Structural Bioinformatics - Introduction to Python: Project
Prediction of protein-ligand binding sites using deep residual neural networks.

Núria Fàbrega, Marc Ciruela

Contents

1 Introduciton

1.1 Introduction

All biological processes involve interactions between biomolecules, which often induce conformational changes in their structure. Specifically in proteins, these interactions occur at specific sites, with the conformational changes being able to affect the protein function, such as activating or deactivating enzymes, or enabling other types of interactions. As a result, identifying these binding sites is crucial for drug discovery and other applications.

Computational methods for predicting binding sites fall into three main categories: geometric, machine learning, and deep learning methods. Geometric methods, such as Fpocket [1], rely on Voronoi tessellation and alpha spheres to identify binding sites. Machine learning-based methods, such as P2Rank [2], use the random forest algorithm to make predictions. Deep learning methods, such as DeepSite [3] , use 3D-convolutional neural networks to identify binding sites.

1.2 Deep learning methods

While deep learning methods, such as DeepSite, have demonstrated better performance than geometry-based methods, there is still room for improvement [3]. It's worth noting that when it comes to machine learning methods, defining the model and descriptors used are important, but the dataset used for training is equally critical. PUResNet [4] is a deep learning method that operates on similar principles to DeepSite, but focuses on improving prediction accuracy through the use of a more carefully curated training dataset. This dataset is designed to minimize redundancy and maximize information content, which results in more effective training of the model. Building on this approach, our project aims to develop a binding site prediction method that benefits from both the model structure, and the same dataset optimization techniques.

1.3 PUResNet

The training set used to train the PUResNet model has been curated from scPDB, a publicly available dataset of protein structures with their binding sites annotated [5]. scPDB is a selection of ligandable binding sites in protein structures from the Protein Data Bank (PDB). Sites are defined from complexes between a protein and a pharmacological ligand and their data is often used in training and validation of binding site predictions methods, such as in Fpocket, DeepSite or PUResNet.

scPDB contains roughly 18000 structures, each corresponding to a distinct protein-ligand interaction. As some proteins can have more than one binding site or ligand, some structures contain the same proteins with different ligands, protein-protein complexes or different conformations. As such, it is important to filter the data to avoid over-representation of certain proteins that may lead to bias in the model. To achieve this, the authors of PUResNet mapped each structure in the scPDB to their UniProt ID [6] to know the number of structures that were representing a same protein. Afterwards, the Tanimoto coefficient [7] was calculated inside each “cluster” representing the same UniProt ID from the MACCS [8] keys of each structure. If enough similarity resulted from Tanimoto coefficient calculation, a representative for each UniProt ID was selected. When this was not the case, manual inspection using PYMOL was performed [9].

With a curated set of proteins, the protein structure was represented as a 3D image of size 36x36x36x18, with a 3D cube of size 36x36x36Å, with each voxel described based on 9 atomic features, such as hybridization, heavy atoms, heteroatoms, hydrophobic, aromatic, partial charge, acceptor, donor, and ring [4]. The ligand was represented as a 3D image of size 36x36x36x1, with each voxel having a value between 0 and 1 indicating the probability of it being part of the binding site.

This was then given to PUResNet model based on ResNet and U-Net. The model has two parts: the encoder, that takes the input matrix of 36x36x36x18 and compresses the information in to a smaller matrix, and then the decoder, that upscales it to a 36x36x36x1 matrix. The encoder is composed of convolution and identity blocks and the decoder is composed of upsampling and identity blocks. The structure of PUResNet can be seen in Figure S3. The model was optimized through K-fold training and later evaluated using the Coach420 and BU48 datasets. PUResNet showed an improved success rate compared to kalasanty methods (such as DeepSite).

2 Our approach

In this project, we aimed to replicate the approach taken by PUResNet using the data provided in their GitHub repository. However, due to our time and computational constraints, we had to simplify certain steps. In the following sections, we will outline the steps necessary to run the program and obtain binding site predictions.

