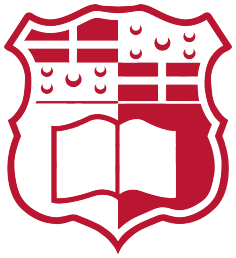


Deep-Learning Approaches in Structure Based Drug Design for Binding Affinity Prediction

Joseph Azzopardi

Supervised by Dr Jean-Paul Ebejer

Department of Artificial Intelligence
Faculty of ICT
University of Malta

April, 2021

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. AI .

WWW.UM.EDU.MT

First edition, Monday 12th April, 2021

L-Università ta' Malta

Declaration by Postgraduate Students

(a) Authenticity of Dissertation

I hereby declare that I am the legitimate author of this Dissertation and that it is my original work.
No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.
I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

(b) Research Code of Practice and Ethics Review Procedures

I declare that I have abided by the University's Research Ethics Review Procedures.
As a Master's student, as per Regulation 58 of the General Regulations for University Postgraduate Awards, I accept that should my dissertation be awarded a Grade A, it will be made publicly available on the University of Malta Institutional Repository.

Faculty/Institute/Centre/School	Faculty of ICT
Degree	M.Sc. AI
Title	Deep-Learning Approaches in Structure Based Drug Design for Binding Affinity Prediction
Candidate (Id.)	Joseph Azzopardi (474488M)

Signature of Student	_____
Date	Monday 12th April, 2021

08.02.2018

To my wife, Candy

for the constant motivation, positivity, support, and endless love.

Acknowledgements

I would like to thank and show my respect and appreciation to my supervisor Dr Jean-Paul Ebejer for his great dedication, work ethic, enthusiasm, and commitment throughout the whole journey. He has guided me through various interesting discussions that have motivated me to bring out the best in me and pursue the best outcome possible. His great talent to impart knowledge has spawned a curiosity to research this domain, and his knowledge, ideas, and support were key to steady progress to finish this dissertation.

My special thanks, goes to the team at AWS Research Credits Team for supporting our research and providing substantial credits to develop our models and run our experiments, without which our work would have been much more difficult.

Last but not least, I would like to thank my whole family and close friends as a source of motivation and encouragement; My wife, Candy for her love, patience, and support, during the challenging times to balance work, university, and late nights, whilst also raising our newborn baby angel Jasmine. I would like to take the opportunity to thank my parents Charles and Mary for their never ending love, that shaped me in the person I am today, and for keeping me in their thoughts and prayers.

Abstract

Scoring functions (SF) are the heart of structure based drug design, where they are used to estimate how strongly the docked pose of a ligand binds to the target. Seeking a SF that can accurately predict the binding affinity is key for successful virtual screening methods. Deep learning (DL) approaches have recently seen a rise in popularity as a means to improve the SFs having the key advantage to automatically extract features and create a complex representation of the data without feature engineering and expert knowledge.

In this study we present LigityScore, a rotationally invariant SF based on convolutional neural networks (CNN). LigityScore descriptors are extracted directly from the structural and interacting properties of the protein-ligand (PL) complex which are input to a CNN for automatic feature extraction and binding affinity prediction. This representation uses the spatial distribution of Pharmacophoric Interaction Points (PIPs), derived from interaction features from the PL complex based on pharmacophoric features conformant to specific family types (HBA, HBD, etc) and distance thresholds. The data representation component and the CNN architecture together, constitute the LigityScore SF. We define two variants of LigityScore — LigityScore1D considers a single distance from each combination of two PIPs from the extracted ligand and protein PIP pools, and generates a feature vector for each pharmacophoric family pair (example HBA-HBD) based on the distribution of the PIP pair distances in discretised space. The different pharmacophoric family pair combinations are stacked to construct a matrix representation of the complex. Similarly, LigityScore3D consider a 3-PIP combinations to create a triangular structure with three distances between the PIPs, which are discretised to represent binning coordinates for a 3D feature hypercube collection per pharmacophoric family set (example HBA-HBD-HBD) describing the distribution of distances in 3D space.

The main contribution for this study is to present a novel PL representation for use as a CNN based SF for binding affinity prediction. LigityScore models are evaluated for scoring power on the latest two CASF benchmarks. The Pearson correlation coefficient, and the standard deviation in linear regression were used to compare LigityScore with the benchmark model, and also other models in literature published in recent years. LigityScore3D has achieved better overall results and showed similar performance in both CASF benchmarks. LigityScore3D ranked 5^{th} place in the CASF-2013 benchmark, and 8^{th} in CASF-2016, with an average R-score performance of 0.713 and 0.725 respectively. LigityScore1D obtained best results when trained using the PBDbind v2018 dataset, and ranked 8^{th} place in the CASF-2013 and 7^{th} place in CASF-2016 with an R-score performance of 0.635 and 0.741 respectively. Our methods show relatively good performance that exceed the Pafnucy performance, as one of the best performing CNN based SF, on the CASF-2013 benchmark, using a less computationally complex model that can be trained 16 times faster.

Contents

List of Figures

List of Tables

List of Abbreviations

CADD Computer Aided Drug Design
VS Virtual Screening
DTI Drug Target Interactions
HTS High-throughput Screening
HBD Hydrogen-bond Donors
HBA Hydrogen-bond Acceptors
LBVS Ligand Based Virtual Screening
SBVS Structure Based Virtual Screening
DBVS Docking Based Virtual Screening
PIP Pharmacophoric Interaction Points
FDA US Food and Drug Administratiom
PL Protein Ligand
SF Scoring Function
PDB Protein Data Bank
CASF Comparative Assessment of Scoring Functions
NN Neural Networks
FF Feed Forward
DL Deep Learning
ANN Artificial Neural Networks
DNN Deep Neural Networks
RNN Recurrent Neural Networks
CNN Convolution Neural Network
FC Fully Connected

ReLU Rectified Linear units
tanh Hyperbolic Tangent
PReLU Parametric Rectified Linear Unit
SGD Stochastic Gradient Descent
RMSE Root Means Squared Error
MAE Mean Absolute Error
SD Standard Deviation in Linear Regression
AMI Amazon Machine Image
AWS Amazon Web Services
EC2 Elastic Cloud Computing
GPU Grpahics Processing Unit
GBT Gradient Boosting Trees
IFP Interaction FingerPrint
PLEC Protein-Ligand Extended Connectivity
ECFP Extended-Connectivity FingerPrints

1

Introduction

Proteins are complex molecules made up of combinations of naturally occurring amino acid chains. Proteins perform many critical functions in the body such as facilitate chemical reactions, provide structure and support for cells, enable signaling, and perform regulation functions. Drug discovery is a process used to identify small molecules which elicit the desired biological response when they bind to target proteins. This bioactivity in the form of a chemical reaction either activates or inhibits a biological function thus acting as a drug to the body. Figure 1.1 illustrates the concept of the small molecule binding to the protein representing the lock and key analogy. The small molecule has the right fit for binding with the protein at its binding site. A small molecule that physically binds to the protein is termed as a *ligand*.

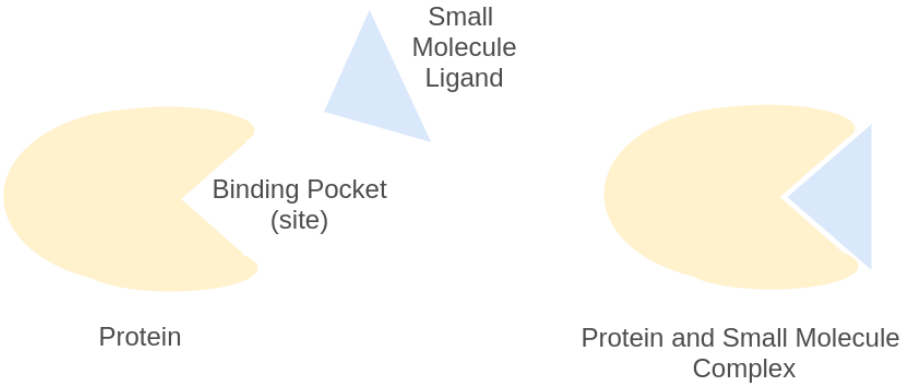


Figure 1.1: Protein and Small Molecule Binding using Lock and Key Analogy.

Computer-Aided Drug Design (CADD) methods have seen a rise in popularity for drug discovery processes in recent years due to the unprecedented amount of data available together with the advent of powerful computational tools available such as cloud

computing platforms (Rifaioglu et al., 2018). CADD methods have recently been successfully applied in various stages of the drug development pipeline. These include hit identification and lead optimisation (Ma et al., 2015; Ragoza et al., 2017), *binding affinity prediction* (Jiménez et al., 2018; Stepniewska-Dziubinska et al., 2017), *ADMET* (absorption, distribution, metabolism, excretion, and toxicity) properties' prediction such as the work done by Mayr et al. (2016) on toxicity prediction, and also *de novo* (Gómez-Bombarelli et al., 2018) methods.

The chemical space is huge and it is estimated that there are 10^{60} molecules that are relevant for drug discovery (Kirkpatrick and Ellis, 2004). It is difficult to clinically test large chemical spaces for potential Drug Target Interactions (DTI) and therefore computational methods such as Virtual Sceening (VS) can be used as the first filtering process to test activity between protein target and small molecules (Cheng et al., 2012). High-throughput screening (HTS) is a widely used approach for hit and lead generation across pharmaceutical companies for the early stages of drug development, as it provides the ability to perform biochemical assays on millions of molecules. However, its low success rate, high cost, and lack of availability for advanced laboratories capable of HTS experiments, have motivated the industry and researchers to seek out *in silico* computational methods for VS (Cheng et al., 2012). The high level process of drug design is illustrated in Figure 1.2, where the VS methods are shown to be applied before HTS as the first step to filter out unlikely drug-target pairs. HTS experiments are expensive and time consuming so filtering out unlikely molecules will accelerate and reduce this costly process. Hopkins (2009) have estimated that the cost to develop a single drug was around 1.8 Billion US dollars over a time span of 13 years on average (Rifaioglu et al., 2018).

Computational methods in drug design can thus assess potential DTI and provide the ability to screen libraries that include millions of compounds, which would otherwise be unfeasible. VS is applied instead of HTS as illustrated in Figure 1.2 for hit identification, however in some cases HTS is also applied after VS for further filtering and *lead* molecule generation (Rifaioglu et al., 2018). After lead optimisation, the potential drug would need to pass a series of rigorous clinical trials before being finally approved.

In this dissertation we apply convolution deep learning models to enhance the binding affinity prediction for scoring functions applied to a protein-ligand complex in VS. Our approach introduces two techniques, *LigityScore1D* and *LigityScore3D* that are based on the Ligity method (Ebejer et al., 2019), that make use of important structural features of both the protein and ligand to create a suitable data representation of the protein-ligand complex. This representation is then used to feed the convolution neural network

to train a model as a scoring function for prediction of binding affinity.

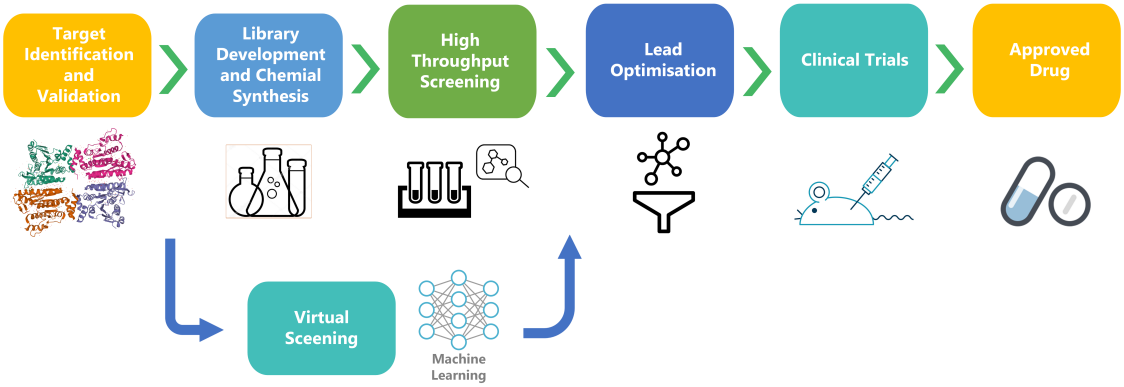


Figure 1.2: High Level Drug Design Process showing virtual screening components. Adapted from Rifaioglu et al. (2018).

1.1 | Virtual Screening

Virtual Screening (VS) is one of the CADD methods that are used to predict chemical activity of small molecules on target proteins *in silico*. VS is considered a critical early step in the drug design process for the discovery of innovative leads (Cheng et al., 2012). VS is practically the computational equivalent to biological screening, and is used in the same way to sift molecule libraries and identify potential molecules that will show biological response. The VS algorithm would typically *rank* the library of molecules under test based on their biological activity, using the binding affinity half-concentration constant, IC_{50}. IC_{50} is a quantitative measure showing the concentration required to inhibit 50% of the biological activity of the protein. Molecules which are below the designated IC_{50} threshold are potential active molecules also known as *hits*. A smaller value of IC_{50} represents a higher binding affinity since a smaller concentration is required to sustain the protein-ligand interaction. Based on the work of Wójcikowski et al. (2017) an $IC_{50} < 25\mu M$ shows bioactivity. The other molecules, which are above threshold, are referred to as *inactives* as they show little biological response and therefore are discarded. These *hits* are allowed to progress to the next phase of drug development for further testing and *lead* molecule identification.

VS is a cheaper and faster alternative to HTS, however it is used as a complementary approach as illustrated in Figure 1.2. Machine learning techniques can be used to generate models for predicting bioactivity. Figure 1.3 shows the typical process involved in a

feature-based VS method which include dataset collection, pre-processing of molecules, feature extractions, followed by training and evaluation of the predictive model.

VS is commonly divided into three main approaches, namely i) Structure-based Virtual Screening (SBVS), ii) Ligand-Based Virtual Screening (LBVS), and iii) Hybrid VS methods where multiple approaches are combined together.

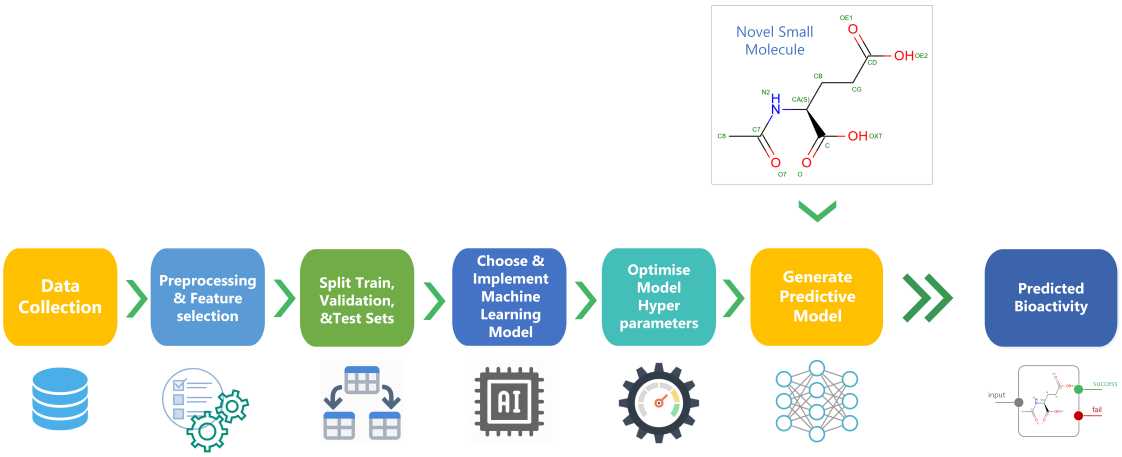


Figure 1.3: Virtual Screening Process for Machine Learning Predictive models. Adapted from Rifaioglu et al. (2018).

1.1.1 | Structure Based Virtual Screening

Structure-based drug design uses the known 3D protein structure to apply computational methods that measure the ability of a small molecule to bind to the target protein structure (Ching et al., 2018). SBVS requires in-depth structural details of the target proteins made available experimentally by advancements in recent years in X-ray crystallography and nuclear magnetic resonance spectroscopy, or by computational modelling (Cheng et al., 2012).

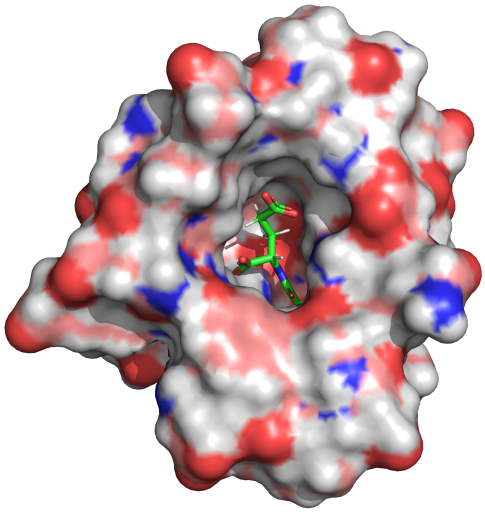
A docking program is normally an integral part of SBVS and is used to validate the ability of the small molecule to bind to the target structure in a typical *'lock and key'* (Figure 1.1) fashion (Ching et al., 2018). During the docking process as many possible ligand orientations, or conformers, are iteratively validated at the binding site to find a suitable ligand pose yielding the best binding affinity. The binding affinity and the strength of the particular pose is determined by the scoring function of the docking program. The docking algorithm needs to tackle a number of challenges for successful simulation of ligand-target interaction such as i) the role of water molecules at the binding site, ii) protein and ligand flexibility (ability to change molecules' orientation at

rotational bonds), iii) metal ions, and iv) the effectiveness of the scoring function used to assess the fitness of the docked pose. From the above mentioned challenges, flexibility of the molecules creates additional complexity as they involve many degrees of freedom for translation and rotation creating a huge number of potential poses. This type of virtual screening is also termed docking-based virtual screening (DBVS) and is arguably the most applied method in SBVS (Cheng et al., 2012). Amongst the most popular docking methods mentioned by Tuccinardi (2009) there are Autodock (Trott and Olson, 2010), Autodock4 (Morris et al., 2009), Dock (Ewing et al., 2001), Glide (Friesner et al., 2004) and Gold (Jones et al., 1997).

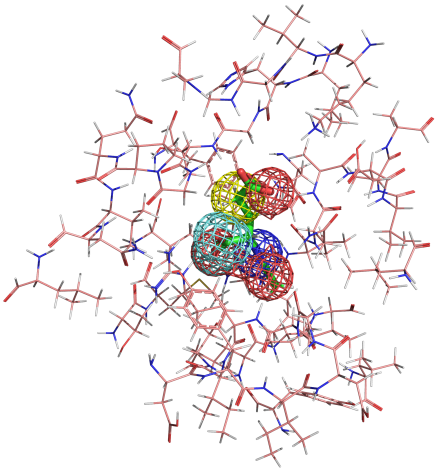
1.1.2 | Ligand Based Virtual Screening

Ligand Based Virtual Screening (LBVS) is used when only the active molecule is known. The properties of known ligands are used to search for similar molecules and LBVS relies on the principle that *similar* molecules should also produce activity to some extent. The open question in LBVS is how to define and measure similar molecules, as the similarity is dependent on the representation used for the molecules. In LBVS the most popular methods for molecule representation include fingerprints, pharmacophore modelling, QSAR (Quantitative Structure-Activity Relationship), and Ultra-Fast Shape Recognition (USR) Ebejer et al. (2019). Fingerprints represent a vector of features describing the ligand and can be in the binary form to represent presence or absence of the ligand features such as presence of donors. Fingerprints can also include non-binary terms such as counts of features such as aromatic rings, or rotatable bonds.

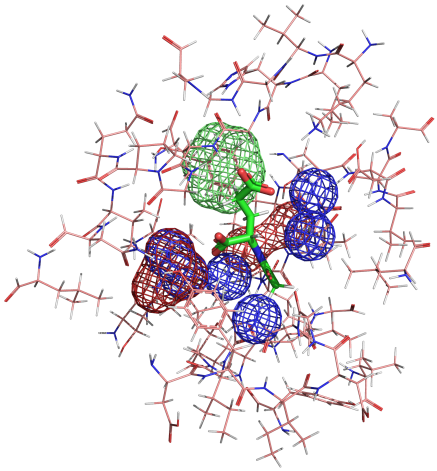
A key aspect used in this study was derived from pharmacophore modeling molecule representation inspired by the work of Ebejer et al. (2019). As described by Gund (1977), a *pharmacophore* is "a set of structural features in a molecule that are recognised at the binding site and is responsible for that molecule's biological activity". Therefore a pharmacophore model represents a number of general structural features such as hydrogen-bond donors (HBD), hydrogen-bond acceptors (HBA), or hydrophobic regions within the molecule, which are used to identify the features at the binding site that are responsible for molecular binding and biological activity. These features at the binding sites may be used to find strong molecular binding interactions or *hot-spots* in the molecule which are used to extract *fingerprint* descriptors to represent the molecule. As indicated by Leach et al. (2010) these features can be used in a 3D pharmacophore model and the spatial relationship between these pharmacophoric features can also be used to represent the molecule. This representation for using pharmacophore keys (or fingerprints) can be extended to include also the protein pockets Leach et al. (2010). An example



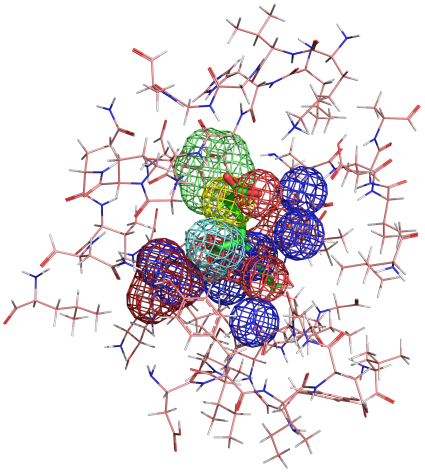
a) 3ZZF Target and NLG Ligand



b) 3ZZF Target and NLG Ligand with Hot-Spots



c) 3ZZF Target with Hot-Spots and NLG Ligand



d) 3ZZF Target and NLG Ligand with Hot-Spots

Figure 1.4: Protein and Ligand Pharmacophore Hot-Spots used in LigityScore. Figure (b) shows hot-Spots for the ligand only whilst, Figure (c) shows those at the protein only. Their combination, as shown in Figure (d) will be used to extract unique combinations of Pharmacophoric Interaction Points (PIPs). Each coloured mesh represents a different pharmacophore family type. For example the blue mesh represents *Donor* atoms, whilst the red mesh represent *Acceptor* atoms.

of pharmacophoric hotspot features are illustrated in Figure 1.4 for the *3ZZF* protein, and *NLG* ligand. Each coloured mesh surrounding different groups of atoms represent a different hot-spot features; for example the red atom mesh represent *Acceptors*, blue

atom represent *Donor* atoms, whilst green atom mesh represent a *Lumped Hydrophobe*. This combination of fingerprints of both ligand and proteins was used as the basis of the feature representation in our approach and is described in detail in Section 3.

Our approach can be categorised as *Hybrid* since a ligand based technique is adapted for a structure based technique to to find a suitable representation of the protein-ligand (PL) complex to construct a scoring function for docking in SBVS.

1.1.3 | Scoring Functions

A scoring function (SF) is an important component in structure-based drug design and can be defined as "Estimating how strongly the docked pose of a ligand binds to the target" (Ain et al., 2015). The scoring function is a crucial component to docking programs as it tells us how good the small molecule binding is. Therefore, optimising the ability of the scoring function to assess the binding capability between a protein and a ligand, helps improve the performance of docking programs, which in turn improves the effectiveness of drug discovery in SBVS methods. The work in this dissertation focuses directly on the scoring function layer of SBVS and tackles this problem by applying a Convolution Neural Network model to implement the algorithm of the scoring function.

The prediction of binding affinity is a regression problem where the model learns the salient information of the input features to output a value of binding affinity. The binding affinity prediction model can then be used in SBVS for classification of the small molecule as inactive or active. Although computational methods has been used in drug design for over three decades, accurate prediction of binding affinity still remains an open problem in computational chemistry (Ching et al., 2018).

The binding affinity constants such as *half-concentration* (IC_{50}), *dissociation* (K_d), or *inhibition* (K_i) are used to measure the binding affinity. Smaller values for these constants result in a stronger binding affinity and the more probable that the small molecule is able to bind the protein. Typically, a small molecule is termed as *binding* when the affinity is $1\mu M$ or better (Wójcikowski et al., 2017). Therefore a SF is typically used to screen millions of small molecules as a *"quick and dirty"* approach to rank the top PL interactions for lead generation.

The SF is used in the following areas for hit and lead discovery and optimization in SBVS and is considered the foundation in structure-based drug design (Ragoza et al., 2017):

1. **Pose prediction:** Predict binding mode or best binding pose of the ligand for pose

generated as part of the Molecular Docking process. Each ligand pose is evaluated at the binding site, and the pose with the best binding affinity is selected. Therefore the scoring function is as described by (Cheng et al., 2012) the "heart of molecular docking".

2. **Ranking:** Ranking of molecules of known pose in order of the binding strength for a given protein target.

3. **Classification:** classification if a small molecule is active or inactive for a given target protein.

Classical SF, that do not include any form of machine-learning, such as the popular Autodock Vina (Trott and Olson, 2010) use the SF function for all of the above areas. In this study the scoring aspect in SBVS will be considered and analysed to be enhanced using machine-learning (ML), specifically using deep-learning (DL) techniques.

1.2 | Motivation

Scoring functions are typically used for fast evaluation of protein-ligand interactions, so building an efficient and powerful SF is a means of accelerating the virtual screening process. Intense research has been done over the years on this problem, however improving the accuracy for binding affinity prediction is shown to be a non-trivial task (Ain et al., 2015).

Optimizing and developing a binding affinity prediction SF is still an open problem with many challenges. One of the main challenges is the huge chemical space available. In order to quantify and qualitatively compare the huge chemical space, Valler and Green (2000) compared the approximate estimate of possible drug molecules (10^{60}), to the number of seconds elapsed since the Big Bang which is *only* 10^{17} seconds. This massive space makes even the most ambitious molecular sampling companies, fated to a hopeless endeavour using HTS and also ultra-HTS techniques (Valler and Green, 2000). This has shifted the focus to computational methods and the use of ML and DL Techniques.

Although there is a huge chemical space available, Rifaioglu et al. (2018) claim that there are only around 9,600 small molecule FDA (US Food and Drug Administration) approved drugs. Therefore there is a huge opportunity for discovery of potential new drug leads within this huge chemical space, but at the same time confirms the difficulty of the drug discovery process. The current computational power is still not sufficient for an exhaustive search in all chemical space as discussed by Kirkpatrick and Ellis (2004).

Another challenge deals with the limited data available in the current datasets. Molecular datasets such as PDBbind (Liu et al., 2017b), only include a small subset of the whole chemical space. Thus, such datasets are not a true representation of the whole chemical space. However, Kirkpatrick and Ellis (2004) argued that most of this space is actually not of biological interest, and therefore the current known chemical space can be used to search for other similar molecules since we know these can bind. However, Kirkpatrick and Ellis (2004) concluded that new molecules are still hard to find.

Additionally, the current datasets such as the PDBbind (Liu et al., 2017b) lack information related to experimentally tested inactive molecules. As commented by Sieg et al. (2019), there is limited documentation on experimentally validated inactives. For this reason, some artificially generated decoys are used to simulate inactive molecules. Although these are not verified most of the decoys created are assumed inactives, although these might add *false negative bias* (Sieg et al., 2019). Due to this lack of documentation there are no datasets that include the binding affinity of an inactive molecules. This creates an additional challenge since it is difficult to include inactive molecules in the SF training since binding data is not available. However such challenges lend themselves to huge opportunities to drive and motivate further research to explore this chemical space in the quest for drug discovery.

Our work on the development of the SF is based on Deep learning (DL) techniques. DL is a class of machine learning algorithm that are composed of multiple layers of artificial neural networks (ANN). DL methods aims to reduce feature engineering or feature selection and automatically extract the salient feature information from the input data, provided it is a suitable representation of the molecular interactions between the protein and ligand (Pérez-Sianes et al., 2019). Therefore, the strategy is to allow deep learning models to *learn* the underlying physics of the molecular interactions so that this learned information can be reapplied to other protein targets for exploration of a novel ligands, without the need to incorporate expert chemical knowledge. The shift from human expert knowledge to automatic feature extraction make such computational methods within reach to more researchers which provides accelerated improvements in this area as witnessed in the last years, whilst also improving on the ability to ingest more and more data for VS.

DL models have achieved remarkable success in various areas such as computer vision (Szegedy et al., 2015) and natural language processing (Goldberg, 2016). Inspired by this work, the use of DL and in particular CNN have naturally become an obvious strategy to apply for CADD as well (Chen et al., 2018). Deep Learning for drug discovery was first introduced during the 2012 Merch Kaggle challenge (Kag, 2012). This was later on published by Ma et al. (2015) where the authors introduced a Deep Neural

Network (DNN) using multi-task deep ANN for QSAR LBVS. Ma et al. (2015) achieved for the first time better results than the standard Random Forest (RF) models with a relative improvement of 15%, which was the best performing ML model in VS before introduction of DL techniques.

Since protein-ligand complexes are 3D structures, their representation can include multi-dimensional tensors. CNNs are known to process multi-dimensional inputs, such as colour images, successfully to extract the relevant features and achieve remarkable accuracy (Szegedy et al., 2015). Similarly, CNNs can be used to extract features for protein-ligand complexes that are represented by multi-dimensional tensors. CNNs are also able to capture the spatial placement of features as described by Goodfellow et al. (2016), which can be an important factor to extract interactions features from 3D structures such as the protein-ligand complexes. For these reasons, and the success achieved with CNNs in the works of Jiménez et al. (2018) and Stepniewska-Dziubinska et al. (2017) we have chosen the CNN as the main DL model to use for automatic features extraction from our multi-dimensional representation of the protein-ligand complexes.

ML methods and later on DL methods have been shown to improve the performance of SF used for structure-based drug design (Ain et al., 2015). In particular the work of Jiménez et al. (2018) and Stepniewska-Dziubinska et al. (2017) have recently developed a scoring function based on Convolutional Neural Networks (CNN) where they achieved a state of the art Pearson Correlation Coefficient, R, of 0.82 and 0.77 respectively between the actual and predicted binding affinities on the PDBbind v2016 (Liu et al., 2017b) dataset. The Pafnucy model developed by Stepniewska-Dziubinska et al. (2017) is used as a baseline model. The success achieved by Pafnucy and its source code availability are the main reasons for motivating us to choose Pafnucy for our baseline model. The availability of their source code made it possible to obtain a deep understanding in the workings of their models, so that a like with like comparison can be made with our developed methods. This would also allow us to perform a better evaluation. The baseline model is discussed in more detail in Section 3.

A major challenge faced by Jiménez et al. (2018) and Stepniewska-Dziubinska et al. (2017) was the requirement of training a CNN model that can handle different views or snapshots of the same representation. If the orientation from where the snapshot is taken is changed, a different representation of the *same* thing is obtained, since the protein-ligand complex is a 3D structure. The authors have worked around this limitation by introducing different systematic rotations of the same input during training. However, these might present additional challenges when testing novel complexes that can take different orientations. This limitation has led us to explore methods that are inherently rotationally invariant. The Ligity (Ebejer et al., 2019) method is based on

the spatial distance distribution between different set of pharmacophoric interactions points. Since these distances remain the same irrespective of the orientation of the structure, the Ligity method is inherently rotationally invariant. The Ligity method was shown to be successful for a virtual screening exercises using similarity measures. The facts that the method is rotationally invariant and that it has not been used for a scoring function application using DL have motivated us to explore the method so that we can apply it to a scoring function application.

1.3 | Aims and Objectives

This dissertation answers the following research question:

Can deep learning approaches be used in scoring functions to augment the predictive scoring power in SBVS techniques?

This research question will be mainly tackled from two aspects to improve the predictive scoring power.

1. Can the deep learning architecture be optimised to improve the SF performance?
2. Which input features should be used to find a suitable *representation* of the protein-ligand complex that can represent the binding interactions and provide good prediction performance? The choice of representation of the protein-ligand structure will determine the flexibility and expressiveness that the model is able to learn and ultimately its scoring power. Although DL methods extract features automatically during training, correct representation of the complex is still required and the chosen representation impacts the feature extraction ability of the DL model.

Therefore, the predictive scoring power effectiveness depends greatly on these two aspects. Improving the scoring power can therefore enhance the successful discovery of drugs, where only the top ranked molecules are experimentally tested saving both on time and cost. Wang and Zhang (2017) claim that the SF is the most critical component for docking, and that a high binding affinity between small molecule and reception is one of the major selecting criteria in drug discovery. To reach the aims described above, the following objectives are tackled:

- Replicate the work of *Pafnucy* (Stepniewska-Dziubinska et al., 2017) to create a baseline model for a scoring function for comparison and evaluation purposes.

- Apply representation engineering to develop different DL input features such as protein ligand complex representation using modified *Ligity* (Ebejer et al., 2019) features applied for binding affinity prediction. Details on the proposed features are highlighted in Section 1.4, and detailed in Section 3.
- Evaluate different CNN architectures to find the best binding affinity prediction model for use with the LigityScore feature representation.
- Hyper-parameter optimisation of the CNN model and the protein-ligand complex feature representation.
- Use the *Comparative Assessment of Scoring Functions*, CASF-2013 (Li et al., 2018) and CASF-2016 (Su et al., 2018), and other models found in literature such as K_{deep} (Jiménez et al., 2018) and *Pafnucy* (Stepniewska-Dziubinska et al., 2017) to rank the scoring power for the model developed. Details of the CASF benchmarks is discussed in Section 4.5.2.

1.4 | Proposed Solution

The first step taken in this dissertation was to replicate and confirm the results obtained by Pafnucy (Stepniewska-Dziubinska et al., 2017) in order to establish a baseline model for our study. The Pafnucy model uses 4D tensors in order to *store* the atom properties in the 4^{th} dimension where the first three dimensions represent the position of the atom on the grid. This representation is used to construct a 3D CNN model for binding affinity prediction.

The high level process described in Figure 1.3 is used in our approach to develop a ML model for a scoring function for VS. This process can be split up into two major tasks i) generate a protein-ligand representation, and ii) design and implementation of the DL model. The first task dealt with finding a suitable representation for the protein-ligand complex and our approach is based on the pharmacophoric features introduced in Section 1.1.2. The pharmacophores include quite general structural features focused at the binding site of the protein-ligand complex which include features such as hydrogen-bond acceptors, hydrogen-bond doners, and hydrophoic regions amongst the most popular features (Leach et al., 2010).

In our representation the 3D structure of experimentally validated ligand and the protein pockets complexes are considered. Protein *pockets* are used as they represents the area of the binding site of the protein. The open-source cheminformatics package *RDKit* (Landrum, 2020) is used to load the 3D molecule structures and search for these

interesting structural features that interact between the protein and the ligand. These interacting features are defined as Pharmacophoric Interaction Points (PIPs) and also are referred to as "hot-spots". Figure 1.4 shows such hot-spots for protein 3ZZF and ligand NLG focused at the binding site as the area of interest. The spatial distribution of the PIPs between combinations of hot-spots shown in Figure 1.4 are used as the basis for feature representation of the protein-ligand complex.

In our approach we have therefore hypothesised that these pharmacophoric interactions across different feature types contain key information to suitably represent the protein ligand structure and their binding properties. We have further hypothesised that this representation would be suitable to train a deep learning model for binding affinity prediction. To this effect we present two methods *LigityScore1D* and *LigityScore3D* that utilise pharmacophoric interactions to build a scoring function based on deep neural networks. This approach was inspired by the work done in *Ligity* (Ebejer et al., 2019) where pharmacophoric features were used to create a hybrid virtual screening method.

The second task dealt with finding the best CNN architecture to fit the new representation of the protein-ligand complex, and then optimise this network using hyperparameter tuning. The CNN approach was inspired by the methods of Stepniewska-Dziubinska et al. (2017) and Jiménez et al. (2018) being amongst top performing scoring function. The Pytorch open-source ML framework was used to build all CNN models.

LigityScore1D uses this pharmacophoric representation to extract a matrix of pharmacophoric distances to be used as input to the CNN model. The size of this matrix depends on the maximum distance threshold applied between pharmacophoric features, and the number of possible combination of different pharmacophoric features. For a maximum distance of 20Å that is discretised at 1Å resolution the size of the CNN input matrix has an input dimension of 21×21. Distance between each PIP combination is used to increment the count at a particular discrete value for a particular feature combination. Therefore each row within the matrix represents the counts of discretised distances for a particular pharmacophoric combination. In *LigityScore1D* 21 possible pharmacophoric feature combinations were possible, hence the 21×21 input dimenstion.

LigityScore3D uses a similar approach to the LigityScore1D method mentioned above however a combination of 3-PIPs are used. A combination of 3-PIPs provide three distances between PIPs, which are discretised and used as coordinates to increment the count of the corresponding voxel in a 3D feature matrix representation. The input size for LigityScore3D is also built using two parameters, namely the max distance allowed between PIPs, and the number of possible pharmocophoric feature combinations. LigityScore3D has a total of 56 possible feature combinations, and assuming also a maxi-

mum distance of 20 angstrom and a distance discretisation of one angstrom, the LigityScore3D input size is of $(56 \times 21) \times 21 \times 21$. This 3D feature matrix was reshaped to $98 \times 98 \times 54$ for a better representation for the input of the CNN model.

The scoring function is a regression type of problem and therefore the output of the model is a single continuous number representing the predicted binding affinity for the given protein-ligand complex representation. During training this is compared with the real values of binding affinity so that the weights of the model are updated using stochastic gradient descent using ADAM optimisation. To the best of our knowledge this is the first approach that utilises spatial distances between PIPs to build a protein-ligand representation to be used for scoring functions.

The CNN model used contains multiple convolutional layers with a combination of pooling layers, which are followed by multiple fully connected layers. Each convolutional layer utilises a set of several small filters or kernels, of size 5×5 for a 2D input, which are used to scan the input volume across its dimensions, using this small receptive field. These filters are randomly initialised and their parameters are updated during training. The subsampling or pooling layers are used for dimensionality reduction of the features, so that a latent representation of the input is provided at the end of the convolution layers which has a suitable dimension to be used as input to the Fully Connected (FC) layers. A number of experiments were done to find the CNN architecture that provides the best output results, and several techniques were used to augment the performance of the CNN have been used such as dropout, convolution dropout, and instance normalisation. These experiments are discussed in detail in Chapter 3. All CNN models were run on AWS EC2 cloud computing platform using G4 type instance.

1.4.1 | Evaluation

LigityScore is evaluated for the scoring power component of the CASF-2013 (Li et al., 2018) and CASF-2016 (Su et al., 2018) benchmarks. The benchmarks use the Core-2013 and Core-2016 test sets composed of high quality complexes from the PDBbind v2013 and v2016 datasets respectively. The rest of the PDBbind protein-ligand complexes are used for the training and evaluation sets. The CASF benchmarks were designed purposely as a benchmark for scoring functions as a means to evaluate the performance of the SF more precisely. The authors in Li et al. (2018) and Su et al. (2018) claim that the performance of a scoring function should be measured outside the context of molecular docking tests, as this approach is not the best evaluation method for SFs as they can be effected by other factors involved in docking. CASF-2016 (Su et al., 2018) is the latest contribution to the benchmark where the authors have made significant im-

provements over the CASF-2013 including a larger test set. The CASF benchmarks have been used in various studies including Stepniewska-Dziubinska et al. (2017), Wang and Zhang (2017), Zheng et al. (2019), and Boyles et al. (2020). These benchmarks are used for objective assessment and evaluation of LigityScore. Additionally, the LigityScore performance is also compared to other literature such as Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018) and Zheng et al. (2019) that are evaluated using the same benchmarks.

LigityScore performance is also evaluated with our own implementation of Pafnucy Stepniewska-Dziubinska et al. (2017) as the chosen benchmark model as it is one of the best performing CNN based scoring functions. The metrics that are used to assess the predictive performance of LigityScore include: i) Pearson correlation coefficient (R), ii) Standard deviation in linear regression (SD), iii) Mean Absolute Error (MAE), and iv) Root Mean Square Error (RMSE) measured from predicted and real binding affinity values. Over the recent years the PDBbind dataset has had considerable updates and contains thousands of experimentally validated Protein-Ligand complexes with bioactivity data. The availability of larger datasets containing compound activity and biomedical data have lead to the increase in use of data-hungry algorithms especially deep learning (DL) models for the development of new scoring functions.

1.4.2 | Contributions and Main Results

LigityScore1D and LigityScore3D show good results in binding affinity prediction with an average pearson correlation coefficient of 0.741 and 0.725 when tested on the Core-2016 set, and 0.635 and 0.713 respectively when tested on the Core-2013 set. Whilst the Core-2016 difference in results for the two methods is only 0.016, LigityScore3D has outperformed LigityScore1D by 0.078 for the Core-2013. Therefore, LigityScore3D has achieved better overall results as it showed similar performance for both CASF-2016 and CASF-2013 benchmarks.

These results were also evaluated on the latest two CASF benchmarks. The Pearson correlation coefficient, and the standard deviation in linear regression were used to compare LigityScore with the benchmark model, and also other models in literature published in recent years. The scoring functions from both the benchmark, and the literature reviewed were compiled in an updated list in order to provide a better understanding of LigityScore performance across a comprehensive list of SFs. With these above mentioned results LigityScore3D ranked 5th place in the CASF-2013 benchmark, and 8th in CASF-2016, where it managed to outperform the Pafnucy (Stepniewska-Dziubinska et al., 2017) method, as one of the best performing CNN based scoring function, in the

CASF-2013 benchmark.

Although our methods do not outperform current state of the art methods, we still managed to find a suitable protein-ligand representation based on pharmacophoric hotspots and this represents our main contribution in the CADD domain. Additionally, our method uses a rotational invariant approach since it is based on distances between hotspots that do not depend on the orientation of the complex, as opposed to the methods of Stepniewska-Dziubinska et al. (2017), and Jiménez et al. (2018) where the feature representation relies on the 3D orientation of the PL complex. Therefore our approach provides an alternative view for tackling this problem, which to our knowledge was not explored in other literature and thus provides a novel approach to the problem of binding affinity prediction in SBVS.

1.5 | Document Structure

The 'Background and Literature Overview' is discussed in Chapter 2 and presents comprehensive background information on SBVS and the CNN, as the chosen deep learning model to implement LigityScore. The related work section focuses on on scoring functions and provides a review of classical methods, ML learning models, and delves into more detail on DL approaches. The latest literature and developments on scoring functions is also presented and discussed, highlighted the current state of the art results. The influence of the presented literature on the development of LigityScore is also discussed.

The 'Methodology' details are presented in Chapter 3 which focuses on the work done to implement the proposed solution to achieve the listed aims and objectives. This chapter describes in detail the feature extraction techniques to generate the LigityScore protein-ligand representation, the datasets and cheminformatics libraries used, together with the design choices and supporting arguments for the CNN architecture used. The steps or *building blocks* for LigityScore are also described in detail.

The 'Results and Evaluation' outlined in Chapter 4, present the results obtained for binding affinity predictions for the various experiments performed, and their evaluation. The experiments include changes to both the protein-ligand complexes feature representation, and also to the CNN architecture components of the model. The experiments performed and their impact are discussed in detail to explain how the final solution was developed. The 'Evaluation' in Section 4.5 details the evaluation methods used for *Scoring Functions* and compares the LigityScore performance with our implementation of Pafnucy. LigityScore is also evaluated with the lastest Comparative Assessment of Scoring Functions benchmarks, namely the CASF-2013 (Li et al., 2018) and CASF-2016

(Su et al., 2018), and other recent literature that also use the same benchmarks.

Finally, Chapter 5 concludes with a summary of the research performed during this study and discusses the aims and objectives achieved. This chapter also highlights contributions to the field, presents a critique to our proposed solutions and highlights any future work that may be performed.

2

Background & Literature Overview

The following chapter provides background information on the scoring functions for SBVS and various aspects of the CNN that are used for the development of our scoring function model. Various aspects of virtual screening were mentioned in Chapter 1 to provide the reader with an initial understanding of the domain and problem being tackled from the onset. However, additional detail focusing on scoring functions are also discussed in Section 2.1, whilst Section 2.2 provides a detailed analysis on artificial neural networks (ANN) and the CNN. Section 2.3 describes the evaluation criteria that are used to assess the experiments performed detailing a description of the metrics and benchmark used. The last section focuses on the related work and discusses in brief the history of the scoring function (SF) in SBVS, and details the work done in the area using ML and DL techniques, providing the current state of the art research.

2.1 | Structure Based Virtual Screening (SBVS)

Virtual Screening (VS) is one of the CADD methods that are used to predict chemical activity of small molecules on target proteins *in silico*. VS is considered a critical early step in the drug design process for the discovery of novel molecular entities (Cheng et al., 2012). VS is practically the computational equivalent to biological screening, and is used in the same way to sift molecular libraries and identify potential molecules that will show biological response. Conventionally, HTS methods are used to synthesize and screen ligands against a target protein for *hit* identification. Such process is slow and expensive making it an unfeasible approach to screen libraries with millions of compounds. Thus, VS methods are a suitable alternative for cheap, rapid assessment of the molecular libraries. The biological response of molecules on target proteins is measured using the binding affinity half-concentration constant, IC_{50}. The binding affinity

represents the strength in association, and IC_{50} is a quantitative measure showing the concentration required to inhibit or activate 50% of the biological activity of the protein. A VS algorithm *ranks* the molecules under tests based on the predicted binding affinity to find potential active molecules also known as *hits*.

Assessment of the efficacy of virtual screening algorithms is normally applied using retrospective virtual screening, where the dataset consists of compounds that are pre-categorised into actives and decoys. VS algorithms that are successful in identifying known actives can then be used in prospective VS so that new uncategorised test compounds can be evaluated for any possible hits.

VS can be mainly split into three major categories as was described in Section 1.1. Structure Based Virtual Screening (SBVS) is the most popular method to estimate the binding affinity (Ashtawy and Mahapatra, 2012). SBVS uses structural information from both the protein's and ligand's 3D structure to deduce their physicochemical interactions. These interactions are approximated by evaluating a scoring function that considers the molecular interactions at the binding site. Ashtawy and Mahapatra (2012) comment that SBVS methods have seen an increase in interest due to the rise of available 3D structures, and because they provide a more accurate approach than ligand-based methods.

Molecular docking is an SBVS approach, where the docking algorithm generates several binding poses of the molecule under test around the area of the protein binding site. The *binding pose* represents a candidate orientation of the ligand relative to the target. The ligand orientation with reference to the protein structure is referred to as *binding mode*.

SBVS theoretical background stems from Fischer's analogy of the lock and key hypothesis developed in the 19th century. Fischer (1894) theory was that a molecule will bind to a protein if its shape matches the space available at the protein binding site, similar to a key matching a lock, as is illustrated in Chapter 1, Figure 1.1. The docking algorithm thus searches for a suitable binding pose in an exhaustive manner all the possible molecule binding modes in a typical 'lock and key' fashion, in search for the perfect fit. The scoring function of the docking program is used to find a suitable ligand pose yielding the best binding affinity. Molecular docking is also termed docking-based virtual screening (DBVS) and is arguably the most applied method in SBVS (Cheng et al., 2012).

The docking algorithm needs to tackle a number of challenges for successful simulation of ligand-target interaction such as i) the role of water molecules at the binding site, ii) protein and ligand flexibility (ability to change molecules' orientation at rotational bonds), iii) metal ions, and iv) the effectiveness of the scoring function used to

assess the fitness of the docked pose. Therefore, the scoring function is a key element in molecular docking and crucial to the docking algorithm as its reliability depends on the accuracy of the scoring function used.

2.1.1 | Scoring Functions

The limitation of docking models have been attributed to the poor performance of scoring functions. Improving this performance is not a trivial task and several developments have been done in the last 30 years, however results are still unsatisfactory (Ain et al., 2015). Zheng et al. (2019) further comment that inaccuracies in the binding affinity prediction leads to high false positive rates during pose prediction, and emphasize the urgent requirement to develop accurate scoring functions. Ragoza et al. (2017) describes the SF as the foundation in structure-based drug design, and Li et al. (2019) attribute four major functions of the SF in VS. These four functions are:

1. **Pose prediction:** Determine the binding mode of the ligand to the target protein. Each ligand pose is evaluated at the binding site, and the pose with the best binding affinity is selected. Therefore the scoring function is as described by Cheng et al. (2012) the "heart of molecular docking".

2. **Prediction:** Predict the absolute binding affinity between protein-ligand complex in lead optimisation.

3. **Ranking:** Ranking of molecules of known pose in order of the binding strength for a given protein target.

