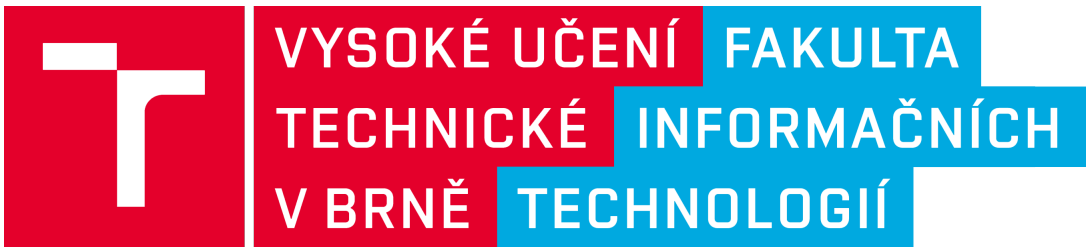


Strojové učení a rozpoznávání

Dokumentace k projektu

Jan Zdeněk (xzdene01)
Michal Bartošák (xbarto0d)

4. května 2025

1 Audio

Tato sekce je věnována řešení identifikace řečníka z audio nahrávek. Bylo využito datasetu, ketrý obsahuje 31 řečníků, kde ke každému bylo k dispozici 6 trénovacích nahrávek a 2 validační nahrávky.

1.1 Popis řešení

Pro účely toho projektu byla, z důvodu malého množství originálních nahrávek, použita **augmentace dat**. Augmentace by měla zvýšit celkovou robustnost systému. Jako metody **extrakce příznaků** byly zvoleny **MFCC** a **fBanks**, které patří ke standardním technikám v oblasti zpracování lidské řeči. Klasifikace řečníka využívá metodu **GMM-UBM** (Gaussian Mixture Model - Universal Background Model) a MAP (Maximum A Posteriori) adaptací.

Obecný popis metody GMM-UBM s MAP adaptací

Tato metoda staví na dvou krocích, nejprve je potřeba natrénovat UBM na celém trénovací korpusu, který reprezentuje obecné akustické vlastnosti řeči. UBM lze chápat jako směsici Gaussovských komponent s parametry $\{\mu_i^{UBM}, \Sigma_i^{UBM}, w_i^{UBM}\}$, které jsou odhadnuty pomocí EM algoritmu. Pro jednotlivé řečníky z trénovací množiny se poté provede MAP adaptace, kdy se původní parametry UBM upraví tak, aby lépe odpovídali datům konkrétního řečníka. Tento způsob řešení je blíže popsán například ve článku *Speaker Verification Using Adapted Gaussian Mixture Models* [1]. Obecné vztahy pro MAP při adaptaci pouze středních hodnot[1] jsou k dispozici ve Vzorci 1 níže.

$$\textbf{(a)} \qquad \gamma_t(i) \;=\; \Pr\big(i \mid \boldsymbol{x}_t\big)$$

$$\textbf{(b)} \qquad n_i \;=\; \textstyle\sum_{t=1}^{T} \gamma_t(i)$$

$$\textbf{(c)} \qquad \boldsymbol{E}_i \;=\; \frac{1}{n_i} \textstyle\sum_{t=1}^{T} \gamma_t(i)\, \boldsymbol{x}_t$$

$$\textbf{(d)} \qquad \hat{\mu}_i \;=\; \frac{n_i}{n_i + r}\, \boldsymbol{E}_i + \frac{r}{n_i + r}\, \mu_i^{\mathrm{UBM}}$$

Vzorec 1: Vzorce výše ukazují postup jak pomocí metody MAP adaptovat střední hodnoty komponent směsice Gaussovských modelů pro každého řečníka. V **(a)** se určí posteriorní pravděpodobnost – tedy jak moc i-tá komponenta „vysvětluje“ t-tý příznak. Hodnota n_i ve vzorci **(b)** poté odpovídá sumě přes všechna data – počet „měkkých“ přiřazení dat ke komponentě. Vzorec **(c)** značí tzv. empirický průměr – zdali považujeme n_i za „přiřazenou masu“, pak $\boldsymbol{E}_i$ označuje centroid dat přiřazených ke komponentě. Poslední vzorec **(d)** poté ukazuje samotnou adaptaci střední hodnoty i-té komponenty, kde r je faktor relevance.

