Fractal Distribution of Medical Data in Neural Network

Nataliya Boyko[0000-0002-6962-9363], Maxim Kuba[0000-0002-7394-5764],

Lesia Mochurad [0000-0002-4957-1512], Sergio Montenegro[0000-0002-0636-5866]

Lviv Polytechnic National University, Lviv 79013, Ukraine
Julius-Maximilians-University Würzburg, Am Hubland, D-97074 Würzburg, German
nataliya.i.boyko@lpnu.ua
maxim.kuba@gmail.com
lesiamochurad@gmail.com
sergio.montenegro@uni-wuerzburg.de

Abstract. Nowadays the topic of deep learning is becoming more and more popular. Moreover, almost every organization want to have at least one specialist in this area, because artificial intelligence can help your medicine to grow and to increase its productivity. Research of one of the types of neural network – fractal neural network. Testing and comparing with other neural networks. We will take one dataset and test it on our neural networks and then compare the results. Trained and tested neural networks with graphs and comparisons of their output. In the current paper we implemented custom neural network and fractal neural network. Then we trained and tested them on CIFAR-10 dataset. Custom neural network showed us worse results, but each iteration took up to 10 seconds, when 1 iteration of fractal neural network took up to 3 minutes. Moreover, our network is pretty simple, so we can say that that is suits better for datasets with lower quantity of classes. Fractal neural network showed us pretty good results, but I am sure that with more powerful computing resources and more time it can perform much better.

Keywords: neural networks; model; medical data; keras; train; dataset; accuracy, loss.

1 Introduction

In the current paper we want to make a research about one part of deep learning – fractal neural networks. A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus, a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1. There are many types of neural networks, and residual neural network is one of them [1, 9].

A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or short-cuts to jump over some layers. Typical ResNet models are implemented with double- or triple-layer skips that contain nonlinearities (ReLu) and batch normalization in between. An additional weight matrix may be used to learn the skip weights; these models are known as HighwayNets. Models with several parallel skips are referred to as DenseNets. In the context of residual neural networks, a non-residual network may be described as a plain network.

One motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights. During training, the weights adapt to mute the upstream layer, and amplify the previously-skipped layer. In the simplest case, only the weights for the adjacent layer's connection are adapted, with no explicit weights for the upstream layer. This works best when a single non-linear layer is stepped over, or when the intermediate layers are all linear. If not, then an explicit weight matrix should be learned for the skipped connection (a HighwayNet should be used) [1-4, 6].

Fractal neural network uses non-residual network approach. Macro-architecture of fractal neural networks is based on self-similarity. Repeated application of a simple expansion rule generates deep networks whose structural layouts are precisely truncated fractals. These networks contain interacting subpath of different lengths, but do not include any pass-through or residual connections; every internal signal is transformed by a filter and nonlinearity before being seen by subsequent layers. The key may be the ability to transition, during training, from effectively shallow to deep. Additionally, fractal networks exhibit an anytime property: shallow subnetworks provide a quick answer, while deeper subnetworks, with higher latency, provide a more accurate answer [3].

2 Review of the Literature

Fractal neural networks are relatively new, that is why there are only a few articles on this theme. Frankly speaking, there is only one brief and complex paper about Fractal neural networks. It was published at ICLR 2017 as a conference paper by Gustav Larsson, Michael Maire and Gregory Shakhanaovich [11]. Their paper is called “FractalNet: Ultra-Deep Neural Networks without Residuals”. They briefly describe fractal neural networks and how do they work. Also, they compare the results of this network with more than 20 other networks on about 10 different datasets. They published code for FractalNet implementation which weare going to update and use in current paper. So, their paper is very useful, full of important information. They have very powerful computing resources, which helps them to train and test networks on a different data for a long time.

3 Materials and Methods

In order to implement and run our networks we will use Python 3 and Google Collaboratory as our working environment.

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser. Also,it provides good GPU in order to operate our networks [4, 12].

For training and testing we pick CIFAR10 dataset from Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research [5, 10].

The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class [6].

Computer algorithms for recognizing objects in photos often learn by example. CIFAR-10 is a set of images that can be used to teach computer how to recognize objects. Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms to see what works. Various kinds of convolutional neural networks tend to be the best at recognizing the images in CIFAR-10.

In order to implement our Sequential model we will use the following layers and functions:

1) ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as y = max(0, x). ReLU is linear (identity) for all positive values, and zero for all negative values. This means that [7]:

It's cheap to compute as there is no complicated math. The model can therefore take less time to train or run.

