

Federal Autonomous Educational Institution of  
Higher Professional Education  
**National Research University «Higher School of Economics»**

Faculty of Computer Science  
Main educational program  
Applied Mathematics and Informatics

**COURSE PAPER**  
**«Deep Learning for EEG-Based Emotion Analysis»**

Authors:

Glazkova Ekaterina Vasilevna, 152, 3rd course  
Soboleva Natalia Alexeevna, 151, 3rd course

Academic adviser:

Head of Laboratory of Big Data Analysis,  
PhD in Physico-mathematical sciences,  
Ustyuzhanin Andrey

Consultant:

Britkov Radomir

Moscow 2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Related work</b>	<b>6</b>
<b>3</b>	<b>Preprocessing</b>	<b>7</b>
3.1	Background . . . . .	7
3.2	Main approaches . . . . .	7
3.2.1	MNE visualization . . . . .	7
3.2.2	Finding Dropouts . . . . .	7
3.2.3	Artifacts processing . . . . .	8
3.3	Results . . . . .	8
<b>4</b>	<b>Fully-Connected Autoencoder</b>	<b>9</b>
4.1	Background . . . . .	9
4.2	Implementation . . . . .	9
4.3	Experiments . . . . .	10
4.3.1	Validation metrics . . . . .	10
4.3.2	Hidden layer size, compression ratio . . . . .	10
4.4	Results . . . . .	11
<b>5</b>	<b>Convolutional Neural Networks</b>	<b>11</b>
5.1	Background . . . . .	11
5.2	Implementation . . . . .	12
5.2.1	Encoder . . . . .	12
5.2.2	Decoder . . . . .	13
5.2.3	Learning process . . . . .	13
5.3	Experiments . . . . .	14
5.4	Second Architecture . . . . .	14
5.4.1	Hidden state size . . . . .	14
5.5	Results . . . . .	15
<b>6</b>	<b>Recurrent Neural Networks</b>	<b>15</b>
6.1	Background . . . . .	15
6.2	Implementation . . . . .	16
6.3	Experiments . . . . .	16
6.3.1	Hidden state size . . . . .	16
6.3.2	Timestamps number . . . . .	17
6.3.3	LSTM vs GRU . . . . .	19

6.4 Results . . . . .	19
<b>7 Results</b>	<b>20</b>
7.1 Work results and their characteristics . . . . .	20
7.2 Future work . . . . .	20
<b>Bibliography</b>	<b>22</b>

### Abstract

Implementation of brain-computer interfaces is one of the most fascinating tasks of scientific world and at the same time one of the most challenging. The peculiarity of nonstationary electroencephalography (EEG) signals is that even short signals, for example, intentions and emotions, are represented in high dimensional space. To simplify the communication between computer and brain we need to be able to extract all meaningful information from initial datasets. Such presentation might be useful for scientific and practical purposes. For example, accurate classification of EEG signals can provide the solution for medical researches on detecting brain behavior. In this study we are looking at this task from slightly another angle – emotions recognition. We design a joint of convolutional and recurrent neural networks with the usage of autoencoder to compress high dimension representation of the initial data.

**Index Terms:** EEG, deep learning, CNN, RNN, emotion recognition, BCI.

Разработка нейрокомпьютерных интерфейсов – одна из самых интересных и сложных задач современной науки. Для получения информации о деятельности головного мозга могут использоваться электроэнцефалографические сигналы (ЭЭГ), сложность анализа которых заключается в большой размерности пространства измерений. Даже непродолжительные факторы оказывают влияние на показания каналов, в то же время изменения по некоторым каналам могут быть коррелированными. При разработке алгоритмов использующих ЭЭГ полезно уметь получать представление имеющихся данных в пространстве другой размерности: например, для сжатия сигналов с наименьшей потерей качества или наоборот, извлечения латентных переменных. Такие представления могут быть эффективно использованы для решения большого количества исследовательских и прикладных задач. Одна из таких задач – распознавание эмоций. В работе рассматривается создание автоэнкодера, сочетающего в себе сверточные и рекуррентные сети. Он используется для получения эмбедингов, а также для уменьшения размерности показаний ЭЭГ.

**Ключевые слова:** ЭЭГ, нейронные сети, сверточные сети, рекуррентные сети, распознавание эмоций, нейрокомпьютерный интерфейс.

## 1 Introduction

Brain-Computer Interface (BCI) is a common name for devices and software which create a communication path between human and machine. Their work is based on brain's neural activity analysis. BCI are used for research and practical purposes. One of the ways to measure brain activity is electroencephalography (EEG). It is a noninvasive monitoring method for recording electrical activity of the brain using placed along the scalp electrodes. EEG consists of several correlated time-series, each measured by a particular electrode. The range of controlled (fixed

or manipulated) and uncontrolled (random) sources of variability in the EEG signals include a lot of factors, such as patient's genotype, chronic anxiety, caffeine intake or attention level change as well as some extraphysiologic factors, for example lighting or electronic equipment [1].

It is impossible to create an exact algorithm which will be able to measure particular factor's influence on EEG signal. But this task might be solved with several machine learning approaches, which allow us to extract hidden regularities without explicit programming. Especially deep learning techniques are used to model difficult non-linear dependencies.

The neural network model was created for input data dimension reduction and feature extraction. It includes Autoencoder (AE), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Autoencoder extracts all the valuable information from EEG and reduce input data representation to the minimum possible size. Recurrent neural network is the effective model for time-series analysis and allows us to recognize moments when data was influenced by some of the mentioned above factors. Features, extracted after processing through convolutional layers correspond to higher-level spatial variations.

To evaluate data representation which was obtained after applying the neural network we will solve certain practical task. We plan to create an algorithm which will recognize patient's emotions using his EEG.

Firstly, we are planning to design a tool to extract valuable information, delete noise from EEG and minimize data representation. It will be general and might be used to solve various practical tasks.

Secondly, we are planning to design an algorithm for EEG-based emotion recognition. Results might be used as a part of medical diagnosis, in polygraphs and for entertainment purposes, for example in recommendation systems.