4. **Classification:** Identification of potential drug leads for a given target protein by evaluating a large molecular database to find if the molecules is active or inactive.

Section 2.4 provides an overview of the history of scoring functions starting with the *classical* methods and provides a comprehensive review on the development and the state of the art techniques on ML-based SF. The next section details the foundation background work in artificial neural networks and the convolution neural network as the basis for building our scoring function.

2.2 | Artificial Neural Networks

Artificial neural network are inspired by the function of the human brain and were loosely modeled to simulate the biological neuron (Rosenblatt, 1957). Figure 2.1 com-

pares the representation of the biological and artificial neurons. Neurons are fundamental units in the brain that use dendrites to receive and collect input signals in the cell body from other neurons. The cell body processes them, and depending on the input received it sends the signal out to neighbouring neurons, which in turn generates an input signal to these neighbouring neurons. This basic representation was used as a a source of inspiration, however ANNs do not attempt to mimic the brain's biological complexities (Goodfellow et al., 2016).

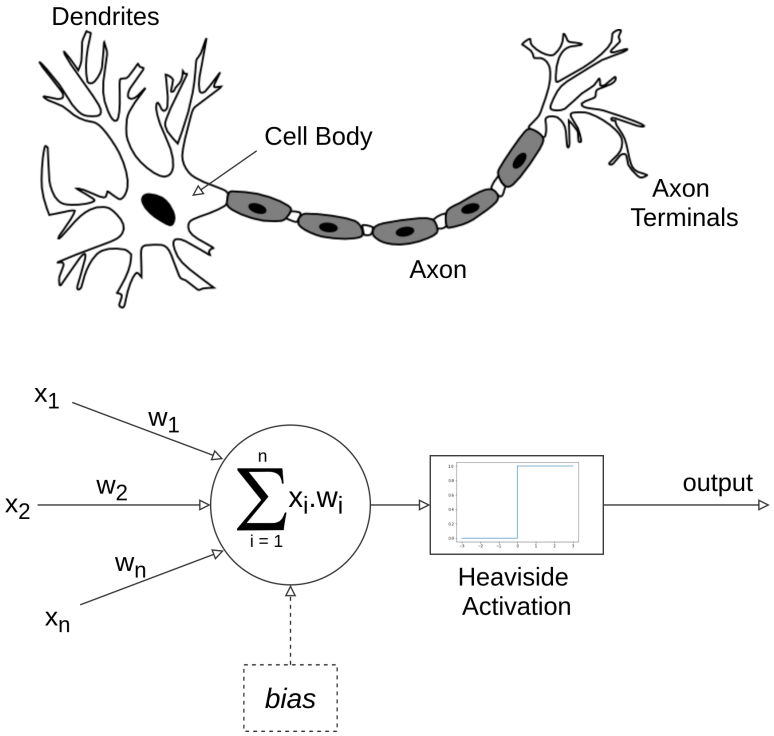


Figure 2.1: The structure of the artificial neuron is analogous to the biological Neuron.

Using the analogy of Figure 2.1 the artificial neuron receives input also from other neighbouring units, weighs their inputs simulating the connection of a synapse, and sum them up. Similar to the cell body function, an activation function determines the output of the weighted sum of inputs to decide if it should be fired (activated) or not, and propagates the signal to the other neurons. The perceptron was the first ANN developed by Rosenblatt (1957) that could learn from examples and solve a limited class of linearly separable problems. The range of solvable problems of the perceptron is limited as it is a binary classifier that uses a step function, known as Heaviside. Due to this major limitation of the perceptron, there was little motivation for further development. Later on, arrays of perceptron arranged in layers interconnected to each other showed

that they could solve more complex functions. This arrangement was defined as a *multi-layer perceptron* or a *feed-forward* (FF) neural network. A FF NN is composed of multiple layers that include an input layer, hidden layer, and output layer as shown in Figure 2.2. Each node within each layer is interconnected to all the other nodes in the previous, and next layers to form a network of layers. It is termed *feedforward* as the output of each layer is only connected to the next layer, and never to a previous layer using a *feedback* connection. Including a feedback connection would result in a Recurrent NN (RNN).

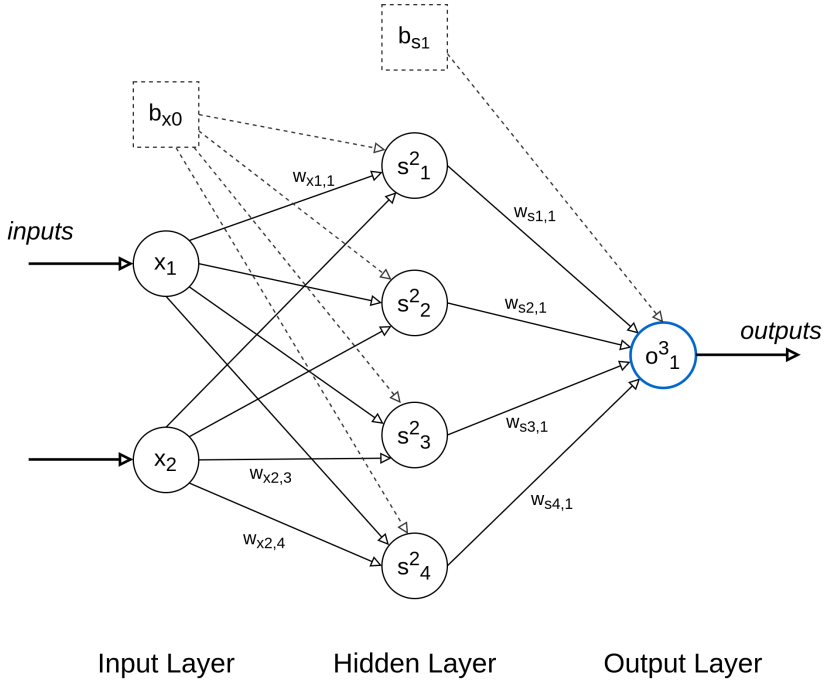


Figure 2.2: Feed Forward Neural Network showing input and output layers, and one hidden layer. w_i represents the weight for a particular for s_i neuron. Bias terms b_i are applied at each hidden and output layers.

A feed-forward neural network can be seen as the classic deep learning models when they have more than one hidden layer (Goodfellow et al., 2016). Deep learning concepts are explained in more detail in the next section. The aim of the feedforward network is to map some underlying function that approximates the required output for the given input. This mapping is achieved through *learning*. The FF NN is a supervised learning method that is capable of solving both classifications and regression tasks. In supervised learning, the training data consists of N samples and each sample is composed of a number of features $x_1, x_2..x_n$ that represent a predetermined output y. For classification tasks, y can be part of finite number of classes $y \in \{C_1, C_2, ..C_k\}$ with k classes. In the regression type y, represents a continuous value without any class label.

Each hidden layer requires the use of an activation function to compute its output. The rectified linear unit (ReLU) activation function, described in more detail in Section 2.2.2.1, is the default function to use as recommended by Goodfellow et al. (2016) in most FF NN. The ReLU function can be defined as $z(x) = max\{0, x\}$, where the output is linear for only positive values of x. This represents a non-linear function and therefore yields also a non-linear output for any linear input. Other activation functions include the sigmoid and the hyperbolic tangent (tanh) function. The non-linearity on the activation functions defined in the hidden layers allow the FF NN to learn non-linear functions. The FF NN with hidden layers was used to build the universal function approximation theorem (Hornik et al., 1990). Hornik et al. (1990) state that a FF NN with a linear output with at least one hidden enabled with ReLU activation (non-bounded such as sigmoid) can approximate any function given it has sufficient hidden units which can be determined empirically.

The strategy for FF NNs is to learn the underlying function through optimisation where the loss function of the NN is minimised. The loss function represents the difference between the real output versus the predicted output of the network, and is also referred to as *criterion*. The loss function is used to measure the performance of the supervised ML model by measuring the *distance* between the predicted output, $\hat{y}$ and the actual output y of training data, parameterised by the parameters $\boldsymbol{\theta}$ of the model. Learning is achieved by modifying the parameters of the model so that $\hat{y}$ is close to y as much as possible. Equation 2.1 shows the common *Mean Square Error* loss function:

$$L_{MSE}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \quad = \quad \frac{1}{n}\sum_{i=1}^{n}(h(\theta_i)) - y_i)^2 \tag{2.1}$$

where $\boldsymbol{\theta}$ are the model parameters, n is the number of samples, and $h(\boldsymbol{\theta})$ is the predicted output in terms of the model parameters, also known as the *Hypothesis Function*. Loss functions are minimised by finding the minimum in its derivative. For example consider the loss function $y = f(x)$, where both x and y are real numbers. The derivative of the loss function is $f'(x) = \frac{dy}{dx}$. If we consider a small change in the input ϵ, then the corresponding output can be represented as:

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{2.2}$$

Therefore for our minimisation objective we can use the derivative of the loss function to indicate the change in x to make an improvement in y. Since we need to minimize the function the negative gradient is used to specify the direction of x to make the im-

provement in y. Therefore:

$$f(x - \epsilon(f'(x))) \quad < \quad f(x) \qquad \textit{for small } \epsilon \tag{2.3}$$

This technique is called *gradient descent* and makes small changes to the opposite sign of the derivative to optimize the output (minimize). Learning of FF NN requires computing gradients of complicated functions using the *back-propagation* algorithm (Rumelhart et al., 1986).

The back-propagation is used in FF NN and several other deep learning models for training and is used to efficiently compute the complex gradients of these models. Back-propagation refers to the method used to compute the gradients, however other algorithm are used for the actual training such as stochastic gradient descent described later on in Section 2.2.4. The learning process utilising the back-propagation can be split into two parts with the objective of making model updates on weight terms, $\boldsymbol{W}$ and bias terms $\boldsymbol{b}$, so that the required output can be mapped for a given input. The bias terms represent a value that is added to each neuron for each layer, and provide a way to shift the activation functions to the left or right to make adjustments to fit the expected output.

- **Forward Propagation**. In the forward pass we predict the output value with the current parameter values. Considering the NN shown in Figure 2.2, the forward propagation steps include:
  - Initialize weights $\boldsymbol{W}_l$ and bias values $\boldsymbol{b}_l$ for each of the total l layers.
  - Get values for training input $\boldsymbol{x}$, and target output $\boldsymbol{y}$.
  - Starting at the input layer and moving forward to the output layer, one layer at a time, find the output from each layer k using:

$$\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)} \tag{2.4}$$

$$\boldsymbol{h}^{(k)} = z(\boldsymbol{a}^{(k)}) \tag{2.5}$$

    where $\boldsymbol{h}^{(k)}$ is a vector output at layer k, $z(\boldsymbol{a}^{(k)})$ is the activation function at layer k, and $\hat{\boldsymbol{y}} = \boldsymbol{h}^{l}$ is the output of the last layer.
  - Find the loss in the network, J using the loss function in Equation 2.1.

$$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{2.6}$$

- **Backward Propagation**. As detailed in the above Forward propagation steps to compute the output error, the particular weight values are transformed using different functions as it flows forward through the network. Partial derivatives using the *chain* rule for each function are used to find the change in loss at the output with respect to the particular weight and bias values. To illustrate this, let us consider the weight update of $w_{s1,1}$ of Figure 2.2. To find out how much change in $w_{s1,1}$ affect the total error, the partial derivative of the loss J with respect to $w_{s1,1}$ can be represented as:

$$\frac{\partial J}{\partial w_{s1,1}} = \frac{\partial J}{\partial h_1^3} \cdot \frac{\partial h_1^3}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial w_{s1,1}} \tag{2.7}$$

where, $\frac{\partial J}{\partial h_1^3}$ is the partial derivatives of the loss with respect to the output neuron o_1^3 (Figure 2.2). $\frac{\partial h_1^3}{\partial a_1^3}$ is the partial derivative for the activation function, and $\frac{\partial a_1^3}{\partial w_{s1,1}}$ is the partial derivative of the summation terms (Equation 2.4) with respect to weight $w_{s1,1}$. Additionally, since each layer has a vector of weights, let us represent the partial derivatives in vector notation as shown in Equation 2.8.

$$\frac{\partial J}{\partial W_i} = \nabla_{W_i} . J = \nabla_{W_i} . L(\hat{y} - y) \tag{2.8}$$

Therefore, using the above example, the model parameters can be updated using:

  - Compute the partial derivatives of the cost function with respect to the output, and store in g

$$g \longleftarrow \nabla_{\hat{y}} . J = \nabla_{\hat{y}} . L(\hat{y} - y)$$

  - **For each** layer l, compute gradients of the output with respect to the activation function, starting from the output layer going backwards to the input, and update g, using element-wise multiplication of g and $(z'(a^{(k)})$.

$$g \longleftarrow \nabla_{a^{(k)}} . J = g \odot z'(a^{(k)})$$

  - Compute partial derivatives for bias and weight terms.

$$\nabla_{b^{(k)}} . J = g$$

$$\nabla_{W^{(k)}} . J = g . h^{(k-1)^\top}$$

since bias is constant, and the weights are multiplied by the outputs of the previous layer with linear sum terms.

- Propagate the gradients with respect to the next layer (previous hidden layer).

$$g \longleftarrow \nabla_{\boldsymbol{h}^{(k-1)}} . J = g . \boldsymbol{W}^{(k)^{\top}} \tag{2.9}$$

- **Repeat** for the number of layers.
- Update all parameters using learning rate ϵ at each layer, similar to the gradient descent update shown in Equation 2.3.

The next section discusses Deep Learning in more detail and described the convolution neural network as a specialised form of deep learning network.

2.2.1 | Deep Learning

Machine Learning (ML) algorithm success is highly dependent on having the correct set of features that represent the problem being tackled. In representation learning, ML algorithms use the raw input to automatically discover a representation to output, and also the representation (or features) of the input itself. Deep Learning (DL) has the ability to enhance this representation learning so that it is able to extract complex features from raw data, that would otherwise be difficult to extract (Goodfellow et al., 2016). Without a good representation for complex features, learning would not be possible — a capability that is lacking in conventional ML algorithms. LeCun et al. (2015) comments that traditional ML techniques are limited to process raw data and that careful feature engineering and expert knowledge in the particular domain are still required for feature extraction. The principal aim of deep learning as described by Glorot and Bengio (2010) is:

> "Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features".

This simply implies that the DL learning models seem to abstract the input representation into simple concepts, and then extract a complex representation by building a hierarchy from these simpler concepts. If we consider an image as an example, it is difficult to create a representation from the raw pixels. However, in a deep learning model each hidden layer within the network extracts different types of simpler concepts such as edges, corners and contours, or small objects. When these simpler concepts, across

all layers, are nested together as a hierarchy they can form the complex representation that describe the object in the image. The key aspect in this representation is that no human input is provided for any feature engineering. The deep network is able to *automatically* extract multiple levels of representation and learn very complex functions. (LeCun et al., 2015).

This has empowered different researcher to use DL to discover features in high dimensional data in various domains including techniques at predicting activity of potential drug molecules (Ma et al., 2015). Goodfellow et al. (2016) describe model depth by measurement of the number of sequential instructions executed to produce the output (number of hidden layers), or else by the depth of the *graph* of how simple concepts relate to each other, however there are no defined rules for this measurement. In the next section we will describe in the convolution neural network DL model.

2.2.2 | CNN

The Convolution Neural Network (LeCun et al., 1989) is a specialised type of neural network that typically processes data with a grid-like input type. As the name of this technique implies, the convolution operation is used in this type of network. The *convolution* is a mathematical operation on two functions to obtain a new function that reflects how one of the input function is modified by the other input function. The convolution $s(t)$ for input functions $f(t)$ and $g(t)$ can be represented as shown in Equation 2.10.

$$s(t) = (f * g)(t) = \int_{a=-\infty}^{\infty} f(a)g(t-a)da \tag{2.10}$$

Goodfellow et al. (2016) define CNN as:

> "Convolution networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."

Equation 2.10 represents data in continuous space. The discrete function of Equation 2.10 applied to discretised time can be represented as:

$$s(t) = (f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t-a) \tag{2.11}$$

In a CNN terminology the $f(t)$ terms represent the input which is usually a multidimensional array of data, commonly referred to as a *tensor*. The $g(t)$ term is the kernel used to transform the input data and is a tensor of parameters that are updated during training. The output $s(t)$ can be referred to as the *feature map*. In multi-dimensional

data applications, the kernel has the same depth, or channels as the input image. The convolution operation when applied to a 2D tensor for an image I with a 2D kernel K is changed to:

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.12}$$

Since the convolution operation is commutitative it can be re-written as shown in Equation 2.13 and requires flipping the kernel tensor in both dimensions (bottom to top, and right to left).

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.13}$$

In Equation 2.13 i and j represent the dimension size of I, whilst m and n represent the dimension size of K. The summation terms are restricted to m and n, to represent the size of the kernel, and the other locations in space are assumed to be 0 so that the limits $-\infty$ to ∞ of Equation 2.11 can be dropped.

As described by Goodfellow et al. (2016) the commutative property in convolution is not important for neural networks and it is common for machine learning libraries to leave it out of the convolution operation. When the kernel is not flipped, in reality the *cross-correlation* operation is performed. Machine libraries such as PyTorch[1] implement cross-correlation and often use the two terms interchangably. In this study we also use the same convention. The kernel values are parameters learned within the network and in general will not make a difference if a flipped version is learned. When the kernel is *not* flipped the Equation 2.13 changes to:

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.14}$$

The convolution operation (without flipping) is shown in Figure 2.3. A window the size of the kernel of the input (examples marked in shades of red) is convolved with the filter to produce a single value. The output dimensions shown in Figure 2.3 can be calculated using Equation 2.15.

$$n^{[l]} = \frac{n^{[l-1]} + 2p^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \tag{2.15}$$

where, $n^{[l]}$ is the output at layer l, n^{l-1} represents the input size of a particular dimension, and $p^{[l-1]}$ is the padding applied to this input that are coming from a previous

[1] `https://pytorch.org/docs/master/generated/torch.nn.Conv2d.html` (last accessed 20-06-2020)

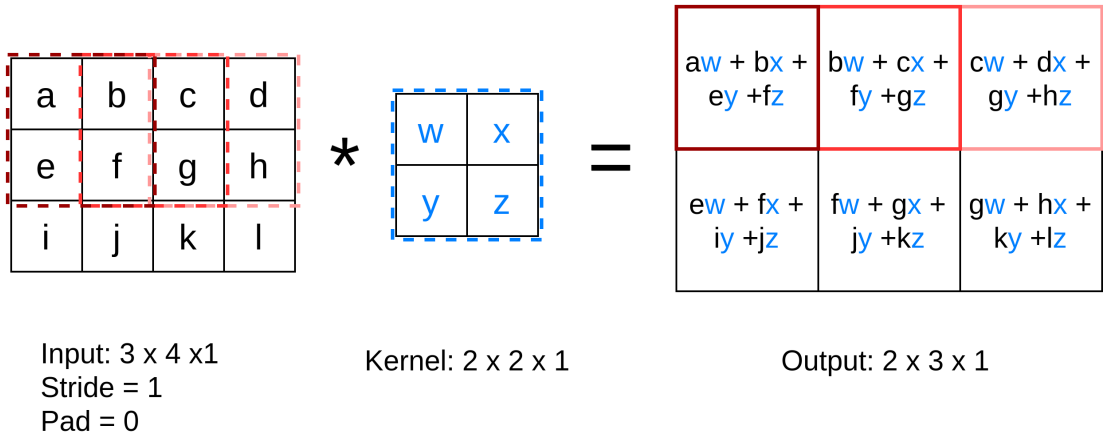


Figure 2.3: 2D Convolution without kernel flipping. The kernel is here of size 2×2 and slided across the input with a stride of 1. In this example the input is not padded and since the kernel needs to lie entirely within the image, the resultant output is of 3×2. Adapted from Goodfellow et al. (2016).

layer, f is the kernel size for the corresponding dimension, and s is the stride used for the convolution. Padding enlarges the input size by adding zeros across all dimensions. This can be used to keep the output of the same size. Padding is also used to retain more information at the border of the input. The stride is the number of steps the kernel windows moves to calculate the next output value.

For input with more than 2 dimensions such as a 3D images, the kernel is required to be of the same dimension, however the output will be always 2D, irrespective of the number of channels with the input. In CNN networks multiple filters are used to increase the channel depth of the convolution output. Figure 2.4 shows an example where the input is convolved with 64 filters each of size $5 \times 5 \times 3$. Convolution of the 64 filters with the 3D input of $32 \times 32 \times 3$ would correspond to an output size of $28 \times 28 \times 64$ using Equation 2.15 assuming a stride of 1 and no padding. The output from all the filters are collected into one multi-dimension tensor to be used as input for the next layer.

2.2.2.1 | Anatomy of the CNN

Figure 2.5 illustrates a typical CNN network made up of following components, which will be discussed further in this section.

- The first part of the network is made up of a series of convolution and pooling layers.

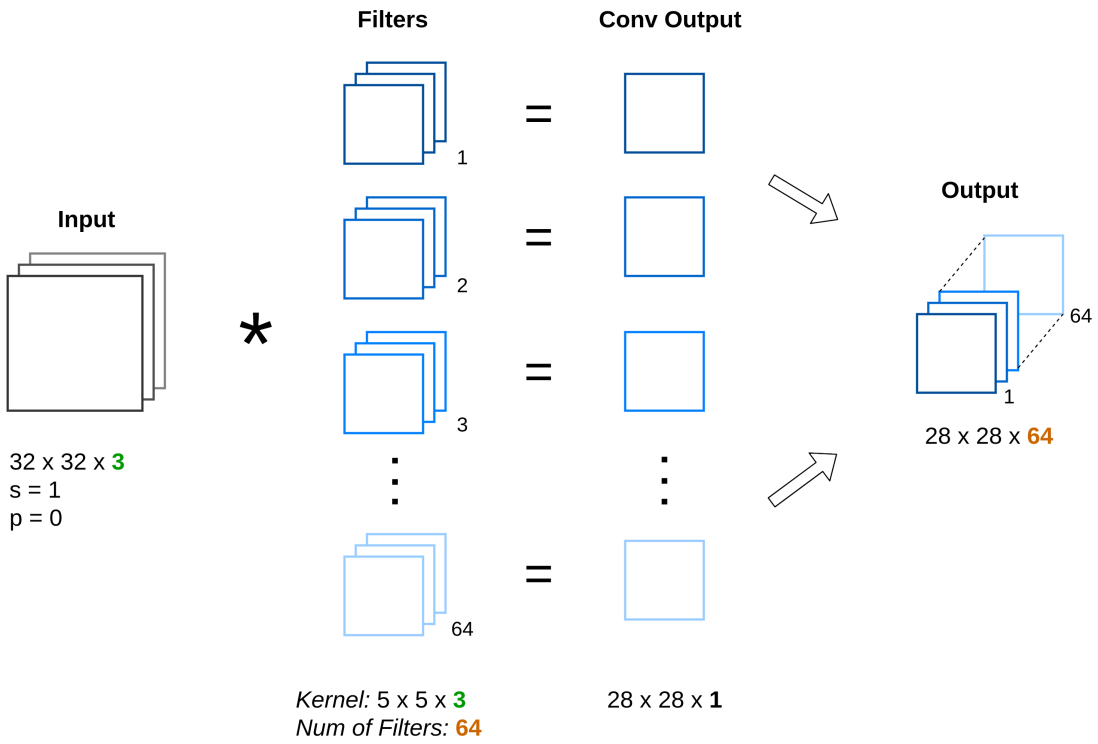


Figure 2.4: Convolution with multiple filters. A 3D input is convolved with an array of 64 filters. Each filter needs to have the same number of channels as the input. In this example the input has 3 channels, so the size of the filter with a kernel of 5 is $5 \times 5 \times 3$. Each filter produces a 2D image, which are grouped in a single array to produce the output of the layer.

- The second part of the network consists of fully connected (FC) components using a number of hidden layers.
- At the output, there is additional softmax layer for classification problems, or a single output neuron for regression type of problems.

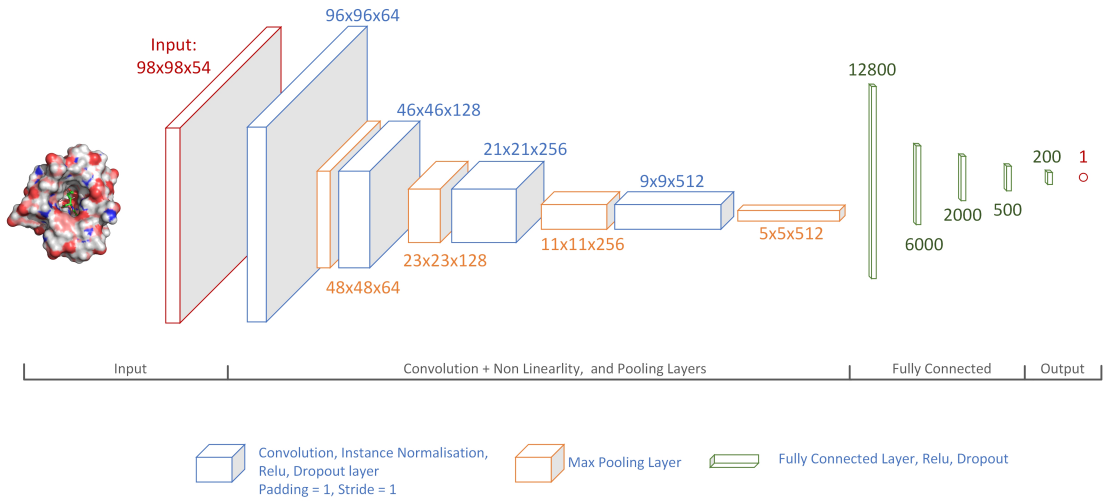


Figure 2.5: CNN Architecture used for LigityScore3D (described in Section 3.7) showing input layer, convolution and pooling layers, fully connected section and a single output neuron for a regression output.

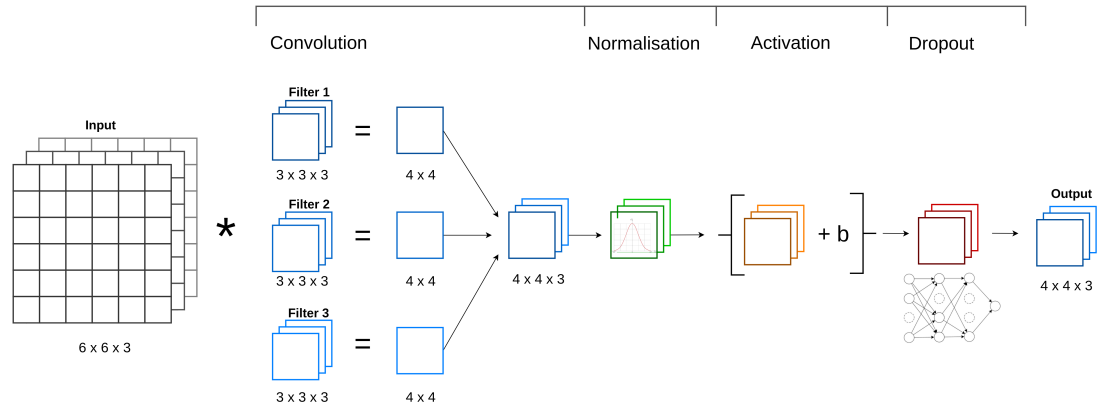


Figure 2.6: A Convolution Layer. Apart from the convolution operation with a number of filters, a convolution layer includes also a non-linear activation function, such as RELU, and a bias component. The normalisation (Section 2.2.6) and dropout components are optional components.

The Convolution Layer

The convolution layer is composed of several components as illustrated in Figure 2.6, and represents a zoomed in view for one of the convolution layers (blue block) in Figure 2.5. The first stage includes a series of parallel convolution operations to compute a set of linear activations. The second stage is normally a non-linear activation function such as the Rectified Linear units (ReLU) function. In our example we have included also the normalisation and dropout components. These are optional components that are not part of the basic convolution layer but are included here as they are mentioned in subsequent sections and were used as additional building block in this study. Although the normalisation can be placed after activation (Mishkin et al., 2017), it used before activation by Ioffe and Szegedy (2015) when used with the ReLU function.

The RELU non-linear activation function is visualised in Figure 2.7 and the function is represented by Equation 2.16. RELU returns 0 for any negative value, whilst it returns the input itself for any positive input.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.16}$$

$$\frac{d}{d(x)} f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2.17}$$

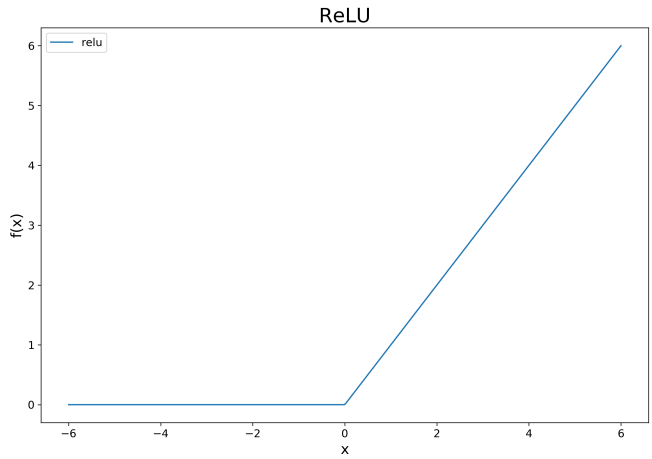


Figure 2.7: The RELU function. The RELU function is non-linear activation function commonly used in deep networks that although its simplicity is was used successfully in various deep learning methods to achieve state of the art results in various years (He et al. (2015); Krizhevsky et al. (2012)).

The RELU activation function (or *detector*) is a commonly used in NN due to its simplicity, performance, and its suitability in deep networks, and was also used for the LigityScore implementation. Krizhevsky et al. (2012) introduced a breakthrough CNN architecture termed AlexNet, which pioneered the adoption of ReLU in deep networks together with dropout layers, and achieved state of the art results for the 2012 ImageNet (Deng et al., 2009) classification challenge. Krizhevsky et al. (2012) reject the use of the *standard* way to model the neuron output that use the *sigmoid* or *hyperbolic tangent* (tanh) activation functions. The authors describe how ReLU enabled their network to train several times faster than the tanh activation counterpart, and allowed the authors to successfully train their deep network on the massive ImageNet dataset which wasn't feasible with saturating neuron models (sigmoid, and tanh).

He et al. (2015) used the ReLU activation to again achieve state of the art results in 2015 on the ImageNet classifications using very deep CNN where their result for first time achieved better than human classification performance. He et al. (2015) continued on the work of Krizhevsky et al. (2012), and to achieve these results they proposed an extension of ReLU termed *Parametric Rectified Linear Unit* (PReLU). For the positive part, the PReLU is the same as ReLU. For the negative part He et al. (2015) introduce the a_i term that is an activation learnable parameter. The subscript i is used to refer to the i^{th} channels as PReLU allows different values to be learned across different channels. He et al. (2015) also devise a sound initialisation method for the CNN weights that is based on ReLU activations, that helps very deep networks to converge. Their method is termed the *Kaiming Initilisation* and is discussed in Section 2.2.5.

Pooling

A pooling layer is used after each *convolution layer* to reduce the dimensionality of its output feature maps by using a summary statistic to summarise the presence of features. The summary statistic for the pooling operation is a commutative function such as *sum*, *average*, and *max*.

The feature map output of the convolution layers encodes also the location of the particular features. Any translation of these features at the input, would create another different feature map. In order to avoid different representations, nearby features of the convolution layer output are grouped together and summarised in *patches* to be downsampled as to create a convolution representation that is invariant to small translations of the input. Therefore the pooled output remains the same even when there are small fluctuations at the input (Goodfellow et al., 2016). This type of invariance is termed *local translation invariance*. Local translation invariance is a useful property to have when we are interested in the presence of the feature rather than its exact location.

The pooling is applied to each feature map seperately so that all channels are pooled

in the same way. A pooling with a kernel size of 2 and a stride of 2, would apply the summary statistic at a 2×2 window, that would half the dimensions of the feature maps. Two of the most common summary statistic include:

- **Max Pooling**. Max pooling summaries the feature maps by choosing the maximum value in the pooling window. Therefore, only the features with the most activated presence are selected, which tends to provide the most salient information from features.
- **Averaging**. Average pooling calculates the average values of the considered in the pooling window to smooth out the features in the pooling window.

Fully connected layers

The output of the last convolution layer is flattened to one dimension so that it is used as input to a fully connected (FC) network. The CNN layers analyse the input and break it down into a map of features. In turn, the fully connected layers, which are deep FF NN as introduced earlier in Section 2.2, are used to map these extracted features to the expected output. The convolution layers on their own are not capable of concluding the output result and therefore the FC layers are crucial for the operation of the CNN.

On the other hand, a fully connected layer on its own can be inefficient especially for large multi-dimension tensors. Moreover the FC layers lack the feature extraction capabilities of the CNN layer which are essential to the overall learning process. These capabilities are discussed in Section 2.2.2.2. Therefore the two parts together, enable the CNN to be one of the most successful NN architectures (Goodfellow et al., 2016).

Output Layer

The output layer of the CNN is the last layer of the FC network and similar to the FF NN, it is expected to provide the final transformation to predict the output. The output layer can have different activation functions, such as the linear or sigmoid functions and the choice depends on the required output distribution. In classification tasks the probability distribution is discrete and contains n possible values. The *softmax* function can be used in these classification scenarios to represent the probability distribution over the n classes, and draw out the probability for each respective class. In classification tasks the output layer contains as many neurons as the number of classes in the dataset. The softmax function is shown in Equation 2.18.

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)}, \quad \text{where} \quad \boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b} \tag{2.18}$$

where z is the summation terms at the output neuron. Softmax exponentiates z for better gradient-based optimisation, and normalises over all other outputs in order to have a valid probability output between 0 and 1 for each output neuron, with a total sum probability equal to 1. Softmax generalises the sigmoid function to extend the binary value probability distribution to multiple classes.

Regression type of problems have a single output neuron that does not make use of squashing activating functions such as sigmoid or tanh, as these would restrict the output values to their limits. Therefore for regression output linear outputs can be used, or else ReLU for positive only values.

The work of Krizhevsky et al. (2012) and his success in the ImageNet challenge on object recognition has sparked and motivated researchers to use CNNs in various domains also outside of the computer vision domain. Although the basic building blocks remain the same, various successful architectures were developed in recent years. These prominent architectures include LeNet LeCun et al. (1998), the VGG (Simonyan and Zisserman, 2014), and the ResNet (He et al., 2016). The CNN methods used for building scoring functions is discussed in detail in Section 2.4.4.

2.2.2.2 | CNN as a better ML model

The CNN network leverages a number of concepts that help to improve a ML model which stand out from traditional fully connected network. These areas can be used as motivation to train better networks using convolution layers. These include:

- **Sparse Interactions**. In traditional NN every output neuron is connected to all inputs. Convolution networks as shown in Figure 2.4 have the windowing filter that is smaller by a number of orders of magnitude. This implies that only the neurons within reach of the windowing filter are connected. This leads to sparse connectivity within convolution networks. The CNN network can thus detect small, meaningful features within the filter window such as edges using fewer connections and learning parameters, which reduce the memory requirements and also improves statistical efficiency. This results in significant efficiency improvement (Goodfellow et al., 2016). Therefore the sparse connections are able to detect important features with less network complexity. A neuron in a deeper layer of the network has still an indirect interaction with a larger portion of the input as sparse connection form a sort of hierarchical tree, where the deep neuron is connected to a number of adjacent neurons, however these neurons are in turn connected to a number of other neurons from the previous layer as illustrated in Figure 2.8. This

concept allows separate feature maps to interact together when constructed as a deep model.

- **Parameter Sharing**. The kernel filters in a CNN are all learnable parameters. In convolution the filter is shifted across all the inputs such that all its parameters are *shared* across all the input elements. This contrasts with traditional NN, where specific weights (parameters) are tied to individual neurons. These *tied* parameters are only used once when computing the output. Therefore in CNN, a set of shared parameters are learned for the whole input, rather than having different distinct parameter sets for every location. The shared aspect implies that less parameters are used in training that also result in significant memory efficiency enhancement.

- **Equivariance**. Convolution function is equivariant, meaning that if the input changes, the output changes in the same way and can be expressed as $f(g(x)) = g(f(x))$. This property is important for feature extraction. This is the result of the parameter sharing principle described in the previous section. In time series data, the same *event* but at a different time would still have the same representation. Similarly, in 2D data, spatial feature maps are created that localise particular features of the input. If these features are moved, the spatial feature maps move by the same amount. This would basically imply that the feature extraction process is independent of the spatial position of the features. However this also means that this property is useful to identity the presence of the feature but not their position. It is important to note that scaling and rotation transformation are not equivariant.

2.2.3 | Dropout

Dropout regularisation technique that is targeted for neural network (NN) models and was proposed by Srivastava (2013). The dropout techniques adds bias or noise to the NN model in order to prevent overfitting. In order to add this bias, the dropout technique removes nodes together with their input and output connections randomly during training. The random dropout action is performed during the training phase only, and it creates a different thinned NN for each epoch as there are less neuron activations. Minimisation, through back propagation, of the loss function is only applied to the thinned network, as the inactive neurons do not participate in the training of that epoch which may lead to slower training. The intensity of the dropout is regulated by hyperparameter p, describing the probability of retaining a unit, where $p = 1$ implies no dropout. Dropout can be applied both for the input layer, and also on each of the

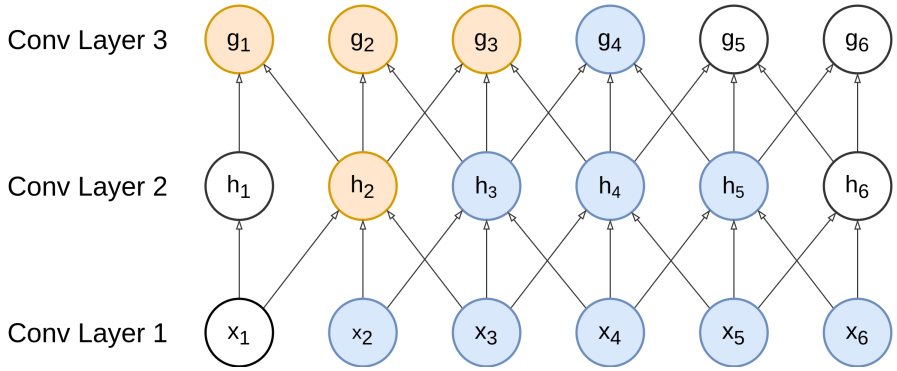


Figure 2.8: Neuron direct and indirect Connections. The diagram represents 3 convolution layers as seen from a *plan* (top) view. Each neuron has sparse connections to the other neurons as represented by neuron h_2 highlighted in orange. h_2 has only direct connections with g_1, g_2, and g_3. However, neurons in a deep network such as g_4 still have an indirect connection to most or all of the neurons of the input through the previous layers. The highlighted neuron in blue shows the sparse connection in a hierarchical tree creating indirect connections from across a number a layers. Adapted from Goodfellow et al. (2016).

hidden layers. Srivastava (2013) have determined that the typical values of p for hidden layers is between $0.5 - 0.8$, whilst for real valued inputs a probability of 0.8 is used. Choosing an incorrect value of p can induce too much bias and may lead to underfitting. In our LigityScore models we use 0.5 dropout probability for all FC layers. Higher probabilities were also used in some experiments but did not show any generalistion improvement and had the negative effect of slowing the convergence time.

During each training epoch a masking vector is created using the probability p applied on a Bernoulli Distribution, where its output is either '1' or '0' to determine which neurons are activated. Therefore, considering a layer of n neurons, $n\,(1-p)$ neurons would be masked at each epoch. At the end of the training each node would have been trained a different number of times, however each epoch still contributes to the same sets of weights. On the other hand, during the testing phases dropout is not switched off and all the neurons are used. Srivastava (2013) has determined that dropout was successful in various domains including speech recognition and document classification to mention a few. The author has concluded that the dropout method is a general technique that can be applied across different domains, however it has the drawback of extending the training time by typically two to three times.

2.2.3.1 | Convolutional Dropout (Spatial Dropout)

Spatial Dropout is used to drop the activations of the entire feature map on a particular channel to improve the generalization performance of a CNN network. Spatial dropout was formulated by Tompson et al. (2015) to reduce overfitting by preventing particular activations in their network from becoming strongly correlated. This contrasts with the dropout method discussed in the previous section where the activations are dropped randomly. Tompson et al. (2015) claim that standard dropout at the convolution layers increases the training time but fails to prevent overfitting.

The feature map is constructed using sparse interactions as described in Section 2.2.2.1. Since the output depends on multiple neurons, simply removing one of the neurons using standard dropout, will still receive a strong activation since most probably neighbouring neurons would also contribute to strong activation. Dropout in this scenario is only effective to decrease the learning rate of the model. Therefore spatial dropout drops out all the feature map at a particular channel to reduce activation strength. Tompson et al. (2015) show that they outperformed the previous state-of-the-art results on human-body part localisation CNN with the addition of spatial dropout. Spatial dropout with a small dropout probability (0.1 or 0.2) was applied to the LigityScore convolution layers and showed improved results and more resilience to overfitting.

2.2.4 | Stochastic Gradient Descent

Machine Learning algorithms utilize optimisation algorithms for learning, based on the gradient descent and back propogation algorithms introduced earlier, where the parameters updated at each iteration do not consider all the training examples. Instead, a sample of the training set is used to find an estimate of the error at each iteration to update the parameters. Such optimisation algorithms are commonly called *minibatch stochastic* or simply *stochastic*.

Stochastic gradient descent (SGD) is one of these *stochastic* optimisers and it was proved to be an effective way to train deep networks (Ioffe and Szegedy, 2015). CNN networks are also typically trained in mini-batches. The CNN is only updated once after the loss for the mini-batch samples are accumulated. This reduces the training time as the network is only updated once per mini-batch. Additionally since each mini-batch is of fixed size, its computation time remains the same allowing steady gradient updates for large datasets. The number of samples to includes in each mini-batch varies per model and can be as low as 5 or large as 256 (Ioffe and Szegedy, 2015). The basic algorithm for SGD is as follows:

1. Initialise learning rate ϵ_k, and parameters $\boldsymbol{\theta}$. The value of ϵ_k is decreased gradually over time, and therefore ϵ_k denotes the learning rate at iteration k.

2. Randomly sample a mini-batch of m samples $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(m)}\}$ from the training set with their output $\boldsymbol{y}^{(i)}$.

3. Compute gradient estimate, $\hat{\boldsymbol{g}}$ using back propagation for each sample, and find their average. Adding the average term and generalising Equation 2.8, the gradient estimate can be represented as:

$$\hat{\boldsymbol{g}} \quad \longleftarrow \quad \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - \boldsymbol{y}^{(i)}) \tag{2.19}$$

   The gradient estimate from the random sample creates an element of noise in the model, since it considers only a sample of the training set, and can be seen as a form of regularisation.

4. Apply updates to parameters:

$$\boldsymbol{\theta} \quad \longleftarrow \quad \boldsymbol{\theta} - \epsilon\, \hat{\boldsymbol{g}} \tag{2.20}$$

5. Repeat until stopping criterion for error is reached.

Momentum is a technique added to the SGD algorithm to cater for its slow learning speed. Momentum introduces another hyperparameter $\alpha \in [0, 1)$ that controls the momentum variable v, termed velocity which represents an exponentially decaying average of the negative gradient. The exponential effect results from accumulated term of v over the iterations. Velocity v is controlled by α so that the gradient estimate of Equation 2.20 changes to $\alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$. The parameters are then updated by this velocity component. Therefore Equation 2.21 is applied before *Step 4* from the SGD algorithm above.

$$\boldsymbol{v} \quad \longleftarrow \quad \alpha \boldsymbol{v} - \epsilon\, \hat{\boldsymbol{g}} \tag{2.21}$$

The larger the value of α, the bigger the impact of the previous gradient on the current gradient.

2.2.4.1 | Adam Optimisation

Goodfellow et al. (2016) describe the difficulty in selecting a good ϵ. A number of algorithms using adaptive learning rates have been developed to alleviate this difficulty. Adaptive optimizers utilise separate learning rates for different parameter which are also updated automatically during learning. The *Adam* is a popular adaptive optimizer developed by Kingma and Ba (2014) that was inspired by other popular adaptive optimizers namely AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman and Hinton, 2012). Adam uses techniques from RMSProp and SGD with momentum. Adam uses squared gradient to change the learning rate like RMSProp as in Equation 2.23, and uses the moving average update shown in Equation 2.21 instead of the gradient itself (Equation 2.20) like momentum SGD.

Adam uses estimates of 1^{st} and 2^{nd} order moments of the gradient taken from the mini-batch to apply adaptive learning rates. The n^{th} moments of the gradient of the cost function, which is a random variable, is the *expected* values to the power of n. The term Adam stems from the *adaptive moment* estimation. Adam introduces a number of parameters. These include β_1 and β_2 used as decay rate for moment estimates, and δ that is used for numerical stability. The Adam algorithm is summarised by the following sequence:

1. Initialze parameters, β_1, β_2, ϵ_k, and δ. The suggested parameters intial values defined by Kingma and Ba (2014) are 0.9, 0.999, 0.001, and 10^{-8} respectively. Apart from ϵ, the others are rarely changed.

2. Initialise time step $t = 0$.

3. Initilise model parameters $\boldsymbol{\theta}$, and $\boldsymbol{s}$, $\boldsymbol{r}$ which are 1^{st} and 2^{nd} moment variable respectively.

4. Sample mini-batch and compute gradient, $\boldsymbol{g}$ as in Steps 2 and 3 of the above mentioned SGD algorithm.

5. Increment t ($t \leftarrow t + 1$).

6. Update 1^{st} and 2^{nd} moment variables:

$$\boldsymbol{s} \quad \longleftarrow \quad \beta_1 \boldsymbol{s_{t-1}} + (1 - \beta_1)\, \boldsymbol{g} \tag{2.22}$$

$$\boldsymbol{r} \quad \longleftarrow \quad \beta_2 \boldsymbol{r_{t-1}} + (1 - \beta_2)\, \boldsymbol{g} \odot \boldsymbol{g} \tag{2.23}$$

These represent exponentially moving averages since as the moments are accumulated the β terms increase in power. The first moment can be seen as the mean of the gradient and a momentum term, whilst the 2nd moment represents the uncentered variance (mean not subtracted).

7. Correct bias for the 1st and 2nd order moment variables so that these estimations are or expected value.

$$\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1 - \beta_1^t} \tag{2.24}$$

$$\hat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1 - \beta_2^t} \tag{2.25}$$

8. Find change in update for parameters and apply updates:

$$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{r}} + \delta}; \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \tag{2.26}$$

9. Repeat Steps 4-9 until the required stopping criterion is reached

Kingma and Ba (2014) conclude that Adam is a *simple* and computationally efficient stochastic optimisation technique which combines advantages from AdaGrad and RMSProp, and is aimed to for use in deep learning methods to enhance the model's convergence. LigityScore uses the Adam optimiser techniques using default values for all parameters except for the learning rate, where smaller values were used (1×10^{-5}).

2.2.5 | Weight Initialisation

Weight initialisation is an important element to consider when training ANNs. Incorrectly initialised weights can lead to poor performance or cause the network not to converge. Weight initialisation that is too large can lead to exploding gradient issue, whilst initialisation that are too small can lead to vanishing gradient issue. Exploding gradients occurs when *large* weights in the network are multiplied across several layers that cause a large change in the error output, which in turn also causes large gradients. The opposite occurs for small weights. Weight initialisation methods such as zero intialisation or random initialisation are rather naive and are not suitable for deep networks. In this section we will discuss the Xavier (Glorot and Bengio, 2010) and the Kaiming (He et al., 2015) methods.

The Xavier initialisation method was developed by Glorot and Bengio (2010). The authors investigate how different activation functions and weight initialisation technique effect model performance, and come up with two rules for weight initialisation to avoid vanishing or exploding gradients.

- Keep the mean of the activations to zero.
- The variances of activation across all the layers should be the same.

To abide but the above two rules, Glorot and Bengio (2010) proved that the weight initialization when using tanh activation can be initialised using Equation 2.27:

$$Xavier_{Init} : \begin{cases} W^{[l]} = \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}) \\ b^{[l]} = 0 \end{cases} \tag{2.27}$$

where $W^{[l]}$ are the weights at layer l, and $\mathcal{N}$ represents the Normal Distribution with mean $\mu = 0$, and variance $\sigma^2 = \frac{1}{n^{[l-1]}}$ and $n^{[l-1]}$ is the number of neurons in the previous network layers. The Xavier method initiliases the bias with zeros.

