Master Thesis

Neural networks for improving drug discovery efficiency

Author(s):
Hassan Harrirou, Hussein

Publication Date:
2019-03

Permanent Link:
https://doi.org/10.3929/ethz-b-000337776

Rights / License:
In Copyright - Non-Commercial Use Permitted
Neural networks for improving drug discovery efficiency

Master Thesis
Hussein Hassan Harrirou
Sunday 31st March, 2019

Supervisor: Dr. Thomas Lemmin, Prof. Ce Zhang
Department of Computer Science, ETH Zürich
Acknowledgements

I would like to thank everyone who helped and supported me through the last semester of my Master’s degree.

I want to express my utmost gratitude to my parents, Ala and Najat, without whom I could never have gotten the possibility to do my Master’s degree abroad and join ETH Zurich, and who gave me the unconditional love and support that I needed while I studied abroad.

I want to show my gratitude to my supervisor Thomas Lemmin, for showing great patience and constant support by filling my gaps in knowledge about biochemistry and how to work systematically on my experiments. Many thanks to Prof. Ce Zhang for his ideas and help figuring out my mistakes during the development of the neural networks, and for helping me realize that I do not know as much about machine learning as I thought I did. Thanks to all the members of the department who helped me by sharing their computational resources, without your help I couldn’t have made it to the deadline.

I would like to thank my friends, who listened to my incessant complaints and helped me stay sane for the last six months. Special thanks: to David Martínez Rubio, Jeniffer Zhou and Nikolai Nikolov for their ideas on how to improve and fix the neural network architectures; to Asuman Inan, Xinyue Yao, and Eric García de Ceca who helped proofreading and correcting this work, and supported me throughout the thesis; to Asuman Inan, Eric García de Ceca, Xinyue Yao, David Martínez Rubio, Jay Li, Kenichi Furuya, Rikushi Yasumatsu, Georgiana Birjovanu, Xi Chen, Renjie Cui, Yaqi Zhao, Qiao Tang, Risako Miyake and Benedikt Petko for their incredible support during the hardships of this last half year.
Abstract

The increase in costs and needs of new drugs has led to the wide use of computational methods for drug discovery. Among the different methods, Machine-learning based methods have demonstrated to have a good efficiency/accuracy trade-off. Here we explore how to improve the binding affinity prediction, a measurement for drug binding effectiveness. To do so, we base our development on the previous Convolutional Neural Network state-of-the-art approach, KDeep, and research how to improve the neural network architecture and also use different architectures, such as ResNet. We also explore the performance of more physically meaningful energy-based features, derived from the Rosetta force field, APBS electrostatics, and electronegativity maps. We examine how these features can be represented in 3D space with different types of filters and interpolations, and compare their performance. Finally, we perform data manipulation and augmentation to improve the generalization capabilities of the neural networks by applying local relaxation of the protein-ligand structures and increasing our dataset tenfold with molecular dynamics simulations. We have shown that energy-based features are able to represent the information needed for good pK binding affinity prediction, achieving 1.36 RMSE on PDBBind core set (v.2016) for Rosetta and electronegativity features. When combined with HTMD features, we lowered the RMSE down to 1.32. In addition, we achieved better predictions for the test datasets of CSAR HiQ and CSAR 2014 and overall better generalization, with very similar Pearson’s correlation coefficients for the validation and test datasets. Finally, we have produced an extensive dataset of more than 48 000 poses from the 4463 complexes offered in PDBBind v.2018, which can be used for further studies.

Keywords: Binding affinity, dissociation constant (K_d), inhibition constant (K_i), Convolutional Neural Networks, Rosetta, ResNet, Molecular Dynamics, PDBBind, Drug Discovery
1 Introduction
1.1 Motivation ............................................. 1
1.2 Background Knowledge ................................. 2
  1.2.1 Binding affinity .................................... 2
  1.2.2 Convolutional Neural Network architectures .... 3
1.3 Previous results ....................................... 4
1.4 Objective .............................................. 5

2 Materials .................................................. 7
2.1 Datasets ................................................ 7

3 Methods ................................................... 9
3.1 Features ............................................... 9
  3.1.1 Molecular descriptors ............................... 10
  3.1.2 Electrostatic charges ............................... 10
  3.1.3 Force-field energies ............................... 12
  3.1.4 Atom identification ............................... 14
3.2 3D representation .................................... 14
3.3 Deep Learning ....................................... 15
3.4 Pipeline ............................................... 18
  3.4.1 Getting the input data ............................ 19
  3.4.2 Relaxing the ligands into the protein .......... 19
  3.4.3 Autodock Vina formatted files ................. 20
  3.4.4 Preprocessing Rosetta pointwise energies .... 20
  3.4.5 Computing features ............................ 20
  3.4.6 Merging the features ............................ 21
  3.4.7 TFRecords for neural network input .......... 21
  3.4.8 Augmenting the data - Molecular Dynamics simulation 22
## Contents

4 Results .................................................. 23
  4.1 Prediction results ................................ 23
    4.1.1 Original KDeep - HTMD features .......... 24
    4.1.2 Larger filter KDeep ......................... 26
    4.1.3 ResNet-101 .................................. 28
    4.1.4 Comparison of results with other networks 31
  4.2 Representation results ......................... 31
  4.3 Data augmentation results ..................... 34

5 Discussion ....................................... 39

6 Conclusions .................................... 43

Bibliography ..................................... 45

Appendices ....................................... 51

A Dataset complexes .............................. 53
  A.1 Training data: PDBBind 2018 Refined set .... 53
  A.2 Validation data: PDBBind 2018 Core set .... 58
  A.3 Test set: CSAR HiQ .......................... 58
    A.3.1 Set 1 .................................. 58
    A.3.2 Set 2 .................................. 58
  A.4 Test set: CSAR 2014 ........................ 58

B Core set clusters ............................... 59

C Pipeline scripts ................................. 61
  C.1 Rosetta relaxation ............................ 61
  C.2 Precompute Rosetta energies ................. 68
  C.3 CNN Architectures ............................ 71
  C.4 Feature generation ............................ 75
  C.5 PDBQT generation ............................ 92
  C.6 Molecular Dynamics simulation .............. 95
Chapter 1

Introduction

1.1 Motivation

In 1928, the antibiotic penicillin was discovered by Alexander Fleming[1]. Shortly after the spread and mass production of the drug, the first case of penicillin-resistant *Staphylococcus aureus* was found[8]. Since then, the use of antibiotics has sky-rocketed as well as the cases of multi-drug resistant bacteria[27]. This continued trend calls for the constant development of new drugs effective against previously treatable bacteria.

The drug development industry by pharmaceutical companies is one of the costliest. The investment required to develop a new drug was estimated to be of $1.4 billion on R&D alone[12].

The growing necessity for new drugs and the increasing costs for their research and development motivate the use of mathematical and computational methods that could narrow down the search space. The drug search space is very large, with an estimate of more than $10^6$[14] possible protein-ligand complexes. The Protein Database (PDB) is a database for proteins, nucleic acids, and other biological molecular 3D structures and their information. In it, there are already more than 144,000 entries, and in recent years around 10,000 more are added every year[45]. It is clear then that using neither experimental testing nor time-intensive computational methods are reasonable approaches to identify well performing combinations of proteins and drugs.

Computer-aided drug discovery has been approached as a set of sub-problems. Virtual Screening (VS) consists in searching large data-sets of small molecules for potential ligands that would properly bind to the target protein. One of the search methods is molecular docking, in which a ligand is spatially manipulated in order to find the binding conformation to the protein with the lowest energy[28]. Binding Affinity Prediction can then be used to com-
1. Introduction

 pare and rank the binding strength between the target protein and potential drugs by using several constants derived from molecular mechanics.

Binding affinity can be predicted with a few different methods. The more precise techniques relying on Quantum Mechanics or Thermodynamic integration are computationally expensive, which limits their large-scale usability. Machine-learning approaches offer a faster alternative; however, their accuracy strongly depends on the adequate selection of features and models.

In this project, we investigated the use of Deep Learning to improve the binding affinity prediction of protein-ligand complexes. The current state-of-the-art system, KDeep, uses a group of features classified as molecular descriptors, indicator functions of certain subsets of atoms with certain properties. We explore the performance of energy-based features such as the ones from all-atom energy functions like Rosetta or electrostatic energies based on the Poisson-Boltzmann equations. We also compare different filters and interpolations to represent these features in a 3D grid. We explore the performance of two different convolutional neural network architectures: KDeep (a variation of SqueezeNet) and ResNet. Finally, we explore the change of performance when augmenting our dataset by introducing changes on the proteins’ structures via molecular dynamics simulations.

1.2 Background Knowledge

In this section we explain two concepts required to understand the development of the project. First, we glance through the definition of binding affinity and its chemical significance. Secondly, we explain the building blocks of the neural network architectures used later in this work, KDeep (with its underlying SqueezeNet architecture) and ResNet.

1.2.1 Binding affinity

Binding affinity represents how well a protein and a ligand bind when combined in a solution. This binding can be represented by the following reaction, where E is the protein, S is the ligand and I is a potential inhibitor.

\[
E + S \xrightleftharpoons[k_f]{k_i} E \cdot S
\]

\[
E + I \xrightleftharpoons[k_{fi}]{k_{ri}} E \cdot I
\]

Binding affinity can be understood as the difference of energy between the unbound and bound complex. This is expressed by the Gibbs free energy at standard conditions, \(\delta G^o\).
1.2. Background Knowledge

Binding affinity is also commonly measured by other values related to $\Delta G^o$, such as the dissociation constant ($K_D$), the inhibition constant ($K_I$) or the half maximal inhibitory concentration ($IC_{50}$).

$$K_D = [E][S]/[ES] = exp(\Delta G^o/RT)$$

$$K_I = [E][I]/[EI]$$

$$IC_{50} = K_I(1 + [S]/K_M) + [E]_0/2$$

In many binding affinity databases only one of them appears per complex, and regardless of their differences, many prediction approaches will consider them as a generic binding affinity constant.

1.2.2 Convolutional Neural Network architectures

In this project we employ two Convolutional Neural Networks (CNN), SqueezeNet and ResNet, to predict the binding affinity.

SqueezeNet was designed to be a compact version of the AlexNet architecture[30]. Both AlexNet and ResNet were originally developed to classify natural images from the ImageNet[11] dataset, but have been widely used in many classification and regression problems in many fields, including problems with 3D images such as 3D pose prediction[36] and MRI detection[29].

Both of these architectures are deep convolutional neural networks designed by sequentially composing different modules. SqueezeNet defines two-layered modules called “fire modules”. The first layer of the module is the squeeze layer, a convolutional layer with filter size 1x1 and small number of filters. This squeeze layer is supposed to constrain the network to a more compact representation in order to find the features that are most important and reject the noisier ones. The second layer is the expand layer, composed by two side-by-side convolutional layers, one of shape 1x1 and the other with shape 3x3, with a larger channel count than the squeeze layer. This layer is dedicated to performing transformations over the “squeezed” representation. These side-by-side convolutions are then concatenated to form an output with a much larger channel count. Each fire module maintains the dimensions of the images, and either maintains or increases the number of channels. The overall network is composed by groups of fire modules, with max pooling layers between groups to reduce the image dimensionality after each block. Finally, the output of the last block is averaged and passed through a fully connected layer. A graphic representation of the fire module is shown in Figure 1.1.

ResNet follows a similar idea with its “residual modules”. In ResNet’s case, the module has three layers, with filter shapes 1x1, 3x3 and 1x1 respectively. Residual modules also include skip connections, following the idea that the
1. Introduction

SqueezeNet’s “Fire module”

Figure 1.1: SqueezeNet’s “Fire module”

ResNet’s “Residual module”

Figure 1.2: ResNet’s “Residual module”

modules shouldn’t learn all attributes of the data, but only the perturbances between input and output. Residual modules, in the same way as fire modules, do not reduce the size of the images, and ResNet composes them in blocks separated by max pooling layers. There are also skip connections between contiguous block, skipping the separating max pooling layers. A graphic representation of a residual module is shown in Figure 1.2.

1.3 Previous results

Machine-learning approaches to binding affinity prediction have become very popular in recent years. Although their accuracy is still not comparable to more costlier methods, such as the ones based on simulations[18], the size of the drug search space makes speed outweigh precision as a bottleneck.
1.4 Objective

A plentiful variety of classical machine-learning algorithms have been applied, from linear regressions\cite{31, 46}, kernel ridge regression\cite{44, 40}, support vector machines\cite{41, 46}, Gaussian processes\cite{31} to random forests\cite{41, 34, 44}. In this regard, model choice seems to favor random forests, with the best performing example of RF-Score\cite{34}.

Until recently, the best performing machine-learning approach was that of the random forest RF-Score, which relies on 42 molecular descriptors extracted from Autodock Vina\cite{43}. RF-Score reported an RMSE of 1.513 for $pK_D$ prediction on their test set (PDBBind 2007 core set), whereas \cite{24} reports an RMSE of 1.39 after training with PDBBind\cite{35} 2016 refined minus core set, and testing with the core set.

On the other hand, neural networks and specifically deep learning, has shown appearance and promising results that improve on RF-Score’s results. Various examples are KDeep\cite{24}, TopologyNet\cite{6}, DLScore\cite{20}.

KDeep, the network we base our project in, focuses on generating 3D maps of different molecular descriptors of the atoms of both protein and ligand, in a bounding box of 24 Angstrom centered at the ligand’s center of mass. These resulting 16 maps are then input for a simplified SqueezeNet. KDeep reports an RMSE on the PDBBind 2016 core set of 1.27 for $pK_D$ prediction.

TopologyNet relies on computing topologically invariant features (i.e. Betty numbers) of the graph of heavy atoms and bonds at different bonding distances and computing so called topological barcodes, which are inputted into a 1D-convolutional neural network with additional geometrical data. TopologyNet reported an RMSE of 1.37 for $pK_D$ prediction when training with PDBBind 2016 refined minus core set and testing with the core set.

DLScore counts the closeness between atoms from the protein and the ligand in buckets depending on the atom types. It also calculates the electrostatic energy of the residues. Finally, all these features are added to the Vina features and inputted to a deep neural network to predict the result. DLScore reports an RMSE of 1.15 for $\Delta G^0$ prediction, when trained with PDBBind 2016 refined minus core set and tested with the core set.

1.4 Objective

In this project, we tried to answer the following questions:

1. How can we incorporate physically meaningful features like electrostatics from APBS\cite{25} and force-field from Rosetta\cite{4}?

2. How should these features be represented in 3D, and how do new feature representations perform compared to the new proposals?
1. **Introduction**

3. Can we find a neural network architecture that performs better than KDeep?

4. Does data augmentation via molecular dynamics simulations help the neural network to learn to predict better?
Chapter 2

Materials

2.1 Datasets

In this section we introduce the datasets used to train, validate and test the machine learning models.

For training and validation, the PDBBind dataset[35] is used. More concretely, we use the “refined set” subset for the dataset version released in 2018. The “PDBBind 2018 general set” version contains 19588 biomolecular structures, for which some binding affinity data (i.e. $K_d$, $K_i$, $IC_{50}$) has been experimentally recorded. Furthermore, the “refined set” contains 4463 selected complexes after applying certain curation filters. Finally, a 290 complex subset of the “refined set” called “core set” is used for validation. The “core set” was constructed by clustering the complexes of the “refined set” into 58 clusters by a 90% similarity cutoff of the protein sequence and taking 5 examples for each cluster, in order to evenly represent the diversity of the complexes. In our case, due to changes in PDBBind v.2018, only 271 complexes from the “core set” are currently classified as part of the “refined set”. 

For testing, we used the 2010 CSAR-HiQ[15] and 2014 CSAR[7] datasets. These datasets are filtered to contain only the complexes not already considered during training and validation. The former is given as two separate sets, CSAR-HiQ Set 1 and CSAR-HiQ Set 2. After filtering, CSAR-HiQ Set 1 contains 55 complexes, CSAR-HiQ Set 2 contains 49 complexes, and CSAR 2014 contains 47 complexes.

In the end, the training dataset consists of 4192 complexes, the validation dataset of 271 complexes, and the test datasets of 201 complexes. A list of these complexes appears in Appendix A

It is important to note that this division between training, validation and testing is common but not unique, and other authors have tested other
2. Materials

approaches\cite{44}, such as a time based split (in which the validation and testing subsets are formed by the newly acquired experimental results in order to simulate the real discovery procedure), structure based (separate the dataset according to substructure similarity) or stratified random sampling (a split that ensures that in each of the subsets the full range of values is represented).
Chapter 3

Methods

To improve the predictive behavior of the current applied methods, the following approaches were considered:

- Identifying features that describe well the concepts related to binding affinity.
- Finding a 3D representation for those features, to be used with convolutional neural networks.
- Modifying the neural network architecture to improve generalization.

3.1 Features

In the field of biochemistry, proteins and ligands are commonly given in a few raw representations, such as single-line strings indicating atoms, molecular motifs and bonds (i.e. SMILES or FASTA), or full 3D representations of atom coordinates (i.e. PDB), with bonds (i.e. MOL2).

These representations fail to satisfy one of the preferable properties of inputs to most machine-learning algorithms: a fixed-size structure. Furthermore, it is unclear if directly using these raw features is enough to represent any attribute interesting for our purpose of predicting binding affinity.

Thus, a great deal of research has focused on obtaining advanced representations developed from the previously mentioned raw ones. Overall, three types of attributes seem to dominate the representation landscape: topological features, based on the atoms and their bonds; energetic features, based on the charges and energies of the atoms and their bonds; and statistical features of both raw data and the previous features.

Examples of different uses of statistical features of atom types and small substructures [31 46], physicochemical markers [31 41 46], and many energy features such as Coulomb matrices [40], and Gasteiger charges [24].
3. Methods

In this project, we have focused on three different feature sets:

- Molecular descriptors
- Electrostatic energies
- Force-field energies

### 3.1.1 Molecular descriptors

Molecular descriptors indicate, as binary maps, certain properties of the atoms in a molecule or protein.

Following KDeep’s procedure, we used the molecular descriptors offered by the Python library HTMD, which are defined in terms of the atom types of the software Autodock Vina. In it, eight different descriptors are defined, as shown in Table 3.1.

<table>
<thead>
<tr>
<th>Channels</th>
<th>Atoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hydrophobic</td>
<td>Aliphatic or aromatic carbons</td>
</tr>
<tr>
<td>Aromatic</td>
<td>Aromatic carbons</td>
</tr>
<tr>
<td>Hydrogen bond acceptor</td>
<td>Nitrogen, oxygen and sulphur acceptors</td>
</tr>
<tr>
<td>Hydrogen bond donor</td>
<td>Donor hydrogens bonded to oxygen or nitrogen</td>
</tr>
<tr>
<td>Positive ionizable</td>
<td>Atoms with positive Gasteiger charge</td>
</tr>
<tr>
<td>Negative ionizable</td>
<td>Atoms with negative Gasteiger charge</td>
</tr>
<tr>
<td>Metallic</td>
<td>Magnesium, Zinc, Manganese or Iron</td>
</tr>
<tr>
<td>Occupancy</td>
<td>All atoms</td>
</tr>
</tbody>
</table>

The intuition behind these descriptors is that a machine-learning algorithm should be able to learn the relation between different atom types that interact creating regions of high or low energy, making the bonds between protein and ligand stronger or weaker.