It converges faster. Linearity means that the slope doesn't plateau, or "saturate," when x gets large. It doesn't have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh.

It's sparsely activated. Since ReLU is zero for all negative inputs, it's likely for any given unit to not activate at all.

2) Softmax is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1, but after applying softmax, each component will be in interval(0,1),and the components will add up to 1, so that they can be interpreted as probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes[8, 10].

3) Dropout is a regularization technique for neural network models. Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations [9, 17]

4) Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation[10, 15].

Also,we will use the optimization algorithms described below:

1) The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster[11, 12].

2) Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients mt, similar to momentum and also the decaying average of the past squared gradients vt, similar to RMSprop and Adadelta. Thus, it combines the advantages of both the methods. Adam is the default choice of the optimizer for any application in general [11, 13].

4 Experiment

So, for training our networks, we chose CIFAR10 dataset. We will train our network to classify 10 different objects: doctor, patient, disease, mode, ward, hospital, surgery, tablet, syringe, prescription. The classes are completely mutually exclusive. There is no overlapping between classes. This means that you will, not find an image with 2 different classes at the same time.

This means, that we could apply our network to solve different medical problems. For example, it can be helpful for predicting diagnosis relying on the cardiogram.

We will make custom sequential model for comparing with fractal one. Sequential model is simply a linear stack of layers. So, you can just create an empty model, and then add as many layers as you want. In this model we add few activation layers, connection layers, regularization layers, convolutional layers, pooling layers. Here is our final version

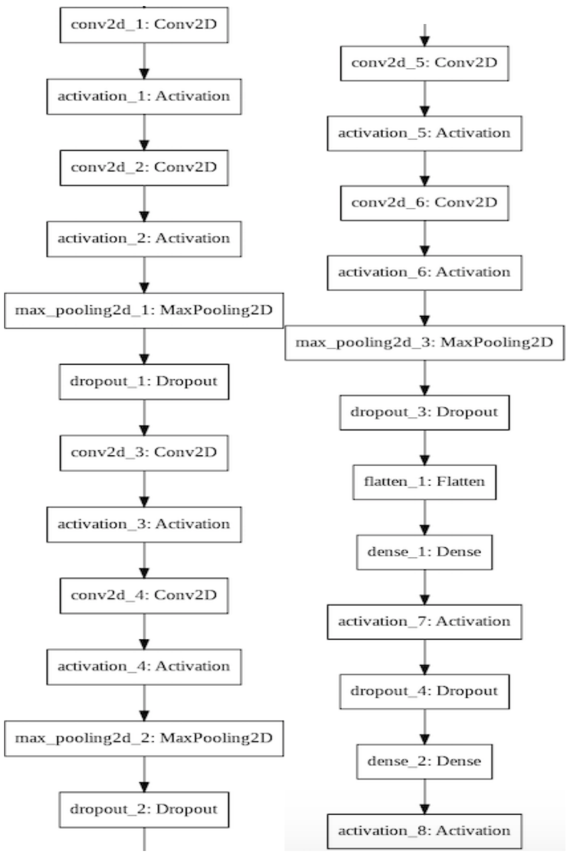


Fig. 1. Final version model.add

Now it is time to train our network. In order to do this, we will iterate over our dataset (which contains 50000 images) 200 times (epochs) and teach our network. We will pass the image and regardless to prediction of our network correct the matrix of the weight in order to get better accuracy.

On the picture below you can see a small piece of our training process, which outputs the result after each epoch (Fig. 2).

```
Epoch 57/70
50000/50000 [======]12s 232us/step – loss: 0.5535 – acc: 0.8144 – val_loss: 0.6030 – val_acc: 0.7995
Epoch 58/70
50000/50000 [=====]12s 234us/step – loss: 0.5535 – acc: 0.8144 – val_loss: 0.6030 – val_acc: 0.7995
Epoch 59/70
50000/50000 [=====]12s 234us/step – loss: 0.5535 – acc: 0.8144 – val_loss: 0.6030 – val_acc: 0.7995
Epoch 60/70
50000/50000 [=====]12s 232us/step – loss: 0.5535 – acc: 0.8144 – val_loss: 0.6030 – val_acc: 0.7995
Epoch 61/70
50000/50000 [=====]12s 233us/step – loss: 0.5535 – acc: 0.8144 – val_loss: 0.6030 – val_acc: 0.7995
```

Fig. 2. Training process

Table 1. Training process.

