3D CN-LSTM for prediction of medical nanoparticle properties

Vid Sustar, 1901627
Supervisors: Sepinoud Azimi Rashti, Sébastien Lafond
Faculty of Science and Engineering
Åbo Akademi University
2021

Abstract

Cancer is an increasing and already one of the most common causes of death in developed countries. One way to fight cancer tumours is with targeted anti-tumour drug delivering nanoparticles (NPs). NPs can be composed of gold core covered with a variety of drug and supporting substances (SS) in varying ratios.

Since chemical synthesis of all potential NPs is costly, to find the most optimal drug and drug-SS ratios out of many potential candidates, NPs are simulated in silico in molecular dynamics (MD) simulations. To further lower the costs and expand coverage of potential optimal NP compositions, computationally demanding MD simulations of NPs could in part be replaced with Deep Learning (DL) neural networks. Here the properties of NPs at later stages of MD simulation would be predicted with DL from NP properties from starting stages of MD simulations.

As MD simulations are time series and NPs simulated are 3D objects, one can join two types of DL: recurrent neural networks (RNN) and convolutional neural networks (CNN) to create a suitable DL network. The scope of this master's thesis is running MD simulations, finding proper DL architecture for the model, refining the input and assessing the predictions of the refined model.

The architecture giving the best prediction of NP drug exposure is a combination of concatenated 3D CNN for NP structure input and dense layers for other types of input fed into Long Short-Term Memory (LSTM) RNN.

Keywords: nanoparticles, molecular dynamics, deep learning, RNN, CNN, LSTM

Abbreviations:
CNN - Convolutional Neural Network
DL - Deep Learning
LSTM - Long Short-Term Memory
MD - molecular dynamics
NP - nanoparticle
RNN - Recurrent Neural Network
SS - supporting substance

Acknowledgements

I would like to thank my supervisors Sepinoud Azimi Rashti and Sébastien Lafond for introducing me to the exciting topic of my thesis. I am also grateful to professor Jan Westerholm for his help in setting up the computations on the local cluster Dione, the assistants Valenting Soloviev and Luca Zelioli for their explanations about bash scripting, deep learning, 3D convolution and LSTMs. I am also grateful to colleague Otto Lindfors as well as other students for discussions and suggestions about deep learning and tensorflow.

Last, but not least I thank my wife Mira for her patience and for allowing me to sacrifice some of our family life for the pursuit of a master's in computer science.

Contents

1 Introduction

In this section the topic of the thesis is explained with introduction to cancer and drug delivering nanoparticle simulations and its possible deep learning upgrade.

1.1 Cancer and EVO-NANO project

Cancer is one of the leading causes of death in developed countries [1]. The main reasons for the high incidence of cancer are mainly consequences of modern lifestyle and toxic environment: tobacco smoking, consumption of alcohol, a diet low in fruit and vegetables, physical inactivity, obesity, sexual transmission of human papillomavirus, exogenous hormones, UV radiation, etc. [2]. Since cancer is an increasing cause of death, it is of even greater importance to find the possible remedies. The best approach is preventive care through avoidance of the above list of cancer-causing agents. However, when the cancer is already present, there are several possible treatments depending on the stage and type of cancer, e.g. surgery, radiation therapy, chemotherapy, targeted therapies, immunotherapy, hormonal therapy, angiogenesis inhibitors and synthetic lethality. Each of these has its own benefits and drawbacks. More novel approaches, as for example the approach of cancer-targeting nanoparticles which can carry an array of drugs, may be a combination of the above.

This master's thesis is a subproject of the 'Evolvable platform for programmable nanoparticle-based cancer therapies' (EVO-NANO) EU project which, in turn, is a project in Horizon 2020, Cordis framework [3]. The main aim of EVO-NANO is to create a cross-disciplinary platform for assessment of drug-delivering nanoparticles (NPs) in-silico. The NPs are proving to be an ever more interesting anti-cancer therapeutical approach, however, their effective distribution in the body remains its limitation. It is crucial to understand the behaviour of the multitude of the synthetic NPs interacting with tumour in its natural environment. In order to find effective NPs, new types of algorithms need to be developed that will satisfy several conditions: creation of new anti-cancer strategies, expansion of the space of possible solutions, being adaptable to changing scenarios. Finally, the best selected candidates will be synthesised and tested *in vivo* and *in vitro* on tumour cells (xenografts and microfluidic testbeds). The research stage is directly continued in production of commercialised product, which can attract additional investors and give the project additional potential. With its relatively fast development and assessment cycle, EVO-NANO is at the forefront of cancer nanomedicine. The project will produce tangible tools to help people fight cancer.

1.2 Nanoparticles and Molecular Dynamics shortcuts

The NPs used are composed of a gold core and a fixed total number of hydrocarbon chains that carry either drug or supporting substance (SS) in varying ratios (see Figure 1). The role of the gold core is as a surface to be covered with desired molecules under controlled conditions and is, otherwise, chemically inert. As such, the NP core is stable and its behaviour predictable. Gold reflects infrared wavelengths and allows potential long wavelength multiphoton tracking. Different drug types also have different water solubility properties. SS improves the solubility of NP-attached-drug in water (blood). Exact theoretical prediction of the behaviour of many drug molecules chained to NP via hydrocarbons is not possible and neither is SS vs drug interaction possible, as it is a matter of interactions of thousands of atoms, and there can be emergent properties. For this reason, and because chemical synthesis of nanoparticles is slow and costly, varying drug and drug vs complementary substance are simulated via molecular dynamics (MD) simulations in-silico.

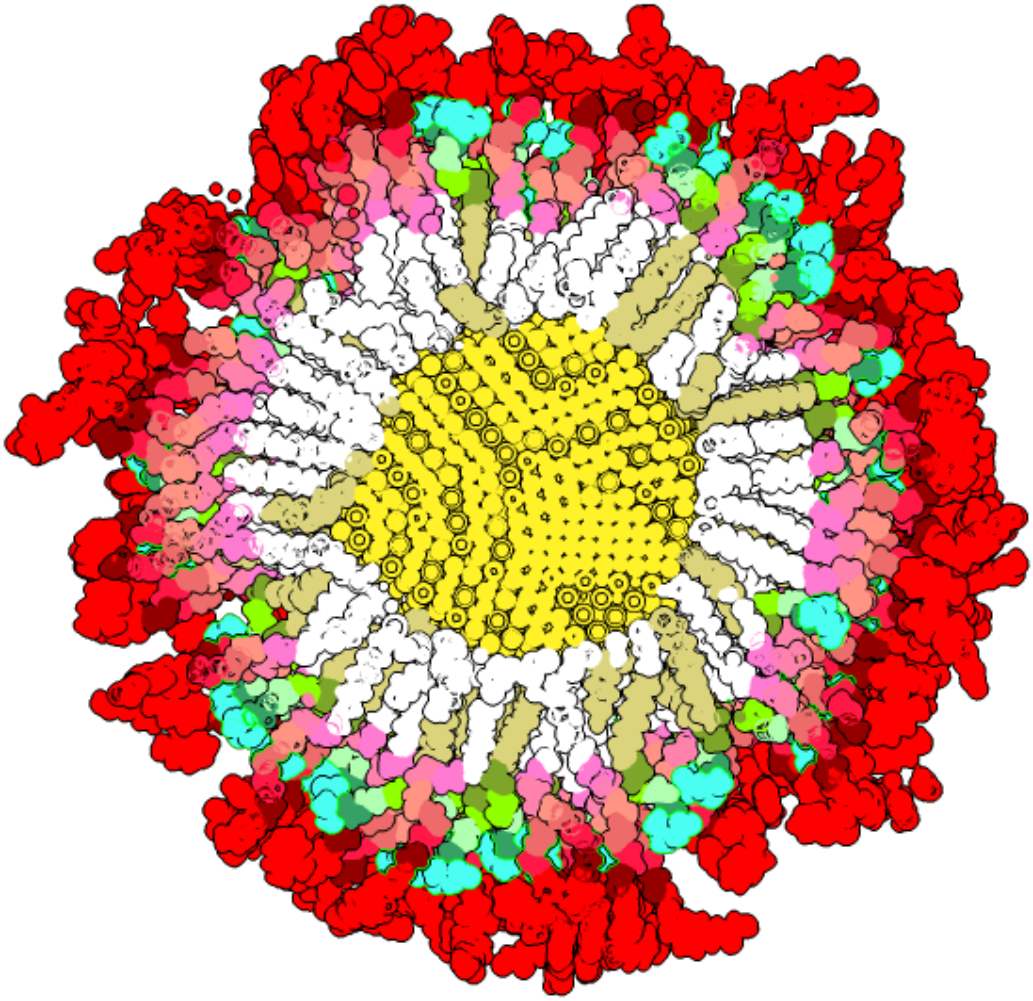

Figure 1: The sliced simulated nanoparticle with gold core (yellow), drug molecules (red), supporting substance (turqoise) and hydrocarbon chains connecting them to NP (redish and greenish hues). The diameter is 5 nm.

Molecular dynamics (MD) is a computer simulation method for analysing the physical movements of atoms in molecules. The atoms are set to interact for a defined period of time, which gives a representation of dynamic "evolution" of the system. Usually, trajectories of atoms are determined by usage of Newtonian motion laws for interacting particles, where forces between atoms are calculated with interatomic potentials or molecular mechanics force fields. NPs used in MDs need to have exactly defined composition. The nanoparticles used in this study, for example, have 6000 gold atoms, 50 000 atoms belonging to combined 420 SS and drug molecules and 50 000 atoms belonging to water molecules. All in all, there are more than 100 000 interacting atoms. The MD is calculated to show the stabilisation of "theoretical" nanoparticles from vacuum into water solvent and its stabilised behaviour in water. This requires the simulation of NPs in water for a range between 300 and 600 nanoseconds. To calculate such a multitude of interactions a modern top shelf GPU needs to calculate one MD simulation for a week.

To further lower the costs and expand coverage of potential optimal NP compositions, computationally demanding MD simulations of NPs could, in part, be replaced with Deep Learning (DL) neural networks, where the properties of NP at later stages of MD simulation would be predicted with DL from NP properties from starting stages of MD simulations. As MD simulations are time series and nanoparticles are 3D objects, one can join two types of DL: recurrent neural networks (RNN) and convolutional neural networks (CNN) to create a suitable DL network. The scope of this thesis is running MD simulations, finding proper DL architecture for the model, and refining the input for chosen DL architecture.

2 Background

In this section deep learning and its basic building blocks, principles are explained as well as more specific solutions to extraction of features from image/matrix data and from sequential data in the form of Convolutional Neural Networks and Long Short-Term Memory.

2.1 Deep Learning

Machine learning is an array of computer algorithms that improve through experience [4]. Machine learning can be either unsupervised or semi/supervised. In supervised machine learning, the algorithms are trained using labeled data, whereas in unsupervised machine learning the algorithms are trained on unlabeled data. Artificial neural networks (ANNs) are part of supervised machine learning methods [5].

Artificial neural networks, as the name implies, were inspired by biological neural networks (BNNs). There are major differences between the two, as ANNs tend to be static and digital, whereas BNNs are dynamic and analog [6].

The artificial neural networks are composed of neurons. Each artificial neuron has one or more inputs and produces an output that can be simultaneously sent forward as input to multiple other neurons.

The output of each neuron is produced in the following manner:

1. All the inputs to the neuron have weights, the weighted inputs are summed up.
2. A bias value is added to the above sum, which forms activation.
3. Activation is passed through nonlinear activation function to produce the output.

The output of a neuron is an activation function to the neurons in the next layer, it determines whether the neurons in the next layer should be activated ("fired") or not. The activation function introduces the non-linearity which separates a neural network from simpler regression models [7]. An example representation of an artificial neuron with inputs, weights, bias, that compose activation function is in figure 2.

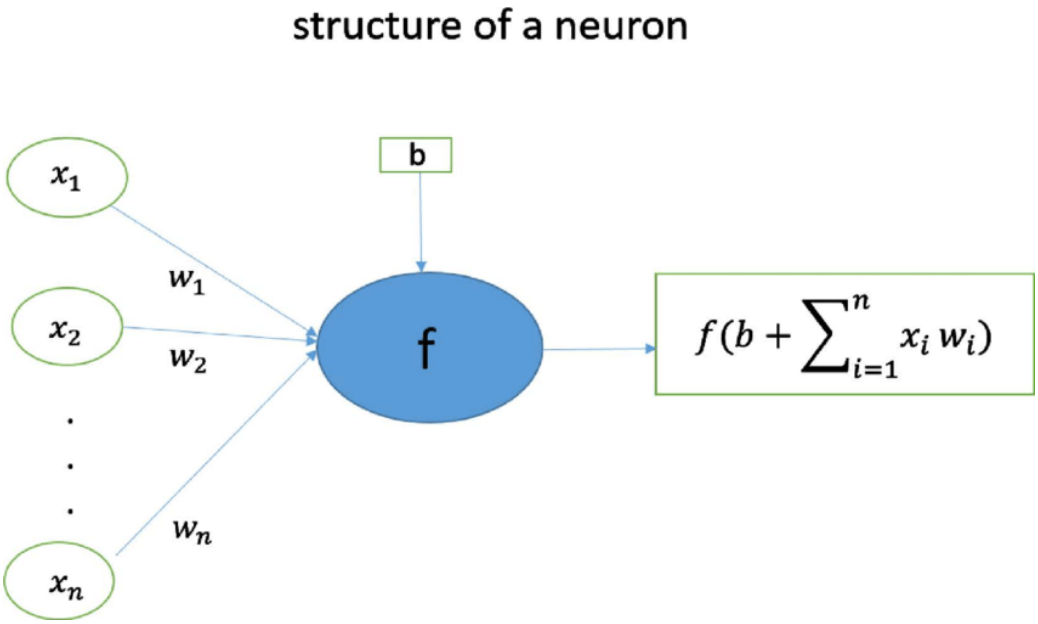

Figure 2: A diagram of an artificial neuron with: inputs x, weights w, bias b, composing an activation function f. From [8].

Neurons are organised in parallel into layers: parallel input neurons - input layer, intermediary and output layers. When there is more than one intermediary hidden layer between input and output layer such a network is called "deep", hence deep learning neural networks.

In case of the input layer, the input values might be pixel values of an image, or vectorised characters of a string, whereas the final outputs of the output layer can be annotated categories of an image, etc. An example of a simplified representation of DL network is in figure 3.

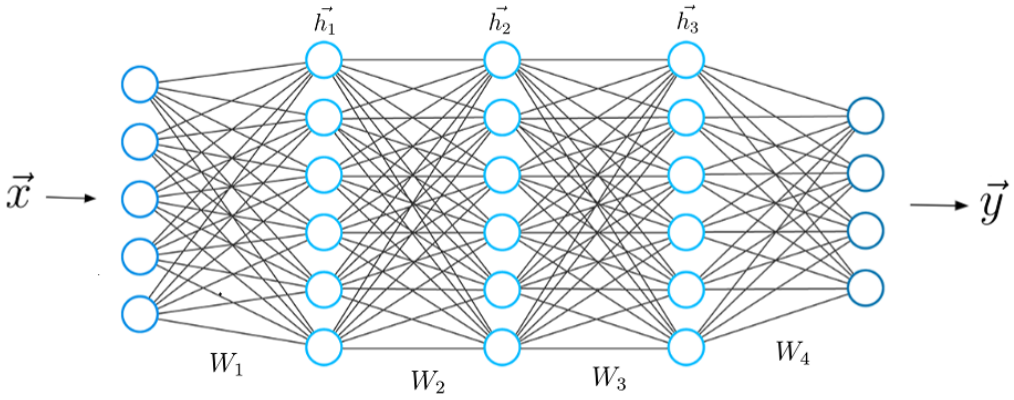

Figure 3: Simplified representation of a DL network, from [9].

Neural networks learn through the algorithm of backpropagation from errors in predictions. The errors (inaccuracy) in predictions are measured and calculated by loss function. The aim of deep learning models is to minimize the loss function value through the process of optimization. Optimization is an algorithm that modifies the

weights of neural network through gradient descent. During each iteration, weights are being tweaked to decrease the loss function gradually approaching a minimum or convergence, as shown in figure 4 [10].

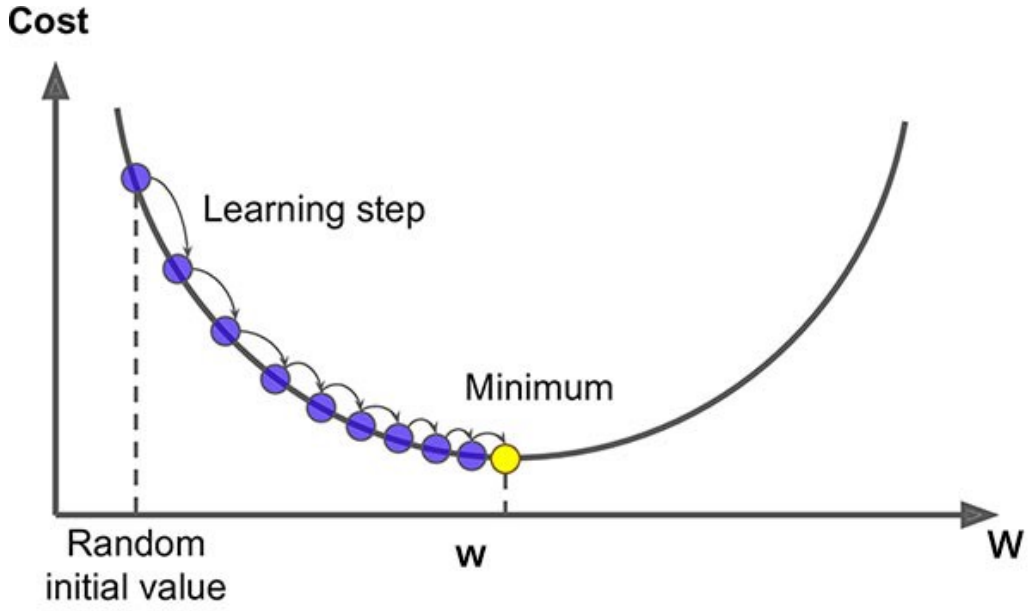

Figure 4: Gradient descent: cost of loss function minimisation with DLN weights optimisation through iterative learning steps, from [10].

When optimising the weights, weights can be adjusted to different extents. The adjustment can be considered as a step size or a learning rate of iterative gradient descent. Higher learning rate means faster descent (learning) by smaller number of iterations, however, it can be at a cost of missing the convergence [11], p. 247.

There are many optimisation algorithms used, one of the most commonly used is for example Adam, which computes adaptive individual learning rates for different parameters from estimates of first and second moments of the gradients [12], p. 1.

2.2 Convolutional Neural Networks

When two functions are merged into a third function, this process is called convolution. Among artificial neural networks, there is a type that is even more closely mimicking biological neural networks. Convolutional neural networks are organised similarly to visual cortex. In animal visual cortex, single neurons respond to the stimulus covering just a small, restricted subregion of the cortex. Similarly, convolutional neural networks apply filters also known as kernels that cover and extract features of individual subregions of multidimensional input. Through this process, CNNs capture the temporal and/or spatial dependencies in data through the application of corresponding filters.

In figure 5 is shown an example of kernel feature detection with dot product multiplication between 3x3 kernel and same-sized part of input matrix. The destination pixel on the resulting matrix is the sum of the aforementioned multiplication product.

The kernel scans through the whole input matrix, creating a feature map composed of product sums.

Many different kernels are used to detect different types of initial simple features like edges etc. The feature maps of each kernel are combined in an output of convolutional layer [13]. Important parameters in the process are strides and padding. Strides are the number of pixels a kernel scans over the source matrix. Padding are the additional edge pixels, if the kernel multiple does not fit into the source matrix. The initial filters can be relatively simple, however, by assembling several consecutive convolutional layers on top of each other, each additional layer contributes to recognition and extraction of patterns of additional levels of complexity. Padding can be done in two ways: zero padding, by adding of zeroes to the edges or via valid padding, by removing a sufficient number of edge cells/pixels both with a goal for kernel multiple to fit into the matrix [14].

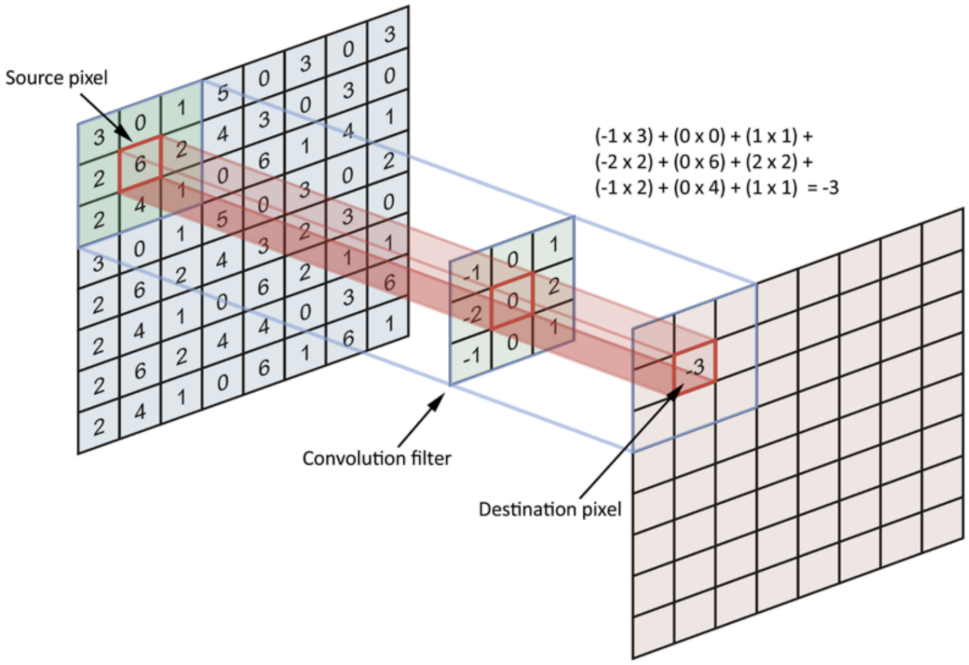

Figure 5: Principle of feature extraction with convolutional kernels of an image (pixel array), from [13].

To extract the features, pooling layers are used to reduce the width and height of feature maps, whereas the depth is preserved. The benefit of pooling is that the most informative features are preserved while required computational power is reduced. Pooling layers have two variations: max pooling, which returns max value within a considered area and average pooling, which returns the average value. Since max pooling extracts dominant features better, it performs better and is more widely used.

2.3 RNN and LSTM

In this section the deep learning solutions to sequential data are explained, namely Recursive Neural Networks and Long Short-Term Memory.

2.3.1 RNN

The CNN is a type of ANN that is a feedforward network, which learns from dataset, where the order of the input is not important and does not affect the output result. However, if time-dependent data are being analysed and the objective is to predict the future events, one needs to employ Recursive Neural Networks. RNNs are capable of modelling sequencesvby remembering past information and processing new events accordingly [15], p. 539. ANNs and RNNs have many components in common, such as neurons, cost functions and back-propagation, but the major difference is that RNNs use sequential data as input. Sequential data contain not only the sample information and their features as such, but also additional information about their order. Few examples of such information are video sequences and sentences. These would lose their original meaning if the order were changed. The data in this thesis, the molecular dynamics trajectories, are sequential data.

In RNNs, besides doing forward and backward propagation, information also flows in cycles. Through cycles, the information of present timepoint influences the future timepoint [15], p. 368.

These cycles are done within the hidden layers of a network, see figure 6 of an RNN, where they are coloured orange. On the right side of the figure, the main difference between simple feedforward ANNs and RNNs is apparent, while the hidden neurons of ANNs receive only one (initial, "outside") input, the RNNs receive two; both outside input, and the output from the preceding sequence element as an input. Since the next time step refers to a previous one, recurrence occurs, hence the name Recurrent NN. As such, RNN can approximate recurrent functions, unlike feedforward ANNs.

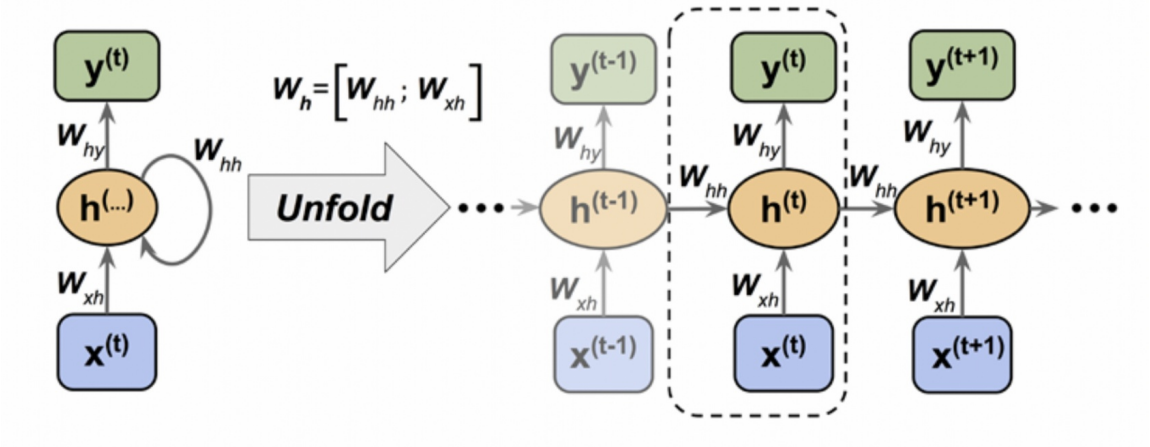

Figure 6: Simple RNN structure, with hidden are coloured orange, input blue, and output green. Looped arrow in the left part represents the cycles in hidden layers, exposed in the unfolded representation on the right, where the hidden layer from a previous sequence element ("timepoint") is connected to the subsequent element. From [15], p. 544.

Sequential data contain ordered progression, and can therefore provide so-called parameter sharing to RNNs and need fewer examples, compared to ANNs [16], p. 371.

2.3.2 LSTM

The RNN uses feedback connections for the storage of input-representations in a form of activations, creating a short-term memory (as opposed to a long-term memory in slower weight-changing). However, in their basic form, RNNs have a drawback: error signals going backwards in time may blow up or disappear, since backpropagated error depends on the size of the weights exponentially. In the first case, weights will oscillate, whereas in the second case, bridging long time lags consumes too much time, or cannot be accomplished [17].

Long Short-Term Memory (LSTM) type of RNNs solves the aforementioned issues of basic RNNs [18]. LSTM transforms the inputs in a sophisticated manner. As can be observed in figure 7, the inputs to each LSTM cell are manipulated and processed to create two outputs, which represent 'long-term memory' (upper horizontal line of arrows) and 'short-term memory' (lower output).

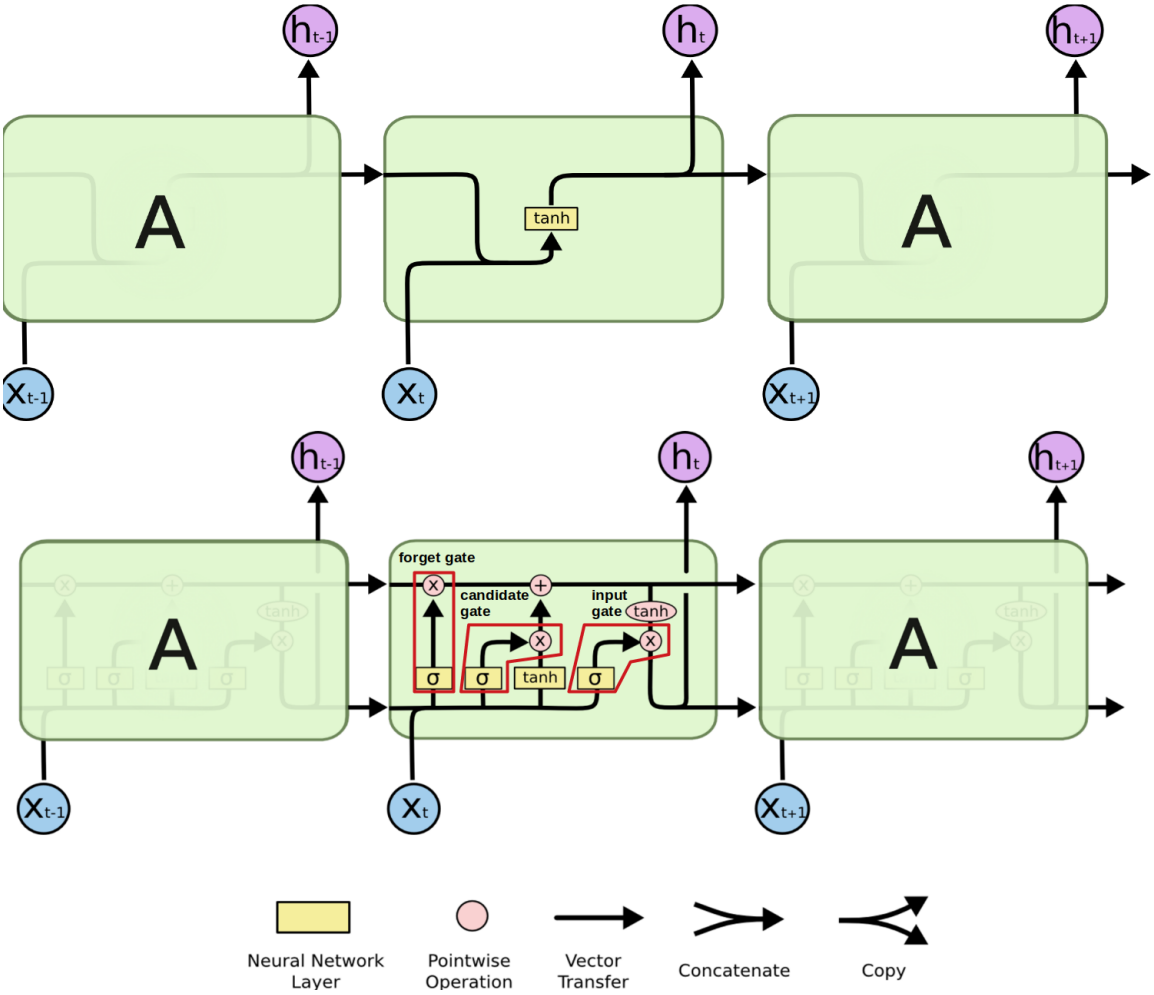

Figure 7: Comparison of RNN (above) and LSTM (below). The repeating cell in RNN contains only one layer, whereas LSTM contains four interacting layers: the original tanh layer + 3 gates (sigmoid NN layer + pointwise operation in red frame) that control the flow of information to the next cell. Modified from [19].

Vectors which go along the upper long-term memory channel may pass without

any changes. The gates (a combination of a sigmoid NN layer and a following pointwise operation) in the figure may block or add information. This way the network can retain data from an arbitrary number of sequence points (cells) in the past.

LSTM works in the following fashion:

1. The preceding hidden state and the current input are combined (concatenated) into a "combine".
2. The combine is fed into the forget gate, which removes non-relevant data.
3. The combine is used to create a "candidate" gate, which holds possible values to add to the cell state.
4. The combine is fed into the "input" gate, which decides what data from the candidate should be added to the new cell state.
5. The cell state is calculated after the gates are computed, from the gate values and the previous cell state.
6. The output is then computed.
7. Pointwise multiplication of the output and the new cell state creates a new hidden state.

The equations for the gates of the LSTM are the following [20]:

1. Input gate, determines what new information will be stored in the cell state:

$$i_t = \delta(w_i)\,[h_{t-1}, x_t] + b_i)$$

2. Forget gate determines which information to discard from the cell state:

$$f_t = \delta(w_f)\,[h_{t-1}, x_t] + b_f)$$

3. Output (candidate) gate which activates the final output of the LSTM cell at timespoint 't':

$$o_t = \delta(w_o)\,[h_{t-1}, x_t] + b_o)$$

4. Candidate cell state:

$$\tilde{c}_t = tanh(w_c)\,[h_{t-1}, x_t] + b_c)$$

5. Cell state:

$$c_t = (f_t) * c_{t-1} + i_t * \tilde{c}_t$$

6. Final output:

$$h_t = (o_t) * tanh(c^t)$$

Where:
$\mathrm{i}_t \rightarrow represents\ input\ gate$
$f_t \rightarrow represents\ forget\ gate$
$o_t \rightarrow represents\ output\ (candidate)\ gate$
$\delta \rightarrow represents\ sigmoid\ function$
$w_x \rightarrow weight\ for\ the\ respective\ gate\ (x)\ neurons$
$h_{t-1} \rightarrow output\ of\ the\ previous\ LSTM\ cell$
$x_t \rightarrow input\ at\ currenttimepoint$
$b_x \rightarrow biases\ for\ the\ respective\ gates\ (x)$
$c_t \rightarrow cell\ state\ (memory)\ at\ timepoint\ (t)$
$\tilde{c}_t \rightarrow represents\ candidate\ for\ cell\ state\ at\ timepoint\ (t)$

The ability of long-term information retention expands the network's attention compared to the basic RNNs. In addition to being able to access the previous cells states, it can also access the learning from the past cells, which enables referencing the context [19].

Because of the additional tanh and three sigmoid layers, compared to the original RNNs the drawback of LSTMs is the need for greater memory and computational power [21].