**Research object:** brain-computer interface, deep learning, emotion recognition.

**Research subject:** extracting useful information from EEG in space with the least possible dimension number.

**Research methods:** research on related papers, methods analysis, framework usage practice, deep learning model implementation, evaluation on real data.

**Research purpose:** create a deep learning model for effective information extraction, which results might be useful for practical purposes, such as emotion recognition.

**Research aims:**

1. Research EEG preprocessing methods and deep learning models
2. Implement preprocessing on initial EEG data
3. Create deep learning model combining AE, CNN and RNN
4. Experiment with implemented algorithms and different models architectures
5. Evaluate useful features extraction on emotion recognition task

## 2 Related work

In the past few years Brain-Computer Interface (BCI) has proved itself as a very unique technique that enables the brain's neural activity to communicate with the computer environment in order to provide some information without direct physical movements. Today's machine-learning techniques (deep learning techniques in particular) allow extracting useful information from electroencephalographic (EEG) recordings. According to that, there is no surprise, that EEG-based researches are mostly medically-directed. There are already several EEG-based BCI systems for clinical applications. The earliest purposes of such systems were to allow people with severe paralysis to communicate with the outside world. There are methods, which allow people with amyotrophic lateral sclerosis (ALS) to enter alternative communication and control technology through choosing characters from characters matrices [2] or even draw pictures [3]. Another medical aspect for the usage of EEG-recognition system is epilepsy detection, as EEG has been a valuable tool in diagnosis of neurological disorders [4].

All the methods and systems above are solving issues of great importance, but still are narrow-focused and directed to solve particular problems. According to the recent trends, the main tendency in EEG recognition area is EEG classification and feature extraction. An effective and accurate feature extraction is usually crucial for proper classification, due to complexity of raw EEG signals. One great deep learning technique proved itself to be one of the best in extracting new features and learning local patterns, especially in computer vision, – Convolutional Neural Networks (CNN). They are widely used in EEG-classification tasks [5].

To make classification even more accurate another popular deep learning technique was applied to EEG-recognition – Recurrent Neural Networks (RNN). That makes sense, since EEG-signals are time-series and RNNs have successfully been used to classify continuous signals and establish through-time connections. Combination of this two approaches was effectively used to classify abnormal brain behaviour [6].

All mentioned methods deal with feature extraction and further classification, but the main problem remains in the high complexity of raw EEG data. Is there a way to reduce the complexity without losing initial information? With the use of autoencoders (AE) that was implemented, for example, for movement [7] or emotion recognition [8].

In conclusion, in current work we combine the usage of AE, CNN and RNN for feature extraction with the main purpose of this work – to implement a deep learning structure, which will be able to extract all crucial features from EEG raw data. At the end we want this features to be applicable to different tasks, and in this particular work – emotion recognition.

## 3 Preprocessing

### 3.1 Background

Although, the information contained in EEG signals can be essential for detection of different brain problems, they are very complicated to process as they depend on various things. First of all, it is brain activity, and both conscious and unconscious parts of it, but there are also a lot of factors, that can influence EEG signal fluctuations, such as environmental noise or physical artifacts. Artifact – that is the term for all recorded activity that is not of cerebral origin. Artifacts can be divided into two groups: physiologic (arise from parts of the body, for example, face muscles) and extraphysiologic (arise from, for example, technical equipment) artifacts.

Extraphysiologic artifacts usually caused by electrode popping, which appears as single or multiple sharp vertical transient. And can easily be detected by setting some reasonable maximum/minimum values. Physiologic artifacts are: muscle (electromyogram) activity, glossokinetic artifact, eye movements, pulse and respiration artifacts [11].

To extract the most important features of EEG observations preprocessing is necessary. For data processing and visualization MNE – open-source Python software for human neurophysiological data including EEG was chosen. In this area there are two main state-of-art methods to process EEG signals: Wavelet transform [9] and independent component analysis (ICA) [10].

Both implementations provide channel correlation depending on validity of time and channel groups division.

### 3.2 Main approaches

Main impact is made by Soboleva Natalia.

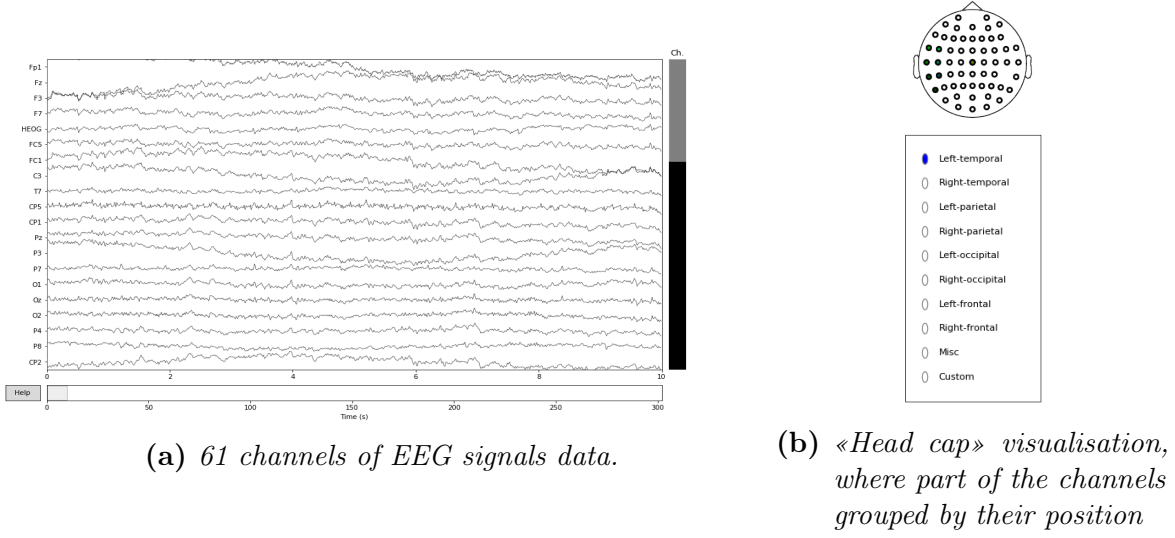
#### 3.2.1 MNE visualization

As it was mentioned, preprocessing is crucial to the accuracy of our future model. Given data consists of 61 channel, which were visualized and studied with MNE Python library. Figure 1.a illustrates 61 visualized channels of initial data.