Epoch	Loss	Accuracy	Epoch	Loss	Accuracy
1	1.9679	0.2735	36	0.6250	0.7914
2	1.5706	0.4101	37	0.6452	0.7824
3	1.4189	0.4774	38	0.6336	0.7900
4	1.2955	0.5239	39	0.6172	0.7879
5	1.2336	0.5527	40	0.6663	0.7797
6	1.2376	0.5506	41	0.6854	0.7718
7	1.1391	0.5954	42	0.6731	0.7751
8	1.0698	0.6177	43	0.6089	0.7985
9	1.1180	0.6023	44	0.6342	0.7872
10	0.9758	0.6512	45	0.6001	0.8020
11	0.9569	0.6608	46	0.6475	0.7883
12	0.9204	0.6739	47	0.6988	0.7812
13	0.9549	0.6634	48	0.6283	0.7920
14	0.8803	0.6881	49	0.6131	0.7992
15	0.8783	0.6897	50	0.6975	0.7771
16	0.8434	0.7002	51	0.6001	0.7994
17	0.8100	0.7149	52	0.6080	0.7946
18	0.8215	0.7104	53	0.5819	0.8074
19	0.8707	0.6978	54	0.6156	0.7963
20	0.7756	0.7322	55	0.5921	0.8055
21	0.7524	0.7425	56	0.6101	0.8031
22	0.7338	0.7416	57	0.6030	0.7995
23	0.7304	0.7481	58	0.6311	0.7939
24	0.7347	0.7471	59	0.6152	0.7958
25	0.6974	0.7580	60	0.6128	0.7972
26	0.7043	0.7576	61	0.6032	0.8007
27	0.6685	0.7675	62	0.6045	0.8018
28	0.6875	0.7655	63	0.6053	0.8033
29	0.7100	0.7567	64	0.6095	0.8017
30	0.6677	0.7714	65	0.5860	0.8075
31	0.6982	0.7617	66	0.6370	0.7964
32	0.6558	0.7782	67	0.6523	0.7887
33	0.6424	0.7838	68	0.6478	0.7870
34	0.6637	0.7792	69	0.5904	0.8111
35	0.6663	0.7776	70	0.6472	0.7929

In the table above you can see the full training process with its accuracy and loss at each step of the training. The best results are highlighted

Also, on the following graphs (Fig. 3, 4) you can see a dependency of accuracy and loss according to epochs. Accuracy is calculated as the amount of right predictions divided by all predictions.

So, from the graph we can see the logarithmic increase of accuracy. Also, we can notice optimal amount of training after which the accuracy increases very slightly.

