné použít několik obdobných neuronů (pro každou třídu jeden).

Ve vrstevnatých sítích se používá velké množství parametrů, mezi které patří všechny váhy a prahy jednotlivých neuronů. Určení hodnot těchto parametrů probíhá v rámci tzv. *učení*, při kterém síť sama mění své parametry na základě trénovacích dat, tedy dat, u kterých je již k dispozici požadovaný výsledek. Hodnoty parametrů jsou upraveny tak, aby výstup sítě pro trénovací data byl co nejbližší požadovaným výsledkům. Za tímto účelem se využívá metoda *zpětného šíření*.



Obrázek 1.3: Základní vrstevnatá neuronová síť se dvěma vstupy a jedním výstupem.

Jednotlivé vrstvy sítě můžeme popsat [7] rovnicemi

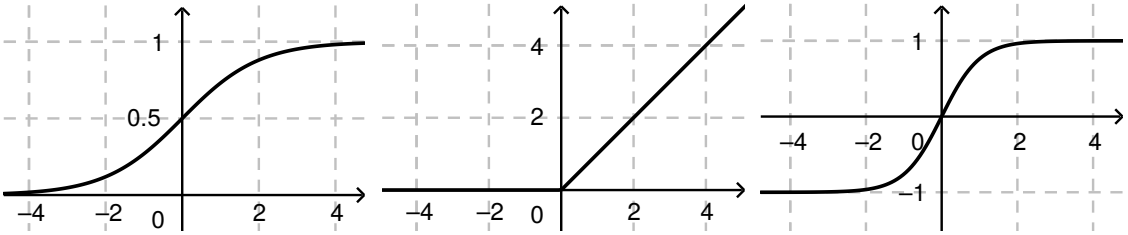
$$\boldsymbol{y}_n = f(\boldsymbol{x}_n), \tag{1.8}$$
$$\boldsymbol{x}_{n+1} = \mathbb{W}_n \cdot \boldsymbol{y}_n, \tag{1.9}$$

kde f značí aktivační funkci (působící po složkách), $\boldsymbol{x}_n$ je vektor vstupu do n-té vrstvy, $\boldsymbol{y}_n$ je vektor výstupu n-té vrstvy a $\mathbb{W}_n$ je matice vah mezi n-tou a $(n+1)$-ní vrstvou.

1.2.1 Aktivační funkce

Jedna z vět lineární algebry říká, že složení lineárních transformací je opět lineární transformace. Z toho vyplývá, že vytváření vrstevnaté sítě z vícero lineárních transformací není přínosné. Existuje proto celá řada nelineárních aktivačních funkcí, které se používají v praxi. Nejznámější z nich jsou například

$$\mathrm{SIGMOID}(x) = \frac{1}{1+\mathrm{e}^{-x}}, \tag{1.10}$$
$$\mathrm{ReLU}(x) = \max(0, x), \tag{1.11}$$
$$\tanh(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}, \tag{1.12}$$
$$\mathrm{Id}(x) = x. \tag{1.13}$$



Obrázek 1.4: Srovnání aktivačních funkcí sigmoidy, ReLU a tanh.

1.2.2 Ztrátová funkce

Ztrátová funkce (*Loss function*, případně *Cost function*) je klíčová pro úpravu parametrů celé sítě. Při procesu učení totiž hodnotí chybu výstupu oproti požadovaným hodnotám. Síť poté mění své koeficienty tak, aby výsledek této funkce byl co nejnižší, tedy aby měla síť co možná nejlepší výsledky na trénovacích datech. V případě, že tato data nebyla čistě náhodná, by pak měla mít dobré výsledky i na datech, na kterých netrénovala. Ztrátových funkcí existuje celá řada a jsou využívány podle typu daného problému [8].

Střední kvadratická chyba (MSE)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}\|^2, \quad \|\mathbf{y}_i - \hat{\mathbf{y}}\| := \sqrt{\sum_j (y_{ij} - \hat{y}_j)^2}, \tag{1.14}$$

kde $\boldsymbol{y}_i$ je výstup sítě pro i-tá data, $\hat{\boldsymbol{y}}$ je námi požadovaný výsledek a N počet dat, která uvažujeme (celý dataset, případně nějaký svazek, či jednotlivá data). Výhodou této funkce je to, že díky kvadratičnosti bere v potaz i data, která jsou velice vzdálená požadovanému výsledku [8]. Tato vlastnost se ale může stát i nevýhodou ve chvíli, kdy je například jedna hodnota velice vychýlená vlivem špatného měření.

Střední absolutní chyba (MAE)

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}\|. \tag{1.15}$$

Výhody i nevýhody MAE jsou přesně opačné oproti MSE. Je totiž méně náchylná vůči přehnaně chybným ojedinělým datům, ale celkově vzato nemusí dávat dobré výsledky [8].

Binární křížová entropie

$$L = -\frac{1}{N}\sum_{i=1}^{N} \big(\hat{y}\log(y_i) + (1-\hat{y})\log(1-y_i)\big), \tag{1.16}$$

kde y je výstup sítě (skalární) a $\hat{y}$ námi požadovaný výsledek. Tato ztrátová funkce nachází využití u binárních klasifikátorů [9][10].

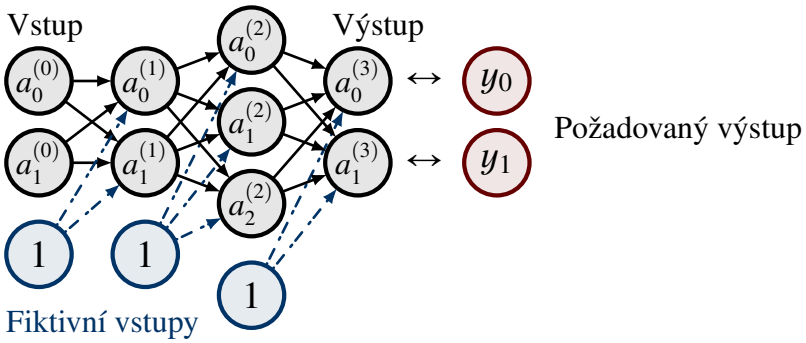
1.2.3 Zpětné šíření

Zpětné šíření (*backpropagation*) je metoda učení, pomocí které síť upravuje své parametry v závislosti na zvolené ztrátové funkci.

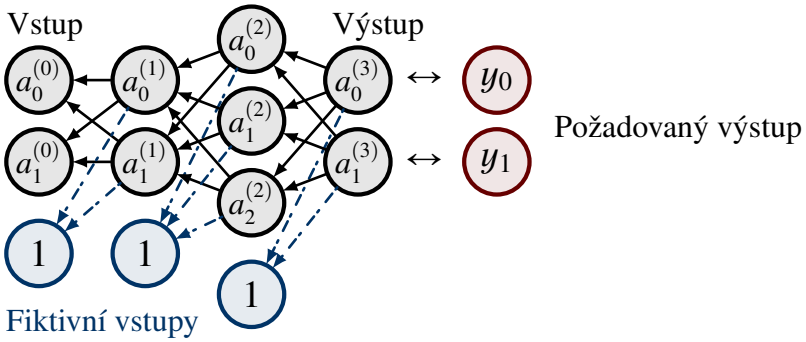
Tato metoda se dá rozdělit na dvě fáze, viz obr. 1.5. Nejprve síť vyhodnotí trénovací data podle parametrů, které již obsahuje. Tento výsledek potom porovná s požadovaným výsledkem, tedy s hodnotou, která je pro trénovací data už předem známá. K tomuto porovnání slouží ztrátová funkce, která je volena v rámci konstrukce sítě.

Ze ztrátové funkce je možno vypočítat tzv. *ztrátu* (*loss*), která je při procesu učení minimalizována. Za tímto účelem je třeba patřičně upravit váhy. Pokud by však byly upraveny tak, aby ztráta na jednom prvku datasetu byla nulová, zhoršilo by to obecně schopnosti vyhodnocování zbylých dat.

Fáze 1: Výpočet výstupu a porovnání s požadovaným výstupem.



Fáze 2: Změna parametrů sítě na základě ztrátové funkce.



Obrázek 1.5: Schéma zpětného šíření.

Z tohoto důvodu je třeba, aby ztrátová funkce sítě klesala do lokálního (ideálně globálního) minima na všech trénovacích datech současně. K tomu se používá metoda gradientního sestupu, při němž se vypočítá gradient ztrátové funkce podle jednotlivých vah a pomocí něho se potom mění jejich hodnoty. Při výpočtu gradientu je výhodnější postupovat směrem od poslední vrstvy k první [11]. Pro výpočet gradientu vah v jednotlivých vrstvách je totiž možno využít dílčí výsledky z následujících vrstev. Odtud také pochází název zpětného šíření.

V praxi se potom nemění hodnoty po jednom vzorku, ale po tzv. *svazku* dat (*batch*), kde se nejprve vypočítají pomyslné změny vah od všech prvků ve svazku a následně jsou váhy změněny podle průměru těchto dílčích změn. Tím je dosaženo plynulejšího sestupu do lokálního minima i snížení výpočetní náročnosti. Za stejný čas se tímto způsobem dospěje k lepšímu výsledku [12].

1.2.4 Inicializace vah a prahové hodnoty

V průběhu učení dochází k postupné úpravě vah tak, aby byla minimalizována ztrátová funkce. V momentě, kdy je navržena architektura sítě, však žádné váhy k dispozici nejsou, a proto je potřeba tyto hodnoty inicializovat. Správná inicializace je důležitá a závisí na ní rychlost konvergence a úspěšnost celé sítě. Pokud by byla například zvolena hodnota pro všechny váhy v síti stejná, síť by se stala symetrickou a učení by nezvyšovalo kvalitu vyhodnocování, protože by byla hodnota gradientu pro všechny váhy v jednotlivých

vrstvách identická. Celá síť by tedy postrádala smysl a nefungovala by správně.

Nabízí se proto inicializovat váhy podle některého rozdělení pravděpodobnosti. Pokud by však byly všechny váhy v celé síti inicializovány podle stejného rozdělení, potom by gradient v jednotlivých vrstvách nabýval výrazně odlišné hodnoty [13]. V extrémním případě by byl tento rozdíl i několik řádů. Parametry v jednotlivých vrstvách sítě by se potom měnily jinou rychlostí, což by bylo pro proces učení nežádoucí [14].

Další problém by nastal, kdyby měly váhy v absolutní hodnotě nastaveny příliš vysokou, nebo nízkou hodnotu. Pokud by totiž byla použita jako aktivační funkce sigmoida (ale i tanh apod.), gradient by byl pro vysoké hodnotách blízký nule [13]. Síť by proto svoje parametry měnila velice pomalu a trénování by bylo neefektivní (pomalé a často i neúspěšné). Tento problém se nazývá *vanishing gradient problem* a dá se částečně redukovat použitím jiných aktivačních funkcí, například ReLU [13]. Naopak pro nízké hodnoty by absolutní hodnota jednotlivých složek gradientu v průběhu učení nabývala příliš nízkých hodnot v důsledku násobení malými čísly a učení by bylo rovněž neefektivní.

Vznikly proto inicializace, které se s těmito problémy umí vypořádat.

Inicializace Xavier/Glorot

Tato inicializace byla navržena pány Xavier Glorot a Yoshua Bengio. Přesto však bývá někde označována jako Xavierova a někde jako Glorotova funkce (například v prostředí Keras, [15]), což může být velice matoucí. Tato metoda je založená na ponechání rozptylu vah ve všech vrstevách na podobné hodnotě. Zjistilo se totiž, že již po inicializaci dochází k tomu, že v posledních vrstvách je hodnota gradientu vyšší, než v těch prvních [14].

Je několik možností, jak s tímto problémem naložit. Běžné doporučení je například to, že pokud je to možné, vstupní hodnoty by měly mít $\mu = 0$, $\sigma^2 = 1$ a všechny skryté vrstvy také. Toho lze docílit například odečtením střední hodnoty ($\mu = \mathrm{E}[x]$) a vydělením standardní odchylkou σ.

Xavier/Glorotova inicializace spočívá v nastavení vah tak, aby měly jejich hodnoty nulovou střední hodnotu a rozptyl o hodnotě

$$\sigma^2 = \frac{2}{n_i + n_{i+1}}, \tag{1.17}$$

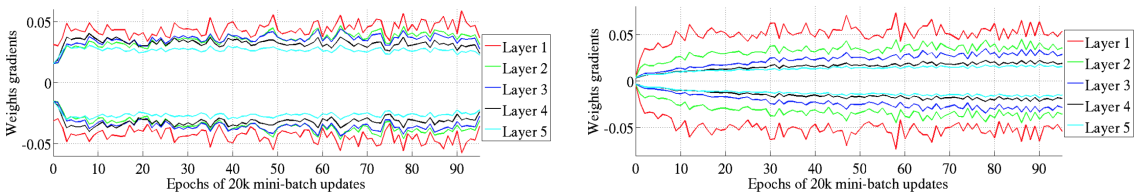
kde n_i označuje počet neuronů v i-té vrstvě [16]. Tomu odpovídají například normální rozdělení $\mathcal{N}\left(0, \frac{2}{n_i+n_{i+1}}\right)$. Často se používá i rovnoměrné rozdělení, které musí kvůli nulové střední hodnotě být symetrické podle 0. Rozptyl rovnoměrného rozdělení na intervalu $[-a, a]$ se spočítá jako $\sigma^2 = \frac{1}{3}a^2$. Z toho vyplývá, že

$$a = \sqrt{3\sigma^2} = \sqrt{3}\sigma. \tag{1.18}$$

Výsledné rovnoměrné rozdělení se tedy nachází na intervalu [16]

$$\mathcal{I}_n = \left[-\frac{\sqrt{6}}{\sqrt{n_i + n_{n+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\right]. \tag{1.19}$$

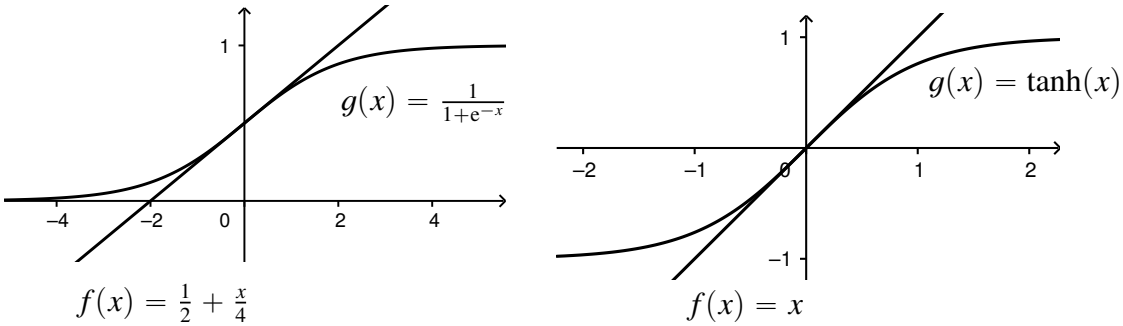
Tento přístup značně přibližuje rozptyly aktivací a gradientů ze zpětné propagace (viz obrázek 1.6). Autoři této inicializace také experimentálně demonstrovali její účinnost



Obrázek 1.6: Porovnání Glorotovy inicializace a "standardní" inicializace na pětivrstvé síti [17].

zdroj: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

na úloze nazvané *CIFAR-10 image classification task*, ve které se síť snaží rozpoznávat 10 tříd obrázků z databáze 60000 snímků [17].



Obrázek 1.7: Lineární aproximace sigmoidy a tanh.

Inicializace He

Předchozí přístup funguje pouze při použití aktivační funkce $f(x) = x$, případně v přiblížení i pro $g(x) = \tanh(x)$, protože je rovněž symetrická v nule a na jejím okolí se dá dobře aproximovat funkcí $f(x) = x$ (viz obr. 1.7). Pro použití této inicializace pro aktivační funkci ReLU je potřeba počítat s tím, že tato funkce vynuluje všechna záporná čísla. Rozptyl je zde dvojnásobný oproti Glorotovi [16] a dalo by se tedy napsat, že $\sigma^2 = \frac{4}{n_i+n_{i+1}}$. Bylo ale dokázáno [18], že pro běžné struktury sítí stačí brát v potaz počet neuronů v právě počítané vrstvě, tedy $\sigma^2 = \frac{2}{n_i}$.

Inicializace pro sigmoidu

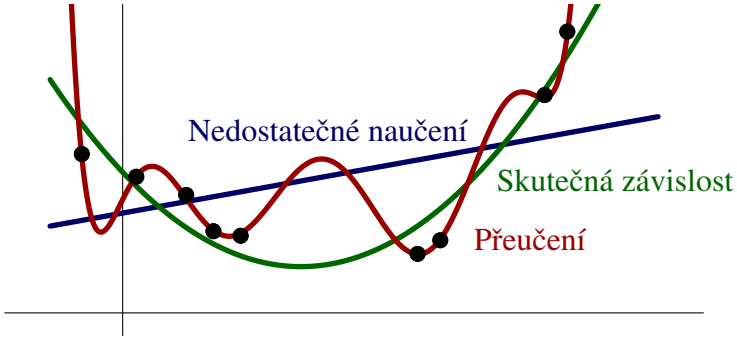
V případě sigmoidy je postup obdobný [16], pouze se aproximuje funkcí $f(x) = \frac{1}{2} + \frac{x}{4}$ (viz obr. 1.7) a po několika výpočtech lze dospět k rozptylu $\sigma^2 = \frac{32}{n_i+n_{i+1}}$.

1.2.5 Přeučení

Přeučení je stav, kdy síť velice dobře vyhodnocuje trénovací data, ale zároveň nedokáže správně vyhodnocovat testovací data (viz obr. 1.8). Existuje několik způsobů, jak se tomuto problému vyhnout. Nejzákladnější z nich je konstruovat síť tak, aby měla optimální množství parametrů. Příliš velká síť by se totiž neučila, jak generalizovat souvislosti v datech, ale místo toho by se naučila testovací data a jejich vyhodnocení podobně, jako by to byl slovník pojmů [5]. Příliš malá síť by naproti tomu neobsahovala dostatečné množství parametrů a neměla by pro učení dostatečnou kapacitu.

Ideálně konstruovaná síť potom musí v rámci minimalizace ztrátové funkce nalézat obecné závislosti. V současné době neexistuje žádný způsob, jak určit vhodný počet neuronů a vrstev, a proto se v praxi vždy porovnávají různě konstruované architektury a až následně se určí, která z nich daný problém vyhodnocuje nejlépe.

Dalšími možnostmi, jak zabránit přeučení, je použít tzv. *regularizace*, *dropout* nebo *svazkovou normalizaci* (*batch normalization*).



Obrázek 1.8: Příklad přeučení a nedostatečného naučení u kvadratických dat.

Regularizace

Celý koncept regularizace spočívá v aplikaci tzv. *Occamovy břitvy* [5]. Ta se dá interpretovat i tak, že pokud existuje několik možných vysvětlení daného problému, je lepší upřednostnit to nejméně komplikované. V případě dvou sítí se srovnatelnou úspěšností je pak lepší upřednostnit menší z nich. Další možností je snižovat váhy sítě na co nejnižší hodnotu při zachování generalizačních vlastností.

Možností existuje celá řada. Velice časté jsou však L1 a především L2 regularizace. V případě L1 se do hodnoty ztrátové funkce zahrnuje i absolutní hodnota jednotlivých vah. U L2 se pak přidává kvadratická hodnota. Tato metoda výrazně zabraňuje přeučení sítě.

Dropout

Další metodou, jak zabránit přeučení, je použití dropoutu [19]. Princip této metody spočívá v tom, že se v procesu učení náhodně vynulují některé hodnoty výstupu vrstvy, na kterou je aplikován dropout [5]. Množství takto vynulovaných hodnot je určeno v rámci

konstrukce sítě tzv. *droupoutovým poměrem*. Velice často se používá [5] hodnota z intervalu $[0.2, 0.5]$. Dropout je aplikován pouze v procesu trénování, přičemž se zároveň přeškálují hodnoty výstupu vrstvy vydělením jejich hodnot dropoutovým poměrem (tedy například číslem 0.5).