Hlavními výhodami použití tohoto řešení jsou **robustnost** modelu při omezeném množství dat a celková **interpretovatelnost** modelu. Mezi hlavní nevýhodu lze požadovat **velký počet parametrů**, kde je nutné uložit $N + 1$ Gaussovských směsic pro N řečníků.

Popis zvolené metody

V tomto projektu se využívá pouze **adaptace středních hodnot**, ostatní parametry (váhy a kovariance) se pouze kopírují z UBM.[2] Tato adaptace minimalizuje tzv. *overfitting* při omezeném počtu řečníků. Tento fakt byl vyvozen z testování různých fúzí, kde prvně byla testována adaptace všech parametrů, která v kontrastu s výzkumem *Separate MAP Adaptation of GMM Parameters for Forensic Voice Comparison on Limited Data* [2] nedopadla nejlépe pro tato data a úlohu.

[1] Po provedení empirického testování byla zvolena jen tato adaptace, ale součástí implementace jsou i adaptace vah a kovariančních matic.

[2] Tohoto by se dalo využít při ukládání pro optimalizaci prostorové složitosti modelu.

Replikace výsledků

Pro replikaci výsledků tohoto řešení lze spustit trénovací skripty podle instrukcí v Sekci 1.3 – Instrukce. Zdrojové soubory jsou k dispozici ve veřejném **Git** repozitáři.[3] Pro úplnost jsou níže parametry, s kterými byl skript spouštěn – Tabulka 1, Tabulka 2 obsahuje podobný výpis, ale pro vyzkoušené způsoby extrakce příznaků.

# komponent	typ kovariance	max. iterací	faktor relevance	normalizace	adaptace
256	diagonální	300*	16.0	minmax	střední hodnoty

Tabulka 1: Výpis hodnot výše představuje nastavení hyperparametrů pro trénování GMM-UBM pro replikaci podobných/stejných výsledků.
* reálně mnohem méně; ~40

metoda	# binů	# fft transformací	délka skoku
MFCC	{20, **30**, 40}	400	160
fBank	{36, 40}	400	160

Tabulka 2: Výše jsou uvedeny všechny hodnoty a metody extrakce příznaků, které byli v průběhu testování vyzkoušeny. **Tučně** je označen způsob vedoucí na nejlepší výsledek – jak z pohledu přesnosti, tak EER (více o testování v následující sekci 1.2 – Vyhodnocování).

1.2 Vyhodnocování

Při vyhodnocování systémů pro identifikaci řečníka se běžně kombinují metriky **klasifikační** a **detekční**, mezi klasifikační se (z použitých pro tento projekt) řadí **přesnost** a **matice záměny**. Tyto metriky hodnotí, jak často systém přiřadí nahrávce odpovídajícího řečníka. Detekční metriky (EER, ROC a DET) naopak zkoumají schopnost modelu rozlišit řečníka od tzv. *impostora* při různých prahových hodnotách. Pro všechny metriky se používá skóre a predikce, výpočet těchto hodnot pro dané trénovací dato $\boldsymbol{X}$ je ukázán ve Vzorci 2.

$$\textbf{(a)} \qquad s_i(\boldsymbol{X}) = logp(\boldsymbol{X}|GMM_i) - logp(\boldsymbol{X}|UBM)$$

$$\textbf{(b)} \qquad pred(\boldsymbol{X}) = argmax_i[s_i(\boldsymbol{X})]$$

Vzorec 2: Výše je zachycen postup výpočtu všech skóre s_i **(a)** a finální predikci $pred$ **(b)** – tedy, kterému řečníkovi model přiřadil danou nahrávku.

