Real-Time Human Emotions Recognition on Mobile Devices using Machine Learning Algorithms

Beimbet Daribayev[1,2][0000−0003−1313−9004] and Nurkhan Zhaksylyk[2]

[1] University of International Business, Almaty, Kazakhstan
[2] Al-Farabi Kazakh National University, Almaty, Kazakhstan
beimbet.daribayev@gmail.com

Abstract Modern methods of deep learning of neural networks are to find the minimum of some continuous error function. Currently, various optimization algorithms are known that use different approaches to update model parameters. This article is devoted to the analysis of convolutional neural networks with a teacher, as well as the most common optimization methods for learning deep networks in order to recognize the human emotion in the picture. Learning with a teacher is one of the ways of machine learning, during which the test system is forcibly trained with the help of " stimulus-reaction "examples. During the analysis, activation functions such as ReLU, softmax were used. And also, categorical crossentropy was chosen to find errors in the network. And they minimized the error by updating weights to correctly output the response using the Adam optimizer. Corrected a network error using the Dropout regulator. To test the performance of the above algorithms, a mobile application was created for the android platform. And also, we accelerated image processing in the video stream by dividing processes into different threads.

Keywords: Multilayer Perceptron, Neural Network, Activation Function, Weighting Factors, Tensors, Convolutional Neural Network, Gradient Descent.

Introduction

People communicating with each other constantly analyze any human manifestations. It is no secret that one of the important aspects of the analysis is human emotions. Therefore, the creation of modern technology of machine systems is relevant application of methods for automatic recognition of emotions.

One of the common ways of recognizing human emotions by another person is the analysis of visual information. Based on this, it can be understood that the automation of this process should be obvious based on the use of computer vision methods, which is implemented using modern technology of neural networks.

Computer vision is a branch of science where theory and methods are studied, as well as algorithms for analyzing images of objects [1]. This technology is the creation of machines that can detect, track and classify objects. As a scientific discipline, computer vision refers to the theory and technology of creating artificial systems that receive information from images.

Automatic recognition of emotions is quite widespread. For example, recently, Kia

Motors introduced [2] a system that can assess the mood of a person in a car and adjust the atmosphere in the cabin for him.
Such artificial systems are well-implemented using convolutional neural networks. At present, the use of convolutional neural networks allows one to recognize human emotion with various methods of data analysis. For example, one can recognize a person's emotion with the help of an audio recording implemented by Reza Chu [3]. Using this technology, we implemented a network that can recognize the emotion in the picture.
In order to recognize human emotion, a convolutional neural network (CNN) [4] was used, which was trained based on Kaggle [5] data, where gray images with 1 channel are stored in 48x48 pixels format. For 2 reasons, we used this particular method and not a simple multilayer perceptron [6]:
– The percentage of accuracy on the verification data is greater. Due to this, the computer determines the emotion of a person in our case better in our cases
– The concept of total weights. Let´s look at it better to understand the above-mentioned pluses of the CNN. First, let's see how version 2 of the usual multilayer perceptron works. A multilayer perceptron consists of an input, hidden (there may be several of them, depending on the task), output layer (Fig. 1).
As you can see, each node (neuron) in each layer is interconnected. The binding strands of neurons are called synapses, which have their own weights (weighting factors). In machine learning, weights are usually marked with the letter "w", and "X" is the input data that is transmitted to the first layer. To get a response from the output layer, in each layer, all neurons must perform several operations and transfer data to the last layer.
Accordingly, each neuron is 1 pixel of the picture. If we multiply the width of 48px by the height of 48px, then we get the value 2304. Our input layer would have 2304 neurons and since each neuron in each layer is interconnected, we would have many weighting factors. And this is not beneficial for us since it would require a large amount of resources. Conversely, convolutional neural networks are limited by the number of weights. And this has become the main reason for the success of the CNN in the recognition of large objects such as pictures, audio, video, etc. Since our emotions are tensors (48x48 matrix), we used this method to train our model.