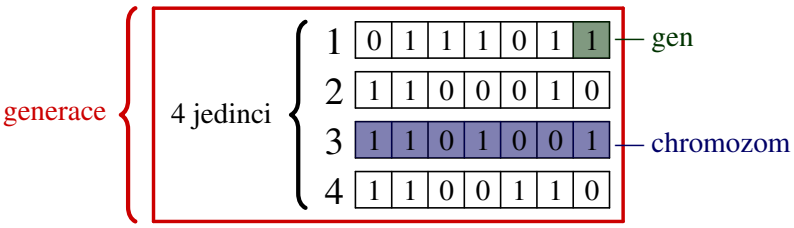
Svazková normalizace

Při procesu učení dochází ke změně vah v celé neuronové síti. Zároveň s tím se mění i rozdělení vstupních hodnot v každé vrstvě. Tato vlastnost musí být kompenzována vhodně zvolenou inicializací a malým koeficientem učení.

Alternativní řešení je použít svazkovou normalizaci [20], při níž je v jednotlivých vrstvách sítě použita standardizace vstupních dat z předchozí vrstvy u všech trénovacích dat ze svazku, na kterých se trénuje. Bylo ukázáno [20], že za použití této metody může nabývat koeficient učení vyšší hodnoty a zároveň nemusí být tak důkladně řešena inicializace vah. Dále se ukazuje, že za použití svazkové normalizace není nutné do sítě zahrnovat vysokou hodnotu dropoutu, protože se tímto krokem snižuje rychlost učení sítě. Ideální hodnota použité L2 regularizace je potom pětinásobně nižší, než jaká by byla použita u sítě bez svazkové normalizace.

1.2.6 Optimalizace hyper-parametrů pomocí genetického algoritmu

Genetický algoritmus (GA) je iterační metoda, pomocí které lze najít vhodné parametry některých problémů, jako například u optimalizace hyper-parametrů neuronové sítě, tedy parametrů, které se netrénují, ale volí se v rámci konstrukce sítě. GA je inspirovaný Darwinovou teorií přirozeného výběru [21] a používá se proto, aby nebyla potřeba zkoušet hrubou sílou všechny kombinace hyper-parametrů, což by bylo výpočetně velice náročné [22].



Obrázek 1.9: Vizualizace základních pojmů GA [21].

Označme tedy jednu sadu parametrů jako *jedinec* (nebo *individuum*) a všechny jedince v jedné iteraci jako *generaci*, viz obr. 1.9. Každý jedinec potom obsahuje všechny své parametry v tzv. *chromozomech*, což jsou soubory jednotlivých *genů* (které už obsahují danou číselnou hodnotu). Často se používá binární zápis parametrů, ale je možno jako parametr brát i samotné číslo. Celý algoritmus probíhá následovně [23].

1. Inicializuje se první generace.

2. Všichni jedinci v dané generaci se ohodnocují pomocí předem zvolené funkce (*fitness function*).

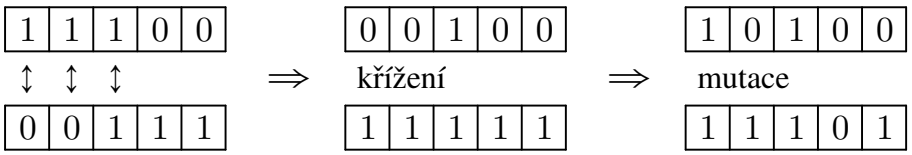
3. Jsou vybráni nejlepší jedinci a dochází ke křížení jejich chromozomů. Po této úpravě je ještě aplikována náhodná mutace na některé geny.

4. Předešlá generace je nahrazena nově vzniklou generací.

5. Body 2-5 se opakují podle potřeby.

Nejprve je tedy inicializována první generace (často náhodně). V rámci neuronových sítí může například každý jedinec obsahovat informaci o počtu neuronů a vrstev, případně i o použití regularizace, dropoutu a svazkové normalizace [23]. Poté se vybere způsob, jakým se budou vyhodnocovat jednotlivá individua. Pro tento účel je zkonstruována funkce, která přiřadí každému individuu číslo na základě jeho úspěšnosti v daném problému. V případě sítí to může být například validační přesnost, nebo nějakým způsobem transformovaná ztráta (hodnota ztrátové funkce).

V další fázi jsou vybráni ke křížení jedinci s pravděpodobností závislou na jejich ohodnocení (jedinci s lepšími vlastnostmi mají větší šanci na reprodukci) [24]. Samotné křížení probíhá tak, že jsou ze dvou jedinců vytvořeni noví jedinci na základě kombinace jejich chromozomů. Na kombinování chromozomů lze použít mnoho metod. Může se například vzít od každého jedince polovina, případně dát každému genu šanci 50%, že se přenese na potomka od každého rodiče, apod. [21][23] Tímto způsobem je ale možné získat pouze kombinaci genů z inicializované generace a parametry s obecně nejlepšími vlastnostmi by v nich nemusely být obsaženy vůbec. Tento problém lze sice vyřešit generací s velkým počtem jedinců, ale zároveň s tím by se zvýšila výpočetní náročnost celého algoritmu. Využívá se proto ještě metoda mutace genů [24].



Obrázek 1.10: Znázornění křížení a následná mutace chromozomů dvou jedinců.

Mutace funguje tak, že se po křížení chromozomů ještě náhodně se zadanou pravděpodobností změní hodnoty jednotlivých genů. Správné nastavení této pravděpodobnosti se ukazuje jako klíčové, protože 0% by znamenalo, že se v populaci nemusí vyskytovat potřebné geny a 100% by znamenalo, že by každá nová generace vznikala čistě náhodně. Vysoké číslo tedy obecně znamená, že se jedinci mění příliš rychle na to, aby bylo dosaženo požadované vlastnosti. V praxi se proto často používá hodnota okolo 1%.

Tímto krokem je získána nová generace a celý postup se může opakovat do té doby, než je dosaženo dostatečně dobrého výsledku.

V rámci experimentální části není tato metoda použita a je ponechána pro budoucí výzkum.

1.2.7 Obecný postup implementace neuronové sítě

Tvorba sítí za účelem vyhodnocování dat je popsána v následujících bodech.

1. **Návrh architektury** sítě. V prvním kroce je určen počet skrytých vrstev, počet neuronů v každé z nich, aktivační funkce, dropout apod. Obecně je vhodné síť implementovat s použitím regularizací, aby se zabránilo přeučení.

2. **Inicializace parametrů** modelu. V této fázi jsou iniciovány váhy sítě.

3. **Učení modelu** (iterací přes trénovací data):

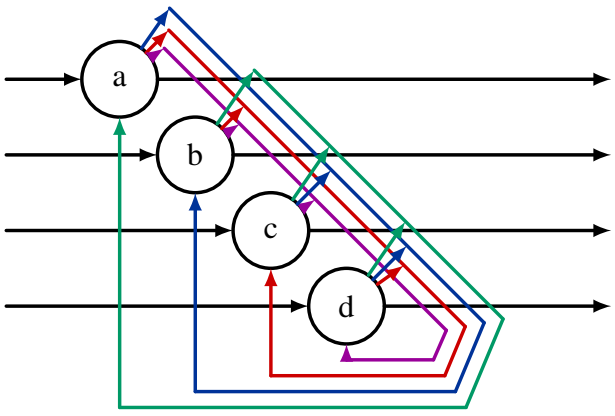
   - Je vypočítán výstup sítě na základě trénovacích dat.
   - Určí se hodnota ztrátové funkce.
   - Pomocí zpětného šíření jsou upraveny hodnoty parametrů sítě. Tento postup je opakován dle potřeby.

4. **Aplikace** natrénované sítě na nová data, tzv. *vybavování*.

1.3 Asociativní paměti

1.3.1 Hopfieldovy sítě

Tyto sítě navrhnul J. J. Hopfield roku 1982. Jednalo se o typ binárních rekurentních sítí, které sloužily jako jednoduchý model paměti. V té době poskytovaly dokonce model fungování lidské paměti. Hopfieldova síť totiž v rámci učení upravuje své váhy tak, aby si zapamatovala určitý vzor. Nová data jsou pak iterativní metodou transformována do těchto naučených vzorů.



Obrázek 1.11: Jednoduchá Hopfieldova síť o 4 neuronech.

V hopfieldově síti neurony nabývají hodnot -1 a 1 namísto obvyklých 0 a 1 [25]. Nechť

$$\boldsymbol{V} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \ \mathbb{W} = \begin{bmatrix} w_{aa} & w_{ab} & w_{ac} & w_{ad} \\ w_{ba} & w_{bb} & w_{bc} & w_{bd} \\ w_{ca} & w_{cb} & w_{cc} & w_{cd} \\ w_{da} & w_{db} & w_{dc} & w_{dd} \end{bmatrix} \overset{\underline{obvykle}}{=\!=} \begin{bmatrix} 0 & w_{ab} & w_{ac} & w_{ad} \\ w_{ab} & 0 & w_{bc} & w_{bd} \\ w_{ac} & w_{bc} & 0 & w_{cd} \\ w_{ad} & w_{bd} & w_{cd} & 0 \end{bmatrix}, \quad (1.20)$$

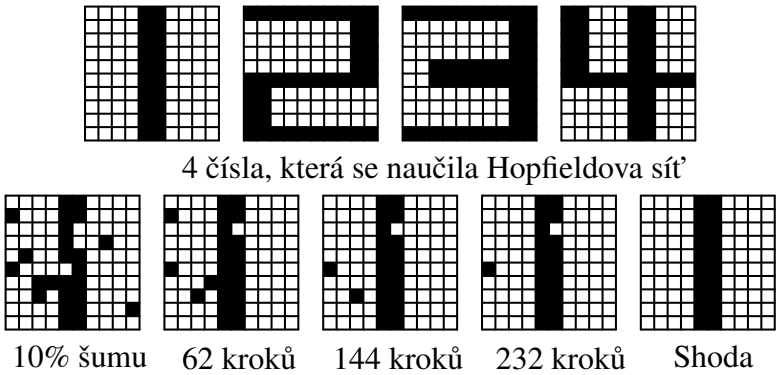
kde $\boldsymbol{V}$ je vektor neuronů a $\mathbb{W}$ je symetrická matice vah s nulami na diagonále (protože žádný neuron není spojen sám se sebou). Dá se ukázat [11], že hodnota tzv. energetické funkce

$$E := -\frac{1}{2}\sum_i \sum_j w_{ij} V_i V_j = -\frac{1}{2}\boldsymbol{V}^T \mathbb{W} \boldsymbol{V} \quad (1.21)$$

klesá monotonně při iteraci

$$V_i = f(x_i) \quad (1.22)$$

$$f(x_i) = \begin{cases} -1 & x_i = \sum_{j \neq i} w_{ij} V_j > 0, \\ 1 & x_i = \sum_{j \neq i} w_{ij} V_j < 0. \end{cases} \quad (1.23)$$



Obrázek 1.12: Ukázka použití Hopfieldovy sítě
upraveno z https://res.infoq.com/articles/emotion-cognition/en/resources/6_hopfield_net.jpg

Celkově vzato tedy Hopfieldova síť funguje tak, že se nejprve naučí některé vzory, (viz obr. 1.12). Po předložení nových dat, resp. poškozených dat (například vlivem šumu), tato síť začne upravovat data výše uvedenou iterací tak, aby se snižovala celková energie systému. Iterace končí ve chvíli, kdy systém dosáhne na lokální minimum ($\Delta E = 0$). V ideálním případě mohou dokonce takto upravená data vypadat identicky s naučeným vzorem (obecně se ale očekává jistá odchylka).

Od té doby bylo vyvinuto mnoho dalších metod, častokrát s lepšími výsledky, než měla Hopfieldova síť. Jednou z nich jsou již uvedené vrstevnaté neuronové sítě a na ně úzce navazující konvoluční neuronové sítě. Ty narozdíl od Hopfieldových sítí nejsou binární, což rozšiřuje využití i na mnohem rozmanitější a komplexnější problémy. Například u rozpoznávání černobílých obrázků mohou být rozlišeny všechny odstíny šedi, nikoliv pouze černá a bílá. Vrstevnaté neuronové sítě mohou mít oproti Hopfieldovým sítím složitější

strukturu, protože v nich není počet neuronů svázaný s počtem vstupů. Hopfieldovy sítě jsou rekurentní, zatímco vrstevnaté neuronové sítě ze základu nejsou. V roce 1997 však vědci Hochreiter a Schmidthuber vyvinuli tzv. *LSTM* sítě (*Long Short-Term Memory*), které fungují na základě vrstevnatých neuronových sítí a zároveň jsou rekurentní [26].

1.4 Kohonenova samoorganizační mapa

Kohonenova samoorganizační mapa je model vyvinutý finským profesorem Teuvo Kohonenem [27]. Zatímco vrstevnaté sítě se učí na základě upravování koeficientů pomocí chybové funkce, u Kohonenových map se využívá tzv. *metody soutěže*.

Kohonenova mapa je síť tvořena dvěma vrstvami, vstupní a výstupní vrstvou. Není zde použita žádná aktivační funkce a každý neuron z výstupní vrstvy má stejný počet vah, jako je počet dimenzí u vstupních dat [28]. Tato situace se dá chápat tak, že v n-dimenzionálním prostoru váhy neuronů představují souřadnice jednotlivých neuronů. Proces trénování pak probíhá třemi procesy: soutěžení, kooperace a adaptace.

Soutěžení

V rámci soutěžení se vypočítá vzdálenost trénovacího prvku a všech neuronů. Soutěž vyhrává neuron, který je mu nejblíže [28]. K tomu se většinou používá Eukleidovská metrika.

Kooperace

V rámci kooperace se podle poloměru vytipují neurony, které jsou vítěznému neuronu nejblíže.

Adaptace

V tomto kroku se mění váhy vybraných neuronů pomocí vzorce [27]

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \alpha_i(t)\beta_{ij}(t)[x(t) - \omega_{ij}(t)], \tag{1.24}$$

kde $\omega_{ij}(t)$ je i-tá váha j-tého neuronu v iteraci podle t,

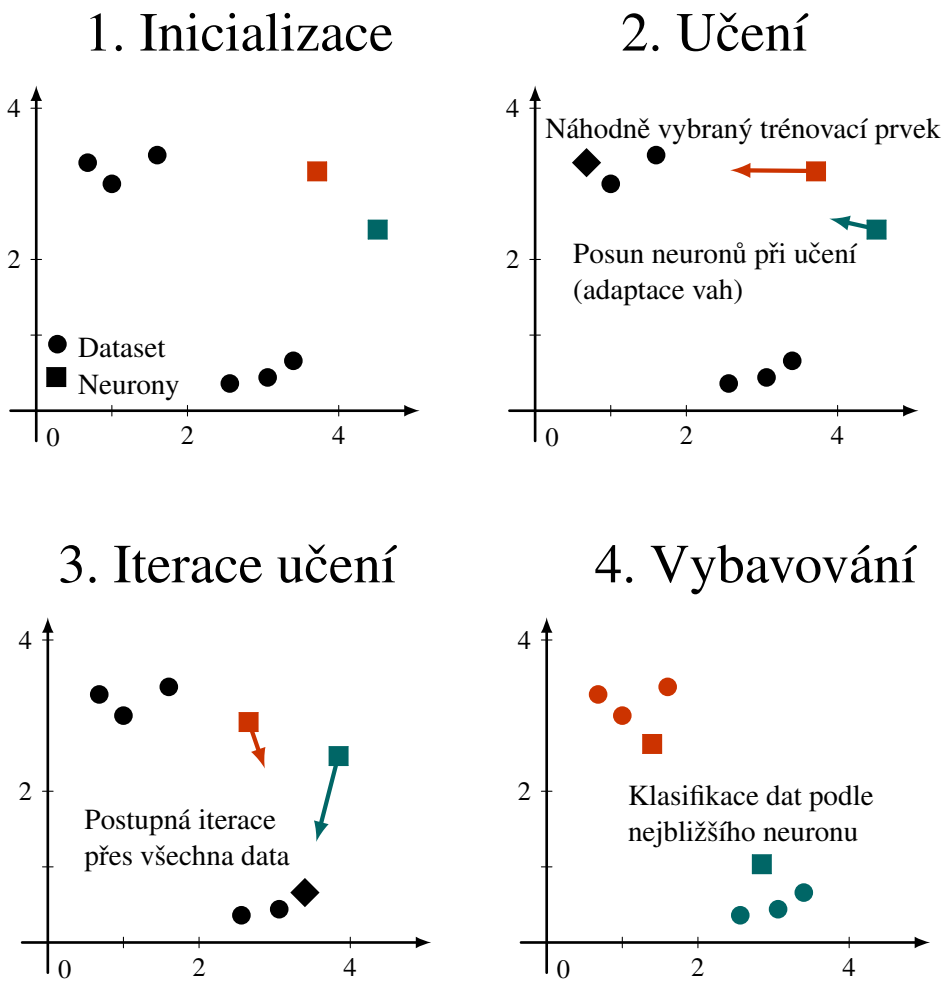
$$\alpha_i(T) = \alpha_{i0}\exp\left(\frac{-t}{\lambda}\right), \text{ kde } \alpha_{i0} \text{ je konstanta } i\text{-tého neuronu,}$$

$$\beta_{ij}(t) = \exp\left(\frac{-d_{ij}^2}{2\sigma^2(t)}\right), \text{ kde } d_{ij} \text{ je vzdálenost } i\text{-tého a } j\text{-tého neuronu a}$$

$$\sigma(t) = \sigma_0\exp\left(\frac{-t}{\lambda}\right), \text{ kde } \sigma_0 \text{ je konstanta.}$$

Vybavování

Klasifikace (vybavování) probíhá tak, že je všem datům přiřazena kategorie příslušící nejbližšímu neuronu.

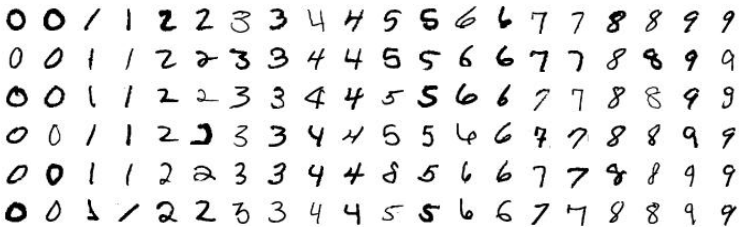


Obrázek 1.13: Vizualizace fungování kohonenových map.

Kapitola 2

Konvoluční neuronové sítě

Pro vektorová data se vrstevnaté sítě osvědčily. Pokud je však řešený problém maticové, či tenzorové povahy (např. klasifikace obrázků), musela by se data pro vrstevnaté sítě nejprve vektorizovat, čímž by zanikly důležité informace o uspořádání jednotlivých prvků. Například z obrázku kruhu by tak vznikla soustava úseček. I pouhé posunutí obrázku o 1 pixel by pak výrazně změnilo hodnoty vektoru. Z tohoto důvodu není úspěšnost vrstevnatých sítí příliš vysoká. Postupem času proto vznikla metoda, ve které se informace o uspořádání zachovávají. Touto metodou jsou *konvoluční neuronové sítě* (*convolutional neural network, CNN*). CNN se dá chápat i jako vícerozměrná neuronová síť, kde se v první (konvoluční) části používají místo vah matice konvoluce. Celá metoda se pak dělí na několik částí.

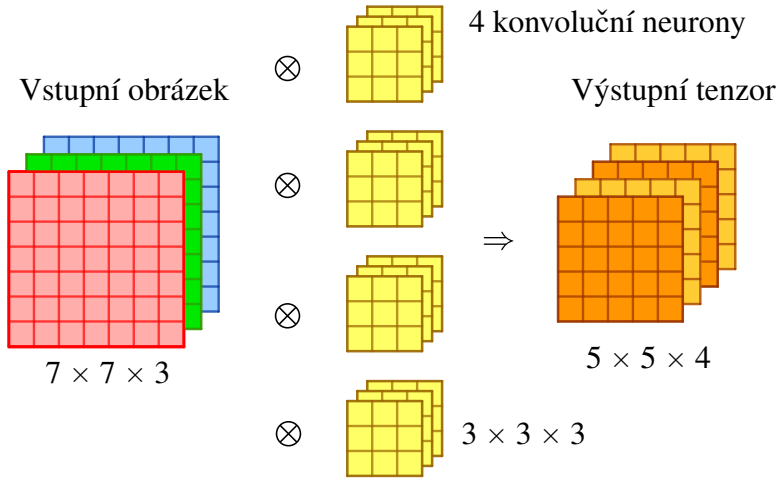


Obrázek 2.1: Příklad databáze ručně psaných číslic MNIST, na kterou lze úspěšně použít konvoluční neuronovou síť.

