

SUR PROJECT 2021/22

 **Skuratovich Aliaksandr**

2BIT student.

Brno University of Technologies
Brno, Božetěchova 1/2
xskura01@vutbr.cz

 **Tikhonov Maksim**

2BIT student.

Brno University of Technologies
Brno, Božetěchova 1/2
xtikho00@vutbr.cz

April 27, 2022

ABSTRACT

This technical report describes the implementation of three classifiers implemented for solving the problem of audio-visual person verification on small unbalanced datasets. The first part of the project describes data augmentation and an attempt to balance the datasets. The second part describes the methods used for audio-visual person verification.

You can find the source code and the installation guide in the github repository¹ with the project, as well as a `Readme.md` file with a tutorial on how to run the experiments and reproduce the results.

Tikhonov Maksim: BGMM; Audio dataset processing.

Skuratovich Aliaksandr: Image classification; Image dataset processing.

¹<https://github.com/SkuratovichA/SUR>

Contents

1	Introduction	3
1.1	Person verification task	3
1.2	Assignment	3
2	Image dataset	4
2.1	Data preparation	4
3	Audio dataset	6
3.1	Data preparation	6
3.1.1	Trimming audios	6
3.1.2	Mel frequency cepstral coefficients	7
3.1.3	Environmental sound augmentations	8
4	Implemented classifiers	9
4.1	Audio	9
4.1.1	Maximum a-posteriori classifier based on Bayesian Gaussian Mixture Models	9
4.2	Images	10
4.2.1	Binary classifier using CNN	10
4.3	Trained Models	10
4.4	PCA and NN	11
5	Conclusion	12

1 Introduction

1.1 Person verification task

Automatic verification of a person by photo or voice is a technology that can find many practical applications. Starting from the fight against terrorists and finding recognition in the school dataset, ending with the state persecution of people who disagree with the political regime.

1.2 Assignment

The task was to train the detector of a particular person based on face images and voice recordings. There were given training and testing data of both types. We implemented three separate models. The first is described in the section subsection 4.1.1. The second is described in the section subsection 4.2.1. The third is described in the section subsection 4.4.

The whole project was implemented in Python. The most used libraries in this project are `Torch Lightning`, `Numpy`, `Speechbrain`.

2 Image dataset

Image dataset is split into train and dev parts. train part contains 120 images for non-target people, and 20 images for the target target person. Each sample in the dataset is a square 3-channel RGB image with sizes of 8080 pixels.

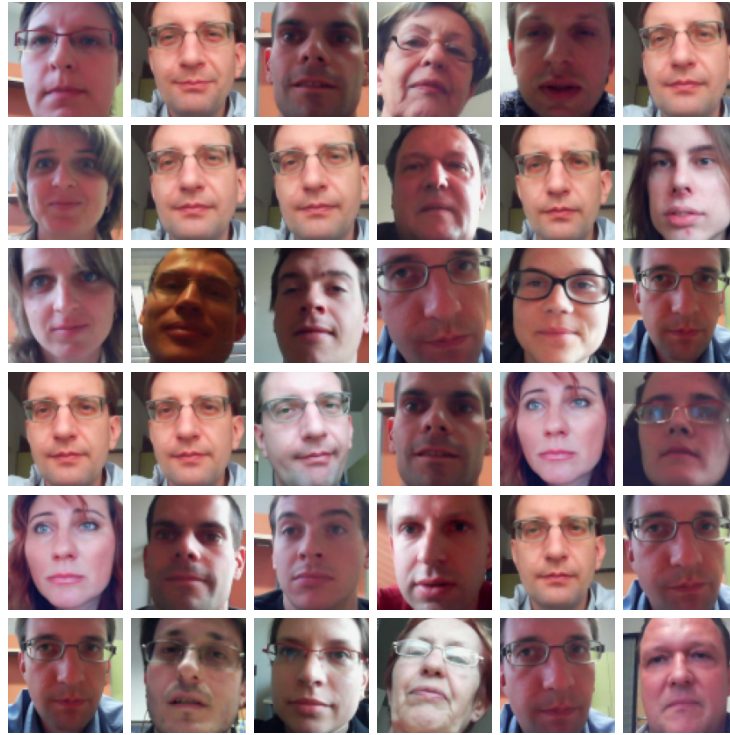


Figure 1

It is clearly seen, that the dataset is unbalanced and too small. Even simple neural networks need thousands of images to train on. The solution can be to use a pre-trained model. However, it was forbidden by the assignment. Therefore, there was a need to augment image data.

