UDC 004.85

Using Docker to deploy computing software

**Tatiana S. Demidova, Anton A. Sobolev,
Anastasia V. Demidova, Migran N. Gevorkyan**

*Department of Applied Probability and Informatics
Peoples' Friendship University of Russia
Miklukho-Maklaya str. 6, Moscow, 117198, Russia*

Email: dem_tatiana@mail.ru, raven357be@gmail.com, demidova_av@rudn.university,
gevorkyan_mn@rudn.university

There are many ways to facilitate the creation of large-scale projects. One of the most commonly used methods is to create virtual machines that contain the program environment. However, software has recently been created to make this process even easier. One example is Docker, a software for automating the deployment and management of applications in an operating system-level virtualization environment.

This paper discusses the Docker software, its features and benefits, which allows you to create images that contain the program and all the necessary components for its operation. The purpose of this work is to study the capabilities of Docker. And also, the creation of a container containing a software implementation of the neural network for recognition of various handwritten characters. Training and test data is a database of handwritten numbers and letters "MNIST" and "EMNIST". To teach the neural network to recognize numbers, a training set containing 60 thousand copies was used, and the test set includes 10 thousand copies. For letters – the training set contains 88800 copies, and the test set includes 14800 copies. The project was created on the basis of the image tensorflow downloaded from public Docker-registry Docker Hub. The program is written in Python3, using the service for interactive computing Jupyter Notebook.

In: K. E. Samouylov, L. A. Sevastianov, D. S. Kulyabov (eds.): Selected Papers of the IX Conference "Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems", Moscow, Russia, 19-Apr-2019, published at http://ceur-ws.org

1. Docker — container virtualisation system

In this paper we use Docker to automate the deployment and management of applications. Docker is collection of programs used to run processes in an isolated environment based on special images [1]. It allows one to create containers that contain the application and its entire environment, and provides an environment for their use.

The main components of Docker are images, registry and containers.

Docker image is a read-only template. For example, an image might contain an operating system with installed applications. Images are used to create containers. Docker makes it easy to create a new image and update existing ones or download ready-only images.

Docker-registry stores images. There are public and private registries. One can download images from registry or upload images to registry. There is free public Docker registry called Docker Hub. It is accessible for all users.

Containers are similar to directories. They contain everything to run the application. Each container is created from an image, which forms an isolated and a secure platform for the application.

Docker container is created by following command

```
$ docker run <attributes> <image> <command>
```

One of the advantages of Docker is the ability to create a container using Dockerfile and the `docker build` command. Dockerfile contains the base image. This image is used to build the desired container by applying commands, specified in Dockefile.

To demonstrate the capabilities of Docker, a project was created using a neural network for handwriting recognition.

Machine learning libraries

Artificial neural network is a mathematical model built on the principle of biological neural networks of nerve cells of a living organism. A neural network is constructed in such a way that its nodes work like neurons in the human brain. The node collects information, processes it, and passes it to the next node [3, 11, 18].

For the implementation of neural networks, there is a huge amount of software. The main differences between them are their functionality. While some frameworks are used as shells to extend functionality and facilitate the writing of neural networks, others are full-fledged languages and can define neural networks of any level of complexity.

TensorFlow is library designed for machine learning. This framework is created by Google. It uses Python languages as back-end, but core parts are written in C++ [**?**,13,14]. We use TensorFlow to train and build a neural network for classifying and finding images that are close to human perception. All calculations in this environment are performed using stream data graphs, where nodes represent different mathematical operations and graph branches represent arrays of data.

The Keras library [12] is an add-in for TensorFlow to create high-level neural networks, written in the Python programming language [15]. The main advantage of this library it is easy usage when working with deep learning networks. Keras can be easily extended with new modules in the form of classes and functions. This environment includes the implementation of optimizers, layers, functions, and other tools for working with images and text.

The Scikit-learn [7, 8] library, written in Python, has many algorithms for training neural networks with and without a teacher. Developers pay much attention to the usability of this environment and optimization issues to improve the speed of its operation. Scikit-learn includes various classification, regression and clustering algorithms. It is designed for interaction with numerical and scientific Python libraries such as NumPy and SciPy.

2. Neural network architecture

The convolutional neural network [6, 9, 17] was chosen as the topology of the neural network to solve the problem of handwriting recognition. Today convolutional neural networks are considered to be the best for solving image recognition problems. The architecture of the neural network, which is based on the convolution operation, was first developed in the late 1990s by Lekun et al.

