# SUR projekt 2024/2025

Autor: Kryštof Andrýsek (xandry12), Ondřej Dacík (xdacik01)

## Vývoj klasifikátoru řeči

Vývoj klasifikátoru na rozpoznávání řeči by se dal rozdělit do několika fází. Během vývoje se sice více skákalo mezi různými přístupy (Hlavně kvůli tomu, že natrénovat neuronovou síť chvíli trvá.), zde ovšem pro přehlednost bude prezentován po tematických celcích.

Hned od začátku jsme využili MFCC a ořezávali jsme hluchá místa nahrávky. Jak se ale konkrétní verze MFCC měnily bude popsáno později.

### K-Nearest, Random Forest a SVM

Začali jsme tím, že jsme vyzkoušeli naimplementovat metody K-Nearest, Random Forest a SVM. U všech těchto metod jsme využívali funkce z knihovny sklearn, takže jejich implementace byla poměrně rychlá. Data jsme předzpracovávali MFCC, kde jsme extrahovali 13 příznaků a prováděli nad tímto polem průměr.

- Random Forest - stejná data: 100 %, testovací: 35.48 %
- K-nearest - stejná data: 100 %, testovací: 37.10 %
- SVM - stejná data: 100 %, testovací: 58.06 %

Později se úpravou funkce provádějící extrakci příznaků z nahrávku podařilo dosáhnout až následujícího zlepšení.

- Random Forest : 35.48 % --> 58.06 %
- K-nearest: 37.10 % --> 41.94 %
- SVM: 58.06 % --> 69.35

Úprava spočívala v tom, že jsme do vektoru přidali i směrodatnou odchylku. Tyto metody byly testovány pouze na ne zašuměných datech.

### Neuronová síť

Neuronová síť používala taktéž MFCC s extrakcí 13 příznaků. V průběhu jsme zkoušeli je různě normalizovat. Augmentací přidávat data. Nebo naopak průměrovat. Architektury modelů jsme trénovali 2 (RNN a MLP). Zkoušeli jsme různé množství epoch, adaptivní learning rate i L2 regularizaci. Jako loss funkci jsme využívali křížovou entropii. Pokusy byly různých výsledků, ale obecně by se dalo říct, že se síť nic nenaučila nebo přetrénovala, neboť nejlepšího výsledku, jehož bylo dosáhnuto bylo 3.64 % na testovacích datech.

Později se vyzkoušelo GMM a díky tomu, že už na poprvé fungovalo dobře a kvůli časové náročnosti trénování neuronové sítě, se vývoj pomocí neuronové sítě zastavil.

## GMM

Začali jsme s tím, že jsme modelovali všechna data s využitím jednoho GMM. K extrakci příznaků se využívala MFCC a extrahovalo se opět 13 příznaků. Model měl úspěšnost přes 70 %.

Další vývojovou větví bylo modelování každé třídy pomocí směsice Gaussovských rozložení. Tento přístup ovšem vedl k horším výsledkům než ten předešlý. Úspěšnost se sice postupem času podařilo z původních 31 % vytáhnout až na okolo 50 %. Ale samotné modelování tak malého množství dat pomocí 31 GMM se nám později nezdálo jako správný přístup.

## Výsledné řešení – GMM

Jako výsledné řešení byl tedy zvolen GMM dohromady pro všechna data. Postupnými změnami se podařilo dosáhnout až úspěšnosti kolem 87 % (85 % - 90 % v závislosti na natrénování klasifikátoru).

Vylepšení, která k tomu napomohly byla:

- Zvýšení extrahovaných příznaků na 30. Zkoušelo se i 10, 13, 20, ale 30 se nám zdálo s ohledem na výsledky modelu a časovou náročnost nejvhodnější.
- Při extrakci se počítají i první a druhé derivace, které se přidávají do výsledného vektoru.
- Trénování a testování na augmentovaných datech. To pomáhalo především s nižším množstvím příznaků.
- Změna toho, kdy se ořezává ticho. Tahle změna je trochu zvláštní, neboť ořezání před MFCC vrací konzistentnější výsledky, ale ořezání po MFCC a případné přepočítání MFCC může dosáhnout lepších výsledků (až 90 %), ale i horších (okolo 85 %).
- Výsledný počet komponent je 16. Lze vzít i více, a potom snížit počet extrahovaných příznaků, ale nevede to k lepším výsledkům. Méně, například deset, vede ke zhoršení klasifikátoru.

## Popis řešení

### *Obecný popis*

Program pro klasifikaci zvuku je v souboru audio_gmm_singleGMM_aug.py. Jeho hlavní částí je třída *GMMVoiceModel*, která obstarává všechny části trénování i evaluaci.

Proces začíná extrakcí příznaků z dat (*extract_features*) a je možné provést nad těmito daty augmentaci přidáním šumu. Vždy se provádí ořezání ticha a taktéž druhé derivace MFCC. Poté probíhá trénování (*train*).

K predikci hodnot lze použít funkci *predict*. K evaluaci celého modelu potom funkci *evaluate*, ta ovšem funguje jen nad daty, které mají obdobný formát jako trénovací. Tzn. Názvy složek odpovídají labelům.