It is important to note that in machine learning applications, a significant amount of work is invested in developing and training the model, which may not be readily apparent in the final application. Therefore, we will provide a brief overview of the steps we took to develop our model.

2.1 Data preparation: the training set and validation set

We decided to use the scPDB data, the same one used in PUResNet, to train and validate our model. Removing duplicate proteins from the 18000 structures of scPDB, we were left with 16000 unique PDB structures, which were mapped to a Uniprot ID using Uniprot API (https://www.uniprot.org/id-mapping), obtaining around 5600 Uniprot IDs. Then, all PDB structures were clustered according to their Uniprot ID. This process was conducted in a Jupyter Notebook (*Clusterizing scPDB.ipynb*) and with the assistance of another script that connected to UniProt API (*uniprot_maping.py*).

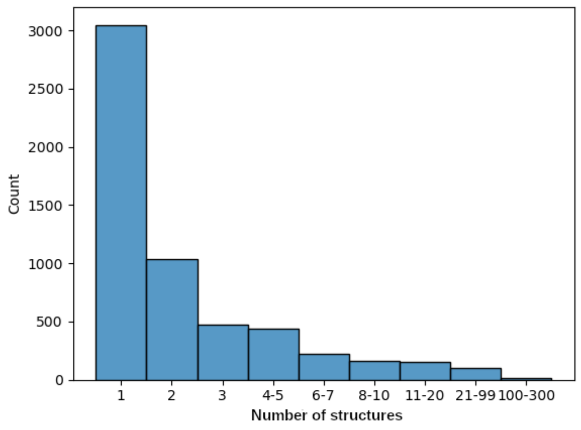

Figure 1: Size of UniProt ID clusters in scPDB database.

As shown in Figure 1, the majority of clusters contained only one protein structure. In these cases, we included all the structures in our next set, as they represented different proteins. However, for clusters containing more than one structure, we calculated the Tanimoto coefficient using Open babel [10], [11] to extract information from the .mol2 files and in-house scripts to calculate the Tanimoto coefficient (*tanimoto_clust.py*). We calculated the average Tanimoto coefficient for each structure in the cluster against the rest of the members. If the minimum mean Tanimoto coefficient for all the members of a cluster was above 0.8, we considered the cluster to have high similarity structures. To select a representative structure of each of those clusters, we chose the one with the largest number of residues. This process resulted in around 680 UniProt IDs with lower similarity between their structures, which we manually inspected using PyMol. Due to time constraints, we could only visualise the proteins with the lowest similarity in each cluster. We found that most of the differences were due to variations in crystallisation, such as the presence or absence of other molecules (like cholesterol) in a complex, or different units of the protein present in the .mol2 file.

In clusters with a minimum mean Tanimoto coefficient below 0.8, the lower similarity between structures does not directly imply that they all have different conformations (as we can foresee, for example, in kinases [12]). In some cases, these dissimilarities were due to bigger but less important changes in the structure (for the purpose of predicting the binding site), such as it being in a complex with a fatty acid, or being in a crystal with several monomers, that masked the minor but more biological relevant changes. In order to take this factor into account and further filter the dataset, a more in-deep analysis of dissimilarity should be conducted, that was outside the scope of this project. Consequently, we decided to apply the same selection that with clusters with higher similarity. Therefore, we selected one representative for each one of these UniProt ID selecting the structures with the biggest number of residues.

With this selection we were left with around 5000 protein structures that needed to be divided into a train and validation set. In order for the validation set to be a good indicator of the prediction power of the model and its ability to predict binding sites in new structures that it had never been trained in, the validation set needed to contain protein structures that were dissimilar to the ones present in the training set. To create these two sets, we first mapped each protein to their annotated family in Uniprot. From the search in Uniprot, around 1700 identifiers for protein families were obtained - for instance, some are a reference to the presence of a certain domain or a conserved region in c-terminal region -. The analysis of each protein family showed that most had less than 50 proteins but that there were a few families with a considerable number of members.