To date, the best results in image recognition are obtained with their help. On average, the recognition accuracy of such networks exceeds conventional perceptrons by 10-15%. CNN is the core technology of Deep Learning.
The main difference between a fully connected layer and a convolutional layer is the following: multilayer perceptron layers study global patterns in the space of input features (in the case of emotions from the Kaggle set, these patterns involve all pixels), while convolutional layers study local patterns (Fig. 2).

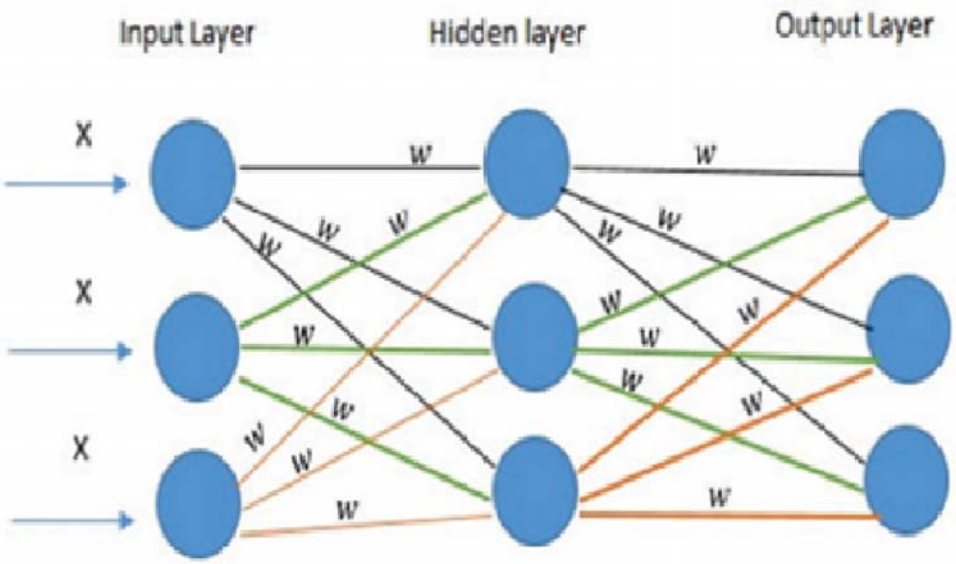

Figure 1. The structure of neural networks.

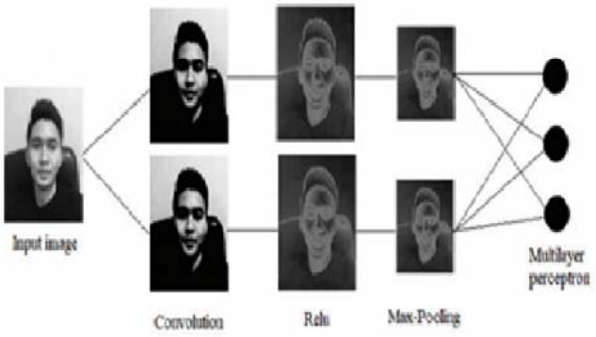

Figure 2. Convolution operation.

Input layer

The input data are gray images of the type JPEG, size 48x48 pixels. If the size is too large, the computational complexity will increase, respectively, the restrictions on the response speed will be violated, the determination of the size in this problem is solved by the selection method. If you select a size too small, then the network will not be able to identify key signs of faces.
The picture with human emotions has only 1 channel (48x48x1). The input layer takes into account the two-dimensional topology of the images and consists of one map (matrices), the map can be one, if the image is presented in shades of gray, otherwise there are 3, where each map corresponds to an image with a specific channel (red, blue and green).
The input data of each specific pixel value is normalized to a range from 0 to 1, according to the formula:

$$f(p, min, max) = \frac{p - min}{max - min} \tag{1}$$

where,
f = normalization function.
p = specific color value in pixels from 0 to 255.
min = minimum pixel value-0.
max = maximum pixel value-255.