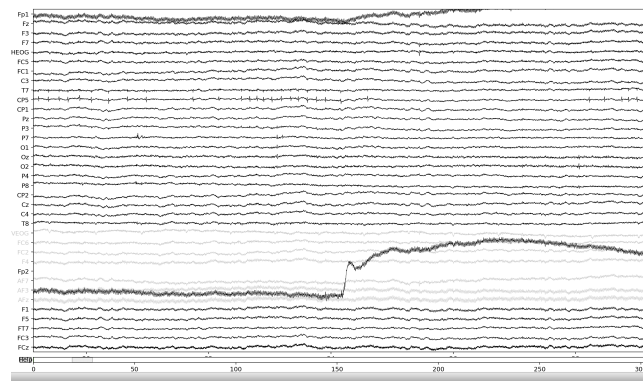
With some additional information «head cap» can be restored (Figure 1.b).

#### 3.2.2 Finding Dropouts

With MNE tools most dropouts, which were caused by electrode popping, can be detected and reduced via addition of reasonable threshold. Figure 2.1 illustrates a dropout example.



**Figure 3.1:** Caption for this figure with two images



**Figure 3.2:** Dropout channel example on EEG signals data.

### 3.2.3 Artifacts processing

Earlier in this paper we referred to Wavelet and ICA, as the two most commonly used methods of EEG signals pre-processing. While there is an implemented ICA method in MNE library, the Wavelet transformation implementation demands certain amount of custom tuning.

Due to the inconsistent initial information on electrodes location, basic ICA method did not show significant improvements to the data quality.

## 3.3 Results

The dataset was preprocessed by Laboratory of Cognitive Research [12] to be cleaned from different artifacts. Moreover, several channels which did not contain any useful information (for example, channel O2 had a huge amplitude, thus the sensor had some defect). After preprocessing number of channels was reduced to 58 and the data was transformed using MinMax scaling with parameters counted on the whole train dataset.



## 4 Fully-Connected Autoencoder

### 4.1 Background

Autoencoder, originally introduced in [1], is a neural network which consists of two parts: encoder and decoder, both are neural networks. In this work we create contractive autoencoder. It means that the layer between encoder and decoder has less neurons than an input layer. It is aimed to reduce input data dimension space, delete the noise and extract valuable information. Decoder transforms reduced representation to the target dimension space. In our task autoencoder's target is the same as input data – it guaranties that we do not lose any data.

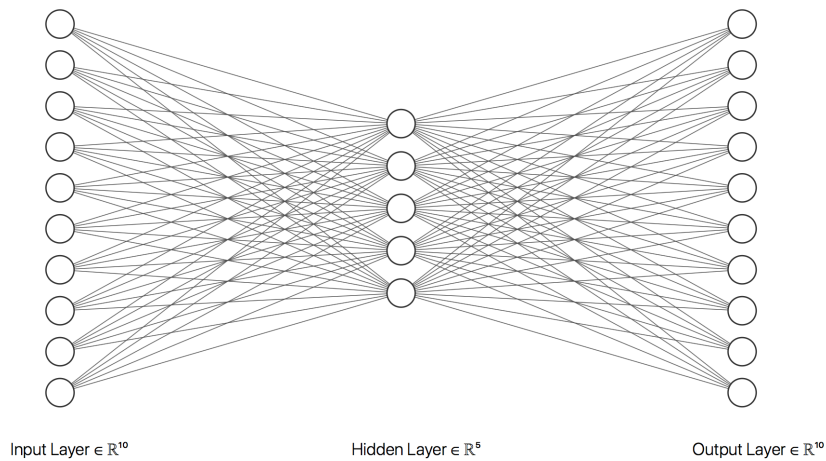
Additionally this approach might be useful for data preprocessing with the aim representation being data cleaned from the noise. The noise consists of several deviations, such as measurement errors, muscle and eyes artifacts. We did not use denoising autoencoder in our work, but we consider it as an opportunity for the future research.

### 4.2 Implementation

Main impact is made by Glazkova Ekaterina.

In our work we implement several types of autoencoders and in this section fully-connected autoencoder is presented.

Our first autoencoder version consists of two fully-connected layers with ReLU activation function. Architecture is presented on figure 4.1. The loss function during training was MSE. In learning process stochastic method ADAM was used.



**Figure 4.1:** *Fully-Connected Autoencoder architecture*

For the implementations we used following technologies: python 3.6, tensorflow, keras, scikit-learn, h5py [13, 14, 16, 17].

## 4.3 Experiments

### 4.3.1 Validation metrics

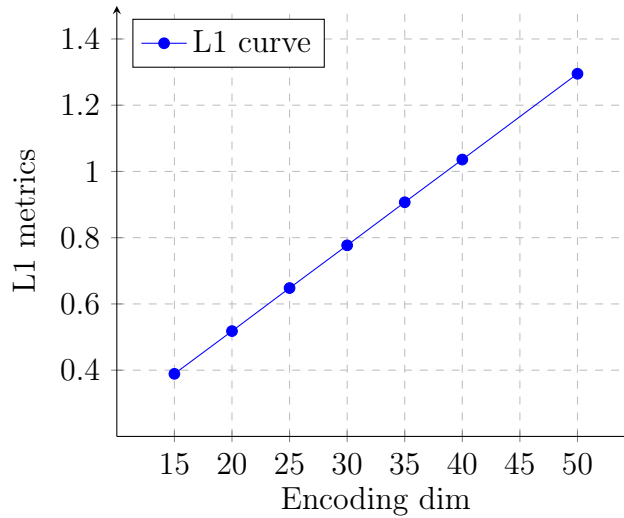
We considered two possible validation metrics:

$$L1(y_{pred}, y_{true}) = \frac{encoded\_dim}{input\_dim} \left( 1 + MSE(y_{pred}, y_{true}) \right) \quad (1)$$

$$L2(y_{pred}, y_{true}) = \frac{RMSE(y_{pred}, y_{true})}{||y_{true}||_2} \quad (2)$$

As L1 we considered combination of MSE and compression ratio, thus we appreciate reduction of representation size.

