# Emotion Recognition In Images From EEG Signals Using Deep Neural Networks

Paweł Figiel, Michał Jakóbczyk, Kamil Janas, Krzysztof Lelonek and Dawid Pieła

**Abstract**—Electroencephalography is a method of measuring electrical activity of brain. It can be used in many fields, as medicine or human-machine interface. But collecting data is just a first step, then they must be interpreted by some software. In this document there is described an experiment, which aim was to measure possibilities of EEG data analysis and emotion recognition using neural network. In this experiment there was used convolution neural network and EMOBRAIN database to train and test it. Results received were below 50% accuracy, what could be caused mainly by low quality of used data. Experiment indicated that high quality and well preparing of data can have key impact for effectiveness of neural network and classification at all.

## 1 Introduction

Living brain generates electric field depending of its current activity, physical state, emotions, etc. Measuring that electrical brain activity enables "brain reading". It can be used in many fields, e.g. medicine (brain diseases diagnosis), criminology (lie detector), human-machine interfaces (especially for paralyzed people). Reading electric signals from brain is possible with using electroencephalography (EEG). Weakness of EEG is that the noninvasive variant is very sensitive for any noises, so it is practically useless in normal, not laboratorial, environment.

Important challenge for EEG is analysis and interpretation of collected data. One of possible approaches to this issue is using neural network to analysis of EEG data. The aim this project is an examination of effectiveness this approach by making an experience of emotion recognition using EEG data and neural network. There will be designed and implemented convolution neural network. Data for training and testing the network will be received from EMOBRAIN database. This work uses Python to implementation of the research instrument. To implement the network there will be used Keras and Tensorflow libraries and for preprocessing and extraction data from EEG files there will be used MNE library.

## 2 Literature review

This literature review contains data gathered from the following articles (each of them called later as stated in bold):

- EEG-based Emotion Recognition, 2006 [1] - **Article I**
- EEG Databases for Emotion Recognition, 2013 [2] - **Article II**
- Evaluating classifiers for Emotion Recognition using EEG, 2013 [3] - **Article III**
- EEG-based subject independent affective computing models, 2015 [4] - **Article IV**
- Evaluation of Classifiers for Emotion Detection while Performing Physical and Visual Tasks: Tower of Hanoi and IAPS, 2019 [5] - **Article V**

*Manuscript received June 26, 2019.*

### 2.1 Performed experiments

#### *Article I*

Protocol of test:

- Subject will be instructed about what will happen during test.
- Subject will be exposed to short test run with 3 neutral stimuli.
- During 9 minutes, 36 additional stimuli will be presented, five seconds of stimuli exposure will be followed by 10 seconds of cooling down. Number of test subjects will be limited to five people, with varying gender, age and background.

User guidelines during the stimulus presentation:

- Try not to blink, move your eyes, or move any other part of body.
- Try to stay relaxed and do not tense up.
- Keep your eyes open.

Instructions during cooling down period:

- Blink and move. Relax your shoulders, jaw, neck, face. Do not damage equipment.
- Count down mentally with the numbers presented on the screen.
- Keep your gaze steady on the center cross, where stimulus will happen during final seconds of cooling down.

#### *Article II*

In the experiment the Emotiv device had been used. It contained 14 electrodes located in specific points, based on the American Electroencephalographic Society standard. The most important configuration aspects have been:

- bandwidth: 0.2-45Hz,
- digital notch filters at 50Hz and 60Hz,
- 16-bit AD converters,
- 128Hz sampling rate.

Both experiments - audio and visual - had been organized into sessions. For audio experiment, each session consisted consecutively of 12 seconds of silence, followed by 5 to 6 sound clips and ending with time for self-assessment. A session in the visual experiment started with black screen, followed by displaying a cross for concentration purposes.

*Article III*

There were 20 participants of the experiment. They were asked to watch at 30 emotional related pictures presented sequentially and assess their emotional state (Self-Assessment Manikin) in two dimensions: valence (positive/negative) and arousal (calmness/excitement). In the same time they were examined by EEG. Pictures used in experiment came from IAPS base and were chosen equally from all the emotion cluster (6 for each: neutral, positive arousing/ calm, negative arousing/calm).

*Article IV*

The group consisted of 26 females. Brain activity was registered by ERP's recording. Images shown to them were eliting either pleasant or unpleasant emotions. Data itself was taken from IAPS repository (International Affective Picture System). Experiment covered 24 different photos with varying rating (positive and negative). Each picture was presented 3 times for 3500 ms in a pseudo-random order. Signals coming from brain were recorded from 21 channels, sampled at 1kHz.

Raw brain signals were:

- filtration using band-pass filter (between 0.1 to 30 Hz),
- eye-movement correction,
- baseline compensation,
- segmentation using epochs using NeuroScan.

*Article V*

Tower of Hanoi is used because it causes strong emotions to appear. Various systems for emotion classification exist. On of them is Pluchtik's one, which considers eight primary states – acceptance, anger, anticipation, disgust, fear, joy, sadness and surprise. In this research though, a two dimensional model, involving valence (pleasure) and arousal (excitement), was used. In the experiment, there participated 20 students of Blekinge Institute of Technology in Sweden, aged between 21 and 35 years, coming from different nationalities, cultures and fields of studies. They were asked to solve the Tower of Hanoi puzzle in which they have to move the tower of growing disks from left pig to right, using the middle one as a support. Only one disc can be moved at the time and a bigger disc can never be on top of a smaller one. The subjects did the task twice, once with 4 discs and once with 5 discs, half in this order, and half in the opposite order, to avoid sequencing effect. There was 5 minutes for each of the two tasks.

