Recognition of Faces, Head Positions, Gender, Age, and Emotions In Real Time Using Deep Convolutional Neural Networks

Eldar Kurbanov
Bachelor's student.
Volgograd, Russia
EldarK@outlook.com

Aleksei Gordeev
Supervisor.
Volgograd, Russia
alexurgor2008@gmail.com

Vladimir Klyachin
Ph.D. Physical and mathematical sciences.
Volgograd, Russia
klyachin.va@volsu.ru

Institute of Mathematics and Informational Technologies
Volgograd State University

Abstract

This paper shows an example of solving the task of real-time recognition of faces, head poses, genders, ages and emotions from USB camera using deep convolution neural network on a Raspberry Pi 3 Model B with Intel Neural Compute Stick 2.

Also, this paper consists some theoretical information about building your own deep convolutional network like VGG Face and AlexNet.

1 Why do we need face recognition?

The task of automated face recognition in real time was set in the last century and today it is perfectly implemented in many software and hardware complexes, including on mobile apps.

Face recognition technology has reached an incredibly high level: it became possible to recognize specific people, including wearing glasses and headgear. This method of identification is recognized so accurate that it has already begun to crowd out the usual methods of identification with a password[1].

However, we will not solve the task of identifying an individual by face. Development of a specific face recognition system requires a specific train data, containing certain individuals. The sample should be great for quality detection. Creating such a sample for specific individuals is a very long and expensive process (since it requires a lot the number of photos of specific people in different situations). While creating a sample for easy recognition individuals are much easier and faster. Because we can make a large training sample using data from the Internet.

In: S. Hölldobler, A. Malikov (eds.): Proceedings of the YSIP-3 Workshop, Stavropol and Arkhyz, Russian Federation, 17-09-2019–20-09-2019, published at http://ceur-ws.org

On the input video stream, we recognize the face, try to guess the gender, age of the person, as well as his current emotion.

The solution to this task is one of the stages in the design of our future robot, which can detect and recognize people faces, head pose, gender, age and current emotion.

2 Deep convolutional neural network

How is the face recognition task solved? More recently the most popular algorithms of face recognition were principal component analysis, template matching, method of Viola-Jones and Hopfield network[10]. We will solve the task of face recognition with the help of more modern method - deep convolutional neural networks.

At the moment, the convolutional neural network and its modifications are considered the best algorithms for finding objects on the stage in accuracy and speed. Since 2012, neural networks have occupied first places in the well-known ImageNet international pattern recognition competition.

Below, we provide methods for training only a few of the neural networks we use, and solely for example.

2.1 Making neural networks like VGG

This section shows - how to train VGG-like neural network from scratch on the training data[11].

2.1.1 Prepare of dataset

In beginning, of course, we had to prepare dataset to use. As a dataset, we used the Fer2013[1] from the Internet. It consists image data and file with information about images in CSV format. Our goal is to now take this CSV file and convert it to series of HDF5 datasets for training, validation and testing so we can more easily train out CNN on top of it.

Our images is a string of integers, so we need to take this string, split it into a list, convert it to an unsigned 8-bit integer data type, and reshape it to a 48x48 grayscale image. The next step of parsing CSV file is check the usage and assign the image and label to the respective training, validation and testing lists. The last step in preparing of dataset is a loop over each of the training, validation and testing sets, get images and labels from it and writing them to disk in HDF5 format[11].

2.1.2 Modeling a neural network

This network inspired by the family of VGG networks and so we agree to do the following:

1. The convolutional layers in the network must be only 3x3.
2. Double the number of filters learned by each convolutional layer the deeper we go in the network.
3. Initialize our convolutional and full connected layers using the using the MSRA/He-et-al method[2] - doing so will enable our network to learn faster.
4. After every convolutional layer apply an activation followed by a batch normalization.

For building of Convolutional Neural Networks we chosen the deep learning framework Caffe because the Intel's Model Optimizer has support of this framework and the topology VGG Face and our network is just VGG-like network[3]. More information about Model Optimizer you see in section 3.3.3.

The so-called rectifier was taken as a function of activation of neurons. In recent years, this activation function has gained great popularity. Neurons with this activation function are called ReLU (rectified linear unit). ReLU has the following formula $f(x) = max(0, x)$ and implements a simple threshold transition at zero. The graph of function at Figure 1[12].

