TENSOR REPRESENTATIONS OF MOLECULAR GRAPHS FOR STRUCTURE BASED VIRTUAL DRUG SCREENING

A PREPRINT

Rick Stevens [*]
Department of Computer Science
University of Chicago
Chicago, IL 60615

Carlos A. Olivares
Stevens Group Intern
University of Chicago
cao826@uchicago.edu

September 3, 2020

ABSTRACT

Using deep learning to predict molecular docking scores could potentially speed up a drug discovery process that is orders of magnitude slower than modern pandemics. Successfully incorporating deep learning in molecular docking based lead identification requires comparatively studying different molecular featurizations for predicting docking scores. In this paper we compare the accuracy of models trained on tensor representations of molecular graphs to models with other standard molecular featurizations and discuss how tensor based models can make a lead identification pipeline faster and more efficient. We perform this comparison in the context of identifying lead molecules for COVID-19 antivirals.

1 Introduction

Long before COVID-19 was discovered, global health experts worried about the international community's pandemic preparedness. Advocates claim that a successful pandemic response requires investing into a platform allowing effective clinical trial results and regulatory approval for medicines (vaccines or antivirals) in 3 months or less. This estimate is appropriate for the timeframe of a pandemic, as the time between the discovery of SARS-CoV-2 and the declaration of COVID-19 as a pandemic was about 3 months. However, the average time from discovery to market for a medication is 13 years with an average cost of over 1 billion USD [1, 2]. For every medication that receives approval to enter the market, there are several candidates that do not receive approval, leading to millions of dollars and years of wasted effort. In its current state, the drug discovery pipeline is orders of magnitude slower than a pandemic with access to modern transportation, highlighting a serious need to introduce faster methods for finding molecules that can serve as medicines. Molecules of this kind are called lead molecules, and searching for them is called lead identification.

One promising technique for speeding up lead identification is to use deep learning to predict molecular docking scores. Molecular docking is a computational method that seeks to predict if and how a molecule would stably bind to a target protein (a protein which drives a deleterious biological effect, like a protein that allows the SARS-CoV-2 virus to invade a host cell). Docking has two stages: pose identification, where an algorithm docks a molecule to a pocket in the protein, and a scoring process that predicts how strongly a molecule will bind to the protein in a given pose. Research shows that scoring methods based on molecular dynamics simulations are very accurate at predicting the binding affinity. However, the simulations used to score a docking pose require expensive high performance computing, which makes the method intractable for effectively exploring the space of all potentially pharmacologically active molecules (estimated to be 10^{60} in magnitude[3]). If researchers were able to train deep neural networks to accurately predict the docking scores from simulation based scoring algorithms, the process would be drastically faster and could efficiently identify lead molecules.

In order to better characterize the potential for deep learning enabled molecular docking, researchers need to compare the performance of different molecule featurizations when applied to this method. Molecules are complicated three-

[*]I don't know if I should put more people here or in the acknowledgements section.

dimensional quantum mechanical objects, and researchers have a hard time developing digital representations (or featurizations) that a machine learning algorithm can take as input and predict high level chemical properties from. Different featurizations can significantly affect model behavior and accuracy. Molecular featurizations can be based on the molecular graph, like images or tensor represention, or they can be based on finding an embedding, like descriptors and fingerprints. For this paper, we will featurize molecules with tensor representations of their molecular graphs and compare the performace of models based on this featurization with that of models based on other featurizations. We perform this comparison in the context of identifying lead molecules for COVID-19 antivirals. In section 2, we explain our methods. We present our results in section 3, and discuss them in section 4.

2 Methods

2.1 Data

The data used for experimentation was a preliminary dataset from the 2019-nCoV group. The 2019-nCoV group is a collaborative effort between researchers at Argonne and other national labs to compute chemical properties for a large number of drug like molecules, train machine learning tools, and use the resulting models for screening large numbers of drugs for COVID-19 medications. The data comes in a `.csv` file with a set of smiles strings used to identify a molecule and a collection of simulation based docking scores that molecule received from 24 different protein pockets. This data can be made available upon request, and more data of this kind will likely be released to the public as part of the 2019-nCov group's data releases[4].