These descriptors, as defined before, only represent the type of single atoms, and do not extend intuitively to a 3D model. In fact, HTMD’s procedures have implemented a voxelization method, but such methods will be discussed in section 3.2.

These molecular descriptors are computed for both protein and ligand, resulting in 16 feature maps per complex. Two examples of these feature maps are shown in Figure 3.1 and Figure 3.2.

### 3.1.2 Electrostatic charges

The electrostatic interactions of the complex should give some information on the interaction between the protein and ligand. The Poison-Boltzmann
3.1. Features

Figure 3.1: HTMD Aromatic channel for protein in complex 10GS

Figure 3.2: HTMD Positively charged channels for protein and ligand in complex 10GS
\[ \nabla^2 \phi = c_0 e \cdot [\exp\left(\frac{-e\phi}{k_B T}\right) - \exp\left(\frac{e\phi}{k_B T}\right)] \]

Figure 3.3: Poisson-Boltzmann Equation

The Poisson-Boltzmann equation \[17\], describing the charge distribution between solution and a charged surface, is used to compute these electrostatics. Figure 3.3 shows the equation.

We use the program Adaptative Poisson-Boltzmann Solver\[25\], designed specifically for biomolecular settings.

With APBS, we compute three electrostatic energy maps: one for the energy distribution of the protein alone, a second for the distribution of the ligand alone, and a third for the distribution when both protein and ligand coexist. This combination of maps should have some information about the difference in energy before and after binding, which is related to our target value, the binding affinity of the complex.

These maps are given as 3D voxelized grids by APBS and need no further treatment.

### 3.1.3 Force-field energies

Force-fields are functions approximating the potential energy of an atomic system in terms of interactions between the different atoms.

Much work has been previously done on force-fields to predict binding affinity and docking of complexes. Specifically on binding affinity, most approaches approximate the change of Gibbs free energy and use that to compute the binding affinity constant. These approaches consider only system-wide (statistical) properties derived from the force-fields. The novelty addition of this project to the already explored approaches of other authors is the use of force-fields as derived 3D features of the complexes. In the 3D setup, many takes on the problem of predicting binding affinity have implicitly used force-fields on the step of ligand docking, in which the position of the ligand with respect to the protein is optimized according to minimizing the energy of that certain force-field.

We instead use a force-field to generate maps between atoms and the different energies and attributes that the force-field offers, which will later be voxelized accordingly.

We focus on the well-performing Rosetta all-atom force field. In particular, we explore the attractive, repulsive, electrostatic and solvative energies between pairs of non-bonded atoms that the Rosetta force-field offers. These features are computed using the Rosetta framework\[33\] and PyRosetta.
3.1. Features

Figure 3.4: Plots of a single contribution of the different energies obtained from Rosetta. Interaction pair C-N

An example of the distribution of the energy with respect to distance between two atoms is shown in [Figure 3.4]

Emulating the separation of positive and negative charges in HTMD, we preprocess these Rosetta features to generate 6 maps from the 4 force maps: attractive, repulsive, positive electrostatic, negative electrostatic, positive solvative, negative solvative. We generate these 6 maps for both protein alone and ligand alone, summing up to 12 maps in total. These maps will have some representation applied (discussed in section 3.2) and then normalized to the range 0 to 1 to match the range of the HTMD features.

Other force-fields are implicitly used in other parts of our pipeline. CHARMM[5] is used in the data augmentation via molecular dynamics simulations. AMOEBA[42] and AMBER[9] were also explored as possible alternatives for Rosetta.
3. Methods

3.1.4 Atom identification

One approach derived from the idea of molecular descriptors is to have channels indicating the elements of the atoms. Due to the high number of elements that take part in our complexes, it is unfeasible to introduce 15+ more channels just for this. Our take on this is to create one map for the protein and one for the ligand, in which we distribute spatially the electronegativity of each atom. These values should bear enough information for the neural network to identify the atoms.

To get these electronegativity values we use the Python library Mendeleev. The values are then distributed following the same procedure applied to the force-field features.

As with the rest of the maps, the electronegativity maps are also normalized to the range 0 to 1. An example of the electronegativity maps is shown in Figure 3.5.

3.2 3D representation

With the advent of Deep Learning and the use of Convolutional Neural Networks in computer vision, so have the applications of CNN’s to machine-learning based biochemistry. The inherent 3D structure of proteins and ligands intuitively leads to the use of CNN’s, capturing local properties of a voxelized complex. In order to use a CNN, we need to represent the data in a fixed-size 3D shape. We apply a common approach of creating a cubic
3.3. Deep Learning

Over the last few years, the focus on applying machine learning algorithms to the problem of drug discovery has increased considerably. Many models using classical methods such as linear regression, kernel ridge regression, support vector machines, Gaussian processes, and many others have been successfully applied in drug discovery. Additionally, deep learning methods have shown promising results in this field, especially in the prediction of protein-ligand binding affinities.

Evenly-spaced grid of 24 Angstrom of length centered around the center of mass of the ligand.

For both the Autodock Vina molecular descriptors obtained through HTMD and the APBS electrostatic fields are already represented in 3D voxels.

For HTMD, voxelization is done by applying a filter for each binary map around each active atom. The filter has the following expression:

\[ f(r) = 1 - \exp\left(-\frac{r_{vdw}}{r}\right)^{12} \]

where \( r \) is the distance from the atom center to the voxel center and \( r_{vdw} \) is the Van der Waals radius of the atom.

Part of the project involves finding a good way to voxelize these pointwise features. Two approaches were tested: applying a filter like the previous one, and applying interpolation algorithms.

In Figure 3.2 we list the filters that were tried to distribute the pointwise values in the 3D grids. Likewise, in Figure 3.3 the tested SciPy interpolations are listed.

In Figure 3.6 the spatial distribution of the filters is shown. In Figure 3.7 the example of the respective distributions are shown.
3. Methods

Figure 3.6: Plots of the density distribution of Gaussian and inverse exponential filters

Figure 3.7: Gaussian and inverse exponential filters applied
or random forests [34, 44] have been used as various approaches to predict the binding affinity of protein-ligand complexes. Until recently, the best performing machine-learning approach was that of the random forest RF-Score [34], which relies on 42 molecular descriptors extracted from Autodock Vina.

We developed on top of KDeep’s convolutional neural network, which achieved a lower validation error on PDBBind’s core set.

KDeep is based on SqueezeNet [23]. We have produced two versions of KDeep: the first one is the most faithful reproduction of the original KDeep network, matching the parameter count reported; the second is a modification of the former by increasing the size of the first convolution filter from 1x1x1 to 7x7x7.

The structure of the first version, which we call “Original KDeep”, has the following structure:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output size</th>
<th>Filter size</th>
<th>Stride</th>
<th>Squeeze</th>
<th>Expand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>25x25x25@29</td>
<td>1x1x1</td>
<td>2x2x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv1</td>
<td>12x12x12@96</td>
<td>3x3x3</td>
<td>2x2x2</td>
<td>32</td>
<td>128</td>
</tr>
<tr>
<td>fire1</td>
<td>12x12x12@128</td>
<td>16</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fire2</td>
<td>12x12x12@128</td>
<td>16</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fire3</td>
<td>12x12x12@258</td>
<td>32</td>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool1</td>
<td>6x6x6@258</td>
<td>3x3x3</td>
<td>2x2x2</td>
<td>32</td>
<td>128</td>
</tr>
<tr>
<td>fire4</td>
<td>6x6x6@258</td>
<td>48</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fire5</td>
<td>6x6x6@384</td>
<td>48</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fire6</td>
<td>6x6x6@384</td>
<td>64</td>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fire7</td>
<td>6x6x6@512</td>
<td>64</td>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avgpool1</td>
<td>3x3x3@512</td>
<td>3x3x3</td>
<td>2x2x2</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>flatten</td>
<td>13824</td>
<td>3x3x3</td>
<td>2x2x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dense</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The structure of the second version or “Large-filter KDeep”, has the same structure as the previous one, except the filter size of layer “conv1” is 7x7x7.

We have also used ResNet, with its original specification in [21]. We use a ResNet-101 with the following structure:
### 3. Methods

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output size</th>
<th>Filter size</th>
<th>Stride</th>
<th>Inner dimension</th>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>25x25x25@29</td>
<td>7x7x7</td>
<td>2x2x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv1</td>
<td>12x12x12@64</td>
<td>3x3x3</td>
<td>2x2x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool1</td>
<td>6x6x6@64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inner-res-0</td>
<td>6x6x6@256</td>
<td></td>
<td></td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>start-res-1</td>
<td>3x3x3@512</td>
<td></td>
<td></td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>inner-res-1</td>
<td>3x3x3@512</td>
<td></td>
<td></td>
<td>128</td>
<td>3</td>
</tr>
<tr>
<td>start-res-2</td>
<td>2x2x2@1024</td>
<td></td>
<td></td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>inner-res-2</td>
<td>2x2x2@1024</td>
<td></td>
<td></td>
<td>256</td>
<td>23</td>
</tr>
<tr>
<td>start-res-3</td>
<td>1x1x1@2048</td>
<td></td>
<td></td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>inner-res-3</td>
<td>1x1x1@2048</td>
<td></td>
<td></td>
<td>512</td>
<td>2</td>
</tr>
<tr>
<td>flatten</td>
<td>2048</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dense</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The loss commonly used in binding affinity prediction is the RMSE:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

where $\hat{y}_i$ is the predicted binding affinity and $y_i$ is the true binding affinity.

In our KDeep architectures, all convolutional layers are followed by a rectifier linear unit (ReLU). Other activation functions were tried (ELU, SELU, Leaky ReLU), but no significant difference in performance was found. On our ResNet architecture, the first convolutional layer is followed by a ReLU, whereas the rest of the convolutional layers, used in the residual modules, are preceded by a ReLU, as is recommended in [22]. Neither Batch Normalization (BN) nor Dropout were used, as in all of our experiments the performance was considerably worse when these types of regularization were applied.

For training, we use the AdaM optimizer[26] with its default parameters and a learning rate of $10^{-4}$. We train the networks for 100 epochs using a batch size of 128. The variables are initialized with the default settings of Tensorflow, which corresponds to the Glorot uniform initializer[19].

The implementation of these neural networks is available in Appendix C.

### 3.4 Pipeline

In order to make the project fully reproducible and easily extensible, a modular pipeline was developed for this project. The vast amount of file formats and intermediate steps in the generation of our features made the system quite complex. In this section, we will explain the steps taken from raw data
3.4. Pipeline

until it is inputted to the CNN. All scripts mentioned in this section appears in Appendix C.

3.4.1 Getting the input data

Our initial dataset is PDBBind refined set v.2018. It can be downloaded from \cite{35} as a compressed .tar.gz file. Inside it, there is a folder per complex, e.g. 10gs. In 10gs we will have some files, of which we will focus on 10gs.protein.pdb and 10gs_ligand.mol2. The .pdb file contains 3D atom positions, as well as atom type, residue name and protein chain. The .mol2 file contains atoms positions with their type and explicit bonds between them.

For the sake of simplicity, we will keep using 10gs as an example.

3.4.2 Relaxing the ligands into the protein

The PDBBind dataset was obtained by X-Ray crystallography. This technique allows for the location of atoms with considerable electron density (usually having trouble identifying hydrogen atoms\cite{37}). Because of the interaction between the X-Rays and the electrons of the complex, exact positions may not be very realistic due to ionization effects and the likes\cite{10}. To amend this, we apply a “relaxation”\cite{38} step using the Rosetta Commons\cite{33} framework. This relaxation process minimized the target energy (in our case, Rosetta’s energy function) by moving the atoms in the volume of 20 Angstroms surrounding the ligand’s center of mass. This distance is selected to have relaxed atoms for any rotation of the 24 Angstrom side-length cube centered at the ligand center of mass. Water molecules located far away from the protein pocket (i.e. waters not at distance 3 Angstrom of the protein nor the ligand) are removed to accelerate the relaxation. In cases where ions are in contact with the ligand, i.e. ions that are at a distance of at most 2.5 Angstrom of heavy ligand atoms such as oxygen or nitrogen, we apply constraints on the relaxation to maintain this maximum distance, so as not to have these important ions diverge outside the focused area.

To begin the relaxation, first we need to compute the .params and .pdb of the ligand, which we can obtain by executing make_ligand_pdb.params. This will produce file 10gs_ligand.params and 10gs_ligand.pdb.

After this, we will need to combine protein and ligand .pdb files with make_complex.pdb. Here, the protein is protonated with PropKa\cite{39} via HTMD\cite{13} to recover the unidentified hydrogens, and residue names are normalized to their standard versions (i.e. HIE to HIS, AR0 to ARG, etc). This produces 10gs_complex.pdb.

With this, we can relax the complex by executing minimize_rosetta. This will potentially generate 10 different randomly generated poses, in files of the
3. Methods

form 10gs_complex_00XX.pdb, where XX ranges from 01 to 10. This will also generate file score.sc, where the metadata of the poses (such as the energy score) is stored. From these poses, we will take the one with the least energy. We can set aside the other poses by executing hide_non_minimal_complexes. This will move the other poses to an auxiliary folder, leaving only one file 10gs_complex_00XX.pdb on the 10gs folder. We shall recover the .mol2 file with bonds for the new pose by executing make_ligand_mol2_renamed and make_ligand_mol2. This creates 10gs_ligand_00XX.mol2. We shall also recover the .pdb file for the protein, by executing make_protein.pdb. After this step we should have three files, with names 10gs_complex_00XX.pdb, 10gs_protein_00XX.pdb and 10gs_ligand_00XX.mol2.

3.4.3 Autodock Vina formatted files

Until now, all of our files were essentially slight transformations of the original .pdb and .mol2 files. To compute HTMD features, we will need to apply a final transformation. We need to produce, from our 10gs_protein_00XX.pdb and 10gs_ligand_00XX.mol2, two files in PDBQT format, i.e. 10gs_protein_00XX.pdbqt and 10gs_ligand_00XX.pdbqt, respectively. These files contain Gasteiger charges and a finer atom classification, with special types for carbons in benzenes (aromatic hydrocarbons), hydrogen bond acceptor versions for oxygen and nitrogen, and so on.

To generate these .pdbqt files, we will execute make.pdbqt.

We now have the needed files to compute the molecular descriptor features with HTMD.

3.4.4 Preprocessing Rosetta pointwise energies

In order to compute both electrostatic features with APBS and Rosetta features, we will need to precompute the radii, charges and forces of the complexes’ atoms.

To do so, we will run compute_rosetta_energy, which produces a file named 10gs_complex_00XX.attr.npz, a Numpy compressed file with the data necessary to generate dictionaries for atom’s radius, charge and Rosetta forces.

3.4.5 Computing features

Molecular descriptors - HTMD

To compute the molecular descriptor maps, we will use the script make_htmd_features. This will take the previously generated .pdbqt files, and produce a file 10gs_complex_00XX.hdf5, which contains a grid with 16 maps. This grid can be accessed by key ‘grid’ in the HDF5 structure.
3.4. Pipeline

Electrostatics - APBS

To compute the electrostatic maps, we will use the script make_apbs_features. This will use 10gs_complex_00XX.pdb and 10gs_complex_00XX.attr.npz, and generate a file 10gs_complex_00XX.hdf5. This file will contain a grid with 3 maps, one for the electrostatics of the protein, one for the ligand and one for the whole complex. Again, this grid is stored with key ‘grid’ in the HDF5.

As an intermediate step, the script generates .pqr files for protein, ligand and complex, which store atom coordinates with their charges. These files are input for APBS. APBS generates the grid results in .dx format, which our script transforms to HDF5.

Force-field energies - Rosetta

To compute the Rosetta energy maps, we will use script make_rosetta_features, which requires both 10gs_complex_00XX.pdb and 10gs_complex_00XX.attr.npz. It will produce 10gs_complex_00XX.hdf5, with 4 maps, one per energy, stored with key ‘grid’ in the HDF5 file.

Electronegativity - Mendeleev

To compute the electronegativity maps, we use the script make_electroneg_features, requiring only 10gs_complex_00XX.pdb. It produces 10gs_complex_00XX.hdf5 with 2 maps, one for protein atoms and one for ligand atoms, stored with key ‘grid’ in the HDF5 file.

3.4.6 Merging the features

In this step, all the HDF5 files are merged to generate a single HDF5 file per complex. This can be achieved by executing script make_supermap.

3.4.7 TFRecords for neural network input

In this last step, we take the HDF5 files, read the grids, find the target value for each complex, and create TFRecords of the recommended 100MB size.

Before creating these TFRecords, we have the option to split the dataset into training and validation sets. We can either randomly split by giving a ratio of how many complexes should be collected for training; or we can split by setting the validation set as the complexes of the PDBBind core set and leaving the rest for training.

To do this, we call script make_tfrecords. To choose between random splitting and core set splitting, we use options –split or –core, respectively.
3. Methods

3.4.8 Augmenting the data - Molecular Dynamics simulation

To increase our training data, we will use simulations to let the protein change shape. For this, we will use OpenMM, a molecular mechanics software.

We run a simulation of 2ns of duration, with 200,000 steps of 1fs, taking snapshots every 20,000 steps, or every 20ps.

As input, we need a .pdb and a .psf file for the protein. We can use 10gs_protein.pdb and generate 10gs_protein.psf by running make_protein_psf.

OpenMM will generate a .dcd file, which can be used to generate 10 protein .pdb file. Each of these .pdb files, together with the ligand .mol2 file, can be processed with the previous steps as any other complex. To run OpenMM, we need to execute molecular_dynamics.
4.1 Prediction results

In this section we show the different experiments and the results obtained. We commence with our reproduction of the KDeep\cite{kdeep} experiment. We then show the results obtained with the large-filter KDeep and ResNet-101 architectures when using the following combinations of feature maps:

- HTMD features
- Rosetta features
- Electronegativity features
- APBS features
- Combination of features:
  - HTMD + Rosetta
  - HTMD + electronegativity
  - Rosetta + electronegativity
  - HTMD + Rosetta + electronegativity

The representation used for Rosetta and electronegativity features in this section is the inverse exponential filter.

A summary of the results appears in Figure 4.1 for the validation errors, Figure 4.2 for the test errors on CSAR HiQ Set 1, Figure 4.3 for test errors on CSAR HiQ Set 2 and Figure 4.4 for the test errors on CSAR 2014. In Figure 4.5 and Figure 4.6 the results for Pearson’s R correlation are shown for the validation set and the average for the test sets, respectively. We also show the Pearson’s R per cluster in Figure 4.7 and Figure 4.8 for the best
4. Results

Figure 4.1: Best validation error per model type

large-filter KDeep and ResNet-101 models, respectively; where the clusters are obtained by 90% similarity cutoff of the complexes’ proteins (See Appendix B for a list of the clusters and their complexes).