3 Methods

The main goal of the model is to predict the solvent accessible surface area of a drug on the nanoparticles in future timepoints with simulation, given the input of earlier timepoints. The input is based on molecular dynamics simulation of the nanoparticles. From these simulations, several parameters are extracted via pre-processing and fed into a model. The whole data creation, pre-processing and modelling pipeline is shown in figure 8.

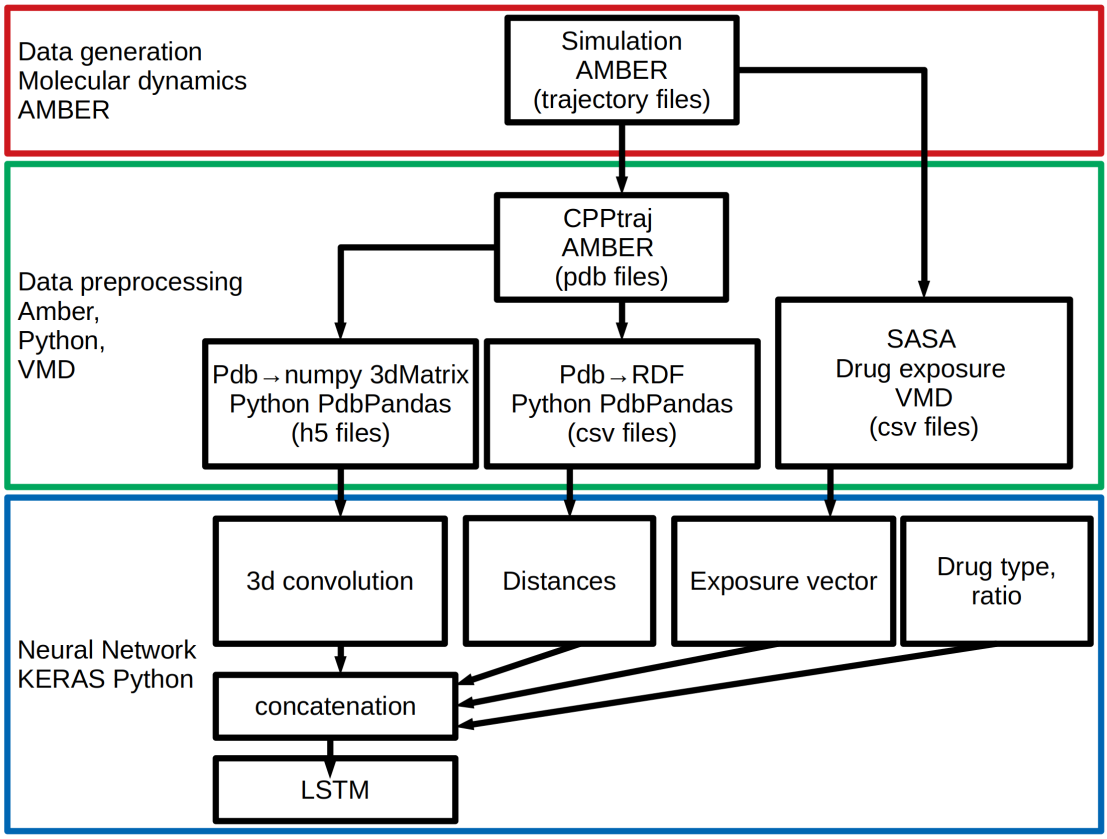

Figure 8: Whole pipeline: Molecular dynamics simulation (red box), data preprocessing (green box), neural network (blue box).

3.1 Software and hardware used

The practical work of this thesis was performed on the following clusters and programs:

1. The molecular dynamics simulations and conversions to PDB format were performed with Amber, a software suite for biomolecular simulation and analysis, version 18 [22] on a CSC cPouta cluster and Dione local cluster, co-owned by UTU/ÅA, both with Nvidia Tesla P100-PCIE 16 Gb graphical cards.

2. The drug SASA (Solvent Accessible Surface Area) exposure calculations were performed with VMD (Visual Molecular Dynamics for LINUXAMD64, version

1.9.4a43 [23]) software on CSC cPouta cluster.

3. NP visualisation was perfomed with UCSF Chimera (UCSF ChimeraX version 1.0, [24]) on a personal laptop with GeForce RTX 2060 6 GB graphical card.

4. The Deep Neural Networks were created with TensorFlow with Keras API in Python. The trial neural networks were tested on the aforementioned personal laptop, whereas longer trainings were performed on CSC cPouta and Dione clusters.

3.2 Molecular Dynamics Simulations

Molecular dynamics simulations (MDS) is a method for analysing the physical movements of atoms and molecules. In MDS, particles (atoms, molecules) interact for a given period of time, providing dynamic evolution of the whole system. Usually, the trajectories of the particles are determined by numerically solving Newton's equations of motion.

In this project, Amber, a software suite for biomolecular simulation and analysis, was used to perform MDS. PMEMD (Particle Mesh Ewald Molecular Dynamics) is the primary engine for running molecular dynamics simulations with Amber.

The input files for PMEMD, are parametric files, which describe the initial positions of the atoms, their associations and system temperature:

1. A primary file in Amber defines the system topology and the parameters for the force field for that system. It has an extension "*.prmtop", sometimes preceded by "*.solv.prmtop", which indicates the presence of a solvent, in this case water. The first few lines of such afile are presented in figure 9.

```
%VERSION  VERSION_STAMP = V0001.000  DATE = 02/02/21  13:18:59
%FLAG TITLE
%FORMAT(20a4)
default_name
%FLAG POINTERS
%FORMAT(10I8)
  363406      16  336807   23940   75180   27930  123480   53130       0       0
  704505  108138   23940   27930   53130      30      56      35      22       1
       0       0       0       0       0       0       0       2     101       0
       0
%FLAG ATOM_NAME
%FORMAT(20a4)
Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au
Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au
Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au
Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au  Au
```

Figure 9: First few lines of an exemplary ".prmtop" file, containing the system topology and the parameters defining the force field for that system. Au stands for NP gold core atoms.

2. Coordinate/restart file has initial coordinates of the atoms with the extension: "*.inpcrd". First few lines of such a file are presented in figure 10.

```
default_name
363406
 89.3858523 89.6374586 88.7229906 89.4423084 87.0865057 90.0490796
 91.7856115 88.0827590 88.4178267 91.0842057 89.4265429 91.0338715
 89.5658911 91.9644606 90.4027846 88.0710085 89.4856583 91.2758718
 86.9098765 88.1775952 88.8090278 89.2058136 87.3104567 87.0431966
 91.8618282 91.0973221 88.6369535 86.9860932 91.1921582 89.0281546
 90.7006962 89.7892589 86.1701095 89.3293962 92.1884116 87.3969017
 87.6874990 89.8483743 86.4121097 89.4992666 84.5347135 91.3753772
```

Figure 10: First few lines of exemplary ".inprcd" file, defining initial coordinates of atoms.

3. File "heat.in" defines the temperature of the system that is being controlled throughout the simulation, in a similar fashion as a thermostat - the kinetic energy of the particles is readjusted to keep a set temperature, as in figure 11.

```
&cntrl
 ioutfm = 1,     ! use netCDF
 ntxo = 2,       ! use netCDF for restart
 iwrap = 0,    ! wrap coordinates to original box (bad for diffusion...)
 cut = 9.0,    ! nonbonded cutoff, in A, def 8.0
 ntc = 2,      ! SHAKE (1 no, 2 H, 3 all)
 tol = 0.000001, ! geom tolerance (max <0.00005, def 0.00001)
 ntf = 2,      ! force evaluation, to use 2 with SHAKE
 ntpr = 500,     ! every NTPR steps print to out-info
 ntwx = 1000,    ! every NTWX to mdcrd-nc
 ntwr = 50000,     ! every X rst; anyway at nstlim...
 nstlim = 10000, ! nstep (in remd nstep between exchanges)
 dt = 0.001,     ! dt, in psec
 ig = -1,      ! random seed
 ntt = 3,      ! 3(Langevin), 0(const Etot, ntb<2)
 gamma_ln = 1.0,   ! collision frequency
 temp0 = 150.0,    ! T at which the system is to be kept
 ntr = 1,      ! crtesian restraint
 restraint_wt = 10,
 restraintmask = '!:WAT,Na+,Cl-',
/

&wt TYPE='TEMP0',
  value1=0.1, value2=150.0,
  istep1=0, istep2=10000,
  /
&wt TYPE='END'
/
```

Figure 11: Example of "heat.in" file, the set temperature of the system throughout the simulation.

4. File "density.in" defines the pressure and density of the system throughout the MDS, as exemplified in figure 12.

```
&cntrl
 ioutfm = 1,     ! use netCDF
 ntxo = 2,       ! use netCDF for restart
 iwrap = 0,    ! wrap coordinates to original box (bad for diffusion...)
 cut = 9.0,    ! nonbonded cutoff, in A, def 8.0
 irest = 1,    ! restart, req velocities in input
 ntc = 2,      ! SHAKE (1 no, 2 H, 3 all)
 tol = 0.000001, ! geom tolerance (max <0.00005, def 0.00001)
 ntf = 2,      ! force evaluation, to use 2 with SHAKE
 ntx = 5,      ! 1(read no velocities, def), 5(read all)
 ntb= 2,       ! PBC, 1(const V [def]), 2(const P)
 ntpr = 500,     ! every NTPR steps print to out-info
 ntwx = 5000,    ! every NTWX to mdcrd-nc
 ntwr = 100000, ! every X rst; anyway at nstlim...
 nstlim = 20000, ! nstep (in remd nstep between exchanges)
 dt = 0.001,     ! dt, in psec
 ig = -1,        ! random seed
 ntt = 3,        ! 3(Langevin), 0(const Etot, ntb<2)
 gamma_ln = 1.0,   ! collision frequency
 temp0 = 300.0,    ! T at which the system is to be kept
 ntp = 1,        ! [0]no_const_P, (1)isotropic, (2)anisotropic
 barostat = 1,     ! [1]Berendsen, (2)Monte Carlo barostat
 mcbarint = 50,   ! interval steps to MC volume changes
 taup = 5.0,     ! pressure relazation time in psec (1-5)
 ntr = 1,        ! crtesian restraint
 restraint_wt = 5,
 restraintmask = '!:WAT,Na+,Cl-',
/

&wt TYPE='TEMP0',
  value1=150.1, value2=300.0,
  istep1=0, istep2=20000,
  /
&wt TYPE='END'
/
```

Figure 12: Example of "density.in" file, the set pressure of the system throughout the simulation.

These input files were produced by our collaborator Marina Kovacevic from University of Novi Sad, Serbia.

PMEMD was run in CUDA version of the program, to use with parallelisation with GPUs. The MDS were run to simulate 300 nanoseconds (ns) of particle interactions. While PMEMD is running, the progress and speed of calculations can be inspected by examining the file "mdinfo". It contains information on how many steps have been completed, how many nanoseconds of simulation can be run per day with this system and how much time there is left before the specific job is finished [25].

Exemplary script to run PMEMD on CSC cPouta virtual machine can be found in the appendix listing 1.

The output of the MDS are a set of files containing trajectories with continuous information on the positions of the particles and the final parameters for every 10 ns of the simulation.

An example of the descriptive file after the completion of the molecular dynamic simulation, which describes the calculated parameters of the system, is found in figure 13 with 10 ns of simulation.

```
 NMR restraints: Bond =  20.409  Angle =   0.000  Torsion =   0.000
 ================================================================================
 | MC Barostat: Decreasing size of volume moves

 NSTEP =   30000  TIME(PS) = 105110.000 TEMP(K) =  300.26 PRESS =   0.0
 Etot   = -494575.2155 EKtot   =  128213.8203 EPtot    = -622789.0358
 BOND   =    5905.2547 ANGLE   =   21430.8566 DIHED    =   12881.1188
 1-4 NB =    5289.5349 1-4 EEL =    2626.8781 VDWAALS  =  -58751.0139
 EELEC  = -612188.9643 EHBOND  =       0.0000 RESTRAINT =     17.2994
 EAMBER (non-restraint)  = -622806.3352
 EKCMT  =       0.0000 VIRIAL  =       0.0000 VOLUME    = 2046811.3109
                        Density    =     1.4594
 --------------------------------------------------------------------------------

 NMR restraints: Bond =  14.773  Angle =   0.000  Torsion =   0.000
 ================================================================================

 NSTEP =   40000  TIME(PS) = 105130.000 TEMP(K) =  299.19 PRESS =   0.0
 Etot   = -495396.3953 EKtot   =  127755.9375 EPtot    = -623152.3328
 BOND   =    5927.0561 ANGLE   =   21625.7849 DIHED    =   12880.5916
 1-4 NB =    5238.9526 1-4 EEL =    2520.1320 VDWAALS  =  -58708.3735
 EELEC  = -612654.0298 EHBOND  =       0.0000 RESTRAINT =     17.5533
 EAMBER (non-restraint)  = -623169.8862
 EKCMT  =       0.0000 VIRIAL  =       0.0000 VOLUME    = 2047928.9211
                        Density    =     1.4586
 --------------------------------------------------------------------------------

 NMR restraints: Bond =  19.318  Angle =   0.000  Torsion =   0.000
 ================================================================================

 NSTEP =   50000  TIME(PS) = 105150.000 TEMP(K) =  300.04 PRESS =   0.0
 Etot   = -494592.1028 EKtot   =  128117.7188 EPtot    = -622709.8216
 BOND   =    5940.7691 ANGLE   =   21510.5865 DIHED    =   13005.4036
 1-4 NB =    5266.6650 1-4 EEL =    2588.3243 VDWAALS  =  -59247.4708
 EELEC  = -611790.7657 EHBOND  =       0.0000 RESTRAINT =     16.6664
```

Figure 13: A descriptive file after the completion of a molecular dynamic simulation, which describes the calculated parameters of the system.

3.3 Data pre-processing

3.3.1 SASA

The trajectories obtained from the MD simulations can be used directly for nanoparticle observation in programs such as Chimera or VMD.

Another, more important direct usage of the PMEMD trajectories for our studies, is the calculation of the solvent accessible surface area (SASA) of the drug vs the whole NP surface in VMD.

SASA is measured by rolling a ball of a diameter that is equal to the radius of the solvent, in this case water (1.4 Å), across the surface of the NP with a command in VMD:

"measure sasa 1.4 $all"

The SASA can be limited only to the SASA of the drug. The percentage of water-accessible surface of the drug vs the total water-accessible surface of the NP is calculated and used as the main parameter of the drug exposure in NP. The goal of the EVO-NANO project is to find this drug and drug vs supporting substance ratio.

An example of calculated SASAs of a NP with Panobistat in ratio vs supporting substance (1:1) can be seen in figure 14 and the corresponding graphs can be seen in figure 15 and figure 16.

timepoint	SASA drug	SASA total	SASA % (drug/total)
1	34802	80055	43.47
2	36069	80337	44.90
3	36489	80403	45.38
4	36177	79663	45.41
5	36829	80290	45.87
6	37944	81384	46.62
7	36309	80102	45.33
8	36670	79562	46.09
9	37620	80290	46.86
10	37902	79792	47.50
11	37329	79385	47.02
12	37011	79559	46.52
13	37900	78993	47.98
14	37602	79521	47.28
15	38541	79410	48.53
16	37593	79173	47.48
17	38176	79351	48.11
18	38543	80213	48.05

Figure 14: The first 18 timepoints of the SASAs, calculated for each ns of the MD simulation of a NP consisting of a gold core, Panobistat and a supporting substance (in ratio 1:1).

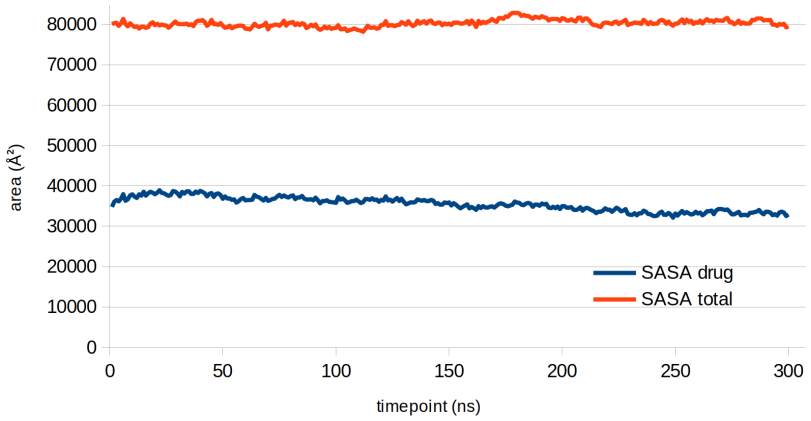

Figure 15: A graph of the drug and the total NP surface SASAs, calculated for all 300 ns of the MD simulation of a NP consisting of a gold core, Panobistat and a supporting substance (in ratio 1:1).

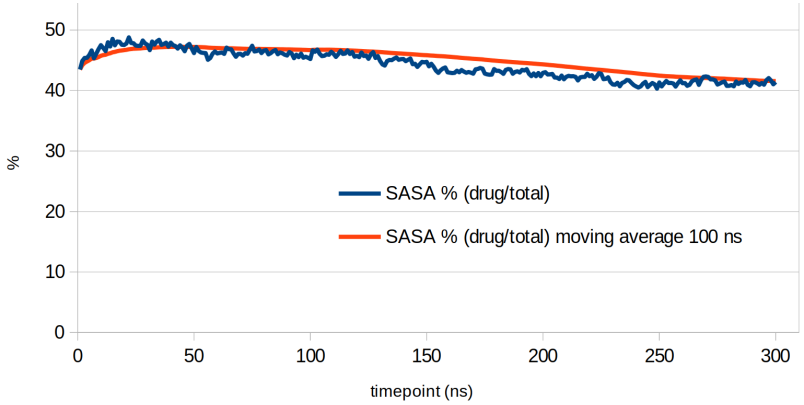

Figure 16: A graph of the percentage of a drug per total NP surface SASAs, calculated for all 300 ns of the MD simulation of a NP consisting of agold core, Panobistat and a supporting substance (in ratio 1:1) (in blue) and the calculated 100 ns moving average (in orange).

A script to run VMD SASA calculations on CSC cPouta virtual machine can be found in the appendix listing 2.

In the graphs in figure 15 and figure 16 (blue line), it can be observed that the SASA behaviour of the NP is stochastic, increasing and decreasing from timepoint to the next for several square Å. The stochasticity does not provide valuable information, but rather a noise that is more difficult to predict with DL. Therefore, the original SASA calculations were smoothed with a moving average for 100 timepoints with a Python Pandas library with *".rolling"* command, as can be seen in appendix 3. The smoothened data example can be seen in figure 16 (orange line).

3.3.2 PDB processing

In order to simplify the extraction of nanoparticle properties, Amber MDS trajectory files can be converted to PDB files. This conversion is performed with an Amber cpptraj program. An example bash script of this can be found in the listing 4.
PDB files are lists of atomic positions in 3D space at a given timepoint. Throughout this study, the conversion was for one PDB file per ns, so that 300 ns long MDS yielded 300 PDB files. Besides containing the information of the spatial positions of the atoms, PDB also contains the information on atomic associations, the covalent bonds that chain the atoms into the molecules. The first few exemplary lines of this file can be seen in figure 17.

```
CRYST1  137.455  137.455  137.455 109.47 109.47 109.47         1
ATOM    1 Au   NP   1    68.728 68.728 68.728 1.00 0.00      AU
TER    2     NP   1
ATOM    2 Au   NP   2    67.724 66.081 68.206 1.00 0.00      AU
TER    3     NP   2
ATOM    3 Au   NP   3    70.592 66.581 68.825 1.00 0.00      AU
TER    4     NP   3
ATOM    4 Au   NP   4    68.454 66.978 70.890 1.00 0.00      AU
TER    5     NP   4
ATOM    5 Au   NP   5    67.969 69.881 71.157 1.00 0.00      AU
TER    6     NP   5
ATOM    6 Au   NP   6    66.158 68.054 69.539 1.00 0.00      AU
TER    7     NP   6
ATOM    7 Au   NP   7    66.784 68.515 66.695 1.00 0.00      AU
TER    8     NP   7
ATOM    8 Au   NP   8    69.506 67.513 66.328 1.00 0.00      AU
TER    9     NP   8
ATOM    9 Au   NP   9    70.685 68.801 70.631 1.00 0.00      AU
TER   10     NP   9
ATOM   10 Au   NP  10     66.918 70.747 68.575 1.00 0.00       AU
TER   11     NP  10
ATOM   11 Au   NP  11     71.228 69.208 67.820 1.00 0.00       AU
TER   12     NP  11
ATOM   12 Au   NP  12     69.645 71.247 69.228 1.00 0.00       AU
TER   13     NP  12
ATOM   13 Au   NP  13     68.983 70.418 66.575 1.00 0.00       AU
TER   14     NP  13
ATOM   14 Au   NP  14     66.797 63.511 67.619 1.00 0.00       AU
```

Figure 17: First few lines of exemplary ".pdb" file of a NP consisting of a gold core, Panobistat and a supporting substance (in ratio 2:1) with atom associations and their coordinates in 3D space, where the units are Å, and finally the occupancy, temperature factor and the element name.

PDB files are the basis for further processing. The main goal of this thesis is to model the SASA percentage as mentioned in 3.3.1, with as little pre-processing of the data as possible and to relay the task of the important feature extraction to the extensive DL. The principle of least-processed data is presented in the mapping of the atomic positions from the simulated trajectories (or its PDB "snapshot") into a 4D array (3D positional data + timepoints). PDB files are opened with PandasPDB Python library as a Pandas data frame. The atoms associated into residues were assigned an "intensity" value based on their association with the drug (higher values close to 255) or supporting substance (lower values, closer to 100). An example of this can be seen in figure 18.

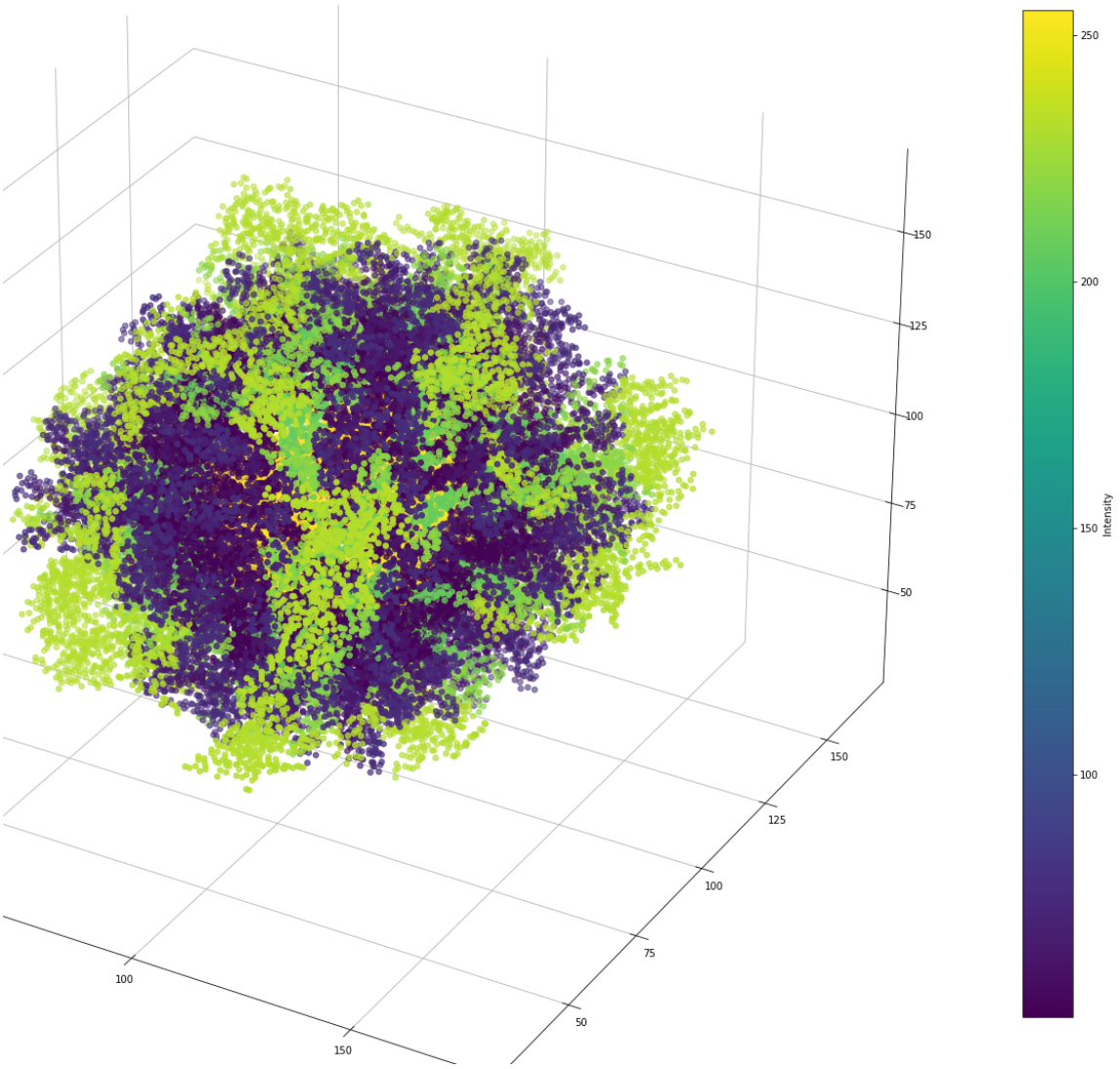

Figure 18: An example of mapping 3D atomic positions of NP into a 3D array. The intensities are assigned based on atomic associations with either a drug or a supporting substance. The example is made from Panobistat and a supporting substance (in ratio 1:2).

The initial mapping of all the NP atoms in PDB required a 3D matrix with a minimum size of 200 ("resolution") in order to avoid the overwriting of the atomic positions with close by atoms. However, the resulting matrices were too large for the GPU memory, also considering the time dimension (x 300). GPU memory is needed in training the DL with parallel processing offered by the GPU. Therefore, only the geometrically central atoms of each residues were considered, reducing the required minimum resolution (size of the 3D matrix) to 120. Because the reduced resolution of 120 was still too large for the GPU memory, an additional approach was to slice the spherical NP into 8 quadrants and flip them into the same orientation: upper front left. In this way, not only was the input made 8 times smaller (matrix of size 60), but also the number of inputs was 8 times greater. An example of this can be seen in figure 19.

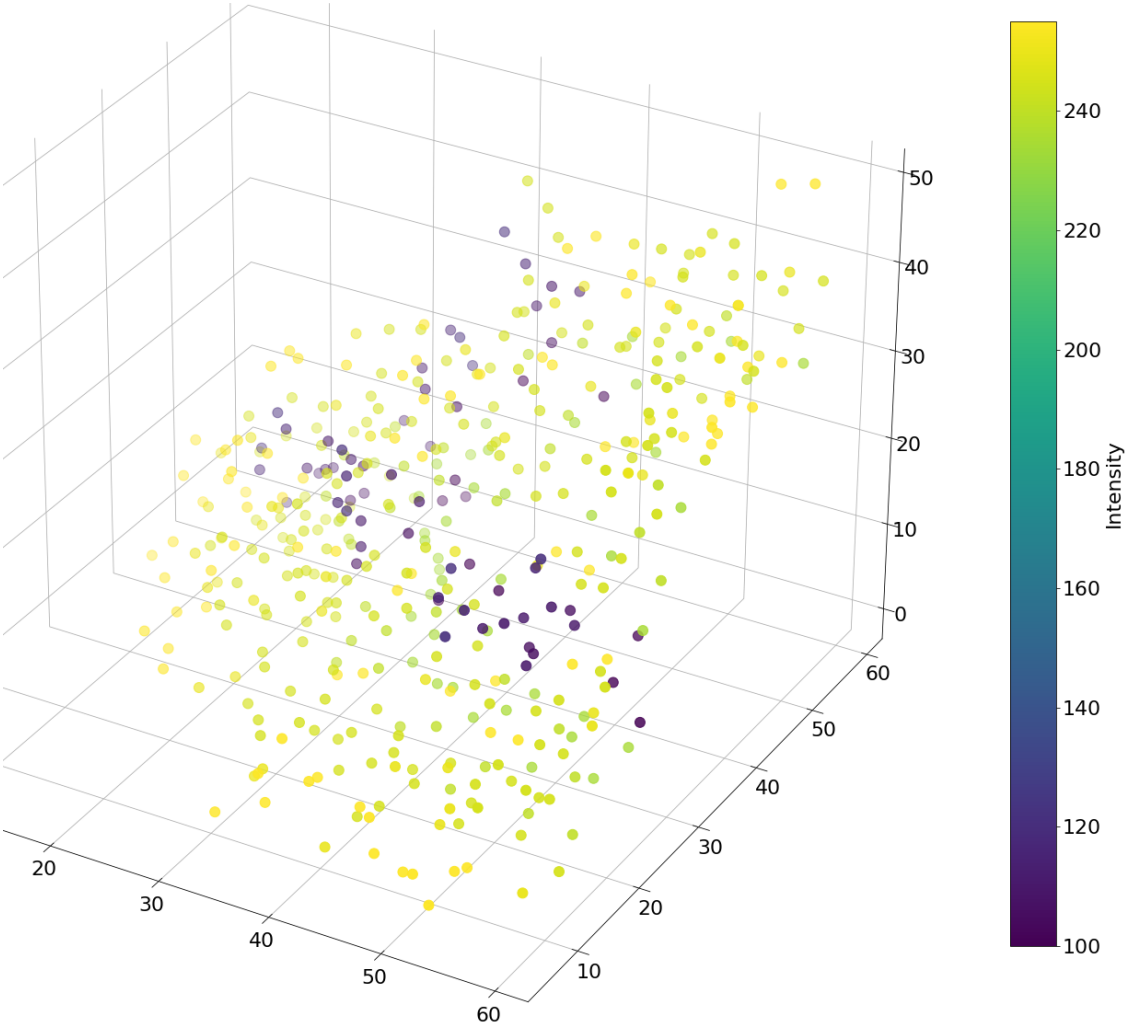

Figure 19: Reducing the size of the 3D matrix input by mapping just the centres of residues and slicing the NPs into individual quadrants. The example is made from quinolinol and a supporting substance (in ratio 5:1).

Each quadrant was additionally flipped in y and z axis by 90 degrees, resulting in a three-fold increase of the input. To additionally increase the amount of input, the coordinate positions were rotated in 15 steps by an angle of 25 degrees concurrently in y and z axis prior to the mapping into matrices, effectively producing 15 times more input. All geometric operations to increase the number of input can be seen in figure 20. The total input was 11 000 samples.

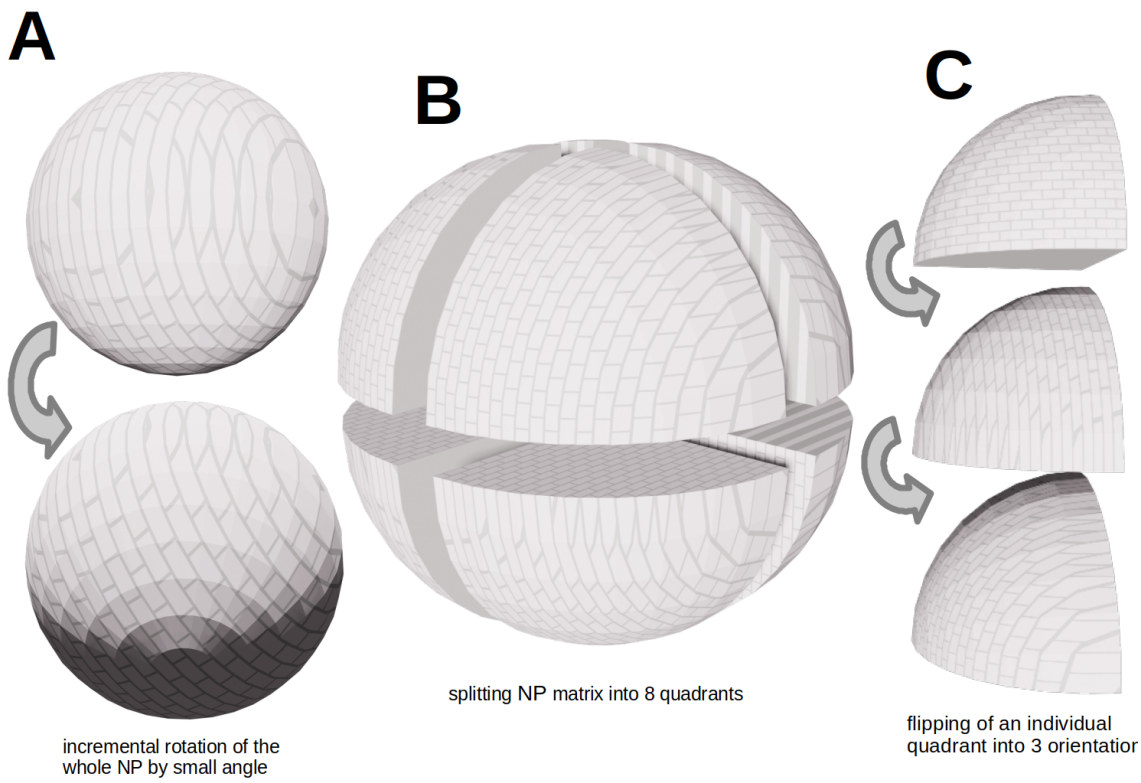

Figure 20: Presentation of all geometric operations to increase the number of samples: rotation by angle of 25 degrees (x15) concurrently in y and z axis (A), splitting the NP into 8 quadrants (B) and flipping each quadrant by 90 degrees in y and z axis (C). The operations are multiplicative, resulting in 360 fold increase of samples from initial whole NPs.

