Convolutional Neural Networks for Red Blood Cell Trajectory Prediction in Simulation of Blood Flow

Chapter · April 2019

DOI: 10.1007/978-3-030-17935-9_26

4 authors, including:

Michal Chovanec
University of Žilina
4 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Hynek Bachraty
University of Žilina
32 PUBLICATIONS **62** CITATIONS
SEE PROFILE

Katarína Bachratá
University of Žilina
44 PUBLICATIONS **73** CITATIONS
SEE PROFILE

Convolutional Neural Networks for Red Blood Cell Trajectory Prediction in Simulation of Blood Flow

Michal Chovanec[1⋆], Hynek Bachratý[2],
Katarína Jasenčáková[2], and Katarína Bachratá[2⋆⋆]

University of Žilina, Faculty of Management Science and Informatics,
[1]Department of Technical Cybernetics,[2]Department of Software Technology, Žilina, Slovakia,
{michal.chovanec,hynek.bachraty,
katarina.jasencakova,katarina.bachrata}@fri.uniza.sk
http://cell-in-fluid.kst.fri.uniza.sk

Abstract. Computer simulations of a blood flow in microfluidic devices are an important tool to make their development and optimization more efficient. These simulations quickly become limited by their computational complexity. Analysis of large output data by machine learning methods is a possible solution of this problem. We apply deep learning methods in this paper, namely we use convolutional neural networks (CNNs) for description and prediction of the red blood cells' trajectory, which is crucial in modeling of a blood flow. We evaluated several types of CNN architectures, formats of theirs input data and the learning methods on simulation data inspired by a real experiment. The results we obtained establish a starting point for further use of deep learning methods in reducing computational demand of microfluid device simulations.

Keywords: microfluidic devices, simulation experiment, deep learning, convolutional neural networks, trajectory prediction

1 Introduction

Computer based simulations of blood flow

Microfluidic devices have been investigated for multiple medical applications. For instance, these devices can be used for capturing of circulating tumor cells (CTCs) and thus for the early diagnostics and the targeted treatment of cancer. Since CTC are greatly outnumbered by other cells, it is highly desirable that the microfluidic devices are very efficient. The efficiency can be optimized by altering the shape, dimensions and other parameters of these devices. However, testing, verification and comparison of their properties on real prototypes of microfluidic devices is labourious and technically demanding. We have proposed replacing it with numerical simulation , which needs to be sufficiently authentic [1], [5].

⋆ Author of this work was supported by the Ministry of Education, Science, Research and Sport of the Slovak Republic under the contract No. VEGA 1/0643/17.

⋆⋆ Authors of this work were supported by the Slovak Research and Development Agency under the contract No. APVV-15-0751.

The model we use considers blood as a suspension. The suspensions' model consists of plasma and of elastic models of its solid components. The dominant part of solid components is formed by red blood cells (RBCs). Thus, the correct setup of elastic parameters of the RBCs' behavior and their interactions with the liquid, surface of device and other objects, are crucial for correctness and accuracy of used simulation model. Our research group uses the software tool ESPResSo for these simulations. We expanded it with a module of immersed elastic objects, which is still under development [6].

Nowadays, there is increasingly obvious need to augment numerical simulations. The need arises from the high modeling complexity together with the scope of simulations. The complexity involves the size and the topology of the simulated device, the amount of the modeled RBCs, the duration of the simulated experiment as well as a range of the monitored and recorded measurements. For these reasons, the simulation experiments performed at present have very high requirements on hardware efficiency as well as on the required computation time. Often times simulations run for several days or even weeks. Hence it happens to be a limitation for rerunning or extension of the simulation experiments with the same or slightly modified parameters. Even though a typical simulation provides a large amount of the observation data, we usually need to use only a fraction of it for a given end goal. Several analyses [2], [3], [4] suggest the experiments can be universally described and characterized by miscellaneous statistics on files from the output data. This inspired us to apply machine learning methods for complex processing of all output data from the simulation experiments. This approach promises to extract relationships from the simulation data which would one allow to obtain more results without the need for new simulation experiments.

