*Educational Technologies*
*Образователни технологии*

# THE PROBLEM OF IMAGES' CLASSIFICATION: NEURAL NETWORKS

**Assoc. Prof. Larisa Zelenina, Assoc. Prof. Liudmila Khaimina, Evgenii Khaimin, D. Khripunov, Assoc. Prof. Inga Zashikhina**
*Northern (Arctic) Federal University named after M.V. Lomonosova (Russia)*

**Abstract.** The article discusses the employment of an artificial neural network for the problem of image classification. Based on the compiled dataset, a convolutional neural network model was implemented and trained. An application for images classification was created. The practical value of the developed application allows an efficient use of a smartphone camera's storage facilities. The article presents a detailed methodology of the application's development.

*Keywords:* image classification; convolutional neural network; Python programming language libraries; graphical user interface (GUI)

## I. Introduction

Nowadays, many users have to deal with images and photos. The quality of cameras in modern smartphones is often as high as that of professional cameras, where the usage of lenses allows perfect images. The number of captured images in the smartphone memory becomes uncontrollable and it is limited only by the smartphone's memory capacity. At the same time, the cooperation with thematic online stores, namely, photobanks – image banks that act as intermediaries between the authors of images and their buyers, is also intensifying. In this regard, the task of classifying all stored images is becoming more and more urgent every day. Sorting images by their content on photobank sites can be performed by an application for their classification. With the help of a neural network, this application will be able to identify the depicted object, and then to sort them according to the recognized classes. This will help to classify quickly all images in the database of the photobank.

At the moment, there are many free software products with similar functionality. Images' sorting is fulfilled with the help of tags, when the user adds each photo manually. In this article, we are presenting the application, which classifies images using a neural network. The suggested application is not overloaded with unnecessary functionality and has a simplified and intuitive user interface.

***Image classification problem. Convolutional neural network***

The classification task in machine learning is usually referred to as supervised learning, but unsupervised learning also exists. In this case, the training sample does not have the specified classes, the classification is based on the similarity of objects. This type of problem is often referred to as clustering and taxonomy problems, and classes are called clusters or taxa.

The statement of the classification problem in mathematical language is as follows: there are two sets X and Y, where X is a set of descriptions of objects, and Y is a finite set of classes. There is an unknown mapping $y^*: X \to Y$, the values of which are known only on the objects of the training set $X^m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. It is necessary to construct such an algorithm $a: X \to Y$ capable of classifying an arbitrary object $x \in X$ [Classification problem].

There are two main types of classification – binary and multiple. In the problems of binary classification, the dependent variable takes only two values – yes/no, 0/1, or alike). For example, the task of predicting the buyer of a certain product belongs to this type. In the problems of multiple classification, the dependent variable takes values from a set of classes. For instance, this type includes the problem of predicting the choice of the buyer, i.e. which class of goods he will prefer to purchase. Evaluation of classification accuracy can be carried out using cross-validation on test data called cross-validation set. In this case, the accuracy on the training set is compared with the accuracy on the test set.

**Application for Images' Classification Development**

In this study, the solution to the image classification problem builds of convolutional neural networks, which have a fairly high ability to recognize patterns in images. There are four main layers used in convolutional neural network modeling: the convolution layer, the activation layer, the subsampling layer, and the fully connected layer. The job of a convolutional neural network is to move from specific features of an image to more abstract details. The network independently filters unimportant image details and highlights its essential features. Training a neural network is reduced to minimizing the loss function by adjusting the weights of synaptic connections between neurons. To train the developed neural network, the backpropagation algorithm or the generalized delta rule is used.

***Neural network design for image classification***

The following libraries of the Python programming language were used to develop the convolutional neural network and application [Python 3.6.5 documentation]:

– *TensorFlow* is an open-source machine learning software library designed specifically for building and training neural networks. Calculations performed by

the *TensorFlow* library are represented as a data flow through a fixed state graph [TensorFlow];

*Numpy* – a library that supports working with multidimensional arrays and high-level mathematical functions for working with them [Numpy Documentation];

*PyQt5* – a library for creating a user interface, providing *Qt Designer* tools to facilitate interface development [PyQt5 Reference Guide];

*Fastai* – Python deep learning library providing high level API;

*Keras* is a neural network library, which is an add-on over the TensorFlow and Theano libraries to facilitate the development of models for machine learning [Keras Documentation].