2.1 Konvoluce

Do této fáze vstupují data obecně v tenzorové podobě, přičemž pro účely klasifikace obrázků (viz obr. 2.1) mají data 3 rozměry. První dva rozměry jsou výška a šířka obrázku a třetí udává počet barevných kanálů. V první části CNN je tento tenzor transformován [29] pomocí konvolučního jádra, viz obr. 2.2. Jeho první dva rozměry jsou voleny dle potřeby už při konstrukci sítě, třetí pak musí odpovídat počtu kanálů vstupního obrázku. Každý konvoluční neuron tedy sestává z tenzorového jádra a prahové hodnoty, na kterou lze pohlížet podobně jako u vrstevnatých sítí. V praxi se velikost jádra obvykle volí ve tvaru $(3, 3, z)$, ale mohou být použity i jiné hodnoty. Vzhledem k tomu, že třetí rozměr

konvolučního jádra je pevně daný povahou vstupních dat, často se v zápisu vynechává. Příkladem toho je i pythonová knihovna Keras, kde se zadává pouze rozměr ve tvaru $(3,3)$ [30]. Výstupem každého neuronu je pak matice [29], která má první dva rozměry zmenšeny o [*rozměry konvolučního jádra*-1], protože je jádro aplikováno pouze tak, aby nepřečnívalo okraje obrázku. Výstup celé vrstvy CNN je potom kompozice matic od jednotlivých neuronů. Celkově tedy třetí rozměr výstupu odpovídá počtu neuronů použitých v dané konvoluční vrstvě. Tento tenzor se nazývá *mapa charakteristik*.



Obrázek 2.2: Struktura konvoluční části CNN. RGB obrázek je zde transformován pomocí 4 konvolučních neuronů o rozměru $(3,3)$ do podoby tenzoru. První dva rozměry tenzoru jsou menší kvůli povaze konvoluce a třetí rozměr odpovídá počtu neuronů.

Tento přístup je užitečný v tom, že každý neuron konvoluční vrstvy extrahuje z obrázku jiný typ informací. Celkově je tedy na obrázek pohlíženo z více perspektiv, což umožňuje snažší hledání souvislostí v prostorové uspořádanosti jednotlivých pixelů. Příklad konvoluční transformace podle matic 2.1 je znázorněn na obr. 2.3. Zde jsou však oproti konvolučním neuronům transformovány jednotlivé vrstvy zvlášť. Například pro první transformační matici ve výsledku vyniknou hrany obrázku, ve druhém případě plochy.

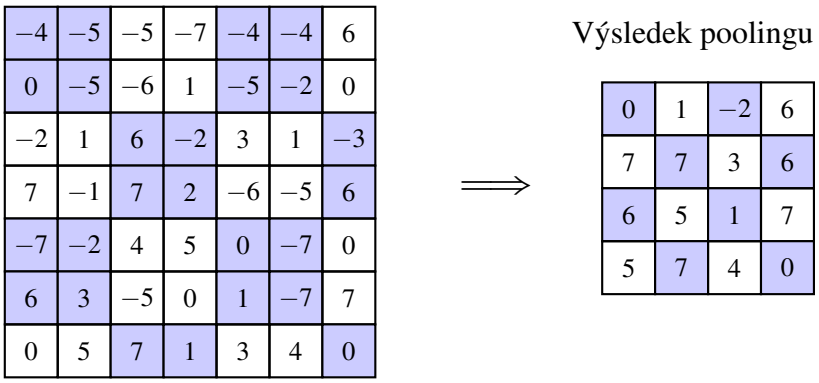
$$\mathbb{A}_1 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -8 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \mathbb{A}_2 = \frac{1}{9}\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \mathbb{A}_3 = \begin{pmatrix} -6 & 0 & 5 & -2 & 4 \\ 2 & -5 & 1 & 5 & -2 \\ 1 & 5 & 1 & 1 & 2 \\ -7 & -5 & 4 & 5 & -2 \\ -1 & -3 & 1 & 3 & 1 \end{pmatrix}. \tag{2.1}$$

Obrázek 2.3: Obrázek květu a jeho konvoluce podle matic $\mathbb{A}_1$, $\mathbb{A}_2$, $\mathbb{A}_3$.

U konvoluce existuje více variant, jak se vypořádat s okrajem obrázku. Jednou z alternativních možností je přidání nul po okrajích, tedy přidání řádku i sloupce nul na začátek i na konec, díky čemuž by měl výsledný obrázek po konvoluci stejný rozměr. V praxi se ale častěji používá již zmíněná varianta, při které probíhá konvoluce pouze v rámci původního obrázku, čímž dochází ke zmenšení prvních dvou rozměrů. Například pro matici konvoluce o rozměru $(3, 3)$ bude mít obrázek po konvoluci o 2 sloupce a 2 řádky méně. Tento přístup se používá proto, aby zároveň došlo k redukci dat. Další možností, jak redukovat data v síti je použít vrstvu, která se nazývá *pooling*.

2.2 Pooling

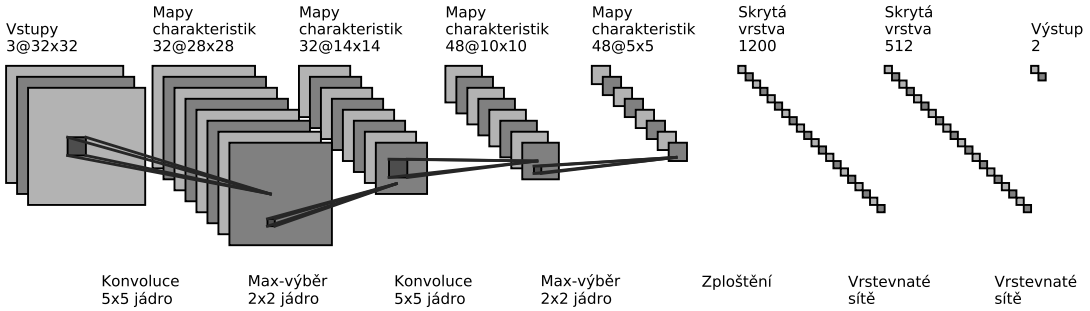
Cílem konvoluční vrstvy je postupně převést vícerozměrná data do jednoho rozměru tak, aby se na ně následně dala použít vrstevnatá neuronová síť. Za tímto účelem je rovněž výhodné redukovat data, která nejsou pro daný úkol potřebná. K tomu slouží tzv. poolingová vrstva, což je nelineární transformace, která výrazně redukuje množství dat v tenzoru. Při této transformaci se iteruje přes jednotlivé vrstvy tenzoru maticí zvoleného rozměru tak, aby se žádné 2 oblasti nepřekrývaly, viz obr. 2.4. Na tyto oblasti je pak aplikována funkce ve tvaru $f : \mathbb{A} \to \mathbb{R}$. Většinou se pro tento účel volí maximum, ale může jím být i průměr apod.



Obrázek 2.4: Příklad maximového poolingu u matice s lichým rozměrem.

2.3 Zploštění

Poté, co jsou pomocí opakování konvoluce a poolingu zmenšeny první dva rozměry tenzoru na dostatečně nízkou hodnotu (ideálně kolem $(3 \times 3 \times z)$), je aplikována poslední vrstva konvoluční části sítě, tzv. *zploštění* (*flattening*). To má za úkol převést mapu charakteristik na vektor, na který se následně použijí vrstevnaté sítě. Na obrázku 2.5 je možno vidět příklad struktury typické konvoluční neuronové sítě. Jako vstup zde slouží obrázek o 3 barevných kanálech s rozměrem (32×32). Na něj je poté aplikována konvoluční vrstva o 32 neuronech a výsledná mapa charakteristik je následně zredukována poolingovou vrstvou. To samé je pak opakováno ještě jednou, ale se 48 neurony u konvoluční vrstvy. Výsledný tenzor je pak zploštěn do vektorové podoby, na kterou jsou aplikovány



Obrázek 2.5: Znázornění architektury běžné konvoluční neuronové sítě.
Vytvořeno úpravou generátorového kódu z https://github.com/gwding/draw_convnet.

dvě skryté vrstvy o 1200 a 512 neuronech. Výstupní vrstva má pouhé 2 neurony, což odpovídá například binárnímu klasifikátoru.

2.4 Učení CNN

Stejně jako v případě vrstevnatých sítí, i konvoluční neuronová síť musí nejprve inicializovat své parametry. Potom už probíhá učení na základě trénovacích dat. Vrstevnatá část sítě pro učení využívá zpětné šíření popsané v sekci 1.2.3. Konvoluční část sítě pak trénuje parametry v konvolučních vrstvách pomocí zpětného šíření modifikovaného pro použití na tenzorech (více v [31]). V poolingových vrstvách žádné trénovací parametry nejsou, a proto k jejich učení nedochází.

Opět lze docílit vyšší přesnosti za použití svazku dat namísto jednotlivých prvků. To však klade větší požadavky na hardware použitý při trénování. Při svazkovém učení je totiž potřeba uložit do paměti všechny změny parametrů sítě pro jednotlivé vzorky ve svazku zvlášť. Velikost svazku se proto musí volit tak, aby se předešlo problému s pamětí.

2.5 HFD spektrogram

Adekvátním postupem při použití konvoluční sítě na vektorová data je využít vhodnou formu časo-frekvenční interpretace. Jednou z běžně používaných metod je *krátkodobá Fourierova transformace* (*short-time Fourier transform, STFT*), při které se počítá diskrétní Fourierova transformace na krátkém úseku. Tímto způsobem je získána informace o výskytu frekvencí v signálu v době odpovídající délce časového okna. Diskrétní Fourierova transformace (*DFT*) se vypočítá pomocí vztahu

$$F(m) = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\frac{2\mathrm{i}\pi mn}{N}}. \tag{2.2}$$

Při STFT dochází k dilematu mezi frekvenčním a časovým rozlišením, které řeší např. Waveletová transformace. Ta ale podstatně závisí na tvaru použité vlnky. Jako alternativní metoda umožňující libovolné pokrytí frekvenční osy bylo použito zobecnění spektro-

gramu, které nazýváme *HFD* (*High Frequency Density*) *spektrogram*, definovaný po odvození vzorcem 2.9. Při jeho použití je navíc zachována konzistence zkoumaných frekvencí, pokud se transformují různě dlouhé úseky signálu.

Zaveďme nyní časo-frekvenční transformaci tak, aby zobrazovala míru korelace signálu na intervalu $[a,b]$ v závislosti na sinu o frekvenci f, tedy $\sin(2\pi f \cdot t)$. Míru korelace spočítáme vztahem

$$F(f) := \max_{\varphi} \int_a^b \sin(2\pi f t + \varphi) s(t) \mathrm{d}t, \tag{2.3}$$

kde $s(t)$ je zkoumaný signál, přičemž nemusí být známo, s jakou fází signálu nastává nejvyšší úroveň korelace. Cílem je tedy najít maximum korelace zkoumaného signálu přes všechna posunutí referenčního sinu. Počítat tuto rovnici numericky by bylo příliš náročné, a proto se nejprve upraví do výhodnější podoby. Za tímto účelem je použita součtová funkce pro sinus ve tvaru

$$\sin(2\pi f t + \varphi) = \sin(2\pi f t)\cos(\varphi) + \cos(2\pi f t)\sin(\varphi). \tag{2.4}$$

Dále tedy můžeme postupně upravovat:

$$\begin{aligned}
F(f) &:= \max_{\varphi} \int_a^b \Big(\sin(2\pi f t)\cos(\varphi) + \cos(2\pi f t)\sin(\varphi)\Big) s(t)\mathrm{d}t = \\
&= \max_{\varphi} \left(\cos(\varphi) \underbrace{\int_a^b \sin(2\pi f t) s(t)\mathrm{d}t}_{c} + \sin(\varphi) \underbrace{\int_a^b \cos(2\pi f t) s(t)\mathrm{d}t}_{d} \right) = \\
&= \max_{\varphi} \Big(c\cos(\varphi) + d\sin(\varphi)\Big) = \max_{\varphi} \left[\sqrt{c^2+d^2} \cdot \sin\left(\varphi + \arccos\left(\frac{c}{\sqrt{c^2+d^2}}\right)\right)\right] = \\
&= \sqrt{c^2+d^2}.
\end{aligned} \tag{2.5}$$

Tento výraz lze interpretovat jako absolutní hodnotu komplexního čísla, která je nezávislá na operaci komplexního sdružení. Podle Eulerovy formule lze psát

$$F(f) := \sqrt{c^2+d^2} = |d - \mathrm{i}c| = \left| \int_a^b \mathrm{e}^{-2\pi \mathrm{i} f t} s(t)\mathrm{d}t \right|. \tag{2.6}$$

Dostáváme tedy výraz obdobný Fourierově transformaci

$$\mathfrak{F}[s(t)](f) = \int_{-\infty}^{+\infty} \mathrm{e}^{-2\pi \mathrm{i} f t} s(t)\mathrm{d}t. \tag{2.7}$$

Vztah 2.6 představuje zjemnění odhadu spektra oproti standardní Fourierově transformaci. V diskrétním případě přechází integrál 2.6 na sumu přes jednotlivé vzorky podle zvoleného časového okna $\omega(x)$. Zároveň s tím je zrelativizována frekvence v závislosti na tzv. *Nyquistově frekvenci* N_q, tedy $f = f_{rel} \cdot N_q$, kde $N_q = \frac{1}{2\Delta_s}$. Pro signál o N vzorcích

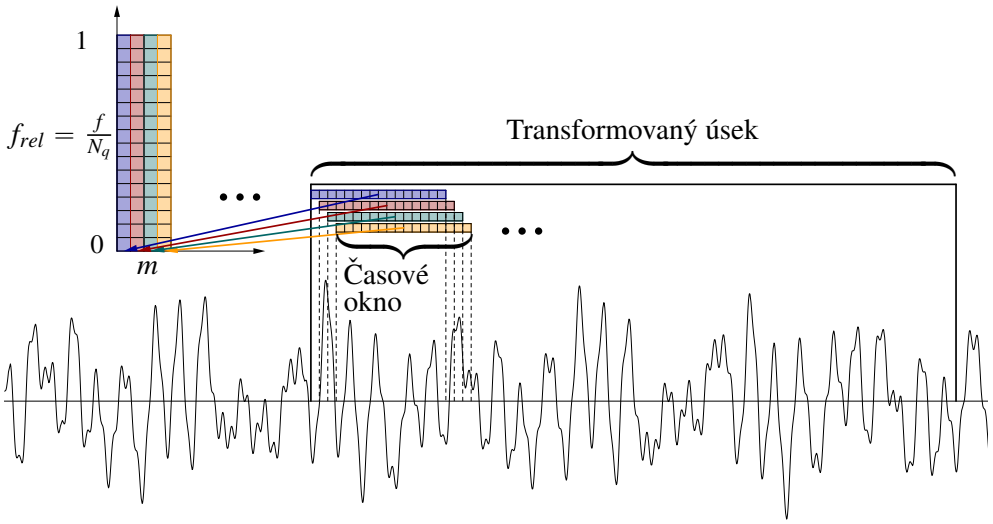
se vzorkovací periodou Δ_s potom platí vztah

$$\overline{F}(m, f_{rel} \cdot N_q) = \left| \sum_{n=0}^{N-1} x(n\Delta_s) \mathrm{e}^{-2\pi \mathrm{i} f n \Delta_s} \omega(n-m) \right| = \left| \sum_{n=0}^{N-1} x(n\Delta_s) \mathrm{e}^{-2\pi \mathrm{i} \cdot f_{rel} \frac{1}{2\Delta_s} \cdot n\Delta_s} \omega(n-m) \right| =$$
$$= \left| \sum_{n=0}^{N-1} x(n\Delta_s) \mathrm{e}^{-\pi \mathrm{i} n f_{rel}} \omega(n-m) \right|, \tag{2.8}$$

kde f_{rel} je relativní frekvence vůči N_q. Nyquistova frekvence se k relativizaci používá, protože bylo dokázáno, že je možno analýzou diskrétního signálu získat informace pouze o frekvencích nižších, než N_q. Z předešlého vztahu se následně vypočítá odhad frekvenčního spektra pro každé posunutí m časového okna, tj. všechny sloupce spektrogramu, s využitím vztahu pro odhad výkonového spektra $P(x) = |\overline{F}(x)|^2$:

$$P(m, f_{rel} \cdot N_q) = \left| \sum_{n=0}^{N-1} x(n\Delta_s) \mathrm{e}^{-\pi \mathrm{i} n f_{rel}} \omega(n-m) \right|^2. \tag{2.9}$$

Jak již bylo zmíněno výše, tento vzorec definuje transformaci signálu, tzv. HFD spektrogram, který lze použít jakožto časo-frekvenční interpretaci signálu. Nejprve je zvolena délka a tvar časového okna, které bude použito na výpočet diskrétní transformace, a celkový počet těchto oken. Z každého okna po transformaci vznikne jeden sloupec ve výsledné matici. Výška těchto sloupců je dána volbou frekvencí, které jsou použity ve výpočtu spektrogramu. Každý řádek tedy odpovídá HFD spektrogramu pro jinou relativní frekvenci vzhledem k Nyquistově frekvenci, která je známá ze zadání experimentu.



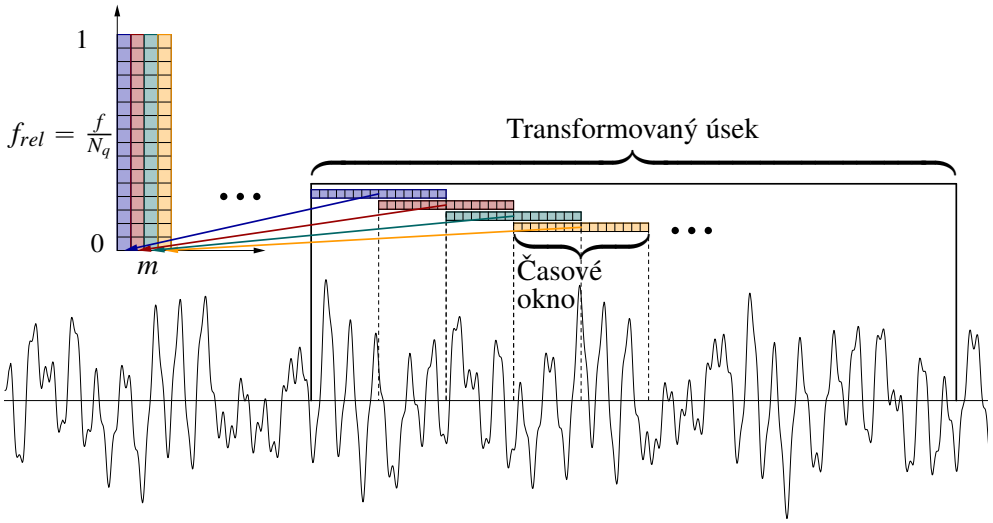
Obrázek 2.6: Znázornění HFD spektrogramu s posunem časového okna o jeden vzorek signálu, tedy $k = 1$. Transformovaným úsekem se rozumí celkový úsek signálu, který je pokrytý alespoň jedním časovým oknem.