In all result tables, $\rho$ and $R$ indicate Spearman’s and Pearson’s correlation coefficients, respectively.

4.1.1 Original KDeep - HTMD features

We tried to reproduce the network that KDeep published on their supplementary information. It was not obvious how to deduce some of the parameters of the architecture, such as the size of some filters, because they were not reported. Using the number of parameters of the network, we managed to infer most of the shapes and sizes of the layers. Only two parameters were unidentified: the sizes of the pooling layers, as they are independent of both output size and parameter count. For these unknown parameters we used the ones given by SqueezeNet, the architecture which KDeep is
4.1. Prediction results

Table 4.1: Original KDeep results with HTMD features

<table>
<thead>
<tr>
<th></th>
<th>RMSE</th>
<th>$\rho$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>1.48</td>
<td>0.65</td>
<td>0.66</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.55</td>
<td>0.69</td>
<td>0.71</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>2.22</td>
<td>0.69</td>
<td>0.64</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.72</td>
<td>0.63</td>
<td>0.69</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>0.85</td>
<td>0.79</td>
<td>0.85</td>
</tr>
</tbody>
</table>

This architecture was trained 50 times for 100 epochs each time using HTMD features only. The results appear in Table 4.1.

These results are far from what was reported by KDeep’s paper [24]. We have checked their prediction results for the validation set with PlayMolecule’s webserver, where KDeep is offered, and the results do match the RMSE of 2.09 baseline [24].
4. Results

This leads us to believe that some kind of preprocessing was applied to the data that was not explicitly mentioned on the paper. Because of the poor performance of this baseline, we decided not to train this specific architecture with any of the other features and representations.

4.1.2 Larger filter KDeep

We implemented a slight variation of the KDeep network, in which we use the filter shapes from SqueezeNet. We performed the 8 experiments mentioned above.

We trained the network at least 50 times per feature combination for 100 epochs. The results appear in Table 4.2.

We can observe that the best performance in the RMSE measure from all but the CSAR 2014 dataset was obtained by the combination of HTMD and Rosetta features, with the combination of HTMD, Rosetta and electronega-
4.1. Prediction results

![Figure 4.4: Best CSAR 2014 error per model type](image)

Activity features as a close second best. This may indicate good interaction between Rosetta and HTMD features. The fact that the larger feature map size led to slightly worse results may also indicate the need to increase the width or depth of the architecture.

It is interesting to notice that, when comparing HTMD features with the combination of Rosetta and electronegativity features, the performance was very similar with respect to RMSE, with the latter falling behind by less than 0.2 units, and outperforming HTMD on CSAR 2014. This last behavior happened consistently in the experiments that did not include HTMD features.

In terms of correlations, it is also clear that the HTMD plus Rosetta features achieve best or close to best performance compared to the other combinations.

In regards to APBS features, the performance was very poor. The RMSE for the validation set was 1.82, very far behind any of the other feature combi-
4. Results

nations. We did not explore any further this representation as it consistently showed poor performance when combined with the rest of the feature maps.

In Figure 4.7, we can see the Pearson’s $R$ coefficients per cluster. We can observe that we obtain less anticorrelation in the bottom group of clusters compared to the results from KDeep’s paper.

In general, the network seems to be generalizing well in terms of correlations, with them staying on a very close range regardless of the dataset in question.

4.1.3 ResNet-101

We use the ResNet with 101 layers. We trained this network with the subsets of features mentioned above with 50 different random seeds for 100 epochs. The results appear in Table 4.3.

With this architecture, it is harder for us to find a best performing feature
Table 4.2: Results for the experiments when using a large filter KDeep network

<table>
<thead>
<tr>
<th></th>
<th>HTMD</th>
<th>Rosetta</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>$\rho$</td>
<td>$R$</td>
</tr>
<tr>
<td>Training set</td>
<td>1.399</td>
<td>0.708</td>
<td>0.712</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.484</td>
<td>0.731</td>
<td>0.745</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>2.058</td>
<td>0.747</td>
<td>0.724</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.592</td>
<td>0.710</td>
<td>0.758</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.342</td>
<td>0.775</td>
<td>0.823</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.259</td>
<td>0.768</td>
<td>0.770</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>1.960</td>
<td>0.787</td>
<td>0.780</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.559</td>
<td>0.769</td>
<td>0.768</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.067</td>
<td>0.750</td>
<td>0.812</td>
</tr>
</tbody>
</table>
4. Results

![Graph showing Pearson's R per model type](image)

**Figure 4.6: Averaged test sets Pearson’s R per model type**

combination. While HTMD seems to perform best for CSAR HiQ Set 2, we also see very good performance for CSAR HiQ Set 1 on both Rosetta and HTMD plus electronegativity features. The combination that achieved the best performance in our validation set was the combination of HTMD, Rosetta and electronegativity features. Overall, both Rosetta and HTMD achieve good results in some of the datasets, while at the same time using fewer feature maps than the combination of the two.

In ResNet’s case the combination of Rosetta and electronegativity improved the prediction of the validation set, but worsened the predictions of all test datasets with respect to Rosetta features alone. This behavior is opposite to the one obtained in the large-filter KDeep architecture. Also, the difference in performance between HTMD plus Rosetta and HTMD, Rosetta and electronegativity was very small.

Here again, the performance of APBS features was very poor, with an RMSE on the validation set of 1.96, and is not included in the result’s table as we
did not continue exploring this feature map.

In Figure 4.8 we can see the Pearson’s $R$ coefficients per cluster. We observe little difference between these correlations and the ones from the previous network. There is a slight worse performance in the bottom clusters, though one of the previously anticorrelated is now uncorrelated. We can also observe that the per cluster performance is not exactly the same as with the large-filter KDeep network.

4.1.4 Comparison of results with other networks

Here, we compare the results published in KDeep’s paper [24], in which they report errors for the PDBBind core set and CSAR HiQ Set 1, Set 2 and v.2014 sets for both their network and their reproduction of RF-Score [34].

We can see that both large-filter KDeep and ResNet-101 outperform the results reported in KDeep’s paper for all the test datasets considered. In fact, large-filter KDeep also achieves a performance similar or slightly better compared to RF-Score in the test datasets. With regards to the validation set, PDBBind Core set, our networks did not manage to beat the results reported in KDeep’s paper. With ResNet-101 we managed to achieve a performance similar to RF-Score, with better result in the validation set and worse result in the CSAR 2014 test dataset.

4.2 Representation results

In this section, we compare the different representation for Rosetta and electronegativity maps. To argue about them, we used the best performing network of ResNet-101 architecture. The results appear in Table 4.5.

We have observed that the distribution of the inverse exponential filter, when applied to our dataset with grids of side length 25, is very binary-like, with values usually either larger than 0.7 or lower than 0.1. This together with the fact that the maps we apply this filter to are normalized to the range 0 to 1, makes the maps look very much like a 3D binary grid.

We observed that performance did not substantially change between the inverse exponential and Gaussian filters. Except for the results in CSAR HiQ Set 2, the values for RMSE and correlations do not differ much between these two filters.

Regarding interpolation methods, linear interpolation showed worse behavior than the other two when comparing the RMSE of the test datasets. There was no significant difference in performance between Gaussian and thin-plate interpolations, with Gaussian interpolation performing slightly better on the validation set and CSAR HiQ 1 but worse on CSAR HiQ 2.
Table 4.3: Results for the experiments when using a ResNet-101 network

<table>
<thead>
<tr>
<th></th>
<th>HTMD</th>
<th>Rosetta</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>ρ</td>
<td>R</td>
</tr>
<tr>
<td>Training set</td>
<td>1.25</td>
<td>0.817</td>
<td>0.821</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.42</td>
<td>0.775</td>
<td>0.790</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>1.976</td>
<td>0.793</td>
<td>0.748</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.524</td>
<td>0.736</td>
<td>0.771</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.258</td>
<td>0.735</td>
<td>0.765</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HTMD + R</th>
<th>HTMD + EN</th>
<th>R + EN</th>
<th>HTMD+R+EN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>ρ</td>
<td>R</td>
<td>RMSE</td>
</tr>
<tr>
<td>Training set</td>
<td>0.900</td>
<td>0.890</td>
<td>0.890</td>
<td>1.063</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.332</td>
<td>0.769</td>
<td>0.788</td>
<td>1.358</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>2.008</td>
<td>0.768</td>
<td>0.742</td>
<td>1.951</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.560</td>
<td>0.718</td>
<td>0.751</td>
<td>1.585</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.425</td>
<td>0.724</td>
<td>0.779</td>
<td>1.259</td>
</tr>
</tbody>
</table>
Figure 4.7: Pearson $R$ per cluster for PDBBind Core set for large-filter KDeep.
4. Results

Table 4.4: Comparison between KDeep’s paper results and our networks’ results.

<table>
<thead>
<tr>
<th>KDeep paper</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KDeep</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
</tr>
<tr>
<td>PDBBind Core set</td>
<td>1.27</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>2.09</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.92</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.75</td>
</tr>
</tbody>
</table>

We saw a larger margin in performance when comparing RMSE between filters and interpolation methods. We observed a consistent lower prediction error in filter methods for all datasets. When comparing correlation coefficients, except for linear interpolation, the performance was very similar.

4.3 Data augmentation results

We augmented the data using molecular dynamics simulations for 2ns. From the 4463 starting complexes, we obtained a total of 48935 poses without errors. Around 20 of the original complexes failed the simulations consistently and had to be excluded.

The simulations took more than one week to complete when using around 10 Titan X GPUs.

We run the best performing experiments once again on the augmented data for 15 epochs (equivalent to more than 100 epochs on the original data). We run the experiments 25 times each. The results appear in Table 4.6. The validation error did not improve when compared to the same networks trained using the original dataset.
4.3. Data augmentation results

Figure 4.8: Pearson $R$ per cluster for PDBBind Core set. for ResNet-101
Table 4.5: Results for the experiments on different 3D representations of Rosetta + electronegativity maps using a ResNet-101 network

<table>
<thead>
<tr>
<th>Filter</th>
<th>Inverse Exponential</th>
<th>Gaussian</th>
<th>Linear</th>
<th>Gaussian</th>
<th>Thin Plate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>$\rho$</td>
<td>$R$</td>
<td>RMSE</td>
<td>$\rho$</td>
</tr>
<tr>
<td>Training set</td>
<td>0.97</td>
<td>0.87</td>
<td>0.87</td>
<td>1.10</td>
<td>0.82</td>
</tr>
<tr>
<td>Validation set</td>
<td>1.36</td>
<td>0.76</td>
<td>0.78</td>
<td>1.38</td>
<td>0.75</td>
</tr>
<tr>
<td>CSAR HiQ set 1</td>
<td>2.00</td>
<td>0.76</td>
<td>0.75</td>
<td>1.97</td>
<td>0.78</td>
</tr>
<tr>
<td>CSAR HiQ set 2</td>
<td>1.68</td>
<td>0.70</td>
<td>0.70</td>
<td>1.74</td>
<td>0.60</td>
</tr>
<tr>
<td>CSAR 2014</td>
<td>1.12</td>
<td>0.74</td>
<td>0.77</td>
<td>0.95</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Table 4.6: Results for the best performing networks on MD augmented data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Large-filter KDeep</th>
<th>ResNet-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation RMSE</td>
<td>1.458</td>
<td>1.389</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion

In this chapter we offer our deductions from the information given in the results.

The goals of this project were:

1. Assessing the performance of energy-based features.
2. Comparing 3D voxelization procedures for pointwise spatial features.
3. Improving performance by tuning or changing the neural network architecture.
4. Exploring whether data preprocessing and augmentation can improve predictive power.

On the first point, we have observed that it is definitely possible to use energy-based features like the Rosetta force-field used here. The performance of Rosetta features with ResNet is close to the performance of previous state-of-the-art RF-Score. We also observed improvement when combined with HTMD or electronegativity maps, which indicates that it could be used as an extension of current molecular descriptor based models. We have also observed that correlation coefficients improved when these features were added to HTMD features, which may showcase the diversity of information that these new maps provide. We also tested using APBS features, but their performance was very poor. Considering the significance of electrostatics in protein-ligand binding, we believe that the drop in performance may be caused by the continuous nature of the maps, as all the other maps had a more stepwise shape due to the nature of the filters.

On the second point, we have tested five different voxelization methods and found no significant improvement to what the state-of-the-art is. It was not surprising to see that linear interpolation was the worst performing overall, considering that energy distributes as an inverse polynomial through space,
but it was interesting to see that the performance was still within acceptable margins when compared to the behavior of all methods shown in this project.

On the third point, we did not manage to reproduce the original results reported by KDeep’s paper. We built our own reproduction based on the diagrams that were released on KDeep’s supplementary information, and based on that, we beat its prediction RMSE with the other architectures by a margin of 0.25, but our best result was still 0.06 units away from the reported KDeep result. We have managed to corroborate KDeep’s result by using the public webserver they offer. This can only indicate that there is some difference in the preprocessing or the neural network that we were not able to account for. Regardless of this, we have shown that our best model based on KDeep’s architecture gets better generalization results when considering not only validation set but the other three datasets. And even in the validation set, we have observed less anticorrelation, which may indicate that the additional Rosetta features add some new information to the HTMD features.

Finally, we have developed a pipeline to preprocess the protein-ligand complexes in a standard way. This pipeline consists in mainly relaxing the ligand inside the protein to lower the energy of the complex, and thus reducing the diverse biases that the data may have due to the inherent characteristics of its extraction methods (i.e. X-Ray crystallography). The results show a better generalization in all combinations of features and neural network architectures, which leads us to think that this minimization may be successful in making the data harder to overfit to. We also augmented the dataset tenfold by simulating the movement of the protein for 2ns, thus enlarging the dataset size from 4.4k to 48k poses. The simulations took more than one week to complete when using around 10 Titan X GPUs. Because of this, we have stored both the resultant poses as well as the checkpoints so that the simulations can be continued if there is any need to increase the dataset. We have seen that augmenting the dataset by molecular dynamics simulations did not directly improve the performance, which may be because the simulations were too short. It could also be the case that Rosetta relaxation may be removing the effect of the simulations when optimizing the position of the atoms around the ligand. Because we only ran the experiments for 25 times, it is a possibility that we got unlucky with the random seeds used in them.

If we were to continue this study, some ideas for next steps would be the following:

- Regarding 3D voxelization, it would be interesting to let the neural network learn the filter. This could be achieved by using deconvolutional layers, for example.
• Regarding data augmentation, it would be interesting to generate fake complexes to give the neural network examples of “bad binding affinity” for all the proteins. We noticed that some protein clusters are over-represented, and this approach could help remove bias in the dataset.

• Because the per cluster correlations changed when comparing large-filter KDeep and ResNet, it would be a possibility to train multiple networks and ensemble them to slightly improve the performance.

• It would probably be beneficial to add structures of different nature: different types of proteins, different extraction methods (i.e. not only structures obtained with X-Ray crystallography) and also repeated complexes from different sources.

• Finding the neural network architecture could be left to some of the recently popularized AutoML techniques. This could find a custom structure that may fit better the nature of the problem treated here.
In this work we have developed on top of the preceding results by KDeep \cite{24} in order to improve the prediction of binding affinity in protein-ligand complexes. We have approached this by four different routes: modification of the features, modification of the neural network architecture, data preprocessing, and augmentation of the dataset.

Our work on the feature maps has shown that it is possible to get decent, albeit not the best, performing out of energy-based feature maps like our maps based in the Rosetta all-atom force field. We have also explored different ways of representing these energy features in 3D space by application of different filters and interpolation methods, and shown that the method used by the state-of-the-art (the inverse exponential filter) was best performing compared to the other ones.

In regards to the neural network architecture, we were not able to reproduce the results of KDeep’s paper. We tried to reproduce the original KDeep network but the results were far from close, which indicates that we may be lacking some step in our data preprocessing or some mistake in the implementation of the network. We developed two other networks, a variant of KDeep with a larger filter on the first convolution, and a ResNet-101 architecture. We have been unable to achieve a better result for the validation set, PDBBind’s core set, but we managed to improve on all the test datasets. We have also obtained Pearson correlation coefficients that are much more consistent between validation and test data, indicating a better generalization behavior. Protein-wise, we have obtained similar positive correlations and less negatively correlated proteins than KDeep’s results. Because of the usage of binding affinity as a ranking measurement to compare how well ligands bind to a single protein, this result indicates a better performance of our networks as ranking tools.

As for data preprocessing, we have introduced a process of relaxation of
6. Conclusions

the ligand inside the protein using the Rosetta force field. We have seen improved results in some of the test datasets when comparing KDeep’s result with our reproduction of their network, indicating that the process of relaxation may better the spatial structure of the inputted complexes. This is reinforced by the later results with large-filter KDeep and ResNet-101, which obtained very balanced correlation coefficients even when using the only HTMD features.