Výše zmíněná skóre a hlavně predikce řečníka jsou dostačující pro přímý výpočet klasifikačních metrik, tedy přesnosti i matice záměny (které třídy se jak často pletou). Problém nastává při určování detekčních metrik, které jsou definovány pouze pro binární klasifikaci.[4] Jako řešení se nabízí problém více-třídní klasifikace převést na klasifikaci binární – toto lze provést pomocí převodu otázky

Jaký z 31 řečníků je na nahrávce X?

na několik (konkrétně 31) binárních otázek typu

Je na nahrávce X řečník T?

‚kde $T \in \{1..31\}$. Tímto způsobem lze vytvořit „pozitivní“ a „negativní“ tzv. *trialy* (31 *trialů* pro každou nahrávku, kde vždy jeden z nich je „pozitivní“).

[3] `https://github.com/xzdene01/SUR`

[4] Lze spočíst i DET/ROC křivku pro vice-třídní úlohy, kde práh má podobu hyperplochy. Tento způsob nebyl v rámci aktuálního řešení brán v potaz.

Vyhodnocování bylo provedeno zejména z pohledu extrakce příznaků, protože jiná nastavení pro samotné trénování GMM-UBM vykazovala znatelně horší výsledky, tímto je myšleno testování různých kombinací/fúzí adaptací, změna parametru relevance a různé normalizace. Jiné než diagonální kovariační matice taktéž nebyly testovány a to z důvodu vysokého nároku na výpočetní prostředky. Všechny výsledky testování jsou zaneseny do Tabulky 3 níže. Ukázky DET a ROC křivek pro nejlépe natrénovaný systém lze vidět na Obrázcích 3a a 3b.

metoda	# binů	přesnost	EER
MFCC	20	.871	.037
	30	**.889**	**.024**
	40	.858	.026
fBank	26	.817	.035
	40	.844	.029

Tabulka 3: Tabulka obsahuje výsledky všech provedených testů pro různé metody extrakce příznaků. Jako výsledek byla zvolena nejlepší hodnota z několika běhů.

1.3 Instrukce

Trénování celého zmíněného systému probíhá v několika následujících krocích:

- **Instalace** – pro správné fungování celé implementace je potřeba mít lokálně instalovanou aplikaci conda a aktivované prostředí, které lze vytvořit pomocí definičního souboru prostředí `env.yaml`.
- **Příprava složek** – trénovací data je potřeba umístit do složky `data/original/{train, dev}`. Je potřeba aby byla zachována stejná struktura jako u zadání tohoto projektu, tedy aby jednotlivé nahrávky byly umístěny do podsložek nesoucí jména jednotlivých tříd.

Dále následuje už jen sekvence příkazů, které je nutno provést pro natrénování systému, pro bližší informace je vhodné nahlédnout do poskytnutých skriptů. Výsledky trénování i modely lze najít v adresáři `models/gmm`.

- **Augmentace dat**
  ```
  python preprocess/augment.py -i data/original -o data/augmented
  ```
- **Extrakce příznaků** `bash feats_all.sh`
- **Trénování** `bash gmms_all.sh`

1.4 Externí nástroje

Výpis všech použitých nástrojů lze najít v souboru `env.yaml`, který lze zároveň použít pro vytvoření *conda* prostředí. Celá implementace pro audio byla napsána v jazyce *python* s použitím knihoven jako například *numpy*, *scipy* nebo *scikit-learn*.

1.5 Další vyzkoušené techniky

Mimo již zmíněné metody GMM-UBM + MAP adaptace byla vyzkoušena i řada dalších (neuronových) technik. Příkladem takových technik jsou konvoluční neuronové sítě (dále jen CNN) jako například TDNN nebo jednodušší CNN, které implementace byla ponechána ve zdrojových souborech pro demonstraci.