2.2 Featurizing a molecule

For this experiment, molecules were featurized using tensor representations of molecular graphs. The representation itself is a pair of tensors, one to represent the adjacency matrix (or connectivity matrix) of the molecular graph, and another tensor to specify the identities of the atoms in the molecule by atomic number. In the raw data, molecules were identified by smiles strings. We used the open source cheminformatics software `rdkit` to compute an adjacency matrix from the smiles string of the molecule and the tensor of atomic identities.

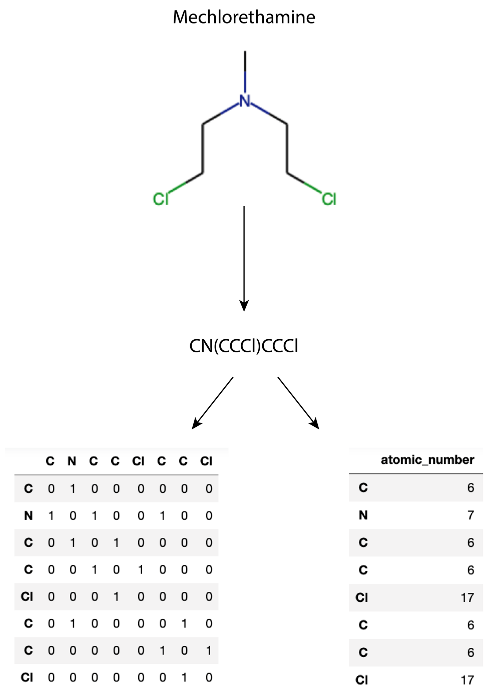
Padding Tensors One reason why featurizing molecules is difficult is because of variations in molecular size. Most neural network models can only accept examples of uniform input size, necessitating that molecular featurizations map different sized molecules to a featurization of just one size. We accomplish this by padding the featurization tensors to standard sizes. The standard sizes were chosen to prevent training models on large molecules (> 65 atoms), as they were infrequent and models that could take their input had too many parameters. The padding value was -1.0, as 0 is used in adjacency matrices to denote the lack of a chemical bond between nodes in the graph. Atom identity tensors were also padded with -1.0.

Cleaning smiles strings A limited number of smiles strings triggered an error the `rdkit` algorithm to compute an adjacency matrix. To adjust for this, we determined which of the smiles strings could be featurized and serialized them as a list with `pickle`. This list can be made available upon request.

2.3 Preparing featurized datasets

The data format is `smiles_string, pocket_1_score, pocket_2_score, ..., pocket_n_score`. Note that `pocket_j_score` for each `j` in the column index, when paired with the `smiles` column, is a full learning problem to be split into training and validation sets. This required a method to generate a learning problem for each individual pocket in the data. We wrote a plain python object called a `DFHandler` which takes as input the paths to the data file and the serialized list of clean smiles strings, and could generate a learning problem for each pocket in the data. The `DFHandler` outputs two dictionaries of `smiles_string:docking_score` pairs, and these dictionaries are used as input to `DataGenerator` objects that dynamically generate batches of featurized molecules and labels and feeds them to a `tensorflow.keras.models.Model` instance [5].

The code for this process, as well as examples and instructions for using this code is available at the GitHub repository for this project.



	C	N	C	C	Cl	C	C	Cl
C	0	1	0	0	0	0	0	0
N	1	0	1	0	0	1	0	0
C	0	1	0	1	0	0	0	0
C	0	0	1	0	1	0	0	0
Cl	0	0	0	1	0	0	0	0
C	0	1	0	0	0	0	1	0
C	0	0	0	0	0	1	0	1
Cl	0	0	0	0	0	0	1	0

	atomic_number
C	6
N	7
C	6
C	6
Cl	17
C	6
C	6
Cl	17

Figure 1: Example of the featurization process for Mechlorethamine. Mechlorethamine was originally an ingredient in mustard gas during the first world war, but in the 1950's it was discovered to be an effective medication against certain types of lung cancers and leukiemias. Note that for readibility, the tensors are not padded.

