Student: Rakhlevski Ilia, ID: 316932847

Subject: Development an application that recognizes digits.

Tools: C++, OpenCV, Visual Studio 2017, Python, Spyder.

Project description:

The goal of this project is development of the application that recognizes digits. The application learns to recognize the printed digits (0 – 9). The digits are created in the Paint application and stored in BMP format. The main methods are used for recognition are: Histogram of Oriented Gradients (HOG), Artificial Neural Network (ANN). The project is developed in C++ using OpenCV library. Also, the project includes HOG part developed in Python, demonstrating how the method works. For HOG development I used the article (Note: the code in this article contains bugs and works not always properly). For ANN development I used the code from my course project that I made in course "Artificial Intelligent" during the study in first degree.

Recognition algorithm description:

Recognition algorithm has several steps:

1) Loading image, containing some digit.
2) Converting the coloured image into grey scale format.
3) Rescale the image to size 32 x 32.
4) Filtering the image in order to find the edges of the digit.
5) Divide the image into cells. The image has 16 cells. Each cell has size 8 x 8 pixels.
6) For each cell calculate its HOG. Each HOG contains 18 buckets (360 grad). They are served as input of the ANN.
7) Activate the ANN and recognize the digit. ANN consists from 3 layers: the input – receives values of HOG, the hidden, the output – result of recognition. Sizes of the input and the hidden layers equal the number of all buckets of all HOG. The output layer size is 10 (10 digits: 0 – 9).

Training:

The training process is performed to adjust the weights in order to recognize both a single image and the set of images. The weights are assigned to connections. The connections exist between each node of input layer and each node of the hidden layer, also, between each node of the hidden layer and each node of the output layer.

There are 3 kind of training:

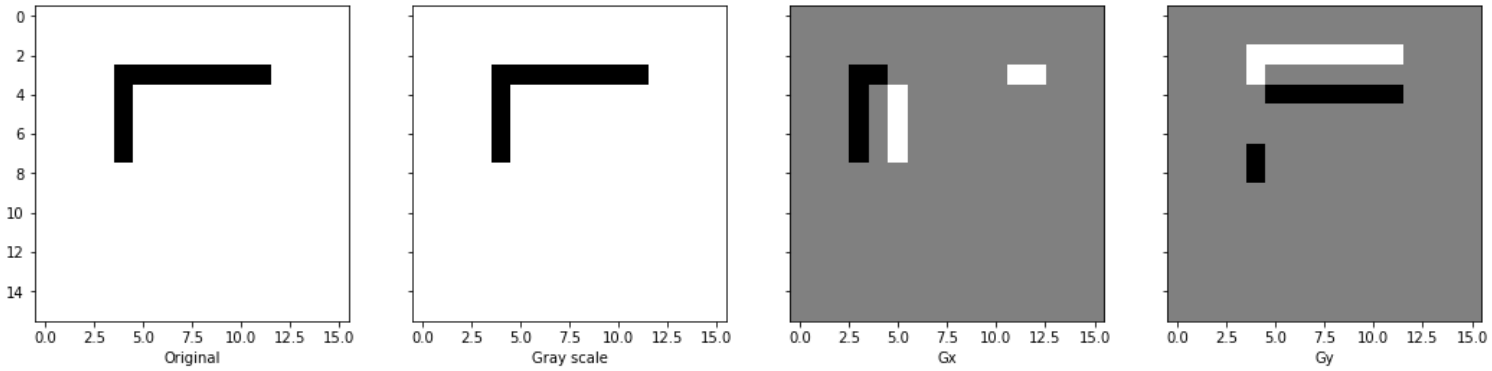
1) One step training. Adjust the weight one time only according to recognition error.
2) Continuous training of a single image. Perform the one step training till the image is correct recognized.
3) Training of set of images. Perform continuous training of all the images in the set till the correct recognition of all these images.