The movement and the behavior of RBCs in general play the key role in our simulation experiments. Therefore, we decided to use machine learning methods firstly on trajectory prediction of RBCs immersed in the blood flow in the artificial microfluidic channel. Our first experience in this topic can be found in [3]. There, the data from the simulation experiments were used as the input for the basis functions of Kohonen neural network. These basis functions allowed miscellaneous types of RBCs' trajectory prediction in the particular device. We obtained acceptable accuracy for the predictions, even with the relatively unsophisticated machine learning method.

In this paper, we decided to verify the possibilities of applications of the newest methods in deep neural networks learning. We used it on the data from a different type of the simulation experiment. To our best knowledge this is the first such application of CNN for prediction of trajectories of RBCs. Therefore, the aim of this work is to test various approaches to data preprocessing, different network networks architectures, various learning methods and utilization.

2 Design of a simulation experiment

We used data from the simulation experiment, which was inspired by the real experiment described in [8]. We simulated the movement of 38 RBCs in the micro channel depicted below. The amount of the RBCs corresponds to the 1% hematocrit used in the laboratory experiment. Initial seeding of the simulated cells was in the left part of the

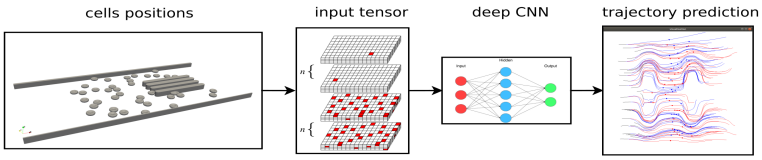

Fig. 1. RBCs' trajectory prediction using CNN from the simulation experiment.

channel, free from any obstacles. The main task of the experiment was to monitor characteristics like cells' deformation and velocity in the special narrow slits in the channel. The channel was periodic in direction of its main horizontal x-axis in the computer simulation. This direction also corresponds to the blood flow. We used the standard and calibrated parameters of the simulation model (see the table below). For the detailed description of the simulation model, see [9]. In terms of using the simulations results for a design and a validation of the CNNs architecture and their learning methods, the more important is the size and the quality of the data observed from the simulation described in the following section.

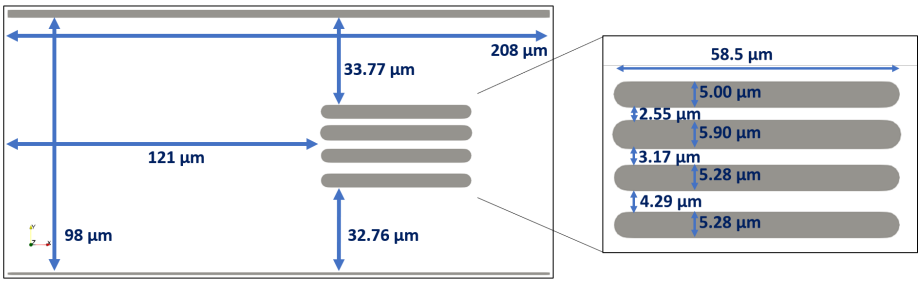

Fig. 2. Schema of the simulation channel with its dimensions.

3 Dataset description

3.1 Data processing

Data for the neural network comes from a simulation experiment. Its output includes information about trajectory properties of each cell. Before giving more in depth description of these data, it is good to acknowledge that the input of a neural network consists of several cell positions and output is the velocity of that cell at a particular time. There is record about movement of each cell during the simulation. Output files include information about the position and velocity of cell centers and also the velocity of border points, which are important when determining cell's rotation. The user can choose how often to record this data, since the time step is usually around tenths of microseconds. Data for this study were recorded every 0.001 s (once in 1000 simulation steps), which results in almost 9 200 records for each cell.

Parameter	SI unit
Radius	$3.91^{*}10^{-6}m$
Stretching coefficient k_s	$6^{*}10^{-6}N/m$
Bending coefficient k_b	$8^{*}10^{-18}Nm$
Coefficient of local area conservation k_{al}	$1^{*}10^{-6}N/m$
Coefficient of global area conservation k_{ag}	$9^{*}10^{-4}N/m$
Coefficient of volume conservation k_v	$5^{*}10^{2}N/m^2$
Membrane viscosity	$0m^2/s$
Density	$1^{*}10^{3}kg/m^3$
Kinematic viscosity	$1^{*}10^{-6}m^2/s$
Friction coefficient	$1.15(-)$

Table 1. First seven parameters are the elastic parameters of the cell model used in our experiments. In the last three rows are numerical parameters of simulation liquid.

