УДК 004.8

DOi 10.30837/ bi.2021.1(96).05

Nataliia Golian[1], Iryna Afanasieva[2], Vira Golian [3], Dmytro Panchenko[4]

[1]associate professor of the department of Software Engineering,
NURE, Ukraine, nataliia.golian@nure.ua
[2]associate professor of the department of Software Engineering,
NURE, Ukraine, iryna.afanasieva@nure.ua
[3]associate professor of the department of Software Engineering,
NURE, Ukraine, vira.golan@nure.ua
[4]master's student of the department of Software Engineering,
Ukraine, dmytro.panchenko@nure.ua

APPLYING GRADIENT BOOSTING AS A STACKING ALGORITHM OVER BOTTLENECK FEATURES TO ACHIEVE HIGH IMAGE CLASSIFICATION ACCURACY

With the development of the Internet, making many images available online for analysis, object recognition software is gaining more and more attention from researchers. Factors are driving the development of computer vision today: mobile devices with built-in cameras, the availability of computing power, the availability of computer vision and analysis equipment, and new algorithms such as convolutional neural networks that take advantage of the power of hardware and software. The work is generally devoted to the consideration of the problem of image classification using convolutional neural networks. And in particular, one of the most popular and applied in practice machine learning algorithms – gradient boosting applied to the bottlenecks of deep convolutional neural networks. It also discusses three scenarios for applying gradient boosting to bottlenecks extracted from the last convolutional layer of the neural network. The essence of boosting, as well as of other ensembles of algorithms, is to collect one strong from several weak models. The general idea of boosting algorithms is to consistently apply predictors so that each subsequent model minimizes the error of the previous one. Gradient boosting works by sequentially adding new models to past models so that errors made by previous predictors are corrected.

ARTIFICIAL INTELLIGENCE, COMPUTER VISION, GRADIENT BOOSTING, IMAGE, MACHINE LEARNING, NEURAL NETWORK, PATTERN RECOGNITION

Голян Н., Афанасьева И., Голян В., Панченко Д. Применение градиентного бустинга в качестве алгоритма стекинга по узким местам для достижения высокой точности классификации изображений. С развитием Интернета, сделавшим многие изображения доступными онлайн для анализа, программное обеспечение для распознавания объектов привлекает все больше внимания исследователей. Факторы стимулируют развитие компьютерного зрения сегодня: мобильные устройства со встроенными камерами, доступность вычислительной мощности, доступность оборудования для компьютерного зрения и анализа, а также новые алгоритмы, такие как сверточные нейронные сети, которые используют аппаратные и программные возможности. Работа в целом посвящена рассмотрению проблемы классификации изображений с помощью сверточных нейронных сетей. И, в частности, одному из самых популярных и применяемых на практике алгоритмов машинного обучения – градиентного бустинга, применяемого к узким местам глубоких сверточных нейронных сетей. Также рассматриваются три сценария применения градиентного бустинга к узким местам, извлеченным из последнего сверточного слоя нейронной сети. Суть бустинга, равно как и других ансамблей алгоритмов, состоит в том, чтобы из нескольких слабых моделей собрать одну сильную. Общая идея алгоритмов бустинга – последовательно применять предикторы так, чтобы каждая последующая модель минимизировала ошибку предыдущей. Градиентный бустинг работает последовательно добавляя к прошлым моделям новые так, чтобы исправлялись ошибки, допущенные предыдущими предикторами.

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ГРАДИЕНТНЫЙ БУСТИНГ, ИЗОБРАЖЕНИЕ, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННАЯ СЕТЬ, РАСПОЗНАВАНИЕ ОБРАЗОВ

