sciendo RIGA TECHNICAL UNIVERSITY *Electrical, Control and Communication Engineering*
ISSN 2255-9159 (online)
ISSN 2255-9140 (print)
2019, vol. 15, no. 2, pp. 54–61
doi: 10.2478/ecce-2019-0008
https://content.sciendo.com

Generator of a Toy Dataset of Multi-Polygon Monochrome Images for Rapidly Testing and Prototyping Semantic Image Segmentation Networks

Vadim Romanuke* (*Professor, Polish Naval Academy, Gdynia, Poland*)

***Abstract* – In the paper, the problem of building semantic image segmentation networks in a more efficient way is considered. Building a network capable of successfully segmenting real-world images does not require a real semantic image segmentation task. At this stage, called prototyping, a toy dataset can be used. Such a dataset can be artificial and thus may not need augmentation for training. Besides, its entries are images of much smaller size, which allows training and testing the network a way faster. Objects to be segmented are one or few convex polygons in one image. Thus, a toy dataset generator is created whose complexity is regulated by the number of edges in a polygon, the maximal number of polygons in one image, the set of scale factors, and the set of probabilities determining how many polygons in a current image are generated. The dataset capacity and image size are concurrently adjustable, although they are much less influential.**

***Keywords* – Dataset complexity; Multi-polygon object; Semantic image segmentation; Segmentation network architecture; Toy dataset; Two-class segmentation.**

I. Introduction to Semantic Image Segmentation

Semantic image segmentation (SIS) is a computer vision task of labeling specific regions of an image by subsequently filling in the regions with respective colors. The colors are chosen so that they would be distinguishable as much as possible (Fig. 1). Sometimes, a perceptible rate of transparency is applied (Fig. 2). More specifically, the goal of SIS is to label each pixel of an image with a corresponding class or category of what is being imaged [1], [2].

Theoretically, a semantic segmentation network (SSN) classifies every pixel in an image [2], [3]. This results in an image of the same resolution that is segmented by classes or categories. Inasmuch as the spatial resolution of an image is not downsampled purposely, the network processes huge amounts of data. Thus, SIS is a challenge for modern machine learning.

Neural network architecture for SIS is based on an encoder/decoder structure [4], [5]. The spatial resolution of the input is downsampled developing lower-resolution feature mappings, and then the feature representations are upsampled into a full-resolution segmentation map. A common SSN consists of three parts: a downsampling subnetwork, an upsampling subnetwork, and a pixel classification layer. A downsampling subnetwork is stacked of convolutional layers (ConvLs), ReLUs, and max pooling layers. The upsampling is executed using the transposed convolutional layer, which is also commonly referred to as deconvolutional layer (DeConvL) [4], [6]. DeConvL simultaneously performs the upsampling and filtering, so the upsampling subnetwork is stacked of DeConvLs and ReLUs. The final set of layers performs pixel classifications. These final layers process an input that has the same spatial dimensions of height and width as the input image. The third dimension, which is equal to the number of filters in the last DeConvL, is squeezed down to the number of classes which are tasked to be segmented. The squeezing is performed by a 1-by-1 ConvL, whose number of filters is equal to the number of classes. The softmax and pixel classification layers, following the 1-by-1 ConvL (which is a fully-connected layer), categorically label each image pixel.

Fig. 1. An image which is to be semantically segmented and the result of segmentation [7]. There are a few tens of classes of labeled pixels, and each class is shown with its own color. Some colors are chosen in accordance with real tints of the corresponding objects or scenes (for instance, the trees in this example are labeled green, although there is no verdure due to a specific season). This is an example of a very accurate complex segmentation.

* Corresponding author.
E-mail: romanukevadimv@gmail.com

Fig. 2. An example of two-class segmentation by transparency in labeling. Here, the task is to segment a complex object, which is the landscape and other items except for the water and sky. The accuracy is not perfect therein.

What should an efficient SSN architecture be like for a given SIS task? Except for the 1-by-1 ConvL inserted before the softmax and pixel classification layers, how many ConvLs and DeConvLs should the SSN have? These questions are not trivial as inserting ConvLs and DeConvLs appropriately cannot be performed in a single step. The appropriateness here is meant by a weaker efficiency, at which an SSN would perform at an acceptable accuracy, although not close to perfect. It will assuredly take a few tries to build an efficient SSN architecture even for simple tasks, with two classes (like a task with the example in Fig. 2). For tasks like that with the segmentation result in Fig. 1, building an appropriate SSN may take too many tries, where each lasts for a few hours [5].

II. Background and Motivation for Toy Datasets

Any neural network being built requires a dataset, on which it is trained, validated, and tested. This stage is an actual prototyping. The goal of the prototyping is to build an SSN capable of segmenting successfully real-world images. The prototyping, however, does not require a real SIS task. Therefore, a dataset for the prototyping can be selected so that it would allow building an SSN (including training, validation, testing) as fast as possible. Obviously, a connection between the dataset and the real SIS task must exist [3], [4], [6].

As of July 2019, SSNs are prototyped still based on the experience rather than a strong theory. Datasets of mostly high-resolution real-world images along with respective labels to each pixel in every image instance are used for this. Such datasets like BSDS500, CamVid, Cityscapes, COCO, DUS, Mapillary Vistas (an example in Fig. 1), MSRCv2, PASCAL VOC, etc., fit excellently for the corresponding SIS tasks but training on them is still expensive [7]. Augmentation of training data, especially needed for smaller datasets (BSDS500 and DUS), is limited [5], [8]. On the contrary, artificial datasets are infinitely scalable and thus they do not need augmentation (Fig. 3), although they fit much simpler SIS tasks (Fig. 4).