Finally, we have augmented the dataset by applying molecular dynamics simulations, growing it by 10 times.
Bibliography


[34] Hongjian Li, Kwong-Sak Leung, Man-Hon Wong, and Pedro J. Ballester. Improving autodock vina using random forest: The growing accuracy


Appendices
## Appendix A

### Dataset complexes

#### A.1 Training data: PDBBind 2018 Refined set

<table>
<thead>
<tr>
<th>ID</th>
<th>PDB ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a94</td>
<td>1a09</td>
</tr>
<tr>
<td>1a7l</td>
<td>1a7m</td>
</tr>
<tr>
<td>1a75</td>
<td>1a7q</td>
</tr>
<tr>
<td>1a7f</td>
<td>1a7g</td>
</tr>
<tr>
<td>1a7m</td>
<td>1a7n</td>
</tr>
<tr>
<td>1a7o</td>
<td>1a7p</td>
</tr>
<tr>
<td>1a7r</td>
<td>1a7s</td>
</tr>
<tr>
<td>1a7t</td>
<td>1a7u</td>
</tr>
<tr>
<td>1a7v</td>
<td>1a7w</td>
</tr>
<tr>
<td>1a7x</td>
<td>1a7y</td>
</tr>
<tr>
<td>1a7z</td>
<td>1a7a</td>
</tr>
<tr>
<td>1a81</td>
<td>1a82</td>
</tr>
<tr>
<td>1a83</td>
<td>1a84</td>
</tr>
<tr>
<td>1a85</td>
<td>1a86</td>
</tr>
<tr>
<td>1a87</td>
<td>1a88</td>
</tr>
<tr>
<td>1a89</td>
<td>1a8a</td>
</tr>
<tr>
<td>1a90</td>
<td>1a91</td>
</tr>
<tr>
<td>1a91</td>
<td>1a92</td>
</tr>
<tr>
<td>1a92</td>
<td>1a93</td>
</tr>
<tr>
<td>1a93</td>
<td>1a94</td>
</tr>
<tr>
<td>1a94</td>
<td>1a95</td>
</tr>
<tr>
<td>1a95</td>
<td>1a96</td>
</tr>
<tr>
<td>1a96</td>
<td>1a97</td>
</tr>
<tr>
<td>1a97</td>
<td>1a98</td>
</tr>
<tr>
<td>1a98</td>
<td>1a99</td>
</tr>
<tr>
<td>1a99</td>
<td>1a9a</td>
</tr>
<tr>
<td>1a9a</td>
<td>1a9b</td>
</tr>
<tr>
<td>1a9b</td>
<td>1a9c</td>
</tr>
<tr>
<td>1a9c</td>
<td>1a9d</td>
</tr>
<tr>
<td>1a9d</td>
<td>1a9e</td>
</tr>
<tr>
<td>1a9e</td>
<td>1a9f</td>
</tr>
<tr>
<td>1a9f</td>
<td>1a9g</td>
</tr>
<tr>
<td>1a9g</td>
<td>1a9h</td>
</tr>
<tr>
<td>1a9h</td>
<td>1a9i</td>
</tr>
<tr>
<td>1a9i</td>
<td>1a9j</td>
</tr>
<tr>
<td>1a9j</td>
<td>1a9k</td>
</tr>
<tr>
<td>1a9k</td>
<td>1a9l</td>
</tr>
<tr>
<td>1a9l</td>
<td>1a9m</td>
</tr>
<tr>
<td>1a9m</td>
<td>1a9n</td>
</tr>
<tr>
<td>1a9n</td>
<td>1a9o</td>
</tr>
<tr>
<td>1a9o</td>
<td>1a9p</td>
</tr>
<tr>
<td>1a9p</td>
<td>1a9q</td>
</tr>
<tr>
<td>1a9q</td>
<td>1a9r</td>
</tr>
<tr>
<td>1a9r</td>
<td>1a9s</td>
</tr>
<tr>
<td>1a9s</td>
<td>1a9t</td>
</tr>
<tr>
<td>1a9t</td>
<td>1a9u</td>
</tr>
<tr>
<td>1a9u</td>
<td>1a9v</td>
</tr>
<tr>
<td>1a9v</td>
<td>1a9w</td>
</tr>
<tr>
<td>1a9w</td>
<td>1a9x</td>
</tr>
<tr>
<td>1a9x</td>
<td>1a9y</td>
</tr>
<tr>
<td>1a9y</td>
<td>1a9z</td>
</tr>
<tr>
<td>1a9z</td>
<td>1a9a</td>
</tr>
<tr>
<td>1a9a</td>
<td>1a9b</td>
</tr>
<tr>
<td>1a9b</td>
<td>1a9c</td>
</tr>
<tr>
<td>1a9c</td>
<td>1a9d</td>
</tr>
<tr>
<td>1a9d</td>
<td>1a9e</td>
</tr>
<tr>
<td>1a9e</td>
<td>1a9f</td>
</tr>
<tr>
<td>1a9f</td>
<td>1a9g</td>
</tr>
<tr>
<td>1a9g</td>
<td>1a9h</td>
</tr>
<tr>
<td>1a9h</td>
<td>1a9i</td>
</tr>
<tr>
<td>1a9i</td>
<td>1a9j</td>
</tr>
<tr>
<td>1a9j</td>
<td>1a9k</td>
</tr>
<tr>
<td>1a9k</td>
<td>1a9l</td>
</tr>
<tr>
<td>1a9l</td>
<td>1a9m</td>
</tr>
<tr>
<td>1a9m</td>
<td>1a9n</td>
</tr>
<tr>
<td>1a9n</td>
<td>1a9o</td>
</tr>
<tr>
<td>1a9o</td>
<td>1a9p</td>
</tr>
<tr>
<td>1a9p</td>
<td>1a9q</td>
</tr>
<tr>
<td>1a9q</td>
<td>1a9r</td>
</tr>
<tr>
<td>1a9r</td>
<td>1a9s</td>
</tr>
<tr>
<td>1a9s</td>
<td>1a9t</td>
</tr>
<tr>
<td>1a9t</td>
<td>1a9u</td>
</tr>
<tr>
<td>1a9u</td>
<td>1a9v</td>
</tr>
<tr>
<td>1a9v</td>
<td>1a9w</td>
</tr>
<tr>
<td>1a9w</td>
<td>1a9x</td>
</tr>
<tr>
<td>1a9x</td>
<td>1a9y</td>
</tr>
<tr>
<td>1a9y</td>
<td>1a9z</td>
</tr>
<tr>
<td>1a9z</td>
<td>1a9a</td>
</tr>
</tbody>
</table>
A. Dataset complexes

10ar 10au 10ba 10cq 10d8 10di 10dj 10e8 10gd 10gg 10gx 10gz
10hr 10if 10kl 10ls 10la 10lx 10lny 10onx 10p19 10p1o 10p57 10p9 10p8 10p9 10p8 10p69 10pbf
10px 10p9 10p8 10p7 10p6 10p5 10p4 10p3 10p2 10p1 10p0 10p1 10p0 10p1 10p0 10p5 10p4 10p3 10p2 10p1 10p0
10pm 10p5 10pro 10pvn 10px 10p9 10p8 10p7 10p6 10p5 10p4 10p3 10p2 10p1 10p0
10q4 10q6 10q7 10q8 10q7 10q6 10q9 10q8 10q7 10q6 10q9 10q8 10q7 10q6 10q9 10q8 10q7 10q6 10q9 10q8 10q7 10q6 10q9
10q9 10q8 10q7 10q6 10q5 10q4 10q3 10q2 10q1 10q0 10q9 10q8 10q7 10q6 10q5 10q4 10q3 10q2 10q1 10q0
10r9 10r8 10r7 10r6 10r5 10r4 10r3 10r2 10r1 10r0 10r9 10r8 10r7 10r6 10r5 10r4 10r3 10r2 10r1 10r0
10s9 10s8 10s7 10s6 10s5 10s4 10s3 10s2 10s1 10s0 10s9 10s8 10s7 10s6 10s5 10s4 10s3 10s2 10s1 10s0
10t7 10t6 10t5 10t4 10t3 10t2 10t1 10t0 10t9 10t8 10t7 10t6 10t5 10t4 10t3 10t2 10t1 10t0
10u2 10u1 10u0 10v9 10v8 10v7 10v6 10v5 10v4 10v3 10v2 10v1 10v0 10w9 10w8 10w7 10w6 10w5 10w4 10w3 10w2 10w1 10w0
A.1. Training data: PDBBind 2018 Refined set
### A. Dataset complexes

<table>
<thead>
<tr>
<th>5uxf</th>
<th>5v0n</th>
<th>5v79</th>
<th>5v7a</th>
<th>5v82</th>
<th>5vb7</th>
<th>5vb6</th>
<th>5vb5</th>
<th>5v83</th>
<th>5v8v</th>
<th>5v8c</th>
<th>5v8w</th>
</tr>
</thead>
<tbody>
<tr>
<td>5vcy</td>
<td>5vcz</td>
<td>5vd0</td>
<td>5vd1</td>
<td>5vd2</td>
<td>5vd3</td>
<td>5vg7</td>
<td>5v8h</td>
<td>5vle</td>
<td>5v7i</td>
<td>5v7j</td>
<td>5v7k</td>
</tr>
<tr>
<td>5v7l</td>
<td>5v7m</td>
<td>5v7n</td>
<td>5v7o</td>
<td>5v7p</td>
<td>5v8d</td>
<td>5v8e</td>
<td>5v8f</td>
<td>5v8g</td>
<td>5v8h</td>
<td>5v8i</td>
<td>5v8j</td>
</tr>
<tr>
<td>5v8k</td>
<td>5v8l</td>
<td>5v8m</td>
<td>5v8n</td>
<td>5v8o</td>
<td>5v8p</td>
<td>5v8q</td>
<td>5v8r</td>
<td>5v8s</td>
<td>5v8t</td>
<td>5v8u</td>
<td>5v8v</td>
</tr>
<tr>
<td>5v8w</td>
<td>5v8x</td>
<td>5v8y</td>
<td>5v8z</td>
<td>5v7a</td>
<td>5v7b</td>
<td>5v7c</td>
<td>5v7d</td>
<td>5v7e</td>
<td>5v7f</td>
<td>5v7g</td>
<td>5v7h</td>
</tr>
</tbody>
</table>

### A.2 Validation data: PDBBind 2018 Core set

<table>
<thead>
<tr>
<th>1a30</th>
<th>1bcu</th>
<th>1bzc</th>
<th>1e66</th>
<th>1eby</th>
<th>1g2k</th>
<th>1gpk</th>
<th>1gpn</th>
<th>1h22</th>
<th>1i23</th>
<th>1k1i</th>
<th>1lpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1mq6</td>
<td>1nc1</td>
<td>1nq7</td>
<td>1o6h</td>
<td>1o7h</td>
<td>1o8h</td>
<td>1o9h</td>
<td>1o10h</td>
<td>1o11h</td>
<td>1o12h</td>
<td>1o13h</td>
<td>1o14h</td>
</tr>
<tr>
<td>1g81</td>
<td>1g82</td>
<td>1g83</td>
<td>1g84</td>
<td>1g85</td>
<td>1g86</td>
<td>1g87</td>
<td>1g88</td>
<td>1g89</td>
<td>1g90</td>
<td>1g91</td>
<td>1g92</td>
</tr>
<tr>
<td>1ygr</td>
<td>1yhr</td>
<td>1ypr</td>
<td>1yqr</td>
<td>1ytr</td>
<td>1ypr</td>
<td>1yqr</td>
<td>1ytr</td>
<td>1ypr</td>
<td>1yqr</td>
<td>1ytr</td>
<td>1ypr</td>
</tr>
<tr>
<td>2oc7</td>
<td>2oc8</td>
<td>2oc9</td>
<td>2oc10</td>
<td>2oc11</td>
<td>2oc12</td>
<td>2oc13</td>
<td>2oc14</td>
<td>2oc15</td>
<td>2oc16</td>
<td>2oc17</td>
<td>2oc18</td>
</tr>
<tr>
<td>2qbr</td>
<td>2qbt</td>
<td>2qbc</td>
<td>2qbb</td>
<td>2qba</td>
<td>2qzb</td>
<td>2qzy</td>
<td>2qzv</td>
<td>2qzq</td>
<td>2qzr</td>
<td>2qzs</td>
<td>2qzt</td>
</tr>
<tr>
<td>2wca</td>
<td>2wct</td>
<td>2wdv</td>
<td>2wde</td>
<td>2wdc</td>
<td>2wdb</td>
<td>2wda</td>
<td>2wca</td>
<td>2wct</td>
<td>2wdv</td>
<td>2wde</td>
<td>2wdc</td>
</tr>
<tr>
<td>2f73</td>
<td>2f74</td>
<td>2f75</td>
<td>2f76</td>
<td>2f77</td>
<td>2f78</td>
<td>2f79</td>
<td>2f7a</td>
<td>2f7b</td>
<td>2f7c</td>
<td>2f7d</td>
<td>2f7e</td>
</tr>
<tr>
<td>3b7v</td>
<td>3b8v</td>
<td>3b9v</td>
<td>3bav</td>
<td>3baw</td>
<td>3bax</td>
<td>3bby</td>
<td>3bcz</td>
<td>3bde</td>
<td>3bdf</td>
<td>3bdg</td>
<td>3bdh</td>
</tr>
<tr>
<td>3g8q</td>
<td>3g8r</td>
<td>3g8s</td>
<td>3g8t</td>
<td>3g8u</td>
<td>3g8v</td>
<td>3g8w</td>
<td>3g8x</td>
<td>3g8y</td>
<td>3g8z</td>
<td>3g8a</td>
<td>3g8b</td>
</tr>
<tr>
<td>3h9i</td>
<td>3h9j</td>
<td>3h9k</td>
<td>3h9l</td>
<td>3h9m</td>
<td>3h9n</td>
<td>3h9o</td>
<td>3h9p</td>
<td>3h9q</td>
<td>3h9r</td>
<td>3h9s</td>
<td>3h9t</td>
</tr>
<tr>
<td>3u8e</td>
<td>3u8f</td>
<td>3u8g</td>
<td>3u8h</td>
<td>3u8i</td>
<td>3u8j</td>
<td>3u8k</td>
<td>3u8l</td>
<td>3u8m</td>
<td>3u8n</td>
<td>3u8o</td>
<td>3u8p</td>
</tr>
<tr>
<td>3v9i</td>
<td>3v9j</td>
<td>3v9k</td>
<td>3v9l</td>
<td>3v9m</td>
<td>3v9n</td>
<td>3v9o</td>
<td>3v9p</td>
<td>3v9q</td>
<td>3v9r</td>
<td>3v9s</td>
<td>3v9t</td>
</tr>
<tr>
<td>3u0e</td>
<td>3u0f</td>
<td>3u0g</td>
<td>3u0h</td>
<td>3u0i</td>
<td>3u0j</td>
<td>3u0k</td>
<td>3u0l</td>
<td>3u0m</td>
<td>3u0n</td>
<td>3u0o</td>
<td>3u0p</td>
</tr>
<tr>
<td>3z8s</td>
<td>3z8t</td>
<td>3z8u</td>
<td>3z8v</td>
<td>3z8w</td>
<td>3z8x</td>
<td>3z8y</td>
<td>3z8z</td>
<td>3z8a</td>
<td>3z8b</td>
<td>3z8c</td>
<td>3z8d</td>
</tr>
</tbody>
</table>

### A.3 Test set: CSAR HiQ

#### A.3.1 Set 1

<table>
<thead>
<tr>
<th>1ax1</th>
<th>1swk</th>
<th>1vot</th>
<th>2add</th>
<th>2add</th>
<th>2ade</th>
<th>2b3f</th>
<th>2c1q</th>
<th>2cp</th>
<th>2hr6</th>
<th>2hik</th>
<th>2k4c</th>
<th>2k4d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2gbb</td>
<td>2snq</td>
<td>2au0</td>
<td>2pq9</td>
<td>2pqw</td>
<td>2pqm</td>
<td>2pqj</td>
<td>2pqd</td>
<td>2pge</td>
<td>2pge</td>
<td>2pge</td>
<td>2pge</td>
<td>2pge</td>
</tr>
<tr>
<td>2fag</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
<td>2fai</td>
</tr>
<tr>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
<td>4icv</td>
</tr>
<tr>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
<td>4lcz</td>
</tr>
<tr>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
<td>4q8j</td>
</tr>
<tr>
<td>5a76</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
<td>5a7b</td>
</tr>
</tbody>
</table>

#### A.3.2 Set 2

<table>
<thead>
<tr>
<th>1ax2</th>
<th>1by1</th>
<th>1c21</th>
<th>1d24</th>
<th>1e27</th>
<th>1f2a</th>
<th>1g2d</th>
<th>1h2g</th>
<th>1i2j</th>
<th>1j2k</th>
<th>1l2m</th>
<th>1m2n</th>
<th>1n2o</th>
</tr>
</thead>
<tbody>
<tr>
<td>1axl</td>
<td>1bym</td>
<td>1c2l</td>
<td>1d2m</td>
<td>1e2n</td>
<td>1f2o</td>
<td>1g2p</td>
<td>1h2q</td>
<td>1i2r</td>
<td>1j2s</td>
<td>1k2t</td>
<td>1l2u</td>
<td>1m2v</td>
</tr>
<tr>
<td>1n2w</td>
<td>1o2x</td>
<td>1p2y</td>
<td>1q2z</td>
<td>1r2a</td>
<td>1s2b</td>
<td>1t2c</td>
<td>1u2d</td>
<td>1v2e</td>
<td>1w2f</td>
<td>1x2g</td>
<td>1y2h</td>
<td>1z2i</td>
</tr>
</tbody>
</table>

### A.4 Test set: CSAR 2014

<table>
<thead>
<tr>
<th>fxa006</th>
<th>fxa102</th>
<th>fxa106</th>
<th>fxa222</th>
<th>syk222</th>
<th>syk224</th>
<th>syk226</th>
<th>syk249</th>
<th>trmd444</th>
<th>trmd446</th>
<th>trmd448</th>
<th>trmd450</th>
</tr>
</thead>
<tbody>
<tr>
<td>trmd452</td>
<td>trmd454</td>
<td>trmd456</td>
<td>trmd458</td>
<td>trmd460</td>
<td>trmd462</td>
<td>trmd464</td>
<td>trmd466</td>
<td>trmd468</td>
<td>trmd470</td>
<td>trmd472</td>
<td>trmd474</td>
</tr>
<tr>
<td>trmd476</td>
<td>trmd478</td>
<td>trmd480</td>
<td>trmd482</td>
<td>trmd484</td>
<td>trmd486</td>
<td>trmd488</td>
<td>trmd490</td>
<td>trmd492</td>
<td>trmd494</td>
<td>trmd496</td>
<td>trmd498</td>
</tr>
<tr>
<td>trmd500</td>
<td>trmd502</td>
<td>trmd504</td>
<td>trmd506</td>
<td>trmd508</td>
<td>trmd510</td>
<td>trmd512</td>
<td>trmd514</td>
<td>trmd516</td>
<td>trmd518</td>
<td>trmd520</td>
<td>trmd522</td>
</tr>
</tbody>
</table>