Convolutional neural network (CNN) consists of the following types of layers: convolutional layers, subsampling (or pooling) layers and perceptron layers. The first two types of layers, alternating with each other, form the input feature vector for the multilayer perceptron.

In convolutional neural network a convolution operation uses a limited matrix of weights of small size, which moves through the processed layer, forming after each shift activation signal for the next layer of the neuron with a similar position.

This weight matrix is called the convolution kernel. In a convolutional neural network, sets of weights encoding image elements are formed independently by training the network. After a convolutional layer comes the pooling layer. It also has maps, the number of which coincides with the previous layer. The purpose of this layer is to reduce the dimension of the maps. In the previous convolution operation, signs were identified that do not require such detail. Filtering already unnecessary parts helps the network avoid overtraining. After several repetitions of convolutional layers and pooling layers, a layer of the usual multilayer perceptron follows. The output layer is connected to all neurons of the previous layer and the number of neurons corresponds to the number of recognized classes. In the case of binary classification, a single neuron and hyperbolic tangent can be used as an activation function. Then, the output of a neuron with a value of 1 means belonging to a class, and the output of a neuron with a value of -1 means not belonging to a class.

The following architecture was used for the convolution network to recognize digits (fig. 1).

This network consists of 6 layers:

1. Convolutional layer with 75 feature maps, convolution kernel size: 5x5.
2. Pooling layer (MaxPooling) with 2x2 poolsize.
3. Another convolution layer with 100 feature maps, convolution kernel size: 5x5.
4. Second pooling layer with 2x2 poolsize.
5. Fully connected layer with 500 neurons
6. Fully connected output layer with 10 neuron, which correspond to the classes of handwritten digits from 0 to 9.

The activation function in hidden layers is ReLU, and the output layer is softmax.

As training and test data we use "MNIST" database of handwritten numbers. To train the neural network to recognize numbers we use a training set containing 60 thousand copies and the test set containing 10 thousand copies.

We use Matplotlib library to create an image of the symbols from the database 2.

3. Neural network classification quality assessment

One of the concepts for describing metrics in terms of classification errors is the error matrix. It is used if there are two classes and an algorithm predicting that each object belongs to one of the classes, then the classification error matrix will look like this [10, 16]:

One of the simplest metrics is accuracy — the proportion of true results (both positive and true negative) among the total number of cases considered, i.e. the probability that the class will be predicted correctly.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 75)	1950
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 75)	0
dropout_1 (Dropout)	(None, 12, 12, 75)	0
conv2d_2 (Conv2D)	(None, 8, 8, 100)	187600
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 100)	0
dropout_2 (Dropout)	(None, 4, 4, 100)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_1 (Dense)	(None, 500)	800500
dropout_3 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 10)	5010

Total params: 995,060
Trainable params: 995,060
Non-trainable params: 0

Figure 1. Neural network architecture

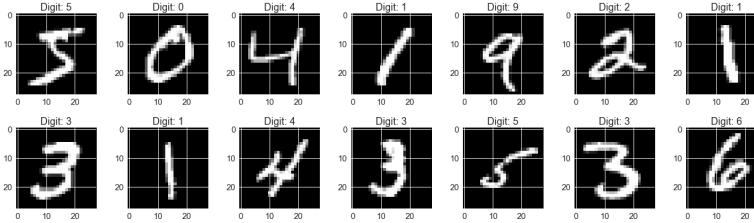

Figure 2. Symbols from MNIST database

Table 1

Error classification matrix

	Condition mark – 1	Condition mark – 0
Algorithm response – 1	True Positive (TP)	False Positive (FP)
Algorithm response – 0	False Negative (FN)	True Negative (TN)

To assess the quality of the algorithm on each of the classes precision and recall metrics are used:

$$precision = \frac{TP}{TP + FP}, \qquad recall = \frac{TP}{TP + FN}.$$

The accuracy of the classification of positive results (precision) is the proportion of positive results that are correctly identified by the classifier among the total number of considered cases.

Recall, also known as sensitivity, reflects the proportion of positive results that are correctly identified by the classifier.

Specificity — reflects the proportion of negative results that are correctly identified by the classifier.