## 2.2 Preprocessing

*Article I*

Bandpass filter provided by EEGLab for Matlab was used, fourier frequency analysis signal can be split up in frequencies. Specific frequencies can be removed and the signal can be transformed back, containing only frequencies of interest. At this point alpha and beta bands were available from recorded channels. As it was unknown what combinations would provide the best classification results, all interesting possibilities were tested for channels. Principal component analysis was applied to reduce total number of features from 1000 to 1-25.

*Article II*

The process of processing the EEG data consisted of several steps. First, the raw data has been filtered by a 2Hz high-pass filter. The first 5 seconds of data, recorded before presenting the stimuli, was considered a baseline. Then a 3-47Hz filter with Welch's method has been applied. Next the baseline power has been subtracted from the data with emotions in order to obtain the relative change of power during the exposure. Then mean changes of power for alpha(8-13Hz), beta(14-29Hz), gamma(30-47Hz) and theta(3-7Hz) signals have been computed. Finally the Spearman correlation coefficients between the power changes and self-assessment ratings for each subject have been calculated. This resulted in the following observations:

- For arousal dimension, a negative correlation in theta, beta and gamma bands has been observed,
- For valence dimension there was a positive correlation in gamma band,
- For dominance dimension, there was a positive correlation between it and the beta band of the frontal lobe electrode.

*Article III*

After collecting data a few results with low emotional ratings were rejected, so data from 15 subjects remained. In the next step signals connected with all the particular emotions were extracted from EEG data, noise, gaps and other artifacts were removed. Finally features were selected according to theirs correlations with emotional ratings. Features were extracted such techniques as Fourier transform, wavelet transform, thresholding, and peak detection and were based on signals from six electrodes and theirs extreme values, mean and standard deviations. Prepared data were used to train classifiers using 10 fold cross validation.

*Article IV*

Initial step to reduce big variability between subjects was to get averaged signals of all positive and negative trials per subject. The averaged ERP signals were filtered using a Butterworth filter of 4th order with passband [0.5-15] Hz. ERP-based detection systems rely on the frequency content of signals.

*Article V*

The data was preprocessed using EEGLAB Matlab Toolbox to extract Epoch and Event information, as well as Indepentent Component Analysis. These should help in removing artifacts caused by eye blinking or system impedance, and also make feature extraction easier.

TABLE 1
Classifiers used in articles

| | Article I | Article II | Article III | Article IV | Article V |
|---|---|---|---|---|---|
| ANN | | | 🟩 | 🟩 | 🟩 |
| LogReg | | | | 🟩 | |
| LDA | | | | 🟩 | |
| kNN | | | 🟩 | 🟩 | 🟩 |
| NB | | | | 🟩 | |
| SVM | | 🟩 | 🟩 | 🟩 | 🟩 |
| DT | | | | 🟩 | |
| BN | | | 🟩 | | 🟩 |
| FDA | 🟩 | | | | |

### 2.3 Classifiers

Classifiers used in articles are marked as green in an appropriate cell in table 1.

Full names:

- Artificial Neural Networks (ANN)
- Logistic Regression (LogReg)
- Linear Discriminant Analysis (LDA)
- k-Nearest Neighbors (kNN)
- Naive Bayes (NB)
- Support Vector Machine (SVM)
- Decision Tree (DT)
- Bayesian Network (BN)
- Fisher's Discriminant Analysis

### 2.4 Conclusions

*Article I*

Literature research pointed to alpha and beta frequency bands for emotion recognitions. Various combinations of the two were tested for each of channels. For each of the different classifications (modality, arousal, valence) different features could give the best results. Modality: compared to the results of arousal and valence classifiers, modality classification is apparently more difficult, 82.1% classification rate. Arousal: 3 errors on 39 classifications, 92.3% performance. Valence: Performance is good for all channels. Highest performance rate 94.9%. Binary modality classification rate of over 80% seems attainable. Visual stimuli appear more difficult to classify than their audio and audiovisual counterparts.

*Article II*

The special case of happy (positive, high arousal, high dominance) and frightened (negative, high arousal, low dominance) emotions has been studied independently using spatial pattern of Fractal Dimensions. As a result it has been observed, that the right hemisphere of negative emotions, when experiencing fear, is more active than the left one. Moreover this was true for both audio and visual experiment. The data gathered during the experiment has been analyzed using SVM supported by Higher Order Crossing, Fractal Dimensions and statistical classifiers, in different combinations. This method has been applied to both the data collected during the experiment and to the benchmark DEAP database. The aim was to check the accuracy of emotion predictions using different methods and to check if these methods are dependant on stimuli type or measuring device used. It has been concluded that the best results in terms of prediction accuracy are obtained using a combination of HOC, FP and statistical classifiers (87% for 2 emotions). It has been also observed that a greater amount of measuring channels improves the accuracy of the prediction (90% for 32 channels and 2 emotions as opposed to 88% for 16 channels and 2 emotions). An important fact has been that the results were consistent between experimental and benchmark databases, proving that the proposed analysis method is stimuli- and device-independent.

*Article III*

The best results were obtained for SVM classificator (accuracy 56.10%) and all the rest results were oscillating around 50%. After dividing data sets (5 subjects each) to three subsets results were better for two subsets (up to 78%) and worse for the third. Still the best results were obtained for SVM in every subset, and the second classificator was kNN. According authors the difference in results for whole set and subsets is caused by problem with selecting enough general features, not connected with subjects specific reactions. There were executed one more experiment for single subject subsets and results were up to 83% (for kNN).