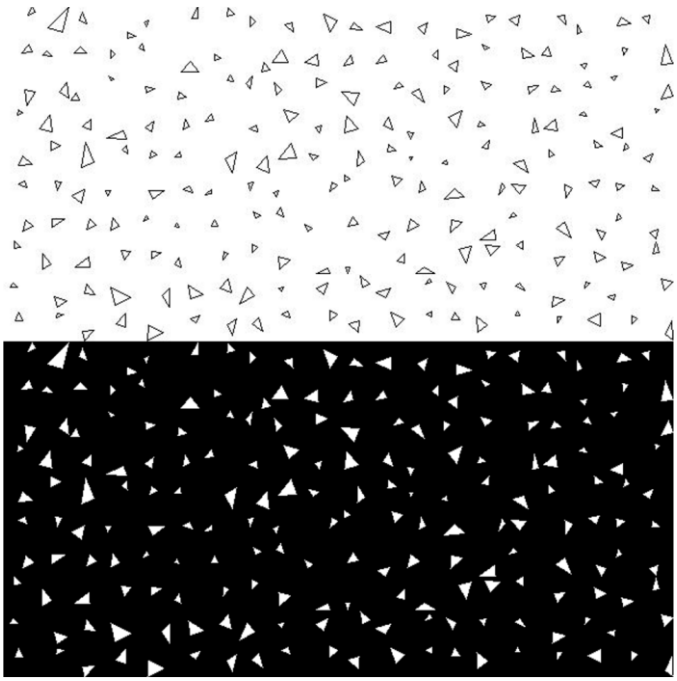

Fig. 3. An example of an artificial dataset of triangles (grayscale images above and labels beneath) used by MATLAB in instructing how to deal with SSNs (https://www.mathworks.com/help/vision/ref/semanticseg.html). The dataset is intended for an SIS task with two classes. It consists of 200 32×32 images, which can be easily reproduced with varying positions of the triangle. The task is to segment the triangle in a single image or triangles in a stack of multiple images. Note that the background color coincides with the color of the triangle interior. This is done intentionally to make the SIS task a bit sophisticated. Otherwise, such a task would be solved easily by thresholding.

```
 1 'imageinput'      Image Input
                     32x32x1 images with 'zerocenter' normalization
 2 'conv_1'          Convolution
                     64 3x3x1 convolutions with stride [1  1]
                     and padding [1  1  1  1]
 3 'relu_1'          ReLU
 4 'maxpool'         Max Pooling
                     2x2 max pooling with stride [2  2]
                     and padding [0  0  0  0]
 5 'conv_2'          Convolution
                     64 3x3x64 convolutions with stride [1  1]
                     and padding [1  1  1  1]
 6 'relu_2'          ReLU
 7 'deconv'          Transposed Convolution
                     64 4x4x64 transposed convolutions
                     with stride [2  2] and output cropping [1  1]
 8 'conv_3'          Convolution
                     2 1x1x64 convolutions with stride [1  1]
                     and padding [0  0  0  0]
 9 'softmax'         Softmax
10 'classoutput'     Pixel Classification Layer
                     Class weighted cross-entropy loss
                     with classes 'triangle' and 'background'
```

Fig. 4. The SSN architecture in MATLAB for segmenting 32×32 images with triangles. There are two ConvLs, each of which is followed by a ReLU. A single DeConvL is used following the second ReLU. The SSN is trained on the dataset in Fig. 3. After 100 epochs of training, the SSN performs at an acceptable accuracy (see its testing in Fig. 5), although it is far from perfect.

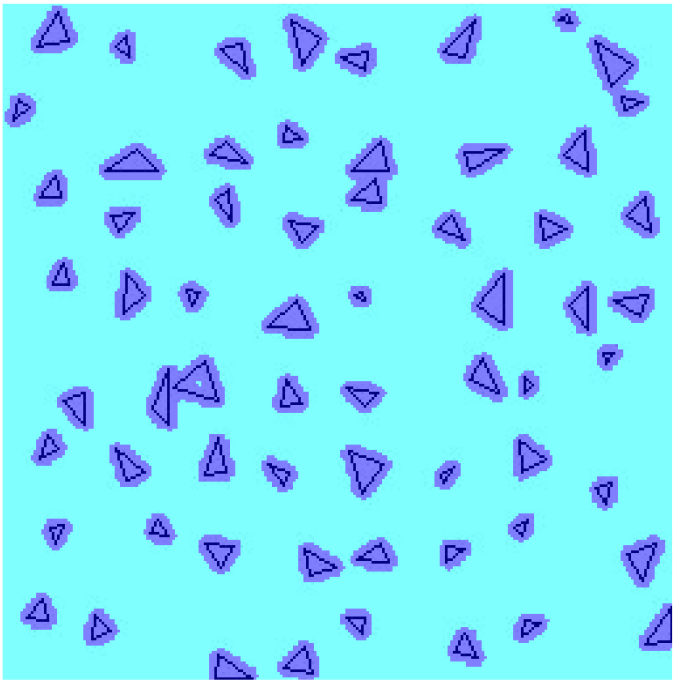

Fig. 5. A stack of 64 test triangle images and the fused overlay image as a result of segmentation by an SSN with the architecture in Fig. 4 trained on the dataset in Fig. 3. Despite all 64 triangles have been spotted, the accuracy of segmentation is believed to be improved by adjusting parameters of training and hyperparameters [9] of the SSN. Note that there are three triangle couples "fused" into one due to lower accuracy. The size of the triangles varies as dramatically as that in the training dataset in Fig. 3, but the triangle shape is still the simplest to pretend to be a toy pattern for some simple real-world SIS tasks (e. g., segmenting one-color objects of not-a-curvilinear shape).

Apparently, a toy dataset can be made more complex. Then, an appropriate SSN will be tested and prototyped faster anyway owing to the dataset's independent augmentation by infinite scalability. Subsequently, this SSN's parameters and hyperparameters could be imparted to real-world SIS tasks with images of the same or slightly higher resolution [10]. The number of classes to be labeled, obviously, must be the same. Nevertheless, for non-toy SIS tasks containing multiple classes (Fig. 1), the same architecture is expected to be appropriate by just increasing the values of hyperparameters (for instance, increasing filter numbers, intensifying pooling).

The connection between a toy dataset and the related real SIS tasks must be controlled via adjusting parameters of the dataset. Such parameters are supposed to be complexity of objects to be labeled and their density in an image. For maintaining a speed gain in training on toy datasets, they nonetheless must contain primitive objects [1], [3], [7].

Therefore, the motivation for toy datasets is explained with a possibility to overcome difficulties in training on real-world datasets. This is about infiniteness of dataset entries (independent augmentation) and faster training. Automatic labeling is another merit of toy datasets. Compared to datasets of natural images, building a toy dataset is much faster and cheaper, as toy entries are generated along with labeling them automatically by an algorithm. Unlike working with natural images, the algorithm does not perform preprocessing, nor does it perform any image format conversions. This additionally saves a lot of time and human resources.

