THESIS

PROTEIN INTERFACE PREDICTION USING GRAPH CONVOLUTIONAL NETWORKS

Submitted by

Alex M. Fout

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2017

Master's Committee:

Advisor: Asa Ben-Hur

Chuck Anderson
Hamid Reza Chitsaz
Wen Zhou

ABSTRACT

PROTEIN INTERFACE PREDICTION USING GRAPH CONVOLUTIONAL NETWORKS

Proteins play a critical role in processes both within and between cells, through their interactions with each other and other molecules. Proteins interact via an *interface* forming a protein complex, which is difficult, expensive, and time consuming to determine experimentally, giving rise to computational approaches. These computational approaches utilize known electrochemical properties of protein amino acid residues in order to predict if they are a part of an interface or not. Prediction can occur in a partner independent fashion, where amino acid residues are considered independently of their neighbor, or in a partner specific fashion, where pairs of potentially interacting residues are considered together. Ultimately, prediction of protein interfaces can help illuminate cellular biology, improve our understanding of diseases, and aide pharmaceutical research. Interface prediction has historically been performed with a variety of methods, to include *docking*, *template matching*, and more recently, *machine learning approaches*.

The field of machine learning has undergone a revolution of sorts with the emergence of *convolutional neural networks* as the leading method of choice for a wide swath of tasks. Enabled by large quantities of data and the increasing power and availability of computing resources, convolutional neural networks efficiently detect patterns in grid structured data and generate hierarchical representations that prove useful for many types of problems. This success has motivated the work presented in this thesis, which seeks to improve upon state of the art interface prediction methods by incorporating concepts from convolutional neural networks.

Proteins are inherently irregular, so they don't easily conform to a grid structure, whereas a graph representation is much more natural. Various convolution operations have been proposed for graph data, each geared towards a particular application. We adapted these convolutions for use in interface prediction, and proposed two new variants. Neural networks were trained on the Docking

Benchmark Dataset version 4.0 complexes and tested on the new complexes added in version 5.0. Results were compared against the state of the art method partner specific method, PAIRpred [1]. Results show that multiple variants of graph convolution outperform PAIRpred, with no method emerging as the clear winner.

In the future, additional training data may be incorporated from other sources, unsupervised pretraining such as autoencoding may be employed, and a generalization of convolution to simplicial complexes may also be explored. In addition, the various graph convolution approaches may be applied to other applications with graph structured data, such as Quantitative Structure Activity Relationship (QSAR) learning, and knowledge base inference.

ACKNOWLEDGEMENTS

I would like to firstly acknowledge and thank my Creator and Savior Jesus Christ, who continues to bless me with new challenges and show me the wonder of His creation. Thank you to my parents for their unconditional love and support that has manifested itself in so many practical ways, and to Kris, Katherine, Steve, Quinn, and members of my extended family for their support and understanding as I've worked towards completion of my degree. My immense gratitude goes to my advisor, Dr. Asa Ben-Hur for his patience, calm guidance, and encouragement to maintain a healthy life balance in graduate school of all places. I thank the members of my committee for lending their expertise to provide feedback and insight on the work presented in this thesis, to include Chuck Anderson, Hamid Reza Chitsaz, and Wen Zhou.

I must thank my fellow interface predictors, Basir Shariat and Jonathon Byrd, who very much compensated for my shortcomings in numerous ways, and who bravely held me back from the precipice of software engineering. We make a good team! I appreciate all of the support and feedback from Gareth Halladay, Mike Hamilton, Don Neumann, Swapnil Sneham, and Fahad Ullah. Thanks to the National Science Foundataion for enabling this research through grant funding: Award Number 1564840, DeepStruct: Learning representations of protein 3-d structures and their interfaces using deep architectures.

Some molecular graphics and analyses were performed with the UCSF Chimera package. Chimera is developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco (supported by NIGMS P41-GM103311).

Soli Deo Gloria

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Many cellular processes rely on proteins, which facilitate these processes via their interactions with one another and with small molecules within the cell [6]. Understanding protein interactions is key to disease and pharmaceutical research, as well as our understanding of basic cellular biology [7,8]. Proteins normally interact via an *interface*, a localized region of the protein with special properties.

Experimentally identifying the interface between two proteins is a time consuming and expensive process which involves crystallization of the protein complex and imaging via x-ray crystallography or nuclear magnetic resonance [9–11]. In general, more than one hundred thousand proteins have been crystallized and imaged in the last 40 years, but doing so for weakly interacting protein complexes remains a challenging problem [12] [13]. In contrast, computational methods are faster, cheaper, and complement wet lab experiments by identifying the most relevant and worthwhile experiments to attempt in vivo.

This thesis presents a novel computational method to predict the interface between a pair of interacting proteins. The method is inspired by the success of convolutional neural networks in image processing [14, 15], but adapts the ideas to this problem domain. Proteins are represented as graphs and fed into a pairwise convolutional neural network with specialized convolution operations. The network makes predictions on pairs of amino acid residues of the likelihood that they constitute a part of the interface. This method outperforms the existing state-of-the-art approach based on a support vector machine with pairwise kernels [1].

The rest of this thesis is organized as follows. The remainder of the introduction presents a primer on proteins and their interfaces. Chapter 2 reviews prior work in interface prediction. Chapter 3 introduces neural networks and specifically convolutional neural networks. Chapter 4 describes how proteins can be modeled as graphs, discusses various methods of graph convolution, and introduces the pairwise neural network architecture used for interface prediction. Chapter 5

describes the data set, experimental procedure, and relevant metrics used to evaluate the various graph convolution methods. Chapter 6 discusses the experimental results. Chapter 7 lays out potential avenues of research which build upon the findings in this thesis. Appendix A contains details pertaining to the features computed for each amino acid residue, and Appendix B describes the software implementation used for this research.

1.1 Proteins

DNA is rightly considered the "blueprint for life," which begs the question, what is built from those blueprints? The answer is, in many cases, information contained in DNA is used to synthesize proteins. Proteins are composed of amino acids linked in a chain and held together by covalent bonds. Amino acids are organic compounds consisting of a central α*-carbon* atom, which binds to an *amine group* (NH_2), a *carboxyl group* ($COOH$), a single hydrogen atom, and a *side chain*, in a tetrahedral geometry. There are 21 unique amino acids encoded in eukaryotic DNA, each of which has a distinct side chain that gives rise to structural and electro-chemical properties. Figure 1.1 depicts the different amino acids.

Two amino acids link together when the nitrogen atom from one's amine group bonds covalently with the carbon atom of the other's carboxyl group, releasing a water molecule. This covalent bond is called a *peptide bond*, and an amino acid involved in at least one such bond is referred to as an *amino acid residue*, or *residue*.

A *peptide*, or *peptide chain*, is a linear chain of amino acids held together by peptide bonds, and the *backbone* of the peptide consists of all atoms participating in peptide bonds, together with the α-carbons. If a peptide contains several residues it is referred to as a *polypeptide*. Proteins consist of one or more polypeptides which are bound together. All peptides have a canonical ordering, from the *N-terminus*, the residue with a free amine group, to the *C-terminus*, the residue with a free carboxyl group. This matches the order that polypeptides are created during biological protein synthesis. Figure 1.2 shows a ball and stick model of a small peptide.