Another set of information from the atom positions in PDBs is a pre-extracted set of features, the statistics on the distances between the atoms. From the positions of the atoms, it is possible to calculate the Radial Distance Function (RDF). The RDF informs about the distance from the centre of the NP. The RDF is calculated by setting the central golden atom as an origin and converting the original Cartesian coordinates of the residues into polar coordinates and then measuring the radial component. Another set of measurements are the distances between molecules of the same type, to reveal a possible clustering of the residues. An example of the statistics, the mean, median, maximum and minimum distances, can be seen in figure 21.

All of the measured distance-statistics in time were averaged with a 100 timepoint window to reflect the averaging of the target data; the SASA percentage as mentioned in 3.3.1). The script used to map the PDB atomic positions into Numpy arrays, perform the rotations, quadrants selections, and calculate the radial and inter-residue distances can be found in The Python script used to build, train and use the model for predictions can be found in 5. The output numpy arrays were flipped and combined with distances tables and SASA calculations into Numpy compressed files for the input during training and prediction of the model with the script6.

	DrgRDMn	DrgRDMdn	DrgRDMx	DrgRDMin	DrgIntDMn	DrgIntDMdn	DrgPrxRDMn	DrgPrxRDMdn	DrgPrxRDMx	DrgPrxRDMin	DrgPrxIntDMn	DrgPrxIntDMdn	SupRDMn	SupRDMdn	SupRDMx	SupRDMin	SupIntDMn
2	58.58	58.63	62.51	54.00	82.85	78.25	52.63	52.68	56.51	47.34	74.56	70.31	48.89	49.04	54.02	41.44	68.53
3	56.51	56.57	61.71	45.68	79.94	75.50	51.16	51.16	56.60	41.62	72.48	68.38	47.06	47.29	52.50	39.88	65.90
4	55.77	56.10	61.73	43.66	78.84	74.54	50.62	50.58	57.09	40.12	71.63	67.67	46.43	46.95	53.56	36.35	64.84
5	55.20	55.60	61.53	44.57	78.06	73.77	50.18	50.48	55.89	40.19	71.00	67.07	46.09	46.46	51.91	36.23	64.57
6	54.77	55.12	62.75	42.37	77.34	73.22	49.93	50.27	57.16	39.35	70.72	66.77	46.12	46.60	53.41	34.22	64.51
7	54.43	55.00	62.66	40.53	76.84	72.76	49.62	49.93	57.06	37.22	70.20	66.34	45.98	46.50	53.56	34.15	64.21
8	54.02	54.58	62.05	42.32	76.24	72.24	49.51	49.64	56.35	40.31	70.07	66.21	45.75	46.19	54.00	34.55	63.70
9	53.67	54.32	62.98	41.98	75.72	71.78	49.22	49.38	56.72	38.09	69.68	65.84	45.56	46.27	53.77	34.69	63.51
10	53.56	54.21	61.80	42.15	75.59	71.65	49.16	49.48	55.87	38.09	69.55	65.78	45.35	46.12	53.08	33.16	63.39
11	53.26	53.93	61.04	41.23	75.18	71.26	48.92	49.38	55.90	37.82	69.22	65.46	45.11	45.44	52.19	32.88	63.19
12	50.64	51.51	61.75	35.21	71.19	67.75	47.19	48.00	55.80	30.61	66.43	63.12	45.24	45.40	53.82	34.91	63.20
13	50.51	51.09	60.56	34.92	71.06	67.58	47.11	47.75	55.11	32.07	66.34	63.03	45.05	45.56	51.64	34.91	62.98
14	50.43	51.10	60.80	34.45	70.89	67.45	47.11	47.79	55.22	30.68	66.30	63.01	45.23	45.48	52.09	34.98	63.21
15	50.52	51.03	60.10	34.82	70.99	67.55	47.14	48.00	54.97	30.52	66.39	63.05	45.11	45.59	52.55	36.48	62.97
16	50.39	51.23	59.34	34.48	70.86	67.42	47.00	47.89	54.79	30.98	66.17	62.91	45.05	45.45	52.42	34.40	62.85
17	50.34	51.04	60.60	35.29	70.84	67.33	47.01	47.93	55.57	30.69	66.21	62.90	45.20	45.45	52.71	36.50	63.14
18	50.28	50.97	60.34	35.77	70.70	67.20	47.01	47.76	55.92	30.86	66.23	62.85	45.11	45.74	52.61	33.96	63.09
19	50.16	50.61	60.45	33.49	70.59	67.08	46.91	47.56	55.65	29.95	66.09	62.76	45.03	44.81	52.32	34.07	62.99
20	50.21	50.95	60.15	33.42	70.59	67.16	46.94	47.79	54.51	28.71	66.08	62.81	45.02	45.49	52.12	32.73	63.06
21	49.98	50.51	59.93	34.24	70.15	66.83	46.81	47.48	55.27	30.36	65.87	62.60	45.11	45.41	53.21	34.68	62.89
22	49.99	50.80	58.91	33.53	70.25	66.87	46.66	47.46	55.37	30.05	65.68	62.44	44.89	45.30	51.14	35.56	62.75
23	49.80	50.55	60.50	34.05	69.98	66.62	46.59	47.14	55.69	29.79	65.53	62.35	45.05	45.79	52.16	36.16	62.74
24	49.88	50.76	60.32	35.01	70.05	66.72	46.63	47.38	54.75	29.74	65.65	62.40	45.18	45.55	52.02	35.82	63.02
25	49.95	50.52	60.30	34.41	70.18	66.82	46.77	47.72	55.57	29.69	65.80	62.59	45.26	45.71	52.19	36.12	63.20
26	49.77	50.39	62.51	34.19	69.93	66.57	46.61	47.37	55.98	29.76	65.49	62.36	45.26	46.00	52.71	37.16	63.17
27	49.74	50.50	62.27	34.33	69.77	66.53	46.72	47.48	56.04	30.18	65.64	62.51	45.37	45.91	52.79	33.40	63.29
28	49.65	50.18	60.29	31.86	69.71	66.40	46.53	47.47	56.00	31.42	65.32	62.22	44.97	45.50	53.08	33.81	62.68
29	49.69	50.17	60.10	31.29	69.73	66.45	46.62	47.73	57.44	31.29	65.50	62.34	45.24	46.00	53.38	35.27	63.01
30	49.57	50.31	59.82	31.40	69.53	66.30	46.47	47.62	56.44	31.16	65.25	62.14	45.16	45.32	53.04	34.98	62.89
31	49.49	50.19	60.31	31.82	69.38	66.15	46.45	47.45	56.18	31.22	65.17	62.11	45.16	45.70	53.78	35.50	62.74
32	49.49	50.02	60.08	31.25	69.42	66.15	46.50	47.26	56.26	30.97	65.24	62.16	45.38	46.23	52.36	36.88	63.09
33	49.45	50.20	59.74	31.37	69.36	66.08	46.40	47.23	55.93	31.07	65.11	62.01	45.44	46.18	53.77	35.88	63.00
34	49.51	49.83	61.21	30.47	69.53	66.17	46.47	47.29	56.00	30.04	65.21	62.12	45.42	45.89	53.27	36.59	63.23

Figure 21: The first few timepoints of an exemplary list of statistics of calculated distances of molecules, either from the core of NP - Radial Distance Function or between the molecules of same type, as a measurement of clustering. The example is made from Panobistat and a supporting substance (in ratio 2:1), units are Å.

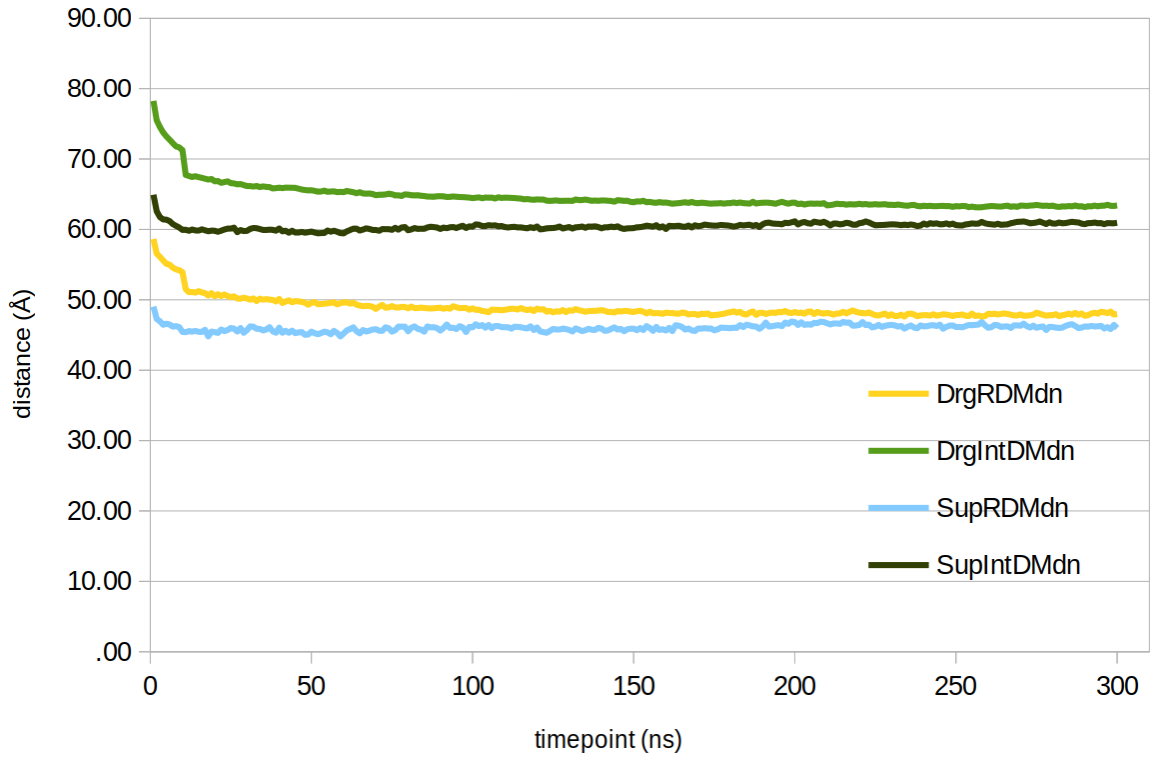

Figure 22: A graph from the list in figure 21, where Drg/SupRDMdn stands for the median of all of the radial distances of a drug (or a supporting substance) molecule from the centre of NP and Drg/SupIntDMdn stands for the median distance between the corresponding types of molecules. The example is made from Panobistat and a supporting substance (in ratio 2:1), units are Å.

The code to map atom positions in PDB files into 4D numpy arrays (3D positional

data + timepoints) and to measure the distances statistics, can be found in listing.

3.4 Technical DL definitions used

The dataset is split into three partitions: The largest part, around 80 percent, of the dataset is used for training of the neural networks. The weights and biases of the neural network are being updated with a training subset of the dataset. About 10 percent of the whole dataset is partitioned into a validation dataset. The validation dataset is used for evaluation of the model after predictions and tuning of the model's hyperparameters. Also, about 10 percent of the whole dataset is dedicated to a test subset. The test subset is only used once the training is completely finalised for the final evaluation. The ratios of training:validation:test can change in accordance to the size of the whole dataset. For large datasets with over 1 million samples, the ratio would be closer to 98:1:1 for training:validation:test, respectively [26].

Out of the many features in the data some have true correlations, whereas some features only appear to be correlated. However, these correlations are actually random. Training data may not be representative in its features in regard to testing data. If this is the case, training the model might capture the random correlations or noise which are occurring only within training data. When the model fits very well only to training data, but not to more general data, this is called overfitting. In contrast, when true correlations are not present well enough or the training is too short, the model cannot capture these correlations. This is called underfitting. In both cases, new datasets cannot be predicted well with under or overfit models [27]. To reduce overfitting a special technique called dropout is used, which drops out units and their connections during training randomly. Another approach to reduce overfitting is batch normalization, which normalizes the input layer by adjusting and scaling the activations. The data is fed into the neural networks in smaller parts as it cannot fit all at once due to memory constraints. These parts of the data that are fed at once are called batches. An epoch is one forward and backward passage of the whole training dataset through the neural network.

3.5 Models used

The general architecture used is composed of concatenated 3D CNN for NP structure input and dense layers for other types of input fed into Long Short-Term Memory (LSTM) RNN, as presented in figure 23.

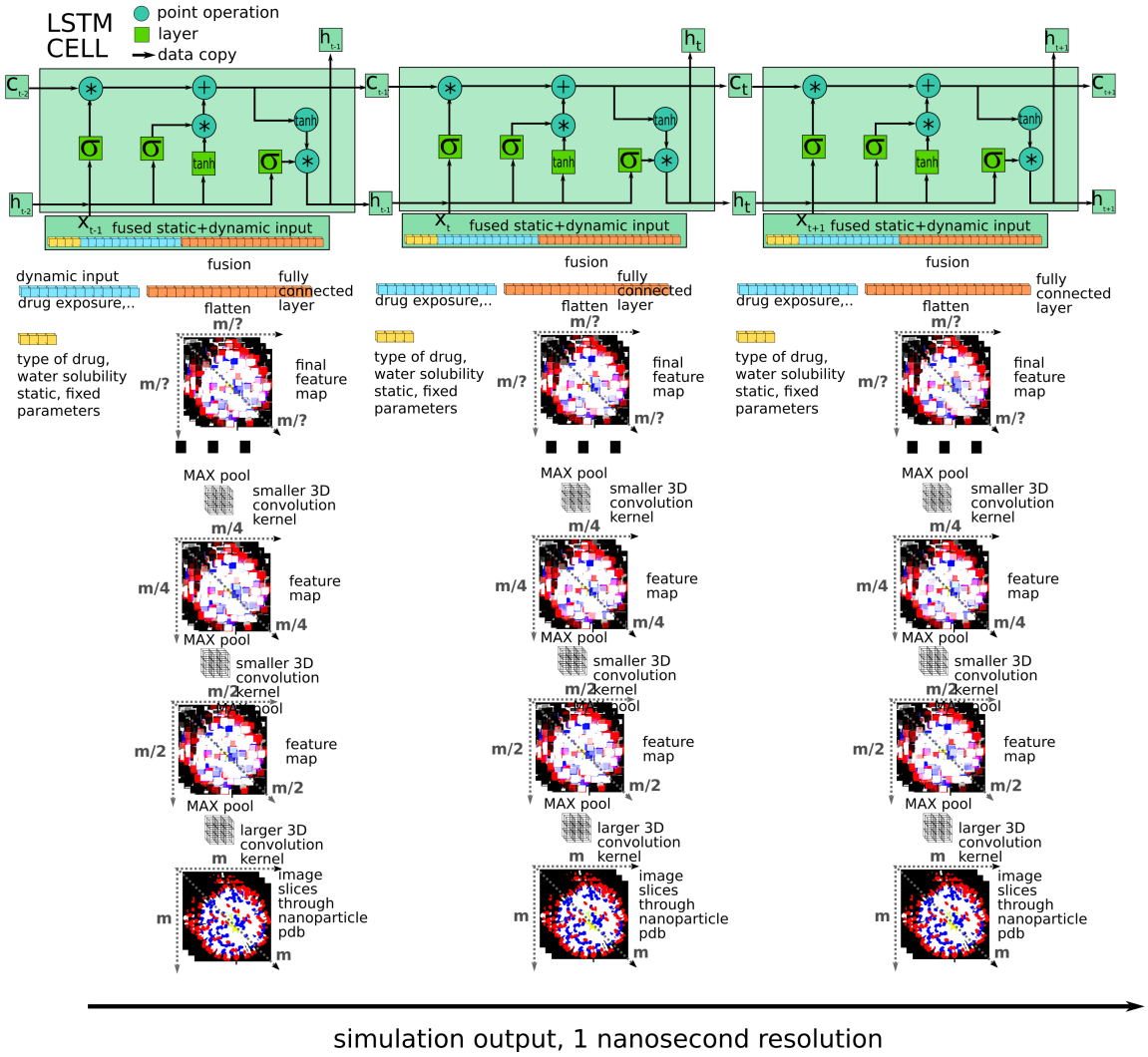

Figure 23: Graphical representation of model architecture inputs, concatenations and DL network components.

This model is composed of four types of input:

1. 3D positions of atoms in NP are input into two 3D convolutional layers with 60x60x60 input units. The 3D convolution layer has kernels of size three. First convolutional layer has 32 kernels, which double in each consecutive convolutional layer. Greater the number of different kernels greater number of distinct features can be detected.
   Each 3D convolution layer is followed by 3D max pooling layer, batch normalization layer and dropout layer with the dropout rate of 0.5 added to prevent overfitting. These are followed by a flattening layer to get a one-dimensional vector to be more easily combined with other input. The resulting vector is relatively large, so an additional dense layer is added to reduce the 3D input to the size order of other inputs.

2. SASA drug exposure input which is fed into a dense layer of the same size as the input.

3. Average distances of centres of molecules from the centre of the NP and distances between atoms which are fed into a dense layer of the same size as the input.

4. Drug type used and the ratio of drug vs supporting substance, also here the input is fed into the same-sized dense layer as the input.

All of the above layers are time distributed as they are input to LSTM layer, with timepoints distributed over time, each input in separate LSTM cell. The inputs cover 75 timepoints.

As can be seen in figure 24, the 3D convoluted layers are concatenated to dense layer with "drug exposure" input. The concatenated output of the aforementioned is, in turn, concatenated to a dense layer with "distances" input. This concatenated output is finally concatenated to a dense layer from "drug type" input. The final concatenation is the input into a single LSTM layer which is an output to two dense layers. The LSTM layer had 256 neurons, as well as the following dense layer. The second dense layer had 170 neurons. The very last layer is again a time distributed dense layer, to distribute the output in timely fashion to predict the SASA exposure at different timepoints in the future. The model architecture is shown in figure 24.

Adam (derived from ADAptive Moment estimation) was used as an adaptive learning rate optimisation algorithm. The Python script used to build, train and use the model for predictions can be found in 7.

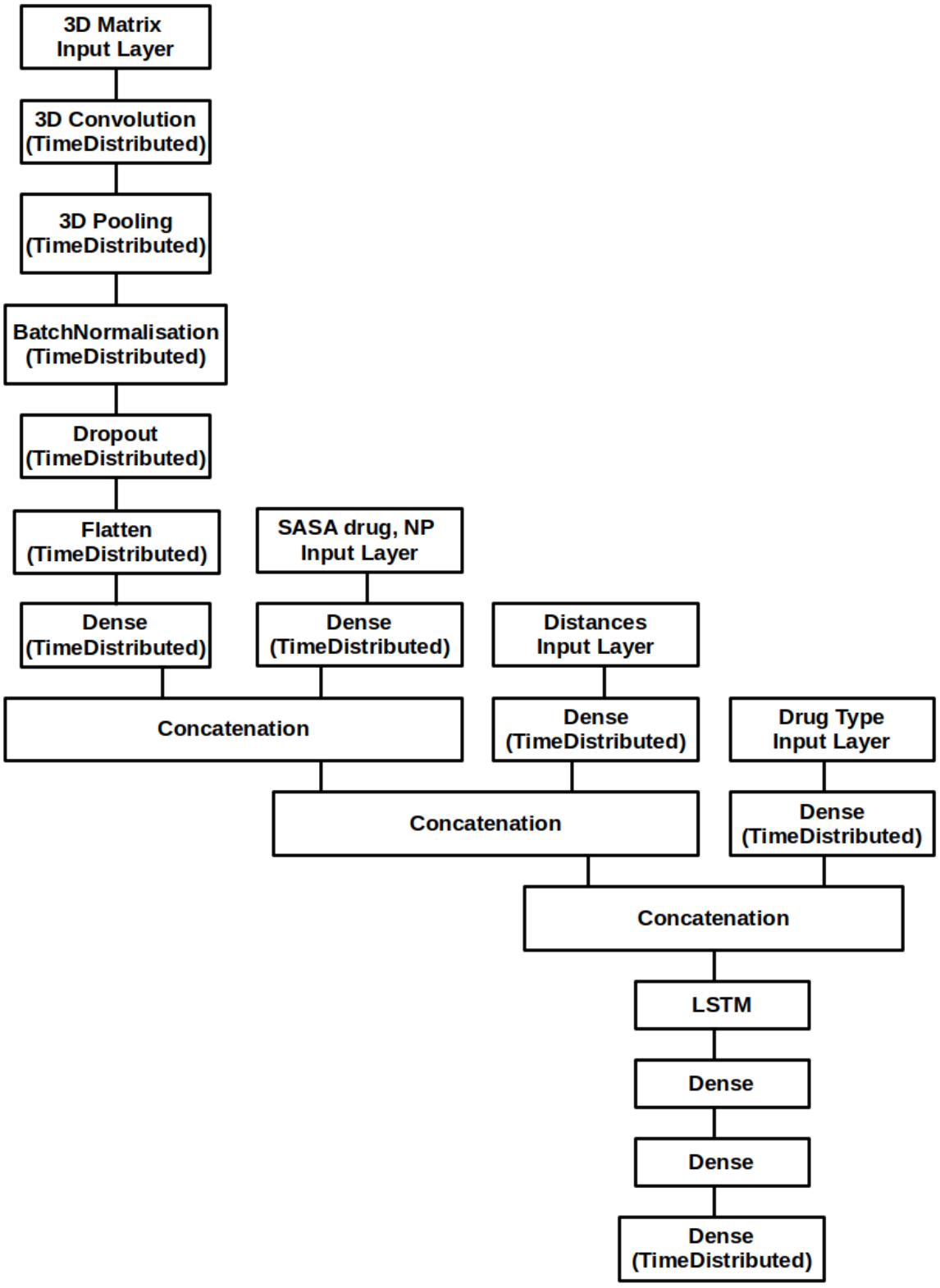

Figure 24: Abstraction of a simple model architecture with its inputs and concatenations.

4 Results

To obtain optimal architecture of the DL model, several architectures were tested. The basic architecture was composed of two 3D convolutional layers concatenated to other input, as described in subsection 3.5, but without the input of the calculated average distances between the types of residues and the average distances from the centre of the NP.

One epoch contained about 11 000 batches, each containing one training sample. The loss function on training data and validation data was observed after each epoch. The model summary can be observed in figure 25. After ten epochs, the training loss was 4.8 and the validation loss was 7.7 (MAE, % of drug vs total NP SASA), as can be seen in figure 26.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 75, 60, 60,	0	
time_distributed (TimeDistribut	(None, 75, 58, 58, 5	896	input_1[0][0]
time_distributed_1 (TimeDistrib	(None, 75, 29, 29, 2	0	time_distributed[0][0]
time_distributed_2 (TimeDistrib	(None, 75, 29, 29, 2	128	time_distributed_1[0][0]
time_distributed_3 (TimeDistrib	(None, 75, 29, 29, 2	0	time_distributed_2[0][0]
time_distributed_4 (TimeDistrib	(None, 75, 27, 27, 2	55360	time_distributed_3[0][0]
time_distributed_5 (TimeDistrib	(None, 75, 13, 13, 1	0	time_distributed_4[0][0]
time_distributed_6 (TimeDistrib	(None, 75, 13, 13, 1	256	time_distributed_5[0][0]
time_distributed_7 (TimeDistrib	(None, 75, 13, 13, 1	0	time_distributed_6[0][0]
time_distributed_8 (TimeDistrib	(None, 75, 140608)	0	time_distributed_7[0][0]
input_2 (InputLayer)	[(None, 75, 3)]	0	
time_distributed_9 (TimeDistrib	(None, 75, 60)	8436540	time_distributed_8[0][0]
time_distributed_10 (TimeDistri	(None, 75, 3)	12	input_2[0][0]
input_3 (InputLayer)	[(None, 75, 7)]	0	
concatenate (Concatenate)	(None, 75, 63)	0	time_distributed_9[0][0] time_distributed_10[0][0]
time_distributed_11 (TimeDistri	(None, 75, 3)	24	input_3[0][0]
concatenate_1 (Concatenate)	(None, 75, 66)	0	concatenate[0][0] time_distributed_11[0][0]
lstm (LSTM)	(None, 75, 256)	330752	concatenate_1[0][0]
dense_3 (Dense)	(None, 75, 256)	65792	lstm[0][0]
dense_4 (Dense)	(None, 75, 128)	32896	dense_3[0][0]
time_distributed_12 (TimeDistri	(None, 75, 1)	129	dense_4[0][0]

Total params: 8,922,785
Trainable params: 8,922,593
Non-trainable params: 192

Figure 25: Model summary of the most basic architecture.

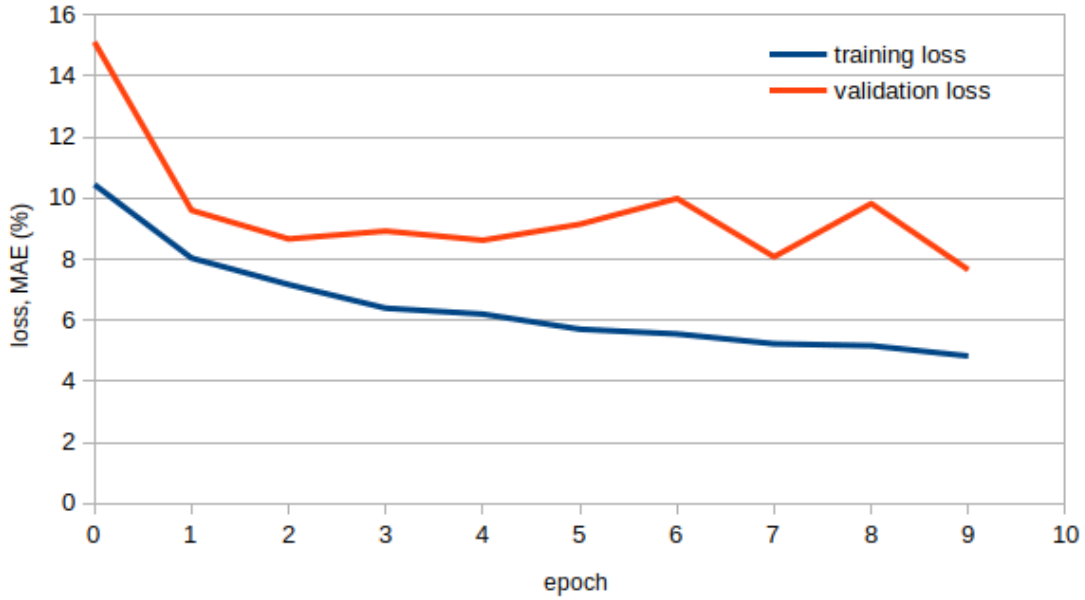

Figure 26: Training performance with basic input, averaged over 100 timepoints.

To improve the loss function, also the calculated average distances between the types of residues and the average distances from the centre of the NP input were included as input. This training resulted in lower validation and training loss after ten epochs, 4.5 and 4.3 MAE, respectively, compared to the training without distances input. The loss per epoch is shown in figure 26.

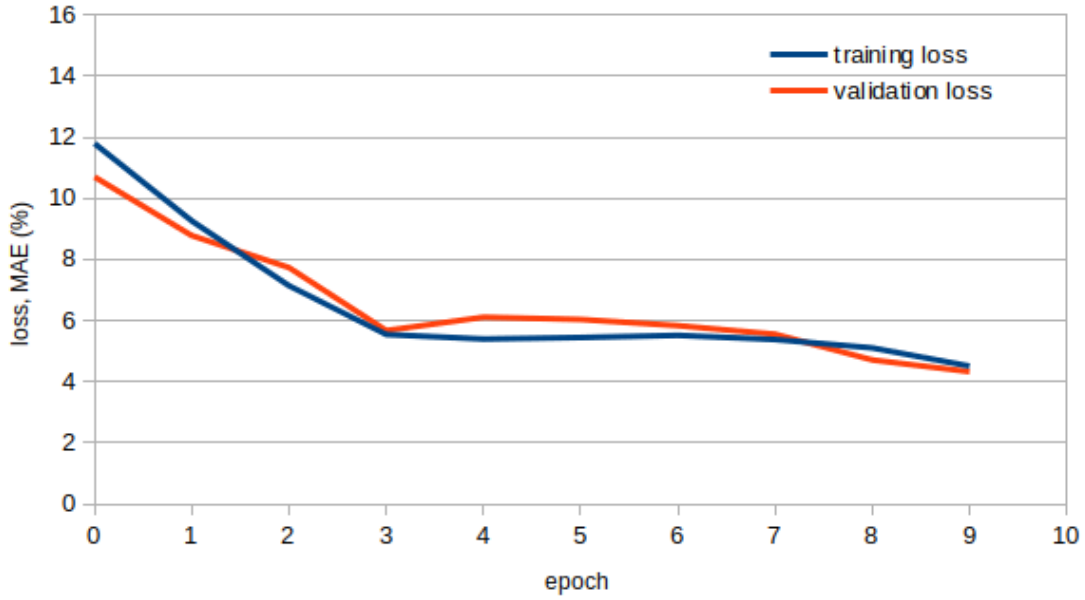

Figure 27: Training performance with basic averaged input with added calculated distances input.

To show the effect of smoothing (averaging) of the input data as a comparison,

also the training on the non-averaged data was performed. After ten epochs the validation and training loss were 6.0 and 6.0 MAE, respectively, as can be seen in figure 28.

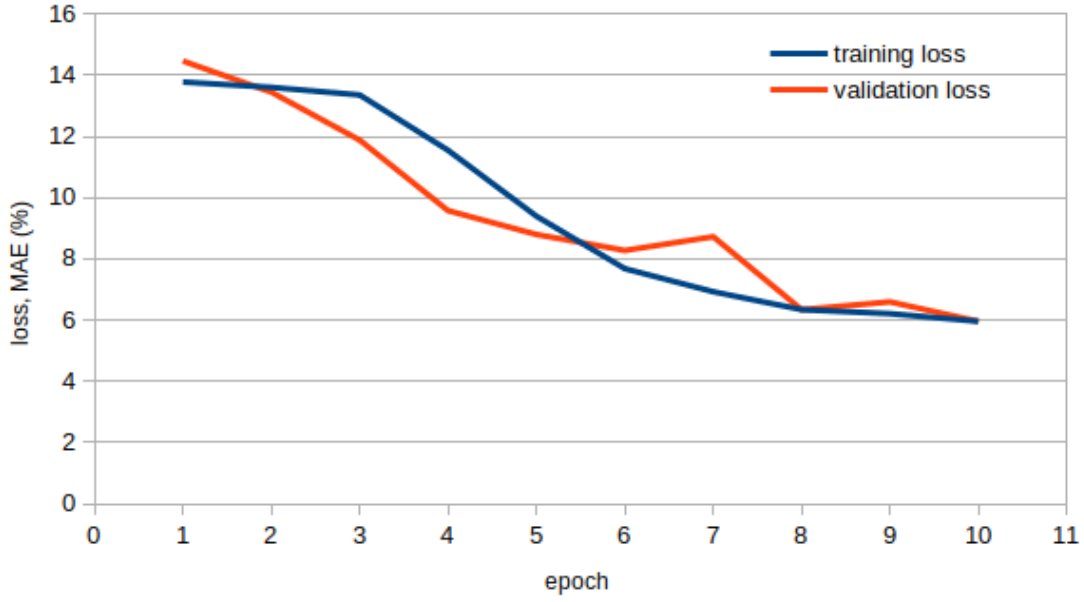

Figure 28: Training performance with added calculated distances input and no averaging.

Next, the effect of additional 3D convolutional layers was examined. After the initial two 3D convolutional layers each followed by a 3D max pooling layer, a batch normalisation layer and a dropout layer, also a third convolutional layer with aforementioned auxiliary layers was added. This type of architecture resulted in the validation and training loss to be 0.8 and 1.1 MAE, respectively, after ten epochs, as can be seen in figure 29.