Tento způsob byl inspirován článkem *Towards Speaker Identification with Minimal Dataset and Constrained Resources using 1D-Convolution Neural Network* [3] a využívá jak konvolučních vrstev, tak reziduálních spojení. Mimo komplikovaný tzv. *backend* tato CNN obsahuje i relativně velký tzv. *frontend*, který obsahuje klasické plně propojené vrstvy. Tato síť však nedosáhla žádných rozumných výsledků (maximální přesnost ~50 % – absolutně nepoužitelné) a je zde zmíněna jen jako příklad.

2 Obraz

Následující sekce popisuje postup vytvoření modelu pro identifikaci 31 osob na základě obrazových dat. Pro každou osobu bylo k dispozici 8 fotografií pořízených během čtyř samostatných sezení. Tyto obrázky pak byly rozděleny na trénovací a validační data v poměru 3:1 – dvě fotografie z jednoho sezení, přičemž tři sezení byla použita pro trénování a jedno pro validaci.

2.1 Popis řešení

Pro identifikaci 31 osob na základě obrázků byla v knihovně PyTorch v rámci programovacího jazyka Python implementována malá konvoluční neuronová síť (CNN). Vzhledem k potřebě velkého množství trénovacích dat pro efektivní učení sítě bylo nutné původní dataset výrazně rozšířit pomocí augmentace dat. K augmentaci byla využita knihovna **albumentations**, která umožňuje aplikovat více transformací současně (každou s různou pravděpodobností) v rámci jedné augmentační pipeline. Tato pipeline zahrnovala geometrické transformace (rotace, horizontální převrácení, afinní transformace, oříznutí), barevné úpravy (změna světlosti, gammy, saturace, převod na odstíny šedi), přidání šumu a rozmazání. Augmentace pak byla realizována ve skriptu *image_augmentations.py*, který načetl trénovací i validační sadu a na každý obrázek aplikoval 25 různých augmentací. Vzniklá datová sada byla následně uložena do nové složky **augmented_image_data**, přičemž byla zachována původní struktura adresáře.

Po provedení augmentace dat následovalo vytvoření architektury samotného modelu konvoluční neuronové sítě (CNN). První návrh obsahoval pouze 2 konvoluční vrstvy (konvoluce, ReLU aktivace a 2×2 max pooling) následované dvěma plně propojenými vrstvami. Tento návrh však neposkytoval dostatečný výkon, jelikož přesnost na validační sadě dosahovala pouze přibližně 37 %. Z tohoto důvodu se do modelu začaly přidávat další konvoluční vrstvy, aby síť měla větší možnost extrahovat relevantní příznaky. Po přidání dalších 2 konvolučních vrstev se validační přesnost sítě zlepšila na více než 60 %. Následně se síť optimalizovala zavedením batch normalizací v konvolučních vrstvách (mezi konvolucí a aktivační vrstvou) a dropout vrstev (s pravděpodobností 0.5) mezi plně propojenými vrstvami. Tyto optimalizace opět dopomohly k lepší generalizaci modelu a validační přesnost se přiblížila k 70 %. Nakonec se přidala pátá konvoluční vrstva a počet výstupních neuronů první plně propojené vrstvy se zvýšil na 256. Tato architektura dávala nejlepší výsledky validační přesnosti (při některých nastaveních hyperparametrů optimalizátoru i přes 80 %).

Kromě úprav architektury byly prováděny i další experimenty s cílem zlepšit validační přesnost modelu. Vyzkoušely se různé typy plánovačů učení (tzv. schedulery), které mění hodnotu learning rate v průběhu trénování. Nejprve se zkusil scheduler **ReduceLROnPlateau**, který snížil hodnotu learning rate na polovinu ve chvíli, kdy se přestala zlepšovat validační loss. Tento přístup však nepřinesl výraznější zlepšení výsledků. Následně se vyzkoušel scheduler **StepLR**, který pravidelně snižoval learning rate o polovinu po každých 10 epochách. Tento přístup vedl k mírnému nárůstu validační přesnosti (v řádu několika procent), a proto byl ponechán v konečné verzi modelu. Dále se také porovnávaly různé optimalizační algoritmy. Nejprve se vyzkoušel algoritmus **Adam**, který bývá často považovaný za univerzálně nejvhodnější optimalizátor. Tento algoritmus se ukázal jako velmi účinný napříč různým nastavením hodnot hyperparametrů learning rate a weight decay (výsledky viz podsekce 2.2). Dále se vyzkoušel optimalizátor **SGD** s momentem okolo 0.9, který vykazoval výrazně horší výsledky než Adam (validační přesnost v některých případech klesla pod 60 %). Z tohoto důvodu nebylo SGD v modelu dále použito.