Based on previous work [13] we decided to use as the validation set a random 20% of the proteins present in families with 1-2 members (268 proteins), and using the rest (4809 proteins) as a training set. All information about how these steps were done is in jupyter notebook *Clusterizing scPDB.ipynb*. When other python scripts or different tools such as *sed* or *awk* were used it is conveniently cited.

The training of the model with the set of 4809 proteins resulted in failure, as we realised that our computer did not have enough computational power to handle this task. Therefore, we decided to reduce the training set by randomly selecting one only protein per family, obtaining a total of 1549 proteins to do the training (*Getting 1 prot for fam.ipynb*). This set was further randomly divided into a train and test set, with a 80%-20% ratio (*train_model.ipynb*), with which the model was trained.

Proteins were loaded into the python script by using the openbabel package, which led to several warnings. We inspected them further, as they could lead to potential problems in the training of the model, if they were compromising the extraction of features from the protein structures needed to train it. To check whether there was any issue with openbabel reading the protein structures we evaluated the kinds of warnings that were given. Two types of warnings arised:

- "Failed to kekulize aromatic bonds", present in almost all proteins. This is due to structures in scPDB coming from crystallographic experiments, where aromaticity is not properly annotated.
- "Cannot interpret atom types correctly", with less frequency. Checking manually some cases, we saw that these were usually ions, hydrogens and in general atoms from ligands. To check whether this warning brought problems into the molecule structure, we selected a set of proteins with this warning and checked whether openbabel was capable of transforming them

from mol2 format into PDB format without losing any information. We could confirm that the atoms were properly saved and that the PDB files could be further used with programs such as pdb4amer. We also confirmed that the OBMol object obtained by loading the file with openbabel had all the residues and atoms properly saved.

As none of the warnings seemed to affect the model, none of the proteins were removed from the training.

2.2 Training, validating and predicting with the model

The training of the model was made (and can be replicated) with the jupyter notebook file **train_model.ipynb**. The validation, with the python file **validate.py**, and the prediction with **predict.py**.

All files required to function the files **PUResNet.py** and **data.py.**

- **PUResNet.py** contains the machine learning model used by the authors of the PUResNet paper to train the model. The content of this file is fully attributable to those authors, and we have not made changes in it.
- **data.py** contains the class Featurizer() and the function make_grid(), used to get the features for each of the atoms of the protein, and convert the coordinates into a grid. This file has been downloaded from tfbio.data.package, and minor changes have been made in order to make it work with the last version of the packages it uses.

The code from **train_model.ipynb** and **validate.py** has been developed based on external sources as inspiration, but has not been directly copied from any such sources.

- **train_model.ipynb** first converts the protein.mol2 and cavity6.mol2 of each of the folders in the training set into a numpy array of the correct shape, and then it uses this numpy array to train the model. The input set of proteins is split into a train and test sets with a ratio of 80% - 20%. The model has been trained with Dice Loss as a loss function, as authors of PUResNet found it was the loss function that achieved the best results, and with a batch size of 5 proteins for the same reason. The model has been trained until the validation loss has stopped improving, with a patience of 15 epochs.
- **predict.py** is mainly reused code from PUResNet. However, a new function has been included that calculates the nearby residues to the ligand binding site (those that have any atom fewer than 4Å away from any of the predicted binding site points), and creates a mol2 file to be opened with a visualization program such as chimera.
- **validate.py** uses the best model weights gotten from training the model to predict the binding site of the proteins specified in the validation folder. From each prediction, it calculates:
    - Percentage of proteins where a binding site has been found (as in some proteins, the model cannot find any binding site)
    - Percentage of proteins with predicted binding site with DCC (Distance Center Center) <4Å: where the distance of the centers of the predicted and actual cavities is lower than 4Å

$$Success\ Rate = \frac{Number\ of\ sites\ having\ DCC < 4\text{Å}}{Total\ number\ of\ sites}$$

