FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Bakalářská práce

Rozpoznávání a detekce klíčových slov

Autor:
Anton Lytvyniuk

Vedoucí práce:
Ing. Luboš Šmídl, Ph.D.

7. srpna 2022

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 7. srpna 2022 ________________________

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Anton LYTVYNIUK**
Osobní číslo:	**A19B0362P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Umělá inteligence a automatizace**
Téma práce:	**Rozpoznávání a detekce klíčových slov**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Nastudujte problematiku automatického rozpoznávání řeči, zaměřte se na metody detekce a rozpoznávání klíčových slov a frází.
2. Navrhněte a natrénujte model pro detekci klíčového slova pro offline a online detekci. Uvažujte možnost nasazení na zařízení s malým výpočetním výkonem.
3. Modely otestujte a vyhodnoťte.

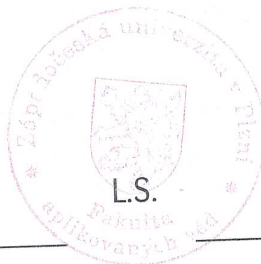
Rozsah bakalářské práce: **30 – 40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

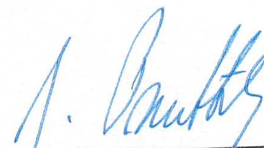
Dodá vedoucí práce.

Vedoucí bakalářské práce: **Ing. Luboš Šmídl, Ph.D.**
Katedra kybernetiky

Datum zadání bakalářské práce: **15. října 2021**
Termín odevzdání bakalářské práce: **23. května 2022**



L.S.



Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2021

ZÁPADOČESKÁ UNIVERZITA

Fakulta aplikovaných věd

Katedra kybernetiky

Rozpoznávání a detekce klíčových slov

Anton Lytvyniuk

Abstract

S rozvojem hlubokého strojového učení roste i jeho využití v chytrých zařízeních, proto existuje velká potřeba provozování modelů v kontextu omezené výpočetní síly. Jednou z významných úloh v oblasti rozpoznávání řeči a komunikace člověk-stroj je detekce klíčových slov (keyword spotting, KWS). Tato práce vyšetřuje různé architektury neuronových sítí a porovnává učení s učitelem a učení částečně s učitelem. Zjistilo se, že z navržených sítí nejlepších výsledků dosahuje síť ResNet8 a že model před-trénovaný na datové sadě LibriSpeech Light se dobře přenáší na úlohu KWS.

Keyword detection and recognition

With the development of deep machine learning its usage becomes increasingly popular in smart devices, consequently there is a significant need of operating such models in the context of limited computational power. One of the frequent tasks in the domain of speech recognition and human-machine communication is keyword spotting (KWS). This paper investigates multiple neural network architectures and compares supervised and self-supervised learning approaches. We show that the ResNet8 architecture achieves the best results and that the model pretrained on LibriSpeech Light dataset transfers well to the KWS task.

Keywords: speech recognition, keyword spotting, deep neural network, self-supervised learning, low footprint.

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Luboši Šmídlovi, Ph.D., za cenné rady a konzultace při opravě skriptů a dokumentace. Dále děkuji organizaci MetaCentrum a společnosti Google za poskytnutí výpočetní síly a vývojového prostředí.

Obsah

1 Formulace problému 12
1.1 Úkoly 12

2 Současný stav řešené problematiky 12

3 Navržený způsob řešení 13
3.1 Neuronové sítě 14
3.1.1 Rozvoj 14
3.1.2 Trénování 15
3.1.3 Návrh architektury sítě 17
3.1.4 Konvoluční vrstvy 18
3.1.5 Rekurentní vrstvy 20
3.1.6 Transformer vrstvy 21
3.1.7 Další použité prvky 21
3.2 Základy Tensorflow 22
3.2.1 Hardwarové zrychlení 22
3.2.2 Nevýhody statické kompilace 22
3.2.3 Eager mode 23
3.2.4 Automatická diferenciace 23
3.2.5 Modul tf.data 23
3.3 Supervised learning 24
3.3.1 Simple dense a Permute dense 25
3.3.2 ResNet8 a dsResNet9 25
3.3.3 Transformer a GRU 27
3.4 Učení částečně s učitelem 27
3.4.1 Autoencoder 30
3.5 Dataset 32
3.5.1 Použité datasety 32
3.6 Předzpracovaní nahrávek 33
3.6.1 Tensorflow dataset 33
3.6.2 LibriSpeech Light 35
3.7 Přetrénování a podtrénování 35
3.8 Validace 36
3.9 Augmentace 37
3.9.1 Augmentace amplitudy 37
3.9.2 Augmentace šumem 38
3.9.3 Augmentace času a frekvence 38
3.10 Regularizace 38
3.11 Volba optimizátoru 39
3.11.1 Stochastic gradient descent 39
3.11.2 Adam 40
3.11.3 LARS a LAMB 40
3.12 Ztrátová funkce 40
3.13 Trénink a optimalizace 41

4 Prezentace a zhodnocení výsledků **42**
4.1 Výsledky učení s učitelem 43
4.2 Výsledky učení částečně s učitelem 45

5 Závěr **47**

6 Bibliografie **47**

Seznam obrázků

1 Architektura Simple dense 26
2 Architektura Permute dense 26
3 Architektura Resnet8 28
4 Architektura dsResnet9 28
5 Architektura Transformer 29
6 Architektura GRU 29
7 Architektura MAE Small 31
8 Architektura MAE Large 31
9 Příklad výrazného přetrénování modelu. 36
10 Příklad podtrénování modelu. 36
11 Průběh přesností pro nejlepší model 43
12 Průběh ztrátové funkce pro nejlepší model 44
13 Průběh přesností pro nejhorší model 44
14 Průběh ztrátové funkce pro před-trénování na LibriSpeech datasetu 46
15 Průběh přesností pro finetuning 46

Seznam tabulek

1 Počet volných parametrů pro jednotlivé modely - učení s učitelem 25
2 Parametry trénování pro jednotlivé modely - učení s učitelem. . 41
3 Validace modelů - učení s učitelem. 43
4 Matice četnosti klasifikace pro model ResNet8. 45
5 Validace modelů - učení bez učitele. 45

Seznam zkratek

KWS - Keyword Spotting, rozpoznávání klíčových slov

BS - Batch size

CPU - Central Processing Unit

GPU - Graphical Processing Unit

SOTA - State of the Art

HMM - Hidden Markov Model

Conv - Konvoluční

dsConv - Depthwise separable convolution

MSE - Mean Squared Error

SGD - Stochastic Gradient Descent

Loss - Ztrátová funkce

LR - Koeficient učení, learning rate

ReLU - Rectified Linear Unit

NAS - Neural Architecture Search

XLA - Accelerated Linear Algebra

JIT - Just In Time

AE - Autoenkodér

NLP - Natural Language Processing, zpracování přirozené řeči

STFT - Short Term Fourier Transform, krátkodobá fourierova transformace

1 Formulace problému

Klíčovou úlohou problému Keyword-detection je neustále sledovat zvukový signál a detekovat znění případné klíčové fráze. Problém je komplikovanější kvůli velké variabilitě vedlejších faktorů, jako např. šum, zvuky prostředí, zkreslení mikrofonu, více hlasům na pozadí.

Z technického hlediska v každém okamžiku v čase se hledá podmíněná pravděpodobnost znění jednoho nebo vice klíčových slov, případně krátkých frází, za podmínky vstupního signálu. Tedy se jedná o úlohu klasifikace do $N + 1$ známých tříd, kde N je počet klíčových slov, a zbylá třída reprezentuje všechny ostatní případy a ticho. V této práci se primárně pracuje s datasetem Speech Commands Data Set v0.01 [60], tedy úkolem je rozpoznávání deseti hlavních klíčových slov. Více o datasetu v sekci (3.5)

V praxi celý algoritmus často běží na embedded zařízení nebo zařízení s malým výpočetním výkonem (např. smartphone), tím nás omezuje v maximální době kalkulace algoritmu, a tedy i jeho komplexitě. Implementace by měla být robustní nejen vůči vlivu již vyjmenovaných vnějších faktorů, ale také vůči individuálním charakteristikám hlasu (přízvuk, tempo řeči, hlasivkové tóny, emoční zabarvení). Zároveň by program měl být dostatečně rychlý, aby se dal spustit na hardwaru Raspberry Pi 3 v reálném čase. Proto jako algoritmus se zvolila metoda z oblasti strojového učení - hluboká neuronová síť (3.1).

1.1 Úkoly

- Prvním úkolem bylo nalézt a experimentálně porovnat optimální architektury neuronové sítě a popsat vliv jednotlivých prvků na přesnost klasifikace.
- Druhým úkolem bylo diskutovat nasazení algoritmu na reálný hardware.

2 Současný stav řešené problematiky

Za první řešení úlohy KWS se v literatuře považuje experimentální rozpoznávání řeči na základě detekce formantů [55]. Potom dlouhou dobu první místa držely přístupy na základě skrytých Markovových modelů (Hidden Markov Models - HMM) [44], které se často používaly pro modelování dat sekvenčního typu.

I když neuronové sítě mají svoje počátky v 60. letech a pro úlohu KWS se používaly i dříve [18], opravdový rozvoj začal až po roce 2012, kdy se pro trénování neuronových sítí začalo široce používat hardwarové zrychlení na GPU.

Za velký milestone se považuje sít AlexNet, která řešila úlohu klasifikace z počítačového vidění pomocí konvolučních sítí [34]. Za hlavní přínos této práce

se považuje implementace těchto konvolučních vrstev na GPU. Díky zlevnění výpočetní síly, ale i obecně technickému progresu v oblastí výpočetní techniky, neuronové sítě zažily prudký rozvoj, který trvá doteď. Dalším faktorem, který podporuje využití neuronových sítí v praxi je velké množství generovaných dat (počítače, smartphony).

V půlce předchozího desetiletí se řešení problému pomocí neuronových sítí dostaly mezi SOTA v úloze KWS, a tuto pozici drží pořád [57]. Ve srovnání s HMM, navržené neuronové sítě nejen mají o řád nižší chybovost, ale také jsou méně výpočetně náročné při dekódování, což se vyplatí u embedded zařízení kvůli menší spotřebě energie a zlepšení odezvy.

Jako jednu ze základních deep-learning architektur se dá považovat klasický vícevrstvý perceptron. Taková jednoduchá síť je rychlá na zpracování a má malé nároky na RAM paměť (zejména uchovávání aktivací vrstev). Kvůli vysoké míře propojení jednotlivých neuronů a absenci biasů (náklonností k typu dat) se ale taková síť při tréningu rychle dostává do stavu přetrénování (přetrénování, 3.7), což nakonec vede na poměrně malou přesnost rozpoznávání na reálných datech a náchylnost k šumů.

Jako pokus využití biasu lokality byly navrženy architektury na bázi konvolučních vrstev. Tento typ sítí je více odolný vůči šumu a absolutní lokaci příznaků na časově-frekvenčním spektrogramu. Nevýhodou konvolučních sítí je neefektivní zachycení vztahů mezi dalekými příznaky a složitější realizace stateful (proudového) rozpoznávání (vyžaduje cachování aktivačních map). Od roku 2018 se začaly častěji používat vrstvy depthwise separable convolution [10], které se dokáží naučit stejnou logiku jako standardní konvoluční vrstvy, ale přitom využívají miň parametrů [64].

Některé práce [52] se vydaly jiným směrem a zkoumaly použití rekurentních sítí, které jsou stejně jako HMM, efektivní pro modelování časových posloupností. Kromě standardních rekurentních sítí se navíc zkoumaly jejich vylepšené verze [61] nebo použití transformerů [6].

Některé modernější práce využívají různé druhy residualních sítí ve spojení s postupy, jako např. triplet loss [57] nebo spojením 1D a 2D konvolucí [32]. Tyto práce značně vylepšují oproti předchozím implementacím a patří mezi SOTA řešení. Na druhou stranu, ConvMixer [41] se soustředí na snižování počtu parametrů a zvýšení efektivity sítě.

3 Navržený způsob řešení

V jádru systému leží neuronová síť, která slouží k dekódovaní řeči a následné klasifikaci do jednotlivých tříd. Byly navrženy a natrénovány neuronové sítí s použitím následujících přístupů:

- Přímé end-to-end natrénování neuronové sítě s učitelem
- Předtrénování a finetuning - učení částečně s učitelem (self-supervised learning)

Pro učení s učitelem se navrhnou několik architektur s využitím různých typů vrstev a biasů, které se porovnají z hlediska přesnosti rozpoznávání jednotlivých slov. U učení částečně s učitelem se vyzkouší natrénovat maskovací autoencoder dvou různých velikostí na obou použitých datasetech (3.5).
Na výsledných modelech provedeme finetuning a porovnáme výsledky s klasickým učením s učitelem. Ve druhé části práce se bude diskutovat nasazení modelu na zařízení s malým výpočetním výkonem (Raspberry Pi). Jako framework se zvolil Tensorflow, který se použil ve spojení s interpretovaným jazykem Python 3. Navíc se použily balíčky Numpy, pro načítání a práci s vektory, Pandas pro manipulaci s datasetem a tf_extensions, ze kterého se použil optimizační algoritmus LAMB.