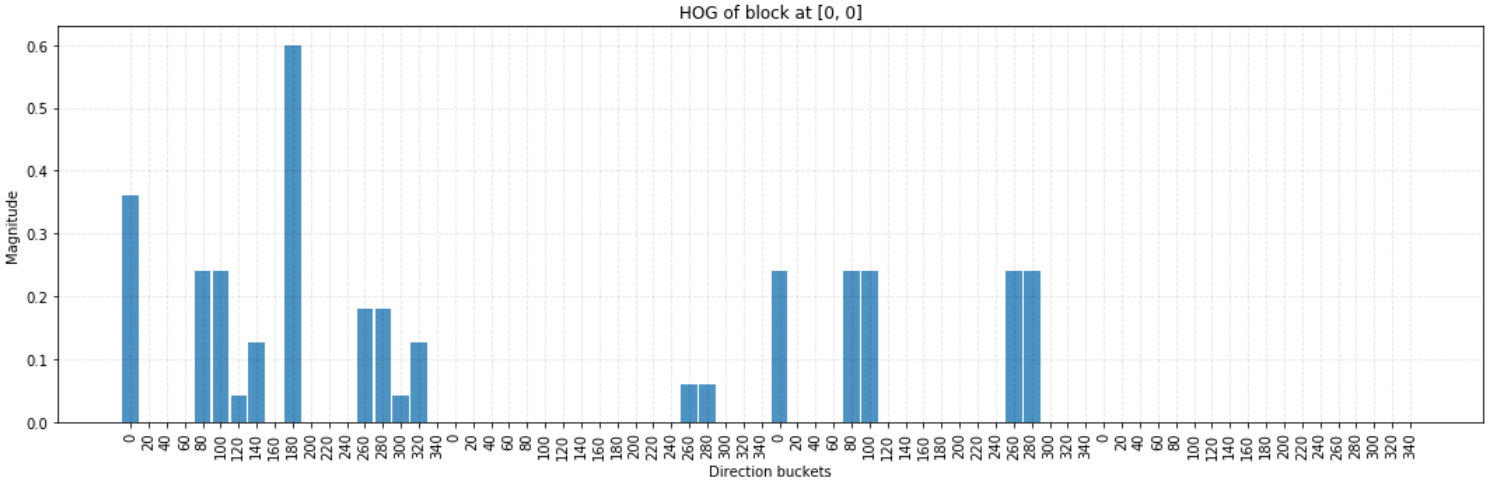
Using of the applications:

Python application:

Files: hog.py, figure.bmp.

Run the script. I use Spyder, IPython. The script loads ‘figure.bmp’ image, create four histograms of oriented gradients (one histogram for each cell), print the images (original, grey scale, edges) and the HOG diagrams. Output:





C++ application:

Run the application. You see the menu:

```
0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice:
```

0 – Exit the application.

1 – Load the image ‘digit.bmp’ that is found in the same directory. The image must contain a digit.

2 – Recognize the previously loaded image. Use menu item 1 to load the image.

3 – Start the process of current loaded image training. In this case you must assign the digit you

are going to train. Select the menu item 4 to do this.

4 - Assign the digit to be trained. The digit must be contained in the loaded image.

5 – Train the images set. These are the images are found in the ‘training_images’ directory.

6 – Save current weights in file ‘weights.dat’.

7 – Load weights saved in file ‘weights.dat’.

Examples of the application work:

Train the ANN to recognize digits from the set of the images.

1) All training images are found in the ‘training_images’ directory.
2) Run the application.
3) Select the option 5 in the application menu.
4) After finish of the training save the weights. Select the option 6.

Recognize a digit.

1) Create an image containing a digit in BMP format or take it from the ‘testing_images’ directory.
2) Copy the image to the application directory and rename it to ‘digit.bmp’.
3) Run the application.
4) If you have already saved the weights load them. Select the option 7.
5) Select the option 1 to load the image ‘digit.bmp’.
6) Select the option 2 to recognize the digit.

```
0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 7


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 1


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 2
```

```
Recognized number: 5


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice:
```

Training of a single image.

1) Run the application.
2) Load the weights (optional) – select the option 7.
3) Load the image saved in the ‘digit.bmp. file. Select the option 2.
4) Assign the digit toy are going to train. Select the option 4. You must choose the digit which is contained in the loaded file.
5) Select the option 3 to train the digit.

```
0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 1


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 0)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 4
Enter the digit to be trained: 5


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 5)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice: 3
```

```
Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 7

Recognized number: 5


0 - Exit
1 - Load image
2 - Recognize
3 - One image training
4 - Digit to be trained (current: 5)
5 - Images set trainig
6 - Save weights
7 - Load weights
Your choice:
```

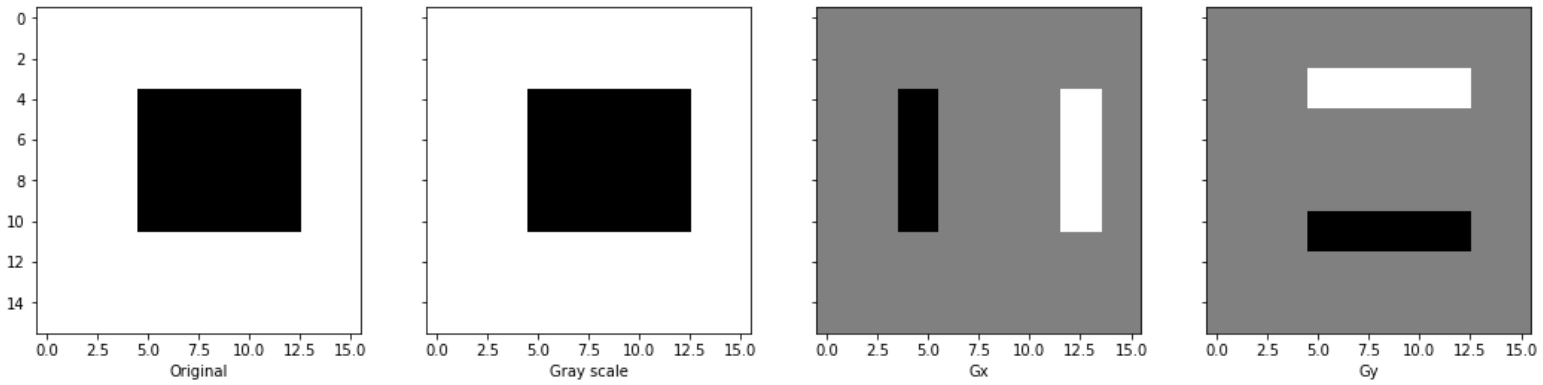
Notes:

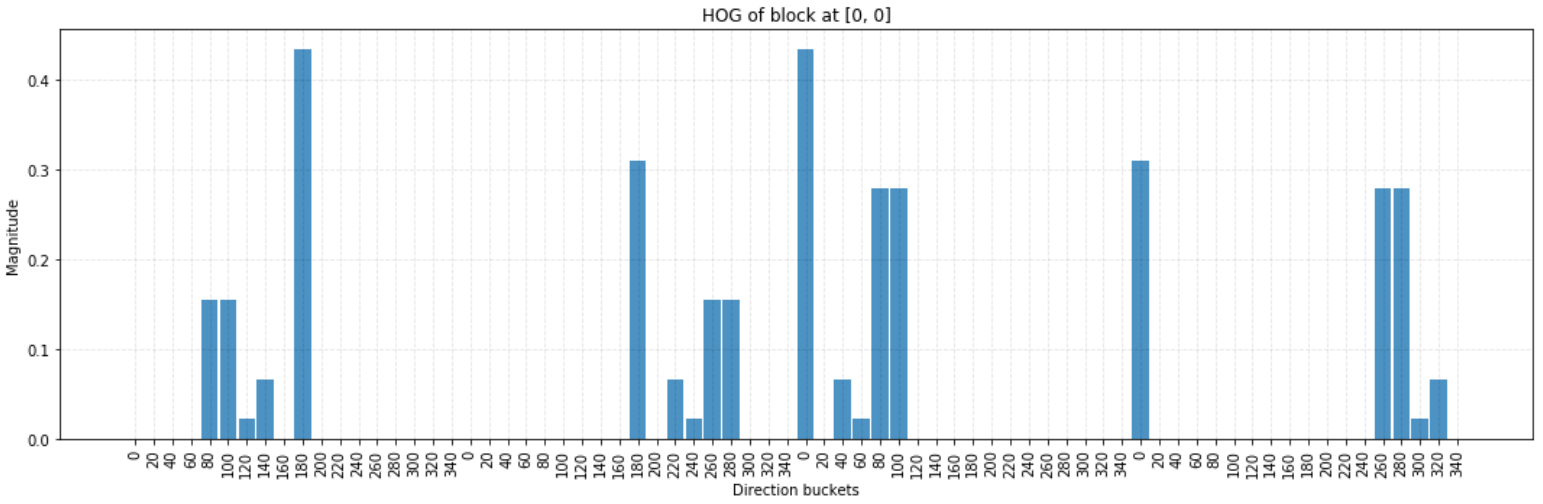
1) The application uses the OpenCV DDL: opencv_core400d.dll, opencv_highgui400d.dll, opencv_imgcodecs400d.dll, opencv_imgproc400d.dll. They are not included to the project because of their sizes.
2) The 'training_images' directory contains the images for training. All the images have the same size: 32x32. Format of the image name: <digit>_<font name>. Example: 6_Tnr – image contains the digit 6 painted with font Times New Roman.
3) The 'testing_images' directory contains the images for testing. The images were created with different sizes and fonts.

