Emotion Recognition in Videos From EEG Signals Using Deep Neural Networks (June 2019)

D. Tyrna, T. Marcol, M. Woźnica, K. Włodarski, A. Piepiora, A. Marzec

ABSTRACT

Recently, deep learning has become popular to analyse EEG signals and recognition human emotion. In this research we applied Convolutional Neural Network (CNN). The prepared model is then used to classify four emotions namely Meditation, Boredom, Excitement and Frustration. Our experiments were carried out on DEAP dataset and the results were represented in confusion matrix.

I. INTRODUCTION

In 1875 the electrical activity was observed for the first time. Since that time scientists try to discover how electrical activity of our brains influence us. Presented article covers method of emotion recognition during watching videos with the usage of EEG signal and deep neural networks. EEG signal is used in medicine to diagnose: epilepsy, sleep disorder, coma, brain death etc. The aim of the experiment to classify EEG data (level of arousal and valence) to appropriate class of emotion. The most difficult part of the this task is to analyze meaningful data from input sets.

II. LITERATURE REVIEW

Various kinds of studies have been conducted on human affective state recognition. The first approach focuses on the audio-visual features like facial expressions and speech. The second method involves physiological signals like electrocardiogram (ECG) , skin conductance (SC), electroencephalograms (EEG), electrooculography (EOG), temperature (TEM), blood volume pressure (BVP), electromyograms (EMG), and many other methods. In addition to these periphery physiological features, recently the advance of devices and processing system enables global assessment of EEG signals, which are captured from central nervous systems. When it comes to the data collection, multiple electrodes are usually attached to the user's scalp in EEG assessment devices. The signals captured in each channel will record the voltage change between a pair of adjacent electrodes. The information contained in these channels, however, can be very noisy due to a variety of artefacts, which stem from different kinds of events, such as the discomfort brought by the device, the interference from outside world, or just the sudden mood change of the participant. **Electroencephalography (EEG)** is a complex signal and can require several years of training to be correctly interpreted. Recently, deep learning (DL) has shown great promise in helping make sense of EEG signals due to its capacity to learn good feature representations from raw data. Whether DL truly presents advantages as compared to more traditional EEG processing approaches, however, remains an open question. Emotion- or affect-recognition is the task of estimating emotional responses to some stimulus. Humans typically recognize emotion by integrating multiple modalities, whether from vocal intonation, perspiration, or facial expressions.

Different modalities may also offer different benefits and challenges than one another for affect recognition, both in terms of their descriptive power, but also in terms of how practical or economical they are to capture.

Electroencephalographs (EEGs) measure activity in the brain by amplifying electrical brain waves created in psycho-physiological processes such as experiencing an emotion. One motivation for using EEG for affect recognition is its use for augmented human-computer interfaces, especially in instances where physical impairments prevent the use of traditional interfaces.

In this work we present a comparison of different approaches to use DL in EEG processing. Deep learning network(DLN) is capable of discovering unknown feature coherences of input signals that is crucial for the learning task to represent such a complicated model. The DLN provides hierarchical feature learning approach. Learned features at high-level are derived from features at low-level with greedy layer-wise unsupervised pre-training. This unsupervised pre-training provides the stage for a final training phase that is fine-tuning

Reading the articles showed us several approaches to neural network like:

Fully Connected Deep Neural Network (DNN). DNN (also known as Multi-layer Perceptron) consists of a three-layer architecture an input layer that functions as an information receiver and has multiple sensory units one more hidden layers that makes a non-linear transformation of the input space into a high dimensional space and an output layer that gives the network response through an activation function. The aim of DNNs is to solve problems that cannot be separated linearly. This type of network is trained with the back-propagation learning algorithm

Convolutional Neural Network (CNN). CNN is a sort of supervised deep learning algorithm that have demonstrated notable results in the classification of data with grid-like topology . CNN architecture is composed of a stack of building blocks with convolution kernels and pooling operators and a feed forward Artificial Neural Network (ANN). Each building block extracts relevant information from input map to its own significant feature maps. Finally the feature maps feed the ANN to compute each class probabilities (using soft-mask function).