Results for the first implementation and evaluation on L1 is presented on figure 4.2. This metrics strongly depends on compression ratio, that is why the best result was reached with the minimum considered number of neurons – 15 neurons.



**Figure 4.2:** *L1 results on train subset*

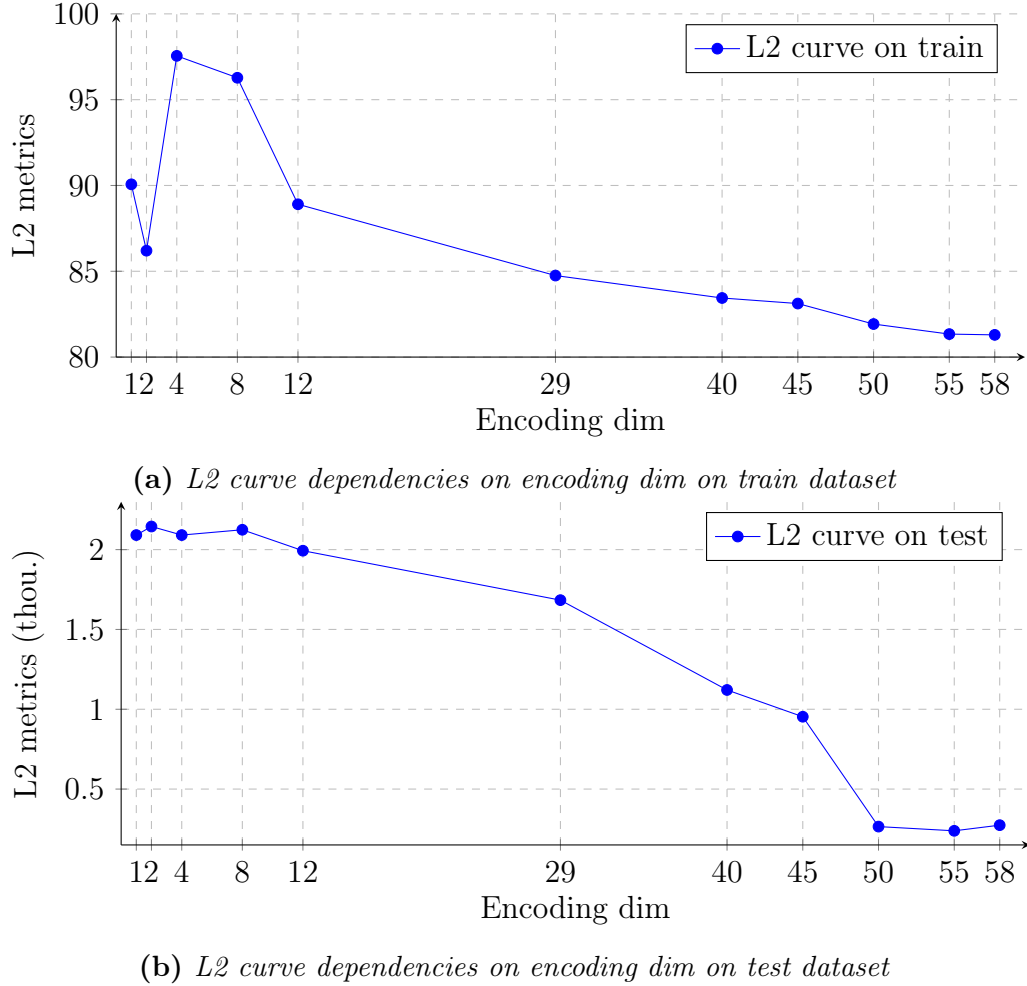
L2 metrics does not depend on compression ratio, thus we will just compare its results with different hidden layer size (see figure 4.3). L2 corresponds to the normalized predicted variance. As our model learns according to the MSE loss function using metrics that includes RMSE is only rational.

L2 metrics correlates less with the compression ratio than L1 metrics, meanwhile shows reasonable quality of predictions analysis. Thus, rationally, it is used in this work.

### 4.3.2 Hidden layer size, compression ratio

It is highly anticipated that L2 has inverse relationship with compression ratio. As we want to reduce hidden layer size as much as possible, we consider 29 neurons (compression ratio

equals 0.5) as the best result for our task. The value was chosen based on series of experiments, see results on figure 4.3.



**Figure 4.3:** *L2 Results for Fully-connected Autoencoder*

## 4.4 Results

We implemented the baseline autoencoder architecture, which consists of 2 fully-connected layers and ReLU activation. Autoencoders with several different hidden layer sizes were trained and validated with two metrics. During the experiments the most appropriate metrics – L2 (2) and the best number of hidden layer neurons – 29 was chosen.

# 5 Convolutional Neural Networks

## 5.1 Background

The motive behind using convolutional neural networks (CNN) is simple, as CNNs usually used in many machine learning areas, such as computer vision, natural languages processing

and even audio signals recognition. The main similarity between all these signals lies in their hierarchical structure: the whole signal consists of small pieces. CNNs can find local dependencies and represent higher-level features as compositions of lower-level dependencies. Another reason is that CNN is well-suited for the dimensional relevance representations. The main purpose of convolutional neural networks is to exploit spatial connections between different features in each signal.