    - For the proteins with DCC<4Å, it calculates the **DVO:** The Dice Volume Overlap, calculated as the division between the intersection and the union of the predicted and actual binding sites.

$$DVO = \frac{V_{pbs} \cap V_{abs}}{V_{pbs} \cup V_{abs}}$$

- Best prediction (highest DVO) and worst prediction (lowest DCC). The worst prediction, however, can be misleading, as the prediction may be correct, but may be a different binding site than the one present in the v

The final model weights, stored in *model_weights.h5* result from training the model with the 1549 proteins previously commented and the hyperparameters present in the *train_model.ipynb* file. The model was run overnight, and it stopped at the 25th epoch. The graphics of the accuracy and loss of the train and test datasets can be observed in Figures S1 and S2.

3 Results

The model was validated by running it against the validation set of 268 proteins not used for training (defined in section 2.1). The percentage of proteins with at least one predicted cavity (as some got none predicted), the number of proteins whose predictions had a DCC with the actual model lesser than 4Å, and from those, the value for the DVO was calculated, giving the following results:

• Proteins with predicted pockets:	82%
• Predictions with a DCC<4Å:	52%
• From those, mean value for DVO:	44%

Figures 2 and 3 show the best and worst predictions of binding sites, with the best being the one with the highest DVO, and the worst the one with the highest DCC. Red dots are the ones of the prediction, while green dots represent the actual cavity. The best prediction comes from the protein 2ivn (DVO = 0,83) whereas the worst one comes from a protein called 2o1x (DCC = 70.7).

We can see how the cases where the program had more difficulties seem to be related with pockets that usually are more difficult to predict as they are more shallow and in the surface of proteins as in the case seen in Figure 3.

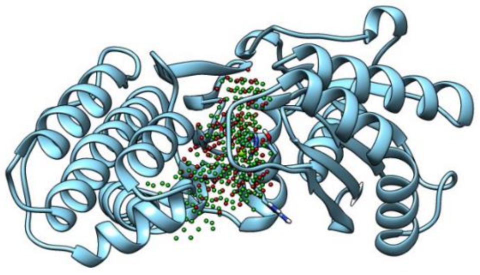

Figure 2. Best prediction of the binding site obtained by the BSPredictor from the validation set (protein 2ivn)

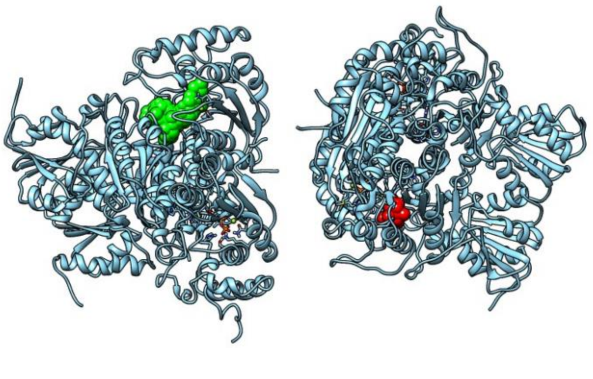

Figure 3. Worst prediction of binding site obtained by the BSPredictor from the validation set (protein 2o1x)

3.1 Comparison with other methods

To further assess the performance of our model, we compared it to PUResNet – the one attempted to replicate – and FPocket – as a standard geometry-based prediction method. This assessment was made by plotting the DCC values against the success rate (Figure 4). Success rate was defined as the true positives divided by the total number of proteins, with the true positives being the pockets considered to be predicted properly as their DCC is below the threshold value.

The assessment was first done using the validation set created for our model. In Figure 4A It can be observed that our model's performance is slightly worse than Fpocket and PUResNet, as its curve is below those of Fpocket and PUResNet for DCC values < 4Å (which is what we consider to be the threshold for properly predicted pockets). But even so, the model behaves very consistently with what we could expect, having a good early enrichment (similarly to PUResNet and better than Fpocket).