### *Nastavitelné parametry (v kódu)*

*number_of_components* – počet komponent modelu

*train_path* - cesta k trénovacím datům

*test_path* - cesta k testovacím datům

*add_noise_train* - zapnutí/vypnutí augmentace dat (přidáním dat se šumem) při trénování

*add_noise_test* - zapnutí/vypnutí augmentace dat (přidáním dat se šumem) při testování

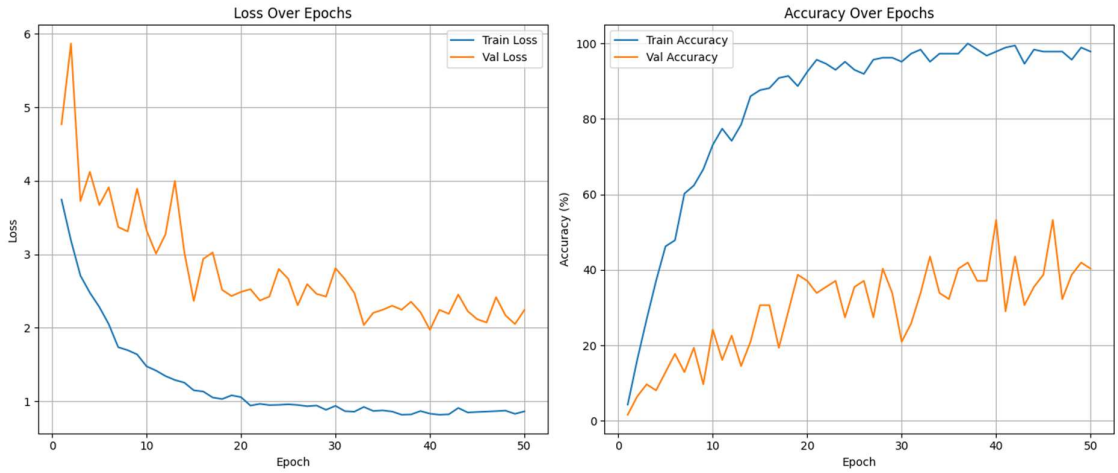
*old_split* - ořezává se před/po extrakcí MFCC

*n_mfcc_parameters* - počet MFCC příznaků, které se zde mají vyextrahovat

*model_name* - jméno, jak má být uložen výsledný model

# Klasifikátor obrazu

Pro obraz jsme zvolili konvoluční neuronovou síť. Testovali jsme, jak vlastní architekturu skládající se z několika konvolučních bloků (ty se skládají z konvoluce, batch normalizace, ReLU aktivační funkce a max pooling) a výstupní plně propojené vrstvy, tak existující architektury jako ResNet18 nebo EfficientNet-B0. Podle výsledků (viz obrázek níže) jsme nakonec zvolili architekturu ResNet18.




*Průběh trénování s architekturou Resnet18*

## Augmentace

Vzhledem k malému počtu trénovacích vzorů jsme hojně využívali augmentace: Obrázky byly během trénování náhodně rotovány, ořezávány, překlápěny a také byla lehce měněna perspektiva a rozmazání.

## Nastavitelné parametry trénování

Experimentovali jsme s různými kombinacemi parametrů, nejlepší a nejstabilnější výsledky jsme získali s tímto nastavením:

LR = 1e-3  (learning rate, rychlost učení sítě)

WEIGHT_DECAY = 1e-4  (síla penalizace větších vah)

BATCH_SIZE = 16  (velikost dávky v rámci jedné epochy)

DROPOUT = 0.3  (síla prevence přetrénování náhodným nulováním aktivací)

PATIENCE_THRESHOLD = 20  (dřívější ukončení trénování pro prevenci přetrénování)

EPOCHS = 200  (maximální počet epoch)

### Průběh trénování

Z dat ze zadání se vytvoří trénovací dataset (adresář train) a validační dataset (adresář dev). Model se začne trénovat na trénovacím datasetu, na konci každé epochy se spolu s trénovací loss a přesností vyhodnotí také validační loss a přesnost. Pokud se po daný počet epoch validační přesnost nezlepší (dáno parametrem PATIENCE_THRESHOLD), trénování se ukončí a jako výsledný model se uloží stav z epochy s nejvyšší validační přesností. Tímto se (alespoň částečně) zamezí přetrénování na trénovacích datech.

### Vyhodnocení modelu

U modelu jsme sledovali hlavně dosaženou přesnost na validačním datasetu. Informativní ale také byl graf srovnávající trénovací a validační loss a graf porovnávající trénovací a validační přesnost. Z nich je možné vyčíst, kdy zhruba začíná být model přetrénovaný.

### Techniky s pozitivním přínosem

- Vyhlazování labelů (label smoothing) pomohlo se stabilitou tréninku a pravděpodobně také zlepšilo generalizaci modelu. Tato technika vyhlazuje one-hot vektor labelů (např. [0, 1, 0] se vyhladí na [0.02, 0.96, 0.02]) a tím zajišťuje, že model nezíská na trénovacích datech příliš velkou jistotu.
- Dropout, regularizace vah a brzké zastavení také pomohly v boji s přetrénováním a s lepší generalizací.

## Kombinovaný klasifikátor

Výsledné řešení používané ke klasifikaci je implementováno souborem eval_both.py. Kombinuje GMM model k vyhodnocení audio nahrávek a Resnet síť používanou ke klasifikaci obrázků. Vzhledem k tomu, že GMM dosahuje lepších výsledků, než neuronová síť přikládáme jí větší váhu 70 %.

Výsledné řešení je získané z kombinované logaritmické pravděpodobnosti obou tříd. Podle nejvyšší je poté vybrána predikovaná třída.

## Získání modelu a spuštění

Natrénovaný model je dostupný zde:
https://drive.google.com/drive/folders/14ohcmKs9YFZEOfGGa0m8uWWyut7AsztN?usp=sharing

Model lze používat pomocí programu *eval_both.py,* ten má nastavitelné parametry jména modelů a cesty ke vstupním a výstupním souborům. Potřebné knihovny jsou v souboru requirements.txt. Soubor lze pustit následovně: *python eval_both.py*.