2.4 Choosing Pockets

During experimentation, we noticed that model behavior and accuracy fluctuated based on the underlying distribution of docking scores for each pocket. We chose pockets based on two considerations: that the distribution should not have more than half of its scores as zero and that the distributions of the pockets we choose should be similar. Base on this, we chose `PLPro_pocket6_round1_dock`, `nsp15-CIT_pocket6_round1_dock`, `Nsp9_pocket2_round1_dock`, and `ADRP-ADPR_pocket1_round1_dock`.

2.5 Model selection

We were able to explore the performance of all of the models defined in the search space below with keras tuner [6]. We performed the model selection with random search. The code used to find an optimal model and explanations for how to use the code are available in the GitHub Repository.

Hyperparameters During the development stage of the experiments, we learned that a batch size of 10 maximized the accuracy of the models. Instead of searching for an optimal static learning rate, we used callbacks to reduce the learning rate by a factor of 10 when the validation loss does not decrease after a specified number of epochs.

Model architectures We hypothesized that under the tensor representation of molecular graphs, models would learn by searching for molecular sub-structures that increase the stability of a bind. Based on this, we decided to explore different sized convolutional windows to see if this affected model performance.

Optimizers During the development stage of the experiment, we ran model selection with all of the optimizers available to us through keras. However, preliminary results showed us that Adam and stochastic gradient descent provided the best results. For our final set of experiments, we chose to perform model selection with just stochastic gradient descent and Adam.

We used four different model architectures to choose the best model from. The details of their architectures can be found in the GitHub repository.

2.6 Training and Evaluating Models

After the optimal models for a given pocket were found, we trained the models with two callbacks: one to drop the learning rate by a factor of ten if validation loss plateaued, and an early stop callback to end the training process if the validation loss stagnates and dropping the learning rate no longer helps. We set the models to train for 600 epochs to increase the chances that training would end by callback. Trained models would be evaluated by computing the pearson and spearman correlation coefficients between model predictions and true scores. For each pocket in the final set of experiments, the trained models, their predictions, and the true scores for the validation set are all serialized and can be made available upon request.

3 Results

Model Selection Results Stochastic gradient descent was the best optimizer for all pockets. Model 4 (largest convolutional windows, 12x12) performed best for three of the four pocket, with model 1 being the best performing model for nsp15 pocket 6. Here are the best to worst performing models for PLPro pocket 6:

Rank	Model type	Optimizer
1	4	SGD
2	1	SGD
3	2	SGD
4	3	SGD
5	2	Adam
6	1	Adam
7	3	Adam
8	4	Adam