3.1 Neuronové sítě

Ve dnešní době se jedná o nejkomplexnější a zároveň nejpřesnější algoritmy strojového učení. Mohou řešit zároveň úlohy klasifikace a regrese, a patří do extrému z hlediska množství potřebného výkonu a dat. Užitečnou vlastností takových sítí je velká flexibilita, která umožňuje jejich aplikaci na velkou množinu úkolů. Využívají se zejména v úlohách rozpoznávání obrazů [22], řeči [4] a zpracování přirozeného jazyka [12]. V posledních několika letech neuronové sítě začínají pronikat do dalších odvětví, jako např. modelování bílkovin [29], strukturovaných dat [62], učení s podporou [8]. Navíc na rozdíl od klasických algoritmů strojového učení pro trénování neuronových sítí není potřeba provádět manuální navržení a selekci příznaků, proto je možné tyto sítě optimalizovat zcela automaticky (end-to-end). Pro dosažení nejlepších výsledků je ale nezbytná optimální architektura sítě a zavedení efektivních augmentací, čemuž se budeme věnovat v rámci této práce.

3.1.1 Rozvoj

Počátky neuronových sítí vedou k prvnímu matematickému modelu umělého neuronu (McCulloch-Pits, 1943) a jeho zobecnění [45], kde se neuron popisuje jako funkce více proměnných z $\mathbb{R}^N$ do $\mathbb{R}$, kde N je počet vstupů. Tato funkce je tvořena váženou sumou vstupů, přidanou konstantou (bias) a binární aktivační funkcí, jejíž účelem je zavést nelinearitu. Za parametry, které se někdy nazývají váhy, se dají považovat koeficienty vážené sumy a biasu (zesílení). Tento model se výrazně liší od reálných neuronu, které jsou navíc dynamické v čase a jejích chování je řízeno desítkami neuromediátorů.

Reálný biologický neuron obsahuje větší počet synapsí, které slouží jako vstupy, a jeden axon - výstup, který je fyzicky napojen na synapse dalších neuronů. Komunikace mezi neurony probíhá pomocí elektrických pulzů (v synapsích) a

komunikaci mezi axonem a synapsí zprostředkovávají neuromediátory (molekuly, které dokážou změnit chování neuronu).

Jeden takový umělý neuron kvůli své lineární povaze nedokáže modelovat některé jednoduché funkce, např. XOR, což ve své době vedlo k poklesu zájmu k neuronovým sítím. Řešením bylo uspořádaní neuronů do vrstev. V tomto případě se jedná o hustě propojenou vrstvu (Fully connected/Dense). Několik vzájemně propojených vrstev tvoří neuronovou sít. Pokud je síť tvořena pouze hustě propojenými vrstvami, pak se taková síť nazývá Multilayer Perceptron [46]. [36] ukázal, že neuronová síť s dostatečně širokou latentní (skrytou) vrstvou a jednou výstupní vrstvou dokáže aproximovat libovolnou spojitou funkci s nulovou odchylkou.

Existuje více topologií takových sítí, ale dnes nejpoužívanější jsou feed-forward sítě, ve kterých informace se propaguje směrem od vstupních vrstev, přes latentní, a končí ve výstupních vrstvách. Každá vrstva je popsaná počtem neuronů a její typem. Starší architektury měly vrstvy uspořádány striktně za sebou, v novějších se můžou objevovat paralelní větve nebo skip-connection [22].
Existují i přesnější modely, jako např. spikové neuronové sítě [39], ale dnes nemají praktické uplatnění a používají se zejména pro výzkum. Hlavními důvody je větší výkonová náročnost a absence robustního trénovacího algoritmu.

3.1.2 Trénování

Pod pojmem trénování (neboli fitování) se myslí vyřešení optimalizační úlohy vzhledem ke ztrátové funkci L a parametrům sítě θ, kde chceme získat parametry příslušné minimální hodnotě ztrátové funkce. V případě učení s učitelem hodnota ztrátové funkce závisí na dvou tenzorech - opravdovém a požadovaném výstupu sítě. Níž je uveden vzorec pro jednu se základních ztrátových funkcí - Mean Squared Error. Víc o ztrátových funkcích v sekci (3.12).

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Na trénování některých neuronových sítí bez skrytých vrstev se lze dívat jako na lineární nebo logistickou regresi, v závislosti na výstupní aktivační funkci. V případě použití ztrátové funkce MSE (víc v sekci 3.12) lze použit metodu nejmenších čtverců a tím rovnou získat parametry odpovídající globálnímu minimu ztrátové funkce.

U hlubších neuronových sítí s nelineárními aktivačními funkcemi se jedná o nekonvexní optimalizaci, tedy o NP-úplnou úlohu. Existuje větší množství optimalizačních algoritmů (tzv. optimizátorů) pro fitování neuronových sítí, ale většina z nich je založena na principu gradientního sestupu (gradient descent). Tento postup spočívá v iterativním pohybu hodnot jednotlivých parametrů sítě proti směru jejich gradientu. Po dostatečném počtu iterací ztrátová funkce skončí v lokálním nebo globálním minimu, ze kterého se již nedostane. Novější

algoritmy používají jisté modifikace, jako např. uvažování druhých derivací vah, inercie, Moving Average v prostoru parametrů, které zrychluji konvergenci. Víc o optimizátorech v sekci (3.11).

Metoda Gradient Descent vyžaduje existenci gradientu pro každou možnou sadu parametrů θ, proto navrhovaná struktura neuronové sítě se často volí tak, aby síť odpovídala spojité funkci. Použití např. binární aktivační funkce by bylo problematické, protože má nulovou derivaci pro všechny hodnoty definičního oboru až na nulu, kde derivace neexistuje. Pro optimalizaci takové sítě by se mohla použit např. metoda konstantních přírůstků.

Jako první se získá gradient ztrátové funkce, tedy výpočet začíná od konce sítě a postupuje ve směru vstupní vrstvy. Gradient každé skryté vrstvy se počítá iterativně z gradientu vstupu následující vrstvy (který se získal v předchozím kroku) za pomocí vzorce pro derivaci složené funkce:

$$\frac{d}{dx}\left[f\left(u\right)\right] = \frac{d}{du}\left[f\left(u\right)\right]\frac{du}{dx}$$

Celý tento proces se nazývá backpropagation. V praxi backpropagation vyžaduje uložení výstupů všech vrstev, což klade vysoké nároky na paměť. V některých případech lze provést optimizaci a přepočítávat celou vrstvu (např. u samotné aktivační funkce) namísto uložení aktivačních map, protože čtení a zápis do pamětí jsou často nejpomalejší části výpočtu na dnešních počítacích strojích [13].

Jedna iterace trénovaní pomocí algoritmů založených na Gradient Descent se dá zobecnit a rozdělit na několik fází:

- Dopředný průchod - při dopředném průchodu neuronová síť zpracuje vstup a uloží výstup a všechny mezivýsledky (aktivační mapy) do paměti [13].
- Výpočet hodnoty ztrátové funkce - reálný výstup sítě se pomocí ztrátové funkce srovná s požadovaným výstupem, výsledkem je loss, který lze chápat jako jakousi "chybovost"sítě.
- Zpětný průchod - pomocí Backpropagation se spočte gradient parametrů vůči lossu.
- Aktualizace vah - v závislosti na implementaci se spočte směrový vektor v prostoru parametrů a v tomto směru se provede malý krok. Velikost kroků pak záleží na konstantě učení (LR) a často i na dynamice vah.

Tyto kroky se opakuji pro každý prvek z datasetu. Optimalizace sítě prvek po prvku se nazývá Online učení a je náchylná na jisté problémy:

- Celý proces se špatně paralelizuje, což znemožňuje efektivní použití hardwarového zrychlení.

- Jednou z obecných vlastností trénování neuronových sítí je katastrofické zapomínání. Některé kroky parametrů ve správném směru se můžou nechtěně vykrátit v následujících iteracích. Tento jev vede na neefektivní učení a nutnost optimalizace na stejných prvcích vícekrát (na rozdíl od např. metody nejmenších čtverců).

Pro snížení vlivu katastrofického zapomínání je potřeba většího počtu průchodů datasetem (epoch), řadově desítky až stovky. Dalším způsobem je využití tzv. mini-batchů - náhodně generovaných podmnožin datasetu o konstantní velikosti. Potom se gradient parametrů nepočítá jen pro jeden prvek datasetu, ale pro celý mini-batch, a výsledný gradient se získá průměrováním gradientů prvků mini-batche. Důsledkem je, že kvůli průměrování jeden prvek nedokáže výrazně negativně ovlivnit váhy sítě. Navíc výsledný gradient má menší rozptyl než v případě Online učení (vyplývá z centrální limitní věty), což umožňuje rychlejší konvergenci sítě.

Výpočet gradientů jednotlivých prvků v mini-batchi je si navzájem nezávislý, proto existuje možnost paralelizace. Větší rozměry mini-batche vedou na vyšší vytížení GPU, ale také můžou způsobit numerickou nestabilitu a případnou divergenci [31]. Toto omezení lze obejít např. pomocí metody gradient clipping nebo použitím specifické třídy optimizátorů - LARS/LAMB [63].

Před začátkem tréningu se musí zvolit počáteční množina parametrů, tedy provést tzv. inicializaci sítě. Obvykle se používá Xavierovo inicializace [20], kde počáteční parametry vrstev se čerpají z normálního rozdělení tak, aby výstupy všech vrstev měly nulovou střední hodnotu a konstantní rozptyl. Toto se umožňuje vyhnout divergenci nebo mizení gradientu (gradient vanishing) s rostoucím počtem vrstev.

Po dokončení všech trénovacích epoch parametry sítě pořád nemusí dokonvergovat do globálního minima a můžou se ustálit v lokálním. Někdy nalezení globálního minima není žádoucí, protože by se jednalo o přetrénování (víc v sekci 3.7). Však při korektně zvolených augmentacích a regularizačních metodách (3.9 a 3.10) i lokální minima jsou dostačující pro generalizaci.

V praxi se jednotlivé neurony nemodeluji samostatně, jejich parametry se v rámci jedné vrstvy seřazují do maticového tvaru za účelem zrychlení trénování a inference. Např. hustě propojená vrstva se dá zapsat jako maticové násobení matice parametrů vrstvy a její vstupu (pokud vynecháme aktivační funkci a bias). Některé vrstvy, jako např. rekurentní, nejdou dobře paralelizovat a proto se postupně zaměňuji transformery.

3.1.3 Návrh architektury sítě

Na vrstvy se může pohlížet jako na základní stavební bloky neuronové sítě, proto při návrhu architektury se volí typy vrstev, jejich počet a počet neuronů v jednotlivých vrstvách. Modernější architektury jsou navíc rozšířené z hlediska

topologie sítě (paralelní vrstvy, skip connection). Zatím ale neexistuji široce používané metody stanovení nejlepší architektury, proto se v úvahu bere empirická znalost a přístup pokus-omyl. V oblasti zpracování přirozeného jazyka se dříve používaly rekurentní sítě [53], kdyžto dnes jednoznačně vedou architektury na bázi transformeru [56]. Strojové vidění je silně spojeno s konvolučními vrstvami [35], které jsou vhodné pro zpracování prostorových dat. Novější práce ale úspěšně přizpůsobily transformery i k úlohám strojového vidění ([15], [21]), tedy tato architektura se dá považovat za univerzální a je vhodná pro zpracování dat z různých modalit, jak jednotlivě tak i pohromadě [28].

Novější praktikou je rozdělení vrstev a jejích aktivačních funkcí, často pro zavedení paralelních větví nebo použití normalizačních vrstev. Podobný princip se používá v sítích ResNet [22].

Existuje možnost použít NAS (Neural Architecture Search), ale je silně omezena výpočetní náročnosti úlohy [17], zejména pro větší datasety. Navíc existuje složitá závislost mezi šířkou (počet neuronů v jedné vrstvě), hloubkou (počet sériově zapojených vrstev) sítě a samotným datasetem, proto se architektura získána pomocí NAS nemusí dobře generalizovat na jiné úlohy.

3.1.4 Konvoluční vrstvy

Původně byly navrženy pro rozpoznávání rukopisného vstupu [35], později se staly hlavním blokem v úloze rozpoznávání obrazů. Jsou postaveny na diskrétní verzi matematické operace konvoluce, kde parametry kernelu jsou objektem optimalizace sítě. Při definici této vrstvy se udávají další parametry - rozměr jádra (kernelu), krok a počet kanálů (channels, paralelně běžících konvolucí s odlišnými váhami kernelů). Existují 1D, 2D, 3D varianty, kde se pro zpracování obrazů využívají 2D konvoluce. Pro každý bod se výstup spočte jako skalární součin hodnot čtvercového okolí bodu pro všechny kanály a váhy jádra, a po přičtení biasu výsledek projde aktivační funkci. Aby se jednalo o opravdovou konvoluci, váhy kernelu by měly být v opačném pořadí, což plyne ze vzorce pro konvoluci.

Tyto vrstvy dokáží dobře zachytit lokální vztahy a patterny, proto se často používají v různých úlohách počítačového vidění, a tedy jsou aplikovatelné i na časově-frekvenční data. V oblasti rozpoznávaní řeči se konvoluční vrstvy ukázaly jako robustní vůči šumu, protože berou v úvahu lokální kontext. Další výhodou je uplatnění principu sdělování vah (weight sharing) - každý kernel se aplikuje na každou možnou pozici v prostoru obrázku. Jako důsledek počet parametru konvolučních vrstev nezávisí na výšce ani šířce vstupního tenzoru (na rozdíl od např. hustě propojených vrstev). Nižší počet parametrů dává menší prostor k přetrénování (overfittingu) sítě a ve spojení s metodami paralelizace dosahují větší efektivity na GPU (menší počet přístupů k paměti, která je často hlavním úzkým hrdlem).