*Article IV*

The most important task is to select subject specific features. Sequential Feature Selection (SFS) provides the most confident features, but is computationally heavy procedure that may take too long to finish. On the other hand, empirical feature elimination is a compromise between computational time and discrimination performance, because effects can be obtained very short but the they will not be as accurate as in case of SFS. All in all, both methods are consistent in selecting the parietal and occipital channels are better encoders of the class information and less variable across subjects. The topic of emotion valence recognition

*Article V*

70% of the subjects considered the Tower of Hanoi problem with 4 disc hard, while 30% treated them easy. On the other hand, the 5 disc version was evaluated hard by 80% of subjects and easy by 20%. SVM proved to be the best at classifying EEG data with specific emotional states with accuracy of 70%, followed by RT with 60% accuracy. The study shows that focused brain activities on one goal may yield better accuracy than diverse activities on diverse goals.

### 2.5 Summary

All listed articles covered creating model for performing emotion recognition. Each article used different classifier (or set of classifiers) for this purpose. The most distinguishing cases were for Article I and Article II. In Article I only Fisher's Discriminant Analysis classifier was used. Calculcated performance in terms of arousal and valance oscilated between 92.3% and 94.9%. In Article II data was analyzed using SVM, giving accuracy around 87%. It turned out that a greater amount of channel improves the accuracy of the prediction. In addition results indicated stimuli-

and device-independency, which is an important feature. In contrast, SVM classifier in Article III offered an accuracy of around 56,10%. Although it is not a very high score, among other tested classifiers that was the best method. The best results for SVM was also obtained in Article V with 70% accuracy. Only in Article IV approach using Sequential Feature Selection gave satisfying results but at a cost of heavy computational power required.

## 3 MATERIALS

### 3.1 Database

In the database there are results of research conducted on a group of 5 patients. Each patient participated in 3 sessions. During each session, the patient was shown 30 blocks of pictures. Each block constisted of 5 pictures and each picture was shown for about 5 seconds. After each block there was a short break and the patient evaluated their arousal and valence feelings. All pictures in each block were meant to cause similar arousal and valence feelings. After each block, the patient evaluated their valence and arousal feelings. During each session the patient had their EEG recorded.

In the EEG directory, there is a .bdf file for each session of each patient. Each contains data from 72 channels with sampling frequency 1024 Hz, except the first patient, where it is 256 Hz. Also, for each session, there is a .mrk file with numbers of samples at which each block of the session started and finished.

In the Common directory there are files with arousal and valence evaluation of each patient after each picture block. Also, there are files with names and arousal and valence values of each picture used in the research, as well as classes corresponding to each block presented to each patient.

In the fNIRS directory there are fNIRS files corresponding to each session conducted during the research. There is also a MATLAB code that can be used to load this data for further analysis.

### 3.2 Data Usefulness Analysis

During analysis of data some failures occurred while trying to read EEG data for 2nd and 3rd session for first subject, so these data were considered as corrupted. Moreover 1st and 2nd series for first patient have the same assessments and timestamps, and 1st series for second patient has wrong, too high, amount of timestamps, so they both were rejected too. As a result there were used only 11 data series.

Eight last channels of data were rejected because according to database description they contain data coming from other than EEG sensors, e.g. Skin Response or Stimuli Occurance.

In experiment there were used data from 64 EEG channels, marker files and assessment of image blocks. FNIRS data and assessments from IAPS database weren't used.

## 4 METHODS

In the experiment data from base were read and extracted from files. Image assessments were transformed into three classes and EEG series were filtered and transformed from time domain to frequency domain. Received data were divided into two sets in 4:1 ratio: training and testing set. Then a neural network was created and trained with training data. Finally the accuracy of the network was measured using test set.

### 4.1 Preprocessing

Preprocessing of EEG data consisted of four parts:

- Extracting sections for individual stimuli according to markers
- Filtering data for EEG-useful frequencies, i.e. 1-100 Hz
- Transforming data to frequency domain
- Removing unused channels
- Standardizing data

To extract data from .To extract data from .bdf files and other operation on these files MNE library was used. Marker files were read as text data. Finding and cutting out segments from data was implemented in `cvapr_data` module. Transforming data to power spectrum was originally implemented by FFT transform but because of inacceptable time of working finally MNE feature was used. All preprocessing steps were processed in some different orders to experimentally find the best one. Finally they were performed in the following order: extracting sections, removing channels, transforming to power spectrum and standarizing.

For representing EEG data in program, a class `PictureBlockData` was defined. That class delivered features such as accessing to raw EEG and corresponding assessment, filtering data and transforming it to frequency domain.

Preprocessing of image assessments was significantly easier. Its only step was to transform two dimensional numerical values to three discrete classes: calm, positive and negative.

### 4.2 Neural Network Models

In the experiment there were used convolution neural networks. Some different models were created and tested. All data were provided as 1D data, so networks use only 1D layers. Structures of networks are described below:

- Network 1
  - Convolution layer with 64 filters of size 11
  - Convolution layer with 128 filters of size 7
  - Convolution layer with 256 filters of size 3
  - Flattening layer
  - Full-connected layer with softmax activation function
- Network 2
  - Convolution layer with 128 filters of size 3
  - Flattening layer
  - Full-connected layer with sigmoid activation function
- Network 3
  - Convolution layer with 128 filters of size 3
  - Flattening layer

TABLE 2
Network accuracies

| Model | Mean accuracy |
|---|---|
| Network 1 | 45.57% |
| Network 2 | 43.19% |
| Network 3 | 47.18% |
| Network 4 | 48.27% |
| Network 5 | 46.70% |

  - Full-connected layer with softmax activation function
- Network 4
  - Convolution layer with 256 filters of size 7
  - Flattening layer
  - Full-connected layer with softmax activation function
- Network 5
  - Two convolution layers with 256 filters of size 7
  - Flattening layer
  - Full-connected layer with softmax activation function

Output of all the networks is three-element vector, whose cells correspond to emotion classes. For the all convolution layers the rectified linear unit function was used as activation function. As loss function there was used categorical crossentropy and as optimizer - adam. The metric user for measuring network prediction quality was accuracy.