Although the Xavier method applies for tanh function, Equation 2.27 were derived using the assumption that activation are linear (which is valid for values close to 0). Therefore the Xavier method is only valid for the tanh function. ReLU functions cannot assume the linearity as all negative values have 0 activation and a different technique is required for ReLU activations. He et al. (2015) continue on the work of Glorot and Bengio (2010) and extend their concepts to find suitable initialisation for deep networks with ReLU activation. The Kaiming method initialises the weight like Equation 2.27 but with the variance multiplied by two as shown in Equation 2.28.

$$Kaiming_{Init} : \begin{cases} W^{[l]} = \mathcal{N}(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}) \\ b^{[l]} = 0 \end{cases} \tag{2.28}$$

In their work He et al. (2015) claim that their initialisation method is more robust and is suitable to training also very deep networks (example 30 layers), which cannot be achieved using Xavier initialisation. However, for model with less layers He et al. (2015) report no significant difference in accuracy of their tests.

2.2.6 | Batch Normalisation

Input normalisation is a standard pre-proprocessing technique that is used to speed up learning when tackling ML problems. Normalisation deals with the process of adjusting

the features in the dataset that have different scales, to fit in common scale across all features whilst keeping the relative differences in the ranges.

Therefore when normalisation of the input is so beneficial, why cannot we extend this concept to the other layers? Batch Normalisation (BatchNorm) is a technique developed by Ioffe and Szegedy (2015) that extends the concept of normalisation to all inputs of the hidden layers in a NN. The normalisation is applied at the output of each hidden layer after every mini-batch is processed during training. Batch normlaisation therefore extends the *basic* improvement obtained by normalising the input, to multiple layers repeated every iteration.

As the network grows deeper, the training becomes more complex. Every input at each layer is dependent on the output of the previous layers. Any *disturbance* in the one layer can be amplified as it is goes deeper in the network. Therefore the need for stability across deep networks is even more important. Ioffe and Szegedy (2015) comment that the training of neural networks is complicated due to the fact that the distribution of layer inputs change every time, after each mini-batch. If we consider a particular hidden layer, its input is coming from the previous layer. The hidden layer needs to try to learn the inputs of the previous layer to map it to the required output. However since the distribution of the input to the hidden layer is also changing, as it is an output of another layer, it is more difficult to train the layer.

The model would therefore need to adapt to these changes continuously. When the input distribution of an ML model changes it is termed as *covariate shift*, and the authors extends this concept to include each layer of the network, treating each layer as though it is a separate model. This can be seen as allowing each layer of the network to be more "independent" than the others. They term this approach *Internal Covariate Shift*. Ioffe and Szegedy (2015) have shown that in order to improve training, the internal covariate shift needs to be reduced. In order to achieve this they *stabilize* the network so that a layer's input remains with a fixed distribution during training. Ioffe and Szegedy (2015) apply the concept by LeCun et al. (2012) who showed that inputs with zero mean and unit variance train faster. Therefore, BatchNorm normalises the input of the next layer by subtracting the batch mean and dividing it by the batch standard deviation, so that each layers has the same mean and variance. During training this normalisation is performed at each layer using the batch mean and batch standard deviation for the particular batch samples only. Hence the name "batch" normalisation.

The optimisation algorithm such as SGD needs to be aware that normalisation is in place at each layer. Otherwise it can undo this normalisation during the update process, and it may change the effect of gradient descent that can cause the network to stop optimising the loss and lose its ability to converge. Batch normalisation is therefore

applied before the activation function, and the BatchNorm function is also included in SGD. In order for normalisation not to change what the layer represents Ioffe and Szegedy (2015) introduce two parameters for each BatchNorm layer function, γ and β, as learnable parameters through SGD. These parameters are used to scale and shift the normalised values so that the normalised output is multiplied by some standard deviation γ, and added some mean β, in order to maintain an identity transform and preserve the representation ability of the network. In summary, the benefits of batch normalisation include:

- **Improved training speed.** A drastic improvement in model performance was achieved in the experiments carried out by Ioffe and Szegedy (2015), that showed slightly better accuracy results with 14 times fewer training steps compared to their baseline model.
- **Allows usage of higher learning rates**. Since the values are normalisated, the network is more stabilised and less chance of large or very small activation outputs. Higher learning rates can lead to faster network convergence.
- **Less sensitive to parameter initialization.** Deep neural network can be sensitive to the initialisation of random weight which can lead to exploding gradients problem.
- **Regularization function.** Ioffe and Szegedy (2015) also showed that networks with BatchNorm required less dropout (Srivastava, 2013), or none at all in some cases. BatchNorm uses the mean and variance computed for each mini batch which represent only a small sample of the total training population. Ideally, the mean and variance of the whole training population are used to remove the internal covariate shift. Therefore this lead to inaccurate values of mean and variance. This lack of accuracy adds some noise to the model that acts a regularisation parameter.

2.2.6.1 | Instance Normalisation

Instance Normalisation (InstanceNorm) was introduced by Ulyanov et al. (2017) where the authors further develops the concept of Ioffe and Szegedy (2015) to optimise the speed and quality of image style transfer in computer vision. Style transfer refers to a technique where the style of one image is imposed on another image whilst keeping its semantic content.

BatchNorm uses all the mini-batch samples to collectively normalize the whole batch at each layer. However in InstanceNorm each sample, image in this scenario, is normalised individually. This normalisation at the sample layer is used to normalise the contrast in the image, effectively discarding the contrast which is not important for stylisation of the image (Ulyanov et al., 2017). Therefore apart from achieving the benefits discussed in the previous section, Ulyanov et al. (2017) managed to enhance the performance of the stylisation technique by incorporating the contrast normalisation function in the CNN architecture itself.

InstanceNorm is used in LigityScore model and apart the benefits highlighted for BatchNorm it also showed slightly better results and therefore was used as the main normalisation function in our experiments. The results are detailed in Section 4.3.

2.3 | Evaluation Criteria

The evaluation of scoring functions is done using a number of metrics that are referenced throughout the literature on ML based SF including Ballester and Mitchell (2010), Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018), and Zheng et al. (2019). These metrics include:

- **Pearson's Correlation Coefficient,** R. This coefficient investigates the relationship between two quantitative, continuous variables assuming a linear relationship, to measure their association strength. The coefficient ranges between 0 and 1, where '0' represents no correlation, whilst a '1' represents identical variables. In the context of scoring functions, R is used to find the correlation between the real binding affinity of experimentally validated protein-ligand complexes, and the predicted binding affinity from the output of our model. The closer R is to 1, the better the prediction ability of the model. Since the predicted affinity y should match the real affinity x at the input, the two variables have a linear relationship. R can be found using Equation 2.29.

$$R = \frac{\sum_i^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i^n (x_i - \bar{x})^2} \sqrt{\sum_i^n (y_i - \bar{y})^2}} \tag{2.29}$$

  where $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are the two independent variables represented the real and predicted binding affinity respectively for n protein-ligand complexes. $\bar{x}$ and $\bar{y}$ are the means of the two variables. Scatter plot are normally used to visualize the association strength.

- **Standard Deviation**. SD is defined as the standard deviation in linear regression and is computed using Equation 2.30.

$$SD = \sqrt{\frac{1}{n-1}\sum_{i}^{n}[y_i - (a + bx_i)]^2} \tag{2.30}$$

Similar to Equation 2.29, x_i and y_i represent the real and predicted binding affinities for the i^{th} complex. The a and b terms represent the intercept and the slope of the regression line when fitting x and y in $y = bx + a$. These terms are computed using a linear regression method to find the intercept and slope of the best straight line. The SD measure the spread of the data from the best straight line. The smaller the SD the more correlated is the data and the better the predictions.

- **Root Mean Squared Error (RMSE)**. The RMSE is used by the loss function to calculate the prediction error and is monitored during training. The RMSE measures the distance between the predicted output and the real values. RMSE is a negatively oriented score where a lower value is better. During training a model might overfit, and although the RMSE continues to decrease the prediction ability of the model does not improve or gets worse. Therefore, a lower RMSE during training is not necessarily the best performing model. The RMSE is also computed during testing for the best performing model so that it can be used for comparison with other experiments, and also with other research such as Stepniewska-Dziubinska et al. (2017) and Jiménez et al. (2018). The RMSE can be defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{2.31}$$

Equation 2.31 is very similar to the Euclidean distance apart from the $\sqrt{n}$ factor in the denominator which is used to make it a good estimator for the standard deviation of error distribution.

- **Mean Absolute Error (MAE)**. MAE is a measure of the error average magnitude ignoring the direction. Similar to the RMSE, the MAE was also monitored during training. The MAE for the test results were also computed for comparison reasons with other tests and literature. MAE is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{2.32}$$

2.3.1 | CASF Benchmark

The Comparative Assessment of Scoring Functions (CASF) benchmarks aim to provide an objective platform to assess scoring functions strengths and weaknesses using high-quality protein ligand complexes. The CASF benchmark was first published by Cheng et al. (2009b) and was termed CASF-2007. A major CASF update was done in Li et al. (2014b), and also in Su et al. (2018) which is the most recent edition. The last two updates are termed CASF-2013, and CASF-2016 respectively. The CASF-2016 includes significant improvements over the CASF-2013 including a larger, better quality test set, and improved evaluation methods. The CASF-2013 was compiled from the PDBbind refined set version 2013, whilst the CASF-2016 was compiled from the PDBbind refined set version 2016. The protein-ligand complexes were selected from the refined sets through a systematic, non-redundant sampling procedure, and are termed the Core Set 2013 and Core Set 2016 respectively.

The CASF benchmarks were designed purposely as a benchmark for scoring functions as a means to evaluate the performance of the SF more precisely. The objectives of the CASF benchmarks are to:

- Build a high quality dataset suitable for evaluation.
- Select suitable performance metrics to assess the SFs.
- The authors in Su et al. (2018) claim that the performance of a scoring functions should be measured outside the context of *molecular docking tests*, as this approach is not the best evaluation method for SFs since it can be effected by other factors involved in docking. Therefore the third objective was to decouple SF performance evaluation from molecular docking tests.

The rest of the discussion will focus on the CASF-2016 version since it is the latest update. The data set for CASF-2016 is based on high-quality crystal structures with reliable binding data. In order to ensure the *high-quality* a number to rules were defined to filter unwanted complexes. These rules are specified in the Supporting Information[2] of Su et al. (2018), and as an example they include; overall resolution < 2.5 Å, and its binding data must be in K_d or K_i.

The conformant complexes were subjected to a systematic, non-rendundant sampling procedure and were clustered using a protein sequence similarity algorithm where each cluster must have at least 90% similarity. Clusters with less than 5 complexes were

[2] https://pubs.acs.org/doi/suppl/10.1021/acs.jcim.8b00545/suppl_file/ci8b00545_si_001.pdf

discarded. From the remaining clusters, 5 complexes were selected to include complexes with the lowest and highest binding affinty, and 3 additional complexes whose affinity spreads as evenly as possible between the range of lowest and highest affinities of the cluster (Su et al., 2018). The authors also did a visual inspection of the electron density map for any defects using data directly from the PDB (Berman et al., 2003). Finally, all ligand molecules for the selected complexes were examined to make sure no molecule is identical or a stereoisomer. The selected high-quality complexes are used as the primary test set in the CASF-2016 benchmark and include 285 complexes.

In this study the CASF benchmark was used to assess the Scoring Power of LigityScore1D and LigityScore3D. The scoring power is quantitatively measured for evaluation using the Pearson's correlation coefficient, R, and the standard deviation (SD) in linear regression introduced earlier in Equations 2.29 and 2.30. The *Scoring Power* measures the ability of the model to map a linear correlation with the predicted and known experimental affinity values. The CASF benchmark includes also evaluation of the scoring function in terms of its Ranking power, Docking power, and the Screening power. These powers refer to the ability of a SF to:

- **Ranking power**. Select the correct, known binding pose of the ligand for the target protein.
- **Docking power**. Identify the correct ligand from a number of generated decoys for a given target.
- **Screening power**. Identify the true ligand from a pool of random molecules for a given target.

This study is focused to optimise the binding affinity prediction. A similar approach was taken in other literature namely Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018), and Zheng et al. (2019). The scoring power benchmarks will be used for objective assessment and evaluation of the proposed SF. Additionally, the model will also be compared to the work done by Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018), Zheng et al. (2019) since the benchmark tests do not include deep learning models. All CASF benchmarks are available online[3].

[3] `http://www.pdbbind-cn.org/casf.asp/` (last accessed 12-07-2020)

2.4 | Related Work

Scoring functions model prediction for binding affinity have been researched and developed for the last 20 years (Ain et al., 2015). Throughout the years SF methodologies have evolved from the classical functions, to a more machine-based approach which was mainly pioneered by Ballester and Mitchell (2010) with the RF-score model and was the first to achieve superior prediction results than classical SF.

In this section we will discuss the literature related to Scoring Function and how they evolved from classical models, to ML based model, and finally to DL models.

2.4.1 | Classical Scoring Functions

According to Li et al. (2019), *classical* scoring functions can be classified into three main categories: *Empirical*, *Physics-based*, and *Knowledge-based*. An overview of the types of SFs is shown in Figure 2.9 where the classical SFs, in general, include summation terms and are based on linear models. The classification of these scoring function is based on the feature components used to derive the equations. This additive terms of the classical SF are also *highlighted* in blue in Figure 2.9.

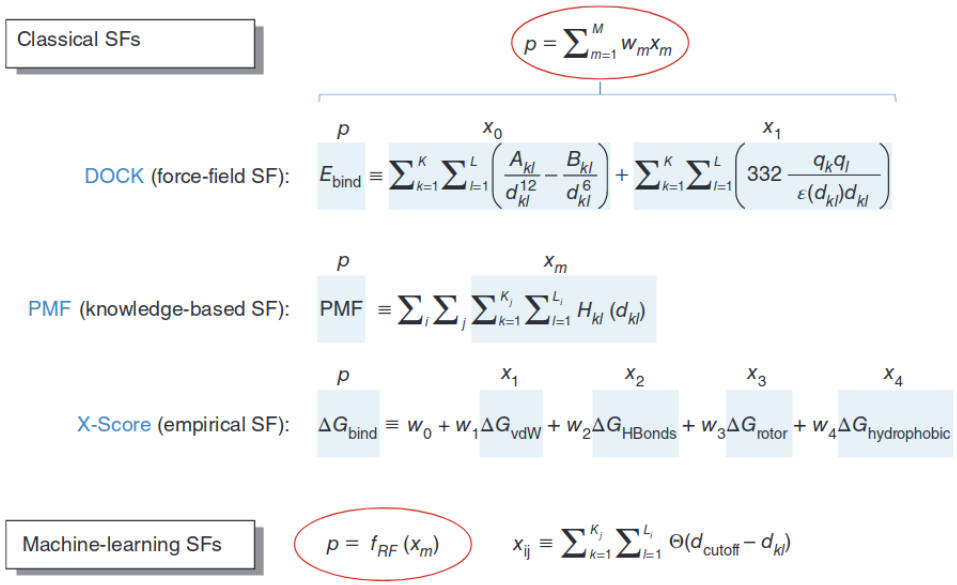


Figure 2.9: Scoring Function showing differences between classical and ML functions. Reproduced from Ain et al. (2015).

Empirical SF. Empirical SF usually include specific terms accounting for intermolecular interactions such as hydrophobic areas, and number of hydrogen bonds as shown

with X-score (Cheng et al., 2009a) example in Figure 2.9. Each term has a weight value that can be determined by linear regression or multi-linear regression. Although these can be deemed as ML models, they are still *classical* SF since they are considered simpler linear models (Ain et al., 2015). Empirical SF are popular as they are commonly used in docking programs such as Autodock VINA (Trott and Olson, 2010) which is one of the most famous classical functions and also one of the highest-ranking classical models on CASF-2016. Empirical scoring functions have a good compute performance however the underlying *simplistic* functional form and reliance on linear regression prohibit the empirical model to learn the true relationship between the protein-ligand complex and its binding affinity (Li et al., 2019).

Physics-field SF. Li et al. (2019) has further sub-categorised the physics-based SFs in another 3 methods — i) Force field, ii) Solvent models, and iii) Quantum mechanics. Force-field SF example includes terms with predetermined energy terms that take in account the interaction energies of the PL complex such as the *DOCK* SF shown in Figure 2.9. These energy terms include the van der Waals and electostatic interations between protein-ligand complexes atom pairs. These energy values are summed to compute the binding energy as represented in Equation 1 of Figure 2.10. Li et al. (2019) comments than force field techniques are somewhat restricted in providing an actual representation of the potential energy and other related parameters that effect prediction accuracy. The *solvent* models, as shown in Equation 2 of Figure 2.10, try to compensate for a better representation that the force-field method by adding solvation/desolvation effects (G_{solv}) to the atom interactions (example water). In Quantum mechanics, the scoring function is based on covalent interaction, polarisation, and charge transfer, to achieve higher accuracy at the expense on computation cost.

Knowledge based functions. Knowledge based functions are based on statistical rules extracted from analysis of a dataset protein-ligand complexes (the knowledge base), such as the Protein Data Bank (PDB). The Potential of Mean force (PMF) enhanced by Muegge (2006) derive statistical preferences as potentials for PL interactions and can be defined as the sum of interaction energies between all the PL atoms pairs for a particular distance.

An additional category has been defined by Su et al. (2018) where the newer *descriptor-based machine learning SF class* is included. Different ML SFs have been developed where most of them are generic for all protein families, however some researchers focused on a protein family specific approach (Ain et al., 2015). Ain et al. (2015) comment that in general specific protein family models do not improve the performance, since some protein families see a gain in performance, but others experience a decline. The following section briefly outlines the differences in the classical SF and discusses the limitations

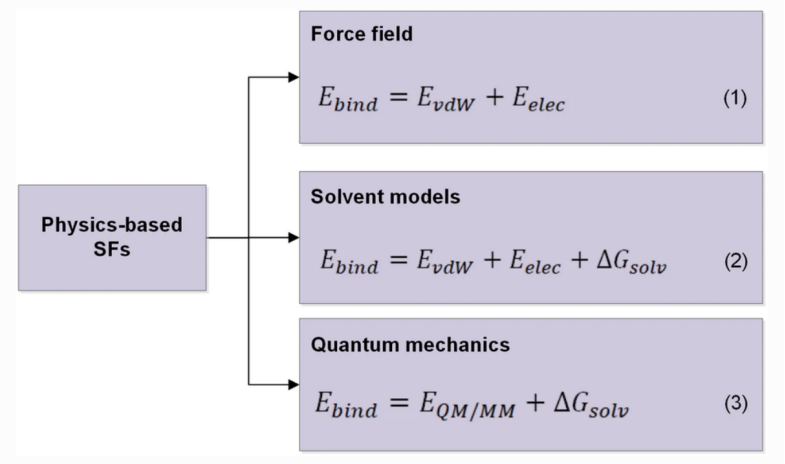


Figure 2.10: Physics-based classical scoring functions can be grouped into 3 categories — Force Field, Solvent Models, and Quantum Mechanics. Reproduced from Li et al. (2019).

experienced in such methods.

2.4.2 | Machine Learning to improve Scoring Functions

The ML models have been shown to outperform the accuracy of the binding prediction for classical methods (Ain et al., 2015). The difference in scoring power is due to several limitations of the classical models. One of the limitations is related to the linear functions used to define the classical models whilst ML techniques have the advantage to fit non-linear functions to the data. Therefore classical SF have a more imposed functional form defining the relationship between PL complexes, that is restricted to the parameters including the pre-defined model.

One of the major reasons for this increase in performance is the ability of ML models to use all available data for feature extraction. In ML models it was shown that adding more training data improved the predicted results. Throughout the years, new training data has been consistently been made available such as through various updates to the PDBbind datasets (Liu et al., 2017b), as is illustrated later on in Figure 3.2. Ain et al. (2015) have shown that as the training data availability increases across the versions of PDBbind, the results of the classical SF are unaffected, on the contrary to machine learning models. Ain et al. (2015) have compared this behavior using Autodock Vina with RF SF models. Therefore, the classical methods seem not to make any use of the new data as consistent results were achieved for bigger datasets. This also implies that as ML models use the new training data, then the performance gap to the classical methods

can also potentially increase.

This behaviour is related to the fact that classical models have predetermined functions (Figure 2.9). Classical scoring functions assume a predetermined relation that can be derived from statistical or empirical knowledge. However, since this is predetermined it is inherently a rigid approach that does not generalise well and would lead to poor predictions for those complexes that do not conform with the predetermined assumptions of the model. This contrasts with ML, where the function is not known before training, and the SF is built from the data describing protein ligand complex itself using representation learning. Therefore, ML models are able to use the additional data to adjust its function and improve its generalisation for better prediction results. Similar conclusions were also highlighted in the work of Li et al. (2015) where the authors showed that better correlation coefficients between actual and predicted affinities where achieved over Autodock Vina (Trott and Olson, 2010) as more training data was used.

2.4.3 | Machine Learning Approaches

The pioneers of ML approaches have used these methods way back in 2004. Deng et al. (2004) have used kernel partial least squares as the first approach to non-liner SFs. The first Neural Network (NN) based SF was also implemented by Artemenko (2008). However, these early approaches achieved modest results, very similar to the results achieved by classical scoring functions. These researchers did not have an available test set for benchmarking, so it was difficult to rank which of these SF performed better and it was difficult to compare to the state of the art.

In 2010, RF-score (Ballester and Mitchell, 2010) was the first machine-learning method to significantly obtain a higher performance in affinity prediction and is one of the most prominent examples of ML SFs. The RF-score used simple element-to-element counts as features between atoms of the protein and ligand around the binding site. These descriptors were used to train a Random Forest (RF). NN-score (Durrant and McCammon, 2010) which uses a number of neural networks was also proposed in 2010 and also showed good results, significantly higher than the classical methods. These two approaches have generated a lot on interest to utilise ML for SBVS binding affinity prediction and inspired the new wave of SF.

However, Gabel et al. (2014), have indicated that although RF-score (Ballester and Mitchell, 2010) achieved great results in terms of scoring power, when tested on docking tests it performed poorly and much worse than the classical methods. Su et al. (2018) define scoring power as the ability of a SF to predict the binding affinity in a linear corre-

lation with the experimentally known affinities. On the other hand they define docking power as the ability of a scoring function to identify the native ligand binding pose among computer-generated decoys. Gabel et al. try to understand how a ML model with very simple descriptors used in RF score (element to element distance counts), can outperform models using more descriptive protein-ligand interaction attributes. Gabel et al. (2014) hint that such models are attributed to non-causal bias, explained more recently by Sieg et al. (2019) and is discussed in more detail later in Section 2.4.4, where the models might be performing well but for the wrong reasons.

Gabel et al. (2014) conclude that the scoring function is used also for VS and pose prediction, and it is important to evaluate the scoring function also in terms of docking power and screening power to validate the capacity of the SF to enrich hit lists, so that the SF is not built in isolation.

Wang and Zhang (2017) have later counter argued the criticism done by Gabel et al. (2014) on the extent that ML models create black boxes, and provided insight into the decisions being taken by the RF. Wang and Zhang (2017) further explain that the poor results in docking tests, occurred because the RF model used by Ballester and Mitchell (2010) is not able to extrapolate the regression function for unseen data, but is limited to interpolation within the range that was available during training. A predicted output, y_{pred} for a given test input is within the range of $min(y_{train})$ and $max(y_{train})$. Therefore, any unseen data outside the binding affinity of the training data would be predicted as either the min or max of the range of the training data. Wang and Zhang (2017) comment that this extrapolation feature is required for docking and screening, and is the reason why RF-score (Ballester and Mitchell, 2010) performed badly in docking tests.

In order to cater for the bottleneck in RF-score (Ballester and Mitchell, 2010), Wang and Zhang (2017) have used RF to optimize the Autodock VINA classical SF. Autodock VINA being one of the best performing classical SF obtained good scoring and screening power results, provided a good base score and tackled the extrapolation issue. The RF model was used as a *correcting* factor to the VINA score. Wang and Zhang's new model, ΔvinaRF20, was therefore composed of two components. The first component was the VINA score and the second is a RF correcting factor based on 20 pharmacophore-based features, to take advantage of the RF improved scoring accuracy. ΔvinaRF20 achieved top rank in all power tests in the CASF-2016 benchmark. Developing a robust protein-ligand scoring function that is able to improve the 3 SF use cases is major challenge in the area of structure based drug design. ML functions still rely on feature engineering as discussed by Stepniewska-Dziubinska et al. (2017), that requires expert knowledge to preprocess the data.

2.4.4 | Deep Learning Approaches

Deep Learning (DL) has been recently applied for drug discovery, where multiple hidden layers are used to build a deep neural network (DNN). The key advantage of DNN is the ability to automatically extract features and create a complex representation of the data through the hidden layers without the need of applying feature engineering (Pérez-Sianes et al., 2019). Feature extractions thus occurs as a natural byproduct of fitting the input to the model's parameters. Deep learning models allow for further reduction in feature engineering and reliance on expert knowledge. The first deep neural network used for VS was introduced by the winning team of the 2012 Kaggle Merck Molecular activity challenge (Kag, 2012) where the team applied a multi-task deep feed forward network for quantitative structure-activity (QSAR) LBVS. This work was later published by Ma et al. (2015) which generated a lot of interest and excitement for the use of DL in this field. Ma et al. (2015) have achieved an average Pearson correlation coefficient of 0.496 using a multi-task DNN compared to the 0.423 obtained when using an RF model.

Figure 2.11 provides an overview of the deep learning methods utilised specifically for SF available in the literature after the work of Ma et al. (2015).

The Convolutional Neural Network (CNN) is one of the most common DL architectures used for SF. The CNN uses a number of sequential layers of convolutions and pooling modules to encode the hidden features of the data, and then use a fully connected feedforward NN layers for classification or regression. One of the advantages of CNNs in the area of structure-based drug design is its ability to capture local spatial information within the input. This property can be used to capture spatial information of interactions between protein-ligands. CNN models have been applied to SF development (Jiménez et al., 2018; Liu et al., 2019; Ragoza et al., 2017; Stepniewska-Dziubinska et al., 2017; Zheng et al., 2019). CNN models have achieved great results for image recognition and is also a popular method for SBVS as reported in Rifaioglu et al. (2018) and Pérez-Sianes et al. (2019). Table 2.1 provides a summary of the methods discussed in this section highlighting the methods, evaluation, and results achieved.

CNN when compared to more traditional ML methods (such as methods used in RF-score and NNscore), do not rely on expert knowledge for feature selection but uncover the hidden features automatically directly from the data. In doing so, the resultant model can be seen as a *black box* where the actual features that are important to the model may be difficult to determine.

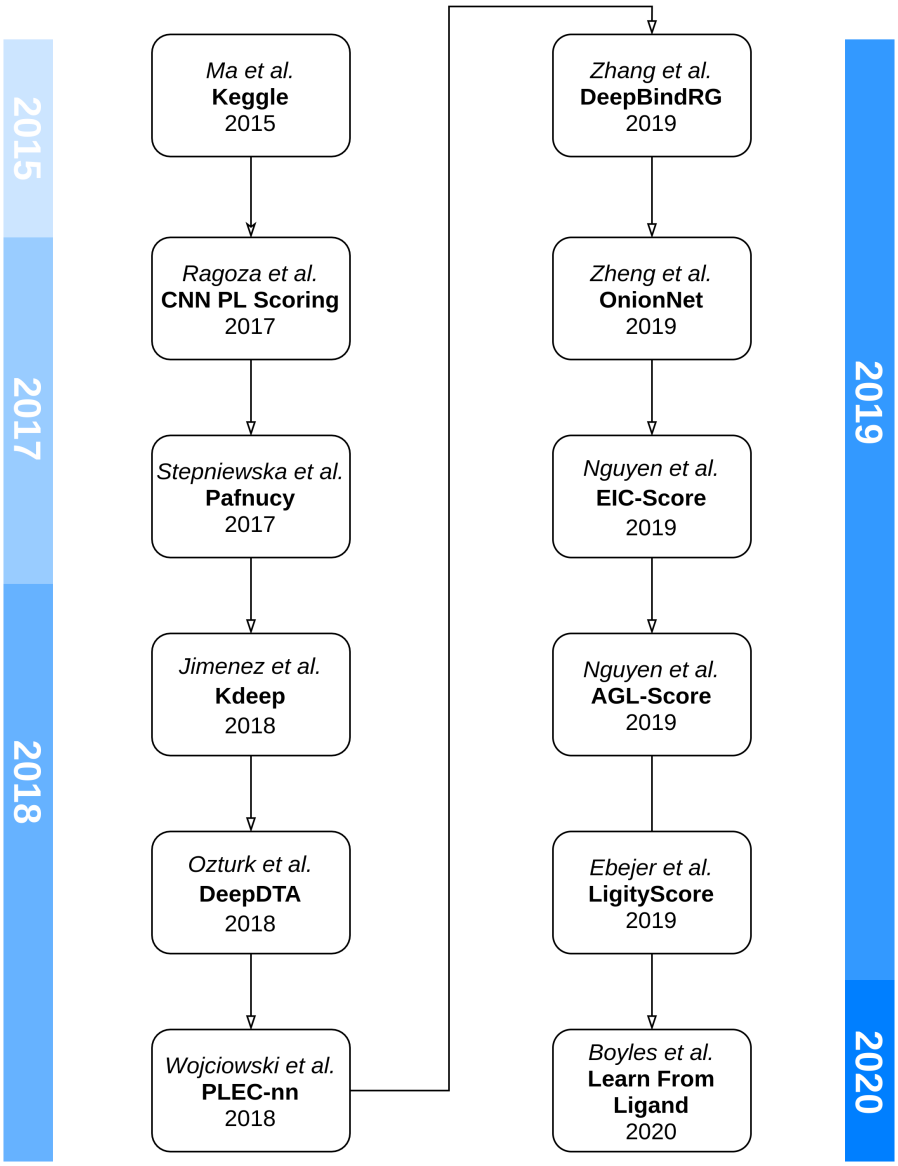


Figure 2.11: Deep Learning based Scoring Function literature overview, providing a comprehensive summary of the literature available. The sequence of literature highlighted here also represented the sequence used to introduce this literature in this section. Deep learning methods for scoring functions are relatively recent considering that a first DL method was published in 2017. The work of Ma et al. (2015) is not directly on scoring functions, however it is included for its importance in defining DL methods for use in VS techniques.

Table 2.1: Summary of literature review highlighting approach, evaluation dataset and metric, and results for the literature discussed in Section 2.4.4 to 2.4.6.

Method	Year	Authors	Approach	Eval Set	Eval Metric	Results
Kaggle Challenge	2015	Ma et al.	Multi-task DNN	Kaggle Merch Set	R-Score	0.496
CNN Scoring	2017	Ragoza et al.	CNN	DUD-E CSAR	AUC	0.868
Pafnucy	2017	Stepniewska et al.	3D CNN	CASF-2013 CASF-2016	R-Score	0.780 0.700
Kdeep	2018	Jimenz et al	3D CNN	CASF-2016	R-Score	0.820
DeepDTA	2018	Ozturk et al.	CNN - Smiles and Protein sequences.	Davice KIBA	CI (Concordance Index)	0.878 0.863
PLEC-nn	2018	Wojciowski et al.	ANN	CASF-2013 CASF-2016	R-Score	0.774 0.817
DeepBindRG	2019	Zhang et al.	ResNet CNN	CASF-2013	R-Score	0.639
OnionNet	2019	Zheng et al.	CNN	CASF-2013 CASF-2016	R-Score	0.782 0.816
EIC-Score	2019	Nguyen et al.	Differential Geometric with ML (GBT)	CASF-2013 CASF-2016	R-Score	0.774 0.826
AGL-Score	2019	Nguyen et al.	Algebriac Graphs with ML (GBT)	CASF-2013 CASF-2016	R-Score	0.792 0.830
Ligity	2019	Ebejer et al.	Knowledge-Based	DUD-E	AUC	0.44-0.99
Learn From Ligand	2020	Boyles et al.	ML + Ligand RDKit features	CASF-2013 CASF-2016	R-Score	0.786 0.826

Ragoza et al. (2017) were the first to successfully use CNNs to implement a DL scoring function that predicts the docking score for a drug target interaction which was then used for SBVS and pose prediction. Earlier attempts in 2015 were made by Wallach et al. (2015) on AtomNet , however their research is yet unpublished. Ragoza et al. (2017) claim that their CNN SF outperforms Autodock Vina on the DUD-E (Mysinger et al., 2012) and CSAR (Dunbar Jr et al., 2011) datasets for both screening and pose prediction tasks respectively. For the input of the CNN, Ragoza et al. have discretized the protein-ligand structure into a 3D grid of 24Å for each side which was centered at the binding site. The 3D grid is composed of multiple 2D grids — one for each heavy atom of the protein-ligand complex analogous to RGB channels in images. Each element in the grid stores information about the type of atom and at that particular location. The atom grid information representing the particular heavy atom at 0.5Å resolution discretised positions is represented using a density distribution function $A(d, r)$, dependent on the distance d from atom centre, and the van der Waals radius, r. Although, Ragoza et al. (2017) utilize the CNN for automatic feature extraction, the authors still applied an element of feature engineering with the way they have represented the atoms using their own density distribution function $A(d, r)$. Ragoza et al. (2017) obtained an AUC of 0.868 for screening using their developed scoring function which outperformed Vina (0.716), RF-Score (0.622), and NN-Score (0.584). Pose prediction assessment of the CNN SF to distinguish top poses obtained results with an AUC of 0.815, compared to Vina's performance 0.645. Ragoza et al. (2017) also comment that rotations of their data structures were used for data augmentation and improved their test results considerably. Since each layer in their interpretation considered atom positions, rotations are essential part of training, to make it less sensitive to different protein-ligand orientations.

However, recently Sieg et al. (2019), have reported that the benchmark datasets such as the DUD-E (Mysinger et al., 2012) amongst others are not suitable for ML approaches to VS. They comment that ML models are considered as black boxes, difficult to interpret and identify which features are contributing to predicted results. Sieg et al. (2019) have also reproduced the work done by Ragoza et al. (2017) to evaluate bias information in the dataset, and have made some bold conclusions. Namely, that the CNN models are not learning from the features of the protein but are instead just learning from the bias found in the small molecule features. To test this claim, Sieg et al. (2019) re-implemented the training method used by Ragoza et al. (2017) but excluded all the protein features from training. This adjusted model was able to reproduce screening results almost identical to those when the protein features were used. The authors term this type of bias as *non-causal* bias, as good results were obtained however the results do not represent any binding mechanism of the protein-ligand interaction.

Additionally, Sieg et al. (2019) also claim that in reality the CNN model provides little improvement in the performance over a more traditional ML approach such as random forest (RF) when the DUD-E dataset was used for training and testing. Sieg et al. (2019) has also investigated their claims on other CNN such as the work done by Pereira et al. (2016) (DeepVS), and have reached similar conclusions both in terms of CNN performance and the learning ability from the structure. Pereira et al. (2016) have used CNN to enhance the performance of DBVS.

Therefore, given the contribution made by Sieg et al. (2019) it seems that with the current available datasets it is difficult to evaluate different VS model as the benchmark datasets are affected by bias. Sieg et al. provide some recommendations on a possible approach moving forward, however do not provide an alternative benchmark for ML in SBVS. Therefore, until a benchmark is curated specifically for ML, it is difficult to test and evaluate different ML and DL models. One of the main reasons of this *failure* is due to the artificially created decoys in the DUD-E dataset. The decoys were created to have a similar structure and physiochemical properties such as molecular weight (MW), LogP, and number of hydrogen bond acceptors, of the actual ligand. This obviously presents a new challenge to find a suitable dataset suitable for ML techniques.

In our approach the PDBbind dataset is used which contains a list of experimentally validated complexes, which include also experimentally measured binding affinity data. This dataset is detailed in Section 3.3. Since these are experimentally validated complexes they do not suffer from bias introduced by artificially created decoys. Due to the unavailability of experimental results for inactive molecules it is difficult to include the inactives in the training. Binding affinity data may also not be available for inactive molecules and therefore it might be impossible to obtain such data to use for training. Wójcikowski et al. (2017) have added training decoys by setting an identical value that represents a known non-binding affinity value for all the decoys used, however do not comment if these improved the results.

2.4.4.1 | Conv3D CNN Scoring Functions

Pafnucy proposed by Stepniewska-Dziubinska et al. (2017) is one of the most promising DL SF models where the authors achieved a Pearson Coefficient, R, for the predicted versus actual binding affinity of 0.70 on the CASF-2013, and 0.78 on the CASF-2016 benchmarks respectively. Stepniewska-Dziubinska et al. (2017) claim that their pafnucy model was outperformed only by RF-scorev3 (Li et al., 2015) with R of 0.74.

Stepniewska-Dziubinska et al. (2017) continued on the work done by Ragoza et al. (2017) to build a 3D CNN model with a 4D tensor to represent 19 protein-ligand fea-

tures in 3D. The 4D tensor includes discretised atom location in the first 3 dimensions, whilst the features for the particular atom are encoded in the 4th dimension. This can be visualised as a 3D object with 19 different channels. Stepniewska-Dziubinska et al. (2017) label their model as 3D CNN as they utilise the *Conv3D* Tensorflow module to train the 3D model with 19 input channels. In order to obtain consistency in input feature sizes, the complex was cropped to a defined size of 20Å, resulting in 21 discrete locations. Any atoms that are *outside* this boundary are not included as they are assumed not important for interactions at the binding site. The concept of input features is similar to those presented by Ragoza et al. (2017), however Stepniewska-Dziubinska et al. extend the input to 19 features, which include:

- **Atom types**. Heavy atoms types use a 9-bit encoding to represent elements B, C, N, O, P, S, Se, halogen, and metal types.
- **Hybridisation**. Hybridisation state are represented by one integer $\in [1, 2, 3]$
- **Heavy Valence**. Integer to count the number of bonds with other heavy atoms.
- **Hetero Valence**.Integer to count the number of bond to other heteroatoms.
- **SMARTS**. 5-bits to encode atom properties from hydrophobic, aromatic, acceptor, donor, and ring. SMARTS is a type of language, similar to SMILES, that describes molecular patterns used to search such molecular substructures.
- **Partial Charge**. Partial charge float value for the atom calculated using UCSF Chimera (Pettersen et al., 2004) software.
- **MolType**. A value of '1' is used to represent a ligand atom, whilst '-1' represents protein atom.

This Pafnucy protein-ligand complex representation can be generated using tf.bio[4] python package developed from the same author. These input features are fed to a 3 tier convolution-pooling layers, and a 3-tier fully connected network with a single output neuron. A key difference from Ragoza et al. (2017) is that the Pafnucy model is used as a regression model for binding affinity prediction only, and was trained on binding affinity data from PDBBind. The PBDbind v2016 was split in training, validation, and testing datasets. The core set was used for testing, whilst 1,000 complexes from the refined set were used during validation, whilst the general set and the rest of the refined set were used for training. A similar approach was used to split the datasets

[4] `https://gitlab.com/cheminfIBB/tfbio` (last accessed 26-06-2020)

for LigityScore. The Pafnucy model includes the locations of the atoms as part of the representation. In order not to have a model that is sensitive to the PL orientation, Stepniewska-Dziubinska et al. (2017) have augmented their data to include all possible 90° rotations of their 20Å space. Therefore training was performed on 24 different rotated variants of each complex and showed that the model extracted the same information from different input representation. To explain how the model is using the input data, the authors have extracted the weight ranges for the 1st layer of the CNN. This has 19 input channels corresponding to each of the features mentioned earlier. Pafnucy uses also L2 regularisation in the Adam optimisation (Kingma and Ba, 2014), which has the capability of zeroing unimportant features. The channel with wider ranges are passing more information to subsequent layers and therefore are more important. The top four influential features includes moltype, hydrophobic, donor and aromatic features, whilst the least important where atom type *B* and *Se* which is expected since these atom type are present less frequently. The feature influence was determined by looking at the distribution of weights in the first convolution layer. Since each feature (example B or C elements) represents a separate input channels, the weights at the first layer would correspond to the particular feature only. Stepniewska-Dziubinska et al. (2017), therefore take the features with the larger range of weight values as the most influential. The 4D tensor used was well suited for CNN learning and effective to keep spatial and chemical information.

The K_{deep} (Jiménez et al., 2018) CNN model achieved state of the art results on the PDBBind v.2016 results core test with R value for 0.82, early in 2018. Jiménez et al. (2018) have used the general and refined sets for training, similar to approach taken by Stepniewska-Dziubinska et al. (2017). However, they have extending their testing to various CSAR Dunbar Jr et al. (2011) datasets due to claims by previous authors Gabel et al. (2014) that testing on core set might provide over optimistic results. Jiménez et al. (2018) have used input descriptors similar to Ragoza et al. (2017). Their input features use a 3D voxel representation where each channel encodes a particular property of the atom. Similar to Pafnucy, K_{deep} also used 3D CNN. Each protein-ligand complex is represented by a 4D tensor, where each 3D hyper-plane representes the protein-ligand complex with respect to a particular property only as shown in Figure 2.12. The eight properties chosen for K_{deep} include: hydrophobic, aromatic, HBA, HBD, Cation, Anion, Metallic, and excluded Volume. These are detailed in Table 2 of Jiménez et al. (2018).

Therefore, the 3D plane represents the PL complex with respect to that property only. If we take the 3D hydrophobic *layer*, all the hydrophobic atoms are selected, whilst the rest are of the atoms are discarded. In order to transform the selected atoms to a suitable numerical representation, a function $n(r, r_{vdw})$ is used to determine the contribution of

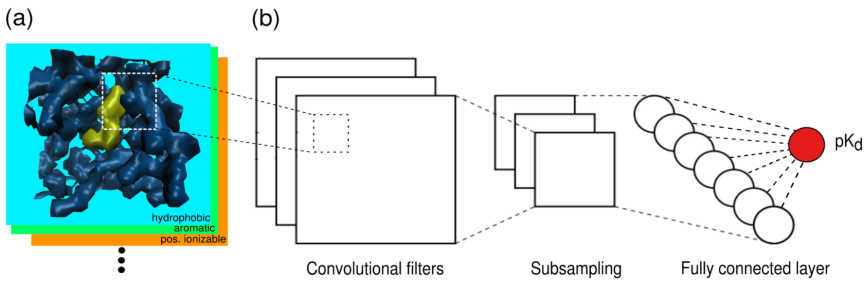


Figure 2.12: The K_{deep} scoring function reproduced from Jiménez et al. (2018). (a) protein and ligand are featurised using eight pharmacophoric-like properties. (b) the descriptors are processed by a Conv3D CNN for training of the binding affinity relationship and prediction of unseen examples.

each atom that depends on the Euclidean distance r and the Van der Waals radius r_{vdw}. This is similar to the method used by Ragoza et al. (2017) of the distribution function. The other properties are extracted the same way to construct the 4D representation.

K_{deep} using an architecture similar to Squeeznet (Iandola et al., 2016) but shallower due to smaller input ($24 \times 24 \times 24 \times 8$). ReLU is used for activation, whilst Adam optimistation with default parameters, mini-batch size of 128, and a learning rate of 10^{-4}.They have achieved good results in the CASF-2016 benchmark on scoring, and have also outperformed Stepniewska-Dziubinska et al. (2017). Other SFs such as RF-score Ballester and Mitchell (2010) and X-Score Cheng et al. (2009a) outperformed K_{deep} in the CSAR test sets, with an average difference in R of 0.11 and 0.05 respectively.

2.4.4.2 | CNN Scoring Functions

Öztürk et al. (2018) propose a binding affinity prediction model based on only sequence information of the protein-ligand complex. This approach varies from the previous mentioned model in the sense that it uses a 1D feature representation instead of multi-dimensional ones. For the 1D feature representation Öztürk et al. (2018) use the raw protein sequences and the ligand SMILES strings to extract information using separate CNN blocks. These two CNN blocks are combined with a common FC layers block, to output binding affinity prediction. This topology was termed *DeepDTA*. The SMILES and protein sequences where analysed and the authors extracted 64 and 25 categories (unique letters) respectively. The SMILES and sequences were then encoded so that each category corresponds to an integer (example, 'C':1, 'H':2, 'N':3, etc). Similar to Pafnucy and K_{deep}, DeepDTA uses Adam optimisation with default parameters including learning rate, and ReLU activation. Öztürk et al. (2018) have evaluated DeepDTA on

the Kinase dataset Davice and KIBA dataset, and report better results from their baselines. Since their dataset and evaluation method is different from the other literature it is difficult to compare DeepDTA with them.

CNN models have also been used for specific VS applications. Liu et al. (2019) have developed a CNN model labeled *DeepSeqPan* for binding affinity prediction specifically for the HLA protein and peptide binding. Their model is interesting as it was designed for multi-task output using two feed forward components after the convolution and pooling layers — opposite of DeepDTA. These components are used to build both regression and classification outputs that predict the binding affinity, and predict the binding probability respectively. For their input they use the sequence of the peptide and the protein to map the sequence using one-hot encoding. Therefore, for a given peptide composed of 9 amino acids, a matrix of 9×20 is created to represent the peptide since there are 20 amino acids in total. A similar approach was taken for the protein sequence encoding, and both encodings were used to feed a CNN network. DeepSeqPan include BatchNorm layers that are also applied before activation, and utilise the Leaky ReLU activation. Liu et al. (2019) have thus used a simple model for the protein and peptide interaction that uses one hot encoding, and use the CNN for automatic feature extraction to uncover any hidden patterns in the data.

Zhang et al. (2019) tackle the problem of binding affinity prediction using a CNN architecture based on the *ResNet* model (He et al., 2016), inspired by the Pafnucy's CNN model. Zhang et al. (2019) mention the difficulty in tackling the open problem of how to best represent the protein-ligand interaction data and use ResNets to include much deeper networks capable of extracting more complex features. The authors have created a 2D input from one hot representation to encode different atom types of the ligand and protein separately. Each ligand atom was represented using an 84 dimension one hot vector, whilst the each protein atom was represented using a 41 dimension one hot vector. The ligands were grouped in 3 categories based on their value of logP. The interaction between ligand and protein atoms is computed for a max distance of 4Å to maintain the important contact information. The extracted atoms pairs are then converted to their respective one hot encoding values. The atom pairs were limited to 1000, to maintain a fixed size input. Complexes with less pairs were padded with zeros, whilst those with more pairs, which were rare, were discarded. Therefore the input size for each complex representation was of size 125×1000. Zhang et al. (2019) label their model as *DeepBindRG*.

The authors used various dataset including the PDBbind v2018 for training and validation, whilst used the CASF-2013 (Li et al., 2014a), CSAR (Dunbar Jr et al., 2011), and Astex Devierse Set (Hartshorn et al., 2007) were used for testing. The ResNet was con-

structed with 7 convolutional layers using 1×1 and 3×3 kernels sizes, one max pool layer and a FC layer with a single output neuron for affinity prediction. Zhang et al. (2019) report a R value of 0.6394 on the CASF-2013 test set. Although DeepBindRG has a better performance than Autodock Vina on the CASF-2013, their results show worst performance than the previously mentioned models.

A recent study proposed by Zheng et al. (2019) compares their model to Stepniewska-Dziubinska et al. (2017) and criticize the Pafnucy model that the protein-ligand interactions in a 3D grid box of 20Å are not sufficient to capture all the PL interactions, and suggest that other long-range interactions outside the 20Å , termed *non-local* electrostatic interactions are also important. In order to capture all the interactions between protein-ligand complexes Zheng et al. (2019) divide all the 3D space of the binding site into a number of shells or zones as shown in Figure 2.13, and count the number of different element to element interactions within each shell. This method is rotationally invariant since the same element to element count are taken irrespective of the orientation of the complex. This element to element interaction count was inspired by the RF-score model Ballester and Mitchell (2010). Zheng et al. (2019) further comment that their model has the advantage of using simple features that do not introduce additional hypothesis and estimations to avoid extra noise or bias. Zheng et al. (2019) mentions that the partial-charge is one such feature used by Stepniewska-Dziubinska et al. (2017) as it calculated differently using various assumptions.

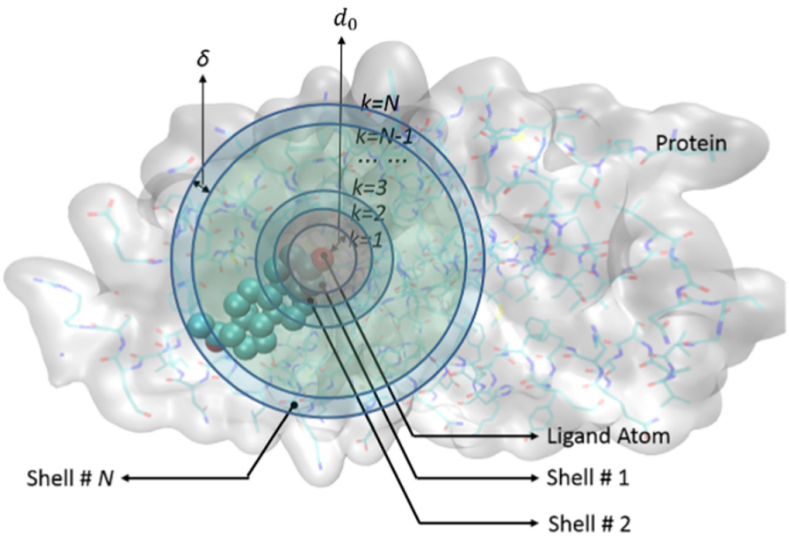


Figure 2.13: OnionNet boundary shells partitioning of the protein around the ligand in 3D space. Reproduced from Zheng et al. (2019).