V tomto procesu není nutné, aby byly transformovány úseky, které se nachází těsně vedle sebe, tedy podle vzorce 2.9. Stejně tak mohou být časová okna posunovány o jiný

počet vzorků. Za tímto účelem lze upravit rovnici 2.9 do tvaru

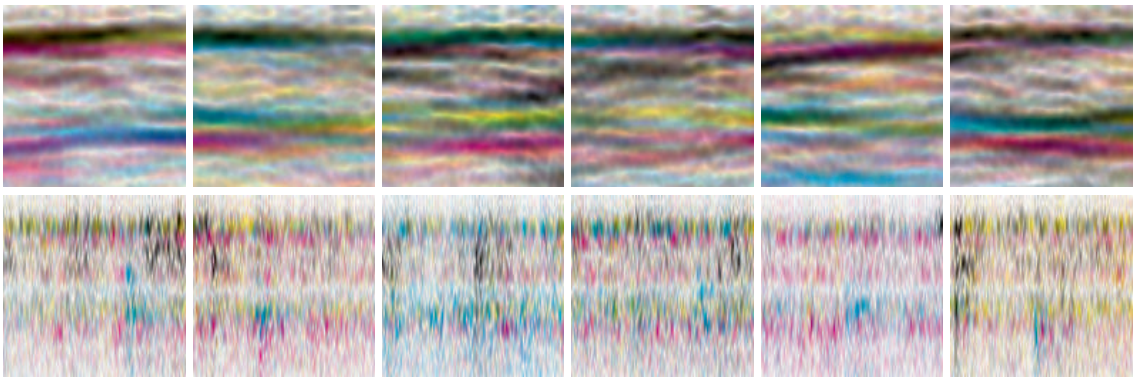
$$P(m, f_{rel} \cdot N_q) = \left| \sum_{n=0}^{N-1} x(n\Delta_s) \mathrm{e}^{-\pi \mathrm{i} n f_{rel}} \omega(n - k \cdot m) \right|^2. \tag{2.10}$$

Takto použitý HFD spektrogram je znázorněný na obr. 2.6 a 2.7. Při větším posunutí jednotlivých oken je celkově transformován delší časový úsek signálu.



Obrázek 2.7: Znázornění HFD spektrogramu s delším časovým posunem jednotlivých časových oken, tedy $k > 1$. Transformovaným úsekem se rozumí celkový úsek signálu, který je pokrytý alespoň jedním časovým oknem.

Pro teoreticky korektní spektrální analýzu by měla být aplikována známá časová okna, jako např. Hannovo, Welchovo apod. V rámci této práce bylo používáno pouze obdélníkové okno, protože předběžné testování ukázalo, že na tvaru časového okna klasifikační výsledky příliš nezávisí.



Obrázek 2.8: Ukázka použití HFD spektrogramu. V obou případech je délka časového okna 128 vzorků. První řada byla transformována s posunem o jeden vzorek, kdežto ve druhé byl použitý posun o 64 vzorků (tedy $k = 64$).

Pomocí HFD spektrogramu je tedy vektor signálu transformován do maticové podoby.

K dispozici jsou ale data ze 4 kanálů. Nabízí se proto tuto transformaci provést na všech čtyřech kanálech zároveň a sloučit výsledná data do jednoho trojrozměrného tenzoru.

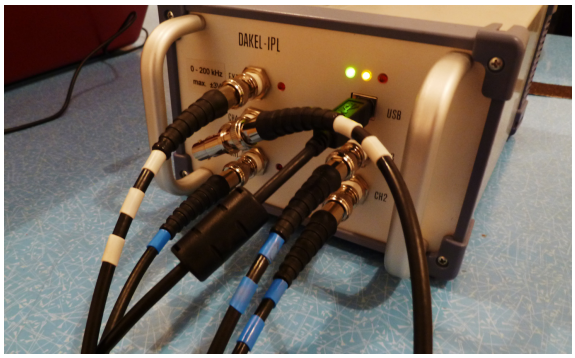
Způsobů uložení tohoto tenzoru existuje celá řada. Například pro 4 kanály se jeví jako nejlepší možnost uložení výsledku v podobě CMYK, což je kombinace azurové, purpurové, žluté a černé barvy, viz obr. 2.8. Výhodou tohoto formátu je podpora knihoven, které umožňují výsledná data komprimovat a šetřit tím místo na disku.

Kapitola 3

Experimenty

3.1 Popis měření signálu

Při zkoušce renovovaného kusu reduktoru vrtulníku bylo provedeno měření spojité akustické emise. Nespojitá (impulzní) akustická emise je generována nevratnými dislokačními a degradačními procesy v mikrostruktuře a makrostruktuře materiálu (např. aktivní měnící se trhlinou) v podobě časově oddělených napěťových pulzů trvajících od několika nanosekund do jednotek milisekund. V případě reduktoru se složitými mechanickými rozvody je však nutné předpokládat podstatně složitější události, které nejsou časově oddělené. Mohou být generovány řadou fyzikálních jevů, jako jsou např. plastické deformace povrchu kovů při tření, či únik kapaliny trhlinou v potrubí nebo nádobě. Jedná se o tzv. spojitou akustickou emisi. Díky současné nejmodernější měřící aparatuře je možné tyto akustické signály kontinuálně zaznamenávat i v ultrazvukové části frekvenčního spektra.

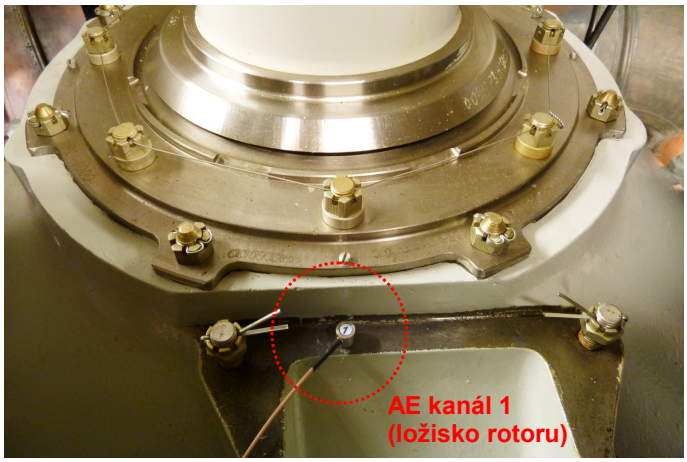


Obrázek 3.1: Analyzátor DAKEL IPL.

Analyzátor IPL firmy Dakel (viz obr. 3.1) je systém umožňující 12-bitové synchronní kontinuální vzorkování 4 kanálů frekvencí 2 MSample/s a nepřetržité ukládání navzorkovaných dat do paměti PC. Komunikace s PC a přenos dat je zajišťován rozhraním USB2.0. Data jsou na disk ukládána rychlostí kolem 16 MB/s, což představuje zhruba 56 GB na hodinové měření, přičemž celková doba měření je v podstatě omezena jen kapacitou diskového prostoru, který je schopen pracovat nepřetržitě alespoň v takové rychlosti ukládání.

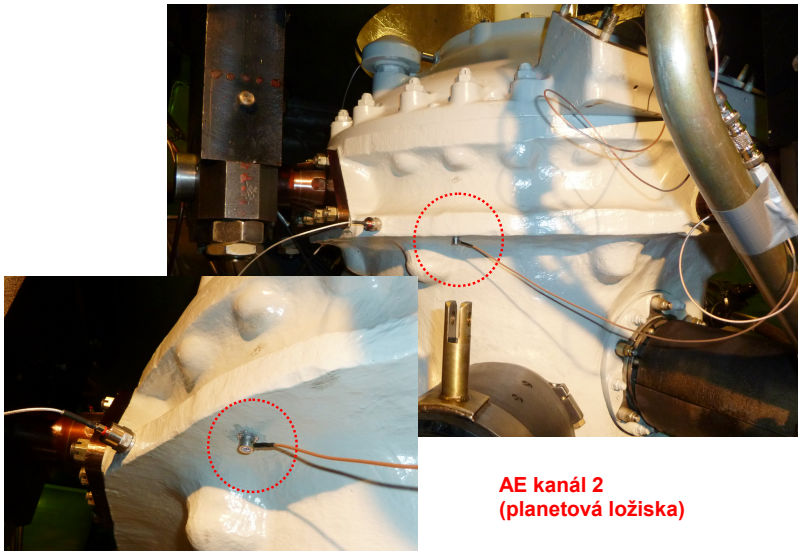
Po ukončení měření je možné zpracování naměřených dat programem Dakel UI, který

pracuje pod operačním systémem Linux. Tento software poskytuje různé možnosti zpracování dat. Pokročilé matematicko-statistické metody zaměřené na identifikaci emisních zdrojů v něm však nejsou implementovány. Proto byl dodávaný software využit pouze pro ovládání měřícího zařízení a konverzi uložených binárních dat do jednoduššího formátu, který lze načítat např. v prostředí Python.



Obrázek 3.2: Umístění snímačů AE pro monitoring spojek motorů.

Snímače typu Dakel IDK-09 (s rezonanční frekvencí mezi 200-300kHz) byly rozmístěny podle potřeb monitoringu kritických částí převodovky a možností jejich uchycení na povrch reduktoru, tj. na fyzicky dostupná rovná místa blízko sledovaných součástí. Snímače AE lze po obroušení, nebo odmaštění spojovaných povrchů lepit přímo na převodovku kyanoakrylátovým lepidlem.



Obrázek 3.3: Umístění snímače AE pro monitoring planetových ložisek.

Po konzultaci s experty provádějícími opravy a údržbu byly zvoleny čtyři kritické součásti:

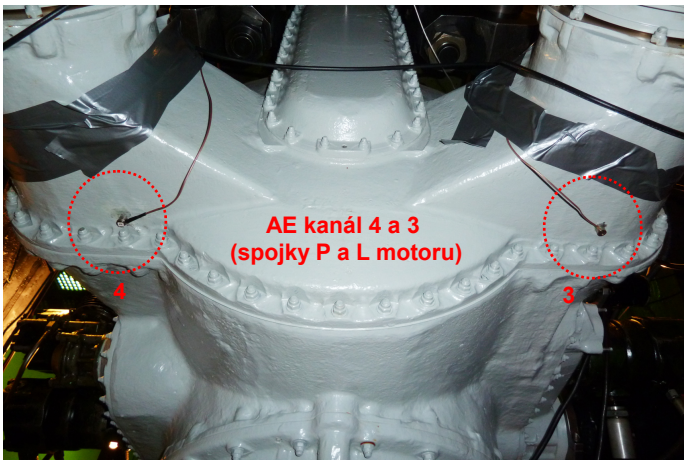
1. Ložisko hlavního rotoru (viz obr. 3.2)

2. Planetová ložiska (viz obr. 3.3)

3. Spojka levého turbomotoru (viz obr. 3.4)

4. Spojka pravého turbomotoru (viz obr. 3.4)

Na vhodná místa na povrchu reduktoru byly nejblíže výše uvedeným součástem upevněny snímače AE, které jsou opatřeny krátkým cca 1m koaxiálním kabelem o průměru 1.7 mm s BNC konektorem se silikonovou izolací odolávající teplotám -20°C až +75°C. Snímač vyžaduje externí koaxiálový impedanční převodník s předzesilovačem, umožňující připojení dalšího koaxiálního kabelu pro přenos signálu k cca 30m vzdálenému analyzátoru IPL.



Obrázek 3.4: Umístění snímačů AE pro monitoring spojek motorů.

3.2 Úvod do experimentů

Experimenty byly původně provedeny s cílem ověřit možnosti detekce poruch v převodových ústrojích měřením povrchových elastických vln v ultrazvukové frekvenční oblasti, což by umožnilo hodnocení stavu sledované součástky a případný odhad její zbytkové životnosti.

V rámci experimentální části práce byla analyzována data z 11 různých letových režimů o celkovém objemu 55.5GB a zadáním bylo navrhnout způsob, jak pomocí neuronové sítě rozpoznat aktuální letový režim na základě signálů, které nebyly použity k jejímu učení.

Tento problém lze řešit několika způsoby. Jedním z nich je přímé aplikování vrstevnaté sítě na zkoumaný signál, což je sice jednoduché na implementaci, avšak neposkytuje žádoucí výsledky a je citlivé na posuny signálu. To znamená, že posun o jeden vzorek může zcela změnit vnitřní hodnoty v celé síti. Další možností je potom nejprve na signál aplikovat časo-frekvenční transformaci a teprve na takto transformovaná data použít konvoluční neuronové sítě. V této kapitole budou oba přístupy experimentálně prověřeny a budou zhodnoceny jejich přínosy a nedostatky.

3.3 Programové prostředí

Existuje mnoho programovacích jazyků a v nich spousta knihoven, které by byly vhodné pro tyto experimenty. Mezi nimi jsou například Python, Java, C++ a MATLAB. Použití C++ je výhodné zejména tehdy, pokud je kladen důraz na vysokou rychlost výpočtu. Pokud je naopak potřeba rychle a jednoduše vyvíjet software, potom je vhodnější použít například Python.

V této práci je softwarová část psaná v Pythonu v rámci pythonové distribuce *Anaconda* (https://anaconda.org/). Tato distribuce zjednodušuje správu knihoven. K tomuto účelu využívá *správce balíčků* (*package manager*) s názvem *conda*. V praxi to znamená, že je možné vytvořit několik pythonových prostředí s různými verzemi knihoven a podle potřeby mezi nimi přepínat.

Pro vytvoření prostředí v linuxové distribuci Ubuntu 18.04.4 LTS byl aplikován následující kód:

```
$ conda create -n tf keras-gpu=2.2.0
$ conda activate tf
$ conda install opencv matplotlib scikit-learn Pillow
        sympy numpy scipy
$ conda install -c conda-forge pyglet
```

Tímto způsobem bylo vytvořeno prostředí s názvem *tf* založené na knihovně Keras-GPU ve verzi 2.2.0. Tato knihovna je postavená na knihovně Tensorflow-GPU a výrazně usnadňuje práci s neuronovými sítěmi za použití grafického procesoru.

3.4 Dataset

Pro úspěšné trénování neuronových sítí je potřeba velké množství trénovacích dat. Ty je nutno rovnoměrně vybírat z jednotlivých letových režimů a následně i rovnoměrně rozdělit na trénovací, validační a testovací. Například pokud by byla síť trénována na datech z prvních 8 letových režimů a validována na zbylých 3, neposkytovala by pro nová data relevantní výsledky, protože by nebyla schopna rozpoznat všechny režimy. V této práci je pro každý letový režim použito 9000 časových úseků, z nichž 64% je využito jako trénovací data, 16% jako validační data a 20% jako testovací data.

Rozdělení na 3 části je výhodné zejména pro optimalizaci hyper-parametrů sítě, např. pomocí genetického algoritmu. Na počátku totiž není možné předpovědět, jaké rozměry sítě budou ideální, protože v současné době neexistuje žádná metoda, která by určila, kolik použít vrstev, kolik neuronů mají obsahovat jednotlivé vrstvy, ani jakým způsobem nastavit regularizace apod. Proto se postupuje tak, že se postupně zkouší různě konstruované sítě a vybere se z nich ta, která nejlépe vyhodnocuje validační data (tj. data, na kterých netrénovala). Tím ale do sítě částečně proniknou i informace z validačních dat, a proto se po vytvoření vyhovující sítě sloučí validační data dohromady s trénovacími a tato síť je trénována na obou z nich. Po trénování je potom vyhodnocena pomocí testovacích dat.

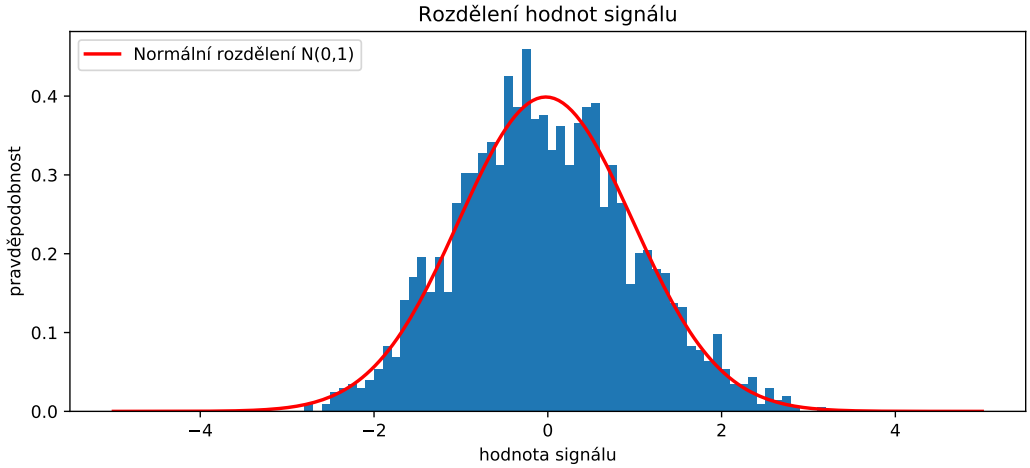
3.4.1 Vytvoření datasetu

Pro tvorbu datasetu byla vytvořena třída *KernelFunction* (4.2.1) tak, aby poskytovala nástroj na čtení požadovaného úseku naměřených dat a jeho správnou interpretaci. Z této třídy dědí všechny třídy transformací, které budou zavedeny později. Zde je využita dědičnost jakožto základní vlastnost objektově orientovaného programování.

Každý objekt z třídy transformací potom obsahuje informaci o umístění naměřených dat, o pozici čteného úseku, o velikosti transformovaného úseku a o případných dalších parametrech použitých transformací.

Vrstevnaté sítě

Tvorbu datasetu pro vrstevnaté sítě zajišťuje třída *Standardizer_1D* (4.2.2), která využívá dědičnosti ze třídy *KernelFunction*. Od signálu je pouze odečten průměr a výsledek se vydělí směrodatnou odchylkou, tzn. je provedena standardizace signálu (viz obr. 3.5).



Obrázek 3.5: Rozdělení hodnot náhodně zvoleného úseku o délce 2048 vzorků (512 vzorků krát 4 kanály) po standardizaci a jeho porovnání s normálním rozdělením $\mathcal{N}(0,1)$.

Pro jednotlivé letové režimy jsou zde brány úseky o délce 512 (případně 4096) vzorků, ze kterých je vytvořen dataset vhodný k trénování. Protože jsou k dispozici 4 ultrazvukové kanály, každý trénovací prvek je složen z $4 \cdot 512 = 2048$, případně z $4 \cdot 4096 = 16384$ hodnot.

HFD spektrogram

Pro použití konvolučních neuronových sítí je potřeba nejprve převést signál do tenzorové podoby. Za tímto účelem je vytvořena třída *CHFDSpektrogram* (4.2.3), která opět využívá dědičnosti ze třídy *KernelFunction*. Vzhledem ke složitosti výpočtu však není tato třída napsána čistě v Pythonu, který by byl příliš pomalý, ale využívá se zde rozšíření Pythonu s názvem *Cython*, který má velice podobný zápis, jako Python, ale zároveň se daný kód kompiluje podobně jako C/C++ a je i patřičně rychlejší. Další výhodou je fakt, že Cython

je s Pythonem v mnoha věcech kompatibilní, a proto lze využít dědičnosti ze třídy psané v Pythonu. Narozdíl od C/C++ má však jednodušší syntaxi a píše se proto rychleji.

Pro psaní kódu v Cythonu je velice užitečné použít prostředí Jupyter notebook, který celou implementaci značně zjednodušuje, neboť se cythonovský kód může psát přímo do buňky pro zdrojový kód. Před použitím cythonovského kódu je nutné pouze aplikovat kód:

```
%load_ext cython
```

Další postup je obdobný třídě *Standardizer_1D*. Díky Cythonu tato třída funguje přibližně 5krát rychleji, než kdyby běžela v samotném Pythonu.

Zprostředkování transformací

Nyní už zbývá tyto transformace paralelizovat, aby bylo možné využít celého potenciálu procesoru. Za tímto účelem je vytvořena funkce *gen_interface* (4.3.3), která zprostředkovává zvolenou transformaci. Tato funkce je navržena tak, aby bylo možno pro každý letový režim rozdělit transformované úseky rovnoměrně mezi jednotlivá jádra. V praxi je tedy funkce *gen_function* spuštěna na každém jádře zvlášť, přičemž jeden z jejích parametrů udává, kterou část signálu bude zpracovávat.

Funkce si vytvoří 4 objekty transformací (pro každý kanál zvlášť), nastaví jim správně pozici, ze které se budou brát data a následně zahájí proces transformace. Poté, co je úsek signálu transformován, je předán funkci, která jej uloží na disk. Zde se však liší způsob ukládání 1D a 3D dat.

Pro ukládání 1D dat (určeno pro vrstevnaté sítě) je zavedena funkce *generate_1D* (4.3.1) a pro 3D data (pro konvoluční sítě) funkce *generate_3D* (4.3.2). V prvním případě dochází ke změně rozměrů z $[4, x]$ na $[4x]$ a uložení vzniklého vektoru ve formátu npy, který využívá pythonová knihovna *numpy*. Ve druhém případě dojde ke změně rozměrů dat z $[4, x, y]$ na $[x, y, 4]$ a jeho uložení jako CMYK ve formátu jpg z důvodu komprese.

