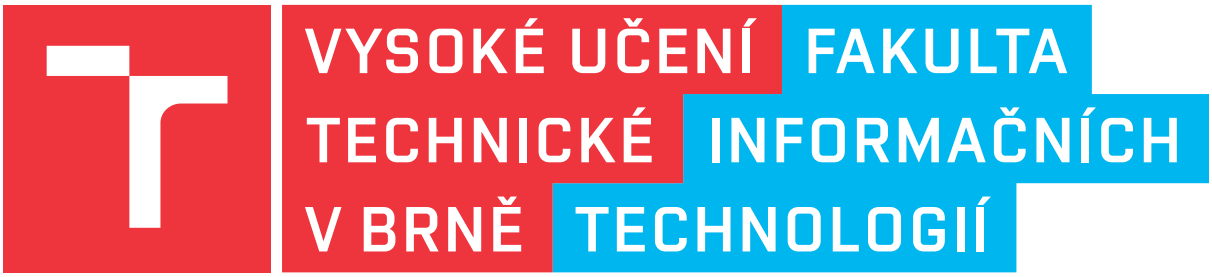



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# SUR

Identifikácia osôb z obrázku tváre a hlasovej nahrávky

5. května 2025

Adam Rajko (xrajko00)
Jakub Škunda (xskund02)

# 1 Úvod

Na implementáciu modelov sme použili programovací jazyk `Python 3.12` a virtuálne prostredie `venv`. Pre inštaláciu prostredia s potrebnými knižnicami použite nasledujúce príkazy v adresári `/src`:

```
python -m venv myenv
source myenv/bin/activate
pip install -r requirements.txt
```

# 2 Klasifikácia reči

Pre klasifikáciu reči sme použili metódu založenú na GMM. Pri implementácii sme sa inšporovali prednáškami a demonštračnými príkladmi[1] a podľa odporúčania sme použili funkciu `wav16khz2mfcc` a upravili podla svojich potrieb.

## 2.1 Vstupy a výstupy

**Vstup**: súbor s trénovacími, testovacími a evaluačnými dátami – cesty sú uložené v premenných v súbore `/src/voice.py`
**Výstup**: súbor s výsledkami klasifikácie, ktorý obsahuje meno súboru, predpovedanú triedu a logaritmické pravdepodobnosti pre každú triedu. Súbor s výstupnými dátami bude vytvorený v adresári so zdrojovými kódmi.

## 2.2 Predspracovanie dát

Na spracovanie dát sme použili nasledujúce kroky:

- **Odstránenie tichých rámcov** – z každ´ho audio súboru sme odstránili tiché rámce pomocou funkcie `remove_silence()`. Tieto rámce sú identifikované na základe short-term energy.
- **Výpočet MFCC koeficientov** – Koeficienty MFCC boli vypočítané pomocou funkcie `librosa.feature.mfcc()` s podobnými parametrami ako v sprístupnených príkladoch
- **Cepstral Mean Normnalization** - Každý extrahovaný MFCC príznak bol následne normalizovaný pomocou funkcie `apply_cmn()`
- **Výpočet delta koeficientov** – pre každú nahrávku boli tiež extrahované delta koeficienty prvého a druhého rádu

Všetky tieto vlastnosti boli následne konkatenované a použité ako vstupy pre trénovanie GMM modelu.

[1] https://www.fit.vutbr.cz/study/courses/SUR/public/prednasky/demos/

## 2.3 Trénovanie GMM

Implementáciu GMM sme použili z knižnice `sklearn`[2]. Model bol trénovaný na všetkých vzorkých trénovacieho datasetu a následne bola určená accuracy na dev datasete. Parametre modelu boli zvolené nasledovne:

- **Počet komponentov** – pre každý model sme použili rovnaký počet komponentov. Najlepšie výsledky sme dosiahli s 10 komponentmi, ktoré sa ukázali ako optimálne pre naše dáta
- **Počet iterácii** – pre každý model sa vykoná 250 iterácii EM algoritmu. Pri nižšom počte iterácií niektoré modely nezkonvergovali správne, čo malo za následok horšie výsledky
- **Typ kovariančnej matice** – použili sme `covariance_type='tied'` čo znamená, že všetky komponenty zdieľajú rovnaký typ kovariančnej matice
- **Počiatočné nastavenie parametrov:** Počiatočné hodnoty pre stredy komponentov boli zvolené náhodne z tréningových dát

Tento proces bol vykonaný pre každú 31 tried, čím sme získali 31 modelov. Následne sa na dev dátach previedla evaluácia modelu, ktorý dosiahla maximálnu úspešnosť cca 85 %.