Samotné trénování modelu pak šablonovitě odpovídá trénování klasické neuronové sítě. Nejprve se načtou trénovací a validační datasety. Následně se model během každé epochy trénuje po dávkách (batchích), přičemž se pro každou dávku vynulují gradienty, provede se dopředný průchod sítí, spočítá se trénovací loss funkce (pomocí vícetřídní křížové entropie), provede se zpětný průchod sítí a aktualizace vah. V rámci každé epochy se provádí i proces validace, ve které se model vyhodnocuje pouze dopředným průchodem bez zpětné propagace a aktualizace vah. Během trénování i validace modelu se zaznamenávají různé statistiky (např. trénovací a validační loss), které slouží k vykreslení grafů a celkovému vyhodnocení výkonnosti modelu. Po dokončení trénování se všechny důležité výstupy uloží do podsložky s názvem ve formátu <*validation_accuracy*>_<*time*> uvnitř adresáře **save**, kde <*validation_accuracy*> reprezentuje nejlepší validační přesnost během trénování a <*time*> představuje čas dokončení trénování. V této podsložce se nachází:

- Soubor „*model.pth*“ s váhami natrénovaného modelu
- Soubor „*parameters.txt*“ se záznamem použitých hyperparametrů
- Soubor „*report.log*“ s výpisem průběhu trénování

Kromě toho jsou zde uloženy i výsledné grafy. Obrázek „*loss_and_accuracy.png*“ obsahuje dva podgrafy: vývoj trénovací a validační loss a průběh validační přesnosti v čase. Soubor „*roc.png*“ znázorňuje ROC křivku, která zachycuje vztah mezi mírou skutečně pozitivních a falešně pozitivních klasifikací. Graf „*det.png*“ zobrazuje DET křivku, která vizualizuje míru falešně pozitivních (false alarms) a falešně negativních (misses) klasifikací v závislosti na nastavení rozhodovací hranice klasifikátoru. Pro vykreslení těchto dvou grafů byla vícetřídní klasifikace převedena na sérii binárních klasifikací, kde se pro každý obrázek řešila otázka, zda na něm je či není osoba X, přičemž X nabývalo hodnot od 1 do 31.

Při přepnutí sítě na režim testování se nejprve načtou specifikované váhy natrénovaného modelu (ze souboru „*model.pth*“) a poté následuje klasifikace obrázků ze zvoleného datasetu pomocí dopředného průchodu bez výpočtu gradientů. Výsledky testování se ukládají do souboru „*image_CNN*“ ve formátu odpovídajícím specifikaci zadání projektu. Soubor je vytvořen ve stejném adresáři, odkud bylo spuštěno testování modelu.

2.2 Vyhodnocování

Při vyhodnocování modelu bylo provedeno tzv. „grid testování“ hyperparametrů, konkrétně hodnot learning rate (LR) a weight decay (WD) u optimalizačního algoritmu Adam. Každá kombinace těchto hodnot byla testována na validační sadě, přičemž se zaznamenávala dosažená validační přesnost (v tabulce uvedena v procentech) a hodnota EER (Equal Error Rate, uvedena jako desetinné číslo). Testování probíhalo při nejlepším nalezeném nastavení ostatních parametrů sítě. Výsledky jednotlivých nastavení jsou shrnuty v následující tabulce:

WD / LR	**0.001**	**0.00075**	**0.0005**	**0.00025**	**0.0001**
0.005	68.80 % 0.18	67.25 % 0.16	74.75 % 0.14	76.49 % 0.11	63.59 % 0.19
0.001	69.23 % 0.19	73.26 % 0.18	76.24 % 0.13	76.36 % 0.12	59.80 % 0.19
0.0005	71.28 % 0.22	76.18 % 0.17	70.53 % 0.17	**82.44 %** **0.10**	60.17 % 0.19

Tabulka 4: Výsledky validační přesnosti and EER pro různé nastavení hyperparametrů.

