# CS5242: Project Report (Authors: Chan Wei Xin, Yuki Sasa)

## 1 Problem Definition

Our work seeks to apply Deep Learning methods to predict the probability of protein-ligand binding through the Keras API. We model our prediction task as a binary classification problem, 'dock' vs 'no dock' between a protein-ligand pair. Our neural network takes in as input the (x, y, z) coordinates and atom types of atoms in a protein-ligand pair. Atom types have been simplified to be either hydrophobic or polar based on their element. Our test data set consists of 824 proteins and 824 ligands. We have been tasked to suggest 10 ligands that will bind to each protein. For each protein, we feed all 824 possible protein-ligand pairs into our neural network to obtain the probabilities of each pair binding. We select ten protein-ligand pairs with the highest probabilities of binding as our suggestions.

## 2 Insights

To obtain some intuition of how to design our neural network and how to preprocess our data to enable successful prediction of protein-ligand binding pairs, we viewed correct and incorrect protein-ligand pairs in PyMol [1]. We observed that correct pairs dock perfectly in 3D space, while incorrect pairs are often spatially distant. We exploited this fact to preprocess our data based on the spatial positions of the proteins and ligands.

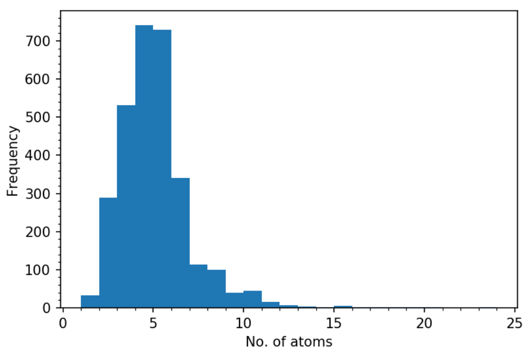
Also, we wanted our neural network to be able to infer some of the biological knowledge that humans have discovered about protein-ligand binding. Namely, that the docking site on a protein has a complementary shape to the ligand, that proteins and ligands only bind at a distance of a few angstroms, and that similar types of atoms tend to exist in both the binding site of the protein and the ligand. This is because the interaction between protein and ligand is often a result of dispersion forces that occur between non-polar molecules, and permanent dipole-induced dipole interactions that occur between polar molecules.

### 2.1 Protein and ligand sizes

Upon inspection of the PDB files provided, we discovered that the proteins and ligands were of different sizes. The protein sizes ranged from 38-14,644 atoms, while ligand sizes ranged from 1-24 atoms. Figure 1 shows the distribution of protein and ligand sizes. In order to feed all the protein-ligand pairs into our neural network, we had to preprocess our data set so that all the protein-ligand pairs would have an equal number of features.

### 2.2 Proportion of classes

We were provided with the PDB files of 3000 proteins and 3000 ligands, which contained information about the 3-dimensional structures of 3000 positive protein-ligand complexes. Assuming that one protein only docks with one ligand in the data set, this meant that out of the 9 million possible protein-ligand combinations, only 3000 were positive and the remaining 8997000 would be negative. This leads to an highly imbalanced data set, which we decided to correct


(a) Histogram of ligand sizes.

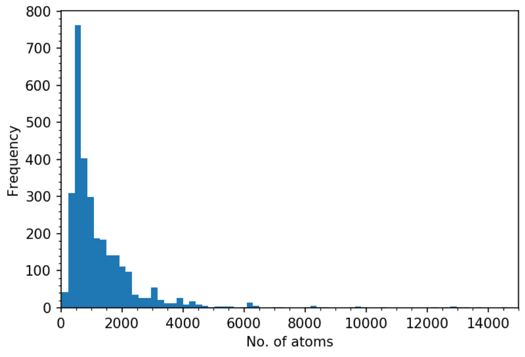

(b) Histogram of protein sizes.

Figure 1: Frequency distribution of ligand and protein sizes in the training data set in terms of number of atoms.

to an equal proportion of positive and negative samples by undersampling the negative samples. Hence, only 3000 negative samples were chosen at random to be used in our training data set. As a result, we ended up with a relatively small data set of 6000 samples. Another possible solution to the problem of a small data set would be to upsample the positive samples by generating synthetic data through slight perturbation of their features.