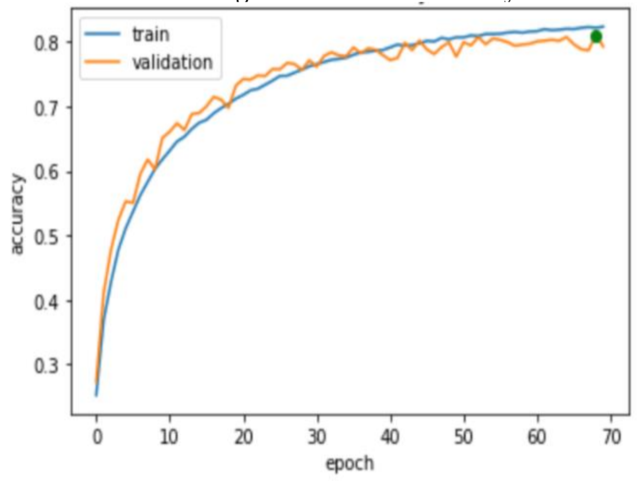


Fig. 3. Model accuracy

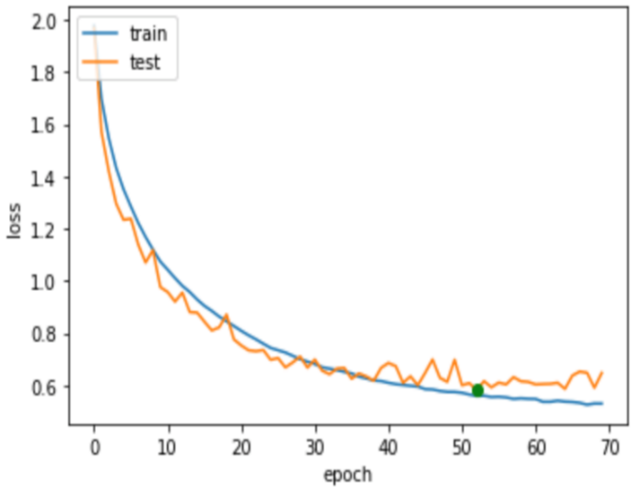


Fig. 4. Model loss

Model for fractal neural network is much more complicated than our custom model. It has much more layers and much more configurations. The full implementation of the fractal neural network model could be found by the link in references [9]. It was published with a paper at ICLR 2017 by Gustav Larsson, Michael Maire and Gregory Shakhanaovich, as I mentioned in literature review section [11].

Now let us train this network the same way as we did with our custom network. This time we will make 70 epochs, because training fractal network takes more time and computing resources. Below you can see a piece of our training process (Fig. 5).

```
Epoch 37 /70
50000/50000 [=====]116s 2ms/step – loss: 0.0804 – acc: 0.9761 – val_loss: 0.5396 – val_acc: 0.8424
Epoch 38/70
50000/50000 [=====]116s 2ms/step – loss: 0.0777 – acc: 0.9761 – val_loss: 0.5090 – val_acc: 0.8492
Epoch 39/70
50000/50000 [=====]116s 2ms/step – loss: 0.0820 – acc: 0.9750 – val_loss: 0.4709 – val_acc: 0.8541
Epoch 40/70
50000/50000 [=====]116s 2ms/step – loss: 0.0735 – acc: 0.9777 – val_loss: 0.4540 – val_acc: 0.8630
Epoch 41/70
50000/50000 [=====]116s 2ms/step – loss: 0.0728 – acc: 0.9783 – val_loss: 0.4439 – val_acc: 0.8691
```

Fig. 5. Training proccess

In the table 2 you can see the full training process with its accuracy and loss at each step of the training. Best results are highlighted.

Table 2. Training process.

Epoch	Loss	Accuracy	Epoch	Loss	Accuracy
1	2.3521	0.1087	36	0.4375	0.8663
2	2.0765	0.1944	37	0.5396	0.8424
3	2.2318	0.2154	38	0.5090	0.8492
4	1.8486	0.3299	39	0.4709	0.8541
5	1.8550	0.3061	40	0.4540	0.8630
6	1.1768	0.6020	41	0.4439	0.8691
7	1.1903	0.6307	42	0.4675	0.8657
8	0.9979	0.7036	43	0.4817	0.8542
9	0.9514	0.6841	44	0.4516	0.8664
10	1.0341	0.6793	45	0.4493	0.8693
11	0.7769	0.7512	46	0.4520	0.8593
12	0.7874	0.7646	47	0.4981	0.8587
13	0.8565	0.7197	48	0.4356	0.8733
14	0.6864	0.7828	49	0.5158	0.8460
15	0.6245	0.8135	50	0.4157	0.8776
16	0.7089	0.7909	51	0.4671	0.8626
17	0.6684	0.7949	52	0.4642	0.8597
18	0.5995	0.8228	53	0.4100	0.8788
19	0.7498	0.7592	54	0.6282	0.8223

20	0.5802	0.8037	55	0.4664	0.8606
21	0.6381	0.7965	56	0.5437	0.8354
22	0.5384	0.8351	57	0.4502	0.8721
23	0.5344	0.8379	58	0.4535	0.8651
24	0.4795	0.8506	59	0.5082	0.8502
25	0.6395	0.7895	60	0.4246	0.8735
26	0.6320	0.8028	61	0.4654	0.8576
27	0.5624	0.8226	62	0.3877	0.8834
28	0.5276	0.8393	63	0.4217	0.8680
29	0.5183	0.8395	64	0.4337	0.8758
30	0.6432	0.7951	65	0.4299	0.8717
31	0.5376	0.8188	66	0.4545	0.8721
32	0.4631	0.8612	67	0.4193	0.8697
33	0.5345	0.8393	68	0.4392	0.8680
34	0.4406	0.8676	69	0.4547	0.8690
35	0.4629	0.8556	70	0.3922	0.8864

Best results are marked with green color.