To better compare those methods, we also conducted calculations for an independent set of proteins, Coach420. We have chosen Coach420 as it is one of the independent sets that are also used in PUResNet [4]. Coach420 is a set of 298 protein structures from PDB with their ligand separated. In this case, however, each protein folder does not also contain a file with their annotated cavity. This implies that the DCC values can't be directly compared with the ones from the previous validation set, as in this case the predicted cavity is compared with the ligand, whereas in the previous case the predicted cavity was compared with the real one. Figure 4B shows how the performance is very similar to the observed in scPDB validation set - albeit a little bit lower due to ligand-cavity being compared instead of cavity-cavity – further validating the previous results. Having obtained a similar performance with a different dataset is important, because as scPDB is the dataset used for training and also validating the model, it could be possible that we introduced some bias, even if the structures in the two sets were ensured to be dissimilar. This way we are confirming the results with a completely different set. Another aspect to notice is the fact that our model obtained fairly similar performance than the geometry method and the PUResNet method, which we consider to be great results considering the limitations of this work.

To better summarise the results obtained, Table 1 shows the success rates for a DCC of 4 Å for each of the 3 compared methods and the 2 validation sets.

Table 1: Success rates for each of the studied methods in scPDB validation set. These success rates are using DCC < 4 Å as threshold to consider a right prediction of a pocket and therefore calculated as: pockets predicted with DCC < 4 Å vs total number of pockets to predict.

Method	**scPDB validation set Success rate (DCC < 4 Å)**	**coach420 validation set Success rate (DCC < 4 Å)**
BSPredictor	0.685	0.542
PUResNet	0.733	0.548
Fpocket	0.729	0.564

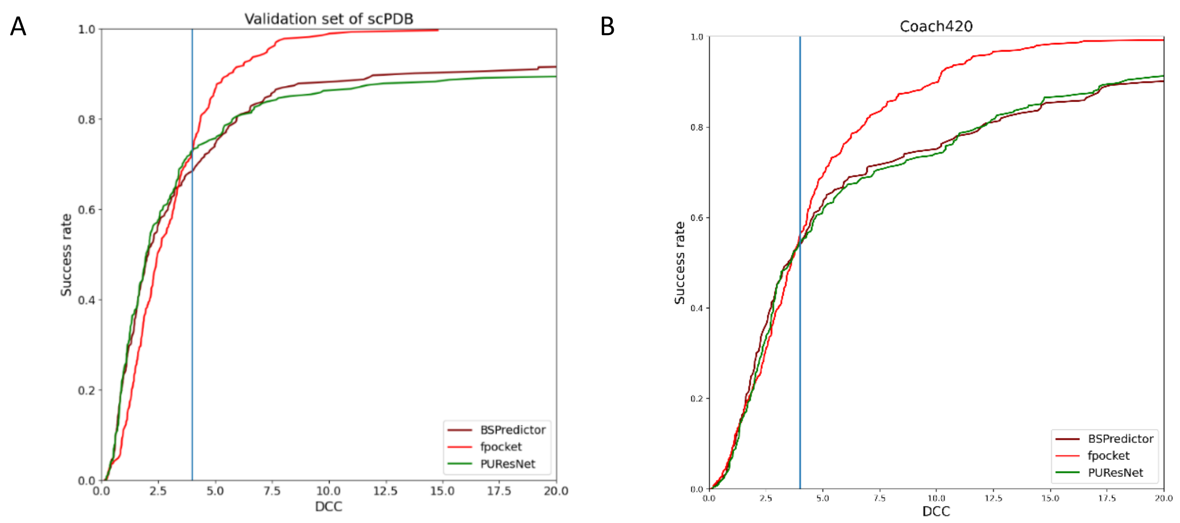

Figure 4. Value of DCC (Distance Center Center) against the success rate: the number of proteins with lower DCC: than the value, for BSPredictor, FPocket and PUResNet. A: usage of the validation set created for the model. B: usage of the Caoch420 validation set

4 Conclusions

Our project has given further evidence that training a binding site predictor requires a combination of a high-quality model architecture and a carefully curated dataset. We found that the PUResNet model performed well in predicting binding sites, but that the quality of the data used for training was just as important. Specifically, we showed that training the model with a curated set of 1700 non-redundant proteins resulted in similar performance to the final weights of the PUResNet paper, which were trained with 5000 proteins. Moreover, our study highlights the importance of computational resources in machine learning research, as limitations in computational power can pose significant challenges to training complex models.