## 3 Data Preprocessing

In order to standardise the number of attributes for each protein-ligand pair, we devised an algorithm that would select an arbitrary number of protein atoms and ligand atoms for each protein-ligand pair. Our algorithm assumes that the protein atoms closest to the ligand are the most relevant to determining whether a protein-ligand pair binds or not. For positive protein-ligand pairs, we will essentially be selecting atoms that are in the ligand binding site of the protein. We decided to select 8 ligand atoms in order to avoid truncating the majority of our ligands after examining the distribution of ligand sizes (Figure 1a). We created 5 data sets selecting 8, 16, 40, 80 and 300 protein atoms together with 8 ligand atoms in order to test which would perform better in our prediction task. We initially thought that 300 protein atoms would give the best performance as it preserves the most information about the protein binding site. However, after evaluating all the data sets, we selected the data set with 16 protein atoms and 8 ligand atoms as it gave the best averaged performance over several multilayer perceptron (MLP) and convolutional neural network (CNN) models.

We describe our algorithm in greater detail with reference to how the test data set was generated. For each of the 824 ligands provided, we first calculated the centroid of each ligand based on the (x, y, z) coordinates of all ligand atoms. If the ligand has more than 8 atoms, the closest 8 atoms to the centroid are selected and are ordered with the closest on top of the ligand matrix. If the ligand has less than 8 atoms, the shortfall in attributes are padded with zeros at the bottom of the ligand matrix (Figure 2). For each of the 824 possible binding protein candidates, we select the 16 protein atoms that are closest to the ligand centroid with the closest atoms ordered at the bottom of the protein matrix. Thus, for each possible protein-ligand pair, we get a 24 by 4 matrix composed of the protein-ligand matrix as illustrated in Figure 2. Iterating through all 824 proteins and 824 ligands, we get 678,976 possible protein-ligand pairs in our test data set.

For the data set provided, we also had to convert the categorical variable, atom type, to a numerical value. We decided to assign a value of 1 to hydrophobic atoms and -1 to polar atoms. We used -1 instead of 0 in order to distinguish it from the 0 values we pad if there is a shortfall in ligand atoms.

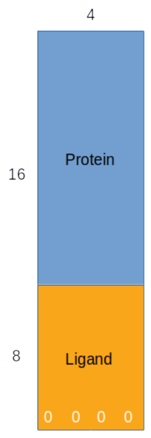


Figure 2: Illustration of processed data matrix for protein-ligand pair. It is formed by concatenating the protein matrix (blue) and ligand matrix (orange). Protein atoms closer to the centroid of the ligand are ordered at the bottom of the protein matrix. Any shortfall in ligand atoms are padded with zeroes starting at the bottom of the ligand matrix.

## 4 Experimental Study

### 4.1 Model Selection

After experimenting with several neural network models including MLPs, CNNs, Long-Short Term Memory (LSTM) [2] networks, and several ensemble machine learning methods, we decided on using the CNN for our prediction task. A test data set was first created from a 20% split off the training data set. We evaluated the models based on their categorical accuracy for the test data set, and also observed their ability to minimise the loss function. We decided to use the categorical cross entropy loss function, also known as the negative log likelihood, together with two output nodes followed by a final softmax activation function. This was used instead of just one output node followed by a sigmoid activation function, as it offers slightly better accuracy. Out of all the models, the CNN model achieved the best prediction accuracy, and also achieved good convergence of the loss function.

### 4.2 Hyperparameter Tuning

After deciding on using the CNN model, we tuned the hyperparameters of our model in an effort to increase the accuracy of our validation data set while ensuring good convergence of the loss function. We decided to use convolutional filters that were 7 by 3 in size, in order to mimic the shape of our protein-ligand data matrix (Figure 2). We also wanted the filters to cover more atoms at one go. The 7 by 3 filters showed the best performance as compared to 3 by 3 filters and 9 by 3 filters. We applied zero padding in order to mitigate the decrease in importance of the values at the edge of the data matrix. Subsequently, we tried to improve the validation accuracy by adding more convolutional layers. However, the use of more than two layers did not seem to yield any significant improvement. Efforts to use more than 8 convolutional filters per layer also did not result in an increase in accuracy. Our final model consists of two convolutional layers with eight 7 by 3 filters followed by 3 fully connected layers of size 64, 64 and 2. Increasing the width and depth of the fully connected layers had little effect on the performance of our model. We use the rectified linear unit (ReLU) activation function after every hidden layer to provide non-linear transformation to our inputs. The ReLU activation function is chosen over the sigmoid or tanh activation function in order to avoid the problem of vanishing gradients during backpropagation.