III. Goal and Tasks

Proceeding from plausible benefits of toy datasets for SIS tasks, the goal is to create a toy dataset generator whose complexity could be regulated from the simplest mode up to the reasonably most sophisticated one. The simplest mode is to generate those triangles in Fig. 3. The reasonably most sophisticated mode will be based on complicating the object's shape, scattering its size, and admitting congestions of multiple objects, which may be eventually perceived as one object. The generator is believed to produce datasets, which could serve as a fast prototyping platform for SIS.

Firstly, the image size along with the shape and interior of the objects to be segmented will be discussed and substantiated. Secondly, an algorithm of the toy dataset generator will be stated. Finally, examples should be presented giving practical recommendations of how real-world SIS tasks could inherit appropriate parameters and hyperparameters from the SSN trained on the toy dataset generated by the stated algorithm.

IV. Image Size

In machine learning, and, particularly, in the training of neural networks on images, there are classical datasets like CIFAR-10, CIFAR-100, MNIST, NORB, EEACL26, etc. [9], [11], [12]. For them, researchers tend to set the image size at dimensions which are raised to some integer power [5]. This ensures faster training owing to consistency with the binary system hardware, on which computational algorithms are physically implemented. Therefore, let the minimal image size be 32×32. Generally, the image size is $h\times w$ (in pixels), where both height h and width w will be set at integers divisible by 32.

V. Shape and Interior of Objects to be Segmented

The triangle is the simplest convex polygon. So, let the image contain one or few convex polygons. The maximal number of polygons in one image is $P_{\max}$ by $P_{\max}\in\mathbb{N}$. Then, occasionally, the object can be non-convex when a few polygons intersect [13], [14].

The minimal number of polygon vertices is 3. As the maximal number of polygon vertices cannot be limited, let a polygon be generated of n vertices, where $n\in\left\{\overline{3, n_{\max}}\right\}$ by $n_{\max}\in\mathbb{N}\setminus\{1, 2\}$. Number $n_{\max}$ is a maximal number of edges in a polygon, and it is specifically selected for a given SIS task. Number n will be randomly chosen for every new image. A greater number $n_{\max}$ makes a given SIS task more complex, which may require a more complex SSN architecture.

The background remains white. Thus, the most appropriate color of the polygon interior is white. The object's color is black that implies the color of the polygon edges is black. In fact, the object is 100 % transparent. At that, whichever image size would be, the thickness of the polygonal edges will be just one pixel. This implies that the segmentation is harder for bigger images because always only a one-pixel border separates the object's white interior off the white background.

VI. Algorithm of the Toy Dataset Generator

The toy dataset generator has eight input arguments: a number of images N to be generated, h, w, a minimal number of edges $n_{\min}$ in a polygon, $n_{\max}$, $P_{\max}$, a set S of scale factors (to scale the polygon with respect to the image size), a set R of different probabilities (to determine how many polygons in an image will be generated). Whereas set S must contain at least one element, set R must consist of at least $P_{\max}-1$ elements:

$$S=\{s_k\}_{k=1}^{K},\ s_k\in(0;1)\ \ \forall k=\overline{1,K},\ K\in\mathbb{N}, \tag{1}$$

$$R=\{r_l\}_{l=1}^{L},\ r_l\in(0;1),\ r_l>r_{l+1}\ \ \forall l=\overline{1,L-1},\ L\geqslant P_{\max}-1. \tag{2}$$

Obviously, $n_{\min}\in\{\overline{3,n_{\max}}\}$.

First of all, a number of polygons, which are to be drawn in one image, is determined. If $L>P_{\max}-1$, then vector

$$\mathbf{R}=\left[\overline{r}_{l_*}\right]_{1\times(P_{\max}-1)} \tag{3}$$

is formed from set (2), from which $P_{\max}-1$ elements are taken out, whose indices are given by function $\pi(L,P_{\max}-1)$ returning a row vector containing a random permutation of $P_{\max}-1$ integers from 1 to L inclusive, and the permutation is sorted in descending order. So,

$$\{\overline{r}_{l_*}\}_{l_*=1}^{P_{\max}-1}\subset R,\ \overline{r}_{l_*-1}>\overline{r}_{l_*}\ \ \forall l_*=\overline{1,P_{\max}-1} \tag{4}$$

by, formally, $\overline{r}_0=1$. Otherwise, if $L=P_{\max}-1$, then

$$\mathbf{R}=\left[\overline{r}_{l_*}\right]_{1\times(P_{\max}-1)}=\left[r_l\right]_{1\times(P_{\max}-1)}. \tag{5}$$

Let θ_1 be a value of a random variable uniformly distributed on the open interval $(0;1)$. If

$$\theta_1>\overline{r}_{l_*}\ \text{ and }\ \theta_1\leqslant\overline{r}_{l_*-1} \tag{6}$$

then $P_{\max}-l_*+1$ polygons are drawn, where $l_*=\overline{1,P_{\max}-1}$. Otherwise, if

$$\theta_1\leqslant\overline{r}_{P_{\max}-1} \tag{7}$$

then just a single polygon is drawn.

An initial number of the polygon vertices is

$$n=\alpha\left(\left(n_{\max}-n_{\min}+1\right)\cdot\theta_2\right)+n_{\min}, \tag{8}$$

where value θ_2 is random generated analogously to θ_1 by a function $\alpha(\xi)$ returning the integer part of number ξ. Initially, coordinates of the vertices are taken from a vector

$$\mathbf{Y}=\left[y_i\right]_{1\times(2n)}=\alpha\left(\Theta(1,2n)\cdot\min(\gamma h,\gamma w)\right)+1, \tag{9}$$

where function $\Theta(1,2n)$ returns a pseudorandom $1\times(2n)$ vector whose entries are drawn from the standard uniform distribution on the open interval $(0;1)$, and $\gamma=s_{\pi(K,1)}$ is a coefficient to scale the polygon with respect to the image size. In this scale factor, $\pi(K,1)$ is a random integer between 1 and K to choose one of elements in set (1). The horizontal coordinates are

$$z_i=y_i+\alpha\left(\left(1-\gamma\right)\cdot h\theta_3\right)+1\ \text{ for }\ i=\overline{1,n} \tag{10}$$

and the vertical coordinates are

$$z_{n+i}=y_{n+i}+\alpha\left(\left(1-\gamma\right)\cdot w\theta_4\right)+1\ \text{ for }\ i=\overline{1,n}, \tag{11}$$

where values θ_3 and θ_4 are random generated analogously to θ_1. So, the i-th vertex is a plane point $\left[z_i\ \ z_{n+i}\right]$ for $i=\overline{1,n}$.