To associate a precision with the fullness F-measure are used. F-measure is a harmonic mean of precision and completeness:

$$F_{measure} = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

One way to evaluate the model is AUC-ROC — square under the curve of error in coordinates True Positive Rate (TPR) and False Positive Rate (FPR):

$$TPR = \frac{TP}{TP + FN}, \qquad FPR = \frac{FP}{FP + TN}.$$

4. Software implementation of neural network

The neural network is implemented in Python 3, using TensorFlow and Keras modules, as well as Jupyter Notebook shell. These modules have built-in tools for building and fine-tuning neural networks. The project was created on the basis of the image tensorflow [2]. The container is created by the command

```
$ docker run -p 8888:8888 tensorflow/tensorflow:latest-py3-jupyter.
```

The source code for building a neural network using the Keras library is:

```
model = Sequential()
model.add(Conv2D(75, kernel_size=(5, 5),activation='relu',
          input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(100, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss="categorical_crossentropy", optimizer="adam",
              metrics=["accuracy"])
```

When creating a neural network using the Keras library, we add layers sequentially using the `model.add` and then compile it using the `model.compile`.

The most important layers are the basic layers, such as `dense`, `activation`, `dropout`, `flatten`, `input`, etc. Dense layer is fully connected layer, `dropout` is used to prevent overfitting, `flatten` layer is used to convert two-dimensional or three-dimensional data into one-dimensional, the input layer used to input data.

For this neural network the error (confusion) matrix 3 was calculated and graphs of accuracy and error dependence were plotted depending on the number of learning epochs 4 and 5.

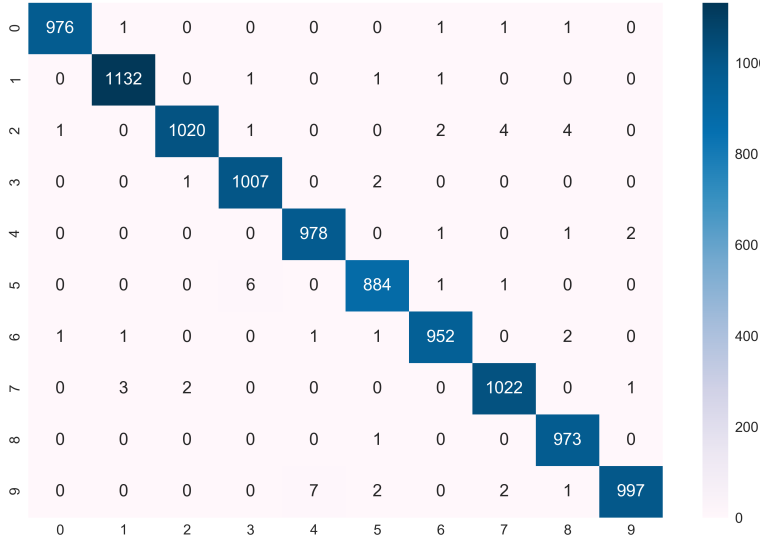

Figure 3. Confusion matrix

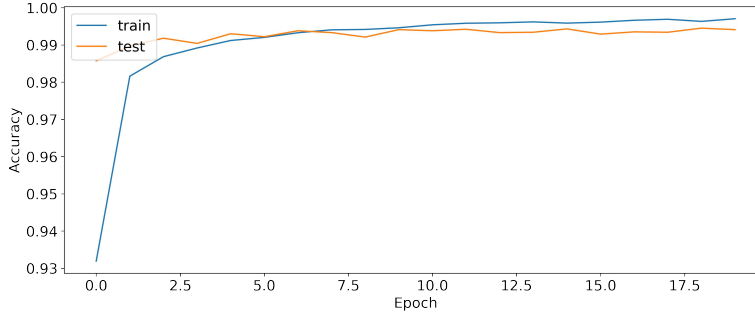

Figure 4. Model Accuracy

The neural network for handwriting recognition based on MNIST, showed an average efficiency of about 99.24%. The average training time of one era is about 160 seconds

Metrics were also calculated for each of the classes:

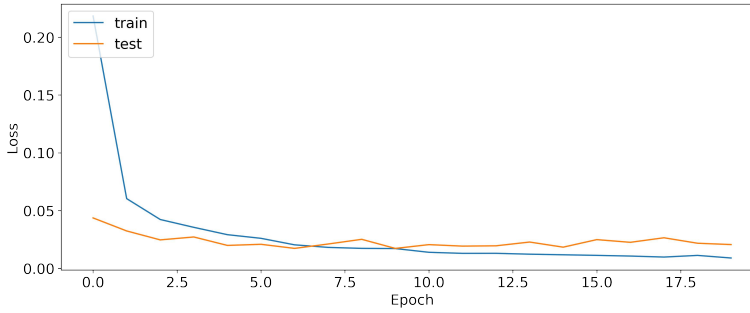

Figure 5. Model Loss

	precision	recall	f1-score
0	0.99	0.99	0.99
1	1.00	0.99	0.99
2	0.99	1.00	0.99
3	0.99	0.99	0.99
4	0.99	0.98	0.99
5	0.97	0.99	0.98
6	0.99	0.98	0.99
7	0.98	0.99	0.98
8	0.99	0.98	0.98
9	0.97	0.98	0.98