Initially, the training loss was converging very well while the validation loss had trouble converging. We applied batch normalization [3] before the hidden layers in order to normalize the distribution of their inputs, which helped the validation loss to converge. Batch normalization also acted as a regularizer and helped to reduce overfitting. However, our model was still showing signs of overfitting and hence dropout was applied to each hidden layer. The addition of dropout was effective in preventing our model from overfitting and caused the validation loss to converge better. However, as randomly selected neurons were dropped during training, this led to an increase in training time.

## 5 Train and Test Procedure

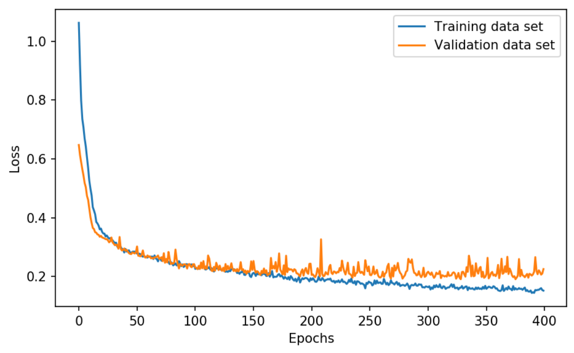
### 5.1 Train Procedure

We trained our CNN model using the 'Adam' optimiser [4] as it showed the best performance in converging the loss function and did not require much parameter tuning. Efforts to tune the parameters of the stochastic gradient descent (SGD) optimizer did not perform as well. We decided on using a batch size of 300 as it resulted in a good convergence of the loss function after trying out a series of sizes from 20 to 500. We decided to train our model over 400 epochs even though the categorical accuracy of the validation data set had plateaued by 250 epochs (Figure 3b) and the validation loss could no longer converge further (Figure 3a). This was to give some allowance when we trained our final prediction model with more samples. We obtained a final accuracy of 0.9361 over the training data set, and 0.9150 over the validation data set.

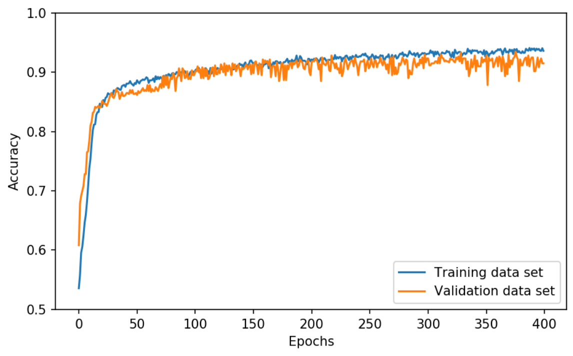
For our final prediction model, we decided to use the entire training data set of 6000 samples, instead of splitting part of it to form the validation data set. We wanted to maximise the number of samples used to train the model, as it was not big to begin with. The final prediction model gave a training accuracy of 0.9382. Figure 4 shows the loss and accuracy curves of our final prediction model.

### 5.2 Test Procedure

We ran our trained CNN model on the 678,976 possible protein-ligand pairs in our test data set to predict the probabilities of each pair binding. For each protein, we obtained its probability of binding with each of the 824 possible ligands. After ranking the ligands based on their probabilities of binding with the protein, we selected the top 10 ligands and presented them as our suggested binding candidates for each protein. An accurate prediction in this case will occur when the correct ligand is present in one of the 10 suggested ligands.

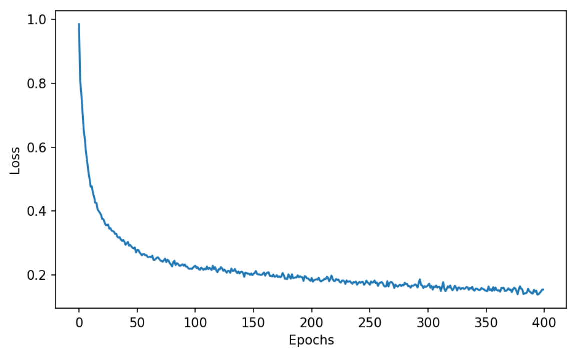


(a) Convergence of training loss (blue) is similar to that of the validation loss (orange).