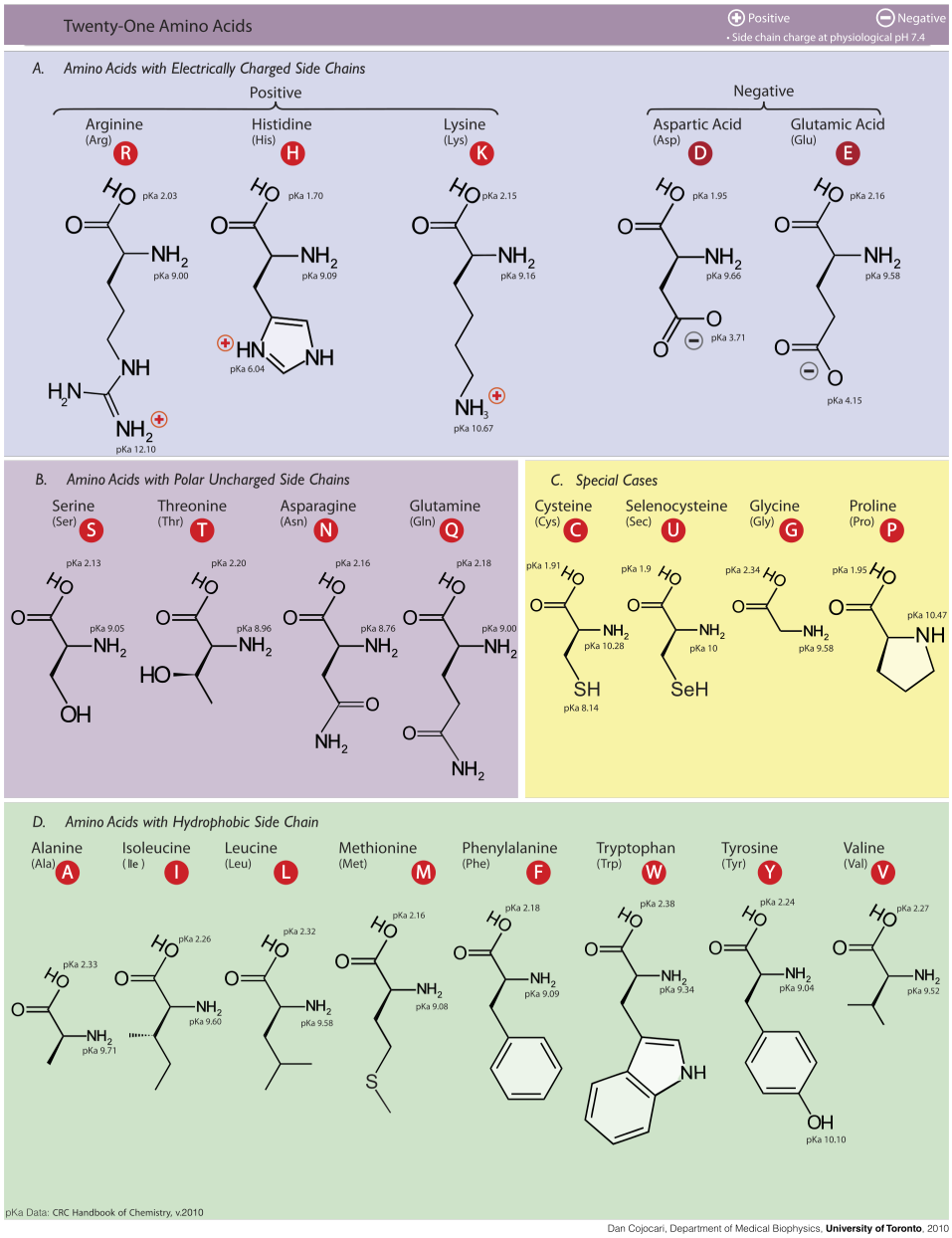


Figure 1.1: The 21 types of amino acid encoded in eukaryotic DNA, categorized by their electrical properties. Full names, along with three letter and single letter abbreviations are given. Chemical structures are oriented so that the carboxyl group, α-carbon, and amine groups are on top, with the side chain extending downwards. This work is "Molecular structures of the 21 proteinogenic amino acids" by Dan Cojocari, Princess Margaret Cancer Centre, University of Toronto. It is under a Creative Commons Attribution-Share Alike 3.0 Unported license. File: https://commons.wikimedia.org/wiki/File:Molecular_structures_of_the_21_proteinogenic_amino_acids.svg, License: https://creativecommons.org/licenses/by-sa/3.0/deed.en

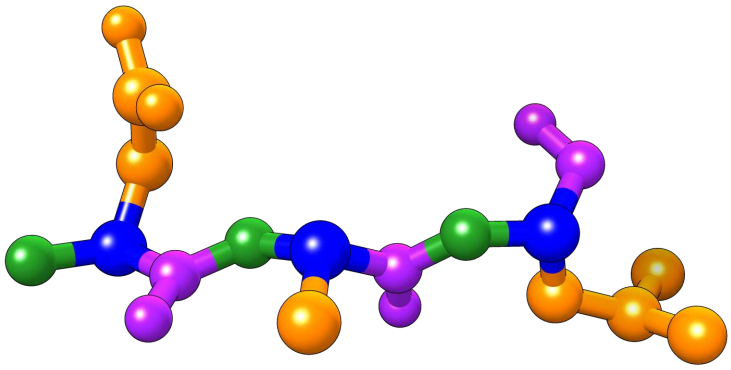


Figure 1.2: Ball and stick model of a peptide containing three amino acid residues (hydogens omitted). Colors indicate, blue: α-carbon , orange: side chain, purple: carboxyl group after loss of OH, green: amine group after loss of H. Peptide bonds are between purple and green atoms. The backbone extends horizontally along blue, purple, and green atoms. Each sequence of atoms starting at an α-carbon and ending at the next α-carbon lay in an amide plane. From left to right are Aspartic Acid, Alanine, and Leucine. Created with Chimera [2].

A residue's side chain influences how it interacts with other amino acids or other atoms and molecules. For example, oppositely charged side chains are attracted to each other. Polar side chains, being hydrophilic, are attracted to water molecules, and non-polar side chains, being hydrophobic, will prefer to avoid water and other polar molecules. Such interactions play an important role in how proteins fold into 3D structures [6].

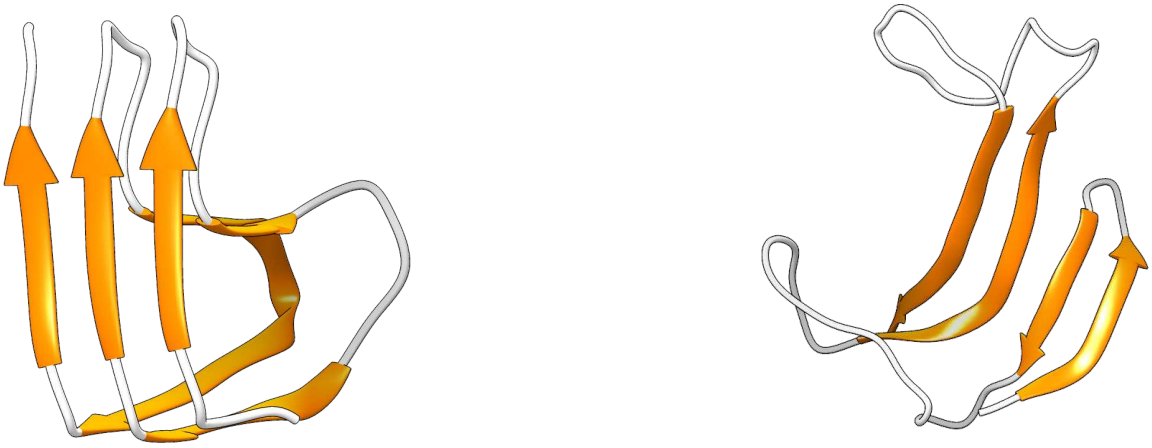
1.1.1 Protein Structure

Protein structure can be described via four levels of abstraction, termed *primary*, *secondary*, *tertiary*, and *quaternary* structure. Primary structure refers to the sequence of amino acid residues (from N-terminus to C-terminus) in a single polypeptide, and is determined by the sequence of codons in the corresponding coding mRNA from which the protein is translated. Sequences are typically written using a string of letters, where each unique letter corresponds to a different residue type.