Convolutional layer

Convolution is connected to three-dimensional tensors, called feature maps, with two spatial axes (height and width), as well as with the profundity hub (or the hub of the channels). For black and white pictures, as within the Kaggle dataset, the profundity pivot incorporates a measurement of 1 (shades of gray). The collapse operation extricates patterns from its input include outline and applies the same changes to all patterns, producing an output feature map. The measure of the output card can be calculated utilizing this equation:

$$(w, h) = mW - kW + 1, mH - kH + 1 \tag{2}$$

where,

(w,h)-calculatedconvolutioncardsize.
mW-width of previous map.
hW-height of previous map.
kW-core width.
kH-core height.

The core is a filter or a window that slides over the complete range of the past map and finds certain signs of objects. Since we trained the network on numerous faces in arrange to recognize human emotion, one of the cores could give the most noteworthy signal in the process of training in the mouth, eyebrow, eye and nose (Fig. 3).

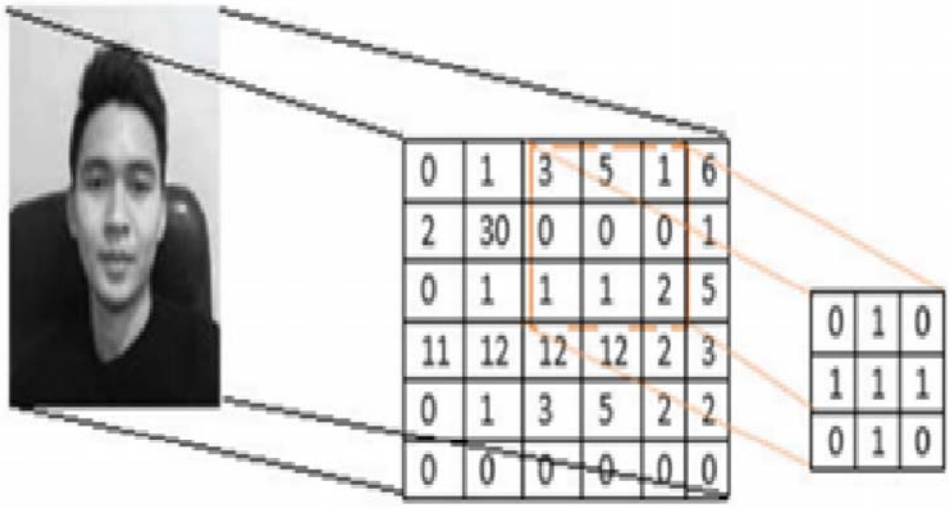

Figure 3. The convolution operation and obtaining the value of the convolution card.

The kernel glides over the past map and performs the convolution operation and transfers the values to the activation function, at that point, the characteristic maps are made (new matrix), the equation:

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l] \tag{3}$$

where,

f-source image matrix,
g-convolution core.

Relu was chosen as the activation function, since it does not lead to problems with attenuation or increasing gradients. And also, for the hidden layers of the perceptron, we used Relu, and the output was Sigmoid.

$$f(s) = max(0, s) - activationfunctionReLU$$

The operation of this function is simple, as you can see this, the function displays 0 if the value passed to the function is s < 0, and if $s \geq 0$ returns the same value.

Having performed the activation function, we transfer the resulting matrix to the subsample layer (Fig. 4).
The reason of the layer is to diminish the measurement of the maps of the past layer. In case a few signs were as of now recognized within the past convolution operation, at that point such a detailed picture is not required for advance preparing, and it is compressed to a less detailed one. In addition, filtering already unnecessary parts helps not to retrain. During filtering by the center of the subsample layer (channel) of the map of the past layer, the checking center does not intersect, unlike the convolutional layer. Ordinarily, each card includes a 2x2 center, which permits you to diminish the past card convolution layer by 2 times. In our task, there will be 3 convolutional layers of 1 subsample layer