### 4.3 Training Networks

Before training networks test and train data sets were prepared. Train set contained 80% of all the data, test set was built from remaining 20%. Both sets were prepared from all used series of data, i.e. every single series was divided in the same ratio. Every model was trained and tested several times because of nondeterministic character of initial weights of network.

Training of network was processed with two stop conditions: target accuracy and no improvements for specified number of iterations. Target accuracy was set to value 0.6 and limit of iterations without improvement to 10. In every iteration training data were shuffled and 5 epochs of training were performed.

## 5 Results

Results of training and testing models are showed in table 2.

Taking into account that 39% of all data are of class 'calm', results are up to 10 percentage points better than primitive predicting always the most common class, which is moderate success. The reason results weren't higher can be caused by low quality of data. Analysis of data indicated that some data (4 of 15 data series) are corrupted, and we can expect that more detailed researches could reveal abnormalities in other series too. Another significant problem can be labeling, which was based on self-assesments of patients, possibly inaccurate or improper. It is also possible that some contribution in low results is caused by bad understanding of data, not sufficient preprocessing or wrong network structure and network parameters.

## 6 Summary

This project aim was to examine of effectiveness of using neural network in EEG data analysis. In the experiment there was examined accuracy of emotion recognition based on EEG data. As parts of this tasks there was implemented a convolution neural network and preprocessing module for data, and finally network was trained and its accuracy was measured for test set. Results weren't very high, which can be caused by small amount and low quality of data, but also by wrong parameters or structure of network or low understanding of data. Results suggests that emotion recognition is a complex problem, that requires huge amount of data and efforts to be solved with good results.

## Appendix A
## Author Contributions

**Data loading** K. Lelonek, P. Figiel
**Data processing** P. Figiel, D. Pieła, M. Jakóbczyk
**Network modelling** D. Pieła, K. Lelonek
**Testing** P. Figiel, K. Lelonek
**Literature review** M. Jakóbczyk
**Data analysis** K. Lelonek
**README** M. Jakóbczyk
**Report - contents** K. Janas
**Report - formatting** K. Janas

## Appendix B
## Code

**main**

```
import numpy as np
import random
import tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Conv1D,Conv2D, GlobalAveragePooling1D, MaxPooling1D
import cvapr_data
import data_processor
import prepared_data_provider
import sys
import os

def get_databse_path():
    if len(sys.argv) < 2 or sys.argv[1] in ("-h", "--help"):
        print("Usage: python main.py <path_to_dir_with_database>")
        print("Directory <path_to_dir_with_database> has to contain enterface06_EMOBRAIN 
            ↪ directory.")
        return None
    elif len(sys.argv) >= 1:
        return sys.argv[1]

#configuration
random_seed = 42
channel_number = 64
freq_samples = 100
max_output_size = 3
data_per_file = 30
test_to_train = 0.2
number_of_files = 11
test_size = int(test_to_train*(number_of_files*data_per_file))
train_size = int((1-test_to_train)*(number_of_files*data_per_file))
test_x = np.zeros(shape=(test_size, channel_number, freq_samples))
test_y = []

#enterface06_EMOBRAIN_path = "C:\\Users\\pawel\\Documents\\Studia\\CVaPR\\Projekt"
enterface06_EMOBRAIN_path = get_databse_path()
if enterface06_EMOBRAIN_path is None or not os.path.isdir(enterface06_EMOBRAIN_path):
    print(f"Cannot find directory {enterface06_EMOBRAIN_path}")
    exit(1)
cvapr_data.configure(enterface06_EMOBRAIN_path, 254)
prepared_data_provider.config(channel_number, freq_samples, test_size, train_size,
    ↪ number_of_files, test_to_train)
random.seed(random_seed)

# load data
train_x = []
train_y = []
for i in range(number_of_files):
    data = cvapr_data.load_data_from_files(i)[:data_per_file]
    #data = cvapr_data.load_data_from_files(i, low_freq = 1, high_freq = 100)[:
        ↪ data_per_file]
    (temp_train_x, temp_train_y) = prepared_data_provider.get_prepared_data(data, int(i *
        ↪ (test_size / number_of_files)), test_x, test_y)
    train_x.append(temp_train_x)
    train_y.append(temp_train_y)
```

```
def create_model_1():  # 46.54%, 44,59% from 7 for standardize, power post split, no
    ↪ filtering, random 42
    model_conv = Sequential()
    model_conv.add(Conv1D(64, 11, activation='relu', input_shape=(channel_number,
        ↪ freq_samples)))
    model_conv.add(Conv1D(128, 7, activation='relu'))
    model_conv.add(Conv1D(256, 3, activation='relu'))
    model_conv.add(keras.layers.Flatten())
    model_conv.add(Dense(max_output_size, activation='softmax'))
    model_conv.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model_conv


def create_model_2():  # 43,30%, 43,07% from 7 for standardize, power post split, no
    ↪ filtering, random 42
    model_conv = Sequential()
    model_conv.add(Conv1D(128, 3, activation='relu', input_shape=(channel_number,
        ↪ freq_samples)))
    model_conv.add(keras.layers.Flatten())
    model_conv.add(Dense(max_output_size, activation='sigmoid'))
    model_conv.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model_conv

def create_model_3():  # 48,05%, 47,40%, 46,10% from 7 for standardize, power post split,
    ↪ no filtering, random 42
    model_conv = Sequential()
    model_conv.add(Conv1D(128, 3, activation='relu', input_shape=(channel_number,
        ↪ freq_samples)))
    model_conv.add(keras.layers.Flatten())
    model_conv.add(Dense(max_output_size, activation='softmax'))
    model_conv.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model_conv

def create_model_4():  # 48,70%, 48,27%, 47,84% from 7 for standardize, power post split,
    ↪ no filtering, random 42
    model_conv = Sequential()
    model_conv.add(Conv1D(256, 7, activation='relu', input_shape=(channel_number,
        ↪ freq_samples)))
    model_conv.add(keras.layers.Flatten())
    model_conv.add(Dense(max_output_size, activation='softmax'))
    model_conv.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model_conv

def create_model_5():  # 48,70%, 48,27%, 47,84%, 42,00% from 7 for standardize, power post
    ↪  split, no filtering, random 42
    model_conv = Sequential()
    model_conv.add(Conv1D(256, 7, activation='relu', input_shape=(channel_number,
        ↪ freq_samples)))
    model_conv.add(Conv1D(256, 7, activation='relu'))
    model_conv.add(keras.layers.Flatten())
    model_conv.add(Dense(max_output_size, activation='softmax'))
    model_conv.compile(loss='categorical_crossentropy',
                    optimizer='adam',
                    metrics=['accuracy'])
    return model_conv

model_conv = create_model_5()

# train
target_accuracy = 0.6
no_improvements_iterations_limit = 10
continue_training = True
no_improvements_iterator = 0
last_improved_accuracy = 0
while continue_training:
    train_xy = list(zip(train_x, train_y))
    random.shuffle(train_xy)
    train_x[:], train_y[:] = zip(*train_xy)
    for i in range(number_of_files):
        model_conv.fit(train_x[i], train_y[i], batch_size=len(train_x[i]), epochs=5)

    # test & check train end criteria
    test_y = np.array(test_y)
    score = model_conv.evaluate(test_x, test_y, batch_size=len(test_y))
    if score[1] > target_accuracy or no_improvements_iterator >
       ↪ no_improvements_iterations_limit:
        continue_training = False
    elif score[1] > last_improved_accuracy:
        last_improved_accuracy = score[1]
        no_improvements_iterator = 0
    else:
        no_improvements_iterator += 1

#test
test_y = np.array(test_y)
score = model_conv.evaluate(test_x, test_y, batch_size=len(test_y))
print(score)
```