Deep Physiological Affect Network (DPAN). This model is based on a convolutional long short-term memory (ConvLSTM) network and a new temporal margin-based loss function. This model improves the performance of emotion recognition. Specifically, the new loss function allows the model to be more confident as it observes more of specific feelings while training ConvLSTM models. The function is designed to result in penalties for the violation of such confidence.

Hybrid deep learning model called Convolutional and LSTM Recurrent Neural Networks (CLRNN) to conduct emotion recognition tasks. This model is a composite of two kinds of deep learning structures: CNN and the LSTM recurrent neural network(LSTMRNN). The CNN is used to extract features from EEG , and the LSTMRNN is used for modelling the context information of the long-term EEG sequences. The features automatically extracted by the CNN reflect the spatial distribution of the EEG signals**.**

This allowed us to understand their structure. On the basis of presented articles, we've chosen Human Emotion Recognition with Electroencephalographic Multidimensional Features by Hybrid Deep Neural Networks as a most interesting one. All occurring 'networks' in fact have some things in common. Main differences are in the model of each layer. Networks are using combinations of models like hybrid deep learning model, recurrent neural network, which have big impact on their performance. We can also observe more differences like using different sampling on the input for each model. It is hard to choose a state of art based on attached articles because some of them like "Deep Physiological Affect Network for the Recognition of Human Emotions" are comparing to general "state-of-art"(means to nothing) In fact these articles seems to be kind of outdated. Nevertheless, we've chosen Human Emotion Recognition with Electroencephalographic Multidimensional Features by Hybrid Deep Neural Networks as a most interesting one.
We find it as a best one cause of performance and uncommon model of network. First, they propose a new method for the EEG feature extraction and representation. EEG frequency features (PSD) are extracted from different EEG channels and mapped to a two-dimensional plane to construct the EEG MFI. Another aspect is proposal of a hybrid deep neural network that deals with the EEG MFI sequences and recognizes the emotions. The hybrid deep neural networks combined Convolution Neural Networks and Long-Short-Term-Memory Recurrent Neural Networks. In the hybrid structure, CNN is used to learn temporary image patterns from EEG MFI sequences, and LSTM RNN is used to classify human emotions. The next thing in common is the joint model of the dataset used in experiments called DEAP using it made results more comparable.

III. MATERIALS

A. DEAP dataset

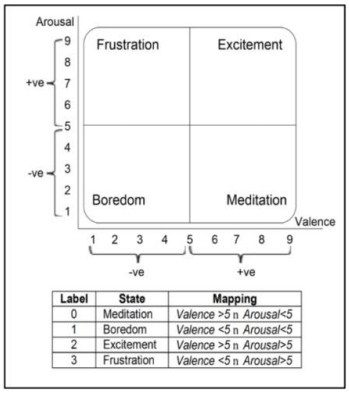
As an input material used for presented project, 'DEAP' dataset was provided.

'DEAP' is a multimodal dataset for the analysis of human affective states. It contains data provided by the electroencephalogram (EEG) and peripheral physiological (using GSR measurement) signals of participants (in presented case: 31 people) as each watched 40 one-minute long videos and was asked about their subjective emotional state (level of valence/arousal, like/dislike, dominance and familiarity). Beside the main sets (which are described in the next part of the work), additional tables were provided, each contains data of for example participant ratings or participant questionnaire. The video list (in which condensed data from all participants in terms of particular videos were presented) was also included. In this project, only the main sets of data were used.

As an input, already preprocessed DEAP dataset was provided. Mentioned preprocessing included bandpass frequency filter (4-40 Hz), downsampling from 512 to 128 Hz and EOG artefacts removal. Each set of data (participant file) contains two arrays with labels (40x4 video/labels) and data (40x40x8064 video/channel/data) respectively. Label array consists of valence, arousal, dominance and liking values computed for each video. In terms of the second array, for this project only first 32 channels were used, as the rest contains data from the GSR measurement, which is not included as an input for dataset processing in this work. Each channel (1-32) refers to particular electrode (EEG technique) and contains data from the experiment provided by the mentioned electrode.

In order to provide labels for validate four-state output of the network, discrete values were extracted from the labels array according to the two-dimensional emotion model approach proposed by Russell, as shown on figure below (as suggested and shown in Al-Nafjan, A., Hosny, M., Al-Wabil, A., & Al-Ohali, Y. (2017). Classification of human emotions from electroencephalogram (EEG) signal using deep neural network. *International Journal of Advanced Computer Science and Applications*, *8*(9).).