2.1 Data preparation

Because two models for images were trained, the used augmentations were slightly different.

The first one is a data augmentation pipeline without rotating images.

```
from imgaug import augmenters as iaa
import torchvision.transforms as transforms
iaa.Sequential([
    iaa.Resize((80,80)),
    iaa.Sometimes(0.8, iaa.GaussianBlur(sigma=(0,2.0))),
    iaa.Sometimes(0.8, iaa.PerspectiveTransform(scale=(0.0, 0.06))),
    iaa.Fliplr(0.8),
    iaa.Sometimes(0.5, iaa.Affine(rotate=(-3, 3), mode='edge')),
    iaa.Sometimes(0.8, iaa.AddToHueAndSaturation(value=(-10,10), per_channel=True)),
    iaa.Sometimes(0.2,
        iaa.blend.Alpha((0.0, 1.0),
            first=iaa.Add(20), second=iaa.Multiply(0.6))
    )]).augment_image,
np.ascontiguousarray,
transforms.ToTensor(),
transforms.Grayscale(num_output_channels=1)])
```

The dataset after augmentations looks as follows:



Figure 2

The second one was prepared for a Convolutional Neural Network. Therefore, there could be used more additional transformations, such as rotation or dilatation.

Here is a list of transforms being applied to each image from the dataset.

```
from imgaug import augmenters as iaa
iaa.Sequential([
    iaa.Resize((80,80)),
    iaa.Sometimes(0.8, iaa.GaussianBlur(sigma=(0,2.0))),
    iaa.Fliplr(0.9),
    iaa.Sometimes(0.8, iaa.Affine(rotate=(-65,65), mode='edge')),
    iaa.Sometimes(0.8, iaa.PerspectiveTransform(scale=(0.0, 0.10))),
    iaa.Sometimes(0.8, iaa.AddToHueAndSaturation(value=(-20,20), per_channel=True)),
    iaa.Sometimes(0.8,
        iaa.BlendAlpha((0.0, 1.0), foreground=iaa.Add(20), background=iaa.Multiply(0.6))
    ),
    iaa.Sometimes(0.1, iaa.SomeOf((0, 2), [
        iaa.Sharpen(alpha=(0, 0.7), lightness=(0.75, 1.5)),
        iaa.Emboss(alpha=(0, 0.7), strength=(0, .5)),
    ]))) .augment_image,
    transforms.Grayscale(num_output_channels=1)])
```

Hence, the transformed dataset looks as follows:

It is seen, that some images are rotated. We have done that, because of the hope, that such an amount of data will help a Neural network learn the underlying features of each person and classify them better.

Also, two csv files were created, to simplify the data preparation pipeline using pytorch lightning data modules².

²<https://pytorch-lightning.readthedocs.io/en/stable/extensions/datamodules.html>



Figure 3

3 Audio dataset

There were given 20 and 120 train (10 and 60 *test*) audio files for target and non-target classes accordingly. All audio recordings from the dataset are sampled with 16kHz. The amount of data provided by the training dataset was considered insufficient, so was decided to double the number of recordings by environmental augmentations. The resulting number of target recordings is 40, and the number of non-target recordings is 240. The length of processed (noise cut, augmented) data is around 210 seconds for target and is around 2300 seconds for non-target. Based on this it is taken into account that the given dataset is unbalanced (since we only have two classes).

3.1 Data preparation

For the training pipeline, there was a need to prepare audio data. This chapter describes this process. First of all, there was a need to trim the audio files. Then, parts of the signal without noise were cut and data was augmented. After that, Mel-frequency cepstral coefficients were extracted and normalized.