The boundary shells, symbolised the *onion* shell, were created around each atom of the ligand. RF-score considered 9 different elements for the protein and ligand, and this resulted in a possible 81 features. On the other hand OnionNet used 8 different elements for a total of 64 features however different zones are considered to count the element interactions and not a single cut-off distance. Zheng et al. (2019) defined 60 zones and obtain an input dimension of 64×60 for each complex, that is analogous to a grayscale image. Using this approach Zheng et al. (2019) trained their model on the PDBbind v2016 dataset and benchmarked on the both v2013, and v2016 core sets, and with the Pafnucy model. They have achieved a value for R of 0.816 on the 2016 core dataset and R of 0.782 for the 2013 dataset — a significant improvement over the Pafnucy model, however similar performance to K_{deep}.

In order to get more insight on the features extracted by OnionNet, Zheng et al. (2019) conducting tests that excluded one shell at a time during the training to uncover which shells contribute more to the Pearson Coefficient. The loss with the removed shell was compared with the loss of the best performing model. The larger the change in loss noticed, the more important is the feature. Their experiments show that the shells closer to the ligand are more important, as was intuitively expected, however also show that non-local interactions have significant importance. Zheng et al. (2019) explore further the different feature importance of element-pair combinations, where the Oxygen-Phosphorus, and Carbon-Sulfur pair were the most significant. Zheng et al. (2019) conclude that the enrichment for these less common atom type might be used reference points during feature extraction. The OnionNet CNN model using 3 convolution layers with kernel size of four and no pooling and BatchNorm, followed by three fully connected layer and a single output neuron.

As described earlier, Gabel et al. (2014) show that the RF-score model performed badly in docking and screening tests. In their study Zheng et al. (2019) do not mention that they have tested OnionNet for docking and screening tests. Therefore since they use similar features to RF-score, it is not known if OnionNet performs also badly in docking and screening tests. However, a recent study by Shen et al. (2020) answers these queries.

Shen et al. (2020) has criticised ML based scoring functions on their lack of usability due to poor performance in screening, ranking, and docking tests. The authors highlighted the need of a systematic assessment of these ML SFs. They report on the limitation of the modern SF to perform in all CASF power scores mentioned in Section 2.3.1. To systematically assess ML SFs, Shen et al. (2020) choose a number of SFs and assess them on all the powers of the CASF v2016 benchmark. These include the Pafnucy and OnionNet models introduced earlier, that were only assessed on scoring power by their authors. Therefore, the work Shen et al. (2020) seems a continuation of the previous

work done Gabel et al. (2014), and seeks to explore the usability of these scoring functions. Shen et al. (2020) comments that most of these ML SFs, including Pafnucy and OnionNet, do not perform well in the CASF screening and ranking powers, and some even perform worst than classical functions.

One of the main reasons for bad performance is the fact that most of these models are trained only on valid protein-ligand complexes. The training does not include any decoys, and prediction for such cases is intuitively difficult. ML SFs rely on representation learning during training to extract the underlying feature maps. Ranking and Screening powers requires the evaluation of non-binding protein-molecule pairs, and if such cases are not used in training it is difficult to have good performance in these areas. Therefore this might suggest that a different training set may be required for different SF tasks.

2.4.5 | Ligity Representation

One of the limitations of $Pafnucy$ and K_{deep} is the dependency on the coordinate frame. 3D CNNs treat the structure like a 3D images. The representation can be thought of as one snapshot of the structure. However if the orientation from where the snapshot is taken is changed, a different representation of the *same* things is obtained. The authors have worked around this limitation by introducing different systematic rotations of the same input during training. However, these might present additional challenges when testing novel complexes that can take different orientations.

This limitation has led us to explore methods that are inherently rotationally invariant. One such model that is not dependent on the coordinate frame is *Ligity* developed by Ebejer et al. (2019). Ligity is a hybrid VS technique that collects key interaction features within the protein-ligand complexes. These key interaction points are known as 'hotspots' and are defined by considering specific pharmacophoric features that lie within a predetermined distance threshold between the protein and ligand feature pairs. Each of these pharmacophoric features that *interact* together are termed *Pharmacophoric Interaction Points* or PIPs. Once these pairs are extracted, the Ligity descriptor for the ligand is created by considering only the PIPs from the ligand space. Two variants of descriptors were considered by Ebejer et al. (2019) that use 3-PIP or 4-PIP combinations. Combinations of all possible 3-PIP or 4-PIP sets from the ligand space are considered when creating the descriptor. Considering the 3-PIP case for simplicity, the distances between the 3-PIP triangular structure created in space are extracted. These distances are discretised to find the co-ordinates of a 3D hypercube, representing the Ligity descriptor, so as to increment the count at that voxel location as shown in Figure 2.14. All 3-PIP combinations from the ligand pool are used to increment the respective location of the 3D

hypercube. This is represented in the schematic of Figure 2.14. Since the Ligity descriptor is built using the spatial distribution of PIPs, this descriptor is rotationally invariant and thus suitable for our required representation.

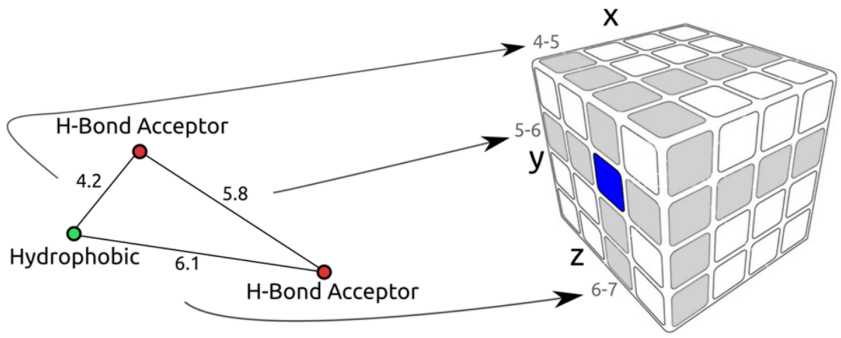


Figure 2.14: Ligity 3-PIP triangular structure showing mappings of the distances as 3D coordiantes to update binning count on the 3D hypercube for features HBA, HBA, hydrophobic. Reproduced from Ebejer et al. (2019).

These hypercube Ligity descriptors are used to build a representation from a known protein-ligand complex. In a virtual screening exercise, Ligity descriptors for libraries of small molecules are used to compute a similar descriptor using the same target protein. The known ligand descriptor and the small molecules descriptors are compared using the Tversky similarity measure to find their respective similarity score. This list of molecules for each protein-ligand complex is then ranked by decreasing similiarity score. A high similarity score represents a potential *hit* for that target protein. Ligity was benchmarked using the DUD-E database (Mysinger et al., 2012), and Ebejer et al. (2019) report a mean area under the receiver operation characteristic curve (AUC) range from 0.44 and 0.99 for different protein families when using 4-PIP descriptors and the Tversky similarity. Ebejer et al. (2019) also showed that their method is computationally efficient and is 20 times faster than docking with Autodock Vina (Trott and Olson, 2010).

2.4.6 | Recent ML approaches

Zheng et al. (2019) (OnionNet) compare their method to another recent model, *AGL-Score*, and comments that its authors, Nguyen and Wei (2019a) provided a more *complete* local environment where they managed to improve the affinity prediction performance with an $R = 0.833$. To date this represents the highest performing ML scoring function. The better results are achieved as they add novel features relating to the physical and biological information of the complex using graph theory.

Nguyen and Wei (2019a) categorise the complex into multiple algebraic sub-graphs that describe different molecules and their interactions. A subgraph can be represented in matrix format to describe the interaction between the subgraph elements, and Nguyen and Wei (2019a) use the *Laplacian* matrix, its pseudo-inverse, and adjacency matrix to achieve this. The eigenvalues and eigenvectors of these matrices are then used to extract the AGL-Score molecule representation that they can use for ML. Although the authors claim that the protein-ligand representation can be used by several ML algorithms they adopted a *simpler* ML model and utilised gradient boosting trees (GBT) with 10,000 estimators, depth of 7, and a learning rate of 0.01. Although this is not a deep learning model it represents an important development in scoring functions for SBVS. AGL-Score is benchmarked using CASF-2007, CASF-2013, and CASF-2016 on scoring, ranking, docking, and screening powers and achieved very positive results results which are the best performing scoring function to date. *AGL-Score* was the top performer for CASF-2013 on scoring, ranking, docking, and screening powers. Therefore, contrary to Shen et al. (2020) ML based scoring functions can still perform better and outperform classical scoring functions in all the benchmark powers of CASF. Shen et al. (2020) on the other hand acknowledged that AGL-Score had balanced powers and confirms it is the best performing ML SF, however they mention that its applicability might be limited for practical cases, and fails to provide a clear description of this. The same authors of AGL-Score, have published another study based on differential geometry earlier in the same year and term this method EIC-Score (Nguyen and Wei, 2019b). The authors use differential geometry based features to represent the protein ligand complex with GBT models, and achieved a best R-score performance of 0.774 and 0.826 for the CASF-2013 and CASF-2016 respectively.

Wójcikowski et al. (2019) introduce a novel interaction fingerprint (IFP) to train a SF for prediction of binding affinity which is termed Protein-Ligand Extended Connectivity (PLEC). Their feature representation is based on Extended-connectivity fingerprints (ECFPs) that describe topological fingerprints about the environments surrounding each atom. These type of fingerprints are then used to train a number of ML models, where Wójcikowski et al. (2019) achieve best predictions results with an ANN, termed PLEC-nn. PLEC-nn achieves an R-score of 0.817 and 0.774 on the CASF-2016 and CASF-2013 scoring power benchmarks. Wójcikowski et al. (2019) state they PLEC was not tested and optimised for the CASF screening and docking powers and highlight that this will be part of their future work.

Another recent study is from Boyles et al. (2020). Their method does not include any deep learning methods however it aims to improve existing methods and we consider it an important recent development in scoring functions. Boyles et al. (2020) claim that

not enough ligand-based features are used in developing scoring function. Therefore, they investigated whether more detailed ligand features can improve binding affinity of the SF. Existing SF already include ligand features in their model such as Autodock Vina Trott and Olson (2010) (number of rotatable bonds) however these are limited to only a few features. Boyles et al. (2020) take a number of classical (Vina) and ML (NN-Score (Durrant and McCammon, 2010), RF-score (Ballester and Mitchell, 2010), RF-score v3 (Li et al., 2015)) models and added an RDKit feature set (Landrum, 2020), where the RDKit feature set is composed of around 200 1D and 2D ligand molecular descriptors features obtained from the RDKit Descriptors module. Boyles et al. (2020) show that they improved results consistently across all models tested achieving approximately an R value of 0.82 for the ML models and 0.792 for the Vina when tested using the PDBbind 2016 core set using 5-fold cross-validation (CV). Boyles et al. (2020) have tested the RDKit features (ligand-based features only) and report results for R similar to classical functions. The top ligand features that consistently improved results included the MollogP, TPSA, Estate-VSA1, and MolMR features.

It is also worth mentioning that Stepniewska-Dziubinska et al. (2017), and Zheng et al. (2019) do not mention the use of CV techniques since the PDBbind core set is predefined. On the other hand, Ragoza et al. (2017) have used 3-Fold CV for all their results. Additionally, test protein structures with a similar structure of 90% or more were grouped into the same fold to avoid test structures that are very similar to the those used in training. A similar approach was taken also by other researchers such as the work of Jiménez et al. (2018) and Boyles et al. (2020).

2.4.7 | Influence on LigityScore

A number of studies mentioned in this review were important for the development of LigityScore, and different techniques inspired various aspects of our model. The Pafnucy model was used as the benchmark for evaluation in our study and has motivated us to find an alternative representation for the protein-ligand complex based on its interacting and structural properties, and use the CNN models for automatic feature extraction. We have used their CNN model for the baseline parameters used in LigityScore. The Pafnucy model was chosen for the baseline as it is well documented and the authors provided the source code that enabled us to understand the details of their work to be able to replicate it for evaluation purposes and be confident in comparing like with like models.

Ligity was used as the basis of our study and has inspired us to select pharmacophoric features from both the protein and ligand that can be useful for protein-ligand binding

representation that is rotationally invariant. This was mainly by intuition as these were not tested for use as a scoring function based on CNN. However, these pharmacophoric features (discussed in depth in the Section 3.5), were in a way partly validated as some of them were also used by the Pafnucy and K_{deep} models. LigityScore method considers 6 pharmacophore features (hydrophoic, acceptor, donor, aromatic, cation, anion), which are all part of the K_{deep} pharmacophric features, and 4 (hydrophoic, acceptor, donor, aromatic) of them are also part of Pafnucy. Stepniewska-Dziubinska et al. (2017) showed that the hydrophobic, donor, and aromatic features ranked 2^{nd} to 4^{th} from Pafnucy's most influential features, as was discussed earlier. Additionally other methods such as OnionNet have inspired us to consider Pharmacophoric Interaction Points (PIPs) that are further apart, whilst using BatchNorm and ReLU to try and enhance our CNN models as done by Liu et al. (2019).

Machine learning model can be seen as black boxes and are sometimes more difficult to explain. Therefore a deeper understanding on the relationship created by these models is required (Li et al., 2019). Although the perfect protein-ligand representation and ML model for the development of a scoring function is still not known, we agree with Li et al. (2019) that with the increase in structural and interaction data, SF based on ML have the potential to direct the development of future SFs.

2.5 | Summary

In this section we have provided background information on the virtual screening domain focusing on the scoring function aspect, and have discussed the CNN architecture in detail together with its related concepts such as normalisation, weight intialisation, and optimisation techniques.

In the related work section we have given an overview on the history of scoring function starting with the classical methods, continuing with ML SFs models, and finally concluding with a comprehensive overview of the deep learning techniques used in scoring functions to date, and how this literature shaped our work. In the next section we explore in detail the components of the LigityScore methods.

3

Methodology

This chapter details the LigityScore implementation covering the methods and processes adopted to achieve the stated aims and objectives of Section 1.3. The following will be discussed:

1. The CNN architecture implemented showcasing the various components and tuning parameters
2. A detailed analysis of the datasets used
3. The algorithm used to generate the Pharmacophoric Interaction Points (PIPs) from the protein and ligand files
4. The algorithm used to extract LigityScore1D and LigityScore3D CNN input features
5. A description of the experiments workflow carried out to develop and validate LigityScore, together with the software and hardware used to run the experiments.

3.1 | LigityScore Implementation Overview

The process to develop the CNN based scoring function is illustrated in Figure 3.1, highlighting also the paramaters that can be changed for each module. The major functional parts for LigityScore1D and LigityScore3D are detailed below:

1. **Pre-Processing.** The process starts with a pre-processing stage where the PDBbind files are processed to build a dataset of the complexes available in PDBbind with their respective binding affinity properties. At this stage the molecular files are

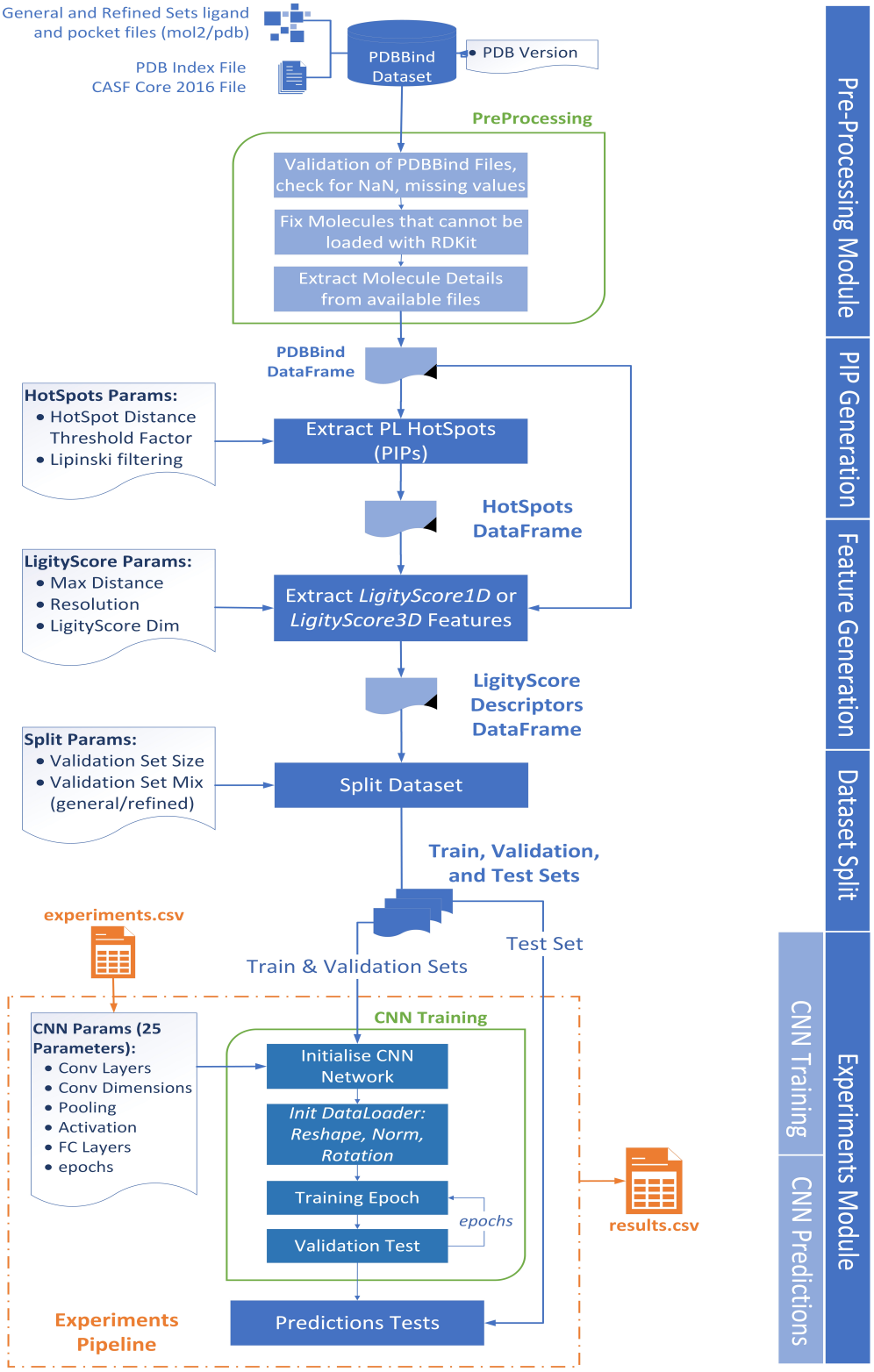


Figure 3.1: The schematic represents the major functional blocks used in our approach to develop a Scoring Function solution for VS. The parameters used in each functional block are included included as reference. For example PIP Hot-spots extraction can take two parameters — Lipinski filtering, and the hot-spots Distance threshold factor.

validated to check that the PL complexes listed have corresponding molecular files, and also to fix any errors that occur whilst loading the files using the RDKit cheminformatics library (Landrum, 2020). This is an important step as LigityScore depends on the correct loading of molecular files in RDKit in order to extract and generate the LigityScore features. If any errors are found, the molecule will not be loaded and the complex would have to be discarded from the dataset. The process taken to resolve this issue, and the total number of complexes discarded are detailed in Section 3.3.

2. **PIP (Hot-spots) Generation.** This module loads the complexes and searches for the pharmacophoric features. All the possible pairs of pharmacophoric features across the protein pocket and the ligand are built, and are then run against a number of constraints. The resultant feature pairs represent the PIPs or interaction hot-spots for the particular complex. The hotspot details are saved to the hot-spots dataset for use by the subsequent task. The constraints used for PIP generation method are described in Ebejer et al. (2019).

3. **Generation of LigityScore Descriptors.** The LigityScore Descriptors module utilises the hot-spots dataset to generate a feature descriptor for each complex. For each complex a feature matrix is generated for LigityScore1D, whilst a feature cuboid is generated for LigityScore3D. Both these feature descriptors are depicted in Figure 3.9 and Figure 3.10 respectively. The name of our models is derived from the dimensionality of the spatial information used to generate the features. LigityScore1D considers two hot-spots at a time that correspond to a particular pharmacophric feature family pair (example Acceptor-Acceptor). For each possible family pair a feature vector is constructed, hence the name LigityScore1D. On the other hand, LigityScore3D uses three hot-spots at a time, and the spatial information for the family set (example Acceptor-Acceptor-Acceptor) is encoded in a feature cube. Generation of the LigityScore descriptors is discussed in depth in Section 3.6.

4. **CNN Training.** This module is mainly built using the Pytorch library (Paszke et al., 2019) and includes a dynamic model to construct a CNN. This dynamic model was constructed in order to facilitate the testing and evaluation of different CNN architectures so that it can be invoked by the 'Experiments Pipeline' highlighted below. The CNN module is used to train the network as a scoring function using the LigityScore descriptors, highlighted previously, as input. The module tackles a regression type of problem and therefore the output is a continuous value

predicting the binding affinity of the complex. This output is compared with the real binding affinity so that the network parameters, or weights, are updated using stochastic gradient descent. Each epoch is validated against the validation set, composed of 1,000 randomly sampled complexes from the PDBbind Refined set, and the model with the lowest root means squared error (RMSE) is stored to disk for use for predicting unseen complexes.

5. **CNN Predictions.** The Predictions module is used to load the best performing model and to compute results for the Test Set. The training and validation sets are also re-evaluated in order to provide RMSE, MAE (Mean Absolute Error), SD (Standard Deviation in Regression), and R (Pearson Correlation Coefficient) values for each dataset as results for the evaluation criteria to assess the performance of the model.

6. **Experiments Pipeline.** This module combines the Training, Validation and Testing in a one pipeline. This step uses a CSV file to describe a series of experiments with different CNN paramaters so that training and testing of the model is done through one pipeline. The CSV file includes parameters such as number of convolution layers, layer filter dimensions, kernel size, training epochs, number of fully connected (FC) layers, and size of each FC layer. A total of 39 parameters are required to initialise one experiment. For each experiment the best epoch is chosen based on the lowest RMSE for the validation set, and the results for the training, validation, and test sets are evaluated using the CNN model parameters for the best epoch. The result for each experiment is then stored to disk as another CSV file.

All the modules listed were developed using Python 3.6 using Ubuntu 18.04 Server. All processes were run on AWS EC2 instances. The feature extractions modules were run using 'r5' memory optimised machines, whilst all CNN training and predictions were run on 'g4dn' GPU instances consuming around 2,400 US dollars. Each of the modules listed above are detailed in the next sections.

3.2 | Baseline for the Study

One of the first steps used to develop LigityScore, was to replicate the work done in Pafnucy (Stepniewska-Dziubinska et al., 2017), as a way for comparison and evaluation which are cornerstones for scientific investigation. Their method was recreated using our own code but using the Pafnucy feature library, tfbio (Stepniewska-Dziubinska,

2020), to generate the PL complex feature representation. The same CNN model parameters were kept. The aim was to achieve similar performance to the results quoted in Stepniewska-Dziubinska et al. (2017). The replication model was used to calculate the Pearson correlation coefficient, R, and the standard deviation (SD) in linear regression as was detailed in Section 2.3. The results of the baseline work we compare our results to are presented in Section 4.2. Some of the modules used to build our implementation of Pafnucy, such as the preprocessing module and the CNN training and prediction modules, were later on used for the development of LigityScore. Correct replication results would confirm the correct functionality of these modules, validating also that a good baseline is available to develop our scoring function models.

Our replication model included a number of differences which might explain the slight difference ($\sim$ 2.5%) in our replication results. These include:

1. **Computation of Charges**. Stepniewska-Dziubinska et al. (2017) have used the Chimera (Pettersen et al., 2004) toolkit to compute the partial charges for the protein pockets. In our replication approach we have chosen to use the OpenBabel (O'Boyle et al., 2011) toolkit as it is available as a python package and is easier to integrate into our code. Therefore, a difference in how the two toolkit compute these charges can also affect the training and prediction results of our version of the Pafnucy model.

2. **Machine Learning Library**. In our replication model we have used the PyTorch ML library (Paszke et al., 2019) instead of the TensorFlow (Abadi et al., 2015) library used by Stepniewska-Dziubinska et al. (2017).

3.3 | Dataset

The PDBBind dataset (Liu et al., 2017b) includes records of experimentally measured binding affinity data for biomolecular complexes taken from the Protein Data Bank (PDB) (Berman et al., 2003) using only the original references directly (Liu et al., 2017a). The first version of the PDBbind was released way back in 2004 and has since then been updated on a yearly basis. The PDBbind dataset includes four types molecular complexes including protein-ligand complexes, nucleic acid-ligand complexes, protein-protein complexes, and protein-nucleic acid complexes. For the purposes of this study only the protein-ligand complexes are relevant for use in drug design and the development of the scoring function. An overview of how the dataset evolved since it was taken over by the Shanghai Institute of Organic Chemistry is highlighted in Figure 3.2

(Liu et al., 2017a). Over the years, a number of publications were released covering the PDBbind dataset (Li et al., 2014a,b; Liu et al., 2015, 2017a), and is regarded as a golden dataset for the development of scoring functions (Liu et al., 2017a). The PDBbind is compiled based on the contents available from Protein Data Bank (PDB) as at the first week of the year.

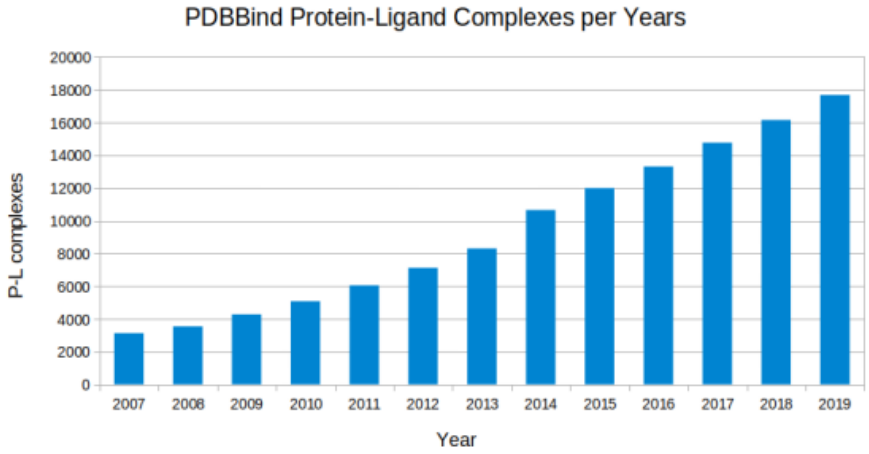


Figure 3.2: Number of protein-ligand complexes included in the PDBBind Dataset over the years maintained by the Shanghai Institute of Organic Chemistry, showing steady improvements in the dataset. The dataset is accessible from `http://www.pdbbind-cn.org`.

The PDBbind v2016 dataset is used for training, validation, and testing of the developed scoring functions primarily as a means to evaluate our models with other research that use the same dataset version. However, our models are also trained and tested against PDBBind v2018 in order to facilitate comparison of our models with future work. The PDBbind dataset is composed of two main components:

- **The General and Refined Sets**. These sets contain different groups of protein-ligand complexes. Each ligand file is available in *mol2* and *sdf* format variants, whilst the protein files are available in *mol2* and *pdb* formats.
- **Index Files**. A set of *INDEX* files document the details of each PL complex. INDEX files refer to each complex name with a PDB Code which is the same as the protein structure name (example 3ZZF). For each complex the files contain details of the ligand name, release year, resolution, and binding affinity data.

The protein-ligand complexes are grouped into different sets and are illustrated in Figure 3.3, whilst Table 3.1 provides a summary of the group sizes. The PDBbind dataset

consists of the following groups:

1. **The General Set.** The general sets represents all the available protein-ligand complexes available in the database.

2. **The Refined Set.** The refined set is a subset of the general set as shown in Figure 3.3. The refined set was constructed after quality controls on structural resolution and experimental precision of the binding data measurement.

3. **The Core Set.** This set in reality is not considered as part of the PDBbind dataset. The core set was established as part of the Comparative Assessment of Scoring Functions (CASF) benchmark and was created as a separate study by the same authors (Li et al., 2018; Su et al., 2018) and is not a direct input in the PDBbind dataset. The CASF benchmark is meant to provide an objective platform to assess scoring functions, using high-quality protein-ligand complexes selected from the refined set, through a systematic, non-rendundant sampling procedure. To filter the complexes the refined set was clustered using a protein sequence similarity algorithm where each cluster must have at least 90% similarity. Five complexes from each cluster were then selected including those with lowest and highest binding affinty (Su et al., 2018). These high-quality complexes are used as the primary test set in the CASF benchmark and were labeled as the PDBbind *core set* by the authors. The core set is not updated on an annual basis as this requires different preparation, and evaluations from the actual PDBbind dataset. The CASF benchmark is discussed in detail in Section 2.3.1. The most recent core sets include the Core Set v2013 (Li et al., 2018), and the Core Set v2016 (Su et al., 2018). The core set v2016 will be used as the main test set for this dissertation as this provides a way to evaluate LigityScore with other research such as Stepniewska-Dziubinska et al. (2017) and Jiménez et al. (2018), and also with other models that are evaluated as part of the CASF benchmark. For comparative reasons the core v2013 set will be also be used for testing as was used by Stepniewska-Dziubinska et al. (2017) and Zheng et al Zheng et al. (2019).

A summary of the PDBBind dataset is listed in Table 3.1 as provided by Liu et al. (2017b).

The PDBbind dataset includes experimentally validated binding affinity values in terms of a dissociation (K_d), inhibition (K_i) or half-concentration (IC_{50}) constant for all types of biomolecular complexes. These are different experimental measures which define how strongly a ligand binds to a protein. It is a common approach when training

PDB Version	Total Files	General Set	Refined Set	Core2016 Set	Core2013 Set
2016	13,308	9,246	3,689	285	195*
2018	16,151	11,657	4,121	285	195**

** 107 complexes overlap the Core2016 set, whilst 83 are from the Refined Set and 5 from the General Set.*
** 107 complexes overlap the Core2016 set, whilst 76 are from the Refined Set and 12 from the General Set.*

Table 3.1: PDBBind summary showing total number of available protein-ligand complexes. The Core Set is defined as part of the Su et al. (2018), and are a subgroup of the Refined Set in PBDbindLiu et al. (2017b).

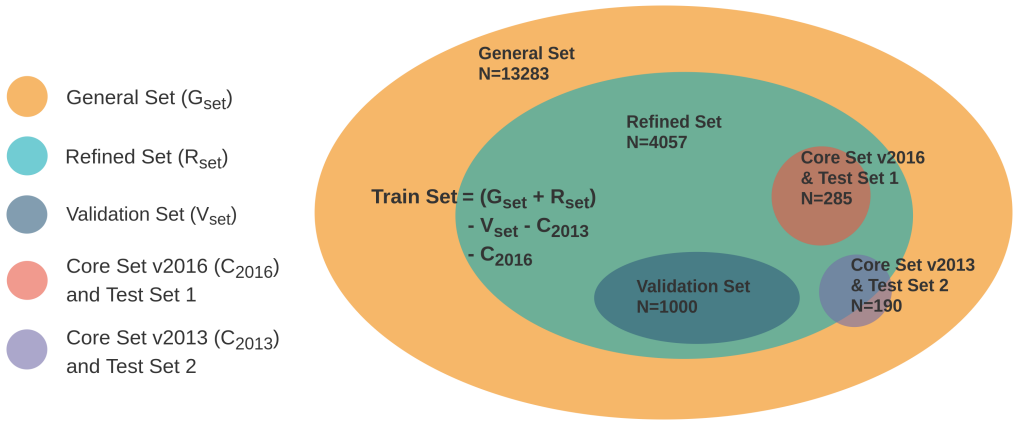


Figure 3.3: PDBBind venn-diagram showing the general, refined and cores sets, together with how these were split up for training, validation and training. The validation set is made up of 1,000 complexes from the refined set, whilst the core sets are the test sets. The remaining complexes are used for training. N represents the count of protein-ligand complexes in each set.

ML models on binding affinity values that no distinction is made between K_d, K_i, and IC_{50}. These constants are also converted into a negative log as shown in Equation 3.1.

$$pK_a = -log_{10}K_x \tag{3.1}$$

where K_x can be K_i, K_d or IC_{50}, and pK_a is the binding affinity. This approach was taken by various researchers such as Stepniewska-Dziubinska et al. (2017) and Jiménez et al. (2018), and is also the approach used in this dissertation.

3.3.1 | Handling Incorrect Molecule Files

Some of the molecular files in the dataset do not load correctly within RDKit, which means that they would have to be excluded from training. The errors are generated due to incorrect mapping of atomic bonds that are not possible. RDKit flags these incorrect

mappings and treats the molecule as invalid. The ligand files generate most of the errors with 1,603 invalid ligand molecules for the PDBbind v2016, and 1,839 for the PDBbind v2018. On the other hand the protein files only had 22 invalid files for PDBbind v2016 and 44 for PDBbind v2018. In order to fix the ligand molecular files the following steps are taken:

1. The format that gave least errors was the .mol2 format. Therefore the first approach was to try to load these molecules in .mol2 format.

2. If mol2 loading is not successful, the .sdf format is then tried. The sdf ligand file is also part of PDBbind.

3. If the sdf file is also not successful, the ligand file is *fetched* from Pymol. The *fetch* utility in PyMol (DeLano, 2002) downloads the molecule directly from the PDB repository.

4. If the Pymol fetch is unsuccessful, the complex is discarded.

5. The fetched molecule is checked to verify it can be loaded. If not, complex is discarded.

The molecules that were fixed or discarded from the dataset using the above mentioned procedure are detailed in Table 3.2. The total usable complexes are also listed after removing the total discarded complexes. The 'No Files' column indicated the molecules that were listed in the INDEX files but did not have corresponding molecular files.

PDB Ver	Total Mol	No Files	mol2 error	sdf fix	sdf pymol	discard	usable	General Set	Refined Set	Core Set
2016	13,308	25	1,603	1,142	289	172	13,111	9065	3763	283
2018	16,151	25	1,839	1,317	315	207	15,919	11446	4188	285

Table 3.2: PDBbind summary showing the total number of complexes available, how many molecular files were absent, errors obtained when loading mol2 files in RDkit, the errors fixed using the sdf format, and also the errors fixes using pymol. The total usable complexes are listed after discarding absent and error molecules, and are categorised by PDBbind set.

3.4 | Pre-Processing Module

The preprocessing module highlighted in Figure 3.1 is used to process, extract and clean the required information from the PDBbind dataset. The following aspects are tackled in the `pdb-file-preprocess.py` script of the preprocessing module.

- **Creation of *PDBbind DataFrame*.** A *Pandas* DataFrame is built to collect details of each individual protein in the PDBBind dataset. The column features include the protein and ligand labels, release year, experimental binding affinity, the file paths corresponding to the molecular files, and the PDBBind set (general, refined, core) of the respective protein-ligand complex. The label of the protein is also used as the PDBCode to identify the record. Additional fields are added to this PDBBind DataFrame by other modules such as:
  - The dataset split script (`pdb-split-dataset.py`) adds a feature to mark if the PL complex should be in the train, validation, or test sets.
  - The PL complex representation (`pdb-LigFeatures.py`) adds the LigityScore features for each PDBCode.
  - The prediction components (`pdb-predictions.py`) adds the predicted binding affinity for each PDBCode.
- **Validation that the listed molecular files are available.** As listed in Table 3.2 there are 25 protein-ligand complexes that are included in the PDBBind INDEX files but do not have corresponding molecular files. These complexes are excluded from the dataset.
- **Fix molecular files that cannot be loaded using RDKit**. Molecules that cannot be loaded using RDKit are fixed in this module as is described in 3.3.1.
- **Validation of compiled dataset.** This consists of other generic steps to make sure there are no errors in the data, such as checks for any duplicate PDB Codes, and checks for any missing values.

Preprocessing of the molecular files such as removal of water molecule, addition of hydrogens, and computation of charges, is not required for the LigityScore methods. The PDBBind protein pocket files do not include any water molecules. Therefore, these molecular processing steps are not included in this module. It is important to highlight that at this stage the actual molecular feature representation is not yet available since this is created in subsequent steps. Therefore, additional pre-processing steps are performed

at later stages on the actual feature representation of the complex before being input to the CNN.

3.4.1 | Dataset Split

The *PDBbind DataFrame* is split into four sets. These four sets and their relationship with the PDBbind sets are illustrated in Figure 3.3. A validation set was selected to evaluate the training progress after each epoch, and select the CNN model with the smallest error to be used for the binding affinity predictions. The validation set was also used for Early Stopping functionality in the training module so that training is stopped after a number of epochs with no loss improvements. The validation set contains 1,000 randomly chosen complexes from the refined set. These are chosen entirely from the refined set, as these provide higher quality protein-ligand complexes and are more reliable for the development of scoring functions (Liu et al., 2017a). This approach was also taken by a number of researchers including Stepniewska-Dziubinska et al. (2017) and Zheng et al. (2019).

Each of the core sets (2013 and 2016) were used entirely as the two test set. Each protein-ligand complex in these sets does not belong to either the training or validation sets, to simulate new and unseen protein-ligand complexes during the prediction stage. For the validation set, 1,000 protein-ligand complexes were randomly selected as was done in the work of Stepniewska-Dziubinska et al. (2017), Zheng et al. (2019), and Zhang et al. (2019). The remainder of the protein-ligand complexes were used as the training set. This includes all the PDBbind general set and the remainder of the refined set, but excluding complexes that are also part of the Core-2013 set.

Stepniewska-Dziubinska et al. (2017) and Zheng et al. (2019) do not perform any cross validation whilst training, due to the dataset splitting approach used where the Test set is bound only to the core set. Testing a scoring function using the core set as a test set is a common approach to evaluate scoring function, as this is also used to assess the *Scoring Power* in the CASF benchmark (Su et al., 2018). Therefore for the same reasons, and for the fact that the CNN training requires several hours of training, cross validation was similarly not used in our approach.

The PDBbind dataset includes molecular complexes from a number of protein families. In our study, training was not performed on individual families but all the families were contributing to the same DL model. This generic model approach is common in ML based SF and is used by a number of researcher Jiménez et al. (2018); Öztürk et al. (2018); Ragoza et al. (2017); Stepniewska-Dziubinska et al. (2017); Zheng et al. (2019).

3.5 | PIP Generation

The protein-ligand features representation for the CNN network is split into two phases. In the first phase the PIPs of each individual protein-ligand complex are extracted to create the *PIP dataset* using all the PDBBind complexes. The algorithm used for PIP generation is based on the Ligity methodology described in Ebejer et al. (2019) and is detailed in Figure 3.4. The PIP generation algorithm can be executed using `pdb-LigHotspots.py` python script.

As a first step the molecules are optionally filtered against the Lipinski Rule of Five (Lipinski et al., 1997). These rules are listed in Table 3.3 and define an approach that suggests whether or not the molecule is likely to be poorly absorbed. When the molecular properties exceed the listed thresholds, the molecule is regarded as non drug-like as it can lead to poor absorption in the body. The distribution of the ligands in PDBBind prior to the Lipinski filtering is shown in Figure 3.5. The filtering of the dataset based on Lipinski rules is an optional paramater for the PIP generation script, and is used as one of the experiments described in Section 3.8 to determine if the "drug-likeness" property of molecules affects the SF prediction ability.

Molecular Property	**Lipinski's Property Threshold**
Molecular Weight (MolW)	500
logP	5
Hydrogen Bond Donors	5
Hydrogen Bond Acceptors	10

Table 3.3: Lipinski Rule of Five used to filter drug-like molecules. Molecules with properties larger than these threshold are considered non drug-like.

The PIPs are then extracted from the query protein-ligand complex using the open-source cheminformatics package *RDKit BuildFeatureFactory* class (Landrum, 2020). The BuildFeatureFactory uses SMARTS patterns to identify these pharmacophoric features within the molecule that include hydrophobic, acceptor, donor, cation, anion, and aromatic family-type features. SMARTS is a type of language, similar to regex for molecules, that describes molecular patterns and is used to search for substructures within the molecule such as aromatic or hydrophobic structures. Each BuildFeatureFactory stores a number of properties of the *feature* including the family, atoms, and coordinates of the particular feature which are then used to filter and identify the suitable PIPs to include in the dataset.

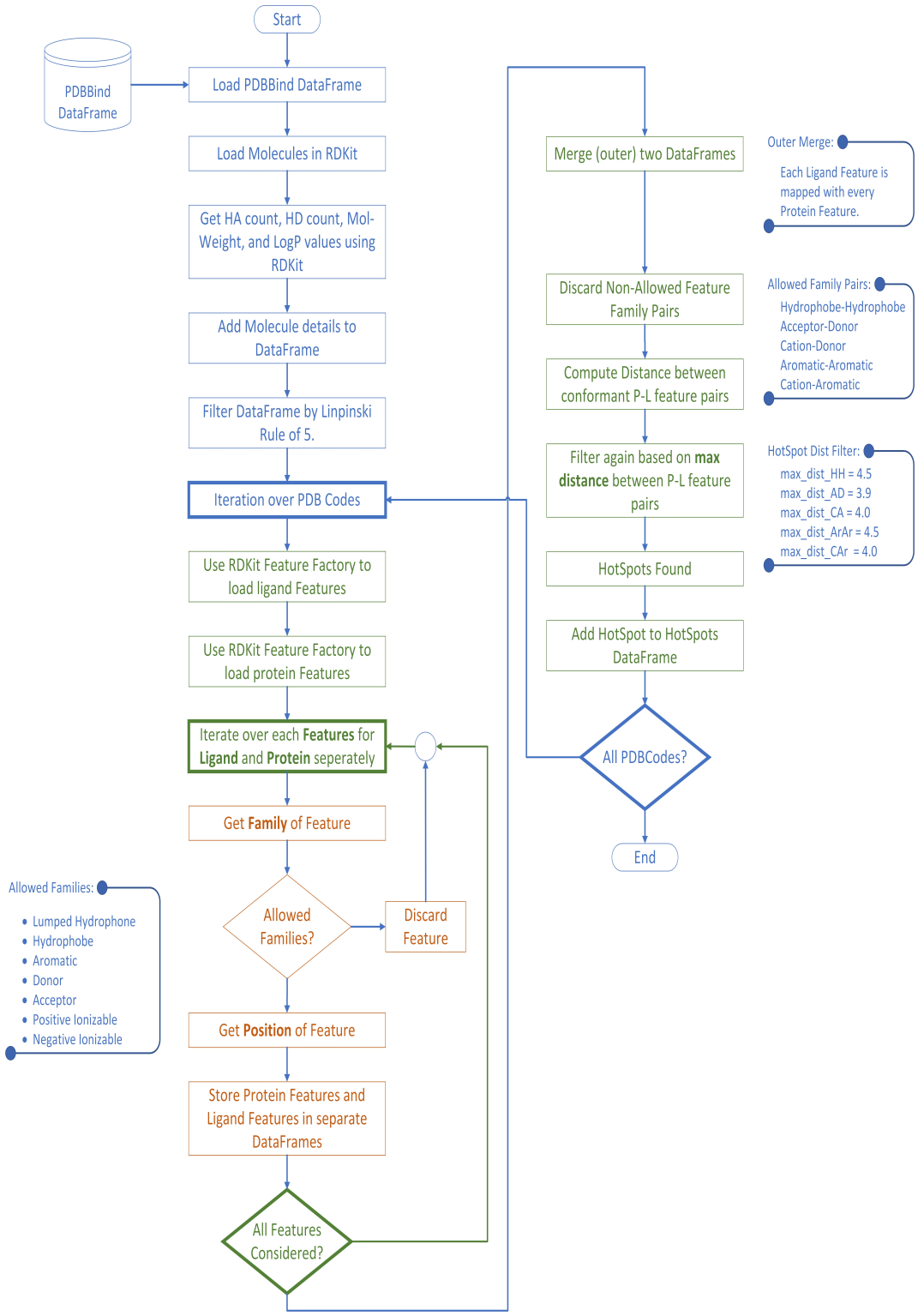


Figure 3.4: PDBbind dataset Hot-Spots or PIP generation algorithm.

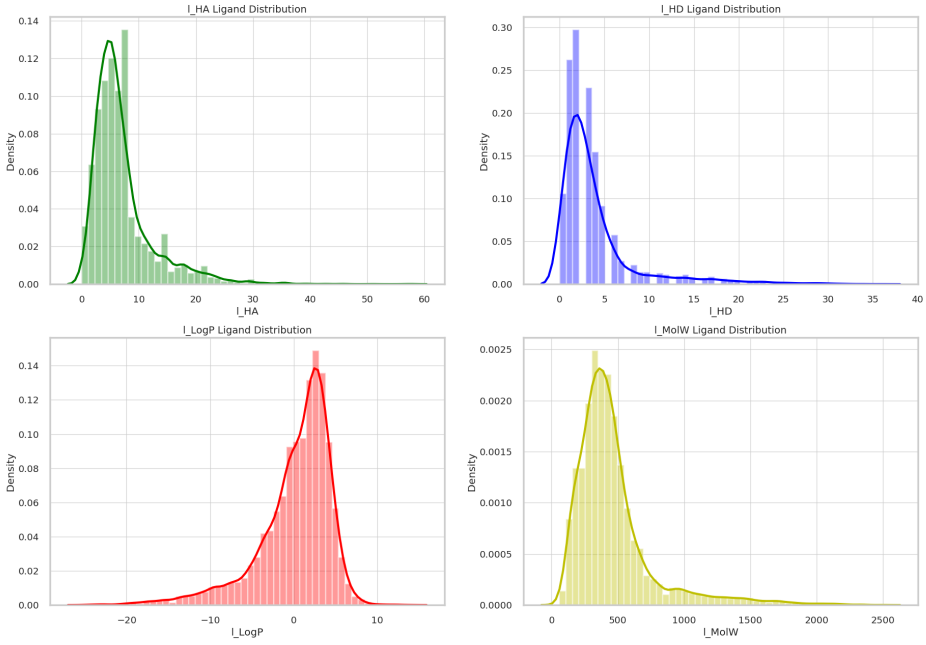


Figure 3.5: PDBBind molecular properties distributions including (a) Hydrogen Bond Acceptors, (b) Hyrdogen Bond donors, (c) logP and (d) molecular weight.

These PIPs, of both the protein and ligand, are then filtered by a set of rules constraining feature family-pairs at a specific distance threshold as indicated by Ebejer et al. (2019) so as to capture only the stronger interactions between the protein's and ligand's pharmacophoric features. For an acceptor-donor PIP pair to be included in the PIP dataset, the hydrogen bond acceptor feature in the ligand requires a corresponding hydrogen bond donor at the protein side, or vice-versa, that has a distance which is less than 4.5Å. The allowed feature family pairs and their corresponding distance threshold are listed in Table 3.4. It is important to note that only these molecular interactions are used to build the PIP dataset. The euclidean distances between the features are calculated using the centre of the atoms making up the feature. For example in an aromatic PIP, only the centre of the atomic structure is considered. Figure 3.6 shows an example of the PIPs for the 3ZZF target and the NLD ligand, where each PIP is represented by a seperate mesh. In order to extract all conformant PIPs from the protein-ligand complex a cartesian product of all PIPs from the protein and ligand is performed, followed by the filtering of the allowed family pairs, and further filtering by the maximum distances allowed. PIP interactions are illustrated in Figure 3.7 showing the calculated distances

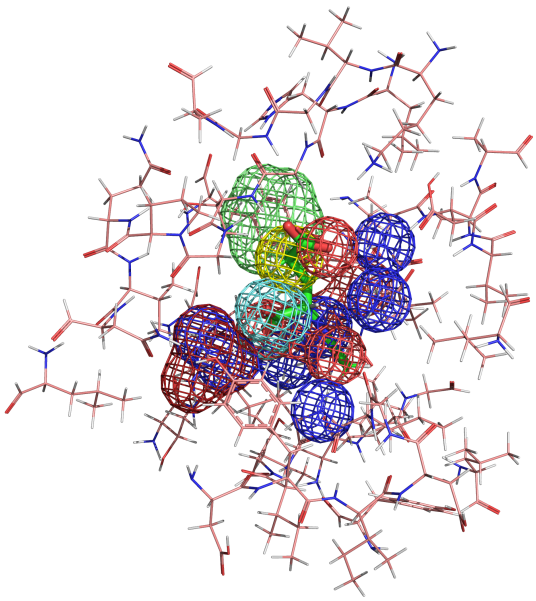


Figure 3.6: Examples of PIPs on the 3ZZF target with NLD ligand. Each coloured mesh represents a different PIP pharmacophore family type. For example the blue mesh represents *Donor* atoms, whilst the red mesh represent *Acceptor* atoms

between centres of PIPs.