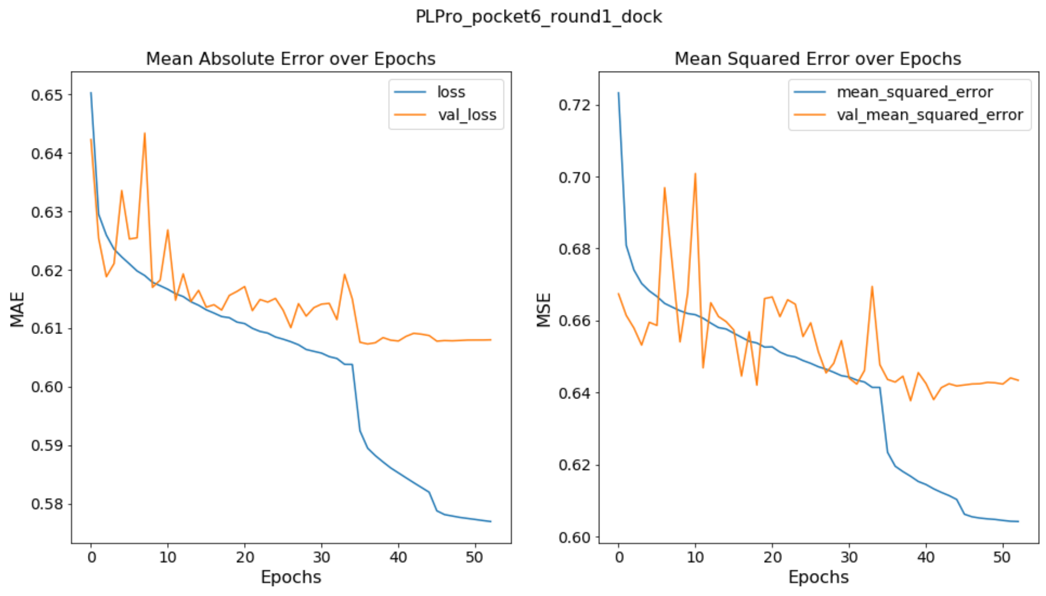


Figure 2: Metrics over epochs for PLPro pocket 6. Mean absolute error was used as the objective function. The model used was Model 4 with SGD.

Pocket	spearman correlation coefficient	pearson correlation coefficient
PLPro_pocket6_round1_dock	0.8217	0.8461
Nsp9_pocket2_round1_dock	0.7058	0.7357
ADRP-ADPR_pocket1_round1_dock	0.6950	0.7119
nsp15-CIT_pocket6_round1_dock	0.7335	0.7566

4 Discussion

The other models used by the 2019-nCoV group use images, fingerprints, and descriptors (2D and 3D). These models typically have a final validation mean absolute error of .5 or under, and spearman correlation between .85 and .95. Compared to these models, our models are not as accurate. The best performing model was just under the cutoff at .82 in the spearman correlation, but the mean absoute error was about .61. Intuitively, this result makes sense, as the featurization we chose does not contain as much information about a molecule as the featurizations. However, featurizing molecules with tensor representations are among the least (if not the least) computationally intesive featurization methods, and the accuracy it yields is surprisingly high. Because of this, we (I) still believe it is worthwhile to understand the performance of these models as part of a broader lead identification pipeline.

Predicting the molecular docking score is a regression problem. But, when we apply deep learning enabled simulation based molecular docking to lead identification, what we want to do is quickly separate good drug candidates from a large pool and further differentiate between them to find the best candidates. Our method of featurization provides relatively little information about a molecule to a model, but the accuracy we get with it is adequate to quickly triage a large number of drug candidates. Using our models for this purpose will allow for quickly discarding bad candidates and generating a set of higher quality candidates which we can use to train more accurate models to further differentiate between the drug targets. The most difficult and important task we could have in this application is accurately predicting the better candidate between two molecules with true scores -0.3 and -0.1. The best way we could do this is probably by actually docking the drug candidates. Using the tensor featurization, we can make the most efficient use of simulation based docking and other sophisticated and computationally intensive tools by rapidly focusing the search to the most promising candidates.

5 Acknowledgements

I would like to thank my advisor, Rick Stevens, for his invaluable help with completing this research. I would also like to thank everyone in the Stevens group at the University of Chicago.

References

[1] Bill Gates. Responding to Covid-19 — A Once-in-a-Century Pandemic? *New England Journal of Medicine*, 382(18):1677–1679, April 2020.

[2] Daniel C. Elton, Zois Boukouvalas, Mark D. Fuge, and Peter W. Chung. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design & Engineering*, 4(4):828–849, 2019.