3.1.1 Trimming audios

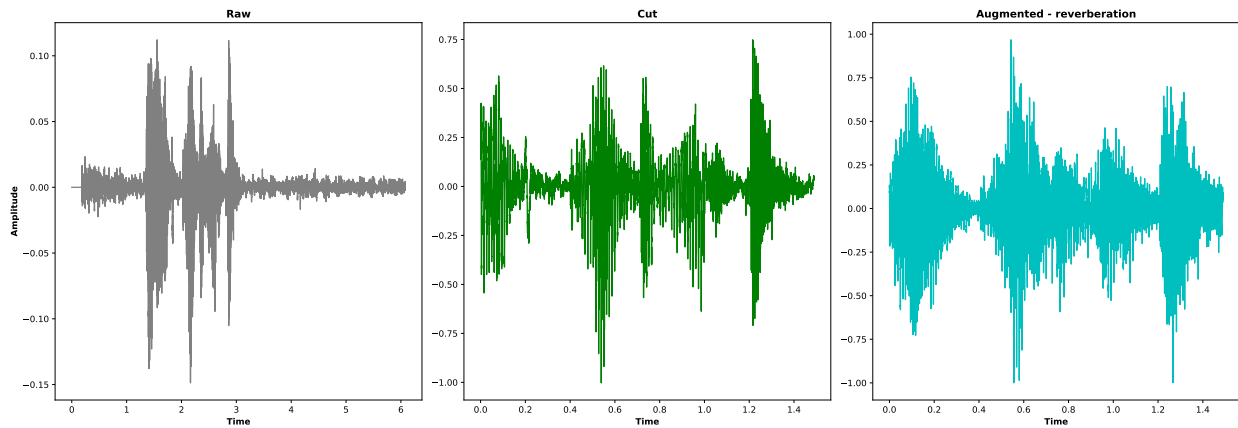
Because the task was to verify the target person, exceeding noise without a speech of a person could be harmful to the accuracy of the statistical model.

Trim the first seconds The microphone the audio was recorded on had to "charge" at the start of each recording, so was decided to trim the first 1.6 seconds of each .wav file, which is 26000 samples.

Voice activity detection There is the Voice Activity Detector³ to trim the remaining noise at the end of the voice line. It divides recording on non-overlapping frames and evaluates them by comparing the mean value of the given frame and the threshold derived from the difference between the maximum and the minimum values of the frame.

```
frame = np.array(_frame) ** 2.
threshold = 0.1 # adaptive threshold
thd = np.min(frame) + np.ptp(frame) * threshold
VADthd = (VADn * VADthd + thd) / float(VADn + 1.)
VADn += 1.
if np.mean(frame) <= VADthd:
    silence_counter += 1
else:
    silence_counter = 0
if silence_counter > 20:
    result = False
```

The following graphs are representing the audio processing pipeline. The first graph represents the raw signal extracted from the given .wav file, the second - cropped signal with only speech, third is an augmented (reverberated) signal. Both second and third signals are going to be in the final dataset for training.



3.1.2 Mel frequency cepstral coefficients

For each audio recording, we have twenty Mel-frequency cepstral coefficients. Which is often derived as follows:

1. Break the signal into overlapping frames
2. Take the (Fast) Fourier transform of those frames.
3. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
4. Take the logs of the powers at each of the mel frequencies.
5. Take the discrete cosine transform of the list of mel log powers as if it were a signal.
6. The MFCCs are the amplitudes of the resulting spectrum.

MFCC function from library `librosa` was used for the feature extraction.

³<https://github.com/mauriciovander/silence-removal>

Normalise MFCC

$$MFCC_{norm} = \frac{MFCC - \mu}{\sigma}$$

Where:

- μ is a mean value of MFCC
- σ is a standard deviation of MFCC

3.1.3 Environmental sound augmentations

As was said before, sound augmentations were used both for the expansion of the dataset and for better results on the evaluation set (was mentioned that data will also have augmentations). All of augmentations are applied "randomly" - one augmentation per .wav file.