In current work we are planning on using three types of layers: convolutional layers, pooling layers and fully connected layers. Building a CNN model means that there are many hyper-parameters to choose from, among others we must decide about sizes of the filters. There are two completely different approaches on how to apply convolutions on the EEG data: convolutions through time, e.g. only inside one channel, or using them to extract correlations between channels, e.g. at once we can look at only two neighbour channels or at all 61 (after preprocessing – 58) of them in general. Another variety of research opportunities lies within the particular size of filters. We can make them really small and explore close lying dependencies, instant signal changes. On the other hand there is a chance, that we are looking here for more durable connections and it will be reasonable to widen our filters. The length of the filters should be considered as hyper-parameter and tuned on cross-validation, for example.

Pooling layer will be applied to reduce the initial size of our EEG data. There are different types of pooling with max pooling being the most frequently used. Pooling layers will also prevent our model from overfitting.

## 5.2 Implementation

Main impact is made by Soboleva Natalia.

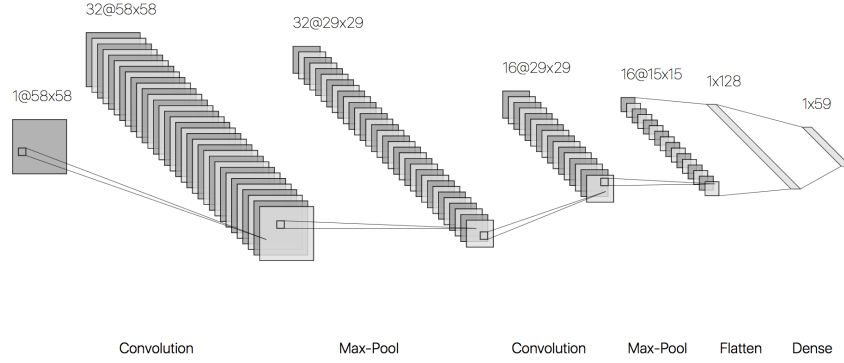
The particular number and order of layers was chosen after a variety of conducted experiments (see experiments for this one) on the filters, pooling strategies, cost functions and other hyper-parameters.

The current version of the CNN encoder consists of a stack of Conv1D and MaxPooling1D layers (max pooling being used for spatial down-sampling), while the decoder consists of a stack of Conv1D and UpSampling1D layers (see figure 5.1). For the particular number and sizes of filters see tables 5.1, 5.2.

For the implementations we used following technologies: python 3.6, tensorflow, keras, MNE, scikit-learn, h5py [13, 14, 15, 16, 17].

### 5.2.1 Encoder

The number of channels is constant in our work and equals 58, window size can be different and is defined by the length of the input data. Thus, formed input has the following shape: (channels\_num=58, window\_size  $\in$  [0, max\_length], 1). Table 5.1 shows the encoder's structure,



**Figure 5.1:** *CNN architecture*

which consists of two one-dimensional convolution layers, with 64 and 32 filters and filters length equals 5; two max pooling layers and dense (flatten) layer.

Layer	channels_num	window_size	nb_filters
Input	58	w	1
Conv1D(64, 5) + ReLU	58	w	64
MaxPooling1D	58	w / 2	64
Conv1D(32, 5) + ReLU	58	w / 2	32
MaxPooling1D	58	w / 4	32
Dense + ReLU	58	w / 4	1

**Table 5.1:** *CNN encoder structure with dimensions shapes*

### 5.2.2 Decoder

Decoder structure is quite similar to the encoder. Instead of pooling now we are using upsampling. Table 5.2 shows current structure and dimensions shapes.

Layer	channels_num	window_size	nb_filters
Conv1D(32, 5) + ReLU	58	w / 4	32
UpSampling1D	58	w / 2	32
Conv1D(64, 5) + ReLU	58	w / 2	64
UpSampling1D	58	w	64
Conv1D(64, 5) + Sigmoid	58	w	1

**Table 5.2:** *CNN decoder structure with dimensions shapes*

### 5.2.3 Learning process

ADAM was chosen as an optimizer with MSE loss function. Each filter was placed on just one channel. In general we have 61 channels in the initial dataset and 58 channels after preprocessing (see section [Preprocessing](#)).

## 5.3 Experiments

The initial and most natural experiments are on window size, as the time line is essential for emotions classification. The window size parameter was set to 0.5 s (chosen from 0.05 s to 5 s) as the most suitable variant, according to biological studies, the target (emotions) happens quick, but not simultaneously.

For the experiments with the number of filters we used the common benchmarks: 16, 32, 64, 128. The chosen variables are 64 for the first convolution layer and 32 for the second. The variety of filters influence the number of model parameters and, as our dataset is comparatively small, huge number of parameters will cause model to overfit. Thus we need to keep the number of parameters to possible minimum. The number of experiments on filters length also were held. In conclusion, filter length equals 5 was chosen from the range [2, 100] as the best working. To avoid overfitting, there were series of experiments on dropout coefficient, and it was set to 0.4.

We use batch generator for model fitting. That means that on each iteration model takes random window sized batch from random file. Batches do not repeat during one epoch. Total of 31 files length from 300 s to 1000 s and 1 for validation.

Current compression is connected only to time stamps and does not include correlation search between different channels. However, the fact that some of the channels have strong connection is well-known. The idea in [7] encouraged us also to try two dimensional convolution in order to also compress channels dimension and extract features, that relate to the similarities between different channels. This approach can also be suitable for the further integration of CNN in the final architecture.

## 5.4 Second Architecture

In this section our convolutional AE consists of only one convolutional layer and uses not one-dimensional convolutions, but two in order to extract dependencies between channels as well. There were several experiences starting with the size and number of layers and all the way to deciding whether to train or model on random batches of data or on the consistent batches, extracted from one file. The resulting architecture is shown in the table 5.4.

### 5.4.1 Hidden state size

As we are looking for the most efficient size of the hidden state – `encoding_dim`, several experiments were conducted in order to find the best. Values were chosen from the set {15, 29, 42, 50, 58}. Figure ?? shows dependencies between hidden state size and loss on train and test data, during several first steps/epochs of learning process.