Training and testing sets used for neural network are extracted from simulations which differ only in initial seeding of the cells. Therefore trajectories of the cells in these experiments are different. However, there should be similar behavior of cells' trajectories, since everything, apart from the initial seeding of the cells, is fixed. This includes the geometry of the channel, the elastic parameters of the cells and fluid parameters. In later work, we would like to apply our framework also on simulations with various geometries and parameter settings.

As was already mentioned, the input for the neural network are the positions of the cell center and the output is the velocity of this cell at given position. The learning set for our neural network consist of the simulation output which also includes redundant data. We processed the data in the following steps:

1. We loaded the data from the experiments and checked their correctness. Afterwards, we extracted data, which were relevant for the neural network and found the extreme values.
2. Next step was the normalization of the data. Using the extreme values from the previous step, we obtained data in interval $\langle 0, 1\rangle$. Normalization of data is common in neural networks. It is needed for instance, when channels used in training and testing have different lengths or if the duration of training and testing simulation is different for some reason.
3. We made the desired pairs of the inputs and the outputs from the normalized data.
4. We created the desired input for the neural network. It is one of the tensors specified in the next section.

3.2 Neural network input

Correct type of input is very important for the results of the neural network. We work with two types of the input tensor in the form of a $3-$dimensional array. Notice, that these tensors have some adjustable functions.

Input tensor based on coordinates of the cell centres
This input type is set by algorithm parameter called *no_spatial_tensor*. Each row represents x, y and z coordinates of the cell centre at given time. Series of rows than represents the movement of the cell centre. To get an ample information about the cell's movement it is sufficient to use every $800th$ row from simulation output. (In the following text we will call this time step.) For this kind of two consecutive rows, the cell usually moves a little bit less than radius of its largest dimension. Which means, that cell at two consecutive positions probably has a small intersection with itself. The distance is computed using Euclidean norm. For our tensor we used zero padding of width p. The horizontal layers of the tensor represent different cells, where the cell we are focused on is on the top of the tensor. By setting boolean variable use_depth true or false, we can choose if we want to consider all cells or only the one cell we are focused on. Thus the dimension of this tensor is:

$$\begin{aligned} &(3+2p) \times (n+2p) \times 1, && \text{if } use\_depth = false, \\ &(3+2p) \times (n+2p) \times c, && \text{if } use\_depth = true, \end{aligned}$$

where n is the number of time steps and c is the number of the cells.

In the experiment we used $n = 8$, number of the cells was $c = 38$ and we used padding $p = 1$. Due to our processing limitations we chose to use_depth false and therefore, the dimension of the input was $5 \times 10 \times 1$.

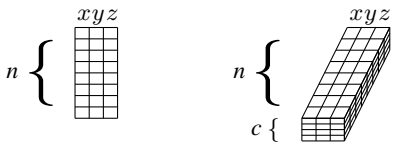

Fig. 3. Schemes of the *no_spatial_tensor* inputs (without padding). The input tensor for *use_depth* = *false* is on the left side. The input tensor for *use_depth* = *true* is on the right side.

Input tensor based on spatial discretization
This input type for the neural network set by parameter called *spatial_tensor* is inspired by the image processing. The spatial tensor consists of n or $2n$ smaller tensors $A^{(i)}$, $i = 1, 2, \ldots, n$ which describe the position of the cell centres. Where n represents the number of the time steps.

We split the channel into $disc\_x \times disc\_y \times disc\_z$ identical rectangular areas. We obtain a small tensor A by determining if the cell is present or not, in each of the $disc\_x \times disc\_y \times disc\_z$ channel areas. Where

$$A_{ijk} = \begin{cases} 1, & \text{if there is a centre of some cell,} \\ 0, & \text{if there is no centre of any cell.} \end{cases} \tag{1}$$