Generation of vector (9) and coordinates (10) and (11) is repeated until the resulting polygon becomes convex. For obtaining the convexity faster, one or a few vertices may be deleted. Sets of values $\{z_i\}_{i=1}^{n}$ and $\{z_{n+i}\}_{i=1}^{n}$ are sorted in ascending order into sets $\{\overline{z}_i\}_{i=1}^{n}$ and $\{\overline{z}_{n+i}\}_{i=1}^{n}$, respectively. Along with that, for a given positive integer λ, inequalities

$$\overline{z}_i-\overline{z}_{i-1}\geqslant\lambda\ \ \text{for}\ \ i=\overline{2,n} \tag{12}$$

and

$$\overline{z}_{n+j}-\overline{z}_{n+j-1}\geqslant\lambda\ \ \text{for}\ \ j=\overline{2,n} \tag{13}$$

help in controlling that the coordinates of the same axis are not too close. In practice, value $\lambda\in\{\overline{5,10}\}$ is acceptable. Nonetheless, the requirement of inequalities (12) and (13) can be relaxed so that one of them or both may be violated for a single $i\in\{\overline{2,n}\}$ and a single $j\in\{\overline{2,n}\}$.

Such a dataset generator does not necessarily return a polygon having at least $n_{\min}$ vertices (or edges). While searching for the convex hull by the given coordinates, those vertices that violate the convexity are deleted. This decreases the factual number of polygon edges, especially for small-sized images and/or by small scale factors. For an increased $n_{\min}$, as the image size increases, the generator produces polygons having a greater number of edges with greater probability.

Given a number N of dataset entries, the dataset generator produces N images with polygonal objects along with respective N labeled images. The dataset represents two classes – "polygonal object/objects" (shortly, "polygon") and "background". The labeled image is almost a negative of the original polygonal image (see an example in Fig. 6). The matter is that when a few polygons intersect, the non-convex object interior in the labeled image does not have "inner" lines. These "inner" lines, invisible for an SSN while training, serve as an additional impediment to improve generalization.

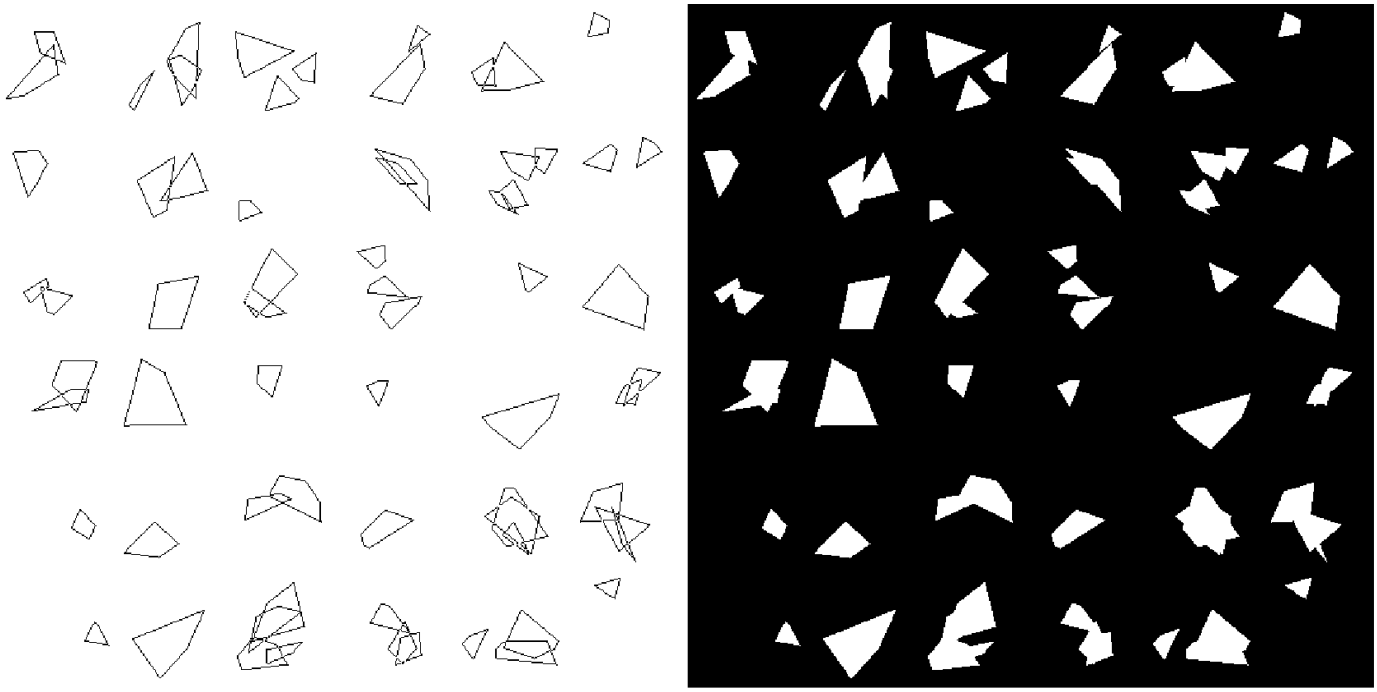

Fig. 6. A toy dataset of 36 polygonal 96×96 images in a stack and their respective labeled images. The images are generated by $n_{\min}=4$, $n_{\max}=6$, $S=\{0.3+0.1\cdot(k-1)\}_{k=1}^{5}$, $R=\{0.9-0.1\cdot(l-1)\}_{l=1}^{9}$, $P_{\max}=4$. None of 2 images, whose polygonal objects are made of 4 convex polygons, factually (in their labels) have 4 convex polygons because they intersect. One of those images contains 2 objects; another one contains just a single object of an intricate shape (the third from the left in the bottom row). There are only 2 images having 3 polygonal objects (which occasionally appear to be convex due to no intersections). Eventually, 16 images are made of one polygon. So, a rate of randomness here is satisfactory, although the 6×6 lattice of polygonal objects can be marked out.

The toy dataset whose entries are generated by the algorithm of (1) – (13) has its own rate of randomness. It shows the dataset complexity. The rate of randomness is complex itself implying the following properties:

1) how scattered and sparse images appear in a stack (like in Fig. 6);
2) how many objects in an image are scattered (for images containing multiple objects);
3) how sophisticated an object of intersecting polygons appear (whether it is single or there are a few such objects in an image);
4) what the range of object's size is (difference between the tiniest and biggest objects).