The modules were also used:

*shutil* – contains a set of functions for processing and working with files and folders, allowing to copy, move and delete them;

*os* – provides functions for working with the operating system, such as executing system commands, gaining access to directories;

*QML* (Qt Meta Language) is a Javascript-based declarative programming language that serves to markup and prototype an interface.

*Google Colab*, a free cloud service based on *Jupyter Notebook*, was chosen to write and edit the source code of the neural network being developed. It contains everything you need for machine learning, as well as free access to fast GPUs and TPUs.

*Preparing a dataset*

To train a neural network, a prepared dataset is required, containing a large number of images of those classes that the network must learn to distinguish. After examining the main sections on the sites for the sale of images, we concluded that the most popular objects of photographers are people, architecture, nature, animals, food, cars. Accordingly, we have developed a dataset of images.

Images for the set were determined from the "Pictures" section of the Google search engine, which contains images of all sorts of topics and directions. To automate the download of a large amount of data required to compile a dataset of images, Javascript was used that collects download links for all images in a *csv* file.

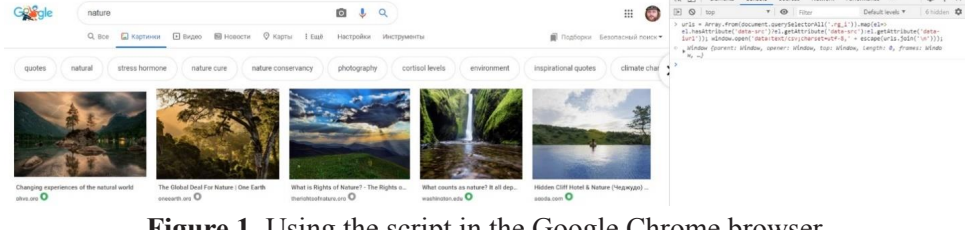


**Figure 1**. Using the script in the Google Chrome browser

As you can see in Figure 1, the script is written to the console of the Google Chrome browser and it is executed at the moment when images of the necessary content are loaded in the "Pictures" section of the Google search engine. Then, according to requests for nature, architecture, car, people, food, animals, 6 *csv* files, containing links to download images, were compiled with the use of the script.

| | A |
|---|---|
| 22 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcTh_YgJoy480ZP8lTc4pW1nZCKbeg4W8nMx3zAcyxUIIlE1w5O&usqp=CAU |
| 23 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRJANOtMIkp6nd3ppylWssXw4iGNuruD8qFTwcu7BMl5AtbDNPk&usqp=CAU |
| 24 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcReBDsiORSua1wXWh6J300pzxPkZ1VFJ_TTq0TKvJncaq3iEV9o&usqp=CAU |
| 25 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcSit20FePNfqF7cML7BUD2ovDFHqAlt9Jsh_g3Weydza_LmAC6Il&usqp=CAU |
| 26 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcT__srJhKycBCNbXHr2NI7jUVEj0M8SKX_xsvyFK0kdJJcebNw4&usqp=CAU |
| 27 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcStV0_r9VZJMqXsx5Z196UYoqt7wK7emjY5bGXKD8cLEO4ZIB-I&usqp=CAU |
| 28 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQlnL_POOhN9-DdlDx_-2NxGVRzNco1xDSiEvF4HZrARAPT-L32&usqp=CAU |
| 29 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRRII79T3M_IovjTvUhECzHtRYBI5EO-JtbqUpwWkYkKl2AtocX&usqp=CAU |
| 30 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcTm02ph4wbuPKlkgtZuPWVVhLiOzFO4XH6AxsqP7xldhlY6edEe&usqp=CAU |
| 31 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRdrUWkw1-zRNqNb5gMdqF2CpDh_gAjc36oIyWzujH8w1h71k_w&usqp=CAU |
| 32 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcSAVLdvWCTeQzitQCpSH-pplecyqsUAnS5uMuwByIAKmLFutQpn&usqp=CAU |
| 33 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQDqjQpPqwZ2q2usT46aXFXeGg08kyzvDg5DVw0AtPLatCNX6b7&usqp=CAU |
| 34 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRlDa3c_xIZteZuOS4bNUgIxSfqOXZ15mO1DnUc7lJ0i61KaEQh&usqp=CAU |
| 35 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcR5OjKlaVoI_HRLpWK7U-ugNivmT_HVCE3AsAeI4Et1qNZINstA&usqp=CAU |
| 36 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcTEUewTqyuFARtqkXQgpdI_iYLxm5FafMnNJdiMCPEeJYNW4VjG&usqp=CAU |
| 37 | https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRHJTCtyNkkucqV9xYfUedHru4w2dg2NJ0rEwmfkafxt0gpfvhU&usqp=CAU |

**Figure 2**. Fragment of the outcome for the script use on the request *animals*

To download all images from the obtained csv files, the *download_images* method of the *fastai* library of the Python programming language was used. The obtained photographs were sent directly to folders with the names corresponding to the classes. As a result, 6 folders were obtained, each of which containing images of its own class (animals, architecture, car, people, nature, food).


