VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AUTOMATICKÝ PŘEPIS ŘEČI LETECKÉ KOMUNIKACE DO TEXTU

AUTOMATIC TRANSCRIPTION OF AIR-TRAFFIC COMMUNICATION TO TEXT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR BALOK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IGOR SZÖKE, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

139625

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Balok Petr**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Automatický přepis řeči letecké komunikace do textu**
Kategorie:	Zpracování signálů
Akademický rok:	2022/23

Zadání:

1. Seznamte se s automatickým rozpoznáváním řeči a nástroji jako jsou ESPnet2, NeMo, SpeechBrain, Whisper.
2. Natrénujte rozpoznávač pro český a anglický jazyk na veřejně dostupných datech.
3. Připravte dodaná VHF trénovací data. Přidejte je do trénovací sady, přetrénujte a vyhodnoťte úspěšnost rozpoznávače.
4. Vytvořte bi-linguální rozpoznávač. Průběžně vyhodnocujte jeho úspěšnost, jak na standardních, tak VHF datasetech.
5. Zhodnoťte výsledky a navrhněte směry dalšího vývoje.
6. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Shinji Watanabe, et al. ESPnet: End-to-End Speech Processing Toolkit, Proceedings of Interspeech, 2018
- ATCO2 projekt, http://atco2.org
- Dále dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce řeší problematiku získání přepsaného textu z audio souborů obsahujících záznamy letecké komunikace a audio soubory obsahující řeč ve dvou jazycích. Zvolenou problematiku řeším pomocí strojového učení. Konkrétně nástroji vytvořenými v jazyce Python, NeMo a Whisper. Před fine-tuningem modelů jsem získal WER 78 % na datech letecké komunikace a 60 % na bilinguálním datasetu. Pomocí těchto technologií se mi podařilo zmenšit chybovost přepisů na 24 % v přepisech letecké komunikace. Na dvojjazyčném datasetu jsem dosáhl 19 % WER (Word Error Rate – četnost chybně přepsaných slov). Výsledky této práce umožňují automatický přepis nahrávek letecké komunikace s nízkým počtem chyb v přepisu. Modely trénované na dvojjazyčném datasetu umožňují přepis nahrávek obsahujících angličtinu i češtinu zároveň.

Abstract

This thesis solves the problem of getting text transcription from audio files containing air-traffic communication and audio files containing speech in two languages. I solved this problem using machine learning, specifically by using toolkits written in Python called NeMo and Whisper. Before fine-tuning, I got a 78 % word error rate on an ATC dataset and a 60 % word error rate on a bilingual dataset. Using these technologies, I managed to lower the word error rate to 24 % in transcriptions of air-traffic communication. I also got a 19 % word error rate for bilingual speech. The results of this thesis allow automatic transcription of air-traffic communication with a low rate of errors in the transcript. Furthermore, models trained on bilingual dataset allow transcribing audio files containing both English and Czech speech in one file.

Klíčová slova

automatický přepis řeči, strojové učení, NeMo, Whisper, letecká komunikace, umělé neuronové sítě

Keywords

automatic speech recognition, machine learning, NeMo, Whisper, air-traffic communication, artificial neural networks

Citace

BALOK, Petr. *Automatický přepis řeči letecké komunikace do textu.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szöke, Ph.D.

Automatický přepis řeči letecké komunikace do textu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Balok
9. května 2023

Poděkování

Chtěl bych poděkovat mému vedoucímu práce Ing. Igorovi Szökemu Ph.D. za odborné rady a skvělé vedení této bakalářské práce. Dále chci poděkovat mojí rodině, která mě po celou dobu studia podporovala.

Obsah

Kapitola 1

Úvod

Přepis textu mluveného v audio souboru se dlouhou dobu musel vytvářet ručně. Bylo nutno poslechnout si audio soubor, přepsat jeho obsah do textu a tento text uložit například jako titulky k danému audio souboru. S touto činností ale velmi pomohlo strojové učení, které se naučilo rozpoznávat řeč v těchto souborech a automaticky vytvářet k těmto souborům přepsaný text.

Přepis audio souborů do psaného textu je v dnešní době velmi rozšířený. Se softwarem vytvořeným pro tento účel je možné se setkat již na velkém množství míst. Velmi známou funkcí je například automatický přepis mluveného textu videa na webovém video serveru YouTube, který pro angličtinu funguje velmi dobře.

V oblasti letecké komunikace je automatický přepis textu velkým pomocníkem. V této oblasti ale čeká velké množství překážek, které tyto přepisy velmi ztěžují. Jednou překážkou pro přepis textu z tohoto druhu audio souborů je obrovské množství šumu. Šum ztěžuje práci nejen strojovému učení, ale hlavně lidem, kteří tyto přepisy vytváří sami, nebo tyto přepisy kontrolují.

Hlavní překážkou pro automatické rozpoznávání řeči z letecké komunikace jsou časté změny jazyka klidně několikrát v jedné nahrávce. Toto způsobuje problémy hlavně tím, že modely jsou často trénovány pouze na jeden jazyk a neporadí si například s názvy měst v jiných jazycích.

Cílem této práce je vytvořit model strojového učení, který bude schopen s přijatelnou chybovostí rozpoznat mluvený text i v tomto hůře srozumitelném audiu. Pro vytvoření tohoto modelu využiji nástroj vytvořený firmou Nvidia jménem Nemo [9]. Tento nástroj je určen pro tvorbu modelů umělé inteligence pro konverzaci počítače s člověkem pomocí přirozeného jazyka. Z tohoto nástroje využiji pouze část ASR (Automatic Speech Recognition – automatické rozpoznávání řeči).

NeMo ASR poskytuje řadu modelů umělých inteligencí pro přepisování audia do textu, včetně předtrénovaných modelů, který mají velmi nízkou chybovost na čistých datech (neobsahujících šum), a to nejen pro angličtinu, ale i další jazyky. Z tohoto nástroje vytvořím model strojového učení, který natrénuji nejdříve na čisté angličtině, poté jej natrénuji na datasetu ATCO2 [20] obsahujícím záznamy letecké komunikace.

Mimo nástroj NeMo využiji také nástroj Whisper od OpenAI [15]. Nástroj Whisper obsahuje nejen modul pro ASR, ale také moduly pro překlad textu do různých jazyků. Tento nástroj byl natrénován na 680 000 hodinách audia ve více jazycích s velmi nízkou chybovostí i na jiných jazycích, než je angličtina. Whisper v mé práci využiji pouze pro ASR a jeho modely natrénuji na již zmíněném datasetu ATCO2 a bilinguálním datasetu.

Kapitola 2

Umělé neuronové sítě

V této kapitole popíši neuronové sítě a jejich typy. Informace jsem převzal z [5, 6, 11, 18]. Umělé neuronové sítě jsou prostředkem, jakým se provádí strojové učení, ve kterém se počítače učí vykonávat různé úkoly. Tyto úkoly se učí na základě analýzy ukázek, které mají popsané správné odpovědi. Například systému pro rozpoznání věci na fotografii je možno dát jako vstup tisíce těchto obrázků a daná neuronová síť najde vzor, který konzistentně koreluje s daným popisem.

Tyto umělé neuronové sítě jsou modelovány podle lidského mozku, který obsahuje miliardy neuronů, které jsou spojené synapsemi. Umělé neuronové sítě jsou v mnohem menším měřítku, než je lidský mozek, jinak by nebyly použitelné z důvodu enormních výkonových požadavků.

Umělé neuronové sítě jsou tedy vytvořeny z uzlů (neuronů), které jsou mezi sebou propojeny. Těmto spojením se přidávají váhy. Při aktivaci neuronu se do něj pošlou data, která si vynásobí váhou na daném spojení. Výsledky se poté sečtou dohromady a odešlou se do aktivační funkce, která hodnoty odešle dál do sítě.

Neurony se dále seskupují do vrstev. Nejnižší vrstvou je vstupní vrstva, do které přijdou vstupy do celé neuronové sítě. Následuje počet vrstev neuronů, které vstupy zpracují, až se data dostanou na výstupní vrstvu. Pokud jsou neurony propojeny pouze jedním směrem, síť je typu *feed-forward*. Data se v takovéto síti zpracují jednosměrně a získá se výsledek.

Na začátku trénování nové sítě se všem neuronům v síti nastaví náhodné hodnoty na váhy a hraniční hodnoty neuronů. Během trénování jsou tyto váhy a hraniční hodnoty neustále upravovány, aby stejná data na vstupu konzistentně dávala stejné výstupní hodnoty. Jednoduchým typem neuronových sítí jsou perceptrony, na kterých dále vysvětluji, jak funguje trénování neuronových sítí.

2.1 Perceptron a feed-forward neuronové sítě

Perceptron je jednoduchý typ neuronu, který lze spojit do vrstev. Perceptron se může také nazývat lineární klasifikátor, určuje totiž, zda vstupní hodnoty do nějaké kategorie patří, či nepatří. Jeho součásti jsou:

- Vstupní hodnoty, nebo vstupní vrstva
- Váhy
- Vážený součet vstupních hodnot

- Aktivační funkce

Vstupní vrstvou mohou být vstupní data, nebo výstup z jiného perceptronu. Tyto hodnoty se vynásobí váhami u jednotlivých vstupů. Když tyto hodnoty sečteme, získáme sumu vstupních hodnot.

$$y = \sum_{i=1}^{m} w_i x_i$$

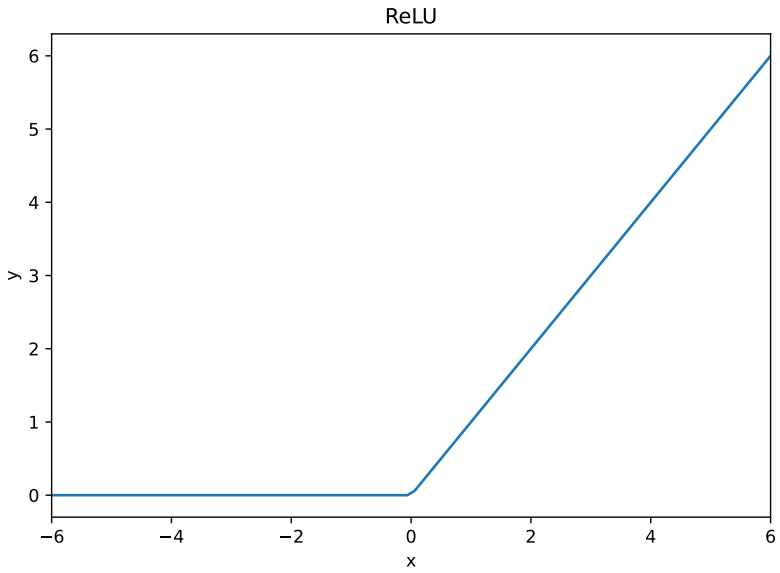
m je počet vstupů perceptronu.
w je vektor vah.
x je vektor vstupních hodnot perceptronu.
y je vážená suma vstupních hodnot

Takto vypočítaná hodnota se následně předá jako vstup do aktivační funkce perceptronu. Úlohou aktivační funkce je namapovat sumu vstupních hodnot do rozmezí určeného zvolenou funkcí. Výstupní hodnota aktivační funkce je zároveň výstupní hodnotou perceptronu a tedy reakcí perceptronu na vstupní data v daný moment. V praxi se používají nelineární aktivační funkce. Hodně používaná je logistická aktivační funkce (sigmoida). Používá se hlavně, protože obor hodnot této funkce je $\langle 0; 1 \rangle$. To znamená, že logistickou funkci lze dobře použít u modelů, které určují pravděpodobnost nějakého výstupu [17].

Nejčastěji se používá funkce ReLU. Má obor hodnot $\langle 0; \infty)$. Tato funkce je definována následovně.

$$f(x) = max(0, x)$$

Pro veškeré vstupní hodnoty menší než 0 vrací hodnotu 0. Je také rychlejší na výpočet, neobsahuje totiž žádné složité matematické konstrukce. Tato funkce má také své nevýhody, neuronová síť může totiž dojít do bodu, kde některé neurony budou úplně neaktivní. Pokud se váhy neuronu dostanou na nulu, již není způsob, aby neuron vracel jinou hodnotu, než nula [10]. Graf ReLU funkce je přiložen v obrázku 2.1.