Aby transformace probíhala rychleji, byla do ní zavedena paralelizace. V našem případě je tedy vytvořeno 9000 souborů pro každý letový režim tak, že jádro #0 transformuje prvních 1125 úseků, jádro #1 druhých 1125 úseků, atd. Díky paralelizaci je možno transformaci několikanásobně urychlit.

Celkově je tímto způsobem vytvořeno 11 složek s daty. Pro trénování je dobré, když jsou data namíchána v náhodném pořadí. K tomu slouží funkce *prepare_dataset* (4.3.4), která v linuxovém prostředí přesune náhodně vybraných 20% souborů do složky *test* a zbytek do složky *data*. Ve složce *data* jsou tedy dohromady trénovací i validační data, a to z důvodu implementace trénovací funkce knihovny *keras*, která umožňuje načíst tento dataset dohromady a zvolit poměr dat, která budou využita pro trénování a která pro validaci sítě.

Schéma generování datasetu pro HFD spektrogram

Iterace přes 11 letových režimů:

- Rozdělení úseků daného režimu mezi všechna dostupná jádra procesoru:
  - Spuštění funkce *gen_interface*.

- Ta inicializuje 4 transformační funkce *CHFDSpectrogram* (pro každý kanál).
  - Ty dědí ze třídy *KernelFunction*, pomocí které načtou naměřená data.
  - Tato data poté transformují a vrátí zpět do *gen_interface*.
- Ta poté transformovaná data předá funkci *generate_3D*:
  - Tato funkce změní rozměry transformovaných dat z $[4, x, y]$ do $[x, y, 4]$ a uloží je jako CMYK do jpg souboru.

Vygenerované datasety

Přehled všech použitých datasetů udává tabulka 3.1. U všech datasetů vygenerovaných pomocí HFD spektrogramu bylo pro diskrétní transformaci použito časové okno o šířce 128 vzorků.

Datasety vytvořené třídou *Standardizer_1D*:

Název	Rozměry	Transformace
R512	[2048]	Standardizace
R4096	[16384]	Standardizace

Datasety vytvořené třídou *CHFDSpectrogram*:

Název	Rozměry	Šířka časového okna	Tvar č. okna	Posun č. oken (k)
H255	[128,128,4]	128 vzorků	obdélníkové	1 vzorek
H382	[128,128,4]	128 vzorků	obdélníkové	2 vzorky
H636	[128,128,4]	128 vzorků	obdélníkové	4 vzorky
H1144	[128,128,4]	128 vzorků	obdélníkové	8 vzorků
H2160	[128,128,4]	128 vzorků	obdélníkové	16 vzorků
H5192	[128,128,4]	128 vzorků	obdélníkové	32 vzorků
H8256	[128,128,4]	128 vzorků	obdélníkové	64 vzorků
H16384	[128,128,4]	128 vzorků	obdélníkové	128 vzorků

Tabulka 3.1: Přehled vygenerovaných datasetů. Číslo v názvu udává celkovou délku transformovaného úseku. Posun časových oken je dán parametrem posunu k, viz rovnice 2.10.

3.5 Schéma implementace

V následující části budou porovnávány sítě tak, aby byla vidět jejich přesnost a hodnota ztrátové funkce v závislosti na epochách. Epocha zde představuje jednorázovou adaptaci vah přes všechna trénovací data.

Nejprve budou porovnány různě veliké sítě a následně i vliv regularizací. Poté budou prozkoumány možnosti využití delšího úseku naměřeného ultrazvukového signálu a jeho vliv na úspěšnost vrstevnaté sítě. Ve druhé části pak budou porovnány konvoluční sítě a jejich regularizace. Potom bude zhodnocena úspěšnost výstupu HFD spektrogramu vzhledem k různě dlouhým posunům časových oken, viz tabulka 3.1.

Vizualizace sítě

Všechny sítě použité v experimentech budou znázorněny graficky, viz tabulka 3.2. Díky tomu bude možno jednoduše znázornit všechny důležité prvky sítě.

	Konvoluční vrstva s jádrem o rozměrech [5,5], L2 regularizací s parametrem p a počtem neuronů N.
	Poolingová vrstva s jádrem o rozměrech [2,2].
	Konvoluční vrstva s jádrem o rozměrech [3,3], L2 regularizací s parametrem p a počtem neuronů N.
	Vrstva zploštění.
	Vrstva vrstevnaté sítě s počtem neuronů N, na kterou je aplikována L2 regularizace s parametrem p.
	Výstupní vrstva s počtem neuronů N.
$\mathcal{N}$	Použití svazkové normalizace.
$\frac{0.1}{\mathcal{N}}$	Použití dropoutu a svazkové normalizace. Pokud jsou použity obě současně, je upřednostněna ta metoda, která je uvedena nad čárou.

Tabulka 3.2: Popis vizualizace konvoluční sítě.