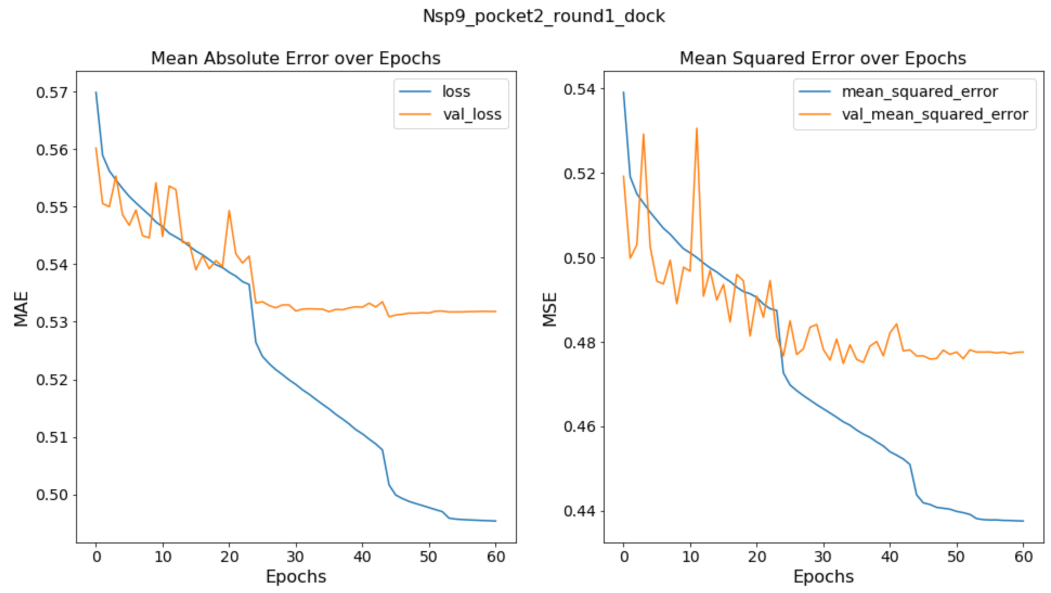
[3] Regine S. Bohacek, Colin McMartin, and Wayne C. Guida. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3–50, 1996. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291098-1128%28199601%2916%3A1%3C3%3A%3AAID-MED1%3E3.0.CO%3B2-6.

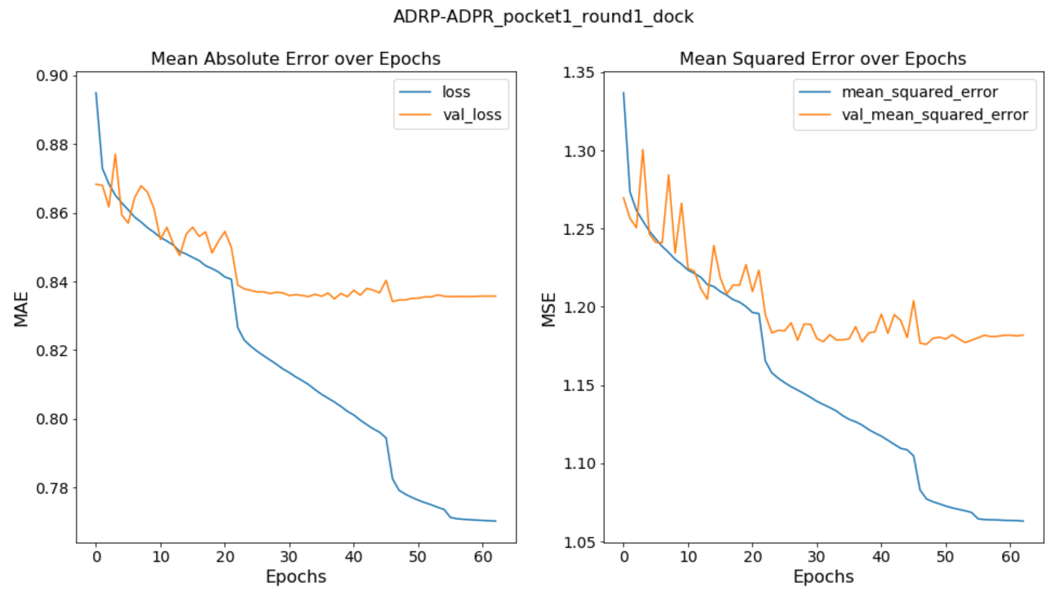
[4] Yadu Babuji, Ben Blaiszik, Tom Brettin, Kyle Chard, Ryan Chard, Austin Clyde, Ian Foster, Zhi Hong, Shantenu Jha, Zhuozhao Li, Xuefeng Liu, Arvind Ramanathan, Yi Ren, Nicholaus Saint, Marcus Schwarting, Rick Stevens, Hubertus van Dam, and Rick Wagner. Targeting SARS-CoV-2 with AI- and HPC-enabled Lead Generation: A First Data Release. *arXiv:2006.02431 [cs, q-bio, stat]*, May 2020. arXiv: 2006.02431.

