Advanced Topics in Artificial Intelligence - Project

Vision recognition of hand gestures

Wojciech Pęciak, Oriana Tadewosjan

Wrocław 2018

Table of Contents

1. Description of the aim

Our aim is to create application which will be kind of interface between human and computer. It combines gesture recognition and motion tracking. The program based on use of camera vision will identify signal made by hand. We will name five types of gestures:

- Fist 
- Ok 
- Hi 
- Rock 
- Victory 

Assuming that the palm will be contrasting compared to the background and background will be uniform and still.

2. Proposed solution

In used approach we develop feedforward neural network used for error based supervised learning process. Then trained network is loaded in main program to perform real time predictions.

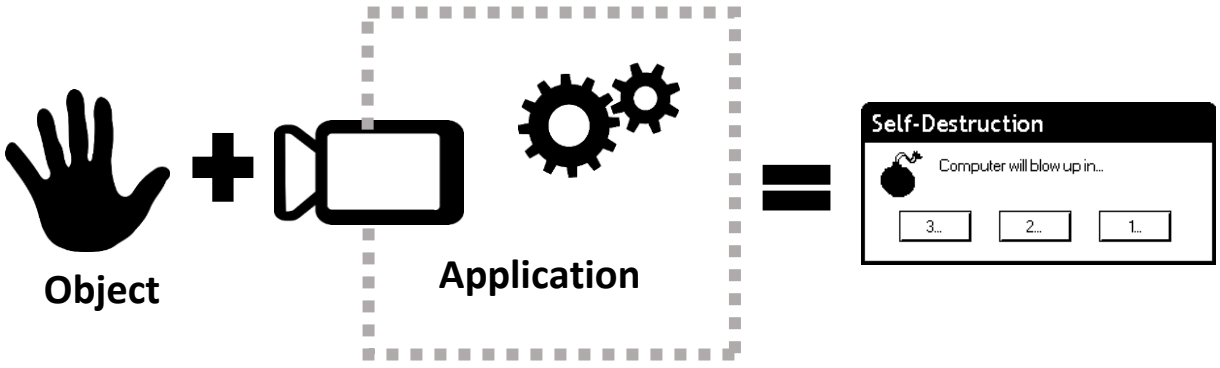

Figure 1 Concept of environment for conceptual application

Program is written in C++ language and based on x64 windows platform. OpenCV library is used for whole task. In details for image acquisition and preprocessing, object detection based on skin color, object classification and execution of action. Additionally, thanks to machine learning part in OpenCV library, we also use it for building and training neural network.

3. About Neural Network

Machine Learning module of OpenCV library implements feed-forward neural networks, more precisely, a multi-layer perceptrons (MLP) which consist of the input layer, one or more hidden layers and output layer. Each layer is made of one or more neurons. Neurons in one layer are connected with neurons in a next layer forming the network. Each connection has a certain weight. Picture below shows a network with 3 layers: three inputs, four nodes in a hidden layer and two outputs.

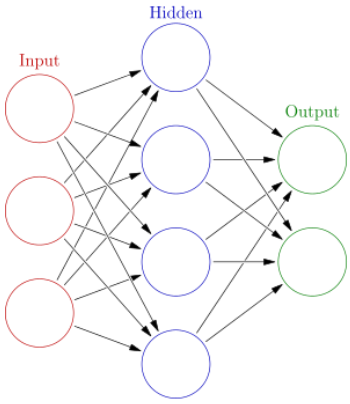

Figure 2 Neural Network example scheme [1]

The goal of a neural network training is to calculate all the weights. To do that, the network must be provided with a set of training data in a form of input vectors (input layer). Each input value is multiplied by initially random weights of a respective connections and summed up with some added bias before passed to the node in a next layer. The sum is transformed with the activation function to make the value fit in a desired range of values. The process is repeated until reaching the last layer. Depending on the used training algorithm, responses in an output layer are evaluated, compared with corresponding given output values. In that way training algorithm can adjust the weights so the network gives correct responses to the provided input. OpenCV implements back-propagation algorithm for MLP training. The only completely supported activation function in OpenCV ML module is symmetrical sigmoid [2] so it chose to use it.