Figure 1. Two-dimensional emotion model



Label	State	Mapping
0	Meditation	*Valence >5 ∩ Arousal<5*
1	Boredom	*Valence <5 ∩ Arousal<5*
2	Excitement	*Valence >5 ∩ Arousal>5*
3	Frustration	*Valence <5 ∩ Arousal>5*

As the part of final results improvement, additional condition was used. The dataset was split additionally into two groups - 'strong' and 'weak' in terms of valence/arousal values for each sample. Strong sample refers to one having both valence and arousal values close to the limits of the scale. After some research, lower band was set to 3, and the higher to 7, that provides approximately six-times-reduced number of samples, but ones having stronger results in terms of presented four-state output, which is thought to be a factor of method improvement. However, in order to extend the set of data, weak sample could be treated as the strong one with the probability of 25%. For the purpose of another slight extension of the dataset, augmentation was provided using Gaussian noise added to samples.

For being more accurate for processing by neural network, data are transformed to frequency domain and split into four bands: *alpha*, *beta*, *gamma* and *theta*, each representing specified range of frequencies.

B. Data preprocessing

To obtain data in form, which can be easily processed by the neural network, the average band power in each channel is mapped into a spatial map (as suggested in [1]). As a result, each trial is represented by four maps presenting spatial distribution of average band power in each of the four mentioned bands: *alpha*, *beta*, *gamma* and *theta*. Details of the method are described below.

After initial preparation of the data, power spectral density for each channel in each trial completed by each participant is computed. As long as we are not interested in processing each of the frequencies separately, the average band power is calculated for alpha, beta, gamma and theta bands (as suggested in [2]). In the computations integration using Simpson's method was used. The values computed this way are then mapped into a 9×9 matrix (the dimensions are maximum point numbers between the horizontal or vertical test points). The empty cells of the matrix are filled with average values stored in the surrounding cells, using following formula [1]:

$$V_{(m,n)} = \frac{V'_{(m+1,n)} + V'_{(m-1,n)} + V'_{(m,n+1)} + V'_{(m,n-1)}}{K}, (0 \le m, n \le 8, n \in N)$$

where V is the value of the empty cell (corresponding to $P_{(m,n)}$ and V' is the value of the point surrounding $P_{(m,n)}$. If the index of the surrounding point exceeds the range of 0 and 8, then the value is 0. K is the number of non-zero elements in the numerator, and the default value of K is 1.
After the feature matrix is filled, it is used as a base table to generate data matrix through the interpolation method. The final data matrix has dimensions of 4×120×120 (corresponding to four bands and dimensions of the matrix of average band power obtained through the interpolation).

IV. METHODS

A. Network

For project execution, convolutional neural network was provided. Used setup (shown in the figure 2) was fully experimental Parameters of the model are typed below.

The activation functions for all of the convolutional layers:
'*ReLU'*
The dropout values:
0.5, 0.3 and 0.5.

Compilation parameters were chosen as follows:

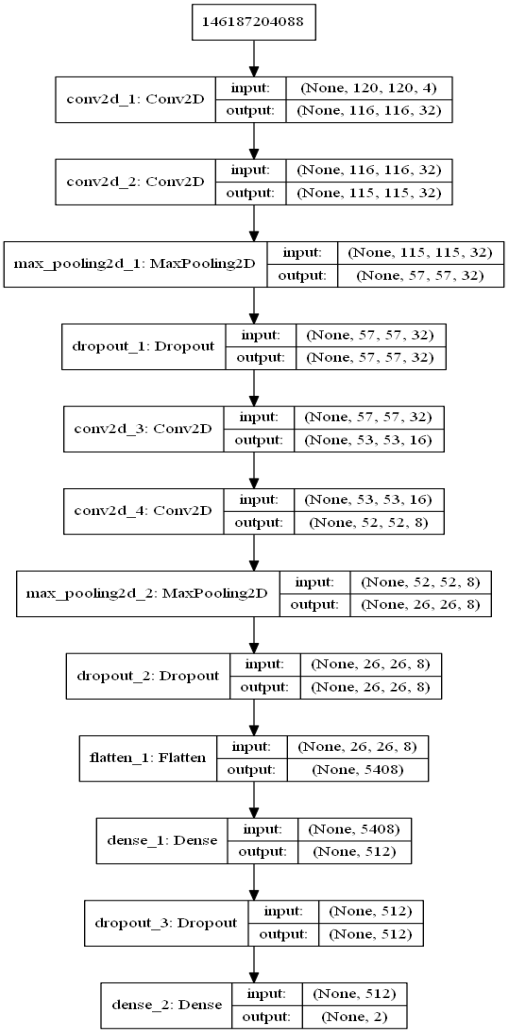
- loss function: categorical cross entropy,
- optimizer: adam,
- metric: accuracy.