Types of environmental augmentations used:

- Speed change.
- Add reverberation.
- Add crowd noise.
- Add random noise.

Speed change was implemented by SpeedPerturb function from SpeechBrain's augmentation module. May change the speed of the input signal by any of these factors: 0.9, 0.95, 1.05, 1.1. Has occurrence chance of 20%.

Reverberation addition was implemented by Reverb function from SpeechBrain's augmentation module. The function convolves the signal with an impulse response, so it creates the echo-like sound file. May have various room impulse response factor (0.5, 1, 1.5), so the impulse response may be compressed (< 1) or dilated (> 1). Is considered the most natural of all augmentation (echo in sound files, just as random noise, is widespread) so it has an occurrence chance of 40%.

Crowd noise addition was implemented by AddNoise function from SpeechBrain's augmentation module. The crowd noise audio is also provided by this toolkit. Has occurrence chance of 20%.

Random noise addition was implemented by the simple addition of a randomly generated signal of the same length as the input signal. This augmentation was also considered one of the most common (just as reverberation) but since the input sound already has some noise, was decided to lower the occurrence chance to 20%.

4 Implemented classifiers

4.1 Audio

4.1.1 Maximum a-posteriori classifier based on Bayesian Gaussian Mixture Models

We used a Maximum a-posteriori classifier to verify the target person. The intuition behind it is the following: we compare posterior probabilities of each class given a sequence of MFCC coefficients. We use $p(C_{\text{target}}|F)$ for the probability of Target class given MFCC coefficients, and $p(C_{\text{non-target}}|F)$ for the probability of non-target class given MFCCs. Then, we can rewrite it using Bayesian rule and compare two probabilities:

$$\frac{p(F|C_{\text{target}})p(C_{\text{non-target}})}{\sum_n p(F|C_n)} > \frac{p(F|C_{\text{non-target}})p(C_{\text{non-target}})}{\sum_n p(F|C_n)}$$

We can get rid of the denominator on both sides because it is a positive number. Hence, we got.

$$p(F|C_{\text{target}})p(C_{\text{target}}) > p(F|C_{\text{non-target}})p(C_{\text{non-target}})$$

For numerical stability, we take a logarithm of each side, because a logarithm is a monotonically increasing function, and it will not change the result of inequality.

$$\log p(F|C_{\text{target}}) + \log p(C_{\text{target}}) > \log p(F|C_{\text{non-target}}) + \log p(C_{\text{non-target}})$$

Then, move everything to the left side, we get:

$$\log p(F|C_{\text{target}}) + \log p(C_{\text{target}}) - \log p(F|C_{\text{non-target}}) - \log p(C_{\text{non-target}}) > 0$$

We model log-likelihood $\log p(F|C_{\{\text{target}, \text{non-target}\}})$ using Gaussian Mixture Model from the `sklearn` library.

The number of components used for target person and non-target people is 1 and 1 accordingly (apparently, the optimal numbers of components are that low due to the low number of data).

Probability density function of the Normal (Gaussian) distribution is given by:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right\}$$

where $\boldsymbol{\Sigma}$ is a covariance matrix, $\boldsymbol{\mu}$ is a vector of mean values, and \mathbf{x} is an input vector.

Gaussian mixture model is then given by an equation:

$$p(\mathbf{x}|\theta) = \sum_c \pi_c \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

where θ is $\{\pi_c, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$.

The Gaussian mixture model was trained using the Expectation-maximization algorithm. You can learn more about in the book (2). Also, a custom loss function was used to control the accuracy of the MAP classifier: Created loss function looks as follows:

$$\frac{|\mu_t - \mu_n|}{\sigma_t + \sigma_n} \cdot (a_t \cdot a_n)^{20},$$

where

$\mu_{t,n}$ and $\sigma_{t,n}$ are a mean and standard deviations for predictions on the target and non-target validation sets.

$a_{t,n}$ are accuracies ($\frac{\text{\#true predictions}}{\text{\#false predictions}}$) on target and non-target validations set.

The power was chosen heuristically to bias the accuracy metric.