[5] Francois Chollet et al. Keras, 2015.

[6] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras Tuner. `https://github.com/keras-team/keras-tuner`, 2019.

Supplementary Information





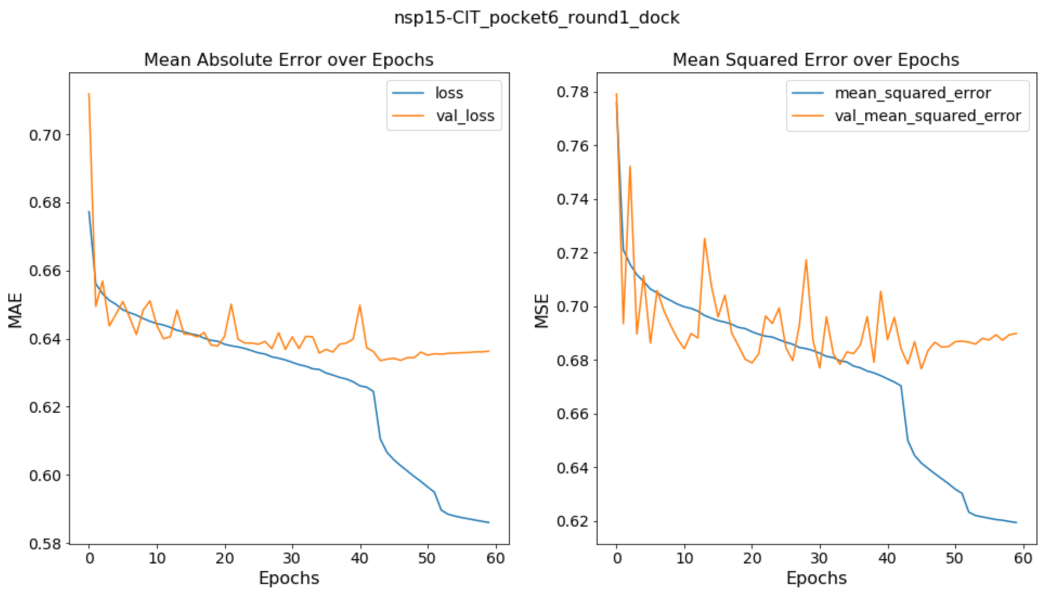
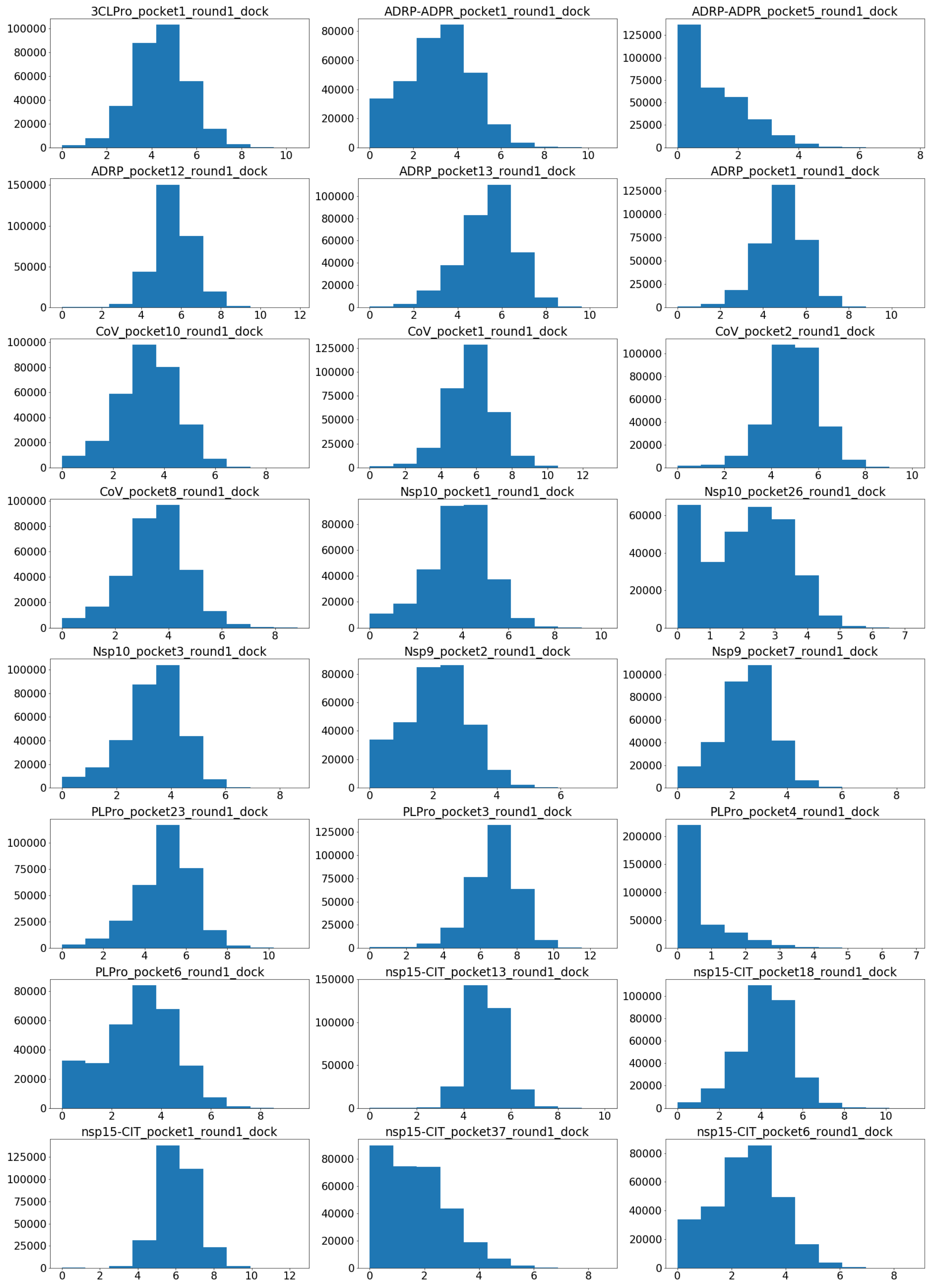