In a way, another characteristic of the randomness is whether it is easy to see a lattice in a stack of images. This property is controlled by changing sets (1) and (2). Namely, the greater number of scale factors, independently of set (2), makes a rectangular stack of dataset images appear more scattered (Fig. 7).

Eventually, when a toy dataset is generated, it is divided into a training set, a validation set, and a testing set. The division is fulfilled without choosing random indices as the dataset entries are already randomly indexed. The percentage of each set type is determined by peculiarities of a SIS task, but it is not essential owing to the fact that the dataset generator produces as many images as needed without any restrictions.

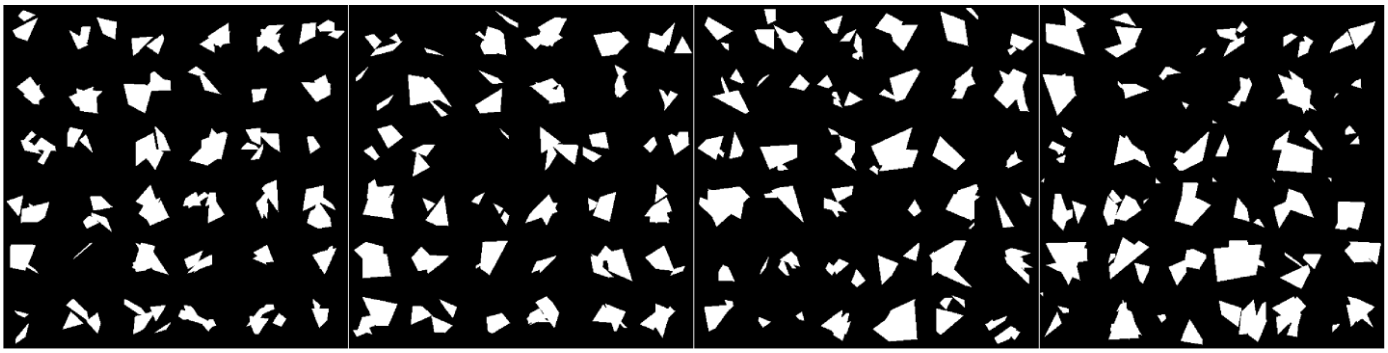

Fig. 7. An example of four datasets stacked in a row, where every dataset is of 36 images stacked as a 6×6 lattice. The set of scale factors has more elements and a wider interval of their values moving from left to right. Set (2) is of three elements: 0.75, 0.5, 0.25 (from 1 to 4 polygons are produced by equal probabilities in any image). Whereas the 6×6 lattice of polygonal objects can be easily marked out in the dataset on the left, such lattice is not so apparent in the dataset on the right. Difference between the tiniest and biggest objects in the dataset on the right is the greatest. Moreover, here images appear the most scattered and sparse. Some objects have pretty intricate shapes but similar intricacy can be found in three other datasets (from the left).

VII. Examples of Training an SSN on the Toy Dataset

For illustration purposes, let a toy dataset be generated by $n_{\min}=3$, $n_{\max}=8$, $P_{\max}=4$ and the following sets (1) and (2):

$$S=\{s_k\}_{k=1}^{7},\ s_1=0.2,\ s_{k+1}=s_k+0.01\cdot(k+2)\ \ \forall k=\overline{1,6}, \quad (14)$$

$$R=\{r_l\}_{l=1}^{14},\ r_1=0.85,\ r_{l+1}=r_l-\frac{l+1}{(2l+1)^2}\ \ \forall l=\overline{1,13}. \quad (15)$$

Let 2000 medium-sized square images be generated. So, $N=2000$, $h=w=64$. This is expected to be sufficient to obtain consistent and reliable results. The first 10 images generated by the mentioned parameters and by sets (14) and (15) are shown along with their labels in Fig. 8.

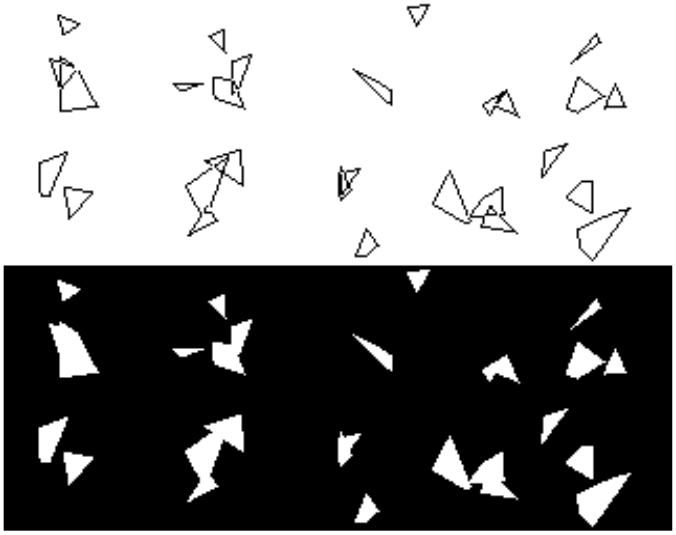

Fig. 8. A stack of 10 images and their labels (only for training) of the dataset of 64×64 images generated by $n_{\min}=3$, $n_{\max}=8$, $P_{\max}=4$, and (14), (15).

For validation and testing, it is sufficient to use 10 % of the dataset, which is 200 images. We can start from a simpler SSN whose architecture is similar to that in Fig. 4. At the first stage, this SSN (Fig. 9) is trained for 120 epochs by the initial learning rate of 0.001 and the learning rate drop factor of 0.975 multiplied by the learning rate after every 10 epochs. Then, at the second stage, the SSN is trained for additional 80 epochs if the SSN does not start segmenting the empty image (in which class "background" exists only). Nevertheless, the accuracy of the SSN in Fig. 9 is pretty low (Fig. 10). In general, it "captures" multi-polygon objects, but the final segmentation looks sloppy. So, the SSN architecture is made deeper by inserting one ConvL and one DeConvL (Fig. 11).

```
 1 'imageinput'    Image Input
                   64x64x1 images with 'zerocenter' normalization
 2 'conv_1'        Convolution
                   128 3x3x1 convolutions with stride [1  1]
                   and padding [1  1  1  1]
 3 'relu_1'        ReLU
 4 'maxpool'       Max Pooling
                   2x2 max pooling with stride [2  2]
                   and padding [0  0  0  0]
 5 'conv_2'        Convolution
                   128 3x3x128 convolutions with stride [1  1]
                   and padding [1  1  1  1]
 6 'relu_2'        ReLU
 7 'deconv'        Transposed Convolution
                   128 4x4x128 transposed convolutions
                   with stride [2  2] and output cropping [1  1]
 8 'conv_3'        Convolution
                   2 1x1x128 convolutions with stride [1  1]
                   and padding [0  0  0  0]
 9 'softmax'       Softmax
10 'classoutput'   Pixel Classification Layer
                   Class weighted cross-entropy loss
                   with classes 'triangle' and 'background'
```

Fig. 9. An SSN architecture for segmenting 64×64 images (Fig. 8). This SSN is a replica of that in Fig. 4, wherein the numbers of convolutions and deconvolutions are just twice increased (as the input size is twice increased).