**Figure 3**. Examples of the images corresponding to the classes

In order for the neural network to learn best, we can first clear the resulting dataset from poorly corresponding images. To simplify further work with images, they were renamed according to the pattern: ""class name" + "picture number" + .jpg.

The final stage of preparing the dataset was to create a training, validation and test sampling. For this, we made folders with the corresponding names *train, val* and *test*. Each of them contained 6 folders corresponding to a certain class. Twenty percent of the dataset was allocated for the validation sample, ten percent for the test sample. The samples were compiled by copying images from the folders of the main set to the *train, val* and *test* folders in a specified percentage. At the same time,

the number of files in the corresponding folder was counted for each class. Then we determined the file which could be used to start copying. The obtained values, along with the paths to the source folder of the class and its folder from the selection, were passed to the input for the *copy images* function. This function is used to copy an image from the resulting range using the *copy2* method of the *shutil* library.

*Implementing and training a convolutional neural network*

To expand the compiled dataset and improve the quality of training of the developed neural network, data augmentation can be performed. Data augmentation is a technique for obtaining additional data from the neural network available for training. To attain this, the *ImageDataGenerator* method from the *Keras* library was used, which allows generating a training sample.

For augmentation of the training sample data, the following parameters are set in the *ImageDataGenerator*:

– **rescale**: data scaling factor, divides the values of all image pixels by 255;
– **rotation_range:** values for randomly rotating the image by a given number of degrees;
– **width_shift_range**: sets a random shift in width;
– **height_shift_range**: sets a random shift in height;
– **zoom_range**: sets the range for random image scaling;
– **shear_range**: sets the range of shift of the image pixels;
– **horizontal_flip**: reverses the image horizontally; from left to right;
– **fill_mode**: when 'nearest' fills points outside the input images framework according to the pattern: **aaaaaaaa | abcd | ddddddd**.

When training a neural network, it is not always possible to achieve the desired accuracy, since it requires a very fine selection of hyperparameters for each type of task. To simplify the process of matching hyperparameters, the *Keras Tuner* optimization library can be used. This library allows to optimize a set of network hyperparameters in an automatic mode. At the end of the tests, it provides a report on the most effective obtained models with the selected parameters.

In the proposed optimization function of a convolutional neural network using *Keras Tuner*, the hyperparameters are|:

– the number of filters for the first convolutional layer of the network (in the range from 32 to 128 with a step of 32)
– the total number of convolutional layers (in the range from 1 to 8, since the problem of image classification does not need to use a deeper network architecture)
– number of data filters of convolutional layers (in the range from 64 to 256 with a step of 32)
– the number of neurons in a fully connected layer (in the range from 128 to 512 with a step of 64).

We received a report on the top 10 models using the *results_summary* method on completion of the optimizer's work. Figure 4 shows an example of the report for one model, which shows the average accuracy during its training and all the fitted hyperparameters of the network.


**Figure 4**. Example of a *Keras Tuner* optimizer report

We modelled a convolutional neural network with the hyperparameters from the optimizer. For this, we employed a sequential type model, which makes it easy to add new layers with the add method.