5. Implementation of the project using Docker

The following describes the process of downloading the created software package in Docker Hub. The first step is to create an image from the container that contains the program. We use the commit command to commit the changes to the new Docker image.

```
$ docker commit container_id repository/new_image_name
```

To show container's id we use command `docker ps -a`. As repository name we use user's Docker Hub login and assign a name to the image. This image is saved locally and to make sure that the new image is saved successfully, we use the command `docker images`.

Next, we upload our image to Docker Hub. To do this we enter the DockerHub account with command

```
$ docker login -u docker-registry-username
```

where `docker-registry-username` — Docker Hub user name. We enter the password and upload the image to DockerHub by the command

```
$ docker push docker-registry-username/docker-image-name
```

After successfully uploading the image, we can see it in the account toolbar (Fig. 6).

Thus, we can conclude that Keras library combined with Docker is a powerful toolset for the development software implementation of various machine learning methods. Keras is the most convenient library for writing neural networks, as it is easy to use

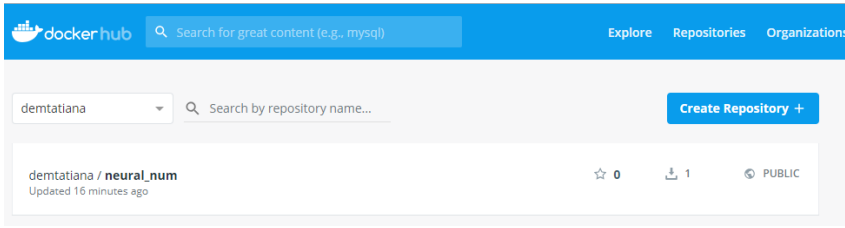

Figure 6. Our image on Docker Hub

and has a high speed of creating neural network models. The ease of use of Keras is due to the huge number of pre-installed functions designed to create different layers of the neural network. And Docker makes it easier to develop applications, because the installation of all the necessary libraries can be replaced by a single command — download the desired image and in addition, is a universal way to deliver the developed applications on local machines and run them in an isolated environment.

6. Conclusion

The paper considered Docker software that allows you to create images that contain the program and all the necessary components for its operation.

Created image based on tensorflow containing the Jupyter notebook implemented in Python convolutional neural network using bibltoteki Keras.

This image was uploaded to the public image repository of Docker Hub containers.

Acknowledgments

The publication has been prepared with the support of the “RUDN University Program 5-100”.

References

1. Docker: Enterprise Container Platform — URL: www.docker.com
2. Docker Hub — URL: https://hub.docker.com/
3. Tariq, Rashid. Make Your Own Neural Network — Spb.: “Alfa-kniga”, 2017. — 272 p.
4. The MNIST database of handwritten digits — URL: http://yann.lecun.com/exdb/mnist
5. The EMNIST Dataset — URL: http://www.nist.gov/itl/iad/image-group/emnist-dataset
6. LeCun, Y. Gradient-based learning applied to document recognition / Y. LeCun [et al.] // Proc. of the IEEE. — 1998. — Vol. 86, No 11. — P. 1–46.
7. Scikit-Learn: Machine Learning in Python. — URL: https://scikit-learn.org/stable/
8. Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.: O’Reilly Media – 2017. – 564 p. – ISBN-13: 978-1491962299
9. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. — Psychological review. — Vol. 65, No. 6. — 1958
10. Werbos P. J., Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

11. Hopfield J. J. Learning algorithms and probability distributions in feed-forward and feed-back networks. — 1987.
12. Keras: The Python Deep Learning library. — URL: https://keras.io/
13. Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.: O'Reilly Media – 2017. – 564 p.
14. Tensorflow: official website — URL: https://www.tensorflow.org/
15. Richert W., Coelho L. Building machine learning systems with Python. — Birmingham: Packt Publ. — 2013. — 290 p.
16. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: data mining, inference, and prediction. -2nd ed. -New York: Springer. — 2013. — 745 p.
17. Hornick K., Stinchcombe M., White H. Multilayer feedforward networks are universal approximators//Neural Networks. 1989. Vol. 2, no. 5. P. 359–366.
18. Ben J. A. Kroese and P. Patrick van der Smagt. An introduction to neural networks. — 1993.