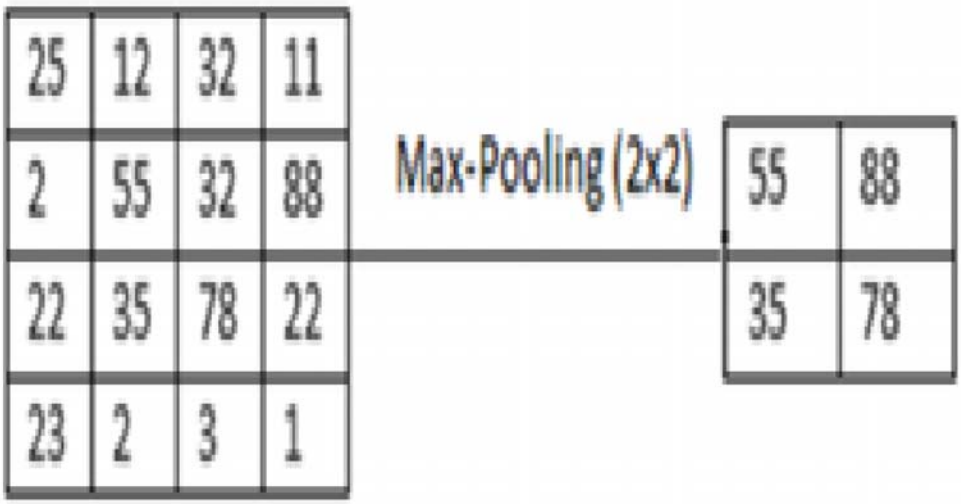

Figure 4. Layer subsampling.

for each and a multilayer perceptron with 1 hidden layer. As you can see after reducing the size, we transfer the reduced image size to the input of the multilayer perceptron without losing important information.
After receiving the vector data, we carry out the training. The essence of training is to reduce the function of loss. Typically executed utilizing the backpropagation strategy. To discover the mistake, we'll utilize los function categorical crossentropy. This work will figure it out by the formula:

$$E = -\sum_{i=1}^{n} D_i * log_2(Y_i) \tag{4}$$

where,

D-expected result, Y-output.

The output is calculated using the softmax activation function. Softmax is a logistic function for the multidimensional case. This means that the function is not applied to a single value, but to a vector. For example, it can be used when the task of multiclass classification is.
The softmax is expressed by the formula:

$$Softmax(x) = \frac{e^{S_i}}{\sum_j e^{S_j}} \tag{5}$$

where,

$$S = \sum_{i=1}^{n} x_i * \omega_i$$

n-number of neurons.

As we said above, in order for the network to work correctly, we must reduce losses. This is implemented using the gradient descent method [7]. For this we will use the Adam optimizer [8].
During the first experiment, we obtained results of loss and network accuracy (Fig. 5a, Fig. 5b):