The physical chemistry associated with peptide bonds gives rise to the property that the nitrogen and carbon atoms involved in the peptide bond, along with the adjacent α-carbons, all lie within a plane, called the *amide plane*. Each α-carbon lies on the intersection between two amide planes, and the planes are free to rotate with respect to each other. In some cases, side chains prohibit certain relative angles due to *steric constraints*, which enforce that no two atoms may occupy the same volume of space at the same time. In general, however, the angle between amide planes provides flexibility in the peptide backbone, which allows for the formation of higher order structures.

Secondary structure describes the local 3D structures that arise from the attraction of non-adjacent residues in a polypeptide. There are three common categories of local structures: α-helices, β-sheets, and loops. An α-helix occurs when the polypeptide coils into a barrel-like structure (like the threads of a screw) and residues from adjacent coils (a distance of three residues from each other along the backbone) form hydrogen bonds with one another. A β-sheet occurs when two non-adjacent sections of the polypeptide align next to each other such that residues in one of the sections form hydrogen bonds with residues in the other section. β-sheets may be parallel or anti-parallel, depending on the relative orientation of adjacent strands in the sheet. Figure 1.3 illustrates the difference between parallel and anti-parallel β-sheets. Some sections of the polypeptide form neither helices nor sheets, and are called loops. These sections are more flexible than helices or sheets due the lack of hydrogen bonds, and therefore are useful in connecting the end of one helix/sheet to the beginning of another. Figure 1.4 shows a cartoon depiction of part of a protein, with different secondary structural elements highlighted in different colors.

Helices and sheets provide some rigidity to a polypeptide, but it typically further folds into a larger, tertiary structure, which may be globular, fibrous, or such that it resides on a cellular membrane. After folding, some residues reside on the surface while others are buried in the core. Because water makes up the majority of the cellular environment, protein surfaces have a higher percentage of hydrophilic residues than cores, and cores likewise have a higher percentage of



(a) Parallel β-sheets

(b) Anti-parallel β-sheets

Figure 1.3: Cartoon examples of parallel and anti-parallel β-sheets. In parallel β-sheets, adjacent strands in the sheet are oriented the same way, whereas in anti-parallel β-sheets, adjacent strands are oriented opposite each other. Created with Chimera [2].

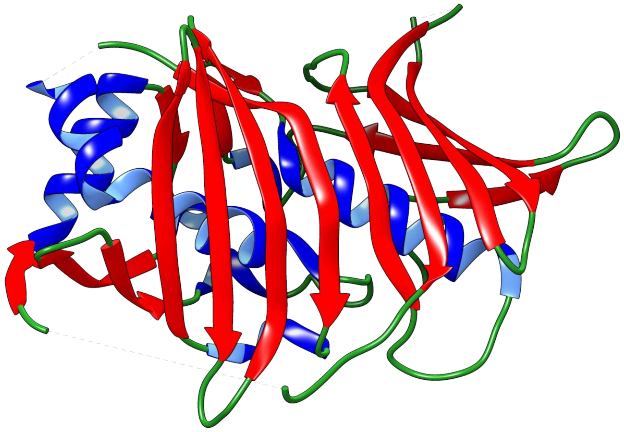


Figure 1.4: 3D cartoon of the the dehydratase domain of PpsC protein from Mycobacterium tuberculosis [3] showing α-helices in blue, β-sheets in red, and loops in green. Created with Chimera [2].

hydrophobic residues than surfaces. Protein cores also show higher evolutionary conservation compared to surfaces [16].

Finally, in many cases multiple polypeptides combine together into a complex [6]. Quaternary structure describes the manner in which the individual polypeptides, known as subunits in this context, combine together to form the complex. Protein complexes can be categorized as either *transient* or *permanent*. This distinction reflects the difference in binding affinity (strength of attraction) between single proteins in the complex, with permanent complexes having higher and transient complexes having lower affinity. Complexes can also be categorized as either *obligate* and *non-obligate*. The constituent proteins of obligate complexes typically exist only in the complex, whereas for a non-obligate complex, each constituent protein exists both in the complex and independently. Transience usually implies non-obligation and obligation usually implies permanence, so a simpler classification of obligate vs. transient is also appropriate [17, 18]. The temporary nature of transient interactions enables complex networks of interaction which give rise to numerous cellular processes and the regulation thereof [18, 19]. Proteins in which all subunits are identical are called homomeric whereas proteins with different subunits are called heteromeric. Homomeric proteins are usually obligate and more conserved than heteromeric proteins [17] [16]

1.1.2 Protein Interfaces And Their Prediction

The locus of a protein-protein interaction is the interface, which is comprised of pairs of residues, one from each interacting protein. These pairs may form a disulfide bond or salt bridge which help anchor the two proteins together [16], but this is not always the case [19]. Residues may not attract each other directly but still be considered part of the interface due to their proximity. This is the case when hydrophobic residues from two proteins appear near each other in the interface, since they do not attract each other, but this configuration is energetically favorable compared to being exposed to water in the surrounding cellular environment [16, 19]. Historically, residues have been considered part of the interface if they are in contact with residues on the adjacent protein. This is typically determined in one of two ways, either the distance between any atom

in the residue in question and any atom in the other protein is below a threshold, or the exposed area of a residue drops sufficiently after complex formation [1, 16, 17, 19].

A survey of known protein complex structures from the Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank (PDB) [12] has shown that interfaces have a higher prevalence of hydrophobic residues, lower prevalence of hydrophilic residues, and more evolutionary conservation compared to non-interface surface regions [16]. The higher overall hydrophobicity of an interface region biases the protein towards configurations in which the interface excludes water by binding to another protein. The higher degree of conservation shows the functional significance of the interfaces, although conservation is somewhat lower for transient compared to obligate complexes [17]. Transient complexes also have a comparatively *lower* hydrophobicity compared to obligate complexes, consistent with the fact that proteins in transient complexes exist outside of the complex, with interfaces exposed to water [17]. To overcome the unfavorable energetic effects of burying more hydrophilic residues in an interface, transient interfaces usually have higher numbers of hydrogen bonds [17]. It has also been shown that transient complexes have less shape complementarity between the participating proteins compared to obligate complexes [17]. Lastly, transient interfaces also tend to either be smaller in size (for weakly interacting complexes) [17, 18], or undergo more conformational change when forming (for strongly interacting complexes) [18] compared to obligate complexes. These differences make transient interfaces more difficult to distinguish from non-interface surface residues [18].

Historically, the problem of interface prediction has been formulated in two ways: *partner-independent* and *partner-specific* prediction. The former variant considers a single residue from a protein and attempts to answer the question: does this residue form part of the interface with some other partner protein? The latter variant considers pairs of residues, each from a different protein, and attempts to answer a more specific question: does this *pair* of residues constitute part of the interface between these two proteins? The pairwise nature of partner-specific prediction allows the consideration of the compatibility of a pair of residues, which has been found to increase

performance [1, 20]. Methods of interface prediction include *docking methods*, *template-based methods*, and *machine learning methods*.

Docking methods predict the 3D bound formation of two proteins in a complex, from which the interface can be extracted. These methods use energy minimization techniques [21,22]. Unlike template and machine learning methods, docking methods do not require a library of known interfaces in order to make good predictions, but are historically poor at accounting for conformational change during complex formation [23].