Each block in convolutional block in our network consists of (Convolutional $->$ ReLU activation function $->$ batch normalization) * 2 $=>$ pooling layer sets. The first convolutional layer will learn 32 3x3 filters. Then apply an ReLU activation followed by batch normalization. The second convolutional layer has same pattern. We then applied max pooling and then a dropout layer with probability 25%. The second block in out network

[1]This database uses in challenges in Representation Learning: Facial Expression Recognition Challenge. The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image[15].

[2]He Normal (He-et-al) Initialization of weights

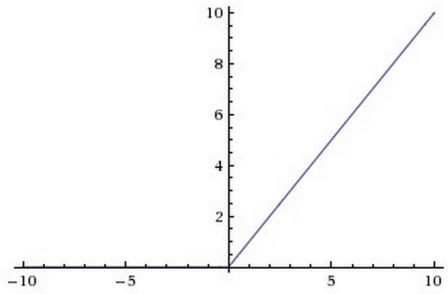


Figure 1: Rectifier function graph

is identical to the first, and we are doubling the number of filters in the convolutional layers to 64 rather than 32. The moving on to the third block, we again apply the same pattern, now increasing the number of filters from 64 to 128 - as we get deeper in the CNN, the more filters we learn.

Next, we needed to construct first fully-connected layer set. We learn 64 hidden nodes, followed by applying an ReLU activation function and batch normalization. Then we apply a second full connected layer in same manner and finally we apply a full connected layer with the supplied number of classes along with a softmax classifier to obtain out output class label probabilities[11]. A complete neural network diagram is shown in Figure 2.

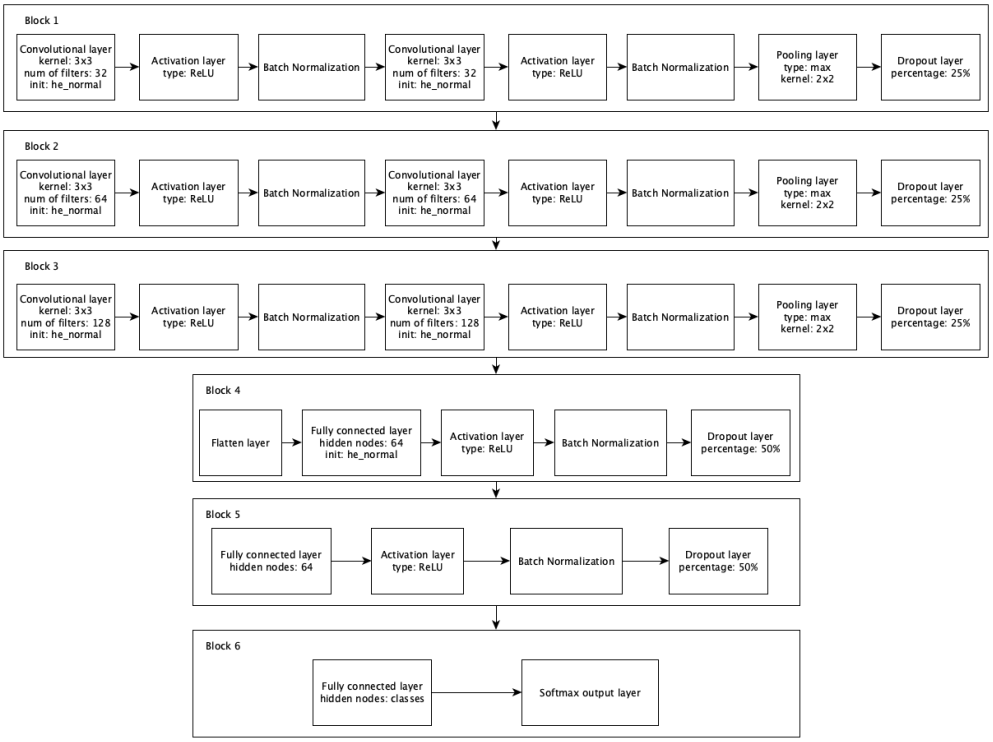


Figure 2: Complete neural network diagram for face and emotions detection

2.1.3 Training the CNN

We was apply data augmentation to the training set to help reduce overfitting and improve the classification accuracy of our model. Earlier we stored our images in dataset as raw, unnormalized RGB images, meaning that pixel values are allowed to exist in the range [0, 255]. However, its common practice to either perform mean normalization or scale the pixel intensities down to a more constricted change. So, we chose scaling and we need

to provide an rescale value of 1/255 - every image will be multiple by this ratio, thus scaling the pixels down to [0,1].