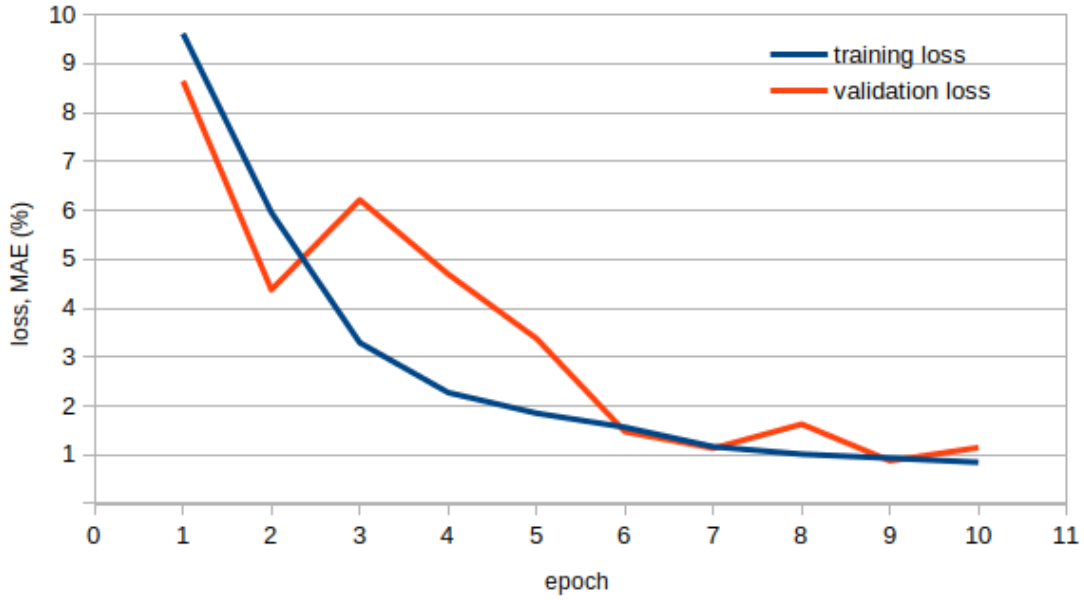

Figure 29: Training performance with added calculated distances input, averaging and an additional 3D convolutional layer.

Additional decrease of loss function was achieved by adding the fourth 3D convolutional layer. Such architecture resulted in the validation and training loss to be 0.7 and 0.8 MAE, respectively, after ten epochs, as can be seen in figure 30.

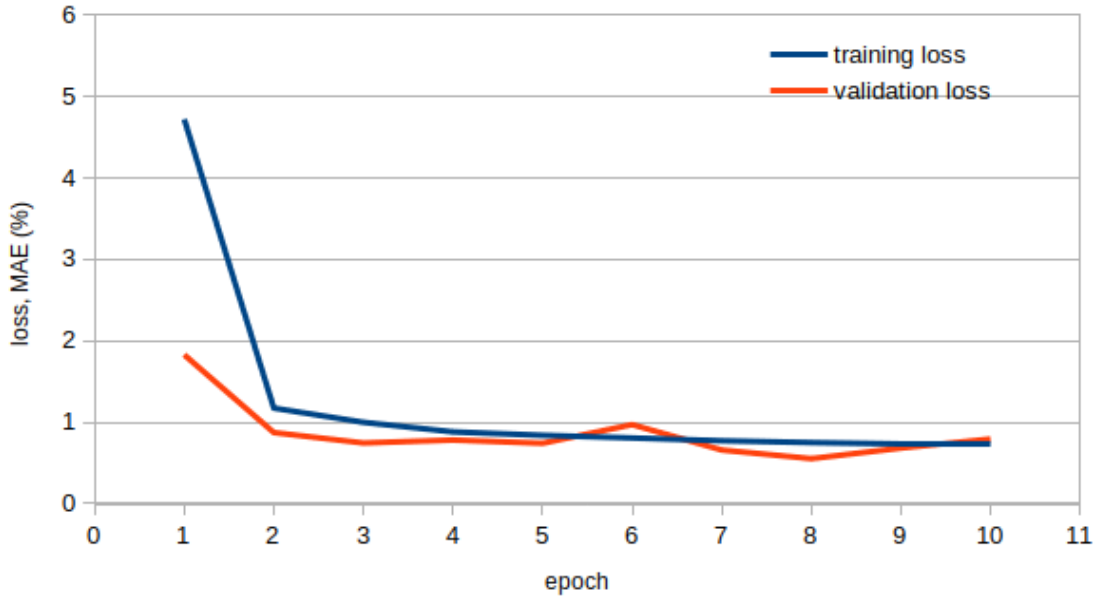

Figure 30: Training performance with added calculated distances input, averaging and additional two 3D convolutional layers.

By adding the fifth 3D convolutional layer with 256 kernels, the training loss

remained the same, whereas the validation loss decreased slightly to 0.7 MAE after ten epochs, as can be seen in figure 31.

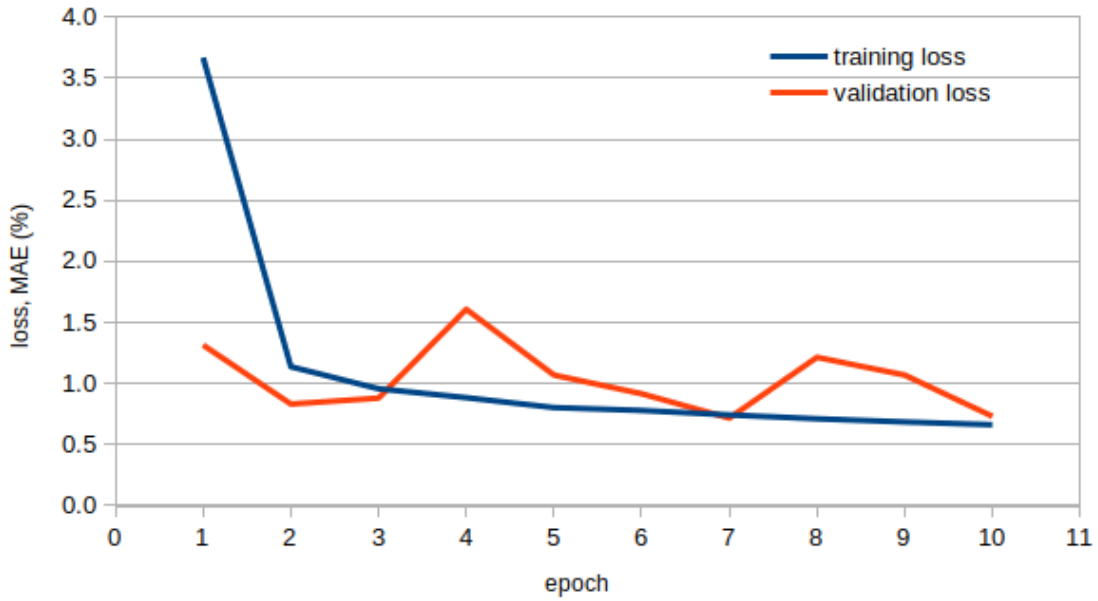

Figure 31: Training performance with added calculated distances input, averaging and additional three 3D convolutional layers.

The sixth layer of 3D convolution could not be applied, since already the five layers of the 3D convolution (and corresponding dropout layers) reduced the number of neurons to one per 3D matrix dimension and, thus, could not be convoluted and reduced further.

Next, the changing of the dropout rate was examined. The purpose of the dropout layer is to randomly set input units to zero with a given rate, which helps to prevent overfitting. When changing the dropout rate from initial 0.5 to 0.2, the validation and training loss were 0.6 and 0.8 MAE, respectively, after ten epochs, as can be seen in figure 32.

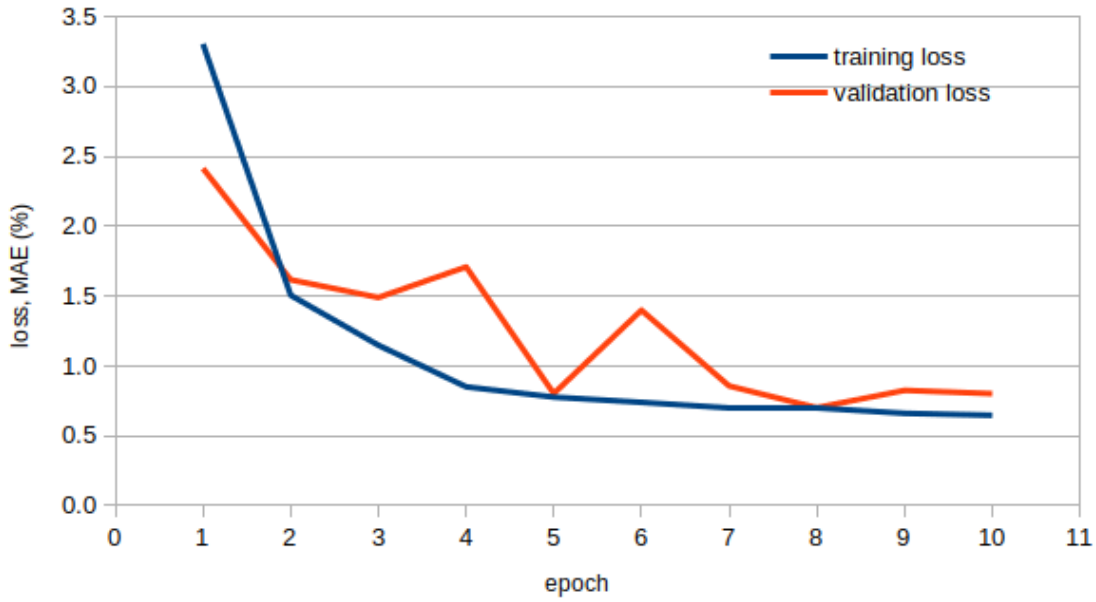

Figure 32: As in figure 31, but with a dropout rate of 0.2 instead of 0.5.

The effect of the number of kernels was examined by increasing the number of kernels in the last fifth 3D convolutional layer from 256 to 512. The dropout rate was 0.5. The model summary can be observed in figure 33. Such a setup of parameters resulted in the validation and training loss of 0.7 and 0.5 MAE, respectively, after ten epochs, as can be seen in figure 34. The 0.5 MAE of the validation loss after ten epochs is an overall best performance for all the tested models. Additional ten epochs lowered the validation and training loss to 0.6 and 0.4 MAE, respectively.

Layer (type)	Output Shape	Param #	Connected to
Input_3D (InputLayer)	[(None, 75, 60, 60,	0	
Conv3D_1 (TimeDistributed)	(None, 75, 60, 60, 6	896	Input_3D[0][0]
MaxPooling3D_1 (TimeDistributed	(None, 75, 30, 30, 3	0	Conv3D_1[0][0]
BatchNormalization_1 (TimeDistr	(None, 75, 30, 30, 3	128	MaxPooling3D_1[0][0]
Dropout_1 (TimeDistributed)	(None, 75, 30, 30, 3	0	BatchNormalization_1[0][0]
Conv3D_2 (TimeDistributed)	(None, 75, 30, 30, 3	55360	Dropout_1[0][0]
MaxPooling3D_2 (TimeDistributed	(None, 75, 15, 15, 1	0	Conv3D_2[0][0]
BatchNormalization_2 (TimeDistr	(None, 75, 15, 15, 1	256	MaxPooling3D_2[0][0]
Dropout_2 (TimeDistributed)	(None, 75, 15, 15, 1	0	BatchNormalization_2[0][0]
Conv3D_3 (TimeDistributed)	(None, 75, 15, 15, 1	221312	Dropout_2[0][0]
MaxPooling3D_3 (TimeDistributed	(None, 75, 7, 7, 7,	0	Conv3D_3[0][0]
BatchNormalization_3 (TimeDistr	(None, 75, 7, 7, 7,	512	MaxPooling3D_3[0][0]
Dropout_3 (TimeDistributed)	(None, 75, 7, 7, 7,	0	BatchNormalization_3[0][0]
Conv3D_4 (TimeDistributed)	(None, 75, 7, 7, 7,	884992	Dropout_3[0][0]
MaxPooling3D_4 (TimeDistributed	(None, 75, 3, 3, 3,	0	Conv3D_4[0][0]
BatchNormalization_4 (TimeDistr	(None, 75, 3, 3, 3,	1024	MaxPooling3D_4[0][0]
Dropout_4 (TimeDistributed)	(None, 75, 3, 3, 3,	0	BatchNormalization_4[0][0]
Conv3D_5 (TimeDistributed)	(None, 75, 3, 3, 3,	3539456	Dropout_4[0][0]
MaxPooling3D_5 (TimeDistributed	(None, 75, 1, 1, 1,	0	Conv3D_5[0][0]
BatchNormalization_5 (TimeDistr	(None, 75, 1, 1, 1,	2048	MaxPooling3D_5[0][0]
Dropout_5 (TimeDistributed)	(None, 75, 1, 1, 1,	0	BatchNormalization_5[0][0]
Flatten_3D (TimeDistributed)	(None, 75, 512)	0	Dropout_5[0][0]
Input_SASA (InputLayer)	[(None, 75, 3)]	0	
Dense_3D (TimeDistributed)	(None, 75, 60)	30780	Flatten_3D[0][0]
Dense_SASA (TimeDistributed)	(None, 75, 3)	12	Input_SASA[0][0]
Input_Distances (InputLayer)	[(None, 75, 42)]	0	
Concatenate_3D-SASA (Concatenat	(None, 75, 63)	0	Dense_3D[0][0] Dense_SASA[0][0]
Dense_Distances (TimeDistribute	(None, 75, 42)	1806	Input_Distances[0][0]
Input_DrugType (InputLayer)	[(None, 75, 7)]	0	
Concatenate_3DSASA-Distances (C	(None, 75, 105)	0	Concatenate_3D-SASA[0][0] Dense_Distances[0][0]
Dense_DrugType (TimeDistributed	(None, 75, 3)	24	Input_DrugType[0][0]
Concatenate_3DSASADistances-Dru	(None, 75, 108)	0	Concatenate_3DSASA-Distances[0][0 Dense_DrugType[0][0]
LSTM (LSTM)	(None, 75, 256)	373760	Concatenate_3DSASADistances-DrugT
Dense_afterLSTM_1 (Dense)	(None, 75, 256)	65792	LSTM[0][0]
Dense_afterLSTM_2 (Dense)	(None, 75, 128)	32896	Dense_afterLSTM_1[0][0]
Dense_afterLSTM (TimeDistribute	(None, 75, 1)	129	Dense_afterLSTM_2[0][0]

Total params: 5,211,183
Trainable params: 5,209,199
Non-trainable params: 1,984

Figure 33: Model summary of the best performing setup, performance of which can be seen in figures 34 and 35.

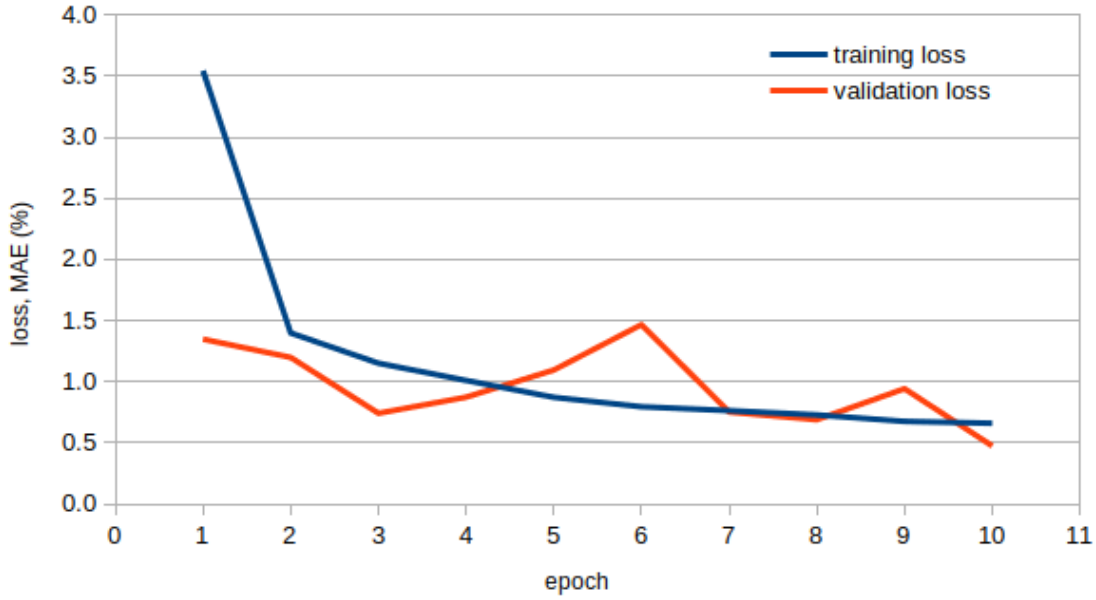

Figure 34: As in figure 31, but with 512 kernels in the last 3D convolutional layer instead of 256.

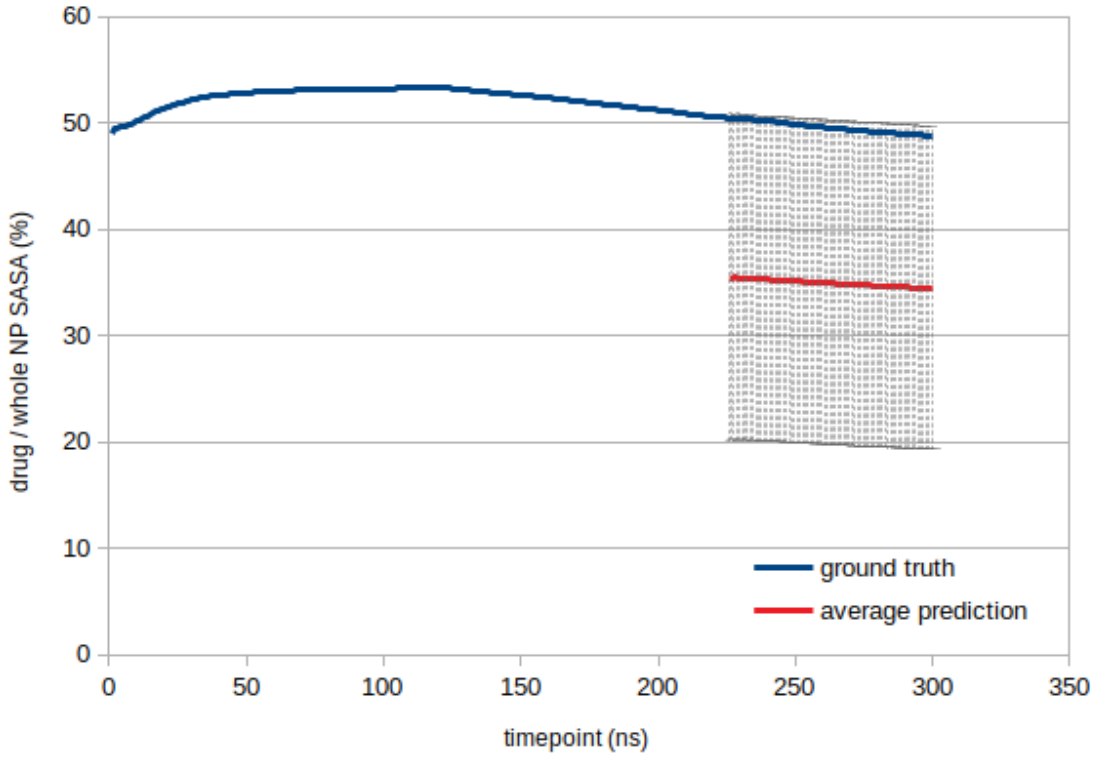

Figure 35: Performance of the model from figures 33 and 34 after 20 epochs training, the average of 14 predictions of drug vs whole NP SASA percentage in red with one standard deviation in gray and ground truth in blue. The 14 predictions were based on the 14 quadrants with different angles of the same NP composition, which were not used in the training process. The NP is composed of Panobistat and a supporting substance (2:1).

The effect of the additional dense layer with 170 neurons after LSTM layer, was examined. Training on this architecture resulted in the validation and training loss of 0.7 and 0.9 MAE, respectively, after ten epochs, as can be seen in figure 36.

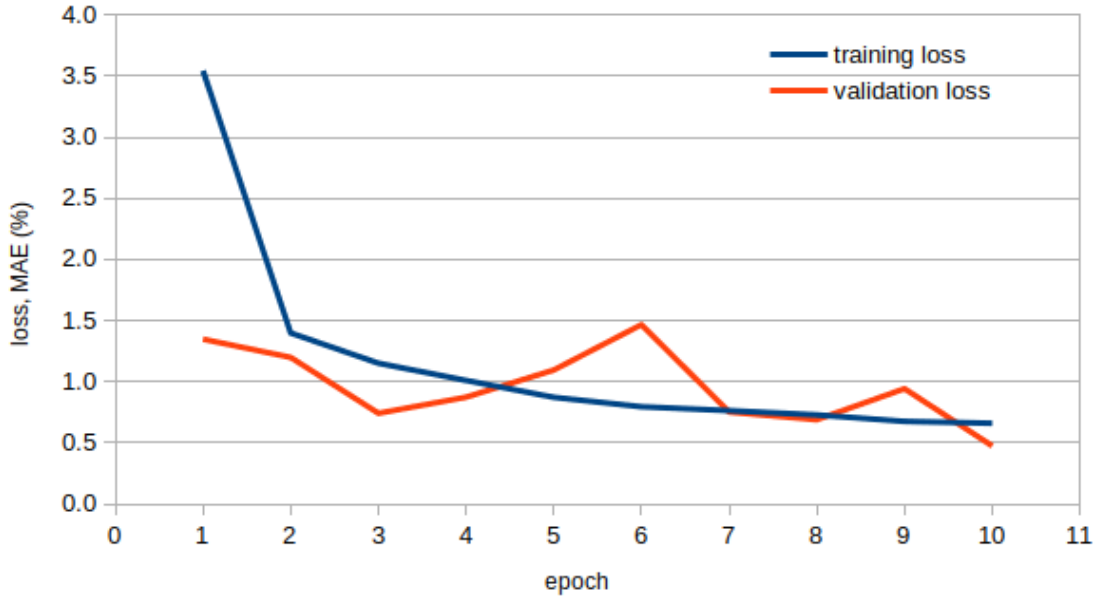

Figure 36: As in figure 34, but with an additional dense layer with 170 neurons after LSTM.

Next, the effect of the type of loss function on the training was inspected. A loss function commonly used for regression is the mean squared error (MSE). To be able to compare this to the previous results, also MAE was calculated. The usage of an MSE as a loss function resulted in the validation and training loss of 0.8 and 1.8 MSE, respectively, and validation and training loss of 0.7 and 1.1 MAE, respectively, as can be seen in figure 37.

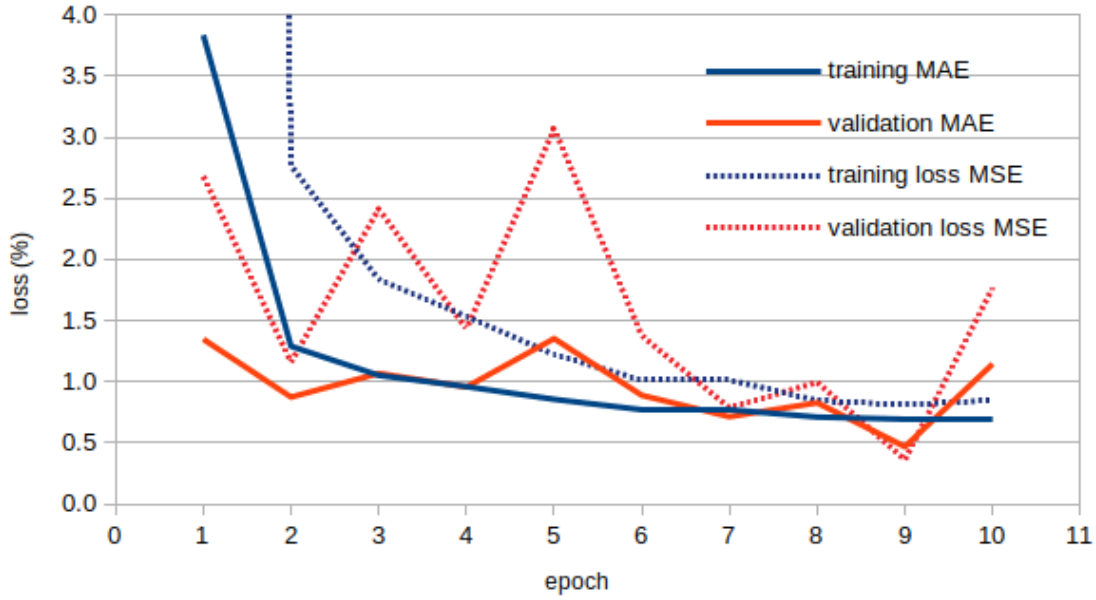

Figure 37: As in figure 36, but with the mean squared error (MSE) for loss function instead of the mean average error (MAE). MAE was calculated in addition for easier comparison.

Finally also the test of possible smaller number of input timpoints to predict also smaller number of final timepoints was examined. The goal of the study was to shorten the simulations as much as possible and be able to predict the final few timepoints. When only first 10 timepoints as opposed to 75 were used as input to predict last 10 timepoints (as opposed to 75), the validation and training loss after 10 epochs were 0.7 and 1.1 of MAE respectively. The result can be seen in figure 38.

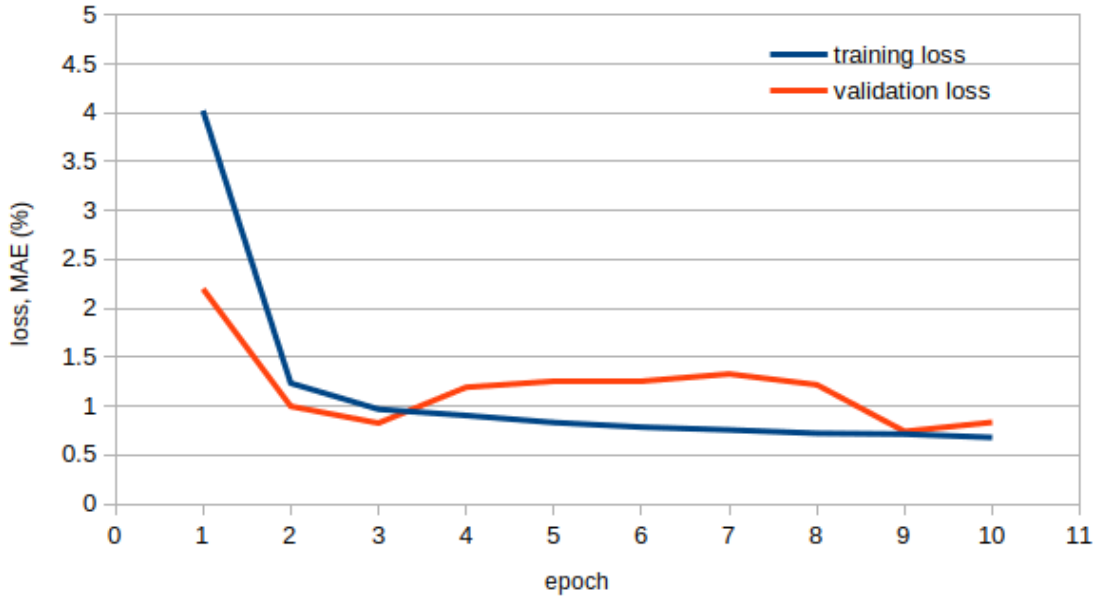

Figure 38: As in figure 34, but with 10 timepoints as prediction for 10 last timepoints SASA ratio, as opposed to 75 in prior cases.

5 Discussion

The aim of this thesis was to establish a set of procedures to prepare the input data for a deep learning model and to develop the deep learning model itself, which would predict the properties of a medical nanoparticle.

The aim of the running molecular dynamics simulations was achieved. In total, 107 simulations were run. Of these, 63 simulations of nanoparticles were run in basic conditions by 27 degrees Celsius and without added ions. The NPs were covered with seven different drugs in various ratios to the supporting substances. Additional 45 simulations were done with the same drugs, but with smaller number of ratios (3:1, 1:1, 1:3) for 27 degrees with ions and 37 degrees with and without ions present in the NP solution. The aforementioned 63 simulations in basic conditions were used for the creation of the deep learning dataset.

In order to create a deep learning dataset, first the comparable solutions used by other researchers were examined. As the goal was to extract features from 3D objects (nanoparticles), one of the closest experiments where 3D convolution was used was finding tumours in computer tomography scans of lungs (citation). There, however, the largest 3D images used were maximally twenty 8-bit voxels in each dimension and no time dimension was used that could additionally increase the size of an individual sample.

Due to a previous experience in the usage of a software for the visualisation of large macromolecular structures, Chimera software was used to visualise the nanoparticles. A script was created in Python to slice through a nanoparticle in stepwise fashion in Chimera, while saving each clip as a separate image, as can be seen in figure 39. However, after optimising a script, the output was not satisfactory. The depth resolution could not reach the resolution of an x,y -plane. The properties of a human friendly visualisation of objects in Chimera created artifacts that added noise and could interfere with the deep learning method. For example, the atoms are coloured with a set colour, however, their edges change the intensity depending on the depth of a slice, which can not be zero. In addition, the created set of slices per each timepoint of the simulation required both a large hard disk storage capacities as well as a large running memory for the graphics card. The optimisation of this approach required time and effort that could have been used for the modeling itself.

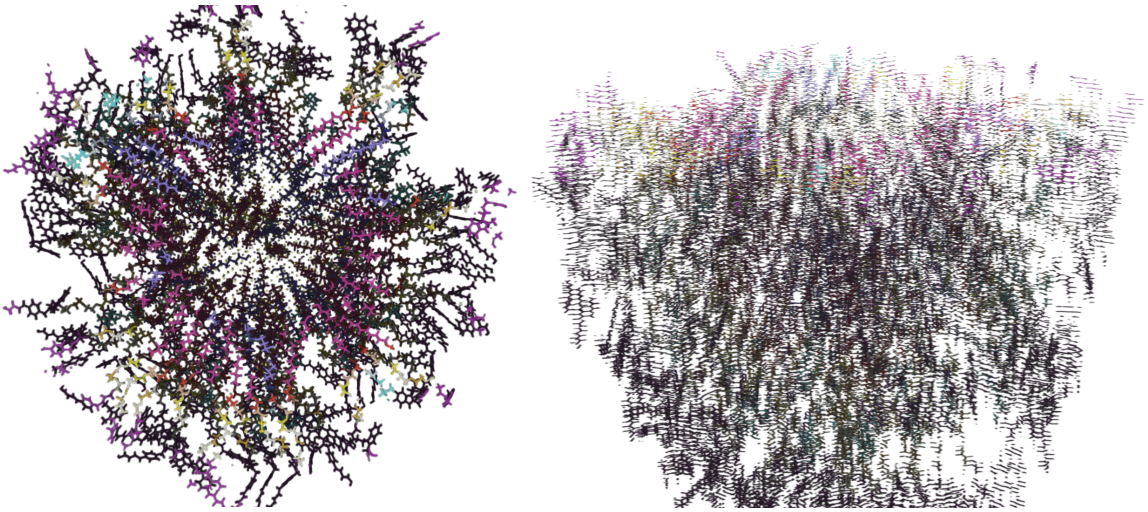

Figure 39: The initial approach of representing nanoparticles in 3D space as a stack of 2D images made with a combination of Python and Chimera. The approach is similar to the 3D CT scans of lungs as a known case of 3D convolutional DL used for detecting tumours. Left is top-down view, right is side view.

The impasse of representing the nanoparticle as a stack of slices led to a search for an alternative, more direct approach to map the atomic coordinates from pdb files directly into a 3D matrix representation. The solution found was a Python library named BioPandas and its submodule PandasPdb. This approach allowed first the loading of the pdb files as pandas dataframes and then selective filtering of the atoms of interest. The atom coordinates from the pdb are translated into a position in the 3D matrix. Similarly as colour coding in an image representation of a nanoparticle in figure 1, the value stored in a given matrix position could encode the type of the atom, whether it is a part of the drug molecule itself, a part of the PEG chain connecting the drug to the gold core or the supporting substance or its PEG chain. The encoding of such properties was also possible with the combined pandas dataframe and the numpy matrix approach, as can be seen in figure 18.

To perform the direct mapping of the atomic coordinates into the 3D numpy matrix via PandasPdb, the matrix of a sufficient size had to be created in order not to overwrite the already occupied cells in the matrix with the nearby atoms. Empirically, the size of 200 was concluded. However, as already mentioned in 3.3.2, the produced input was too large to be fed into the 3D convolutional LSTM model. Therefore, several actions of additional pre-processing were taken. Firstly, only the geometrically central atoms of the molecules were considered to be mapped into 3D matrices. This reduced the minimal non-overlapping matrix size to 120. As this step was not sufficient, additionally only the individual quadrants of the matrix were fed separately into the neural network. The quadrants of the same matrix were flipped for all to match the same orientation as to normalise them. The individual quadrant matrix size was reduced to 60 cells in each dimension, which allowed the training of the deep learning model on the graphs cards at disposal.

The separation of the matrix into eight quadrants produced an eight-fold increase of the input data. Additional increase of the data was augmentation of the whole

NP atom coordinates prior to the mapping into a matrix by incrementally rotating the NP (atomic coordinates) by a small angle. Through these multiplicative steps, the total number of samples was increased by 360-fold.

The goal of the Evonano project is to develop nanoparticles that deliver anti-tumour drugs to the site of the tumour. It is desirable that as much drug as possible is delivered per nanoparticle. The varying drugs used can have varying levels of hydrophobicity, which lowers their exposure to a water-based environment, like blood, as well as to the targeted receptors of the tumour. To compensate these properties, a supporting substance is added to a nanoparticle. By empirically trying out various ratios of drug vs supporting substance, the optimal ratio is obtained.