Na základě výsledků validační přesnosti a EER se jako nejlepší varianta jevilo nastavení learning rate = 0.00025 a weight decay = 0.0005. Při podrobnější analýze výstupních grafů se však ukázalo, že toto nastavení způsobuje nestabilitu modelu, jelikož malé změny u trénovací loss vedly k výrazným změnám ve validační loss a validační přesnosti. Proto byly dále analyzovány i jiné konfigurace. Z tohoto průzkumu se podařilo najít 3 stabilnější varianty s dostatečně vysokou validační přesností (nad 74 %), konkrétně kombinace hyperparametrů (LR, WD): (0.0005, 0.005), (0.0005, 0.001) a (0.00025, 0.005). Pro tyto kombinace následně probíhalo opakované trénování modelu. Jako optimální se ukázala konfigurace (0.00025, 0.005), která dosahovala nejvyšší validační přesnosti a nejnižší EER (až 79.9 % a 0.11). Nejlepší síť natrénovaná na této konfiguraci byla zvolena jako finální model pro testování na ostrých datech. Výstupní grafy tohoto modelu (na validačních datech) jsou uvedeny na konci dokumentace.

2.3 Instrukce

Pro reprodukci výsledků uvedených v sekci 2.2 je nutné provést následující kroky:

- **Instalace externích nástrojů:** Nejprve je nutné si doinstalovat knihovny nutné pro spuštění skriptů. Knihovny je možné nainstalovat v rámci virtuálního prostředí příkazem: **pip install -r requirements.txt** nebo lze alternativně spustit instalační skript příkazem: **source requirements_install.sh**, který automaticky nastaví virtuální prostředí a nainstaluje potřebné závislosti.
- **Augmentace datasetu:** Augmentace dat se provádí pomocí skriptu *image_augmentation.py*, který se spouští příkazem: **python3 image_augmentation.py -d <dataset_path>**, kde <*dataset_path*> udává cestu k datasetu. Skript je navržen tak, aby pracoval se strukturou, která odpovídá datasetu přiloženému v zadání projektu. Výstup s augmentovanými daty je uložen do stejného adresáře jako původní dataset.
- **Trénování a testování sítě:** Trénování i testování sítě se provádí pomocí skriptu *image_model_CNN.py*. Trénování modelu se spouští příkazem: **python3 image_model_CNN.py --train <dataset_path>**, kde <*dataset_path*> udává cestu k trénovacímu datasetu. Testování modelu se následně provádí prostřednictvím příkazu: **python3 image_model_CNN.py --test <dataset_path> --model <model_path>**, kde <*dataset_path*> udává cestu k testovacímu datasetu a <*model_path*> specifikuje cestu k natrénovanému modelu (soubor model.pth). Výstupy trénování se ukládají do posložky adresáře save. Výstupní soubor s výsledky testování se ukládá do stejného adresáře, ze kterého bylo spuštěno testování modelu.
- **Nastavení hyperparametrů a spuštění grid testování:** Hodnoty hyperparametrů (learning rate a weight decay) se mohou nastavit v konfiguračním souboru *hyperparams.json*. Samotné grid testování se spouští příkazem: **hyperparams_test.sh <dataset_path>**, kde <*dataset_path*> specifikuje cestu k trénovacímu datasetu.

