UDC 004.932

# APPLICATION OF DEEP LEARNING AND COMPUTER VISION FRAMEWORKS FOR SOLVING VIDEO CONTEXT PREDICTION PROBLEM

*D. Voloshyn*

Authors describe an application for solving video context detection problem. Application architecture use state-of-the-art deap learning TensorFlow framework together with the computer vision library OpenCV in isolated agent environment. The experimental results are shown to demonstrate the effectiveness of developed product.

Key words: deep learning, tensorflow, computer vision, video context prediction.

У роботі описано програмний продукт для вирішення задачі прогнозування змісту відеопотоку. Архітектура розробки використовує інфраструктуру програмних рішень TensorFlow для глибинного навчання разом з бібліотекою комп'ютерного зору OpenCV в ізольованому агентному середовищі. Наведені експериментальні результати, які демонструють ефективність розробленого програмного забезпечення.

Ключові слова: глибинне структурне навчання, tensorflow, комп'ютерний зір, прогнозування змісту відеопотоку.

В работе описано программный продукт для решения задачи прогнозирования содержания видеопотока. Архитектура приложения использует инфраструктуру программных решений TensorFlow для глубинного обучения вместе с библиотекой компьютерного зрения OpenCV в изолированном агентном окружении. Приведены экспериментальные результаты которые демонстрируют эффективность программных разработок.

Ключевые слова: глубинное обучение, tensorflow, компьютерное зрение, прогнозирование содержания видеопотока.

## Introduction

Computer vision is a rapidly growing field of science aimed to analyse and understand images and video streams at a high level. Its objective is to determine the structure and type of the object in the front of a camera and use that understanding to control a computer system, or to provide people with information about the object. Generally, video context prediction problem is used to determine or predict the presence of object or entity, for example a person or car, in the video stream given some prior knowledge about video's nature. Application areas for computer-vision technology include military intelligence, video surveillance, movie production, Web search, medicine, augmented reality gaming, processing videos from unmanned aerial vehicles, and many more.

As a great theoretical research is done in this area, especially with a lot of works in deep learning, it still has a limited amount of practical application because of computational power and amount of training data required. This implies a strong restriction on the class of problems that could be approached. Therefore, there is a room for experiments with different technology stacks that will turn theory into practice. A true result can be achieved only by using results and tools from different fields.

In this paper we narrow the compare vision domain to predicting context of video input on frames where human faces are present. This restriction is imposed as predicting the context of the all frames on the video is very computationally expensive, and from the empirical experiments usually frames with humans give a fair estimate of the context of the whole video. The main goal of our application is to use the state of the art deep learning algorithms to predict the context of the video and classify video content based on that.

At the core of our application we use – TensorFlow framework. TensorFlow [1], developed by Google Brain team, is an interface for expressing machine learning algorithms and an implementation to execute them. The TensorFlow API is used to describe a dataflow-like model, and the implementation then maps those models onto the underlying machine hardware. Realised in 2016, Tensorflow express various types of parallelism by replicating the dataflow model across multiple machines and running them in parallel.