Layer	window	channels	filters
Input	10	58	1
Conv2D(5, (5,5))	6	54	5
MaxPooling2D	3	27	5
Flatten	405	None	None
Dense	enc_dim	None	None

**Table 5.3:** *CNN encoder structure with dimensions shapes*

Layer	window	channels	filters
Dense	405	None	None
Reshape	3	27	5
UpSampling2D	6	54	5
Conv2D.T(1, (5,5))	10	58	1

**Table 5.4:** *CNN decoder structure with dimensions shapes*

## 5.5 Results

We have considered a problem of feature extraction and dimension reduction using CNN architecture. The flexibility of the current model is quite limited, due to the constant compression ratio – 0.25. But the results of the conducted experiments show that current compression is suitable for the initial data.

Second version of convolutional autoencoder is much lighter and can now easily be used in come mobile technologies, which is usefull for the further practical applications, but experiments on the second architecture are not yet over and current model can be improved or suited for the special practical task, such as emotions recognition.

## 6 Recurrent Neural Networks

### 6.1 Background

From another point of view EEG signals are basically time-series, and recurrent neural networks (RNN) work great to build time dependent connections, extract temporal features. So, we include RNNs as part of this project to represent the temporal feature of the input EEG signal. There are two main more complicated known types of RNN: Long-Short-Term-Memory (LSTMs) and gated recurrent units (GRUs), which are better at establishing longer connections through time. Future analysis of these structures, number of layers and other hyper-parameters will establish the most suitable model for this particular task.

According to the previous similar research [7], LSTMs show better results than conventional RNNs, but some authors also find gated recurrent units structure suitable for the task of EEG signals processing. We are planning on implementing both LSTM and GRU and experiment with architecture, hidden representation size and timebatch length.

## 6.2 Implementation

Main impact is made by Glazkova Ekaterina.

We implemented base RNN Autoencoder architecture, which consists of two recurrent layers: the first one – encoder – takes batch of size  $58 \times \text{timestamp}$  (batch from EEG), processes it and returns one vector and hidden state of size  $\text{encoding\_dim}$ . Decoder's hidden state is initialized with encoder's output hidden state and takes batch of the same size as encoder, but it consists of "unusual" values (we used  $-2$ ), which never occurs in real data – we show that the beginning of the sequence is presented. The output of our contractive autoencoder is the same as encoder input - has size  $58 \times \text{timestamp}$ .

Layer	input size	hidden states size	output size
LSTM 1 (Encoder)	$58 \times \text{timestamp}$	$\text{encoding\_dim}$	$58 \times \text{timestamp}$
Encoded representation	$\text{encoding\_dim} \times 2$		
LSTM 2 (Decoder)	$58 \times \text{timestamp of } -2$	$\text{encoding\_dim}$	$58 \times \text{timestamp}$

**Table 6.1:** *LSTM Autoencoder architecture*

Layer	input size	hidden states size	output size
GRU 1 (Encoder)	$58 \times \text{timestamp}$	$\text{encoding\_dim}$	$58 \times \text{timestamp}$
Encoded representation	$\text{encoding\_dim}$		
GRU 2 (Decoder)	$58 \times \text{timestamp of } -2$	$\text{encoding\_dim}$	$58 \times \text{timestamp}$

**Table 6.2:** *GRU Autoencoder architecture*

Time stamps parameter might be changed during the training and execution - it is equal to training object size.

To decrease size of input data we plan to experiment with different values of encoding dim and taken time stamps.

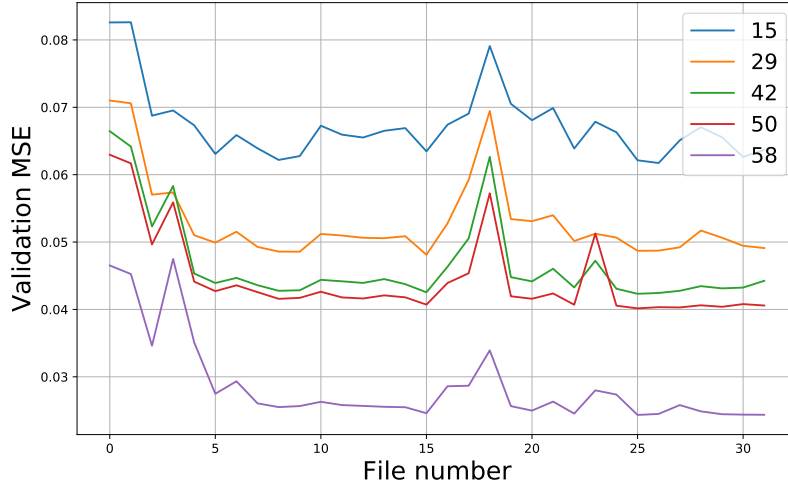
The technologies are the same, as mentioned before (see section 5).

## 6.3 Experiments

### 6.3.1 Hidden state size

We conducted experiments with hidden states size -  $\text{encoding\_dim}$  and implemented algorithms with  $\text{encoding\_dim}$  from set  $\{15, 29, 42, 50, 58\}$ . This experiment was set for LSTM version of autoencoder. The results are provided on 6.1. They show metrics evaluation on test (during this experiment it was a randomly chosen and fixed EEG file) after learning on all the rest EEG files, 2 epochs of learning each. All the experiments were conducted with the same order of files and the same batch size during learning. Timestamps number was fixed and equal to 1.





**Figure 6.1:** *MSE Validation results during first epoch for LSTM AE*

The learning was conducted in the same training files order for all experiments, that is why at some points all the curves have the same abnormal trend – those training files are different from the validation file – it is an EEG signal feature.

As it was expected, MSE results correlate with hidden state size and the more hidden state size is, the better autoencoder performs. It is quite interesting, that results for 42 and 50 are almost similar and on some iterations autoencoder with *encoding\_dim* = 50 performs worse. Our aim is to reduce data size but it might be achieved with timesteps reduction (see next experiment), so for the final model we consider 58 to be the best hidden state size. In some further experiments we also tried to use value of 42, because it provides quite good result with compression rate equal to 0.75.