### A.5 Test set: CSAR 2014

| 58 |
### Core set clusters

<table>
<thead>
<tr>
<th>ID</th>
<th>PDB Codes</th>
<th>ID</th>
<th>PDB Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3ao4, 3so, 3sx, 3z2, 4cig</td>
<td>30</td>
<td>1p1n, 1p1q, 1syi, 2al5</td>
</tr>
<tr>
<td>1</td>
<td>1a30, 1ebv, 1g2k, 2qng, 3o9i</td>
<td>31</td>
<td>1bcu, 1yvt, 2zda, 3bv9, 3utu</td>
</tr>
<tr>
<td>2</td>
<td>3oe4, 3oe5, 3ozs, 3ozt</td>
<td>32</td>
<td>3u8k, 3u8n, 3wtj, 3zdq, 4qac</td>
</tr>
<tr>
<td>3</td>
<td>2zb1, 3e92, 3e93, 49w</td>
<td>33</td>
<td>1e66, 1gpk, 1gpn, 1h22, 1h23</td>
</tr>
<tr>
<td>4</td>
<td>1bzc, 2hbl, 2qbp, 2qbp, 2qbr</td>
<td>34</td>
<td>100h, 1u11, 1w4o</td>
</tr>
<tr>
<td>5</td>
<td>2weg, 3dd0, 3kwa, 3ryj, 4js</td>
<td>35</td>
<td>2vce, 2zcr, 2zy1, 3aca, 4ea2</td>
</tr>
<tr>
<td>6</td>
<td>2vot, 2xii, 4j28, 4jfs, 4pcs</td>
<td>36</td>
<td>1vso, 3f1v, 3f2v, 3gbh, 4dld</td>
</tr>
<tr>
<td>7</td>
<td>1s9s, 3b5r, 3b65, 3b68, 3g0w</td>
<td>37</td>
<td>3nq9, 3ueu, 3uve, 3uwe, 3uex</td>
</tr>
<tr>
<td>8</td>
<td>2c3i, 3bgz, 3jya, 4k18, 5dwr</td>
<td>38</td>
<td>3kr8, 4j21, 4j3l</td>
</tr>
<tr>
<td>9</td>
<td>1kl1, 1o3f, 1uto, 3gy4, 4abg</td>
<td>39</td>
<td>4e5w, 4ivb, 4ive, 4ivd, 4k77</td>
</tr>
<tr>
<td>10</td>
<td>2vkm, 3rsx, 3udb, 4dv, 4gid</td>
<td>40</td>
<td>3f3a, 3f3c, 3f3d, 3f3e, 4mne</td>
</tr>
<tr>
<td>11</td>
<td>3arp, 3arq</td>
<td>41</td>
<td>2v7a, 3k5v, 3ms, 3py, 4wp</td>
</tr>
<tr>
<td>12</td>
<td>3eby</td>
<td>42</td>
<td>1qf1, 1z9g, 3fcq, 4tmn, 5tmn</td>
</tr>
<tr>
<td>13</td>
<td>1e66, 3e66, 4e66, 5e66</td>
<td>43</td>
<td>3cyz</td>
</tr>
<tr>
<td>14</td>
<td>2v00, 3pr, 3pww, 3uri, 3ws2</td>
<td>44</td>
<td>2cbv, 2cet, 2j28, 2j7h, 2wbg</td>
</tr>
<tr>
<td>15</td>
<td>1qkt, 2p15, 2pog, 3e4, 4mdd</td>
<td>45</td>
<td>2v2v, 3e5a, 3myg, 3u04, 3up2</td>
</tr>
<tr>
<td>16</td>
<td>1ry5, 1s38, 3c5, 3c6, 3r3e</td>
<td>46</td>
<td>3ehy, 3lka, 3x7, 3tsk, 4gr0</td>
</tr>
<tr>
<td>17</td>
<td>4agp, 4agq, 4agq, 5a7b, 5aba</td>
<td>47</td>
<td>3g2z, 3g31, 4de1, 4de2</td>
</tr>
<tr>
<td>18</td>
<td>1ps3, 3d4z, 3d31, 3dx2, 3ej</td>
<td>48</td>
<td>3ui7, 3uo, 4lx, 4mnr, 4mrz, 4msn, 5c1w, 5c2, 5c2h</td>
</tr>
<tr>
<td>19</td>
<td>1pxn, 2vfd, 2xnb, 3px, 4eor</td>
<td>49</td>
<td>4bkt, 4w9c, 4w9h, 4w9i, 4w9j</td>
</tr>
<tr>
<td>20</td>
<td>1pg, 1mq6, 1z6e, 2xb, 2y5h</td>
<td>50</td>
<td>1yc1, 2xsd, 2yki, 3b27, 3br</td>
</tr>
<tr>
<td>21</td>
<td>3coy, 3coz, 3cvg, 4ddh, 4ddk</td>
<td>51</td>
<td>4eeq, 4f09, 4fpm, 4he, 4ja</td>
</tr>
<tr>
<td>22</td>
<td>2wn9, 2wnc, 2x00, 2xys, 2ymd</td>
<td>52</td>
<td>4cr9, 4cra, 4cr, 4ty7</td>
</tr>
<tr>
<td>23</td>
<td>1nc1, 1nc3, 1yer, 4f2w, 4f3c</td>
<td>53</td>
<td>3cq4, 3gw, 4e08, 4ih5, 4ih7</td>
</tr>
<tr>
<td>24</td>
<td>2fxs, 2wxx, 2v5, 2wer, 2yge</td>
<td>54</td>
<td>2p4y, 2yfe, 3b1m, 3fur, 3u9q</td>
</tr>
<tr>
<td>25</td>
<td>3gqy, 4m0y, 4q6, 4rfm</td>
<td>55</td>
<td>4kzq, 4ku</td>
</tr>
<tr>
<td>26</td>
<td>10wh, 1sqa, 3kgp</td>
<td>56</td>
<td>3p50, 3u5j, 4lzs, 4ogj, 4wiv</td>
</tr>
<tr>
<td>27</td>
<td>3qq, 3r88, 3wtp, 4gkm, 4owm</td>
<td>57</td>
<td>1q8t, 1q8u, 1ydr, 1ydt, 3ag9</td>
</tr>
<tr>
<td>28</td>
<td>1nvf, 2br1, 2brb, 3jvr, 3jvs</td>
<td>58</td>
<td>2xb8, 3n76, 3n7a, 3n8, 3ciw</td>
</tr>
<tr>
<td>29</td>
<td>29w, 3gr2, 3gv9, 4jx, 4kz6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Pipeline scripts

C.1 Rosetta relaxation

Listing C.1: thesis/appendix/make_ligand_pdb_params.sh

```bash
#!/bin/bash
# Input argument: A Mol2 file
mol=$(basename "$1")
filename=$(basename "$1" .mol2)
if [ ! -f $filename.params ]; then
    python2 $ROSETTA/main/source/scripts/python/public/molfile_to_params.py -n WER -p $filename --conformers=1 --keep-names --clobber $mol
fi
```

Listing C.2: thesis/appendix/make_complex_pdb.py

```python
#!/cluster/apps/python/3.6.0/x86_64/bin/python3
from prody import parsePDB, writePDB
from multiprocessing import Pool, cpu_count
from pathlib import Path
import numpy as np
import sys
import string
from htmd.molecule.molecule import Molecule
from htmd.molecule.voxeldescriptors import getVoxelDescriptors
from htmd.builder.preparation import proteinPrepare

def get_files(folder):
    for pdb in folder.glob('*/'):
        protein_pdb = pdb / f'{pdb.stem}.protein.pdb'
        ligand_pdb = pdb / f'{pdb.stem}.ligand.pdb'
        complex_pdb = pdb / f'{pdb.stem}.complex.pdb'
        if complex_pdb.exists():
            continue
        yield (protein_pdb, ligand_pdb, complex_pdb)
```
def make_complex_pdb(protein_pdb, ligand_pdb, complex_pdb):
    if complex_pdb.exists():
        print(complex_pdb, "exists")
        return
    protein = parsePDB(str(protein_pdb))
    ligand = parsePDB(str(ligand_pdb))
    p_chains = protein.getChids()
    individual_chains = set(p_chains)
    possible_chains = string.ascii_uppercase.translate(str.maketrans("WXZ","wxz")) + string.digits + string.ascii_lowercase.translate(str.maketrans({"w":"","x":"","z":""}))
    chain_dict = dict(zip(individual_chains, possible_chains[:len(individual_chains)]))
    protein.setChids(np.vectorize(chain_dict.get)(p_chains))
    ligand.setResnames(np.array(['W E R'] * ligand.numAtoms()))
    ligand.setChids(np.array(['X'] * ligand.numAtoms()))
    complex = protein + ligand
    res = complex.getResnames()
    res[res=='H O H'] = 'W A T'
    res[res=='C Y X'] = 'C Y S'
    res[res=='C Y M'] = 'C Y S'
    res[res=='H I E'] = 'H I S'
    res[res=='H I D'] = 'H I S'
    res[res=='H E D'] = 'H I S'
    res[res=='H I P'] = 'H I S'
    res[res=='T R Q'] = 'T R P'
    res[res=='K C X'] = 'L Y S'
    res[res=='L L P'] = 'L Y S'
    res[res=='A R N'] = 'A R G'
    res[res=='A S H'] = 'A S P'
    res[res=='G L H'] = 'G L U'
    res[res=='L Y N'] = 'L Y S'
    res[res=='A R O'] = 'A R G'
    res[res=='H S E'] = 'S E R'
    chain = complex.getChids()
    chain[chain=='W A T'] = 'W'
    for metal in ['MN', 'MG', 'ZN', 'CA', 'NA']:
        chain[chain==metal] = 'Z'
    complex.setResnames(res)
    complex.setChids(chain)
    writePDB(str(complex_pdb), complex)
    complex = Molecule(str(complex_pdb))
    prot = complex.copy()
    prot.filter("protein")
    lig = complex.copy()
    lig.filter("not protein and same residue as ((rename WAT and within 3 of rename WER and within 3 of protein) or (rename MN MG ZN CA NA and within 5 of rename WER) or rename WER)")
    prot = proteinPrepare(prot, pH=7.0)
    mol = Molecule(name="complex")
    mol.append(prot)
    mol.append(lig)
    mol.write(str(complex_pdb))
    complex = parsePDB(str(complex_pdb))
C.1. Rosetta relaxation

```python
res = complex.getResnames()
res[res=="HOH"] = 'WAT'
res[res=="CYX"] = 'CYS'
res[res=="CYM"] = 'CYS'
res[res=="HIE"] = 'HIS'
res[res=="HID"] = 'HIS'
res[res=="HSD"] = 'HIS'
res[res=="HIP"] = 'HIS'
res[res=="TRQ"] = 'TRP'
res[res=="KCX"] = 'LYS'
res[res=="LLP"] = 'LYS'
res[res=="ARN"] = 'ARG'
res[res=="ASH"] = 'ASP'
res[res=="GLH"] = 'GLU'
res[res=="LYN"] = 'LYS'
res[res=="AR0"] = 'ARG'
res[res=="HSE"] = 'SER'
complex.setResnames(res)
writePDB(str(complex.pdb), complex)

def process(args):
    try:
        make_complex_pdb(*args)
    except Exception as e:
        print(e)
    return False
def params(root, pdb):
    protein_pdb = root/ pdb / (pdb+"protein.pdb")
    ligand_pdb = root/ pdb / (pdb+"ligand.pdb")
    complex_pdb = root/ pdb / (pdb+"complex.pdb")
    return protein_pdb, ligand_pdb, complex_pdb

if __name__ == "__main__":
    parent_folder = #Set root folder for complex folders
    p = Pool(cpu_count())
    p.map(process, get_files(parent_folder))
```

Listing C.3: thesis/appended/minimize_rosetta.sh

```bash
#!/bin/bash
root= #Set root path for complex folders
rosetta=$ROSETTA/main/source/bin/rosetta_scripts.static.linuxgccrelease
rosettadb=$ROSETTA/main/database/
script_folder=$(pwd)
export rosetta
export rosettadb
export script_folder

function make_rosetta() {
    folder=$(dirname "$1")
    cd $(dirname "$1")
```
C. Pipeline scripts

if [[ $? -ne 0 ]]; then
  exit 1
fi
if [ ! -f "processed" ]; then
touch "processed"
  echo "######################################################## $(basename $folder)"
  complex=$(basename "$1") name=$(basename "$1" .pdb) params="${name}/complex/ligand}.params" envsubst < $script_folder/flags_relax.txt > "flags_relax.txt"
  echo $folder
  python3 $script_folder/get_closest_lig_atom.py ./$(basename "$1") ./constraints ($rosetta @flags_relax.txt -parser:protocol "$script_folder/relax.xml" -database $rosettadb ) & & echo "Finished $(basename $folder) correctly"
fi
export -f make_rosetta
find $root -name "*.complex.pdb" | shuf | parallel -j 24 -n 1 --ungroup bash -c "& & make_rosetta {}"

Listing C.4: thesis/appendix/make_protein_pdb.py

from prody import parsePDB, writePDB
from multiprocessing import Pool
from pathlib import Path
import numpy as np
import sys

def get_files(folder):
  for complex_pdb in folder.glob('*/complex_*.pdb'):
    protein_pdb = complex_pdb.parent / complex_pdb.name.replace('complex', 'protein')
    if not protein_pdb.exists():
      yield complex_pdb

def make_complex_pdb(complex_pdb):
  complex = parsePDB(str(complex_pdb))
  protein_pdb = complex_pdb.parent / complex_pdb.name.replace('complex', 'protein')
  protein = complex.select('not resname WER')
  writePDB(str(protein_pdb), protein)

def process(args):
  print(args)
  try:
    make_complex_pdb(args)
  except Exception as e:
    print(e)
  return False

return True
C.1. Rosetta relaxation

```python
if '__name__' == '_main_':
    root = '# Set root path for complex folders
p = Pool(48)
p.map(process, get_files(parent_folder))
```

Listing C.5: thesis/appendix/make_ligand_mol2_renamed.py

```python
import pdb
from pathlib import Path
from multiprocessing import Pool

def read_pdb(pdb_path):
    with pdb_path.open('r') as f:
        lines = f.readlines()
    hetatm = filter(lambda x: x.startswith('HETATM'), lines)
    atom_num_name = map(lambda x: x.split()[1:3], hetatm)
    return dict(atom_num_name)

def read_mol2(mol2_path, name_map):
    with mol2_path.open('r') as f:
        lines = f.readlines()
    mode = "search"
    for i, line in enumerate(lines):
        if mode == "search":
            if line.startswith('@<TRIPOS>ATOM'):
                mode = "rename"
        elif mode == "rename":
            if line.startswith('@<TRIPOS>BOND'):
                mode = "end"
            else:
                atom_num, atom_name = line.split()[0:2]
                new_name = name_map[atom_num].ljust(len(atom_name))
                position = line.find(atom_name)
                end = position + len(new_name)
                new_line = line[0:position] + new_name + line[end:]
                lines[i] = new_line
        elif mode == "end":
            return lines

def process_file(path):
    pdb_code = path.stem
    new_mol2_path = path / f'{pdb_code}_ligand_renamed.mol2'
    try:
        name_map = read_pdb(path / f'{pdb_code}_ligand.pdb')
        with new_mol2_path.open('w') as f:
            f.write(''.join(read_mol2(path / f'{pdb_code}_ligand.mol2', name_map)))
    except:
        print(e)

def get_files(path):
    return path.glob('*/')
```

65
C. Pipeline scripts

```python
if __name__ == "__main__":
    p = Pool(48)
    root=# Set root path for complex folders
    p.map(process_file, get_files(root))
```

Listing C.6: thesis/appendix/make_ligand_mol2.sh

```bash
#!/bin/bash
root= # Set root path for complex folders
script=$ROSETTA/main/source/src/apps/public/ligand_docking/
pdb_to_molfile.py
export script

function make_ligand_mol2 () {
    dir=$(dirname $1)
    complex=$1
    name=$(basename "$1" .pdb)
    code=$(echo $name | cut -f 1 -d " ")
    mol="$dir/$code_ligand_renamed.mol2"
    out=$dir/$name/complex/ligand_renamed.mol2
    python2 $script $mol $complex > $dir/$name/complex/ligand_renamed.mol2
}
export -f make_ligand_mol2

find ${root} -name "*.complex.*.pdb" -maxdepth 2 | parallel -n 1
bash -c " & & make_ligand_mol2 {}"
```

Listing C.7: thesis/appendix/flags_relax.txt

```bash
-in
  -file
  -s ./complex
  -extra_res_fa ./$params
  -packing
  -ex1
  -exlaro
  -ex2
  -mute core.util.prof ## dont show timing info
  -mute core.io.database
  # -mute all
  # -unmute protocols.jd2.JobDistributor
  -constraints:csf_file ./constraints
  -score:set_weights atom_pair_constraint 1.0
  -in: auto_setup_metal
  -in: metals_angle_constraint_multiplier 3.0
  -in: metals_distance_constraint_multiplier 3.0
  -in: ignore_waters false
  -ignore_zero_occupancy false
  -keep_input_protonation_state true
  -nstruct 10
  -overwrite
```

Listing C.8: thesis/appendix/get_constraint.py

66
from prody import *
import numpy as np
import sys

def process(file, output):
    complex = parsePDB(file)
    metals = complex.select("chain Z")
    # import pdb; pdb.set_trace()
    result = []
    if metals:
        for atom in metals:
            pos = atom.getCoords()
            close_ligand = complex.select("rename WER and (not (element H or element C)) and within 2.5 of t", t=pos)
            if close_ligand:
                # close = close_ligand[np.argmin(np.linalg.norm(close_ligand.getCoords() - pos, axis=1))]
                for close in close_ligand:
                    result.append((atom.getName(), str(atom.getResnum())+atom.getChid(), close.getName(), str(close.getResnum())+close.getChid()))
    with open(output, "w") as f:
        for r in result:
            f.write(f"AtomPair {{r[0]} \{r[1]} \{r[2]} \{r[3]} SQUARE WELL 2.5 \-2000\n")

if __name__ == "__main__":
    process(sys.argv[1], sys.argv[2])

Listing C.9: thesis/appendix/relax.xml

```xml
<ROSETTASCRIPTS>
  <SCOREFXNS>
  </SCOREFXNS>
  </FILTERS>
  </FILTERS>
  <TASKOPERATIONS>
    <DetectProteinLigandInterface name="ligandInterface" cut1="0.0" cut2="0.0" cut3="16.0" cut4="20.0"
     design="0" catres_interface="0" catres_only_interface="0"
     arg_sweep_interface="0"/>
  </TASKOPERATIONS>
  <MOVERS>
    <ConstraintSetMover name="constraint" add_constraints="true"
    cst_file="/constraints"/>
    <EnzRepackMinimize name="relax" scorefxn_repack="REF2015"
    scorefxn_minimize="REF2015" csopt="0" design="0" repack_only="1"
    fix_catalytic="0" minimize_RB="1" minimize_BB="1"
    minimize_sc="1" minimize_lig="1" min_in_stages="0" backrub="0"
    cycles="1" task_operations="ligandInterface"/>
  </MOVERS>
  <PROTOCOLS>
    <Add mover name="constraint"/>
  </PROTOCOLS>
</ROSETTASCRIPTS>
```
C. Pipeline scripts