At the beginning of the study, the metric for the drug exposure given by our Serbian collaborators was the ratio of the drug molecules which had no neighbouring molecules in a given radius vs those that did. Later, this metric was replaced by the solvent accessible area (SASA) of a drug relative to SASA of the whole nanoparticle. This thesis developed a non-GUI automatised approach to calculate this metric based on instructions from the aforementioned collaborators. This metric is more widely used and faster to calculate.

SASA ratio is the main objective of the DL prediction. If it were possible to predict the SASA ratio after 300 ns of simulations based on the first few timepoints (ns), then the actual simulation could be shortened to those few initial timepoints. In the first models (not presented), only the next timepoints were predicted. Those models performed relatively well. However, as already mentioned, the goal of this thesis was to predict the last portion of the simulation based on the first portion. The last model settled to the 75 last timepoints based on the first 75 points, so that the first timepoint would be predictive of the 226th timepoint. The reason for this number of timepoints was the limitations of the graphics card memory.

Effects of preprocessing were examined, where the input was averaged with 100 timepoint moving average and compared to non-averaged unprocessed input. The smoothing of data by averaging showed to have improved the learning rate and resulted in a smaller validation loss.

To find the optimal architecture number of 3D convolutional layers and corresponding pooling, batch normalisation and dropout layers was examined. Adding maximum possible number of 3D convolutional layers, given the input dimensions prooved to be best performing. The initial dimensions of one quadrant matrix were 60x60x60, after each 3D convolutional layer the dimension was reduced by two, and after each max pooling layer by half, which allowed for five 3D convolutional and max pooling layers to finaly extract the matrix dimensions to 1x1x1.

The dropout rate was examined as it helps to prevent overfitting higher dropout rate of 0.5 resulted in smaller validation loss compared to lower dropout rate of 0.2. Greater number of kernels in 3D convolutional layers resulted in smaller validation loss, as well as using of mean average error comparred to mean squared error as loss function.

Also the length of input in regard to number of timepoints was examined. The goal of the study was to shorten the simulations as much as possible and be able to predict the final few timepoints. Using 75 initial timepoints to predict last 75 timepoints resulted in smaller validation loss compared to 10 timepoint input-output.

The best result of predicting the SASA ratio in the last 75 timepoints of the simulation based on the first 75 timepoints (or 226 timepoints in the future), can be seen in figure 35. This is still far from the level where it could be used beside the simulations itself. It does, however, show a similar trend in the decline of the SASA ratio through time. It was also improved with a larger number of learning epochs. It might be possible to improve it further with an even larger number of epochs. If the memory limitations could be overcome to input whole NP matrices, the prediction might improve as well, since the range of error (standard deviation) observed in figure 35 stems from the predictions of 14 quadrants at different angles of the same NP.

Also, the simplification of the constituent molecules of the nanoparticles to their central atoms might have brought a penalty to the predictive power of the model.

The model could be upgraded to include an attention layer that would report on which input had the greatest contribution in the prediction of the SASA ratio. This would create so called explainable artificial intelligence (XAI), which is on the rise for its understandability, as opposed to the classical "black box" deep learning.

6 Conclusions

The main aims of the study were achieved:

1. Running of molecular dynamics simulations on CSC and local cluster, gathering and sharing of obtained MDS trajectories.

   The basis code for running MDS on CSC cluster was preset. In this study the transfer to running MDS on local cluster with 24 GPUs was made, which allows much greater throughput in MDS.

2. Processing of the MDS trajectories into the input of interest.

   MDS trajectories were used with VMD software to calculate the SASA ratio of drug vs whole NP at given time resolution. MDS trajectories were used to create PDB lists of atomic positions in chosen timepoints. PDB lists were the source to create 3D matrix representation of NP via PandasPdb as input for 3D convolution. Additionally PDBs were used to calculate the radial and inter-residue distances as an additional abstracted information input for the final model.

3. Selection of deep learning approaches and their fusion into one architecture.

   3D convolutional layers were used to extract features from 3D matrix NP representation and were concatenated to other linear vector input as SASA ratios, distances and drug types. The concatenated input was fed into LSTM layer.

4. Optimisation of the concatenated 3D convolutional-LSTM DL model was made. Several versions of the model were tried with assesment of validation loss function. The architecture and parameters with lowest loss function/fastest learning at set number of epochs was identified.

In the future, an attention layer could be added to create explainable AI to allow comprehension of the most important contributing input in DL and also other more sophisticated types of LSTM could be considered, such as social LSTM, to predict the behaviour of individual atoms or molecules.

References

[1] Jemal A, Siegel R, and Ward E. Cancer statistics, 2008. *CA Cancer J Clin*, 58 :(2):71–96, 2008.

[2] Doll R and Peto R. The causes of cancer: quantitative estimates of avoidable risks of cancer in the united states today. *J. Natl. Cancer Inst.*, 66(6):1191–308., 1981.

[3] Evolvable platform for programmable nanoparticle-based cancer therapies. https://cordis.europa.eu/project/id/800983, 2018. Accessed: 2021-01-21.

[4] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[5] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 35(8):1798–1828, 2013.

[6] Adam H. Marblestone, Greg Wayne, and Konrad P. Kording. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94, 2016.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 4:303–314, 12 1989.

[8] Wei Hu. Towards a real quantum neuron. *Natural Science*, 10:99–109, 01 2018.

[9] Qin T. *Deep Learning Basics. In: Dual Learning.* Springer, 2020.

[10] Agnes Sauer. Quick guide to gradient descent and it's variants. https://morioh.com/p/15c995420be6, 2020. Accessed: 2021-02-21.

[11] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[13] D. Cornelisse. An intuitive guide to convolutional neural networks. https://medium.com/free-code-camp/an-intuitive-guide-to-convolutionalneural-networks-260c2de0a050, 2018. Accessed: 2021-01-21.

[14] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. https://arxiv.org/abs/1603.07285, 2018.

[15] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow, 2nd Edition*. Packt Publishing, 2nd edition, 2017.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning, Adaptive computation and machine learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] J. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. diploma thesis, institut fur informatik, lehrstuhl prof. brauer, technische universitat munchen. www7.informatik.tu-muenchen.de/ hochre/, 1991.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[19] C. Olah. colah's blog: Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[20] Divyanshu Thakur. Lstm and its equations. https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af, 2018.

[21] Aditi Sinha. Understanding of lstm networks. https://www.geeksforgeeks.org/understanding-of-lstm-network, 2020.

[22] Romelia Salomon-Ferrer, David A. Case, and Ross C. Walker. An overview of the amber biomolecular simulation package. *WIREs Computational Molecular Science*, 3(2):198–210, 2013.

[23] William Humphrey, Andrew Dalke, and Klaus Schulten. Vmd: Visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, 1996.

[24] Huang CC Meng EC Couch GS Croll TI Morris JH Ferrin TE. Pettersen EF, Goddard TD. Ucsf chimerax: Structure visualization for researchers, educators, and developers. *Protein Sci.*, 30(1):70–82, 2021.

[25] Abigail Held and Maria Nagan. Running md with pmemd. https://ambermd.org/tutorials/basic/tutorial14/index.php, 2020.

[26] Goodacre R Xu Y. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *J Anal Test*, 3(2):249–262, 2018.

[27] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995.

7 Appendix

The used code

```
export syn='PegCY5_11.solv'
export inpcrd='PegCY5_11.solv'

export CUDA_VISIBLE_DEVICES=1
export NCPU=20

mkdir -p equil
mkdir -p prod
mkdir -p pre_equil

mpirun -np $NCPU -mca btl ^openib pmemd.MPI -O -p "$syn".prmtop -i ./mdin/min_sol.in \
  -o ./pre_equil/min_sol_"$syn".out \
  -x ./pre_equil/min_sol_"$syn".nc \
  -r ./pre_equil/min_sol_"$syn".rst \
  -c   "$inpcrd".inpcrd \
  -ref "$inpcrd".inpcrd

mpirun -np $NCPU -mca btl ^openib pmemd.MPI -O -p "$syn".prmtop -i ./mdin/min_all.in \
  -o ./pre_equil/min_all_"$syn".out \
  -x ./pre_equil/min_all_"$syn".nc \
  -r ./pre_equil/min_all_"$syn".rst \
  -c   ./pre_equil/min_sol_"$syn".rst \
  -ref ./pre_equil/min_sol_"$syn".rst

mpirun -np $NCPU -mca btl ^openib pmemd.MPI -O -p "$syn".prmtop -i ./mdin/heat.in \
  -o ./pre_equil/heat_"$syn".out \
  -x ./pre_equil/heat_"$syn".nc \
  -r ./pre_equil/heat_"$syn".rst \
  -c   ./pre_equil/min_all_"$syn".rst \
  -ref ./pre_equil/min_all_"$syn".rst

mpirun -np $NCPU pmemd.MPI -O -p "$syn".prmtop -i ./mdin/density.in \
  -o ./pre_equil/density_"$syn".out \
  -x ./pre_equil/density_"$syn".nc \
  -r ./pre_equil/density_"$syn".rst \
  -c   ./pre_equil/heat_"$syn".rst \
  -ref ./pre_equil/heat_"$syn".rst

mpirun -np $NCPU pmemd.MPI -O -p "$syn".prmtop -i ./mdin/density2.in \
 -o ./pre_equil/density2_"$syn".out \
  -x ./pre_equil/density2_"$syn".nc \
  -r ./pre_equil/density2_"$syn".rst \
  -c   ./pre_equil/density_"$syn".rst \
  -ref ./pre_equil/density_"$syn".rst


pmemd.cuda -O -p "$syn".prmtop -i ./mdin/equil.in \
  -o ./equil/equil1_"$syn".out \
```

```
  -x ./equil/equil1_"$syn".nc \
  -r ./equil/equil1_"$syn".rst \
  -c   ./pre_equil/density2_"$syn".rst \
  -ref ./pre_equil/density2_"$syn".rst
export x=2
while [[ $x -le 10 ]] ; do
let y=x-1
pmemd.cuda -O -p "$syn".prmtop -i ./mdin/equil.in \
  -o ./equil/equil"$x"_"$syn".out \
  -x ./equil/equil"$x"_"$syn".nc \
  -r ./equil/equil"$x"_"$syn".rst \
  -c   ./equil/equil"$y"_"$syn".rst \
  -ref ./equil/equil"$y"_"$syn".rst
let x+=1
done

pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_BER.in \
  -o ./equil/NPT_BER1_"$syn".out \
  -x ./equil/NPT_BER1_"$syn".nc \
  -r ./equil/NPT_BER1_"$syn".rst \
  -c   ./equil/equil10_"$syn".rst \
  -ref ./equil/equil10_"$syn".rst
export x=2
while [[ $x -le 8 ]] ; do
let y=x-1
pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_BER.in \
  -o ./equil/NPT_BER"$x"_"$syn".out \
  -x ./equil/NPT_BER"$x"_"$syn".nc \
  -r ./equil/NPT_BER"$x"_"$syn".rst \
  -c   ./equil/NPT_BER"$y"_"$syn".rst \
  -ref ./equil/NPT_BER"$y"_"$syn".rst
let x+=1
done

pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_MCeq.in \
  -o ./equil/NPT_MC1_"$syn".out \
  -x ./equil/NPT_MC1_"$syn".nc \
  -r ./equil/NPT_MC1_"$syn".rst \
  -c   ./equil/NPT_BER8_"$syn".rst \
  -ref ./equil/NPT_BER8_"$syn".rst
export x=2
while [[ $x -le 5 ]] ; do
let y=x-1
pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_MCeq.in \
  -o ./equil/NPT_MC"$x"_"$syn".out \
  -x ./equil/NPT_MC"$x"_"$syn".nc \
  -r ./equil/NPT_MC"$x"_"$syn".rst \
  -c   ./equil/NPT_MC"$y"_"$syn".rst \
  -ref ./equil/NPT_MC"$y"_"$syn".rst
let x+=1
done

pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_MC.in \
  -o ./prod/NPT_MC1_"$syn".out \
  -x ./prod/NPT_MC1_"$syn".nc \
  -r ./prod/NPT_MC1_"$syn".rst \
```

```
  -c   ./equil/NPT_MC5_"$syn".rst \
  -ref ./equil/NPT_MC5_"$syn".rst
export x=2
while [[ $x -le 60 ]] ; do
let y=x-1
pmemd.cuda -O -p "$syn".prmtop -i ./mdin/NPT_MC.in \
  -o ./prod/NPT_MC"$x"_"$syn".out \
  -x ./prod/NPT_MC"$x"_"$syn".nc \
  -r ./prod/NPT_MC"$x"_"$syn".rst \
  -c   ./prod/NPT_MC"$y"_"$syn".rst \
  -ref ./prod/NPT_MC"$y"_"$syn".rst
let x+=1
done
```

Listing 1: Example script to run MDS with Amber PMEMD on CSC cPouta Virtual Machine

```

touch "../vmd_SASA_calc.tcl" #creating a command text file with
   commands to be executed within VMD

#printing the lines (commands) into command text file
echo "mol new {prod/$FN.solv.prmtop} type {parm7} first 0 last -1
   step 1 waitfor 1" >> "../vmd_SASA_calc.tcl" #opening the
   trajectory file
echo "mol addfile {prod/NPT_MC1_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC2_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC3_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC4_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC5_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC6_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC7_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC8_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC9_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC10_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC11_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC12_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC13_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC14_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC15_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
echo "mol addfile {prod/NPT_MC16_$FN.solv.nc} type {netcdf} first 0
   last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
```

```
22 echo "mol addfile {prod/NPT_MC17_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
23 echo "mol addfile {prod/NPT_MC18_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
24 echo "mol addfile {prod/NPT_MC19_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
25 echo "mol addfile {prod/NPT_MC20_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
26 echo "mol addfile {prod/NPT_MC21_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
27 echo "mol addfile {prod/NPT_MC22_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
28 echo "mol addfile {prod/NPT_MC23_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
29 echo "mol addfile {prod/NPT_MC24_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
30 echo "mol addfile {prod/NPT_MC25_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
31 echo "mol addfile {prod/NPT_MC26_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
32 echo "mol addfile {prod/NPT_MC27_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
33 echo "mol addfile {prod/NPT_MC28_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
34 echo "mol addfile {prod/NPT_MC29_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
35 echo "mol addfile {prod/NPT_MC30_$FN.solv.nc} type {netcdf} first 0
       last -1 step 10 waitfor all " >> "../vmd_SASA_calc.tcl"
36 echo "set selmode all" >> "../vmd_SASA_calc.tcl"
37 echo "set all [atomselect top all]" >> "../vmd_SASA_calc.tcl"
38 echo 'set drg [atomselect top "resname '$DRUG_NAME'"]' >> "../
       vmd_SASA_calc.tcl"
39 echo "set n [molinfo top get numframes]" >> "../vmd_SASA_calc.tcl"
40 echo 'set output [open "SASA_$selmode.csv" w]' >> "../vmd_SASA_calc.
       tcl"
41 echo '      puts $output "timepoint,SASArestricted,SASAtotal,
       SASAperc"' >> "../vmd_SASA_calc.tcl"
42 # sasa calculation loop
43 echo "for {set i 0} {\$i < \$n} {incr i} {" >> "../vmd_SASA_calc.tcl
       "
44 echo "      molinfo top set frame \$i" >> "../vmd_SASA_calc.tcl"
45 echo "      set sasar [measure sasa 1.4 \$all -restrict \$drg]" >> "
       ../vmd_SASA_calc.tcl"
46 echo "      set sasat [measure sasa 1.4 \$drg]" >> "../vmd_SASA_calc
       .tcl"
47 echo "      set sasaperc [expr {(\$sasar / \$sasat)*100}]" >> "../
       vmd_SASA_calc.tcl"
48 echo "      set ione [expr {\$i+1}]" >> "../vmd_SASA_calc.tcl"
49 echo '      puts $output "$ione,$sasar,$sasat,$sasaperc"' >> "../
       vmd_SASA_calc.tcl"
50 echo '      ' >> "../vmd_SASA_calc.tcl"
51 echo "}" >> "../vmd_SASA_calc.tcl"
52 echo 'puts "Done."  ' >> "../vmd_SASA_calc.tcl"
53 echo 'puts "output file: SASA_GEM_test_\$selmode.csv"' >> "../
       vmd_SASA_calc.tcl"
54 echo "close \$output" >> "../vmd_SASA_calc.tcl"
```

```
echo "mol delete 0" >> "../vmd_SASA_calc.tcl"
echo "exit" >> "../vmd_SASA_calc.tcl"
cd ..
echo "2pwd"
pwd


vmd -dispdev text -e vmd_SASA_calc.tcl

end=`date +%s`;runtime=$((end-start));hours=$((runtime / 3600));
    minutes=$(( (runtime % 3600) / 60 )); seconds=$(( (runtime %
    3600) % 60 ));

echo "Runtime: $hours:$minutes:$seconds (hh:mm:ss) - VMD SASA
    exposure " >> "exec_report.txt"
```

Listing 2: Part of trajectory processing Bash script that calculates NP SASA drug exposures via VMD in non GUI mode. Bash script first writes a custom TCL script which is in turn read by VMD.

```
import pandas as pd
import numpy as np
import glob
import csv


path='/home/user/Downloads/MastersNNinput/NDL/SASA100/'
input_files_csv=sorted(glob.glob(path+"*.csv"))
wind=100

for k,filename_csv in enumerate(input_files_csv):
        print("FILENAME:",filename_csv)
        print("K:",k)

        fn_idx_strt = filename_csv.rfind('/')
        print ("Substring 'csv' found at index:", fn_idx_strt )
        fn_idx_end = filename_csv.find('.csv')
        print ("Substring 'csv' found at index:", fn_idx_end )
        df = pd.DataFrame(pd.read_csv(filename_csv))
        for i in range(len(df.columns)-1):
            df.iloc[:,i+1] = df.iloc[:,i+1].rolling(window=wind,
    min_periods=1).mean()
            df.head()
        filename_csv_avg=filename_csv[:fn_idx_end] + '_avg_'+str(
    wind) + filename_csv[fn_idx_end:]
        print (filename_csv_avg)
        df.to_csv(filename_csv_avg, sep=',', encoding='utf-8', index
    =False)
```

Listing 3: Python script to smoothen SASA drug exposure with moving average

```
#!/bin/bash
#prior to execution run:
#chmod +x pdb_auto_08.sh
#sudo chmod 777 pdb_auto_08.sh

#author: Vid Sustar
```

```
7
8  #this script:
9  #1. copies given folder
10 #2. creates pdbs
11 #3. renames, copies pdbs
12
13 #./pdb_auto_07.sh NCL12 NCL 2.2 10 2 1 2 200
14 #new way of exposure calculations with VMD
15
16 #do screen
17
18 INPUT_FOLDER=$1 #assigning the variable for drug folder name from
       command line input parameter #1
19 DRUG_NAME=$2 #initialising the variable for  drug name (pdb name,
       later also for exposure accessibility calculations) from command
       line input parameter #2
20
21 echo "Input folder set to: "$INPUT_FOLDER #printing
22 echo "Drug set to: "$DRUG_NAME #printing
23
24 start=‘date +%s‘ #to measure total execution time
25
26 cp -r "$INPUT_FOLDER" "$INPUT_FOLDER"_5 #copying the folder to new
       name (added _2)
27
28 export AMBERHOME=/home/cloud-user/amber
29 export AMBERHOME=/home/cloud-user/amber/amber18 #if one path not
       found, no problem, script will continue to execute
30 source /home/cloud-user/amber/amber.sh
31 source /home/cloud-user/amber/amber18/amber.sh
32 export CUDA_HOME=/usr/local/cuda
33
34 pattern="$INPUT_FOLDER/*.solv.prmtop" #all the files in folder found
        with this extension
35 files=($pattern)
36 FN="${files[0]}"#first one
37 echo "pattern set to: "$pattern #printing
38 echo "files set to: "$files #printing
39 echo "FN init set to: "$FN #printing
40 FN=$files
41 FN=${FN##*/} #taking just the part after the path/
42 FN=${FN%.*} #taking the part in front of last .
43 FN=${FN%.*} #as above
44 echo "FN finaly set to: "$FN #printing
45 cd "$INPUT_FOLDER"_5
46 pwd
47 #Create input file for cpptraj7
48 touch "../cpp_traj_input.txt" #creating a command text file with
       commands to be executed within cpptraj
49 echo "set sysname = "$FN >> "../cpp_traj_input.txt" #printing the
       lines (commands) into command text file
50 echo "parm $FN.solv.prmtop" >> "../cpp_traj_input.txt"
51 #1. batch of commands
52 echo "trajin equil/equil?_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
53 echo "trajin equil/equil??_$FN.solv.nc 1 last 1">> "../
```

```
    cpp_traj_input.txt"
54 echo "autoimage">> "../cpp_traj_input.txt"
55 echo "trajout equil/equil_2ns_100psf.$FN.solv.nc offset 5">> "../
    cpp_traj_input.txt"
56 echo "go">> "../cpp_traj_input.txt"
57 echo "strip :WAT outprefix noWAT">> "../cpp_traj_input.txt"
58 echo "trajout equil/equil_2ns_20psf.$FN.noWAT.nc">> "../
    cpp_traj_input.txt"
59 echo "go">> "../cpp_traj_input.txt"
60 echo "clear trajin">> "../cpp_traj_input.txt"
61 echo "exit" >> "../cpp_traj_input.txt"
62 cat ../cpp_traj_input.txt | cpptraj  #running cpptraj with above
    cpptraj txt command file
63 rm ../cpp_traj_input.txt #removing the command .txt file
64
65
66 touch "../cpp_traj_input.txt" #creating a command text file with
    commands to be executed within cpptraj
67 echo "set sysname = "$FN >> "../cpp_traj_input.txt" #printing the
    lines (commands) into command text file
68 echo "parm $FN.solv.prmtop" >> "../cpp_traj_input.txt"
69 #2. batch of commands
70 echo "trajin equil/NPT_BER?_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
71 echo "autoimage">> "../cpp_traj_input.txt"
72 echo "trajout equil/NPT_BER_8ns_500psf.$FN.solv.nc offset 5">> "../
    cpp_traj_input.txt"
73 echo "go">> "../cpp_traj_input.txt"
74 echo "strip :WAT outprefix noWAT">> "../cpp_traj_input.txt"
75 echo "trajout equil/NPT_BER_8ns_100psf.$FN.noWAT.nc">> "../
    cpp_traj_input.txt"
76 echo "go">> "../cpp_traj_input.txt"
77 echo "clear trajin">> "../cpp_traj_input.txt"
78 echo "exit" >> "../cpp_traj_input.txt"
79 cat ../cpp_traj_input.txt | cpptraj  #running cpptraj with above
    cpptraj txt command file
80 rm ../cpp_traj_input.txt #removing the command .txt file
81
82
83 touch "../cpp_traj_input.txt" #creating a command text file with
    commands to be executed within cpptraj
84 echo "set sysname = "$FN >> "../cpp_traj_input.txt" #printing the
    lines (commands) into command text file
85 echo "parm $FN.solv.prmtop" >> "../cpp_traj_input.txt"
86 #3. batch of commands
87 echo "trajin equil/NPT_MC?_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
88 echo "autoimage">> "../cpp_traj_input.txt"
89 echo "trajout equil/NPT_MC_5ns_500psf.$FN.solv.nc offset 5">> "../
    cpp_traj_input.txt"
90 echo "go">> "../cpp_traj_input.txt"
91 echo "strip :WAT outprefix noWAT">> "../cpp_traj_input.txt"
92 echo "trajout equil/NPT_MC_5ns_100psf.$FN.noWAT.nc">> "../
    cpp_traj_input.txt"
93 echo "go">> "../cpp_traj_input.txt"
94 echo "clear trajin">> "../cpp_traj_input.txt"
```

```
95 echo "exit" >> "../cpp_traj_input.txt"
96 cat ../cpp_traj_input.txt | cpptraj  #running cpptraj with above
       cpptraj txt command file
97 rm ../cpp_traj_input.txt #removing the command .txt file
98
99
100
101
102 touch "../cpp_traj_input.txt" #creating a command text file with
       commands to be executed within cpptraj
103 echo "set sysname = "$FN >> "../cpp_traj_input.txt" #printing the
       lines (commands) into command text file
104 echo "parm $FN.solv.prmtop" >> "../cpp_traj_input.txt"
105 #4. batch of commands
106 echo "set ProdRange = 000-300ns">> "../cpp_traj_input.txt" #setting
       the time-resolution, how many pdbs per ns
107 #instead of for loop, because it does not work:
108 echo "trajin prod/NPT_MC1_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
109 echo "trajin prod/NPT_MC2_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
110 echo "trajin prod/NPT_MC3_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
111 echo "trajin prod/NPT_MC4_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
112 echo "trajin prod/NPT_MC5_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
113 echo "trajin prod/NPT_MC6_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
114 echo "trajin prod/NPT_MC7_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
115 echo "trajin prod/NPT_MC8_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
116 echo "trajin prod/NPT_MC9_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
117 echo "trajin prod/NPT_MC10_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
118 echo "trajin prod/NPT_MC11_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
119 echo "trajin prod/NPT_MC12_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
120 echo "trajin prod/NPT_MC13_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
121 echo "trajin prod/NPT_MC14_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
122 echo "trajin prod/NPT_MC15_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
123 echo "trajin prod/NPT_MC16_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
124 echo "trajin prod/NPT_MC17_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
125 echo "trajin prod/NPT_MC18_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
126 echo "trajin prod/NPT_MC19_$FN.solv.nc  1 last 1">> "../
       cpp_traj_input.txt"
127 echo "trajin prod/NPT_MC20_$FN.solv.nc  1 last 1">> "../
```

```
    cpp_traj_input.txt"
128 echo "trajin prod/NPT_MC21_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
129 echo "trajin prod/NPT_MC22_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
130 echo "trajin prod/NPT_MC23_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
131 echo "trajin prod/NPT_MC24_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
132 echo "trajin prod/NPT_MC25_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
133 echo "trajin prod/NPT_MC26_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
134 echo "trajin prod/NPT_MC27_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
135 echo "trajin prod/NPT_MC28_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
136 echo "trajin prod/NPT_MC29_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
137 echo "trajin prod/NPT_MC30_$FN.solv.nc  1 last 1">> "../
    cpp_traj_input.txt"
138 echo "autoimage">> "../cpp_traj_input.txt"
139 echo "trajout prod/NPT_MC_000-300ns_1nsf.$FN.solv.nc offset 10">> "
    ../cpp_traj_input.txt"
140 echo "go">> "../cpp_traj_input.txt"
141 echo "strip :WAT outprefix noWAT">> "../cpp_traj_input.txt"
142 echo "autoimage">> "../cpp_traj_input.txt"
143 echo "trajout prod/NPT_MC_$ProdRange_100psf.$FN.noWAT.nc">> "../
    cpp_traj_input.txt"
144 echo "go">> "../cpp_traj_input.txt"
145 echo "clear trajin">> "../cpp_traj_input.txt"
146
147 echo "exit" >> "../cpp_traj_input.txt"
148
149 cat ../cpp_traj_input.txt | cpptraj  #running cpptraj with above
    cpptraj txt command file
150 rm ../cpp_traj_input.txt #removing the command .txt file
151 ######################################################################
152 echo "copying "$INPUT_FOLDER"_5/"$FN.solv.prmtop" "$INPUT_FOLDER"_5/
    prod/"
153 pwd
154 echo "1"
155 cp -r "$INPUT_FOLDER"_5/"$FN.solv.prmtop" "$INPUT_FOLDER"_5/prod/ #
    copying the *.solv.prmtop to prod folder
156 echo "5"
157 cp -r $(pwd)/$FN.solv.prmtop $(pwd)/prod/$FN.solv.prmtop #ing the *.
    solv.prmtop to prod folder
158
159
160 cd $(pwd)/prod # executing cpptraj inside of prod folder
161 #cpptraj commands withing prod folder to produce pdbs there
162 echo "1pwd"
163 pwd
164 touch "../cpp_traj_input.txt" #creating a command text file with
    commands to be executed within cpptraj
165 echo "set sysname = "$FN >> "../cpp_traj_input.txt" #printing the
```

```
   lines (commands) into command text file
echo "parm $FN.solv.prmtop" >> "../cpp_traj_input.txt"
echo "trajin NPT_MC_000-300ns_1nsf.$FN.solv.nc 1 last 1">> "../
   cpp_traj_input.txt"
echo "trajout $DRUG_NAME.pdb multi #exchange OQL with drug name ">>
   "../cpp_traj_input.txt"
echo "go">> "../cpp_traj_input.txt"
echo "exit" >> "../cpp_traj_input.txt"
echo "2pwd"
pwd
cat ../cpp_traj_input.txt | cpptraj  #running cpptraj with above
   cpptraj txt command file
rm ../cpp_traj_input.txt #removing the command .txt file
echo "3pwd"
pwd
```