B. Training and validation

Set of samples generated from the original dataset was split into two main parts: train and test set, with the percentage of 70% dedicated for training and 30% for testing. Test set has been chosen randomly through all data. For proper preparation of the datasets for training process, data generator was implemented, which allows to prepare set of training and validation samples throughout directories, each represents samples correspond to another class of output classification.

Two validation methods were proposed for the solution. First of them is the simple approach, which aim is to train data set only once, with the validation percentage set to 30% (could be differentiated across the trials and treated as a parameter of execution). Second one is commonly used K-fold technique, which provide K trainings, where for each one the different validation set is chosen as the package of [nr of samples / K] samples, chosen by random or straight splitting. Number of iterations is one of the most important parameters of execution and could have a strong impact on the results.

Figure 2: experimental setup of used network

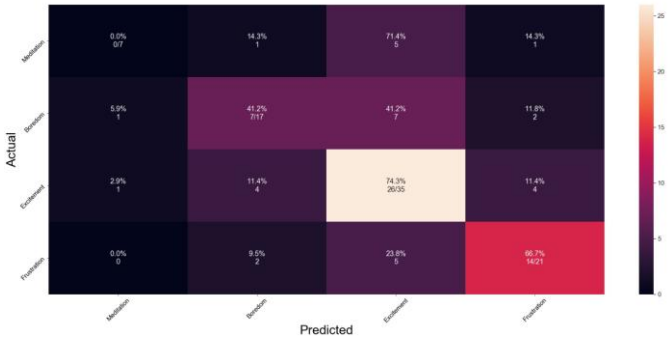


Training itself was prosecuted using Keras library method, fit generator, working on previously implemented model. As the parameters of execution, number of epochs and training steps per epoch were provided. For simple validation, number of epochs equals 15 were proposed and for K-fold method, for each iteration – 5. For both methods algorithm performs 64 training steps and two validation steps. Accuracy and loose of training for validation set are being shown for each step.

V. Results

To present the results of work, we implemented classify function which creates shown confusion matrix (figure 3). Results were obtained for test set and given parameters: batch size = 8, target size = (128,128), shuffle = false.

Figure 3: confusion matrix



As can we observe for Meditation emotion – true positive result was 0.0%. For Boredom and Frustration he results were higher in sequence: 41.2% and 66.7%. The highest result was for Excitement and reached 74.3%. This results may be caused by spotty train set division to classes. We have used 7 samples for Meditation, 17 for Boredom, 35 for Excitement and 21 for Frustration.

VI. Summary

In this paper, we presented a recognition of four emotions: Meditation, Boredom, Excitement and Frustration. We used DEAP dataset, it contains data collected during 40 one-minutes videos and subjective assessment of patients.

To achieve our goal we modelled the network as shown in the schematic figure 2. To obtain better results we have chosen the best samples for each classes and we trained network on this set. But on the confusion matrix it turned out, that the dataset obtained this way is highly imbalanced – it contains 35 samples depicting excitement, whereas there are only 7 samples of meditation. Other classes (boredom and frustration) contain respectively 17 and 21 samples This imbalance resulted in high percentage of correctly classified samples of excitement (74.3%), but no correct recognition of meditation. The percentage of correctly classified boredom and frustration samples was also correlated with their strength (frustration, a class containing more samples than boredom, were correctly recognized in 66.7% samples, whereas boredom in 41.2%). This implies how important is class balance – if one of the classes is much bigger that other, the network learns almost only the features describing this one class. This leads to poor performance in the classification of samples belonging to the other classes.

