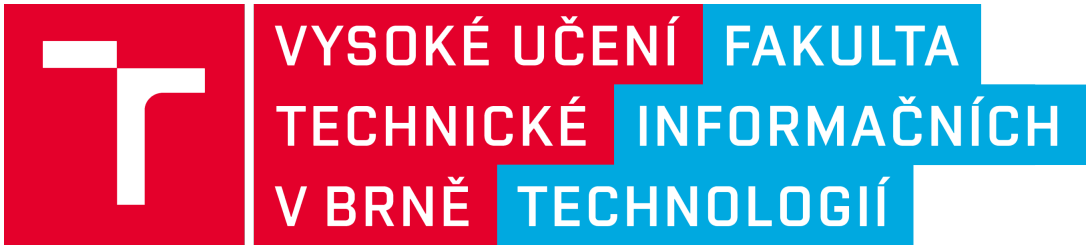


# Strojové učení a rozpoznávání

Dokumentace

Adam Kučík (xkucik00)

Robin Volf (xvolfr00)

May 5, 2025

# 1 Introduction

This paper contains documentation for the project SUR. The goal was to build a classifier for 31 people. The first section discusses the image classifier, followed by the voice classifier. Finally, the conclusion summarizes the achieved results.

# 2 Image classifier

For the image classification, the most common and straightforward approach was to train a convolutional neural network. However, the primary limitation of this approach was the limited size of the given dataset, which limited the power of the performance neural network. To address this problem, we used data augmentation to improve the generalization of the neural network. The training data was divided into training and validation sets. The dev data were used as testing data. In the following subsections, we will discuss in more depth the development process of the image classifier.

## 2.1 Used libraries and technology

The code for the image classifier was written in Python 3.8, using Jupyter Notebook. We used following libraries:

- **Numpy** for common mathematics operations
- **PIL** for image processing
- **Os, Pathlib and Shutil** for work with dataset
- **Pytorch** for training of neural network for image classification
- **Albumentations** for online augmentations during training
- other: **sklearn**, **matplotlib**

To run the code, all libraries need to be installed with *pip install*. By default, *image.ipynb* expects the *train* and *dev* directories to be located in the same directory as the notebook. However, the path can be manually changed in the code.

## 2.2 Data process

We started by processing the dataset. From the train data, we only kept png files, while audio files were removed. The training set was then divided into training and validation subsets. For each person's set of 6 images, the last 2 images were used as validation and the remaining were kept for training.

## 2.3 Data augmentation

To enlarge the limited dataset size, we augmented the training data. First, all images were horizontally flipped. Then for each original and flipped image, we generated 3 additional variants by applying gaussian noise, brightness reduction, and brightness enhancement. During training, we further applied more augmentations with the **Albumentations** library to dynamically apply random augmentations, including rotation, cropping, gaussian blur, color jitter, among others.

During experimenting, various augmentetions were applied, but aggressive augmentations of the neural network were unable to generalize, so we stayed with slight augmentations. Another insight into the future was not to include augmented validation data in training, which led to massive overfitting, 100 % validation accuracy vs 20 % test accuracy.

### 2.4 Neural network

The creation of a neural network required finding the right balance between its size and its ability to generalize patterns without memorizing the training data.

The best results were achieved by architecture that consist of 3 blocks, each with convolutional layers, batch normalization, ReLU and either max pooling or adaptive average pooling. To avoid overfitting, we added dropout layers with a rate of 0,3. Most of the time, we used CrossEntropy as activation, but due to task requirements, we switched to log-softmax activation, which provides better training stability and improved overall model results.

For optimization we chose stochastic Gradien descent, which outperformed Adam in our experiments. The training ran for 50 epochs, achieving 77.42 % validation accuracy and 74.19 % test accuracy. The model demonstrated solid generalization, with only a small accuracy gap, suggesting generalization. With additional non-augmented training data we might further improve results, extending training beyond 50 epochs showed no improvements.

### 2.5 Summary and potential improvements

Our best model achieved about 75 % accuracy on the test data. The model is not perfect, and there are several ways to improve its performance. We could try to optimize the model further by experimenting with alternative optimizers, fine-tuning hyperparameters, or implementing more sophisticated regularization techniques. As with most classification tasks, increasing the amount of real training data would likely lead to better results. This was achieved in some manners by data augmentations, but is not perfect. Probably the easiest way would be to use deep models with pre-trained weights, which was denied by the task. Overall, our model was able to perform the task to a certain extent.

## 3 Voice Classifier

For voice classification we decided to use a GMM model on Mel-Frequency Cepstral Coefficients computed from the input signal. First the signal is split into overlapping windows and then MFC Coefficients are computed on them using FFT and then DCT. This is then used as the input into a MAP (Maximum a-Posteriori) classifier which computes the likelihood of each class ($P(x|C)$) using a GMM. We are then left with a 2D matrix of class likelihoods for each window. Matrix is converted into a vector of mean likelihood for each class (mean class likelihood across all windows). This likelihood is then used in a classic generative MAP classifier to get $P(C|x)$ like this:

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)}$$

Where we infer $P(C)$ as a proportion of that class in the training data and $P(x) = \sum_C P(x|C)$.

### 3.1 Hyperparameter tuning

Model was trained and then validated on provided data (`dev/` and `train/` datasets). Hyperparameters were tuned on validation data by training the model with given hyperparameters and then measuring the accuracy on the validation dataset. In the end, the best accuracy ($\approx 74\%$) was achieved with 5 gaussians per class with 100 iterations of EM (Expectation-Maximization) to train the GMM.

Included ZIP file contains a README (`SRC/sound/README.md`) outlining how to compile and use the voice classifier. MFCC computation and GMM model were implemented from scratch in Rust using the ndarray library and its extensions for linear algebra, FFT and DCT (Discrete Cosine Transform) computation.

# 4 Conclusion

In this documentation, we discuss our approaches for image and voice classification. Both classifiers achieved an accuracy of approximately 70 %. The image classifier was implemented by the convolutional neural network, and the voice classifier was implemented by the GMM model with MFCC features.