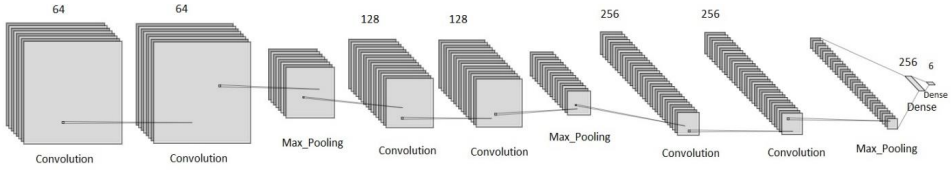


**Figure 5.** Model of a convolutional neural network

The first pair of Conv2D layers are convolutional layers containing 64 filters, a 3 x 3 filter matrix, a ReLU activation function, a form for the input image input_shape = (150, 150, 3), where 150 x 150 is the dimension of the image, the number 3 is the presence of three *rgb* channels. The next layer, MaxPooling2D, is a

subsampling (maximum pooling) layer, with the compaction of a 2 x 2 pixels group to one pixel group. Then comes *Dropout*. This is a process of selectively shutting down neurons during training. It is also a way to avoid overfitting. When some neurons are selectively disabled during training, other neurons are activated much more actively, which has a positive effect on the overall quality of the training. The network becomes more resilient and can rely on all of its neurons in task completion when this training method is applied. The parentheses indicate the probability of excluding each of the neurons at any training iteration.

The next blocks of the convolutional neural network model contain similar layers, differing only in convolutional layers containing 128 filters, and then 256. Between the convolutional layers and the fully connected layer there is *Flatten*. That is an alignment layer that converts the received data into a vector. This is followed by the fully connected Dense layer (256 neurons, *relu* activation function), the Dropout layer, the final Dense layer, consisting of the number of neurons equal to the number of classes. Using the *softmax* activation function, it converges the resulting amount to one in order to interpret the result as a series of possible outcomes.

Three parameters were used to compile the constructed model: *the adam* optimizer, *the categorical_crossentropy* loss function used in classification problems, and *the accuracy* metric used to display the accuracy score at validation during training. The optimizer controlled the learning rate, that is, the rate at which the optimal weights were calculated for the model. At a low speed, the weights are determined more accurately, but it takes a lot of time.

The compiled network model was trained over 200 epochs. Starting from the 180th epoch of learning, the accuracy on the test sample began to decline slightly. Further training of the neural network on this dataset could lead to overfitting. Then the trained convolutional neural network model was saved to a separate file using the *save* method of the *Keras* library for further use in the application.

Figure 6 shows an example of testing a neural network on arbitrary photographs.

The loaded image is first scaled to 150 x 150 pixels, converted to an array, and normalized so that all pixel intensities are in the range from 0 to 1. Then, based on the *predict* method, the network recognizes the class of the transformed image.

*Implementing an application based on a trained neural network*

To implement the application for classifying the content of an online store, we need a program that sorts all images in the specified directory into their corresponding folder classes. This will require a trained neural network as described above, as well as a graphical interface that simplifies user interaction with the application.

In order to simplify the creation of a graphical interface and save time, we chose to use a cross-platform environment for GUI development, namely *Qt Designer*.

```
[63] img_path = '/content/chmaska.jpg'
     img = image.load_img(img_path, target_size=(150, 150))
     plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7fd4836b7828>
```


```
[64] x = image.img_to_array(img)
     x /= 255
     x = np.expand_dims(x, axis=0)
```

```
[71] prediction = loaded_model.predict(x)
     prediction = np.argmax(prediction)
     print(classes[prediction])
```

```
people
```