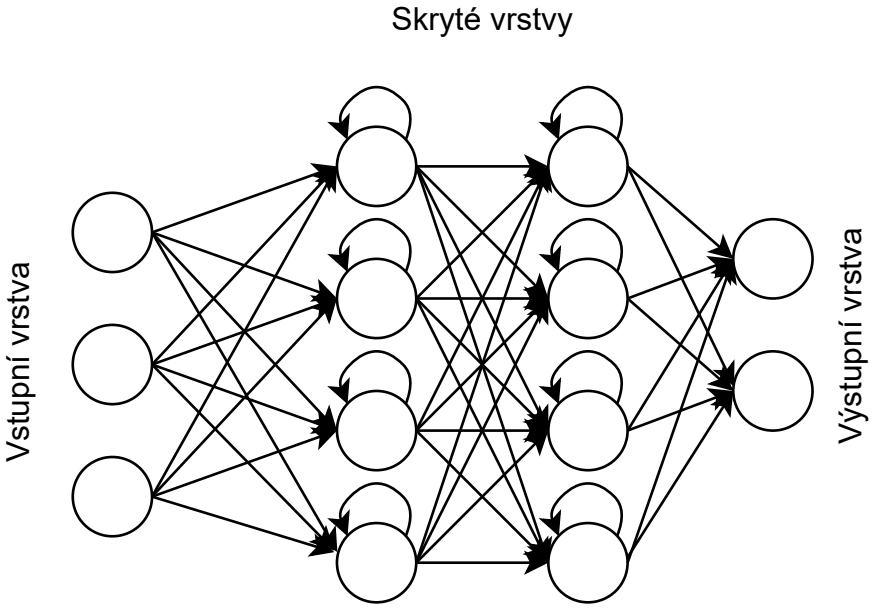
Obrázek 2.1: Graf ReLU funkce

Aby se perceptron mohl učit, je třeba mít připravena popsaná trénovací data. Ze začátku se nastaví váhy v perceptronu náhodně. Po zpracování vstupních dat se vypočte chyba oproti reálné hodnotě. Tato chyba vynásobená hodnotou *learning rate* je přičtena k dané váze u vstupu.

Perceptrony mohou fungovat jako jedna, ale i více vrstev. Z takto spojených perceptronů vzniká neuronová síť typu *feed-forward*. V sítích tohoto typu postupují data pouze jedním směrem. Má-li síť nějaké skrté vrstvy, je také nutno použít algoritmy jako *backpropagation* pro změnu vah ve skrytých vrstvách.

2.2 Rekurentní neuronové sítě

Rekurentní neuronové sítě jsou takové neuronové sítě, které používají pro vstup sekvenční data, nebo data z časových řad. Tento typ neuronových sítí je velmi rozšířený v oblasti překládání přirozeného jazyka, zpracování přirozeného jazyka (Natural Language Processing – NLP), rozpoznávání řeči a popisu obrázků. Od *feed-forward* sítí se liší hlavně tím, že mají „paměť“, ve které si pamatují předchozí vstupy, pomocí nichž vyhodnotí aktuální výstup. Každý neuron má tedy jedno spojení, které použije výstup tohoto neuronu jako jeho vstup. Pro tyto sítě by bylo vhodné dodat i data, která se stanou po aktuálním čase, ale tato data nemusí být vůbec dostupná. Například model pro automatické rozpoznávání řeči, která je dodána z mikrofonu nemůže dostat žádná data z budoucnosti, protože je řečník do mikrofonu ještě vůbec neřekl. Proto se používají pouze data z minulosti. V obrázku 2.2 se nachází schéma jednoduché rekurentní neuronové sítě.



Obrázek 2.2: Schéma jednoduché rekurentní neuronové sítě

Další změnou oproti *feed-forward* neuronovým sítím je sdílení parametrů na jednotlivých vrstvách. Ve *feed-forward* modelech má každý neuron vlastní váhy, u rekurentních neuronových sítí jsou tyto váhy stejné pro celou jednu vrstvu. Tyto váhy se trénují pomocí zpětného šíření v čase (backpropagation through time), model se tedy učí pomocí chyb na jeho výstupních vrstvách, které se šíří vrstvami zpět a upravují parametry vstupních vrstev.

Tímto způsobem učení ale rekurentní neuronové sítě naráží na dva problémy. Explodující gradienty a mizící gradienty. Tyto chyby jsou závislé na sklonu loss funkce. Pokud je gradient příliš malý, bude pokračovat ve zmenšování. To způsobí, že se váhy budou zmenšovat také, dokud nedosáhnou hodnoty 0 a budou úplně bezvýznamné, gradient vymizí. Pokud se naopak bude gradient zvětšovat a s ním také jednotlivé váhy, může se stát, že váhy budou obsahovat tak velké číslo, že se z něj stane NaN (Not a Number – programová reprezentace hodnoty, která není číslo) a model bude nestabilní. Řešením je snížit počet skrytých vrstev, aby se snížila komplexita modelu.

Jedno z řešení mizejících gradientů je také Long short-term memory (LSTM) RNN. Tyto sítě mají ve skrytých vrstvách buňky, které si pamatují různé informace podstatné pro danou síť. Pomocí toho mohou uchovávat informace i dlouhodobě. V nástroji NeMo jsou modely vytvořené na základě rekurentních neuronových sítí a LSTM také dostupné.

2.3 Konvoluční neuronové sítě

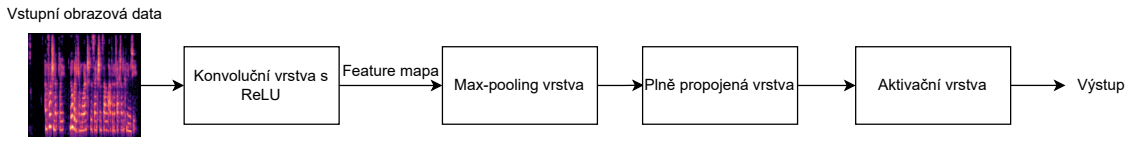
Konvoluční neuronové sítě (anglicky Convolutional Neural Networks – CNN) se používají, pokud mají vstupní data velké rozměry. Těmito daty bývají často obrázky, které mohou mít velikosti tisíce, či miliony pixelů. Specialitou tohoto typu neuronových sítí jsou konvoluční vrstvy.

Tyto vrstvy dostanou na vstup velkou matici dat, kterou konvolucí zmenší do malých matic, přičemž zachovávají důležité informace. Konvoluční vrstvy obsahují konvoluční kernel, což je malá matice hodnot použitých pro výpočet konvoluce hodnot v dané oblasti. Konvolučním kernelem se postupně přepočítají všechny oblasti v obrázku a vznikne tak aktivační mapa (feature map), která se zpracuje zbytkem neuronové sítě.

Další výhodou jsou max-pooling vrstvy, které v některých lokacích fungují jako výstupy z neuronů. Tyto vrstvy zmenší velikost aktivačních map.

Výstupy z max-pooling vrstev se následně vyhodnotí plně propojenými neuronovými sítěmi. Další částí těchto neuronových sítí je aktivační vrstvy, které využívají aktivační funkce, například logistickou funkci, nebo ReLU. Z těchto vrstev vychází výstup CNN. Grafické znázornění těchto vrstev je ukázáno na obrázku 2.3.

Tento typ neuronových sítí dělá zpracování velkého množství dat rychlejší. Není totiž nutno používat tisíce klasických neuronů využívajících maticové operace. Konvoluční bloky či moduly využívá například NeMo ASR v jeho různých modelech, například Jasper nebo Conformer.



Obrázek 2.3: Diagram vrstev konvoluční neuronové sítě

Kapitola 3

Automatické rozpoznávání řeči

V této kapitole jsou popsány přístupy, pomocí kterých se automatické rozpoznávání řeči řešilo v minulosti [4, 12], a jaké způsoby jsou používány dnes [16]. Popisuji jaké součásti tvoří systém automatického rozpoznávání řeči [3]. V této kapitole je také popsána role tokenizerů v ASR (Automatic speech recognition) systémech [7, 19]. Další částí je popis transformer modelů pro ASR [1]. Dále je zde také popsáno, jaké metriky se používají pro vyhodnocení kvality modelu pro automatické rozpoznání řeči. Tuto kapitolu zakončuji popisem problémů, se kterými je možno se setkat při vytváření ASR systému na přepis letecké komunikace.

3.1 Historie rozpoznávání řeči

První experimenty s automatickým rozpoznáváním řeči se objevily již v roce 1952. V tomto roce byl vytvořen rozpoznávač číslic z audia nazvaný Audrey. V roce 1960 poté IBM vyvinulo systém, který byl schopen rozpoznávat číslice a aritmetické operace. V Japonsku se v tomto roce vyvíjí také systém pro rozpoznání menších částí řeči, například samohlásek.

Na začátku 70. let 20. století spustilo oddělení obrany ARPA ve Spojených státech výzkum pro porozumění řeči. V tomto 5-ti letém výzkumu vzniklo několik nových ASR systémů, nejúspěšnějším byl systém s názvem Harpy, který uměl rozpoznat přes 1 000 slov.

V 80. letech přišly na řadu Hidden Markov Modely (HMM). Tyto modely se nesnažily pochopit řeč počítačem jako člověk, ale byly to statistické modely pro rozpoznávání řeči. Tyto modely se již trénovaly vstupními datasety a na dalších datasetech se hodnotily.

V 90. letech se automatické rozpoznávání řeči začalo využívat spotřebiteli. Jedním z prvních programů byl Dragon Dictate, který měl slovník o velikosti 80 000 slov a obsahoval také funkce pro zpracování přirozeného jazyka.

Po příchodu neuronových sítí a hlubokého učení se začaly objevovat end-to-end modely pro automatické rozpoznávání řeči. Tyto modely se trénují celé najednou a není příliš obtížné do nich natrénovat třeba další jazyk.

3.2 Moderní přístupy rozpoznávání řeči

Moderním přístupem pro zpracování je vytvořit model, který bude obsahovat všechny potřebné součásti. Tyto typy modelů se nazývají *end-to-end* modely. Daný model dostane na vstup audio signál a jeho výstupem bude textový přepis. Dnešní modely obsahují minimálně audio enkodér a textový dekodér. Součástí může být ale také preprocesor pro vstupní audio signál. Enkodér zpracovává vstupní signál převedením na logaritmický mel-spektrogram.

Pomocí mel-spektrogramu se detekují znaky obsažené v audio signálu. Logaritmický mel-spektrogram se používá, protože lidé nevnímají mluvenou řeč na lineární ose frekvencí, ale právě na logaritmické. Tyto detekované znaky se předají do dekodéru, který z těchto znaků vytvoří slova a dále také věty, které se v audiu nachází. Důležité je, že *end-to-end* modely se mohou trénovat jako celek zároveň, i když obsahují několik neuronových sítí.

Enkodér pro detekci znaků může používat různé přístupy, může to být právě HMM, spíše se ale využívají neuronové sítě. Zde se mohou použít neuronové sítě typu RNN (Recurrent Neural Networks – Rekurentní neuronové sítě) a CNN (Convolutional Neural Networks – konvoluční neuronové sítě). Nejlepší modely ale využívají např. attention-based modely, transformer modely atd. Neuronové sítě pomáhají lépe generalizovat, důsledkem je lepší detekce textu při změně mluvčího, přízvuku, domény (např. letecká komunikace), dokonce i jiného jazyka.

Dekodér má také různé možnosti implementace. Může dekódovat jednotlivé znaky, které se poté poskládají do slov a vět, nebo může využívat slovník, pomocí kterého skládá slova a následně i věty. Pro dekodéry se používá například algoritmus Greedy search, výstupem bude tedy znak, nebo slovo, s nejvyšší podmíněnou pravděpodobností z dostupných znaků či slov. Greedy search má nevýhodu, že může způsobit výběr nejvíce pravděpodobného slova, které zní podobně, ale v přepisu se nenachází. Toto může dále způsobit horší WER (Word Error Rate).