# 3 Klasifikácia tváre

Ukážky trénovania a klasifikácie modelov na rozpoznávanie tváre sú v jupyter notebooku: `image-classification.ipynb`.

## 3.1 Dataset

### Augmentácia

Na každý pôvodný obrázok bolo vygenerovaných 75 augmentovaných verzií.

- geometrických transformácií (rotácia, translácia, zoom, perspektíva),
- rozostrenia a šumu (Gaussian, Motion, ISO),
- oklúzií (náhodné polygóny, `CoarseDropout`),
- zmien farebnosti (jas, kontrast, sýtosť),
- horizontálneho zrkadlenia a náhodného orezu.

Tieto transformácie boli aplikované náhodne, pričom bola zabezpečená konzistencia výstupného rozlíšenia **80 × 80 px**.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

### Normalizácia

Pre úspešné trénovanie neurónovej siete bolo potrebné normalizovať vstupné obrázky. Preto sme pre celý trénovací dataset (augmented) vypočítali **strednú hodnotu (mean)** a **štandardnú odchýlku (std)** pre každý z troch kanálov (RGB).

Tieto hodnoty boli následne použité v normalizačnej transformačnej funkcii pred vstupom do modelu.

### Rozdelenie datasetu

Dataset bol rozdelený na trénovací a validačný v pomere `0.8` a `0.2`. Toto rozdelenie zabezpečuje, že validačné dáta sú nezávislé od trénovacích a model sa na ne nemôže priamo naučiť.

## 3.2 Modely

V rámci experimentov bolo testovaných viacero architektúr konvolučných neurónových sietí, od jednoduchých vlastných modelov až po známe architektúry. Všetky modely boli upravené tak, aby zodpovedali počtu tried v zadanom datasete.

### Prehľad modelov

V nasledujúcej tabuľke sú uvedené použité modely a ich počet trénovateľných parametrov:

| **Model** | **Počet parametrov** |
|---|---:|
| ResNet18 | 11,192,415 |
| CNN [8, 16, 32] + FC[32] | 109,599 |
| CNN [8, 16, 32, 64] + FC[64] | 129,247 |
| CNN [8, 16, 32, 64, 128] + FC[128] | 168,543 |
| CNN [16, 32, 64] + FC[64] | 435,487 |

Tabulka 1: Architektúry modelov a počet parametrov

### Vlastné CNN modely

Vlastné modely pozostávajú z niekoľko blokov a jednou alebo viacerými plne prepojenými (fully connected) vrstvami. Pod blokom rozumieme (konvolučnú vrstvu + batch normalizáciu + ReLu + MaxPooling). Použitý bol dropout na plne prepojenú vrstvu na zníženie pretrénovania.

### ResNet18

Okrem vlastných modelov bol použitý aj štandardný ResNet18, prispôsobený pre náš počet tried. Do architektúry bol pridaný dropout pred finálnou vrstvou, aby sa model ľahko nepretrénoval. Vzhľadom na robusnosť siete a zlým výsledkom na malom datasete sme sa rozhodli túto sieť nepoužiť.

## 3.3 Trénovanie a hyperparametre

### Optimalizácia

Na trénovanie modelov bol použitý optimalizátor `Adam`, ktorý bol zvolený pre svoju efektivitu a stabilitu pri konvergencii. Počiatočná hodnota learning rate bola nastavená na $0.0001$ a L2 regularizácia (tzv. weight decay) $1e^{-}4$, penalizácia veľkých hodnôt váh a pomáha znižovať overfitting.

### Scheduler

Pre adaptívne riadenie učenia počas trénovania bol použitý plánovač učenia `ReduceLROnPlateau`, ktorý monitoruje validačnú stratu. Ak sa validačná strata po určitý počet epoch nezlepšila (`3`), learning rate bol automaticky znížený. Táto stratégia pomáha modelu dosahovať lepšiu generalizáciu.

### Loss funkcia

Ako stratová funkcia bola použitá `CrossEntropyLoss`.

### Batch size a epochy

Batch size bol zvolený ako 32. Každý model bol trénovaný po dobu 30 – 50 epoch, v závislosti od rýchlosti konvergencie, pričom trénovanie bolo ukončené pri ustálení validačnej presnosti.