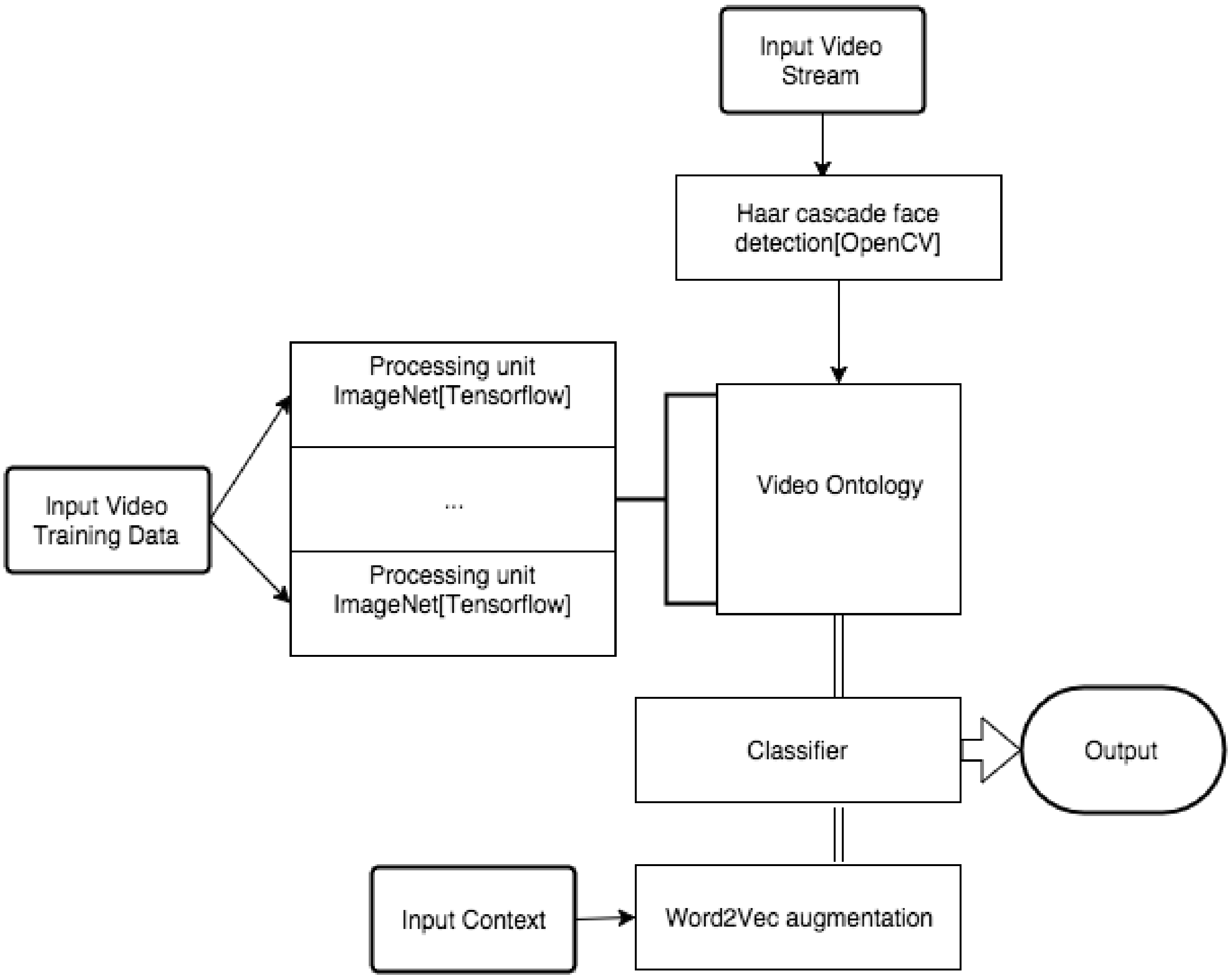
For processing video and using it as a training data we use OpenCV [2] – an open source library for video and image analysis, originally introduced in 2000 by Intel. Since then, a number of programmers have contributed to the most recent library developments. The latest version – OpenCV 3 was realised in the end of 2016 the library has >2500 optimised algorithms.

As mentioned above we filter frames on which we predict context by the presence a human face there. Face Detection is done using Haar feature-based cascade classifiers [3]. It has proven to be one of the quickest face detection algorithms. This is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. OpenCV library uses Haar classifiers with Adaboost algorithm for best feature detection.

To achieve architectural scalability and isolation we use prebuilt Docker [4] image. Developed software could be run on general purpose machines, or as hardware in specialised video processing units.