**Figure 6**. Example of testing a neural network

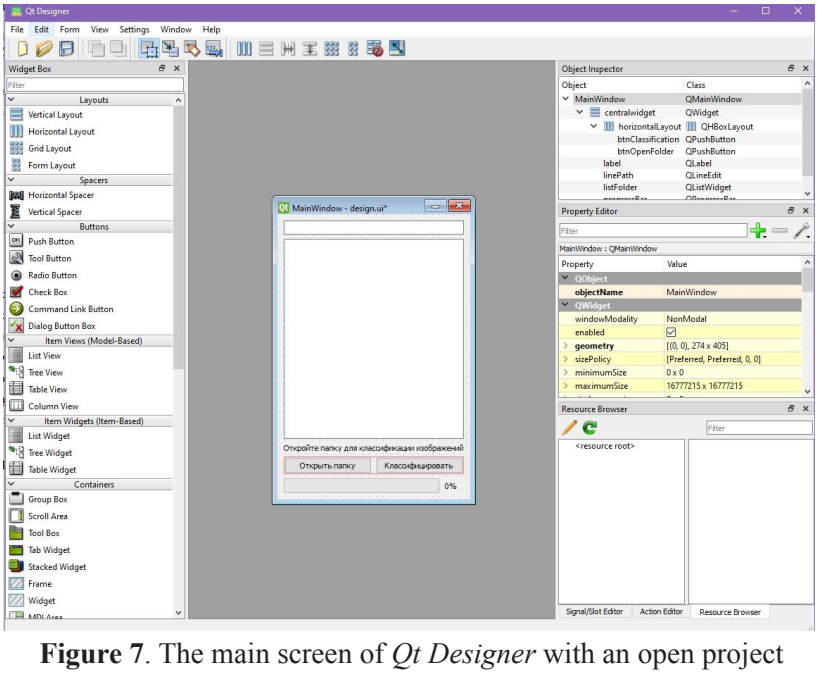


**Figure 7**. The main screen of *Qt Designer* with an open project

All the used form elements and their hierarchy are displayed in the *Object Inspector*, located in the upper right corner of the screen. In total, 6 objects were used in the project:

– *btnClassification*: by pressing this button classification and sorting of images occurs;

– *btnOpenFolder*: by pressing this button a window with a choice of a directory appears;

– *labe*l: a text string notifying the current status of the application;

– *linePath*: the top window that displays the selected directory;

– *listFolder*: a large window that displays all found images in the selected directory;

– *processBar*: a bar at the very bottom of the application, indicating the percentage of the sorted images.

For the objects of the graphical interface not to move out when the window is stretched, we need layouts. Layouts are containers for widgets that hold them in a certain position relative to other elements. Both app buttons use a horizontal layout, while all other widgets use vertical layout.

The resulting interface design must be saved in a separate *ui* extension file. To import and use this file in Python code, you need to convert it to a *py* file. The command "pyuic5 design.ui -o design.py" from the command line, converting the *ui* file to a Python file, will help with this.

The resulting converted design file is imported into the main file containing the application code. Next, the main functionality of the application is implemented: a function, that is activated after clicking, is assigned to each button. For example, the browse_folder function is linked to the "Open folder" button, the *classification* function was linked to the "Classify" button.

The model of the developed and trained convolutional neural network is loaded into the application from a file with the h5 extension using the *load_model* method.

When the application is launched, the user is presented with the main and only screen with a minimalistic and intuitive interface. The screen contains two buttons for interacting with the application ("Open folder" and "Classify"), a status line displaying the start caption "Open a folder for image classification" and a progress bar.

When the user clicks the "Open folder" button, a window prompting to select a directory for image search appears.

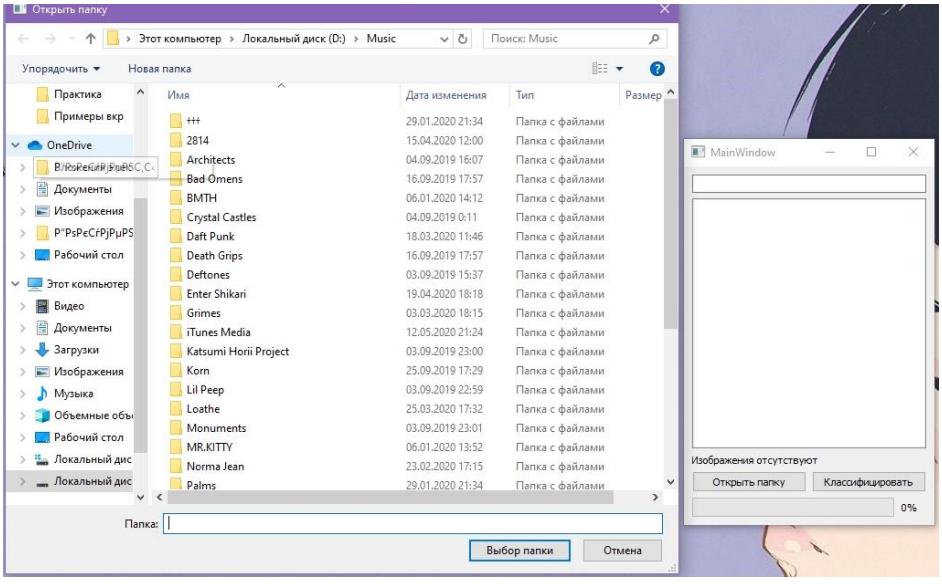


**Figure 8**. Demonstration of the “Open folder” button operation

If the user clicks the “Classify” button accidentally without selecting a directory, the message “Folder is not open” appears in the status bar. If there are no images in the selected directory and the user clicks the button again, the message “There are no images” appears. When the user selects the required directory, its path will be displayed in the upper part of the window. All the detected images will appear as a list in the central part of the window, the number of all found images will be displayed in the status line. When the user clicks on the “Classify” button, the application starts working and the bottom progress bar shows the percentage of sorted images. The application is shown in Figure 9.