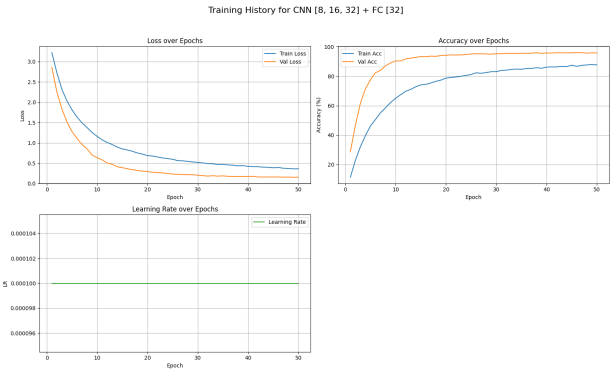
## 3.4 Výsledky

Tabulka 2: Trénovacie a validačné metriky

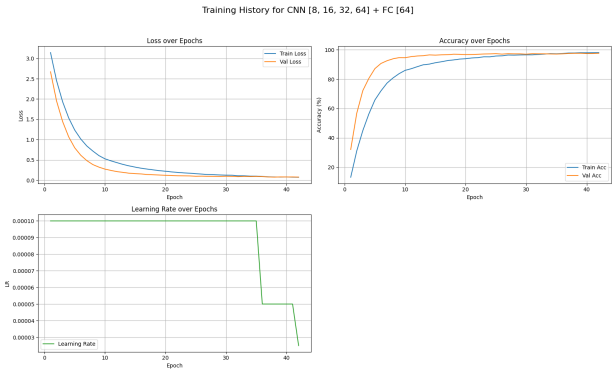
| Model | Train Loss | Train Acc (%) | Val Loss | Val Acc (%) | Epochs |
|---|---|---|---|---|---|
| CNN (a) | 0.3596 | 87.66 | 0.1549 | 95.78 | 50 |
| CNN (b) | 0.0705 | 98.10 | 0.0748 | 97.59 | 43 |
| CNN (c) | 0.0075 | 99.92 | 0.0907 | 97.67 | 40 |
| CNN (d) | 0.0780 | 97.60 | 0.0792 | 97.80 | 50 |

## 3.5 Návrh možných zmien alebo experimentov

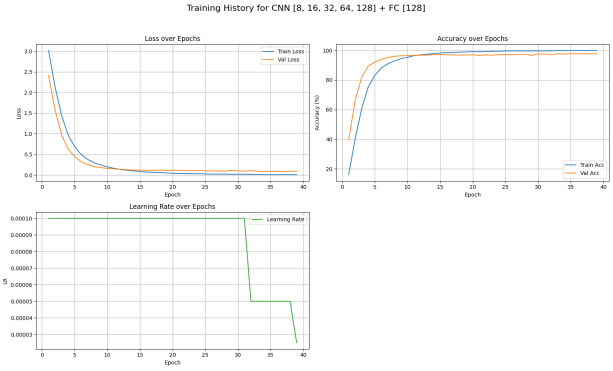
Preskúmať viac možnosť dropout vrstiev aj na konvolučné vrstvy, táto možnosť je predpripravená v kóde a bola do istej miery vyskúšaná ale nemala nejak skvelé výsledky. Použiť **label smoothing**, čo predstavuje techniku, ktorá znižuje pretrénovanie modelu a zvyšuje jeho robustnosť.



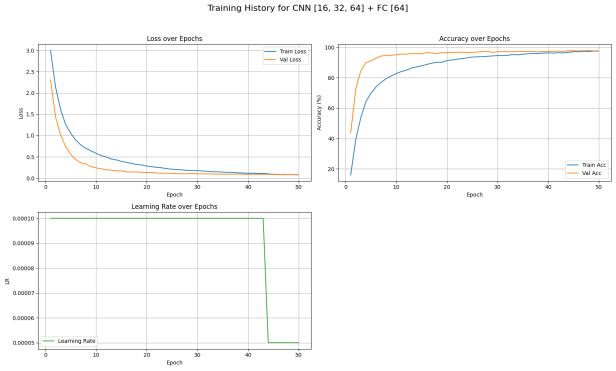

(a) CNN [8, 16, 32]-[32]




(b) CNN [8, 16, 32, 64]-[64]




(c) CNN [8, 16, 32, 64, 128]-[128]




(d) CNN [16, 32, 64]-[64]

Obrázek 1: Porovnanie histórie trénovania pre rôzne CNN modely