The purpose of this loss function is to increase the distance between mean values and decrease standard deviations for predictions for target and non-target persons on the validation set.

Even though both the Gaussian mixture model and the Bayesian Gaussian mixture model have shown relatively small differences in terms of classification the described loss function has demonstrated that the Bayesian Gaussian mixture model is a bit better than the common one: the latter has a bigger variance.

Why the results are so good? - Because of a good augmentation, the specific voice of the target person, and the overall simplicity of the task. The final model was trained on both train and validation data.

4.2 Images

Pytorch implementation `torch.nn.BCELoss()` of a Binary Cross-Entropy loss was used for training both models. Binary cross-entropy is defined as following:

$$L_{\text{BCE}} = -\frac{1}{N} \sum_{n=1}^N (t_n \cdot \log(y_n)) + (1 - t_n) \log(1 - y_n)$$

where y_n is an output from a sigmoid activation function, and t_n is a ground truth.

Both models were trained using google colaboratory platform⁴.

Library pytorch lightning⁵ was used both to prepare the data and train the model.

WandB (Weights and Biases) (1) toolkit was used for watching the training results and comparing models by looking at the produced metrics.

4.2.1 Binary classifier using CNN

Convolutional networks (4), also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. (3)

4.3 Trained Models

Several models were trained using CNN architecture. After CNN layers, there are feed-forward layers and a Sigmoid activation function at the end. Here, you can see WandB metrics for different models.

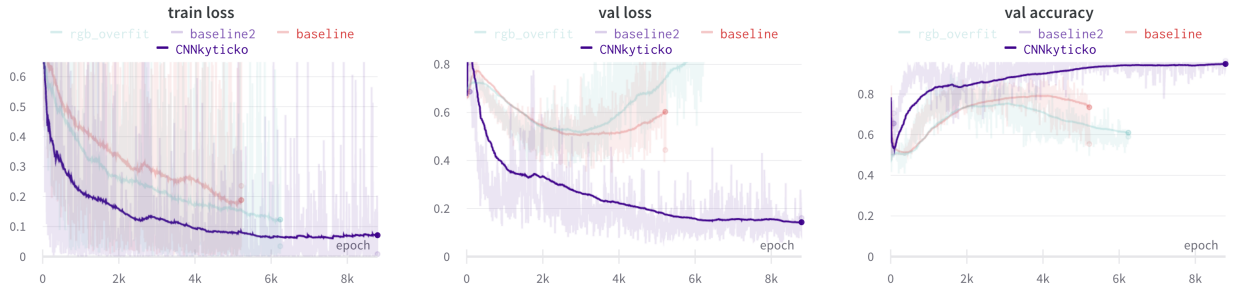


Figure 4: Learning curves of trained models

Surprisingly, the model trained without overfitting and showed good results on the validation set. The model consisted of four CNN blocks. Where each layer is represented by a relation $in_channels \rightarrow out_channels$. $1 \rightarrow 8$, $8 \rightarrow 16$, $16 \rightarrow 32$, $32 \rightarrow 64$. Each block had a MaxPool2d layer after the activation function `torch.nn.GELU()`⁶, batch normalization for better learning, and dropout to prevent overfitting. This was followed by a linear layer $1600 \rightarrow 128$ with the same activation function, and then another linear layer with the Sigmoid activation function to represent the probabilities.

“Small batches can offer a regularizing effect (Wilson and Martinez, 2003), perhaps due to the noise they add to the learning process. Generalization error is often best for a batch size of 1. Training with such a small batch size might require a small learning rate to maintain stability because of the high variance in the estimate of the gradient. The total runtime can be very high as a result of the need to make more steps, both because of the reduced learning rate and because it takes more steps to observe the entire training set.” (3)

Thus, the model was trained for 7000 epochs with a surprisingly low learning rate of $8e10 - 8$ and a batch size of 1 for the sake of better generalization. A `torch.optim.RMSprop()`⁷ optimizer. A `weight_decay` regularization with the value 0.00156, and a momentum of 0.999 were used in the optimizer.