Template based methods make predictions based on similar known interfaces. The protein of interest is compared to a library of interfaces, and the interface is predicted from the most similar interfaces from the library. Template methods rely on a non-redundant library and can make predictions only when there is a sufficiently close match to an interface in the library [24].

Machine learning methods attempt to directly predict the interface rather than compare against a template complex or predict the bound formation, however they still use information from a library of complexes whose interfaces are known. Machine learning approaches have included use of neural networks and support vector machines [20] [1]. The latest SVM based approach, PArtner-specific Interacting Residue PREDictor (PAIRpred), uses pairwise kernels which operate on pairs of residues and incorporates both sequence and structural information of residues. The method performs well compared to existing docking and machine learning methods [1]. More detail of prior work in interface prediction will be given in Chapter 2.

Chapter 2

Prior Work in Interface Prediction

As previously mentioned, the experimental determination of protein complexes is time and resource intensive, prompting computational modeling approaches. Esmaielbeiki, Krawczyk, Knapp, Nebel, & Deane [25] describe three slightly different computational problems: protein interaction prediction, protein interface prediction, and protein-protein docking. The first problem seeks to identify pairs of proteins which interact, elucidating the complicated protein interaction networks that give rise to cellular processes. The second problem, and the focus of this thesis, is concerned with identifying the specific residues or pairs of residues which make up the interface. The third problem considers two specified proteins and seeks the bound 3D structure of their complex.

2.1 Docking

Docking begins with the known unbound structures of two proteins known to interact, and conducts two main steps: search and scoring. During search, the proteins are translated and rotated relative to each other and brought into contact to create several putative 3D bound structures for the complex. The putative structures are then evaluated by a scoring function to identify the most likely conformation of the complex. Different docking methods differ in their the search algorithms and scoring functions. Scoring functions may incorporate complementarity in geometry, chemistry, and electrostatics, or incorporate van der Walls forces or evidence based (i.e. statistical) potentials [24] [26]. It is worth noting that docking methods can also be used to predict interfaces, by first solving for the 3D structure of the complex and then extracting the interface from the complex. Indeed, docking methods were some of the earliest computational approaches developed for modeling protein interactions [26]. One of the major advantages of docking is its ability to produce interface predictions *ab initio* without requiring examples of known interfaces, which is particularly useful when experimental complex data are sparse or absent. Unfortunately, docking methods traditionally suffer from relatively high false positive rates, are considerably less effective

for complexes which undergo conformational change when binding than those that do not, and are computationally expensive because of the vast search space [26] [24].

2.2 Other Early Methods

Some early alternatives to docking used sequence information, residue properties, and unbound structures for each protein in the complex to directly predict the interface without predicting the structure of the whole complex. Lichtarge, Bourne, & Cohen [27] used inferred evolutionary relationships between different proteins to identify conserved residues and then identified those conserved residues which lay on the surface of the protein. This method was based on the hypothesis that conserved surface residues must be vital to a protein's function and therefore probably constitute an interface. Pazos, Helmer-Citterich, Ausiello, & Valencia [28] took a similar approach, but instead looked at evolutionary relationships between protein complexes and identified pairs of residues between the proteins in the complex which have co-evolved. This method requires only sequence information and therefore is applicable even in cases where the protein structures are unknown, but relies on having sufficient data to infer evolutionary relationships. Additionally, this method is partner-specific since it identifies residue pairs which show correlated changes. Gallet, Charloteaux, Thomas, & Brasseur [29] used a sliding window on a protein sequence and calculated measures of hydrophobicity in a region, which can easily be calculated knowing the residue identities and secondary structure. This method requires no phylogenetic information so is applicable even when no close evolutionary relatives can be identified.

Early methods such as those listed above were crafted for the available data and computational resources of the time, but were unable to fully account for the growing body of research surrounding protein interfaces and their properties, as in Jones & Thornton [17]. It was Jones & Thornton [30] that proposed a method which incorporates multiple structural features such as surface planarity, protrusion, and accessible surface area, with residue level features such as solvation potential, hydrophobicity, and interface residue propensity. They constructed a manual scoring function whose inputs are the aforementioned features and output is a score, where higher scores

are intended to correspond with members of the interface. They constructed a different scoring function for each of three different categories of complex, reflecting observations made into the characteristics of different complex types. These categories were homomeric and small heteromeric proteins, larger heteromeric proteins, and antibody/antigen complexes. Prediction was performed on small patches of residues. Like docking, these methods avoid using examples of known interfaces when making predictions.

Evaluation and comparison of early methods was challenging due to the paucity of experimentally determined protein interfaces [25]. Thankfully, the turn of the twenty first century coincided with an increase in the number of experimentally determined structures added to databases such as the Protein Data Bank [12]. Curated subsets also emerged which focused on evaluating protein-protein docking methods, such as the Critical Assessment of Predicted Interactions (CAPRI) [31] and the Docking Benchmark Dataset (DBD) [32]. These datasets also became useful in the evaluation (and sometimes training) of interface prediction methods.

2.3 Data Driven Methods

The increasing availability of data and increased interest in interface prediction led to a growing number and diversity of approaches. Template based methods emerged which utilize a non-redundant library of known protein interfaces to make predictions about unknown proteins. For a given query protein, a search is made in the library for known complexes where a partner is similar to the query protein. The interface of the query protein is then inferred from the interfaces of the most similar query results. Similarity may be measured via sequence or structural similarity [25].

Other data-driven methods have appeared which are based on either machine learning or statistical methods. Some early machine learning based approaches used a support vector machine (SVM) to classify residues as either belonging to an interface or not. An SVM essentially provides a scoring function which is dependent on training data rather than being manually constructed. Koike & Takagi [33] trained an SVM classifier to perform partner-independent prediction of interfaces. They represented a residue by its profile, a vector of relative abundances of each amino

acid type among homologous proteins at that location. They experimented with different feature representations to make predictions at a particular residue, finding that incorporating profiles from sequential or spatially neighboring residues improves performance, as does incorporating accessible surface area and accounting for the relative interface size. Bradford & Westhead [34] also performed partner-independent prediction with an SVM classifier, but instead made predictions for surface patches rather than individual residues. Zhou & Shan [35] were among the first to train a neural network for partner-independent prediction. They incorporated profile and solvent exposure of residues and their neighbors to make predictions at the residue level. Their method uses residue profiles. In a follow up paper, Chen & Zhou [36] used an ensemble of neural networks to make a consensus prediction concerning a residue of interest.

Various statistical approaches to interface prediction have also been proposed, many of which attempt to model the interdependence between different residues and between residue features. Bradford, Needham, Bulpitt, & Westhead [37] compared a naive Bayes approach to a Bayesian network, which accounts for observed correlations between features, and found that both perform equivalently when predicting interface patches. They also found that these methods perform well even when some data are missing, particularly when conservation scores can't be determined due the absence of homologs. Friedrich, Pils, Dandekar, Schultz, & Müller [38] adapted a hidden Markov model (HMM) originally used for homology detection [39] in order to detect interacting residues. The advantage of an HMM is the ability to jointly model all residues in a sequence at once. Li, Lin, Wang, & Liu [40] generalized this joint modeling to an undirected graphical model using conditional random fields (CRFs) which performs comparably to other data based approaches.

Early machine learning and statistical methods for interface prediction provide predictions at the individual residue or patch level, in contrast to docking methods which generate global solutions for the complex. These methods also typically incorporate both sequence and structural information in order to make partner-independent predictions. However, in a 2007 review paper,

Zhou & Qin [41] identified the need for partner-specific methods in order to improve prediction specificity.