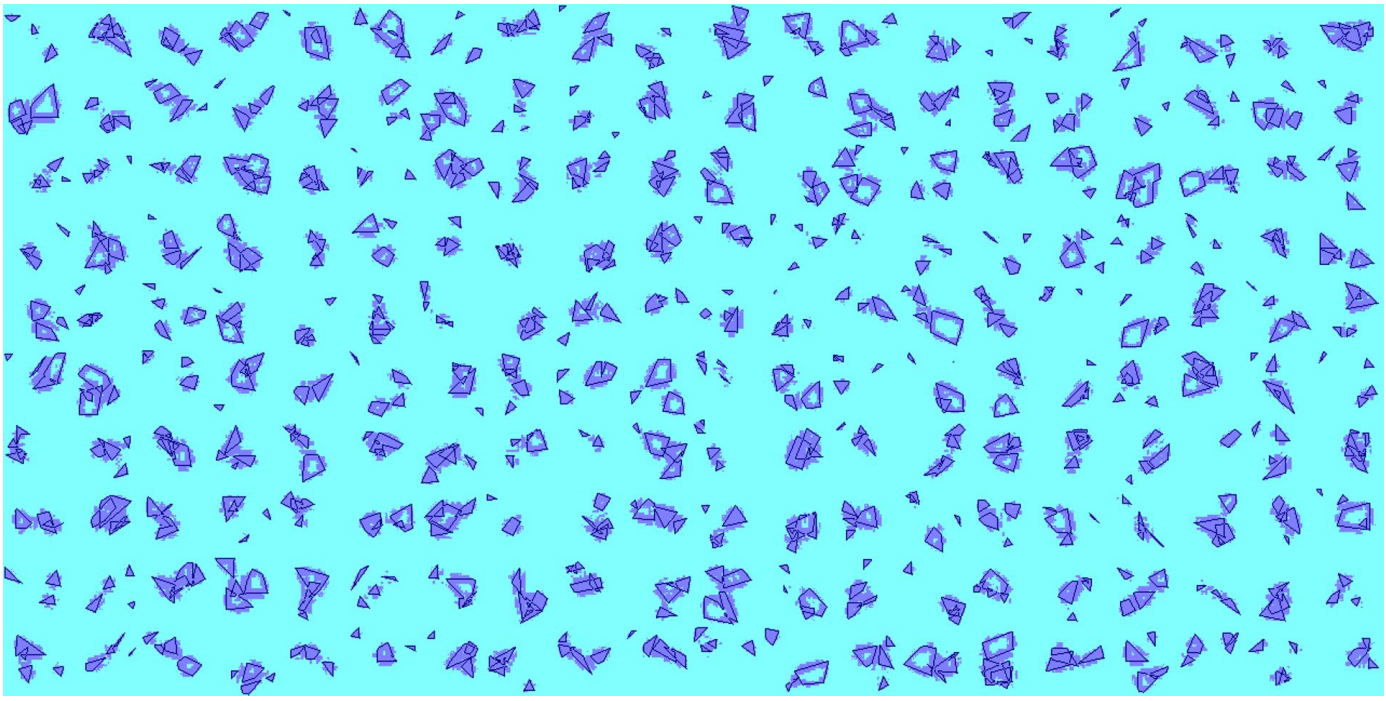

Fig. 10. A stack of 200 images with the segmentation result by the trained SSN (Fig. 9). The weighted IoU [15] is 0.88, whereas the mean IoU is just 0.75402.

```
 1 'imageinput'    Image Input
                   64x64x1 images with 'zerocenter' normalization
 2 'conv_1'        Convolution
                   128 3x3x1 convolutions with stride [1  1]
                   and padding [1  1  1  1]
 3 'relu_1'        ReLU
 4 'maxpool_1'     Max Pooling
                   2x2 max pooling with stride [2  2]
                   and padding [0  0  0  0]
 5 'conv_2'        Convolution
                   128 3x3x128 convolutions with stride [1  1]
                   and padding [1  1  1  1]
 6 'relu_2'        ReLU
 7 'maxpool_2'     Max Pooling
                   2x2 max pooling with stride [2  2]
                   and padding [0  0  0  0]
 8 'conv_3'        Convolution
                   128 3x3x128 convolutions with stride [1  1]
                   and padding [1  1  1  1]
 9 'relu_3'        ReLU
10 'deconv_1'      Transposed Convolution
                   128 4x4x128 transposed convolutions
                   with stride [2  2] and output cropping [1  1]
11 'deconv_2'      Transposed Convolution
                   128 4x4x128 transposed convolutions
                   with stride [2  2] and output cropping [1  1]
12 'conv_4'        Convolution
                   2 1x1x128 convolutions with stride [1  1]
                   and padding [0  0  0  0]
13 'softmax'       Softmax
14 'classoutput'   Pixel Classification Layer
                   Class weighted cross-entropy loss
                   with classes 'triangle' and 'background'
```

Fig. 11. A deeper SSN architecture in MATLAB for segmenting 64×64 images (Fig. 8). Here, one more ConvL and one DeConvL have been inserted.

The deeper SSN trained by the same parameters for those two stages performs much better (Fig. 12). Multi-polygon objects are segmented accurately now, although some sloppy segmentation results are still observed, where background pixels in the neighborhood of the object are highlighted. Besides, there are a few objects, whose interiors are not fully highlighted. Such a poor one-image segmentation result is compared to the best one in Fig. 13 (8-bit JPEG format conversion is seen).