Pomoci hierarchického uspořádaní Conv2D a případně dalších vrstev pro redukci dimenzí (pooling), dokáže síť spolehlivě detekovat komplexní prostorové patterny. Nevýhodou by bylo horší zachycení vztahů mezi dalekými prvky na obrázku (protože by tato informace musela bezztrátově propagovat do hlubších vrstev). Empirické pokusy ukázaly, že efektivní strategii je postupné zvýšení počtu kanálů s rostoucí hloubkou, protože hlubší vrstvy mají větší receptivní pole (receptive field) a kódují komplexnější patterny v porovnání s vrstvami, které se nachází blíž ke vstupu sítě.

Při návrhu konvolučních architektur se obecně vyplatí dávat pozor na velikost receptivních polí [43]. V závislosti na velikosti kernelu, kroku (střídě) a vstupním rozlišení vzniká omezení na maximální efektivní hloubku sítě. Po dosažení této hloubky zavedení dalších vrstev málo ovlivňuje přesnost sítě, proto efektivnější by bylo škálovat síť do šířky. Větší modely obsahující víc nastavitelných parametrů jsou obvykle nejen náročnější na trénování, ale také mají větší spotřebu paměti a delší dobu výpočtu. Větší počet parametrů ale automaticky neznamená delší dobu výpočtu, která závisí hlavně na implementaci samotné vrstvy. Např. při stejném počtu parametrů konvoluční vrstva bude výrazně pomalejší než hustě propojená. Ani menší počet matematických operací (násobení a sčítání) nemusí znamenat kratší dobu výpočtu, např. v případě když je hardware omezen rychlostí přístupu do paměti.

Pokud se na Conv2D vrstvu podíváme jako na operaci transformace tenzorů, pak se dá popsat její vliv na tvar výstupu. Vstupní tenzor musí obsahovat čtyři dimenze $[batchsize, width, height, channels]$, kde v případě obrázků dimenze width a height popisují polohu jednoho pixelu. Pokud vstupem jsou černobílé obrázky bez dimenze kanálů, pak se má provést expanze poslední dimenze, která nastaví počet kanálů na jedna. Výsledné rozměry šířky a délky obrázku závisí především na velikosti kroku a v menší míře na velikosti kernelu.
Při jednotkovém kroku obě prostorové dimenze se redukuji o $(kernelsize - 1)$ pixelů, což plyne z toho, že konvoluce kromě samotného pixelu uvažuje i jeho okolí. Někdy se takové chování nemusí vyplatit, například když chceme zavést identickou vazbu ze vstupu na výstup. Řešením je doplnění nul na okrajích prostorových dimenzí vstupního tenzoru, které se ve Tensorflow zapne jednoduchým nastavením parametru *padding*.

Velikost kroku násobně redukuje výsledný tenzor a je často rychlejší, než použití ustálené kombinace konvoluční a Pooling vrstvy. Krok se ale obvykle nenastavuje na hodnoty větší než kernel size, protože by pak docházelo ke ztratě informace. Někdy je krok roven kernel size, což vede na rozdělení obrázku na nepřekrývající se vzorky (patche). Takový přístup se používá jako jedna z prvních vrstev vizuálních transformerů [15] nebo novějších konvolučních architektur [37].

Pro každou vrstvu obecně platí, že neovlivňuje první dimenzi, která popisuje počet prvků v mini-batchi. Výsledný počet kanálů ale přímo odpovídá počtu

kernelu a nezávisí na počtu kanálů na vstupu. Této vlastnosti lze využít pro expanzi a redukci kanálů, a tedy i zavedení úzkého hrdla (bottleneck) [49], nebo pro rozšíření vstupního obrázku, kde každý kanál odpovídá výstupu jednoho filtru.

Existuje více druhů konvolučních vrstev, jako např. pointwise a depthwise convolution. V prvním případě se jedná o obyčejnou 2D konvoluci s jednotkovým rozměrem kernelu, která se chová stejně, jako hustě propojená vrstva na ose kanálů. Depthwise convolution zpracovává každý kanál zvlášť, a proto taková vrstva obsahuje méně volných parametrů při stejné velikosti kernelu a počtu kanálů.

3.1.5 Rekurentní vrstvy

Hlavní vlastnost rekurentních vrstev je uchovávání stavového vektoru (paměť systému), který shromažďuje data ze všech uplynulých iterací a vyvíjí se v čase [50]. Z tohoto hlediska je koncept velmi podobný stavovému popisu dynamického systému. Hlavní odlišnosti je použití nelineární aktivační funkce při přiřazení hodnot stavu a výstupu, a absence přímé vazby ze vstupu na výstup. Rekurentní vrstvy se tradičně používaly pro zpracování časových posloupností, včetně přirozeného jazyka a řeči.

Způsob výpočtu celé sekvence je založen na rekurenci, tedy pro výpočet stavu n-tého vzorku je potřeba mít stav vzorku předchozího. Takový postup by byl efektivní při zpracování streamu dat, protože v každém kroce je potřeba pouze přepočítat jeden časový vzorek, což vede na složitost $O(N)$ vzhledem k délce posloupnosti. Navíc pro tento výpočet je postačující znalost stavového vektoru z předchozího kroku, tedy paměťová složitost při inferenci je konstantní. Existují ale dvě velké nevýhody rekurence - mizející gradient a nemožnost paralelizace.

I když problém paralelizace se v některých případech dá obejit rozbalením rekurentní vrstvy do série hustě propojených vrstev, tato metoda vyžaduje velké množství paměti, což často znemožňuje efektivní tréning pro delší posloupnosti. Počet těchto vrstev odpovídá počtu časových vzorků, proto v některých případech se může jednat o velmi hluboké sítě se stovkami vrstev. Bez použití sofistikovanějších normalizačních nebo inicializačních metod (jako např. DeepNorm [58]) by docházelo k mizení nebo explozi gradientu, které by vedlo buď k nestabilitě tréninku nebo k neúčinném zachycení vztahů vzdálených v čase. Pro zachycení větší časové vzdálenosti mezi příznaky by síť musela propagovat informaci přes více iterací téměř beze změn, což často není možné vzhledem ke konečné velikosti stavového vektoru.

Některé modifikace, především LSTM a GRU ([25], [9]) vyřešily problém nestabilních gradientů pomocí použití speciálních bran, které řídí tok informace. LSTM vrstva obsahuje tři brány - vstupní, výstupní a bránu zapomínání. Tyto brány propouští pouze důležité příznaky a odhazují již nerelevantní data. GRU

je novější, ale funguje na podobném principu jako LSTM. Liší se počtem bran, kterých pouze má dvě. Ve dnešní době rekurentní sítě jsou pro velkou část úloh nahrazeny transformery.

3.1.6 Transformer vrstvy

První architektura transformeru byla vyvinuta specificky pro strojový překlad jazyků [56], ale později se rozšířila i do dalších domén. Jedná se o univerzální vrstvy bez inherentního biasu, které nedávno posunuly celý obor dopředu kvůli jejím vlastnostem. Na rozdíl od rekurentních sítí, které zpracovávají posloupnosti dat v přesně daném pořadí, transformery je zpracovávají jako množiny. Výhodou množin je nezávislost na pořadí tokenů (časových vzorků), a proto existuje možnost například náhodného odstranění prvků z množiny, což je u rekurentních a konvolučních sítí problematické. Z hlediska časových posloupností je tato vlastnost velice přínosná, protože umožňuje práci s posloupnostmi, které nemají konstantní vzorkovací periodu. Navíc se dají spojit víc modalit do jedné latentní reprezentace, což vedlo k vzestupu odvětví text-to-image [48]. Transformery obecně pracují s tokeny, pomocí kterých jde popsat nejen signály, ale také obrázky (pomocí patchů) a grafy [16].

Další výhodou je velká míra paralelizace, jelikož se při výpočtu sítě nemusí omezovat rekurentním vztahem. Byl to jeden z důvodů přechodu z rekurentních sítí na transformery pro řešení velké části úloh. Práce s množinou ale vyžaduje stanovení relace mezi jednotlivými tokeny, proto algoritmus má kvadratickou složitost vůči velikosti množiny (délce posloupnosti). Proto se intenzivně zkoumaly způsoby optimalizace algoritmu. Nakonec byly vyvinuty některé varianty, které mají lineární nebo téměř lineární složitost ([11], [59]). Varianta Performer se odlišuje zpětnou kompatibilitou se standardní vrstvou, což umožňuje zrychlení již natrénovaných modelů, a také teoretickou garanci malé chyby odhadu.

Jeden blok se skládá ze dvou části - promíchání tokenů pomocí Self-Attention [5] a následném perceptronu o dvou vrstvách. Obě části mají zapojenou identickou paralelní vazbu ze vstupu na výstup s následnou normalizaci. Původní článek dále rozděloval celou architekturu na enkodér a dekodér, které se lišily hlavně použitím cross-attention vrstev a kauzálního maskování.

3.1.7 Další použité prvky

Jedná se o operace, které mají za úkol zrychlit konvergenci parametrů při trénování, zvýšit výslednou přesnost modelu nebo provádět obecné manipulace s tenzory. Některé vrstvy také nemají žádné nastavitelné parametry.

Pro zrychlení konvergence se do architektury často přidávají normalizační vrstvy. Jejich hlavním úkolem je manipulace s průměrem a rozptylem vstupního tenzoru. Dvě nejčastěji používané varianty jsou BatchNorm [27] a LayerNorm [3], a v rámci této práce se použiji obě. Batch Normalizace se tradičně používá v architekturách pro zpracování obrazů, když Layer-Norm je nezbytnou části

transformerů. Tyto dvě vrstvy se liší ve způsobu získávání průměrů a rozptylů, které se prvním případě počítají při tréningu pro celý mini-batch, ve druhém se získávají dynamicky při inferenci.

Positional embeddings jsou užitečné pro transformery, které pracují s daty jako s množinou a jsou invariantní k jejím pořadí. Tato vrstva obohacuje každý vstupní token informací o jeho pozici mezi dalšími tokeny. Dál jsou nezbytné vrstvy pro manipulaci s tenzory. K nim patří např. Reshape, Flatten, Average Pooling.

3.2 Základy Tensorflow

Jako základní stavební kámen se použil framework Tensorflow [2]. Je dostatečně flexibilní pro vytvoření komplexních systémů a zároveň obsahuje některé užitečné moduly, jako např. vysokoúrovňový interface pro návrh neuronových sítí (keras), interface pro efektivní práci s datasety (tf.data) a Tensorflow Lite, který umožňuje nasazení modelů na reálný hardware. Navíc podporuje vektorové výpočty a podporu výpočtu na GPU, což je velmi užitečné pro řešení optimalizačních úloh.

Základní jednotkou, umožňující výpočty na CPU, GPU a případně dalším hardwaru, jsou tenzory. Jedná se o imutabilní více-dimenzionální pole. Jedním z parametrů tenzorů je jejich tvar (shape), který udává velikosti jednotlivých dimenzí a jejich počet. V rámci této práce se vstupní data uvažují ve formě čáso-frekvenčních spektrogramů, tedy při učení neuronová síť dostává tenzor ve tvaru $[Batchsize, timesteps, melbins]$, a cílový (target) tenzor je potom ve tvaru $[batchsize, classcount]$. Dalším důležitým parametrem tenzorů v tomto frameworku je datový typ, který ovlivňuje přesnost dat, využití pamětí a často i rychlost výpočtu operací s daným tenzorem.

3.2.1 Hardwarové zrychlení

Pro zrychlení výpočtů Tensorflow používá framework CUDA. Každá operace má k dispozici vlastní CUDA kernel (zkompilovaný kód, který běží přímo na GPU). Při prvním spuštění skriptu probíhá trasování kódu a sestrojí se statický graf operací, kde každý vrchol grafu odpovídá jedné operaci, a tedy i jednomu CUDA kernelu. Znalost tohoto grafu umožňuje současné vykonávání paralelních větví a lepší plánování operací. Tensoflow navíc podporuje JIT (Just-In-Time) kompilaci kódu (knihovna XLA), která dokáže optimalizovat graf mimo jiné uspořádáním a sjednocením jednotlivých operací do jednoho kernelu. Výsledkem je často několikanásobné zrychlení výpočtů [1].

3.2.2 Nevýhody statické kompilace

Statická kompilace má i nevýhody - pro sestrojení grafu je třeba mít k dispozici tvar vstupních a výstupních tenzorů, který se získává pomocí operace trasování.

Problém ale nastává, když jeden z těchto tenzorů nemá stálý tvar (např. audio nahrávka o proměnné délce). V tomto případě operace trasovaná probíhá pro každý z možných tvarů, což vede ke zpomalení běhu programu. V případě JIT kompilace práci s proměnným tvarem tenzorů nejde realizovat. Toto klade značná omezení na počet použitelných metod (např. operace tf.where). Mezi další nevýhody patří obtížnost debugování kódu na GPU (některé funkce Pythonu se nepodporují a při kompilaci se ignorují), které vede na často zbytečně složité logy, a nemožnost dynamické alokace paměti.

3.2.3 Eager mode

Za účelem snažšího debugování Tensorflow podporuje Eager Mode - režim, kdy se operace spouští okamžitě bez sestavení grafu. Takové chování umožňuje využití nástrojů jazyka Python, jako např. funkce *timeit*, standardní datové struktury, dynamickou alokaci paměti. Tedy se jedná o zvýšenou flexibilitu na úkor rychlostí. Proto je tento režim zapnutý jako výchozí nastavení. Na druhou stranu, i když tento režim umožňuje vykonávání kódu na GPU, je výrazně pomalejší než statická kompilace.