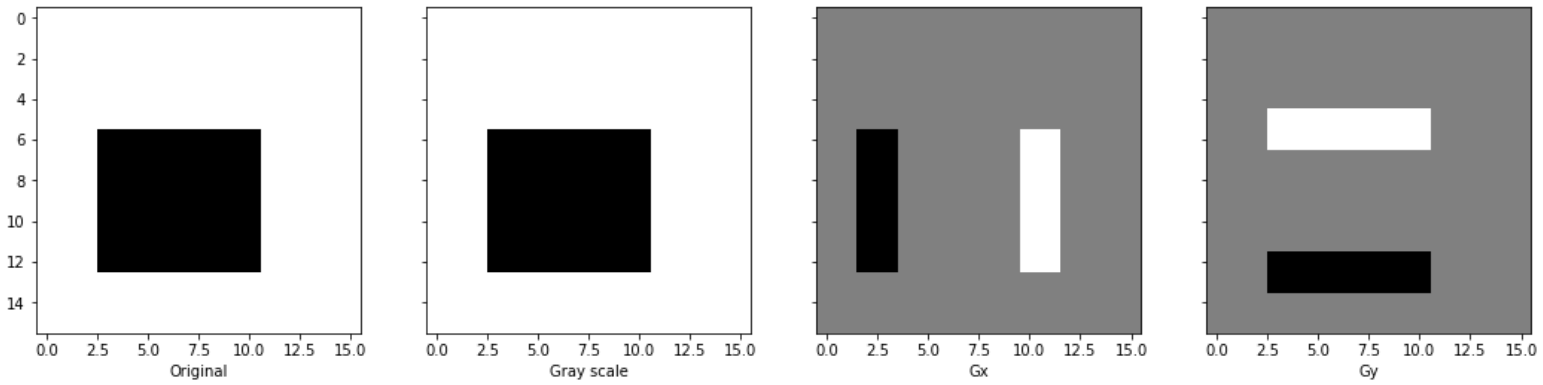
Comparison of the used algorithm with others:

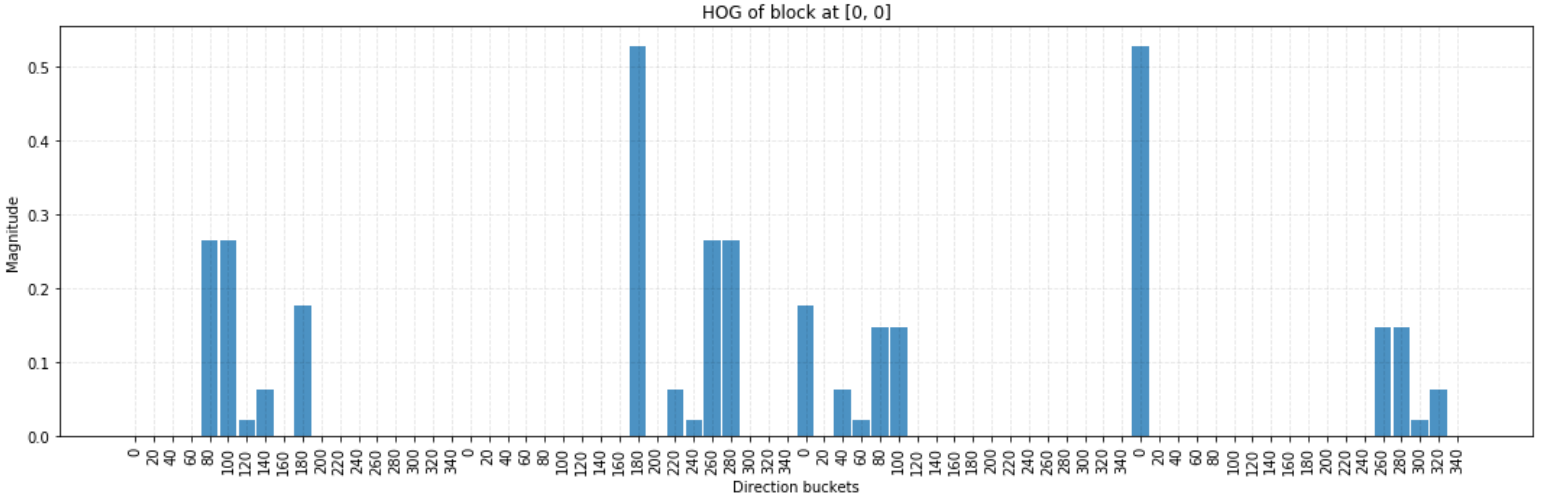
In my project I use the method Histogram of Oriented Gradients (HOG) + Artificial Neural Network (ANN). Except of these methods for pattern recognition also are used other methods such: SVM, KNN. Within this course I have no opportunity to investigate all these algorithms. So first of all, I have chosen the ANN because I learned this algorithm while my study for the first degree and I have already some experience with using of the ANN for digits recognition. But in this project, I have added the HOG algorithm to improve the digits recognition. How it can help me?

When we use ANN for digits recognition, we assign certain areas of an image to relevant inputs of an ANN. But what occurs if we change a digit location on the image? In this case other inputs of the ANN will be activated and we can receive incorrect output. The HOG helps us to solve this problem. In the next example we create a histogram for the same image, but the figure on this image is shifted.









The same figure has the different locations, but both histograms are very similar and we can use them as inputs of the ANN. In this case the result is more accurate. This is the reason that HOG + ANN is better

than ANN.

According to the Internet information: SVM and ANN give about similar results, but KNN works worse.

Example of such comparison:

Performance analysis of K-nearest neighbours, support vector machine, and artificial neural network classifiers for driver drowsiness detection with different road geometries