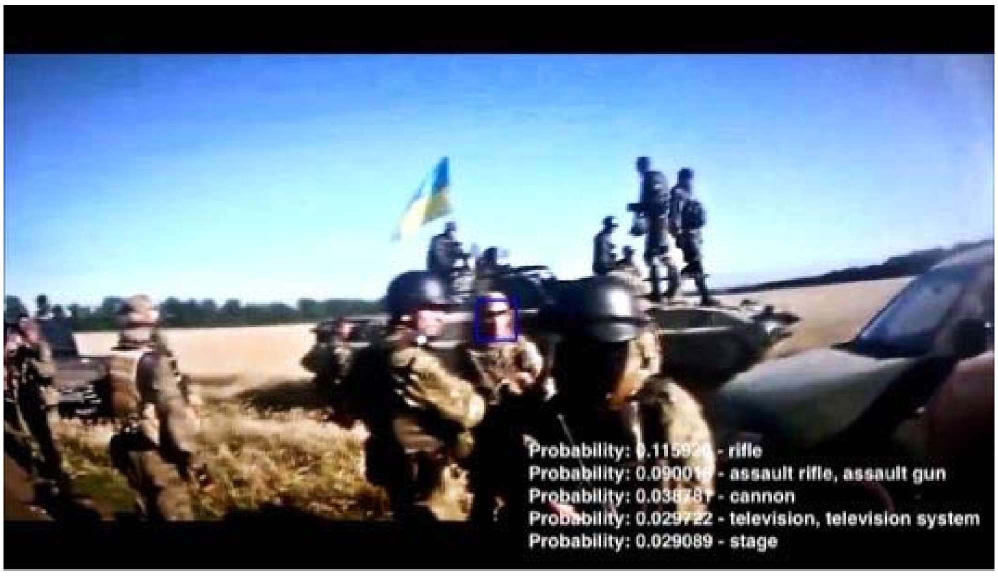
## Application architecture

Basic workflow of our application architecture consists of TensorFlow framework which operates on the graph of operations. Nodes/vertices in the graph represent operations (i. e., machine learning functions, mathematical operations), and the edges represent the multidimensional data arrays also known as tensors communicated between the nodes. Special edges, called control dependencies, can also exist in the graph and denote that the source node must finish executing before the destination node starts executing. Should be noticed that Tensorflow operates in a way when developer first design algorithm flow and computation architecture and after that framework itself runs the code and optimises the flow. Nodes are assigned to computational devices and execute asynchronously and in parallel once all the tensors on their incoming edges becomes available. Our application uses Python implementation of Tensorflow. Despite of the fact it is slower then speed-optimised C option, but we are able to use fundamental Python package for scientific computing Numpy and Scikit-learn – library for machine learning and data mining. Therefore, the whole project is a standalone Python application, that run in Docker container with all required packages and libraries. To ensure images/video output we connect to the running application using VNC technology. Initial video processing and extract, transform, load operations are done using native methods of the OpenCV library. At data processing step we also use few filters to diminish resolutions of input videos to prevent processing resolution higher then required.


Input Video Stream
Haar cascade face detection[OpenCV]
Processing unit ImageNet[Tensorflow]
...
Processing unit ImageNet[Tensorflow]
Input Video Training Data
Video Ontology
Classifier
Output
Input Context
Word2Vec augmentation