Pro dekodéry, které používají slovníky pro kontext jednotlivých dekódovaných slov se více využije Beam search. Beam search pro určení slova použije předchozí detekovaná slova a vybere N kandidátů, kteří mají v tomto kontextu největší pravděpodobnost, že budou následovat. Podle největší pravděpodobnosti se rozhodne, které slovo se do výstupu přidá jako další.

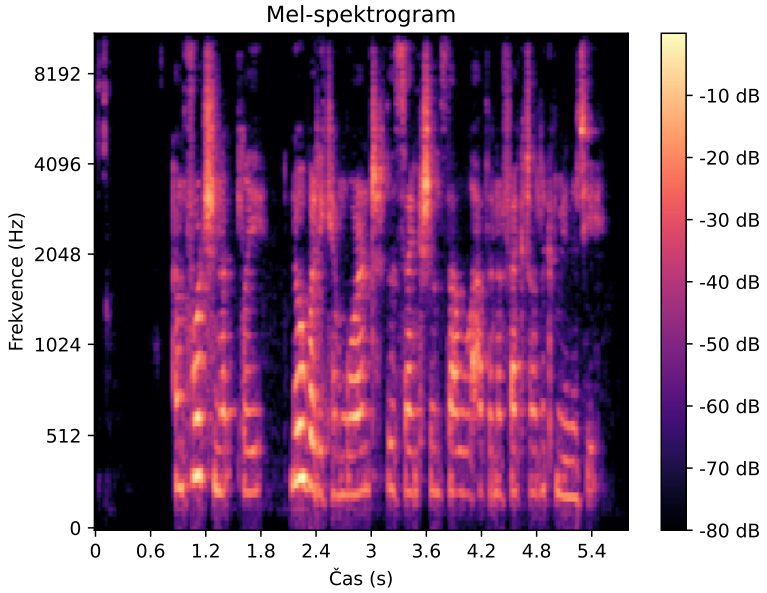
3.3 Součásti systému pro rozpoznávání řeči

Automatické rozpoznávače řeči začínají u preprocesoru dat. Úkolem preprocesoru je přečíst zdrojová audio data a připravit je do formátu, který přijme neuronová síť pro rozpoznávání řeči. Prvním krokem je samotná data načíst z různých formátů, jako jsou například *wav* nebo *mp3*. Po přečtení ze souboru se data načtou do 2D pole (v Pythonu se většinou používá knihovna Numpy z důvodu efektivnější implementace), které obsahuje hodnoty reprezentující amplitudu zvuku v daném čase. Velikost tohoto pole závisí na vzorkovací frekvenci audio souboru. Ve zdrojovém audio souboru se také nachází audio s různým počtem kanálů. Pro tyto účely většinou mono, nebo stereo, tedy jeden či dva audio kanály. Máme-li stereo audio soubor, naše pole narůstá o další dimenzi, abychom mohli uložit i druhý kanál.

Tímto ale práce preprocesoru nekončí, audio soubor může totiž mít různou vzorkovací frekvenci, různý počet kanálů a hlavně různou délku. S těmito všemi komplikacemi se preprocesor musí vypořádat, protože neuronová síť, do které se data předávají, očekává určitou velikost vstupních dat. Data se tedy musí převzorkovat, upravit počet kanálů a následně oříznout, případně doplnit, aby měla správnou délku. Pokud mají vstupní data špatnou kvalitu, je možno také provést odstranění šumu.

Další možností preprocesoru je upravit vstupní data drobnými změnami. Data je možno trochu časově posunout nebo změnit rychlost audia. Tyto úpravy pomáhají modelu lépe generalizovat. Výsledný model by poté měl být schopen lépe pracovat s různými daty, nemělo by tedy příliš záležet na změně řečníka, či rychlosti mluvení atp.

Poslední úlohou pro preprocesor je extrakce příznaků (features) ze vstupních dat. Příznaky mohou mít formát mel-spektrogramu, nebo MFCC (Mel Frequency Cepstral Coeffi-



Obrázek 3.1: Mel spektrogram nahrávky z anglického datatasetu Common Voice

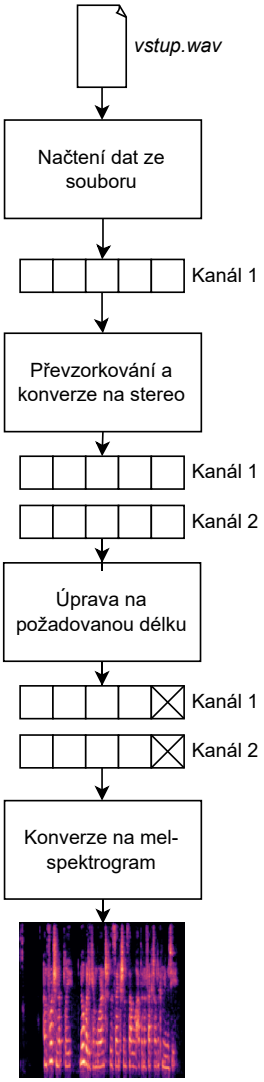
cients). Na obrázku 3.1 je možno vidět, jak mel-spektrogram vypadá. V tomto bodě je také možno provést různé úpravy spektrogramu (spectrogram augmentation). Tyto úpravy mohou být časové nebo frekvenční maskování, tedy odmazání určité frekvence nebo času, aby model rozuměl také neúplným datům. Schéma tohoto procesu je možno vidět v obrázku 3.2.

Takto upravená data mohou být použita jako vstup neuronové sítě. Posledním krokem je načíst přepisy trénovacích audio souborů.

Pro samotný model rozpoznávače řeči se používají neuronové sítě, kterých je mnoho typů. Pro účely ASR (Automatic Speech Recognition – automatické rozpoznávání řeči) se často používá kombinace CNN (Convolutional Neural Network – konvoluční neuronové sítě) s RNN (Recurrent Neural Network – rekurentní neuronové sítě), případně pouze RNN.

První neuronovou sítí, do které zamíří vstupní data je CNN, která vytvoří mapy příznaků (feature map). Tyto mapy se předají jako vstup do RNN. V RNN se s pomocí LSTM (Long Short Term Memory) vrstev tyto mapy zpracují a vytvoří rámce, na které se mapuje detekovaný text. Ze spojitého vstupu se stává diskrétní výstup, který se pošle do *softmax* funkce. Ze *softmax* funkce se získají hodnoty pravděpodobností jednotlivých znaků pro daný rámec. Toto je náš enkodér.

Poslední překážkou je z těchto znaků vytvořit smysluplnou větu. To je ale docela obtížné, protože enkodérem mapované rámce nemají určeno, že rámec je jedno písmeno. Každé písmeno může na spektrogramu mít jinou délku, takže pouhé přepsání detekovaných písmen není řešením. K těmto účelům slouží CTC (Connectionist Temporal Classification) algoritmus. CTC použije pravděpodobnosti jednotlivých znaků a vytvoří z nich jednotlivá slova. Využívá k tomu také speciální znak „-“, který funguje podobně jako `NULL`, tedy žádný znak nebyl detekován. Tímto znakem se vytvoří mezery mezi ostatními znaky. Těmito mezerami je možno správně detekovat například dva stejné znaky jdoucí za sebou (např. ve slově „good“). Než CTC vrátí finální text, vymaže z něj tyto zástupné znaky a výsledkem by mělo být detekované slovo.



Obrázek 3.2: Diagram operací prováděných preprocesorem

Pro trénování modelů s CTC se používá *CTC loss*. CTC loss je hodnota správnosti přepisu mezi odhadnutým přepisem neuronové sítě a reálným přepisem audia. Společně s CTC se také může použít Beam search.

Pro lepší výsledky dekodéru je možno použít také jazykový model. Jazykové modely by měly vytvářet smysluplné slova a věty a jejich použití s ASR je také možné.

3.4 Tokenizery a jejich použití v ASR

Tokenizery se používají pro rozdělení vstupních dat do menších částí. Vstupní data se pomocí slovníku rozdělí na skupiny znaků, které se nazývají tokeny. Pro rozdělení textu na tokeny se nejčastěji používají algoritmy BPE (Byte-Pair Encoding), WordPiece a SentencePiece. Rozdělení na tokeny může probíhat jednoduše tím, že se tokeny vytvoří ze slov. Dělící znak je tedy oddělovač slova. Může se také použít rozdělení na úrovni části daného slova, toto se nazývá *sub-word tokenization*. Dále se také může používat dělení po jednotlivých

znacích. Rozdělení samotných slov je sice jednoduché, ale při zpracování většího množství dat bude slovník obrovský, tímto se také sníží rychlost tokenizeru. V případě dělení tokenů na jednotlivé znaky bude slovník malý, ale jednotlivé tokeny ztrácí význam v kontextu věty. Nejlepším řešením je právě sub-word tokenization, pomocí kterého fungují nejčastěji používané algoritmy pro tokenizaci textu.

BPE (Byte Pair Encoding) dostane na vstupu jednotlivá slova z textu. Poté si vytvoří základní slovník, což je kolekce nejčastějších párů znaků ze vstupních slov. Tyto páry znaků se do slovníku přidávají, dokud slovník nedosáhne specifikované velikosti.

WordPiece funguje na podobném principu jako BPE, ale neupřednostňuje četnost výskytů párů daných znaků, ale pravděpodobnost že daný pár znaků se v textu vyskytne.

SentencePiece se na tyto problémy zaměřuje úplně jinak. Veškeré znaky totiž klasifikuje jako tokeny a nepotřebuje tedy žádné dělení slov na vstupu. Navíc funguje pro jazyky jako je například japonština, které nemají slova oddělená mezerami. S tímto přístupem je navíc rychlejší, než ostatní algoritmy tokenizerů.

V ASR se tokenizery používají pro dekódování výstupního textu, vzhledem k lepším výsledkům, než Greedy nebo Beam search.

3.5 Transformer modely

Aktuálně se často pro ASR používají transformer modely, které dosahují skvělých výsledků. Transformery jsou také encoder-decoder modely, rozdíl je v tom, jak vypadá jejich enkodér a dekodér.

Jejich enkodér se skládá ze self-attention modulu a feed–forward neuronové sítě. V self-attention vrstvě jsou přítomny matice, pomocí kterých se přidělí skóre jednotlivým vstupům. Tyto matice obsahují hodnoty, kterými se vynásobí vstupní matice, tyto hodnoty se upravují během trénování. Tyto trénovatelné matice se nazývají *query*, *key* a *value*. Toto jsou matice vah.

$$
\begin{aligned}
Q &= X \cdot W^Q \\
K &= X \cdot W^K \\
V &= X \cdot W^V \\
Z &= softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V
\end{aligned}
$$

X je matice vstupních hodnot
W^Q je query matice vah
W^K je key matice vah
W^V je value matice vah
K^T je transponovaná key matice
d_k je velikost key matice
Z je výstupní matice

Takto vypočítaná matice se nazývá attention head, což je matice nesoucí pravděpodobnosti různých slov. Pro větší přesnost je lepší vytvořit těchto attention head matic více, aby byly brány v úvahu všechny možné formy výstupu. Po vytvoření attention head matic se tyto matice spojí a vynásobí maticí vah W^O, která se také mění během trénování. Po

tomto kroku již zůstane jedna matice, která bude vstupem do feed–forward neuronové sítě. Tímto získáme pravděpodobnosti výskytu jednotlivých vstupních dat, většinou slov.

Dekodér poté získá výstupní slova pomocí greedy nebo beam search. Jeho výstupem je index slova ve slovníku, které se vypíše jako výstup. Dekodér vyhodnocuje výstupní slova po jednom, pokud není další slovo, které by se do věty přidalo, dekodér to indikuje speciálním tokenem značícím konec věty. Tyto modely používají pro trénování loss funkci, kterou upravují váhy v jednotlivých maticích.