<Add mover_name="relax"/>
</PROTOCOLS>
</ROSETTASCRIPTS>

C.2 Precompute Rosetta energies

Listing C.10: thesis/appendix/compute_rosetta_energy.py

```python
#!/usr/bin/env python3
from __future__ import print_function
import argparse
import os
import h5py
from pyrosetta.rosetta.protocols.scoring import Interface
from pyrosetta.rosetta import *
from pyrosetta import *
from pathlib import Path
import numpy as np
from multiprocessing import Pool, cpu_count
from collections import defaultdict
from pyrosetta.toolbox.atom_pair_energy import print_residue_pair_energies
init('--in: auto_setup_metals')  # mute core.conformation.Conformation

def compute_atom_pair_energy(pdb_filename, ligand_params, interface_cutoff = 21.0):
    if type(ligand_params) is str:
        ligand_params = [ligand_params]
    ligand_params = Vector1([str(ligand_params)])
    pose = Pose()
    res_set = pose.conformation().modifiable_residue_type_set_for_conf()
    res_set.read_files_for_base_residue_types(ligand_params)
    pose.conformation().reset_residue_type_set_for_conf(res_set)
    pose_from_file(pose, str(pdb_filename))
    scorefxn = create_score_function('ref2015')
    pose_score = scorefxn(pose)
    #detect interface
    fold_tree = pose.fold_tree()
    for jump in range(1, pose.num_jump()+1):
        name = pose.residue(fold_tree.downstream_jump_residue(jump)).name()
        if name == 'WER':
            break
    interface = Interface(jump)
    interface.distance(interface_cutoff)
    interface.calculate(pose)
```
C.2. Precompute Rosetta energies

```python
energies = []
en = defaultdict(lambda:np.zeros((1,4)))
keys = []
for rnum1 in range(1, pose.total_residue() + 1):
    if interface.is_interface(rnum1):
        r1 = pose.residue(rnum1)
        for a1 in range(1, len(r1.atoms()) + 1):
            seq1 = pose.pdb_info().pose2pdb(rnum1).strip().replace(' ','-')
            at1 = r1.atom_name(a1).strip()
            key1 = seq1 + '-'+at1
            for rnum2 in range(rnum1+1, pose.total_residue() + 1):
                if interface.is_interface(rnum2):
                    r2 = pose.residue(rnum2)
                    for a2 in range(1, len(r2.atoms())+1):
                        seq2 = pose.pdb_info().pose2pdb(rnum2).strip().replace(' ','-')
                        at2 = r2.atom_name(a2).strip()
                        key2 = seq2 + '-'+at2
                        ee = establish_atom_pair_energies(r1, a1, r2, a2, scorefxn)
                        if all(e == 0.0 for e in ee):
                            continue
                        en[key1] += np.array(ee)
                        en[key2] += np.array(ee)

energy_matrix = np.array([v for v in en.values()])
return list(en.keys()), energy_matrix
```

```python
def get_radii_and_charges(pdb_filename, ligand_params):
    keys = []
    charges = []
    radii = []

    if type(ligand_params) is str:
        ligand_params = [ligand_params]
    ligand_params = Vector1([str(ligand_params)])

    pose = Pose()
    res_set = pose.conformation().modifiable_residue_type_set_for_conf()
    res_set.read_files_for_base_residue_types(ligand_params)
    pose.conformation().reset_residue_type_set_for_conf(res_set)
    pose_from_file(pose, str(pdb_filename))
    for rnum1 in range(1, pose.total_residue() + 1):
        r1 = pose.residue(rnum1)
        for a1 in range(1, len(r1.atoms()) + 1):
            seq1 = pose.pdb_info().pose2pdb(rnum1).strip().replace(' ','-')
            at1 = r1.atom_name(a1).strip()
            key1 = seq1 + '-'+at1
            charges.append(r1.atomic_charge(a1))
            radii.append(r1.atomic_typ(a1).lj_radius())
```

69
keys.append(key1)

return keys, charges, radii

def extract_and_save(pdb_file):
    folder = pdb_file.parent
    pdb_code = folder.stem
    ligand_params = folder / f'{pdb_code}_ligand.params'
    output_file = folder / (pdb_file.stem + '.attr')
    try:
        e_keys, e_values = compute_atom_pair_energy(pdb_file, ligand_params)
        rc_keys, charges, radii = get_radii_and_charges(pdb_file, ligand_params)
    except Exception as e:
        print(f'Error at {pdb_file}
        print(e)
        return

    energy_keys = np.array(e_keys)
    energy_values = np.array(e_values)
    rc_keys = np.array(rc_keys)
    radius_values = np.array(radii)
    charge_values = np.array(charges)
    np.savez_compressed(str(output_file),
                        energy_keys=energy_keys,
                        energy_values=energy_values,
                        rc_keys=rc_keys,
                        radius_values=radius_values,
                        charge_values=charge_values)

    print(f'COMPLETED{pdb_file}

def update_radii_charges(pdb_file):
    folder = pdb_file.parent
    pdb_code = folder.stem
    print(pdb_code)
    ligand_params = folder / f'{pdb_code}_ligand.params'
    output_file = folder / (pdb_file.stem + '.attr')
    rc_keys, charges, radii = get_radii_and_charges(pdb_file, ligand_params)
    rc_keys = np.array(rc_keys)
    radius_values = np.array(radii)
    charge_values = np.array(charges)
    old_data = np.load(str(output_file) + '.npz')
    e_keys = old_data['energy_keys']
    energy_values = old_data['energy_values']
    np.savez_compressed(str(output_file),
                        energy_keys=e_keys,
                        energy_values=energy_values,
                        rc_keys=rc_keys,
                        radius_values=radius_values,
                        charge_values=charge_values)
C.3. CNN Architectures

Listing C.11: thesis/appenix/original_kdeep.py

```python
import tensorflow as tf

LEARNING_RATE = 0.0001

def fire_module(net, squeeze, expand, training):
    net = tf.layers.conv3d(net, squeeze, [1, 1, 1], activation=tf.nn.relu)
    net1 = tf.layers.conv3d(net, expand, [1, 1, 1], activation=tf.nn.relu)
    net2 = tf.layers.conv3d(net, expand, [3, 3, 3], padding='same',
                            activation=tf.nn.relu)
    return tf.concat(axis=-1, values=[net1, net2])

def conv_net(X, reuse, training):
    with tf.variable_scope('SqueezeNet', reuse=reuse):
        print(X)
        net = tf.layers.conv3d(X, 96, 1, 2, padding='same', activation=tf.nn.relu)
        print(net)
        net = fire_module(net, 16, 64, training=training)
        net = fire_module(net, 16, 64, training=training)
        net = fire_module(net, 32, 128, training=training)
        net = tf.layers.max_pooling3d(net, 3, 2)
        print(net)
        net = fire_module(net, 32, 128, training=training)
        net = fire_module(net, 48, 192, training=training)
        net = fire_module(net, 48, 192, training=training)
        net = fire_module(net, 64, 256, training=training)
        net = tf.layers.average_pooling3d(net, 3, 2)
        net = tf.layers.flatten(net)
        net = tf.layers.dense(net, 1)
        return net

def model_fn(features, labels, mode, rotate):
    training = mode == tf.estimator.ModeKeys.TRAIN
    ```
predictions = conv_net(features, reuse=tf.AUTO_REUSE, training=training)
loss = None
train_op = None
if training:
    loss = tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
else:
    if rotate:
        predictions = tf.reduce_mean(tf.reshape(predictions, (-1, 24, 1)), axis=1)
        loss = tf.losses.mean_squared_error(labels, predictions)
    else:
        loss = tf.losses.mean_squared_error(labels=labels, predictions=predictions)
return tf.estimator.EstimatorSpec(
    mode=mode,
    predictions=predictions,
    loss=loss,
    train_op=train_op)

Listing C.12: thesis/appendix/large_kdeep.py

import tensorflow as tf

LEARNING_RATE = 0.0001

def fire_module(net, squeeze, expand, training):
    net = tf.layers.conv3d(net, squeeze, [1, 1, 1], activation=tf.nn.relu)
    net1 = tf.layers.conv3d(net, expand, [1, 1, 1], activation=tf.nn.relu)
    net2 = tf.layers.conv3d(net, expand, [3, 3, 3], padding='same',
                            activation=tf.nn.relu)
    return tf.concat(axis=-1, values=[net1, net2])

def conv_net(X, reuse, training):
    with tf.variable_scope('SqueezeNet', reuse=reuse):
        print(X)
        net = tf.layers.conv3d(X, 96, 7, 2, padding='same', activation=tf.nn.relu)
        print(net)
        net = fire_module(net, 16, 64, training=training)
        net = fire_module(net, 16, 64, training=training)
        net = fire_module(net, 32, 128, training=training)
        net = tf.layers.max_pooling3d(net, 3, 2)
        print(net)
        net = fire_module(net, 32, 128, training=training)
        net = fire_module(net, 48, 192, training=training)
C.3. CNN Architectures

Listing C.13: thesis/appendix/resnet_101.py

```python
import tensorflow as tf

LEARNING_RATE = 0.0001

def start_block(net, channels, training):
    input_net = tf.layers.conv3d(net, 4*channels, 1, 2, padding='same')
    net = conv3bn(net, channels, 1, training, stride=2)
    net = conv3bn(net, channels, 3, training)
    output_net = net + input_net
    return output_net

def inner_block(net, channels, training):
    input_net = tf.layers.conv3d(net, 4*channels, 1, padding='same')
    net = conv3bn(net, channels, 1, training)
    return output_net
```

```python
net = fire_module(net, 48, 192, training=training)
net = fire_module(net, 64, 256, training=training)
net = tf.layers.average_pooling3d(net, 3, 2)
net = tf.layers.flatten(net)
net = tf.layers.dense(net, 1)
return net

def model_fn(features, labels, mode, rotate):
    training = mode == tf.estimator.ModeKeys.TRAIN
    predictions = conv_net(features, reuse=tf.AUTO_REUSE, training=training)
    loss = None
    train_op = None
    if training:
        loss = tf.losses.mean_squared_error(labels=labels, predictions=predictions)
        optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE)
        train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    else:
        if rotate:
            predictions = tf.reduce_mean(tf.reshape(predictions, [-1, 24, 1]), axis=1)
        else:
            loss = tf.losses.mean_squared_error(labels, predictions)

    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op)
```

Listing C.13: thesis/appendix/resnet_101.py

```python
import tensorflow as tf

LEARNING_RATE = 0.0001

def start_block(net, channels, training):
    input_net = tf.layers.conv3d(net, 4*channels, 1, 2, padding='same')
    net = conv3bn(net, channels, 1, training, stride=2)
    net = conv3bn(net, channels, 3, training)
    output_net = net + input_net
    return output_net

def inner_block(net, channels, training):
    input_net = tf.layers.conv3d(net, 4*channels, 1, padding='same')
    net = conv3bn(net, channels, 1, training)
    return output_net
```
C. Pipeline scripts

```python
net = conv3bn(net, channels, 3, training)
net = conv3bn(net, 4*channels, 1, training)
output_net = net + input_net
return output_net

def conv3bn(net, channels, filt, training, stride=1):
    #net = tf.layers.batch_normalization(net, training=training)
    net = tf.nn.relu(net)
    net = tf.layers.conv3d(net, channels, filt, stride, padding='same')
    return net

def conv_net(X, reuse, training):
    with tf.variable_scope('ResNet', reuse=reuse):
        layers = [3,4,23,3]
        k = 64
        net = tf.layers.conv3d(X, k, 7, 2, padding='same')
        #net = tf.layers.batch_normalization(net, training=training)
        net = tf.nn.relu(net)
        net = tf.layers.max_pooling3d(net, 3, 2)
        for i in range(0, layers[0]):
            net = inner_block(net, k, training)
        for i, l in enumerate(layers[1:], 1):
            for j in range(0, l-1):
                net = inner_block(net, k*(2**i), training)
            net = tf.reduce_mean(net, axis=(1,2,3))
        net = tf.layers.flatten(net)
        net = tf.layers.dense(net, 1)
        return net

def model_fn(features, labels, mode, rotate):
    training = mode == tf.estimator.ModeKeys.TRAIN
    predictions = conv_net(features, reuse=tf.AUTO_REUSE, training=training)

    loss = None
    train_op = None
    if training:
        loss = tf.losses.mean_squared_error(labels=labels, predictions=predictions)
        optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE)
        update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
        with tf.control_dependencies(update_ops):
            train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    else:
        if rotate:
            predictions = tf.reduce_mean(tf.reshape(predictions, (1,-1,1,24,1)), axis=1)
            loss = tf.losses.mean_squared_error(labels, predictions)
        else:
```
### C.4 Feature generation

Listing C.14: thesis/appendix/make_hfmd_features.py

```python
from pathlib import Path
import numpy as np
from collections import defaultdict
import h5py
from htmd.molecule.molecule import Molecule
from htmd.molecule.voxelDescriptors import getVoxelDescriptors
from multiprocessing import Pool, cpu_count
from htmd.builder.preparation import proteinPrepare

def save_grid(saving_path, pdb, image):
    with h5py.File(str(saving_path), "w", libver='latest') as f:
        f.create_dataset("grid", dtype='f4', data=image)

def build_images(protein_file, ligand_file, size):
    protein = Molecule(str(protein_file))
    ligand = Molecule(str(ligand_file))
    protein.filter("not (water or name CO or name NI or name CU or name NA)"
                   
                   #protein.filter('not name HG and not name CD and not name K and
                   #not name CU and not name SE and not name LI and not name NI and
                   #not name CO and not name CS and not name SR and not name MN and
                   #not name NA')
    center = np.mean(ligand.get('coords'), axis=0)
    ligand.moveBy(-center)
    protein.moveBy(-center)
    size_ang = 12
    center = np.mean(ligand.get('coords'), axis=0)
    ex_min = center - size_ang
    ex_max = center + size_ang
    x = np.linspace(ex_min[0], ex_max[0], size)
    y = np.linspace(ex_min[1], ex_max[1], size)
    z = np.linspace(ex_min[2], ex_max[2], size)
    position_matrix = np.stack(np.meshgrid(x, y, z, indexing='ij'),
                               axis=-1).reshape((-1,3))
    prot_features = getVoxelDescriptors(protein, usercenters=position_matrix)[0].reshape(3*(size,) + (-1,))
    inh_features = getVoxelDescriptors(ligand, usercenters=position_matrix)[0].reshape(3*(size,) + (-1,))
    final_grid = np.concatenate((prot_features, inh_features), axis=-1)
    #import pdb; pdb.set_trace()
```
C. Pipeline scripts

```python
return final_grid

def get_files(folder):
    for protein_file in folder.glob('*/_protein_0*.pdbqt'):
        pdb = protein_file.parent
        try:
            ligand_file = next(pdb.glob('*_ligand_*_.pdbqt'))
        except:
            continue
        yield (pdb.stem, try:
        image = build_images(prot, lig, size)
        except Exception as e:
            print(e)
            return False
        save_grid(saving_path, pdb, image)
        return True

if __name__ == '__main__':
    size = 25
    root = # Set root path for complex folders
    saving_folder = # Saving HDF5 folder
    saving_folder.mkdir(parents=True, exist_ok=True)
    p = Pool(cpu_count())
    p.map(process_file, get_files(root))
    m, protein_file, ligand_file

    def process_file(params):
        pdb, prot, lig = params
        saving_path = saving_folder / (prot.name.replace('.pdbqt', '.hdf5')
        .replace('protein', 'complex'))
        if saving_path.exists():
            return True
        print(pdb)

    from operator import itemgetter
    from prody import parsePDB, calcCenter, parseDCD, moveAtoms
    import itertools
    from apbs import APBS
    from collections import defaultdict
    from multiprocessing import Pool, cpu_count
    from pathlib import Path
    import numpy as np
    from collections import defaultdict
    import h5py
    from scipy.interpolate import Rbf
    from scipy.spatial.distance import cdist

    def save_grid(saving_path, image):
        with h5py.File(str(saving_path), "w", libver='latest') as f:
            f.create_dataset("grid", dtype='f4', data=image)

    def get_files(folder):
```

Listing C.15: thesis/appendix/make_rosetta_features.py
C.4. Feature generation

```python
print(folder)
return folder.glob('*/*.complex_*_.pdb')

def import_data(pdb_path):
data_path = str(pdb_path).replace('.pdb', '.attr.npz')data = np.load(data_path)radii = dict(zip(data['rc_keys'], data['radius_values']))charges = dict(zip(data['rc_keys'], data['charge_values']))energies = data['energy_values'].squeeze()energy_keys = data['energy_keys']
return radii, charges, energy_keys, energies

from scipy.spatial import KDTree
def apply_filter(filter_type, points, values, targets, radii):
c = 0
mask = np.linalg.norm(points, axis=-1) <= 12.5*np.sqrt(3)
points = points[mask]
values = values[mask, :]
radii = radii[mask]
dists = cdist(points, targets)
aux = np.where(dists < 5, filter_type(dists, radii), 0)
delete dists
import pdb; pdb.set_trace()
result = np.array([values[np.argmax(aux, axis=0), i] * np.max(aux, axis=0) for i in range(values.shape[-1])])
return result

def interpolate(int_type, points, values, targets):
mask = np.linalg.norm(points, axis=-1) <= 12.5*np.sqrt(3)
points = points[mask]
values = values[mask, :]
import pdb; pdb.set_trace()
points_x, points_y, points_z = [c.flatten() for c in np.split(points, 3, axis=1)]
targets_x, targets_y, targets_z = [c.flatten() for c in np.split(targets, 3, axis=1)]
res = np.stack([Rbf(points_x, points_y, points_z, values[..., i], function=int_type')(targets_x, targets_y, targets_z) for i in range(values.shape[-1])], axis=-1)
return res

GRIDS = ['fa_atr', 'fa_rep', 'fa_sol', 'fa_elec']
def grid_around(center, size, spacing=1.0):
size_ang = ((size - 1) / 2.) * spacing
ex_min = center - size_ang
ex_max = center + size_ang

x = np.linspace(ex_min[0], ex_max[0], size)
y = np.linspace(ex_min[1], ex_max[1], size)
z = np.linspace(ex_min[2], ex_max[2], size)
return np.stack(np.meshgrid(x, y, z, indexing='ij'), axis=-1)
```
C. Pipeline scripts

```python
def get_keys(pdb):
    resnums = pdb.getResnums()
    chids = pdb.getChids()
    names = pdb.getNames()
    keys = np.char.add(np.char.mod('%s−', np.char.replace(np.char.add(np.char.mod('%s ', resnums), chids), ' ')), names)
    return keys

def build_images(pdb_path, size, interpolation_mode):
    complex = parsePDB(str(pdb_path))
    protein = complex.select("not (resname WER or water)")
    ligand = complex.select("resname WER")
    center = calcCenter(ligand.getCoords())
    moveAtoms(complex, by=-center)
    center = calcCenter(complex.select("resname WER"), getCoords())
    size_ang = ((size - 1) / 2.) * (24/(size - 1))
    ex_min = center - size_ang
    ex_max = center + size_ang
    position_matrix = grid_around(center, size, spacing=24/(size - 1))
    radii, charges, energy_keys, energies = import_data(pdb_path)
    coordinates = complex.getCoords()
    try:
        keys = get_keys(complex)
    except:
        print('FAILED AT KEYS', pdb_path)
        raise
    keys_p = get_keys(protein)
    keys_l = get_keys(ligand)
    key_map = dict(zip(keys, range(len(keys))))
    energy_ids = [i for i, x in enumerate(energy_keys) if x in key_map]
    energy_ids_p = [i for i, x in enumerate(energy_keys) if x in key_map and x in keys_p]
    energy_ids_l = [i for i, x in enumerate(energy_keys) if x in key_map and x in keys_l]
    energy_coordinates = np.array([coordinates[key_map[x]] for x in energy_keys if x in key_map])
    energy_coordinates_p = np.array([coordinates[key_map[x]] for x in energy_keys if x in key_map and x in keys_p])
    energy_coordinates_l = np.array([coordinates[key_map[x]] for x in energy_keys if x in key_map and x in keys_l])
    radii_c = np.array([radii[x] for x in energy_keys if x in key_map])
    radii_p = np.array([radii[x] for x in energy_keys if x in key_map and x in keys_p])
    radii_l = np.array([radii[x] for x in energy_keys if x in key_map and x in keys_l])
```

C.4. Feature generation