Our findings suggest that future research in this area should focus on further improving the efficiency of machine learning models for binding site prediction, while also exploring new ways of curating high-quality datasets. One potential avenue for future research could be to incorporate other sources of data, such as protein-protein interaction networks or structural information, to improve the accuracy of binding site prediction. Additionally, it will be important to continue developing approaches that are more computationally efficient, as this will allow for larger and more diverse datasets to be used for training. Overall, our project provides valuable insights into the challenges and opportunities for advancing the field of protein-ligand interaction prediction.

5 Tutorial

5.1 Download the model

The model can be used by cloning the github repository https://github.com/macija/sbi_project. Inside the folder BSPredictor, the file *train_test_families_1rep_best_weights.h5*, which contains the model weights, weights 160MB and will not be downloaded by simply cloning the github repository. We recommend downloading this file manually from the repository, and situating it into the BSPredictor folder in your computer.

5.2 How to do a prediction using the trained model

First, all dependencies need to be installed. We recommend using conda and create a conda environment with:

- python 3.10
- openbabel
- numpy
- keras
- skimage

Dependencies can be installed using the commands:

```
conda create -n "myenv" python=3.10
conda install -c conda-forge keras
conda install -c conda-forge tensorflow
conda install -c anaconda scikit-learn
conda install -c conda-forge openbabel
```

FIles data.py, PUResNet.py and predict.py need to be present in the same directory.

Predict.py should be used to run predictions and it can take as arguments:

```
predict.py  [-h]  --file_format  FILE_FORMAT  --mode  MODE  --
input_path
INPUT_PATH [--output_format OUTPUT_FORMAT] [--output_path
OUTPUT_PATH] [--gpu GPU]
```

With a pdb or mol2 file the prediction would be run as follows:

```
python predict.py –file_format mol2 –mode 0 –input_path
myprotein.mol2
```

But the program can perform batch prediction for several proteins using -mode 1. The output path can be given with –output_path and gpu usage by tensorflow can be established with –gpu.

List of complete options:

```
--help, -h              show this help message and exit
--file_format FILE_FORMAT, -ftype FILE_FORMAT
                        File Format of Protein Structure like:
                        mol2,pdb..etc. All file format supported by
                        Open Babel is supported (default: None)
--mode MODE, -m MODE    Mode 0 is for single protein structure. Mode
                        1 is for multiple protein structure (default:
                        None)
--input_path INPUT_PATH, -i INPUT_PATH
                        For mode 0 provide absolute or relative path
                        for protein structure. For mode 1 provide
                        absolute or relative path for folder
                        containing protein structure (default: None)
--output_format OUTPUT_FORMAT, -otype OUTPUT_FORMAT
                        Provide the output format for predicted
                        binding side. All formats supported by Open
                        Babel (default: mol2)
--output_path OUTPUT_PATH, -o OUTPUT_PATH
                        path to model output (default: output)
--model_weights, -mw    path to model weights
                        (default: model_weights.h5)
--gpu GPU, -gpu GPU     Provide GPU device if you want to use GPU
                        like: 0 or 1 or 2 etc. (default: None)
--verbose, v            Activate OpenBabel warnings
```

When the prediction finishes, two types of files are written: pocketX and site_pocketX. pocketX can have the format that we have defined in the program arguments. By default it will be a mol2. This describes with a series of points the predicted cavity. site_pocketX will have the same format as pocketX and describes the residues around that cavity for further analysis and comprehension of the binding site by the user. There will be as many files (X) as binding sites have been found by the program.