2.4 Externí nástroje

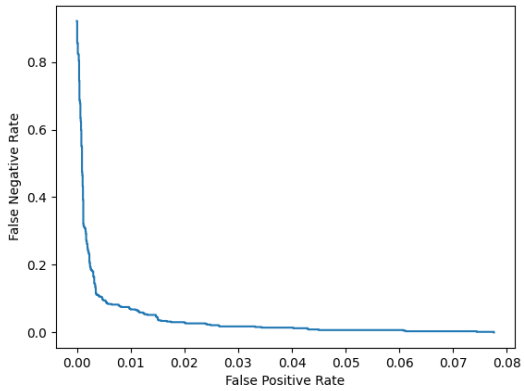
Veškeré externí knihovny použité v projektu jsou uvedeny v souboru „*requirements.txt*“. Tyto nástroje slouží například k vytváření augmentací (*Albumentations*), práci s obrazovými daty (*OpenCV*), manipulaci s vícerozměrnými poli (*NumPy*), vykreslování grafů (*Matplotlib*), získávání statistických metrik (*Scikit-learn*) a trénování či vyhodnocování konvolučních neuronových sítí (*PyTorch* a *Torchvision*).

2.5 Další vyzkoušené techniky

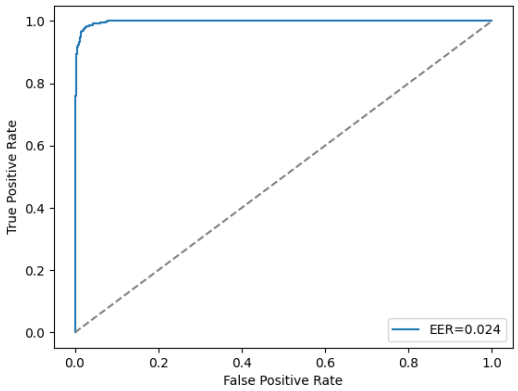
Kromě návrhu konvoluční neuronové sítě byly testovány i alternativní přístupy. Jedním z nich bylo použití klasifikátoru Support Vector Machines (SVM) s příznaky extrahovanými pomocí metody Histogram of Oriented Gradients (HOG). Tento přístup však dosahoval výrazně nižších výsledků (validační přesnost dosahovala pouze kolem 55 %), proto se od tohoto přístupu upustilo. Dalším přístupem bylo natrénování autoenkodéru metodou self-learning na rozsáhlém souboru augmentovaných dat a následné trénování klasifikační hlavy s využitím natrénovaného enkodéru. Hlavní myšlenkou bylo, že páteřní síť se naučí rozpoznávat důležité rysy v obrazech (extrahovat relevantní informace), zatímco klasifikační hlava se naučí tyto rysy správně interpretovat a zařazovat. Tento přístup však zcela selhal (validační přesnost nepřesáhla ani 40 %), pravděpodobně kvůli nedostatečnému množství trénovacích dat.

3 Kombinace modelů

Návrh fúze klasifikačních modelů byl vytvořen podle doporučení z přednášky k projektu. Nejprve se načtou výstupní soubory s výsledky obou modelů a jejich skóre se normalizují pomocí min-max normalizace zvlášť pro každý sloupec. Tím se zajistí, že hodnoty všech tříd budou převedeny do společného rozsahu. Následně se provede sloučení skóre z obou modelů sčítáním hodnot na odpovídajících pozicích. Na základě výsledného skóre se pro každý záznam určí nové klasifikační rozhodnutí a vypočítají se nové logaritmované pravděpodobnosti pomocí log-softmax funkce. Tyto výsledky se pak uloží do souboru *„fuse.csv“*.