3.2.4 Automatická diferenciace

Většina algoritmů pro optimalizaci neuronových sítí jsou založeny na výpočtu nebo odhadu gradientu vah vůči hodnotě ztrátové funkce. Samotný výpočet sice lze naprogramovat manuálně, ale takový přístup je zbytečně komplexní a náchylný k chybám pro hluboké sítě. Automatická diferenciace je nástrojem, který výrazně zrychlil vývoj v oblasti hlubokého strojového učení a zjednodušil návrh nových architektur neuronových sítí. V Tensorflow výpočet gradientu probíhá pomocí modulu GradientTape, u kterého lze nastavit množinu sledovaných parametrů a po provedení výpočtu získat jejich gradient vůči požadované veličině (ve strojovém učení se často jedná o loss).

3.2.5 Modul tf.data

Používá se pro vytváření efektivních datových pipelinů (zejména načítání a zpracování datasetu). Hlavní výhodou modulu je dynamické načítání a uvolnění datových prvků. Za pomocí této funkce lze provádět trénink na velkých datasetech, které se nevejdou do RAM paměti stroje. Nad daty se mohou provádět transformace (mapování funkcí), které jsou vhodné např. pro zavedení stochastických augmentací přímo za běhu tréninku sítě. Deterministická část pipelinu se pro účely optimalizace dá uložit na disk pomocí cachování, čímž se lze vyhnout zbytečným opakovaným výpočtům. Všechny operace se provádí na CPU, aby nebránily GPU provádět mnohem výpočetně náročnější část kódu (optimalizaci sítě). V tomto případě se příprava dat a výpočet sítě stále probíhá synchronně, tedy po dokončení zpracování jednoho mini-batche GPU bude čekat na nový mini-batch ze strany CPU. Řešením je funkce Prefetch, která slouží pro zvýšení vytížení (utilizace) obou procesorů. Umožňuje přípravu dat o jeden krok (mini-batch) dopředu a tím zavádí asynchronní zpracování na CPU a

GPU. Mezi další užitečné funkce patří Shuffle, Repeat a Batch, které provádí promíchání jednotlivých datových prvků (datapointů), umožňují generaci více prvků než je velikost datasetu a seřazují data do mini-batchů s předem nastavenými parametry. Všechny tyto operace se provádí až za běhu programu.

3.3 Supervised learning

Jedná se o optimalizaci neuronové sítě tak, aby co nejvíc vyhovovala cílovým datům podle jistých kritérií, které jsou realizovány v podobě ztrátové funkce. Úlohy strojového učení se dají rozdělit do dvou skupin:

- Regrese - spočívá v predikci jedné nebo více spojitých veličin ze vstupních dat.
- Klasifikace - zařazuje prvek do jedné nebo více předem definovaných skupin.

Úloha klasifikace se od regrese odlišuje použitím speciálních ztrátových funkcí (víc v sekci 3.12) a aktivačních funkcí výstupu, jako např. softmax nebo sigmoid (logistická funkce). Tato práce řeší zejména úlohu klasifikace, regrese se ale také použije v sekci učení částečně s učitelem 3.4.

Učení s učitelem vyžaduje data ve tvaru dvojice (Vstup, Výstup). Pro vytváření takových datasetů je potřeba manuální anotace dat expertem v řešené oblasti, a proto je jejich tvorba často zdlouhavá a drahá. Navíc taková data mohou obsahovat lidský faktor, který vede na biasy a chybné anotace.

V rámci této paradigmy bylo navrženo šest architektur. Při návrhu se orientovalo na chtěný počet parametrů, který je zhruba $1e5$. Jako aktivační funkce se u všech architektur používala ReLU.

- Simple Dense
- Permute Dense
- ResNet8
- dsResNet9
- GRU
- Transformer

Všechny sítě dostávají na vstup dvourozměrné časově-frekvenční spektrogramy, a na výstupu obsahují hustě propojenou vrstvu s počtem neuronů odpovídajícím počtu tříd (jedenáct) a aktivační funkci softmax. Tato poslední vrstva slouží jako výsledný klasifikátor.

Velikosti modelů		
Model	Statefull	Počet parametrů
Simple dense	Ne	106203
Permute dense	Ne	100843
ResNet8	Ne	103051
dsResNet9	Ne	103491
GRU	Ano	101259
Transformer	Ne	100363
MAE Small	Ne	108448
MAE Large	Ne	2751488

Tabulka 1: Počet volných parametrů pro jednotlivé modely - učení s učitelem

3.3.1 Simple dense a Permute dense

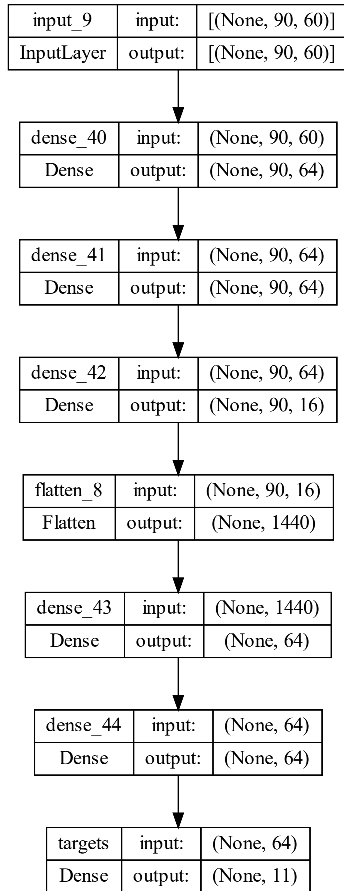
První architektura je velmi jednoduchá a využívá pouze hustě propojené vrstvy (1). Nejprve se první tři vrstvy aplikují na každý časový vzorek zvlášť a tím získají vnitřní reprezentaci znějících tonů. Následně se výsledky sloučí dohromady a aplikuje se další dvě hustě propojené vrstvy, které zpracovávají nahrávku globálně. Následuje projekce do počtu tříd s aktivační funkci softmax. Počty neuronů byly zvoleny intuitivně v souladu s požadovaným počtem parametrů. Víc než polovinu parametrů zabírá vrstva napojená na vrstvu Flatten z důvodu velikosti vstupního vektoru (dimenze 1440). Může ale být vhodné tyto parametry rozložit rovnoměrněji, proto se dál navrhla architektura Permute dense.

Myšlenkou je střídání zpracování dat na časové a frekvenční ose. Tuto funkci úspěšně plní operace Permute, která v případě spektrogramu funguje jako transpozice matice. Základním stavebním prvkem je posloupnost vrstev Dense-Dense-Permute, která je zapojena pětkrát za sebou. U posledního bloku se místo Permute použil Flatten a výsledek se pustil přes další tři vrstvy, včetně klasifikační (2).

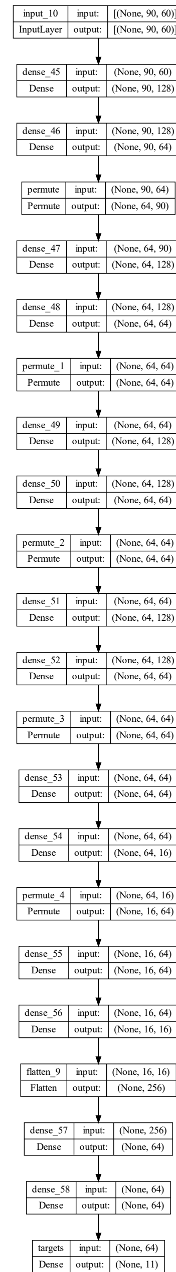
Výhodou obou architektur je vysoká rychlost inference a tréninku. Na druhou stranu hustě propojené vrstvy neobsahují časový ani lokální bias, a proto se mohou rychleji dostat do stavu přetrénování.

3.3.2 ResNet8 a dsResNet9

Tato architektura založena na konvolučních vrstvách se osvědčila v oblasti zpracování obrazů [22], kde výrazně posunula celý obor. Hlavním přínosem byl Skip Connection, který umožňoval informaci průchod do následující vrstvy beze změny (jednotková paralelní větev). Výsledkem byla lepší propagace gradientu, což umožnilo trénování výrazně hlubších sítí. Tato architektura je vhodná i pro zpracování zvuku, např. architektura ResNet15 za použití Triplet lossu dosáhla SOTA výsledků na Tensorflow datasetu [57].



Obrázek 1: Architektura Simple dense



Obrázek 2: Architektura Permute dense

Dál se používaly normalizační vrstvy, jejichž úkolem je zrychlení konvergence modelu a případné zvýšení přesnosti. Normalizace při tréningu slouží jako další nelinearita a snižuje pravděpodobnost exploze nebo mizení gradientu udržením konstantního rozptylu výstupu. Hlavním stavebním prvkem jsou bloky, které v sobě zahrnují dvě konvoluční vrstvy s normalizací a jednotkovou vazbou. V rámci této práce se zvolila varianta ResNet8 (3), která obsahuje tři takové bloky.

Po dalším výzkumu se ukázalo, že ve světě počítačového vidění hybridní architektury na bázi střídání depthwise a pointwise konvolucí dosahují lepších výsledků při nižším počtu parametrů a rychlejším trénování [49]. Proto se navrhla síť s velmi podobnou architekturou jako ResNet8, ale tří ResNet bloky se zaměnily čtyřmi InvertedResidual bloky. Po nahrazení se, i přes větší hloubku, počet parametrů výrazně snížil. Proto pro vynahrazení se zvýšil počet kanálů ze 40 na 52 (4).

Obě architektury obsahují konvoluční vrstvu na vstupu, která slouží jako detektor příznaků. Následuje AveragePooling2D pro snížení prostorových dimenzí a zvýšení rychlosti tréninku a inference. Celkově se vstupní tenzor zredukoval 12 krát, což ale bylo pořád dostačující pro řešení úlohy.

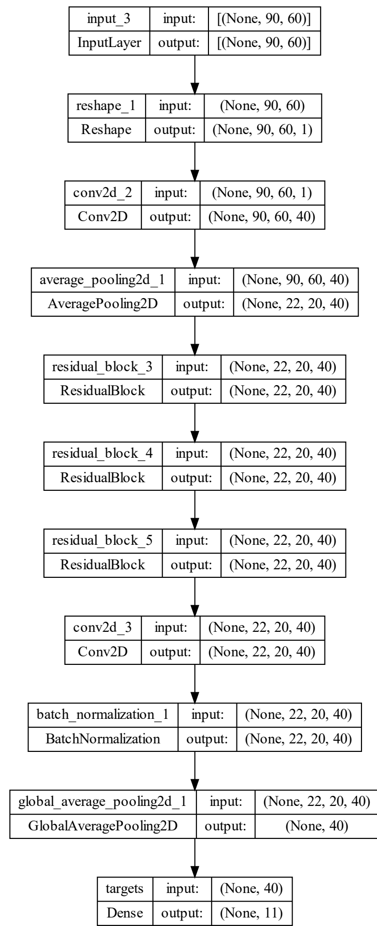
3.3.3 Transformer a GRU

V této sekci byly navrženy dvě architektury, které využívají časovou spojitost vzorků. Sítě na základě transformerů tento bias ale nemají (nejsou citlivé k pořadí tokenů), a proto je třeba její vstup obohatit o pozice jednotlivých tokenů v posloupnosti. Pro tento účel byla vytvořena vrstva Positional Embeddings, která je postavena na standardní Embedding vrstvě. Počet neuronů v hustě propojené vrstvě se vždy rovnal dvojnásobku latentní dimenze a parametr *headcount* byl nastaven na čtyři. Architektura je vidět na obrázku (5).
Druhou síť tvoří hlavně rekurentní vrstva Gated Recurrent Unit (6). Proto jde o jedinou z navržených architektur, která podporuje statefull detekci a je vhodná pro proudové rozpoznávání na zařízeních se slabším výkonem.

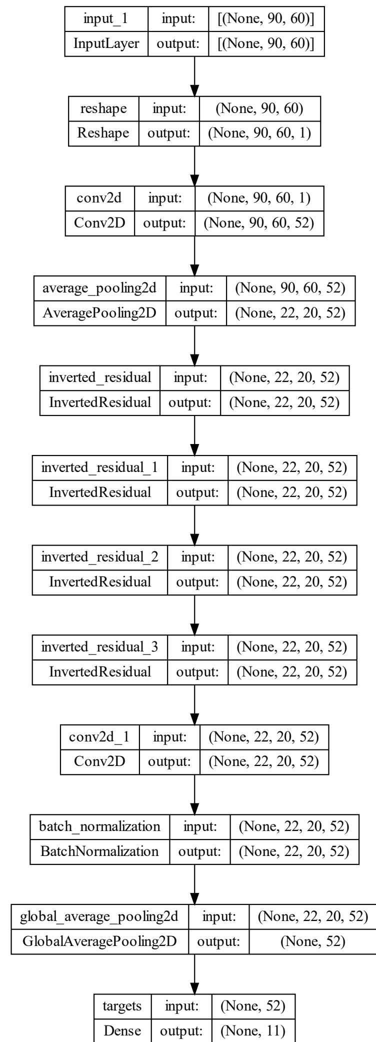
3.4 Učení částečně s učitelem

Učení částečně s učitelem (self-supervised learning) řeší problém velkého množství potřebných anotovaných prvků pomocí neanotovaných dat, které se dají získat mnohem snadněji. Dvěma základními pojmy jsou před-trénování (pretraining) a dotrénování.