In our approach we have also considered using larger distance thresholds than those stated in Table 3.4. The `pdb-LigHotspots.py` program provides a *threshold-factor* argument that can be used to multiply this baseline distance. As an example a threshold-factor of 1.5 would increase the maximum distance by half. A number of experiments were carried out using various distance threshold-factors between 1.0 and 1.6 in order to capture additional PIP interactions in our feature representation. This additional information includes also other weaker interactions, since the protein and ligand features are further apart and thus have a weaker charge attraction, which leads to a more information rich representation of the protein-ligand complex. Considering a larger distance threshold factor implies that more PIPs are selected which in turn leads to a larger number of combinations that would be considered during the feature generation process. Therefore, this contributes to a more information rich representation of the protein-ligand complex. The idea of using larger distance threshold factors was motivated from the work of Zheng et al. (2019) where they showed that long-range interactions outside the 20 angstrom range, termed *non-local* electrostatic interactions are also important for the protein-ligand binding. To capture all the interactions between

protein-ligand complexes Zheng et al. (2019) divide all the 3D space of the binding site into a number of shells or zones, and count the number of different element to element interactions within each shell. To prove their results, Zheng et al. (2019) compared the loss of the model when a particular shell is removed to the loss of the best performing model. Their results show that these non-local interactions have significant impact on the loss of their model and claim their contribution is important for binding affinity prediction. Our experiment using different distance threshold-factors are discussed in Section 3.8, whilst the next section tackles the second phase of the LigityScore protein-ligand feature representation process.

Interacting Protein-Ligand PIP Family Pairs	**Distance Threshold (Å)**
hydrophobic, hydrophobic	4.5
acceptor, donor	3.9
cation, anion	4.0
aromatic, aromatic	4.5
cation, aromatic	4.0

Table 3.4: Pharmacophoric Features and Distance thresholds used to build the PIP dataset reproduced from Ebejer et al. (2019).

3.6 | Protein-Ligand Complex Representation

The second phase of the protein-ligand complex representation uses the *PIP dataset* described in the previous section to create a feature matrix, or a feature cube collection for every complex for LigityScore1D or LigityScore3D respectively. The algorithm used to generate these features is illustrated in detail in Figure 3.8. The LigityScore methods were inspired from the Ligity Ebejer et al. (2019) method mainly due to its rotationally invariant nature. LigityScore has four main differences from the Ligity implementation:

1. LigityScore uses PIPs from both the protein and ligand sides for feature generation, to capture additional information from the 3D protein structure.

2. LigityScore considers PIPs that are further apart since these interactions have also a significant contribution to binding affinity prediction models as demonstrated by Zheng et al. (2019).

3. LigityScore is a scoring function for binding affinity prediction function, whereas Ligity is a virtual screening technique.

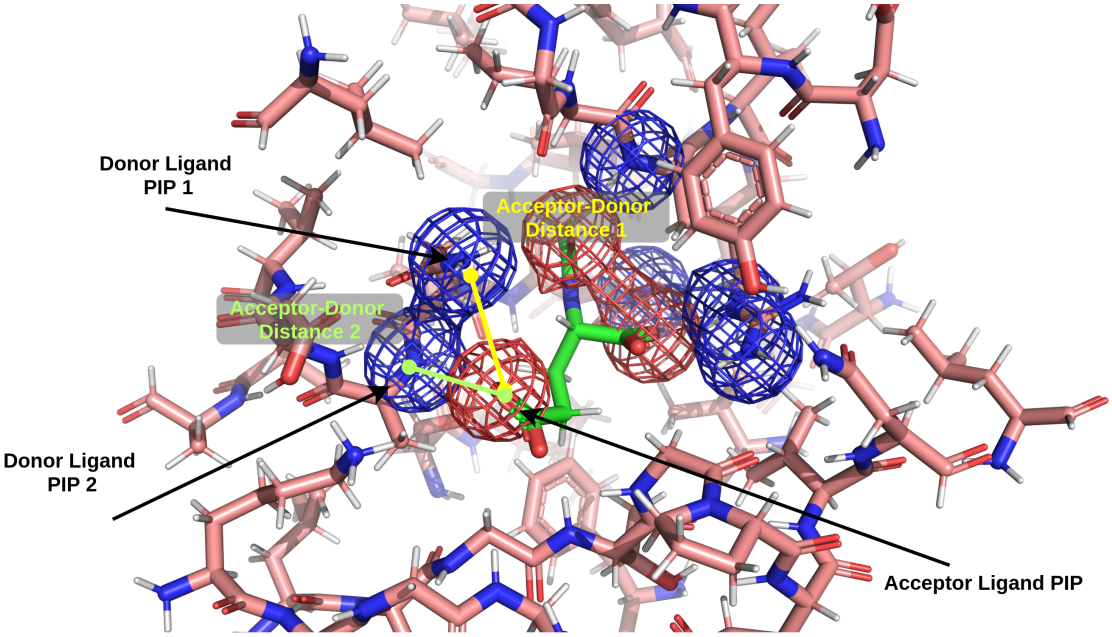


Figure 3.7: PIP pair interaction between a donor protein PIPs (blue mesh), and an acceptor ligand PIP (red mesh). The ligand acceptor interactions with all the protein PIPs are considered during PIP generation however only two are shown here for better visualisation. For each interaction the distance between the centres of the PIPs is calculated.

4. LigityScore uses a CNN to automatically extract features and perform binding affinity prediction instead of similarity based methods.

Both LigityScore methods use the same algorithm using the `pdb-LigFeatures.py` python script, however there are some slight differences in the two approaches and therefore they are described in the next sections separately. Through the LigityScore models we have generated the required protein-ligand representation for use with CNNs and thus achieved the objective outlined in Section 1.3.

3.6.1 | LigityScore1D

Each feature matrix is calculated using the PIPs from the *PIP-dataset* related to the particular protein-ligand complex. The PIPs for the ligand side and those of the protein side are extracted to obtain two seperate sets — the ligand PIP set, and the protein PIP set.

The feature matrix is constructed by considering all the possible combinations when choosing one PIP from the ligand-PIP set, and one PIP from the protein-PIP set. The distance between each combination is discretised using 1Å resolutions. These PIP com-

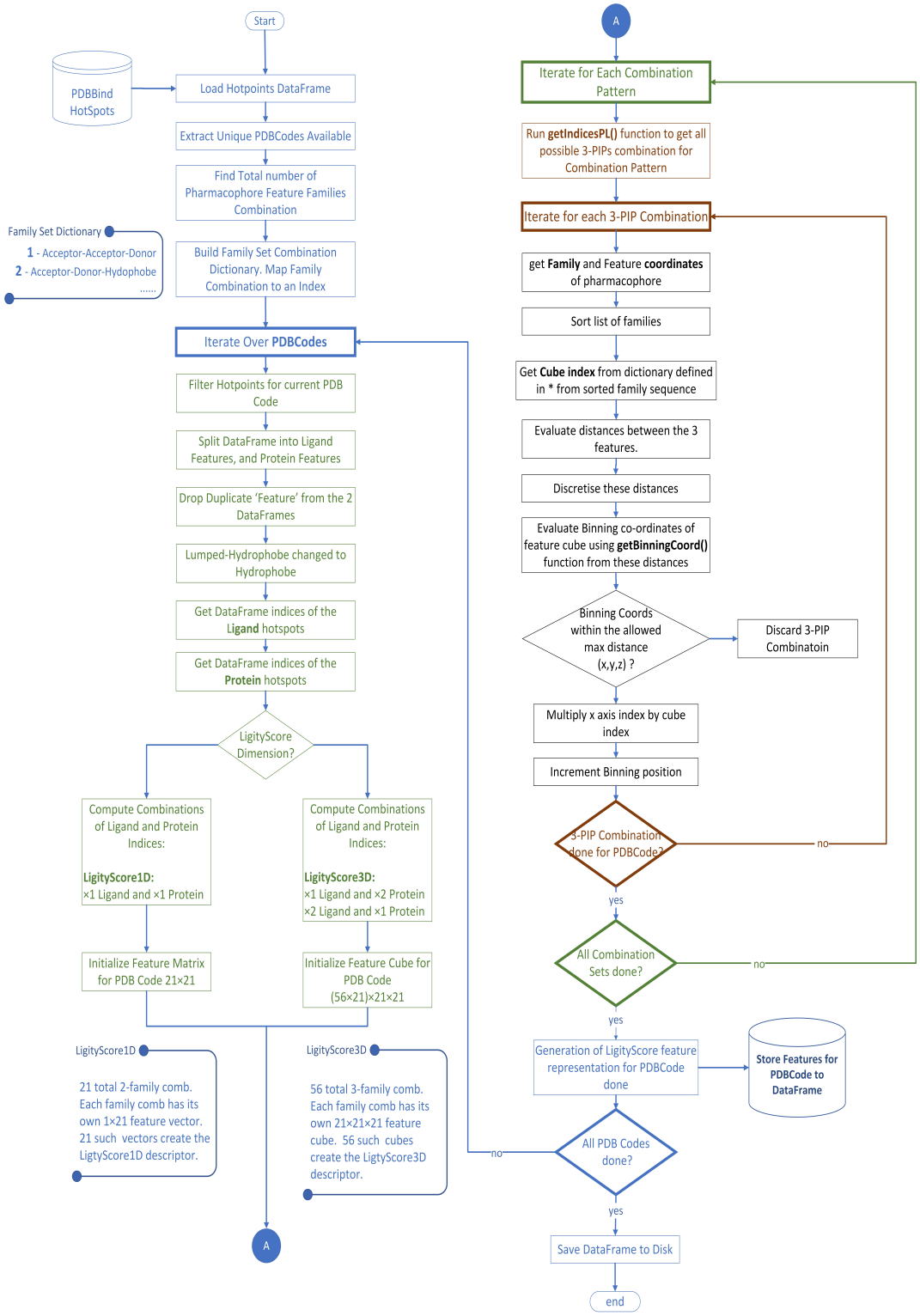
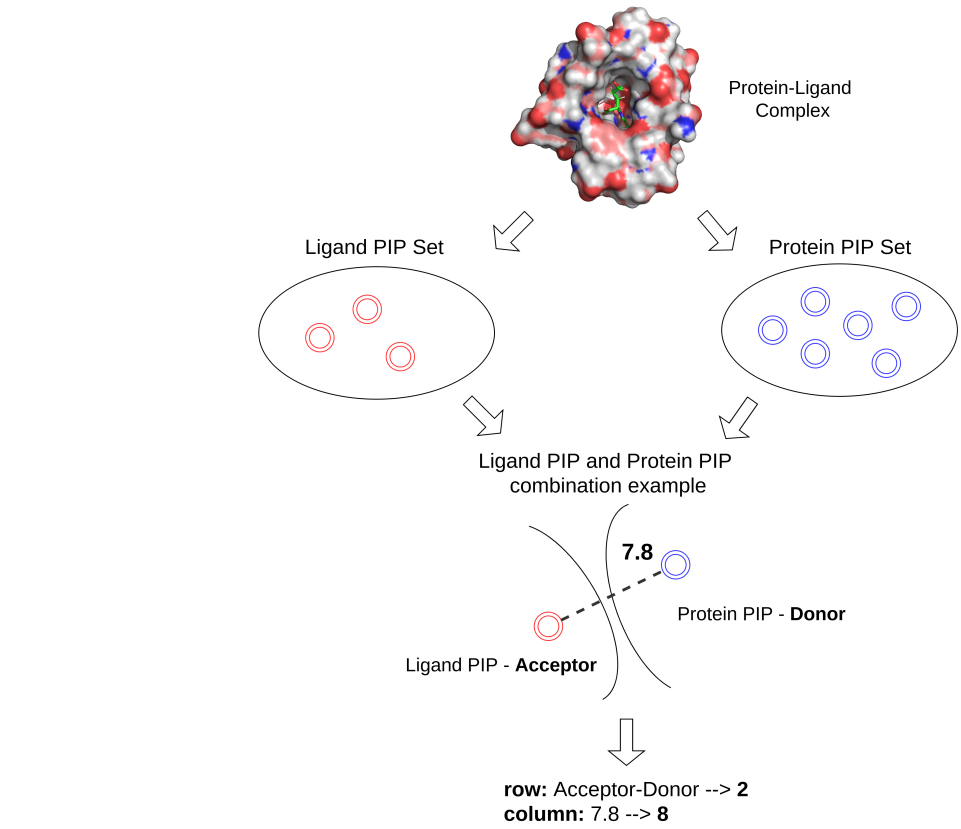


Figure 3.8: Algorithm used for the Feature Generation Process for LigityScore1D and LigityScore3D.



Feature Matrix — **Discrete Distance**

PIP Family Pair		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(Acceptor, Acceptor)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(Acceptor, Aromatic)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(Acceptor, Donor)	2	0	0	0	0	0	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0	0
⋮																						
(PosInoizable, PosIonizable)	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.9: LigityScore1D Feature Matrix Generation. A protein-PIP set and a ligand-PIP set are extracted from a PL complex. From each PIP combination taken from the PIP pool, the family pair, and their discrete distance are used to update the binning count in the feature matrix. The rows of the feature matrix represents the allowed family-pairs, whilst the columns represent the discrete spaces available to bin the PIP combination distance.

binations also represents a particular PIP family set (example Acceptor-Donor). Each PIP family pair represents a different row in the feature matrix. Therefore the PIP-family pair is used to index the row of the feature matrix, whilst the discretised distance is used to index the column of the PIP. These coordinates in the feature matrix are then used to increment the bin count of that location. This process is illustrated in Figure 3.9. In this example an *Acceptor* ligand-PIP is selected, whilst a Donor protein-PIP is selected from the protein-PIP set. A distance of 3.7Å is discretised to position 4. In this dummy example the Acceptor-Donor family pair represents row 1. Therefore in this example location (1,4) is incremented by 1. If the discrete distance exceeds the allowed max distance the PIP combination is discarded. The feature matrix is initialised to all 0's. All the possible combinations are iterated so that each PIP pair distance increments the bin position in the feature matrix space.

The rules in Table 3.4 defined in Ebejer et al. (2019) allow six possible PIP families. This corresponds to a total of 21 possible family-family combinations when selecting two families from six with replacement. A total of 21 possible discrete locations are available when considering a maximum distance of 20Å, and a 1Å resolution. Therefore each feature matrix has a size of (21×21). Each PIP family combination corresponds to a row vector that represents the discrete distribution from a single distance of the 2-PIP combination, and hence the name LigityScore1D was chosen for this representation.

3.6.2 | LigityScore3D

The method used LigityScore3D is similar to LigityScore1D but is more complex as it considers a combinations of 3-PIPs at a time. The 3-PIP combination creates a triangular structure amongst the PIP as shown in Figure 3.10 a generates a set of three distances. The three distances are discretised as is done in LigityScore1D approach to extract a binning coordinate in 3D space. The voxel, or bin at this location is incremented by one. Additionally, in this case the 3-PIP family combination represents a unique feature cube. Taking three out of six families with replacement creates a total of 56 possible 3-family set combinations. The unique family set combination is used to index the particular feature cube in the feature cube collection to update the binning count using the coordinates from the discrete distances. Considering the example in Figure 3.10, one ligand PIP and two protein are considered. These generate a PIP-family combination of *Acceptor-Acceptor-Donor*, so the *Acceptor-Acceptor-Donor* will be updated at the (10, 8, 3) voxel location.

Considering a maximum distance of 20Å in each dimension, each feature cube has a dimension of $(21 \times 21 \times 21)$. Since each 3-PIP family set has its own feature cube, 56

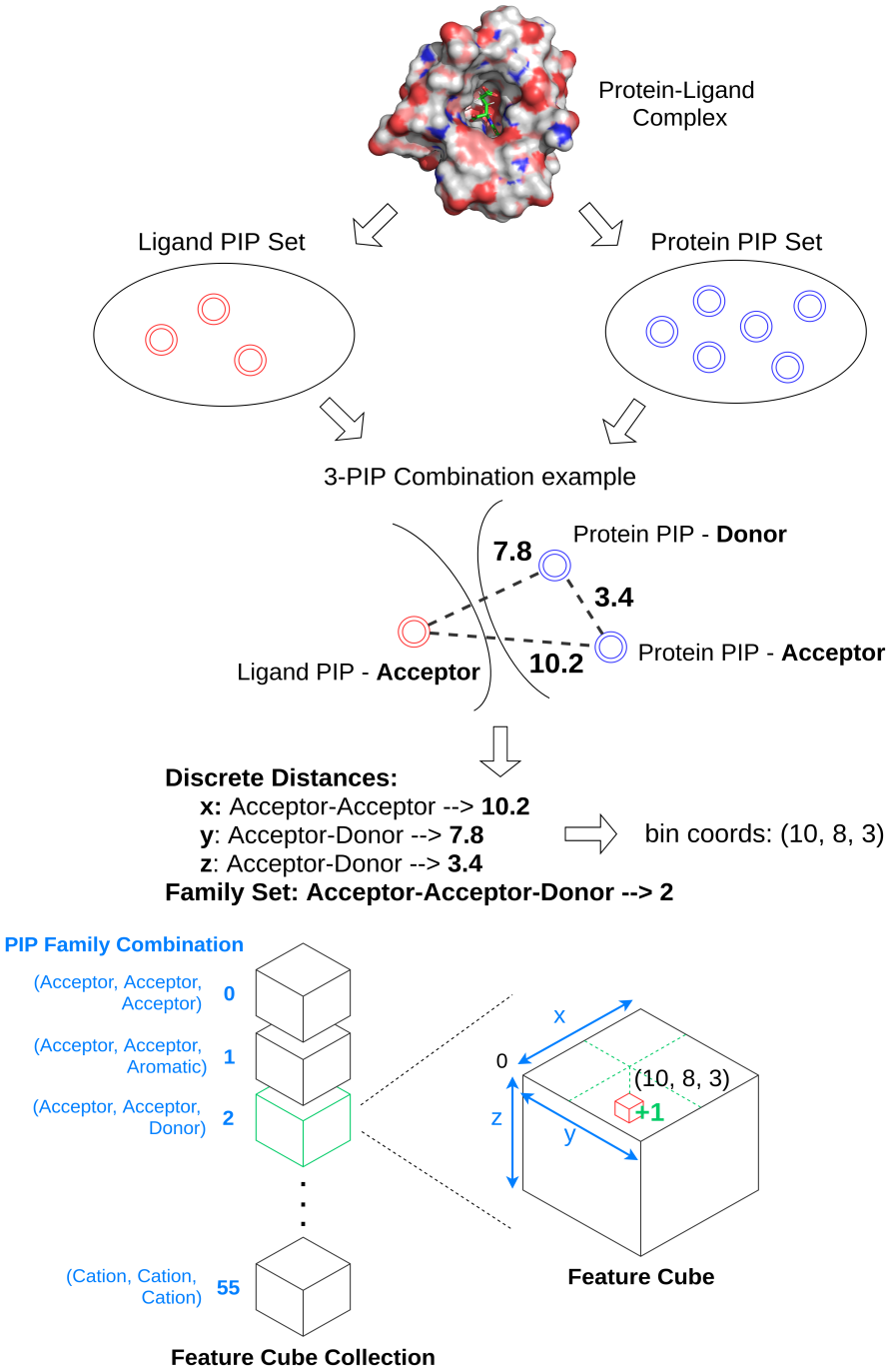


Figure 3.10: LigityScore3D Feature Cube Collection Generation. A protein-PIP set and a ligand-PIP set are extracted from a PL complex. From each 3-PIP combination taken from the PIP pool, the family set, and their discrete distances are used to update the binning count in the feature cube. A feature cube is built for every available family-set. Choosing 2 protein-PIPs and 1 ligand-PIP, and choosing 2 ligand-PIPs and 1 ligand-PIP combinations are both evaluated when compiling the feature cube collection.

features cubes are stacked together to create a protein-ligand LigityScore3D representation of size ($1176 \times 21 \times 21$). In order to ensure that the correct binning coordinates are selected the following measures are taken:

- **Family Set Ordering**. The families of the 3-PIP combinations are ordered by the first two family names so that a unique family sequence is created. These sequences are stored in a python dictionary with a corresponding index. Therefore all 3-PIP combinations considered can only belong to one of the 56 family sequences, each corresponding to a particular index. This is also illustrated in Figure 3.10 (*PIP Family Combination*).

- **Binning Coordinates**. When a particular PIP family is repeated two or three times for a particular 3-PIP combination (example Acceptor-Acceptor-Donor, or Donor-Donor-Donor), there can be the situation where the *same* distance-pairs within the triangular structure correspond to a different binning coordinate. In order to avoid this situation, the binning coordinates are calculated by ordering both the family-pairs and their distances at the same time. This concept is illustrated in Figure 3.11.

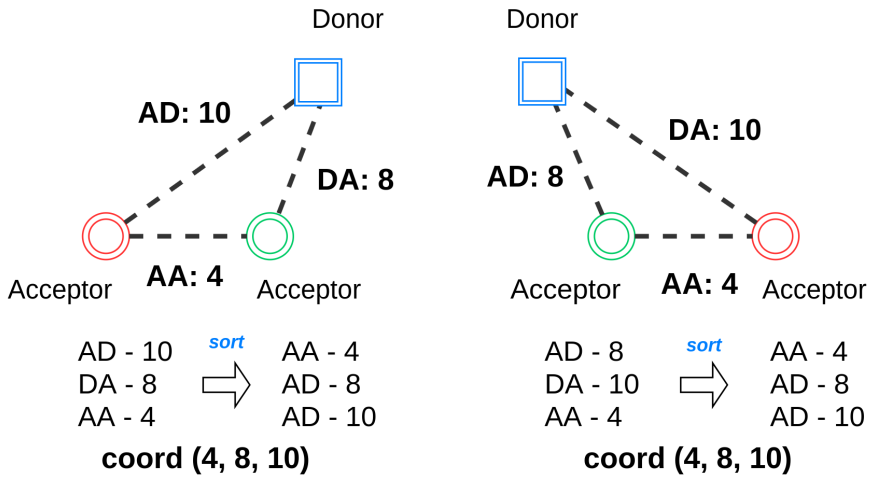


Figure 3.11: LigityScore3D 3-PIP binning coordinates. In order to extract the correct binning coordinates, both the families pairs and their corresponding distances are used to sort the distance. This ensures that the same binning coordinates are used for the same structure. Although the structure are a mirror of each other, they still update the same binning coordinate (4,8,10).

As indicated by Figure 3.10, the PIP distances are calculated by using combinations across both the ligand-PIPs and the protein-PIPs. In LigityScore3D, a combination a

3-PIP is considered at a time. All the possible combinations using two PIPs for the protein, and one PIP from the ligand, plus the combinations where two ligand-PIPs and one protein-PIP are considered. This contrasts with the approach used in Ebejer et al. (2019) where only the ligand-PIP pool was considered to take 3-PIP and 4-PIP combinations. Our hypothesis is that since the protein is essential for SBVS, considering also the protein-PIP in the feature generations strengthens our model for feature representation for development of the SF.

3.7 | Convolutional Neural Network Implementation

The architecture used for LigityScore is a deep convolutional neural network with a single regression output neuron used for prediction of binding affinity. The CNN architecture used for both LigityScore1D and LigityScore3D are illustrated in Figure 3.12 and 3.13. The dimensions at each layers are also highlighted. The model automatically extracts patterns from the protein-ligand representation and encodes these patterns in the weights of the model. The patterns extracted should differentiate the spatial information between different complexes captures from the PIP interactions. A python package "`sbvscnn`" was developed to configure and build the CNN network. The package provides a dynamic way to construct different CNN architectures using a number of input parameters such as the number on convolution layers, if maxpooling, normalisation, and dropout are implemented at each layer, number and dimensions of the fully connected layers. Table 3.5 includes a complete list of parameters used to define our CNN implementation, highlighting also an example of the expected values.

As described in previous sections the input for LigityScore1D is (21×21), whilst the input for LigtyScore3D is $(98 \times 98 \times 54)$. These inputs are treated as 2D and 3D tensors respectively, and our approach treats them similar to a greyscale and colour image respectively. This analogy to an *image* stems from the fact that the protein-ligand complex representation includes a matrix of integer values representing counts of spatial distances across different pharmacophoric family pairs which are very similar to an image matrix. This analogy allowed us to explore and use image processing techniques to optimise the scoring function model. Data augmentation using rotations of the *"image"*, and normalisation at the convolution layers (Ulyanov et al., 2017) are both techniques that are used in image processing and are also used in our experiments to verify if they can improve the prediction of the ligityScore SF.

The network consists of two components namely the convolutional block, and the fully connected block. The model shown in Figure 3.12 for LigityScore1D has three con-

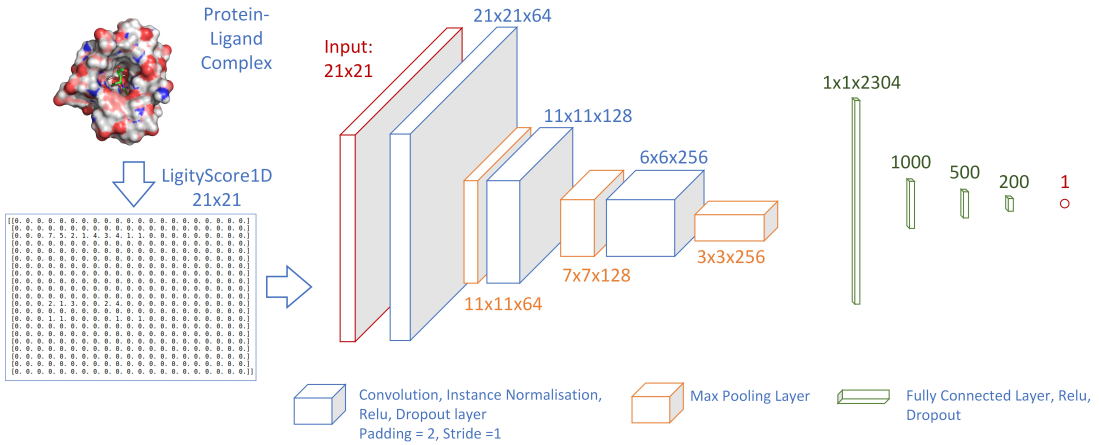


Figure 3.12: CNN Architecture for LigityScore1D. The feature representation is directly input to the network with 4 convolutional layers with instance normalisation, RELU activation, and spatial dropout applied in each layer. The output of the last convolution layers is flattened to input to a fully connected network with 3 hidden layers. The output is a single neuron predicting the binding affinity.

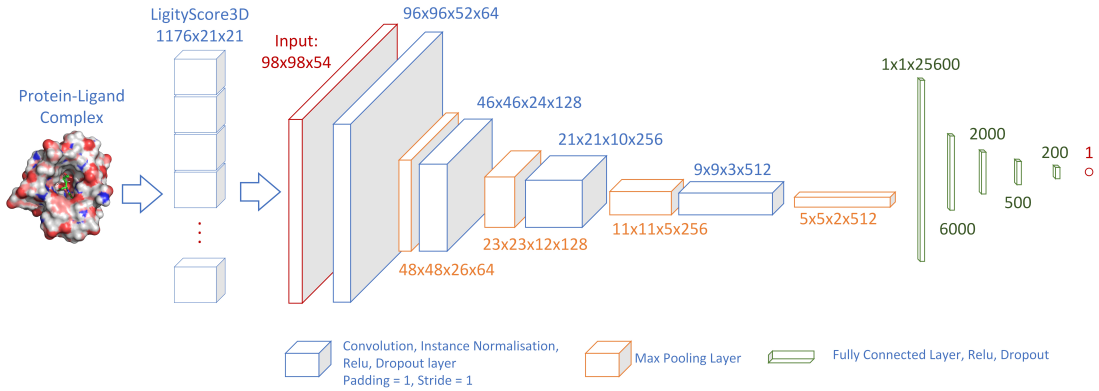


Figure 3.13: CNN Architecture for LigityScore3D. The input is reshaped to ($98 \times 98 \times 54$). Four convolutional layers are applied each layer enabled for instance normalisation, RELU activation, and spatial dropout. The output of the last convolution layers is flattened to input to a fully connected network with 4 hidden layers. The output is a single neuron predicting the binding affinity.

volutional layers with filter dimensions 64, 128, 256, whilst the model for LigityScore3D in Figure 3.13 has four convolutional layers with filter dimensions of 64, 128, 256, and 512, which are all randomly initialised. The PyTorch *conv2D* module was used in both cases for each of the convolutional blocks. A convolutional kernel size of 5x5 was used, with a padding of two in order to keep the output size the same as the input after convolution. Each convolution layer included instance normalisation, RELU activation, and convolutional dropout components, which is then followed by a maxpooling layer with a patch of two to reduces the dimensions by half. The output of the last convolution layers is flattened to be used as input to four fully connected layers. LigityScore1D had a dimension of $(3 \times 3 \times 256)$ at the FC layers whilst LigityScore3D had an $(5 \times 5 \times 2 \times 512)$. To cater for the difference in inputs the fully connected layers were assigned dimensions of (2000, 1000, 500, 200) and (6000, 2000, 1000, 200) respectively.

All the weights of network layers are initialised with the Pytorch defaults. The layers are therefore initialised using the random uniform distribution using the Kaiming method presented in He et al. (2015). This initialising method is well suited for use with the RELU activation function as this method keeps the standard deviation of the layer's activation close to 1. Correctly initialising the weights of the network is important for the training of deep neural networks as this prevents the output of the activation layers from exploding or vanishing, which would produce an insignificant output.

Stochastic gradient descent with the Adam optimisation (Kingma and Ba, 2014) is used with default parameters for momentum scheduling ($\beta_1 = 0.99$, $\beta_2 = 0.999$) to train the network with a learning rate of 10^{-5}. Adam optimisation was also used by Jiménez et al. (2018) and Stepniewska-Dziubinska et al. (2017). Various batch sizes were used for training ranging from mini-batch sizes of 5 to 25. The datasets are shuffled and split into mini-batch is order to speed up the training process. These mini-batches allow the network to converge faster by making fewer adjustments (Mishkin et al., 2017). A number of experiments were done to explore different architectures and optimise these parameters. These experiments are described in Section 3.8 whilst Table 3.5 provides a summary of all the parameters used in the network.

A custom PyTorch *Dataset* and *Dataloader* class is used to prepare the data for input to the CNN. These classes are used to load, preprocess and augement the data, and create a mini-batch for the input to the CNN. The following preprocessing and data augmentation techniques are implemented:

- **Scaling.** The Min-Max scaling function of Equation 3.2 is used to scale the input

Table 3.5: Summary of parameters used for CNN models for LigityScore1D and LigityScore3D that are available in the `sbvscnn` python package with a brief descriptions for each parameter.

Model Parameter	**Parameter Example**	**Parameter Description**
Convolution Module	conv2D	Pytorch Module used.
Input Channels	3	Input depth dimension (C_{in}).
Convolution Layers	3	Number of Convolution Layers.
Convolution Filter Dimensions	64, 128, 256	Filter depth at each layer.
Convolution Kernel Size	5×5	Filter windowing size.
Convolution Stride	1	Steps moved during convolution.
Convolution Padding	2	Padding applied before convolution.
Normalisation Function	batchnorm	*batchnorm* or *instance* norm options.
Normalisation Layer Activation	[1, 1, 0]	Boolean List to indicate where batchnorm is applied. Length equal to number of conv layers.
Convolution Dropout Activation	[1, 0, 0]	Boolean List to indicate presence of spatial dropout.
Convolution Dropout Probability	[0.1, 0, 0]	Spatial Dropout probability to configure at each layer.
Max-Pooling Layer Activation	[1, 1, 1]	Boolean List to indicate where pooling is applied. Length equal to number of Conv layers.
Max-Pooling Kernel Size	2	Patch size used for pooling
Max-Pooling Stride	2	Steps move at a timein
Max-Pooling Padding	0	Padding applied before pooling
FC Layers	4	Number of fully connected layers, incl. output.
FC Dimensions	[4000, 512, 128, 1]	Size of each FC layers
Activation Function	RELU	Activation func used across all layers.
Activation Function Activation	[1, 1, 1, 1]	Boolean List for RELU layer activation
FC Dropout Activation	[1, 1, 1, 0]	Boolean List for dropout activation.
FC Dropout Probability	[0.5, 0.5, 0.5, 0]	Dropout probability at each layer.
Batch Size	5	Mini-batch Size.
Epochs	200	Training epochs.
Learning Rate	10^{-5}	Learning Rate used in Pytorch Optimiser.
L2 Regularization Decay	10^{-3}	λ parameter for L2 regularisation.
Training Rotations	4	Allowed values 1, 4 or 8 for LigityScore1D. 1, 6 allowed for LigityScore3D.
Weight Initialisation	0	boolean enable custom weights.

between 0 and 1.

$$X_s = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.2}$$

where, X is the input element to scale, X_{min} is the minimum value in the matrix, whilst X_{max} is the maximum value in the matrix. X_s represents the scaled output element.

- **Reshaping.** In the case of LigityScore3D, the feature representation has a dimension of $(1176 \times 2 \times 21)$. This is reshaped to $(94 \times 94 \times 54)$ in an effort to have uniform dimension sizes at the input of the CNN. In this case, using the analogy of a coloured image, the image has input dimension of 94×94 with 54 colour channels. No reshaping was done for LigityScore1D.

- **Rotation.** Rotations were used for data augmentation of the dataset. Rotated feature matrices are used to present a different *view* of the same protein-ligand representation at the input of the CNN. This is used in our experiments to verify if more input data can improve the generalisation ability of the model. The inputs to the CNN must have a consistent input dimensions, so the rotated matrix be of the same size as the original matrix. A slightly different approach was used to handle rotations for LigityScore1D and LigityScore3D:

  - LigityScore1D has the option of four or eight rotations where the feature matrix is rotated by 90° and 45° angles respectively. The feature matrix has as a square input dimension, however the length and width of the matrix increase when rotated by 45°. To avoid discarding any elements in the matrix at the edges, the matrix is pre-padded so that when it is rotated it can accommodate all of the feature elements. Ninety degree angles do not have this issue and the feature matrix remains of the same size. Rotations at 90° showed improved generalisation results and showed an increase in performance of this scoring function.
  - LigityScore3D has the option to rotate the 3D matrix by 90° along any of the planes about the 3 axis. Since the feature representation has a cuboid shape, six possible rotated views are available across all the planes. Four rotation are allowed across the x-axis plane, while an additional rotation is achieved across the other two planes.

Early Stopping is an optional parameter for the CNN training module and can be used to stop training when the validation errors loss does not improve over a number

of epochs. A minimum number of epochs can be configured so that tracking of the performance of the loss only starts after this minimum number of epochs are complete. If early stopping is not enabled, the epoch with the smallest RMSE value is chosen as the best epoch.

Apart from Early Stopping, a number of techniques are also used to prevent the CNN network from overfitting. These include L2 regularisation in the loss function, dropout at the fully connected layers, spatial dropout for the convolutional layers, data augmentation using new protein-ligand complexes from more recent PDBbind versions, artificial data augmentation through rotations, minimizing the complexity of the neural network, and also by introducing batch normalisation that can also have a regularisation effect (Mishkin et al., 2017). The experiments related to these techniques are described in Section 3.8.

3.8 | Experiments

Figure 3.14 provides the high level workflow used to carry out a number of experiments to develop the LigityScore SF. After replicating the work in Pafnucy using our own code, as described earlier in Section 4.2, the first implementation of LigityScore was carried out. The LigityScore1D PL representation was tackled first as it is a simpler approach than LigityScore3D. As indicated in Figure 3.14 a series of initial experiments were carried out to get a valid result from the model. Once a valid result was obtained a series of additional experiments were done using a two pronged approach, inline with the objectives presented in Section 1.3. These include:

- **CNN Model Optimisation**. In this aspect, different parameters of the network were tested to find the parameter set that maximised the prediction results both in terms of computations speed and in prediction performance. These experiments are highlighted in Section 3.8.2. After trying a number of options with no tangible improvements, a second wave of experiments is triggered focusing on changing the molecular representation. The experiments were evaluated by comparing the results obtained for the predictions of the validation and test sets using the R, SD, and RMSE metrics detailed in evaluation criteria of Section 2.3.

- **Molecular Representation Optimisation**. In this second aspect the objective was to change the data used to represent the protein-ligand complex in order to find a better representation that encodes as much as possible information of the protein-

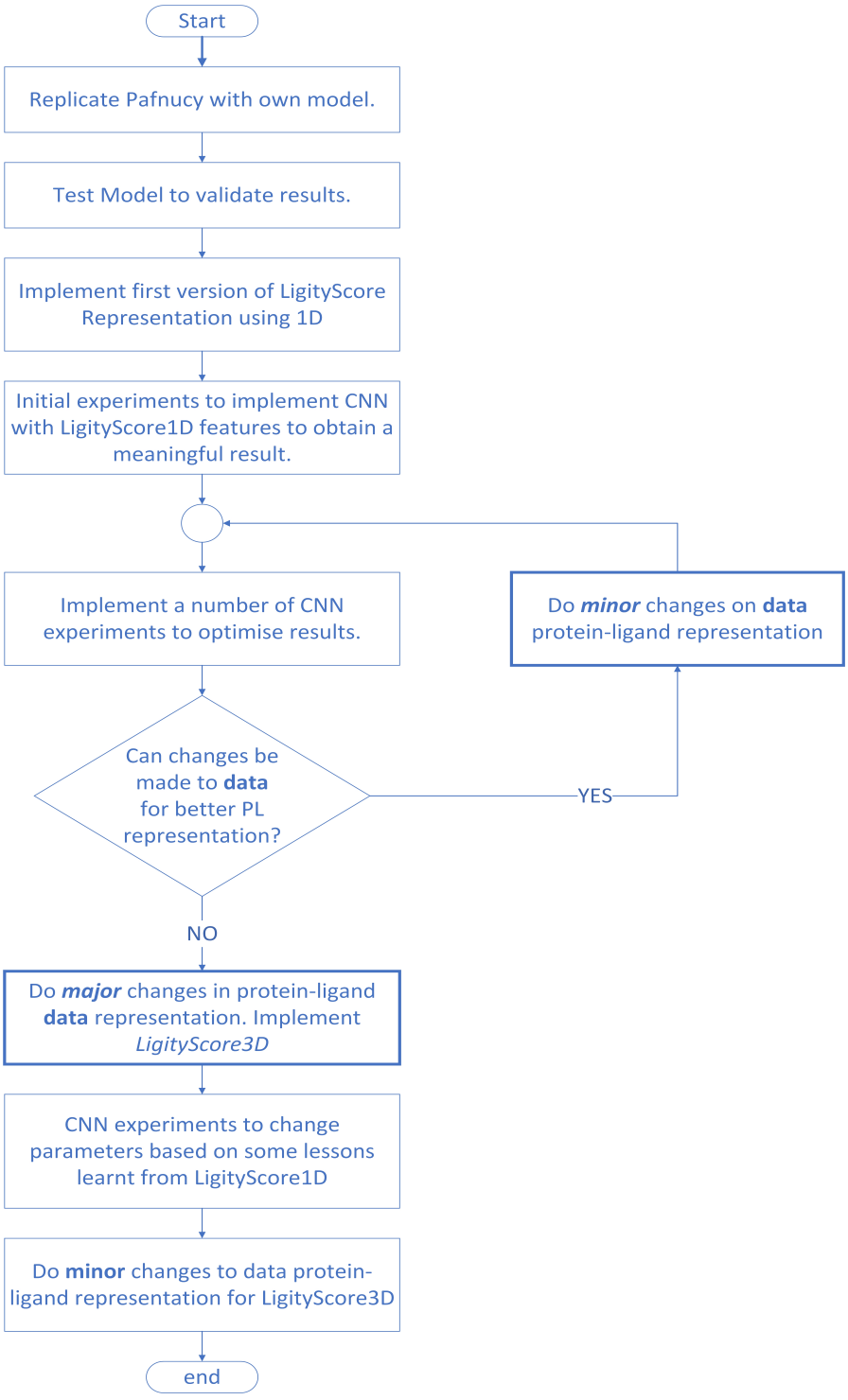


Figure 3.14: High Level experiment workflow to provide an overview on the approach taken to implement different experiments to develop the LigityScore scoring function.

ligand complex, so that this new information can then be harvested by the CNN network. These experiments are highlighted in Section 3.8.1.

After performing a number of minor changes to the representation used as input to CNN such as changing the allowed max distance during feature generation, or changing the PIP generation distance threshold factor, a major change was implemented that lead to the development of LigityScore3D. LigityScore3D as detailed in Section 3.8.1 is more complex and required additional methods for feature extraction. Some of the experiments carried out on LigityScore1D were re-implemented to reassess their impact on the new model. The following sections summarise the experiments done for these two approaches on both LigityScore1D and LigityScore3D.

3.8.1 | Molecular Representation Optimisation

The following experiments were done to optimise the LigityScore protein-ligand complex representation. The results for these experiments are provided in Chapter 4.3.

- **LigityScore Dimension**. This represents major changes in the feature generation module as described in the previous sections. Since the LigityScore3D is a more complex model that includes more detailed spatial information on the PIP interactions it was used to validate if this additional information could achieve better results.

- **PIP Generation Threshold Factor.** The recent work of Zheng et al. (2019) has showed that intermolecular interactions further away from the binding site are still important. The authors used these long range interactions to create their *OnionNet* descriptors to achieve superior results than the *Pafnucy* (Stepniewska-Dziubinska et al., 2017) model. Inspired by this, we have tried to capture additional interactions by using larger PIP distance thresholds. The PIP distance thresholds as defined by Ebejer et al. (2019) are listed in Table 3.4. To achieve this we have conducted a number of experiments where these thresholds were increased by a factor given as an argument to the PIP generation module. In these experiments, various factors were considered ranging between 1.0 (representing thresholds used in Ebejer et al. (2019)) to 1.6. Although the PIP distance threshold were increased the maximum distance allowed between PIP combinations was left to 20Å.

  Increasing the PIP distance threshold implies that more PIPs are extracted from the protein-ligand complexes. This leads to a higher number of PIP combinations to

generate the feature matrix for LigityScore. Each variation in PIP distance threshold requires the re-generation of the PIPs, followed by the feature representation generation for both LigityScore1D and LigityScore3D.

- **Max distance allowed in Feature representation.** In line with the above, the maximum distance allowed between combinations of PIPs during the 'feature generation' process is increased. The feature generation scripts are re-run using different maximum distance thresholds. Maximum distance of 30Å and 40Å were tested for LigityScore1D keeping the discretisation resolution of the distance to 1Å.

- **Lipinski Rules of 5**. In this experiment we have used the Linpinski rule of Five (Table 3.3) to determine if drug-like molecules can achieve better performance. The filtering of PDBbind molecules based on these rules is implemented as part of the PIP generation module.

- **Data Augmentation.** LigityScore was trained using the PDBbind v2016 and v2018 in order to determine if the model can learn and generalise better when there are more protein-ligand complexes available for training. As shown in Figure 3.2 there are considerable additional complexes in the more recent versions, so the aim of this experiment was to determine if additional training data leads to better prediction performance.

  Apart from data augmentation through the addition of more real protein-ligand complexes for training, artificial data augmentation was also used during the CNN training by providing different rotations of the feature matrix as was described in Section 3.7. The LigityScore descriptors are rotationally invariant, however rotations were included in the experiments to test if they improve the generalisation ability of the model. Both data augmentation techniques were tested to determine if they lead to better affinity prediction results .

3.8.2 | CNN Hyperparameter Tuning

The following list describes the parameters relating to the CNN architecture that were tested to optimise the binding affinity prediction of LigityScore. The CNN architecture used in Pafnucy was used as a baseline for the parameter optimisation. The Tensorboard toolkit from the TensorFlow platform was used for visualising how various metrics such training loss, validation loss, and also R of the training and validation sets, change as training progresses.

- **PyTorch Convolution Module**. The conv2D and conv3D modules were evaluated. The conv2D module expects a 4D tensor as input. The 4D tensor represents, the batch size, number of input channels, width and height. LigityScore1D was trained using the conv2D module and therefore the 4D input tensor for a batch size of 20 was $(20 \times 1 \times 21 \times 21)$. LigityScore1D has only one input channel and is analogous to a greyscale image.

  The conv3D module expects a 5D tensor as input with batch size, number of input channels, width, height, and depth. LigityScore3D was tested using both Pytorch conv2D, and conv3D modules. The conv3D module has an input dimension of $(20 \times 1 \times 54 \times 98 \times 98)$, representing a 3D input with only one channel. On the other hand the conv2D modules has an input dimension of $(20 \times 54 \times 98 \times 98)$ which is analogous to a colour images with *54* colour channels. These experiments were carried out in order to determine if the convolution module used, has any impact on the prediction performance. The conv3D model has substantial more training parameters which results in additional compute and GPU resources, and training time.

- **Convolution Layers**. A number of experiments were performed to test CNN architectures with different number, and sizes of convolution layers. Additional convolutional computations were added so that multiple convolutions with the same filter depth are performed at the each stage. The VGG16 (Simonyan and Zisserman, 2014) model is also tested in our approach. These tests were motivated from the series of breakthroughs achieved by deep networks in image classification (He et al., 2016; Simonyan and Zisserman, 2014).

- **Convolution Filters**. Convolution kernel sizes of (3×3) and (5×5) were tested as they are amongst the most popular and are used to achieve state of the art results in image classification in Simonyan and Zisserman (2014) and Szegedy et al. (2015) respectively. The input at each layer has a padding of 1 or 2 respectively so that the output of the convolution layer does not change in size. A stride of 1 was maintained throughout all experiments.

- **Normalisation**. Batch normalisation (batchnorm) or instance normalisation (instancenorm) apply a normalisation layer after each convolution operation so that the inputs of each layer in the network is also normalised. These normalisation techniques are presented in detail in Section 2.2.2. Batchnorm applies a normalising function across all the weights together at a particular layer, whilst instancenorm applies normalisation for each individual weight channel. As reported by

(Mishkin et al., 2017) batch normalisation can increase the stability of the network as it acts as a regularisation function and helps solve the exploding/vainishing gradient problem. These experiments were performed to determine if *batchnorm* or *instancenorm* can increase the generalisation ability of the network, and also if they can improve speed of convergence of the network. Although *batchnorm* can be applied before or after the activation functions (Mishkin et al., 2017), in our approach it is applied before the activation function, as was done in Ioffe and Szegedy (2015) when using the RELU activation function.

- **Pooling**. Pooling affects the size at the input of the fully connected layers. Experiments were performed to determine if it is beneficial to disable pooling at particular layers to vary the inputs at the fully connected layers and test if this effects the prediction performance. Max pooling was used for all experiments with pooling patch size of 2, to reduce the size of its input by half.

- **Dropout**. In this experiment various dropout probabilities, ranging between 0.5 to 0.8 are tested, to determine the affect on generalisation performance and hence it effect on prediction performance. Dropout is applied at the fully connected layers where a number of activations are *dropped* or zeroed during the training phase with a probability p_{drop} that can be input as a parameter in our model.

- **Spatial Dropout**. Since it is a common approach to use dropout to prevent overfitting in fully connected layers, this test explores if dropout at the convolution layers, known as *spatial dropout*, can have a similar effect. Spatial dropout is formulated by Tompson et al. (2015) and considers the spatial location of the features. Instead of dropping activations randomly, spatial dropout drops the activation of a entire convolution filter or feature map as described by Tompson et al. (2015).

- **Batch Size**. Mini-batch sizes ranging between 5 and 25 samples were tested to find the best trade-off between model computational performance and generalisation ability. Larger mini-batch sizes increase the performance of the CNN since less updates are done on the model, however this might have a degradation in prediction performance. Additionally extremely low mini-batch sizes might not provide any gain in prediction preformance but could significantly slow down training (Mishkin et al., 2017).

- **Fully Connected Layers and Dimensions**. The fully connected layer is used to process extracted features from the convolution layer, and predict the binding

affinity. In this experiments we test different number of layers and size and evaluate their effect on model performance.

3.8.3 | Implementation Details

In this section we highlight details on the software packages and hardware used to generate the results of this research project. This project was supported by Amazon Web Services (AWS) through their "AWS Cloud Credits for Research" program (aws, 2020) who awarded us 4,500 US dollars after successful submission of our proposal for this study. Therefore, all our experiments were conducted on AWS EC2 instances. In order to optimise the costs, different EC2 instances were used to handle different workloads. The feature extraction and representation processes are more heavy on memory but do not require any hardware acceleration. On the other hand all the CNN model training and predictions were run on hardware accelerated GPU machines. The list below provides a summary of the instances used. Each instance was running Ubuntu 16.04 as part of the AWS deep learning amazon machine image (AMI).