Since this input is orthogonal the neuron networks gives us more accurate output values.
We also considered another approach, where the value 1, at the particular position A_{xyz}, is replaced by values of a Gauss function. More precisely, we replace all values in the s^3 neighborhood of the element A_{xyz}. We set s to be an odd number in order for A_{xyz} to be in the centre with the maximum value of Gauss function. To set up this approach, we set parameter $use\_gaussian\_kernel$.
We make n tensors, $A^{(1)}, \ldots, A^{(n)}$, for each time step. Each tensor records the position of the centre of one cell. Thus, if we don't use the gaussian kernel, $A^{(i)}$ is a sparse matrix with only one value 1. The input tensor consists of smaller tensors $A^{(n)}, \ldots, A^{(1)}$ in that order.
If we want to incorporate all the cells, we set $use\_depth = true$ to create another n tensors $B^{(1)}, \ldots, B^{(n)}$. Therefore each of these tensors includes the information about all of the cell centres. Then the input tensor consists of small tensors $A^{(n)}, \ldots, A^{(1)}$, $B^{(n)}, \ldots, B^{(1)}$ in this order. The small tensor $A^{(n)}$ is at the top horizontal layer of the input tensor and $B^{(1)}$ is at the bottom layer of the input tensor. Therefore the dimension of the input tensor is:

$$\begin{array}{ll} disc\_x \times disc\_y \times disc\_z * n, & \text{if } use\_depth = false, \\ disc\_x \times disc\_y \times disc\_z * 2n, & \text{if } use\_depth = true, \end{array}$$

We can see the schemes for these inputs in the Figure 4. Until now, we did not use the inputs for the neural network, where all cells are included. In our computations, we used $disc\_x = 16, disc\_y = 16, disc\_z = 3$ and $n = 8$. We also used the gaussian kernel. Hence, the dimension of the input tensor for the neural network was $16 \times 16 \times 24$.

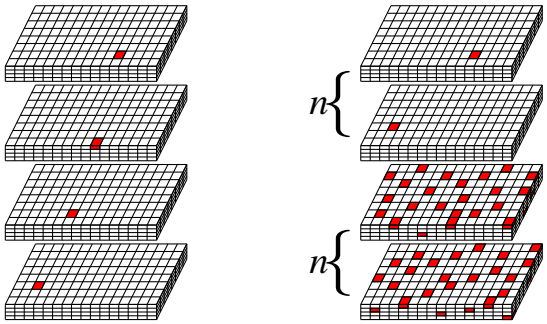

Fig. 4. Schemes of the *spatial_tensor* inputs, for $disc\_x = 16$, $disc\_y = 8$, $disc\_z = 4$. Marked areas represent positions of the cell centres. There are tensors $A^{(i)}$ illustrated on the left side, where position of the one cell centre is recorded. If we join these tensors together as is described above, we obtain *spatial_tensor* for *use_depth* = *false* with $n = 4$. There is *spatial_tensor* for *use_depth* = *true* depicted on the right side.

3.3 Trajectory prediction based on network's output

At the beginning we choose time $T > 0$, from which we want to predict cell's trajectories. In order to do so, our framework first predicts the velocity of the cell from past n positions of this cell recorded in the input tensor. Then it predicts positions from these predicted velocities. This is computed for all of the cells at given time. And then it is repeated with the data in the input tensor shifted by one simulation step. Data for the input tensors are selected as follows. The data from *simulation time* $< T$ are from the simulation output and data from *simulation time* $\geq T$ are predicted by the algorithm.

The next scheme (Figure 5) illustrates the prediction of RBCs' trajectories from theirs velocities.

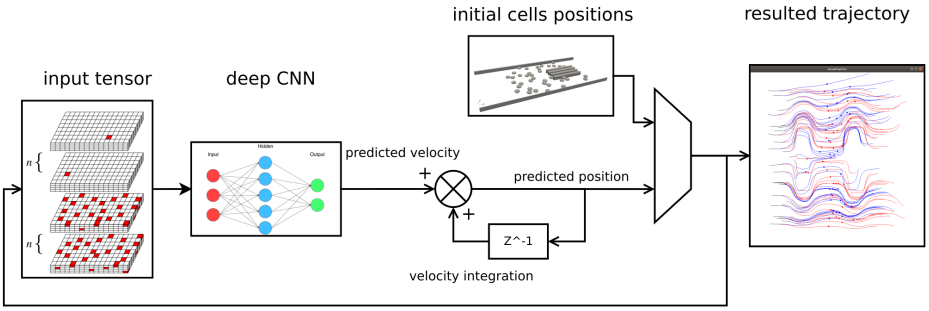

Fig. 5. RBCs' trajectory prediction from predicted velocities using our CNN from the simulation experiment.