Listing 4: Example Bash script to run Amber Cpp-traj to convert trajectory files into PDB files on CSC cPouta Virtual Machine

```


print("INPUT FORMAT: 'python3 SCRIPT(pdb2mtrx09.py) DRUGNAME(PAN)
   RATIODRUG(2) RATIOSUPSUBSTANCE(1) MTRXSIZE(50) ALSO-SUPD(1) ALSO-
   OTHRESID(1)'  ")

import sys
import h5py
import os, psutil, numpy as np
from biopandas.pdb import PandasPdb
import glob
import pandas as pd

from datetime import datetime

drug_list=['PAN', 'OQL', 'ZIL', 'NHQ', 'GEM']#ADD TO THE LIST! make
   sure same order and length for drug proximals
drugproximals=['OCP','OCQ','OCZ','OCN', 'OCG']#CHECK and add drug
   proximal for NHQ!

nonzero_elements_list=[]

compression_level=9 #h5 file gzip level of compression, 9 is highest
   ->smallest files, possibly slower opening
path=os.getcwd()+'/'

#INPUT PARAMETERS
compression_level=9 #h5 file gzip level of compression, 9 is highest
   ->smallest files, possibly slower opening
drugname=sys.argv[1]#finddrug(filename) #change appropriately
drug=drugname
rdrug=int(sys.argv[2])#int(ratio[0]) #check SYS replace/uncomment
   appropriately
rsupd=int(sys.argv[3])#int(ratio[1])
size=int(sys.argv[4])
supd = int(sys.argv[5]) #if one wants only drug residues make 0, if
   also supdrug 1
```

```
resid = int(sys.argv[6]) #if one wants also other non drug/sup subs residues 1 else 0
ending="s"+str(supd)+"r"+str(resid)
window_list=[1,50,100] #list of averaging window lenghts
angle_step=int(sys.argv[7]) #there will be 360/anglestep versions of the matrix calculated each cumulatively rotated by anglestep in y and z direction!


def usage(): #function to show the usage of memory
    process = psutil.Process(os.getpid())
    print ("procces memory:",process.memory_info()[0] / float(2 ** 20))
    print("memory percent:",process.memory_percent())

print("intial",usage())


def finddrugprox(drugname):


    drugproximal=""

    for i in range(len(drug_list)):

        if drug_list[i] == drugname:
            drugproximal=drugproximals[i]
            #print ("drug_list[i],drugname: ",drug_list[i],drugname)

    return drugproximal

from copy import deepcopy
from scipy.spatial import distance
def transform_rotate(atoms, azim_angle, polar_angle): #function from Otto Lindfors

    # copy in order to not manipulate the original object
    df = deepcopy(atoms)

    # Rotational matrices ("clockwize" rotation)
    R_z = np.array([
        [np.cos(polar_angle), np.sin(polar_angle), 0],
        [-np.sin(polar_angle), np.cos(polar_angle), 0],
        [0, 0, 1]
    ])
    R_y = np.array([
        [np.cos(azim_angle), 0, -np.sin(azim_angle)],
        [0, 1, 0],
        [np.sin(azim_angle), 0, np.cos(azim_angle)]
    ])
    R = np.dot(R_y, R_z)

    # coordinates
    A = df[['x_coord', 'y_coord', 'z_coord']].values.T

    # Rotate
```

```
    A = np.dot(R, A)

    # Save
    df.loc[:, ['x_coord', 'y_coord', 'z_coord']] = A.T

    # Update or add the spherical coordinates
    df = add_spherical_coordinates(df)

    return df


def shift_atoms_to_origin(atoms, center_atom_number=1):
    atoms = deepcopy(atoms)
    # the nanoparticle center
    center_atom = atoms.loc[atoms['atom_number'] ==
   center_atom_number]
    x_center = center_atom['x_coord'][0]
    y_center = center_atom['y_coord'][0]
    z_center = center_atom['z_coord'][0]
    # Center the nanoparticle to origo
    atoms.loc[:, 'x_coord'] -= x_center
    atoms.loc[:, 'y_coord'] -= y_center
    atoms.loc[:, 'z_coord'] -= z_center

    return atoms
def add_spherical_coordinates(atoms):#function from Otto Lindfors
    # add radius column to atoms df
    r = PandasPdb.distance_df(atoms)
    atoms['radius']= r

    x = atoms['x_coord']
    y = atoms['y_coord']
    z = atoms['z_coord']

    # polar angle
    atoms['polar_angle'] = np.arctan2(y, x)           # angles in
   range [-pi, pi]
    atoms.loc[atoms['polar_angle'] < 0, 'polar_angle'] += 2 * np.pi
    # add np.pi to negative angles to make the angles in range [0, 2
   pi[

    # azimuth angle
    atoms['azim_angle'] = np.arccos( z / r )
    atoms.loc[atoms['azim_angle'].isna(), 'azim_angle'] = 0

    return atoms

def shift_atoms_from_origin(atoms, gdata, center_atom_number=1):
    """Centers the atoms so that center_atom_number is at origin"""

    atoms = deepcopy(atoms)

    # the nanoparticle center
    center_atom = gdata.loc[atoms['atom_number'] ==
   center_atom_number]
    x_center = center_atom['x_coord'][0]
```

```
132     y_center = center_atom['y_coord'][0]
133     z_center = center_atom['z_coord'][0]
134     # Center the nanoparticle to origo
135     atoms.loc[:, 'x_coord'] += x_center
136     atoms.loc[:, 'y_coord'] += y_center
137     atoms.loc[:, 'z_coord'] += z_center
138 
139     return atoms
140 
141 def resid_distance(pdata,residuename):
142     #function to calculate radial distances between centre of
    nanoparticle, origin -> and their mean, median, max, min)
143     pdata_residue = pdata['residue_name'] == residuename
144     pdata_residue_rdistance_mean = pdata.loc[pdata_residue, 'radius'
    ].mean()
145     pdata_residue_rdistance_median = pdata.loc[pdata_residue, '
    radius'].median()
146     pdata_residue_rdistance_max = pdata.loc[pdata_residue, 'radius'
    ].max()
147     pdata_residue_rdistance_min = pdata.loc[pdata_residue, 'radius'
    ].min()
148 
149     #function to calculate distances between all the residues of
    certain type -> and their mean, median)
150     pdata_residue_2 = pdata[pdata['residue_name'] == residuename]
151     pdata_residue_2 = pdata_residue_2[['x_coord','y_coord','z_coord'
    ]]
152     distances=distance.cdist(pdata_residue_2, pdata_residue_2, '
    euclidean')
153     distances=skip_diag_strided(distances) #def skip_diag_strided(A)
    : #function to remove diagonal 0s from euclidean distance matrix
    (self distances) to keep only calculated distances between
    residues
154 
155     pdata_residue_intr_distance_median=np.median(distances)
156     pdata_residue_intr_distance_mean=np.mean(distances)
157 
158     distance_list_residue=[pdata_residue_rdistance_mean,
    pdata_residue_rdistance_median,pdata_residue_rdistance_max,
    pdata_residue_rdistance_min,pdata_residue_intr_distance_median,
    pdata_residue_intr_distance_mean]
159     return distance_list_residue
160 
161 def skip_diag_strided(A): #function to remove diagonal 0s from
    euclidean distance matrix (self distances) to keep only
    calculated distances between residues
162     m = A.shape[0]
163     strided = np.lib.stride_tricks.as_strided
164     s0,s1 = A.strides
165     return strided(A.ravel()[1:], shape=(m-1,m), strides=(s0+s1,s1))
    .reshape(m,-1)
166 
167 def pandasdf_np_appends(pdata, k):
168     Row_list =[]
169     # Iterate over each row
170     for index, rows in pdata.iterrows():
```

```
171         # Create list for the current row
172         my_list =[rows.atom_number, rows.atom_name, rows.
    residue_name,rows.residue_number, rows.x_coord, rows.y_coord,
    rows.z_coord, rows.element_symbol, rows.intensity]
173         # append the list to the final list
174         Row_list.append(my_list)
175     pdata_np = np.array(Row_list)
176     pdata_np = np.expand_dims(pdata_np, axis=1) #adding an extra
    dimension so later next time points can be added
177     return pdata_np
178
179
180 def pdb2matrix(k, filename,angle,sec_port):
181     if 1:
182         fulpath=filename
183         input_file_name_idx = fulpath.rfind("/")
184         input_file_name=fulpath[input_file_name_idx+1:len(fulpath)
    -4]
185         input_file_name_sqnum_idx = input_file_name.rfind("_")
186         input_file_name_sqnum=input_file_name[
    input_file_name_sqnum_idx+1:len(input_file_name)]
187         ppdb=PandasPdb().read_pdb(fulpath) #reading from path
188         # Using DataFrame.insert() to add a column
189         ppdb.df['ATOM']["intensity"] = np.nan
190         #CALCULATE AND ADD INTENSITIES based on atom and residue:
191         ppdb.df['ATOM'].head(3)
192         data=ppdb.df['ATOM']
193         #usage() # initial memory usage
194         #drop unnecessary columns
195         pdata=data.drop(['record_name','blank_1','alt_loc','blank_2'
    ,'chain_id','insertion','segment_id','occupancy','b_factor','
    charge','line_idx','blank_3','blank_4'], axis=1)
196         #check dropped columns
197         pdata.head(3)
198         num_pdb_rows=len(pdata)
199         num_pdb_columns=len(pdata.columns)
200         #calculating positions of residues in PDB, to get the ranges
     for colorations
201
202         ngoldresidues=2869
203         common_drugresidues=["C11","OCB","OCC","OCC"]
204         drug_addit_res=["OCC","OCC"]
205         sup_drugresidues=["OCN","ONC"]
206         #filename_front = input_file.split("_")[0]
207         #print(filename_front)
208         n_total=420
209
210         trat=rdrug+rsupd
211         ndrug=int((n_total/trat)*rdrug)
212         nsupd=int((n_total/trat)*rsupd)
213         drugproximal=finddrugprox(drugname)
214         #print("drugproximal",drugproximal)
215         drugresidues=[drugproximal,drugname]
216         drug_assc_residues=common_drugresidues+drug_addit_res+
    drugresidues
217         suppdrug_assc_residues=common_drugresidues+sup_drugresidues
```

```
218         starting_drug_resnum=ngoldresidues+1
219         ending_drug_resnum=starting_drug_resnum+(ndrug*len(
    drug_assc_residues))-1
220         starting_supdrug_resnum=ending_drug_resnum+1
221         ending_supdrug_resnum=starting_supdrug_resnum+(nsupd*len(
    suppdrug_assc_residues))-1
222         #COLORING!
223         #could be by residue number range and then residue name and
    element symbol
224 
225         #gold brightest
226         pdata['intensity'] = np.where(pdata['element_symbol'] == 'AU
    ',255, pdata['intensity'])
227         #brighter grey tones
228         list_drug_residue_colors= [205, 206,  207,  208,  209,  210,
      211,  212,  213,  214,  215,  216,  217,  218,  219,  220,
    221,  222,  223,  224,  225,  226,  227,  228,  229,  230,  231,
      232,  233,  234,  235,  236,  237,  238,  239,  240,  241,
    242,  243,  244,  245,  246,  247,  248,  249,  250,  251,  252,
      253,  254,  255]
229         #darker grey tones
230         list_supdrug_residue_colors=[50,  51, 52, 53, 54, 55, 56,
    57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
    73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
    89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
231         maxlist_drug_residue_colors=max(list_drug_residue_colors)
232 
233         drug_assc_residues = list(dict.fromkeys(drug_assc_residues))
234         pdata['intensity'] = 0#0 #1 for testing
235         pdata['intensity'] = np.where((pdata['residue_number'] <=
    ending_drug_resnum) & (pdata['residue_name'] == drug)&(pdata['
    atom_name'] == 'H31'),255, pdata['intensity'])
236         if resid:
237             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == drug)&(pdata['
    atom_name'] == 'C5'),255, pdata['intensity'])
238             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == drugproximal)&
     (pdata['atom_name'] == 'C1'),250, pdata['intensity'])
239             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == 'OCC')& (pdata
    ['atom_name'] == 'C1'),245, pdata['intensity'])
240             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == 'OCB')& (pdata
    ['atom_name'] == 'C1'),240, pdata['intensity'])
241             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == 'C11')& (pdata
    ['atom_name'] == 'H72'),235, pdata['intensity'])
242             pdata['intensity'] = np.where((pdata['residue_number']
    <= ending_drug_resnum) & (pdata['residue_name'] == 'NP')& (pdata[
    'atom_number'] == 1),5, pdata['intensity'])
243 
244         if supd:
245             pdata['intensity'] = np.where((pdata['residue_number']
    >= starting_supdrug_resnum) & (pdata['residue_name'] == 'ONC')&(
    pdata['atom_name'] == 'N'),120, pdata['intensity'])
```

```
246             if resid:
247                 pdata['intensity'] = np.where((pdata['residue_number
    '] >= starting_supdrug_resnum) & (pdata['residue_name'] == 'OCN')
    & (pdata['atom_name'] == 'C1'),115, pdata['intensity'])
248                 pdata['intensity'] = np.where((pdata['residue_number
    '] >= starting_supdrug_resnum) & (pdata['residue_name'] == 'OCC')
    & (pdata['atom_name'] == 'C1'),110, pdata['intensity'])
249                 pdata['intensity'] = np.where((pdata['residue_number
    '] >= starting_supdrug_resnum) & (pdata['residue_name'] == 'OCB')
    & (pdata['atom_name'] == 'C1'),105, pdata['intensity'])
250                 pdata['intensity'] = np.where((pdata['residue_number
    '] >= starting_supdrug_resnum) & (pdata['residue_name'] == 'C11')
    & (pdata['atom_name'] == 'H72'),100, pdata['intensity'])
251         gold_x_mean=pdata[pdata['element_symbol'] == 'AU']["x_coord"
    ].mean()
252         gold_y_mean=pdata[pdata['element_symbol'] == 'AU']["y_coord"
    ].mean()
253         gold_z_mean=pdata[pdata['element_symbol'] == 'AU']["z_coord"
    ].mean()
254         #calculating the max distance from gold centre for all
    dimensions
255         max_val_x=pdata["x_coord"].max()
256         min_val_x=pdata["x_coord"].min()
257         maxdist_goldc_x=max(abs(gold_x_mean-max_val_x),abs(
    gold_x_mean-min_val_x))
258         max_val_y=pdata["y_coord"].max()
259         min_val_y=pdata["y_coord"].min()
260         maxdist_goldc_y=max(abs(gold_y_mean-max_val_y),abs(
    gold_y_mean-min_val_y))
261         max_val_z=pdata["z_coord"].max()
262         min_val_z=pdata["z_coord"].min()
263         maxdist_goldc_z=max(abs(gold_z_mean-max_val_z),abs(
    gold_z_mean-min_val_z))
264         #finding absolutely max distance among all dimensions
265         maxdist_goldc=max(maxdist_goldc_x,maxdist_goldc_y,
    maxdist_goldc_z)
266         gdata=pdata[pdata['atom_number'] == 1]
267         #remove zero elements of pdata
268         pdata = pdata.drop(pdata[(pdata.intensity == 0)&(pdata['
    atom_number'] != 1)].index)
269         pdata=shift_atoms_to_origin(pdata,1)
270         #rotate everything by certain angle!
271         phi=angle #CHANGE TO WHATEVER YOU NEED!
272         phi = phi / 180 * np.pi
273         polar_angle=phi
274         theta=angle
275         theta = theta / 180 * np.pi
276         azim_angle=theta
277         pdata=transform_rotate(pdata, azim_angle, polar_angle)
278         pdata=shift_atoms_from_origin(pdata, gdata,1)
279         pdata=pdata.drop(['radius','polar_angle','azim_angle'], axis
    =1)
280         pdata=pdata.round(3)#rounding all the numbers to 3 decimals,
     so it is a bit easier to handle
281
282         return pdata,k,maxdist_goldc,gold_x_mean,gold_y_mean,
```

```
    gold_z_mean,num_pdb_rows,drugname,drugproximal
283
284
285 def matrix_mapping(pdata,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean,num_pdb_rows):
286     factor=size/((maxdist_goldc+3)*2)#+4) #+3 because of variation!,
        actually the np distance shrinks,
287     #so if this is constant by the first maxgold distance and not
    dynamic, then the shrinking will be shown on the matrix
288     xcordlist = pdata["x_coord"].tolist()
289     ycordlist = pdata["y_coord"].tolist()
290     zcordlist = pdata["z_coord"].tolist()
291     xcordlistnp = pdata["x_coord"].to_numpy().astype(float)
292     xcordlistnp =np.asarray(xcordlistnp, dtype=float)
293     ycordlistnp = pdata["y_coord"].to_numpy().astype(float)
294     zcordlistnp = pdata["z_coord"].to_numpy().astype(float)
295     xcordlist=[int(((float(element)-gold_x_mean)* factor)+(size-1)
    /2.0) for element in xcordlist]
296     ycordlist=[int(((float(element)-gold_y_mean) * factor)+(size-1)
    /2.0) for element in ycordlist]
297     zcordlist=[int(((float(element)-gold_z_mean) * factor)+(size-1)
    /2.0) for element in zcordlist]
298     npmatrix = np.zeros([size,size,size])#creating matrix with 0s of
        size given
299     npmatrix[xcordlist,ycordlist,zcordlist] = pd.to_numeric(pdata["
    intensity"].tolist()) #filling it up with intensities to rounded
    coordinates
300     npmatrix = np.expand_dims(npmatrix, axis=0) #adding one extra
    dimension "in front", so I can concatenate them with others!
301     num_nonzero_elements=np.count_nonzero(npmatrix)
302     nonzero_elements_list.append(num_nonzero_elements)
303     return npmatrix
304
305 def pandas_averaging(all_res_tmpnts_temp,wind):
306     #function to do the averaging along timepoints
307     data_xyz=all_res_tmpnts_temp[:,4:7].copy()
308     data_xyz = data_xyz.astype(float)
309     data_xyz_df = pd.DataFrame(data_xyz, columns = ['x_coord','
    y_coord','z_coord'])
310     df3=data_xyz_df.copy()
311     for i in range(len(df3.columns)):
312         df3.iloc[:,i] = df3.iloc[:,i].rolling(window=wind,
    min_periods=1).mean()
313     return df3
314
315
316 def matrix_averaged2dist(k,averaged_data_df,sec_port,drugname,
    drugproximal):
317     if 1:
318         gold_x_mean=averaged_data_df[averaged_data_df['
    element_symbol'] == 'AU']["x_coord"].mean()
319         gold_y_mean=averaged_data_df[averaged_data_df['
    element_symbol'] == 'AU']["y_coord"].mean()
320         gold_z_mean=averaged_data_df[averaged_data_df['
    element_symbol'] == 'AU']["z_coord"].mean()
321
```

```
322         #calculating the max distance from gold centre for all
    dimensions
323         max_val_x=averaged_data_df["x_coord"].max()
324         min_val_x=averaged_data_df["x_coord"].min()
325 
326         maxdist_goldc_x=max(abs(gold_x_mean-max_val_x),abs(
    gold_x_mean-min_val_x))
327         max_val_y=averaged_data_df["y_coord"].max()
328         min_val_y=averaged_data_df["y_coord"].min()
329         maxdist_goldc_y=max(abs(gold_y_mean-max_val_y),abs(
    gold_y_mean-min_val_y))
330 
331         max_val_z=averaged_data_df["z_coord"].max()
332         min_val_z=averaged_data_df["z_coord"].min()
333         maxdist_goldc_z=max(abs(gold_z_mean-max_val_z),abs(
    gold_z_mean-min_val_z))
334 
335         maxdist_goldc=max(maxdist_goldc_x,maxdist_goldc_y,
    maxdist_goldc_z)
336 
337         gdata=averaged_data_df[averaged_data_df['atom_number'] == 1]
338         averaged_data_df = averaged_data_df.drop(averaged_data_df[(
    averaged_data_df.intensity == 0)&(averaged_data_df['atom_number']
     != 1)].index)
339         averaged_data_df=shift_atoms_to_origin(averaged_data_df,1)
340         #rotate everything by certain angle!
341         phi=0 #CHANGE TO WHATEVER YOU NEED!
342         phi = phi / 180 * np.pi
343         polar_angle=phi
344         theta=0
345         theta = theta / 180 * np.pi
346         azim_angle=theta
347         averaged_data_df=transform_rotate(averaged_data_df,
    azim_angle, polar_angle)
348 
349         distance_list_drug=resid_distance(averaged_data_df,drugname)
350         distance_list_drugprox=resid_distance(averaged_data_df,
    drugproximal)
351         distance_list_sups=resid_distance(averaged_data_df,"ONC")
352         distance_list_supsprox=resid_distance(averaged_data_df,"OCN"
    )
353 
354         distance_list_OCC=resid_distance(averaged_data_df,"OCC")
355         distance_list_OCB=resid_distance(averaged_data_df,"OCB")
356         distance_list_C11=resid_distance(averaged_data_df,"C11")
357         distance_list_joined = distance_list_drug+
    distance_list_drugprox+distance_list_sups+distance_list_supsprox+
    distance_list_OCC+distance_list_OCB+distance_list_C11
358         averaged_data_df=shift_atoms_from_origin(averaged_data_df,
    gdata,1)
359         averaged_data_df=averaged_data_df.drop(['radius','
    polar_angle','azim_angle'], axis=1)
360         averaged_data_df=averaged_data_df.round(3)#rounding all the
    numbers to 3 decimals, so it is a bit easier to handle
361          return averaged_data_df,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean, distance_list_joined
```

```
362
363
364 start_time = datetime.now()
365 def main(path,size,wind,angle):
366
367     sec_port=0
368     input_files=sorted(glob.glob(path+"*.pdb"))
369     counter=0
370     for k,filename in enumerate(input_files): #enumerate is for
    greater control, to put some extra constraints on steps of for
    loop
371         if (k == 0):
372             pdata,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean,num_pdb_rows,drugname,drugproximal=pdb2matrix(k,
    filename,angle,sec_port)
373             pdata_np=pandasdf_np_appends(pdata, k)
374
375             all_res_tmpnts=np.zeros([pdata_np.shape[0], 300, 9],
    dtype=object)
376             all_res_tmpnts[:,k:k+1,:]=pdata_np
377             pdata_index=pdata.index #to be used when reconstituting
    individual timepoint dataframe from numpy matrix
378             pdata_columns=pdata.columns #later in the main function
379
380         if (k!=0):
381             #print("new_run")
382             pdata,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean,num_pdb_rows,drugname,drugproximal=pdb2matrix(k,
    filename,angle,sec_port)
383             #actually gold parameters might need to be fixed on the
    initial values
384             pdata_np=pandasdf_np_appends(pdata, k)
385             all_res_tmpnts[:,k:k+1,:]=pdata_np
386
387             counter=0
388     z = 2 #number of steps to have printed reporting for debugging
389     #do the averaging
390     for i in range(all_res_tmpnts.shape[0]): #looping through all
    the residues, averaging by wind(ow) size
391         paverage=pandas_averaging(all_res_tmpnts[i],wind)
392         paverage=paverage.to_numpy()
393         all_res_tmpnts[i][0:all_res_tmpnts.shape[1],4:7]=paverage
394
395     sec_port=1
396     print("2ND LOOP, AFTER AVERAGING")
397
398     temp_data_columns=['atom_number', 'atom_name', 'residue_name', '
    residue_number', 'x_coord','y_coord', 'z_coord', 'element_symbol'
    , 'intensity']
399
400     for k, filename in enumerate(input_files): #enumerate is for
    greater control, to put some extra constraints on steps of for
    loop
401         if (k == 0):
402
403             temppdata_final_np=all_res_tmpnts[:,k:k+1,:]
```

```
404             temppdata_final = temppdata_final_np[:, 0, :] #removing
    the dimension that was used to enumerate the timepoint
405             temp_data_columns=['atom_number', 'atom_name', '
    residue_name', 'residue_number', 'x_coord','y_coord', 'z_coord',
    'element_symbol', 'intensity']
406             temppdata_df_inp = pd.DataFrame(data=temppdata_final,
    index=pdata_index, columns=temp_data_columns)
407             temppdata_df_inp = temppdata_df_inp.astype(dtype= {"
    atom_number":"int64","atom_name":"object","residue_name":"object"
    ,"residue_number":"int64","x_coord":"float64","y_coord":"float64"
    ,"z_coord":"float64","element_symbol":"object","intensity":"int64
    "})
408             temppdata_df_inp.round(3)
409             tempdata_df_out,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean, distance_df_inp=matrix_averaged2dist(k,
    temppdata_df_inp,sec_port,drugname,drugproximal)
410
411             array=matrix_mapping(tempdata_df_out,k,maxdist_goldc,
    gold_x_mean,gold_y_mean,gold_z_mean,temppdata_final_np.shape[0])
412
413             distance_df = pd.DataFrame(columns=['DrgRDMn','DrgRDMdn'
    ,'DrgRDMx','DrgRDMin','DrgIntDMn','DrgIntDMdn','DrgPrxRDMn','
    DrgPrxRDMdn','DrgPrxRDMx','DrgPrxRDMin','DrgPrxIntDMn','
    DrgPrxIntDMdn','SupRDMn','SupRDMdn','SupRDMx','SupRDMin','
    SupIntDMn','SupIntDMdn','SupPrxRDMn','SupPrxRDMdn','SupPrxRDMx','
    SupPrxRDMin','SupPrxIntDMn','SupPrxIntDMdn','OCCRDMn','OCCRDMdn',
    'OCCRDMx','OCCRDMin','OCCIntDMn','OCCIntDMdn','OCBRDMn','OCBRDMdn
    ','OCBRDMx','OCBRDMin','OCBIntDMn','OCBIntDMdn','C11RDMn','
    C11RDMdn','C11RDMx','C11RDMin','C11IntDMn','C11IntDMdn'])
414             distance_df_inp_series = pd. Series(distance_df_inp,
    index = distance_df.columns)
415             distance_df = distance_df. append(distance_df_inp_series
    , ignore_index=True)
416
417         if ((k!=0)):#&(k<3)):
418             temppdata_final_np=all_res_tmpnts[:,k:k+1,:]
419             temppdata_final = temppdata_final_np[:, 0, :] #removing
    the dimension that was used to enumerate the timepoint
420             temppdata_df_inp = pd.DataFrame(data=temppdata_final,
    index=pdata_index,columns=temp_data_columns)
421             temppdata_df_inp = temppdata_df_inp.astype(dtype= {"
    atom_number":"int64","atom_name":"object","residue_name":"object"
    ,"residue_number":"int64","x_coord":"float64","y_coord":"float64"
    ,"z_coord":"float64","element_symbol":"object","intensity":"int64
    "})
422             temppdata_df_inp.round(3)
423
424
425             tempdata_df_out,k,maxdist_goldc,gold_x_mean,gold_y_mean,
    gold_z_mean, distance_df_inp=matrix_averaged2dist(k,
    temppdata_df_inp,sec_port,drugname,drugproximal)
426
427             distance_df_inp_series = pd. Series(distance_df_inp,
    index = distance_df.columns)
428             distance_df = distance_df. append(distance_df_inp_series
    , ignore_index=True)
```

```
            array_temp=matrix_mapping(tempdata_df_out,k,
    maxdist_goldc,gold_x_mean,gold_y_mean,gold_z_mean,
    temppdata_final_np.shape[0])

            array = np.concatenate((array, array_temp), axis = 0)

    return distance_df, array

for wind in window_list:
    print("wind:",wind)
    for angle in range (0,360,angle_step):
        print("angle:",angle)
        inds = [i for i,c in enumerate(path) if c=='/']
        try:
            main_fold_idx_1=(inds[-3])
            main_fold_idx_2=(inds[-2])
        except:
            print("ERROR! main folder finding not working")
        mainfolder=path[main_fold_idx_1+1:main_fold_idx_2]
        print("main folder name:",mainfolder)
        h5name='pdb2mtrx_2_'+mainfolder+str(size)+'_'+ending+'_'+str
    (angle)+'_'+str(wind)+'.h5'
        dist_csv_name='distancescsv_2_'+mainfolder+str(size)+'_'+
    ending+'_'+str(angle)+'_'+str(wind)+'2.csv'
        print('h5name: ',h5name)

        try:
            h5f = h5py.File(h5name, 'w')
        except:
            print("ERROR! COULD NOT CREATE H5 FILE ")
        distance_df_out,array1=main(path,size,wind, angle)
        print("############FINAL ARRAY SHAPE:",array1.shape)
        print("len(distance_df_out.index)",len(distance_df_out.index
    ))

        dset = h5f.create_dataset('pdb2_3dmatrix',data=array1,
    compression="gzip", compression_opts=compression_level)

        distance_df_out.to_csv(dist_csv_name, encoding='utf-8', sep=
    ',', index=False)

        h5f.close()
        usage()

time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
usage()
```

