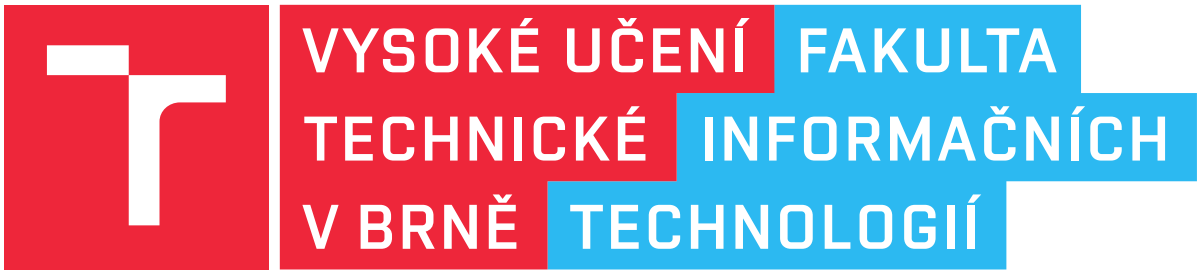


# Projekt SUR

Tomáš Lukáč
xlukac16, xlukac16@stud.fit.vutbr.cz
5.5. 2025

## 1.1 Dataset

Prvou úlohou pri vytváraní klasifikátoru bolo vyriešenie problému malého množstva trénovacích dát. Na dátach augmentovaných ďalej popísanou metódou boli trénované obe popísané siete, preto spôsob augmentácie uvediem v samostatnej kapitole.

### 1.1.1 Šum

Prvé rozšírenie datasetu bolo vykonané pridaním šumu, čím mala byť simulovaná klasifikácia na základe poškodených údajov. Na vygenerovanie šumu bola použitá funkcia knižnice numpy - np.random.normal v tvare odpovedajúcom údajom. Z jedného súboru boli vygenerované 3 varianty pripočítaním šumu s intenzitou 0.2, 0.4 a 0.6 a následnou normalizáciou. Tieto novo vytvorené varianty boli popárované tak, aby z každej dvojice údajov vzniklo 7 nových tak, že jeden údaj z páru zostal nepoškodený a ten sa spároval so všetkými úrovňami poškodenia druhého údaju.

### 1.1.2 Zvukové dáta

Na spracovanie zvukových dát do použitej podoby bola použitá knižnica torch.transforms a torchaudio.transforms. Zvukové údaje boli spracované nasledovne:

1. Prevzorkovanie na rovnakú frekvenciu - 16000
2. Odstránenie prvých dvoch sekúnd klipu
3. Zarovnanie údajov na rovnakú dĺžku odstránením koncu nahrávky alebo predĺženým prázdnymi symbolmi
4. Aplikácia MFCC / MelSpectrogramu na nahrávku
5. Konvertovanie získaných koeficientov na decibely pomocu funkcie torchaudio.transforms.AmplitudeT

Zvukové dáta boli zarovnané na 240000 vzorkov, čiže 15 sekúnd. Pri trénovaní zmiešaného klasifikátoru bol použitý MelSpectrogram, pri tréningu audio klasifikátoru bolo použité kompaktnejšie MFCC.

### 1.1.3 Obrazové dáta

Na spracovanie obrazových dát bola použitá knižnica torch.transforms. Obrazové údaje boli spracované nasledovne:

1. Konvertovanie na RGB formát obrazu.
2. transforms.RandomResizedCrop - Náhodný výrez okna zo zadaného obrazu
3. transforms.RandomHorizontalFlip - Náhodné prevrátenie obrazu po zvislej ose
4. transforms.RandomRotation - Náhodné natočenie obrazu s limitom 30 stupňov
5. transforms.ColorJitter - Náhodná úprava farieb obrazu

Pri úprave farieb obrazu sa mení svetlosť, kontrast, saturácia a odtieň farieb.

## 2.1 Klasifikátor podľa zvukovej nahrávky

### 2.1.1 Vytvorenie vektoru vlastností

Na extrakciu vektoru vlastností bola použitá sieť s nasledujúcou architektúrou:

- Konvolučná 1D vrstva - (in=64, out=16, kernel_size=5, stride=2)
- ReLU vrstva
- Konvolučná 1D vrstva - (in=16, out=32, kernel_size=5, stride=2)
- ReLU vrstva
- Adaptívna average pool vrstva

### 2.1.2 Klasifikácia

Samotný klasifikátor pozostáva z nasledujúcich vrstiev:

- Lineárna vrstva - (in=dimenzia vstupu, out=128)
- ReLU vrstva
- Dropout vrstva - 0.3
- Lineárna vrstva - (in=128, out=počet tried)

## 3.1 Kombinovaný klasifikátor

### 3.1.1 Vytvorenie vektoru vlastností

#### Obraz

Bola využitá predtrénovaná sieť na klasifikáciu obrazu typu resnet18. Sieť má štruktúru popísanú na obrázku.

| Layer Name | Output Size | ResNet-18 |
|---|---|---|
| conv1 | $112 \times 112 \times 64$ | $7 \times 7$, 64, stride 2 |
| conv2_x | $56 \times 56 \times 64$ | $3 \times 3$ max pool, stride 2 |
| | | $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$ |
| conv3_x | $28 \times 28 \times 128$ | $\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$ |
| conv4_x | $14 \times 14 \times 256$ | $\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$ |
| conv5_x | $7 \times 7 \times 512$ | $\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$ |
| average pool | $1 \times 1 \times 512$ | $7 \times 7$ average pool |
| fully connected | 1000 | $512 \times 1000$ fully connections |
| softmax | 1000 | |

Obr. 3.1: Resnet18 [1]

Extraktor bol získaný príkazom models.resnet18(pretrained=False). Posledná FC vrstva bola ale odstránená a nahradená vyrobeným klasifikátorom.

#### Audio

Sieť na extrakciu vektoru vlastností má rovnakú štruktúru ako pri samostatnom audio klasifikátore.

#### Výsledný vektor

Vektory získané z extraktorov boli stlačené na riadkové vektory a spojené pomocou konkatenácie. Takto získaný vektor bol vložený do klasifikátoru.

### 3.1.2 Klasifikácia

Bola použitá rovnaká architektúra klasifikátoru ako pri audio klasifikátoru.

## 4.1 Nastavenie trénovania

Obidva klasifikátory boli trénované s rovnakými nastaveniami, ktoré sú nasledujúce.

- Na výber dát z datasetu bola použitá trieda torch.utils.data.DataLoader, ktorá vytvárala skupiny (batch) požadovanej veľkosti - 32 audio, 8 kombinovaný klasifikátor
- Bol použitý optimalizátor Adam (torch.optim.Adam), ktorý bol nastavený na parametre konkrétnej siete, s rýchlosťou učenia (lr=1e-4)
- Bola použitá chybová funkcia krížovej entropie (nn.CrossEntropyLoss)

## 5.1 Použité skripty

- data_noise.py - vytvorenie zašumeného datasetu
- prog_simple.py - tréning klasifikátoru z audia
- prog.py - tréning klasifikátoru z audia a obrazu
- load_eval_model.py - načítanie modelu a jeho vyhodnotenie nad datasetom
- eval_fin_multi.py - použite klasifikátoru z oboch zrojov na vytvorenie výstupu
- eval_fin_single.py - použite klasifikátoru z audia na vytvorenie výstupu

# Literatúra

[1] Napoletano, P. a Piccoli, F. Anomaly Detection in Nanofibrous Materials by CNN-Based Self-Similarity. *Sensors*. Január 2018, zv. 18. DOI: 10.3390/s18010209.