At the end of the work, the application notifies the user with the message “Done!” in the status bar and clears the window displaying all found images.

The application has been successfully tested on a set of 392 images.

**Conclusion**

In this study, we have analyzed the existing photobanks in order to identify the most popular image classes searched by the users. To optimize the users’ search we have developed an application that is friendly in terms of the smartphone camera’s memory. We have created a dataset and designed a convolutional neural network. The latter has been trained to solve the image classification problem. Due to the ability of a trained neu-

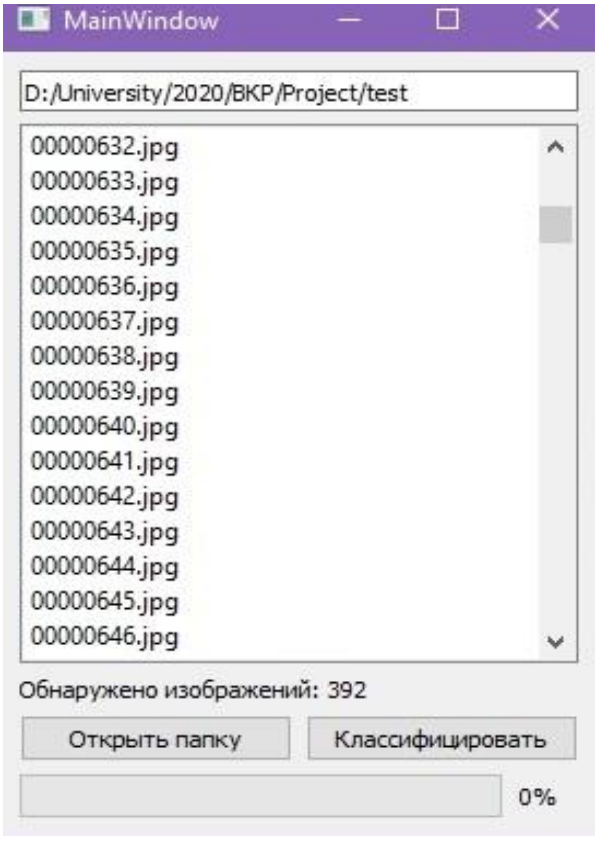


**Figure 9**. Application operation during image classification

ral network created for the application, images' classification occurs. The process of images' classification allows for the efficient use of a smartphone camera's capacity. The application has been tested and received positive feedback.

**REFERENCES**

*Classification problem*. Loginom. Accessed: https://wiki.loginom.ru/articles/classification-problem.html. Last visited on 15.11.20.

*Keras Documentation.* Accessed: https://keras.io/. Last visited on 15.11.20.

*Numpy Documentation.* URL: https://docs.scipy.org/doc/. Accessed online 15.11.20.

*PyQt5 Reference Guide*. Accessed: http://pyqt.sourceforge.net/Docs/PyQt5/. Last visited on 15.11.20.

*Python 3.6.5 documentation*. Accessed: https://docs.python.org/3/index.html. Last visited on *15.11.20*.

*TensorFlow*. Accessed: https://www.tensorflow.org/. Last visited on 15.11.20.

✉ **Dr. Larisa I. Zelenina, Assoc. Prof.**
ORCID iD: 0000-0002-0155-3139

**Mr. D.S. Khripunov**
Department of Applied Mathematics and High Performance Computing
Higher School of Information Technologies and Automated Systems
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: l.zelenina@narfu.ru
E-mail: hripunov.d@edu.narfu.ru

**Dr. Liudmila E. Khaimina, Assoc. Prof.**
ORCID iD: 0000-0003-4552-0440

**Evgenii S. Khaimin, Senior Lecturer**
ORCID iD: 0000-0003-0523-3623
Department of Applied Informatics and Information Security
Higher school of Information Technologies and Automated Systems
Northern Arctic Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: l.khaimina@narfu.ru
E-mail: e.khaymin@narfu.ru

**Dr. Inga M. Zashikhina, Assoc. Prof.**
ORCID iD: 0000-0002-8217-2302
Department of Philosophy and Sociology
Higher School of Social Sciences, Humanities and International Communication
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: i.zashikhina@narfu.ru