These examples show that the deeper SSN has successfully "inherited" the architecture of simpler SSN. Further optimization is surely possible by manipulating with the numbers of convolutions and deconvolutions. As the optimization of the SSN is completed, the toy dataset generation should be made more complicated, step-by-step moving towards the most complicated SSN for such datasets, where the image size can be set to roughly the same values as the objects to be segmented for a real-world SIS task. Eventually, an initial SSN for a real-world SIS task will be that most complicated SSN optimized for the most complicated toy dataset. The initial SSN will be further fine-tuned by just manipulating with (mainly, increasing) the numbers of convolutions and/or deconvolutions, without changing the network layers. The second training stage plays a quite important role here. With an appropriate maximal number of additional epochs $c_{\max}$, the algorithm of this stage is as follows:

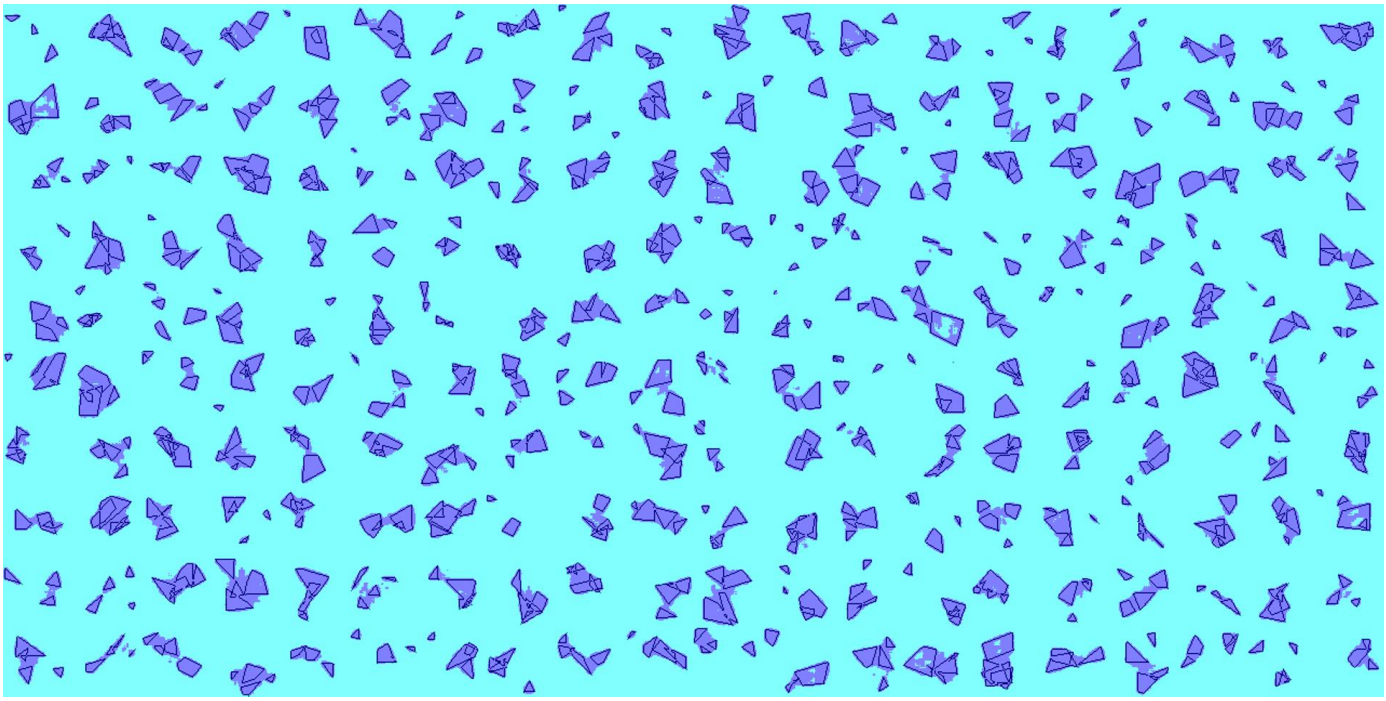

Fig. 12. The stack of 200 images with the segmentation result by the trained SSN (Fig. 11). The weighted and mean IoUs are 0.91331 and 0.81367, respectively.

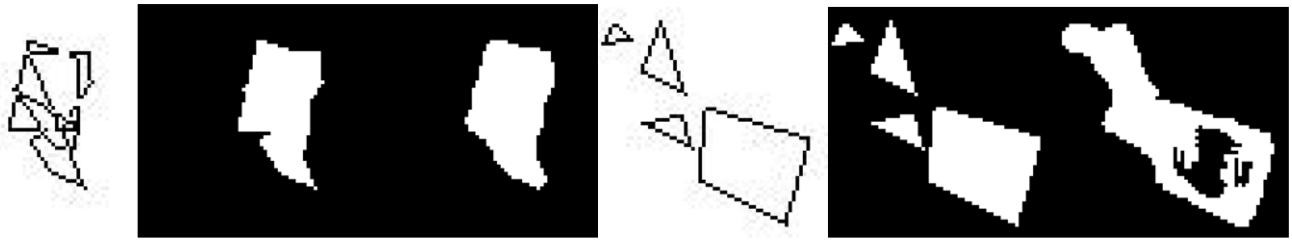

Fig. 13. The best and worst cases of segmentation (the test image versus truth and segmented images) from Fig. 12 by IoUs equal to 0.93 and 0.67, respectively.

1. Train for some number of epochs (the first stage);
2. Set the current number of additional epochs to zero ($c_{\text{AE}} = 0$);
3. While the number of segmented pixels in the empty image for class "polygon" is zero and $c_{\text{AE}} < c_{\max}$ do:
   3.1. Drop the learning rate after every new g_{LRD} epochs;
   3.2. Train;
   3.3. Increase c_{AE} by 1.

In the examples above, $g_{\text{LRD}} = 10$ but integer g_{LRD}, generally speaking, can be non-constant. Surely, the other training parameters may be slightly adjusted as well [9], [16].

VIII. Discussion

Obviously, it is very likely that the adjustment of an SSN for a real-world SIS task will be needed anyway. Nevertheless, "playing" with the toy dataset allows significantly reducing the volume of such an adjustment. Apart from the faster training, a toy-dataset SSN is tested a way faster: those exemplary SSNs occupy only 1.48 MB (Fig. 9) and 2.93 MB (Fig. 11), respectively, that makes them rapid.