**cvapt_data**

```
"version␣1.0.0"  # by Krzysztof Lelonek


'''
This module allows loading the data from the enterface06_EMOBRAIN database.
It requires Python3 and MNE https://martinos.org/mne/stable/index.html , https://pypi.org/
   ↪ project/mne/.
Everything that starts with _ is private, imports are imports, else is the module
   ↪ interface,
you can check what those function do in their docs.


FILTERING AND POWER SPECTRA

Basic usage is to simply use load_data_from_files(numbers...) to get list of EEG samples
   ↪ for all blocks.

Then, each of those picture block data objects can be filtered,
or the data can also be filtered while loading (if you add keyword parameters to
   ↪ load_data_from_files).
In the first case, the data is first split into blocks and then each block is filtered,
in the second case, the data is first filtered, then split into blocks,
and the resulting samples will be different in both cases.
```

```
You can also get the power spectra from the EEG, then you always have to define range of
    ↪ frequencies.
Summing up, you can:
    - load blocks and do power spectra on each
    - load blocks, filter each block and do power spectra on each
    - load blocks, filtering whole file while loading, and do power spectra on each
    - get power spectra for blocks directly from each EEG
    - filter whole EEG and then get power spectra for blocks directly from it
and in each case the resulting power spectra may be, sometimes even very, different.
'''


import os
import mne
import numpy as np
from mne.time_frequency import psd


_env = {
    "repo_path": None,
    "bss": None,
}


def _eval_path_from_eeg_path(eeg_path):
    eeg_filename = os.path.basename(eeg_path)
    patient_num = eeg_filename[4]
    session_num = eeg_filename[14]
    eval_path = os.path.join(_env['repo_path'],
                             *('enterface06_EMOBRAIN', 'Data', 'Common'),
                             f"Part{patient_num}SES{session_num}.log")
    return eval_path


def _load_block_ranges_from_marker(marker_path):
    current_line = ""
    block_ranges = []
    with open(marker_path, 'r') as marker_file:
        while not current_line.strip().endswith(f'"{_env["bss"]}"'):
            current_line = marker_file.readline()
        range_start = int(current_line.split("\t")[-2])  # [-1] is stimuli
        current_line = marker_file.readline()
        range_end = int(current_line.split("\t")[-2])
        block_ranges.append((range_start, range_end))
        for _1 in range(29):
            current_line = marker_file.readline()
            range_start = int(current_line.split("\t")[-2])
            current_line = marker_file.readline()
            range_end = int(current_line.split("\t")[-2])
            block_ranges.append((range_start, range_end))
    return block_ranges


def _load_evals(eval_path):
    current_line = ""
    eval_list = []
    with open(eval_path, 'r') as eval_file:
        for _1 in range(30):
            for _2 in range(5):
                current_line = eval_file.readline()
                if not current_line.strip():
```

```
                current_line = eval_file.readline()
            line_parts = current_line.split("␣")
            eval_list.append( ( int(line_parts[-2][0]), int(line_parts[-1][0]) ) )
    return eval_list


class PictureBlockData:

    '''Represents data gathered during one picture block'''

    def __init__(self, single_eval, block_eeg, low_freq = None, high_freq = None):
        '''Constructs PictureBlockData object based on data gathered from a picture block

        Parameters:
            - single_eval - tuple of patient's arousal and valence evaluation
                (or may be other way around, it's just the way they are ordered in file)
            - block_eeg - EEG gathered during the picture block, provided by mne library
            - low_freq - low frequency at which the signal was filtered, in Hz
                or None in case of no filtering
            - high_freq - low frequency at which the signal was filtered, in Hz
                or None in case of no filtering
        '''
        self._low_freq = low_freq
        self._high_freq = high_freq
        self._mne_eeg = block_eeg
        self._evaluation = single_eval

    @property
    def mne_eeg(self):
        '''Wrapped mne EEG object'''
        return self._mne_eeg

    @property
    def evaluation(self):
        '''Tuple of patient's arousal and valence evaluation
        (or may be other way around, it's just the way they are ordered in file)'''
        return self._evaluation

    @property
    def raw_eeg(self):
        '''Samples of patient's EEG,
        It is a np.array, with first dimension being channel number, and second - sample
            ↪ number'''
        return self.mne_eeg[:,:][0]

    def power_spectrum(self, low_freq = None, high_freq = None, step = 1):
        '''Returns a tuple,
        where first element is a 2D np.array
        with first index corresponding to channel number
        and second index corresponding frequency
        and second element is a np.array
        where i-th element is the i-th frequency, in Hz.

        The frequencies will be from low_freq to high_freq, inclusive, with step step.'''
        if low_freq is None:
            low_freq = self._low_freq if self._low_freq is not None else 0
        if high_freq is None:
            high_freq = self._high_freq if self._high_freq is not None else float("inf")
        sample_freq = self.mne_eeg.info["sfreq"]
        return psd.psd_welch(self.mne_eeg, low_freq, high_freq, n_fft = round(sample_freq
            ↪ / step))
    def copy(self):
        '''Creates and returns a copy of the object'''
        return PictureBlockData(self.single_eval, self.mne_eeg.copy(), self._low_freq,
            ↪ self._high_freq)

    def filter(self, low_freq = None, high_freq = None):
        '''Filters the EEG between low_freq and high_freq (in Hz, can be None).
        Modifies the object in place, use copy first to preserve initial object.
        Return the object after modification.'''
        self.mne_eeg.filter(low_freq, high_freq)
        self._low_freq = low_freq if low_freq is not None else self._low_freq
        self._high_freq = high_freq if high_freq is not None else self._high_freq
        return self


def configure(repo_path, block_start_stimuli=254):
    """Sets up parameters needed for the module to work

    repo_path is the directory that containts the enterface06_EMOBRAIN directory (and
        ↪ everything else inside)
    block_start_stimuli is the stimuli that indicates the beginning of the picture block
        - the other one is the end, it has to be either 254 or 255"""

    _env['repo_path'] = repo_path
    _env['bss'] = block_start_stimuli


def available_eeg_paths():
    """Returns list of paths for found EEG files

    The first patient data and second patient's first EEG are discarded"""

    eeg_dir_path = os.path.join(_env['repo_path'], *('enterface06_EMOBRAIN', 'Data', 'EEG'
        ↪ ))
    eeg_dir_content_list = os.listdir(eeg_dir_path)
    eeg_file_list = filter(lambda el: el.endswith(".bdf"), eeg_dir_content_list)
    eeg_file_list = filter(lambda el: not el.startswith("Part1"), eeg_file_list)
    eeg_file_list = filter(lambda el: not el.startswith("Part2_IAPS_SES1"), eeg_file_list)
    eeg_file_path_list = [os.path.join(eeg_dir_path, eeg_file) for eeg_file in
        ↪ eeg_file_list]
    return eeg_file_path_list


def load_data_from_files(*file_numbers, low_freq = None, high_freq = None):
    '''Returns a list with PictureBlockData objects
    where each element corresponds to one picture block.

    Parameters:
        - file_numbers - indices of available EEG files (from 0 to 10).
            There are loaded all blocks for each number in the order their indices are
                ↪ passed,
            i.e. load_data_from_files(3,1,2,3) will first return all blocks
            from file of index 3, then index 1, 2 and 3 again (duplicated).
        - low_freq - low frequency used for filtering, in Hz, or None
        - high_freq - high frequency used for filtering, in Hz, or None

    Notice that low_freq and high_freq are keyword-only parameters.
    Loading data without filtering should be faster.'''

    available_eeg_paths_snapshot = available_eeg_paths()
    eegs_to_load = [available_eeg_paths_snapshot[i] for i in file_numbers]
    markers_to_load = [single_eeg_to_load + '.mrk' for single_eeg_to_load in eegs_to_load]
    evals_to_load = [_eval_path_from_eeg_path(single_eeg_to_load)
                     for single_eeg_to_load in eegs_to_load]
    no_filtering = low_freq is None and high_freq is None
    loaded_data = []
    for eeg_path, marker_path, eval_path in zip(eegs_to_load, markers_to_load,
        ↪ evals_to_load):
        block_ranges = _load_block_ranges_from_marker(marker_path)  # list of tuples
        eval_list = _load_evals(eval_path)
        raw_eeg = mne.io.read_raw_edf(eeg_path)
        if no_filtering:
            pass
        else:
            raw_eeg.load_data().pick_channels(raw_eeg.info["ch_names"][:-8]).filter(
                ↪ low_freq, high_freq)
        for (eeg_start, eeg_end), single_eval in zip(block_ranges, eval_list):
            if no_filtering:
                sample_freq = raw_eeg.info["sfreq"]
                block_eeg = raw_eeg.copy().crop(eeg_start / sample_freq, eeg_end /
                    ↪ sample_freq).load_data().pick_channels(raw_eeg.info["ch_names"
                    ↪ ][:-8])
            else:
                block_eeg = raw_eeg.copy().crop(raw_eeg.times[eeg_start], raw_eeg.times[
                    ↪ eeg_end])
            picture_block_data = PictureBlockData(single_eval, block_eeg, low_freq,
                ↪ high_freq)
            loaded_data.append(picture_block_data)
    return loaded_data


def load_power_spectra_from_files(*file_numbers, low_freq = None, high_freq = None, step =
    ↪ 1, filter = False):
    '''Returns a list,
    where each element is a tuple corresponding to a picture block,
    where first element is a tuple,
    where first element is a 2D np.array
    with first index corresponding to channel number
    and second index corresponding frequency
    and second element is a np.array
    where i-th element is the i-th frequency, in Hz
    and second element is a tuple
    of patient's arousal and valence evaluation
    (or may be other way around, it's just the way they are ordered in file).

    Visually:
    [ (   ( [[ch1pow1,ch1pow2,...],[ch2pow1,ch2pow2...],...], [freq1, freq2, ...] ), (
        ↪ arousal, valence)   ),
      (   ( [[ch1pow1,ch1pow2,...],[ch2pow1,ch2pow2...],...], [freq1, freq2, ...] ), (
          ↪ arousal, valence)   ),
      ...
          ↪
          ↪ ]
                                                                                    ]
    Parameters:
        - file_numbers - indices of available EEG files (from 0 to 10).
            There are loaded all blocks for each number in the order their indices are
                ↪ passed,
            i.e. load_data_from_files(3,1,2,3) will first return data for all blocks
            from file of index 3, then index 1, 2 and 3 again (duplicated).
        - low_freq - low frequency used for power spectra calcuation, in Hz
        - high_freq - high frequency used for power spectra calcuation, in Hz
        - step - the frequency step in returned power spectrum
        - filter - if True, the data will be explicitly filtered before power spectra
            ↪ calculation
            between the given frequencies

    This function uses mne function to get power spectra directly from whole EEG while
        ↪ defining frequencies.
    Notice that low_freq and high_freq are keyword-only parameters.'''

    available_eeg_paths_snapshot = available_eeg_paths()
    eegs_to_load = [available_eeg_paths_snapshot[i] for i in file_numbers]
    markers_to_load = [single_eeg_to_load + '.mrk' for single_eeg_to_load in eegs_to_load]
    evals_to_load = [_eval_path_from_eeg_path(single_eeg_to_load)
                     for single_eeg_to_load in eegs_to_load]
    loaded_data = []
    for eeg_path, marker_path, eval_path in zip(eegs_to_load, markers_to_load,
        ↪ evals_to_load):
        block_ranges = _load_block_ranges_from_marker(marker_path)  # list of tuples
        eval_list = _load_evals(eval_path)
        raw_eeg = mne.io.read_raw_edf(eeg_path, preload = True)
        raw_eeg.pick_channels(raw_eeg.info["ch_names"][:-8])
        if filter and not (low_freq is None and high_freq is None):
            raw_eeg.filter(low_freq, high_freq)
        low_freq = low_freq if low_freq is not None else 0
        high_freq = high_freq if high_freq is not None else float("inf")
        for (eeg_start, eeg_end), single_eval in zip(block_ranges, eval_list):
            sample_freq = raw_eeg.info["sfreq"]
            block_frequencies = psd.psd_welch(raw_eeg, low_freq, high_freq, raw_eeg.times[
                ↪ eeg_start], raw_eeg.times[eeg_end], n_fft = round(sample_freq / step))
            picture_block_data = (block_frequencies, single_eval)
            loaded_data.append(picture_block_data)
            # raw_eeg[channel's, sample's][tuple_elem(0=samples,1=times)]
            block_eeg = np.array(raw_eeg[0:-1, eeg_start:eeg_end + 1][0])
            block_data = (block_eeg, single_eval)
            loaded_data.append(block_data)
    return loaded_data
```