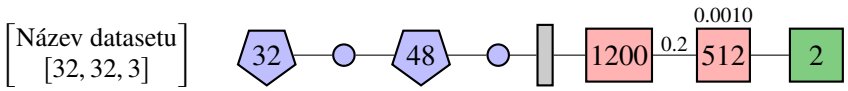
Generování sítě

Pro jednoduché (nevětvené) struktury sítí (v této práci všechny) je možné použít model *Sequential* z knihovny keras. V tomto modelu se přidávají jednotlivé vrstvy za sebe, přičemž jejich napojování i funkčnost obstarává samotná knihovna.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout, AlphaDropout
from keras import regularizers
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
classifier = Sequential()
classifier.add(Convolution2D(32, (5, 5), input_shape = (32,32,3),
    activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Convolution2D(48, (5, 5), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(activation = 'relu',units=1200))
classifier.add(Dropout(0.2))
classifier.add(Dense(activation = 'relu',units=512, kernel_regularizer=
    regularizers.l2(0.001)))
classifier.add(Dense(2, activation='softmax'))
```

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

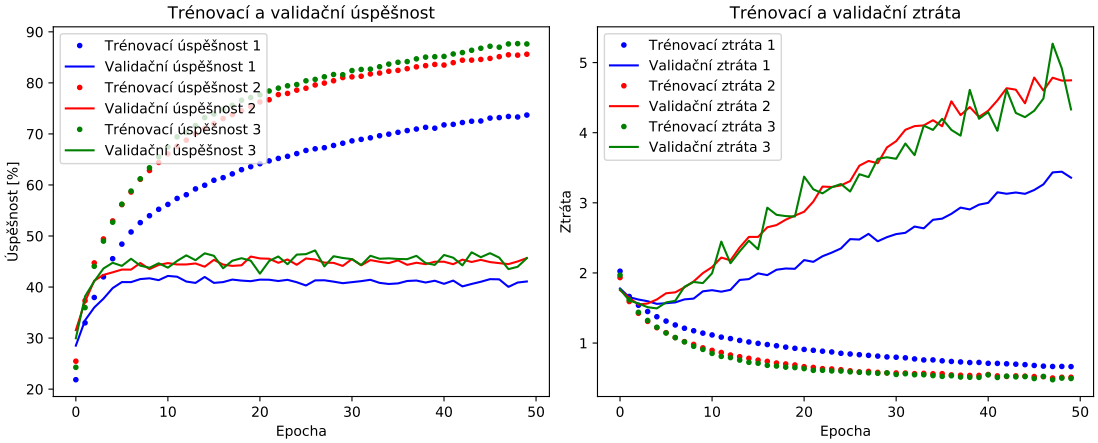
Takto je sestavena konvoluční neuronová síť podle obrázku 2.5. Grafické znázornění sítě pak poskytuje obrázek 3.6. Aktivační funkce všech skrytých vrstev u všech sítí v této práci bude výhradně *ReLU*.



Obrázek 3.6: Vizualizace konvoluční neuronové sítě (viz 2.5).

3.5.1 Terminologie hodnocení výsledků

Poté, co jsou sítě zkonstruovány, přichází na řadu jejich trénování. V procesu trénování jsou zaznamenávány hodnoty ztrátové funkce (v těchto experimentech je to *softmax*) na trénovacích a validačních datech. Zároveň s tím je měřena úspěšnost dané sítě na trénovacích a validačních datech jako poměr dat, která síť rozpoznala správně. Záznam těchto hodnot probíhá po každé epoše, tj. po každé iteraci přes všechna trénovací data. Pro přehlednost jsou poté tyto záznamy znázorněny graficky pro všechny sítě, které daný experiment porovnává, viz obr. 3.7.



Obrázek 3.7: Příklad hodnocení experimentu pro 3 vrstevnaté sítě.

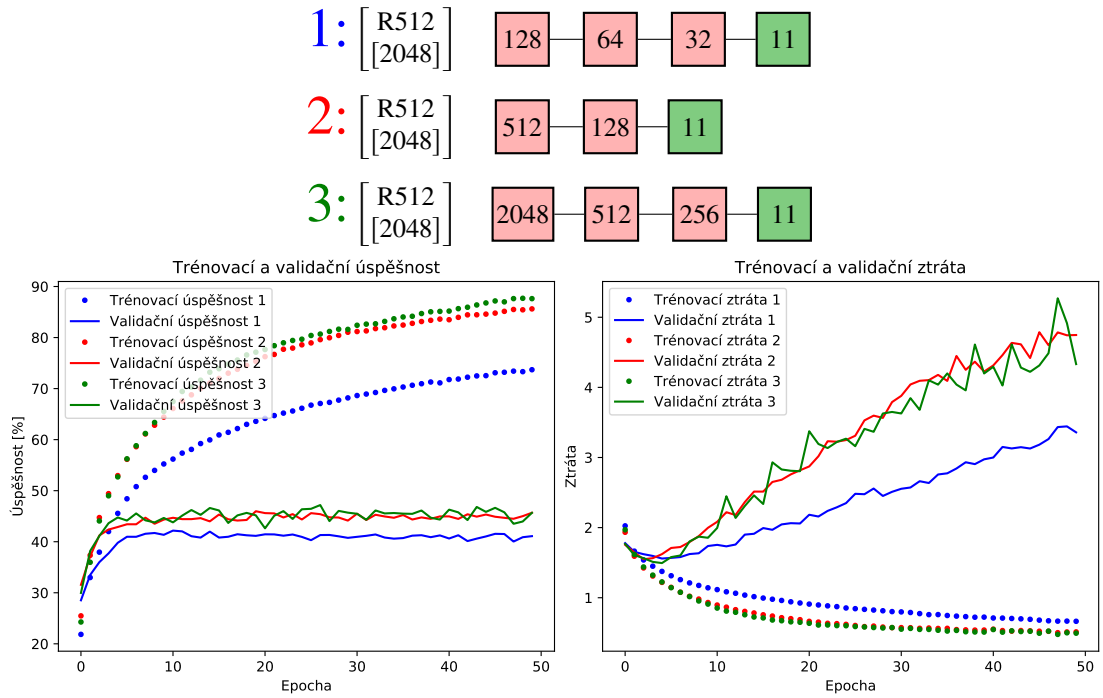
3.6 Aplikace vrstevnatých sítí

3.6.1 Vliv velikosti sítí

V této části jsou na naměřená data aplikovány vrstevnaté sítě bez konvoluční části architektury. Zároveň je zde demonstrováno několik možností použití regularizace, svazkové normalizace a dropoutu.

Nejprve je potřeba si rozmyslet, jaká struktura bude na daný problém vhodná. Pokud by byla síť příliš malá (málo vrstev s málo neurony v každé vrstvě), neměla by dostatečnou kapacitu na to, aby se dokázala naučit rozpoznávat trénovací data a nová (validační) data by pak nevyhodnocovala správně. Pokud by naopak byla příliš velká, mohlo by docházet k přeučení, tedy ke ztrátě schopnosti generalizace. V takovém případě se pak síť učí trénovací data tak, jako by to byl slovník pojmů. Jednou z možností, jak najít vhodnou architekturu, je použít genetický algoritmus. Tento přístup je obecně velice náročný na hardware, jelikož je potřeba trénovat velké množství sítí.

V tomto experimentu (obr. 3.8) je porovnán pouze malý počet sítí bez regularizace, dropoutu, i svazkové normalizace.



Obrázek 3.8: Experiment 1 (velikost sítí): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

Z výsledků je možné usoudit, že první síť neměla dostatek parametrů, a proto ve srovnání s ostatními dvěma nedosahuje tak vysoké přesnosti. Druhá a třetí síť hodnotí validační data podobně, rozdíl v nich je jen nepatrný. Druhá síť však obsahuje méně parametrů, než třetí, a proto je v průběhu trénování stabilnější.

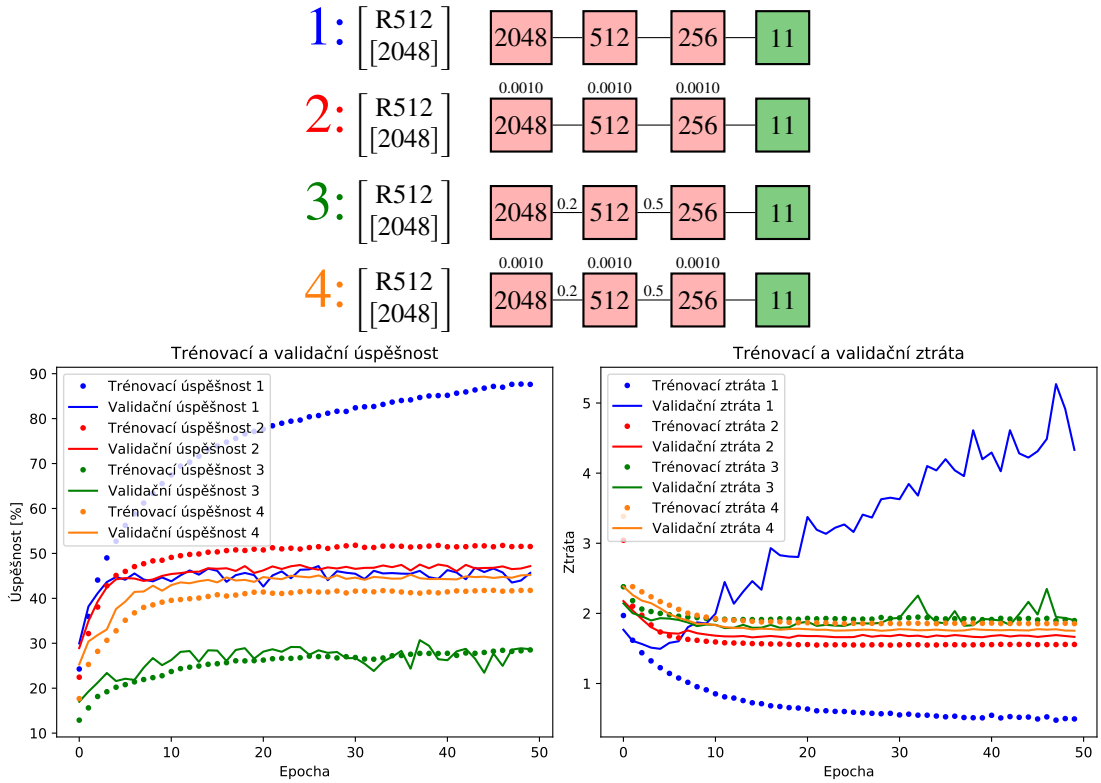
3.6.2 Vliv regularizace a dropoutu

Nyní bude do sítí zavedena regularizace a dropout. U dropoutu se doporučuje, aby v první vrstvě nebyl příliš silný a v poslední vrstvě nebyl přítomen vůbec.

Experiment ukazuje (obr. 3.9), že dropout na tato data pravděpodobně nebude příliš vhodný. Použití samotného dropoutu i jeho kombinace s L2 regularizací zhoršilo přesnost vyhodnocování validačních dat. Naproti tomu se zde ukazuje přínos L2 regularizace. Trénovací přesnost již po dvacáté epoše příliš nemění svoji hodnotu a nedochází ani k přeučení.

Validační přesnost je proto stabilnější a dosahuje vyšší hodnoty, než v případě sítě bez regularizace.

V následujících experimentech budou porovnány další kombinace dropoutu s regularizací a se svazkovou normalizací.

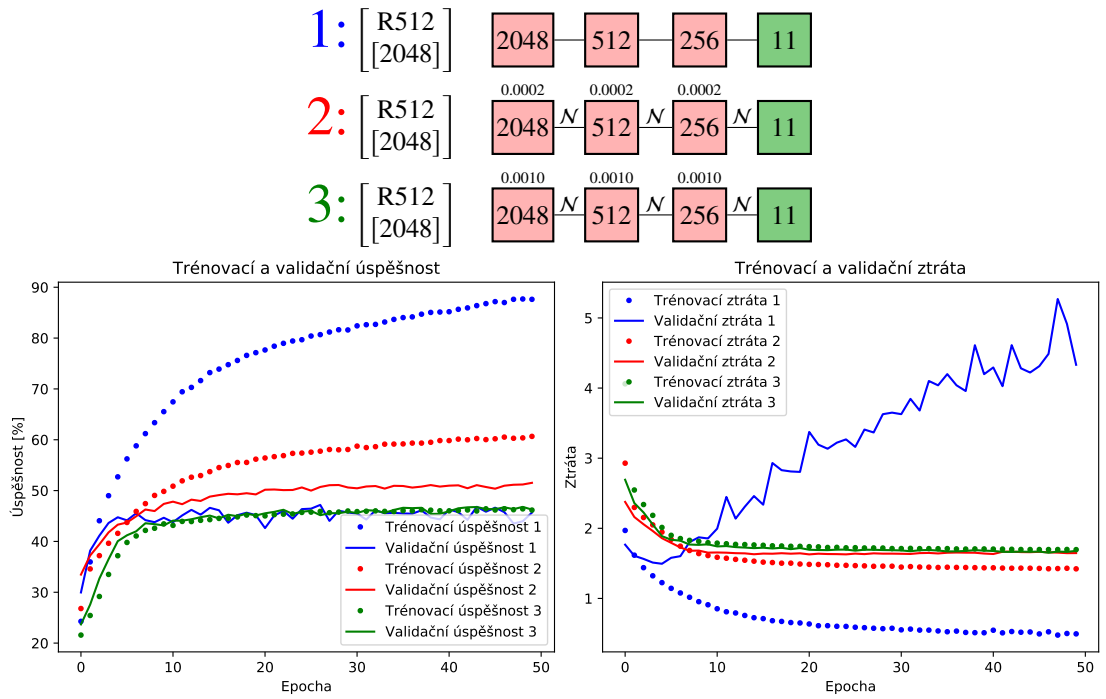


Obrázek 3.9: Experiment 2 (regularizace a dropout): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

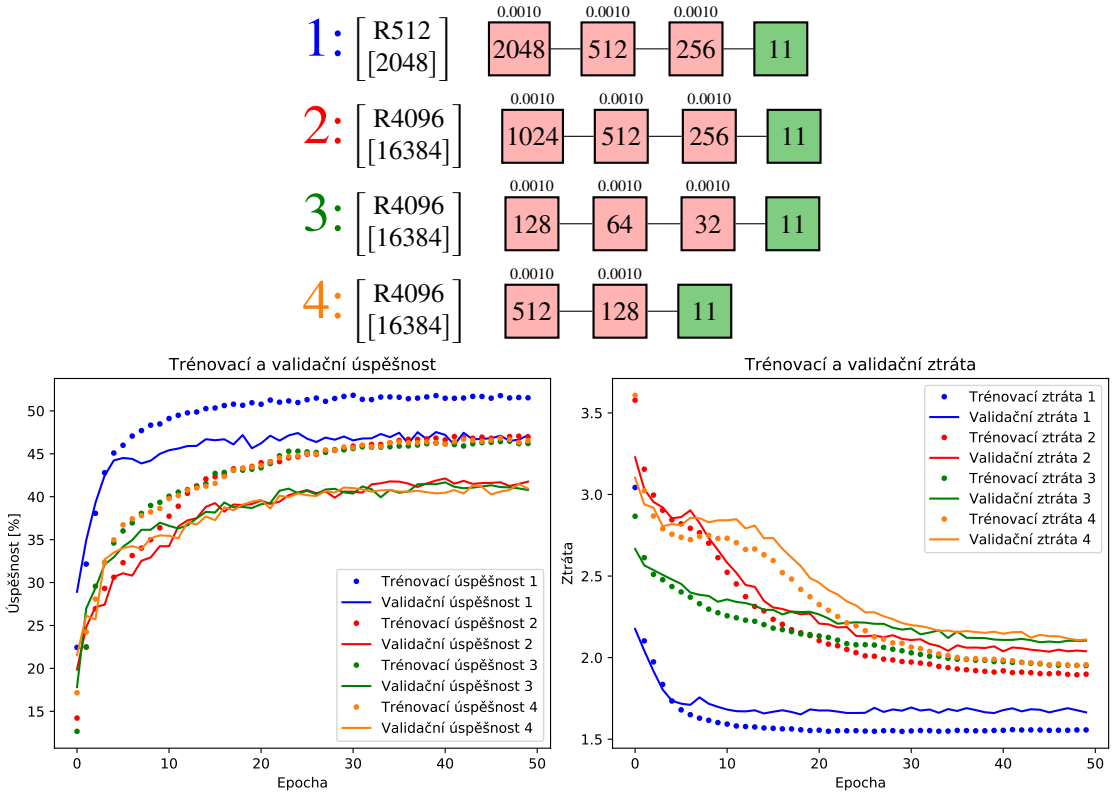
3.6.3 Vliv svazkové normalizace

Při tvorbě dat již byly jednotlivé úseky signálu normalizovány (viz 3.4.1), a proto není potřeba použít svazkovou normalizaci jako první vrstvu. Pro další vrstvy to ale neplatí, takže zde bude svazková normalizace zahrnuta do všech skrytých vrstev. Vzhledem k tomu, že u svazkové normalizace je doporučeno [20] použít pětkrát nižší hodnotu L2 regularizace, bude porovnána i takto vytvořená síť.

Výsledek experimentu (obr. 3.10) je v souladu s doporučením autorů svazkové normalizace [20]. Síť, ve které byla použita svazková normalizace a pětkrát nižší hodnota L2 regularizace, měla přibližně o 5% lepší výsledky u validačních dat. Síť s pouhým přidáním svazkové regularizace měla sice podobný výsledek, jako bez jejího použití, avšak už nedocházelo k přeučení, neboť trénovací a validační přesnost dosahovala přibližně stejné hodnoty.



Obrázek 3.10: Experiment 3 (svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.



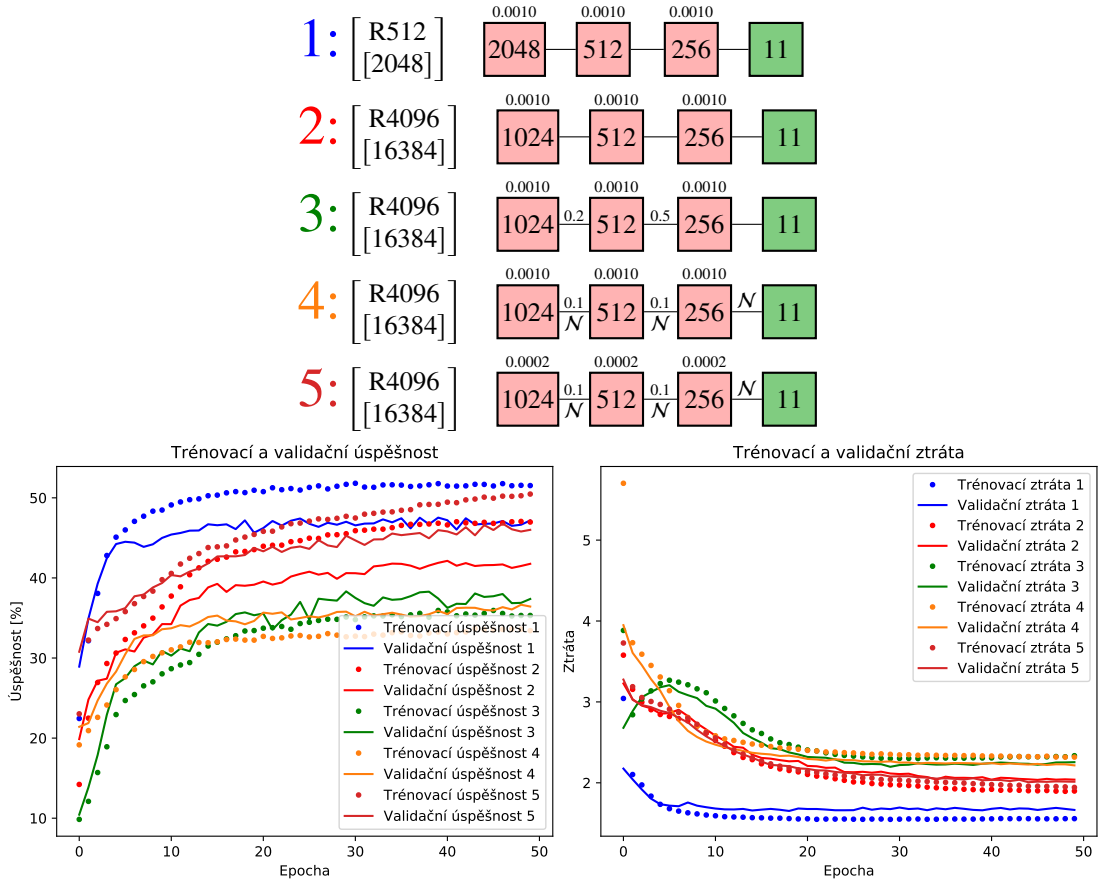
Obrázek 3.11: Experiment 4 (délka časového úseku): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

3.6.4 Závislost na délce časového úseku

V této části experimentů bude brán do vstupní vrstvy čtyřnásobně delší časový úsek. Počet parametrů se tímto krokem zvýší na velice vysokou hodnotu (řádově i desítky milionů), což výrazně zvyšuje hardwarové požadavky na výkon a paměť. Bude proto zkoumán přínos tohoto kroku.

Úseky jsou opět normalizovány již v rámci jejich vytvoření, a proto není potřeba brát jako první vrstvu sítí svazkovou normalizaci. Postup bude stejný, jako v předešlé části experimentů. Nejprve budou prozkoumány různé velikosti sítě, a jelikož se v předchozích experimentech osvědčily L2 regularizace, budou zde využívány po celou dobu. U nejúspěšnější sítě bude dále prozkoumán přínos použití různě nakombinovaných normalizací a dropoutů.

Výsledky experimentu (obr. 3.11) jsou pro všechny zkoumané sítě velice podobné. Ve srovnání s předešlým experimentem však došlo k výraznému zhoršení (asi o 5%), a to i přes to, že mají nyní sítě výrazně více parametrů (okolo 17 milionů). To ve výsledku znamená, že je zde větší požadavek na výkon a na paměť hardwaru, a přesto dochází ke zhoršení trénovací i validační přesnosti. Ze tří porovnávaných sítí má síť číslo 2 nejlepší přesnost i nejnižší hodnotu ztrátové funkce, a proto bude vybrána pro další experimenty.



Obrázek 3.12: Experiment 5 (dropout): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

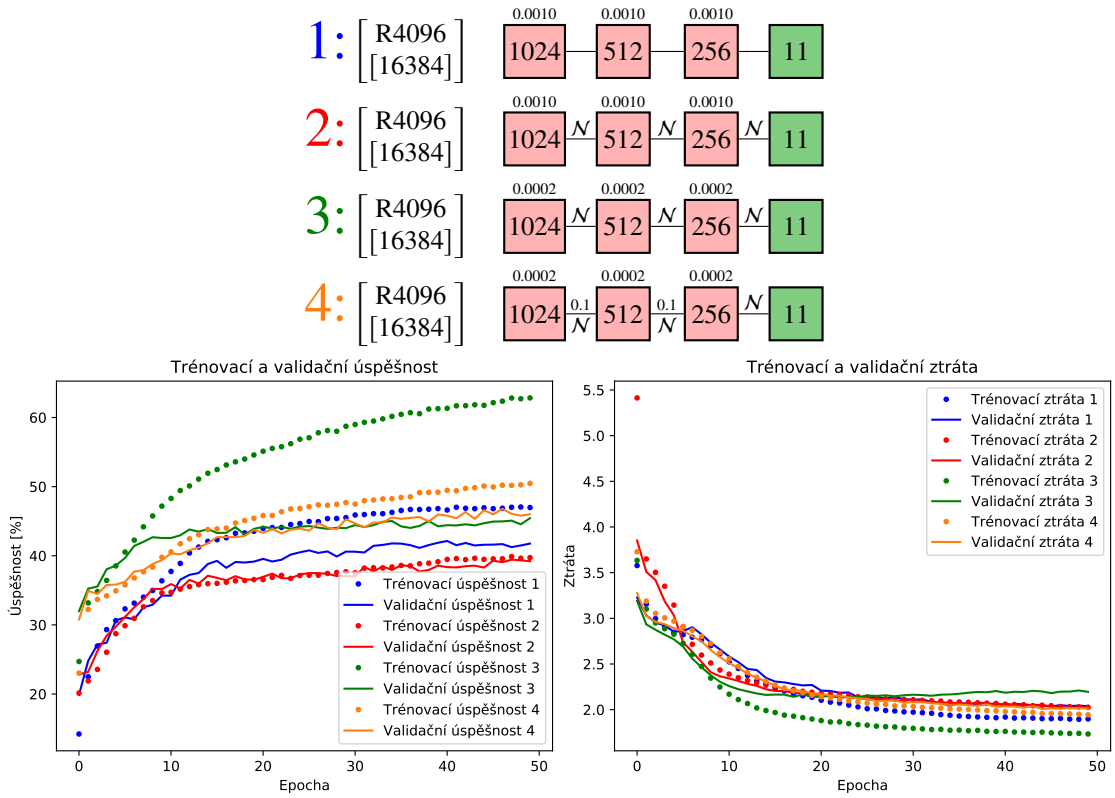
3.6.5 Vliv dropoutu

V následujícím experimentu bude otestován vliv dropoutu. V minulých experimentech se dropout osvědčil pouze v kombinaci se svazkovou normalizací. Nyní bude otestován jeho přínos na tomto datasetu (s delšími úseky signálu).

Experiment ukazuje (obr. 3.12), že samotné použití dropoutu opět snižuje úspěšnost sítě, a to i s použitím svazkové normalizace, pokud není zároveň pozměněna hodnota L2 regularizace. Pokud se však pětinásobně sníží hodnota L2 regularizace, dosahuje síť mnohem vyšší přesnosti, než síť, která používá pouze L2 regularizaci. I přesto má ale nižší přesnost, než síť z minulé části experimentů.

3.6.6 Vliv svazkové normalizace

Nyní je potřeba ukázat, jestli by síť se svazkovou normalizací nebyla bez dropoutu úspěšnější, tedy jestli dosahuje lepších výsledků díky dropoutu, nebo navzdory dropoutu.



Obrázek 3.13: Experiment 6 (svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

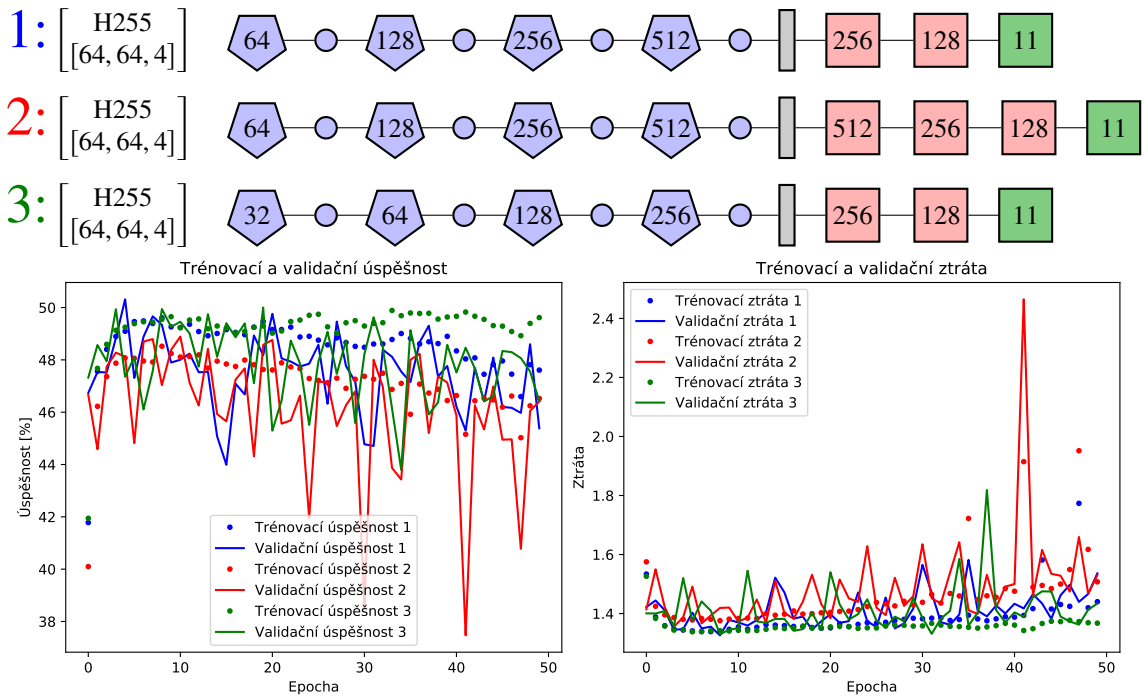