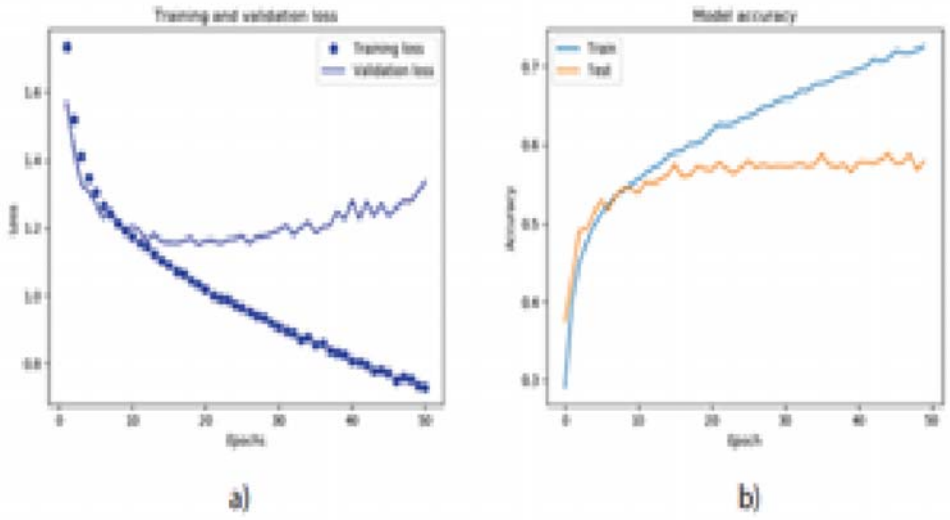

Figure 5. a) Losses on training and validation data in the 50th epochs. b). Accuracy on training and validation data in the 50th epochs.

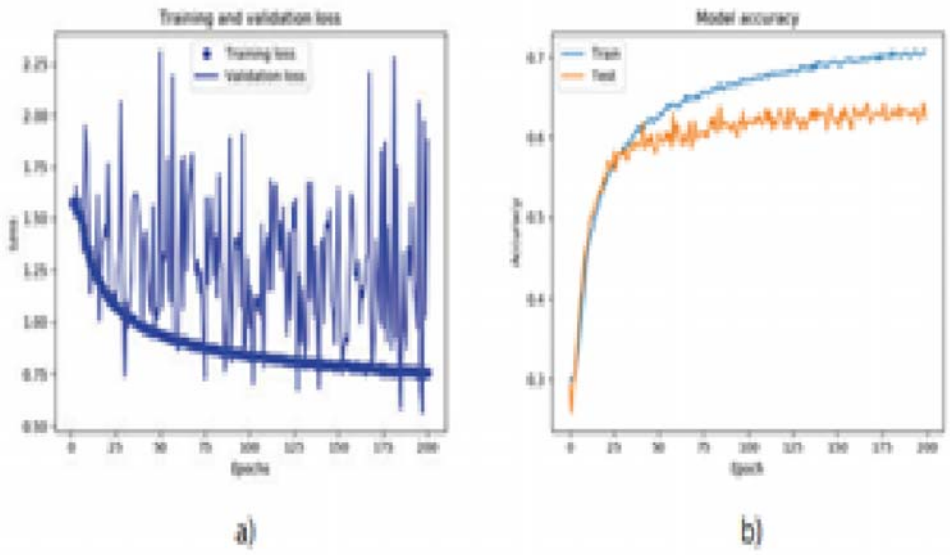

Figure 6. a) Losses on training and validation data in the 200th epochs. b). Accuracy on training and validation data in the 200th epochs.

Increasing the number of epochs from 50 to 200, we got the following results (Fig. 6a, Fig. 6b):

Entrance training model we are confronted with retraining. This term implies that our network recognizes emotion on the training data well, in any case, the precision on the validation data did not rise. Due to this, the strategy of diminishing (Dropout) was used. Retraining could be a issue for numerous neural systems. Due to truth that the network starts to memorize training data, our network will start to memorize excessively. Any methods that we utilize to anticipate retraining is called the regularization method.

One of these regularization methods is loss (thinning), which is utilized in profound systems by counting drop-down layers. This layer does not contain neurons; appropriately, it does not calculate anything. Instead, it incidentallydisengages a few neural from the past layer. Such a layer will be temporarily active in the learning process. When we use the network for prediction, the drop-down layer does not work. The trained model can now be used to implement software. Using this model, a mobile application for recognizing human emotions in real time was implemented on the android platform.

For the implementation, we used the Android Studio development environment, the OpenCV library for image processing. We chose Java as the programming language, and also used our trained model in the tflite format, since the mobile application does not support the h5 file.

The program consists of 5 classes: MainActivity, MultiThread, Classifier and Result. In the first class, we connect the library we need, in our cases it is openCV.