3.6 Metriky pro hodnocení kvality modelu rozpoznávání řeči

V automatickém rozpoznávání řeči je nutno vědět, jak kvalitní natrénovaný model je. V oblasti kvality modelu se jako nejlepší metrika jeví počet slov, které model správně dekódoval, dále WER (Word Error Rate – chybovost přepisu slov). WER se vyhodnocuje pomocí následující rovnice.

$$WER = \frac{S + I + D}{C}$$

S — Počet přepsaných slov oproti původnímu přepisu
I — Počet slov, která byla přidána oproti původnímu textu
D — Počet slov, která nejsou v přepisu, ale nachází se v původním textu
C — Celkový počet slov v původním textu

Ideální WER by měl být co nejblíže 0, tedy automatický přepis neobsahuje téměř žádné chyby oproti původnímu textu, se kterým se generovaný přepis porovnává.

Další metrika, která se může použít místo WER, je CER (Character Error Rate – chybovost přepisu znaků), tato metrika je mírnější, protože se porovnávají pouze přepsané znaky a ne celá slova. Výpočet metriky je stejný, jako u WER, akorát se nepočítají celá slova, pouze jednotlivé znaky.

Dále se může jako metrika použít SER (Sentence Error Rate), která tuto hodnotu počítá z celých přepsaných vět.

Z těchto metrik je nejvíce používanou metrikou WER (Word Error Rate). Tato metrika kontroluje, zda přepsaná slova modelem dávají smysl, je tedy vhodná pro většinu jazyků. CER (Character Error Rate) je mírnější, ale pro některé jazyky vhodnější (například japonština). SER (Sentence Error Rate) je vhodné použít, pokud se model bude používat pro delší texty. V mojí práci využívám jako metriku hlavně WER (Word Error Rate).

3.7 Automatické rozpoznávání letecké komunikace

V mém projektu se zabývám leteckou komunikací mezi piloty a ATC (Air Traffic Control – řízení letového provozu). Tato komunikace se přenáší přes VHF (Very High Frequency – vysoko frekvenční) rádio. Ať se již jedná o špatný mikrofon, nebo šum z ostatních frekvencí, vždy se v této komunikaci bude nacházet šum. Tento šum bude také různé intenzity, dále se mohou vyskytnout hlasité rušivé frekvence, nebo utlumení. Tyto faktory mají obrovský vliv na tuto komunikaci, proto se některé informace musí po přijetí říct znovu, aby se věž ujistila, zda pilot informaci převzal správně. Navíc se pro volání letadel používají volací znaky, což je další způsob eliminace chyb.

Při poslechu této komunikace dokáže být někdy velmi obtížné rozumět, co zrovna bylo řečeno. Zde neleží výzva pouze pro člověka, ale v mé práci hlavně pro strojové učení. Modely

pro rozpoznávání řeči bývají natrénované na čistých datech, neobsahujících téměř žádný šum. Proto nedokáží příliš dobře rozpoznat právě leteckou komunikaci. Tyto modely se musí natrénovat tak, aby si poradily se šumem a model mohl na relativně čistých datech provést detekci mluveného slova. Další možnost je data před přepisem odšumět, toto ale nebude možné, pokud se bude přepis vytvářet z audio streamu.

Piloti jsou z různých zemí světa, je zde tedy hodně přízvuků, které musí ASR systém rozpoznat. Největším problém ale nastává, když pilot komunikuje více jazyky v jedné nahrávce. Často to bývají jen malé části nahrávky, například pozdravy nebo názvy měst. Tyto faktory ale dost zvyšují WER (word error rate) modelu, pokud si s těmito částmi nahrávek neporadí. Modely, které přepisují tento text, si musí poradit s různými úrovněmi šumu, změnou řečníka několikrát v jedné nahrávce, také se změnou mluveného jazyka v jedné nahrávce.

Kapitola 4

Rozpoznávání řeči pomocí nástroje NeMo

Tato kapitola se věnuje nástroji NeMo, jeho závislostem a konfiguraci modelů obsažených v této knihovně. Také trénování a vyhodnocení těchto modelů.

NeMo je knihovna vytvořená v jazyce Python. Tato knihovna je dostupná pro verzi jazyka Python 3.6 a vyšší. Já jsem tuto knihovnu použil s verzí Pythonu 3.7. Další závislostí této knihovny je knihovna pro strojové učení PyTorch [14].

Knihovna NeMo obsahuje všechny potřebné součásti pro vytváření, trénování, validaci a testování různých modelů pro ASR (Automatic Speech Recognition – automatické rozpoznávání řeči). Obsahuje tedy preprocesor s nastavitelnými parametry pro runtime augmentaci dat. Je zde přítomen enkodér pro zpracování mel-spektrogramů a dekodér pro vytváření smysluplného textu. NeMo umí také pracovat s tokenizery, které jsou pro některé mnodely vyžadované. Pro správný průběh trénování je zde scheduler, který manipuluje s hodnotou learning rate a experiment manager, který z experimentů ukládá logy pro stavy modelu během trénování. Součástí této knihovny jsou i moduly pro zpracování přirozeného jazyka a převod textu na řeč. V mojí práci ale použiji pouze modul pro ASR.

Modely strojového učení knihovny NeMo již obsahují všechny potřebné součástí pro vytvoření jejich instancí. Pro otestování funkčnosti knihovny je možno využít předem natrénované modely a zkusit si přepsat nějaký audio soubor nebo dataset. Nejdůležitější součástí NeMo modelu je ale jeho konfigurace.

Tyto modely strojového učení jsou v NeMo reprezentovány jako propojení neurálních modulů. Knihovna NeMo je navržena tak, aby vše bylo abstrakcí jako neurální modul, ať už se jedná o enkodér, dekodér, loss funkci nebo manipulátor dat. Tyto neurální moduly obsahují funkce *forward* a *backward*, které určují, co se stane při předání dat do dalšího modulu. Aby tyto moduly fungovaly pro nějaký konkrétní účel, spojí se dohromady a vznikne tím neuronová síť pro různé účely, jako je třeba ASR (Automatic Speech Recognition – automatické rozpoznávání řeči).

Knihovna NeMo je navržena tak, aby implementace neurálních modulů, či neuronových sítí nebyla závislá na žádném frameworku. V budoucnu je vývojáři plánováno, že bude možno si vybrat framework strojového učení, na kterém neuronová síť poběží. Zatím je možno NeMo využít pouze za použití frameworku PyTorch.

4.1 Konfigurační soubory knihovny NeMo

Konfigurační soubory NeMo modelů obsahují hodnoty, které jsou v modelu použity pro jeho různé části. Tyto soubory obsahují konfiguraci pro *preprocesor, trénovací, validační a testovací datasety, enkodér, dekodér, optimalizátor a spektrogramový augmentor*. Mimo konfiguraci samotného modelu se v konfiguračních souborech nachází také nastavení pro *trainer* modelu a *experiment manager*. Dále se zde nachází také konfigurace jednotlivých bloků modelu.

Informace o datasetech

V konfiguračních souborech jsou také obsažena data o použitých datasetech. Pevně jsou zde nastavena data, která se nebudou měnit, jako například vzorkovací frekvence vstupních souborů, počet zároveň zpracovávaných souborů (batch size), maximální délka audia v jednom souboru (aby se přeskočily dlouhé soubory, které by zaplnily velké množství paměti) atd. Batch size je nutno nastavit, tak aby při trénování nedošla paměť.

Některé informace se doplní až při spuštění skriptu. To jsou hlavně systémové cesty k manifestům datasetu. Všechna tato nastavení je možno nastavit pro datasety jednotlivě. Rozlišuje se zde mezi trénovacím datasetem, validačním datasetem a testovacím datasetem.

Pro rozpoznávač řeči je nutno mít dostatek dat, na kterých se bude trénovat. Pro tento účel byly vytvořeny popsané datasety, které stačí upravit do formátu, v jakém je akceptuje NeMo a může se začít trénovat. Pro anglický jazyk existuje mnoho datasetů, já si vybral dataset LibriSpeech [13]. LibriSpeech je dataset délky 1 000 hodin, obsahující čtenou řeč z audio knih ve veřejné doméně. Pro tento dataset je možno využít skript dostupný v repozoitáři NeMo, který dataset, nebo jeho část, stáhne a převede do formátu, který NeMo ASR očekává.

Pro trénování modelu na český jazyk jsem použil dataset Mozilla Common Voice [2]. Tento dataset obsahuje různé věty, které mluví různí lidé, kteří se dobrovolně rozhodli tomuto projektu přispět namluvením různých vět. Mozilla Common Voice neobsahuje pouze češtinu, ale také řadu dalších jazyků, já pro NeMo používám pouze dataset pro český jazyk. Tento dataset má přepisy audio souborů uloženy v *TSV* souborech, tyto soubory nelze okamžitě použít pro NeMo ASR, musel jsem tedy tyto soubory převést do požadovaného formátu, který je popsán v dokumentaci nástroje NeMo.

Pro trénování modelů na datech letecké komunikace jsem využil ATCO2 dataset [20]. Tento dataset má řeč k jednotlivým audio souborům přepsanou v obsažených souborech. Bylo nutné si převést přepsané texty do formátu, který NeMo očekává. Tento formát využívá zápisů v JSON souboru, kde na každém řádku je JSON řetězec odpovídající jedné nahrávce obsažené v datasetu. Jeden záznam má následující formát

```
{
"audio_filepath": "cesta/k/audio/souboru",
"text": "Přepsaný text nahrávky",
"duration": Délka nahrávky v sekundách
}
```

Jakmile je dataset připraven ve formátu pro NeMo ASR, je možno přidat cestu k manifestu do konfigurace modelu a začít na něm model trénovat nebo provést na datasetu fine-tuning.

Experiment manager

Pro trénování modelu je vhodné nastavit *experiment manager*, umožňuje totiž ukládat vývoj trénování modelu do formátu, který se poté zobrazí pomocí programu *tensorboard*. Vedle těchto výstupů budou také uloženy *checkpointy* během trénování. Pro každý trénovaný model vytvoří *experiment manager* novou složku s jeho daty. Je tedy potom možné si zobrazit statistiky všech natrénovaných modelů. V těchto složkách se ukládají data pro tensorboard, ze kterých je možno zjistit vývoj různých hodnot během trénování modelu. Data obsahují hodnoty jako například WER a hodnotu loss funkce.

4.2 NeMo modely

NeMo ASR obsahuje dva typy modelů, modely založené na *CTC* a *sequence-to-sequence attention-based* modely. V mých experimentech používám pouze CTC modely, protože jsou méně náročné na trénování.

Tyto modely se trénují pomocí *CTC-loss* funkce. Pro instance modelů je možno použít konfigurační soubor, který popisuje, jaké bloky použijí, a jak se propojí, také je ale možno zavolat `NeuralFactory` (továrnu na neurální moduly) a definovat si neuronovou síť kompletně podle sebe pomocí API od NeMo.

Použité neurální moduly mají implementované sémantické kontroly vstupů a kontrolují se také velikosti tenzorů. Toto jsou velmi užitečné nástroje, které vyhovují vytváření vlastních neuronových sítí. Mé experimenty využívají konfigurace neuronových sítí zapsané v konfiguračních souborech, které se doplňují o data, která nejsou známa do doby běhu skriptu. Po spuštění trénovacího skriptu se chybějící data doplní, vytvoří se instance traineru a NeMo modelu a spustí se trénování. Po konci trénování je důležité zavolat metodu pro uložení modelu, aby se mohl dál použít pro přepis, případně fine-tuning.

4.3 Trénování NeMo modelů na anglických, českých a ATC datech

V této sekci popíši nutnosti potřebné pro trénování ASR (Automatic Speech Recognition) modelu a jak se model trénuje. V této sekci se nachází také vyhodnocení jednotlivých natrénovaných modelů.

V NeMo ASR je velké množství modelů, jejichž konfigurace se různě liší. Je možno si vybrat z malých modelů (Jasper, QuartzNet), po velké modely (Conformer). Různé typy modelů vyžadují různé množství paměti a budou se také různě dlouho trénovat.

Pro trénování modelu je nutno mít jeho konfigurační soubor a skript pro jeho trénování. U modelů, které to vyžadují, je nutno mít také tokenizer.

Mým cílem bylo hlavně si natrénovat model na rozpoznávání řeči, který by měl přijatelný WER (Word Error Rate). Od mých očekávaných 20 % WER se ale reálná data dost liší. Dále jsem byl částečně limitován velikostí paměti mého GPU (Graphics Processing Unit – grafický procesor), toto způsobilo delší dobu trénování modelu, a také nemožnost natrénovat Conformer model. Tento model totiž vyžaduje minimální *batch size* 256, kterého jsem ale nemohl dosáhnout z důvodu nedostatku paměti VRAM na mém systému.