2.4 Partner Specific Methods

Whereas early partner-specific interface prediction methods are based on sequence co-evolution or derived from docking solutions, recent approaches have also included machine learning based methods. Notably, two such methods have incorporated the same types of features as the partner-independent machine learning methods, but have instead considered pairs of residues from separate proteins when making predictions.

In Prediciton of Protein-protein Interacting Position Pairs (PPiPP), Ahmad & Mizuguchi [20] used a neural network which only uses sequence based features. They experimented with two types of sequence based features, a sparse encoding and a position specific scoring matrix (PSSM) encoding. The sparse encoding is a one-hot binary array of length 20 indicating the amino acid type. The PSSM encoding instead represents each amino acid type by its log-odds frequency in iterative multiple-sequence-alignment results. Using these features, they also experimented with different sized sequence-windows, where a residue of interest is represented by the concatenation of feature vectors of all residues inside a sequence window. Both the feature representations and the sequence windows are in keeping with prior work in machine learning based partner-independent predictors. Residues whose windows extended past the end of the bounds of the sequence were excluded from the training set.

The data used are from version 3.0 of the Docking Benchmark Dataset [42], which includes 124 unbound and bound structures of both proteins. A pair of residues, one from each protein was considered positive (part of the interface) if they are within 6Å of each other and negative otherwise. Training examples were created by concatenating the feature representation of each residue in a pair together. Due to the inherent asymmetry of this concatenation, two examples were produced for each pair by concatenating the representations in both orders (AB and BA). Predictions for pairs were made by taking the average prediction of the two orderings. There

are significantly fewer positive than negative examples, so negative examples were sampled to prevent extreme class imbalance. Specifically, either 2% or 1000 negative examples were randomly selected, whichever was smaller.

The model consists of an ensemble of 24 neural networks, each using different window sizes for the sparse and PSSM encodings. The predictions from each of these models are averaged to produce a final prediction for a residue pair. Networks were evaluated in a leave-one-out fashion (train on all but one complex and test performance on the omitted complex). Performance was measured using area under the receiver operating characteristic curve (AUC) for each left out complex and averaged.

The ensemble achieved an AUC of 72.9% compared to 67.9% for a single neural network with windows of size 7 for both encodings. The authors compared this to an analogously constructed neural network ensemble which performs partner-independent prediction. Examples consist of a single residue which is positive if it is part of an interface and negative otherwise. Partner independent predictions were naively converted to partner-specific predictions by averaging the scores of each residue in the pair, yielding an AUC of 71.0%, worse than the partner-specific predictor. Conversely, partner-specific predictions were converted to partner-independent predictions by taking the max over all potential neighbors. This yielded an AUC of 66.1% for the partner-independent prediction problem, better than the 63.8% AUC of the partner-independent model. Thus, the partner-specific model outperformed the partner-independent model on both partner-specific and partner-independent predictions.

In PAIRPred, Minhas, Geiss, & Ben-Hur [1] incorporated custom symmetric kernels into an SVM formulation [1]. In addition, this method includes structural information for each residue as well. The structure based features consisted of the relative accessible surface area (rASA), residue depth, half sphere amino acid composition, and protrusion index. The sequence based features included the same PSSM encoding as PPiPP, a position frequency scoring matrix (PSFM) encoding (like the PSSM encoding but with raw frequencies instead of log-odd frequencies), and predicted rASA (prASA).

The authors used complexes from DBD version 3.0 [42] for comparison to PPiPP. They also utilized the updated DBD version 4.0 [43], which is a superset of the complexes in version 3.0, and complexes from the CAPRI [44] experiment.

The authors constructed specialized symmetric pairwise kernels which compute a similarity between any two pairs of residues, independent of the ordering of each pair. Each pairwise kernel is constructed by taking a symmetric combination of kernels for individual residues, where residue kernels were themselves sums of radial basis function kernels for each feature type. Several individual pairwise kernels are summed and the result is normalized to produce the final pairwise kernel. The authors also investigated a postprocessing step where pair scores are smoothed based on the scores in a neighborhood around each residue. Cross validation was used to tune the soft margin parameter C and the residue kernels and pairwise kernels were optimized in similar fashion. Following the same leave-one-out procedure as Ahmad & Mizuguchi, PAIRPred achieved an AUC of 87.3% before any postprocessing, and 88.7% after post processing. When using only sequence based features, PAIRPred achieved an AUC of 80.9%, which demonstrates a significant improvement over PPiPP even in the absence of structural features. Partner-independent prediction was performed by taking the maximum score across all potential neighbors, and achieved an AUC of 70.8% and 77.0% using only sequence features and all features respectively, which also outperforms PPiPP's best partner-independent performance.

Minhas et al. [1] note that the extreme class imbalance inherent to the partner-specific classification problem means AUC is not as easy to interpret. Therefore they also calculate the rank of the first positive prediction (RFPP) at a given percentile, where $\text{RFPP}(p) = q$ means that p of the complexes in the test set have at least one true positive pair among the top q predictions. Low values of q corresponding to high values of p indicate better classifier performance because a higher percentage of complexes have a true positive near the top predictions. The authors argue that this is more relevant to a biologist because it indicates the trustworthiness of the top few predictions of the classifier. PAIRPred outperforms PPiPP for values of p equal to 10, 25, 50, and 75, but is worse at the 90% level.

The authors conclude by noting that there is much room for improvement in partner-specific interface prediction, particularly for complexes with a high degree of conformational change. They propose adding new features to PAIRPred which capture shape complementarity between binding interfaces, co-evolution, and protein flexibility. However, PAIRPred's improvement over PPiPP suggests that not only is the choice of features important to classifier performance, but also their representation and the construction of the predictor itself. This supports the investigations of this thesis into the graph representation of proteins and new convolution methods which operate on graphs. Chapter 3 gives a primer on the convolutional neural networks which helped inspire these new methods.

Chapter 3

Artificial Neural Networks

Artificial neural networks are a class of machine learning algorithms which are loosely inspired by neuroscientific models of how biological neuronal networks operate. Early work in neuroscience by Santiago Ramón y Cajal indicated that the nervous system is composed of complex networks of individual cells, later called neurons [45]. These neurons consist of a soma and axon. The soma is the main part of the cell, and the axon extends from the soma, attaching to one or more somata from other neurons. A neuron creates a so called "action potential" if it is sufficiently excited by the axons attached to it, which propagates down its own axon to help excite or inhibit other neurons to which it is attached [46]. In this way, electrical signals are propagated through a neronal network. Work by Hodgkin & Huxley described mathematically how action potentials are generated in neurons and propagate from one neuron to another [47].

Artificial neural networks mimic this behavior with a mathematical network of *artificial* neurons. Mathematical signals propagate through this network, whereby each neuron performs a simple mathematical operation on its inputs to determine its output. *Feed forward artificial neural networks* are a subclass of artificial neural networks which contain no cycles, that is, there is no sequence of input/output connections that start and end at the same neuron. This class of artificial neural network always contains at least one input neuron with nothing feeding into it, and at least one output neuron which feeds into nothing. They therefore constitute a function, where the input is applied to the input neurons, the signal propagates through the network, and the output is extracted from the output neurons. If the neuron operations are parameterized, then it may be possible to approximate various other functions through the appropriate selection of parameters. Let the function defined by a feed forward artificial neural network be denoted by $f(x|\Theta)$, where x is the input vector and Θ is the complete set of parameters.