Listing 5: Python script to rotate and convert PDB atom positions into Numpy 3D matrices as separate quadrants and calculate the radial and inter-residue distances in NP.

```

import numpy as np
import tensorflow.keras
import h5py
```

```
import math
import time
from pympler import asizeof #to check the memory size of objects
from numpy import save
import csv
import pandas as pd


import sys
import os #psutil

# PLOTING WITH MATPLOT
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([1,2,3])
fig.savefig('test.png')

exp_n_col_y=17 #### number of columns from final exposure y

size=int(sys.argv[1]) #the size of the input matrix
typeinp=str(sys.argv[2]) #the type of the matrix, whether there are drugcentres, supsubscentres, other molecule centres etc
interval=int(sys.argv[3]) #the number of timepoints (pdbs, rows in exposure csv) to be skipped (to make timedistributed input for LSTM smaller and hence allow for bigger 3d matrix)
quad=int(sys.argv[4]) #quadrant (1) or no (0), taking just one eighth front top left quadrant of 3D matrix, to lower the size of input
conc_quad=int(sys.argv[5]) #if one wants to take the other 7 quadrants into consideration, it will concatenate them!

sect=int(sys.argv[6]) # sectioning of the 3d matrix (can be quadrant even further) (1 - yes) (0 - no), taking the right most portion of the matrix, see below
width=int(sys.argv[7]) # width of the section from above, how many cells, how much of the right most portion of the matrix
cpu=int(sys.argv[8]) # CPU mode - disable usage of GPU! (1 - CPU, 0 - GPU)
exp_n_col_y=int(sys.argv[9]) #### number of columns from final exposure y
num_epochs=int(sys.argv[10]) #number of epochs
sim_len_f=int(sys.argv[11]) #input length (the output should be of same length)
inp_len=int(sys.argv[12]) # numb of inputs per LSTM
wind=int(sys.argv[13]) # averaging window of SASA csv
train_end=int(sys.argv[14]) #trainfolder_ending_digit
if quad: #adjusting reported size of input matrix according to taking only quadrants
    nsize=int(size/2)
else:
    nsize=size

if sect: #adjusting the first dimension (width) of 3d matrix
```

```
    appropriatelty according to sectioning
48      nsize1=width
49  else:
50      nsize1=nsize
51
52  print(">>>>>>>>INPUT PARAMETERS:")
53  print("size: ",size)
54  print("interval between timepoints: ",interval)
55  print("use of only single quadrant of matrix bool: ",quad)
56  print("use only section of matrix bool: ",sect)
57  print("width of section:",width)
58  print("usage of cpu only bool:",cpu)
59
60
61  pathtrain=os.getcwd()+'/train'+str(train_end)+'/'#sys.argv[3] #path
        to the input #make sure the train data in train folder!
62  pathtest=os.getcwd()+'/test/'#sys.argv[3] #path to the input
63  import glob
64
65  from datetime import datetime
66  start_time = datetime.now()
67
68  if cpu: #disabling gpu to force run it on cpu
69      os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
70      os.environ["CUDA_VISIBLE_DEVICES"] = ""
71
72
73  import subprocess as sp
74  import tensorflow as tf
75
76  def gpu_memory_usage(gpu_id):
77      command = r"nvidia-smi --id=0 --query-gpu=memory.used --format=
        csv"
78      output_cmd = sp.check_output(command.split())
79      memory_used = output_cmd.decode("ascii").split("\n")[1]
80      # Get only the memory part as the result comes as '10 MiB'
81      memory_used = int(memory_used.split()[0])
82      return memory_used
83
84  def gpu_memory_total(gpu_id):
85      command = r"nvidia-smi --id=0 --query-gpu=memory.total --format=
        csv"
86      output_cmd = sp.check_output(command.split())
87      memory_total = output_cmd.decode("ascii").split("\n")[1]
88      # Get only the memory part as the result comes as '10 MiB'
89      memory_total = int(memory_total.split()[0])
90      return memory_total
91
92  # The gpu you want to check
93  gpu_id = 0
94
95  initial_memory_usage = gpu_memory_usage(gpu_id)
96  initial_memory_total = gpu_memory_total(gpu_id)
97
98  # Set up the gpu specified
99
```

```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)

sim_size = 1
full_sim_len=300
sim_len_init = 150 #because we are using one timepoint to predict the next one,
sim_len = math.ceil(sim_len_init/interval)
#sim_len_f = 150 #input length (the output should be of same length)
#so if there are 300 timepoints, the last needs to be used for prediction...and we set sim_len 299
sample_shape = (sim_len_f, size, size, size)

def make_input_3d (path):
    namestring=path[len(path)-6:len(path)]
    print("\n\n-3dmatrix--h5--3dmatrix--h5--3dmatrix--h5--3dmatrix--h5--3dmatrix--h5--3dmatrix--h5--3dmatrix--h5--\n")
    atomtrain=[]
    print("namestring",namestring)
    for typeinp in ['s1r1']:
        print(typeinp)
        input_files=sorted(glob.glob(path+"*.h5"))

        input_files_csv=sorted(glob.glob(path+"*SASA.csv"))
        input_files_csv_dist=sorted(glob.glob(path+"*s1r1*.csv"))
        print("input_files_len  len(input_files), len(input_files_csv), len(input_files_csv_dist)", len(input_files), len(input_files_csv), len(input_files_csv_dist))
        for k,filename in enumerate(input_files): #enumerate is for greater control, to put some extra contraints on steps of for loop
            sasa_k=int(k/(len(input_files_csv_dist)/(len(input_files_csv))))
            try:
                if 1:#(k==0): #for testing purposes (set to if 1: otherwise)
                    print("FILENAME:",filename)
                    print("K:",k)
                    filename_clean=filename[filename.rfind('/')+1:len(filename)-3]
                    #filename_clean=filename[:len(filename)-3]
                    print("filename_clean first:",filename_clean)
                    print("input_files_csv[k]",input_files_csv[sasa_k])
                    #break #comment out
                    if (k==0): #initial file
                        with h5py.File(filename, 'r') as dataset:
                            for key in dataset: #if there were more datasets in one h5 file!
                                print("key from h5py",key)
                                #ASSUMPTION: 300 datapoints in data
                            atomtrain = dataset[list(dataset.keys())[0]]#[:sim_len_f] #taking just first portion of the simulations
                            #the number of timepoints (pdbs, rows in exposure csv) to be skipped (to make timedistributed input for
```

```
    LSTM smaller and hence allow for bigger 3d matrix)
142                             print("atomtrain = dataset[list(dataset.
    keys())[0]][:-1]",atomtrain.shape)
143                             atomtrain=atomtrain[0::interval] #
    subsampling taking just every interval-th matrix,
144                             print("atomtrain=atomtrain[0::interval]"
    , atomtrain.shape)
145                             if quad:
146                                 #selecting quadrants:
147                                 #LEFT[:int(h / 2)]RIGHT[int(size /
    2):]
148                                 #front[:,:int(size  / 2)]back[:,int(
    size  / 2):]
149                                 #top[:,:,int(size  / 2):]bot[:,:,:
    int(h  / 2)]
150                                 #flipping into front top left
151                                 #atomtrain =np.array([atomtrain[i][:
    int(size / 2)][:,:int(size  / 2):][:,:,int(size  / 2):] for i in
    range(int(sim_len_f / 1))]) #taking just front top left quadrant
    of the matrix
152                                 front_top_left =np.array([atomtrain[
    i][:int(size / 2)][:,:int(size  / 2):][:,:,int(size  / 2):] for i
     in range(int(sim_len_f / 1))])
153                                 atomtrain1=front_top_left
154                                 atomtrain1b=np.rot90(atomtrain1, k
    =1, axes=(1,2))
155                                 atomtrain1b=np.rot90(atomtrain1b, k
    =1, axes=(1,3))
156
157                                 atomtrain1c=np.rot90(atomtrain1b, k
    =1, axes=(1,2))
158                                 atomtrain1c=np.rot90(atomtrain1c, k
    =1, axes=(1,3))
159                                 #print("atomtrain.shape",atomtrain.
    shape)
160                                 if conc_quad:
161                                     sim_size = 8
162                                     front_top_right =np.array([
    atomtrain[i][int(size / 2):][:,:int(size  / 2):][:,:,int(size  /
    2):] for i in range(int(sim_len_f / 1))])
163                                     print("atomtrain2.shape",
    front_top_right.shape)
164
165                                     atomtrain2=front_top_right
    [:,::-1, :, :]
166                                     atomtrain2b=np.rot90(atomtrain2,
     k=1, axes=(1,2))
167                                     atomtrain2b=np.rot90(atomtrain2b
    , k=1, axes=(1,3))
168
169                                     atomtrain2c=np.rot90(atomtrain2b
    , k=1, axes=(1,2))
170                                     atomtrain2c=np.rot90(atomtrain2c
    , k=1, axes=(1,3))
171
172                                     print("atomtrain2.shape",
```

```
    atomtrain2.shape)
173                                 front_bot_left =np.array([
    atomtrain[i][:int(size / 2)][:,:int(size  / 2)][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
174                                 print("atomtrain3.shape",
    front_bot_left.shape)
175                                 atomtrain3=front_bot_left[:,:,
    :, ::-1]
176                                 atomtrain3b=np.rot90(atomtrain3,
     k=1, axes=(1,2))
177                                 atomtrain3b=np.rot90(atomtrain3b
    , k=1, axes=(1,3))
178
179                                 atomtrain3c=np.rot90(atomtrain3b
    , k=1, axes=(1,2))
180                                 atomtrain3c=np.rot90(atomtrain3c
    , k=1, axes=(1,3))
181                                 print("atomtrain3.shape",
    atomtrain3.shape)
182                                 front_bot_right =np.array([
    atomtrain[i][int(size / 2):][:,:int(size  / 2)][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
183                                 atomtrain4=front_bot_right
    [:,::-1, :, ::-1]
184                                 atomtrain4b=np.rot90(atomtrain4,
     k=1, axes=(1,2))
185                                 atomtrain4b=np.rot90(atomtrain4b
    , k=1, axes=(1,3))
186
187                                 atomtrain4c=np.rot90(atomtrain4b
    , k=1, axes=(1,2))
188                                 atomtrain4c=np.rot90(atomtrain4c
    , k=1, axes=(1,3))
189                                 print("atomtrain4.shape",
    atomtrain4.shape)
190                                 back_top_left =np.array([
    atomtrain[i][:int(size / 2)][:,int(size  / 2):][:,:,int(size  /
    2):] for i in range(int(sim_len_f))])
191                                 atomtrain5=back_top_left[:,:,
    ::-1, :]
192                                 atomtrain5b=np.rot90(atomtrain5,
     k=1, axes=(1,2))
193                                 atomtrain5b=np.rot90(atomtrain5b
    , k=1, axes=(1,3))
194
195                                 atomtrain5c=np.rot90(atomtrain5b
    , k=1, axes=(1,2))
196                                 atomtrain5c=np.rot90(atomtrain5c
    , k=1, axes=(1,3))
197                                 print("atomtrain5.shape",
    atomtrain5.shape)
198                                 back_top_right =np.array([
    atomtrain[i][int(size / 2):][:,int(size  / 2):][:,:,int(size  /
    2):] for i in range(int(sim_len_f))])
199                                 atomtrain6=back_top_right
    [:,::-1, ::-1, :]
```

```
200                                     atomtrain6b=np.rot90(atomtrain6,
    k=1, axes=(1,2))
201                                     atomtrain6b=np.rot90(atomtrain6b
    , k=1, axes=(1,3))
202
203                                     atomtrain6c=np.rot90(atomtrain6b
    , k=1, axes=(1,2))
204                                     atomtrain6c=np.rot90(atomtrain6c
    , k=1, axes=(1,3))
205                                     print("atomtrain6.shape",
    atomtrain6.shape)
206                                     back_bot_right =np.array([
    atomtrain[i][int(size / 2):][:,int(size  / 2):][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
207                                     atomtrain7=back_bot_right
    [:,::-1, ::-1, ::-1]
208                                     atomtrain7b=np.rot90(atomtrain7,
    k=1, axes=(1,2))
209                                     atomtrain7b=np.rot90(atomtrain7b
    , k=1, axes=(1,3))
210
211                                     atomtrain7c=np.rot90(atomtrain7b
    , k=1, axes=(1,2))
212                                     atomtrain7c=np.rot90(atomtrain7c
    , k=1, axes=(1,3))
213                                     print("atomtrain7.shape",
    atomtrain7.shape)
214                                     back_bot_left =np.array([
    atomtrain[i][:int(size / 2)][:,int(size  / 2):][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
215                                     atomtrain8=back_bot_left[:,:,
    ::-1, ::-1]
216                                     atomtrain8b=np.rot90(atomtrain8,
    k=1, axes=(1,2))
217                                     atomtrain8b=np.rot90(atomtrain8b
    , k=1, axes=(1,3))
218
219                                     atomtrain8c=np.rot90(atomtrain8b
    , k=1, axes=(1,2))
220                                     atomtrain8c=np.rot90(atomtrain8c
    , k=1, axes=(1,3))
221                                     print("atomtrain8.shape",
    atomtrain8.shape)
222                                     print("atomtrain.shape prior to
    concatenation with quadrants",atomtrain.shape)
223                                     print(atomtrain.shape)
224                                     csv_np_array = np.genfromtxt(
    input_files_csv[sasa_k], delimiter=",", skip_header=1)
225                                     csv_np_array_dist = np.
    genfromtxt(input_files_csv_dist[k], delimiter=",", skip_header=1)
226                                     if 0: #checking the saved stuff
227                                         print("#########
    $$$$$$$$$$$$$$$$$$ LOADING FROM NPZ:")
228                                         loaded = np.load(
    filename_clean+'_01.npz')
229                                         #GEM11_s0r0_120_01.npz
```

```
230                                         print(np.array_equal(
    atomtrain1, loaded['a']))
231                                         print(loaded['a'].shape)
232
233                                         print(np.array_equal(
    csv_np_array, loaded['b']))
234                                         print(loaded['b'].shape)
235
236                                         print(np.array_equal(
    csv_np_array_dist, loaded['c']))
237                                         print(loaded['c'].shape)
238                                     if 1: #if saving
239                                         #filename_clean=
    filename_clean[filename_clean.rfind('/')+1:len(filename_clean)-3]
240                                         filename_clean='/media/
    volume/TF_TEST_20-50SS100q_2/npzs/'+typeinp+'/'+filename_clean
241                                         print("filename_clean,
    before save",filename_clean)
242                                         np.savez_compressed(
    filename_clean+'_01', a=atomtrain1, b=csv_np_array, c=
    csv_np_array_dist)
243                                         np.savez_compressed(
    filename_clean+'_01b', a=atomtrain1b, b=csv_np_array, c=
    csv_np_array_dist)
244                                         np.savez_compressed(
    filename_clean+'_01c', a=atomtrain1c, b=csv_np_array, c=
    csv_np_array_dist)
245                                         np.savez_compressed(
    filename_clean+'_02', a=atomtrain2, b=csv_np_array, c=
    csv_np_array_dist)
246                                         np.savez_compressed(
    filename_clean+'_02b', a=atomtrain2b, b=csv_np_array, c=
    csv_np_array_dist)
247                                         np.savez_compressed(
    filename_clean+'_02c', a=atomtrain2c, b=csv_np_array, c=
    csv_np_array_dist)
248                                         np.savez_compressed(
    filename_clean+'_03', a=atomtrain3, b=csv_np_array, c=
    csv_np_array_dist)
249                                         np.savez_compressed(
    filename_clean+'_03b', a=atomtrain3b, b=csv_np_array, c=
    csv_np_array_dist)
250                                         np.savez_compressed(
    filename_clean+'_03c', a=atomtrain3c, b=csv_np_array, c=
    csv_np_array_dist)
251                                         np.savez_compressed(
    filename_clean+'_04', a=atomtrain4, b=csv_np_array, c=
    csv_np_array_dist)
252                                         np.savez_compressed(
    filename_clean+'_04b', a=atomtrain4b, b=csv_np_array, c=
    csv_np_array_dist)
253                                         np.savez_compressed(
    filename_clean+'_04c', a=atomtrain4c, b=csv_np_array, c=
    csv_np_array_dist)
254                                         np.savez_compressed(
    filename_clean+'_05', a=atomtrain5, b=csv_np_array, c=
```

```
    csv_np_array_dist)
255                                     np.savez_compressed(
    filename_clean+'_05b', a=atomtrain5b, b=csv_np_array, c=
    csv_np_array_dist)
256                                     np.savez_compressed(
    filename_clean+'_05c', a=atomtrain5c, b=csv_np_array, c=
    csv_np_array_dist)
257                                     np.savez_compressed(
    filename_clean+'_06', a=atomtrain6, b=csv_np_array, c=
    csv_np_array_dist)
258                                     np.savez_compressed(
    filename_clean+'_06b', a=atomtrain6b, b=csv_np_array, c=
    csv_np_array_dist)
259                                     np.savez_compressed(
    filename_clean+'_06c', a=atomtrain6c, b=csv_np_array, c=
    csv_np_array_dist)
260                                     np.savez_compressed(
    filename_clean+'_07', a=atomtrain7, b=csv_np_array, c=
    csv_np_array_dist)
261                                     np.savez_compressed(
    filename_clean+'_07b', a=atomtrain7b, b=csv_np_array, c=
    csv_np_array_dist)
262                                     np.savez_compressed(
    filename_clean+'_07c', a=atomtrain7c, b=csv_np_array, c=
    csv_np_array_dist)
263                                     np.savez_compressed(
    filename_clean+'_08', a=atomtrain8, b=csv_np_array, c=
    csv_np_array_dist)
264                                     np.savez_compressed(
    filename_clean+'_08b', a=atomtrain8b, b=csv_np_array, c=
    csv_np_array_dist)
265                                     np.savez_compressed(
    filename_clean+'_08c', a=atomtrain8c, b=csv_np_array, c=
    csv_np_array_dist)
266
267                                 #atomtrain=np.concatenate((
    atomtrain1, atomtrain2, atomtrain3,atomtrain4,atomtrain5,
    atomtrain6,atomtrain7,atomtrain8))
268
269                                 print("atomtrain.shape post
    concatenation with quadrants",atomtrain.shape)
270                             else:
271                                 atomtrain=front_top_left
272                             if sect:
273                                 atomtrain=atomtrain[:,(nsize-
    width):nsize,:,:,] #--taking the right portion of the matrix
274                         atomtrain=atomtrain.reshape(sim_size,
    sim_len_f, nsize1, nsize, nsize)
275                         print("atomtrain = atomtrain.reshape(
    sim_size, sim_len, size, size, size)",atomtrain.shape)
276                         atomtrain=atomtrain.reshape(list(
    atomtrain.shape) + [1])
277                         print("atomtrain = atomtrain.reshape(
    list(atomtrain.shape) + [1])",atomtrain.shape)
278                     dataset.close()
279
```

```
280                     else: #every non-initial file, gets concatenated
    to the first one
281                         with h5py.File(filename, 'r') as dataset:
282                             for key in dataset: #if there were more
    datasets in one h5 file!
283                                 print("key from h5py",key)
284                             print("sim_len_f",sim_len_f)
285                             print("type(dataset[list(dataset.keys())
    [0]]",type(dataset[list(dataset.keys())[0]]))
286                             print("dataset[list(dataset.keys())[0]].
    shape",dataset[list(dataset.keys())[0]].shape)
287                             atomtrain2= dataset[list(dataset.keys())
    [0]]#[:sim_len_f] #taking just first portion of the simulations
288                             #save(filename_clean+'_01.npy',
    atomtrain2)
289                             atomtrain2=atomtrain2[0::interval] #
    subsampling taking just every interval-th matrix,
290                             #the number of timepoints (pdbs, rows in
     exposure csv) to be skipped (to make timedistributed input for
    LSTM smaller and hence allow for bigger 3d matrix)
291                             if quad:
292                                 front_top_left =np.array([atomtrain2
    [i][:int(size / 2)][:,:int(size  / 2):][:,:,int(size  / 2):] for
    i in range(int(sim_len_f / 1))]) #taking just front top left
    quadrant of the matrix
293                                 atomtrain1=front_top_left
294                                 atomtrain1b=np.rot90(atomtrain1, k
    =1, axes=(1,2))
295                                 atomtrain1b=np.rot90(atomtrain1b, k
    =1, axes=(1,3))
296
297                                 atomtrain1c=np.rot90(atomtrain1b, k
    =1, axes=(1,2))
298                                 atomtrain1c=np.rot90(atomtrain1c, k
    =1, axes=(1,3))
299                                 if conc_quad:
300                                     sim_size = 8
301                                     front_top_right =np.array([
    atomtrain2[i][int(size / 2):][:,:int(size  / 2):][:,:,int(size  /
     2):] for i in range(int(sim_len_f / 1))])
302                                     atomtrain_2=front_top_right
    [:,::-1, :, :]
303                                     atomtrain_2b=np.rot90(
    atomtrain_2, k=1, axes=(1,2))
304                                     atomtrain_2b=np.rot90(
    atomtrain_2b, k=1, axes=(1,3))
305
306                                     atomtrain_2c=np.rot90(
    atomtrain_2b, k=1, axes=(1,2))
307                                     atomtrain_2c=np.rot90(
    atomtrain_2c, k=1, axes=(1,3))
308                                     front_bot_left =np.array([
    atomtrain2[i][:int(size / 2)][:,:int(size  / 2)][:,:,:int(size  /
     2)] for i in range(int(sim_len_f))])
309                                     atomtrain3=front_bot_left[:,:,
    :, ::-1]
```

```
310                                     atomtrain3b=np.rot90(atomtrain3,
    k=1, axes=(1,2))
311                                     atomtrain3b=np.rot90(atomtrain3b
    , k=1, axes=(1,3))
312
313                                     atomtrain3c=np.rot90(atomtrain3b
    , k=1, axes=(1,2))
314                                     atomtrain3c=np.rot90(atomtrain3c
    , k=1, axes=(1,3))
315                                     front_bot_right =np.array([
    atomtrain2[i][int(size / 2):][:,:int(size  / 2)][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
316                                     atomtrain4=front_bot_right
    [:,::-1, :, ::-1]
317                                     atomtrain4b=np.rot90(atomtrain4,
    k=1, axes=(1,2))
318                                     atomtrain4b=np.rot90(atomtrain4b
    , k=1, axes=(1,3))
319
320                                     atomtrain4c=np.rot90(atomtrain4b
    , k=1, axes=(1,2))
321                                     atomtrain4c=np.rot90(atomtrain4c
    , k=1, axes=(1,3))
322                                     back_top_left =np.array([
    atomtrain2[i][:int(size / 2)][:,int(size  / 2):][:,:,int(size  /
    2):] for i in range(int(sim_len_f))])
323                                     atomtrain5=back_top_left[:,:,
    ::-1, :]
324                                     atomtrain5b=np.rot90(atomtrain5,
    k=1, axes=(1,2))
325                                     atomtrain5b=np.rot90(atomtrain5b
    , k=1, axes=(1,3))
326
327                                     atomtrain5c=np.rot90(atomtrain5b
    , k=1, axes=(1,2))
328                                     atomtrain5c=np.rot90(atomtrain5c
    , k=1, axes=(1,3))
329                                     back_top_right =np.array([
    atomtrain2[i][int(size / 2):][:,int(size  / 2):][:,:,int(size  /
    2):] for i in range(int(sim_len_f))])
330                                     atomtrain6=back_top_right
    [:,::-1, ::-1, :]
331                                     atomtrain6b=np.rot90(atomtrain6,
    k=1, axes=(1,2))
332                                     atomtrain6b=np.rot90(atomtrain6b
    , k=1, axes=(1,3))
333
334                                     atomtrain6c=np.rot90(atomtrain6b
    , k=1, axes=(1,2))
335                                     atomtrain6c=np.rot90(atomtrain6c
    , k=1, axes=(1,3))
336                                     back_bot_right =np.array([
    atomtrain2[i][int(size / 2):][:,int(size  / 2):][:,:,:int(size  /
    2)] for i in range(int(sim_len_f))])
337                                     atomtrain7=back_bot_right
    [:,::-1, ::-1, ::-1]
```

```
338                                     atomtrain7b=np.rot90(atomtrain7,
    k=1, axes=(1,2))
339                                     atomtrain7b=np.rot90(atomtrain7b
    , k=1, axes=(1,3))
340
341                                     atomtrain7c=np.rot90(atomtrain7b
    , k=1, axes=(1,2))
342                                     atomtrain7c=np.rot90(atomtrain7c
    , k=1, axes=(1,3))
343                                     back_bot_left =np.array([
    atomtrain2[i][:int(size / 2)][:,int(size  / 2):][:,:,:int(size  /
    2)] for i in range(int(sim_len_f / 1))])
344                                     atomtrain8=back_bot_left[:,:,
    ::-1, ::-1]
345                                     atomtrain8b=np.rot90(atomtrain8,
    k=1, axes=(1,2))
346                                     atomtrain8b=np.rot90(atomtrain8b
    , k=1, axes=(1,3))
347
348                                     atomtrain8c=np.rot90(atomtrain8b
    , k=1, axes=(1,2))
349                                     atomtrain8c=np.rot90(atomtrain8c
    , k=1, axes=(1,3))
350                                     print("atomtrain2.shape prior to
    concatenation with quadrants",atomtrain2.shape)
351                                     #atomtrain2=np.concatenate((
    atomtrain1, atomtrain_2, atomtrain3,atomtrain4,atomtrain5,
    atomtrain6,atomtrain7,atomtrain8))
352                                     print("atomtrain2.shape post
    concatenation with quadrants",atomtrain2.shape)
353
354                                     #csv_np_array = np.genfromtxt(
    input_files_csv[sasa_k], delimiter=",", skip_header=1)
355
356                                     df = pd.DataFrame(pd.read_csv(
    input_files_csv[sasa_k]))
357                                     for i in range(len(df.columns)
    -1):
358                                         df.iloc[:,i+1] = df.iloc[:,i
    +1].rolling(window=wind,min_periods=1).mean()
359                                     csv_np_array=df.to_numpy()
360
361
362                                     csv_np_array_dist = np.
    genfromtxt(input_files_csv_dist[k], delimiter=",", skip_header=1)
363                                     if 1: #if saving
364                                         #filename_clean=
    filename_clean[filename_clean.rfind('/')+1:len(filename_clean)-3]
365                                         filename_clean='/media/
    volume/TF_TEST_20-50SS100q_2/npzs/'+typeinp+'/'+filename_clean
366                                         #print("############
    filename_clean",filename_clean)
367                                         print("filename_clean,
    before save",filename_clean)
368                                         np.savez_compressed(
    filename_clean+'_01', a=atomtrain1, b=csv_np_array,c=
```

```
    csv_np_array_dist)
369                                     np.savez_compressed(
    filename_clean+'_01b', a=atomtrain1b, b=csv_np_array,c=
    csv_np_array_dist)
370                                     np.savez_compressed(
    filename_clean+'_01c', a=atomtrain1c, b=csv_np_array,c=
    csv_np_array_dist)
371                                     np.savez_compressed(
    filename_clean+'_02', a=atomtrain_2, b=csv_np_array,c=
    csv_np_array_dist)
372                                     np.savez_compressed(
    filename_clean+'_02b', a=atomtrain_2b, b=csv_np_array,c=
    csv_np_array_dist)
373                                     np.savez_compressed(
    filename_clean+'_02c', a=atomtrain_2c, b=csv_np_array,c=
    csv_np_array_dist)
374                                     np.savez_compressed(
    filename_clean+'_03', a=atomtrain3, b=csv_np_array,c=
    csv_np_array_dist)
375                                     np.savez_compressed(
    filename_clean+'_03b', a=atomtrain3b, b=csv_np_array,c=
    csv_np_array_dist)
376                                     np.savez_compressed(
    filename_clean+'_03c', a=atomtrain3c, b=csv_np_array,c=
    csv_np_array_dist)
377                                     np.savez_compressed(
    filename_clean+'_04', a=atomtrain4, b=csv_np_array,c=
    csv_np_array_dist)
378                                     np.savez_compressed(
    filename_clean+'_04b', a=atomtrain4b, b=csv_np_array,c=
    csv_np_array_dist)
379                                     np.savez_compressed(
    filename_clean+'_04c', a=atomtrain4c, b=csv_np_array,c=
    csv_np_array_dist)
380                                     np.savez_compressed(
    filename_clean+'_05', a=atomtrain5, b=csv_np_array,c=
    csv_np_array_dist)
381                                     np.savez_compressed(
    filename_clean+'_05b', a=atomtrain5b, b=csv_np_array,c=
    csv_np_array_dist)
382                                     np.savez_compressed(
    filename_clean+'_05c', a=atomtrain5c, b=csv_np_array,c=
    csv_np_array_dist)
383                                     np.savez_compressed(
    filename_clean+'_06', a=atomtrain6, b=csv_np_array,c=
    csv_np_array_dist)
384                                     np.savez_compressed(
    filename_clean+'_06b', a=atomtrain6b, b=csv_np_array,c=
    csv_np_array_dist)
385                                     np.savez_compressed(
    filename_clean+'_06c', a=atomtrain6c, b=csv_np_array,c=
    csv_np_array_dist)
386                                     np.savez_compressed(
    filename_clean+'_07', a=atomtrain7, b=csv_np_array,c=
    csv_np_array_dist)
387                                     np.savez_compressed(
```

```
    filename_clean+'_07b', a=atomtrain7b, b=csv_np_array,c=
    csv_np_array_dist)
388                                         np.savez_compressed(
    filename_clean+'_07c', a=atomtrain7c, b=csv_np_array,c=
    csv_np_array_dist)
389                                         np.savez_compressed(
    filename_clean+'_08', a=atomtrain8, b=csv_np_array,c=
    csv_np_array_dist)
390                                         np.savez_compressed(
    filename_clean+'_08b', a=atomtrain8b, b=csv_np_array,c=
    csv_np_array_dist)
391                                         np.savez_compressed(
    filename_clean+'_08c', a=atomtrain8c, b=csv_np_array,c=
    csv_np_array_dist)
392                                 else:
393                                     atomtrain2=front_top_left
394                                 if sect:
395                                         atomtrain2=atomtrain2[:,(
    nsize-width):nsize,:,:,] #--taking the right portion of the
    matrix
396                                 atomtrain2=atomtrain2.reshape(sim_size,
    sim_len_f, nsize1, nsize, nsize)
397                                 atomtrain2=atomtrain2.reshape(list(
    atomtrain2.shape) + [1])
398                                 print("prior to np.concatenate((
    atomtrain, atomtrain2), axis = 0)")
399                                 print("atomtrain.shape",atomtrain.shape)
400                                 print("atomtrain2.shape",atomtrain.shape
    )
401                                 #atomtrain = np.concatenate((atomtrain,
    atomtrain2), axis = 0)
402                                 print("asizeof.asizeof(atomtrain)/1000
    000 000(Gb)",int(asizeof.asizeof(atomtrain))/1000000000)
403                                 print("postconc")
404                                 print("atomtrain2.shape ",atomtrain2.
    shape)
405                                 print("atomtrain.shape ",atomtrain.shape
    )
406                         dataset.close()
407             except:
408                 print("error occured!
                                55555555555555555555
    ****************")
409         return atomtrain
410
411 def make_input_expo (path):
412     print("\n\n-exposure--CSV--exposure--CSV--exposure--CSV--
    exposure--CSV--exposure--CSV--exposure--CSV--exposure--CSV--\n")
413     import csv
414     input_files_csv=sorted(glob.glob(path+"*.csv"))
415     for k,filename_csv in enumerate(input_files_csv): #enumerate is
    for greater control, to put some extra contraints on steps of for
     loop
416         if 1:#k==0:  #for testing! (set to if 1: otherwise)
417             print("FILENAME:",filename_csv)
418             drugratio=filename_csv[filename_csv.rfind('/')+4:
```

```
    filename_csv.rfind('/')+5] #finding the ratio in the filename if
    the filename is in format /PAN11 fore example
419             ssratio=filename_csv[filename_csv.rfind('/')+5:
    filename_csv.rfind('/')+6] #finding the ratio in the filename if
    the filename is in format /PAN11 fore example
420             print("K:",k)
421             ratio_add=np.array([int(drugratio), int(ssratio)])
422             if "PAN" in filename_csv: #adding the information about
    drug #needs to be refined, reall one hot encoding
423                 drug_add = np.array([1, 0, 0, 0, 0])
424             if (("OQL" in filename_csv) or ("S1" in filename_csv)):
    #adding the information about drug #needs to be refined, reall
    one hot encoding
425                 drug_add = np.array([0, 1, 0, 0, 0])
426             if "GEM" in filename_csv: #adding the information about
    drug #needs to be refined, reall one hot encoding
427                 drug_add = np.array([0, 0, 1, 0, 0])
428             if "NCL" in filename_csv: #adding the information about
    drug #needs to be refined, reall one hot encoding
429                 drug_add = np.array([0, 0, 0, 1, 0])
430             if "NHQ" in filename_csv: #adding the information about
    drug #needs to be refined, reall one hot encoding
431                 drug_add = np.array([0, 0, 0, 0, 1])
432             if (k==0): #initial file
433                 exp_val = np.array([])
434                 arr_len = 0
435                 #expo_train = np.array([])
436                 with open(filename_csv) as csv_file:
437                     csv_reader = csv.reader(csv_file, delimiter=',')
438                     line_count = 0
439                     for row in csv_reader:
440                         if line_count > 0:
441                             row = [float(v) for v in row]
442                             row = np.array(row)
443                             row=np.concatenate([ratio_add,row])
444                             row=np.concatenate([drug_add,row])
445 
446                             #print("row", row)
447                             #print("row.shape", row.shape)
448                             if len(exp_val) == 0:
449                                 exp_val = row
450                                 arr_len = len(row)
451                             else:
452                                 exp_val = np.vstack((exp_val,row))
453                         if(line_count > full_sim_len): #taking just
    first portion of the simulations):
454                             break
455                         line_count += 1
456                 exp_train = exp_val.copy()[:sim_len_f]
457                 print("exp_train = exp_val.copy()[0:-1]",exp_train.
    shape)
458                 exp_train=exp_train[0::interval] #subsampling taking
     just every interval-th matrix,
459                 print("exp_train=exp_train[0::interval] ",exp_train.
    shape)
460                 #the number of timepoints (pdbs, rows in exposure
```

```
    csv) to be skipped (to make timedistributed input for LSTM
    smaller and hence allow for bigger 3d matrix)
461                 exp_out = exp_val.copy()[-sim_len_f:]#[1:]
462                 exp_out=exp_out[0::interval] #subsampling taking
    just every interval-th matrix,
463                 exp_train=exp_train.reshape(sim_size, sim_len_f,
    arr_len)
464                 if conc_quad:
465                     exp_train=np.concatenate((exp_train,exp_train,
    exp_train,exp_train,exp_train,exp_train,exp_train,exp_train)) #
    simply concatenating - oktadupling the exposure input
466                 print("exp_train=exp_train.reshape(sim_size, sim_len
    , arr_len)",exp_train.shape)
467                 #exp_out=exp_out.reshape(sim_size, 1, arr_len)
468                 exp_out=exp_out.reshape(sim_size, sim_len_f, arr_len
    )
469                 if conc_quad:
470                     exp_out=np.concatenate((exp_out,exp_out,exp_out,
    exp_out,exp_out,exp_out,exp_out,exp_out))
471
472                 print("exp_out.shape",exp_out.shape)
473                 #save(filename_clean+'_02.npy', atomtrain2)
474             else: #every non-initial file, gets concatenated to the
    first one
475                 exp_val = np.array([])
476                 arr_len = 0
477                 #expo_train = np.array([])
478                 with open(filename_csv) as csv_file:
479                     csv_reader = csv.reader(csv_file, delimiter=',')
480                     line_count = 0
481                     for row in csv_reader:
482                         if line_count > 0:
483                             row = [float(v) for v in row]
484                             row = np.array(row)
485                             row=np.concatenate([ratio_add,row])
486                             row=np.concatenate([drug_add,row])
487                             #print("row", row)
488                             if len(exp_val) == 0:
489                                 exp_val = row
490                                 arr_len = len(row)
491                             else:
492                                 exp_val = np.vstack((exp_val,row))
493                         if(line_count > sim_len_f): #taking just
    first portion of the simulations):
494                             break
495                         line_count += 1
496                 exp_train2 = exp_val.copy()[:sim_len_f]
497                 exp_train2=exp_train2[0::interval] #subsampling
    taking just every interval-th matrix,
498                 exp_train2=exp_train2.reshape(sim_size, sim_len_f,
    arr_len)
499                 exp_out2 = exp_val.copy()[-sim_len_f:]
500                 exp_out2=exp_out2[0::interval] #subsampling taking
    just every interval-th matrix,
501                 exp_train2=exp_train2.reshape(sim_size, sim_len_f,
    arr_len)
```

```
                #if conc_quad:
                    #exp_train2=np.concatenate((exp_train2,
    exp_train2,exp_train2,exp_train2,exp_train2,exp_train2,exp_train2
    ,exp_train2))
                print("exp_train2.shape",exp_train2.shape)
                #exp_out2=exp_out2.reshape(sim_size, 1, arr_len)
                exp_out2=exp_out2.reshape(sim_size, sim_len_f,
    arr_len)
                #if conc_quad:
                    #exp_out2=np.concatenate((exp_out2,exp_out2,
    exp_out2,exp_out2,exp_out2,exp_out2,exp_out2,exp_out2))
                print("exp_out2.shape",exp_out2.shape)
                exp_train= np.concatenate((exp_train, exp_train2),
    axis = 0)
                print("exp_train.shape",exp_train.shape)
                exp_out= np.concatenate((exp_out, exp_out2), axis =
    0)
                print("exp_out.shape",exp_out.shape)

     exp_out=exp_out[:,:,:exp_n_col_y] #taking just first four
    columns (so up to, including 2.2Hdn) out of 15
     print("FINAL!!!!!!!!!!! exp_out.shape",exp_out.shape)
     return_train_test = [exp_train,exp_out, arr_len]#, 4 because of
    above, taking just first four columns instead of 15...
     return(return_train_test)

#calling functions to import the train and test data
print("CHECKCHEKCHC##########CHEKCHC#####CHEKCHC#####CHEKCHC#####
    CHEKCHC#####CHEKCHC##########")


print("TRAIN DATA:")
atomtrain_inp=make_input_3d(pathtrain)
```

