SUR projekt – Identifikace osob

Dokumentace implementace

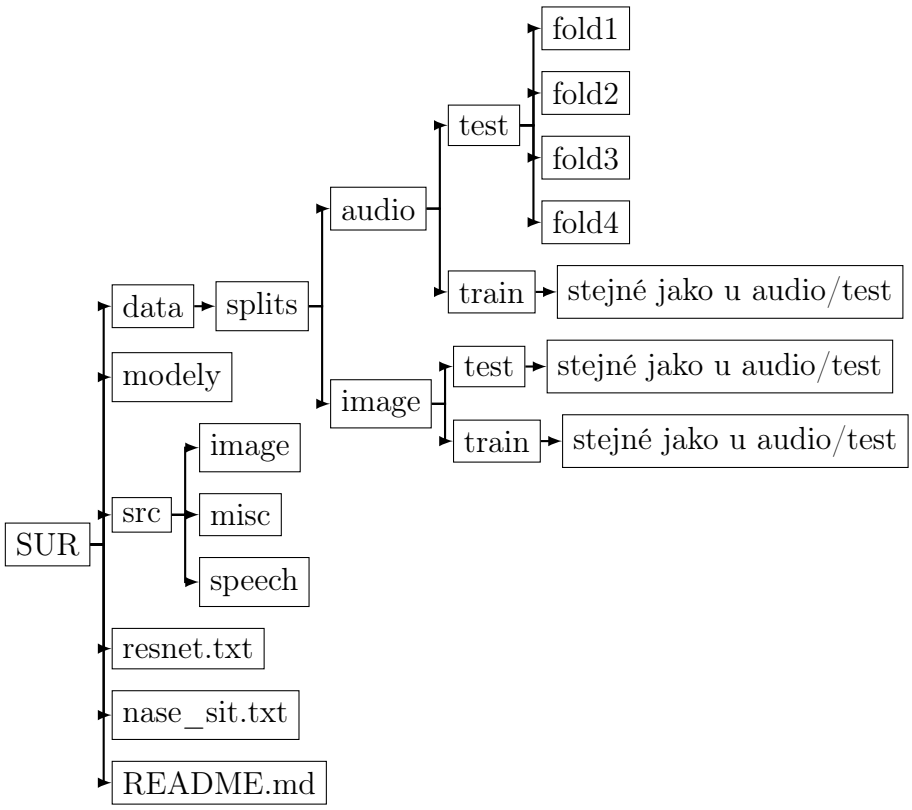
Petr Kaška (xkaska01)
Martin Hemza (xhemza05)

4. května 2025

Obsah

1 Spuštění

V této sekci probereme spuštění našeho projektu a reprodukci výsledků. Pro správné fungování našeho projektu je nutné dodržet adresářovou strukturu níže. Při používání z kořenového adresáře se program používá následovně, kde:



Obrázek 1: Adresářová struktura projektu

1.1 Trénování Image modelu

```
python3 src/image/image_train.py --root DIR --model MODEL --fold
        FOLD --epochs EPOCH --bs BS --lr LR --out OUT
```

- **DIR** - udává adresář datasetu, který chceme použít (v našel případě by to bylo data).
- **MODEL** - údává model, který chceme použít při trénování, buď (face_resnet nebo my_model).

- **FOLD** - údává složku z jejíchž dat se provede trénink (fold[1-4]).
- **EPOCH** - počet epoch.
- **OUT** - název modelu, který bude natrénován.

1.2 Inference Image modelu

```
python src/image/image_eval.py --root DIR --weights WEIGHTS --model
                        MODEL --out OUT
```

- **DIR** - udává adresář datasetu, který chceme vyhodnotit naším modelem (v našel případě by to bylo data/splits/image/fold[1-4]).
- **WEIGHTS** - údává složku, ve kteté se nachazí model, který chceme při inferenci použít.
- **MODEL** - údává model, který chceme použít při trénování, buď (face_resnet nebo my_model).
- **OUT** - název souboru, kam se vypíší výsledky ve formátu ze zadání.