4. Description of main program – AI_app

This program is main part of the work, because from beginning it is responsible for image handling. Thanks to it we were able to gather and prepare database for training and testing parts of neural network development.

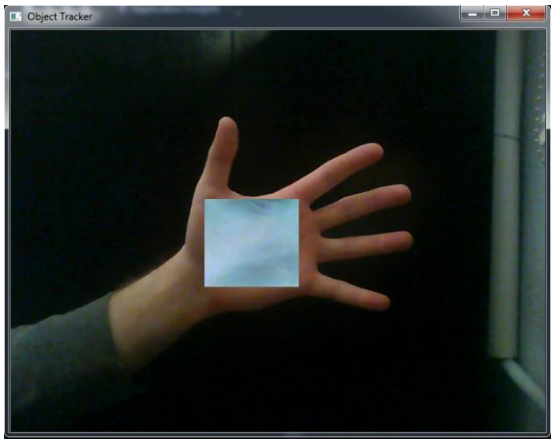

Figure 3 User window - region selection

For start program is in “NO DETECTION” mode. User has to choose region of interest on showed image by mouse like shown in figure 2. Basically program picks every captured frame. Transform its from RGB to HSV color space, thus in this mode calculates ranges of HSV values in selected region. Then switch to “TRACKING” mode. In this case image is thresholded with previously calculated values. Everything in rages is marked as white and pixels out of ranges are marked as black. This operation transform image from three channels type to one channel binary type. Next functions finds edges to find contours and spreads a convex hull on them. Always the one with biggest area is chosen, marked with a rectangle and transferred further to resizing function, which makes samples of 32 by 32 pixels images like in figure 3.

Figure 4 Neural network input before transformation

This part of work was used to produce dataset for neural network.

Later on when developing of network has been finished, new part of program was added. It consist of part transforming image two dimensional matrix to one dimensional row of zeros and ones, which is a form understandable by the network. Then neurons make a prediction. Every one second we receive about 40 answers. For stabilization, user’s output is calculated as most common answer in 0,5 second loop. Last step is to write chosen prediction on upper boundary of rectangle spread on detected object.

From user perspective before starting program we need only to provide name of neural network XML file in defined constant.

5. Dataset description

Dataset consist of 5000 binary images in resolution 32 by 32 pixels stored as JPG. Each of 5 gestures has 1000 samples. Due to the type of task, each single pixel in the image is treated as a input attribute of the neural network, therefore network in first layer has 1024 nodes.

In data set were used two types of hands male left one and female right one.

Additionally validation dataset was prepared with 2500 samples, 500 images per gesture. It was used to make comparison of accuracy in different networks configurations.

6. Description of training and testing program – NN_OpenCV

This part of project was created for training, testing and validating process of prepared neural network.

Basic flow of program handles finding all files in specified dataset directory. Splitting the data into two - training and testing sets. This function offer two types of choosing the samples. Uniform, where every gesture has equal number of training samples or random, where samples are picked randomly. Function randomly picks order of samples. Then data is loaded from JPG files. Each sample is stored in structure containing matrix with image data and vector with appropriate class. After loading process, structures are shuffled once again and transformed into two containers where in the first one, each row in two dimensional matrix is one sample and in the second matrix analogically each row stands for corresponding answer.

From now there are two options: to train network with previously declared topology, save it as XML file and then perform testing part or to load network from file and perform only testing with a dataset.

Each way ends with statistics on accuracy, missed answers and optionally with time of training.