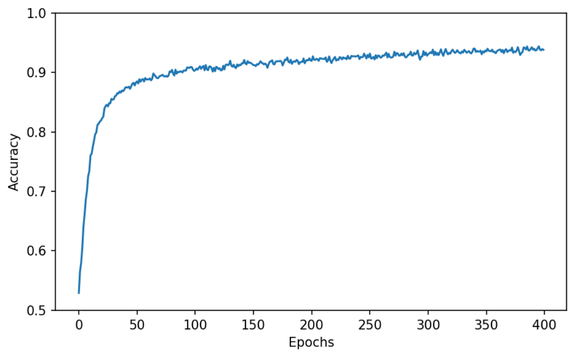


(b) Validation accuracy (orange) is close to training accuracy (blue).

Figure 3: Loss and accuracy curves of our simple CNN prediction model after optimization.



(a) Loss Curve



(b) Accuracy Curve

Figure 4: Training curves for our final CNN prediction model trained with the entire training data set containing 6000 samples (3000 positive and 3000 negative samples).

## 6 Discussion

### 6.1 Convolutional Neural Network

In this section, we seek to explain our intuition behind using the CNN model. CNNs such as AlexNet [5], VGGNet [6] and GoogLeNet [7] were optimized to process 2D image data and perform very well in image classification tasks. They comprise of convolutional filters that essentially process images to extract significant features such as the edges and contours in an image. The fact that our CNN model works well on a protein-ligand data matrix might be a little surprising. One feature of convolutional filters that may have enabled them to perform well on the data matrix is that CNNs are able to conserve spatial relationships between values in the data matrix, much like pixels in an image. During our preprocessing step, we ordered protein atoms that are closest to the ligand centroid near the middle of the data matrix (Figure 2)z. We also made sure to pad the missing ligand atoms with zeros starting from the bottom of the data matrix. This was done as the application of convolutional filters results in values at the edges of the data matrix contributing less to the prediction than the values near the center. Care was taken to pad the data matrix with zeroes before applying convolutional filters as well.

We adapted the VGGNet model developed by Simonyan and Zisserman [6] for our protein-ligand binding prediction task and tested its performance against our simple CNN model. However, the deeper VGGNet model that consisted of 16 convolutional layers achieved a poorer prediction accuracy than our simple CNN model. The deeper VGGNet model might be better suited for a larger input size, such as the 224 by 224 images it was optimized for. Our data matrix in comparison, is only 24 by 4 and is a very narrow rectangle compared to the square 224 by 224 images.

### 6.2 Multilayer Perceptron

The MLP is composed of an input layer with a number of nodes equal to the number of attributes of the data, followed by a variable number of hidden layers and an output layer. The nodes in between the layers are fully connected, hence all the nodes have an equal chance of contributing to the decision. The MLP will try to learn a representation of our training data in feature space that will enable it to discern between our two prediction classes. Our final MLP model performed almost as well as our CNN model, achieving a training accuracy of 0.9385 and validation accuracy of 0.8933.

### 6.3 Ensemble Machine Learning Methods

We also experimented with using several state-of-the-art ensemble machine learning methods such as the random forest [8], XGboost [9] and LightGBM [10] algorithms for our protein-ligand binding prediction task. The random forest algorithm trains a collection of decision trees and makes a prediction by averaging the output of each tree. We obtained a training accuracy of 0.946 and validation accuracy of 0.753 using the random forest algorithm. The XGBoost algorithm uses presorted and histogram-based algorithm for computing the best split. However, it achieved poor accuracy for both the training data set: 0.525 and validation data set: 0.431. LightGBM algorithm on the other hand uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value. It achieved a good training accuracy of 1.000, but its validation accuracy of 0.470 was poor.

Across the ensemble learning methods, we faced the problem of the models overfitting to the training data, even after the parameters were optimized to reduce overfitting. Validation accuracy was also low as compared to our neural network models.

## 7 Conclusion

Deep learning has shown to be a useful approach when dealing with high-dimensional data sets. During our evaluation, the CNN model performed the best in predicting protein-ligand binding, as compared to the MLP, LSTM and other ensemble machine learning model. Our final CNN prediction model consists of two convolutional layers with eight 7 by 3 filters followed by 3 fully connected layers of size 64, 64 and 2 and achieved a final accuracy of 0.9150.

## References

[1] Warren L DeLano. Pymol: An open-source molecular graphics tool. *CCP4 Newsletter On Protein Crystallography*, 40:82–92, 2002.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[8] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.