- **R5 Instance**. The R5 instances were used for PIP generation and the Feature generation algorithms. These are memory optimised instances that have a higher memory-to-cpu ratios than general compute instances. The R5 instances are based on the Intel Xeon Platinum 8000 series processors with a core frequency of 3.1 GHz. The "r5.large" instance with 16GB of RAM and 2vCPU was used for the LigityScore1D, whilst the "r5.2xlarge" with 64GB and 8vCPU was used for LigityScore3D. In the latter case the $(1176 \times 21 \times 21)$ numpy array initialed using the *int16* data type for each protein-ligand complex uses considerable memory. All generated datasets were constructed using the *pandas* data structures and saved to disk using the *DataFrame.to_pickle* module to serialize the DataFrame object to a file. These IO disk processes requires additional memory utilisation for correct loading or saving of the datasets. For these reasons R5 instances were selected.

- **G4 Instance**. The G4 instances are GPU based machines that are equipped with Intel Xeon processors and Nvidia T4 Tensor Core GPUs dedicated which is dedicated to the instance. The T4 GPU has 320 Turing Tensor cores, 2,560 CUDA cores, and 16GB of memory. The "g4dn.2xlarge" instance with 8vCPU and 32GB RAM was used to train and test LigityScore1D. On the other hand the "g4dn.4xlarge" with 16vCPU and 64GB RAM, and "g4dn.8xlarge" 32vCPU with 128GB RAM were used to train and test LigityScore3D, depending on the complexity of the CNN architecture used. The choice of G4 instance size was primarily based on the RAM

requirements, since all of the mentioned instance have one T4 GPU with 16GB of RAM.

All the development of the LigityScore modules were done using Python 3.6 using an Ubuntu 18.04 machine. Table 3.6 provides a summary of the python packages used for development of LigityScore.

Table 3.6: Summary of Python 3.6 packages used with versions with short description where each module is utilised.

Python Package Name	Package Version	Usage Summary
RDKit	2019.09.3	Used for pre-processing and filtering based on Lipinski Rule of 5. Used for PIP Generation and Feature Generation.
PyMOL	2.3.5	Usine in pre-processing to fix molecules with RDKit errors.
pandas	1.0.3	Dataset Generation. Used in all modules.
NumPy	1.18.1	Dataset Generation. Used in all modules.
PyTorch	1.4.0	CNN Training and Predictions
TensorBoard	2.1.0	CNN loss, RMSE, and R metric monitoring.
OpenCV	3.4.2	Rotations for artificial data augmentation
scikit-learn	0.22.1	Used for linear regression to find SD performance metric.
seaborn	0.10.0	Used for statistical plots.

3.9 | Summary

In this Chapter we have presented a detailed description for LigityScore implementation including all the modules developed that were required to meet the objectives of this research project. At its core, LigityScore uses the PIP generation (Section 3.5) and Feature Generation (Section 3.6) algorithms modified from Ebejer et al. (2019) to be used as an input to the CNN where they include PIP combinations from both the ligand and protein PIP pools, using different PIP distance threshold to extract our own protein-ligand representations. This representation can be used to train a CNN network to predict the protein-ligand binding affinity. A detailed analysis of the datasets used,

and the CNN architecture implemented were also presented. We have also described the experiment workflow process that was followed to generate the results required to answer the research question hypotesised in this study.

In the following chapter, we present and discuss the results generated from the experiments described here and outline the best performing model.

4

Results & Evaluation

In this chapter we will review the experiments executed to test and optimise the performance of LigityScore in line with the aims and objectives of this study. This study aims to find a suitable representation of the protein-ligand complex that can leverage the ability of the CNN architecture to extract representative features automatically from the protein-ligand complex, and use them for binding affinity prediction. The following experiments were carried out using the methodology described in Section 3.8.

- Baseline implementations based on the work of *Pafnucy* (Stepniewska-Dziubinska et al., 2017) to create a benchmark model of the scoring function for comparison and evaluation of LigityScore.
- Evaluate the suitability and effectiveness of the LigityScore protein-ligand representation for use as a scoring function for SBVS. We created two different representations in this study, namely LigityScore1D and LigityScore3D, and each approach was tested separately due to differences in the data representation. We performed several experiments to optimise these representations and determine if different LigityScore feature generation parameters can create a representation that encodes more details of the protein-ligand structure that provides a better signal to noise ratio.
- Evaluate different CNN architectures to find the best model for use with the data representation presented by LigityScore.
- Optimisation of the CNN hyperparameters to obtain the best binding affinity predictor.
- All the experiments were evaluated using the CASF-2013 and CASF-2016 test sets using the Pearson correlation coefficient, R, and the standard deviation in linear

regression, SD, as these metrics are standard practice when measuring the performance of scoring functions for binding affinity predictions.

LigityScore, as a scoring function, depends on two main elements. The data representation of the protein-ligand complex, and the underlying CNN architecture for feature extraction and affinity prediction. Therefore the experiments detailed in this chapter aim to enhance these two elements of LigityScore.

4.1 | Pafnucy Replication Results

Our own implementation of the Pafnucy model achieved R values of 0.718, 0.690, 0.761 for the training, validation, and CASF2016 sets respectively, whilst Stepniewska-Dziubinska et al. (2017) achieved 0.77, 0.72, 0.78. Our results show a slight difference that can be attributed to a number of factors that are divergent from the original $Pafnucy$ model. These include:

- **PyTorch**. Stepniewska-Dziubinska et al. (2017) build their CNN using TensorFlow, whilst our models are built using the Pytorch framework as it is preferred tool for development of LigityScore for its ease of use, simplicity, and pythonic framework. Differences in the implementation of the CNN modules might have contributed to this change.

- **Partial Charge Calculations**. The *Pafnucy* model uses the partial charge of the atoms in its representation. The partial charges in *Pafnucy* were calculated using the Chimera software package (Pettersen et al., 2004), whilst our method uses Pybel (O'Boyle et al., 2011) as this was better suited for our pre-processing modules based on Python. As was discussed by Zheng et al. (2019), such *features* are prone to introduce additional estimations that can lead to extra noise or bias. The partial charge calculation is one such features and their calculation varies across different software packages and therefore might have also effected the difference in our results. The partial charge feature is an important component in the Pafnucy representation and was ranked the 6^{th} most influential features out of the 19 used.

- The 1,000 protein-ligand complexes of the validation set are randomly selected from the PDBbind Refined set. These complexes were not listed in the work of Stepniewska-Dziubinska et al. (2017) and therefore our approach also took 1,000 random samples from the refined set. The different selection of validation, and hence training set, can also have an effect on the difference in results. This was also

experienced in our approach and the standard deviation for 10 different validation and training sets are presented in Table 4.5.

Therefore, the results 0.718, 0.690, 0.761 for the training, validation, and CASF2016 sets respectively, confirm that our implementation provides similar results to those published by Stepniewska-Dziubinska et al. (2017). Some of the modules used to build our implementation of Pafnucy, such as preprocessing module and the CNN training and prediction modules, are used also for the development of LigityScore. The replication results confirm the correct functionality of these modules, whilst also validating that we have a good baseline to develop our scoring function models.

The work involved in replicating Pafnucy was also used for pedagogical purposes so that we can learn the model specifics and determine the exact procedures carried out in Pafnucy implementation. Equipped with such knowledge on the Pafnucy implementation, it enabled us to make a like with like comparison of the LigityScore and Pafnucy implementations during evaluation. Pafnucy was successfully implemented using our own code achieving one of the objectives outlined in Section 1.3.

4.2 | Baseline Results

In order to compare the change in performance when modifying CNN parameters a baseline CNN model was established using the parameters used by the Pafnucy model. Table 4.1 lists the baseline parameters used for LigityScore1D for PDBbind v2016 and PDBbind v2018 datasets, and LigityScore3D for the PDBbind v2016 dataset. The PDBbind v2018 was used as a data augmentation experiment as it includes an additional 2,843 protein-ligand complexes that were added to the training set. The baseline results together with the results of the various experiments are listed in Table 4.2 and Table 4.3 discussed in detail in Section 4.3. As was described in Section 3.3 the CASF-2013 and CASF-2016 were used for testing.

The Pearson correlation coefficients that were computed for the training and validation sets for LigityScore1D baseline are depicted in Figure 4.1. On the other hand Figure 4.2 show the RMSE of the training and validation sets evaluated at each epoch. The validation set includes 1,000 protein-ligand complexes that are not included in either training or test sets.

The validation set is used to test the performance of the model to evaluate if the model is overfitting. As can be seen from Figure 4.2 the training RMSE continues to decrease as more epochs are computed. Therefore the model is continuously learning the underlying protein-ligand feature maps. However, the validation RMSE does not follow

Table 4.1: Baseline parameters used for CNN models for LigityScore1D and LigityScore3D used to initialise our `sbvscnnmsc` dynamic CNN python module. Any changes to the CNN or data model parameters are compared to this baseline.

Model Parameter	**LigityScore1D v2016**	**LigityScore1D v2018**	**LigityScore3D**
Convolution Parameters			
Convolution Module	conv2D	conv2D	conv2D
Input Channels	1	1	54
Convolution Layers	3	3	3
Convolution Filter Channels	(64, 128, 256)	(64, 128, 256)	(64, 128, 256, 512)
Convolution Kernel Size	5×5	5×5	5×5
Convolution Stride	1	1	1
Convolution Padding	2	2	1
Activation Function	ReLU	ReLU	ReLU
Normalisation Layer Activation	-	-	-
Normalisation Function	-	-	-
Convolution Dropout Activation	-	-	-
Convolution Dropout Probability	-	-	-
Pooling Parameters			
Max-Pooling Layer Activation	all conv layers	all conv layers	all conv layers
Max-Pooling Kernel Size	2	2	2
Max-Pooling Stride	2	2	2
Max-Pooling Padding	0	0	0
Fully Connected Parameters			
FC Layers	4	4	4
FC Dimensions	(1000, 500, 200, 1)	(1000, 500, 200, 1)	(6000, 2000, 500, 200, 1)
Activation Function	ReLU	ReLU	ReLU
FC Dropout Activation	(1, 1, 1, 0)	(1, 1, 1, 0)	(1, 1, 1, 1, 0)
FC Dropout Probability	(0.5, 0.5, 0.5, 0)	(0.5, 0.5, 0.5, 0)	(0.5, 0.5, 0.5, 0.5, 0)
Optimisation Parameters			
Batch Size	20	20	20
Epochs	140	140	60
Learning Rate	0.00001	0.001	0.00001
L2 Regularization Decay	0.001	0.001	0.001
Training Rotations	1	1	1
Weight Initialisation	Kaiming	Kaiming	Kaiming
Data Model Parameters			
Rotations	1	1	1
PIP Generation Threshold Factor	1.0	1.0	1.0
Feature Generation Max Distance	20Å	20Å	20Å
Feature Generation Resolution	1.0	1.0	1.0

this pattern, and is stable for a number of epochs, and then starts increasing again. This is a sign of overfitting (after epoch 115) since the model has learned a complex representation which does not generalise well for the unseen validation samples. In these experiments the epoch that yields the lowest RMSE for the validation set is taken as the best performing epoch, and is used to predict the binding affinity of the CASF-2013 and the CASF-2016 test sets.

The baselines R and $RMSE$ plots for LigityScore1D (2018) and LigityScore3D baseline are similar to those presented for the LigityScore1D (2016) baseline in Figure 4.1 and 4.2.

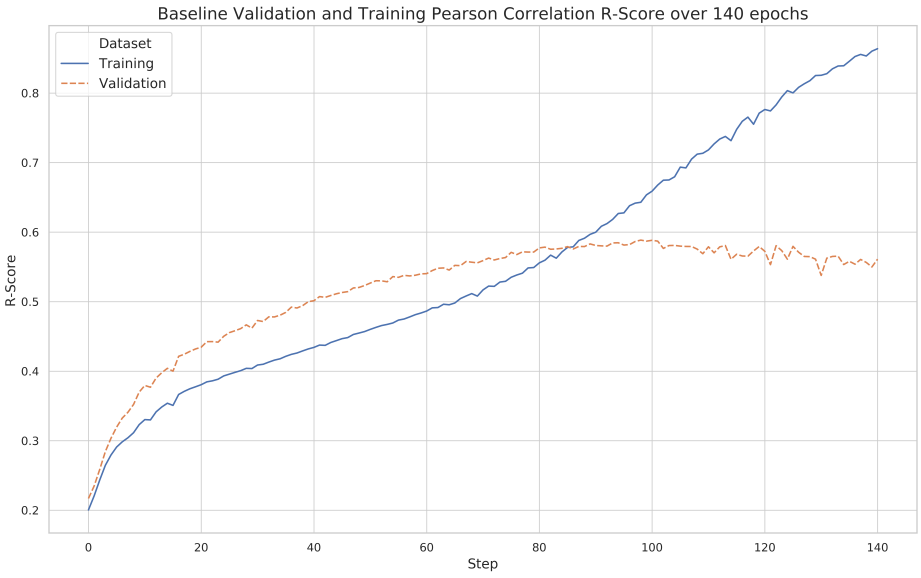


Figure 4.1: LigityScore 1D Baseline R-Score Performance. Model is trained for 140 epoch and the R-score value for the Validation set (orange line) saturates after around 100 epoch, achieving a performance of 0.59.

4.3 | LigityScore Results and Discussion

The next sections presents the results for LigityScore1D and LigityScore3D for various experiments performed on changes in CNN architecture and the data model of the protein-ligand representation.

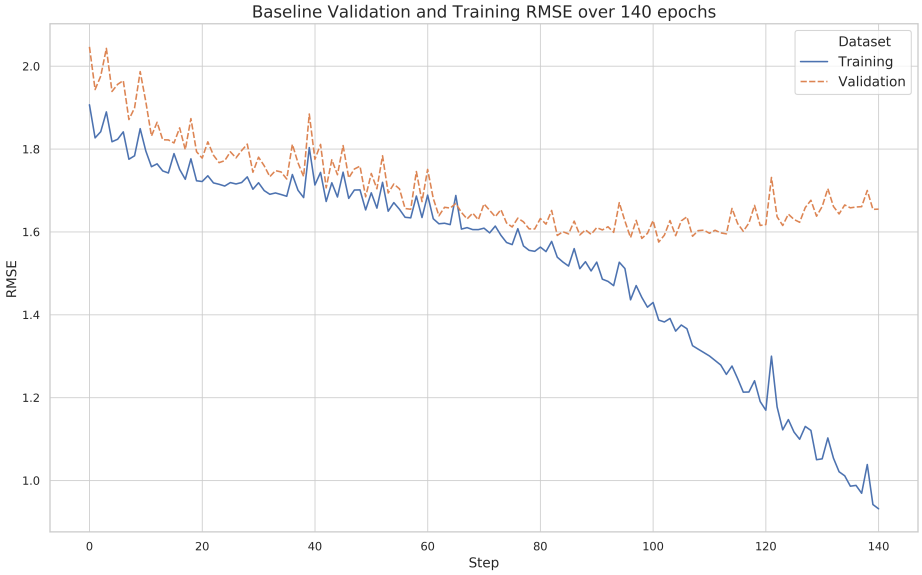


Figure 4.2: LigityScore 1D Baseline RMSE Performance. The RMSE decreases gradually for the first 80 epochs, stabilises for a number of epoch, and starts increasing again after 115 epochs showing signs of overfitting.

4.3.1 | LigityScore1D

LigtyScore1D (v2016 and v2018) results are presented in Tables 4.2 and 4.3 showing performance results in terms of RMSE, and R values. Different parameters in the baseline model were changed to evaluate its impact. The parameters changed from the baseline model are indicated in the *"Exp Shorthand"* column of Tables 4.2 and 4.3 for quick reference, however more descriptive details of the experiments are provided in Table C.1.

For the Pafnucy model the best results were obtained when the weights for the convolutional and fully-connected layers were randomly initialized using a truncated normal distribution with zero mean and standard deviation of 0.001. Using the same type of weight initialisation led to instability in our model and consequently opted to used the *Kaiming* method recommended by He et al. (2015) for DL models with ReLU activation. After using the *Kaiming* method we managed to produce the first meaningful results using the LigityScore representation. Therefore one of the primary objectives to find a suitable representation of the PL complex was reached. The next major objective was to provide the best network architecture that provides the best affinity score predictions. Therefore our next approach was to take measures to increase the generalisation aspect

of the trained model, and optimise the CNN for LigityScore input features.

A similar test execution was followed for both 2016 and 2018 datasets used in LigityScore1D. The first series of experiments (*Exp# 1 to 18*) were done to search for the best CNN hyperparameters. Different mini-batch sizes (25, 15, 10, and 5) and kernel size of 3×3 were tested however these did not show any significant improvement.

Rotations of the LigityScore representation were added for data augmentation during training. Rotations increased the training time as four or eight different rotated LigityScore represention for each PL complex were trained at each epoch. Rotations increase the input size from 21×21 to 37×37 in order to avoid *clipping* any values of the representation. Therefore an additional convolution layer plus pooling was added for these tests to reduce the size of the final convolution layer, and have a suitable input size to the FC network. The Fully Connected layers contain the majority of the parameters of the CNN network. The *baseline* model contains a total of approximately 3.9M learnable parameters, out of which 74% are used to train the Fully Connected layers. The smaller number of learnable parameters at the convolution layers is attributed to the sparse interactions introduced earlier in Section 2.2.2.2. Therefore, to reduce the number of learnable parameters, memory utilisation, and overfitting, it is important to provide an input to the FC layer that is not large. Apart from increasing the training time, the rotations have made the performance of the model worse. The rotated representations contain the same information as LigityScore uses rotationally invariant descriptors, explaining why they did not show improvement in prediction performance.

Increasing the dropout at the FC layers, and increasing the regularisation factor in the ADAM optimizer also did not show improvements in results. For this reason these parameters were not changed from the baseline in other experiments.

Table 4.2: Performance of LigityScore 1D trained on PDBbind v2016. The best results are recorded for *Exp #37* (ID 137) highlighted in green. An R value of **0.695** and **0.725** were recorded for CASF2013 and CASF-2016 respectively. Experiment details are listed in Appendix C Table C.1. The $RMSE_{tr}$, $RMSE_{v}$, $RMSE_{2013}$, and $RMSE_{2016}$ represents the RMSE for training, validation, CASF-2013, and CASF-2016 sets respectively, whilst R_{tr}, R_{v}, R_{2013}, and R_{2016} represent the Pearson correlation for the training, validation, CASF-2013, and CASF-2016 sets.

Exp#	*ExpID*	*ExpShorthand*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_{v}$	R_{v}	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
0	100	*Baseline*	68.88	1.387	0.668	1.575	0.587	1.709	0.626	1.955	0.545
1	101	mini-batch 25	66.64	1.383	0.673	1.581	0.588	1.769	0.612	1.999	0.534
2	102	mini-batch 15	72.75	1.438	0.630	1.574	0.592	1.731	0.622	1.945	0.524
3	103	mini-batch 10	81.57	1.277	0.717	1.548	0.592	1.703	0.609	1.871	0.546
4	104	mini-batch 5	96.88	1.383	0.629	1.521	0.582	1.702	0.593	1.877	0.497
5	105	kernel 3×3	68.26	1.669	0.444	1.695	0.507	1.880	0.518	2.304	0.410
6	106	4 Rotations, 512 Conv layer	211.41	1.359	0.685	1.584	0.583	1.762	0.570	2.233	0.441
7	107	8 Rotations, 512 Conv layer	351.39	1.389	0.663	1.590	0.583	1.779	0.563	2.089	0.452
8	108	LR 0.00005	71.61	1.537	0.570	1.596	0.579	1.768	0.584	2.008	0.477
9	109	LR 0.0001	69.93	1.513	0.597	1.603	0.576	1.767	0.593	2.070	0.451
10	110	BatchNorm (BN)	71.31	1.376	0.682	1.583	0.585	1.758	0.593	1.957	0.497
11	111	InstanceNorm (IN)	72.99	1.355	0.697	1.563	0.599	1.749	0.603	1.985	0.518
12	112	Dropout 0.6 0.6 0.5	70.97	1.407	0.659	1.575	0.593	1.708	0.614	1.936	0.535
13	113	Dropout 0.8 0.8 0.5	71.62	1.437	0.638	1.579	0.587	1.728	0.613	1.987	0.516
14	114	L2 Reg 0.002	71.33	1.475	0.617	1.579	0.589	1.711	0.616	1.980	0.507
15	115	L2 Reg 0.004	71.65	1.426	0.642	1.573	0.590	1.729	0.602	2.005	0.503
16	116	FC dim - 512, 256, 64	70.97	1.379	0.673	1.579	0.586	1.695	0.609	2.216	0.516
17	117	FC dim - 500, 200	70.92	1.353	0.689	1.562	0.593	1.738	0.592	2.073	0.512
18	118	FC dim - 1024, 512, 256, 64	72.90	1.360	0.684	1.575	0.585	1.756	0.587	2.243	0.514
19	119	PIP Threshold 1.1	71.28	1.224	0.755	1.524	0.620	1.704	0.617	1.834	0.589
20	120	PIP Threshold 1.25	72.08	1.211	0.761	1.453	0.661	1.565	0.695	1.885	0.629
21	121	PIP Threshold 1.4	72.27	1.001	0.847	1.431	0.693	1.588	0.684	2.130	0.571
22	122	PIP Threshold 1.5	72.64	1.204	0.761	1.480	0.659	1.599	0.686	1.920	0.576
23	123	PIP Threshold 1.6	72.96	1.248	0.743	1.467	0.663	1.613	0.695	1.864	0.604
24	124	PIP Threshold 1.4 w/Lipinski	38.36	1.364	0.677	1.484	0.601	1.761	0.643	1.744	0.600
25	125	PIP Threshold 1.4 w/Max dist 30	76.84	1.077	0.819	1.433	0.696	1.577	0.684	1.990	0.575
26	126	PIP Threshold 1.4 w/Max dist 40	78.40	1.494	0.593	1.551	0.624	1.764	0.597	1.940	0.574
27	127	PIP Threshold 1.4 w/resolution0.5	78.28	1.522	0.573	1.586	0.604	1.775	0.597	1.910	0.579
28	128	BN–1-4	71.82	1.118	0.810	1.477	0.678	1.664	0.670	2.114	0.610

(continued…)

Exp#	*ExpID*	*details*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_{v}$	R_{v}	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
29	129	IN—1-4	73.71	1.086	0.814	1.444	0.691	1.614	0.681	1.998	0.659
30	130	IN–1-4–Drop0-8	74.11	0.850	0.892	1.458	0.690	1.571	0.694	2.024	0.628
31	131	IN–1-25	74.32	0.378	0.981	1.461	0.664	1.653	0.640	1.957	0.633
32	132	IN–1-4–minibatch15	78.17	0.358	0.983	1.438	0.691	1.557	0.710	1.844	0.669
33	133	IN–1-4–r4	142.58	0.442	0.979	1.440	0.689	1.660	0.645	1.808	0.645
34	134	IN–1-4–r8	220.99	0.402	0.980	1.462	0.680	1.619	0.664	1.916	0.657
35	135	IN-1-4–Cdrop0-1-all	75.92	1.146	0.786	1.485	0.684	1.564	0.707	2.083	0.636
36	136	IN–1-4–Cdrop0-2-all	76.21	0.957	0.865	1.545	0.683	1.561	0.718	2.048	0.649
37	**137**	**IN–1-4–Cdrop0-1-mid**	**76.36**	**0.361**	**0.985**	**1.467**	**0.690**	**1.490**	**0.725**	**1.793**	**0.695**
38	138	IN–1-4–Cdrop0-1-mid–BS15	78.27	0.326	0.985	1.475	0.685	1.558	0.710	1.831	0.665
39	139	IN–1-4–Cdrop0-2-mid	77.12	0.408	0.98	1.502	0.685	1.489	0.726	1.873	0.681
40	140	IN–1-4–Cdrop0-1-all-r4	145.94	0.514	0.96	1.439	0.709	1.539	0.697	1.885	0.629
41	141	IN–1-4–Cdrop0-1-all-r8	226.47	0.637	0.941	1.456	0.695	1.498	0.730	1.860	0.679
42	142	BN–1-4–Cdrop0-1-mid	71.41	0.383	0.979	1.464	0.704	1.534	0.711	1.894	0.660
43	143	IN–1-4–Cdrop0-1-mid–d0.8	73.32	0.479	0.972	1.473	0.685	1.522	0.712	1.948	0.653
44	144	IN–1-4–Cdrop0-1-mid–d0.8–r4	138.93	0.507	0.972	1.420	0.702	1.555	0.705	1.794	0.656
45	145	IN–1-4–Cdrop0-1-mid–4ConvL	76.38	0.300	0.987	1.496	0.670	1.576	0.696	1.775	0.710
46	146	IN–1-4–Cdrop0-1-mid–5ConvL	103.45	0.347	0.982	1.489	0.678	1.631	0.663	1.972	0.651
47	147	IN–1-4–Cdrop0-1-mid–3Convx2	96.90	0.340	0.985	1.458	0.690	1.531	0.706	2.023	0.629
48	148	IN–1-4–Cdrop0-1-mid–4Convx2	151.41	0.320	0.986	1.419	0.706	1.516	0.709	1.742	0.673
49	149	IN–1-4–Cdrop0-1-mid–VGG16	248.43	0.348	0.983	1.427	0.707	1.573	0.698	1.732	0.643

Table 4.3: Performance of LigityScore1D trained on PDBbind v2018. The best results are recorded for Exp #36 (ID 150) highlighted in green. An R value of **0.662** and **0.733** were recorded for CASF2013 and CASF-2016. Experiment details are listed in Appendix C Table C.1. The $RMSE_{tr}$, $RMSE_{v}$, $RMSE_{2013}$, and $RMSE_{2016}$ represents the RMSE for training, validation, CASF-2013, and CASF-2016 sets respectively, whilst R_{tr}, R_{v}, R_{2013}, and R_{2016} represent the Pearson correlation for the training, validation, CASF-2013, and CASF-2016 sets.

Exp#	*ExpID*	*ExpShorthand*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_{v}$	R_{v}	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
0	100	baseline	90.32	1.252	0.736	1.628	0.556	1.808	0.544	2.086	0.424
1	101	mini-batch 25	88.52	1.501	0.577	1.632	0.543	1.935	0.553	2.190	0.434
2	102	mini-batch 15	95.57	1.414	0.635	1.631	0.553	1.824	0.556	2.055	0.466
3	103	mini-batch 10	106.44	1.523	0.545	1.631	0.534	1.826	0.522	2.050	0.444
4	104	mini-batch 5	126.09	1.510	0.526	1.567	0.550	1.810	0.504	1.961	0.434
5	105	kernel 3×3	85.86	1.501	0.574	1.647	0.533	1.804	0.541	2.142	0.474
6	106	4 Rotations, 512 Conv layer	260.23	1.646	0.458	1.718	0.484	1.897	0.501	2.140	0.363
7	107	8 Rotations, 512 Conv layer	432.62	1.700	0.403	1.755	0.436	1.845	0.525	2.137	0.393
8	108	LR 0.00005	93.52	1.457	0.608	1.613	0.561	1.769	0.574	2.041	0.472
9	109	LR 0.0001	90.24	1.853	n/a	1.947	n/a	2.174	n/a	2.428	n/a
10	110	BatchNorm (BN)	91.94	1.447	0.636	1.644	0.548	1.839	0.537	2.221	0.423
11	111	InstanceNorm (IN)	94.19	1.289	0.799	1.700	0.531	1.861	0.557	2.118	0.441
12	114	L2 Reg 0.002	92.00	1.519	0.566	1.652	0.534	1.865	0.515	2.157	0.373
13	115	L2 Reg 0.004	92.41	1.414	0.655	1.644	0.540	1.869	0.506	2.193	0.370
14	116	FC dim - 512, 256, 64	91.34	1.364	0.675	1.615	0.561	1.828	0.545	2.221	0.409
15	117	FC dim - 500, 200	91.41	1.457	0.612	1.596	0.575	1.795	0.561	2.203	0.424
16	118	FC dim - 1024, 512, 256, 64	93.78	1.704	0.384	1.808	0.410	1.955	0.491	2.163	0.403
17	119	PIP Threshold 1.1	90.34	1.087	0.815	1.540	0.609	1.753	0.570	2.169	0.512
18	120	PIP Threshold 1.25	91.28	1.306	0.709	1.492	0.641	1.671	0.653	1.939	0.580
19	121	PIP Threshold 1.4	91.63	1.401	0.668	1.467	0.644	1.609	0.699	1.969	0.598
20	122	PIP Threshold 1.5	92.12	1.328	0.689	1.478	0.659	1.635	0.674	2.184	0.564
21	123	PIP Threshold 1.6	92.29	1.214	0.763	1.449	0.660	1.548	0.709	1.947	0.591
22	124	PIP Threshold 1.4 w/Lipinski	47.88	1.387	0.663	1.451	0.601	1.784	0.622	1.838	0.527
23	125	PIP Threshold 1.4 w/Max dist 30	95.27	1.299	0.721	1.437	0.661	1.668	0.664	2.119	0.610
24	126	PIP Threshold 1.4 w/Max dist 40	98.70	1.332	0.701	1.460	0.646	1.675	0.648	1.974	0.591
25	127	PIP Threshold 1.4 w/resolution0.5	98.83	1.358	0.681	1.474	0.641	1.725	0.623	1.929	0.549
26	128	BN–1-4	106.61	0.352	0.983	1.416	0.687	1.546	0.710	1.824	0.638
27	129	IN–1-4	106.55	1.256	0.751	1.507	0.664	1.627	0.702	1.983	0.624
28	130	IN–1-4–Drop0-8	107.33	2.317	0.890	2.549	0.627	2.710	0.693	2.837	0.634

(continued...)

Exp#	*ExpID*	*details*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_{v}$	R_{v}	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
29	132	IN–1-4–BS15	113.10	1.163	0.785	1.429	0.676	1.642	0.681	1.952	0.586
30	133	IN–1-4–r4	202.54	1.522	0.927	1.910	0.652	2.064	0.672	2.178	0.607
31	134	IN–1-4–r8	314.29	1.334	0.914	1.730	0.613	1.965	0.579	2.183	0.548
32	135	IN–1-4–Cdrop0-1-all	110.19	1.088	0.820	1.418	0.679	1.539	0.719	1.887	0.635
33	136	IN–1-4–Cdrop0-2-all	110.07	0.470	0.967	1.413	0.694	1.510	0.715	1.939	0.651
34	137	IN–1-4–Cdrop0-1-mid	110.42	1.438	0.667	1.533	0.656	1.791	0.646	2.053	0.601
35	139	IN–1-4–Cdrop0-2-mid	111.05	1.111	0.844	1.491	0.676	1.621	0.722	2.027	0.613
36	**150**	**IN–1-6–Cdrop0-1-mid–lr0-0001**	**107.89**	**0.790**	**0.910**	**1.447**	**0.676**	**1.510**	**0.733**	**1.777**	**0.662**
37	151	BN–1-4–Cdrop0-1-mid–lr0-0001	109.14	1.056	0.826	1.389	0.689	1.517	0.729	1.921	0.630
38	142	BN–1-4–Cdrop0-1-mid	92.80	1.347	0.683	1.441	0.659	1.611	0.691	1.920	0.607
39	143	IN–1-4–Cdrop0-1-mid–drop0.8	94.79	1.843	0.268	1.908	0.313	2.161	0.296	2.500	0.320
40	144	IN–1-4–Cdrop0-1-mid–drop0.8–r4	175.14	1.843	0.096	1.910	0.187	2.164	0.072	2.539	0.057
41	145	IN–1-4–Cdrop0-1-mid–4ConvLayers	98.72	1.396	0.787	1.620	0.633	1.834	0.679	2.172	0.587
42	146	IN–1-4–Cdrop0-1-mid–5ConvLayers	130.23	1.060	0.903	1.516	0.657	1.805	0.621	2.019	0.608
43	147	IN–1-4–Cdrop0-1-mid–3Convdouble	123.25	1.702	0.389	1.681	0.514	1.926	0.504	2.272	0.499
44	148	IN–1-4–Cdrop0-1-mid–4Convdouble	187.71	1.757	0.415	1.889	0.442	2.108	0.399	2.440	0.379

Normalisation techniques such as *BatchNorm* and *InstanceNorm* were proved to be effective in optimising the model efficiency. Figure 4.3 and Figure 4.4 illustrate the R-score and RMSE performance when BatchNorm and InstanceNorm are introduced. These models (blue and green line plots) show convergence for the training R-score after only 40 epochs. Comparing this to the baseline performance in Figure 4.1 the model is not seen to converge even at epoch 140. The normalisation models are also compared with a model trained with a higher learning rate (increased 10-fold). Although this shows improved convergence time over the baseline it converges around epoch 80. Additionally the validation R-score is significantly less than that of the normalisation models. Although the normalisation techniques did not improve the prediction ability of the model, they were still used throughout the rest of the experiments due to significant increase in efficiency. InstanceNorm shows slightly better convergence times and R-score results for validation. Therefore, it was used as the preferred choice to be used in further experiments. However, BatchNorm is also tested in other experiments to validate this assumption (Exp ID 128 and 142).

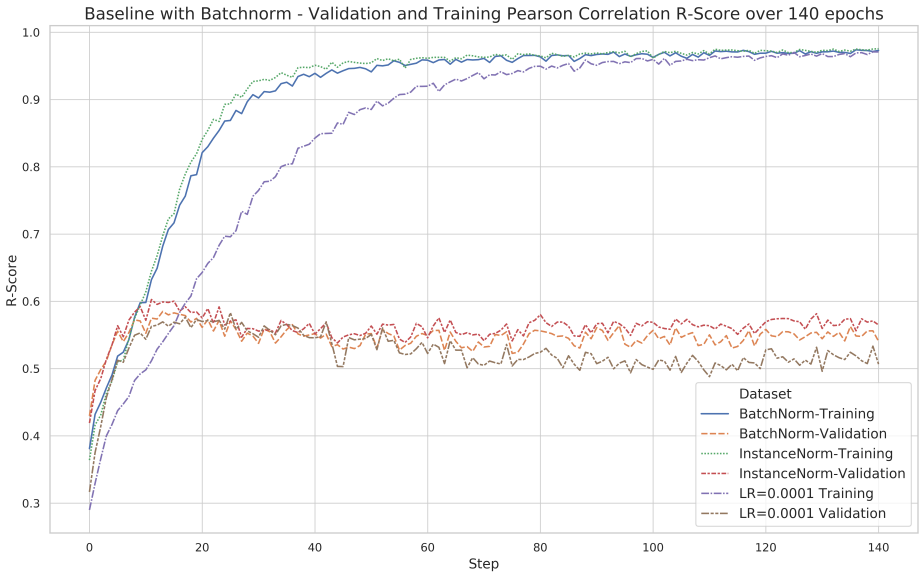


Figure 4.3: LigityScore1D Baseline-with-BatchNorm and InstanceNorm R-score Performance for the *Training* and *Validation* sets. The normalisation performance is also compared with a model with a higher learning rate. The training R-score converges after 40 epochs for both InstanceNorm and BatchNorm. In contrast, the baseline shown in Figure 4.1 does not converge even after 140 epochs.

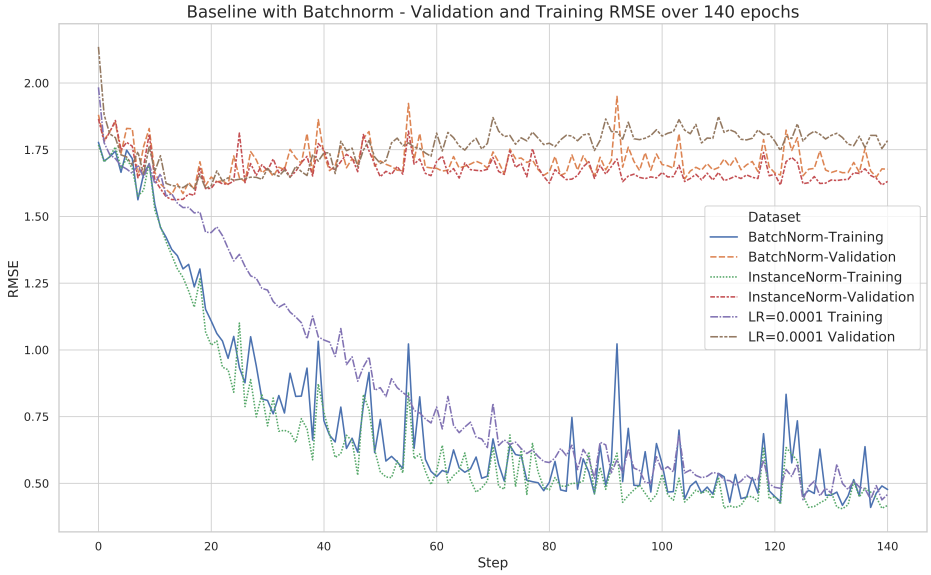


Figure 4.4: LigityScore1D Baseline-with-BatchNorm and InstanceNorm RMSE Performance for *Training* and *Validation* sets. Normalisation reduces the loss of the model more rapidly than with a ten-fold increased learning rate.

After several changes in the CNN parameters a different strategy was adopted to change the model representation. It seemed that with the baseline data representation no further learning was possible, and thus a different data representation was required. A considerable improvement in prediction performance was achieved when different PIP threshold factors are used. A higher PIP threshold factor implies that pharmacophoric *HotSpots* that are further apart are considered during the LigityScore PIP generation, as is described in Section 3.5. Figure 4.5 shows the validation R-score performance as the PIP threshold factor is incremented from the baseline 1.0 to 1.6. The PIP threshold factor of 1.4 (brown line graph) reaches a validation R-score close to 0.69 recording an improvement of 19% from the baseline (blue line). The PIP threshold factor of 1.4 clearly outperforms the baseline and the other models in terms of R-Score for the validation set.

Other changes in data representation were also considered such as increasing the maximum distance allowed to generation HotSpots (Exp #25-26), and using 0.5 resolution for the discritised space of the LigityScore representation (Exp #27). However these did not show improvements in results and were not considered for further experiments. Tests using drug-like ligands that conform with the Lipinski rule of five were also implemented, however such filtering also did not show any improvements in the results.

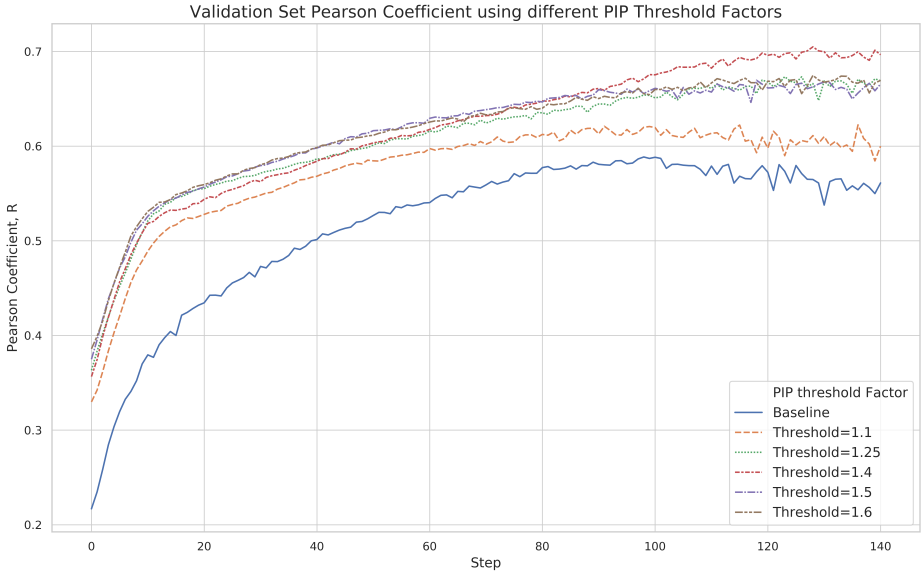


Figure 4.5: Validation Set performance using different PIP thresholds. The R-Score performance increases as the PIP threshold increases and achieves best results with a threshold factor of 1.4.

Subsequent tests (Exp #28-40) were done to find if changes using different combination of CNN parameters can further enhance the model performance. Apart from instance norm and higher PIP generation threshold factors, spatial dropout was found to be effective to increase the generalisation of the model and hence its performance. Positive results were achieved in Exp #37 where the InstanceNorm, PIP threshold factor of 1.4, and spatial dropout of 0.1 were combined and achieved the best performance R-score of 0.725 for CASF-2016 and 0.695 for CASF-2013 test sets. Spatial dropout was applied after the second convolution layers (middle layer with 128 channels) similar to the usage by it authors in Tompson et al. (2015).

The RMSE and R-Score plot for Exp #37 are illustrated in Figures 4.6 and 4.7. Figure 4.6 shows that the model is not overfitting for the duration of the training. Although the RMSE error for validation (orange) varies and spikes throughout training, overall it continues to slowly decrease. In fact the best validation result is achieved at the 139^{th} epoch. The validation R-score in Figure 4.7 (orange line) also has a very slow increase throughout training. Therefore, it seems that with the introduction of spatial dropout in the middle convolution layer, the model was made more resilient to overfitting. The training loss for the model is illustrated in Figure 4.8.

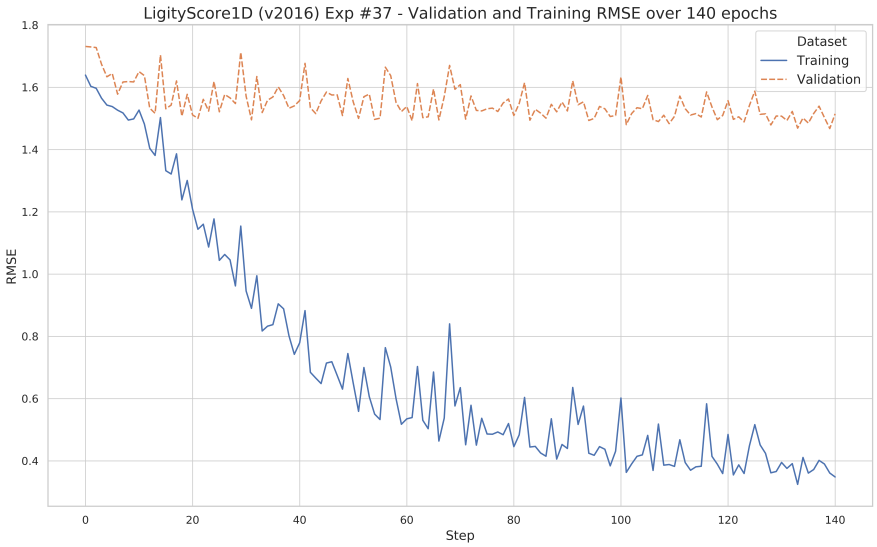


Figure 4.6: LigityScore1D (v2016) best performing RMSE training and validation plots. The validation error continued to slowly decrease over the training epoch, which is a sign that the model is more resilient to overfitting. The least RMSE error was recorded at epoch 139.

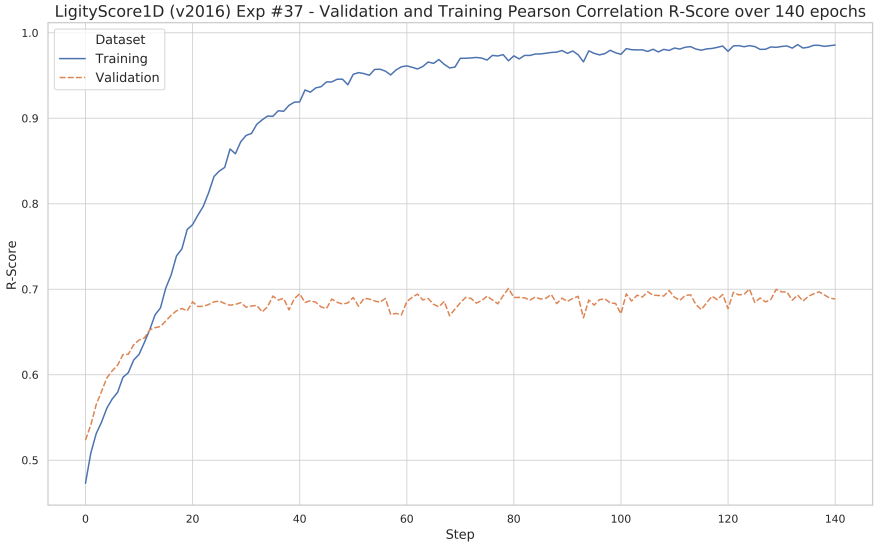


Figure 4.7: LigityScore1D (v2016) best performing R-Score training and validation plots. The R-Score for validation continues to slowly increase over the training, in line the decrease in validation RMSE error highlighted in Figure 4.6.

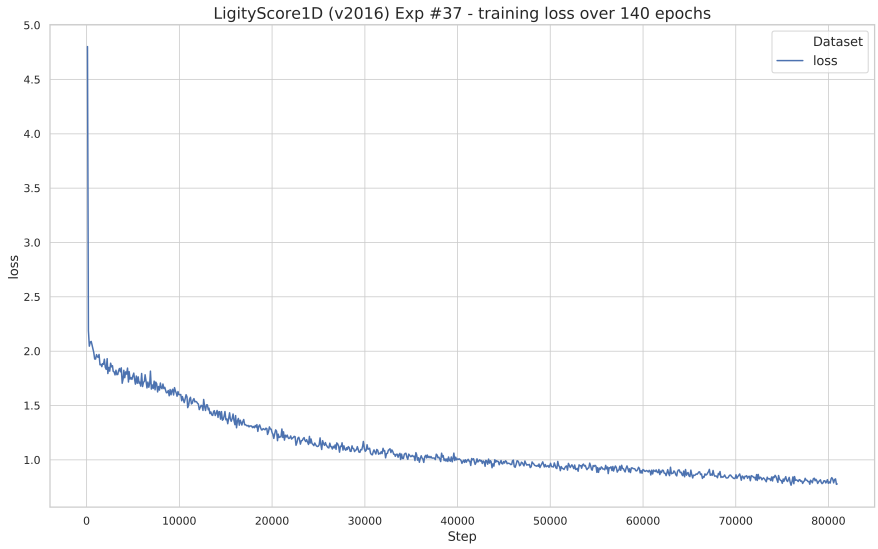


Figure 4.8: LigityScore1D (v2016) best performing training loss computed at 100 mini-batches intervals over 140 epochs.

The real and predicted binding affinity distributions using the model of Exp #37, for all datasets are depicted in Figures 4.9 and 4.10 respectively. These two plot highlight the similarity between the real and predicted results.

Another method to show the similarity of results is through scatter plots for the predicted affinity, versus the experimental (real) affinity. The ideal model would produce a plot of function $y = x$, as the predicted affinity should be equal to the real. This imposes an underlying linear model, which can be trustly evaluated using Pearson coefficient since this correlation measure assumes a linear model. The scatter plots for all sets for the model of Exp #37 are shown in Figure 4.11. The *training* scatter plot (top-left) shows a function very close to $y = x$ as it achieved an R-score of 0.985. The bottom scatter plots represent the Core-2016 (left) and the Core-2013 (right) sets showing good correlation between predicted and real affinities.

Further tests were done to change the depth of the convolution layers and the number of channels used in each layers. These experiments include Exp #45-49. Exp #45 uses an additional 512 convolution layer and also showed some promising results. The model achieved an R score fo 0.71 for the core-2013 set. This only varies from the chosen best model by 0.01 which is too small to claim it as a better result. Such small variation are easily attributed to changes in the model due to random initialisations across parame-

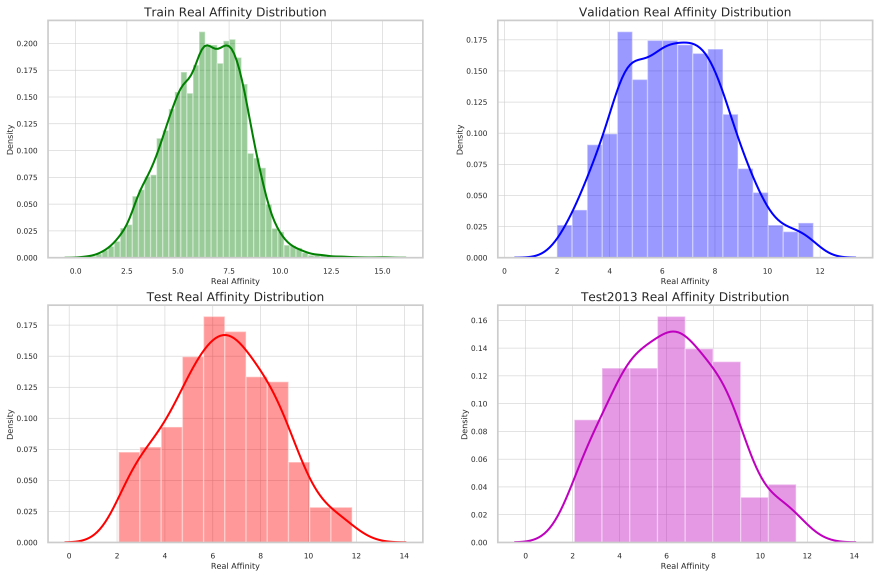


Figure 4.9: PDBBind v2016 experimental binding affinity distribution grouped by Training, Validation, and Test sets.