Further we provide path to input HDF5 files, batch size and number of classes in our dataset and thus we have already indicated all the necessary data. All preparations have been made, so obviously the next step was already the long process of training the neural network. The graph of training shows in Figure 3[11].

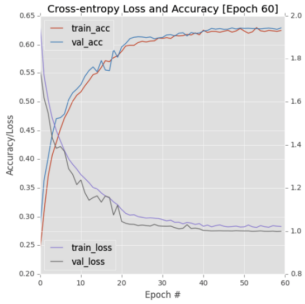


Figure 3: VGG-like neural network training

2.2 Making neural network to detect age and gender

This section shows how we are implementing deep convolution network for detecting the age and gender on image. At this time we are skip the step about preparing of dataset (we used the Adience[3] dataset from the Internet).

2.2.1 Implementing network architecture

This network architecture is similar to AlexNet[4], but:

1. More shallow with no multiple convolutional and ReLU layers stacked on top of each other.

2. Fewer nodes in the fully-connected layers.

3. We utilized batch normalization rather than the now deprecated Local Response Normalization used in earlier convolutional neural network architectures.

4. We introduced a small amount of dropout after every pooling layer to help reduce overfitting.

We defined a series of convolutional, ReLU and pooling layer.

The first convolutional layer in the network learned 96, 7x7 kernels with a stride of 4x4 used to reduce the spatial dimensions of the input 227x227 images. An activation is applied after the convolution, followed by a batch normalization. Max pooling is applied with a kernel size of 3x3 and stride of 2x2 to once again reduce spatial dimensions. Dropout with a probability of 25% is used to help reduce overfitting.

Our second series of layers applies the same structure, only this time adjusting the convolutional layer to learn 256, 5x5 filters. The final convolutional, ReLU, and pooling layer set is near identical, only the number of filters is increased to 384 and the filter size reduced to 3x3. The next comes our first set of fully-connected layers.

Unlike the AlexNet architecture where 4096 hidden nodes are learned, we only learn 512 nodes here. We also applied an activation followed by a batch normalization in this layer set as well. The second set of fully-connected layers is then applied. And the final step is simply an fully connected layer to learn desired number of classes along with a softmax classifier[11]. A complete neural network diagram is shown in Figure 4.

[3]The dataset included in this collection is intended to be as true as possible to the challenges of real-world imaging conditions. In particular, it attempts to capture all the variations in appearance, noise, pose, lighting and more, that can be expected of images taken without careful preparation or posing[16].

[4]AlexNet is the name of a convolutional neural network, designed by Alex Krizhevsky, and published with Ilya Sutskever and Krizhevsky's PhD advisor Geoffrey Hinton, who was originally resistant to the idea of his student[13].

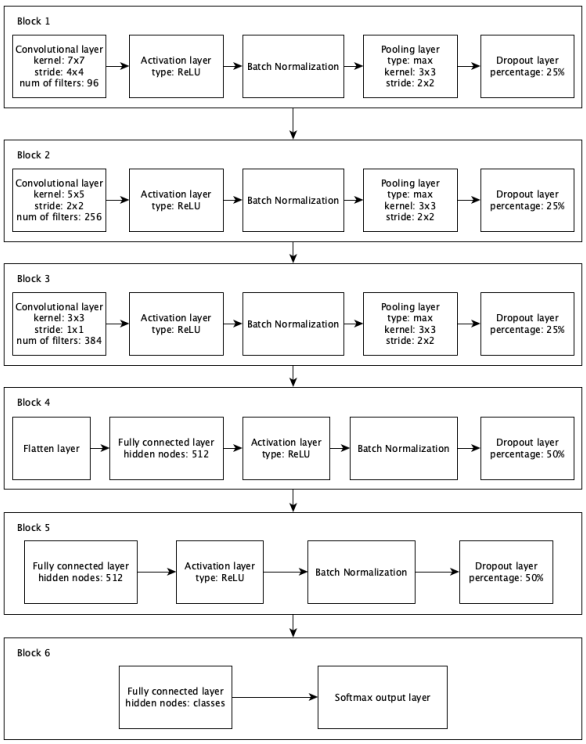


Figure 4: Complete neural network diagram to detect age and gender

2.2.2 Training the network

First, we handles loading our RGB means for either the age of gender dataset, respectively. Since our images are already pre-aligned in our dataset, we don't want to apply too much rotation, so a rotate of $\pm$ 7 degrees seems appropriate here. We was also randomly crop 227x227 regions from the input 256x256 image.