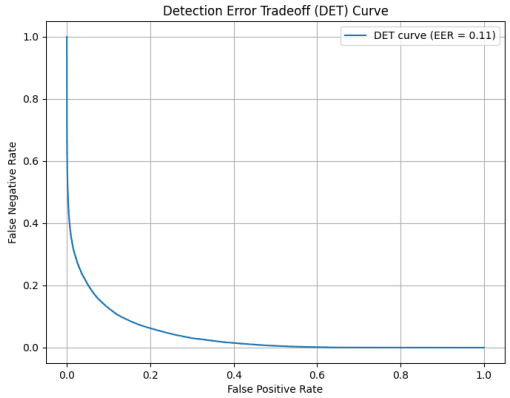


(a) DET křivka

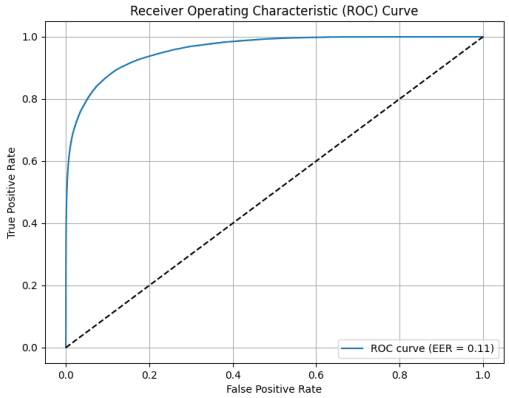


(b) ROC křivka

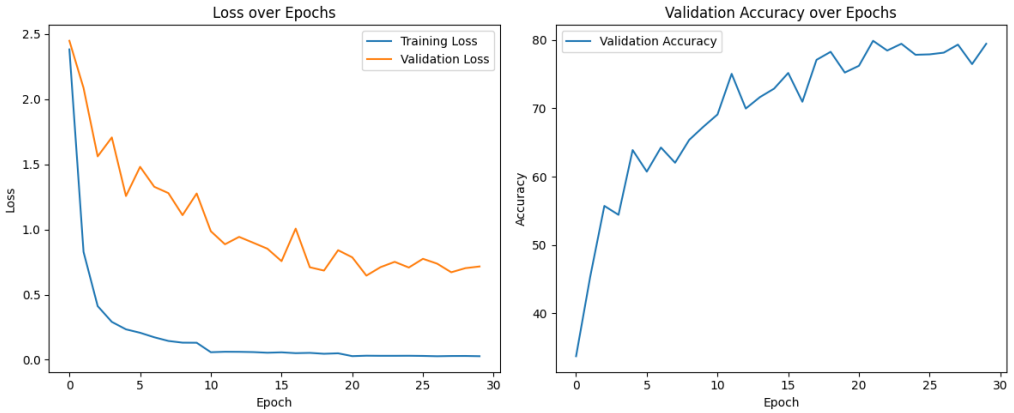
Obrázek 3: Obrázky výše zobrazují DET a ROC křivky nejlepšího natrénovaného modelu při klasifikaci zvukových nahrávek na validační sadě.



(a) DET křivka



(b) ROC křivka



(c) Graf porovnání trénovací a validační loss a graf validační přesnosti

Obrázek 4: Obrázky výše zobrazují DET křívku, ROC křivku, porovnání trénovací a validační loss a validační přesnost nejlepšího natrénovaného modelu při klasifikaci obrázků na validační sadě.

Odkazy

[1] Douglas A. Reynolds, Thomas F. Quatieri a Robert B. Dunn. „Speaker Verification Using Adapted Gaussian Mixture Models“. In: *Digital Signal Processing* 10.1 (2000), s. 19–41. ISSN: 1051-2004. DOI: https://doi.org/10.1006/dspr.1999.0361. URL: https://www.sciencedirect.com/science/article/pii/S1051200499903615.

[2] Chee Cheun Huang, Julien Epps a Ewald Enzinger. „Separate MAP adaptation of GMM parameters for forensic voice comparison on limited data“. In: lis. 2013, s. 1–6. DOI: 10.1109/WIFS.2013.6707785.

[3] Irfan Nafiz Shahan a Pulok Ahmed Auvi. *Towards Speaker Identification with Minimal Dataset and Constrained Resources using 1D-Convolution Neural Network*. 2024. arXiv: 2411.15082 [cs.SD]. URL: https://arxiv.org/abs/2411.15082.