4 Different types of network architectures

In 2018, Bachratý et al. [3], trained and then used network for trajectory prediction, inspired by the radial basis function network and Kohonens' self-organized maps. Our research advanced to framework using CNNs. This type of network is mostly applied to image and video recognition. It has quite little pre-processing in comparison to other image classification algorithms. Since spatial discretization based input is derived from the image of the channel, we expect the use of framework based on CNN as appropriate. Nevertheless, it doesn't mean that this approach cannot be used with the coordinates based input. We confirmed our hypothesis, that framework works better with the *spatial_tensor* parameter. Besides using convolution or fully connected layers, we also used a relatively new type of layers, the dense convolution layers, introduced in [7].

Here we present results for the 8 neural network experiments of trajectory predictions. There were 6 different architectures of neural networks, see the Table 2. First four experiments, named Experiment_0 to Experiment_3 have *no_spatial_tensor* input with the same parameters, but mutually different networks architectures. Similarly for the Experiment_4, to Experiment_7, but with the *spatial_tensor* parameter. The network architectures for experiment pairs 2 and 4 and for 3 and 5 are the same.

layer	net 0	net 1	net 2	net 3	net 4	net 5	net 6	net 7
0	fc 256	conv 3x3x32	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
1	fc 64	fc 64	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
2	fc 32	fc 32	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
3	fc 3	fc 3	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
4			conv 1x1x32	conv 1x1x16	conv 1x1x32	conv 1x1x16	conv 1x1x16	conv 1x1x32
5			fc 3	dense conv 3x3x8	fc 3	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
6				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
7				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
8				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
9				conv 1x1x32		conv 1x1x32	conv 1x1x16	conv 1x1x32
10				fc 3		fc 3	dense conv 3x3x8	dense conv 3x3x8
11							dense conv 3x3x8	dense conv 3x3x8
12							dense conv 3x3x8	dense conv 3x3x8
13							dense conv 3x3x8	dense conv 3x3x8
14							conv 1x1x32	conv 1x1x64
15							fc 3	fc 3

Table 2. Network architectures.

Hyperparameters
Used neural network architecture hyperparameters are these: Weights are initialized in the range *xavier*, bias is set to 0. Learning rate is 0.0002, $\lambda_1 = \lambda_2 = 0.000001$, dropout $= 0.02$ and minibatch size is 32.

Computations ran on CUDA, graphics card GeForce GTX 1080 Ti. Our algorithm is written in C++ and Python. The training phase lasts about 6 hours for more complicated network architectures. The trajectory predictions calculated using already trained network took about half an hour on the graphics card GTX 960.

5 Results

Comparing the difference between the target and resulted values given from neural networks, we observed that networks with *spatial_tensor* parameter (Experiments 4 to 7) give better results than for *no_spatial_tensor* parameter. The Experiment_7, where the network has the most layers gave us the best result. In Figure 6 the histograms of target and resulted output values of velocities are depicted. Note, that these values are normalized, but it doesn't qualitatively affect the information which it provides. These histograms are necessary verification of the framework we used, but not sufficient. The

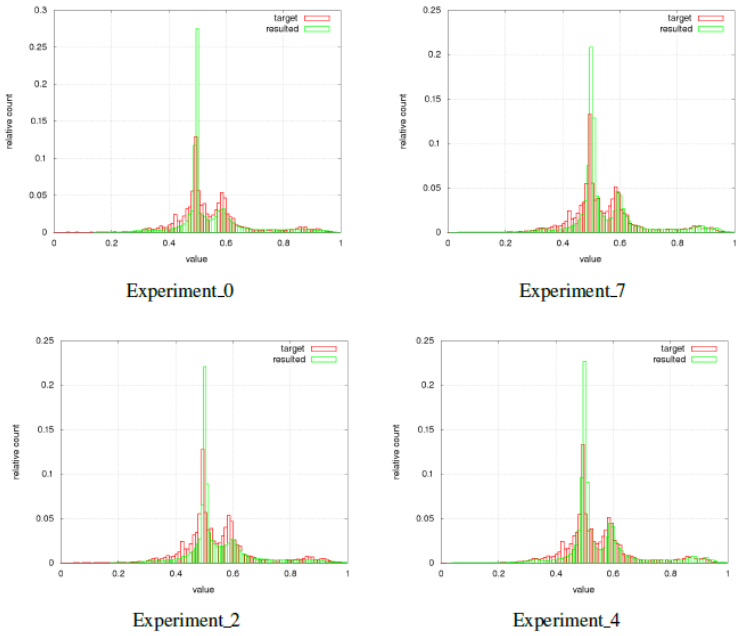

Fig. 6. Histograms of target (red) and resulted (green) values. The experiments with *no_spatial_tensor* input are on the left side and experiments with *spatial_tensor* input are on the right side. Note, that Experiment 0, 2 and 7 have different architecture and 4 has the same architecture as 2.