1.3 Trénování hlasového modelu

```
python src/speech/speech_gmm.py --train_dir DIR --gmm_components NUM
                        --model_out OUT
```

1. **DIR** - název souboru, kam se vypíší výsledky ve formátu ze zadání.
2. **NUM** - Číslo udávající počet různých Gaussovek, které společně (váženě) co nejlíp popíšou rozložení našich MFCC příznaků pro daného mluvčího.
3. **OUT** - název souboru, kam uloží hotový balík modelů.

1.4 Inference hlasového modelu

Vyhodnocení běží na jeden příkaz:

```
python src/speech/speech_eval.py --model_in IN --eval_dir DIR --out
                                OUT
```

1. **IN** - název souboru, ve kterém se nachází model, který chceme použít při inferenci.

2. **DIR** - název složky, ve které jsou složky mluvčích, nad kterými chceme provézt inferenci.

3. **OUT** - název souboru, kam je vypsán výsledek inference v požadovaném formátu.

2 Dataset

V této sekci rozebereme, jaké úpravy jsme v datasetu provedli a jak jsme ho použili ke trénování a testování. Původní dataset obsahuje pro každou osobu 8 nahrávek hlasu a 8 fotografií obličeje, což je malý počet pro trénování modelu, proto jsme se rozhodli použít augmentaci k získání většího počtu dat. Také jsme si všimli, že 8 vstupních nahrávek pro jednu osobu je složeno ze čtyř dvojic, kde každá dvojice je nahrána/vyfocena ve stejný den. Tento fakt jsme zohlednili při úpravě, tak že jsme data pro jednu osobu rozdělili na 6 snímků/nahrávek na trénování a 2 snímky/nahrávky na testování. S tím že zmíněné 2 testovací vzorky byly každý získán v jiný den. Nakonec jsme tedy vytvořili celkem 4 různé složky abychom vyzkoušeli při tréninku a testovaní kombinaci dat ze všech dní. Vysvětlení na příkladu a na jedné osobně (První složka tedy obsahoval pro osobu č. 1 trénovací nahrávky tři až osm a testovací jedna a dva, druhá složka pro osobu č. 1 obsahovala trénovací nahrávky pět až osm a jedna a dva a testovací tři a čtyři, a tímto způsobem jsme protočili všechny nahrávky). Toto jsme se rozhodli udělat, protože při manuálním průchodu vzorků, byli u některých osob v některé dny pořízeny fotografie např s prsty před obličejem, rozmazané, ... Tedy jsme chtěli najít nejlepší kombinaci.

2.1 Dataset obrázků

V této sekci popíšeme úpravu datasetu speciálně pro obrázky. Zprvu když jsme objevili jak fungují augmentace v pythonu jsme na data aplikovali přehnaně moc augmentací a to obrázky kompletně znetvořilo a vzhledem k tomu, že jsme měli i málo původních obrazků, tak jsme spíš vytvořili "detektor šumu". Po této zkušenosti jsme se uchýlili pouze k základním augmentacím obličejů, které jsme našli na této stránce [1]. A z každého ze šesti obrázků v trénovacím datasetu jsme tímto způsobem vytvořili 80 naaugmentovaných pomocí online augmentace.

[1] https://www.datacamp.com/tutorial/complete-guide-data-augmentation

2.2 Dataset hlasu

Trochu rozdílné úpravy než na obličejovém datasetu byly potřeba provést i na tomto hlasovém, jelikiž se v něm nacházelo docela dost nahrávek, kde byla valná většina nahrávky ticho a poté třeba vyslovení jedné slabiky nebo například byly v nahrávce 2 slova, první bylo řečeno na začátku, poté 10 sec ticho a vyslovení druhého slova. Tak jsme se rozhodli ticho odstranit kompletně a k tomu jsme využili python knihovnu *librosa* a také jsme nahrávky normalizovali, protože osoby zněli, soudě podle poslechu z reproduktorů, v různé dny jemně odlišně.