Голян Н., Афанасьєва І., Голян В., Панченко Д. Застосування градієнтного бустінгу в якості алгоритму стекінгу по вузьких місцях для досягнення високої точності класифікації зображень. З розвитком Інтернету, що зробив багато зображень доступними онлайн для аналізу, програмне забезпечення для розпізнавання об`єктів привертає все більше уваги дослідників. Фактори стимулюючі розвиток комп'ютерного зору сьогодні: мобільні пристрої з вбудованими камерами, доступність обчислювальної потужності, доступність обладнання для комп'ютерного зору і аналізу, а також нові алгоритми, такі як згорткові нейронні мережі, які використовують можливості обладнання і програмне забезпечення. Робота в цілому присвячена розгляду проблеми класифікації зображень за допомогою згорткових нейронних мереж. І, зокрема, одному з найпопулярніших і застосовуваних на практиці алгоритмів машинного навчання – градієнтному бустінгу, що застосовується до вузьких місць глибоких згорткових нейронних мереж. Також розглядаються три сценарії застосування градієнтного бустінгу до вузьких місць, добутих з останнього згорткового шару нейронної мережі. Суть бустінга, так само як і інших ансамблів алгоритмів, полягає в тому, щоб з кількох слабких моделей зібрати одну сильну. Загальна ідея алгоритмів бустінга – послідовно застосовувати предиктори так, щоб кожна наступна модель мінімізувала помилку попередньої. Градієнтний бустінг працює послідовно додаючи до минулих моделей нові так, щоб виправлялися помилки, допущені попередніми предикторами.

ШТУЧНИЙ ІНТЕЛЕКТ, КОМП'ЮТЕРНИЙ ЗІР, ГРАДІЄНТНИЙ БУСТІНГ, ЗОБРАЖЕННЯ, МАШИННЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, РОЗПІЗНАВАННЯ ОБРАЗІВ

1. Introduction and preliminaries

During the last few years, computer vision (and image classification in particular) became one of the fastest developing areas in computer science. Different methods and algorithms to solve this problem were developed, and the most prominent of them is the usage of convolutional neural networks (CNN).

A neural network is a system of many neurons (processors). Separately, these processes are quite simple, but connected into a system, neurons perform very complex tasks of collecting information.

Among the main areas of application of neural networks are forecasting, decision making, pattern recognition, optimization, data analysis. Neural networks are at the heart of most modern speech recognition and synthesis systems, as well as image recognition and processing.

CNN is a class of deep neural networks which was originally created specifically for image processing and analysis. CNN is based on multilayer perceptron, but instead of using only fully-connected layers (which are prone to overfitting) it includes specific types of layers designed for extracting patterns from complex input image. Moreover, those layers allow CNNs to have less connections between neurons therefore drastically decreasing time required for network running.

The most basic CNNs are created using following building blocks:

– convolutional layers;
– pooling layers;
– fully-connected layers.

Convolutional layers consist of neurons, each of them covers specific part of the image (i.e. receptive field). Kernel (filter) of the layer defines how data from receptive field will be transformed into the output by its convolution with an input. It means that network will learn filters that activate when some specific type of feature is present at that part of the image.

Apart from receptive field width and height convolutional layers are also parametrized by their depth. Depth means number of convolutions (channels) in the layer which point to the same location of the input. These convolutions represent different features of the input area.

Pooling layers are used to reduce size of the input by partitioning it into a set of non-overlapping squares. For each such region single value is output, the most common is to use maximum of all values in this region (max pooling), but alternative approaches exist as well (e.g. average pooling). As the result, minor and unimportant for the problem at hand features are discarded, and important ones are kept.

Common architectures of CNNs use alternating sequence of convolutional and pooling layers (latter ones are inserted between successive convolutional layers). Such approach allows to separate more and more general features step by step.

After several convolutional and pooling layers usually go one or more fully-connected layers. They are identical to layers in regular (non-convolutional) neural networks and used for actual classification of input based on features extracted by convolutional layers.

2. Subject area

Neural networks derive their strength, firstly, from the parallelization of information processing and, secondly, from the ability to self-learn, that is, to create generalizations. The term generalization refers to the ability to obtain a reasonable result based on data that was not encountered in the learning process. These properties allow neural networks to solve complex (large-scale) problems that are considered intractable today. However, in practice, when working autonomously, neural networks cannot provide ready-made solutions. They need to be integrated into complex systems. In particular, a complex problem can be divided into a sequence of relatively simple ones, some of which can be solved by neural networks.

First attempts at applying deep neural networks to computer vision problems were made in 90s by Yann LeCun et al [1] – researchers achieved substantial results in recognizing handwritten numbers in images.

However, deep learning did not become a commonly used approach in computer vision until 2012, when it was used by Alex Krizhevskiy et al [2] to achieve state-of-the-art accuracy in ImageNet LSVRC-2010 competition. Krizhevskiy, Hinton and LeCun used convolutional neural networks to develop automatic feature extraction that is trained in an end-to-end manner with the classifier.

Main constraints that slowed down development of deep learning at that point were:

– absence of the dataset big enough to fit models with millions of trainable parameters;
– insufficient computational powers.