What about learning optimization? We used the SGD[5] optimizer to train our network. So, the preparing steps are complete, the final step is a training of our network[11].

In this part of the document, we described the process of creating a neural network. There are 4 such neural networks in our work. The basis of the construction of the remaining neural networks remained the same meaning and the same principles as described above. The general scheme for building our neural networks is shown in Figure 5.

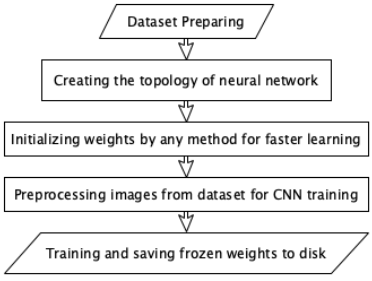


Figure 5: The workflow of building of our neural networks

[5]Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It is called stochastic because the method uses randomly selected (or shuffled) samples to evaluate the gradients, hence SGD can be regarded as a stochastic approximation of gradient descent optimization[14].

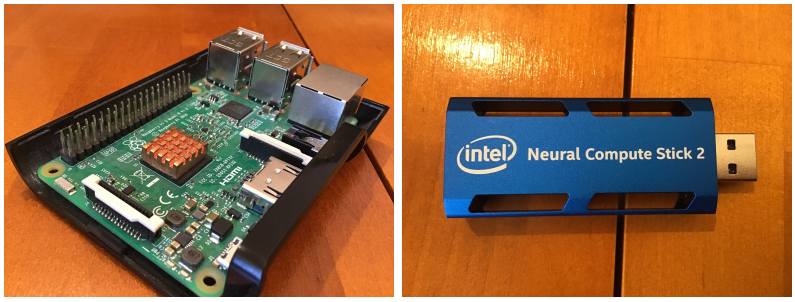
3 What else is needed to solve the task?

Now we have trained neural networks, it's time to find good use for them.

3.1 The equipment

Since our common task is to build a mobile autonomous robot with video analytics, we have the following requirements for the recognition system: mobility, compactness and low power consumption.

For the decision it was decided to use the Raspberry Pi 3 Model B - single-board mini-computer shown on Figure 6a.



(a) Raspberry Pi 3 Model B (b) Intel Neural Compute Stick 2

Figure 6: The equipment of robot

Real-time face detection on the Raspberry Pi 3 Model B works outright badly, yielding less than 1 FPS. In this regard, we will be forced to use additional computing power[4].

That's why we choose the recently released Intel Neural Compute Stick 2 shown on Figure 6b. This is an auxiliary USB device of compact dimensions on which the neural network will be placed and worked. Also we need an USB camera. We will used Logitech C310.

3.2 Benefits of this equipment

The advantage of this choice of hardware is its relatively low cost, compactness, and cost-effective power consumption during operation. The complex is easy to place, connect, maintain.

The Intel Movidius Visual Processor Myriad X on Intel Neural Compute Stick 2 uses 16 powerful computational cores (another name is SHAVE) and has a specialized neural network hardware accelerator to support high-performance computer vision and artificial intelligence data processing applications, along with very little power consumption[5].

3.3 Software components

Raspberry Pi 3 Model B running the Raspbian 19 operating system. To ensure the interaction between Raspberry Pi and Intel Neural Compute Stick 2, we will use the OpenVINO toolkit, which provides an API for interacting with a computer stick[6].

3.3.1 What is OpenVINO?

OpenVINO is a toolkit designed for developing software for the Intel platform and its abbreviation stands for Open Visual Inference and Neural Network Optimization. It contains the Deep Learning Deployment Toolkit (DLDT) for Intel processors (for CPUs), Intel Processor Graphics (for GPUs), and heterogeneous support[7].