We also initialize files such as Emoji.tflite (our trained model), haarcascade frontalface atl2.xml for high speed face recognition. Our class implements the interfaces: CameraBridgeViewBase.CvCameraViewListener2 for working with the camera, View.OnClickListener for listening to buttons to change the camera. We initialized global variables with the following types: JavaCameraView, File, CascadeClassifier, Mat, Interpreter, int, ImageButton. In the onCreate () method, which sets the initial setting of the initialization parameters, we connect our layout for display, create a JavaCameraView object, load the openCV 3.4.0 version using the OpenCVLoader.initAsync method. On successful loading, we call the onManagerConnected () callback function. In this method, we initialize our cascading face recognition file. After that, we go back to the onCreate () method and start installing our models. The android lifecycle is not only about onCreate (), so we need to implement other methods, in particular onDestroy (). After completing the work or when rolling, we must turn off the camera, for this we use this method. Also, the camera itself has a certain analogue, similar to the life cycle of an android application: onCameraViewStarted (), onCameraViewStopped (), onCameraFrame (). In the first method, we create a Mat object. This object stores images in a two-dimensional array. In the next method, when the camera is turned off, we delete the object, since we need to free memory. In the last method, we load our images that we receive in the video stream. After assigning the image to the Mat, we start the thread to make the program work faster and in this thread we pass 4 parameters to the MultiThread class: Mat, CascadeClasifier, Interpreter, absoluteFaceSize. The parameters passed to the stream are needed in order to recognize the face and draw the recognition result in it. But the work of the first class did not end there. Then we worked to ensure that during installation it was possible to configure access to the camera without going into the settings. For this, built-in methods have been redefined, such as: checkPermission (), onRequestPermissionsResult ().

After completing all the necessary tasks, we switched to the third grade in which we will recognize a person's emotion. In this class, we have created variables with the types Mat, CascadeClassifier, Interpreter, int, ByteBuffer. We assigned the received data from the first class to the variables we needed that we created. The thread that we started in the first class is implemented in the second class. This thread creates a MatofRect object in which we will store our rendered faces in a three-dimensional array. That is, a set of two-dimensional arrays can be stored in MatofRect, which are the faces of the decoded people. This is implemented using the detectMultiScale () method of the CascadeClassifier object. In this method, we pass our Mat, CascadeClssifier, specify scaleFactor, minNeighbors, flags and create a new Size object with the absoluteFaceSize attributes we took from the first class. Next, we create a list for our emotions that we will decode. For the list, ArrayList () was used (Fig. 7).

Next, we create a new Mat object for faces. Each time a face is recognized, a new Mat object will be created. Next, we will draw a rectangle on the found faces. It was implemented using the rectangle () method of the Imgproc object. In this metot, we passed Mat, specified the coordinates using Point () and the color of the square using Scaler (). Further from the picture, we cropped the faces and transferred them

```
ArrayList<String> outputString = new ArrayList<>();
outputString.add("Angry");
outputString.add("Happy");
outputString.add("Neutral");
outputString.add("Sad");
outputString.add("Surprise");
```

Figure 7. List of emotions.

to the newly created Mat object. Then we reduced the cropped image by 48x48 since our model was trained with 48x48 images. Then we created a new class Classifier and passed the Interpreter to its constructor, which we got from the first class, so we create an object of the third class. The putText () method of the ImgProc object was used to draw the label of the resolved emotion. This is where we pass Mat and coordinates to Point (), as well as our text that we got using the classify (matFace) method of the Classifier object. As you can see, we are passing our thumbnails into the classify method.

In the class where we will perform the classification, we created variables:

DIM_ BATCH_ SIZE=1,
DIM_ PIXEL_ SIZE=3,
DIM_ HEIGHT=48,
DIM_ WIDTH=48,
BYTES=4.