Figure 7 shows relative error of predicted velocities summed in all three dimensions. Note, that the ranges on the vertical axis are different. We can immediately see, that the experiments with *spatial_tensor* parameter have smaller error. This holds for all of the 8 experiments.
We can see a considerable difference between testing progress of the first four experiments and the other four experiments on the left side of the Figure 8. The Figure on the right shows, in detail, the testing progress from Experiment_4 to Experiment_7.

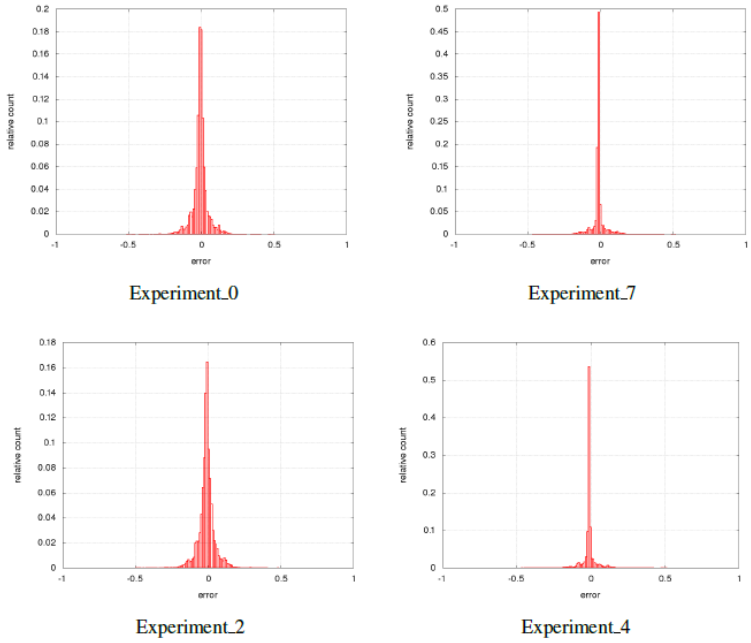

Fig. 7. Histograms of sums of errors in x, y and z axis.

The Figure 9 shows the trajectory prediction of RBCs. Red trajectories are target and blue trajectories are resulted from neural networks experiments. The experiments on right side with *spatial_tensor* parameter give better results.

6 Conclusion

We introduced our approach on trajectory prediction by neural networks. We described data processing and our first results. From these we can determine, that spatial discretization based input and CNN architectures with the largest number of hidden layers are the best the trajectory prediction. There seems to be more applications of this prediction, e.g.:

– prediction of RBCs' trajectory prolongation from executed simulation experiment,

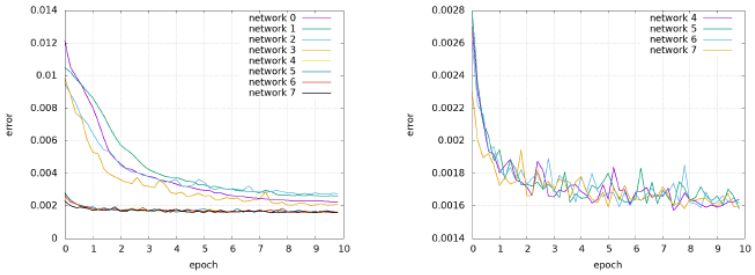

Fig. 8. Testing progress of networks.