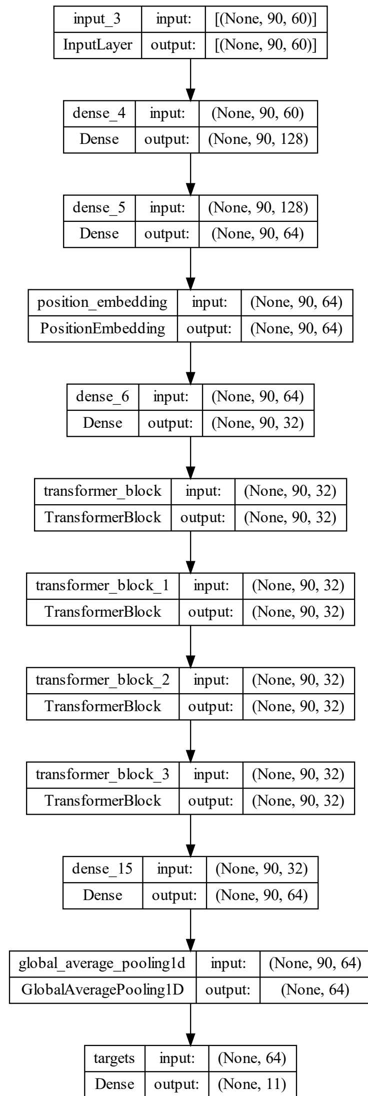
- **Pretraining** je fáze trénování pomocí speciálních ztrátových funkcí (např. InfoNCE [42]), kdy cílem je porozumění vstupních dat.
- **Dotrénování** (finetuning) využívá již natrénovaný model a pouze provádí poměrně krátkou optimalizaci na cílových datech.



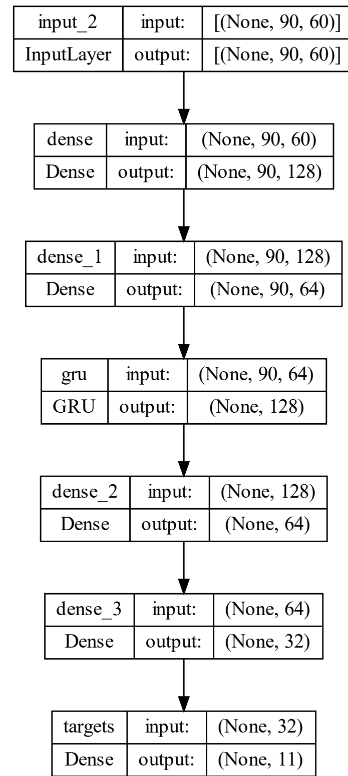
Obrázek 3: Architektura Resnet8



Obrázek 4: Architektura dsResnet9



Obrázek 5: Architektura Transformer



Obrázek 6: Architektura GRU

Dnes existují dva různé směry - modelování s maskou a kontrastní metody. V následujících sekcích se vyzkouší zejména modelování s maskou.

3.4.1 Autoencoder

Tato architektura má za úkol rekonstrukci svého vstupu na výstupu [23]. Toto vyžaduje průchod informace sítí s co nejmenšími ztrátami, a jelikož šířka sítě je často mnohokrát menší než dimenze vstupu, síť je nucena filtrovat nepotřebné detaily a pouze ponechávat nejdůležitější příznaky.

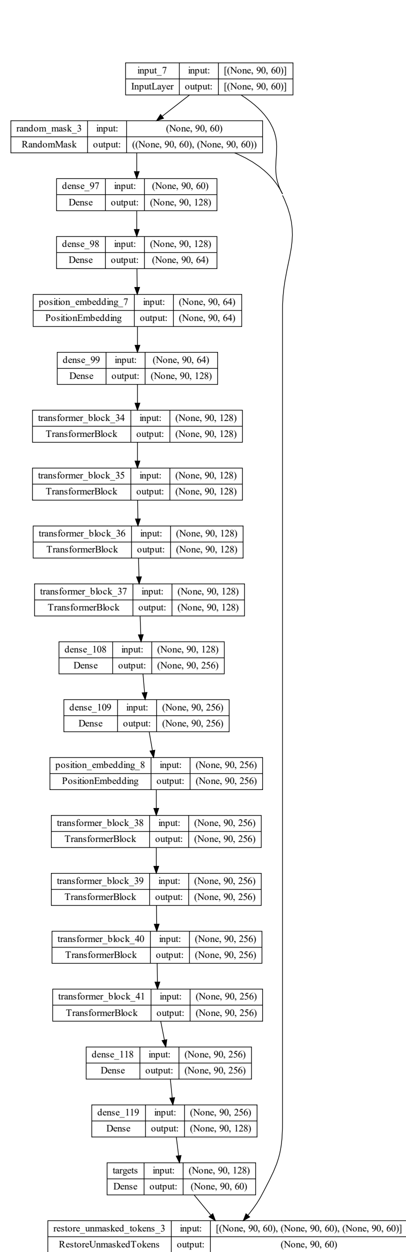
Autoencoder se skládá ze dvou části - enkodéru, který převádí vstup na nizkodimenzionální embedding vektor, a dekodéru, který se z tohoto vektoru snaží co nejpřesněji rekonstruovat vstup. Autoencoder se trénuje jako celek, kde se stejný prvek podává zároveň na vstup i na výstup. Po natrénovaní se enkodér a dekodér od sebe oddělí a využívají se pro downstream úlohy:

- Embedding vektor enkodéru se často používá pro trénování klasifikátoru (linear probing).
- Dekodér může plnit generativní funkci, kde z náhodného embedding vektoru se generuje zcela nový prvek z distribuce vstupních dat.

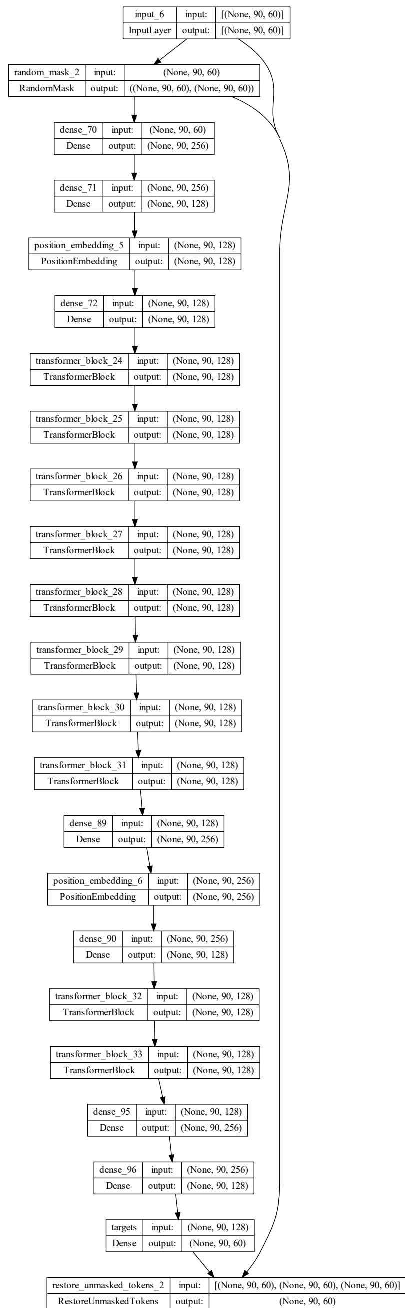
Jako efektivní způsob zlepšení kvality embedding prostoru v oblasti počítačového vidění se považuje maskování [21], kdy se na vstup podává malá část obrázku a úlohou sítě je rekonstruovat zamaskované oblasti. Podobný postup se použil i v oblasti rozpoznávání řeči [26]. Jako maskovací strategie se zvolilo maskovaní časových vzorků s pravděpodobností 75%.

Byly navrženy dva maskované autoencodery s různým počtem parametrů. Architektura MAE Small (7) byla navržena tak, aby samotný enkodér měl podobný počet parametrů, jako u sítí z předchozí sekce. MAE large se liší čtyřikrát vyšší embedding dimenzí a dvakrát větším počtem vrstev (8). Další odlišností byla maskovací vrstva na začátku sítě. Dekodér nebude použit při klasifikaci, proto nebyl omezen v počtu parametrů, kterých nakonec obsahuje asi čtyřikrát víc, než enkodér. Specifikou původního výzkumu byla rekonstrukce pouze zamaskovaných částí obrázků, přitom že viditelná část neovlivňovala loss. V Kerasu se toto realizovalo pomocí vlastní vrstvy RestoreUnmaskedTokens (na schématu jako poslední).

Dále se z důvodu rychlosti tréningu do enkodéru posílaly pouze viditelné vzorky. V rámci této práce autoencoder zpracovával i maskované vzorky (které obsahovaly pouze informaci o pozici v posloupnosti), zejména z důvodu snazší implementace.



Obrázek 7: Architektura MAE Small



Obrázek 8: Architektura MAE Large

3.5 Dataset

Kvůli vysoké flexibilitě, neuronové sítě vyžadují větší množství trénovacích dat než klasické metody strojového učení, jako např. Random Forest. Aby se síť správně natrénovala (nedocházelo k přetrénování) a měla vyhovující generalizační schopnosti, je potřeba velkého množství dat. Správná volba augmentačních postupů dokáže toto množství řádově snížit, ale i přesto, potřebný počet datových prvků se pro některé úlohy pohybuje v řádu tisíc nebo desítek tisíc pro každou třídu.

3.5.1 Použité datasety

V rámci této práce se použili dva datasety - Tensorflow[60] a LibriSpeech Light [30]. Tensorflow dataset se přímo týká úlohy detekování klíčových slov v anglickém jazyce a obsahuje kolem 65 tisíc nahrávek o délce asi 1s. Audio je uloženo ve formátu wav. Nahrávky jsou dále roztříděny do složek, které reprezentují příslušnou třídu. Navíc názvy souborů obsahují data o mluvčím ve formě ID (krátký hash) a číslo pokusu záznamu (každý mluvčí namluvil slova několikrát). Dataset obsahuje 30 klíčových slov, kde 10 z nich tvoří třídy, do kterých se klasifikuje (‘yes‘, ‘no‘, ‘up‘, ‘down‘, ‘left‘, ‘right‘, ‘on‘, ‘off‘, ‘stop‘ a ‘go‘), a zbytek včetně šumu prostředí se používá pro reprezentaci třídy Unknown. Z hlediska rozložení datasetu, 20 slov mají srovnatelný počet nahrávek, kdyžto zbylých 10 tříd jsou méně zastoupené. Nahrávek se šumem prostředí je pouze 6, ale mají délku kolem jedné minuty.

Načítání a operace s datasetem se realizovaly pomocí modulu tf.data, který řídí zejména on-demand načítání jednotlivých souborů nahrávek, paralelní zpracování dat ve více vláknech, rozdělení dat do mini-batchů a prefetch na pozadí. Pro trénování v prostředí Google Colab [7] bylo potřeba co nejvíc minimalizovat častý náhodný přístup k souborům (kvůli limitaci Goodle Drive) a snížit využití CPU, protože se z něj často stává úzké hrdlo. Původně se na CPU počítaly časově-frekvenční spektrogramy a převod na melovské koeficienty. Jako náhradní řešení se zvolilo předzpracování všech nahrávek a uložení do jednoho souboru.

Navíc z úvahy, že k rozpoznání slova ze spektrogramu není potřeba velké přesnosti jednotlivých amplitud, převedlo se jejich rozmezí hodnot na 0 až 255 a po zaokrouhlení se změnil datový typ z float32 na uint8 (provedla se kvantizace v hodnotě). Tato změna zmenšila velikost výsledného souboru čtyřikrát. Největší výhodou je ale reprezentace celého datasetu pomocí jednoho souboru, který se bez problému vejde do RAM paměti a nemá potíže s práci v prostředí Google Colab. Na straně trénovacího skriptu neuronová síť dostává tenzor typu uint8 jako vstup a pak ho pomocí metody tf.cast konvertuje na float32, což eliminuje povinnost převádět celý dataset na float32 a tím šetří RAM paměť. Samotné augmentace nahrávek z důvodu rychlosti probíhají rovnou na GPU (3.9).

Nahrávky šumu a zvuků prostředí mají délku zhruba jedna minuta, proto je

potřeba je rozsekat na části. Opět se vytvořil dynamický pipeline, který náhodně volil část nahrávky o délce 1s. Dále se prováděla standardní procedura výpočtu melovských koeficientů a převod na decibely. Po těchto manipulacích výstupem by ale bylo 6 nahrávek o délce 1s, což není dostačující pro vhodnou reprezentaci šumu a zvuků prostředí. Proto se použila metoda Repeat a dataset se šumem se nakopíroval tolikrát, aby výsledné zastoupení nahrávek šumu bylo stejné, jako u jiných tříd, tedy zhruba 1800 vzorků. Nakonec pomocí metody zip se původní dataset a dataset se šumem a zvuky prostředí spojily dohromady, přitom že nahrávky šumu se zařadily do třídy Unknown. Po všech transformacích datasetu počet vzorků v kategorii Unknown značně převažuje, proto pro úspěšné trénování je potřeba si s tím počítat (více v sekci 3.13).