nsp15-CIT_pocket6_round1_dock
Mean Absolute Error over Epochs
MAE
Epochs
loss
val_loss
Mean Squared Error over Epochs
MSE
Epochs
mean_squared_error
val_mean_squared_error

Figure 3: Distribution of docking scores for all pockets.


3CLPro_pocket1_round1_dock
ADRP-ADPR_pocket1_round1_dock
ADRP-ADPR_pocket5_round1_dock
ADRP_pocket12_round1_dock
ADRP_pocket13_round1_dock
ADRP_pocket1_round1_dock
CoV_pocket10_round1_dock
CoV_pocket1_round1_dock
CoV_pocket2_round1_dock
CoV_pocket8_round1_dock
Nsp10_pocket1_round1_dock
Nsp10_pocket26_round1_dock
Nsp10_pocket3_round1_dock
Nsp9_pocket2_round1_dock
Nsp9_pocket7_round1_dock
PLPro_pocket23_round1_dock
PLPro_pocket3_round1_dock
PLPro_pocket4_round1_dock
PLPro_pocket6_round1_dock
nsp15-CIT_pocket13_round1_dock
nsp15-CIT_pocket18_round1_dock
nsp15-CIT_pocket1_round1_dock
nsp15-CIT_pocket37_round1_dock
nsp15-CIT_pocket6_round1_dock