The identified problems were solved in the following three ways.

First of all, several huge datasets were created and labeled. The most well-known one is ImageNet [3] that consists of several millions of images labeled to a thousand classes (multiclass classification task).

Secondly, modern GPU architectures made researchers able to train deep models quite fast using hardware acceleration. Even specialized hardware optimized for tensor computations was developed [4].

It allowed researchers to train huge models and even apply hyperoptimization algorithms that build large architectures utilizing hundreds of GPUs at a time as shown by Zoph et al [5]. Network architecture developed by Zoph et al called NASNet contained 88.9 million parameters and achieved unprecedented at that time result of 0.960 top-5 accuracy on ImageNet. This example shows that nowadays computational complexity is not a problem. However, model speed still matters sometimes, and there

are different tradeoffs, such as tradeoff of accuracy versus inference speed (it is important for mobile applications and IoT devices using deep learning models for instance) and tradeoff of accuracy versus training speed (it can be important for baselining and exploratory analysis, when researcher wants to quickly assess how much accuracy can be expected from the given data at all). At last, there are cases when model speed does not matter at all, and accuracy is the only crucial metric.

Lastly, researchers developed a way to reuse weights of model trained on one dataset to another machine learning problem. This approach is called transfer learning.

Today, in most applications of deep learning, especially in the field of computer vision, training a deep neural network from scratch is impractical. The use of transfer training is currently the key to the top results in many problems. Transfer learning is an approach that takes a network that has already been trained on a larger dataset and then uses it as an initialization of the weight for further training. Typically, ImageNet acts as such a large dataset for preliminary training.

Transfer learning is based on the fact that convolutional neural network consists of two parts – convolutional part and fully-connected part.

Convolutional part acts essentially as a feature extractor that generates structured and highly separable features (sometimes called *bottleneck features*) from the unstructured input data – images in case of computer vision – by applying several stacked spatial transformations – convolutional and pooling layers.

Fully-connected layers in turn act as a classifier that models decision boundary for specific categories of objects.

So the idea of network-level transfer learning [6] is to reuse convolutional part of the network to apply weights pre-trained on a large dataset to the target dataset which is smaller (and in many cases even too small to train network from scratch at all).

Another important aspect of using convolutional part of a convolutional neural network as a feature extractor is studied in a paper by Zeiler & Fergus [7]. The fact is demonstrated that convolutional layers differ in terms of complexity and level of abstraction of structures recognized by these layers. It is shown that the deep layers of the network, close to the input, are studied low-level structures and simple geometric shapes, such as colored spots, lines, gradients. The closer the layer is to the bottleneck the more high-level features are detected by this layer. It means that latest layers of the deep convolutional neural network represent dataset-specific structured features (for example, for ImageNet last convolutional layer represents such features as cat ears, car wheel, etc.) that act as effective descriptors for solving not only a problem at hand for this particular dataset but a set of computer vision tasks for all datasets that are visually similar to training data.

Meanwhile, in the field of tabular data classification, where features are structured by nature, gradient boosting has established itself as one of the state-of-the-art classifiers outperforming other algorithms in various problems with complex feature spaces and non-linear decision boundaries [8]. The most popular and powerful implementations of gradient boosting over decision trees are LightGBM [9], xgboost [10] and CatBoost [11].

LightGBM is a fast, distributed, high-performance gradient boosting structure based on a decision tree algorithm used for ranking, classification, and many other machine learning tasks. Because it is based on decision tree algorithms, it splits the best-matched leaf of the tree, whereas other boosting algorithms divide the tree by depth or level rather than leaf. Thus, when grown on the same leaf in Light GBM, the leaf algorithm can reduce more losses than the layer-by-layer algorithm, and therefore results in much better accuracy, which can rarely be achieved by any of the existing boosting algorithms.

LightGBM uses the gradient one-sided sampling technique (GOSS) technique to filter data instances and find the split value. At the same time, XGBoost uses a pre-sorted algorithm and a histogram based algorithm to calculate the best split. Observations / Samples are examples here.

More specifically, a histogram-based algorithm breaks all data points for an object into discrete elements and uses them to find the split value of the histogram. It is more efficient than the pre-sorted algorithm in learning speed, which lists all possible split points on the pre-sorted feature values, but it still lags behind GOSS in terms of speed.