**prepared_data_provider**

```
import numpy as np
from data_processor import scale, standardize

_config = {
    "channel_number": None,
    "freq_samples": None,
    "test_size": None,
    "train_size": None,
    "number_of_files": None
}


def config(channel_number, freq_samples, test_size, train_size, number_of_files,
    ↪ test_to_train):
    _config["channel_number"] = channel_number
    _config["freq_samples"] = freq_samples
    _config["test_size"] = test_size
    _config["train_size"] = train_size
    _config["number_of_files"] = number_of_files
    _config["test_to_train"] = test_to_train
```

```
def calculate_emotion(valence, arousal):
    if arousal >= 3 and valence >= 3 :
        return np.array([1,0,0])
    elif arousal >= 3 and valence < 3 :
        return np.array([0,1,0])
    else :
        return np.array([0,0,1])


def get_prepared_data(data, test_start, test_x, test_y):
    train_elements = int(_config["train_size"]/_config["number_of_files"])
    input_train = np.zeros(shape=(train_elements, _config["channel_number"], _config["
       ↪ freq_samples"]))
    output_train = []
    i = test_start
    for j in range(len(data)):
        processed_data = data[j].power_spectrum(1, 100, 1)
        for channel_data in processed_data:
            #scale(channel_data)
            standardize(channel_data)
        block_eeg = []
        for channel_data in processed_data[0]:
            for sample in channel_data:
                block_eeg.append(sample)
        single_eval = data[j].evaluation

        if j < train_elements:
            output_train.append(calculate_emotion(single_eval[0], single_eval[1]))
            input_train[j][0:_config["channel_number"]][0:_config["freq_samples"]] =
                ↪ processed_data[0][0:_config["channel_number"]][0:_config["freq_samples"
                ↪ ]]
        else:
            test_y.append(calculate_emotion(single_eval[0], single_eval[1]))
            test_x[i][0:_config["channel_number"]][0:_config["freq_samples"]] =
                ↪ processed_data[0][0:_config["channel_number"]][0:_config["freq_samples"
                ↪ ]]
            i+=1

    output_train = np.array(output_train)
    return (input_train, output_train)
```