LibriSpeech Light sloužil pro před-trénování modelů. Jedná se o dataset pro rozpoznávání řeči, kde každá nahrávka má délku v rozmezí kolem 5s až 20s a odpovídá jedné celé větě v anglickém jazyce. Ve stejných složkách s nahrávkami se nachází textový soubor s anotacemi. Samotné soubory nahrávek mají datový typ flac, který umožňuje bezztrátovou kompresi audio dat. Nejlehčí verze datasetu celkově obsahuje 100 hodin řeči (28 tisíc souborů), ale kvůli použití komprese nahrávky zabírají pouze dvakrát víc místa než Tensorflow dataset. Pro účely před-trénování se použily samotné audio soubory bez anotací. Načtení probíhalo pomocí prohledávání adresářů a použití jednoduchého regulárního výrazu, který hledal soubory s koncovkou flac. Pro generaci nahrávek o délce 1s se použil stejný postup jako u souborů se zvuky prostředí v prvním datasetu (náhodný výsek). Nakonec se vygenerovalo 530 tisíc vzorků, které byly následně převedeny na uint8 a exportovány do jednoho souboru. Rozměr tohoto souboru je 2.6GB, tedy je 10x větší než rozměr souboru pro Tensorflow dataset.

3.6 Předzpracovaní nahrávek

3.6.1 Tensorflow dataset

Hned po načtení je nahrávka reprezentována jako průběh amplitud zvukové vlny v čase a metadata. Prvním krokem předzpracování je již načtenou do paměti nahrávku převzorkovat na předem zvolenou frekvenci, a případně převést stereo signál na mono. Jako cílová vzorkovací frekvence se zvolila 16 kHz. Z Nyquist–Shannonovy věty o vzorkování, maximální frekvence, kterou může takový signál popsat, je polovina vzorkovací frekvence - 8 kHz, což je postačující pro reprezentaci lidského hlasu.

Po převzorkování každá nahrávka je uložena v paměti jako jedno-dimenzionální pole o velikosti $samplingrate \cdot time$. Data o amplitudě zvukové vlny jsou popsány jedním int16 číslem se znamínkem pro každý bod. Z důvodů kompatibility se provádí převod amplitudy na float32 a normalizace na interval (-1, 1) - vydělením maximální možnou hodnotou čísla typu int16.

Pro trénování neuronových sítí vstupní a výstupní data by také měla být ve formě tenzorů (v novějších verzích frameworku se toto omezení dá obejít pomocí

třídy RaggedTensor), z čehož plyne požadavek, aby každý prvek v datasetu měl stejný tvar (dimenze). Délka samotných nahrávek ale není stejná, pohybuje se v okolí jedné vteřiny. Proto je nezbytné nahrávky zarovnávat. Při normalizaci délky mohou tedy nastat dva případy, kdy je audio nahrávka buď delší, nebo kratší než požadovaná délka.

Požadovaná délka byla nastavena na 16384 bodů, protože je toto číslo blízké jedné vteřině audio při zvolené vzorkovací frekvenci a zároveň je to mocnina dvojky. Pokud je nahrávka delší, pak se na náhodné pozici vystřihne kus nahrávky o požadované délce. V případě kratší nahrávky se časová posloupnost musí doplnit, což se realizovalo přidáním ticha na začátku posloupnosti. Alternativně by se také dál přidávat bílý šum s nízkou amplitudou. Jelikož se bude pracovat s řečí, je vhodná aplikace metody mel filterbank. Použití pásmových filtrů umožňuje výrazně snížit počet prostorových dimenzí vstupu bez značné ztráty potřebné informace, což se projeví v kratším čase zpracování sítě.

Dál se normalizovaný vstup v délce a hodnotě zvukový signál převádí do časově-frekvenční oblastí pomocí algoritmu krátkodobé fourierovy transformace (STFT). Takový spektrogram lépe popisuje průběhy frekvencí v čase a proto se tradičně používá v úlohách rozpoznávání řeči. Na spektrogram se také lze dívat ne jako na časovou posloupnost, ale jako na obrázek, což umožňuje využití řady nástrojů z oblasti počítačového vidění. Audio posloupnost se rozdělí na překrývající se kusy o délce 512 bodů tak, aby krok mezi sousedními pozicemi okénka byl 184 body. Pro zjemnění detailů spektrogramu se na každé okénko aplikuje okénková funkce, jejíž úkolem je vyhladit okraje okénka. Důvod aplikace plyne z vlastností fourierovy transformace, přesněji z požadavku periodicity vstupního signálu. Existuje víc okénkových funkcí, v této práci se ale aplikuje Hamming window. Výsledný signál, který už je periodický, prochází diskrétní fourierovou transformaci. Algoritmus se dá zrychlit použitím rychlé verze fourierovy transformace (FFT), která je ale efektivní pouze v případech, kdy je velikost okénka mocninou dvojky.

Velikost okénka a jeho krok byly zvoleny experimentálně tak, aby výsledný spektrogram dostatečně reprezentoval řeč a zároveň neměl zbytečně velké rozměry, které by přímo ovlivňovaly rychlost inference sítě. Rozměr okénka ovlivňuje hladkost spektrogramu a omezuje maximální počet frekvenčních složek. Menší skok pak zvětšuje počet časových vzorků. Z STFT dostaneme dvourozměrný pole komplexních čísel, kde osa X je časová, a Y frekvenční. Pro účely rozpoznávání řeči je postačující pouze informace o amplitudě. Hodnota amplitudy se z komplexního čísla spočte jako jeho absolutní hodnota.

Poté se po umocnění amplitudy na druhou aplikuje metoda mel filterbank, která agreguje jednotlivé frekvence do menšího počtu pásem (pomocí pásmových filtrů) a aplikuje melovskou škálu. Celou transformaci lze provést pomocí maticového násobeni spektrogramu a matice filtrů, která se dá vygenerovat pomocí speciální metody frameworku Tensorflow. Hodnoty amplitudy jsou ale

pořád v absolutních jednotkách, proto se tato amplituda převede na decibely v rozmezí -80 až 0 dB a výsledek se lineárně zobrazí na interval $(0, 1)$, který je vhodný pro trénování neuronových sítí. S danými parametry docílíme tenzoru o rozměrech (90, 60), kde velikost první dimenze odpovídá počtu časových vzorků.

Cílový (target) vektor obsahuje informaci o příslušné třídě, která je převedena na tenzor pomocí metody one-hot encoding. Takový vektor má stejnou délku jako je počet tříd, do kterých se klasifikuje. Každé třídě se přiřadí pozice v tenzoru a na příslušné pozici vektor bude obsahovat jedničku, všechny zbylé dimenze se nastaví na nulu.

3.6.2 LibriSpeech Light

Prvky z tohoto datasetu jsou určený pro před-trénování. Jelikož se před-trénované modely budou dotrénovávat na Tensorflow datasetu, je rozumné zachovat typ a tvar vstupních dat. Proto se nahrávky před-zpracovávají stejným způsobem jako u Tensorflow datasetu. Metody z oblasti self-supervised učení pouze vyžadují vstupní data, na rozdíl od učení s učitelem, které také vyžaduje anotace. Potom jedinou odlišnosti ve srovnání s Tensorflow datasetem je absence cílového vektoru, protože se nejedná o úlohu klasifikace.

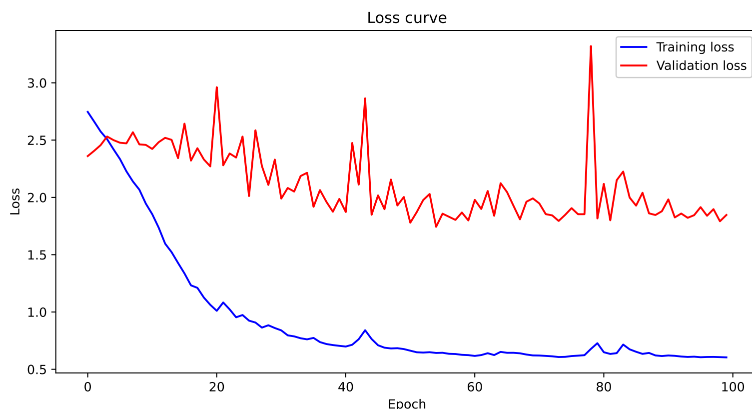
3.7 Přetrénování a podtrénování

Důležitou podmínkou modelu je zobecnění (generalizace) na neviděné prvky, ze které plyne požadavek na hledání reálných závislostí mezi příznaky. Při příliš velkém počtu volných parametrů nebo epoch se po tréninku síť ale může dostat do stavu přetrénování (overfittingu). To znamená, že model dokázal zapamatovat část dat a dává velkou váhu malým a nepodstatným detailům, které ale dobře popisují trénovací data. Taková síť může mít vynikající výsledky při trénování, ale bude zcela nepoužitelná na reálných datech, proto se tomuto stavu chceme co nejvíc vyvarovat.

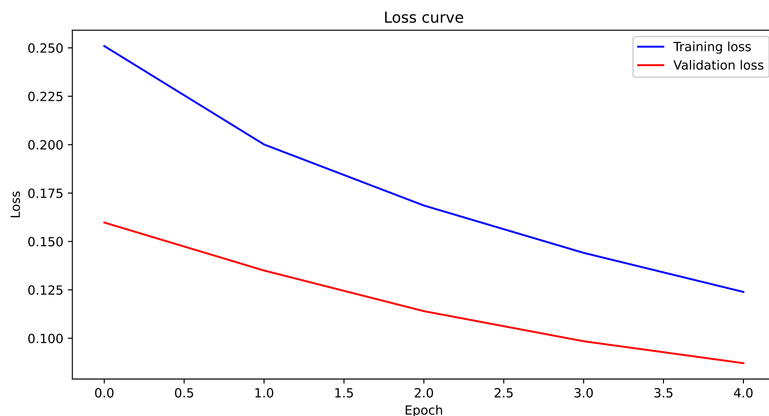
Odhalení přetrénování je hlavním důvodem zavedení validační sady. Poznat, kdy se model začne overfitovat lze z průběhu ztrátové funkce pro trénovací a validační sadu. Jedná se o okamžik v čase, kdy vzdálenost mezi trénovacím a validačním lossem se začne zvětšovat, za podmínky, že trénovací loss pořád klesá. Stejným způsobem se overfitting také dá odhalit i z průběhů přesností. Velký rozdíl mezi lossy obecně poukazuje na nízkou kvalitu modelu nebo datasetu. Existuje velké množství způsobů jak zacházet s přetrénováním, některé z nich jsou popsány v sekcích augmentace (3.9) a regularizace (3.10).

Naopak podtrénování (underfitting) popisuje stav, kdy trénování sítě neproběhlo optimální počet iterací a proto se parametry sítě stále nachází ve stavu konvergence. Nejjednodušším způsobem nápravy je zvýšení počtu trénovacích epoch.

Na obrázku (9) je vidět průběh ztrátových funkcí modelu, který se dostal do stavu přetrénování. Z grafu lze vidět, že trénovací loss se ustálí výrazně níž než validační. Příklad podtrénování je zobrazen na obrázku (10). Tento model se trénoval pouze po dobu čtyř epoch. Zajímavostí je nižší hodnota validačního lossu v počátečních epochách trénování. Tento jev se ale běžně objevuje v praxi.



Obrázek 9: Příklad výrazného přetrénování modelu.



Obrázek 10: Příklad podtrénování modelu.

3.8 Validace

Validace je nezbytnou součástí modelování, která dokáže odhalit případné chyby implementace a zkrátit dobu vývojového cyklu produktu. Ve světě neuronových

sítí se validace provádí pomocí správně zvolených validačních dat. Rozšířenou praxi je provádět validaci na konci každé epochy a porovnávat přesnost modelu na trénovací a validační sadě. Pro lepší názornost se průběhy vykreslují do grafu. Trénování neuronových sítí ale často zahrnuje ruční optimalizaci hyperparametrů, jako jsou např. koeficient učení, rozměr mini-batche a počet epoch. Tyto manipulace mohou způsobit to, že nejlepší dosažený výsledek opět nemusí odpovídat realitě. Proto se obzvlášť u neuronových sítí zavádí další podmnožina dat, která je určena pouze pro výslednou validaci již zvoleného nejpovedenějšího modelu.

Validační a testovací sada byla navržena tak, aby co nejvíce napodobovala reálné podmínky, tedy obsahuje pouze ty mluvčí, kteří nebyly v trénovací sadě. Při evaluaci se nahrávky neaugmentují, protože by to mohlo ovlivnit distribuci reálných dat a výsledek by nemusel být validní. Jako počáteční bod pro generaci těchto sad se použily již obsažené v datasetu testovací a validační seznamy.

Použitý dataset ale pouze slouží ke klasifikaci nahrávek a proto neobsahuje žádnou kategorii pro ticho, v trénovací ani ve validační a testovací sadě. Výjimkou jsou šest delších nahrávek zvuků okolního prostředí a šumu. Při tak malém počtu vzorků ale nejde jednoduše sestavit potřebné sady dat. Proto se použila stejná metoda jako u LibriSpeech datasetu, kde se z jedné nahrávky vygenerovalo více vzorků o délce 1s náhodným vystřihnutím. V tomto případě ale existuje možnost úniku dat (data leakage), kdy se v testovací sadě objeví identický soubor jako v trénovacím. V našem případě je tato pravděpodobnost velmi malá, protože náhodná selekce segmentu probíhá v prostoru zvukové vlny.

3.9 Augmentace

Jelikož se jako řešení používají neuronové sítě, z důvodu vysokého stupně volnosti (nastavitelných parametrů) optimalizace sítě vyžaduje velkého množství trénovacích dat aby nedošlo k přetrénování. Pro snížení potřebného množství a pro zlepšení generalizace systému je nutná augmentace dat a promyšlená architektura, která využívá potřebné biasy.

Při zpracování zvuku se často pracuje s čáso-frekvenčními spektrogramy, na které se dá pohlížet jako na obrázky s jedním barevným kanálem, stejně jako na černobílý obrázek. Proto můžeme čerpat inspiraci z metod augmentací ve světě strojového vidění.