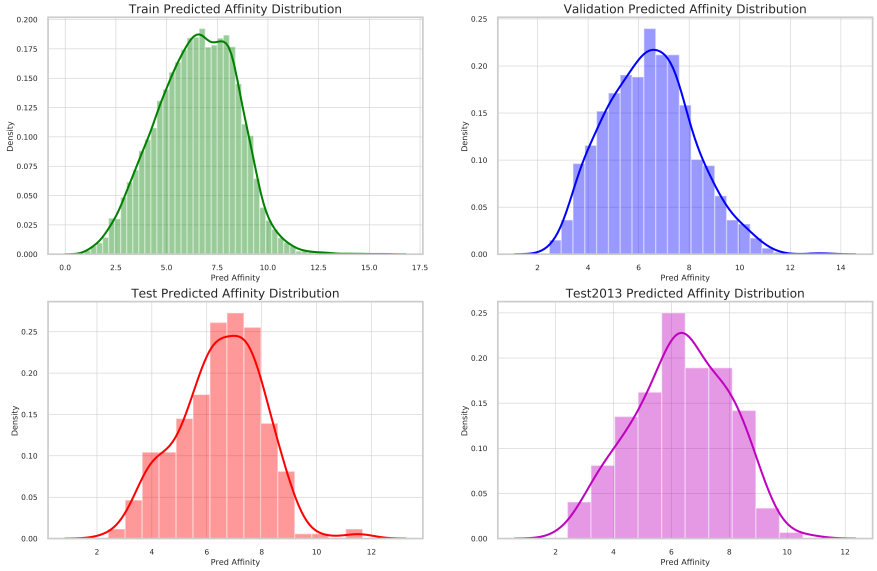


Figure 4.10: PDBBind v2016 predicted binding affinity distribution grouped by Training, Validation, and Test sets, for the best performing LigityScore1D Model (Exp #37).

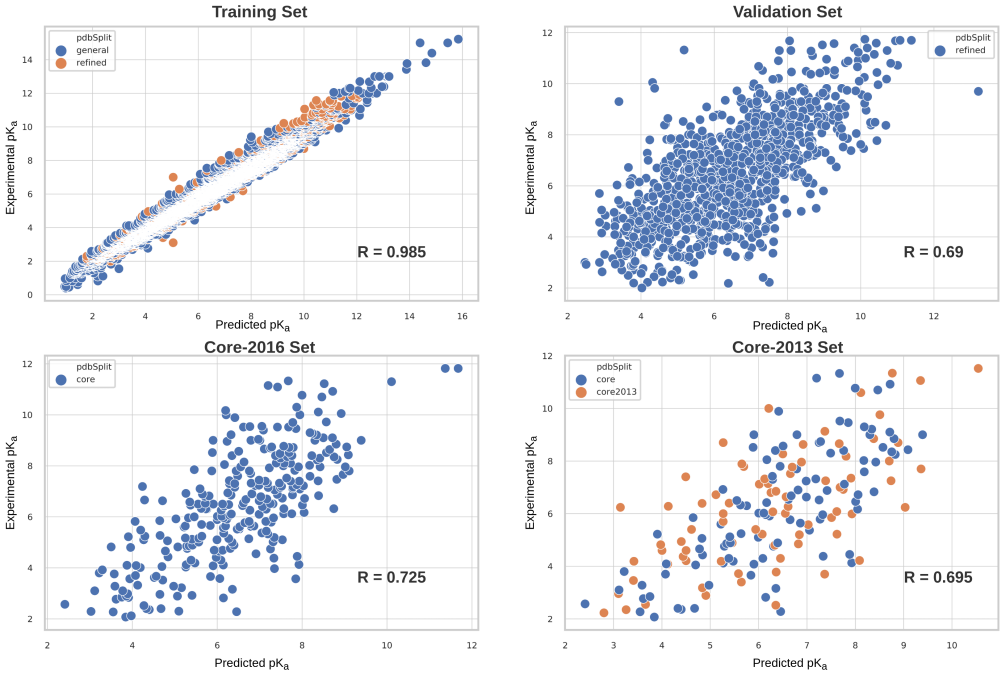


Figure 4.11: Experiment Vs Predicted Binding Affinity in pK_a (negative log of disassocation contant, IC50, and inhibition constant) for best LigityScore1D (v2016) model (Exp #37). The top left plots illustrates the optimal predication rate for the training set. The top right represented the validation set taken from the refined set. The bottom plot are the core set v2016 (left), and the core set v2013 (right).

ters of the model, and also difference in training or validation sets. On the other hand the R-score for core-2016 set is 0.696 (best model 0.725). Additionally the model of Exp #37 was chosen as it is a simpler model with fewer trainable parameters. Exp #46 includes five convolution layers, whilst Exp #47 and 48 include six and eight layers respectively. Exp #46 adds the *32-channel* convolution filter, whilst Exp #47-48 double the convolution layer for each channel depth. Finally, the VGG16 (Simonyan and Zisserman, 2014) architecture was also tested, that was modified to include less pooling layers, due to a small input, and to include spatial dropout. However, these more complex architectures did not improve the performance over those achieved in Exp #37.

The v2018 includes approximately 2,700 new protein-ligand complexes over the v2016. Therefore the v2018 dataset was used to verify if additional data applied during training would result in better performance. The same approach as that just outlined for v2016 as although almost the same CNN configuration was used, different datasets might have

different optimal parameters. The results were presented in Table 4.3. The baseline for v2018 was the same as v2016 except for the learning rate parameter (Table 4.1). A high learning rate of 0.001 was used as originally proposed by Kingma and Ba (2014). The learning rate to 0.00001 led to the problem of vanishing gradients and meaningful results were only achieved after several epochs, if at all. Therefore, a higher learning rate was used almost throughout all experiments with PDBbind v2018.

The same increase in performance was also registered when a larger PIP threshold factor was used. In this scenario the PIP threshold factor of 1.6 achieved the best validation score with an R value of 0.66. The difference between the validation R-score of PIP threshold factor 1.4 is only 0.016, and the test set results also show such small difference in performance (+0.1). Nonetheless the PIP threshold factor of 1.4 was also selected to be used for further tests, but the threshold factor of 1.6 was also included in some tests. In line with the results for the v2016, best results were also achieved for the model trained with InstanceNorm, and spatial dropout of 0.1 after the middle convolution layer, but with a PIP threshold factor of 1.6. The learning rate was also decreased from 0.001 to 0.0001 as this was causing oscillating changes in the loss of the network.

The additional training data has increased the results for the core-2016 test, but at the same time reduced the results for the core-2013 set. Therefore we cannot claim a significant improvement with the additional training data. The best models for LigityScore1D v2016 and v2018 were evaluated again over a number of iterations using different validation and training sets. The results are presented in Section 4.4. The test sets were not changed from CASF-2013 Li et al. (2014b) and CASF-2016 (Su et al., 2018) in order to evaluate their performance against these benchmarks.

4.3.2 | LigityScore3D

The LigityScore3D was developed as an enhancement to LigityScore1D where a more complex 3D representation of the protein-ligand complex is used as was detailed in Section 3.6. The results obtained for LigityScore3D are presented in Table 4.4, and a more detailed description on the experiments is provided in Table C.2.

From the results obtained from LigityScore1D, we have decided to use normalisation (InstanceNorm or BatchNorm) for all experiments, except the baseline, due to the noticeable improvement in performance and training time. We have taken an approach similar to the previous two sets of experiments reported in Tables 4.2 and 4.3 for LigityScore1D. In Exp #2 and #3 normalisation is introduced and a clear enhancement in performance is achieved in this case as well, where the R-score for Core-2016 test set shows a difference of approximately of 0.11 from the baseline. The enhancement pro-

vided by BatchNorm and InstanceNorm is very similar and inline with the results of LigityScore1D, and so we decided to use InstanceNorm as the main normalisation function. However, BatchNorm is also tested in later tests to validate this assumption such as experiment Exp #21. All the experiments were run for 60 epochs, apart from the baseline model which was run for 100 epochs since it takes longer to converge. Exp #31-34 were run for 30 epochs due to the cost of using the *Conv3D* module, described later on in this section.

In parallel with LigityScore1D, the increase in performance from the baseline was obtained by using different PIP threshold factors. Figure 4.12 illustrates the validation set R-Score performance for Exp #14-18, using different PIP threshold factors from 1.1 to 1.6.

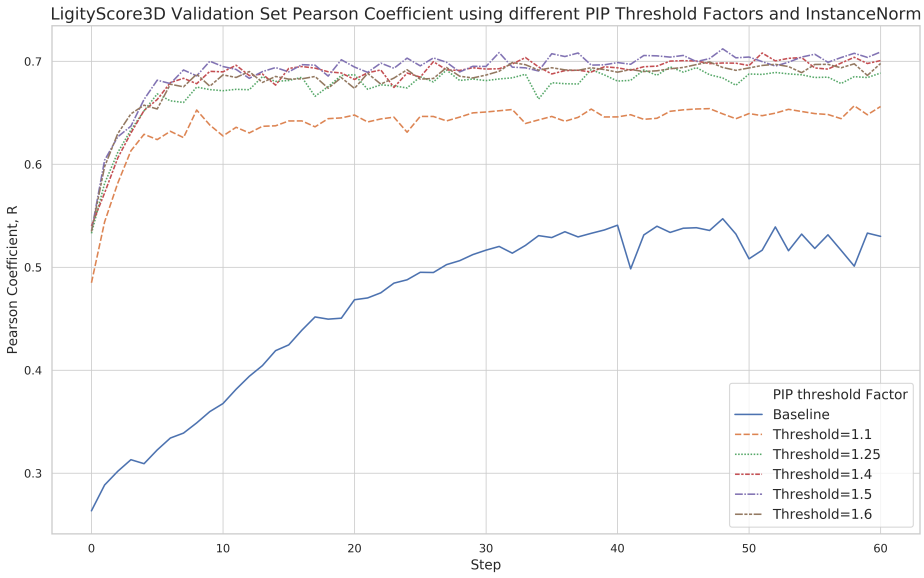


Figure 4.12: Validation Set Performance using different PIP thresholds and InstanceNorm. The R-Score performance increases considerably over the baseline as the PIP threshold increases. LigityScore3D achieves similar results for threshold factors of 1.4, 1.5, and 1.6.

Figure 4.12 clearly illustrate the enhancement on the validation set R-score over the baseline for threshold factors greater than 1.1 and achieve an R-score that are close to 0.7. Experiment #16 shows good results that outperform LigityScore1D with a Core-2016 R-Score of 0.726, and a surprisingly similar R-Score of 0.725 for Core-2013. Throughout the literature such as Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018), lower

performance was achieved on the Core-2013 set. This is also apparent in most of the other tests performed in this study. Experiments #17 and #18 also achieved a good result for Core-2016 and recorded a performance of 0.75 and 0.757 respectively, however they have lower Core-2013 performance at 0.65 which is similar to the performance of LigityScore1D. Due to the similarity between the results obtained for both test sets in Exp #16, further experiments were performed using the PIP threshold factor of 1.4.

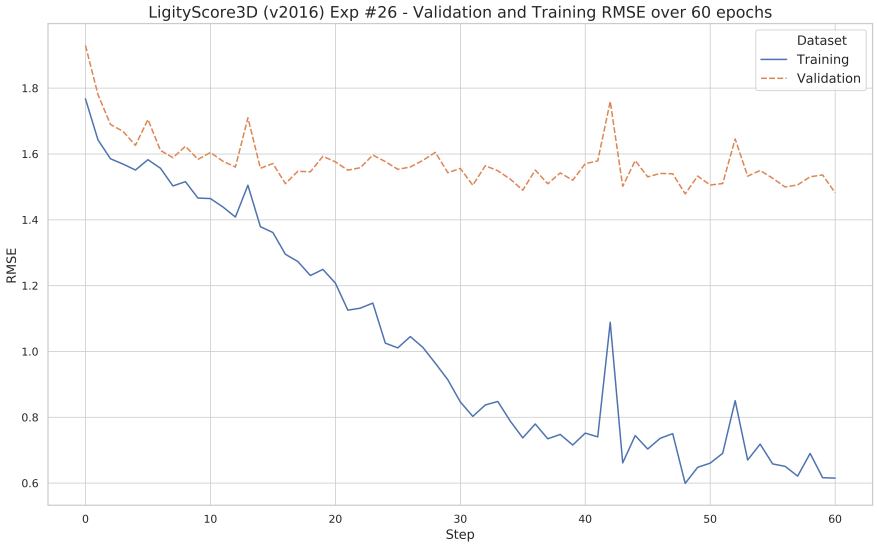


Figure 4.13: LigityScore3D (v2016) best performing RMSE training and validation plots. The validation error continued to slowly decrease over the training epoch, which is a sign that the model is more resilient to overfitting. The least RMSE error was recorded at epoch 48.

The next set of experiments builds on the knowledge acquired from LigityScore1D to enhance the LigityScore3D model using mainly spatial dropout. The best results where achieved with spatial dropout with a dropout probability of 0.2 on all the convolution layers and obtained a prediction performance on the Core-2016 R-score of **0.739**, and a Core-2013 R-Score of **0.745**. The best performance was achieved in Exp #26 and is highlighted in green in Table 4.4. This model is the same used in Exp #14 but with added spatial dropout components.

The additional scoring power results from LigityScore3D comes at the expense of a more complex network. The LigityScore3D model in Exp #26 has 94M learnable parameters, whilst the best model for LigityScore1D (Exp #37) has only 3.9M parameters. This represents a 24 times increase in learnable parameters. This also shows in the train-

ing time required. LigityScore3D took 133 minutes for 60 epochs, with the best epoch recorded in epoch 48, whilst LigityScore1D took 76 minutes for 140 epochs with the best epoch recording in step 139. Therefore, training LigityScore3D is five times slower than LigityScore1D. The timing reported in the Tables 4.2, 4.3, and 4.4 also include the evaluation of the model at each epoch for the training and validation sets.

Figure 4.13 shows the RMSE for the training and validation set for Exp #6. Similar to the results shown in Figure 4.6, Figure 4.13 shows that overall the validation error continues to gradually decrease during training, showing its resiliency to overfitting. The best validation error is recorded at epoch 48. Adding additional dropout at the FC connected layers does not improve the generalisation results as shown in Exp #22, but rather simply increase the convergence time of the CNN due to slower training. The scatter plots for the training, validation, and test sets are presented in Figure 4.14 showing strong correlations between the real and predicted binding affinity, confirming our model can correctly predict the binding affinity for protein ligand complexes.

The LigityScore3D model was also tested using the Conv3d PyTorch module. The Conv3d module allows training of our model using a 4D tensor of dimension $1 \times 54 \times 98 \times 98$, which is essentially the LigityScore3D representation with one input channel. The previous experiments (Exp #1-30) used the Conv2d PyTorch module using a 3D tensor of size $54 \times 98 \times 98$ which can be seen as a 2D protein-ligand complex input representation with 54 input channels. Experiments #31-34 utilise the Conv3D module using a combination of parameters that showed an increase in performance. Although these models do achieve relatively good results, they do no outperform the model in Exp #26. The Conv3d module further increases the complexity of the model as it requires additional training parameters due to the additional input tensor. Experiment #33 uses the same architecture as Exp #26 but uses the Conv3D module, and has a total of approximately 133M learnable parameters, or 42% increase over Exp #26. This model is also heavy on the compute side where each model's training took approximately 15 hours to complete 30 epochs, a drastic increase over the 2.22 hours required to train Exp #26 on 60 epochs.

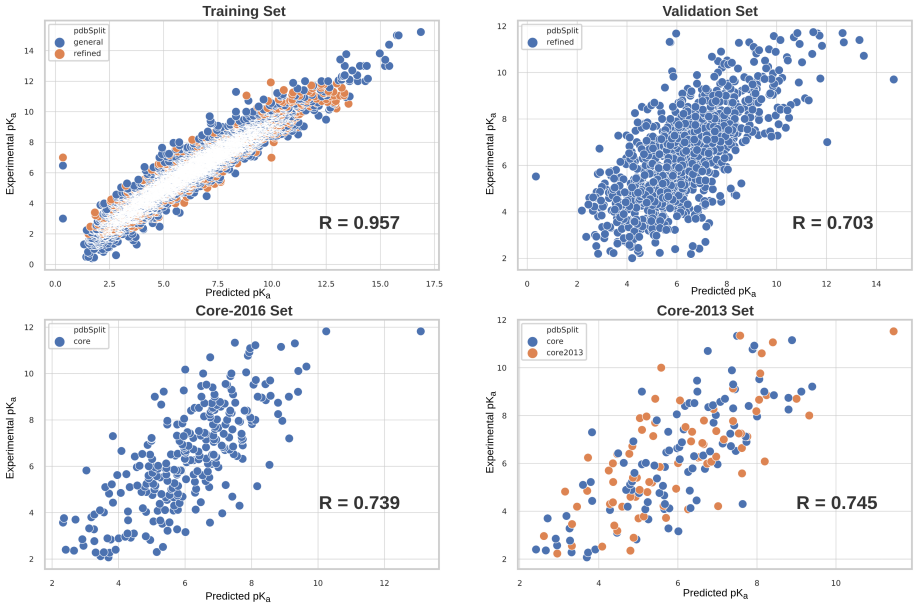


Figure 4.14: Experiment Vs Predicted Binding Affinity in pK_a (negative log of disassociation contant, IC50, and inhibition constant) for best LigityScore3D (v2016) model. The top left plots illustrates the optimal predication rate for the training set. The top right represented the validation set taken from the refined set. The bottom plot are the core set v2016 (left), and the core set v2013 (right).

Table 4.4: Performance of LigityScore 3D trained on PDBbind v2016. The best results are recorded for Exp #26 (ID 306) highlighted in green. An R value of **0.739** and **0.745** were recorded for CASF2013 and CASF-2016. Experiment details are listed in Appendix C Table C.2. R_{tr}, R_v, R_{2013}, and R_{2016} represent the Pearson correlation for the training, validation, CASF-2013, and CASF-2016 respectively.

Exp#	*ExpID*	*ExpShorthand*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_v$	R_v	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
0	300	Baseline	213.18	1.508	0.596	1.657	0.541	1.899	0.515	2.368	0.436
1	301	BatchNorm (BN)	131.21	0.431	0.972	1.562	0.596	1.694	0.631	1.972	0.548
2	302	InstanceNorm (IN)	132.56	0.443	0.970	1.591	0.582	1.721	0.626	1.978	0.515
3	303	IN–mini-batch 25	121.31	0.423	0.972	1.597	0.581	1.751	0.616	1.997	0.571
4	304	IN–mini-batch 15	139.81	0.457	0.97	1.546	0.609	1.702	0.626	1.893	0.591
5	305	IN–mini-batch 10	169.31	0.421	0.972	1.538	0.597	1.728	0.593	1.856	0.545
6	306	IN-no-padding	107.30	0.442	0.972	1.589	0.583	1.730	0.625	1.921	0.526
7	307	IN–rotations6 (r6)	501.37	0.488	0.966	1.602	0.567	1.790	0.571	2.070	0.506
8	308	IN–LR 0.00005	131.17	0.531	0.960	1.575	0.596	1.716	0.609	1.999	0.519
9	309	IN–LR 0.0001	131.36	1.052	0.826	1.631	0.561	1.757	0.594	1.963	0.500
10	310	IN–Dropout 0.8 0.8 0.5	131.59	1.148	0.960	1.925	0.572	2.115	0.606	2.151	0.548
11	311	IN–FC dim - 4000, 500, 200, 1	119.53	0.420	0.974	1.565	0.598	1.742	0.600	2.217	0.534
12	312	IN–FC dim - 6000, 2000, 500, 200, 1	129.62	0.424	0.973	1.574	0.595	1.730	0.608	2.130	0.584
13	313	IN–kernel-3x3–no-padding	117.30	0.412	0.974	1.604	0.572	1.783	0.573	2.227	0.458
14	314	IN–PIP Threshold 1.1	131.94	0.424	0.974	1.463	0.654	1.779	0.566	2.002	0.54
15	315	IN–PIP Threshold 1.25	131.98	0.417	0.978	1.394	0.695	1.614	0.701	1.749	0.692
16	316	IN–PIP Threshold 1.4	132.50	0.376	0.980	1.422	0.698	1.537	0.726	1.717	0.725
17	317	IN–PIP Threshold 1.5	132.98	0.391	0.976	1.381	0.708	1.501	0.750	1.793	0.651
18	318	IN–PIP Threshold 1.6	132.81	0.359	0.981	1.409	0.697	1.472	0.757	1.793	0.656
19	319	IN–PIP Threshold 1.4 w/Lipinski	71.24	0.379	0.98	1.385	0.662	1.665	0.685	1.659	0.642
20	320	IN–PIP Threshold 1.5 w/Lipinski	71.60	0.392	0.977	1.433	0.653	1.652	0.682	1.636	0.654
21	321	BN–PIP Threshold 1.4	130.72	0.376	0.980	1.414	0.705	1.530	0.720	1.646	0.716
22	322	IN–1-4–Drop0-8	131.78	1.330	0.962	1.953	0.688	2.115	0.703	2.212	0.675
23	323	IN–1-4–BS15	138.75	0.372	0.979	1.418	0.703	1.543	0.730	1.668	0.712
24	324	IN–1-4–r6	504.70	0.413	0.975	1.456	0.680	1.547	0.714	1.758	0.698
25	325	IN–1-4–Cdrop0-1-all	133.37	0.546	0.979	1.454	0.703	1.482	0.725	1.650	0.722
26	**326**	**IN–1-4-Cdrop0-2-all**	**133.17**	**0.599**	**0.957**	**1.479**	**0.703**	**1.485**	**0.739**	**1.541**	**0.745**
27	327	IN–1-4–Cdrop0-1-mid	134.27	0.394	0.978	1.432	0.697	1.544	0.722	1.725	0.702
28	328	IN–1-6–Cdrop0-1-mid	133.78	0.36	0.981	1.394	0.706	1.438	0.754	1.712	0.671
29	329	IN–1-4–Cdrop0-2-mid	133.00	0.397	0.979	1.428	0.697	1.491	0.738	1.703	0.693
30	330	IN–1-4–Cdrop0-1-all-r6	507.50	0.4	0.98	1.442	0.69	1.532	0.708	1.708	0.688

(continued...)

Exp#	*ExpID*	*details*	*time*	$RMSE_{tr}$	R_{tr}	$RMSE_{v}$	R_{v}	$RMSE_{2016}$	R_{2016}	$RMSE_{2013}$	R_{2013}
31	331	Conv3D–IN–1-4	896.05	0.785	0.912	1.422	0.702	1.529	0.710	1.799	0.657
32	332	Conv3D–IN–1-4-Cdrop0-1-mid	898.57	0.551	0.956	1.410	0.713	1.533	0.728	1.839	0.683
33	333	Conv3D–IN–1-4–Cdrop0-2-all	890.03	1.010	0.854	1.450	0.694	1.639	0.682	1.929	0.650
34	334	Conv3d–IN–1-4–Cdrop0-2-all-BS15	937.86	0.807	0.902	1.448	0.695	1.502	0.721	1.717	0.686

4.4 | LigityScore Best Performance Results

Table 4.5 summarises the results of the three models presented in previous sections. Each of the best performing method highlighted previously was run ten times using different validation sets, to remove any bias that might be caused from testing using a single validation set. Each validation set consists of 1,000 randomly sampled protein-ligand complexes from the PDBbind refined set. Throughout all experiments, the test sets were kept to the complexes outlined in the CASF-2013 and CASF-2016 for evaluation purposes with the CASF benchmark and other literature. The rest of the protein-ligand complexes were used for training. Therefore each different validation set variation also represents a slightly different training set. The results in Table 4.5 represent the mean results over the 10 tests with their respective standard deviation. As can be noted from Table 4.5 the models have a low standard deviation from the mean across the different metrics for each of the datasets used.

LigityScore3D shows a significant performance improvement for the Core-2013 model with an average of 0.71 R-score that is well above the 0.66 and 0.63 achieved for LigityScore1D (v2016) and LigityScore1D (v2018) respectively. On the other hand the results for Core-2016 for LigityScore3D shows comparable performance to the LigityScore1D (v2018) models with only 0.01 difference in R-score. Each LigityScore3D model takes considerable time for training, and due to limited time and resources it was not tested on the PDBbind v2018 dataset. For these reasons the best performing model is LigityScore3D with an R-score of 0.725 and 0.713 for Core-2016 and Core-2013 test sets respectively.

Throughout the experiments described in Tables 4.2, 4.3, and 4.4 a number of techniques were introduced to reduce the effect of overfitting. These include:

- **Normalisation (BatchNorm or InstanceNorm)**. The normalisation alogrithms introduce some noise when calculating the estimate of mean and variance from the mini-batch sample. This estimate introduces additional noise in the model that acts as a regularising parameter. In fact, Ioffe and Szegedy (2015) detail that when normalisation is applied, dropout probability rate can be reduced.

- **Data Augmentation**. Two main forms for data augmentation where applied. In the first method artificial samples were created with rotated representation of the original input. This method of artificial data augmentation did not enhance LigityScore performance. The second data augmentation method dealt with adding new protein-ligand complexes from the latest PDBbind versions. As reported in

Table 4.5, improved results (+0.04) were seen for Core-2016 test set, whilst less accurate results (-0.03) were obtained for Core-2013 test set.

- **Dropout**. Dropout was applied in all baselines with a drop probability of 0.5. Experiments to increase dropout were performed that showed longer convergence times and no prediction improvements.
- **Spatial Dropout**. Spatial Dropout was added at the convolution layer and results showed that this method help improve the generalisation of the models where the models did not overfit for the throughout the training epochs as was shown in Figure 4.6 and 4.13.
- **L2 Regularisation**. L2 regularation was added as part of the Adam optimisation, that would minimize unimportant features in model.
- **Reduction in Network Complexity**. A number of experiments were done to determine if less complex models would improve network performance. Different number of convolution layers, and also different number and sizes of fully connected layers were tested. Simpler model that provided similar performance to more complex ones were preferred in the selection of model parameters. In LigityScore1D thre convolutation layers and three fully connected layers are used, whilst in LigityScore3D four convolution layers, and four fully connected layers are used since the latter has a larger input dimension.

Table 4.5: Performance of LigityScore1D when trained with PDBbind v2016 and v2018, and LigityScore3D trained with PDBbind v2016, showing average and standard deviation for 10 tests using different validations sets from the refined set. Prediction performance for training, validation, Core-2016, and Core-2013 are listed.

	$RMSE_{tr}$	MAE_{tr}	STD_{tr}	R_{tr}	$RMSE_{v}$	MAE_{v}	STD_{v}	R_{v}	$RMSE_{2016}$	MAE_{2016}	STD_{2016}	R_{2016}	$RMSE_{2013}$	MAE_{2013}	STD_{2013}	R_{2013}
	LigityScore1D (v2016)															
mean	0.406	0.323	0.393	0.974	1.438	1.144	1.432	0.698	1.556	1.234	1.555	**0.699**	1.861	1.485	1.701	**0.657**
std ±	0.151	0.118	0.157	0.027	0.038	0.031	0.032	0.020	0.039	0.031	0.038	0.018	0.076	0.051	0.042	0.021
	LigityScore1D(v2018)															
mean	0.964	0.764	0.947	0.845	1.447	1.158	1.436	0.684	1.516	1.223	1.461	**0.741**	1.831	1.472	1.743	**0.635**
std ±	0.295	0.237	0.287	0.076	0.037	0.033	0.029	0.017	0.066	0.058	0.038	0.016	0.072	0.064	0.054	0.028
	LigityScore3D (v2016)															
mean	0.621	0.490	0.531	0.957	1.479	1.182	1.435	0.692	1.509	1.224	1.497	**0.725**	1.676	1.335	1.583	**0.713**
std ±	0.077	0.059	0.116	0.021	0.020	0.013	0.021	0.009	0.034	0.031	0.034	0.015	0.050	0.040	0.044	0.019

4.5 | Evaluation

In the previous sections we have presented the results and discussion on the experiments done for LigityScore. In order to gauge the effectiveness of our work, LigityScore needs to be benchmarked with other scoring functions that use a similar methodology. The following evaluations will be presented in the next section.

- LigityScore will be evaluated against the Pafnucy (Stepniewska-Dziubinska et al., 2017) model that was used to build our baseline. LigityScore results will be compared with our implementation of Pafnucy as well as the implementation using the authors' published scripts.
- LigityScore will be evaluated using the CASF scoring power benchmark using the test sets listed in CASF-2013 (Li et al., 2014a) and CASF-2016 (Su et al., 2018). Apart from specifying the method and metrics used for evaluation, the CASF benchmarks also include a list of 29 and 34 scoring functions respectively, together with their results, permitting a direct comparison against their performance.
- LigityScore will be evaluated against other recent literature that utilize ML and DL techniques, that were also evaluated by either the CASF-2013 or the CASF-2016 benchmarks. To the best of our knowledge a comprehensive list of the scoring functions utilizing ML and DL techniques for the last four years are listed in Tables 4.7 and 4.8 to indicate the ranking of LigityScore with respect to these methods and also to those within the CASF benchmarks.

The metrics used for evaluation with the CASF benchmark include the Pearson correlation coefficient, R, and the standard deviation in linear regression, SD. Other metrics included for our evaluation include the RMSE and MAE, that although they are not part of the CASF benchmark that are used in various literature (as highlighted in Tables 4.7 and 4.8) and therefore will facilitate the comparison of LigityScore to these methods. These evaluation metrics are detailed in Section 2.3.

The CASF-2013 (core set 2013) and CASF-2016 (core set 2016) contain 195 and 285 protein-ligand complexes respectively. Due to some parsing errors returned by RDKit when loading some of the molecules, and also due to the fact that not all protein-ligand complexes have PIPs that are conformant with the distance thresholds during feature extraction, some complexes were excluding from the testing set keeping 182 and 279 complexes for the CASF-2013 and CASF-2016 respectively. The RDKit errors were mainly generated due to invalid atomic bonds. Boyles et al. (2020) also mention molecule parsing issues when using OpenBabel and RDKit. As described in Section 3.3.1 we have

attempted to fix these errors using a different file format, and also by downloading the structure directly from PDB. Such complexes exclusions are also present in other scoring functions defined in CASF benchmarks, and the recommendation by the authors for the scoring power computation is to simply remove them (Li et al., 2014a). Therefore, all quoted results for LigityScore use the 182 and the 279 complexes for CASF-2013 and CASF-2016 respectively.

4.5.1 | Evaluation with Pafnucy

Table 4.6 illustrates a comparison of the LigityScore results with Pafnucy for the core-2016 set. Two variants of Pafnucy are highlighted, where the first model was run using the authors' provided scripts, whilst the second model was run using our own implementation scripts as is described in Section 3.2.

As can be noted there is a significant difference in the training time per epoch from the LigityScore models to the Pafnucy models. The original Pafnucy model has a training time per epoch of 34.7 minutes which is approximately 16 times more than LigityScore3D, and 43 times more than LigityScore1D. This difference is mainly attributed to the use of the *Conv3d* convolution module. LigityScore3D was also tested using the Conv3d convolution module and as reported in Table 4.4, this module increases the model complexity due to higher multi-dimensional convolutions computations and also increases the number of learnable parameters. Our results also show an increase in training time when using the Conv3d module, and the LigityScore3D model with Conv3d layers requires approximately 30 minutes per epoch. It is important to note that our implementation includes a model evaluation at each epoch for the training and validation sets, so if these are removed the actual training time required is less. This is also one of the main reasons why our Pafnucy implementation has a higher training time than the original implementation.

Additionally, Pafnucy uses longer training cycles than LigityScore, as each epoch considers 24 different rotated representations of the input that simulate a rotationally invariant input. Therefore, although there are fewer learnable parameters than our LigityScore3D, the rotations and complex Conv3d computations increase training time. The larger number of learnable parameters are due to larger inputs to the FC layers and also due to wider FC layers.

LigityScore was trained using a mini-batch size of 20, whilst Pafnucy was trained using a mini-batch size of 5. Therefore, LigityScore performs less CNN backward passes and parameter updates which also contributes to the improved execution time. Therefore our models are computationally more efficient due to the simpler models, rotation-

Table 4.6: LigityScore Evaluation with the original Pafnucy Stepniewska-Dziubinska et al. (2017) and our implementation of the Pafnucy Model. The best results of LigityScore3D and 1D are compared with Pafnucy and show slightly lower prediction performance, and significant improvement in computational efficiency. E_T is the total epochs used in training, and E_B represents the epoch with the best validation set loss.

Scoring Function	**E_T**	**E_B**	**total train time /min**	**epoch train time /min**	**model params in M**	**RMSE**	**MAE**	**SD**	**R**
CASF-2016									
Pafnucy*	20	14	693	34.7	12.8	1.468	1.181	1.301	0.750
Pafnucy (our impl.)	20	11	874	43.7	12.8	1.388	1.153	1.411	0.761
LigityScore3D	60	48	134	2.2	94.3	1.509	1.224	1.497	0.725
LigityScore1D (v2018)	140	23	111	0.8	3.9	1.516	1.223	1.461	0.741

** calculated using Stepniewska-Dziubinska et al. (2017) provided scripts. Time calculation does not include model evaluation for training and validation sets after each epoch.*

ally invariant methodology, and use of larger mini-batch samples. All models listed in Table 4.6 were run on AWS EC2 g4dn instances as was described in Section 3.8.3.

Stepniewska-Dziubinska et al. (2017) report the Pafnucy R-score performance on Core-2016 at 0.78. Our replication results using the authors scripts, and our own implementation of Pafnucy produce similar results which are also close to that reported in Stepniewska-Dziubinska et al. (2017) at 0.750 and 0.761 respectively. This difference from the published results can be attributed to small changes used in our implementation and also different samples in the training and validation sets due to random sampling. Stepniewska-Dziubinska et al. (2017) do not mention if their results were computed using different randomly sampled validation and training sets.

As noted previously, the LigityScore models achieve a similar average prediction performance for the Core-2016. The results of the average LigityScore1D (trained on the 2018 PDBbind dataset) has an R-Score performance difference of 0.04 from the original results reported in Stepniewska-Dziubinska et al. (2017), and a difference of only 0.02 from our implementation of Pafnucy. Therefore we can conclude that LigityScore was shown to have a suitable protein-ligand representation that shows comparable performance in binding affinity predictions to the chosen baseline model, which is significantly more computationally efficient. LigityScore is evaluated with other models using the CASF-2013 and CASF-2016 benchmarks in the next section.

4.5.2 | CASF Scoring Power Benchmark

The Comparative Assessment of Scoring Functions (CASF) was designed as a benchmark specifically for scoring functions. CASF includes a number of power scores including scoring, ranking, docking, and screening which are used to evaluate and rank the scoring function in each area. In this study we focus on the scoring power benchmark. The CASF uses a specific set of high-quality complexes taken from the refined set of the PDBbind datasets. The terms Core-2016 test set and CASF-2016 test set are synonymous, and the year label represents the PBDbind versions used to extract the set. The same applies for the Core-2013 and CASF-2013. The core set details are described in Section 3.3, whilst the CASF benchmark is discussed in Section 2.3.1.

The scoring power uses the LigityScore predicted binding affinities for each protein-ligand complex in the test set to extract R and SD values. Equation 2.29 is used to compute R, whilst SD is computed by Equation 2.30 together with a linear regression model to fit the experimental and predicted affinity values in a linear model.

The ranking of LigityScore for CASF-2013 and CASF-2016 are presented in Tables 4.7 and 4.8. The scoring functions ranked directly as part of the CASF-2013 and CASF-2016 are listed in black, and their results are taken directly from Li et al. (2014a) and Su et al. (2018) respectively. All CASF reported results are listed in terms of R and SD using the former to rank the scoring function. The scoring functions listed in blue represent results reported as per literature reviewed in individual publications that utilise also the CASF to benchmark their work. Tables 4.7 and 4.8 thus provide to the best of our knowledge, a comprehensive list of the scoring functions developed in recent years to date, that compare and rank the different scoring functions available.

LigityScore3D, highlighted in green, achieved 5th place in the CASF-2013 benchmark with an average R-score of 0.713, and exceed the reported CASF-2013 score for Pafnucy. The 4th place is taken by the models of EIC-Score (Nguyen and Wei, 2019b) and PLEC-nn Wójcikowski et al. (2019) that achieve both an R-Score of 0.774.

The top performing *CNN* based model is in the OnionNet model developed by Zheng et al. (2019) where it achieves an R-Score of 0.782. The 2nd place was achieved by NNScore2.0 enhanced with RDKit features (Boyles et al., 2020), whilst AGL (Nguyen and Wei, 2019a) represents the best performing model to date with an R-Score of 0.792. It is important to note that the top five models have a performance difference range of only 0.018, whilst the difference of LigityScore3D from the AGL is 0.079. LigityScore1D achieved slightly worse performance but still outperformed the classical scoring function in terms of scoring power.

On the CASF-2016 benchmark, LigityScore model achieve 7th and 8th places. Ligi-

Table 4.7: LigityScore Evaluation on the **CASF-2013** Scoring Power benchmark. Our results are highlighted in green achieving **5th** and **8th** places out of the scoring functions listed in the CASF benchmark (black) and other literature that use the same benchmark (blue) published after Li et al. (2014a). Only the top 20 scoring function of the CASF-2013 benchmark are included in this list. Full table is available in the supplementary information of Li et al. (2014a). The SF are ranked using the Pearson Correlation, R.

Scoring Function	Scoring Power Rank	N	RMSE	SD	R
AGL (Nguyen and Wei, 2019a)	1	195	-	1.45	0.792
LearningFromLigand NNScore+RDKit (Boyles et al., 2020)	2	180	-	-	0.786
OnionNet (Zheng et al., 2019)	3	195	1.45	1.45	0.782
EIC-Score (Nguyen and Wei, 2019b)	4	195	-	-	0.774
PLEC-nn (Wójcikowski et al., 2019)	4	195	-	1.43	0.774
LigityScore3D	**5**	**182**	**1.68**	**1.58**	**0.713**
Pafnucy (Stepniewska-Dziubinska et al., 2017)	6	195	1.62	1.61	0.700
DeepBindRG (Zhang et al., 2019)	7	195	1.82	-	0.639
LigityScore1D	**8**	**182**	**1.83**	**1.74**	**0.635**
X-Score	9	195	-	1.77	0.622
X-ScoreHS	10	195	-	1.77	0.620
X-ScoreHM	11	195	-	1.78	0.614
X-ScoreHP	12	195	-	1.79	0.607
ΔSAS	13	195	-	1.79	0.606
ChemScore@SYBYL	14	195	-	1.82	0.592
ChemPLP@GOLD	15	195	-	1.84	0.579
PLP1@DS	16	195	-	1.86	0.568
PLP2@DS	17	195	-	1.87	0.558
GScore@SYBYL	18	195	-	1.87	0.558
ASP@GOLD	19	195	-	1.88	0.556
ASE@MOE	20	195	-	1.89	0.544
ChemScore@GOLD	21	189	-	1.90	0.536
DScore@SYBYL	22	195	-	1.92	0.526
Alpha-HB@MOE	23	195	-	1.94	0.511
LUDI3@DS	24	195	-	1.97	0.487
GoldScore@GOLD	25	189	-	1.97	0.483
Affinity-dG@MOE	26	195	-	1.98	0.482
LigScore2@DS	27	190	-	2.02	0.456
GlideScore-SP	28	169	-	2.03	0.452

Table 4.8: LigityScore Evaluation on the **CASF-2016** Scoring Power benchmark. Our results are highlighted in green achieving **7th** and **8th** places out of the scoring functions listed in the benchmark (black), and other literature that use same benchmark (blue) published after Su et al. (2018). Only the top 20 scoring function of the CASF-2016 benchmark are included in this list. Full table is available in the supplementary information of Su et al. (2018). The SF are ranked using the Pearson Coefficient, R.

Scoring Function	Scoring Power Rank	N	RMSE	SD	R
AGL (Nguyen and Wei, 2019a)	1	285	1.27	-	0.830
EIC-Score (Nguyen and Wei, 2019b)	2	285	-	-	0.826
LearningFromLigand	2	276	-	-	0.826
NNScore+RDkit (Boyles et al., 2020)					
K_{Deep}(Jiménez et al., 2018)	3	285	1.27	-	0.820
PLEC-nn (Wójcikowski et al., 2019)	4	285	-	1.26	0.817
OnionNet (Zheng et al., 2019)	5	285	1.28	1.26	0.816
$\Delta VinaRF20$	5	285	-	1.26	0.816
Pafnucy (Stepniewska-Dziubinska et al., 2017)	6	285	1.42	1.37	0.780
LigityScore1D	**7**	**279**	**1.52**	**1.46**	**0.741**
LigityScore3D	**8**	**279**	**1.51**	**1.50**	**0.725**
X-Score	9	285	-	1.69	0.631
X-ScoreHS	10	285	-	1.69	0.629
ΔSAS	11	285	-	1.70	0.625
X-ScoreHP	12	285	-	1.70	0.621
ASP@GOLD	13	282	-	1.71	0.617
ChemPLP@GOLD	14	281	-	1.72	0.614
X-ScoreHM	15	285	-	1.73	0.609
Autodock Vina	16	285	-	1.73	0.604
DrugScore2018	17	285	-	1.74	0.602
DrugScoreCSD	18	285	-	1.75	0.596
ASE@MOE	19	285	-	1.75	0.591
ChemScore@SYBYL	20	285	-	1.76	0.590
PLP1@DS	21	285	-	1.77	0.581
ChemScore@GOLD	22	279	-	1.78	0.574
G-Score@SYBYL	23	284	-	1.79	0.572
Alpha-HB@MOE	24	285	-	1.79	0.569
PLP2@DS	25	285	-	1.80	0.563
Affinity-dG@MOE	26	285	-	1.81	0.552
LigScore2@DS	27	285	-	1.83	0.540

tyScore1D achieving 7th place with a slightly higher performance than LigityScore3D with an average R-score performance 0.741. The Pafnucy model is in 6th place with a difference of 0.04 from ligityScore1D and is discussed in detail the previous section. The top seven scoring function report a very similar performance for the top five positions with a range difference of only 0.014, and a difference to LigityScore1D of 0.089.

Although our results do not make it to the top five in the CASF-2016 benchmark evaluation, the LigityScore protein-ligand representation is shown to be successful for use for binding affinity prediction. From the experiments carried out the performance gain for LigityScore representation was mostly achieved through changes in the data representation by increasing the PIP threshold factors, providing motivation to seek additional improvements to the LigityScore data model that can further enhance the prediction performance.

4.6 | Summary

In this chapter we have presented the results achieved for our two variants of LigityScore. Various experiments for each LigityScore version were carried out changing different parameters of the data model and also the CNN architecture. Noticeably the best results are achieved when using the InstanceNorm normalisation technique at the convolution layers, increasing the PIP threshold factors, and by applying a small spatial dropout at the convolution layers. The model with the best performance was further tested using 10 variations of training and validation sets to remove bias from the results tied to a particular training and validation set.

Our results have showed that our model is successful to be used as a suitable protein-ligand representation where it achieved an average performance on the Core-2013 that is marginally higher than the result reported in Pafnucy as our baseline study. On the other hand Pafnucy had slight better results for the Core-2016 set. Additionally we have also successfully used a number of techniques to enhance the performance of our CNN model achieving the two main aspects of our research question presented in Section 1.3.

We have also evaluated our model with the CASF-2013 and CASF-2016 benchmarks, and also with other recent literature that also used either of these benchmarks. LigityScore3D achieved the best performance in the CASF-2013 benchmark and ranks 5th place, whilst LigityScore1D trained on the PDBbind v2018 showed best performance in the CASF-2016 benchmark achieving 7th place.

5

Conclusions

In this study we explored the use of convolutional neural networks to develop a scoring function, called LigityScore for binding affinity prediction. Machine learning (ML) scoring functions (SF) have been developed to address the limitations of classical models, such as the use of linear models, imposed functional form, and their inability to learn from new data. However, ML based scoring functions still rely on a degree of feature engineering that requires expert knowledge to preprocess the data, which led to the introduction of deep learning methods. In this dissertation we use a representation to describe the protein-ligand complex extracted directly from its structural and interacting properties without relying of expert knowledge. This is then fed to a CNN for feature extraction and binding affinity prediction.

Chapter 1 provides an introduction to the virtual screening domain describing the motivation to enhance and search for new scoring functions. Scoring functions are the heart of structure based drug design, where they are used to estimate how strongly the docked pose of a ligand binds to the target. Therefore, seeking a scoring function that can accurately predict this binding affinity is key for successful virtual screening methods.

Chapter 2 gives comprehensive background information on scoring functions used for SBVS, and details the CNN architecture and various techniques that were developed in recent years to enhance its effectiveness and performance. Chapter 2 provides a literature overview of the scoring function methods, starting with the classical methods, followed by ML methods as better scoring functions, and then leading to deep learning (DL) methods that can perform automatic feature extraction.

Chapter 3 provides implementation details on the algorithms used to generate the LigityScore representation. The data pipeline used to train and test the model is also described in detail. The data representation process is split into two main components

where first, the pharmacophoric interactions points (PIPs) are extracted from the protein and ligand around the binding site, and secondly the PIPs are processed to generate a feature matrix or feature hypercube for LigityScore1D and LigityScore3D respectively, as is illustrated in Figures 3.9 and 3.10. Chapter 3 describes also the CNN architecture used for each model and discusses the experiments carried out to optimise the data representation and the CNN architecture.

Results and Evaluation are presented in Chapter 4. The experiments performed are described in detail, and a discussion on the results obtained is provided to highlight the strategy used to find the best performing models. The average performance of the best models are evaluated using 10 different randomly sampled validation sets with their corresponding training set, to reduce the bias introduced from a single validation and training set combination. The average performance is then evaluated with results from the CASF-2013, and CASF-2016 benchmarks.

5.1 | Achieved Aims and Objectives

The main focus of this study was to answer the following research question.

Can deep learning approaches be used in scoring functions to augment the predictive scoring power in SBVS techniques?

This research question was tackled in two aspects to improve the predictive scoring power. The first aspect dealt with finding a suitable representation of the protein-ligand complex that can be used as input to the CNN. A key aspect of this representation is that it is built without using expert drug design knowledge, but rather it utilises a more *simplistic* representation that is passed to the CNN to automatically extract the underlying complex representations. These CNN feature maps can then be used to distinguish the relationship between the protein and the bound ligand and estimate their affinity using a regression type output.

To this effect we have developed two different protein-ligand representations that are extracted directly from the 3D structure of both the protein and ligand using pharmacophoric features inspired by the Ligity method of Ebejer et al. (2019). The pharmacophoric features across both the protein and ligand are used to extract PIPs that need to conform to specific family types and distance thresholds so as to capture only the stronger interactions between the protein's and ligand's pharmacophoric feature. The two methods developed in this study include the LigityScore1D that uses the *single* dis-

tance between the protein PIP and the ligand PIP, hence the name LigityScore1D. Each distance for each PIP pair combination is discretised to increment binning counters for its corresponding pharmacophoric family-pair vector. The different pharmacophoric family pair combinations are stacked to construct a matrix representation of the complex. The other representation, termed LigityScore3D, uses a combination of 3-PIPs at a time to extract three different distances from the triangular structure formed by the 3-PIP combination. For the selection of the 3-PIPs combination, the cases where two PIPs are chosen from the protein PIP pool and one from the ligand PIP pool, and vice versa, are both considered. In each combination at least one of the three PIPs must be on the other molecule. This varies from the Ligity method where only combinations of ligand PIPs were considered. These distances are also discretised and are used as binning coordinates in a feature hypercube for each PIP family set.

The choice of representation of the protein-ligand structure determines the flexibility and expressiveness that the model is able to learn and ultimately its scoring power. Although DL methods extract features automatically during training, correct representation of the complex is fundamental for the feature extraction ability of the DL model. Various parameters relating the selection of PIPs were tested to seek out the best possible representation of the protein-ligand complex.