### 6.3.2 Timestamps number

We experimented with number of timestamps to process. Our experiments covered values from set  $\{1, 5, 10, 20\}$ .

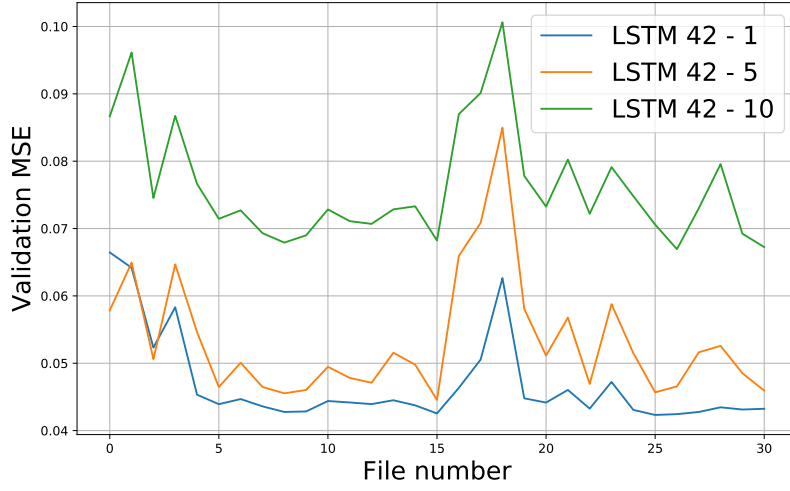
First experiment (see 6.2) was conducted with LSTM and hidden states size equal to 42. Batching and training set were the same as in the previous experiment.

The second experiment was set using GRU with 58 hidden units. The training was presented only on a part of the training set (6 files out of 32) and validated on one test file.

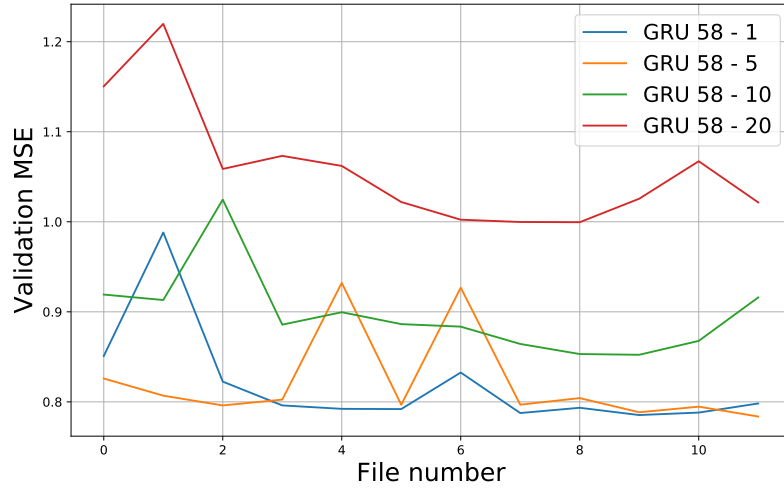
During this experiment learning batching was more probabilistic: on each iteration random shuffle of files was conducted, each file was partitioned on  $58 \times \text{timestamp}$  batches and batches were shuffled. That is why trends, which depend on files, are not correlated for different learning curves.

The results are presented on 6.3.

The amount of information stored in the hidden state is increasing with timestamps number



**Figure 6.2:** *MSE Validation results for first epoch on training set for LSTM AE*

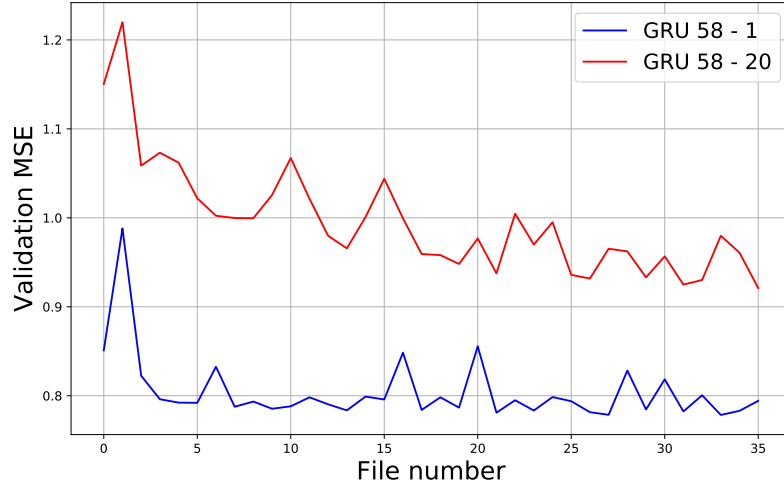


**Figure 6.3:** *MSE Validation results during two epochs on train subset for GRU AE*

increase. Hidden state size was equal during all experiments, that is why learning is harder and takes more time with more timestamps. But after several epochs loss of training with higher amount of timestamps goes down to loss of a more simple model (illustrated on 6.4).

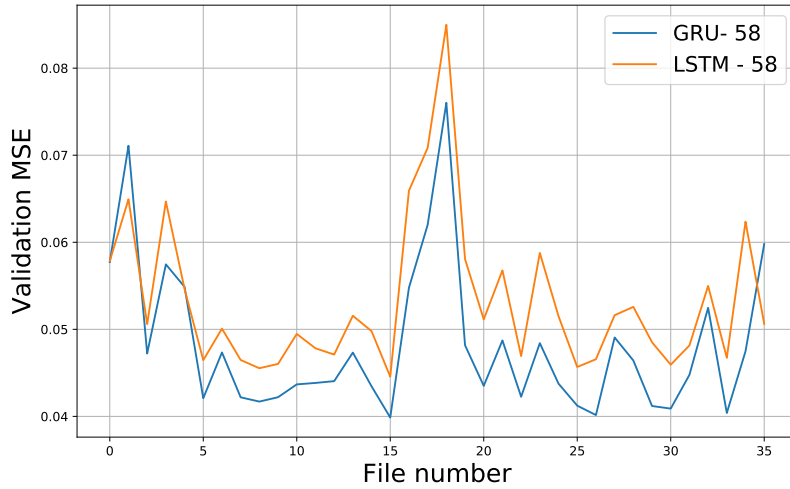
During both experiments results for  $timestamps = 1$  and  $timestamps = 5$  were quite close to each other. At the same time in terms of data compression it means compression ratio 1 and 0.2 respectively. That is why we consider  $timestamps = 5$  to be the best choice of this parameter.