VII. Appendix

A. Credit

The tasks included by the project were done by the following group (navigation to the mentioned parts of the report is available by clicking on the numbers of the chapters):

- Tomasz Marcol: in the project: loading the data from the DEAP database, assigning labels to the samples, splitting whole dataset into training and testing sets,

splitting data into folds for k-fold cross validation, implementation of modes of network training (k-fold cross validation and simple). In the report: III.A (description of the DEAP database), IV.B (description of training and validation processes),

- Angelika Marzec: in the project: presentation of the classification results in the form of confusion matrix. In the report: **ABSTRACT** (abstract), I (introduction), V (results), VI (summary),
- Artur Piepiora: in the report: II (literature review),
- Krystian Włodarski: in the report: II (literature review),
- Marcel Woźnica: in the project: loading the data from the DEAP database, preview of the data in the frequency domain using Fourier transform,
- Diana Tyrna: data preprocessing, augmentation and construction of the initial data set (ready for splitting), implementation of the data generator for the network, design of the network architecture. In the report: III.B (description of data preprocessing) IV.A (network).

B. *Code*

1) *Labels assignment*

```
    # States - labels IMPORTANT
    # 0 - Meditation  Val > 5 Arousal < 5
    # 1 - Boredom     Val < 5 Arousal < 5
    # 2 - Excitement  Val > 5 Arousal > 5
    # 3 - Frustration Val < 5 Arousal > 5

    for i in range(n):
        labels2[i, 0] = i
        curr_valence = labels[i, 0]
        curr_arousal = labels[i, 1]
        if curr_valence > 5:
            if curr_arousal <= 5:
                labels2[i, 1] = 0
            else:
                labels2[i, 1] = 2
        else:
            if curr_arousal <= 5:
                labels2[i, 1] = 1
            else:
                labels2[i, 1] = 3

        if curr_valence > 7 or curr_valence < 3:
            if curr_arousal > 7 or curr_arousal < 3:
                labels2[i, 2] = 1
```

2) *Computation of average bad power*

```
def get_avg_band_power_single_channel(signal, fs):
    # PSD

    psds, freqs = psd_array_welch(signal, sfreq=fs, n_per_seg=7,
n_fft=np.shape(signal)[0])

    # 1. find average band powers (for alpha, beta, gamma and
theta bands)
    freq_bands = {
        'alpha': [8, 13],
        'beta': [13, 30],
        'gamma': [30, 40],
        'theta': [4, 8]
    }

    freq_res = freqs[1] - freqs[0]  # frequency resolution
    avg_power = []

    # calculate absolute avg band power in each of the bands
    # The absolute delta power is equal to the area under the
plot of already calculated PSD.
    # This can be obtained by integrating. As suggested in
    # https://raphaelvallat.com/bandpower.html, let's choose
Simpson's method, which is relying on
    # decomposition of the area into several parabola and then
summing up the area of these parabola.
    for band in ['alpha', 'beta', 'gamma', 'theta']:
        # find indices, which satisfy the limits of the specific
band
        idx = np.logical_and(freqs >= freq_bands[band][0], freqs
<= freq_bands[band][1])
        avg = simps(psds[idx], dx=freq_res)
        avg_power.append(avg)

    return avg_power
```