Výsledky experimentu ukazují (obr. 3.13), že je tomu tak díky dropoutu, neboť bez něho síť dosahuje nepatrně nižší úspěšnosti. V tomto případě je dokonce použití svazkové normalizace vhodné pouze tehdy, když je zároveň upravena hodnota L2 regularizace.

3.7 Použití konvolučních vrstev

Nejlepší vrstevnaté sítě dosáhly úspěšnosti okolo 50%. Vzhledem k rozpoznávání 11 letových režimů je to výrazně více, než kolik by dosáhlo náhodné tipování. Ukazuje se však, že dosáhnout vyšší úspěšnosti může být problematické. Zvýšení délky úseku, kterou se síť učí rozpoznávat, nevede s použitím podobně konstruovaných sítí ke zvýšení úspěšnosti, ale k jejímu snižování. Tento problém by se dal pravděpodobně kompenzovat sestrojením větší sítě, ale s přehnaně vysokými nároky na hardware.

Dalším problémem zůstává nespojitost rozpoznávání úseků v tom smyslu, že posun signálu o pouhý jeden naměřený vzorek zcela mění hodnoty neuronů v celé síti. Všechny tyto problémy by se mohly vyřešit zavedením časo-frekvenčních transformací signálu a následným využitím konvolučních sítí, které tyto problémy nemají. Výběr vhodné transformace se ukazuje být klíčový, neboť správná interpretace dat síti výrazně urychlí proces učení a umožní daný problém vyřešit za použití menšího množství trénovacích dat i s menším počtem parametrů. V této části experimentů bude za tímto účelem použit HFD spektrogram popsaný v sekci 2.5.

Teoreticky by mohla být použita jakákoliv interpretace dat, pokud by byl k dispozici dostatečně velký dataset. Na velké množství komplikovaných dat by se musela aplikovat síť o velkém počtu parametrů. To však není prakticky proveditelné, protože v praxi je počet trénovacích dat omezený, stejně jako výpočetní možnosti používaného hardwaru.



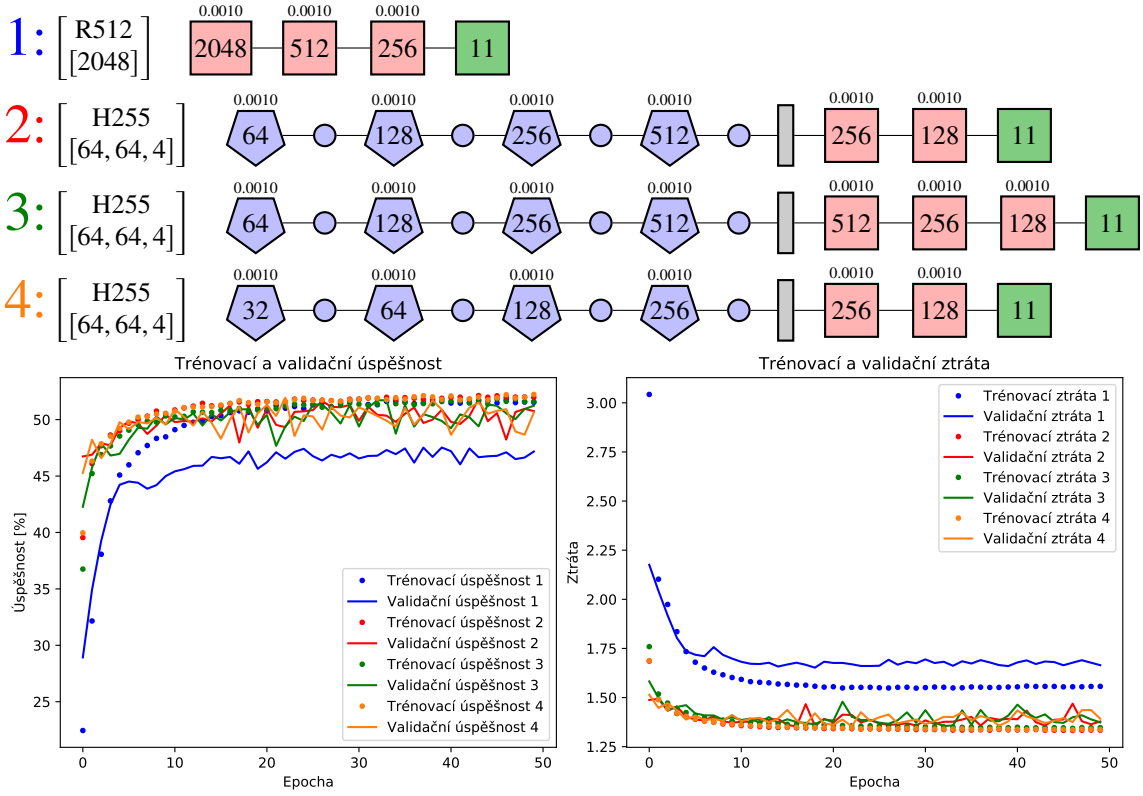
Obrázek 3.14: Experiment 7 (velikost sítí a L2 regularizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

3.7.1 Vliv velikosti sítí a L2 regularizace

Stejně jako u vrstevnatých sítí zde neexistuje žádné pravidlo, které by určovalo, jak vytvořit nejúspěšnější síť. Nejprve budou vyzkoušeny sítě, kde se na začátku střídají konvoluční a poolingové vrstvy, poté dochází ke zploštění a na výsledný vektor jsou aplikovány vrstevnaté sítě.

Počet vrstev konvoluční části je proto vybrán tak, aby první dva rozměry výstupu celé části byly dostatečně malé na to, aby proces zploštění nevytvořil příliš velký vektor. V takovém případě by totiž značně narostl počet parametrů sítě, což by kladlo velké nároky na hardware a navíc by se zvýšilo riziko přeučení. Vzhledem k velikosti vstupního tenzoru o rozměrech $[64, 64, 4]$ tedy budou použity 4 konvoluční vrstvy o rozměru jádra $[3, 3, z]$ a 4 poolingové vrstvy o rozměru $[2, 2]$. Počet vrstev u nekonvoluční části žádné takové podmínky neklade, a proto je zde vyzkoušeno několik možností.

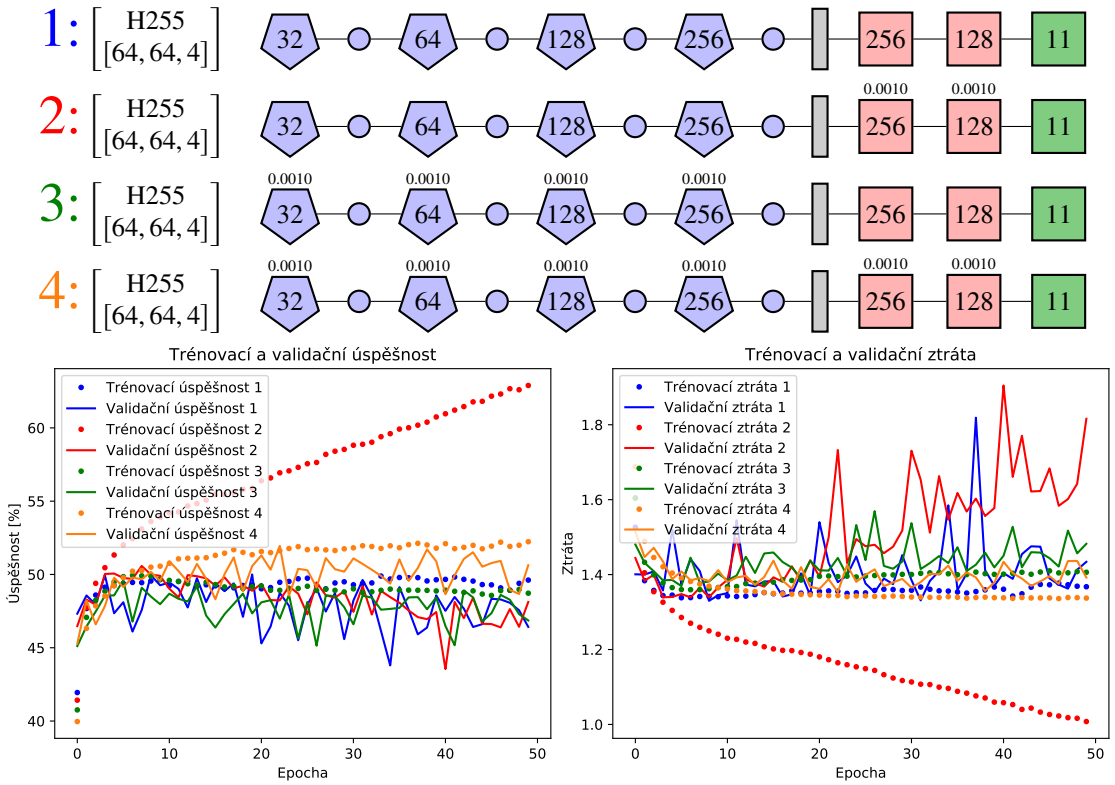
Z výsledku experimentu vyplývá (obr. 3.14), že bez použití L2 regularizací je konvoluční síť velice nestabilní a v každé epoše vyhodnocuje validační data zcela jinak. V praxi pak není možné odhadnout přesnost sítě, ani kolik epoch by bylo ideální pro její trénování. Z tohoto důvodu je v dalším experimentu (obr. 3.15) na stejné sítě aplikována L2 regularizace.



Obrázek 3.15: Experiment 8 (velikost sítí a L2 regularizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

Z grafu je patrné, že se při použití L2 regularizace staly sítě mnohem stabilnějšími. Zároveň se ukazuje, že počet neuronů v různých vrstvách u těchto tří sítí nemá na úspěšnost zásadní vliv, a proto bude k dalším experimentům sloužit nejmenší z nich kvůli jednoduchosti trénování.

Ve srovnání se samotnými vrstevnatými sítěmi dosahují konvoluční přibližně o 5% lepšího výsledku, a to dokonce na polovičním časovém úseku. Nyní bude prověřen přínos použití L2 regularizace na jednotlivé části sítě.



Obrázek 3.16: Experiment 9 (velikost sítí a L2 regularizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

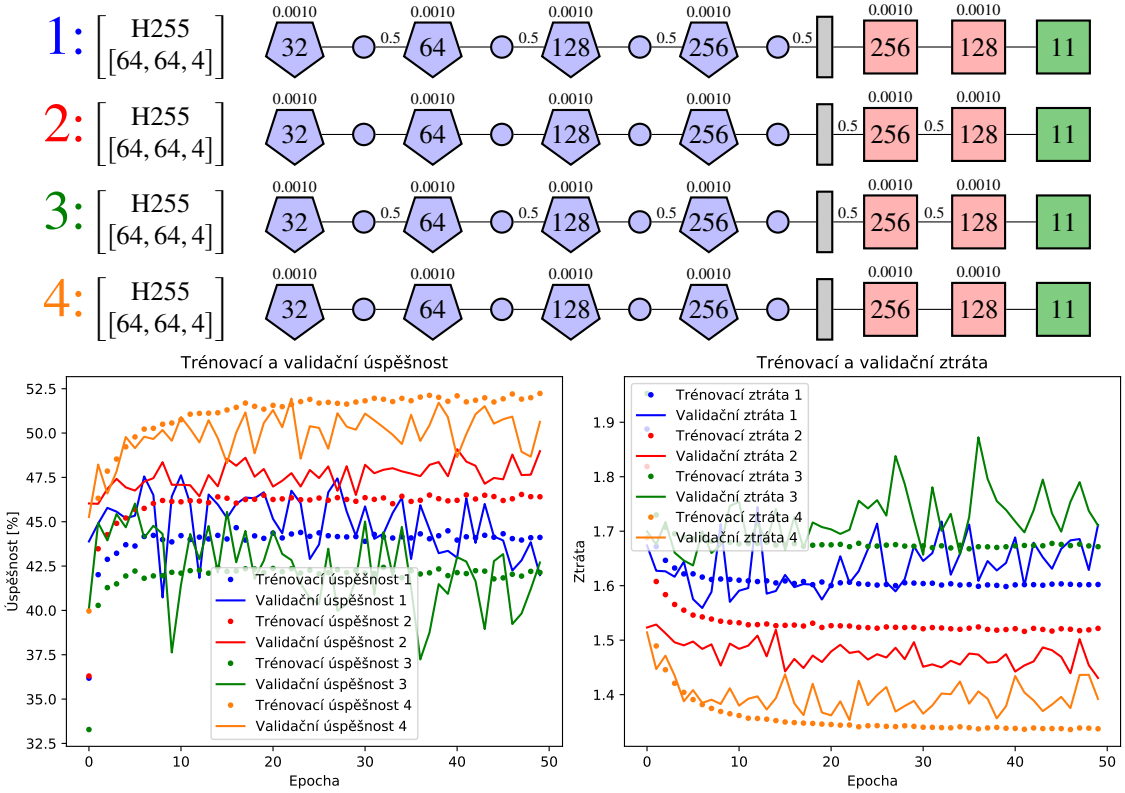
V tomto případě (obr. 3.16) má však výrazně větší úspěšnost síť, kde byla regularizace použita na všech vrstvách současně. Zároveň se ukazuje, že pokud je L2 regularizace použita pouze na nekonvoluční části sítě, začne docházet k přeučení, protože zatímco trénovací přesnost výrazně narůstá, validační přesnost začíná kolem 15-té epochy postupně klesat. V dalších experimentech proto bude L2 regularizace aplikována na všechny vrstvy a bude zkoumáno použití dalších typů regularizací, tj. dropoutu a svazkové normalizace.

3.7.2 Vliv dropoutu a svazkové normalizace

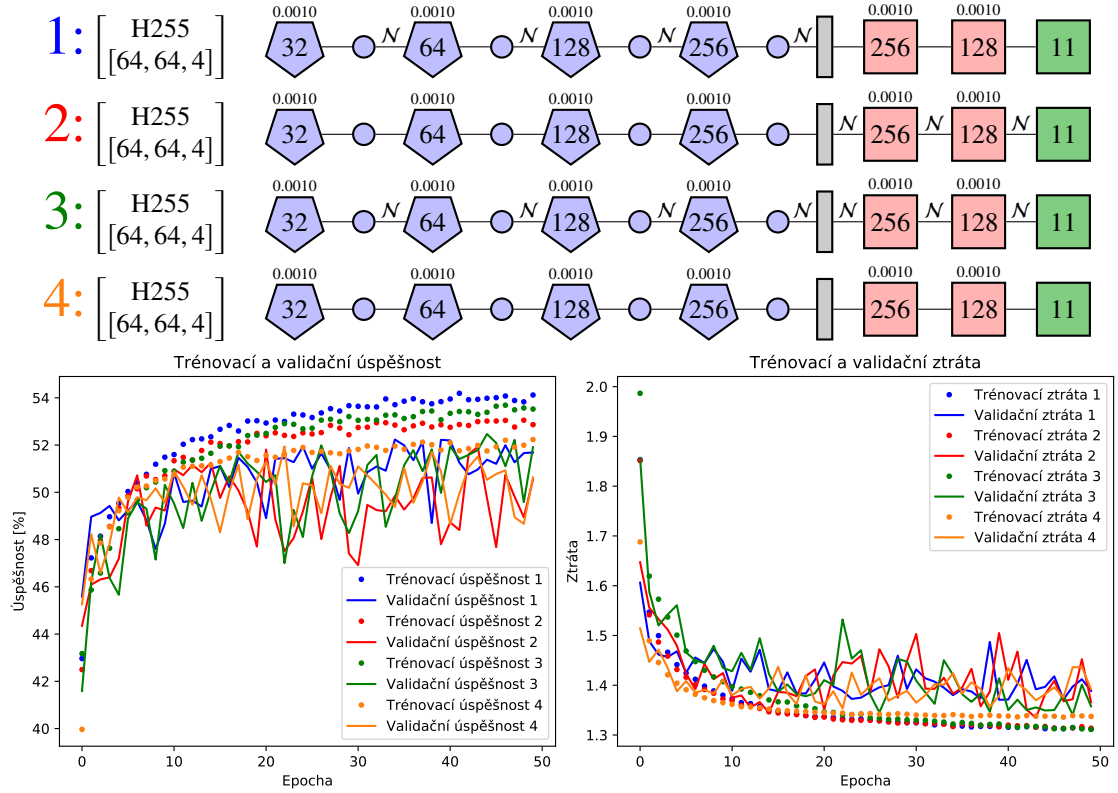
Nejprve bude zkoumán dropout na jednotlivých částech sítě. Vzhledem k úspěšnosti zavedení L2 regularizace na všechny vrstvy budou porovnány pouze takto regularizované sítě.

Výsledek tohoto experimentu (obr. 3.17) ukazuje, že použití dropoutu bez svazkové normalizace není na těchto datech vhodné ani u konvolučních sítí.

Dalším krokem bude otestování samotné svazkové normalizace. Nejprve bude zkoumán účinek na jednotlivých částech sítě bez použití dropoutu, či změny hodnoty L2 regularizace (obr. 3.18).

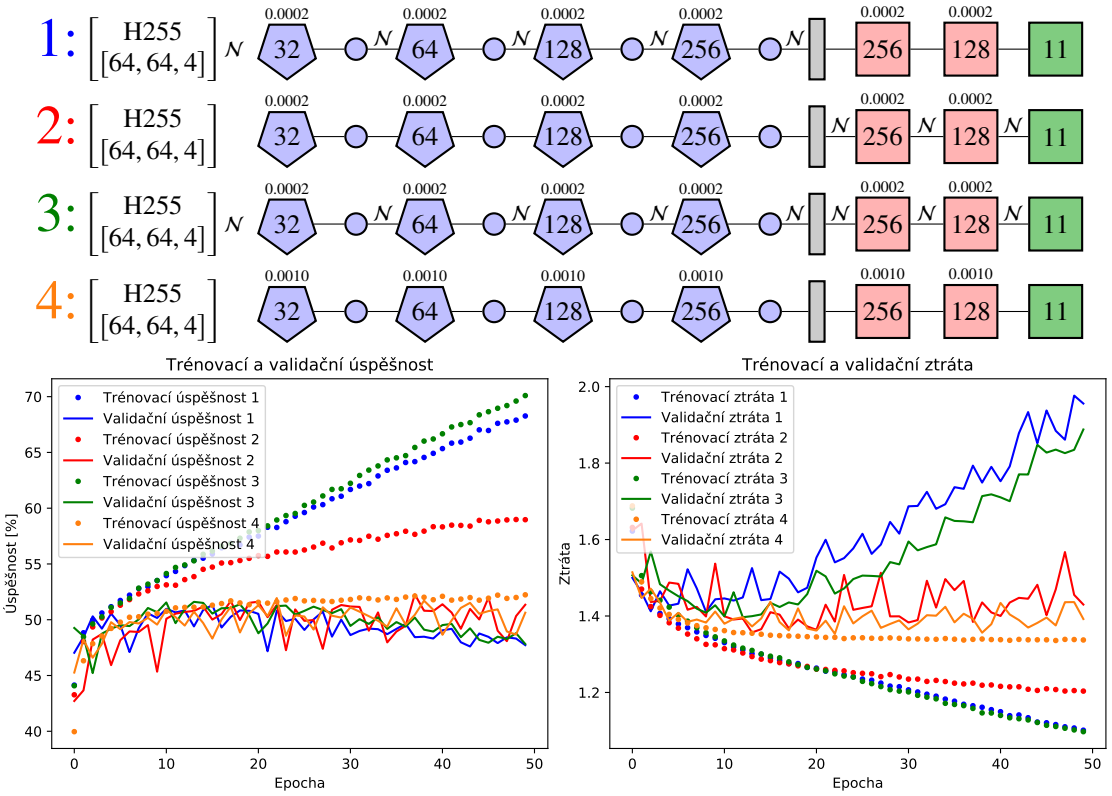


Obrázek 3.17: Experiment 10 (dropout a svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.



Obrázek 3.18: Experiment 11 (dropout a svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

Nejvyšší validační přesnost byla naměřena u sítí, ve kterých byla svazková normalizace použita na konvoluční vrstvy, avšak celkově vzato jsou výsledky všech sítí podobné. Další experiment opět posoudí, jestli se úspěšnost těchto sítí zvýší, pokud je pětinásobně snížena hodnota L2 regularizace.

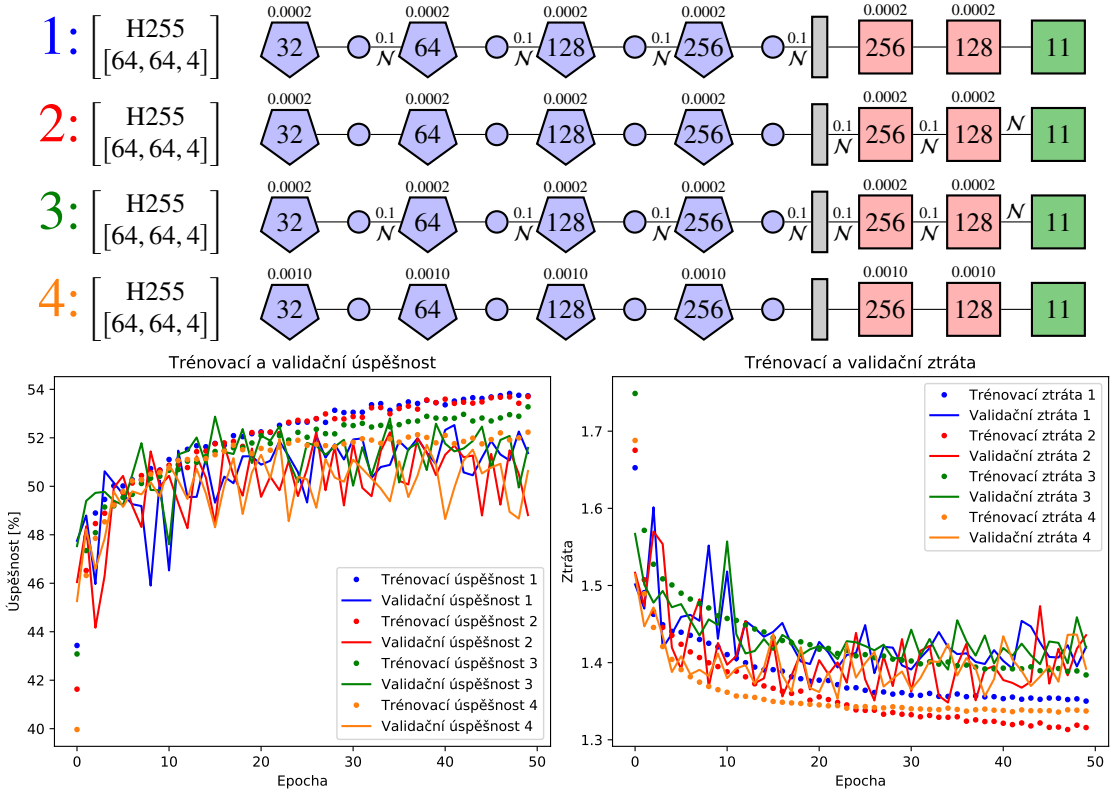


Obrázek 3.19: Experiment 12 (dropout a svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

Experiment (obr. 3.19) je v souladu s tvrzením [20], že normalizovaná síť dospěje do nejúspěšnějšího stavu za kratší dobu (zde kolem 15-té epochy), zatímco síť bez normalizace tohoto stavu dosáhne až kolem 38-té epochy. Tento přístup tedy nemusí nutně zlepšovat kvalitu sítě, ale výrazně snižuje dobu potřebnou k trénování modelu.

Dále se ukazuje, že normalizovaná síť může být úspěšnější, pokud je do ní zakomponován dropout s nízkou hodnotou.

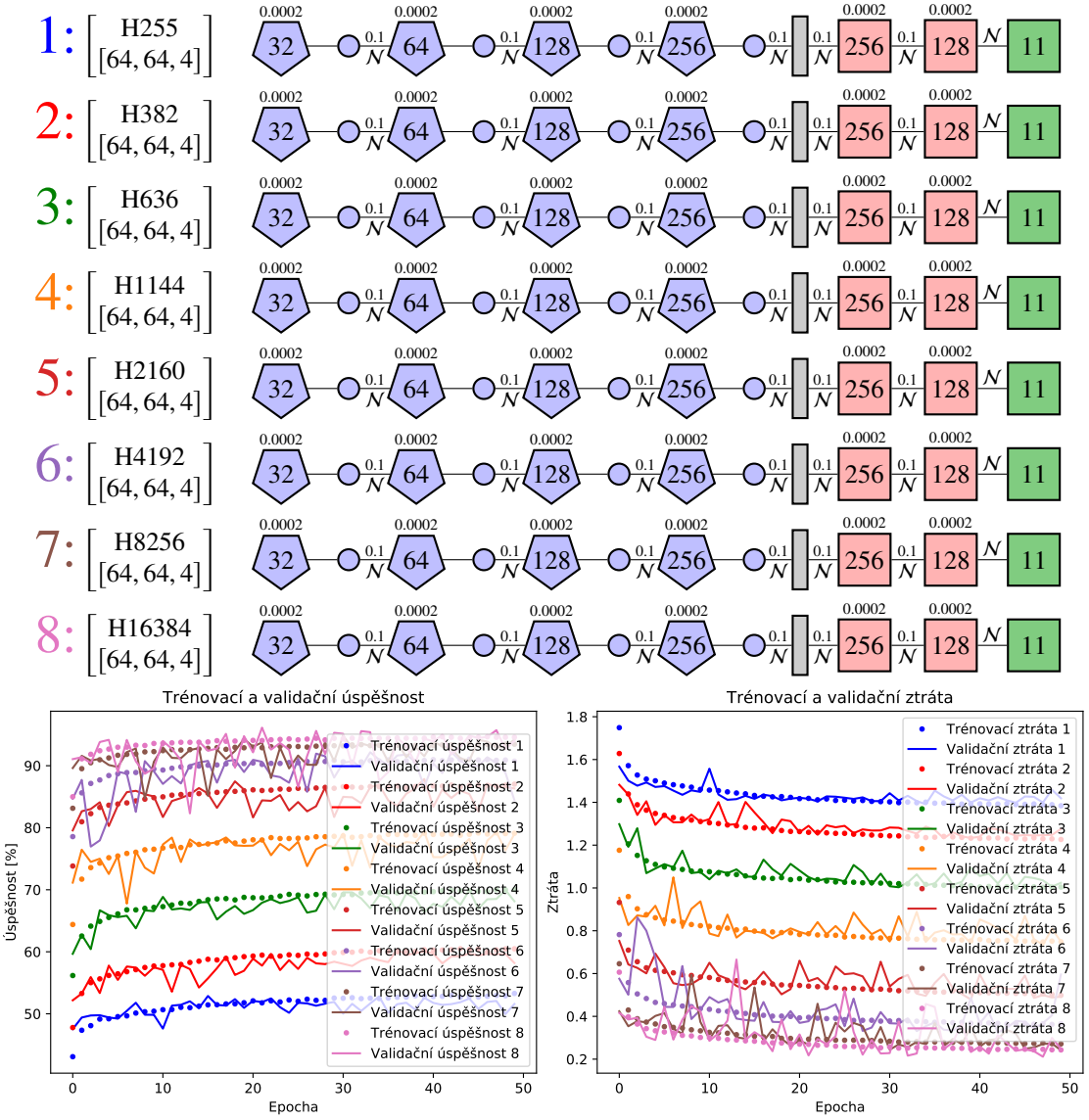
Na tomto experimentu (obr. 3.20) je vidět, že normalizovaná síť dosahuje nejvyšší hodnoty kolem 18-té epochy. Ve srovnání s nenormalizovanou sítí dosahuje vyšší hodnoty téměř v každé epoše.



Obrázek 3.20: Experiment 13 (dropout a svazková normalizace): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

3.8 Vliv posunu časového okna HFD spektrogramu

V následujícím experimentu bude ukázána další výhoda použití HFD spektrogramu. Při delším posunu časového okna u HFD spektrogramu (viz 2.5) totiž dochází k výraznému zlepšení úspěšnosti za použití stejného výpočetního času, pouze za cenu transformování delšího časového úseku (stále však řádově v jednotkách milisekund).

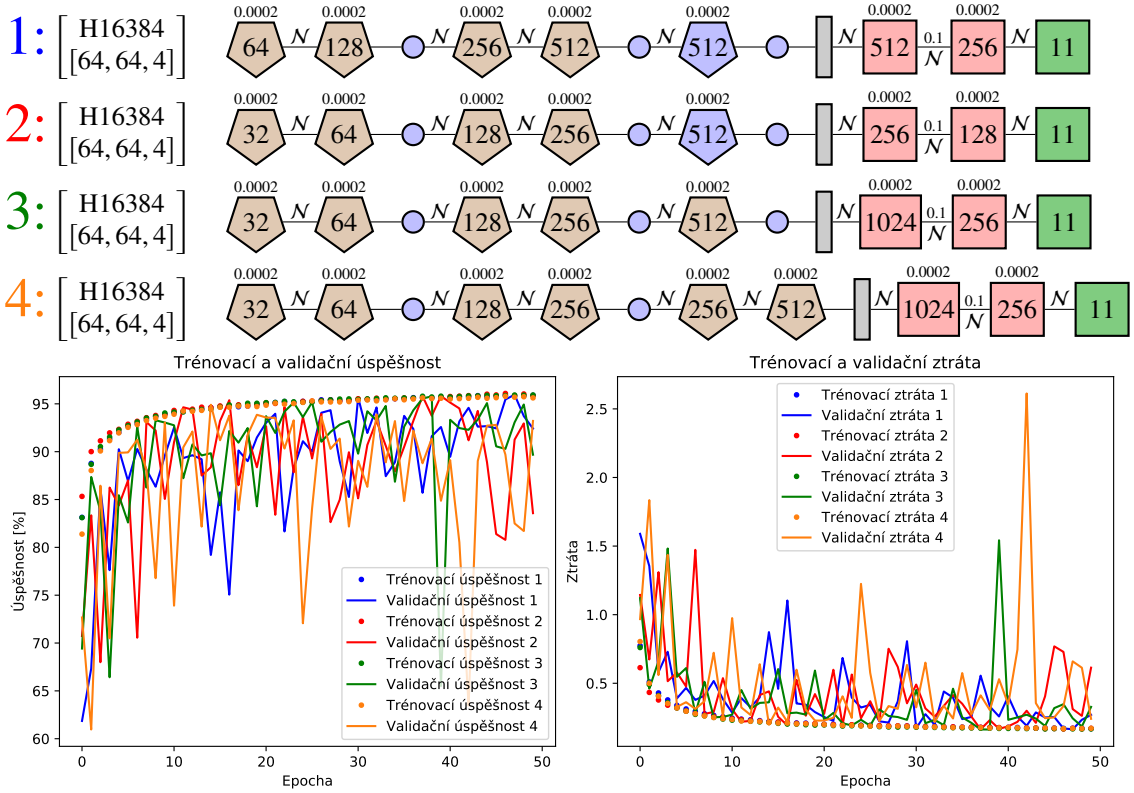


Obrázek 3.21: Experiment 14 (posun časového okna HFD spektrogramu): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

Na tomto experimentu (obr. 3.21) je vidět zlepšení s postupným zvětšováním přeskakovaných úseků. Maximální hodnoty je pak dosaženo, pokud se dílčí transformované úseky vůbec nepřekrývají. Tímto krokem je dosaženo úspěšnosti okolo 94%.

3.9 Další konstrukce sítí

Konvoluční sítě nemusí nutně vypadat tak, že se střídají konvoluční a poolingové vrstvy. V této části je ukázka použití sítí, kde takovéto střídání není.



Obrázek 3.22: Experiment 15 (další konstrukce sítí): struktura sítí, jejich trénovací a validační úspěšnost a ztráta.

U těchto sítí (obr. 3.22) však může být problémem nedostatečná stabilita. V každé epoše vyhodnocují validační data zcela jinak, což znemožňuje určení optimálního času trénování i odhad skutečné úspěšnosti sítě (tedy jestli je síť použitelná do praxe).

Kapitola 4

Implementace

4.1 Inicializace

4.1.1 Import knihoven