Architecture of the model does not depend on amount of timestamps, that is why it is possible to combine different approaches during the learning. For example, learn with smaller timestamps number during first epochs to find main data features and increase it later.



**Figure 6.4:** *MSE Validation results during 6 epochs on train subset for GRU AE*

### 6.3.3 LSTM vs GRU



**Figure 6.5:** *MSE Validation results for first epoch on training set for LSTM and GRU*

We conducted experiments with two different recurrent units types: LSTM and GRU. Results are presented on 6.5.

Results are almost similar, but GRU performs better. Moreover in terms of compression and our architectures (see 6.1 and 6.2) LSTM has two times more parameters and encoded representation is two times bigger. So, for our task GRU performs better.

## 6.4 Results

We implemented feature extraction and dimension reduction using RNN architecture. We conducted experiments on hidden layer size, processed timestamps number and type of RNN

cell (LSTM and GRU). According to our experiments results the best choice of GRU cell with 58-unit hidden state applied to 5 EEG timestamps. With described architecture compression ratio is 0.2. Moreover, our architecture might be used for time batches of arbitrary size, but it was trained with 5-timestamps batches and should perform better in that case.

Current compression affects only time dimension, as hidden state size equals to number of channels.

## 7 Results

### 7.1 Work results and their characteristics

We have considered a problem of extracting features from raw EEG data and using them to improve the existing results for practical tasks, emotion recognition in particular. During the process, we constructed several network architectures - Fully-Connected Autoencoder, Convolutional Autoencoder and Recurrent Autoencoder. They might be used separately or in combination to transform EEG signals and get valuable features. We combine several approaches and use spatial (channels) transformations as well as time transformations.

The resulting architectures are suitable for EEG data compression, but also might be applied for EEG feature extraction.

The best achieved compression ratio equals to 0.2. It was reached with recurrent neural network with GRU cells.

Our research and educational aims were achieved: we studied the most common neural networks architectures and approaches commonly used in similar tasks and implemented several approaches using Python and Keras.

All of the presented research aims were reached, except for embeddings usage for emotion recognition. Unfortunately, the initial dataset does not include emotions markers and is currently undergoing processing by specialists from Laboratory of Cognitive Research [12].

### 7.2 Future work

Obvious and appealing direction of future research is evaluation of the provided and implemented method as well as experimenting with approaches, solving relative EEG-based tasks.

1. Implement machine learning solutions based on constructed embeddings for real-world problems, such as emotions or temptations recognition.
2. Implement denoising autoencoder for raw EEG preprocessing, for example noise deletion.
3. Use masks in CNN model to use convolutions based on 3D-position of EEG sensors. Several different studies show that correlations between EEG sensors can be explained

by their position on the reconstructed «head cap» (see figure 3.1b). We believe that this connections can be restored via additional neural network.

4. Experiment with different network architectures, combine implemented models.

## References

- [1] S. J. van Albada, C. J. Rennie, P. A. Robinson (2018). Variability of model-free and model-based quantitative measures of EEG.
- [2] F. Nijboer, E.W. Sellers, J. Mellinger, M.A. Jordan, T. Matuz, A. Furdea, S. Halder, U. Mochty, D.J. Krusienski, T.M. Vaughan, J.R. Wolpaw, N. Birbaumer, A. Kübler (2008). A P300-based brain–computer interface for people with amyotrophic lateral sclerosis.
- [3] Münßinger, J. I., Halder, S., Kleih, S. C., Furdea, A., Raco, V., Hösle, A., and Kübler, A. (2010). Brain Painting: First Evaluation of a New Brain–Computer Interface Application with ALS-Patients and Healthy Volunteers.
- [4] Andre Rosado, Agostinho C Rosa (2011). Automatic Detection of Epileptiform Discharges in the EEG
- [5] Robin Tibor Schirrmester, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggersperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG
- [6] Subhrajit Roy, Isabell Kiral-Kornek, and Stefan Harrer (2018). ChronoNet: A Deep Recurrent Neural Network for Abnormal EEG Identification
- [7] Xiang Zhang, Lina Yao, Quan Z. Sheng, Salil S. Kanhere, Tao Gu, Dalin Zhang Converting Your Thoughts to Texts: Enabling Brain Typing via Deep Feature Learning of EEG Signals
- [8] Suwicha Jirayucharoensak, Setha Pan-Ngum, and Pasin Israsena EEG-Based Emotion Recognition Using Deep Learning Network with Principal Component Based Covariate Shift Adaptation
- [9] M.H.Alomari, A.Abubaker, A.Turani, A.M.Baniyounes, and A.Manasreh (2014) EEG Mouse : A Machine Learning-Based Brain Computer Interface
- [10] T. C. Major and J. M. Conrad The effects of pre-filtering and individualizing components for electroencephalography neural network classification
- [11] Selim R Benbadis (2017) EEG Artifacts
- [12] <https://cogres.hse.ru/en/about/> (Laboratory of Cognitive Research)
- [13] <https://www.tensorflow.org> (Python library for Deep Learning – Tensorflow)
- [14] <https://keras.io> (Python library for Deep Learning based on Tensorflow – Keras)

- [15] <https://www.martinos.org/mne/stable/index.html> (Python library for MEG and EEG analysis and visualization – MNE)
- [16] <https://www.h5py.org> (Python library for processing h5 files – h5py)
- [17] <http://scikit-learn.org/> (Python library for Machine Learning – scikit-learn)