GOSS (Gradient One-Way Sampling) is a new sampling technique that downsamples based on gradients. Cases with small gradients are well trained (small learning error), and cases with large gradients are underscored. A naive approach is to discard instances with small gradients, focusing exclusively on instances with large gradients, but that would change the distribution of the data. GOSS keeps instances with large gradients by randomly sampling instances with small gradients.

Benefits of LightGBM:

– high efficiency and fast learning speed. LightGBM uses a histogram-based algorithm, that is, it combines continuous feature values into discrete cells, which speed up the learning process;

– lower memory usage. Replaces continuous values with discrete cells, which in turn results in lower memory usage;

– better accuracy than any other gain algorithm. It creates much more complex trees using a sheet rather than level approach, which is a major factor in achieving higher accuracy;

– compatible with large datasets. It is capable of performing equally well on large datasets with a significant reduction in training time.

Finally, ensembles are used either in computer vision or tabular data tasks. Ensembling is a technique of combining outputs of several base learners to reach prediction quality that is better than any of the standalone models is able to reach by itself. Many researchers have studied ensembling and different ways of combining classifiers starting with the bootstrap aggregation [12] and ending with stacking [13].

Stacking is an ensembling technique that allows to combine several models (in our case – classifiers) by using their predictions as a set of input features to the second level model (so called meta-classifier) [14].

The main idea of stacking is to use basic classifiers to get predictions and use them as features for some general algorithm. That is, the essence of stacking is the transformation of the original space of features of the problem into some new space, the points of the latter are the predictions of the basic algorithms.

First, a set of pairs of arbitrary subsets is selected from the training sample, and then, for each pair, it is necessary to train the basic algorithms on the first subset, and also predict the target variable for the second subset with them. In this case, the predicted values become objects of the new space. Stacking has been the primary way to ensemble the underlying algorithms of many machine learning competitions.

It is proposed to use gradient boosting as a classification algorithm for image classification task by applying it over bottleneck features of deep convolutional neural networks (in fact – stacking of bottleneck descriptors by gradient boosting). The main contribution and novelty is studying different scenarios of such stacking.

3. Methods

This article explores three scenarios of applying gradient boosting over bottleneck features extracted from the last convolutional layer of a CNN:

– a convolutional neural network is taken, previously trained on ImageNet. The bottleneck features of the frozen network are extracted for all images in the training and test parts part of the target dataset. After that the gradient boosting is fitted on the training set and evaluate it on the test set;

– a convolutional neural network is taken, pre-trained on ImageNet. The network is fine-tuned to the target dataset by fitting it on the whole training subset. After that, the bottleneck features of the fine-tuned network are extracted for all images in the training and test parts part of the target dataset, the gradient boosting isfited on the training set and evaluate it on the test set;

– several CNNs of different architectures are taken with pre-trained ImageNet weights. The bottleneck features are extracted from those networks, concatenated together and trained gradient boosting on the joint feature vector on the training set. After that, it is evaluated again on the test set.

The experiments are restricted by several assumptions. First of all, the scenario when several networks are fine-tuned to the target set is not tested, because in such case it is possible to apply classical stacking over models' predictions instead of learning from the bottleneck layers.

Only scenarios in which meta-classifier is trained on intermediate features extracted from a neural network are discussed. Also, the folds-in-folds stacking approach that prevents data leakage in the training set is not tested, because, firstly, out-of-fold model training and prediction are impractical in real world situation and, secondly, because there is no any guarantee that particular bottleneck features learnt by a neural network would have similar meaning at different folds which is a crucial assumption for using gradient boosting classifier after all.

The hypothesis is that structured features extracted by a convolutional neural network from raw input data (such as images) can serve as a suitable feature space.

4. Experiments

The experiments use data from the iMaterialist Challenge (Furniture) [15]. Dataset consists of 194 828 images in the training set and 6400 images in the validation set. Each image in dataset represents one of 128 classes of furniture and household items (e.g. chairs, beds, cookware, etc.), so it is a multiclass classification problem.

Examples of images (cookware) from dataset are presented in the Figure 1.

Fig. 1. Examples of images from iMaterialist Challenge (cookware)

Also taken for research, examples of images (chairs) from the dataset are presented in Figure 2.

Fig. 2. Examples of images from iMaterialist Challenge (chairs)

LightGBM used as a gradient boosting classifier implementation.

For testing purposes, the following CNN architectures must be configured, pre-trained on ImageNet:

– Xception [16];
– NASNet Large [5];
– DenseNet-201 [17];
– ResNet-152 [18].