Quartznet pro anglický jazyk

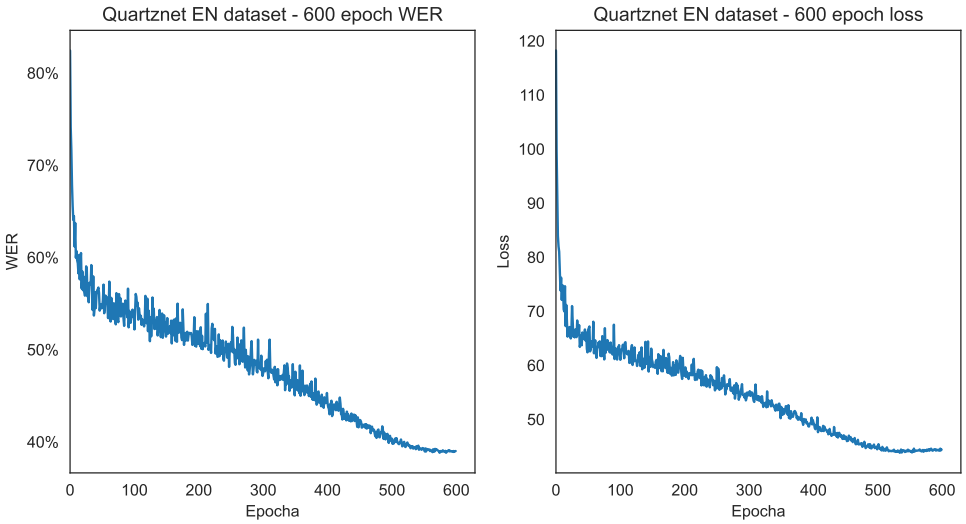
Pro začátek jsem si vybral model Quartznet, protože není příliš paměťově náročný a docela rychle se trénuje. Quartznet používá pro výstup přepisu jednotlivé detekované znaky, je tedy možné, že WER bude docela vysoký, ale CER bude o dost nižší.

Quartznet modely jsou tvořeny konvolučními bloky, které obsahují ReLU aktivační funkci. Počet těchto bloků je nastaven v konfiguračním souboru, v těchto modelech se také nastavuje, kolikrát se data zpracují, než dostaneme výsledek.

Konfiguraci modelu jsem zvolil ukázkovou konfiguraci dostupnou ve složce *examples* v NeMo repozitáři[1]. V konfiguraci jsem pouze upravil *batch size* na hodnotu 4, z důvodu malého množství paměti. Dále jsem si vytvořil skript, ve kterém jsem vytvořil instanci modelu a *traineru*, předal do modelu konfiguraci a spustil jsem trénování.

Pro trénování jsem použil dataset LibriSpeech (100 hodin), konkrétně jeho část *train-clean*. Pro vyhodnocení jsem použil část datasetu *test-clean*. Pro další vyhodnocení po trénování jsem použil části *dev-clean, dev-other, test-clean a test-other*. Mimo tyto datasety jsem do vyhodnocení zahrnul i ATC dataset.

Na grafu 4.1 můžete vidět hodnotu WER a loss během trénování na LibriSpeech datasetu. Z grafu loss funkce 4.1 je ale zřejmé, že nedochází k přetrénování (overfitting).



Obrázek 4.1: Vlevo: WER během trénování Quartznet modelu na anglickém datasetu LibriSpeech. Vpravo: Graf hodnoty loss funkce Quartznet modelu pro anglický jazyk

Jak je možno vidět v tabulce 4.1, podle WER na LibriSpeech datasetu, se mi model nepodařilo natrénovat na lépe, než 39 %. Důvodem bude nejspíše velikost modelu. Tato hodnota však není ideální, zároveň ale není nejhorší. Má cílená hodnota WER tohoto modelu byla pod 20 %, ideálně jsem se chtěl co nejvíce přiblížit hodnotě předtrénovaného Quartznet modelu, v tabulce 4.1 poslední řádek. Tento předtrénovaný model byl trénován po 400 epoch na LibriSpeech datasetu [8].

[1]Github repozitář NeMo: https://github.com/NVIDIA/NeMo

Model	test-clean	ATC	test-other	dev-clean	dev-other
Quartznet 60 epoch	42 %	104 %	68 %	42 %	67 %
Quartznet 240 epoch	39 %	104 %	66 %	38 %	65 %
Quartznet 600 epoch	39 %	107 %	66 %	38 %	65 %
Pretrained	3,9 %	85 %	11,28 %	3,83 %	11,08 %

Tabulka 4.1: WER pro model Quartznet trénovaný na anglickém jazyce, Pretrained je předtrénovaný Quartznet model s názvem `QuartzNet15x5Base-En`.

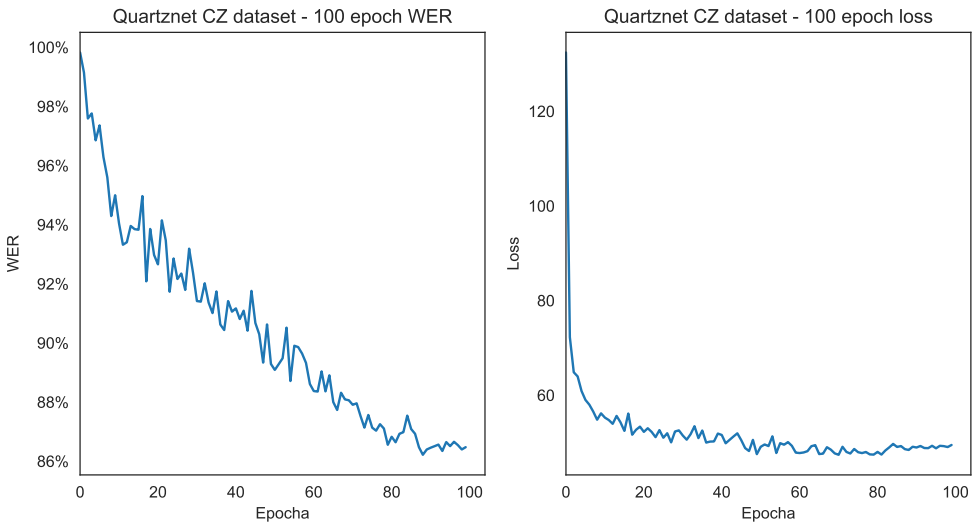
V tabulce 4.2 je možno vidět, že CER (Character Error Rate) se pod 20 % dostal.

Model	test-clean	ATC	test-other	dev-clean	dev-other
Quartznet 60 epoch	13 %	71 %	29 %	13 %	28 %
Quartznet 240 epoch	12 %	71 %	28 %	12 %	27 %
Quartznet 600 epoch	12 %	71 %	28 %	12 %	27 %

Tabulka 4.2: CER pro model Quartznet trénovaný na anglickém jazyce

Quartznet pro český jazyk

Pro český jazyk jsem zvolil také model Quartznet. Konfiguraci jsem zvolil stejnou, jako u modelu pro jazyk anglický s *batch size* hodnotou 4, a také jsem přidal chybějící česká písmena do *labels*. Jedna epocha tohoto modelu trvala natrénovat déle než model anglický, vzhledem k tomu, že tato konfigurace má mnohem větší abecedu. Tento model jsem nejprve natrénoval na 10 epoch, abych si ověřil, že konfigurace funguje. Dále jsem počet epoch zvýšil na 20, abych si ověřil, že WER klesá. Nakonec jsem tento model trénoval po 100 epoch, ale WER nedosáhl příliš dobrých výsledků.



Obrázek 4.2: Vlevo: Průběh WER při trénování Quartznet modelu na datasetu Common Voice v českém jazyce. Vpravo: Hodnota loss funkce během trénování modelu Quartznet na českém datasetu Common Voice

Hodnotu WER a loss během trénování na českém datasetu Common Voice můžete vidět v grafu 4.2.

Počet epoch	WER
Quartznet-cz 10 epoch	96 %
Quartznet-cz 20 epoch	95 %
Quartznet-cz 100 epoch	88 %

Tabulka 4.3: WER pro model typu Quartznet trénovaném na českém jazyce

Pro dosažení lepších výsledků je nutno model trénovat déle, nebo použít jiný typ modelu. Loss funkce opět napovídá, že model se trénoval v pořádku. WER dosažený při různém počtu epoch naleznete v tabulce 4.3.

Trénování modelu typu Citrinet pro anglický jazyk

Pro anglický jazyk jsem také zkusil natrénovat model typu Citrinet. Tento model je založen na modelu *Quartzunet* a navíc obsahuje také tokenizer, který řeší kontext dekódovaného znaku v rámci aktuálního slova.

Pro tento model jsem využil konfiguraci dostupnou ve složce *examples/asr/conf/citrinet* s názvem *config_bpe.yaml* dostupnou v repozitáři od NeMo[2]. Mimo této konfigurace je pro trénink tohoto modelu nutno mít vytvořen tokenizer, který se při inicializaci trénovacího skriptu připojí do konfigurace modelu.

Pro vytvoření tokenizeru jsem využil skript dostupný v repozitáři NeMo[2] ve složce *scripts/tokenizers* s názvem *process_asr_text_tokenizer.py*. Tento skript vytvoří *tokenizer* se všemi potřebnými částmi do specifikované složky. Je zde možnost vybrat si mezi Google SentencePiece (SPE) a HuggingFace BERT Word Piece (WPE) *tokenizerem*. Také se zde musí uvést parametr, kolik slov bude ve vytvořeném slovníku pro daný *tokenizer*.

Pro moji implementaci jsem zvolil *tokenizer* typu WPE. Dále jsem vytvořil skript pro trénování modelů typu Citrinet a ověřil jsem funkčnost skriptu. Model jsem natrénoval na 100 a poté na 300 epoch. Pro trénování jsem opět využil datasetu *LibriSpeech* a jeho částí, podobně jako u modelu Quartznet.

Model	test-clean	ATC	test-other	dev-clean	dev-other
Citrinet 100 epoch	67 %	102 %	83 %	67 %	83 %
Citrinet 300 epoch	61 %	104 %	81 %	61 %	80 %

Tabulka 4.4: Hodnoty WER pro Citrinet model s tokenizerem WPE

V tabulce 4.4 se nachází hodnoty WER pro různé počty trénovacích epoch. Tyto hodnoty je možno zlepšit větším slovníkem tokenizeru nebo delším časem trénování.

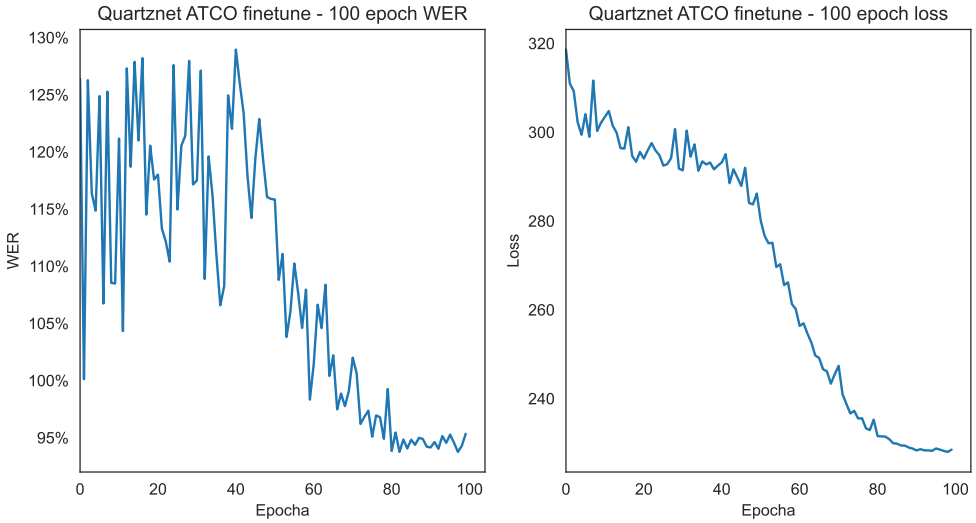
Trénování modelu Quartznet na ATC datech

Pro vytvoření modelu fungujícího na ATC datech jsem použil již natrénovaný model typu Quartznet pro anglický jazyk. Do tohoto modelu jsem přidal trénovací ATC data a následně jej vyhodnotil.