5.3 How to generate the user's own models

If a user wishes to generate their own deep learning models using the same approach followed in this project, they can train a new model with other data. First, the user should provide a directory with subdirectories for each protein. Each subfolder should contain a file for the protein structure and another one where the cavity is defined. A ligand can be used instead but we recommend the use of grids filling the cavity for better performance.

The model can be trained by adapting the jupyter notebook *train_model.ipynb* to the needs of the user. If it wants to be used, protein files should be called "protein.mol2" and cavity files "cavity6.mol2". If they have different names, the function *get_training_data* from the file *train_functions.py* should be changed to the appropriate names. The path of the folder must be indicated for the variable *data_folder_path*. This path will also be used to save the numpy training data (to be able to train the model quicker if it was to be run a second time with the same data), the best weights of the model, and the plots of the accuracy and loss of the model (such as Figures S1 and S2).

To do predictions using your trained model, *predict.py* can be used by using the same commands as described, above, but adding the path of the model with the argument `--model_weights` or `-mw`

6 References

[1] V. Le Guilloux, P. Schmidtke, and P. Tuffery, ‘Fpocket: An open source platform for ligand pocket detection’, *BMC Bioinformatics*, vol. 10, no. 1, pp. 1–11, May 2009, doi: 10.1186/1471-2105-10-168/TABLES/1.

[2] R. Krivák and D. Hoksza, ‘P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure’, *J. Cheminformatics*, vol. 10, no. 1, p. 39, Aug. 2018, doi: 10.1186/s13321-018-0285-8.

[3] J. Jiménez, S. Doerr, G. Martínez-Rosell, A. S. Rose, and G. De Fabritiis, ‘DeepSite: protein-binding site predictor using 3D-convolutional neural networks’, *Bioinformatics*, vol. 33, no. 19, pp. 3036–3042, Oct. 2017, doi: 10.1093/BIOINFORMATICS/BTX350.

[4] J. Kandel, H. Tayara, and K. T. Chong, ‘PUResNet: prediction of protein-ligand binding sites using deep residual neural network’, *J. Cheminformatics*, vol. 13, no. 1, p. 65, Sep. 2021, doi: 10.1186/s13321-021-00547-7.

[5] J. Desaphy, G. Bret, D. Rognan, and E. Kellenberger, ‘sc-PDB: a 3D-database of ligandable binding sites—10 years on’, *Nucleic Acids Res.*, vol. 43, no. D1, pp. D399–D404, Jan. 2015, doi: 10.1093/nar/gku928.

[6] The UniProt Consortium, ‘UniProt: the universal protein knowledgebase in 2021’, *Nucleic Acids Res.*, vol. 49, no. D1, pp. D480–D489, Jan. 2021, doi: 10.1093/nar/gkaa1100.

[7] D. Bajusz, A. Rácz, and K. Héberger, ‘Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations?’, *J. Cheminformatics*, vol. 7, no. 1, p. 20, May 2015, doi: 10.1186/s13321-015-0069-3.

[8] J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse, ‘Reoptimization of MDL Keys for Use in Drug Discovery’, *J. Chem. Inf. Comput. Sci.*, vol. 42, no. 6, pp. 1273–1280, Nov. 2002, doi: 10.1021/ci010132r.

[9] L. L. C. Schrödinger and W. DeLano, ‘PyMOL’. [Online]. Available: http://www.pymol.org/pymol

[10] N. M. O’Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, ‘Open Babel: An Open chemical toolbox’, *J. Cheminformatics*, vol. 3, no. 10, pp. 1–14, Oct. 2011, doi: 10.1186/1758-2946-3-33/TABLES/2.

[11] N. M. O’Boyle, C. Morley, and G. R. Hutchison, ‘Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit’, *Chem. Cent. J.*, vol. 2, no. 1, p. 5, Mar. 2008, doi: 10.1186/1752-153X-2-5.