A conventional convolutional layer handles the simultaneous correlation of adjacent points within one channel, spatial information (spatial information) and inter-channel information, because convolution is applied to all channels as a whole.

The Xception architecture assumes that these two types of information can be processed sequentially and without sacrificing network quality.

Xception decomposes regular convolution into spatial convolution (processes spatial correlation in terms of a single channel) and pointwise convolution (processes inter-channel correlation).

The initial depth-separable convolution is the depth convolution followed by the point convolution. Depth convolution is a channel-by-channel n×n spatial convolution. Point convolution is a 1×1 convolution to change a dimension.

The modified depth-separable convolution is a point convolution followed by a deep convolution. Modified deeply split convolution is used as the seed module in Xception (an extreme version of the seed module).

Differences:

– sequence. The original depth-separable convolutions first perform channel-by-channel spatial convolution, and then perform 1×1 convolution, as, for example, in TensorFlow. The modified depth-separable convolution first performs a 1×1 convolution followed by spatial wise convolution. This is denoted as not so important as, when used in a multi-layered setup, only slight differences appear at the beginning and end of all related starter modules;

– non-linearity. Non-linearity is observed in the entry-level starter module after the first operation. In Xception (modified depth-separable convolution), there is no intermediate non-linearity of ReLU [16].

NASNet-Large is a pretrained model on a subset of the ImageNet database. It belongs to the models of the NASNet architecture family. The NASNet architecture has been learned from data using a repetitive neural network, instead of fully developed by humans like other pretrained models [5].

DenseNet-201 is a convolutional neural network with 201 layers deep. It is possible to download a pretrained version of the network that has been trained on over a million images from the ImageNet database. A pretrained network can classify images into specific categories of objects such as keyboard, mouse, pencil, and others. As a result, the network studied the representations of functions for a fairly large range of images.

ResNet is a deep residual learning framework for image classification problem. Supports multiple architectural configurations to achieve the right balance between speed and quality. The ResNet architecture (with three of its implementations: ResNet-50, ResNet-101 and ResNet-152) has received successful results in ImageNet competitions. The basic idea used in these models, residual couplings, greatly improves gradient flow. This allows you to train much deeper models with tens and hundreds of layers.

For each experiment two metrics are calculated: accuracy and logloss.

Equation (1) displays the metric for calculating the accuracy.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Equation (2) shows the calculation for the metric relative to logloss.

$$H(p,q) = \sum_{x \in \S} p(x) \log q(x) \quad (2)$$

For performance reasons, gradient boosting is not trained on the full concatenated feature vector from four networks. Instead, PCA [19] is first applied to reduce the feature space to 2048 vectors, and only then is gradient boosting applied. Also, added flipped images to the train dataset as a mean of simple offline augmentation.

Principal component analysis (multivariate statistical analysis technology) is used to reduce the size of the feature space with minimal loss of useful information. The implication is that each principal component is associated with a certain proportion of the total variance of the original dataset (load). In turn, variance, which is a measure of data variability, can reflect the level of their information content. Principal component analysis is included in most analytical platforms and is widely used to reduce the dimension of input data at the stage of their preprocessing.

The main limitations of the principal component analysis are:

– impossibility of semantic interpretation of the components;

– the method can only work with continuous data.

The method is sometimes considered as part of a more general approach to data dimensionality reduction - factor analysis. In analytical platforms, it is the principal component method that is often practically implemented in factor analysis modules.

Networks of same architectures are used, fine-tuned to the dataset as a baseline.

The data obtained from the results of the experiments are presented in Table 1.

Table 1

Experiment data

Name	Accuracy	Logloss
Xception	0.8677	0.5299
LightGBM over ImageNet pre-trained Xception bottleneck features	0.7332	0.8920
LightGBM over fine-tuned Xception bottleneck features	0.6698	1.1906
NASNet Large	0.8677	0.4956
LightGBM over ImageNet pre-trained NASNet Large bottleneck features	0.7469	0.8879
LightGBM over fine-tuned NASNet Large bottleneck features	0.6565	1.2727

5. Results and discussion

Experiments show that stacking bottleneck features from a bunch of neural networks pre-trained on the ImageNet performs best in terms of logloss (i.e. outputs the most optimal probabilistic predictions of all tested models). In terms of accuracy it is only 0.77% worse than fine-tuned deep convolutional neural networks, however training gradient boosting on bottleneck features requires only a single inference from each network and fitting of boosting itself, while fine-tuning of a single neural network with Xception architecture takes around 30 GPU-hours and fine-tuning of a single NASNet takes around 40 GPU-hours on GTX 1080.