Also, on the following graphs (Fig.6, 7) you can see a dependency of accuracy and loss according to epochs. As with our custom network accuracy is calculated as the amount of right predictions divided by all predictions. So, from the graph we can see the logarithmic increase of accuracy. Also, we can notice optimal amount of training after which the accuracy in creases very slightly. So, it looks similar to our custom neural network graph, but as we see, that the accuracy here is better.

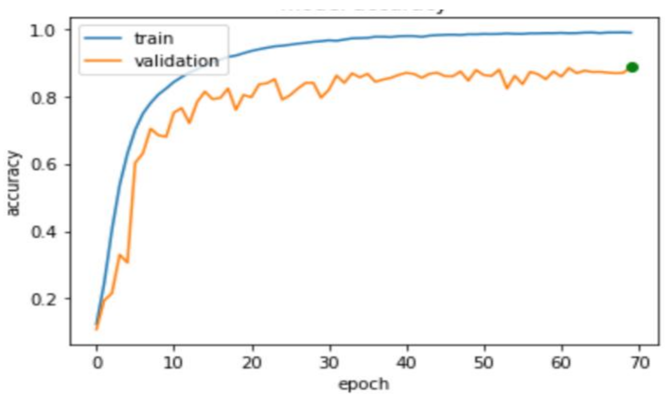


Fig. 6. Model accuracy

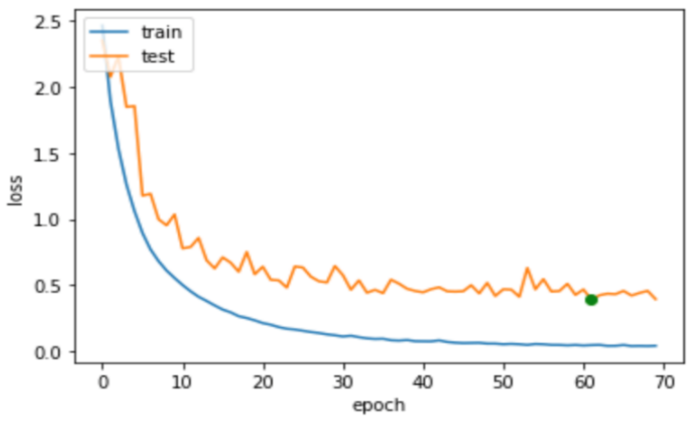
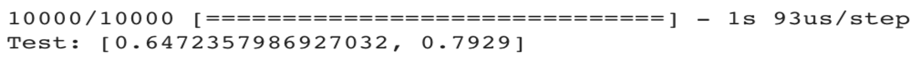


Fig. 7. Model loss

5 Results

Now it is time to test our trained models on a test dataset. It is the set of images which haven't been used during training process. The process is similar, but we iterate through our dataset only once, and output the results immediately. The results for our custom network are the following (Fig. 8).

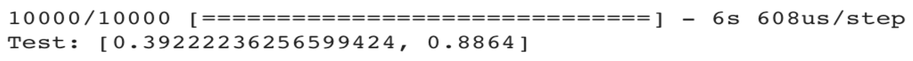


```
10000/10000 [==============================] - 1s 93us/step
Test: [0.6472357986927032, 0.7929]
```

Fig. 8. Test results

This test showed us 0.7929 accuracy, which means that from 10000 labeled images with 10 different object classes our network predicted 7929 images right and 2071 images wrong. Our test accuracy become lower than the training one (8.111), which means that we overfit our model on a train dataset a little bit. It means that our weight fits a bit better for our train dataset. Lowering the training time may improve our accuracy a little bit.

Now let us head back to our fractal network. Our best accuracy was achieved at the very the end of epochs, which means, that further training may lead to better results. But it will take more time and more computing resources. Our accuracy is pretty good, but first let us test in on test data set and check if we didn't overfit our network (Fig. 9).



```
10000/10000 [==============================] - 6s 608us/step
Test: [0.39222236256599424, 0.8864]
```

Fig. 9. Test results

This test showed us 0.8864 accuracy,which means that from 10000 labeled images with 10 different object classes our network predicted 8864 images right and 1136 images wrong.Our test accuracy is the same as train one(0.8864),which means that we didn’t overfit our model on a train dataset.

In the table below you can see the final comparison of our models. All trainings and testing were made inside Google Collaboratory with its own GPU.

Table 3. The final comparison of models.