3 Implemetance Obličejové části

V této sekci popíšeme náše myšlenkové pochod při vypracovávání části projektu pro identifikaci lidí podle obličeje a přiblížíme její implemeční části. Pro klasifikaci obličejů jsme se rozhodli použít neuronovou síť, pro tento úkol jsem si zvolili knihovnu *PyTorch*, protože jsme ji používali již v jiných projektech. Již z prvních přednášek předmetu KNN mě osobně zajímalo jak například takové síťě *ResNet, AlexNet, ...* fungují v praxi, proto jsme se rozhodli vyzkošet architekturu *ResNetu* na tomto problému. Víme, že nejspíše nebude nejlepší řešení, hlavně kvůli malému počtu trénovacích dat a složitosti vybrané sítě, ale stejně jsme to chtěli vyzkoušet. Zprvu jsme si, ale zkusili nadefinovat svoji síť od nuly, která je definována ve třídě *MyModel*. S touto síťí jsme moc velikých úspěchů zezačátku nedosahovali, konkrétně něco okolo přesnosti 0.05, ale později se nám ji podařilo zlepšit až na přesnost 0.7 hraním s různými parametry (přidávání vrstev, změnou parametrů vrstev, aktivační funkce, ...) z čehož jsme byli spokojeni a její konečné parametry přikládám do souboru *nase_sit.txt*. Nakonec jsme vyzkoušeli architekturu *ResNet18*, kterou jsme přimo zavolali z *PyTorch* knihovny, takto snadno získanou síť jsme ovšem ještě museli upravit, jelikož původní *ResNet* je klasifikátorem obrázků do 1000 tříd. Takže jsme změnily implementaci hlavy, aby naše verze *ResNetu* klasifikovala do 31 tříd, přidali jsme na konec lineární vrstvu, která z 256-dimenzionálního embeddingu udělá 31 výstupních logitů. Očekávali jsme, že s takto upravenou sítí dosáhneme vysoké úspěšnosti místo toho jsme s téměř o 4 miliony více parametrů dosáhli stejné úspěšnosti jako s námi vytvořenou síťí. Proto jsme se rozhodli přidat *CosFace* místo klasického softmaxu, kterou jsme znali opět z knn a čerpali jsme z článku Wanga [3]. A dále jsme přidali dropout a *MixUP*[2]. S těmito úpravami jsme byli schopni dosáhnout přesnosti 0.95.

[2]Stránk, ze které jsme čerpali: https://paperswithcode.com/method/mixup

Dále si podrobněji rozebereme jednotlivé moduly.

3.1 `image_augment.py`

Modul definuje sadu **albumentations** transformací, které jsou definované stejno jmennou python knihovnou, a které slouží k augmentaci dat v různých režimech: *train* a *val*.

Použité augmentace Pro trénink jsme použili sekvenci: horizontální převrácení (0.5), afinní transformace (translace 5 %, rotace $\pm 10°$, shear $\pm 4°$), jedna z *rozmazání/šumu*, mírná úprava barev, *Coarse Dropout* a nakonec změna velikosti, normalizace a `ToTensorV2`. Transfomace jsme vybrali tak, aby nenarušily strukturu obličeje, ale zároveň zvýšily variabilitu vstupů. A vyšly jsme z článku[3]. Při validaci používáme pouze změnu velikosti a normalizaci, protože jsme chtěli porovnávat s nezašumělými daty, (zprvu jsme měli augmentaci i pro validaci, to nám ale model dostatečně negeneralizoval a na čistých datech bez augmentace jsme měli horší úspěšnost než na augmentovaných datech).

3.2 `image_models.py`

V tomto souboru jsou implementovány zmíněné modely, tedy *ResNet18* a *Náš-model*, které jsme podrobněji rozebírali výše.