3.3.2 Structure of OpenVINO

The OpenVINO toolkit consists of several components:

1. Intel Deep Learning Deployment Toolkit - a developer toolkit that allows you to deploy pre-trained deep learning models through a high-level interface. The toolkit supports execution on a processor, video card, external device (for example, Intel Neural Compute Stick and FPGA (Field-Programmable Gate Array). In our case neural network will execute on USB device Intel Neural Compute Stick 2 based on Intel Movidius

Visual Processor Myriad X, which belongs to VPU type of devices. This hardware, OpenVINO and the firmware embedded in VPU are intended for the Inference Engine of real-time neural networks.

2. Model Optimizer (section 3.3.3)

3. Inference Engine (section 3.3.4)

3.3.3 Model Optimizer

To successfully use a pre-trained Deep Learning neural network, it is necessary to convert it from our legacy format used in deep learning frameworks such as TensorFlow, Caffe (our case), ONNX, to an optimized IR model, which consists of BIN and XML files. For such a conversion, you can use the included Model Optimizer program. The typical workflow for deploying a trained deep learning model is shown on Figure 7[3].

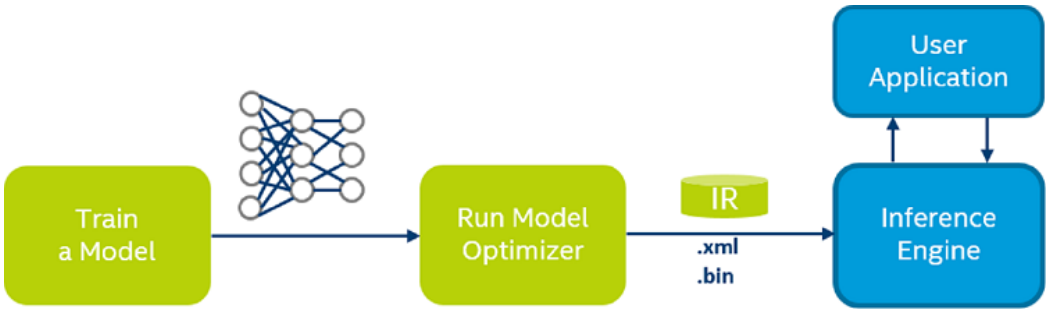


Figure 7: Typical workflow for deploying a trained deep learning model

Model Optimizer generates BIN and XML files. The BIN file contains frozen neural network weights. The XML file is a description of the topology of neural network, serialized a set of layers with their types[8].

Further, this optimized model is loaded into the Inference Engine, which already has all the necessary tools and optimizations for execution on various types of devices (described in Figure 3.3.2).

The model optimizer is a prerogative of external software (external framework).

3.3.4 Inference Engine concept

We need to take a pre-trained model and prepare it for output with the help of special software - the Model Optimizer from OpenVINO.

The optimizer performs various kinds of model manipulations, optimizes it and converts it into IR format. All these manipulations are hardware independent, but the result of OpenVINO can be played on various devices[8].

Inference Engine is certain API for providing inference to optimized pre-trained neural networks operated in real time. Inference Engine API is common to all hardware devices and supports C ++ and Python. In our case Inference Engine is located at MYRIAD firmware. It is also worth noting that the Inference Engine is able to work simultaneously with several neural networks at the same time, which we used in our work by launching several neural networks simultaneously.

Thus, the logic engine has different plug-ins for each of the supported devices. When the API developer calls, the application developer must indicate the device on which its model should run.

The logical inference mechanism will use the necessary plug-in and thus use the correct hardware implementation of the functions necessary for the neural network model (for example, multiplication can be performed on the processor using the assembler command, and the OpenCL command is already needed on the video card).

Our application takes an frame from video camera and send it to Inference Engine Common API. Inference Engine processing frame through neural network and on output we have parameters which should be handle by our application. The Inference Engine structure is shown on Figure 8[9].

3.3.5 Preparing for launch

Operation process using Inference Engine plugin on our device (MYRIAD) looks like on Figure 9.

Before a neural network is sent to a specific plugin which takes optimized neural networks as inputs obtained from model optimizer, it goes through several stages of optimization:

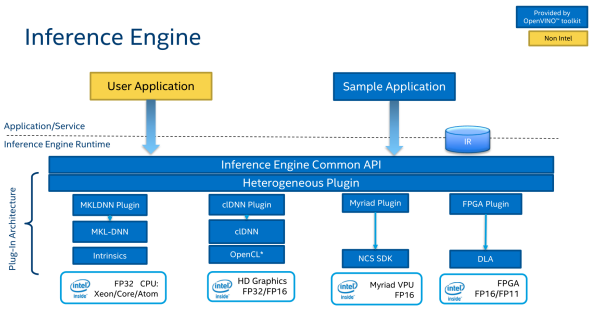


Figure 8: Inference Engine structure

1. Optimization at the neural network level: this includes the mapping of operations used in the neural network to a specific core. By performing this step in advance, the Inference Engine improves performance and minimizes output time.
2. Memory level optimization. This step basically involves reordering the data in memory according to the specific requirements of the device.
3. Kernel Optimization. The output mechanism selects the right implementation that is best suited for the instruction set of the target architecture.