Net-work	Training time (1 epoch)	Ram usage	Training accuracy	Training loss	Test time	Test accuracy	Test loss
Custom	12s	2.9GB	0.8111	0.5819	1s	0.7929	0.6472
Fractal	116s	5.5GB	0.8864	0.3877	6s	0.8864	0.3922

6 Conclusions

In the current paper we run custom neural network and fractal neural network inside Google Collaboratory using given GPU. Then we trained and tested them on CIFAR-10 dataset. Custom neural network showed us worse results than fractal one, but each iteration took up to 10 seconds, when 1iteration of fractal neural network took up to 3 minutes. Moreover, our network is pretty simple, so we can say that that is suits better for datasets with lower quantity of classes. Fractal neural network showed us pretty good results, but we are sure that with more powerful computing resources and more time it can perform much better.

As we mentioned before, we can apply this technology on a different medical data to solve various kinds of medical problems. This can help to decrease the amount of human mistakes.

References

[1] Estivill-Castro, V., Lee, I.: Amoeba: Hierarchical clustering based on spatial proximity using Delaunay diagram, 9th Intern. Symp. on spatial data handling, pp. 26–41 Beijing, China (2000).

[2] Kang, H.-Y., Lim, B.-J., Li, K.-J.: P2P Spatial query processing by Delaunay triangulation, Lecture notes in computer science, vol. 3428, pp. 136–150, Springer/Heidelberg (2005).

[3] Boehm, C., Kailing, K., Kriegel, H., Kroeger, P.: Density connected clus-tering with local subspace preferences, IEEE Computer Society, Proc. of the 4th IEEE Intern. conf. on data mining, pp. 27–34, Los Alamitos (2004).

[4] Boyko, N., Shakhovska, N., Basystiuk, O.: Performance evaluation and comparison of software for face recognition, based on dlib and opencv library, Second International Conference on Data Stream Mining and Processing, pp. 478-482, DSMP (2018).

[5] Boehm, C., Kailing, K., Kriegel, H., Kroeger, P. : Density connected clus-tering with local subspace preferences” IEEE Computer Society, Proc. of the 4th IEEE Intern. conf. on data mining, pp. 27–34, Los Alamitos (2004).

[6] Harel, D., Koren, Y. :Clustering spatial data using random walks, Proc. of the 7th ACM SIGKDD Intern. conf. on knowledge discovery and data mining, pp. 281–286, San Francisco, California (2000).

[7] Tung, A.K., Hou, J., Han, J. :Spatial clustering in the presence of obstacles, The 17th Intern. conf. on data engineering (ICDE’01), pp. 359–367, Heidelberg (2001).

[8] Veres, O., Shakhovska, N.: Elements of the formal model big date, The 11th Intern. conf. Perspective Technologies and Methods in MEMS Design (MEMSTEH), pp. 81-83, Polyana (2015).

[9] Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic sub-space clustering of high dimensional data, vol. 11(1), pp. 5–33, Data mining knowledge discovery (2005).

[10] Ankerst, M., Ester, M., Kriegel, H.-P.: Towards an effective cooperation of the user and the computer for classification, Proc. of the 6th ACM SIGKDD Intern. conf. on knowledge discovery and data mining, pp. 179–188, Boston, Massachusetts, USA (2000).

[11] Guo, D., Peuquet, D.J., Gahegan, M.: ICEAGE: Interactive clustering and exploration of large and high-dimensional geodata, vol. 3, N. 7, pp. 229–253, Geoinfor-matica (2003).

[12] Boyko, N., Shakhovska, N., Sviridova, N.: Use of machine learning in the forecast of clinical consequences of cancer diseases, In 7th Mediterranean Conference on Embedded Computing, pp. 531-536, IEEE MECO'2018 (2018).

[13] Boyko, N.: Advanced technologies of big data research in distributed information systems, Radio Electronics, Computer Science, Control. № 4, pp. 66-77, Zaporizhzhya: Zaporizhzhya National Technical University (2016).

[14] Larsson, G., Maire, M., Shakhnarovich, G.: FractalNet: Ultra-Deep Neural Networks without Residuals, http://people.cs.uchicago.edu/~larsson/fractalnet/

[15] Mochurad, L., Solomiia, A.: Optimizing the Computational Modeling of Modern Electronic Optical Systems. In: Lytvynenko V., Babichev S., Wójcik W., Vynokurova O., Vyshemyrskaya S., Radetskaya S. (eds) Lecture Notes in Computational Intelligence and Decision Making, pp 597-608, ISDMCI 2019. Advances in Intelligent Systems and Computing, vol 1020. Springer, Cham. (2019)