We need these variables in order to create a buffer for storing the image in order to use them in the Interpreter. We need an Interpreter to recognize an emotion. The buffer was created using the ByteBuffer.allocateDirect () method. The 1x5 array was used to store the recognition result. The classify method that does the classification converts the resulting Mat into a byte buffer. For this, the convertMatToBteBuffer (Mat) method was created. In this method, if the buffer is empty, we stop the method, if there is data, then we convert Mat to byte and add it to the buffer. We pass the resulting buffer and the array for the result that was created above to the run method of the Interpreter object to get the result of the classification. After that, we create the Result class and pass our result to its constructor. In this class, using the maximum index, we recognize a person's emotion. The result is drawn in the second class.

The developed application can recognize 5 emotions: 1. Angry (Fig. 8a), 2. Happy (Fig. 8b), 3. Neutral (Fig. 9a), 4. Sad (Fig. 9b), 5. Surprise (Fig. 10).

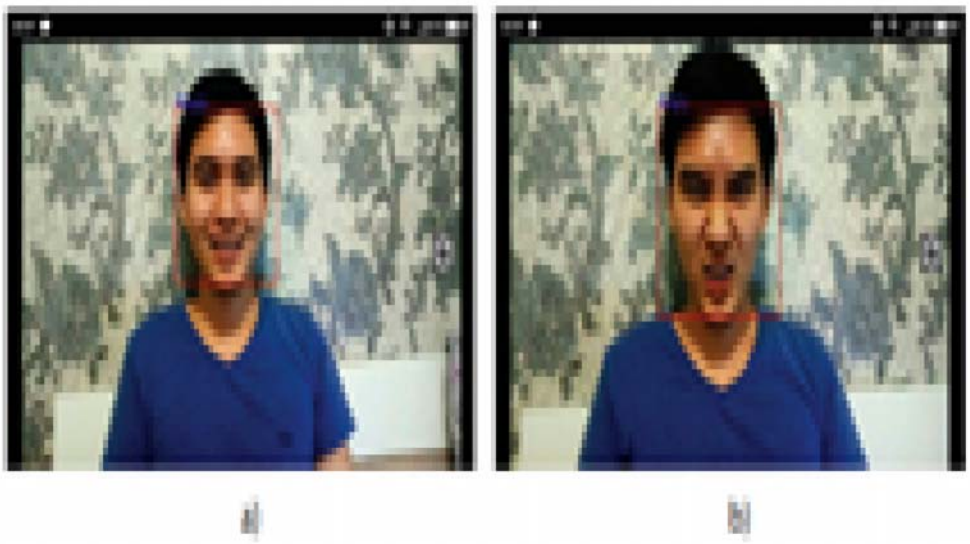

Figure 8. a) Happy. b). Angry

Figure 9. a) Neutral. b). Sad.

Figure 10. Surprise.

Figure 11. Surprise.

Conclusion

During the study, we achieved 70validating data. This is not a bad result for not more data. By further increasing our data, we will improve prediction accuracy.
The given approach to automatic recognition of emotions can be effectively applied in various intelligent human-machine systems. The mobile application of emotion recognition using by machine learning algorithms was developed and image processing in the video stream was accelerated.

References

1. Klette, R.: Concise Computer Vision. An Introduction into Theory and Algorithms. Springer, UK(2014).
2. Emotion Recognition Adjusts the Self-driving Car Interior to the Passenger's Mood, https://nplus1.ru/news/2019/01/07/kia-read. Last accessed 13 June 2019
3. Speech Emotion Recognition with Convolutional Neural Network, https://towardsdatascience.com/speech-emotion-recognition-with-convolutionneural-network-1e6bb7130ce3. Last accessed 19 Sept 2019
4. Chollet, F.: Deep Learning with Python. Manning Publications Co., USA(2018)
5. Data sources, https://www.kaggle.com/deadskull7/fer2013. Last accessed 11 May 2019
6. Rashid, T.: Make your own neural network. CreateSpace Independent Publishing Platform (2018)
7. Glassner, A.: Deep Learning, Vol. 1: From Basics to Practice. Kindle Edition(2018)
8. Neural network optimization methods, https://habr.com/ru/post/318970/. Last accessed 04 Jan 2017