[12] Y. Pan and M. M. Mader, ‘Principles of Kinase Allosteric Inhibition and Pocket Validation’, *J. Med. Chem.*, vol. 65, no. 7, pp. 5288–5299, Mar. 2022, doi: 10.1021/acs.jmedchem.2c00073.

[13] D. Repecka *et al.*, ‘Expanding functional protein sequence spaces using generative adversarial networks’, *Nat. Mach. Intell.*, vol. 3, no. 4, Art. no. 4, Apr. 2021, doi: 10.1038/s42256-021-00310-5.

7 Supplementary information

Graphics of model accuracy and loss over the time. Model stopped training

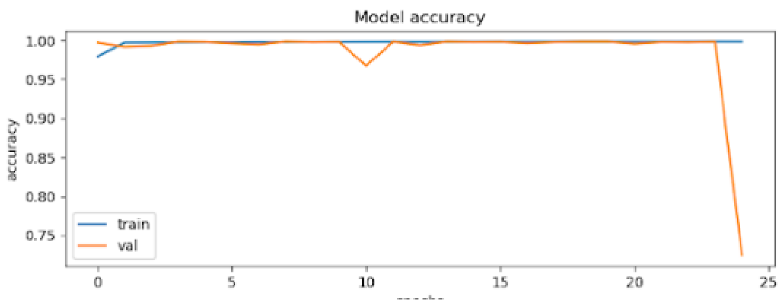

Figure S1. Train and Test model accuracy throughout the epochs

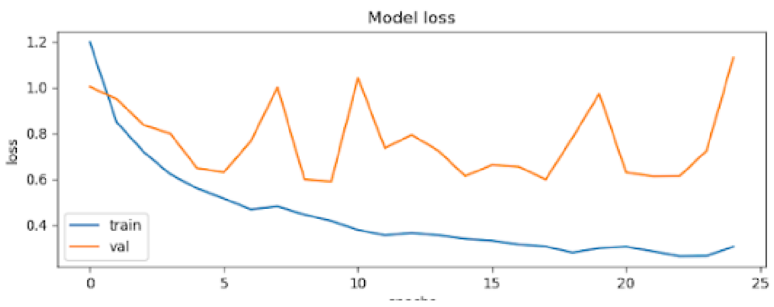

Figure S2. Train and test model loss throughout the epochs

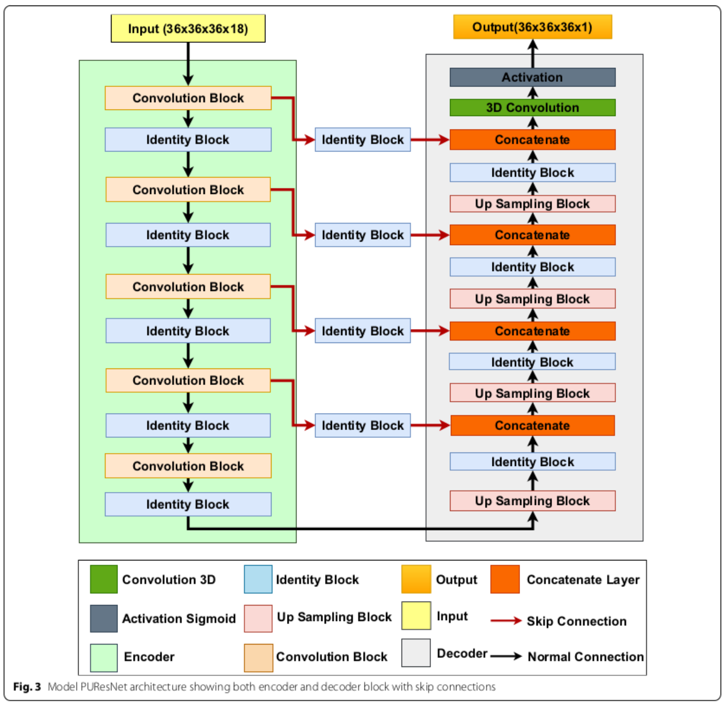

Figure S3. PUResNet structure. Image taken from PUResNet article[4]