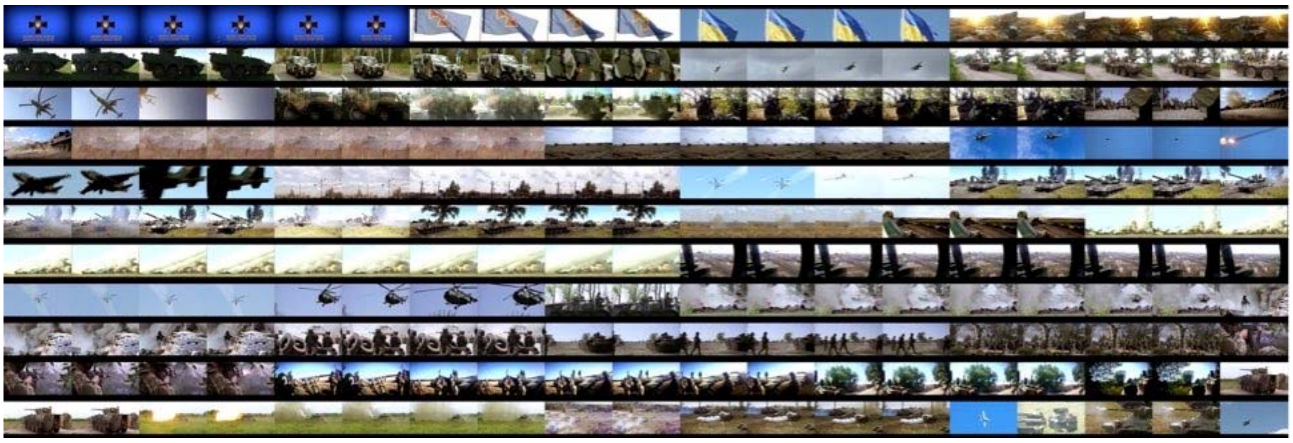
Picture 1. Application architecture

At a first step, we analyse the training set of the videos and label each frame with a set of labels, ordered by their probability. For labelling we use canonical ImageNet [5] deep neural network. Doing that, frame by frame, we obtain a sequence of labels with form "sentences" in natural language. We simply concatenate lists of the labels between frames. In the end, we get a corpus of labels in natural ordering, based on the ImageNet confidence about the video scene. We screen the input video stream with simple Haar filter to detect faces. On each frame where the faces are detected we run ImageNet network again and detect a set of objects' labels predicted to be present on the frame together with their probabilities.



```
Probability: 0.115920 - rifle
Probability: 0.090016 - assault rifle, assault gun
Probability: 0.038781 - cannon
Probability: 0.029722 - television, television system
Probability: 0.029089 - stage
```

Picture 2. Result of face detection and context classification filters



Picture 3. Video frames example - https://www.youtube.com/watch?v=Sr32W9IqPuo

Table 1. Video context prediction

| Frames 0 - 20 | Frames 20 - 30 | Frames 40 - 50 |
|---|---|---|
| 'tank', ' army tank', ' armored combat vehicle', ' armoured combat vehicle', 'cannon', 'torch', 'amphibian', ' amphibious vehicle', 'projectile', ' missile', '', 'half track', 'projectile', ' missile', 'lumbermill', ' sawmill', 'chain saw', ' chainsaw', 'amphibian', ' amphibious vehicle', 'missile', 'crane', 'missile' | 'jeep', ' landrover', 'tow truck', ' tow car', ' wrecker', 'garbage truck', ' dustcart', 'half track', 'ambulance', '', 'tank', ' army tank', ' armored combat vehicle', ' armoured combat vehicle', 'amphibian', ' amphibious vehicle', 'bulletproof vest', 'military uniform', 'ambulance', 'warplane', ' military plane', 'wing', 'kite', 'pelican', 'airliner', 'parachute', ' chute', 'projectile', ' missile' | 'amphibian', ' amphibious vehicle', 'fire engine', ' fire truck', 'oxcart', 'jeep', ' landrover', 'horse cart', ' horse-cart', '', 'tank', ' army tank', ' armored combat vehicle', ' armoured combat vehicle', 'half track', 'military uniform', 'golfcart', ' golf cart', 'cannon', 'steam locomotive', 'cannon', 'harvester', ' reaper', 'warplane', ' military plane', 'projectile', ' missile', 'missile', 'wing', 'aircraft carrier', ' carrier', ' flattop', ' attack aircraft carrier', 'bow', 'walking stick', ' walkingstick', ' stick insect', 'long-horned beetle', ' longicorn', ' longicorn beetle', 'matchstick', 'drumstick', 'wall clock', 'nail', 'folding chair' |