3) *Sample construction*

```
def prepare_sample(avg_band_power, channel_positions, points):
    # 2. calculate unknown points in the feature matrix
    # start from the middle, so the matrix filling won't ever
start from a place surrounded by zeros
    vertical = False
    forward = True
    x, y = 4, 4
    max_x = x + 1
    min_x = x - 1
    max_y = y + 1
    min_y = y - 1
    matrix_len = 9

    cnt = 0

    while cnt < matrix_len * matrix_len:
        if not np.any((channel_positions[:] == [y,
x]).all(axis=1)):
            for z in range(4):
                k = 0
                a = (avg_band_power[y + 1][x][z] if y + 1 < 9
else 0.0)
                k += 0 if a == 0.0 else 1
                b = (avg_band_power[y - 1][x][z] if y - 1 > -1
else 0.0)
                k += 0 if a == 0.0 else 1
                c = (avg_band_power[y][x + 1][z] if x + 1 < 9
else 0.0)
                k += 0 if a == 0.0 else 1
                d = (avg_band_power[y][x - 1][z] if x - 1 > -1
else 0.0)
                k += 0 if a == 0.0 else 1

                avg_band_power[y][x][z] = (a + b + c + d) / k if
k != 0.0 else 0.0

        if not vertical:
            if forward:
                if x < max_x:
                    x += 1
                else:
                    vertical = True
                    y += 1
                    max_x += 1
            else:
                if x > min_x:
                    x -= 1
                else:
                    vertical = True
                    y -= 1
                    min_x -= 1
        else:
            if forward:
                if y < max_y:
                    y += 1
                else:
                    vertical = False
                    forward = False
                    x -= 1
                    max_y += 1
            else:
                if y > min_y:
                    y -= 1
                else:
```

```
                    vertical = False
                    forward = True
                    x += 1
                    min_y -= 1

        cnt += 1

    print('\t\t2 done')

    # 3. Interpolating the sample to the size 120 by 120
    # store the data in the 1D structure for single channel
    avg_bands = avg_band_power.reshape((81, 4))
    # coordinate vector in the X dimension - Xs' of the points,
in which we should interpolate the values
    nx = np.linspace(np.min(points[:, 1]), np.max(points[:, 1]),
num=120)
    # coordinate vector in the Y dimension - Ys' of the points,
in which we should interpolate the values
    ny = np.linspace(np.min(points[:, 0]), np.max(points[:, 0]),
num=120)
    # coordinate matrices from coordinate vectors
    grid_x, grid_y = np.meshgrid(nx, ny)

    out = []
    for band in range(4):
        grid = griddata(points, avg_bands[:, band], (grid_x,
grid_y), fill_value=0.0, method='cubic')
        out.append(grid)

    return out
```

4) *Augmentation*

```
def augment_signal(signal, snr=20):
    sig_avg_watts = np.mean(signal ** 2)
    sig_avg_db = 10 * np.log10(sig_avg_watts)

    noise_avg_db = sig_avg_db - snr
    noise_avg_watts = 10 ** (noise_avg_db / 10)

    mean_noise = 0
    return np.random.normal(loc=mean_noise,
scale=np.sqrt(noise_avg_watts), size=len(signal))
```

5) *Splitting data into the training and test sets*

```
    random.shuffle(tab_nr)

    train_amount = int(nr_files * train_split)

    for i in range(nr_files):
        file = files[tab_nr[i]]
        name = str(file[1])
        if i < train_amount:
            temp_path = set_path + "\\" + str(dirs[file[0]])
            if not os.path.exists(temp_path):
                os.makedirs(temp_path)
            copy(directory + '\\' + str(dirs[file[0]]) + '\\' +
name, temp_path)
        else:
            temp_path_test = set_path_test + "\\" +
str(dirs[file[0]])
            if not os.path.exists(temp_path_test):
                os.makedirs(temp_path_test)
            copy(directory + '\\' + str(dirs[file[0]]) + '\\' +
name, temp_path_test)
```

6) *Splitting data into folds*

```
    mod = nr_files % k
    nr_files = nr_files - mod
    nr_sets = k
    nr_samples = int(nr_files / k)

    for i in range(nr_sets):
        k_tem = nr_samples
        # k_tem = k
        if i == 0 and mod != 0:
            k_tem = mod + nr_samples
            # k_tem = mod + k

        # Choosing test samples
        for j in range(ch_point, k_tem + ch_point):
            test_files.append(tab_nr_shuffle[j])
            temp_files[tab_nr_shuffle[j]] = [0, 0]

        # Saving validation samples
        for nr_of_file in test_files:
            file = files[nr_of_file]
            name = str(file[1])
            temp_path_test = set_path_test + "\\" +
str(dirs[file[0]])
            if not os.path.exists(temp_path_test):
                os.makedirs(temp_path_test)
            copy(directory + '\\' + str(dirs[file[0]]) + '\\' +
name, temp_path_test)

        # Saving training samples
        for file in temp_files:
            if file != [0, 0]:
                name = str(file[1])
                temp_path = set_path + "\\" + str(dirs[file[0]])
                if not os.path.exists(temp_path):
                    os.makedirs(temp_path)
                copy(directory + '\\' + str(dirs[file[0]]) + '\\'
+ name, temp_path)
```