For feed forward neural networks, f is computed in a sequence of layers, each consisting of some number of neurons, which perform comparatively simple mathematical operations on their

inputs to produce an output. A layer's output is either used as input to the subsequent layer, or taken as the network output if there are no subsequent layers. By convention, the input vector x is considered the first layer of a network, with one neuron per input, where each neuron takes the value of the respective input. The last layer in the network produces the function's output, either a scalar or vector of values. All intermediate layers are termed *hidden layers*, each of which accepts the output of the preceding layer and produces one output per neuron in that layer. The number of layers in a network (besides the first layer) is its *depth*. Mathematically, we can express the output for layer i as $h_i(x_i|\theta_i)$, where x_i is the input to the layer and θ_i is the set of parameters for that layer.

Some important notes on terminology: artificial neurons are also called *units*, which will be the preferred term hereafter to avoid confusion with biological neurons. Also, omitting the *artificial* from artificial neural networks is common parlance in the literature, and will be done in this thesis for convenience. To avoid possible confusion, biological neuronal networks will always be explicitly referred to as such.

A common form of neural network layer is a *dense layer*, in which each unit calculates a weighted sum of the layer inputs, where the set of weights is unique to each unit. It is also common to add a scalar bias term and apply a nonlinear activation function to the weighted sum. Dense layers have a convenient mathematical representation:

$$h_i(x_i|W, b) = \sigma(W_i x_i + b_i), \tag{3.1}$$

where i indexes the layer, W_i is a matrix of weights, x_i is the layer input, b_i is a vector of biases, and $\sigma(\cdot)$ is a nonlinear activation function. If there are n inputs to a layer and m units in the layer, then $W \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$. The expression $Wx + b$ is the *signal*, and h is the *activation*. For hidden layers, activations have traditionally taken the form of a sigmoidal, or "s-shaped", function such as the logistic function, $\sigma(x) = 1/(1 + e^{-x})$, or hyperbolic tangent, $\sigma(x) = tanh(x)$. These sigmoidal functions switch from "off" (low value) to "on" (high value) if the weighted sum in the argument is sufficiently high. This loosely mimics the way that signals from attached axons

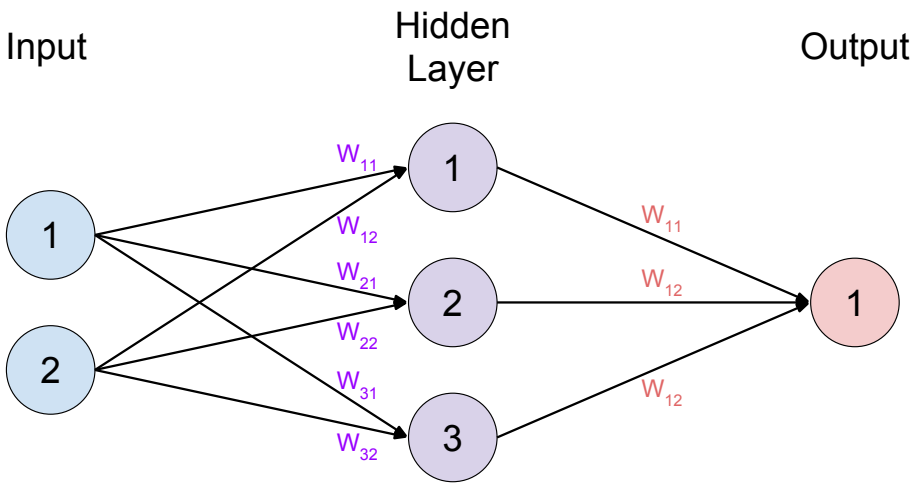


Figure 3.1: A two layer neural network with two inputs, three hidden units, and a single output. The hidden layer and output are both dense layers, and arrows depict the weighted sum computed for each unit. Weights for each layer are numbered according to their index in the weight matrix W

combine to trigger an action potential in a neuron. Figure 3.1 shows a graphical depiction of a parameterized, feed forward artificial neural network.

Despite being conceptually simple, this formulation of neural networks is capable of approximating any continuous function on a compact subset in $\mathbb{R}$ with arbitrarily small error, given enough hidden units [48]. The challenge of using neural networks for function approximation is in finding the appropriate set of parameters to approximate the true function. For simple feed forward neural networks, this is accomplished by quantifying the error between the true function $Y = (y_1, y_1, ..., y_N)$, and the approximation for a given training data set $X = (x_1, x_2, ..., x_N)$, using a differentiable loss function $L(\Theta|Y, X)$. Recall that Θ is the total set of neural network parameters, and note that L is usually a function of the network output and the true output for a The weights are updated by differentiating L with respect to Θ and "taking a step" in parameter space opposite the direction of the gradient. This update step can be repeated iteratively in a process called *gradient descent*, which was first proposed by Augustin-Louis Cauchy [49]. Mathematically, parameters are updated according to:

$$\Theta_{k+1} = \Theta_k - \eta \nabla L(\Theta_k|Y, X) = \Theta_k - \eta \sum_{n=1}^{N} \nabla L(\Theta_k|y_n, x_n), \tag{3.2}$$

where k indicates the iteration and η is a tunable step size. The gradient can be efficiently calculated using an algorithm called *backpropagation* [50]. Weights are usually initialized by drawing from random distributions that have some empirical or theoretical justification [51]. A variant of this algorithm, called *stochastic gradient descent* (SCG) performs an update using a gradient computed from a random sample of the training data at each iteration. There is also a batched version where the training data are randomly shuffled and divided into a fixed number of equal sized *mini-batches*, and an update is performed on each one. When all mini-batches have been used in an update step, this constitutes an *epoch*. Training can be repeated for a fixed number of epochs or until the parameters sufficiently converge. Gradient descent is a first order method because it uses only the gradient of the loss function, however various second order methods which incorporate the Hessian have also been used to train neural networks [52–55].

Like many machine learning algorithms, neural networks risk *overfitting*, in which the network learns to approximate the training data (which usually contain noise), rather than the underlying function from which the training data are assumed to be drawn. This hinders the ability of neural networks to generalize to unseen data. In neural networks, overfitting is commonly the result of an overcomplete parameterization coupled with overtraining [56, 57]. Many regularization techniques have been introduced to prevent overfitting of neural networks, including early stopping [58], model pruning [56], weight decay [59], lateral inhibition [60], induced sparsity [61,62], and dropout [63], with dropout being one of the most common.

Dropout consists of randomly dropping units from the network during the training phase. Each time a training example is presented to the network, each unit in the network is dropped with probability p, commonly 0.5. Each application of dropout can be seen as generating a new network whose units are a subset of the units in the original network. Given n units in a network, there are 2^n possible dropout networks which all share weights with each other. During testing, no dropout is performed, and weights are rescaled by p. This is equivalent to averaging the prediction of each dropout network. Dropout helps prevent overfitting by limiting the effective number of parameters in each dropout network and reducing the amount of co-adaptation among units in the network [63].

In recent years, more advanced forms of neural networks under the moniker *deep learning* have demonstrated success in sophisticated tasks, particularly for image and speech recognition [60,64–67]. A common task is labeling an image or audio sample according to its content. Labeling tasks are commonly decomposed into many binary tasks, where a neural network must indicate whether a particular label is appropriate. Positive examples are those where the label is appropriate, and negative examples are those where it is not. Deep learning has also been applied in a variety of other applications such as quantitative structure activity relationship (QSAR) prediction [68], particle detection [69], and reinforcement learning [70]. These advances were catalyzed by the availability of large volumes of labeled data [71, 72] and the improvements in computing power from general purpose graphical processing units (GP-GPUs) and distributed systems [73,74]. Such factors allow experimentation with larger networks and more complicated layer operations such as convolution, which are described in the following section.