```
import numpy as np # knihovna pro matematiku
import os # knihovna pro práci se systémem
from scipy import signal # zpracování signálu
from multiprocessing import Pool # použití více jader procesoru
```

4.1.2 Definice konstant

```
CHANNELS=4 # počet kanálů (osciloskop)
SOURCES=['Data/1.bdat','Data/2.bdat','Data/3.bdat','Data/4.bdat'] #
    umístění dat
FREQ=2000000 # vzorkovací frekvence - 2MHz
NUM_OF_PICTURES=9000 # počet souborů, které chceme vytvořit pro každý letový režim
NUM_OF_CORES=8 # počet jader procesoru
TEST_PROPORTION=0.2 # kolik procent dat použijeme k testování (zbytek tvoří trénovací a validační
    data dohromady)
```

4.2 Třídy

4.2.1 KernelFunction

```
class KernelFunction: # základní třída pro naše transformace (pouze z jednoho zdroje)
  def __init__(self,source_index,windows,param): # konstruktor
    self.source_index=int(source_index) # informace o tom, který zdrojový soubor je
        používán
    self.windows=windows # [window_length,window_number,shift]
    self.window_length=int(windows[0]) # šířka okna transformace
    self.window_number=int(windows[1]) # počet oken transformace
    self.shift=int(windows[2]) # posun oken transformace
    self.parametres=param # list případných ostatních parametrů transformace
    self.opened_file=open(SOURCES[self.source_index],'rb') # otevření
        souboru
```

```
        self.transformed_length=self.window_length+(self.window_number-1)*
            self.shift
    def __del__(self): # destruktor
        self.opened_file.close() # zavření souboru
    def get_data(self): # načítání dat
        loaded_raw=np.array([int.from_bytes(self.opened_file.read(2),
            byteorder='little', signed=True) for i in range(self.
            transformed_length)])
        #window_values=np.array([(1-(2*i/N-1)**2) for i in range(self.transformed_length)]) # Welchovo
            okno
        window_values=np.array([1 for i in range(self.transformed_length)])
             # Obdélníkové okno
        return loaded_raw*window_values
        # zdrojový soubor je uložen v 2B hodnotách v Little-Endianu
    def set_reader(self,position): # nastaví pozici, ze které se budou číst data
        self.opened_file.seek(12+position*FREQ*2,0) # přeskočí prvních 12B dat,
            protože jsou v nich uloženy informace o měření
    def start_pic(self,section_position): # přeskočí úsek dat tak, aby bylo možné začít
        např. tisícím úsekem (který transformujeme)
        self.opened_file.seek(self.transformed_length*section_position*2,1)
```

4.2.2 Standardizer_1D

```
class standardizer_1D(KernelFunction):
    def transform(self): # transformace, zde pouze odstranění lineární závislosti
        loaded_data=self.get_data()
        loaded_data=np.array(loaded_data-np.mean(loaded_data),dtype="
            float32")
        return loaded_data/np.std(loaded_data)
```

4.2.3 CHFDSpektrogram

```
class CHFDSpektrogram(KernelFunction):
def transform(self): # samotná funkce transformace, parametry: [frekvence]
cdef int x,k,y

cdef double[:] frequency=self.parametres[0]
cdef double[:] loaded_data=signal.detrend(self.get_data())

cdef int y_wide = self.parametres[0].shape[0]
cdef int x_wide = self.window_number

cdef double transformed_data[128][128]
cdef int shift =self.shift
cdef int window_length =self.window_length
cdef complex *exp_core = <complex *> malloc(window_length * sizeof(
    complex))
cdef complex temp
for y in range(y_wide): # iterace přes frekvence
for k in range(window_length):
exp_core[k] = cos(frequency[y]*pi*k)+1j*sin(frequency[y]*pi*k)
for x in range(x_wide): # zde se některá místa přeskakují
temp=0*1j
for k in range(window_length):
temp+=exp_core[k]*loaded_data[x*shift+k]
```

```
transformed_data[y][x]=abs(temp)
free(exp_core)
return np.array(transformed_data)
```

4.3 Funkce

4.3.1 generate_1D

```
def generate_1D(path,transformed_data):
  transformed_data=np.array(transformed_data)
  transformed_data=transformed_data.reshape((transformed_data.shape[0]*
      transformed_data.shape[1])) # převedení na
      vektor
  np.save(path,transformed_data) # uložení jako binární soubor numpy
```

4.3.2 generate_3D

```
def generate_3D(path,transformed_data):
  transformed_data=[np.array(i*255/np.max(i),dtype=np.uint8) for i in
      transformed_data] # škálování
      dat
  picture=np.zeros((transformed_data[0].shape[0],transformed_data[0].
      shape[1],CHANNELS),dtype=np.uint8) # alokace
      paměti
  for x in range(transformed_data[0].shape[0]): # převod dat z rozměru (4,x,y) na
      (x,y,4)
    for y in range(transformed_data[0].shape[1]):
      picture[x,y]=[j[x,y] for j in transformed_data]
  Image.fromarray(picture, mode="CMYK").save(path) # uložení 4-kanálového
      obrázku jako CMYK
```

4.3.3 gen_interface

```
def gen_interface(folder_ID,core_ID,shift=1): # (číslo letového režimu, jádro
    procesoru)
  # v této části se inicializují objekty transformací
  frekvence = np.array([i / 10000.0 for i in range(0, 2048, int
      (2048/128))]) # frekvence v poměru k
      Nq
  fctions=[CHFDSpektrogram(i,windows=[128,128,shift],param=[frekvence])
      for i in range(CHANNELS)]
  #fctions=[CSTransform(i,4096,128,[64,int(512/64)]) for i in range(CHANNELS)]
  #fctions=[Standardizer_1D(i,windows=[512,1,shift],[]) for i in range(CHANNELS)]
  start_position=int(NUM_OF_PICTURES*core_ID/NUM_OF_CORES) # pozice úseku
  for i in range(CHANNELS): # nastavení správného času
    fctions[i].set_reader(times[folder_ID])
    fctions[i].start_pic(start_position)
  for section_ID in range(int(start_position),int(start_position+
      NUM_OF_PICTURES/NUM_OF_CORES)):
    # jednotlivá jádra procesoru si rozdělí práci, např. první procesor si vezme 0-1124, druhý
        1125-2249,...
    transformed_data=[fctions[i].transform() for i in range(CHANNELS)]
        # transformuje všechny
        kanály
```

```
        generate_3D("/home/martin/Bachelor/{0}/{0}-{1}.jpg".format(
            folder_ID,section_ID),transformed_data)
        #generate_1D("/home/martin/Bachelor/0/0-1".format(folder_ID,repeat),transformed_data)
```

4.3.4 prepare_dataset

```
def prepare_dataset(target_folder): # roztřídí vytvořená data
    for number in range(1,12):
        names=np.array(["{0}/{0}-{1}.*".format(number,repeat) for repeat in
            range(NUM_OF_PICTURES)])
        np.random.shuffle(names)
    for i in range(int(NUM_OF_PICTURES*TEST_PROPORTION)):
        os.system("mv␣/home/martin/Bachelor/{}␣{}/test/".format(names[i],
            target_folder)) # všechna testovací
            data
        os.system("mv␣/home/martin/Bachelor/{}/*␣{}/data/".format(number,
            target_folder)) # trénovací a validační data
            dohromady
```

Závěr

Cílem této práce bylo seznámit se s teoretickým modelem konvolučních neuronových sítí (CNN) a za pomoci dostupných knihoven tento model implementovat a aplikovat na rozpoznání letových režimů vrtulníku na základě ultrazvukových signálů z kriticky namáhaných součástí hlavní převodovky. Důležitým úkolem bylo interpretovat již naměřená data pomocí vhodně zvolené časo-frekvenční transformace a následně provést numerické experimenty pro ověření rychlosti učení a úspešnosti rozpoznávání zcela nových dat v závislosti na navržené architektuře sítí.

První kapitola představuje historický průřez teorie umělých neuronových sítí, zejména dále nejvíce používaných vrstevnatých sítí. Součástí kapitoly jsou i architektury, které nebyly uvažovány v rámci experimentální části, ale historicky představují alternativu při návrhu klasifikátorů. Detailněji jsou samostatně v druhé kapitole rozebrány především konvoluční neuronové sítě jakožto rozšíření vrstevnatých.

Dosažené výsledky popisované ve třetí kapitole, zaměřené na numerické experimenty, lze shrnout do následujících poznatků.

Porovnáním výsledků CNN s vrstevnatými sítěmi se potvrdilo, že CNN jsou méně citlivé na posun signálu o jednotlivé vzorky. Jejich výhoda spočívá ve využití časo-frekvenční interpretace signálů. Zejména se osvědčilo využití zobecněného HFD spektrogramu, pomocí kterého lze analyzovat libovolně zvolené frekvence i pro časová okna různých délek.

Provedené numerické experimenty porovnávající různá nastavení parametrů vrstevnatých i konvolučních neuronových sítí prokázaly nevhodnost vrstevnatých sítí pro analýzu delších časových úseků signálu. První vrstva sítě totiž musí obsahovat příliš mnoho neuronů a celkový počet jejích parametrů se proto zvyšuje na příliš vysokou úroven. Trénování pak trvá dlouho, zvyšují se nároky na hardware a nedochází ani ke zlepšení rozpoznávacích schopností sítě. Naopak je demonstrován přínos použití L2 regularizací a je ukázáno, že dropout nemusí vždy zlepšovat kvalitu sítí. Toho je dosaženo například tehdy, pokud se dropout použije současně se svazkovou normalizací a L2 regularizací s pětinovou hodnotou.

Klíčovým poznatkem při rozpoznávání signálů je přínos zavedení konvolučních vrstev do architektury ANN. Ukazuje se, že CNN dosahuje lepší úspešnosti už na polovičním časovém úseku transformovaném pomocí HFD spektrogramu. Významný nárůst úspešnosti pak nastává s postupným zvyšováním posunu časového okna, přičemž se transformuje delší časový úsek signálu za stejné výpočetní náročnosti. Tímto krokem dochází k efektivnímu vytěžení informace obsažené v nameřeném signálu. Bylo prokázáno, že delší posuny časového okna výrazně zvyšují úspěšnost rozpoznávání sítě až na hodnotu kolem

95% pro takový posun, kde už vůbec nedochází k překrývání jednotlivých časových oken.

Závěrem lze konstatovat, že CNN byly úspěšně aplikovány při identifikaci původu ulrazvukových signálů a mají tak velký potenciál být v praxi použity pro detekci vnitřních procesů a poruch v materiálech na základě detekovaných ultrazvukových elastických vln.

Literatura

[1] *Neural Networks - History*. URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html (cit. 04. 03. 2020).

[2] Akshay L. Chandra. *Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works*. en. Zář. 2018. URL: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975 (cit. 04. 03. 2020).

[3] Akshay L. Chandra. *Perceptron: The Artificial Neuron (An Essential Upgrade To The McCulloch-Pitts Neuron)*. en. Zář. 2018. URL: https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d (cit. 04. 03. 2020).

[4] vibhor nigam. *Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning*. en. Ún. 2020. URL: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90 (cit. 04. 03. 2020).

[5] François Chollet. *Deep learning with Python*. OCLC: ocn982650571. Shelter Island, New York: Manning Publications Co, 2018. ISBN: 9781617294433.

[6] *Keras Documentation*. URL: https://keras.io/ (cit. 04. 03. 2020).

[7] *The Math Behind Backpropagation | Big Theta*. URL: https://bigtheta.io/2016/02/27/the-math-behind-backpropagation.html (cit. 04. 03. 2020).

[8] George Seif. *Understanding the 3 most common loss functions for Machine Learning Regression*. en. Červ. 2019. URL: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3 (cit. 10. 03. 2020).

[9] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. en. Ún. 2019. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a (cit. 14. 03. 2020).

[10] *Visual Information Theory – colah's blog*. URL: http://colah.github.io/posts/2015-09-Visual-Information/ (cit. 14. 03. 2020).

[11] Raúl Rojas. *Neural Networks: a Systematic Introduction*. English. OCLC: 1086530934. 1996. ISBN: 9783642610684. URL: https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=3093727 (cit. 04. 03. 2020).

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

[13] James Dellinger. *Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming*. en. Dub. 2019. URL: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79 (cit. 03. 03. 2020).

[14] Daniel Godoy. *Hyper-parameters in Action! Part II - Weight Initializers*. en. Pros. 2018. URL: https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404 (cit. 03. 03. 2020).

[15] *Initializers - Keras Documentation*. URL: https://keras.io/initializers/ (cit. 03. 03. 2020).

[16] *Xavier Initialization · Manas George*. URL: https://mnsgrg.com/2017/12/21/xavier-initialization/ (cit. 03. 03. 2020).

[17] Glorot Xavier a Bengio Yoshua. *Understanding the difficulty of training deep feedforward neural networks*. en. publisher: DIRO, Université de Montréal, Montréal, Québec, Canada. URL: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (cit. 03. 03. 2020).

[18] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv:1502.01852 [cs]* (ún. 2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852 (cit. 03. 03. 2020).

[19] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), s. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. 15. 03. 2020).

[20] Sergey Ioffe a Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]* (břez. 2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (cit. 14. 04. 2020).

[21] Vijini Mallawaarachchi. *Introduction to Genetic Algorithms — Including Example Code*. en. Břez. 2020. URL: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (cit. 14. 03. 2020).

[22] Swanand Mhalagi. *The Quest of Higher Accuracy for CNN Models*. en. Květ. 2019. URL: https://towardsdatascience.com/the-quest-of-higher-accuracy-for-cnn-models-42df5d731faf (cit. 14. 03. 2020).

[23] Matt Harvey. *Let's evolve a neural network with a genetic algorithm—code included*. en. Dub. 2017. URL: https://blog.coast.ai/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164 (cit. 14. 03. 2020).

[24] *Introduction to Optimization with Genetic Algorithm*. en. URL: https://www.linkedin.com/pulse/introduction-optimization-genetic-algorithm-ahmed-gad (cit. 14. 03. 2020).

[25] Filippo Galli. *Hopfield Networks are useless. Here's why you should learn them.* en. Dub. 2019. URL: https://towardsdatascience.com/hopfield-networks-are-useless-heres-why-you-should-learn-them-f0930ebeadcd (cit. 04. 03. 2020).

[26] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (říj. 2017). arXiv: 1503.04069, s. 2222–2232. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2016.2582924. URL: http://arxiv.org/abs/1503.04069 (cit. 04. 03. 2020).

[27] Eklavya. *Kohonen Self-Organizing Maps*. en. Říj. 2019. URL: https://towardsdatascience.com/kohonen-self-organizing-maps-a29040d688da (cit. 04. 03. 2020).

[28] Achraf KHAZRI. *Self Organizing Maps*. en. Srp. 2019. URL: https://towardsdatascience.com/self-organizing-maps-1b7d2a84e065 (cit. 04. 03. 2020).

[29] Jiwon Jeong. *The Most Intuitive and Easiest Guide for CNN*. en. Čvc 2019. URL: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480 (cit. 15. 03. 2020).

[30] *Convolutional Layers - Keras Documentation*. URL: https://keras.io/layers/convolutional/ (cit. 16. 03. 2020).

[31] Mukul Rathi. *Backpropagation in a Convolutional Neural Network*. en. URL: https://mukulrathi.com/demystifying-deep-learning/conv-net-backpropagation-maths-intuition-derivation/ (cit. 01. 04. 2020).