Before starting the program we need to complete the defined constants in the code such as (default values in brackets) :

- `NEURAL_NETWORK_FILE_NAME ("trained_model_5_class_2_hands_uniform_distribution_4_nodes.xml")`
- `DATABASE_PATH ("D:/data_whole")`
- `DATA_SPLIT_TRAINING_RATIO (0.75)`
- `NUMBER_OF_SAMPLES (5000)`
- `ATTRIBUTES_PER_SAMPLE (1024)`
- `NUMBER_OF_HIDDEN_LAYER_NODES (4)`
- `NUMBER_OF_CLASSES (5)`

Default values represent our approach to develop best neural network.

7. Test phase

7.1. Tests of different dataset splitting

In the beginning of development we tested different split ratios for our basic dataset to feed network with appropriate amount of samples, but not to unnecessarily stretch training time.

We gathered statistics for three ratios in two topologies:

Table 1 Comparison of accuracy in different split ratios for two topologies

Nodes in hidden layer	Training ratio	Testing accuracy	Validation accuracy
4	0.5 (2500 samples)	98,56%	93,48%
4	0.75 (3750 samples)	99,01%	92,36%
4	0.9 (4500 samples)	99,40%	93,72%
5	0.5 (2500 samples)	98,44%	92,00%
5	0.75 (3750 samples)	99,41%	94,68%
5	0.9 (4500 samples)	99,40%	93,72%

Table 1 shows that with ratio equal to 50% we have worse accuracy than with ratio equal to 75%. In validating set it reaches difference about 2,5%, but in comparison of 75% ratio and 90% difference is not so clear, because it depends on number of nodes in hidden layer. Taking in consideration longer time for training with ratio of 90%, we assume ratio equal to 75% as the optimal one.

7.2. Tests of adjusting parameters of training process

Initially we chose training parameters, according to these recommended in OpenCV documentation, but we tested different values to see how it influences the accuracy of our network. We did not notice significant changes in classification results.

7.3. Tests of topology

One type of tests touches a problem of choosing the best network topology. To solve this we prepared different types of nets. Starting from 2 hidden nodes ending with 14 hidden nodes. Each type was prepared 3 times and presented statistics are average of registered values.

Table 2 Comparison of training time and accuracy depending on number of hidden nodes

Nodes in hidden layer	Training time [s]	Accuracy (training set) [%]	Accuracy (validation set) [%]
2	950	79,20%	72,84%
3	109	98,64%	93,52%
4	87	99,01%	92,36%
5	208	99,41%	94,68%
6	237	99,68%	93,48%
8	426	99,60%	94,56%
10	590	99,92%	95,48%
14	1022	100,00%	96,68%