LigityScore representation tackled successfully one of the two aspects of our research question. The second aspect dealt with finding a suitable CNN model that can extract features from our representation and provide a successful prediction for binding affinity of the complex. A number of experiments were carried out using a random hyperparameter optimisation technique. The best results were achieved when using higher PIP threshold factors, InstanceNorm normalisation, and small spatial dropout at the convolution layers. The data representation component and the CNN architecture, tuned for the representation to provide the best prediction performance, together constitute the LigityScore scoring function.

In line with the objectives, LigityScore models were also evaluated on the latest two CASF benchmarks. The Pearson correlation coefficient, and the standard deviation in linear regression were used to compare LigityScore with the benchmark model, and also other models in literature published in recent years. The scoring functions from both the benchmark, and the literature reviewed were compiled in an updated list in order to provide a better understanding of LigityScore performance across a comprehensive list of SFs. The LigityScore3D has achieved better overall results and showed similar performance for the CASF-2016 and CASF-2013 benchmarks. LigityScore3D ranked 5^{th} place in the CASF-2013 benchmark, and 8^{th} in CASF-2016, with an average R-score performance of 0.713 and 0.725 respectively. LigityScore1D obtained best results when trained

using the PBDbind v2018 dataset, and ranked 8th place in the CASF-2013 and 7th place in CASF-2016 with an R-score performance of 0.635 and 0.741 respectively.

Therefore in summary, the objectives of this study have been met, and its major contribution is in the presentation of a novel protein-ligand representation for use as a scoring function for binding affinity prediction adapted from Ebejer et al. (2019). Since LigityScore is based on distances between pharmacophoric features, it also presents a rotationally invariant representation. Additionally, the method shows relatively good performance that exceed the Pafnucy performance on the CASF-2013 benchmark, using a less computationally complex model that can be trained 16 times faster. The LigityScore models can potentially be used for affinity predictions for novel molecules, as a scoring function for docking-based virtual screening, and also in structure based ligand discovery campaigns.

5.2 | Critique and Limitations

A very recent paper Shen et al. (2020) highlighted the importance of assessing the scoring function in all four powers (scoring, ranking, docking, and screening) of the CASF benchmark for a 360 degree performance evaluation. Shen et al. (2020) use the CASF benchmark to assess all the power metrics for 14 machine learning-based scoring functions.

Unfortunately due to time constraints for this project, and the recent release of Shen et al. (2020) it was not possible to extend evaluation of LigityScore on the rest of the powers. This is one of our limitations in the sense that these results are not known, and additional work would need to be done to extend LigityScore for testing the other powers in the CASF benchmark. This will be considered for future work. Recent literature for deep learning scoring functions also focused on only the scoring power aspect such as Stepniewska-Dziubinska et al. (2017), Jiménez et al. (2018), and Zheng et al. (2019), and therefore a similar approach was taken for our work. Shen et al. (2020) has shown that Stepniewska-Dziubinska et al. (2017) and Zheng et al. (2019) do not perform well for rest of the benchmark powers, and even report performance lower than the classical functions.

Although the ideal scoring function should perform well in all areas, we argue that this is not necessarily the case and that not every use case scenario needs to use the exact same scoring function model. A particular ML scoring function may not be suited for every scenario, and therefore a different version, trained for a particular scope, might be better. As an example, the screening power would require the scoring function model to

differentiate between actives and decoys. However, the models trained with the PDBbind dataset do not include any decoy information.

ML models, including DL, use learning by representation to extract the underlying function in the data. Therefore, if the dataset does not include the decoy class it is intuitive that the model might not respond well when presented with decoy information. To the best of our knowledge there are no available protein-molecule decoy data that include also their binding affinity. As this might not be possible to record it presents a challenge, as it presents a limitation that restricts the DL model's generalisation ability. One way to workaround this is to use decoys with a manual preset binding affinity set to zero or negative value to indicate its non-binding nature. However this approach would need to be validated. Therefore in this sense a ML model trained with decoys, might not be suited for screening, whilst a model trained on experimental data only, might be better suited for scoring power assessment. A ML SF model can be developed to cater for the particular requirements, leveraging on the flexibility they provide to adjust and derive their parameters from the given training data.

As was shown for the LigityScore1D experiments in Section 4.3.1, training on different versions of PDBbind datasets requires slight variations in the tuning of the CNN model. Therefore, the model might be sensitive to changes in training sets which would require different optimisation. However this can be considered as a more general limitation for ML models (Zheng et al., 2019). Due to limited computational resources we have not performed an exhaustive sweep on all the model parameters, and have used some defaults or values recommended in literature. Therefore it is not excluded that a better parameter set, different than those listed in Table 3.5, can provide a better performance. LigityScore is tested only on the CASF benchmarks. Additional test sets could be used to confirm the generalisation ability of the model.

In an effort to better explain the deep learning model, it would be beneficial to determine which PIP families pairs contribute most to the predicted output. One way to explore this is to determine their effect on the weights of the model. However, since the feature vectors or feature cubes, for LigityScore1D and LigityScore3D respectively, are stacked together and treated as one input, it is difficult to determine the variation in size and distributions of the weights with respect to each of the PIP's family sets. Another method would be to exclude certain PIP families from the representation during the processing of the feature generation. Each PIP family combination would be systematically excluded, or zeroed to determine the effect it has on the prediction performance when compared to the baseline model.

5.3 | Future Work

There are various opportunities for future work to overcome the limitations mentioned in the previous section, and also to enhance the LigityScore performance. As a first step, additional work is required to further develop our method so that it can be benchmarked on all the powers in the CASF benchmark. This avenue might also lead to developing variants of LigityScore that would be optimised for specific tasks, such as screening power. In this case, experiments could be done to include decoy protein-ligand complexes with manually set, poor affinity values, so that they can be included in the training of the CNN.

Subsequent steps for future work would involve optimising the LigityScore performance. The starting point to improve the LigityScore performance should be to focus on the data representation component of the protein-ligand complex. Further research would be required into how to build on the existing respresentation, and possibly seek ways to incorporate alternative types of features within LigityScore. In this regard one of the research tasks would be to look into additional pharmacophoric feature families that could help create a better descriptor. Additionally, other features such as the counts for distances between key atom combinations could be considered as another dimension to the LigityScore representation. Therefore combining different types of descriptors as multiple channels as input to the CNN could potentially be a suitable method to enhance the representation. The key challenge is not only to find a suitable descriptor, but also to build it in a form that is a suitable input to a CNN. Future work for improvement can also be tackled from the CNN model perspective by testing different architectures that have better generalisation ability.

5.4 | Final Remarks

On a final note, finding a suitable representation of the protein-ligand complex is a major challenge to build a successful scoring function, and is key for accurate predictions using deep learning techniques. In our work we have successfully found a suitable representation that to the best of our knowledge was never used for binding affinity prediction, which provides good results and ranked 5th in the CASF-2013 benchmark. Therefore, although our work did not outperform the top scoring function we deem it is still a valid contribution to the area and may be further enhanced in future work, or may also serve as motivation and inspiration for other researchers to seek out alternative methods that increase the effectiveness of scoring functions and virtual screening

in general.

We believe a deeper understanding of CNN in the domain of SBVS is still required, and a breakthrough like the work of Krizhevsky et al. (2012) in the computer vision domain is still being sought after in this challenging domain. Nonetheless, we also believe that ML and DL techniques will lead the future of the development of scoring functions.

A

Media Content

The media content folder structure provided with this study is highlighted in Figure A.1. These directories contain the artifacts used to build, test, and evaluate the LigityScore scoring function. The contents of the each directory are summarised below:

- **ligityscore**. This is the main directory that contains all source code, datasets, and folders used to store the various script outputs. This folder can be used to replicate LigityScore as detailed in Appendix B. The *ligityscore* directory contains the below folders:

  - **tensoboard-runs.** This directory contains the TensorBoard data files for all the logs saved during training. These include the loss every 100 mini-batches for the training set, and the RMSE and R-score values for the training and validation sets computed after each epoch. Each experiment directory also contains a number of CNN model checkpoints to save the model parameters for the epoch with the lowest validation RMSE error. In this folder only the best performing LigityScore experiments are included.

  - **pdbbind.** This folder contains the v2016 and v2018 PDBbind datasets. Each PDBbind versions includes the $INDEX\_general\_PL$.2016 INDEX file that describes the protein-ligand complex properties, and molecular files for the general and refined sets in *sdf* and *mol2* formats. The Core-2013 (*CoreSet2013.dat*) and Core-2016 (*CoreSet.dat*) files are also included with each version of PDBbind.

  - **sbvscnnmsc.** Contains the source files for the *sbvscnnmsc* Python package used to construct the LigityScore SF. The following Python files are included:

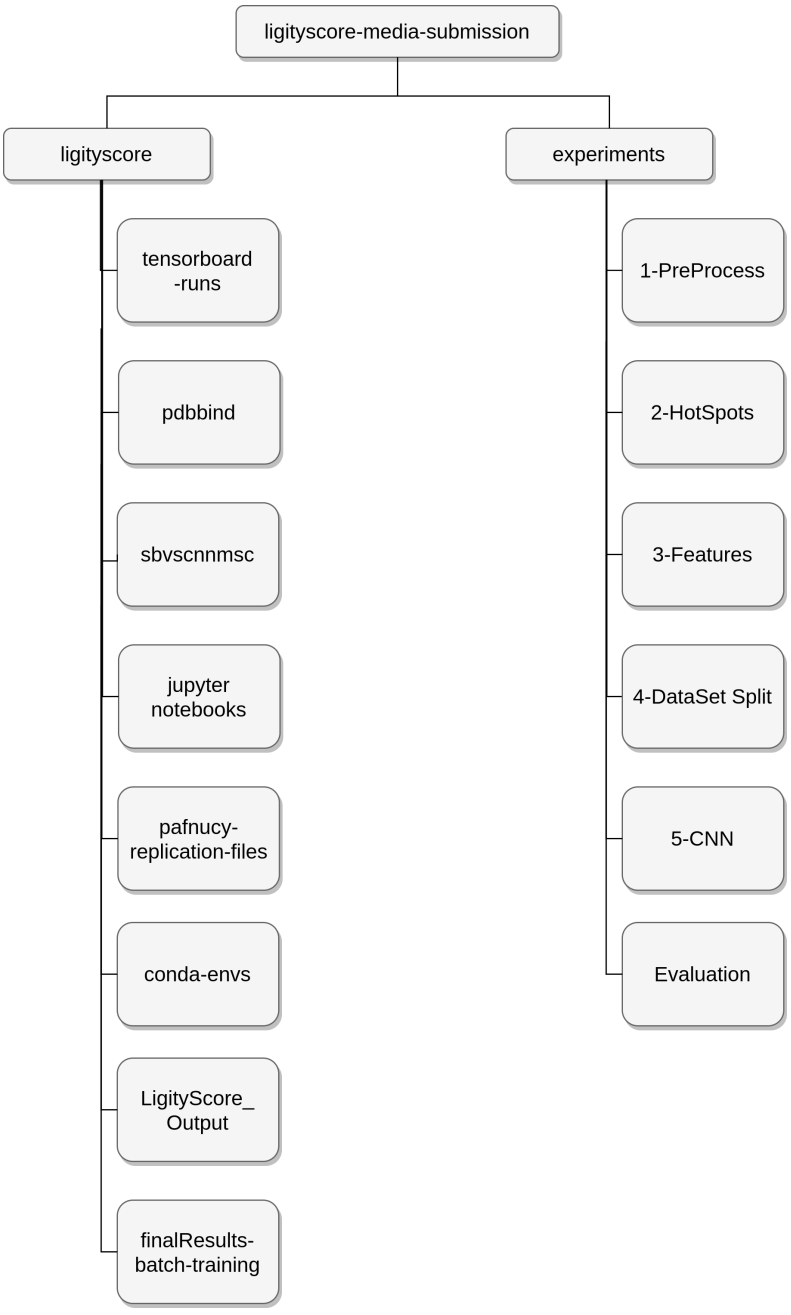


Figure A.1: Media Content Directory Structure.

    - *sbvsCNN.py*. PyTorch Neural Network class definition. Used to create CNN models dynamically using different initialisation parameters.
    - *sbvsData.py*. Used to store custom PyTorch *Dataset* classes for LigityScore1D, LigityScore3D, and Pafnucy. These are then used by the PyTorch *DataLoader* class to load the input to the CNN.
    - *sbvsHelper.py*. Used to store multiple helper classes and functions that are used across multiple LigityScore scripts.
    - *sbvsHelperPafnucy.py*. Use to store multiple helper classes and functions that are used to replicate the Pafnucy model.
  - **jupyter-notebooks.** This directory contains a number of jupyter notebooks that were used during development of LigityScore for initial testing. Some of the notebooks (*results-plots.ipynb* and *Hotpoint Features Diagram.ipynb*) were used to generate figures and plots used in this dissertation.
  - **pafnucy replication**. Contains the source code used for our Pafnucy implementation.
  - ***LigityScore_Output***. This directory is used to store any output files produced by the LigityScore scipts. These include the output from pre-processing, HotSpots extraction, feature generation and its split into Training, Validation, Test, and Test2013 sets. This directory contains all the data files used to run the experiments listed in Tables C.1 and C.2. As an example the *PDB2016Hotpoints1-4.zip* represents the HotSpots dataset using PDBbind v2016 with a PIP distance threshold of 1.4. The folders with names *Ligty1D1-4* contains the LigityScore features, *PDB2016-Lig-Features1D1-4.zip* extracted using *PDB2016Hotpoints1-4.zip* hotspots. *Ligty1D1-4* also include the datasets split from *PDB2016-Lig-Features1D1-4.zip* for Training, Validation, Test, and Test2013.
  - **finalResults-batch-training**. This folder includes all the csv files used to defined the experiments of Tables 4.2, 4.3, and 4.4.
  - **conda-envs**. Contains .yml files for Anaconda Python environments used in LigityScore.
  - The root *ligityscore* directory contains all the source code for LigityScore models. The LigityScore scripts are summarised in Section A.1 below.

- **experiments**. Several LigityScore experiments were performed as is detailed in Chapter 4. The files used or generated for these experiments during each step of

the LigityScore pipeline (Figure 3.1) were stored in separate folders. The console output for each script execution were also saved. The separate folders belonging to each stage of the LigityScore pipeline stages are summarised below:

- **1-PreProcess**. Includes the DataFrames generated using the *pdb-Lig-preprocess.py* script after pre-processing the PDBbind dataset files. The output is saved as a *.zip* compressed pickle object.
- **2-HotSpots**. Includes the HotSpots DataFrames generated using the *pdb-LigHotpoints.py* script, saved as a *.zip* compressed pickle object.
- **3-Features**. Includes the DataFrames generated using the *pdb-ligFeature.py* script saved as a *.zip* compressed pickle object. The features are grouped in separate folders based on the PDBbind version used, and the LigityScore dimension. As an example, the file 'PDB2016-Lig-Feature1D1-4.zip' contains the LigityScore features for the PDBbind 2016 dataset when the HotSpots algorithm was executed using a PIP distance threshold of 1.4. Different folders are used to separate the PDBbind version and the LigityScore version used.
- **4-DataSet Split**. This folder contains a set of Training, Validation, Test2016, and Test2013 datasets created for each LigityScore feature DataFrame generated in the previous step.
- **5-CNN**. The CNN folders contain the files used to carry out the experiments discussed in Chapter 4. The parameter files used to describe each experiments are provided. Additionally the prediction results for the particular experiment are also listed. The TensorBoard data files are also including together with the best checkpoints for each experiment. Since several experiments were performed, only the best performing models are included.
- **Evaluation**. The Evaluation folder contain the output from the Pafnucy replication, and also the CASF scripts that were run to confirm that the same results were obtained using our predictions scripts.

A.1 | LigityScore Scripts

The scripts used in LigityScore are summarised below:

- *pdb-file-preprocess.py*. Used to extract a DataFrame containing information on each complex available in PDBbind (PDBCode, experimental binding affinity, year, ligand name etc), and to check that their corresponding molecular files are available.

- *pdb-LigHotpoints.py*. Program to determine the LigityScore Hotpoints in a PL complex. All the hotspots for each protein-ligand complex in PDBbind are extracted and collected in one DataFrame.
- *pdb-LigFeatures.py*. Program to generate LigityScore Features using the HotSpots DataFrame. It is used to generate both LigityScore1D feature matrix, and the LigityScore3D 3D feature hypercube. Parameter 'LigDimension' controls which LigityScore feature to generate.
- *pdb-split-dataset.py*. Script to split the DataFrame obtained from *pdb-LigFeatures.py* into Test, Test2013, Training, and Validation sets.
- *pdb-training-ligityScore1D.py*. Train a CNN model based on LigtyScore1D features. Model parameters can be defined by a number of script arguments.
- *pdb-training-ligityScore3D.py*. Train a CNN model based on LigtyScore3D features. Model parameters can be defined by a number of script arguments.
- *pdb-predictions-ligityScore.py*. Load a previously trained LigityScore CNN Model, and run predictions for Training, Validation, CASF-2013, and CASF-2016 datasets.
- *pdb-batch-training.py*. Training and Predictions for the LigityScore Scoring Function Experiments. Multiple experiments can be defined in a csv file to be executed sequentially. For each experiment a model is trained using *pdbtrainingligityXDfunction.py* and predictions results are calculated using *pdbpredictionsligityfunction*. The test protein-ligand complexes in CASF-2013 and CASF-2016 are used to measure the performance of the experiment using RMSE, MAE, R, and SD metrics. The full results for each experiments are output as a csv file.
- *pdbtrainingligity1Dfunction.py*. Used by *pdb-batch-training.py* to launch CNN training for LigityScore1D. This script is the same as *pdb-training-ligityScore1D.py* but re-written as a function.
- *pdbtrainingligity3Dfunction.py*. Used by *pdb-batch-training.py* to launch CNN training for LigityScore3D. This script is the same as *pdb-training-ligityScore3D.py* but re-written as a function.
- *pdbpredictionsligityfunction.py*. This script is the same as *pdb-predictions-ligityScore.py* but was re-written as a function to be used by pdb-batch-training.py. The function returns the predictions for Training, Validation, Core2016, and Core2013 sets, using RMSE, MAE, R, and SD metrics.

B

LigityScore User Manual

LigityScore is a CNN based scoring function that predicts the binding affinity for protein-ligand complexes. LigityScore is trained on PDBbind v2016 or v2018 and tested on the CASF-2013 and CASF-2016 scoring power benchmarks. The following user guide lists the steps required to run LigityScore3D for the PDBbind v2016. In our case, the CNN training was run on a 'g4dn.8xlarge' AWS EC2 instance as it requires more than 64G of RAM. For LigityScore1D 'g4dn.2xlarge' instance type can be used.

1. Clone the contents of the provided *ligityscore* folder.

2. Make *ligityscore* folder as your 'present working directory'.

```
$ cd ligityscore
```

3. Unzip the v2016.zip file under the *pdbbind* folder. This contains the PDBbind v2016 dataset.

4. Create conda environments — two environments are required. *msc-ligityscore-features* environment is required for preprocessing, PIP generation, and feature generation. Environment *msc-ligityscore-cnn* is used for CNN training and predictions. The *.yml* files for these conda environments are located in *conda-env* directory.

```
$ conda env create -f conda-envs/msc-ligityscore-features.yml
$ conda env create -f conda-envs/msc-ligityscore-cnn.yml
```

5. In the next steps we are going to 1) Process the PDBbind data, 2) Extract the LigityScore PIP Generation, 3) Generate the LigityScore Descriptors. The files generated for the experiments run in Chapter 4 by these 3 modules are provided in the *LigityScore_Output* folder. Therefore, if you do not want to generate the LigityScore features again, you can skip to *Step 11* directly to start the CNN training.

   Activate *msc-ligityscore-features* conda environment

   ```
   $ conda activate msc-ligityscore-features
   ```

6. **LigityScore PDBBind Preprocessing**. This step will preprocesses the PDBbind dataset and output a file named *PDB2016dataframe-example.zip* containing a DataFrame saved in compressed pickle format of all the protein-ligand complexes available and their properties.

   ```
   $ python3 pdb-file-preprocess.py --input_dir="./pdbbind/v2016"
     --output_dir='./LigityScore_Output'--output_Filename=
     'PDB2016dataframe-example' --pymolFetch 2>&1 |
     tee preprocess-console-output-pdb-2016-example.txt
   ```

7. **LigityScore PIP Generation**. The below command will execute the *pdb-LigHotpoints.py* to generate the PIP dataset with a PIP distance threshold factor of 1.4. The file *PDB2016Hotpoints1-4-example.zip* will be saved in the *LigityScore_Output* folder.

   ```
   $ python3 pdb-LigHotpoints.py --input_filename
     './PDB2016dataframe-example' --it -1 --verbose 1
     --output_filename_Hot "PDB2016Hotpoints1-4-example"
     --pip_threshold 1.4 2>&1 | tee
       hotpoints-console-output-threshold-1-4-2016-example.txt
   ```

8. **LigityScore Feature Generation**. The below command will execute the *pdb-LigFeatures.py* script to generate a hybpercube for each protein-ligand complex from the PIPs generated in previous step (*PDB2016Hotpoints1-4-example.zip*).

   ```
   $ python3 pdb-LigFeatures.py --df_in_hotpoint_filename
     PDB2016Hotpoints1-4-example  --input_dir_df "./LigityScore_Output"
     --output_dir "./LigityScore_Output" --df_out_features_filename
     "PDB2016-Lig-Features3D1-4-example"  --LigDimension 3
     2>&1 | tee ligfeatures-console-output-3D-1-4-2016.txt
   ```

9. Activate *msc-ligityscore-cnn* conda environment.

```
$ conda activate msc-ligityscore-cnn
```

10. Create folder named *Ligity3D1-4-example*. Move the file *PDB2016-Lig-Features3D1-4-example* to *Ligity3D1-4-example*.

11. **LigityScore DataSet Split**. The *pdb-split-dataset.py* script will take the LigityScore features generated in Step 3, and split it into Training, Validation, Test (Core-2016), and Test-2013 (Core-2013) sets. Four files in $./LigityScore_Output/Ligity3D1-4-example$ folder will be created corresponding to these four datasets.

```
$ python3 pdb-split-dataset.py --input_dir
  "./LigityScore_Output/Ligity3D1-4-example" --df_in_filname
   "PDB2016-Lig-Features3D1-4-example"  --output_dir
  "./LigityScore_Output/Ligity3D1-4-example"
  2>&1 | tee dataSplit-console-output-3D-1-4-2016.txt
```

12. Create experiment batch file (*.csv*) containing a list of experiments and their respective CNN parameters. File *msc-exp-LigScore3D-2016-experiment-example.csv* is provided as an example containing the all parameters to train the best LigityScore3D model. This file is located in the *finalResults-batch-training* folder.

13. **Launch Training and Prediction script**. The *pdb-batch-training.py* will launch the CNN training using the parameters defined in *msc-exp-LigScore3D-2016-experiment-example.csv*. The RMSE for the validation set is computed after every epoch and up to 10 CNN model checkpoints are saved at *tensorboard-run/exp-LS-2016-3D-example/models*, that correspond to the lowest RMSE values achieved during training. The prediction script will be executed right after training is complete, to predict binding affinity values using the CNN model checkpoint with the best RMSE error for the validation set.

```
#python3 pdb-batch-training.py --input_dir
"finalResults-batch-training" --output_dir
"finalResults-batch-training" --input_file
"msc-exp-LigScore3D-2016-experiment-example.csv" 2>&1
| tee experiments-console-output-example-2016-3D.txt
```

14. Start TensorBoard and browse to its portal (`http://localhost:6006`) to view training progress. From another shell:

    ```
    $ conda activate msc-ligityscore-cnn
    $ cd ./ligityscore
    $ tensorboard --logdir="tensorboard-runs"
    ```

    To redirect your local port (6006) directly to your server's port (6006) you can use the following ssh command.

    ```
    ssh -i <path-to-your-ssh-server-cert> -N -f -L
    6006:localhost:6006 <username>@<your-server-ipaddress>
    ```

15. Collect results from *finalResults-batch-training* folder when training and predictions are ready. The output file is named *results-msc-exp-LigScore3D-2016-experiment-example.csv*, and includes the following results:

    - Experiment name, description, and best training epoch
    - Training Time
    - Results in RMSE, MAE, SD, and R for the training, validation, Core-2013, and Core-2016 datasets.

C

Experiment Details

Tables C.1 and C.2 provide details on the parameters used for each experiment performed on LigityScore1D and LigityScore3D respectively. Each experiment is assigned an *Exp ID*, that is used to map the experiment details with the experiment results listed in Tables 4.2, 4.3, and 4.4. The 'Experiment Details' column describes the parameters that differ from the baseline CNN models.

Table C.1: LigityScore1D experiments details with corresponding *Exp ID* matching the experiment results listed in 4 for Table 4.2 and Table 4.3.

Exp ID	Exp Shorthand	Experiment Details
100	Baseline	Baseline Models Full Details in Table 4.1.
101	mini-batch 25	Baseline with mini-batch of 25
102	mini-batch 15	Baseline with mini-batch of 15
103	mini-batch 10	Baseline with mini-batch of 10
104	mini-batch 5	Baseline with mini-batch of 5
105	kernel 3x3	Baseline model with kernel size of 3x3
106	4 Rotations, 512 Conv layer	Baseline model with 4 rotations, and 512 dimension 5x5 filter with pooling. Additional convolution layer was added due to increase in input size due to rotation padding.
107	8 Rotations, 512 Conv layer	Baseline model with 8 rotations, and 512 dimension 5x5 filter with pooling. Additional convolution layer was added due to increase in input size due to rotation padding.
108	LR 0.00005	Baseline model with learning rate increased to 0.00005
109	LR 0.0001	Baseline model with learning rate decreased to 0.0001
110	BatchNorm (BN)	Baseline with BatchNorm added after each convolution layer before activation
111	InstanceNorm	Baseline with batchnorm added after each convolution layer before activation
112	Dropout 0.6 0.6 0.5	Baseline model with slightly increased dropout on the FC layers.
113	Dropout 0.8 0.8 0.5	Baseline model with increased dropout on the FC layers.
114	L2 Reg 0.002	Baseline model with L2 regularisation increased to 0.002.
115	L2 Reg 0.004	Baseline model with L2 regularisation increased to 0.004.
116	FC dim - 512, 256, 64	Baseline model with FC connected layers changed number of neurons to 512, 256, 64.
117	FC dim - 500, 200	Baseline model with FC connected layers decreased to 2, and number of neurons changed to 500, 200.
118	FC dim - 1024, 512, 256, 64	Baseline model with FC connected layers increased to 4 and number of neurons changed to1024, 512, 256, 64
119	PIP Threshold 1.1	Baseline model using data model with PIP threshold factor of 1.1
120	PIP Threshold 1.25	Baseline model using data model with PIP threshold factor of 1.25
121	PIP Threshold 1.4	Baseline model using data model with PIP threshold factor of 1.4
122	PIP Threshold 1.5	Baseline model using data model with PIP threshold factor of 1.5
123	PIP Threshold 1.6	Baseline model using data model with PIP threshold factor of 1.6
124	PIP Threshold 1.4 w/Lipinski	Baseline model using data model with PIP threshold factor of 1.4 with Lipinski drug-like filtering applied to PL complexes.
125	PIP Threshold 1.4 w/Max dist 30	Baseline model using data model with PIP threshold factor of 1.4 and PIP Generation Max distance of 30 A.
126	PIP Threshold 1.4 w/Max dist 40	Baseline model using data model with PIP threshold factor of 1.4 and PIP Generation Max distance of 40 A.
127	PIP Threshold 1.4 w/resolution0.5	Baseline model using data model with PIP threshold factor of 1.4 and PIP Generation resolution of 0.5.
128	BN–1-4*	Baseline model with PIP threshold factor of 1.4, and BatchNorm after each Conv Layer.
129	IN—1-4*	Baseline model with PIP threshold factor of 1.4, and InstanceNorm after each Conv Layer.

Exp ID	Exp Shorthand	Experiment Details
130	IN–1-4–Drop0-8*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and increased dropout at FC layers.
131	IN–1-25*	Baseline model with PIP threshold factor of 1.25, and InstanceNorm after each Conv Layer.
132	IN–1-4–BS15*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and mini-batch of 15.
133	IN–1-4–r4*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, with added rotations at 90 degrees.
134	IN–1-4–r8*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, with added roations at 45 degrees.
135	IN–1-4–Cdrop0-1-all*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after all Conv Layers.
136	IN–1-4–Cdrop0-2-all*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after all Conv Layers.
137	IN–1-4–Cdrop0-1-mid*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only.
138	IN–1-4–Cdrop0-1-mid–BS15*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, spatial dropout of 0.1 added after middle layer only, and mini-batch of 15.
139	IN–1-4–Cdrop0-2-mid*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after middle layer only.
140	IN–1-4–Cdrop0-1-all-r4*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after all Conv Layers, with rotations at 90 degrees.
141	IN–1-4–Cdrop0-1-all-r8*	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after all Conv Layers, with rotations at 45 degrees.
142	BN–1-4–Cdrop0-1-mid*	Baseline model with PIP threshold factor of 1.4, BatchNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only.
143	IN–1-4–Cdrop0-1-mid–drop0.8–r4	Experiment 135 with increased dropout at FC layers.
144	IN–1-4–Cdrop0-1-mid–drop0.8–r4	Experiment 135 with increased dropout at FC layers, and 90 degree rotations.
145	IN–1-4–Cdrop0-1-mid–4ConvLayers	Experiment 135 with 4 Conv Layers - 32, 64, 128, 256 (LigityScore1D only)
146	IN–1-4–Cdrop0-1-mid–5ConvLayers	Experiment 135 with 5 Conv Layers - 32, 64, 128, 256, 512 (LigityScore1D only)
147	IN–1-4–Cdrop0-1-mid–3Convdouble	Experiment 135 with 6 layers. Each filter is executed twice (64, 64, 128, 128, 256, 256) (LigityScore1D only)
148	IN–1-4–Cdrop0-1-mid–4Convdouble	Experiment 135 with 8 layers. Each filter is executed twice (64, 64, 128, 128, 256, 256, 256, 256) (LigityScore1D only)
149	IN–1-4–Cdrop0-1-mid–r4–VGG16 –no-last-pool	Experiment 135 using VGG model, withouth last pooling layers and with added spatial dropout. (LigityScore1D only)
150	IN–1-6–Cdrop0-1-mid–lr0-0001	Baseline model with PIP threshold factor of 1.6, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only, and lr of 0.0001. (LigityScore1D only)
151	BN–1-4–Cdrop0-1-mid–lr0-0001	Baseline model with PIP threshold factor of 1.4, BatchNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only, and lr of 0.0001.(LigityScore1D only)

Table C.2: LigityScore3D experiments details with corresponding *Exp ID* matching the experiment results listed in 4 for Table 4.4.

Exp ID	Exp Shorthand	Experiment Details
300	Baseline	Baseline Models Full Details in Table 4.1.
301	BatchNorm (BN)	Baseline with BatchNorm added after each convolution layer before activation
302	InstanceNorm (IN)	Baseline with instancenorm added after each convolution layer before activation
303	IN–mini-batch 25	Baseline with instancenorm and mini-batch of 25
304	IN–mini-batch 15	Baseline with instancenorm and mini-batch of 15
305	IN–mini-batch 10	Baseline with instancenorm and mini-batch of 10
306	IN-no-padding	Baseline with instancenorm added after each convolution layer before activation. No padding is used to decrease size on FC input.
307	IN–rotations6 (r6)	Baseline with Instancenorm and 6 rotations across x, y, x planes.
308	IN–LR 0.00005	Baseline model with Instancenorm and learning rate increased to 0.00005
309	IN–LR 0.0001	Baseline model with Instancenorm and learning rate decreased to 0.0001
310	IN–Dropout 0.8 0.8 0.5	Baseline model with Instancenorm and l slightly dropout increased to 0.8 on the FC layers. Last FC layer dropout is kept at 0.5.
311	IN–FC dim - 4000, 500, 200, 1	Baseline model with Instancenorm and FC connected layers changed number of neurons to 4000, 500, 200, 1
312	IN–FC dim - 6000, 2000, 500, 200, 1	Baseline model with Instancenorm and FC connected layers changed number of neurons to 6000, 2000, 500, 200, 1
313	IN–kernel-3x3–no-padding	Baseline model with Instancenorm and kernel size of 3x3. Padding is changed to 0 to keep input to FC layer small.
314	IN–PIP Threshold 1.1	Baseline model with instancenorm and using data model with PIP threshold factor of 1.1
315	IN–PIP Threshold 1.25	Baseline model with instancenorm and using data model with PIP threshold factor of 1.25
316	IN–PIP Threshold 1.4	Baseline model with instancenorm and using data model with PIP threshold factor of 1.4
317	IN–PIP Threshold 1.5	Baseline model with instancenorm and using data model with PIP threshold factor of 1.5
318	IN–PIP Threshold 1.6	Baseline model with instancenorm and using data model with PIP threshold factor of 1.6
319	IN–PIP Threshold 1.4 w/Lipinski	Baseline model with instancenorm and using data model with PIP threshold factor of 1.4 with Lipinski drug-like filtering applied to PL complexes.
320	IN–PIP Threshold 1.5 w/Lipinski	Baseline model with instancenorm and using data model with PIP threshold factor of 1.5 with Lipinski drug-like filtering applied to PL complexes.
321	BN–PIP Threshold 1.4	Baseline model with batchnorm and using data model with PIP threshold factor of 1.4
322	IN–1-4–Drop0-8	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and increased dropout at FC layers.
323	IN–1-4–BS15	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and mini-batch of 15.
324	IN–1-4–r6	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, with 6 added rotation along x, y, z planes.

Exp ID	Exp Shorthand	Experiment Details
325	IN–1-4–Cdrop0-1-all	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after all Conv Layers.
326	IN–1-4–Cdrop0-2-all	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after all Conv Layers.
327	IN–1-4–Cdrop0-1-mid	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only.
328	IN–1-6–Cdrop0-1-mid	Baseline model with PIP threshold factor of 1.6, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only.
329	IN–1-4–Cdrop0-2-mid	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after middle layer only.
330	IN–1-4–Cdrop0-1-all-r6	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after all Conv Layers, with 6 rotations along the x, y, z planes
331	Conv3D–IN–1-4	Baseline model with PIP threshold factor of 1.4, and InstanceNorm after each Conv Layer. Conv layers use Conv3D modules. Used in LitgityScore3D only.
332	Conv3D–IN–1-4-Cdrop0-1-mid	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.1 added after middle layer only. Conv layers use Conv3D module. Used in LitgityScore3D only.
333	Conv3D–IN–1-4–Cdrop0-2-all	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after all Conv Layers. Conv layers use Conv3D module. Used in LitgityScore3D only.
334	Conv3d–IN–1-4–Cdrop0-2-all-BS15	Baseline model with PIP threshold factor of 1.4, InstanceNorm after each Conv Layer, and spatial dropout of 0.2 added after all Conv Layers and mini-batch of 15. Conv layers use Conv3D module. Used in LitgityScore3D only.

References

Kaggle: Merck molecular activity challenge, 2012. URL https://www.kaggle.com/c/MerckActivity. https://www.kaggle.com/c/MerckActivity, Accessed Feb 8, 2019.

AWS: Cloud credits for research, 2020. URL https://aws.amazon.com/grants/. https://aws.amazon.com/grants/, Accessed June 15, 2020.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. tensorflow.org, Accessed 7 September, 2019.

Qurrat Ul Ain, Antoniya Aleksandrova, Florian D Roessler, and Pedro J Ballester. Machine-learning scoring functions to improve structure-based binding affinity prediction and virtual screening. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 5(6):405–424, 2015.

Natalia Artemenko. Distance dependent scoring function for describing protein- ligand intermolecular interactions. *Journal of chemical information and modeling*, 48(3):569–574, 2008.

Hossam M Ashtawy and Nihar R Mahapatra. A comparative assessment of ranking accuracies of conventional and machine-learning-based scoring functions for protein-ligand binding affinity prediction. *IEEE/ACM Transactions on computational biology and bioinformatics*, 9(5):1301–1313, 2012.

Pedro J Ballester and John BO Mitchell. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, 26(9):1169–1175, 2010.

Helen Berman, Kim Henrick, and Haruki Nakamura. Announcing the worldwide protein data bank. *Nature Structural & Molecular Biology*, 10(12):980–980, 2003.

Fergus Boyles, Charlotte M Deane, and Garrett M Morris. Learning from the ligand: using ligand-based features to improve binding affinity prediction. *Bioinformatics*, 36(3):758–764, 2020.

Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

Tiejun Cheng, Xun Li, Yan Li, Zhihai Liu, and Renxiao Wang. Comparative assessment of scoring functions on a diverse test set. *Journal of chemical information and modeling*, 49(4):1079–1093, 2009a.

Tiejun Cheng, Xun Li, Yan Li, Zhihai Liu, and Renxiao Wang. Comparative assessment of scoring functions on a diverse test set. *Journal of chemical information and modeling*, 49(4):1079–1093, 2009b.

Tiejun Cheng, Qingliang Li, Zhigang Zhou, Yanli Wang, and Stephen Bryant. Structure-based virtual screening for drug discovery: a problem-centric review. *The AAPS Journal*, 14(1):133–141, 2012. ISSN 1550-7416.

Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141): 20170387, 2018.

Warren Lyford DeLano. Pymol, 2002.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Wei Deng, Curt Breneman, and Mark J Embrechts. Predicting protein- ligand binding affinities using novel geometrical descriptors and machine-learning methods. *Journal of chemical information and computer sciences*, 44(2):699–703, 2004.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

James B Dunbar Jr, Richard D Smith, Chao-Yie Yang, Peter Man-Un Ung, Katrina W Lexa, Nickolay A Khazanov, Jeanne A Stuckey, Shaomeng Wang, and Heather A Carlson. Csar benchmark exercise of 2010: selection of the protein–ligand complexes. *Journal of chemical information and modeling*, 51(9):2036–2046, 2011.

Jacob D Durrant and J Andrew McCammon. Nnscore: a neural-network-based scoring function for the characterization of protein- ligand complexes. *Journal of chemical information and modeling*, 50(10):1865–1871, 2010.

Jean-Paul Ebejer, Paul W Finn, Wing Ki Wong, Charlotte M Deane, and Garrett M Morris. Ligity: A non-superpositional, knowledge-based approach to virtual screening. *Journal of chemical information and modeling*, 59(6):2600–2616, 2019.

Todd JA Ewing, Shingo Makino, A Geoffrey Skillman, and Irwin D Kuntz. Dock 4.0: search strategies for automated molecular docking of flexible molecule databases. *Journal of computer-aided molecular design*, 15(5):411–428, 2001.

Emil Fischer. Einfluss der configuration auf die wirkung der enzyme. *Ber*, 27:2985–2989, 1894.

Richard A Friesner, Jay L Banks, Robert B Murphy, Thomas A Halgren, Jasna J Klicic, Daniel T Mainz, Matthew P Repasky, Eric H Knoll, Mee Shelley, Jason K Perry, et al. Glide: a new approach for rapid, accurate docking and scoring. 1. method and assessment of docking accuracy. *Journal of medicinal chemistry*, 47(7):1739–1749, 2004.

Joffrey Gabel, Jérémy Desaphy, and Didier Rognan. Beware of machine learning-based scoring functions on the danger of developing black boxes. *Journal of chemical information and modeling*, 54(10):2807–2815, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Peter Gund. Three-dimensional pharmacophoric pattern searching. In *Progress in molecular and subcellular biology*, pages 117–143. Springer, 1977.

Michael J Hartshorn, Marcel L Verdonk, Gianni Chessari, Suzanne C Brewerton, Wijnand TM Mooij, Paul N Mortenson, and Christopher W Murray. Diverse, high-quality test set for the validation of protein-ligand docking performance. *Journal of medicinal chemistry*, 50(4):726–741, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Andrew L Hopkins. Predicting promiscuity. *Nature*, 462(7270):167–168, 2009.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

José Jiménez, Miha Skalic, Gerard Martinez-Rosell, and Gianni De Fabritiis. K deep: protein–ligand absolute binding affinity prediction via 3d-convolutional neural networks. *Journal of chemical information and modeling*, 58(2):287–296, 2018.

Gareth Jones, Peter Willett, Robert C Glen, Andrew R Leach, and Robin Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of molecular biology*, 267(3):727–748, 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. arxiv:1412.6980.

Peter Kirkpatrick and Clare Ellis. Chemical space. *Nature*, 432(7019):823, 2004.

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.

Greg Landrum. Rdkit: Open-source cheminformatics, 2020. URL `http://www.rdkit.org`. Accessed April, 2020.

Andrew R Leach, Valerie J Gillet, Richard A Lewis, and Robin Taylor. Three-dimensional pharmacophore methods in drug discovery. *Journal of medicinal chemistry*, 53(2):539–558, 2010.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19:143–155, 1989.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Hongjian Li, Kwong-Sak Leung, Man-Hon Wong, and Pedro J Ballester. Improving autodock vina using random forest: the growing accuracy of binding affinity prediction by the effective exploitation of larger data sets. *Molecular informatics*, 34(2-3):115–126, 2015.

Jin Li, Ailing Fu, and Le Zhang. An overview of scoring functions used for protein–ligand interactions in molecular docking. *Interdisciplinary Sciences: Computational Life Sciences*, pages 1–9, 2019.

Yan Li, Li Han, Zhihai Liu, and Renxiao Wang. Comparative assessment of scoring functions on an updated benchmark: 2. evaluation methods and general results. *Journal of chemical information and modeling*, 54 (6):1717–1736, 2014a.

Yan Li, Zhihai Liu, Jie Li, Li Han, Jie Liu, Zhixiong Zhao, and Renxiao Wang. Comparative assessment of scoring functions on an updated benchmark: 1. compilation of the test set. *Journal of chemical information and modeling*, 54(6):1700–1716, 2014b.

Yan Li, Minyi Su, Zhihai Liu, Jie Li, Jie Liu, Li Han, and Renxiao Wang. Assessing protein–ligand interaction scoring functions with the casf-2013 benchmark. *Nature protocols*, 13(4):666, 2018.

Christopher A Lipinski, Franco Lombardo, Beryl W Dominy, and Paul J Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews*, 23(1-3):3–25, 1997.

Zhihai Liu, Yan Li, Li Han, Jie Li, Jie Liu, Zhixiong Zhao, Wei Nie, Yuchen Liu, and Renxiao Wang. Pdb-wide collection of binding data: current status of the pdbbind database. *Bioinformatics*, 31(3):405–412, 2015.

Zhihai Liu, Minyi Su, Li Han, Jie Liu, Qifan Yang, Yan Li, and Renxiao Wang. Forging the basis for developing protein–ligand interaction scoring functions. *Accounts of chemical research*, 50(2):302–309, 2017a.

Zhihai Liu, Minyi Su, Li Han, Jie Liu, Qifan Yang, Yan Li, and Renxiao Wang. Forging the basis for developing protein–ligand interaction scoring functions. *Accounts of chemical research*, 50(2):302–309, 2017b.

Zhonghao Liu, Yuxin Cui, Zheng Xiong, Alierza Nasiri, Ansi Zhang, and Jianjun Hu. Deepseqpan, a novel deep convolutional neural network model for pan-specific class i hla-peptide binding affinity prediction. *Scientific reports*, 9(1):794, 2019.

Junshui Ma, Robert P Sheridan, Andy Liaw, George E Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55 (2):263–274, 2015.

Andreas Mayr, Günter Klambauer, Thomas Unterthiner, and Sepp Hochreiter. Deeptox: toxicity prediction using deep learning. *Frontiers in Environmental Science*, 3:80, 2016.

Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017.

Garrett M Morris, Ruth Huey, William Lindstrom, Michel F Sanner, Richard K Belew, David S Goodsell, and Arthur J Olson. Autodock4 and autodocktools4: Automated docking with selective receptor flexibility. *Journal of computational chemistry*, 30(16):2785–2791, 2009.

Ingo Muegge. Pmf scoring revisited. *Journal of medicinal chemistry*, 49(20):5895–5902, 2006. ISSN 0022-2623.

Michael M Mysinger, Michael Carchia, John J Irwin, and Brian K Shoichet. Directory of useful decoys, enhanced (dud-e): better ligands and decoys for better benchmarking. *Journal of medicinal chemistry*, 55 (14):6582–6594, 2012.

Duc Duy Nguyen and Guo-Wei Wei. Agl-score: Algebraic graph learning score for protein–ligand binding scoring, ranking, docking, and screening. *Journal of chemical information and modeling*, 59(7):3291–3304, 2019a.

Duc Duy Nguyen and Guo-Wei Wei. Dg-gl: Differential geometry-based geometric learning of molecular datasets. *International journal for numerical methods in biomedical engineering*, 35(3):e3179, 2019b.

Noel M O'Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. Open babel: An open chemical toolbox. *Journal of chem informatics*, 3(1):33, 2011.

Hakime Öztürk, Arzucan Özgür, and Elif Ozkirimli. Deepdta: deep drug–target binding affinity prediction. *Bioinformatics*, 34(17):i821–i829, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Janaina Cruz Pereira, Ernesto Raul Caffarena, and Cicero Nogueira dos Santos. Boosting docking-based virtual screening with deep learning. *Journal of chemical information and modeling*, 56(12):2495–2506, 2016.

Javier Pérez-Sianes, Horacio Pérez-Sánchez, and Fernando Díaz. Virtual screening meets deep learning. *Current computer-aided drug design*, 15(1):6–28, 2019.

Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. Ucsf chimera—a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.

Matthew Ragoza, Joshua Hochuli, Elisa Idrobo, Jocelyn Sunseri, and David Ryan Koes. Protein–ligand scoring with convolutional neural networks. *Journal of chemical information and modeling*, 57(4):942–957, 2017.

Ahmet Sureyya Rifaioglu, Heval Atas, Maria Jesus Martin, Rengul Cetin-Atalay, Volkan Atalay, and Tunca Dogan. Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. *Brief. Bioinform*, 10, 2018.

Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Chao Shen, Ye Hu, Zhe Wang, Xujun Zhang, Jinping Pang, Gaoang Wang, Haiyang Zhong, Lei Xu, Dongsheng Cao, and Tingjun Hou. Beware of the generic machine learning-based scoring functions in structure-based virtual screening. *Briefings in Bioinformatics*, 2020.

Jochen Sieg, Florian Flachsenberg, and Matthias Rarey. In need of bias control: Evaluating chemical data for machine learning in structure-based virtual screening. *Journal of chemical information and modeling*, 59 (3):947–961, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *InICLR, 2015*, 2014.

Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182(566):7, 2013.

Marta Stepniewska-Dziubinska. tfbio, 2020. URL `https://gitlab.com/cheminfIBB/tfbio`. Accessed June, 2020.

Marta M Stepniewska-Dziubinska, Piotr Zielenkiewicz, and Pawel Siedlecki. Pafnucy–a deep neural network for structure-based drug discovery. *stat*, 1050:19, 2017.

Minyi Su, Qifan Yang, Yu Du, Guoqin Feng, Zhihai Liu, Yan Li, and Renxiao Wang. Comparative assessment of scoring functions: the casf-2016 update. *Journal of chemical information and modeling*, 59(2): 895–913, 2018.

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

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.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

Oleg Trott and Arthur J Olson. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2): 455–461, 2010.

Tiziano Tuccinardi. Docking-based virtual screening: recent developments. *Combinatorial chemistry & high throughput screening*, 12(3):303–314, 2009.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017.

Martin J Valler and Darren Green. Diversity screening versus focussed screening in drug discovery. *Drug discovery today*, 5(7):286–293, 2000.

Izhar Wallach, Michael Dzamba, and Abraham Heifets. Atomnet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *arXiv preprint arXiv:1510.02855*, 2015.

Cheng Wang and Yingkai Zhang. Improving scoring-docking-screening powers of protein-ligand scoring functions using random forest. *Journal of computational chemistry*, 38(3):169–177, 2017.

Maciej Wójcikowski, Pedro J Ballester, and Pawel Siedlecki. Performance of machine-learning scoring functions in structure-based virtual screening. *Scientific reports*, 7:46710, 2017.

Maciej Wójcikowski, Michał Kukiełka, Marta M Stepniewska-Dziubinska, and Pawel Siedlecki. Development of a protein–ligand extended connectivity (plec) fingerprint and its application for binding affinity predictions. *Bioinformatics*, 35(8):1334–1341, 2019.

Haiping Zhang, Linbu Liao, Konda Mani Saravanan, Peng Yin, and Yanjie Wei. Deepbindrg: a deep learning based method for estimating effective protein–ligand affinity. *PeerJ*, 7:e7362, 2019.

Liangzhen Zheng, Jingrong Fan, and Yuguang Mu. Onionnet: a multiple-layer intermolecular-contact-based convolutional neural network for protein–ligand binding affinity prediction. *ACS omega*, 4(14): 15956–15965, 2019.