[2]Github repozitář NeMo: https://github.com/NVIDIA/NeMo

Zde bylo opět nutno řešit formát datasetu, který jsem musel převést do formátu pro NeMo ASR. Mimo to jsem také musel dataset rozdělit na trénovací a validační část. Trénovací část má délku nahrávek přibližně 3 hodiny a validační přibližně 1 hodinu, dataset obsahuje celkem 4 hodiny ATC nahrávek.

Po trénování modelu po dobu 100 epoch je možno vidět zlepšení, ale WER (Word Error Rate) je stále velmi vysoký. V grafu 4.3 je možno vidět hodnotu WER a loss během fine-tuningu na ATCO datasetu.



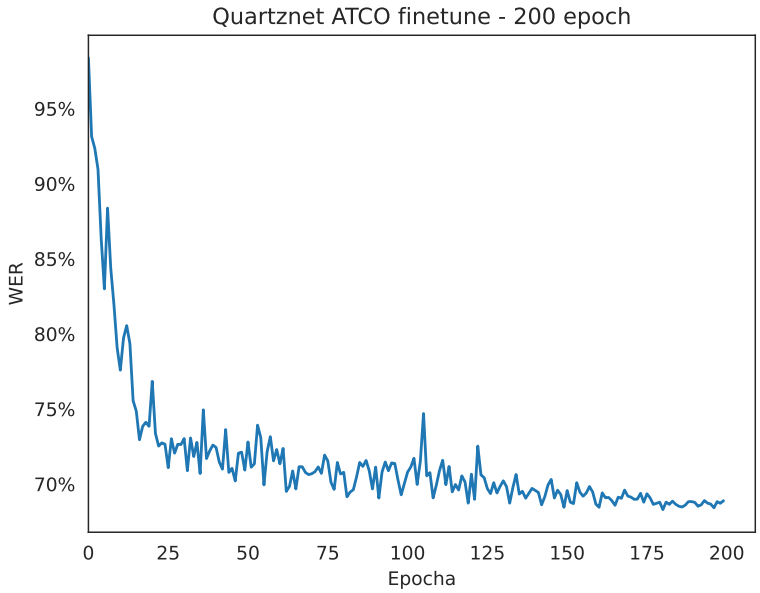
Obrázek 4.3: Vlevo: WER během fine-tuningu Quartznet modelu na ATC datech po dobu 100 epoch. Vpravo: Hodnota loss funkce během fine-tuningu Quartznet modelu na ATC datech po dobu 100 epoch.

V grafu 4.4 je zobrazeno, jak se měnila hodnota WER během fine-tuningu na ATCO datasetu po dobu 200 epoch.

Model	**ATC dataset WER**
Quartznet ATC 100 epoch	95 %
Quartznet ATC 200 epoch	69 %

Tabulka 4.5: WER pro Quartznet model trénovaný na ATC datech

V tabulce 4.5 je výpis WER po jednotlivé počty trénovaných epoch. Tento WER je možno dále zlepšovat trénováním po více epoch, také je možno vytvořit například Citrinet model, který bude mít dobrý WER na angličtině a zkusit natrénovat tento model na ATC datech.



Obrázek 4.4: WER během fine-tuningu Quartznet modelu na ATC datech po dobu 200 epoch

4.4 Souhrn modelů

V nástroji NeMo jsem se naučil vytvářet vlastní modely strojového učení pro angličtinu a češtinu. Získal jsem znalosti, jak se NeMo používá pro vytvoření a ukládání modelů strojového učení a jaké problémy vytváření modelů přináší. Vytvořil jsem modely typu Quartznet a Citrinet, které jsem následně vyhodnotil. Pro jejich trénování jsem použil datasety LibriSpeech pro angličtinu a Common Voice pro češtinu. Vytvořil jsem také vlastní skripty pro převod Common Voice datasetu do formátu, který specifikuje NeMo, aby se na datech mohlo trénovat. Získaný WER není nejlepší, ale pochopitelný vzhledem k omezením hardwaru, na kterém jsem trénoval. WER jednotlivých natrénovaných modelů je uveden v tabulce 4.6.

Model	**test-clean**	**ATC**	**test-other**	**dev-clean**	**dev-other**
Quartznet 60 epoch	42 %	104 %	68 %	42 %	67 %
Quartznet 240 epoch	39 %	104 %	66 %	38 %	65 %
Quartznet 600 epoch	39 %	107 %	66 %	38 %	65 %
Citrinet 100 epoch	24 %	72 %	39 %	24 %	39 %
Citrinet 300 epoch	21 %	72 %	37 %	21 %	36 %
Finetune ATC 100 epoch	99 %	95 %	99 %	99 %	99 %
Finetune ATC 200 epoch	99 %	69 %	99 %	99 %	99 %

Tabulka 4.6: WER NeMo modelů trénovaných na angličtině a modelů s fine-tuningem na ATCO datasetu

Kapitola 5

Automatický rozpoznávač řeči Whisper

V této kapitole popíši automatický rozpoznávač řeči Whisper. Dále také popisuji, jak jsem Whisper využil v mojí práci, jak je možno provést jeho fine-tuning a dosažené výsledky po jeho fine-tuningu.

Whisper od společnosti OpenAI je vysoce výkonný model pro automatické rozpoznávání řeči, který byl natrénován na 680 000 hodinách vícejazyčného audia. Tento model je možno použít mimo ASR (Automatic Speech Recognition) také k překládání textu do různých jazyků. Whisper má velmi nízký WER (Word Error Rate) hlavně na anglickém jazyce, ale skvělé výsledky ukazuje také na různých dalších jazycích včetně českého jazyka.

Whisper je vytvořen pomocí encoder-decoder Transformer modelu pro rozpoznávání řeči. Vzorkování vstupních dat je 16 kHz a mel-spektrogramy se generují po 25 milisekundách audia s krokem 10 milisekund. Vstupní signál je normalizován mezi hodnoty -1 a 1. Aktivační funkcí modelu je GELU. Mimo přepisů Whisper do textu přidává také řídící tokeny, které označují start přepisu, zda se jedná o přepis či překlad, identifikaci jazyka, zda má model rozlišovat časové značky v textu a konec přepisu [15].

Whisper je dostupný jako modul do jazyka Python a byl vytvořen pomocí knihovny PyTorch [14].

Tento modul je možno spouštět buď přímo z terminálu, nebo jej lze spustit ve vlastním Python skriptu. Whsiper má API (Application Programming Interface), přes které je možno jej ovládat.

Já jsem Whisper na mém systému zprovoznil a vyhodnotil jsem jeho úspěšnost na ATCO2 datasetu, vyhodnocení na ostatních datasetech provedli již autoři. Získané hodnoty jsou v tabulce 5.1, první dva modely Quartznet a Citrinet jsou bez fine-tuningu a byly navíc trénovány na čistých datech, což vysvětluje jejich vysoký WER. Model Quartznet ATC má již dotrénovaná data letecké komunikace a WER má již o něco lepší. Whisper si se základním modelem vede velmi dobře, proto se zaměřuji na jeho fine-tuning.

Model	WER	Počet parametrů	Délka trénovacích dat
Quartznet – 600 epoch	107 %	1,2M	100 hodin
Citrinet – 300 epoch	104 %	1,2M	100 hodin
Whisper – medium	**78 %**	769M	680 000 hodin
Quartznet ATC – 200 epoch	69 %	1,2M	4 hodiny

Tabulka 5.1: WER natrénovaných modelů NeMo a Whisperu na ATC datasetu

Součástí této práce je také fine-tuning různých Whisper modelů. Vzhledem k velkému množství různých dat, na kterých byl natrénován, by mohl také velmi dobře fungovat pro účel přepisu této velmi zašuměnné komunikace.

5.1 Fine-tuning modelů rozpoznávače Whisper

Automatický rozpoznávač řeči Whisper disponuje velkým množstvím různě velkých modelů určených pro různé jazyky. V mojí práci se zaměřuji na fine-tuning nejen v oblasti dat letecké komunikace, ale také vytvoření bilinguálních modelů pro rozpoznávání textu. Pro tyto účely bych mohl Whisper využít, vzhledem k jeho skvělým výsledkům na standardních datasetech, proto jsem se rozhodl provést jeho fine-tuning.

Pro tyto účely jsem hledal, jak je možné provést fine-tuning Whisper modelu. Žádný oficiální skript jsem pro tento účel nenašel. Využil jsem tedy skript vytvořený komunitou, který se zaměřuje na fine-tuning Whisperu na japonštině[1]. Pro fine-tuning pomocí tohoto Python notebooku je třeba dodržet jeho formát souborů a složek. Pro každý audio soubor stačí vytvořit samostatnou podsložku, která obsahuje daný soubor ve formátu `wav` a soubor `transcripts_utf8.txt`, který obsahuje přepis textu obsaženého v přiloženém `wav` souboru ve formátu: `<název souboru>:<přepis textu>` oddělené jednotlivými řádky, pokud se ve složce nachází více audio souborů.
Struktura složek datasetu může vypadat například takto:

```
/
├── 1/
│   ├── transcripts_utf8.txt
│   └── 1.wav
├── 2/
│   ├── transcripts_utf8.txt
│   └── 2.wav
└── ⋮
```

Poté stačí spouštět jednotlivé buňky trénovacího Python notebooku, který pomocí data loaderu od PyTorch dataset načte a spustí trénink zvoleného modelu. Zde je také nastaveno, aby se po každé epoše uložil stav modelu do checkpoint souboru. Python notebook jsem využil pro rychlé testování, zda mám dataset ve správném formátu, z tohoto Python notebooku jsem vytvořil Python skript, abych nemusel všechny buňky neustále procházet manuálně.

Pro fine-tuning Whisper modelů jsem použil datasety Mozilla Common Voice, konkrétně datasety pro angličtinu a češtinu. Pro angličtinu jsem použil Common Voice 1, který má 583 hodin audia. Pro češtinu jsem použil Common Voice 13, který má 73 hodin audia. Tyto datasety jsem po odzkoušení fine-tuningu na jednom jazyce spojil a zkusil jsem vytvořit bilinguální model Whisperu. Pro fine-tuning na datech letecké komunikace jsem využil dataset ATCO2 [20].

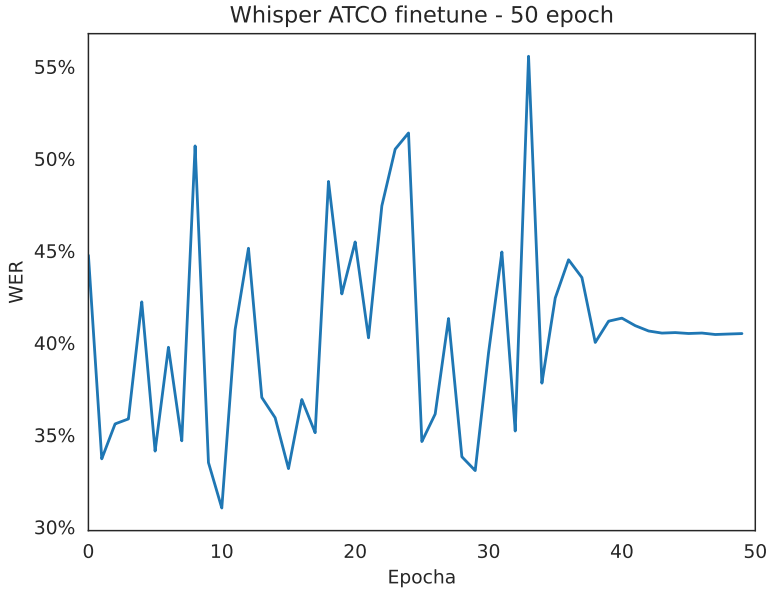
[1] Python notebook dostupný z:
https://colab.research.google.com/drive/1P4ClLkPmfsaKn2tBbRp0nVjGMRKR-EWz?usp=sharing
(navštíveno 8. 4. 2023)

5.2 Fine-tuning Whisper modelů na ATCO datasetu

Po otestování a drobných úpravách trénovacího kódu jsem začal s vlastními experimenty. Ze začátku jsem pracoval s `base` modelem. Tento model obsahuje 74 miliónů parametrů. U base modelu bylo trénování jedné epochy docela rychlé, proto jsem na něm začal vytvářet moje experimenty. Tento model má ale nevýhodu, že je řádově menší, než `large` model, large model má 1,5 miliardy parametrů. Zároveň je ale base model lepší pro ladění trénování.