V oblasti rozpoznávání řeči se často používá augmentace amplitudy, přidání šumu ke vstupu, roztažení a posunutí podle časové a frekvenční osy.

3.9.1 Augmentace amplitudy

Amplituda časofrekvenčních spektrogramů obsahuje informace o hlasitostí jak jednotlivých frekvencí, tak i celé nahrávky. Tato augmentace se zavedla za

úvahy, že se změnou hlasitosti sémantický obsah nahrávky zůstává stejný. Softwarová realizace se skládá ze dvou části - absolutní a relativní:

$$y = x \cdot A_{rel} + A_{abs}$$

Intenzita augmentace se volí náhodně (rovnoměrné rozdělení) v rozmezí hyperparametrů. V rámci této práce se použily hodnoty $A_{rel} = \pm 5\%$ a $A_{abs} = \pm 0.05$. Alternativou ze světa počítačového vidění by byla augmentace jasu a gammy.

3.9.2 Augmentace šumem

Jednou z univerzálnějších metod zvýšení robustností sítě je přidávání šumu s malou amplitudou ke vstupu do neuronové sítě. Jednotlivé složky šumu ovlivňují malé detaily na spektrogramu a proto snižují pravděpodobnost přetrénování sítě. Realizace šumu se čerpají z normálního rozdělení s nulovou střední hodnotou a konstantní směrodatnou odchylkou, jejichž hodnota se volí náhodně pro každý prvek datasetu, přitom maximální odchylka byla nastavena na $stddev = 0.05$. Výstupem tohoto algoritmu muže být jak zašuměný, tak i čistý spektrogram, v závislosti na vygenerované amplitudě. Toto umožňuje neomezovat neuronovou síť pouze na zašuměné nahrávky. Po změně amplitudy a přidání šumu se hodnoty některých dimenzí může uniknout z intervalu $<0, 1>$. Proto se jako poslední prvek augmentačního pipelinu použila operace Clamp s horní a dolní mezi odpovídající intervalu:

$$Clamp(x) = min(max(0, x), 1) = \begin{cases} 0, & \text{v případě x } \leq 0 \\ 1, & \text{v případě x } \geq 1 \\ \text{x}, & \text{jinak.} \end{cases}$$

3.9.3 Augmentace času a frekvence

Jedná se o posunutí a roztažení spektrogramu v horizontálním a vertikálním směru. Projevem těchto augmentací je zrychlené nebo zpomalené tempo řeči a změna hlasivkového tónu. Při pretrainingu se posunutí v čase aplikovalo hned při generaci výstřižků, jejichž pozice se volila náhodně. Augmentace posunutím obvykle není efektivní v případě konvolučních sítí, které již jsou prostorově invariantní, ale je žádaná při použití transformerů a hustě propojených sítí, které tento bias nemají. Augmentace frekvence patří k těm základním v oblasti rozpoznávání řeči, avšak tato práce se omezila na dvě předchozí augmentace z důvodu jednodušší implementace. Opět můžeme provést paralelu s počítačovým viděním, kde se tyto augmentace podobají klasickým zoom a crop.

3.10 Regularizace

Regularizací se myslí položení umělých omezení na trénování neuronové sítě tak, aby se zvýšila přesnost na validační a testovací sadě, často na úkor chyby na trénovací sadě. Tedy sice síť hůř popisuje trénovací data, ale přitom je

robustnější k patternům, které v trénovací sadě nejsou. Regularizačních metod je velké množství, některé hlavní jsou popsány níž:

- Dropout vrstvy - náhodně vynulují výstupy části neuronů, což zvyšuje tolerantnost sítě vůči poruchám a snižuje přetrénování [51].
- L1 a L2 regularizace (weight decay) přidávají další složku ke ztrátové funkci, která penalizuje vysoké hodnoty vah.
- Augmentace se také dají považovat za regularizační metodu, protože jejich hlavním cílem je snížení přetrénování.
- Weight sharing - méně volných parametrů obvykle znamená menší šanci přetrénování.
- Early Stopping - trénink se ukončí, když validační loss výrazně neklesá po daný počet epoch.
- Sharpness Aware Minimization (SAM) - modifikace optimizátorů, která se snaží najít hladká minima, která jsou spojená s lepší generalizací [19].
- Label Smoothing - úprava cílového vektoru u klasifikačních úloh [54].

V rámci této práce se kromě augmentací použily metody SAM a Label Smoothing. Nevýhodou SAM je potřeba přepočítávat celou síť dvakrát pro odhad hladkosti minim. Label Smoothing ale nemá žádné dopady na rychlost a dá se realizovat jako modifikace ztrátové funkce.

3.11 Volba optimizátoru

Základní optimalizační metodou pro trénink neuronových sítí je gradient descent - iterativní pohyb hodnot parametrů proti směru jejich gradientu dokud nedokonvergují k lokálnímu nebo globálnímu minimu chyby. Optimizátor je algoritmus, který se snaží nastavit hodnoty jednotlivých vah tak, aby co nejvíce vyhovovaly trénovacím datům, za co nejmenší počet iterací. Existuje velké množství takových algoritmů [47], kde každý má vlastní klady a zápory. Při volbě optimizátoru se brala v úvahu hlavně jeho stabilita a rychlost konvergence. Další důležitou vlastností byla paměťová složitost vzhledem k počtu parametrů sítě.

3.11.1 Stochastic gradient descent

Jednou ze základních metod je stochastic gradient descent (SGD). Tento algoritmus je jednoduchý na implementaci, ale přesto dokáže dosáhnout globálního minima [65].

Kroky v prostoru vah závisí pouze na hodnotě gradientu a velikosti LR, a neobsahují žádnou dynamiku, proto SGD často vyžaduje větší počet trénovacích epoch. Na druhou stranu výhodou jsou malé nároky na paměť, protože není

třeba uchovavat další parametry pro zavedení dynamického chování.

Za hlavní faktor, který se pomáhá dostat z lokálních minim se považuje stochasticita, která se plyne z náhodné volby vzorků v mini-batchi (obsah mini-batche se při každé iteraci náhodné promíchá). Využití mini-batchu umožňuje vyšší míru paralelizace tréninku na GPU a zároveň stabilizuje odhad směru gradientu. Při příliš velké velikosti mini-batche se stochastická složka výrazně potlačuje, proto se batch size považuje za hyperparametr a je potřeba hledat kompromis.

Úspěšným pokusem o zavedení dynamiky je momentum, který napodobuje chování integrátoru v prostoru parametrů. Tedy velikost kroku jednotlivých vah závisí na předchozích krocích. Toto pomáhá překonávat lokální minima a zvyšuje rychlost konvergence. Nevýhodou by byla větší spotřeba pamětí.

3.11.2 Adam

Jedním z nejpoužívanějších optimizátorů je Adam [33], který patří do skupiny optimizátorů s adaptivním LR. Využívá hodnoty dynamiku velikosti gradientu váhy pro individuální naplánovaní kroků. Tedy každý parametr nakonec má svůj vlastní LR. V poslední době se ale rozšiřuje použití verze AdamW, která disponuje vylepšenou implementaci regularizace vah [38]. Žádná z navržených architektur weight decay nepoužívá, proto se jako optimizátor zvolil LAMB (3.11.3).

3.11.3 LARS a LAMB

Jedná se o modifikované verze SGD a Adam algoritmů, které je vylepšují o logické rozdělení sítě do vrstev, kde se na gradient jedné vrstvy hledí jako na jeden celek. Hlavním vylepšením je normalizace gradientu vrstvy za použití váhové matice, které pomáhá dosáhnout podobného efektu, jako Weight Clipping. Takové škálování kroku vah zvyšuje stabilitu tréninku, zejména v počátečních epochách. V praxi toto umožňuje trénování s velmi velkými rozměry mini-batche [63].

3.12 Ztrátová funkce

Jedna z nejpoužívanějších ztrátových funkcí pro regresi je středně-kvadratická odchylka (MSE), která penalizuje větší odchylky víc, než malé. Proto líp zachycuje drobné detaily, ale je také náchylná na outliery (odlehlé vzorky). V této práci se tato ztrátová funkce použila při před-trénování autoencoderů.

Na klasifikaci se lze dívat jako na regresi hodnoty podmíněné pravděpodobnosti $P(c_i|X)$. To znamená, že ztrátové funkce pro regresi se dají použit i pro klasifikaci. Lepších výsledků ale obecně dosahuje Cross Entropy Loss, který penalizuje nejistotu predikce, a proto tlačí výstup klasifikátoru buď na nulu nebo k jedné:

$$-\sum_{c=1}^{M} y_{x,c} \log(p_{x,c})$$

Existují i složitější ztrátové funkce, jako např. kontrastní a triplet loss. Kontrastní ztrátová funkce [42] se snaží tlačit embedding vektory od sebe pro různé třídy a k sobě pro stejné třídy. Triplet loss funguje na podobném principu, ale navíc zavádí opěrný prvek [14].

3.13 Trénink a optimalizace

Parametry trénování se dají popsat pomocí tabulky:

Učení s učitelem					
Model	Batch size	Learning rate	Epochs	Optimizer	Dataset
Simple dense	1024	0.001	100	LAMB	Tensorflow
Permute dense	1024	0.001	100	LAMB	Tensorflow
ResNet8	1024	0.001	100	LAMB	Tensorflow
dsResNet9	1024	0.001	100	LAMB	Tensorflow
GRU	1024	0.001	100	LAMB	Tensorflow
Transformer	1024	0.001	100	LAMB	Tensorflow
MAE - Large	512	0.001	400	LAMB	Tensorflow
MAE - Small	1024	0.001	100	LAMB	Tensorflow
MAE - Large	512	0.001	200	LAMB	LibriSpeech
MAE - Small	1024	0.001	100	LAMB	LibriSpeech
Dotrénování	128	0.001	10-20	LAMB	Tensorflow

Tabulka 2: Parametry trénování pro jednotlivé modely - učení s učitelem.

Pro optimalizaci každého modelu se použil algoritmus LAMB, který umožnil použití většího rozměru mini-batche. Všechny modely s učitelem měly podobný počet parametrů a také řádově podobnou dobu trénování, proto se pro ně také zvolila stejná doba trénování - 100 epoch. Architektura Simple dense však byla nejrychlejší. Naopak nejpomalejší architekturou byla Depthwise convolution, jejíž trénink trval kolem 90 minut na GPU Nvidia T4 (v prostředí Google Colab). Všechny pretraining modely se trénovaly v prostředí Metacentra se 4xA100 GPU, kde na každém ze čtyr GPU probíhal trénink odlišné verze modelu. Větší model MAE ale vyžadoval snížení batch size na 512, aby se vešel do paměti GPU. Alternativně by se dal využit optimizátor s menšími nároky na paměť, jako např. SGD, ale taková změna by mohla výrazně ovlivnit výsledek. Také se pro korektní porovnání nepoužívala metoda dočasného ukončení tréningu při stoupání validačního lossu (Early Stopping).

Na všechny optimalizované vrstvy se naložila váhová restrikce, která omezovala maximální amplitudu každého parametru na hodnotu 20. Tento typ regulari-

zace snižuje pravděpodobnost divergence sítě, která může nastat na začátku trénování nebo při příliš vysoké hodnotě gradientu některých vah.

Jedním z požadavků byl malý počet parametrů sítě, který se omezil na zhruba sto tisíc. Proto se vyzkoušelo i trénování některých modelů na CPU (Intel i5 8250U). Simple dense model se natrénoval za 45 minut a optimalizace Permute dense modelu trvala hodinu a půl. Tedy menší modely je možné optimalizovat i na laptopu. U větších enkodérů by doba trénování na CPU byla nepraktická.

Jednou ze široce používaných metod při trénování je plánování průběhu koeficientu učení (learning rate schedule), jehož použití může vylepšit přesnost modelu a vyvarovat se manuálním změnám tohoto koeficientu. V rámci této práce tréning trval menší počet epoch a samotné sítě nebyly příliš hluboké, proto se použila konstantní hodnota learning rate.

U novějších GPU existuje možnost využití mixed precision trénování [40], které značně zrychluje dobu konvergence a snižuje paměťové nároky při zanedbatelném vlivu na přesnost. Tato práce ale pouze používá datový typ $float32$.

Jelikož distribuce datasetu není rovnoměrně rozmístěná (třída Unknown obsahuje 40% nahrávek), bylo použito vážení jednotlivých tříd ve ztrátové funkci. Bez vážení neuronová síť zdánlivé rychleji konvergovala, ale ve skutečnosti většinu nahrávek zařazovala do třídy Unknown a tím dosahovala přesnosti kolem 65% (záleží na zastoupení tříd).

Pro výpočet váhy každé třídy se použil vzorec:

$$w_i = N/(C * N_i),$$

kde N je počet prvků v datasetu, C je počet tříd a N_i je počet prvků třídy i.

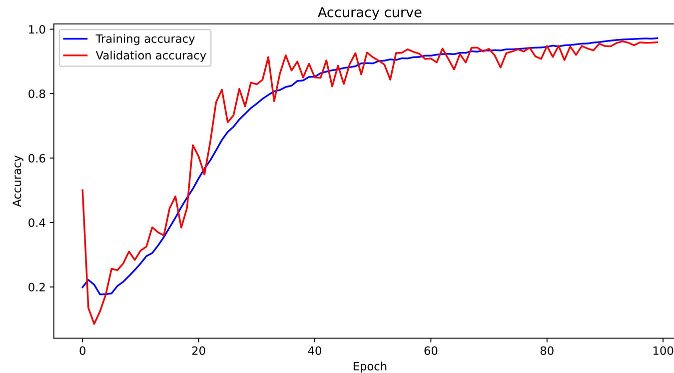
4 Prezentace a zhodnocení výsledků

Jako hlavní metrika pro evaluaci modelu se použila přesnost (accuracy). Výpočet pro jednu třídu je uveden na vzorci níž. Hodnota přesnosti pro klasifikaci do více tříd se spočte jako vážený aritmetický průměr jednotlivých přesností.

$$Acc_c = \frac{TP_c + TN_c}{N_c}$$

kde c označuje třídu, TP počet správně odhadnutých pozitivních prvků (true positives), TN počet správně odhadnutých negativních prvků (true negatives) a N je zastoupení třídy c.