```python
image = {}
image_p = {}
image_l = {}
for field in GRIDS:
    image[field] = np.zeros(3 * (size, ))
    image_p[field] = np.zeros(3 * (size, ))
    image_l[field] = np.zeros(3 * (size, ))
#image[k] = apply_filter(filter_mode, energy_coordinates, energies
#    [energy_ids, i], position_matrix.reshape((-1,3)), radii_c).reshape
#    (3*(size,) + (-1,))
rosetta_p = apply_filter(filter_mode, energy_coordinates_p,
    energies[energy_ids_p, :], position_matrix.reshape((-1,3)),
    radii_p).reshape(3*(size,) + (-1,))
rosetta_l = apply_filter(filter_mode, energy_coordinates_l,
    energies[energy_ids_l, :], position_matrix.reshape((-1,3)),
    radii_l).reshape(3*(size,) + (-1,))
#rosetta_p = interpolate(interpolation_mode, energy_coordinates_p,
#    energies[energy_ids_p, :], position_matrix.reshape((-1,3))).
#    reshape(3*(size,) + (-1,))
#rosetta_l = interpolate(interpolation_mode, energy_coordinates_l,
#    energies[energy_ids_l, :], position_matrix.reshape((-1,3))).
#    reshape(3*(size,) + (-1,))
channel_max = np.array([-1.97975219e-02, 4.11150004e+02,
    4.01129569e+00, 1.28136470e+00, -0.04185846, 0.59311066,
    2.67294434, 0.40072521])
channel_min = np.array([-2.27475644, 0., -0.37756078,
    -3.79981632, -1.7244696, 0., -0.44943017, -2.00621753])
rosetta_atr_p = np.clip(rosetta_p[...,:], channel_min[0], 0) /
    channel_max[0]
rosetta_rep_p = np.clip(rosetta_p[...,:], 0, channel_max[1]) /
    channel_max[1]
rosetta_sol_p_pos = np.clip(rosetta_p[...,:], 0, channel_max[2]) /
    channel_max[2]
rosetta_elec_p_pos = np.clip(rosetta_p[...,:], 0, channel_max[3]) /
    channel_max[3]
rosetta_sol_p_neg = np.clip(rosetta_p[...,:], channel_min[2], 0) /
    channel_min[2]
rosetta_elec_p_neg = np.clip(rosetta_p[...,:], channel_min[3], 0) /
    channel_min[3]
rosetta_atr_l = np.clip(rosetta_l[...,:], channel_min[4], 0) /
    channel_min[4]
rosetta_rel_l = np.clip(rosetta_l[...,:], 0, channel_max[5]) /
    channel_max[5]
rosetta_sol_l_pos = np.clip(rosetta_l[...,:], 0, channel_max[6]) /
    channel_max[6]
rosetta_elec_l_pos = np.clip(rosetta_l[...,:], 0, channel_max[7]) /
    channel_max[7]
rosetta_sol_l_neg = np.clip(rosetta_l[...,:], channel_min[6], 0) /
    channel_min[6]
rosetta_elec_l_neg = np.clip(rosetta_l[...,:], channel_min[7], 0) /
    channel_min[7]
rosetta = np.stack((rosetta_atr_p, rosetta_rep_p,
    rosetta_sol_p_pos, rosetta_elec_p_pos, rosetta_sol_p_neg,
    rosetta_atr_l, rosetta_rel_l, rosetta_sol_l_pos,
    rosetta_elec_l_pos, rosetta_sol_l_neg, rosetta_elec_l_neg),
    axis=-1)
```
C. Pipeline scripts

Listing C.16: thesis/appendix/make_electroneg_features.py

```python
from operator import itemgetter
from prody import parsePDB, calcCenter, parseDCD, moveAtoms
from itertools import
from apbs import APBS
from collections import defaultdict
from multiprocessing import Pool
from pathlib import Path
from numpy import as np
from collections import defaultdict
from h5py import
from scipy.interpolate import Rbf
```

```python
    def exp_12(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = rvdw[:, None] / r
        ret = np.where(r == 0, 1, 1 - np.exp(-(rr)**12))
        return ret
    
    def gaussian(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = r / rvdw[:, None]
        ret = np.exp(-(rr)**2)
        return ret
    
    if __name__ == "__main__":
        size = 25
        # interpolation_mode = "thin_plate"
        filter_mode = exp_12
        root= # Set root path for complex folders
        saving_folder = # Path for HDF5 storage
        saving_folder.mkdir(parents=True, exist_ok=True)
        p = Pool(cpu_count())
        p.map(process_file, get_files(root))
```

```python
    def process_file(file):
        saving_path = saving_folder / (file.name.replace('.pdb', '.hdf5'))
        print(saving_path)
        if saving_path.exists():
            return True
        try:
            image = build_images(file, size, interpolation_mode)
            print(file)
            print(e)
            return False
        except Exception as e:
            save_grid(saving_path, image)
            return True
    
    def exp_12(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = rvdw[:, None] / r
        ret = np.where(r == 0, 1, 1 - np.exp(-(rr)**12))
        return ret
    
    def gaussian(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = r / rvdw[:, None]
        ret = np.exp(-(rr)**2)
        return ret
    
    if __name__ == "__main__":
        size = 25
        # interpolation_mode = "thin_plate"
        filter_mode = exp_12
        root= # Set root path for complex folders
        saving_folder = # Path for HDF5 storage
        saving_folder.mkdir(parents=True, exist_ok=True)
        p = Pool(cpu_count())
        p.map(process_file, get_files(root))
```

```python
    def process_file(file):
        saving_path = saving_folder / (file.name.replace('.pdb', '.hdf5'))
        print(saving_path)
        if saving_path.exists():
            return True
        try:
            image = build_images(file, size, interpolation_mode)
            print(file)
            print(e)
            return False
        except Exception as e:
            save_grid(saving_path, image)
            return True
    
    def exp_12(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = rvdw[:, None] / r
        ret = np.where(r == 0, 1, 1 - np.exp(-(rr)**12))
        return ret
    
    def gaussian(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = r / rvdw[:, None]
        ret = np.exp(-(rr)**2)
        return ret
    
    if __name__ == "__main__":
        size = 25
        # interpolation_mode = "thin_plate"
        filter_mode = exp_12
        root= # Set root path for complex folders
        saving_folder = # Path for HDF5 storage
        saving_folder.mkdir(parents=True, exist_ok=True)
        p = Pool(cpu_count())
        p.map(process_file, get_files(root))
```

Listing C.16: thesis/appendix/make_electroneg_features.py

```python
from operator import itemgetter
from prody import parsePDB, calcCenter, parseDCD, moveAtoms
from itertools import
from apbs import APBS
from collections import defaultdict
from multiprocessing import Pool
from pathlib import Path
from numpy import as np
from collections import defaultdict
from h5py import
from scipy.interpolate import Rbf
```

```python
    def exp_12(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = rvdw[:, None] / r
        ret = np.where(r == 0, 1, 1 - np.exp(-(rr)**12))
        return ret
    
    def gaussian(r, rvdw):
        rvdw = rvdw.reshape((-1,))
        rr = r / rvdw[:, None]
        ret = np.exp(-(rr)**2)
        return ret
    
    if __name__ == "__main__":
        size = 25
        # interpolation_mode = "thin_plate"
        filter_mode = exp_12
        root= # Set root path for complex folders
        saving_folder = # Path for HDF5 storage
        saving_folder.mkdir(parents=True, exist_ok=True)
        p = Pool(cpu_count())
        p.map(process_file, get_files(root))
```
from scipy.spatial.distance import cdist
from mendeleev import element

def save_grid(saving_path, image):
    with h5py.File(str(saving_path), "w", libver='latest') as f:
        f.create_dataset('grid', dtype='f4', data=image)

def get_files(folder):
    return folder.glob('*/complex_*.pdb')

def import_data(pdb_path):
    data_path = str(pdb_path).replace('.pdb', '.attr.npz')
    data = np.load(data_path)
    radii = dict(zip(data["rc_keys"], data["radius_values"]))
    charges = dict(zip(data["rc_keys"], data["charge_values"]))
    energies = data["energy_values"].squeeze()
    energy_keys = data["energy_keys"]
    return radii, charges, energy_keys, energies

from scipy.spatial import KDTree

def apply_filter(filter_type, points, values, targets, radii):
    c = 0
    mask = np.linalg.norm(points, axis=-1) <= 12.5*np.sqrt(3)
    points = points[mask]
    values = values[mask, :]
    radii = radii[mask]
    dists = cdist(points,targets)
    aux = np.where(dists < 5, filter_type(dists, radii), 0)
    del dists
    result = values[np.argmax(aux, axis=0),0] * np.max(aux, axis=0)
    return result

def interpolate(int_type, points, values, targets):
    mask = np.linalg.norm(points, axis=-1) <= 12.5*np.sqrt(3)
    points = points[mask]
    values = values[mask, :]
    #import pdb; pdb.set_trace()
    points_x, points_y, points_z = [c.flatten() for c in np.split(points,3,axis=1)]
    targets_x, targets_y, targets_z = [c.flatten() for c in np.split(targets,3,axis=1)]
    res = Rbf(points_x, points_y, points_z, values, function=int_type)(targets_x, targets_y, targets_z)
    return res

def grid_around(center, size, spacing=1.0):
    size_ang = ((size - 1) / 2.) * spacing
    ex_min = center - size_ang
    ex_max = center + size_ang
    x = np.linspace(ex_min[0], ex_max[0], size)
    y = np.linspace(ex_min[1], ex_max[1], size)
    z = np.linspace(ex_min[2], ex_max[2], size)
    return np.stack(np.meshgrid(x, y, z,indexing='ij'), axis=-1)

C.4. Feature generation
C. Pipeline scripts

```python
def get_keys(pdb):
    resnums = pdb.getResnums()
    chids = pdb.getChids()
    names = pdb.getNames()
    keys = np.char.add(np.char.mod('%s−', np.char.replace(np.char.add(np.char.mod('%s ', resnums), chids), ' ', '−')), names)
    return keys

def build_images(pdb_path, size, interpolation_mode):
    complex = parsePDB(str(pdb_path))
    protein = complex.select("not (resname WER or water)")
    ligand = complex.select("resname WER")
    center = calcCenter(ligand.getCoords())
    moveAtoms(complex, by=−center)
    center = calcCenter(complex.select("resname WER").getCoords())
    size_ang = ((size − 1) / 2.) * (24/(size − 1))
    ex_min = center − size_ang
    ex_max = center + size_ang
    position_matrix = grid_around(center, size, spacing=24/(size − 1))
    radii, charges, energy_keys, energies = import_data(pdb_path)
    coordinates = complex.getCoords()
    keys = get_keys(complex)
    keys_p = get_keys(protein)
    keys_l = get_keys(ligand)
    existing_els = set(complex.getElements())
    el_dict = {name.capitalize(): element(name.capitalize()).en_pauling
               for name in existing_els}
    print(el_dict)
    names_p = protein.getElements()
    names_l = ligand.getElements()
    elec_p = [el_dict[name.capitalize()] for name in names_p]
    elec_l = [el_dict[name.capitalize()] for name in names_l]
    elec_p = [(elec, coord, radii[key]) for elec, coord, key in zip(elec_p, protein.getCoords(), keys_p) if key in energy_keys]
    elec_l = [(elec, coord, radii[key]) for elec, coord, key in zip(elec_l, ligand.getCoords(), keys_l) if key in energy_keys]
    key_map = dict(zip(keys, range(len(keys))))
    elec_p, coord_p, radii_p = zip(*elec_p)
    elec_l, coord_l, radii_l = zip(*elec_l)
    elec_p = np.array(elec_p).reshape((-1,1))
    coord_p = np.array(coord_p).reshape((-1,3))
    radii_p = np.array(radii_p).reshape((-1,1))
    elec_l = np.array(elec_l).reshape((-1,1))
    coord_l = np.array(coord_l).reshape((-1,3))
    radii_l = np.array(radii_l).reshape((-1,1))

    #image[k] = apply_filter(filter_mode, energy_coordinates, energies
                             [energy_ids, i], position_matrix.reshape((-1,3)), radii_c.reshape
```
C.4. Feature generation

$$\text{map}_p = \text{apply\_filter} (\text{filter\_mode}, \text{coord\_p}, \text{elec\_p}, \text{position\_matrix} \cdot \text{reshape}((-1,3)), \text{radii\_p}) \cdot \text{reshape}(3 \cdot (\text{size},) + (-1,))$$

$$\text{map}_l = \text{apply\_filter} (\text{filter\_mode}, \text{coord\_l}, \text{elec\_l}, \text{position\_matrix} \cdot \text{reshape}((-1,3)), \text{radii\_l}) \cdot \text{reshape}(3 \cdot (\text{size},) + (-1,))$$

#map_p = \text{interpolate} (\text{interpolation\_mode}, \text{coord\_p}, \text{elec\_p}, \text{position\_matrix}.\text{reshape}((-1,3))).\text{reshape}(3 \cdot (\text{size},) + (-1,))

#map_l = \text{interpolate} (\text{interpolation\_mode}, \text{coord\_l}, \text{elec\_l}, \text{position\_matrix}.\text{reshape}((-1,3))).\text{reshape}(3 \cdot (\text{size},) + (-1,))

map_p = \text{map}_p / 3.44

map_l = \text{map}_l / 3.98

return \text{np\_concatenate} ((\text{map}_p, \text{map}_l), \text{axis}=-1)

def process_file (file):
    saving_path = saving\_folder / (file.name.replace('\.pdb', '\.hdf5'))
    if saving_path.exists():
        return True
    try:
        image = build\_images (file, size, interpolation\_mode)
    except Exception as e:
        print(file)
        print(e)
        return False
    save\_grid (saving\_path, image)
    return True

def exp\_12 (r, rvdw):
    rvdw = rvdw.\text{reshape}((-1,))
    rr = rvdw[:, None]/r
    ret = np.where(r==0, 1, 1 - np.exp(-(rr)**12))
    return ret

def gaussian (r, rvdw):
    rvdw = rvdw.\text{reshape}((-1,))
    rr = r/rvdw[:, None]
    ret = np.exp(- (rr)**2)
    return ret

if __name__ == "__main__":
    size = 25
    #interpolation\_mode = "thin\_plate"
    filter\_mode = exp\_12
    root = # Set root path for complex folders
    saving\_folder = #Saving path for HDF5
    saving\_folder.makedir(parents=True, exist\_ok=True)
    p = Pool(48)
    p.map(process\_file, get\_files(root))
from collections import defaultdict
from multiprocessing import Pool
from pathlib import Path
import numpy as np
import h5py

def save_grid(saving_path, image):
    with h5py.File(str(saving_path), "w", libver='latest') as f:
        f.create_dataset("grid", dtype='f4', data=image)

def get_files(folder):
    return folder.glob('/*.complex*.pdb')

def import_data(pdb_path):
    data_path = str(pdb_path).replace('.pdb', '.attr.npz')
    data = np.load(data_path)
    radii = dict(zip(data['rc_keys'], data['radius_values']))
    charges = dict(zip(data['rc_keys'], data['charge_values']))
    return radii, charges

def build_images(pdb_path, size):
    complex = parsePDB(str(pdb_path))
    ligand = complex.select("resname WER")
    center = calcCenter(ligand, getCoords())
    moveAtoms(complex, by=-center)
    radii, charges = import_data(pdb_path)
    res = complex.getResindices() + 1
    resnums = complex.getResnums()
    chids = complex.getChids()
    names = complex.getNames()
    atoms = np.char.add(np.char.mod('%s', resnums), chids)
    try:
        atom_charges = np.array(itemgetter(*atoms)(defaultdict(float, charges)))
        atom_radii = np.array(itemgetter(*atoms)(defaultdict(float, radii)))
    except KeyError as e:
        print(f"Error in protein: {str(pdb_path)}")
        raise e
    complex.setCharges(atom_charges)
    complex.setRadii(atom_radii)
    protein_potential, ligand_potential, complex_potential =
    compute_electro_potential(pdb_path, complex_potential)
    grid = np.stack((protein_potential, ligand_potential, complex_potential), axis=-1)
    return grid

def compute_electro_potential(pdb_path, complex, size):
    path = pdb_path.parent
    center = np.array2string(calcCenter(complex.select('resname WER').getCoords()))[1:-1]
    grid_dim = f'.join(3 * [str(size)])
grid_space = ' '.join(3 * [str(1)])
cglen = ' '.join(3 * [str(size + 10)])
fglen = ' '.join(3 * [str(size - 1)])

protein_potential = APBS.run(path, pdb_path.stem+'.protein',
    complex,
        'not resname WER',
    grid_dim, grid_space, center, cglen,
    fglen)
ligand_potential = APBS.run(path, pdb_path.stem+'.ligand',
    complex,
        'resname WER',
    grid_dim, grid_space, center, cglen,
    fglen)
complex_potential = APBS.run(path, pdb_path.stem+'.complex',
    complex,
        'all',
    grid_dim, grid_space, center, cglen,
    fglen)

return protein_potential, ligand_potential, complex_potential

def process_file(file):
    saving_path = saving_folder / (file.name.replace('.pdb', '.hdf5'))
    print(file)
    if saving_path.exists():
        return True
    try:
        image = build_images(file, size)
        except Exception as e:
            print(e)
        return False
    save_grid(saving_path, image)
    return True

if __name__ == "__main__":
    size = 25
    root = # Set root path for complex folders
    saving_folder = # Saving path for HDF5
    saving_folder.mkdir( parents=True, exist_ok=True)
    p = Pool(24)
    files = filter(lambda x: not (saving_folder/x.name.replace('.pdb',
        '.hdf5')).exists(), get_files(root))
    p.map(process_file, files)

# READ IN MOLECULES
read
    mol pqr XXX.pqr
end

elec # Electrostatics calculation on the solvated state
C. Pipeline scripts