3.3.6 The basic idea

The model optimizer performs a hardware-independent optimization, the output mechanism selects the correct plugin for the target device and performs device-specific optimization.

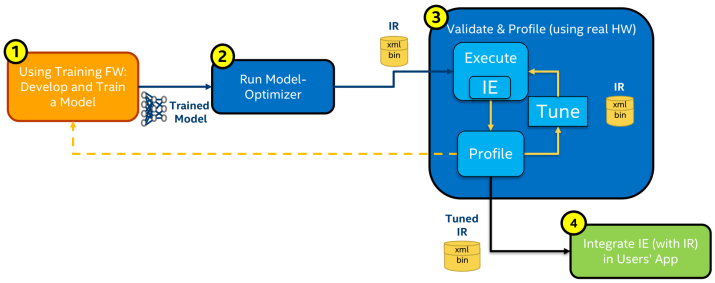


Figure 9: End-To-End Workflow

4 Display the result on the screen

So, the general scheme of the application is shown in Figure 10.

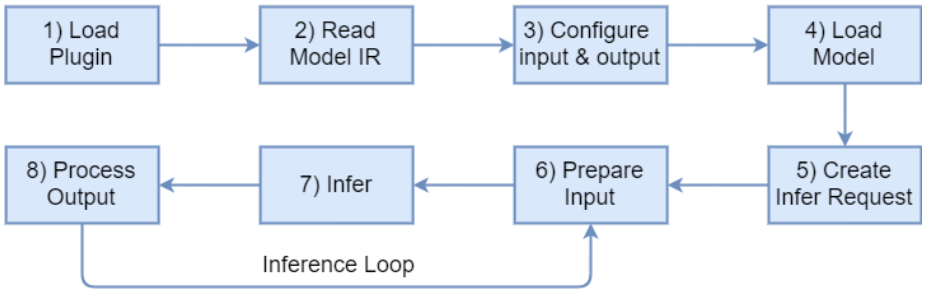


Figure 10: The general scheme of the application

4.1 Used models

To demonstrate the advantages of the Neural Compute Stick 2 in our developed mobile robot, we used our builded convolutional neural networks in section 2.1. A total of 4 neural networks were trained and used:

- The Face Detection model is the main model. It allows you to identify faces on the image, includes convolutional depth data, reducing the amount of computation for a 3x3 convolutional block.
- The Age Gender Recognition model is a convolutional network for simultaneous recognition of age and gender. The network can recognize the age of people in the range of [18, 75] years.
- The Head Pose Estimation model is a head posture assessment network based on CNN architecture. The angular regression layers are convolutions, and packet norms completely associated with a single output.
- The Emotions Recognition model is a fully convolutional network for recognizing five emotions (neutral, happy, sad, surprise, anger).

As mentioned in Chapter 5, these neural networks will be simultaneously launched in real time on the MYRIAD processor: this operation allows us to do the Inference Engine.

4.2 Displaying the result

So, we have a Raspberry Pi 3 Model B, one Neural Compute Stick 2 connected via a USB cable, USB Camera, and our trained and then converted neural network. It remains only to install Raspbian, compile on it an open library of computer vision OpenCV, install the OpenVINO library and write the code.

To display the result on the screen we wrote the code on C++ in which we used the open computer vision library OpenCV and OpenVINO toolkit provided by Intel.