**data_processor**

```
from scipy import fftpack
import numpy as np
import matplotlib.pyplot as plt
from enum import Enum

class Emotion(Enum):
    positive_exciting = 1
    calm = 2
    negative_exciting = 3

def calculate_emotion(valence, arousal) -> Emotion :
    if arousal >= 3 and valence >= 3 :
        return np.array([1,0,0])
    elif arousal >= 3 and valence < 3 :
        return np.array([0,1,0])
    else :
        return np.array([0,0,1])

def process_batch(data):
    """
    WARNING - this method has been deprecated due to being too slow. It is now recommended
        ↪ to use power_spectrum
    method belonging to data object generated by cvapr_data.

    Processes batch of raw data.
    A batch is an array of five arrays corresponding to five blocks in a session.
    Each of these arrays consists of 71 arrays representing EEG data for each electrode.
    """

    print("Processing data...")
    processed_data = []
    for i in range(len(data)):
        for j in range(len(data[i][0])):
            processed_data.append(time_to_freq_domain(data[i][0][j]))
    print("Done")
    return processed_data


def time_to_freq_domain(data):
    """
    Processes signal and returns its power spectrum.
    """

    magnitudes = np.abs(fftpack.rfft(data))
    frequencies = fftpack.fftfreq(len(magnitudes)) * 1024
    start_freq = 1.0
    end_freq = 100.0
    freq_step = 0.1
    expected_results_list_length = (end_freq - start_freq) / freq_step - 1

    start_freq_index = int(len(frequencies) / 2 / max(frequencies) * start_freq) + 1
    end_freq_index = int(len(frequencies) / 2 / max(frequencies) * end_freq) + 1
    magnitudes = magnitudes[start_freq_index:end_freq_index]
    frequencies = frequencies[start_freq_index:end_freq_index]

    compressed_magnitudes = []
    temp_magnitudes_to_compress = []
    current_freq = start_freq + freq_step

    for i in range(len(frequencies)):
        if frequencies[i] < current_freq:
            temp_magnitudes_to_compress.append(magnitudes[i])
        else:
            compressed_magnitudes.append(sum(temp_magnitudes_to_compress) / len(
                ↪ temp_magnitudes_to_compress))
            temp_magnitudes_to_compress = []
            current_freq += freq_step

    # Trim and normalize frequencies against max, comment if not needed.
    compressed_magnitudes = compressed_magnitudes[0:int(expected_results_list_length) - 1]
    compressed_magnitudes = [float(i) / max(compressed_magnitudes) for i in
        ↪ compressed_magnitudes]

    # Display bar chart of frequencies against their corresponding magnitudes.
    # compressed_frequencies = np.arange(start=start_freq, stop=end_freq, step=freq_step).
        ↪ tolist()
    # compressed_frequencies = compressed_frequencies[:len(compressed_magnitudes) - len(
        ↪ compressed_frequencies)]
    # plt.bar(compressed_frequencies, compressed_magnitudes, freq_step)
    # plt.xlabel('Frequency in Hertz [Hz]')
    # plt.ylabel('Frequency Domain (Spectrum) Magnitude')
    # plt.show()
    return compressed_magnitudes

def scale(np_arr, new_min = -1, new_max = 1):
    """
    Scales given np.array in place between passed values and returns it.
    https://stats.stackexchange.com/questions/178626/how-to-normalize-data-between-1-and-1
    """

    min_val = np_arr.min()
    max_val = np_arr.max()
    np_arr -= min_val
    np_arr *= new_max - new_min
    np_arr /= max_val - min_val
    np_arr += new_min
    return np_arr


def standardize(np_arr):
    """
    Standardizes the np.array in place.
    https://kharshit.github.io/blog/2018/03/23/scaling-vs-normalization
    """

    mean_val = np_arr.mean()
    std_dev = np_arr.std()
    np_arr -= mean_val
    np_arr /= std_dev
    return np_arr
```