Obvykle se nejedná o vystihující metriku v případě nerovnoměrně rozmístěného datasetu, ale použitím vah jednotlivých tříd se tento efekt dá vynahradit. Dále se pro nejlepší model vykreslila matici četností, která lépe popisuje výsledky klasifikace pro každou třídu.



Obrázek 11: Průběh přesností pro nejlepší model

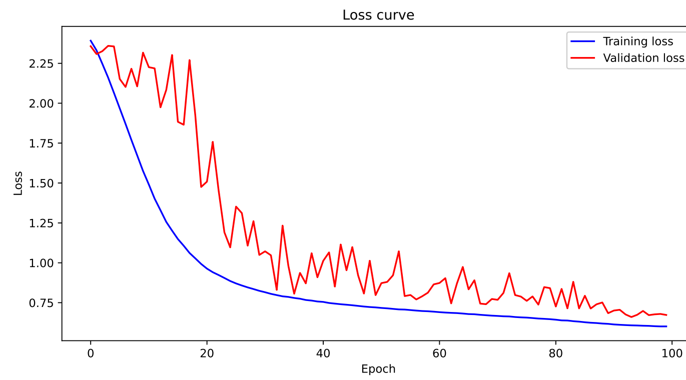
4.1 Výsledky učení s učitelem

Po natrénování všech modelů se provedl další krok validace, tentokrát i na testovací sadě a bez použití vah tříd, aby se data co nejvíc přiblížily realitě. Níž je vidět tabulka přesností modelů (3):

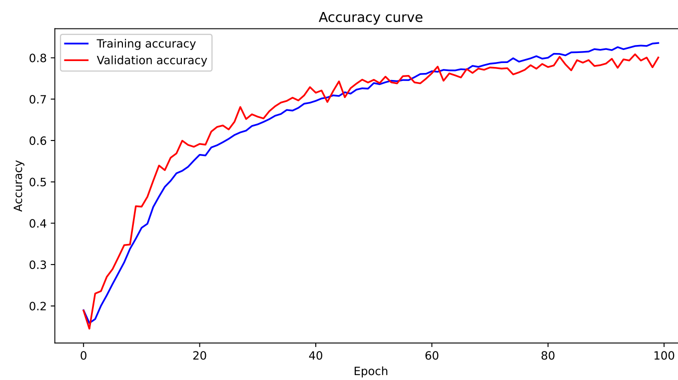
Učení s učitelem - přesnost testovací a validační sadě		
Model	Test sada	Přesnost - Validační sada
Simple dense	0.8010	0.8007
Permute dense	0.8905	0.8895
ResNet8	0.9593	0.9595
dsResNet9	0.9454	0.9455
GRU	0.8330	0.8330
Transformer	0.9231	0.9227

Tabulka 3: Validace modelů - učení s učitelem.

Z výsledků lze usoudit, že nejpřesnějším modelem je ResNet8, i když dsResNet9 má pouze o 1.5 % horší absolutní přesnost. Přesnost pro validační a testovací sadu je v mezích drobné odchylky stejná, tedy model dobře vyhovuje realitě. Pří trénování rekurentní sítě bylo poznamenáno zvláštní chování přesnosti, která na začátku trénování vyrostla na 40% a následně se snížila na 10-15%, kde se držela po dobu zhruba 10 epoch. Rekurentní síť také nedosáhla očekávaných výsledků a dosáhla podobné přesností jako Simple dense model. Níže se vykreslila matice četností pro jednotlivá slova v testovací sadě (4).



Obrázek 12: Průběh ztrátové funkce pro nejlepší model



Obrázek 13: Průběh přesností pro nejhorší model

Matice četnosti											
	Yes	No	Up	Down	Left	Right	On	Off	Stop	Go	UNK
Yes	252	0	0	0	3	0	1	2	0	0	2
No	1	237	0	4	1	0	0	0	1	8	18
Up	0	0	238	0	0	0	0	11	0	2	9
Down	0	1	0	254	0	0	0	0	0	1	8
Left	3	0	0	0	241	0	0	0	1	0	2
Right	0	0	1	0	1	248	0	1	0	0	5
On	0	0	2	0	1	0	239	8	1	0	6
Off	0	0	6	0	0	0	1	246	0	1	2
Stop	0	0	1	2	1	0	0	2	235	0	5
Go	0	0	1	2	0	0	0	6	0	231	20
UNK	7	2	6	18	9	26	19	18	5	22	4341

Tabulka 4: Matice četnosti klasifikace pro model ResNet8.

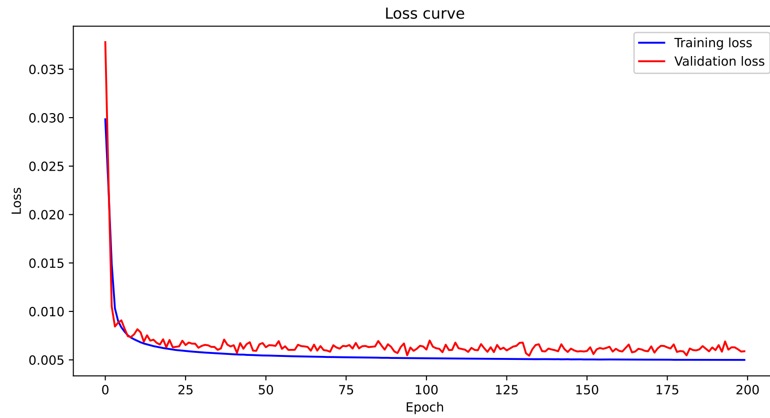
4.2 Výsledky učení částečně s učitelem

Natrénoval se malý a velký model na LibriSpeech a Tensorflow datasetu a následně se pro oba modely vytvořili jednoduché lineární klasifikátory. Při následném dotrénování se všechny vrstvy před-trénovaného modelu zmrazily. Tento postup se nazývá linear probing. Navíc za použití stejných před-trénovaných modelů se po odmrazení provedl finetuning poslední hustě propojené vrstvy, která se nachází před vrstvou GlobalAveragePooling. Jak je vidět z tabulky (5), dotrénování výrazně zvyšuje přesnost rozpoznávání bez dopadu na rychlost výpočtu sítě. Linear probing se trénoval po dobu 10 epoch, finetuning po dobu 20. Z průběhu ztrátové funkce se dá říct, že klasifikátory jsou ve stavu podtrénování a vyplatil by se větší počet epoch. Při provádění dotrénování se u před-trénovaných modelů snížil koeficient maskování na nulu.

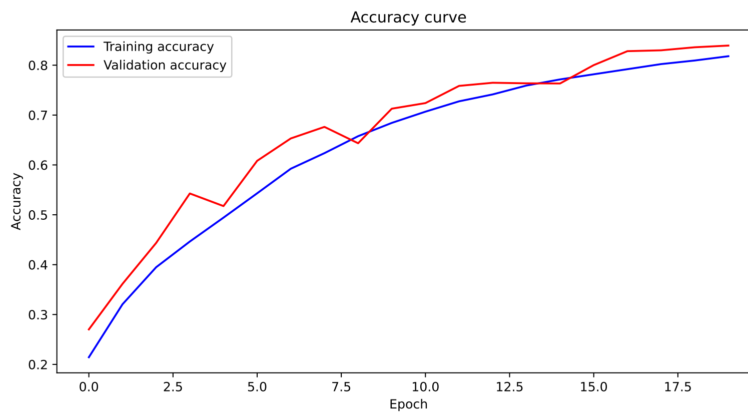
Učení bez učitele		
Model	Přesnost - probing	Přesnost - finetuning
MAE-libri-large	0.4462	0.8391
MAE-tensorflow-large	0.5310	0.8322

Tabulka 5: Validace modelů - učení bez učitele.

Výsledky na validační sadě opět byly v rámci odchylky stejné. Také se natrénovaly menší modely, ale kvůli nedostatečné kapacitě nebyly schopné dostatečně zachytit vztahy mezi příznaky, a proto ani po dotrénování přesnost klasifikátorů nepřesahovala 30%. Z tabulky je také vidět, že po dotrénování se přesnost klasifikátorů pro oba datasety vyrovnala a je poměrně vysoká. Znamená to, že před-trénování na libriSpeech datasetu se dobře přenáší na úlohu rozpoznávání klíčových slov. Výsledné klasifikátory jsou očekávaně horší, než trénované s učitelem. Navíc velikost MAE modelu není vhodná pro nasazení na slabší hardware, ale je možné se pokusit provést destilaci znalostí [24].



Obrázek 14: Průběh ztrátové funkce pro před-trénování na LibriSpeech datasetu



Obrázek 15: Průběh přesností pro finetuning

5 Závěr

V průběhu práce se úspěšné natrénovaly modely a prošly validací. Pro nasazení na reálný hardware je vhodné provést kvantizaci vah sítě, která dokáže zmenšit velikost modelu a jeho paměťové nároky. Z hlediska výkonu by se preferovaly stateful modely (GRU) nebo modely na bázi hustě propojených vrstev. V případě požadavku na vyšší přesnost by se vyplatilo použití výkonově náročnější ResNet architektury.

Všechny naměřené přesnosti jsou pouze bodovými odhady. Jelikož neuronové sítě začínají trénink s náhodnými počátečními parametry (inicializací), při každém opakování tréninku na stejném datasetu se výsledné metriky budou (často v rámci malé odchylky) lišit.

Dalším objektem výzkumu by mohla být například trénovatelná mel-filterbank matice nebo použití Hartley transformace namísto Fourierovy.

6 Bibliografie

Reference

[1] Xla: Optimizing compiler for machine learning, 2022.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc., 2020.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[6] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. Keyword transformer: A self-attention model for keyword spotting. In *Interspeech 2021*. ISCA, aug 2021.

[7] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[8] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[10] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[13] Tensorflow documentation. Introduction to gradients and automatic differentiation.

[14] Xingping Dong and Jianbing Shen. Triplet loss in siamese network for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[16] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2020.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[18] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *International Conference on Artificial Neural Networks*, pages 220–229. Springer, 2007.

[19] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[21] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners, 2021.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[26] Po-Yao Huang, Hu Xu, Juncheng Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, and Christoph Feichtenhofer. Masked autoencoders that listen, 2022.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[28] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021.

[29] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. Alphafold 2. *In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book*, 2020.

[30] J. Kahn, M. Rivière, W. Zheng, E. Kharitonov, Q. Xu, P. E. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, T. Likhomanenko, G. Synnaeve, A. Joulin, A. Mohamed, and E. Dupoux. Libri-light: A benchmark for asr with limited or no supervision. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7669–7673, 2020. `https://github.com/facebookresearch/libri-light`.

[31] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[32] Byeonggeun Kim, Simyung Chang, Jinkyu Lee, and Dooyong Sung. Broadcasted residual learning for efficient keyword spotting, 2021.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[35] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[36] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[37] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

[38] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

[39] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[40] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

[41] Dianwen Ng, Jin Hui Pang, Yang Xiao, Biao Tian, Qiang Fu, and Eng Siong Chng. Small footprint multi-channel convmixer for keyword spotting with centroid based awareness. *arXiv preprint arXiv:2204.05445*, 2022.

[42] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2018.

[43] Mats L. Richter, Julius Schoning, Anna Wiedenroth, and Ulf Krumnack. Should you go deeper? optimizing convolutional neural network architectures without training. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, dec 2021.

[44] R.C. Rose and D.B. Paul. A hidden markov model based keyword recognition system. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 129–132 vol.1, 1990.

[45] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[46] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2):40–48, 1990.

[47] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[48] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018.

[50] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 2019.

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[52] Ming Sun, Anirudh Raju, George Tucker, Sankaran Panchapagesan, Gengshen Fu, Arindam Mandal, Spyros Matsoukas, Nikko Strom, and Shiv Vitaladevuni. Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting. In *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, dec 2016.

[53] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[54] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision.

In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[55] C Teacher, H Kellett, and L Focht. Experimental, limited vocabulary, speech recognizer. *IEEE Transactions on Audio and Electroacoustics*, 15(3):127–130, 1967.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[57] Roman Vygon and Nikolay Mikhaylovskiy. Learning efficient representations for keyword spotting with triplet loss. In *Speech and Computer*, pages 773–785. Springer International Publishing, 2021.

[58] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers, 2022.

[59] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[60] P. Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints*, April 2018.

[61] Martin Wollmer, Florian Eyben, Joseph Keshet, Alex Graves, Bjorn Schuller, and Gerhard Rigoll. Robust discriminative keyword spotting for emotionally colored spontaneous speech using bidirectional lstm networks. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3949–3952, 2009.

[62] Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. Tableformer: Robust transformer modeling for table-text encoding. *arXiv preprint arXiv:2203.00274*, 2022.

[63] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes, 2019.

[64] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers, 2017.

[65] Yi Zhou, Junjie Yang, Huishuai Zhang, Yingbin Liang, and Vahid Tarokh. Sgd converges to global minimum in deep learning via star-convex path, 2019.