3.1 Convolutional Neural Networks

Accurately labeling an image using a neural network often relies on the network's ability to detect low level features in the image such as edges and textures, which collectively indicate the image's content [61]. This is accomplished by the appropriate parameter settings, which maximize a unit's activation when presented with a feature of interest, and can be learned through the standard backpropagation algorithm using training images. The challenge, however, is that the relevant features may only occur in certain regions of the training images, so the network may not be able to recognize that feature in a new region. Convolutional layers explicitly allow detection of features in a translation invariant way, through a clever weight sharing strategy.

To illustrate the limitation of dense layers in translation invariant feature detection, consider the following example. Suppose the labeling task is to determine if an image does or does not contain a cat. Suppose further that no training image contains a cat in the upper left corner of the image. Therefore it is possible, even if the network has learned which low level features indicate the presence of a cat, that it will not recognize a cat contained in the upper left corner of the image,

since it has not been explicitly shown examples of this. This may seem unlikely, but consider that the network does not incorporate any semantic meaning of the word "cat", only that a certain set of example images are positive examples and a certain set are negative examples. Indeed, consider the slightly different labeling task where the classes being considered are "contains a cat anywhere but the top left corner" and its negation. In this case, the same set of training images are appropriate and the hope is that the network would *not* classify images with cats in the upper left corner as positive examples. In most cases, the labeling task is more like the former example than the latter, in which case it would be beneficial for the network to learn to detect objects in a *translation invariant* fashion. This is the primary purpose of convolutional layers.

The term convolution is inherited from the field of functional analysis, where two functions, f and g, are combined in the following way to generate a third function, $(f * g)$:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau. \tag{3.3}$$

The convolution involves reflecting one function about the y axis, shifting it with respect to the other function, and integrating the product. The value of t indicates the magnitude of shift. If $f, g \colon \mathbb{Z} \to \mathbb{R}$, then the discrete convolution is analogously defined as:

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)\, g(t - \tau). \tag{3.4}$$

where $t, \tau \in \mathbb{Z}$. If we imagine f and g to be infinite dimensional vectors indexed by τ, then convolution amounts to reversing g, shifting it by t units relative to f, and taking the dot product of the two vectors. The same operation can be defined for two finite vectors by making them infinite by appending infinitely many zeros before and after them. The zero portions contribute nothing to the convolution and allow the original finite vectors to be partially overlapping. A finite vector can be extracted from the result, since the convolution function is nonzero only when the convolved vectors overlap. Figure 3.2 illustrates the analogy between continuous convolution and discrete convolution of finite vectors.

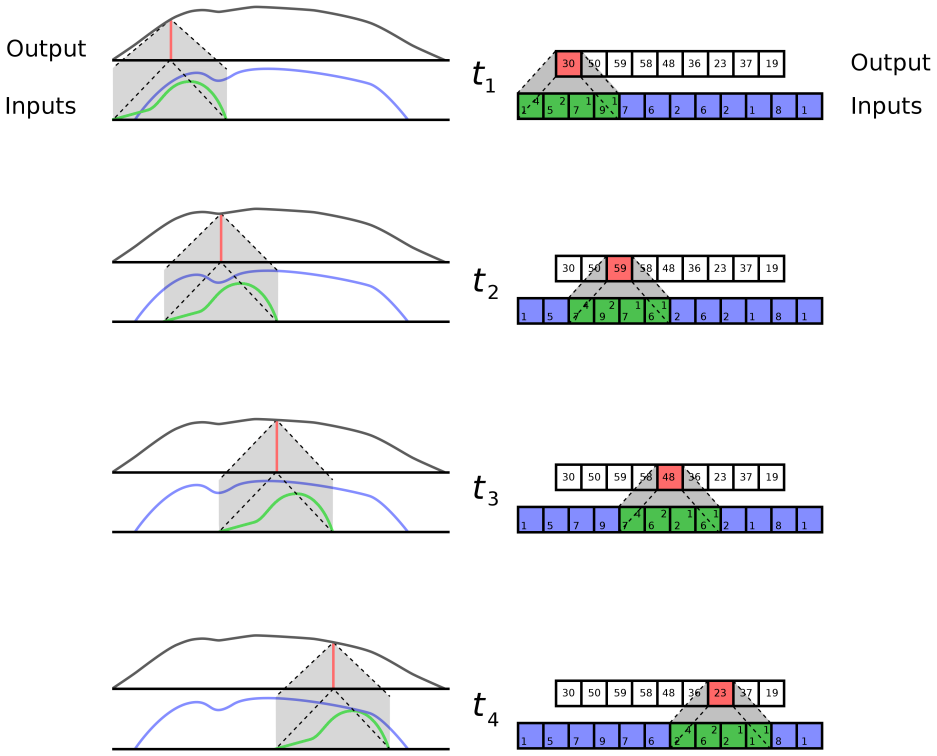


Figure 3.2: The relative position of two functions at four different values of t, for both continuous (left) and discrete (right) convolution. The input functions are blue and green, whereas the output for that time is indicated in red. In the continuous case, the functions are multiplied and integrated whereas in the discrete case, the dot product of the two arrays is taken (extending either array with zeros when necessary). Note that numbers shown are for illustration purposes only.

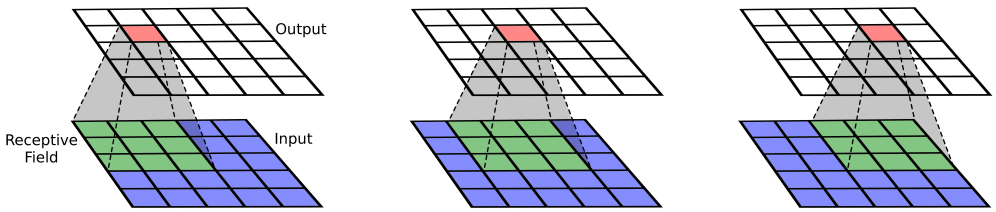


Figure 3.3: Application of a 3x3 convolution filter to an input image. The dot product is taken between the filter (green) and the function (blue) to produce a scalar output (red). Three positions of the filter are shown, each with a unique receptive field and output position.

Convolution can be extended for functions of two variables:

$$(f * g)(t_1, t_2) = \sum_{\tau_1=-\infty}^{\infty} \sum_{\tau_2=-\infty}^{\infty} f(\tau_1, \tau_2)\, g(t_1 - \tau_1, t_2 - \tau_2), \tag{3.5}$$

where $t_1, t_2, \tau_1, \tau_2 \in \mathbb{Z}$. Here, f and g can be interpreted as infinite matrices, where convolution involves reversing g in both indices, shifting it in both indices relative to f, and taking the dot product with f. Again, this may be adapted for two finite matrices using the same infinite extension of zeros as for vectors. This discrete, two dimensional, finite analog to convolution forms the basis of convolutional layers in neural networks.

To understand convolution of a monochrome pixelated image, consider it as a finite matrix where the matrix entries contain pixel values. Further, consider a smaller matrix called a *filter* which entries are filter *weights*. By convolving the filter over the image (as previously described), a new matrix is created where each value is the dot product of the matrices for a given shift of the filter relative to the image. Technically the filter can be assumed reflected already, since the filter weights are trainable parameters anyways. The region of the image covered by the filter is called the *receptive field*, and the dot product can be understood as taking a weighted sum of values in the receptive field, where the weights are precisely the filter weights. As with dense layers, each weighted sum is typically passed through some nonlinear activation function. Figure 3.3 depicts the application of a filter to the input of an image in three distinct positions. The dot product between filter and image naturally extends to images with multiple *channels* (colors), as long as the filter has the same number of channels as the image.