Listing 6: Python script to flip the quadrants of NP in Numpy 3D matrices and combine with distances and SASA exposure into Numpy compressed (.npz) files to be feed into the model.

```

#run in Dione terminal with example line:" srun -p gpu --nodelist=
    di37 -t 99:00:00 --mem=20G python3
    CNN_LSTM_29_3inp_4conv_drp05_pd_512krn_d.py 120 s1r1 3 1 1 0 0 0
    60 10 1 299 299 0 3 mae 100 1 5
    convdrp05preLSTMdrpLSTM256neur_299TMPNTSinterval3 0 256 1 &>
    printout_5conv_drp05_pd_512krn_5convdrp05p256neur_299TMPNTSinterval3
    .out &
#change the parameters as explained bellow #INPUT PARAMETERS

from tensorflow.keras.layers import Conv3D, MaxPooling3D, Flatten,
    Dense,LSTM, TimeDistributed
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.layers import Dropout, Input,
    BatchNormalization
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adadelta
from tensorflow.keras.models import Model, Sequential
```

```
from tensorflow.keras import models
from tensorflow.keras import utils
from tensorflow.keras.callbacks import CSVLogger
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf
import numpy as np
import tensorflow.keras
import h5py
import math
import time
from pympler import asizeof #to check the memory size of objects
from numpy import load
import csv
import sys
import os #psutil
import datetime
import shutil
import glob
import subprocess as sp
import random


from datetime import datetime
start_time = datetime.now()
# PLOTING WITH MATPLOT
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([1,2,3])
fig.savefig('test.png')

cnn_fin_dense_size=17 #### number of columns from final exposure y


#INPUT PARAMETERS

size=int(sys.argv[1]) #the size of the input matrix
typeinp=str(sys.argv[2]) #the type of the matrix, whether there are drugcentres, supsubscentres, other molecule centres etc
interval=int(sys.argv[3]) #the number of timepoints (pdbs, rows in exposure csv) to be skipped (to make timedistributed input for LSTM smaller and hence allow for bigger 3d matrix)
quad=int(sys.argv[4]) #quadrant (1) or no (0), taking just one eighth front top left quadrant of 3D matrix, to lower the size of input
conc_quad=int(sys.argv[5]) #if one wants to take the other 7 quadrants into consideration, it will concatenate them!

sect=int(sys.argv[6]) # sectioning of the 3d matrix (can be quadrant even further) (1 - yes) (0 - no), taking the right most portion of the matrix, see below
width=int(sys.argv[7]) # width of the section from above, how many cells, how much of the right most portion of the matrix
cpu=int(sys.argv[8]) # CPU mode - disable usage of GPU! (1 - CPU, 0
```

```
    - GPU)
cnn_fin_dense_size=int(sys.argv[9]) #### number of columns from
    final exposure y
num_epochs=int(sys.argv[10]) #number of epochs
batch_size=int(sys.argv[11])
sim_len_f=int(sys.argv[12]) #input length (the output should be of
    same length)
sim_len_f = math.ceil(sim_len_f/interval)
out_len=int(sys.argv[13]) # numb of inputs per LSTM
out_len = math.ceil(out_len/interval)
CUDA_VISIBLE_DEVICES=str(sys.argv[14]) #either 1 or 0
kernel_size=int(sys.argv[15])
loss_funct=str(sys.argv[16]) #mae, mse!
average=int(sys.argv[17])#input_how much it was averaged
files_nth=int(sys.argv[18])#taking just every nth file from original
     list, for testing purposes do not want to run through all the
    files for faster outcome
unique_about_this_run=str(sys.argv[19]) #write some unique
    descriptive notion about this training used for discerning saved
    models etc
load_weights=int(sys.argv[20])#whether this training should load
    previous loads
n_neurons=int(sys.argv[21])#number of neurons in LSTM layer
fit_model=int(sys.argv[22])#whether to fit the model or not
if quad: #adjusting reported size of input matrix according to
    taking only quadrants
     nsize=int(size/2)
else:
     nsize=size

if sect: #adjusting the first dimension (width) of 3d matrix
    appropriatelty according to sectioning
     nsize1=width
else:
     nsize1=nsize

#SAVING THE INFORMATION ON TRAINING
path_new_folder = os.getcwd()+"/"+str(sim_len_f)+"_"+str(width)+"_"+
    typeinp+"_"+str(kernel_size)+"_"+loss_funct+"_"+str(average)+"_"+
     datetime.now().strftime("%Y%m%d-%H%M%S")+unique_about_this_run+"
    /"
os.mkdir(path_new_folder)

#function to print the printout also into a file
def fprint(*argv):
     for arg in argv:
          print (arg)
     original_stdout = sys.stdout # Save a reference to the original
    standard output
     with open(path_new_folder+'/'+str(sim_len_f)+"_"+str(width)+"_"+
    typeinp+"_"+str(kernel_size)+"_"+loss_funct+"_"+str(average)+'.
    txt', 'a') as f:
          sys.stdout = f # Change the standard output to the file we
    created.
          for arg in argv:
               print (arg)
```

```
97          sys.stdout = original_stdout # Reset the standard output to
    its original value
98
99
100 fprint(">>>>>>>>INPUT PARAMETERS:")
101 fprint("datetime.now(),",datetime.now())
102 fprint("size: ,",size)
103 fprint("typeinp: ,",typeinp)
104 fprint("interval between timepoints: ,",interval)
105 fprint("use of only single quadrant of matrix bool: ,",quad)
106 fprint("use only section of matrix bool: ,",sect)
107 fprint("width of section: ,",width)
108 fprint("usage of cpu only bool: ,",cpu)
109 fprint("cnn_fin_dense_size: ,",cnn_fin_dense_size) #### number of
    columns from final exposure y
110 fprint("num_epochs: ,",num_epochs) #number of epochs
111 fprint("batch_size: ,",batch_size) #number of epochs
112 fprint("sim_len_f: ,",sim_len_f) #input length (the output should be
     of same length)
113 fprint("out_len: ,",out_len) # numb of inputs per LSTM
114 fprint("CUDA_VISIBLE_DEVICES: ,",CUDA_VISIBLE_DEVICES) #either 1 or
    0
115 fprint("kernel_size: ,",kernel_size)
116 fprint("loss_funct: ,",loss_funct) #mae, mse!
117 fprint("average: ,",average)#input_how much it was averaged
118
119
120 path=os.getcwd()+'/npzs/'+str(typeinp)+'/'#sys.argv[3] #path to the
    input #make sure the train data in train folder!
121
122
123 if cpu: #disabling gpu to force run it on cpu
124     os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
125     os.environ["CUDA_VISIBLE_DEVICES"] = ""
126
127
128 gpus = tf.config.list_physical_devices('GPU')
129
130
131 #giving chmod rights to files in path
132 def recursive_chmod(path, permission): #0o777
133     for dirpath, dirnames, filenames in os.walk(path):
134         os.chmod(dirpath, permission)
135         for filename in filenames:
136             os.chmod(os.path.join(dirpath, filename), permission)
137
138
139
140 def gpu_memory_usage(gpu_id):
141     command = r"nvidia-smi --id=0 --query-gpu=memory.used --format=
    csv"
142     output_cmd = sp.check_output(command.split())
143     memory_used = output_cmd.decode("ascii").split("\n")[1]
144     # Get only the memory part as the result comes as '10 MiB'
145     memory_used = int(memory_used.split()[0])
146     return memory_used
```

```
147
148 def gpu_memory_total(gpu_id):
149     command = r"nvidia-smi --id=0 --query-gpu=memory.total --format=
    csv"
150     output_cmd = sp.check_output(command.split())
151     memory_total = output_cmd.decode("ascii").split("\n")[1]
152     # Get only the memory part as the result comes as '10 MiB'
153     memory_total = int(memory_total.split()[0])
154     return memory_total
155
156 # The gpu you want to check
157 gpu_id = 0
158
159 initial_memory_usage = gpu_memory_usage(gpu_id)
160 initial_memory_total = gpu_memory_total(gpu_id)
161
162 # Set up the gpu specified
163
164 gpu_devices = tf.config.experimental.list_physical_devices('GPU')
165 for device in gpu_devices:
166     tf.config.experimental.set_memory_growth(device, True)
167
168
169
170 sim_size = 1
171 full_sim_len=300
172 sim_len_init = 150 #because we are using one timepoint to predict
    the next one,
173 sim_len = math.ceil(sim_len_init/interval)
174 #sim_len_f = 150 #input length (the output should be of same length)
175 #so if there are 300 timepoints, the last needs to be used for
    prediction...and we set sim_len 299
176 sample_shape = (sim_len_f, size, size, size)
177
178 files=glob.glob(path+"*.npz")
179 print("len(files) pre",len(files))
180 files=files[0::files_nth]
181 print("len(files) post",len(files))
182
183 #filtering for the files that have the correct averraging:
184 files_filt=[]
185
186 for s_file in files:
187     seclast_index=s_file[:s_file.rfind("_")].rfind("_") #looking for
     second last occurence of "_", in file like"
    S1_11R4_3_s1r1_50_50_03c.npz"
188     last_index=s_file.rfind("_")#looking for last occurence of "_",
    in file like"S1_11R4_3_s1r1_50_50_03c.npz"
189     files_cur_averaging=int(s_file[(seclast_index+1):last_index])
190     if files_cur_averaging == average:
191         files_filt.append(s_file)
192
193 files=files_filt
194
195 print("FILES POST FILTERING",files)
196
```

```
fprint("CHECK THIS!,")
fprint('path,',path)
fprint('type(files),',type(files))
fprint('len(files),',len(files))
test_t=0 #####IMPORTANT CHECK CHANGE to print the filenames of test files
test_t_filenames=[]


#data generator function that feeds tensorflow data on the fly
def tf_data_generator(file_list,sim_len_f,test_t, batch_size = batch_size):
    print("DGW1")
    i = 0
    fprint("len(file_list),",len(file_list))
    random.shuffle(file_list)
    while True:
        if i*batch_size >= len(file_list):
            i = 0
            np.random.shuffle(file_list)
        else:
            file_chunk = file_list[i*batch_size:(i+1)*batch_size]
            #data = []
            atom_x= np.array([])
            exp_x = np.array([])
            dist_x = np.array([])
            exp_y = np.array([])
            for k,file in enumerate(file_chunk):

                print("k",k)
                temp = np.load(file)
                filename=str(file)
                if test_t:
                    filename_sh=filename[(filename.rfind("/")+1):filename.rfind(".npz")]
                    print("!!!in gen TEST file:",filename_sh)
                    test_t_filenames.append(filename_sh)
                drugratio=tf.constant(filename[filename.rfind("/")+4:filename.rfind("/")+5]) #finding the ratio in the filename if the filename is in format /PAN11 fore example
                ssratio=tf.constant(filename[filename.rfind("/")+5:filename.rfind("/")+6]) #finding the ratio in the filename if the filename is in format /PAN11 fore example

                ratio_add=np.array([int(drugratio), int(ssratio)]) #creating an array that informs of the drug ratio
                if "PAN" in filename: #adding the information about drug #needs to be refined, reall one hot encoding
                    drug_add = np.array([1, 0, 0, 0, 0])
                if (("OQL" in filename) or ("S1" in filename)): #adding the information about drug #needs to be refined, reall one hot encoding
                    drug_add = np.array([0, 1, 0, 0, 0])
                if "GEM" in filename: #adding the information about drug #needs to be refined, reall one hot encoding
                    drug_add = np.array([0, 0, 1, 0, 0])
```

```
241                 if "NCL" in filename: #adding the information about
    drug #needs to be refined, reall one hot encoding
242                     drug_add = np.array([0, 0, 0, 1, 0])
243                 if "NHQ" in filename: #adding the information about
    drug #needs to be refined, reall one hot encoding
244                     drug_add = np.array([0, 0, 0, 0, 1])
245                     ######################ADD OTHER DRUG TYPES!,
    expand the array!
246 
247                 drug_add=np.concatenate([ratio_add,drug_add])#JOIN
    ratio+drug
248 
249                 if k == 0:
250                     #atom_3d_matrix_numpy_array
251                     atom_x=temp['a']
252                     atom_x=atom_x[0::interval] #subsampling taking
    just every interval-th matrix,
253                     atom_x=atom_x[:sim_len_f]
254                     if sect:
255                         atom_x=atom_x[:,(nsize-width):nsize,:,:,] #
    --taking the right portion of the matrix
256                     atom_x=atom_x.reshape(sim_size, sim_len_f,
    nsize1, nsize, nsize)
257                     atom_x=atom_x.reshape(list(atom_x.shape) + [1])
258 
259                     drug_add = np.expand_dims(drug_add, axis=0)
260                     drug_static = np.repeat(drug_add, repeats=
    full_sim_len, axis=0) #full_sim_len = 300 #creating an array that
     informs of the drug type, one hot encoding
261                     drug_static=drug_static[0::interval]
262                     drug_static_len=drug_static.shape[-1]
263 
264                     drug_static=drug_static[:sim_len_f]
265                     drug_static=drug_static.reshape(sim_size,
    sim_len_f, drug_static_len)
266 
267                     #exposures_matrix_numpy_array
268                     exp_temp=temp['b']
269 
270                     exp_temp=exp_temp[0::interval] #subsampling
    taking just every interval-th matrix,
271                     arr_len=exp_temp.shape[-1]
272                     exp_x=exp_temp[:sim_len_f]
273 
274                     exp_y =exp_temp[-out_len:] #taking the last
    portion of the exposures of the size out_len specified by user
    input
275                     exp_x=exp_x.reshape(sim_size, sim_len_f, arr_len
    )
276                     exp_y = exp_y[:, -1] #keeping just the
    percentage of SASA
277                     exp_y=exp_y.reshape(sim_size, out_len, 1)
278                     exp_y=np.round(exp_y, 2)
279 
280                     dist_temp=temp['c']
281                     dist_temp=dist_temp[0::interval] #subsampling
```

```
    taking just every interval-th matrix,
282                 dist_x=dist_temp[:sim_len_f]
283 
284                 dist_len=dist_temp.shape[-1]
285 
286                 dist_x=dist_x.reshape(sim_size, sim_len_f,
    dist_len)
287 
288 
289             if k!=0:
290                 atom_x_temp=temp['a']
291                 atom_x_temp=atom_x_temp[0::interval] #
    subsampling taking just every interval-th matrix,
292                 atom_x_temp=atom_x_temp[:sim_len_f]
293                 if sect:
294                     atom_x=atom_x[:,(nsize-width):nsize,:,:,] #
    --taking the right portion of the matrix
295                 atom_x_temp=atom_x_temp.reshape(sim_size,
    sim_len_f, nsize1, nsize, nsize)
296                 atom_x_temp=atom_x_temp.reshape(list(atom_x_temp
    .shape) + [1])
297                 atom_x = np.concatenate((atom_x, atom_x_temp),
    axis = 0)
298 
299 
300                 drug_add = np.expand_dims(drug_add, axis=0)
301                 drug_static_temp = np.repeat(drug_add, repeats=
    full_sim_len, axis=0) #full_sim_len = 300 #creating an array that
     informs of the drug type, one hot encoding
302                 drug_static=drug_static[0::interval]
303                 drug_static_len=drug_static_temp.shape[-1]
304                 drug_static_temp=drug_static_temp[:sim_len_f]
305                 drug_static_temp=drug_static_temp.reshape(
    sim_size, sim_len_f, drug_static_len)
306                 drug_static= np.concatenate((drug_static,
    drug_static_temp), axis = 0)
307                 exp_temp=temp['b']
308                 exp_temp=exp_temp[0::interval] #subsampling
    taking just every interval-th matrix,
309                 exp_x_temp = exp_temp[:sim_len_f]
310                 exp_x_temp=exp_x_temp.reshape(sim_size,
    sim_len_f, arr_len)
311                 exp_x= np.concatenate((exp_x, exp_x_temp), axis
    = 0)
312                 exp_y_temp = exp_temp[-out_len:]
313                 exp_y_temp = exp_y_temp[:, -1] #keeping just the
     percentage of SASA
314                 exp_y_temp=exp_y_temp.reshape(sim_size, out_len,
     1)
315                 exp_y_temp=np.round(exp_y_temp, 2)
316                 exp_y= np.concatenate((exp_y, exp_y_temp), axis
    = 0)
317                 dist_temp=temp['c']
318                 dist_temp=dist_temp[0::interval] #subsampling
    taking just every interval-th matrix,
319                 dist_x_temp=dist_temp[:sim_len_f]
```

```
320                     dist_x_temp=dist_x_temp.reshape(sim_size,
    sim_len_f, dist_len)
321                     dist_x= np.concatenate((dist_x, dist_x_temp),
    axis = 0)
322 
323             if test_t: ###just a small function to list all the test
     files used to be compared later
324                 with open("test_files_list_in_gen.csv", 'w', newline
    ='') as myfile:
325                     wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
326                     wr.writerow(test_t_filenames)
327             latest_gpu_memory = gpu_memory_usage(gpu_id)
328             latest_memory_total = gpu_memory_total(gpu_id)
329             print("(GPU %:", 100*(latest_gpu_memory -
    initial_memory_usage)/latest_memory_total)
330             exp_x=exp_x[:,:,:3] #take just last 3 columns of
    exposure!
331             yield ((atom_x, exp_x, dist_x, drug_static), exp_y)
332             i = i + 1
333 print("preDGW1")
334 check_data = tf_data_generator(files, sim_len_f,test_t, batch_size =
     1)
335 num = 0
336 print("postDGW1")
337 arr_len=0
338 arr_len_y=0
339 drug_st_len=0
340 fprint("arr_len,",arr_len)
341 
342 #to get the parameters for the dimensions of model input
343 for (atom_x, exp_x, dist_x, drug_static),exp_y in check_data:
344     fprint("check_data:atom_x, exp_x, dist_x, exp_y, drug_static,",
    atom_x.shape, exp_x.shape, dist_x.shape, exp_y.shape, drug_static
    .shape)
345     arr_len=exp_x.shape[-1]
346     arr_len_y=exp_y.shape[-1]
347     arr_len_dist=dist_x.shape[-1]
348     drug_st_len=drug_static.shape[-1]
349     print()
350     num = num + 1
351     if num > 5: break
352 
353 
354 #spliting the data into train, and val
355 from sklearn.model_selection import train_test_split
356 train, test = train_test_split(files, test_size = int(len(files)
    *0.05), random_state = 54321) #check the test size etc, I took
    10% of all!
357 train, val = train_test_split(files, test_size = int(len(files)
    *0.05), random_state = 12345)#check the val size etc, I took 10%
    of all!
358 
359 random.shuffle(train)
360 
361 
362 
```

```
363 fprint("Number of train_files:," ,len(train))
364 fprint("Number of validation_files:," ,len(val))
365 fprint("Number of test_files:," ,len(test))
366
367 batch_size = 1
368
369
370 print("===> Num GPUs Available: ", len(tf.config.
    list_physical_devices('GPU')))
371
372 if 1:
373
374     train_dataset = tf.data.Dataset.from_generator(tf_data_generator
    ,args= [train,sim_len_f,test_t, batch_size],output_types = ((tf.
    float32, tf.float32,tf.float32,tf.float32),tf.float32),
375                                                    output_shapes =
    (((None,sim_len_f, nsize1, 60, 60, 1),(None,sim_len_f, 3),(None,
    sim_len_f, 42),(None,sim_len_f, 7)),(None,out_len, 1)))
376     validation_dataset = tf.data.Dataset.from_generator(
    tf_data_generator,args= [val,sim_len_f,test_t, batch_size],
    output_types = ((tf.float32, tf.float32,tf.float32,tf.float32),tf
    .float32),
377                                                    output_shapes =
    (((None,sim_len_f, nsize1, 60, 60, 1),(None,sim_len_f, 3),(None,
    sim_len_f, 42),(None,sim_len_f, 7)),(None,out_len, 1)))
378     test_t=1
379     test_dataset = tf.data.Dataset.from_generator(tf_data_generator,
    args= [test,sim_len_f,test_t, batch_size],output_types = ((tf.
    float32, tf.float32,tf.float32,tf.float32),tf.float32),
380                                                    output_shapes =
    (((None,sim_len_f, nsize1, 60, 60, 1),(None,sim_len_f, 3),(None,
    sim_len_f, 42),(None,sim_len_f, 7)),(None,out_len, 1)))
381
382     #FOR CHECKING PURPOSES SOME PRINTOUTS!
383     fprint("TEST DATASET CHECK:,")
384     for (atom_x, exp_x, dist_x, drug_static), exp_y in test_dataset.
    take(2):
385         fprint("atom_x.shape ,",atom_x.shape)
386         fprint('type(atom_x) ,',type(atom_x))
387         fprint('exp_x.shape ,',exp_x.shape)
388         fprint('exp_x ,',exp_x)
389         fprint('type(exp_x) ,',type(exp_x))
390         fprint("dist_x.shape,",dist_x.shape)
391         fprint('dist_x ,',dist_x)
392         fprint('type(dist_x ,',type(dist_x))
393
394         fprint('drug_static.shape ,',drug_static.shape)
395         fprint('type(drug_static) ,',type(drug_static))
396         fprint('exp_y.shape ,',exp_y.shape)
397         fprint('type(exp_y) ,',type(exp_y))
398         fprint('exp_y ,',exp_y)
399
400         steps_per_epoch = np.int(np.ceil(len(train)/batch_size))
401         validation_steps = np.int(np.ceil(len(val)/batch_size))
402         steps = np.int(np.ceil(len(test)/batch_size))
403         fprint("steps_per_epoch = ,", steps_per_epoch)
```

```
404         fprint("validation_steps = ,", validation_steps)
405         print("validation_steps = ,", validation_steps)
406         fprint("steps = ,", steps)
407
408
409
410     window_length = out_len
411
412     cnn_droupout=0.5
413 #model architecture
414     cnn_i = Input(shape=(sim_len_f, nsize1, nsize, nsize,1),name='
    Input_3D') #try without this
415     fprint("########################################INPUT SHAPE:,")
416     fprint("cnn_i = Input(shape=(sim_len_f,",sim_len_f,", nsize1, ",
    nsize1,",nsize,",nsize,", nsize,",nsize,",1))")
417     cnn = TimeDistributed(Conv3D(32,kernel_size=(kernel_size,
    kernel_size, kernel_size), activation='relu', padding='same',
    kernel_initializer='he_uniform'),name='Conv3D_1')(cnn_i) #
    initially Conv3D(64, lowered to 32 to have less variables, less
    needed mempry
418     cnn = TimeDistributed(MaxPooling3D(pool_size=(2, 2, 2)),name='
    MaxPooling3D_1')(cnn)
419     cnn = TimeDistributed(BatchNormalization(center=True, scale=True
    ),name='BatchNormalization_1')(cnn)
420     cnn = TimeDistributed(Dropout(cnn_droupout),name='Dropout_1')(
    cnn)
421     cnn = TimeDistributed(Conv3D(64, kernel_size=(kernel_size,
    kernel_size, kernel_size), activation='relu', padding='same',
    kernel_initializer='he_uniform'),name='Conv3D_2')(cnn) #initially
     Conv3D(64, lowered to 32 to have less variables, less needed
    mempry
422     cnn = TimeDistributed(MaxPooling3D(pool_size=(2, 2, 2)),name='
    MaxPooling3D_2')(cnn)
423     cnn = TimeDistributed(BatchNormalization(center=True, scale=True
    ),name='BatchNormalization_2')(cnn)
424     cnn = TimeDistributed(Dropout(cnn_droupout),name='Dropout_2')(
    cnn)
425     cnn = TimeDistributed(Conv3D(128, kernel_size=(kernel_size,
    kernel_size, kernel_size), activation='relu', padding='same',
    kernel_initializer='he_uniform'),name='Conv3D_3')(cnn) #initially
     Conv3D(64, lowered to 32 to have less variables, less needed
    mempry
426     cnn = TimeDistributed(MaxPooling3D(pool_size=(2, 2, 2)),name='
    MaxPooling3D_3')(cnn)
427     cnn = TimeDistributed(BatchNormalization(center=True, scale=True
    ),name='BatchNormalization_3')(cnn)
428     cnn = TimeDistributed(Dropout(cnn_droupout),name='Dropout_3')(
    cnn)
429     cnn = TimeDistributed(Conv3D(256, kernel_size=(kernel_size,
    kernel_size, kernel_size), activation='relu', padding='same',
    kernel_initializer='he_uniform'),name='Conv3D_4')(cnn) #initially
     Conv3D(64, lowered to 32 to have less variables, less needed
    mempry
430     cnn = TimeDistributed(MaxPooling3D(pool_size=(2, 2, 2)),name='
    MaxPooling3D_4')(cnn)
431     cnn = TimeDistributed(BatchNormalization(center=True, scale=True
```

```
    ),name='BatchNormalization_4')(cnn)
432  cnn = TimeDistributed(Dropout(cnn_droupout),name='Dropout_4')(
    cnn)
433  cnn = TimeDistributed(Conv3D(512, kernel_size=(kernel_size,
    kernel_size, kernel_size), activation='relu', padding='same',
    kernel_initializer='he_uniform'),name='Conv3D_5')(cnn) #initially
     Conv3D(64, lowered to 32 to have less variables, less needed
    mempry
434  cnn = TimeDistributed(MaxPooling3D(pool_size=(2, 2, 2)),name='
    MaxPooling3D_5')(cnn)
435  cnn = TimeDistributed(BatchNormalization(center=True, scale=True
    ),name='BatchNormalization_5')(cnn)
436  cnn = TimeDistributed(Dropout(cnn_droupout),name='Dropout_5')(
    cnn)
437  cnn = TimeDistributed(Flatten(),name='Flatten_3D')(cnn)
438  cnn = TimeDistributed(Dense(cnn_fin_dense_size,activation='relu'
    ),name='Dense_3D')(cnn)
439
440  exposure_i = Input(shape=(sim_len_f,  arr_len),name='Input_SASA'
    )
441  exposure = TimeDistributed(Dense(arr_len, activation='sigmoid',
    input_shape=(sim_len_f, arr_len)),name='Dense_SASA')(exposure_i)
442
443
444  distance_i = Input(shape=(sim_len_f,  arr_len_dist),name='
    Input_Distances')
445  distance = TimeDistributed(Dense(arr_len_dist, activation='
    sigmoid', input_shape=(sim_len_f, arr_len_dist)),name='
    Dense_Distances')(distance_i)
446
447
448  drug_i = Input(shape=(sim_len_f,  drug_st_len),name='
    Input_DrugType')
449  drug = TimeDistributed(Dense(arr_len, activation='sigmoid',
    input_shape=(sim_len_f, drug_st_len)),name='Dense_DrugType')(
    drug_i)
450
451  n_neurons = n_neurons#128#1024#128#512 #check
452
453  #some formula to set the size of batches in regard to the input
    size, if input is fed in small windows and the matrix size small
    then there should be largers batches
454  n_batch = batch_size#int(300/((size/(1+quad*7)+sim_len_f)))
455  if(n_batch < 1):
456      n_batch = 1
457  print("
    #######|||||||||||#####|||||#####|||||#####|||||#####|||||||||||||||||||#########
    n_batchn_batchn_batch",n_batch)
458
459  merged1 = Concatenate(name='Concatenate_3D-SASA')([cnn, exposure
    ])
460  merged2 = Concatenate(name='Concatenate_3DSASA-Distances')([
    merged1, distance])
461  merged3 = Concatenate(name='Concatenate_3DSASADistances-DrugType
    ')([merged2,drug]) #maybe it works to concatenate all in one
    level like merged1 = Concatenate()([cnn, exposure,drug])
```

```
462     merged = LSTM(n_neurons, return_sequences=True,name='LSTM')(
        merged3) #return_sequences=False would return just last timestep?
463     merged = Dense(n_neurons,activation='relu',name='
        Dense_afterLSTM_1')(merged)
464     merged = Dense(int(n_neurons/2),activation='relu',name='
        Dense_afterLSTM_2')(merged)
465     merged = TimeDistributed(Dense(arr_len_y,activation='relu',
        dtype=tf.float32),name='Dense_afterLSTM')(merged)
466
467     model = Model(inputs=[cnn_i, exposure_i,distance_i, drug_i],
        outputs=merged)  #POTENTIAL PROBLEM, MAYBE DRUG CONCATENATION
        NEEDS TO BE DONE BEFORE!
468     model.compile(loss=str(loss_funct), optimizer='adam', metrics=[
        tensorflow.keras.metrics.MeanAbsoluteError()]) #optimizer adjust
        the learning rate, uses scheduled learning rate
469
470     fprint(model.summary())
471
472     # Open the file
473     with open(path_new_folder+'/' + 'model_summary'+
        unique_about_this_run+'.txt','w') as fh:
474         # Pass the file handle in as a lambda function to make it
        callable
475         model.summary(print_fn=lambda x: fh.write(x + '\n'))
476
477     utils.plot_model(model, path_new_folder+"
        multi_input_and_output_model_mi01"+"_"+str(size)+"_"+str(interval
        )+"_"+str(quad)+"_"+str(sim_len_f)+str(kernel_size)+"_"+
        loss_funct+"_"+str(average)+unique_about_this_run+".png",
        show_shapes=True)
478
479     utils.plot_model(model, path_new_folder+"TB"+"_"+str(size)+"_"+
        str(interval)+"_"+str(quad)+"_"+str(sim_len_f)+str(kernel_size)+"
        _"+loss_funct+"_"+str(average)+unique_about_this_run+".png",
        show_shapes=False, show_dtype=False,show_layer_names=False,
        rankdir='TB', expand_nested=False, dpi=300)
480     utils.plot_model(model, path_new_folder+"TBnames"+"_"+str(size)+
        "_"+str(interval)+"_"+str(quad)+"_"+str(sim_len_f)+str(
        kernel_size)+"_"+loss_funct+"_"+str(average)+
        unique_about_this_run+".png", show_shapes=False, show_dtype=False
        ,show_layer_names=True, rankdir='TB', expand_nested=False, dpi
        =300)
481     utils.plot_model(model, path_new_folder+"TBdtype"+"_"+str(size)+
        "_"+str(interval)+"_"+str(quad)+"_"+str(sim_len_f)+str(
        kernel_size)+"_"+loss_funct+"_"+str(average)+
        unique_about_this_run+".png", show_shapes=False, show_dtype=True,
        show_layer_names=False, rankdir='TB', expand_nested=False, dpi
        =300)
482
483
484     latest_gpu_memory = gpu_memory_usage(gpu_id)
485     print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
        )#####(GPU) Memory used:", latest_gpu_memory -
        initial_memory_usage)
486     latest_memory_total = gpu_memory_total(gpu_id)
487     print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
```

```
)#####(GPU) memory_total :", latest_memory_total)
488  print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
)#####(GPU) Memory used of total % :", 100*(latest_gpu_memory -
initial_memory_usage)/latest_memory_total)
489
490
491  ######## to time the epochs
492  class timecallback(tf.keras.callbacks.Callback):
493      def __init__(self):
494          self.times = []
495          # use this value as reference to calculate cummulative
time taken
496          self.timetaken = time.process_time()
497      def on_epoch_end(self,epoch,logs = {}):
498          self.times.append((epoch,time.process_time() - self.
timetaken))
499      def on_train_end(self,logs = {}):
500          fig = plt.figure()
501          plt.xlabel('Epoch')
502          plt.ylabel('Total time taken until an epoch in seconds')
503          plt.plot(*zip(*self.times))
504          fig.savefig('epochs.png')
505  timetaken = timecallback()
506
507  ###to be able to plot loss per batch!
508  class LossHistory(tf.keras.callbacks.Callback):
509      def on_train_begin(self, logs={}):
510          self.history = {'loss':[],'val_loss':[],'
mean_absolute_error':[],'val_mean_absolute_error':[]}
511
512      def on_batch_end(self, batch, logs={}):
513          self.history['loss'].append(logs.get('loss'))
514          self.history['mean_absolute_error'].append(logs.get('
mean_absolute_error'))
515          self.history['val_loss'].append(logs.get('val_loss'))
516          self.history['val_mean_absolute_error'].append(logs.get(
'val_mean_absolute_error'))
517
518  history = LossHistory()
519
520
521  csv_logger = CSVLogger(path_new_folder+'/'+'training_'+str(
sim_len_f)+"_"+typeinp+str(kernel_size)+"_"+loss_funct+"_" +str(
average)+"_"+ datetime.now().strftime("%Y%m%d-%H%M%S")+
unique_about_this_run+'.log')
522
523  n_epoch=num_epochs
524
525  checkpoint_path = "training_1_cp/cp29_"+unique_about_this_run+"_
"+str(sim_len_f)+"_"+typeinp+"_"+str(kernel_size)+"_"+loss_funct+
"_"+".ckpt"
526  model_path="model_"+unique_about_this_run+".h5"
527
528  checkpoint_dir = os.path.dirname(checkpoint_path)
529
530  # Create a callback that saves the model's weights
```

```
531     cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=
    checkpoint_path,monitor='val_loss', save_best_only=True,
    save_weights_only=True,verbose=1)
532
533     #TENSORBOARD
534     logfold=os.getcwd()+'/logs'
535     print(logfold)
536
537
538     log_dir = "logs/fit/"+str(sim_len_f)+"_"+typeinp+"_"+str(
    kernel_size)+"_"+loss_funct+"_"+str(average)+"_"+datetime.now().
    strftime("%Y%m%d-%H%M%S")
539     tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=
    log_dir, histogram_freq=1)
540
541     if load_weights:
542         model.load_weights(checkpoint_path)
543
544     if fit_model:
545         model.fit(train_dataset, validation_data =
    validation_dataset, steps_per_epoch = steps_per_epoch,
    validation_steps = validation_steps,callbacks=[csv_logger,
    tensorboard_callback,history],epochs = num_epochs)
546
547
548
549     # Calling `save('my_model')` creates a SavedModel folder `
    my_model`.
550     model.save("my_model"+str(sim_len_f)+"_"+str(width)+"_"+"_"+
    typeinp+"_"+str(kernel_size)+"_"+loss_funct)
551
552     latest_gpu_memory = gpu_memory_usage(gpu_id)
553     fprint("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)### Memory used:,"
    , latest_gpu_memory - initial_memory_usage)
554     latest_memory_total = gpu_memory_total(gpu_id)
555     fprint("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)### memory_total
    :,", latest_memory_total)
556     fprint("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)### Memory used of
     total % :,", 100*(latest_gpu_memory - initial_memory_usage)/
    latest_memory_total)
557
558     test_loss, test_accuracy = model.evaluate(test_dataset, steps =
    20)
559
560     steps = np.int(np.ceil(len(test)/batch_size))
561     predictions = model.predict(test_dataset,steps = steps)
562     fprint("type(prediction),",type(predictions))
563     fprint("prediction.shape,", predictions.shape)
564     i=0
565     n_test_sims=len(test)
566     numtest=predictions.shape[0]
567     pred_strt=0
568     pred_end=int(numtest/n_test_sims)
569     fprint("Number of test_files:," ,len(test))
570     fprint("test files:," ,test)
571
```

```
572     test_t_filenames_out=[]
573     for testfile in test:
574         testfile=str(testfile)
575         testfile_sh=testfile[(testfile.rfind("/")+2):testfile.rfind(
    ".npz")]
576         test_t_filenames_out.append(testfile_sh)
577 
578 
579     with open("test_files_list_outside.csv", 'w', newline='') as
    myfile2:
580         wr = csv.writer(myfile2, quoting=csv.QUOTE_ALL)
581         wr.writerow(test_t_filenames_out)
582     fprint("PREDICTIONS: ,", type(predictions))
583     fprint("PREDICTIONS: ,", predictions.shape)
584     fprint("predictions,", predictions[0])
585     fprint("predictions,", predictions[0][0])
586     for i in range(n_test_sims):
587         fprint(" print(i),predictions[i][0]),",i,predictions[i][0])
588         np.savetxt(path_new_folder+test_t_filenames[i]+"_"+str(i)+"_
    "+str(size)+"_"+str(interval)+"_"+str(quad)+"_"+str(sim_len_f)+"_
    "+str(out_len)+".csv", predictions[i], delimiter=",")
589         pred_strt=pred_end
590         pred_end=pred_end+int(numtest/n_test_sims)
591 
592     recursive_chmod(path_new_folder, 0o777) #
593     time_elapsed = datetime.now() - start_time
594     fprint('Time elapsed (hh:mm:ss.ms), {}'.format(time_elapsed))
595 
596     latest_gpu_memory = gpu_memory_usage(gpu_id)
597     print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
    )#####(GPU) Memory used:", latest_gpu_memory -
    initial_memory_usage)
598     latest_memory_total = gpu_memory_total(gpu_id)
599     print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
    )#####(GPU) memory_total :", latest_memory_total)
600     print("##(GPU)#(GPU)#(GPU)##(GPU)#(GPU)#(GPU)###(GPU)#(GPU)#(GPU
    )#####(GPU) Memory used of total % :", 100*(latest_gpu_memory -
    initial_memory_usage)/latest_memory_total)
601     model.save(model_path)
```

Listing 7: Example Python - Tensorflow script to train and use the 3D CNN LSTM model to predict the SASA exposure as shown in Results.