⁴<https://colab.research.google.com>

⁵<https://www.pytorchlightning.ai>

⁶<https://pytorch.org/docs/stable/generated/torch.nn.GELU.html>

⁷<https://pytorch.org/docs/stable/generated/torch.optim.RMSprop.html>

4.4 PCA and NN

For the sake of interest, we trained a two-layer model with feature extraction using PCA (2). After dataset augmentation presented in subsection 2.1. Principal Component Analysis was used to reduce dimensions and simplify the classification. PCA was performed during the stage of the data preparation step and the procedure of transforming the data looks as follows:

```
def pca_transform(datas, U, mean):
    d_U = torch.tensor((datas.numpy() - mean.numpy()).dot(U.T.numpy()))
    return d_U
_, _, U = torch.linalg.svd(train_d, full_matrices=False)
train_d, train_l = get_images('train.csv', is_train=True)
val_d, val_l = get_images('dev.csv', is_train=False)
```

Then, the data was passed to a neural network. Several configurations were trained, and you can see the metrics on the graphs.

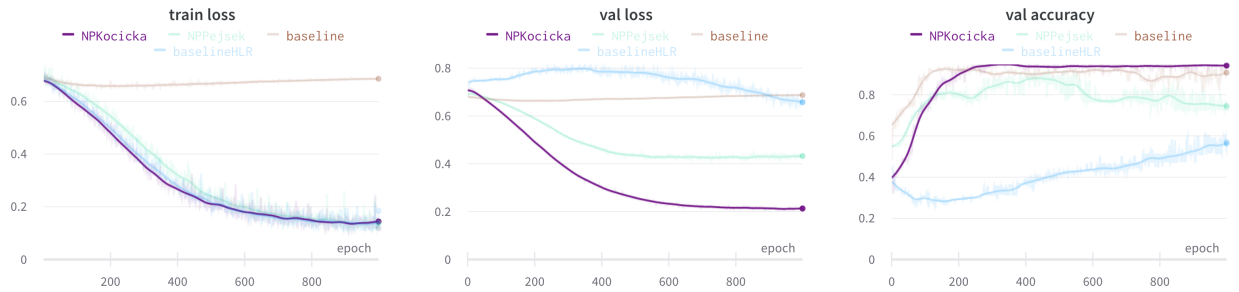


Figure 5: Learning curves of trained models

The best model consists of two feed-forward layers. Before the first feed-forward layer, the `torch.nn.BatchNorm1d`⁸ was used to normalize the input batch. The purpose was to normalize the input vector, in which most of the values were small compared to others. Hence, they could be reduced more. After the batch normalization, there were two feed-forward layers.

The first layer had the input shape of 319, which is the number of features returned by a PCA and a Sigmoid activation function.

The second layer consisted of 32 input neurons and one output neuron. Neuron output is then passed to a Sigmoid activation function.

We used a Binary Cross Entropy loss function from `pytorch`⁹ as a loss function.

`Adam`¹⁰ was chosen as an optimizer with learning rate 0.00009, weight decay 0.01, and amsgrad. This optimizer with these parameters showed the best results.

⁸<https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html>

⁹<https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html>

¹⁰<https://pytorch.org/docs/stable/generated/torch.optim.Adam.html>

5 Conclusion

The results of photo verification showed that with the choice of the correct hyperparameters as an extremely low learning rate, even a fairly global model prone to overfitting on such an unbalanced dataset can show adequate results while training. Also, good results were achieved by massive augmentations of an image dataset. This, together with dataset upsampling (simply cloning images from the target set), helped to achieve compatible results.

References

- [1] BIEWALD, L. *Experiment Tracking with Weights and Biases*. 2020. Software available from wandb.com. Available at: <https://www.wandb.com/>.
- [2] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1. vyd. Springer, 2007. Available at: <http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738>. ISBN 0387310738.
- [3] GOODFELLOW, I., BENGIO, Y. a COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [4] LECUN, Y., BOTTOU, L., BENGIO, Y. a HAFFNER, P. Gradient-Based Learning Applied to Document Recognition. In: *PROCEEDINGS OF THE IEEE*. 1998, s. 2278–2324.