It should be noted that the SSN performance on the test set can be estimated by using minimal IoUs, without referring to the topmost IoUs. It is believed so because the main problem seen in Fig. 12 is those interiors not fully highlighted, which are characterized by low IoUs. Such interiors, by the way, spring from objects composed of bigger polygons. Another open question is how to reduce highlighting outside polygons. Similarly to missed internal regions, this question is addressed to bigger objects, composed of more than a single polygon (a few such cases are easily seen in Fig. 12).

IX. Conclusion

The created generator for toy datasets is a simple mathematical object whose complexity is regulated by the number of edges in a polygon, the maximal number of polygons in one image, the set of scale factors, and the set of probabilities determining how many polygons in a current image are generated. The dataset capacity and image size are concurrently adjustable, although they are much less influential.

The toy dataset generator is a convenient tool for prototyping SSNs capable of segmenting real-world (complex) objects. In particular, the complex object can be an aggregate of landscape scenes (Fig. 2). The generator works best for a two-class SIS. However, it can be used for prototyping a more complex toy dataset generator fitting multi-class SIS.

References

[1] J. Rogowska, "Overview and Fundamentals of Medical Image Segmentation," in: *Handbook of Medical Image Processing and Analysis, 2nd edition*, Bankman I. N. (ed.). Academic Press, San Diego, 2009, pp. 73–90. https://doi.org/10.1016/B978-012373904-9.50013-1

[2] H.-J. He, C. Zheng, and D.-W. Sun, "Image Segmentation Techniques," in: *Computer Vision Technology for Food Quality Evaluation, 2nd edition*, Sun D.-W. (ed.). Academic Press, San Diego, 2016, pp. 45–63. https://doi.org/10.1016/B978-0-12-802232-0.00002-5

[3] Ç. Kaymak and A. Uçar, "A Brief Survey and an Application of Semantic Image Segmentation for Autonomous Driving," in: *Handbook of Deep Learning Applications. Smart Innovation, Systems and Technologies*, Balas V., Roy S., Sharma D., Samui P. (eds). Springer, Cham, 2019, pp. 161–198. https://doi.org/10.1007/978-3-030-11479-4_9

[4] J.-T. Chien, "Deep Neural Network," in: *Source Separation and Machine Learning*, Chien J.-T. (ed.). Academic Press, 2019, pp. 259–320. https://doi.org/10.1016/B978-0-12-804566-4.00019-X

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. https://doi.org/10.1109/TPAMI.2016.2644615

[6] H. Liu, J. Xu, Y. Wu, Q. Guo, B. Ibragimov, and L. Xing, "Learning deconvolutional deep neural network for high resolution medical image reconstruction," *Information Sciences*, vol. 468, pp. 142–154, 2018. https://doi.org/10.1016/j.ins.2018.08.022

[7] G. Neuhold, T. Ollmann, S. R. Bulò, and P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes," *2017 IEEE International Conference on Computer Vision*, Venice, 2017, pp. 5000–5009. https://doi.org/10.1109/ICCV.2017.534

[8] J.-J. Lv, X.-H. Shao, J.-S. Huang, X.-D. Zhou, and X. Zhou, "Data augmentation for face recognition," *Neurocomputing*, vol. 230, pp. 184–196, 2017. https://doi.org/10.1016/j.neucom.2016.12.025

[9] V. Romanuke, "Optimal training parameters and hidden layer neuron number of two-layer perceptron for generalised scaled object classification problem," *Information Technology and Management Science*, vol. 18, no. 1, pp. 42– 48, 2015. https://doi.org/10.1515/itms-2015-0007

[10] H. Hofbauer, E. Jalilian, and A. Uhl, "Exploiting superior CNN-based iris segmentation for better recognition accuracy," *Pattern Recognition Letters*, vol. 120, pp. 17–23, 2019. https://doi.org/10.1016/j.patrec.2018.12.021

[11] V. V. Romanuke, "Appropriateness of DropOut layers and allocation of their 0.5 rates across convolutional neural networks for CIFAR-10, EEACL26, and NORB datasets," *Applied Computer Systems*, vol. 22, no. 1, pp. 54–63, 2017. https://doi.org/10.1515/acss-2017-0018

[12] V. V. Romanuke, "An attempt of finding an appropriate number of convolutional layers in CNNs based on benchmarks of heterogeneous datasets," *Electrical, Control and Communication Engineering*, vol. 14, no. 1, pp. 51–57, 2018. https://doi.org/10.2478/ecce-2018-0006

[13] D. Avis, H. ElGindy, and R. Seidel, "Simple On-Line Algorithms for Convex Polygons", in: *Machine Intelligence and Pattern Recognition (vol. 2)*, Toussaint G. T. (ed.). North-Holland, 1985, pp. 23–42. https://doi.org/10.1016/B978-0-444-87806-9.50007-4

[14] E. Horowitz and M. Papa, "Polygon Clipping: Analysis and Experiences", in: *Theoretical Studies in Computer Science*, Ullman J. D. (ed.). Academic Press, 1992, pp. 315–339. https://doi.org/10.1016/B978-0-12-708240-0.50016-2

[15] M. A. Rahman and Y. Wang, "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation", in: *Advances in Visual Computing (ISVC 2016)*, Bebis G. et al. (eds). Springer, Cham, 2016, pp. 234–244. https://doi.org/10.1007/978-3-319-50835-1_22

[16] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20–24, 2017. https://doi.org/10.1515/itms-2017-0003

Vadim V. Romanuke was born in 1979 and received higher education in 2001. In 2006, he received the Degree of the Candidate of Technical Sciences in Mathematical Modeling and Computational Methods. The Candidate Dissertation suggested a way of increasing interference noise immunity of data transferred over radio systems. The degree of Doctor of Technical Sciences in Mathematical Modeling and Computational Methods was received in 2014. The Doctor-of-Science Dissertation solved a problem of increasing efficiency of identification of models for multistage technical control and run-in under multivariate uncertainties of their parameters and relationships. In 2016, he received the status of a Full Professor.

He is a Professor at the Faculty of Navigation and Naval Weapons of the Polish Naval Academy. His research interests concern decision making, game theory, statistical approximation, job scheduling, and control engineering based on statistical correspondence. Vadim Romanuke has good programming skills in MATLAB. For practical implementations, Vadim Romanuke uses Python.

Address for correspondence: 69 Śmidowicza Street, Gdynia, Poland, 81-127.
E-mail: romanukevadimv@gmail.com
ORCID iD: https://orcid.org/0000-0003-3543-3087