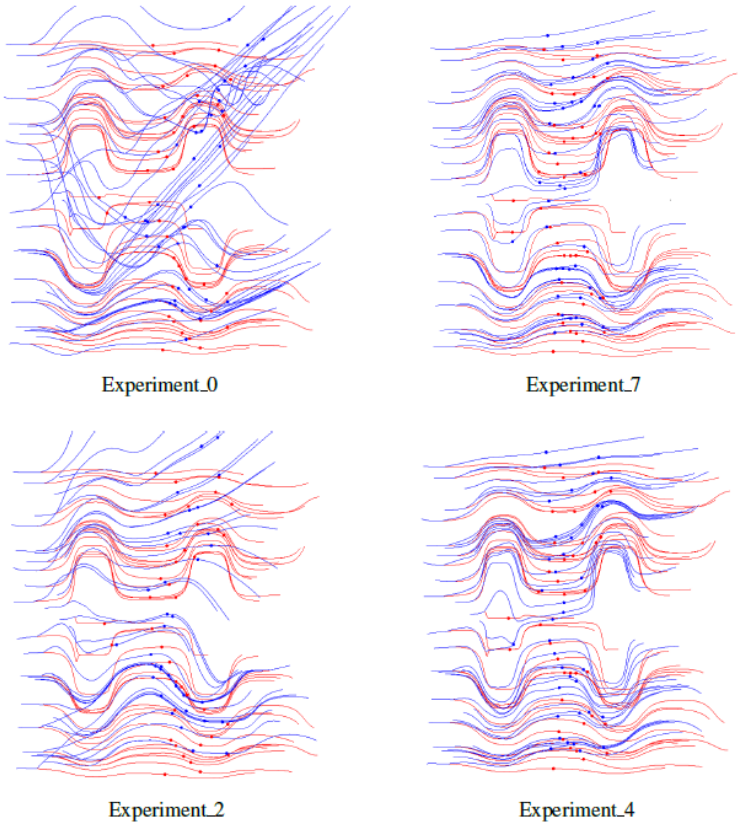

Fig. 9. RBCs' trajectory prediction. Red trajectories are target and blue trajectories are resulted.

- prediction of RBCs' local behavior in a random position of a microfluidic channel. Including such positions, where cell never occurred during the simulation experiment,
- prediction and construction of fictive RBCs' trajectory with an arbitrary starting position in a channel,
- construction of completely virtual simulation experiment with given initial seeding of a larger number of RBCs,
- prediction of RBCs' trajectory on video records, useful for tracking algotithms.

We would like to apply our framework on other types of blood flow simulation experiments based on real laboratory experiments. Because it is not easy to find suitable examples of precisely described experiments, we welcome cooperation in this area.

References

1. Bachratá, K., Bachratý, H.: On modeling blood flow in microfluidic devices, ELEKTRO 2014: 10th International Conference, IEEE, ISBN 978-4799-3720-2, 518–521 (2014)
2. Bachratá, K., Bachratý, H., Slavík, M.: Statistics for comparison of simulations and experiments of flow of blood cells, EPJ Web of Conferences, Vol. 143, art. no. 02002 (2017)
3. Bachratý, H., Bachratá, K., Chovanec, M., Kajánek, F., Smieková, M., Slavík, M.: Simulation of blood flow in microfluidic devices for analysing of video from real experiments, Rojas, I., Ortuño, F. (eds.) Bioinformatics and Biomedical Engineering, 279-289 (2018)
4. Bachratý, H., Kovalčíková, K., Bachratá, K., Slavík, M.: Methods of exploring the red blood cells rotation during the simulations in devices with periodic topology, 2017 International Conference on Information and Digital Technologies (IDT), Zilina, 36–46 (2017)
5. Cimrák, I., Bachratá, K., Bachratý, H., Jančigová, I., Tóthová, R., Bušík, M., Slavík, M., Gusenbauer, M.: Object-in-fluid framework in modeling of blood flow in microfluidic channels, Comunications, Scientific Letters of the University of Zilina, vol. 18/1a, 13–20 (2016)
6. Cimrák, I., Gusenbauer, M., Jančigová, I.: An ESPResSo implementation of elastic objects immersed in a fluid, Computer Physics Communications, vol. 185, 900–907 (2014)
7. Huang, G., Liu, Z., Maaten, L.V., Weinberger, K.Q.: Densely Connected Convolutional Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2261–2269 (2017)
8. Tsai C. H. D. et all: An on-chip RBC deformability checker significantly improves velocity-deformation correlation, Micromachines, 7, 176 (2016)
9. Kovalčíková, K., Bachratý, H., Bachratá, K., Jasenčáková, K.: Influence of the Red Blood Cell Model on Characteristics of a Numerical Experiment. In Experimental Fluid Mechanics conference, Prague, (2018), in press.