7) *Network architecture*

```
def build_model(input_shape):
    model = Sequential()

    model.add(Conv2D(32, (5, 5), activation='relu',
input_shape=input_shape))
    model.add(Conv2D(32, (2, 2), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
    model.add(Dropout(0.5))

    model.add(Conv2D(16, (5, 5), activation='relu'))
    model.add(Conv2D(8, (2, 2), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
    model.add(Dropout(0.3))

    model.add(Flatten())

    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model
```

8) *Training the network (exemplified on the k-fold validation method)*

```
for i, directory_set in enumerate(directories):

    print('Training iteration: ', i+1)
    train_data =
NpyDataGenerator().flow_from_dir(directory=directory_set,
                                 batch_size=batch_size,
                                 shape=input_shape,
                               preprocessing_function=normalize,
                                 validation_split=0.0,

training_set=True)

    val_data = NpyDataGenerator().flow_from_dir(
                              directory=validation_directories[i],
                              shape=input_shape,
                              preprocessing_function=normalize,
                              validation_split=1.0,
```

```
                                training_set=False)

    checkpoint = ModelCheckpoint('best_model',
monitor='val_loss', save_best_only=True)

    model.fit_generator(train_data,
                        steps_per_epoch=64,
                        epochs=5,
                        validation_steps=2,
                        callbacks=[checkpoint],
                        validation_data=val_data)
```

9) *Inference of the model and presentation of the results*

```
def classify(input_shape, test_dir):

    test_data =
NpyDataGenerator().flow_from_dir(directory=test_dir,

shape=input_shape,

preprocessing_function=normalize,
                                                batch_size=8,
                                                shuffle=False)

    model = load_model('best_model')

    output = model.predict_generator(test_data,
steps=len(test_data), verbose=1)

    rcParams['toolbar'] = 'None'

    predicted_classes = np.argmax(output, axis=1)
    true_classes = test_data.classes

    labels = ["Meditation", "Boredom", "Excitement",
"Frustration"]

    pred = []
    true = []

    for i in range(len(predicted_classes)):
        pred.append(labels[predicted_classes[i]])
        true.append(labels[true_classes[i]])

    conf_matrix = confusion_matrix(true, pred, labels)

    width, height = conf_matrix.shape
    annots = np.empty_like(conf_matrix).astype(str)

    sums = np.sum(conf_matrix, axis=1, keepdims=True)
    percentage = conf_matrix / sums * 100

    for i in range(width):
        for j in range(height):
            if i == j:#'%.1f%%\n%d/%d' % (p, c, s)
                annots[i, j] = ("%.1f%%\n%d/%d" % (percentage[i,
j], conf_matrix[i, j], sums[i]))
            else:
                annots[i, j] = ("%.1f%%\n%d" % (percentage[i, j],
conf_matrix[i, j]))

    conf_matrix = DataFrame(conf_matrix)

    fig, ax = plt.subplots(figsize=(30, 15))
    plt.tight_layout(pad=2.5, h_pad=2.5, w_pad=2.5, rect=[0.2,
0.3, 1, 1])

    sns.set(font_scale=1.4)
    sns.heatmap(conf_matrix, annot=annots, fmt='', ax=ax)

    ax.set_xlabel("Predicted", fontsize=30)
    ax.set_ylabel("Actual", fontsize=30)

    ax.set_xticklabels(labels, rotation=45, fontsize=15)
    ax.set_yticklabels(labels, rotation=45, fontsize=15)

    plt.savefig('confusion_matrix.jpg')
```

REFERENCES

[1] Li, Y., Huang, J., Zhou, H., & Zhong, N. (2017). Human emotion recognition with electroencephalographic multidimensional features by hybrid deep neural networks. *Applied Sciences*, *7*(10), 1060.

[2] Al-Nafjan, A., Hosny, M., Al-Wabil, A., & Al-Ohali, Y. (2017). Classification of human emotions from electroencephalogram (EEG) signal using deep neural network. *International Journal of Advanced Computer Science and Applications*, *8*(9).