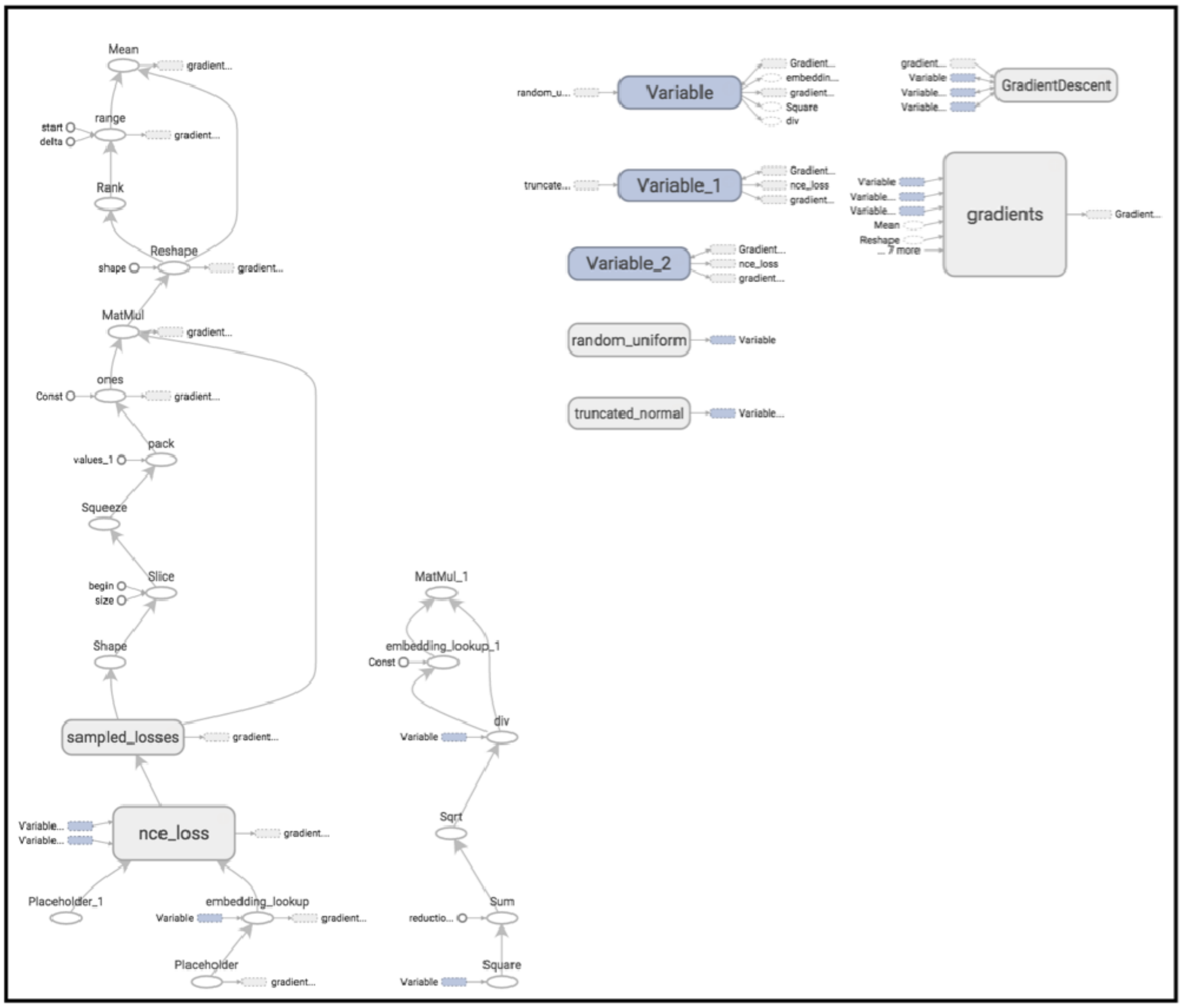
For each label we form a kernel of other labels close to through Word2Vec framework [6], using stochastic samples and pruning. After that, we get an augmented set of objects detected and predicted to be on the scene by expansion with k-nearest neighbours for each word. Lets *L* denote a set of labels for each frame and *F* will denote frames. Recurrent formula will be the following:

$$L_i = L_{i-1} \bigcup (F_{i-1} \setminus L_{i-1})$$

To measure a distance between two neighbouring labels we use a cosine distance:

where vectors *A* and *B* represent the vectors of labels in the Word2Vec space. If the similarity distance is greater then the threshold – we add a label to our set.

To predict the most probable objects not detected on the frame or that will appear on the next frames, we use TensorFlow graph model for the Word2Vec. The Word2Vec tool has two models: Skip-gram and continuous bag of words (CBOW). Given a window size of number of words around a word, the skip-gram model predicts the neighbouring words given the current word. In contrast, the CBOW model predicts the current word, given the neighbouring words in the window.



Picture 4. Word2Vec TensorFlow graph example

## Performance considerations

Video processing can be very expensive in terms of CPU power. In our case, computer-vision tasks often contain subtasks that can be run faster on special-purpose hardware architectures than on the CPU, while other subtasks are computed on the CPU. The GPU (graphics processing unit), for example, is an accelerator that is available not only on desktop computers, but also on mobile devices such as smart phones and tablets. Fortunately, TensorFlow has an built-in ability to use GPU units along with CPU which solves a problem of utilising machine resources in the most efficient way. In real world applications it is also much more efficient to distribute execution not only on GPU of a single machine but rather on a cluster of machines. To do that we use container virtualisation through Docker technology.