6. Conclusion

The paper considers the use of gradient boosting as a classification algorithm for the image classification problem. A study is presented to investigate scenarios for applying gradient boosting to bottlenecks extracted from the last CNN convolutional layer.

It can be concluded that gradient boosting on bottleneck features of the pre-trained networks performs well as a quick solution that does not require a lot of training. It is important to notice that boosting on ImageNet features works better than boosting on the features of already fine-tuned model despite the fact that ImageNet features are less optimized for this particular task. It can be explained by the fact that training convolutional neural network as a feature extractor and then gradient boosting as a classifier on a single dataset leads to a data leakage which presents itself in a form of overfitting. Though it is anyway impractical to use gradient boosting as a classifier when the network is already fine-tuned, since full convolutional network tuned in an end-to-end manner always performs better.

As a point for further research it is proposed to compare the considered approach stacking of bottleneck features with gradient boosting as a meta-classifier, with classical stacking approaches to determine optimal usage strategies.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] *LeCun Y., Boser B., Denker J., Henderson D., Howard R., Hubbard W., Jackel L.* Handwritten digit recognition with a back-propagation network // Advances in Neural Information Processing Systems 2, 1990. – P. 396-404.

[2] *Krizhevsky A., Sutskever I., Hinton G.* ImageNet classification with deep convolutional neural networks // Neural Information Processing Systems, 2012.

[3] *Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L.* ImageNet: A Large-Scale Hierarchical Image Database // CVPR09, 2009.

[4] *Jouppi N.P., Young C., Patil N.* In-datacenter performance analysis of a tensor processing unit // CoRR, 2017. – http://arxiv.org/abs/1704.04760.

[5] *Zoph B., Vasudevan V., Shlens J., Le Q.V.* Learning transferable architectures for scalable image recognition // CoRR, 2017. – http://arxiv.org/abs/1707.07012.

[6] *Tan C., Sun F., Kong T., Zhang W., Yang C., Liu C.* A Survey on Deep Transfer Learning // 27th International Conference on Artificial Neural Networks, 2018.

[7] *Zeiler M.D., Fergus R.* Visualizing and Understanding Convolutional Networks // Computer Vision – ECCV 2014.

[8] *Friedman J.* Greedy function approximation: a gradient-boosting machine // Ann. Statist. 29, 2001. – P. 1189-1232.

[9] *Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye O., Liu T.-Y.* LightGBM: A Highly Efficient Gradient Boosting Decision Tree // Advances in Neural Information Processing Systems 30, 2017. – P. 3146-3154.

[10] *Chen T., Guestrin C.* XGBoost: A Scalable Tree Boosting System // CoRR, 2016. – http://arxiv.org/abs/1603.02754.

[11] *Dorogush V., Ershov V., Gulin A.* CatBoost: gradient boosting with categorical features support // CoRR, 2018. – http://arxiv.org/abs/1810.11363.

[12] *Machova K., František B., Bednár P.* A Bagging Method using Decision Trees in the Role of Base Classifiers // Acta Polytechnica Hungarica 3, 2006.

[13] *Wolpert D. H.* Stacked generalization // Neural networks, 5(2), 1992. – P. 241-259.

[14] Marios Michailidis StackNet, StackNet Meta Modelling Framework, 2017. – https://github.com/kaz-Anova/StackNet.

[15] iMaterialist Competition 2018 https://sites.google.com/view/fgvc5/competitions/imaterialist.

[16] *Chollet F.* Xception: Deep Learning with Depthwise Separable Convolutions // CoRR, 2016. – http://arxiv.org/abs/1610.02357.

[17] *Huang G., Liu Z., Weinberger K.* Densely Connected Convolutional Networks // CoRR, 2016. – http://arxiv.org/abs/1608.06993.

[18] *He K., Zhang X., Ren S., Sun J.* Deep Residual Learning for Image Recognition // CoRR, 2015. – http://arxiv.org/abs/1512.03385.

[19] *Shlens J. A Tutorial on Principal Component Analysis // CoRR, 2014. – http://arxiv.org/abs/1404.1100.*

The article was delivered to editorial stuff on the 13.01.2021