3.3 `image_train.py`

Mix-Up. Bez implementace Mix-up jsme dosahovali úspěšnosti zhruba 0.8 a po přidání do zdrojového kódu 0.91. Samotná implementace funkce provádí lineární kombinaci dvojic vzorků s koeficientem α.

Optimalizace. `AdamW` s výchozím $lr = 10^{-3}$ a váhovým úbytkem 10^{-4}. Scheduler implementuje *warm-up* (5 epoch).

Early-Stopping. Pokud se validační přesnost nezlepší během 14 epoch, trénování končí. Nejlepší model se ukládá do složky `modely`.

[3]Inspirace `https://www.datacamp.com/tutorial/complete-guide-data-augmentation`

3.4 Další podpůrné skripty

Mezi další námi vytvořené skripty patří *image_ eval.py*, tento skript slouží k vytvoření predikcí na základě natrénovaného modelu a *datasets.py* na načtení datasetu.

4 Implementace hlasové části

V této části popíšeme kompletní pipeline pro identifikaci řečníka z krátkých nahrávek řeči.Bohužel nám nezbylo moc času na testování více metod hlasové identifikace, jak jsme původně s kolegou chtěli, tak jsem zvolili pouze jednu a to GMM - **Gaussian Mixture Model**, která je trénovaná *na osobu*, tedy jeden model na každého mluvčího. Experimentovali jsme pouze s různým počtem komponent a jako nejlepší počet se nám osvědčil počet osm. Využili jsme proto funkci *GaussianMixture* z knihovny *sklearn*. Na validačních částech čtyř foldů jsme naměřili průměrnou přesnost **0.78**. Chyby se objevují hlavně u klipů, kde po odfiltrování ticha zbylo jen pár slabik.

4.1 Přehled architektury

Celý proces rozpoznání řečníka probíhá ve třech krocích:

1. **Extrakce příznaků** – z každé wav-ky vytáhneme 20 *MFCC* koeficientů a k nim ještě jejich první a druhou derivaci. Dohromady je to 60 čísel pro jedno krátké okénko zvuku. Počítá to funkce `_mfcc_from_signal`, která se opakuje ve všech našich skriptech.

2. **Trénink GMM** – vezmeme všechny okénka daného mluvčího, sesypeme je do jednoho pytle a naučíme pro něj samostatný *Gaussian Mixture Model*. Standardně používáme $K = 8$ směsí a o všechno se stará třída `GMMSpeakerID`.

3. **Vyhodnocení** – u neznámé nahrávky spočítáme, jak moc jí „věří“ každý model. Ten s nejvyšším skóre vyhrává a oznámí ID řečníka.

4.2 Extrakce příznaků

Na vstupu ze signálu o 16kHZ uděláme nejprve toto:

- rozseká zvuk na okénka 25 ms s posunem 10 ms,
- z každého okénka spočítá 20 MFCC,

- přidá první a druhé derivace (Δ, Δ^2).

Vrátí tak matici velikosti ($T \times 60$) s desetinnými čísly. Žádná další normalizace (CMVN) se nekoná – našimi experimenty si s parametry se ukázalo, že pro naše jednoduché GMM to není potřeba.

4.3 `speech_model.py`

Tento soubor obsahuje jedinou třídu `GMMSpeakerID`:

- `fit(feats, labels)` – natrénuje jeden GMM (diag. kovariance) pro každý unikátní ID řečníka,
- `score\_matrix(feats)` – vrátí tabulku log-pravděpodobností,
- `predict(feats)` – vybere ID s nejvyšším skóre.

Reference

[1] Buslaev A. et al., *Albumentations: Fast and Flexible Image Augmentations*, 2020.

[2] Deng J. et al., *ArcFace: Additive Angular Margin Loss for Deep Face Recognition*, CVPR 2019.

[3] CosFace: Large Margin Cosine Loss for Deep Face Recognition Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu *Tencent AI Lab* wliu@ee.columbia.edu