Po odladění trénovacího skriptu jsem trénoval medium model, který je větší měl by tedy dosahovat lepších výsledků.

Začal jsem tedy s fine-tuningem `base` modelu. Pro ověření, že fine-tuning funguje správně, jsem zkusil trénovat anglický model na ATCO datasetu. Nejprve bylo třeba tento dataset převést na formát, který vyžaduje trénovací skript Whisperu. K tomuto účelu jsem si vytvořil vlastní Python skript, který tuto konverzi provede. Tomuto skriptu stačí předat parametry s cestou k ATCO datasetu a cestou, kam uložit převedený dataset. Skript zkontroluje, zda složka s ATCO datasetem existuje, následně čte postupně audio soubory s nahrávkami letecké komunikace. Při načtení audio souboru ve formátu *wav* se přečte soubor ve formátu *xml* se stejným názvem, kde se nalezne element s textem dané nahrávky. Tento text se zapíše do souboru `transcripts_utf8.txt` vedle zkopírovaného souboru s nahrávkou. Skript tedy automaticky vytvoří strukturu složek, kterou trénovací skript vyžaduje. Dále stačí spustit trénovací skript s cestou k převedenému datasetu a nechat jej provést fine-tuning.



Obrázek 5.1: WER během trénování base modelu na ATCO2 datasetu.

Po fine-tuningu na ATCO datasetu jsem vytvořený model vyhodnotil a dostal hodnoty WER uvedené v tabulce 5.2, WER během trénování se nachází v grafu 5.1.

Počet epoch	Word Error Rate
0 (základní model)	115 %
10	47 %
50	42 %

Tabulka 5.2: WER Whisper base modelu po fine-tuningu na ATCO datasetu. 0 epoch je model bez fine-tuningu.

Zde je vidět velké zlepšení oproti výsledku bez fine-tuningu. Fine-tuning větších modelů by tedy mohl dosáhnout mnohem lepších výsledků, než `base` model, na kterém trénovací skript testuji. Takto vysoký WER je způsoben malým počtem parametrů base modelu. Base model je dobrý pro běh na systémech s malou pamětí VRAM, ale nelze od něj očekávat perfektní výsledky.

Po experimentech s base modelem jsem se rozhodl pro fine-tuning `medium` modelu. Medium model má mnohem více parametrů, než base model, předpokládal jsem tedy, že WER bude lepší než u base modelu. Výsledky medium modelu na ATC datech jsou uvedeny v tabulce 5.3. Přepisy modelů bez fine-tuningu ukázaly, že Whisper má na těchto datech

Počet epoch	Word Error Rate
0 (základní model)	78 %
1	30 %
10	24 %

Tabulka 5.3: WER `medium` modelu po fine-tuningu na ATC datasetu. 0 epoch je model bez fine-tuningu.

problémy se šumem. Přepisoval totiž mnoho slov špatně a většinou nebyla výslovnost vůbec podobná původnímu slovu. Medium model s fine-tuningem se výrazně zlepšil a to o 49 % za jednu epochu trénování. Toto zlepšení je dobře vidět v přepsaných datech, kde je již většina nahrávek přepsána téměř korektně. Whisper si dokonce poradil i s trochou češtiny, která se na konci některých nahrávek nacházela. Whisper je navíc trénován na čistých datech, je tedy skvělé, že i s tak malým datasetem dokázal WER takto rapidně snížit.

Kapitola 6

Fine-tuning bilinguálního modelu Whisperu

V této kapitole popisuji, jak jsem vytvořil bilinguální model rozpoznávače Whisper, jaká data jsem pro tento účel použil a jakých výsledků tento model dosáhl.

Ze začátku jsem vyhodnotil český a anglický model na Common Voice datasetu se stejným jazykem, jako je jazyk vyhodnocovaného modelu. Poté jsem provedl fine-tuning po dobu jedné epochy u každého modelu na daném datasetu. Tím je myšleno, že anglický Whisper model jsem vyhodnotil a trénoval na anglickém Common Voice datasetu, český model jsem vyhodnotil a natrénoval na českém datasetu.

Jazyk modelu	**WER před fine-tuningem**	**WER po fine-tuningu**
angličtina	15 %	11 %
čeština	72 %	47 %

Tabulka 6.1: Word Error Rate (WER) modelů Whisper před fine-tuningem a po fine-tuningu na anglickém a českém datasetu Common Voice

Jak je možno v tabulce 6.1 vidět, dosažený WER je podobný výsledkům testování autory Whisper modelů na datasetu Common Voice [15]. Po fine-tuningu je vidět zlepšení ve WER, tímto jsem si ověřil, že skript pro fine-tuning funguje správně.

Cílem bilinguálního modelu, vytvořeného pomocí Whisperu, je mít model vycházející z jednoho předtrénovaného modelu Whisperu, který bude schopen přepisovat audio soubory, které obsahují řeč ve dvou různých jazycích zároveň. Tento vytvořený model by měl být schopen například přepsat komunikaci, ve které se mluví anglicky, ale obsahuje název českého města, který by čistě anglický model nejspíše přepsal částečně, nebo úplně špatně.

Abych tento model vytvořil, potřeboval jsem mít dataset, který má v každé nahrávce namíchanou angličtinu i češtinu zároveň. Pro tento účel jsem využil datasety Mozilla Common Voice v českém a anglickém jazyce. Vytvořil jsem k tomuto účelu vlastní Python skript, který čte soubory *train.tsv* a *test.tsv* z každého datasetu. Pro každý záznam v souboru *train.tsv* načte pomocí knihovny Pydub[1] třídy *AudioSegment* soubor nahrávky ve formátu *mp3*, pro každý jazyk. Záznam ze souboru *train.tsv* je načten jako slovník (dictionary) obsahující název souboru, přepis textu a různá další data o daném souboru. Pomocí třídy *AudioSegment* tyto dva soubory spojí a exportuje ve formátu *wav* do hierarchie složek popsané v kapitole 5.1. S audio souborem se do této vytvořené složky uloží i přepis textu

[1]Github repozitář Pydub: https://github.com/jiaaro/pydub

z obou souborů. Aby model lépe rozpoznal libovolný z těchto dvou jazyků, vytvořené soubory spojených nahrávek mají pořadí jazyků náhodné. Po zpracování celé trénovací části datasetu (souborů *train.tsv*) se zbývající část druhého datasetu zahodí. Následuje spojení testovacích částí obou datasetů (souborů *test.tsv*) stejným způsobem.

Trénovací Python notebook sice obsahuje kód, který rozdělí celý načtený dataset na trénovací a testovací část, tento kód ale nepoužívám, protože není zaručeno, že testovací část datasetu bude vždy stejná. Dále jsem na takto vytvořeném datasetu natrénoval anglický a český `base` model Whisperu. Výsledky fine-tuning `base` modelu těchto dvou jazyků jsou uvedeny v tabulce 6.2, jako baseline jsem použil WER anglického a českého Whisper modelu na bilinguálním datasetu.

Jazyk modelu	**Baseline WER**	**WER po fine-tuningu**
angličtina	70 %	44 %
čeština	113 %	41 %

Tabulka 6.2: WER `base` modelů Whisper na bilinguálním datasetu po jedné epoše fine-tuningu

S anglickým `base` modelem před fine-tuningem je možno vidět, že WER je docela vysoký. To je způsobeno přepisem pouze anglické části textu v audio souboru. Pokud je na začátku souboru čeština, model píše nesmyly a tím se zvyšuje WER. Někdy se také stane, že model začne opakovat stejný text několikrát za sebou, což má za následek zvýšení WER (někdy i nad 100 %).

Po provedení fine-tuningu je možno pozorovat velký skok ve WER. Tento skok nastal pouze po jedné epoše fine-tuningu na tomto datasetu, což Whisperu dává velký potenciál při větším počtu epoch a správně nastavené *learning rate* konstantě. Očekávám také, že větší modely dosáhnou lepších výsledků.

Z tabulky 6.2 je také možno vidět, že český model nedosáhl bez fine-tuningu tak dobré hodnoty WER, jako model anglický. Toto se dalo očekávat, jelikož na Common Voice datasetu má český Whisper base model WER 63,1 % [15]. Po provedení fine-tuningu je ale možno sledovat výrazné zlepšení, podobně jako u anglického base modelu.

Následně jsem se rozhodl provést fine-tuning `medium` modelu Whisper. Medium model má 769 miliónů parametrů a na Common Voice datasetu dosáhl 11,2 % WER na anglickém jazyce a 18,8 % WER na českém jazyce [15]. V tabulce 6.3 jsou uvedeny hodnoty WER anglického a českého modelu získané po vyhodnocení na bilinguálním datasetu Common Voice. Baseline je WER medium modelu na tomto datasetu před fine-tuningem.

Jazyk modelu	**Baseline WER**	**WER po fine-tuningu**
angličtina	60 %	19 %
čeština	70 %	19 %

Tabulka 6.3: WER (Word Error Rate) `medium` modelů Whisperu na bilinguálním datasetu, fine-tuning byl proveden po dobu 1 epochy

Vzhledem k velikosti modelu jsou tyto hodnoty lepší, než u `base` modelu, hlavně v případě českého medium modelu. Na angličtině je 10% zlepšení oproti base modelu a čeština již nestoupá nad 100 % WER.

Po fine-tuningu na bilinguálním datasetu jsem tento model vyhodnotil na původních Common Voice datasetech, obsahujících buď jen angličtinu, nebo jen češtinu. Jako baseline jsem použil WER medium modelu na konkrétním datasetu bez fine-tuningu. Naměřené hodnoty WER jsou uvedeny v tabulce 6.4. Také jsem na bilinguálním datasetu vyhodnotil `large` model, který dosáhl WER 55 % bez fine-tuningu. Předpokládám, že `large` model Whisperu by mohl dosáhnout při fine-tuningu ještě lepších výsledků, vzhledem k jeho počtu parametrů (1,5 miliardy parametrů).

Model	EN WER	CZ WER	Bilingual WER
Základní medium (baseline)	14 %	122 %	60 %
Finetune medium	16 %	19 %	19 %

Tabulka 6.4: WER základního a natrénovaného medium modelu na původním anglickém a českém datasetu a na bilinguálním datasetu.

Po zhlédnutí přepisů medium modelu bez fine-tuningu je možno vidět vysvětlení, proč má model takto vyšší WER. Protože audio je náhodně míchané, tedy ne vždy je prvním jazykem čeština, se stává, že anglický model mnohdy přepisuje češtinu, jako podobně znějící anglická slova.

```
GT: Souhlas? Have you seen it?
P:  So, class? Have you seen it?
```

V tomto příkladu je možno vidět slovo „Souhlas“ přepsáno jako „So, class“, řádek GT (Ground Truth) je správný přepis textu z datasetu, P (Predicition) je přepis Whisperu.

Také jsem viděl pokus anglického modelu o překlad češtiny, i když nebyl v režimu překladu, ale v režimu přepisu.

```
GT: Měl se vrátit včera. The quickening of time
P:  He was supposed to be back yesterday.
```

V tomto případě se Whisper rozhodl spíše přeložit českou část nahrávky a anglickou část vypustil.