Container can be considered as a lightweight equivalent of a virtual machine. In the core of DockerLinux Containers there is an LXC, a user-space control package for Linux Containers. LXC uses kernel-level namespaces to isolate the container from the host machine. The user namespace separates the container's and the host's user database, thus ensuring that the container's root user does not have root privileges on the host. The process namespace is responsible for displaying and managing only processes running in the container, not the host. And, the network namespace provides the container with its own network device and virtual IP address which serves great for achieving reproducible scientific results.

Our application is packaged into Docker image stored on DockerHub and deployed through Amazon Elasticbeanstalk (EBS). As an input for EBS we provide Dockerfile [7] and configuration for auto-scaling group. Each container can be used for executing tasks by packaging the tasks into a container and deploying the container into a cluster. By running containers on an existing cluster, it is possible to auto-scale resources in response to demand. Capacity can also be shared with other processes such as application containers, GPU hardware, taking advantage of fluctuations in demand on the existing cluster or using Amazon EC2 Spot pricing to service the load. Moreover, EBS provides with GPU instances, which provide access to NVIDIA GPUs with up to 1,536 CUDA cores which leverages TensorFlow ability to utilise the GPU.

## Conclusions

To investigate the applicability of our application for practical video context prediction tasks authors experimented with a lot of different video streams. Application has shown good results and satisfying accuracy, thus proving the correctness and efficiency of TensorFlow usage for video processing tasks. With the development of container technology, many of the benefits of parallelising application can be achieved at a significantly lower cost. Moreover it enables for repeatable scientific results. Having at our disposal such powerful tools as deep learning algorithms, especially neural networks with the deepness that could not be imagined decade ago, the application architecture and optimisation will drive the progress of practical implementation of theoretical results. Further research aims to develop an automated system capable of detecting more complex objects and using and benchmarking the video structure ontology more efficiently.

1. *Abadi M., Agarwal A., Barham P., et al.* TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. *Culjak I., Abram D., Pribanic T., Dzapo H. and Cifrek M.* "A brief introduction to OpenCV," MIPRO, 2012 Proceedings of the 35th International Convention, Opatija, 2012. – P. 1725–1730.
3. *Viola P., Jones M.* Rapid object detection using a boosted cascade of simple features, Proceedings of the CVPR // IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2001.
4. *Boettiger C.* An introduction to Docker for reproducible research, with examples from the R environment // ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts. – 2012. – 49(1). – P. 71–79.
5. *Deng J., Dong W., Socher R., Li L.-J., Li K. and Fei-Fei L.* ImageNet: A Large-Scale Hierarchical Image Database // IEEE Computer Vision and Pattern Recognition (CVPR), 2009.
6. *Tomas M., et al.* "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781, 2013.
7. *Repository* with Docker config files: https://github.com/cubicova17/tensorflow-opencv, 2016.

## References

1. Abadi M., Agarwal A., Barham P., et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. Culjak I., Abram D., Pribanic T., Dzapo H. and Cifrek M. "A brief introduction to OpenCV," MIPRO, 2012 Proceedings of the 35th International Convention, Opatija, 2012. – P. 1725–1730.
3. Viola P., Jones M. Rapid object detection using a boosted cascade of simple features, Proceedings of the CVPR // IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2001.
4. Boettiger C. An introduction to Docker for reproducible research, with examples from the R environment // ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts. – 2012. – 49(1). – P. 71–79.
5. Deng J., Dong W., Socher R., Li L.-J., Li K. and Fei-Fei L. ImageNet: A Large-Scale Hierarchical Image Database // IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

6. Tomas M., et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781, 2013.
7. Repository with Docker config files: https://github.com/cubicova17/tensorflow-opencv, 2016.

## *Information about author:*

*Voloshyn Dmytro,*
Junior Research Associate.
Publications: 5.
http://orcid.org/0000-0002-9160-2746

## *Affiliation:*

Institute of Software Systems NAS Ukraine,
03187, Kyiv - 187, Academician Glushkov ave, 40.
Tel.: (095) 490 0641.
E-mail: wdmytriy@gmail.com