## References

[1] Danny Oude Bos, *EEG-based Emotion Recognition - The Influence of Visual and Auditory Stimuli,* University of Twente, Enschede, The Netherlands, 2006

[2] Yisi Liu, Olga Sourina *EEG Databases for Emotion Recognitio,* Nanyang Technological University Singapore, 2013

[3] Ahmad Tauseef Sohaib, Shahnawaz Qureshi, Johan Hagelback, Olle Hilborn, Petar Jercic *Evaluating classifiers for Emotion Recognition using EEG,* Blekinge Institute of Technology, Karlskrona, Sweden, 2013

[4] Lachezar Bozhkov, Petia Georgieva, Isabel Santos, Ana Pereira and Carlos Silva *EEG-based subject independent affective computing models,* 2015 INNS Conference on Big Data, 2015

[5] Shahnawaz Quresh, Johan Hagelback, Syed Muhammad Zeeshan Iqbal, Hamad Javaid and Craig A. Lindley *Evaluation of Classifiers for Emotion Detection while Performing Physical and Visual Tasks: Tower of Hanoi and IAPS,* Intelligent Systems Conference, 2018

[6] Arman Savran, Koray Ciftci, Guillame Chanel, Javier Cruz Mota, Luong Hong Viet, BülentSankur, Lale Akarun, Alice Caplier and Michele Rombaut *Emotion Detection in the Loop from Brain Signals and Facial Images,* Proceedings of the eNTERFACE 2006 Workshop, 2006