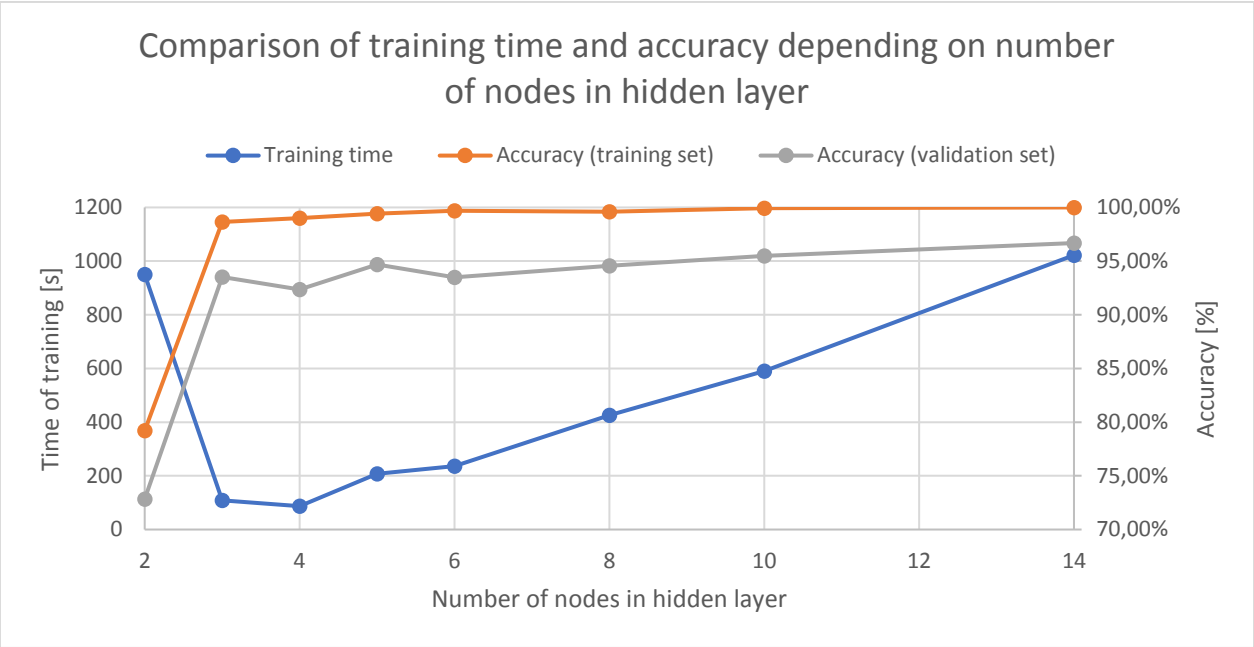

Figure 5 Presenting the correlation between the number of hidden nodes, training time and accuracy

Data presented above gives opportunity to distinguish best network topology. In our case 2 hidden nodes give the worst result. On the other hand we have best accuracy with 14 nodes, but taking into account training times, improvement in accuracy do not balance the increase in time needed for training. In our opinion the optimal solution will be to choose 5 hidden nodes.

Next stage in topology testing was adding second hidden layer. We tried with 4 nodes in second layer, but it decreased accuracy and increased training time giving no advantages. Average accuracy was equal to 94,84% with 530s long time of training.

8. Best configuration and summary

The most optimal for our task configuration of the neural network was chosen basing on tests described above. The most important factor was a relation of a training time and accuracy. The final network consist of 1024 input nodes, one hidden layer with 5 hidden nodes and output layer also with 5 nodes. Below are the statistics showing the performance of chosen network.

Table 3 Statistics about chosen network

	Testing dataset (1250 samples)	Validation dataset (2500 samples)
Correct	1240 (99,2%)	2367 (94,68%)
Wrong	10 (0,80%)	133 (5,32%)

Table 4 Detailed statistics about chosen network

		Fist	Hi	Ok	Rock	Victory
Training dataset (1250 samples)	False positive	1 (0,08%)	0 (0%)	2 (0,16%)	7 (0,56%)	0 (0%)
	Correct positive	249 (99,92%)	250 (100%)	248 (99,84%)	243 (99,44%)	250 (100%)
Validation dataset (2500 samples)	False positive	42 (1,68%)	18 (0,72%)	6 (0,24%)	40 (1,60%)	27 (1,08%)
	Correct positive	2458 (98,32%)	2482 (99,28%)	2494 (99,76%)	2460 (98,40%	2473 (98,92%)

As we can see on a data presented in table 3, our network is good in classifying data provided with a testing dataset, predicting almost 100% accurately. It performs well on a new data with not much noise, however the accuracy drops when network is fed with a real time data provided by a camera. The reason is the noise coming from vison capturing part of our application. Overall performance of the application depends highly on background type so that is why we need it to be still and uniform.

We made a version of a network with an additional 6^{th} class, named “not hand” but it did not improve significantly enough the accuracy of hand gesture prediction of our application to test it further and made training unnecessarily longer.

Code of programs has been loaded into the github repository and is available at this link:

https://github.com/wagon15/Neural_Network_OpenCV.git

[1] https://en.wikipedia.org/wiki/File:Colored_neural_network.svg

[2] https://docs.opencv.org/2.4/modules/ml/doc/neural_networks.html