Nejčastěji se však stávalo, že model přepsal pouze anglickou část textu a českou úplně ignoroval. Toto se stalo hlavně, pokud byla angličtina na začátku audio nahrávky.

```
GT: How do you know my name? V herní krabici je jen mapa.
P:  How do you know my name?
```

Zde je anglická část nahrávky bezchybně přepsaná, ale česká Whisper vůbec nezajímá. Tyto ukázky přepisů jsou z vyhodnoceného medium modelu bez fine-tuningu. Dále zde přikládám ukázku přepisu medium modelu po fine-tuningu.

```
GT: There's that reprieve if they ever find out. Co bych dělal?
P:  There's that reprieve if they ever find out. Co bych dělal?
```

Například tato nahrávka je přepsána celá správně. Pokud na začátku nahrávky byla angličtina, anglickému modelu to většinou nedělalo problémy. Po provedení fine-tuningu ale anglický model již neignoruje českou část nahrávky.

```
GT: Je třeba se pokusit nalézt autora a chránit jeho práva.
Mr. Lee can't be bothered now.
P:  Je třeba se pokusit nalézt autora a chránit jeho práva.
Mr. Lee can't be bothered now.
```

V případě nechtěných překladů se Whisper také zlepšil.

```
GT: Měl se vrátit včera. The quickening of time
P:  Měl se vrátit včera. The beginning of time
```

Většina chyb po provedení fine-tuningu je nahrazení nějakého slova podobně znějícím slovem. Občas také může nastat chyba v interpunkci.

Český model před fine-tuningem měl podobný problém jako anglický, přepisoval pouze českou část nahrávky a anglickou úplně vynechal.

```
GT: Téměř každý souhlasil s účastí na Schengenu.
A hundred nuns stampeded the Vatican.
P:  Téměř každý souhlasil s účastí na Schengenu.
```

Po fine-tuningu se Whisper naučil přepisovat i anglické části nahrávek.

6.1 Zhodnocení experimentů

Rozpoznávač Whisper jsem prvně vyhodnotil na ATCO2 datasetu, konkrétně jeho medium model. Toto vyhodnocení ukázalo dobré výsledky pro model, jenž byl natrénován pouze na čistých datech, konkrétně 78 % WER medium modelu. Rozhodl jsem se tedy zkusit experimentovat s různými Whisper modely.

Ze začátku jsem ladil trénovací skript, využíval jsem tedy k trénování base model, který má 74 miliónů parametrů a fine-tuning netrvá příliš dlouho. Hodnota WER base modelu na ATCO2 datasetu byla 115 % před provedením fine-tuningu. Po fine-tuningu bylo vidět výrazné zlepšení při vyhodnocení na ATCO2 datasetu, WER klesl na hodnoru 42 % po 50-ti epochách trénování, oproti 115 % WER bez fine-tuningu. Zkusil jsem tedy provést fine-tuning medium modelu (769 miliónů parametrů), ten dosáhl ještě lepších výsledků, oproti base modelu. Před fine-tuningem dosáhl 78 % WER na ATCO2 datasetu, po fine-tuningu WER klesl na 24 % po 10 epochách trénování. Úspěšně jsem tedy vytvořil rozpoznávač letecké komunikace.

Další částí bylo vytvoření bilinguálního modelu rozpoznávače Whisper. Pro tento model jsem vytvořil dataset, který obsahuje v jedné nahrávce dva jazyky, konkrétně angličtinu a češtinu. Toho jsem dosáhl spojením anglického a českého datasetu Common Voice. Při spojování jednotlivých nahrávek jsem náhodně vybíral, zda bude první angličtina, či čeština. Na takto vytvořeném datasetu jsem vyhodnotil a poté přetrénoval base model Whisperu. Před fine-tuningem dosáhl Anglický base model WER 70 %, český model získal 113 % WER. Po přetrénování se ukázalo velké zlepšení oproti původnímu WER, po fine-tuningu dosáhl anglický base model WER 44 % a český base model 41 %. Zkusil jsem také vyhodnotit medium model, který dosáhl ještě lepších výsledků. Před fine-tuningem získal anglický medium model 60 % WER a český medium model získal 70 % WER. Po fine-tuningu WER klesl u obou modelů (anglického i českého) na 19 %. Takto jsem vytvořil bilinguální rozpoznávač z jednoho modelu rozpoznávače Whisper.

Kapitola 7

Závěr

V této práci jsem vytvořil vlastní modely pro rozpoznávání řeči nástrojem NeMo. NeMo modely jsem trénoval na dvou různých datasetech. Pro angličtinu jsem zvolil dataset LibriSpeech, pro češtinu jsem použil dataset Common Voice. Tyto modely měly vyšší WER (Word Error Rate), než jsem očekával, ale vyzkoušel jsem si způsob, jakým se tyto modely vytváří.

Dále jsem provedl fine-tuning jednoho takto natrénovaného NeMo modelu na datasetu ATCO2. Z tohoto fine-tuningu jsem získal mnohem lepší výsledky, než tyto modely původně na tomto datasetu dosahovaly.

Po fine-tuningu NeMo modelů jsem na ATCO2 datasetu vyhodnotil Whisper, který dosáhl 78 % WER bez fine-tuningu. Přetrénoval jsem tedy base model Whisperu na ATCO2 datasetu a získal WER 47 % po 10-ti epochách fine-tuningu. Po zvýšení počtu epoch na 50 jsem získal WER 42 %.

Také jsem zkusil provést fine-tuning většího medium modelu Whisperu. Bez fine-tuningu jsem tímto modelem dosáhl již zmiňovaných 78 % WER. Po fine-tuningu medium modelu jsem jej vyhodnotil a získal WER 30 % po jedné trénovací epoše. Při zvýšení počtu trénovacích epoch na 10 jsem získal WER 24 %.

Mimo fine-tuningu Whisper modelů na datasetu ATCO2 jsem vytvořil dvojjazyčný dataset spojením anglického a českého Common Voice datasetu. Pro odzkoušení jsem prvně vyhodnotil a přetrénoval base modely Whisperu na takto vytvořeném datasetu. Anglický model dosáhl před fine-tuningem 70 % WER a po fine-tuningu 44 % WER. Český model Whisperu získal 113 % WER před fine-tuningem a zlepšil si WER na 41 % po fine-tuningu.

Zkusil jsem tedy to stejné s medium modelem Whisperu. Po vyhodnocení dosáhl anglický model 60 % WER a český model 70 % WER. Po fine-tuningu dosáhly oba modely hodnoty 19 % WER.

Po těchto experimentech jsem dosáhl dobré hodnoty WER, jak na datech letecké komunikace, tak na dvojjazyčném datasetu. Další možností vývoje je zkusit provést fine-tuning large modelu Whisperu na obou datasetech. Dále by bylo vhodné vytvořit nějaké grafické uživatelské rozhraní, které by zjednodušilo použití těchto modelů, zatím je lze použít, ale pouze přes příkazovou řádku. Dalším zajímavým rozšířením by bylo použití těchto modelů pro přepis živě zachytávané letecké komunikace.

Touto prací jsem si rozšířil znalosti v oblasti strojového učení a automatického přepisu textu, prakticky jsem si vyzkoušel použití takovýchto nástrojů a zjistil jsem, jaké překážky přináší fine-tuning těchto modelů.

Literatura

[1] Alammar, J. *The Illustrated Transformer* [online], 27. června 2018. Citováno: 28. 4. 2023. Dostupné z: http://jalammar.github.io/illustrated-transformer/.

[2] Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M. et al. Common Voice: A Massively-Multilingual Speech Corpus. *CoRR*. 2019, abs/1912.06670. DOI: 10.48550/arXiv.1912.06670. Dostupné z: http://arxiv.org/abs/1912.06670.

[3] Doshi, K. *Audio Deep Learning Made Simple: Automatic Speech Recognition (ASR), How it Works* [online], 25. března 2021. Citováno: 18. 4. 2023. Dostupné z: https://towardsdatascience.com/audio-deep-learning-made-simple-automatic-speech-recognition-asr-how-it-works-716cfce4c706.

[4] Fabien, M. *Introduction to Automatic Speech Recognition (ASR)* [online]. Citováno 12. 1. 2023. Dostupné z: https://maelfabien.github.io/machinelearning/speech_reco/.

[5] Hardesty, L. *Explained: Neural networks: Ballyhooed artificial-intelligence technique known as "deep learning" revives 70-year-old idea.* [online]. Massachusetts Institute of Technology, 14. dubna 2017. Citováno 12. 1. 2023. Dostupné z: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[6] IBM. *What is recurrent neural networks?* [online]. Dostupné z: https://www.ibm.com/topics/recurrent-neural-networks.

[7] Kosar, V. *Tokenization in Machine Learning Explained* [online], 16. září 2022. Citováno: 26. 4. 2023. Dostupné z: https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained.

[8] Kriman, S., Beliaev, S., Ginsburg, B., Huang, J., Kuchaiev, O. et al. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, s. 6124–6128. DOI: 10.48550/arXiv.1904.03288.

[9] Kuchaiev, O., Li, J., Nguyen, H., Hrinchuk, O., Leary, R. et al. Nemo: a toolkit for building ai applications using neural modules. *ArXiv preprint arXiv:1909.09577*. Září 2019. DOI: 10.48550/ARXIV.1909.09577.

[10] Liu, D. *A Practical Guide to ReLU* [online], 30. listopadu 2017. Citováno: 14. 4. 2023. Dostupné z: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

[11] Mishra, M. *Convolutional Neural Networks, Explained* [online], 26. srpna 2020. Citováno: 17. 4. 2023. Dostupné z: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[12] Niklas, D. *A Brief History of ASR: Automatic Speech Recognition* [online], 31. března 2019. Citováno: 29. 4. 2023. Dostupné z: https://towardsdatascience.com/a-brief-history-of-asr-automatic-speech-recognition-95de6c014187.

[13] Panayotov, V., Chen, G., Povey, D. a Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, s. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*. 2019, sv. 32.

[15] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C. et al. Robust speech recognition via large-scale weak supervision. *ArXiv preprint arXiv:2212.04356*. 2022. DOI: 10.48550/arXiv.2212.04356.

[16] Sable, A. *End to End Automatic Speech Recognition: Introduction* [online]. Citováno 12. 1. 2023. Dostupné z: https://blog.paperspace.com/end-to-end-automatic-speech-recognition/.

[17] Sharma, S. Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED !!! *Activation Functions in Neural Networks* [online], 6. září 2017. Citováno: 14. 4. 2023. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[18] Sharma, S. The Fundamentals of Neural Networks. *What the Hell is Perceptron?* [online], 9. září 2017. Citováno: 14. 4. 2023. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[19] Sharvil. *Tokenization Algorithms Explained* [online], 3. srpna 2021. Citováno: 26. 4. 2023. Dostupné z: https://towardsdatascience.com/tokenization-algorithms-explained-e25d5f4322ac.

[20] Zuluaga Gomez, J., Veselý, K., Szöke, I., Motlicek, P., Kocour, M. et al. *ATCO2 corpus: A Large-Scale Dataset for Research on Automatic Speech Recognition and Natural Language Understanding of Air Traffic Control Communications*. 2022. DOI: 10.48550/arXiv.2211.04054.

Příloha A

Obsah přiloženého paměťového média

```
/
├── graphs/ ..... Notebook použitý pro vytvoření grafů
├── latex/ ..... Zdrojové soubory textu práce
├── NeMo/ ..... Programové řešení nástrojem NeMo
│   ├── artifacts/ ..... Natrénované NeMo modely
│   ├── configs/ ..... Konfigurace NeMo modelů
│   ├── scripts/ ..... Skripty pro trénování, vyhodnocení a použití NeMo modelů
│   ├── tokenizers/ ..... Tokenizery pro Citrinet modely
│   └── nemo.yml/ ..... Soubor pro vytvoření Conda environmentu pro NeMo
├── Whisper/ ..... Programové řešení nástrojem Whisper
│   ├── artifacts/ ..... Natrénované Whisper modely
│   ├── scripts/ ..... Skripty pro trénování a použití Whisper modelů
│   ├── transcriptions/ ..... Přepisy bilinguálního datasetu Whisper modely
│   └── whisper.yml ..... Soubor Conda environmentu pro použití Whisper skriptů
├── README.md ..... Návod k použití
├── video.mp4 ..... Video použití rozpoznávače a ukázka přepisů
├── plakat.pdf ..... Plakát k této práci
├── text.pdf ..... Text této bakalářské práce
└── text_print.pdf ..... Vytištěný text
```