```python
from pathlib import Path
from subprocess import call
import numpy as np
from prody import writePQR
from utils import isfloat

class APBS:
    _APBS_BIN_PATH = "apbs"
    _TEMPLATE_FILE = "apbs.in"
    @staticmethod
    def run(
        output_path,
        name,
        pose,
        selection,
        grid_dim,
        grid_space,
        center,
    )
```

Listing C.19: thesis/appendix/apbs.py
C.4. Feature generation

cglen,
f=len):
apbs_bin_path = APBS'_APBS_BIN_PATH
apbs_template_file = Path(APBS'_TEMPLATE_FILE)
apbs_input_file = output_path / f'apbs_{name}.in'
apbs_output_file = output_path / f'{name}_potential.dx'
if apbs_input_file.exists():
apbs_input_file.unlink()
if apbs_output_file.exists():
apbs_output_file.unlink()
writePQR(f'{output_path/name}.pqr', pose.select(selection))
with apbs_template_file.open('r') as f:
  file_data = f.read()
file_data = APBS'.replace(apbs,
  file_data, str(output_path/name), grid_dim, grid_space, center, cglen, f[len)
with apbs_input_file.open('w') as f:
f.write(file_data)
call([apbs_bin_path,
  f'\{apbs_input_file.absolute()}'],
cwd=str(output_path))
apbs_input_file.unlink()
o, d, potential = APBS'.import_dx(apbs_output_file)
apbs_output_file.unlink()
return potential

@staticmethod
def _replace_apbs(
  file_data,
  xxx,
  grid_dim,
  grid_space,
  center,
  cglen,
  f[len):
  file_data = file_data
    .replace('XXX', xxx)
    .replace('GRID_DIM', grid_dim)
    .replace('GRID_SPACE', grid_space)
    .replace('INH_CENTER', center)
    .replace('CG_LEN', cglen)
    .replace('FG_LEN', f[len
  return file_data

@staticmethod
def _import_dx(filename):
  origin = delta = data = dims = None
  counter = 0
  with open(filename, 'r') as dxfile:
    for row in dxfile:
      row = row.strip().split()
      if not row:
        continue
      if row[0] == '#':
C. Pipeline scripts

```python
continue
elif row[0] == 'origin':
    origin = np.array(row[1:], dtype=float)
elif row[0] == 'delta':
    delta = np.array(row[2:], dtype=float)
elif row[0] == 'object':
    if row[1] == '1':
        dims = np.array(row[3:], dtype=int)
data = np.empty(np.prod(dims))
elif isfloat(row[0]):
    data[3 * counter:min(3 * (counter + 1), len(data))]
        ] = np.array(row, dtype=float)
counter += 1
data = data.reshape(dims)
return origin, delta, data

@staticmethod
def export_dx(filename, density, origin, delta):
    nx, ny, nz = density.shape
    with open(filename, 'w') as dxfile:
        dxfile.write(f'object 1 class gridpositions counts {nx} {ny} {nz}
        )
        dxfile.write(f'origin {origin[0]} {origin[1]} {origin[2]}
        )
        dxfile.write(f'delta {delta} 0.0 0.0
        )
        dxfile.write(f'delta 0.0 {delta} 0.0
        )
        dxfile.write(f'delta 0.0 0.0 {delta}
        )
        dxfile.write(f'object 2 class gridconnections counts {nx}, {ny},
        {nz}
        )
        dxfile.write(f'object 3 class array type double rank 0 items {nx * ny * nz}
        data follows
        )
i = 1
    for d in density.flatten(order='C'):
        if i % 3:
            dxfile.write('} \'.format(d))
        else:
            dxfile.write('} \'.format(d))
i += 1
    dxfile.write('}n')
```

Listing C.20: thesis/appendix/make_supermap.py

```python
import h5py
import numpy as np
import os
from pathlib import Path
from multiprocessing import Pool

@override
suffi = ""
```

88
C.4. Feature generation

```python
def get_files():
    rosetta = [x for x in os.listdir(rosetta_path) if '.hdf5' in x]
    htmmd = [x for x in os.listdir(htmd_path) if '.hdf5' in x]
    apbs = [x for x in os.listdir(apbs_path) if '.hdf5' in x]
    elec = [x for x in os.listdir(electroneg_path) if '.hdf5' in x]
    files = set.intersection(set(rosetta), set(htmd), set(elec))
    return files

def combine_files(file):
    rosetta = os.path.join(rosetta_path, file)
    htmmd = os.path.join(htmd_path, file)
    apbs = os.path.join(apbs_path, file)
    elec = os.path.join(electroneg_path, file)
    output = os.path.join(output_path, file)
    if Path(output).exists():
        return
    try:
        with h5py.File(rosetta, 'r') as f:
            rosetta_grid = np.array(f['grid'])
        except:
            print("Error rosetta ", file)
        return
    try:
        with h5py.File(htmd, 'r') as f:
            htmmd_grid = np.array(f['grid'])
        except:
            print("Error htmmd ", file)
        return
    try:
        with h5py.File(elec, 'r') as f:
            elec_grid = np.array(f['grid'])
        except:
            print("Error elec ", file)
        return
    try:
        with h5py.File(apbs, 'r') as f:
            apbs_grid = np.array(f['grid'])
        except:
            print("Error apbs ", file)
        return
    grid = np.concatenate((htmmd_grid, elec_grid, rosetta_grid), axis=-1)
    print(grid.shape)
    with h5py.File(output, 'w', libver='latest') as f:
        f.create_dataset("grid", dtype='f4', data=grid)

if __name__ == "__main__":
    output_path = # Set HDF5 output folder
    rosetta_path = # Set HDF5 input folder
    htmmd_path = # Set HDF5 input folder
    electroneg_path = # Set HDF5 input folder
    apbs_path = # Set HDF5 input folder
    Path(output_path).mkdir(parents=True, exist_ok=True)
    p = Pool(48)
```

89
import os
import tensorflow as tf
import h5py
import numpy as np
from pathlib import Path
import shutil

# Path to core pdb list file
with Path(pdb_core).open('r') as core_list_file:
    pdb_lines = [x.replace('
', '') for x in core_list_file.readlines()]
return set(pdb_lines)

if source == 'PDBBind':
    csv_file = 'INDEX.refined.data.2018'
    csv = pd.read_csv(csv_file,
        comment='#',
        delim_whitespace=True,
        header=None,
        usecols=[0, 3])
    return dict(zip(csv[0], csv[3]))

def get_file_list(path):
    return list(folder.glob('*.hdf5'))

def split_dataset(files, train_percentage):
    n = len(files)
    indexes = np.arange(n)
    shuffled_indexes = np.random.permutation(indexes)
    percentages = np.arange(n, dtype=float) / n
    train_indexes = shuffled_indexes[percentages <= train_percentage]
    test_indexes = shuffled_indexes[percentages > train_percentage]
    train_files = [files[i] for i in train_indexes]
    test_files = [files[i] for i in test_indexes]
    return train_files, test_files

def generate_tfrecord(files, output_file):
    pdb_labels = get_pdb_labels('PDBBind')
    with tf.python_io.TFRecordWriter(output_file) as writer:
        for file in files:
            pdb_code = file.stem[:4]
            with h5py.File(str(file)) as hdf5_file:
C.4. Feature generation

datapoint = np.array(hdf5_file['grid'], dtype=np.float32)
X = datapoint.flatten()
y = np.array([pdb_labels[pdb_code]])
example = tf.train.Example(features=tf.train.Features(
    feature={
        'X': tf.train.Feature(float_list=tf.train.FloatList(value=X)),
        'y': tf.train.Feature(float_list=tf.train.FloatList(value=y))
    }))
writer.write(example.SerializeToString())
file.unlink()
print('Done writing {}'.format(output_file))

def chunk_by_size(files, recommended_tf_size=float(100*(2**20))):
    average_size = sum(map(lambda x: x.stat().st_size, files)) / float(len(files))
    files_per_chunk = np.ceil(recommended_tf_size / average_size)
    chunks = int(np.ceil(float(len(files)) / files_per_chunk))
    return np.array_split(np.array(files), chunks)

def hdf5_to_tftrecords(folder, split):
    files = get_file_list(folder)
    train_files, test_files = split_dataset(files, split)
    train_chunks = chunk_by_size(train_files)
    test_chunks = chunk_by_size(test_files)
    train_folder = folder / 'train'
    test_folder = folder / 'test'
    shutil.rmtree(train_folder, ignore_errors=True)
    shutil.rmtree(test_folder, ignore_errors=True)
    train_folder.mkdir(parents=True, exist_ok=True)
    test_folder.mkdir(parents=True, exist_ok=True)
    for i, chunk in enumerate(train_chunks):
        chunk_output = train_folder / 'train{}.tftrecords'.format(i)
        generate_tftrecord(chunk, str(chunk_output))
        print('Generated {}'.format(chunk_output))
    for i, chunk in enumerate(test_chunks):
        chunk_output = test_folder / 'test{}.tftrecords'.format(i)
        generate_tftrecord(chunk, str(chunk_output))
        print('Generated {}'.format(chunk_output))

def split_dataset_refined_core(files):
    pdb_core = get_pdb_core_set_list()
    train_files = [file for file in files if file.stem[:4] not in pdb_core]
    test_files = [file for file in files if file.stem[:4] in pdb_core]
    return train_files, test_files

def hdf5_to_tftrecords_refined_core(folder):
    files = get_file_list(folder)
    train_files, test_files = split_dataset_refined_core(files)
    train_chunks = chunk_by_size(train_files)
    test_chunks = chunk_by_size(test_files)
    train_folder = folder / 'train'
C. Pipeline scripts

```python
test_folder = folder / 'test'
shutil.rmtree(train_folder, ignore_errors=True)
shutil.rmtree(test_folder, ignore_errors=True)
train_folder.mkdir(parents=True, exist_ok=True)
test_folder.mkdir(parents=True, exist_ok=True)
for i, chunk in enumerate(train_chunks):
    chunk_output = train_folder / 'train_{}.tfrecords'.format(i)
    generate_tfrecord(chunk, str(chunk_output))
    print('Generated {}'.format(chunk_output))
for i, chunk in enumerate(test_chunks):
    chunk_output = test_folder / 'test_{}.tfrecords'.format(i)
    generate_tfrecord(chunk, str(chunk_output))
    print('Generated {}'.format(chunk_output))

import argparse
parser = argparse.ArgumentParser(usage='%(prog)s A ? [B | C]
parser.add_argument('-f', '--folder', dest='folder', default=str(Path.home()))
parser.add_argument('-c', '--core', dest='core', action='store_true')
parser.add_argument('-s', '--split', dest='split', type=float)
if __name__ == '__main__':
    args = parser.parse_args()
    folder = args.folder
    split = args.split
    folder = Path(folder)
    if args.core:
        hdf5_to_tfrecords_reined_core(folder)
    else:
        hdf5_to_tfrecords(folder, split)
```

C.5 PDBQT generation

Listing C.22: thesis/appendix/make_pdbqt.sh

```bash
#!/bin/bash
root= # Set root path for complex folders
pythonsh=$({MGLTOOLS}/pythonsh
find $root -type d -maxdepth 1 -mindepth 1 | sort | parallel -n 1 $pythonsh preprocess_vina.py {}
```

Listing C.23: thesis/appendix/preprocess_vina.py

```python
#Code extracted from MGLTools. AutoDock 4 is distributed under GNU GPL license. http://autodock.scripps.edu/
import os
from MolKit import Read
import MolKit.molecule
import MolKit.protein
from AutoDockTools.MoleculePreparation import AD4ReceptorPreparation, AD4LigandPreparation
```
import sys
import getopt

def preprocess_receptor(receptor_filename, output_filename):
    repairs = '"
    charges_to_add = 'gasteiger'
    preserve_charge_types= None
    cleanup = ""
    mode = "automatic"
    delete_single_nonstd_residues = False
    dictionary = None

    mols = Read(receptor_filename)
    mol = mols[0]
    preserved = {}
    if charges_to_add is not None and preserve_charge_types is not None:
        preserved_types = preserve_charge_types.split(' , ')
        for t in preserved_types:
            if not len(t): continue
            ats = mol.allAtoms.get(lambda x: x.autodock_element==t)
            for a in ats:
                if a.chargeSet is not None:
                    preserved[a] = [a.chargeSet, a.charge]

    if len(mols) > 1:
        ctr = 1
        for m in mols[1:]:
            ctr += 1
            if len(m.allAtoms) > len(mol.allAtoms):
                mol = m
        mol.buildBondsByDistance()

    RPO = AD4ReceptorPreparation(mol, mode, repairs, charges_to_add, cleanup, output_filename=output_filename,
    preserved=preserved,
    delete_single_nonstd_residues=
    delete_single_nonstd_residues,
    dict=dictionary)

    if charges_to_add is not None:
        for atom, chargeList in preserved.items():
            atom._charges[chargeList[0]] = chargeList[1]
            atom.chargeSet = chargeList[0]

def preprocess_ligand(ligand_filename, output_filename):
    verbose = None
    repairs = "" #"hydrogens_bonds"
    charges_to_add = 'gasteiger'
    preserve_charge_types=""
    cleanup = ""
    allowed_bonds = "backbone"
    root = 'auto'
C. Pipeline scripts

```python
check_for_fragments = True
bonds_to_inactivate = ""
inactivate_all_torsions = True
attach_nonbonded_fragments = True
attach_singletons = True
mode = "automatic"
dict = None

mols = Read(ligand_filename)
if verbose: print 'read', ligand_filename
mol = mols[0]
if len(mols)>1:
    ctr = 1
    for m in mols[1:]:
        ctr += 1
        if len(m.allAtoms)>len(mol.allAtoms):
            mol = m
coord_dict = {}
for a in mol.allAtoms: coord_dict[a] = a.coords

mol.buildBondsByDistance()
if charges_to_add is not None:
    preserved = {}
    preserved_types = preserve_charge_types.split(',
    for t in preserved_types:
        if not len(t): continue
        ats = mol.allAtoms.get(lambda x: x.autodock_element==t)
        for a in ats:
            if a.chargeSet is not None:
                preserved[a] = [a.chargeSet, a.charge]
LPO = AD4LigandPreparation(mol, mode, repairs, charges_to_add,
cleanup, allowed_bonds, root,
outputfilename=outputfilename,
dict=dict, check_for_fragments=
check_for_fragments,
bonds_to_inactivate=bonds_to_inactivate,
inactivate_all_torsions=
attach_nonbonded_fragments=
attach_nonbonded_torsions=attach_singletons=attach_singletons)
if charges_to_add is not None:
    for atom, chargeList in preserved.items():
        atom._charges[chargeList[0]] = chargeList[1]
        atom.chargeSet = chargeList[0]
bad_list = []
for a in mol.allAtoms:
    if a in coord_dict.keys() and a.coords!=coord_dict[a]:
        bad_list.append(a)
if len(bad_list):
    print len(bad_list), 'atom coordinates changed!'```
C.6. Molecular Dynamics simulation

Listing C.24: thesis/appendix/make_protein.psf.tcl

```tcl
package require autopsf

proc gen_psf {pdb_path} {
    set molID [mol load pdb $pdb_path]
    autopsf -mol $molID -protein -regen
}

gen_psf [lindex $argv 0]
```

C.6 Molecular Dynamics simulation
C. Pipeline scripts

Listing C.25: thesis/appendix/molecular_dynamics.py

```python
import argparse
import os
import subprocess
import queue
import threading
from pathlib import Path
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
from mdtraj.reporters import DCDReporter

def generate_snapshots(pdb_path, psf_path, dcd_path):
    psf = CharmmPsfFile(psf_path)
    pdb = PDBFile(pdb_path)
    params = CharmmParameterSet(
        'toppar/stream/misc/toppar_amines.str',
        'toppar/stream/misc/toppar_dum_noble_gases.rtf',
        'toppar/stream/misc/toppar_hbond.str',
        'toppar/toppar_water_iions.str',
        'toppar/stream/prot/toppar_all36_prot_model.rtf',
        'toppar/top_all122_metals.rtf',
        'toppar/par_all122_metals.inp',
        'toppar/top_all36_carb.rtf',
        'toppar/top_all36_carb.prm',
        'toppar/top_all36_cgenff.rtf',
        'toppar/par_all36_cgenff.prm',
        'toppar/top_all35_ethers.str',
        'toppar/par_all35_ethers.prm',
        'toppar/top_all36_lipid.rtf',
        'toppar/par_all36_lipid.prm',
        'toppar/top_all36_na.rtf',
        'toppar/par_all36_na.prm',
        'toppar/top_all36_prot.rtf',
        'toppar/par_all36_prot.prm',
        'toppar/top_all27_prot_na.rtf',
        'toppar/par_all22_prot.prm',
        'toppar/stream/prot/toppar_all36_prot_aldehydes.rtf',
        'toppar/stream/prot/toppar_all36_prot_pyridines.rtf',
        'toppar/stream/prot/toppar_all36_prot_fluoro_alkanes.rtf',
        # 'toppar/toph19.inp',
        # 'toppar/paramh19.inp',
        # 'toppar/hbond.inp'
    )
    modeller = Modeller(pdb.topology, pdb.positions)
    modeller.addHydrogens()
    system = psf.createSystem(params, nonbondedMethod=NoCutoff,
        nonbondedCutoff=1*nanometer, constraints=HBonds, implicitSolvent =HCT, implicitSolventSaltConc=0.1*moles/liter)
```
C.6. Molecular Dynamics simulation

```python
integ = LangevinIntegrator(300*kelvin, 1/picosecond, 1*femtoseconds)
simulation = Simulation(psf.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.minimizeEnergy(maxIterations=2000)
simulation.reporters.append(DCDReporter(dcd_path, 200000))
simulation.step(200000)
simulation.saveCheckpoint(dcd_path + "chk")

sp = lambda pdb, gpu: subprocess.run(["python3", "molecular.dynamics.py", str(gpu), "--run", pdb])

def worker(gpu):
    while True:
        pdb = q.get()
        if pdb is None:
            break
        spawn(pdb, gpu)
        q.task_done()

def set_queue(gpus, pdbs):
    threads = []
    for gpu in gpus:
        t = threading.Thread(target=worker, args=(gpu, ))
        t.start()
        threads.append(t)
    for pdb in pdbs:
        q.put(pdb)
    q.join()
    for t in threads:
        t.join()

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(‘gpu’)  
    parser.add_argument(‘--file’)  
    parser.add_argument(‘--run’)  
    args = parser.parse_args()

gpu = args.gpu
if args.run:
    os.environ["CUDA_VISIBLE_DEVICES"] = gpu
    pdb_path = str(Path(‘.pdb_autopsf’) / (pdb_code + ‘.pdb’))
    psf_path = str(Path(‘.psf_autopsf’) / (pdb_code + ‘.psf’))
    dcd_path = str(Path(‘.dcd’) / (pdb_code + ‘.dcd’))
    if not Path(dcd_path).exists():
        print("ERROR: No DCD file exists", pdb_path)
        generate_snapshots(pdb_path, psf_path, dcd_path)
else:
```

C. Pipeline scripts

```python
q = queue.Queue()
gpus = GPU.split(',')
pdbs = []
with open(args.file) as f:
    files = f.read().split()
    for pdb_code in files:
        pdb_path = (Path('./pdb_autopsf') / (pdb_code + '_protein_autopsf.pdb'))
        dcd_path = Path('./dcd') / (pdb_path.stem + '.dcd')
        if not dcd_path.exists():
            pdbs.append(pdb_path.stem)
set_queue(gpus, pdbs)
```
Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

NEURAL NETWORKS FOR IMPROVING DRUG DISCOVERY EFFICIENCY

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

HASSAN HARRIROU

First name(s):

HUSSEIN

Place, date

ZÜRICH, 31.03.2019

With my signature I confirm that
− I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
− I have documented all methods, data and processes truthfully.
− I have not manipulated any data.
− I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.