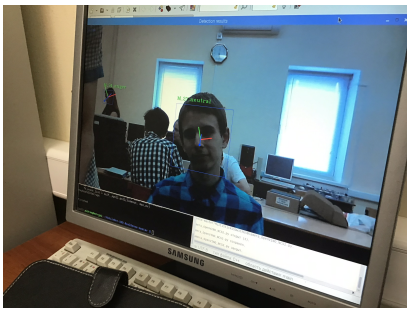


Figure 11: Photo of work

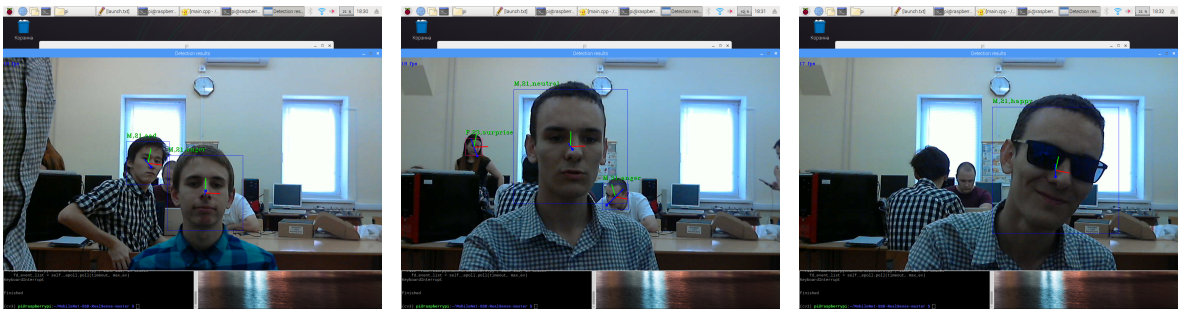


Figure 12: Work program screenshots

5 Resume

In Figure 11 and Figure 12 you can see an example output of the program.

Thus, we solved the task of recognizing the faces, positions of the head, gender, age and emotions of people in real time using deep convolutional neural networks. The main advantage of this system is good performance with such low cost and mobility of the device.

We have achieved 18 FPS. For better results, we should use two or more Intel Neural Compute Stick's. The number of frames per second is not high enough, however, given the number of parameters to be detected, this is a normal result, especially head pose estimation is a very difficult task for the Inference Engine. If we exclude some specifications, then we can say that the Raspberry Pi 3 Model B and the Intel Neural Compute Stick 2 detect faces in real time.

Thus, we found that the use of such a hardware and software complex is well suited for our future mobile autonomous robot with video analytics.

References

[1] People's Daily Online Is facial recognition the future of smart payment in China? *The Telegraph*, https://www.telegraph.co.uk/peoples-daily-online/science/facial-recognition/, January 2019

[2] Integrate the Inference Engine New Request API with Your Application *Intel*, https://docs.openvinotoolkit.org/latest/_docsIE_DG_Integrate_with_customer_application_new_API.html

[3] Model Optimizer Developer Guide *Intel*, https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

[4] Raspberry Pi Face Recognition *Adrian Rosebrock*, https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/

[5] Intel Neural Compute Stick 2 *Intel*, https://software.intel.com/ru-ru/neural-compute-stick

[6] Install OpenVINO toolkit for Raspbian OS *Intel*, https://docs.openvinotoolkit.org/latest/_docs_install_guides_installing_openvino_raspbian.html

[7] Intel Distribution of OpenVINO Toolkit *Intel*, https://software.intel.com/en-us/openvino-toolkit

[8] Deep Learning For Computer Vision *Intel*, https://software.intel.com/en-us/openvino-toolkit/deep-learning-cv

[9] Intel CV School *Intel*, https://delta-course.org/Intel-CV-School

[10] E. S. Mishchenkova. *Vestnik of the Volgograd State University. Series 9: Research of young scientists.* Federal state budgetary educational institution of higher professional education "Volgograd State University", 2013.

[11] Adrian Rosebrock *Deep Learning for Computer Vision with Python Practitioner Bundle 1st Edition (1.2.1)*. PyImageSearch, 2017.

[12] Things to Know: Key Recommendations for Deep Learning (Part 2) Stanislav Petrenko *DataReview*, http://datareview.info/article/eto-nuzhno-znat-klyuchevyie-rekomendatsii-po-glubokomu-obucheniyu-chast-2/

[13] AlexNet *Wikipedia*, https://en.wikipedia.org/wiki/AlexNet

[14] Stochastic gradient descent *Wikipedia*, https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[15] Challenges in Representation Learning: Facial Expression Recognition Challenge *Kaggle*, https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[16] Unfiltered faces for gender and age classification *The OUI-Adience Face Image Project*, https://talhassner.github.io/home/projects/Adience/Adience-data.html#agegender