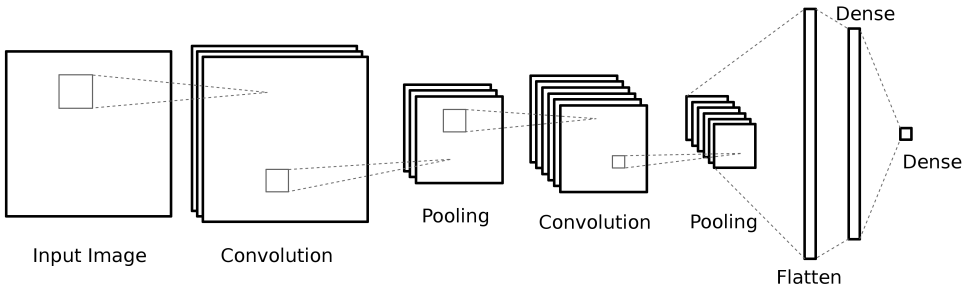


Figure 3.4: A convolutional neural network, consisting of alternating convolution and pooling layers, followed by two dense layers, producing a scalar output. Stacked rectangles indicate multiple channels. These convolutional layers increase the number of channels, which is typical but not necessary. Pooling downsamples the image but retains the same number of channels.

The result of a convolution in a particular position depends on the values of the input image and the weights, and indeed there exist weights which produce larger outputs only in the presence of particular local features in the input. Hence convolution can be used in a neural network to detect patterns in the input. Furthermore, since a filter is convolved across the whole image, feature detection can be performed in a translation invariant way as desired. A filter may only detect a particular local feature, so multiple filters can be used to detect multiple features simultaneously, each producing a different channel in the output. Note that the output of a convolution layer resembles the structure of the input, so convolutional layers may be stacked.

Convolutional neural networks often incorporate *pooling layers*, which reduce a region of the input (e.g., 3x3 pixels for an image) to a single grid element that takes either the maximum or average value (per channel) of the region. For simplicity in this thesis, grid elements will be called pixels, regardless of grid dimensionality. Pooling allows downsampling of the grid, and is often performed between convolutional layers. It's also common to include a series of dense layers after all convolutional layers which combine the results of convolution in a way relevant to the learning task. In order to do this, the multichannel input must first be flattened into a one dimensional vector. Figure 3.4 shows an example of a complete convolutional neural network.

The design of convolutional layers gives rise to certain properties which prove useful in image related tasks:

- *Structural awareness.* Convolutional filters are constructed so that they exploit the structural characteristics of the input. Rather than treat each pixel as independent of all others, convolution operates on local regions of pixels, allowing filters to detect spatial correlations in a straightforward manner.

- *Locality*. Filters are typically significantly smaller than the size of the input, so for a given position, they are operating on a local neighborhood (receptive field) of the input image. Any features learned by a filter will necessarily be local as well. The advantage of local features is that they are simpler and more universal than global features, since they can be combined to construct a wide variety of complex patterns (see *abstraction* below). For example, filters which detect edges or textures will be more useful to subsequent layers in the network than a filter which recognizes only a single object or animal.

- *Translation invariance*. As discussed, a filter "inspects" each part of the image for a particular pattern, regardless of where that pattern occurred in the training images. This improves generalizability because during training certain features may be presented in limited regions of the image, but detection of those features can still operate on a global scale.

- *Abstraction*. Whereas a network's first convolutional layer detects patterns in the input, subsequent layers detect patterns in the output of previous layers. This promotes a hierarchical abstraction of the input, where early layers detect local features, and subsequent layers combine features into more complex patterns. For example, early filters may detect edges of varying orientations, middle filters may detect combinations of edges which indicate curves in space, and latter filters may detect combinations of curves which indicate a particular handwritten letter or digit.

- *Resistence to overfitting*. One interpretation of a convolutional layer is that each filter is a series of units, each responsible for a distinct receptive field on the input. A unit has nonzero weights only for the regions of the input in its receptive field, and all units in the filters share the same weights. This combination of weight sharing and sparsity make a convolutional

layer less prone to overfitting compared to a dense layer with the same number of units. Because of the weight sharing in convolutional layers, the final dense layers often contain the majority of the parameters in the network.

The above properties make convolutional neural networks well suited for structured data with a natural hierarchy of abstraction. The structural hierarchy of proteins suggests that convolution might be useful in interface prediction. Unfortunately, neither protein residues, nor their constituent atoms are aligned in a regular grid, so the above definition of convolution cannot be directly applied. In Chapter 4, convolutions are introduced which operate on irregular structures, namely graphs, and a graphical representation of proteins is presented which allows use of graph convolution for protein interface prediction.

Chapter 4

Pairwise Graph Convolutional Networks for Interface Prediction

The methods presented in this thesis were motivated by a desire to exploit the local structure around a residue when performing interface prediction. The biological reasoning for this is that a residue's neighborhood influences its propensity to participate in an interface. It was noted in Chapter 3 that convolutional neural networks are one way of detecting features in a local neighborhood, but they are limited to regular grids. Unfortunately, proteins cannot naturally be represented as a regular grid, so convolution must be developed for a more natural representation: graphs.

4.1 Proteins As Graphs

An undirected, unweighted graph G consists of a set of vertices, $V = \{v_1, v_2, ..., v_n\}$, and a set of edges, $E = \{e_1, e_2, ..., e_m\}$ where each edge is incident to two vertices and there is at most one edge between two vertices. One way of representing proteins as graphs is to let each vertex represent a residue in the protein, and each edge represent the relationship between two residues. Thus any information pertaining to a particular residue can be associated with the relevant vertex in the form of a feature vector. The features used in this work are drawn from features used in prior interface prediction work [1]. Vertex features include the folloeing:

- *Position specific scoring matrix*, which measures the prevalence of each amino acid type amongst protein domains similar to a region around the residue of interest.
- *Relative Accessible Surface Area*, which reflects the fraction of a residue that is exposed to a potential solvent.
- *Residue Depth*, which is the minimum distance from the residue to the surface of the protein.

- *Protrusion Index*, which measures the degree to which a sphere of radius 10 Å is filled with atoms.
- *Hydrophobicity*, which indicates the tendency for an amino acid to avoid water.
- *Half Sphere Amino Acid Composition*, which captures the amino acid composition in two half spheres around the residue of interest.

Likewise, any information about the relationship between two residues can be associated with the relevant edge. The edge features used here include:

- *Inter-residue distance*, which gives greater weights to residues that are closer together.
- *Relative orientation*, which captures the normalized angle between the peptide plane of each residue.

These edge features are defined between any two residues in the protein, so the graph is complete. A detailed explanation of each feature is contained in Appendix A.

This representation is an abstraction of the original protein to a well studied mathematical object, with two notable facts. First, the graph itself does not rely on a coordinate system, as is the case when working with raw 3D positions. Second, the features contained on the graph are also not tied to a coordinate system. This is because they either describe coordinate free attributes of individual residues, or they describe relative spatial relationships between two residues. These facts make a protein graph invariant to rotations or translations in space. However, since the graph was constructed from points in 3D space, local neighborhoods of vertices can be defined using spatial proximity. This is useful when designing convolutions that use a local neighborhood as a receptive field. We now turn our attention to graph convolution

4.2 Graph Convolution

Recent years have seen increased attention to problems involving graph structured data, prompting developments in graph convolution to perform various tasks on those data [75]. These developments allow leveraging of deep learning techniques, which have shown great success for problems

on regular grids. Though each variant of graph convolution is tailored to suit the data and problem being addressed, there are some common approaches which merit describing generally. These approaches generally fall into two categories: *spectral* and *spatial*.

4.2.1 Spectral and Spatial Graph Convolution

Spectral methods of graph convolution are based on linear functions in the frequency domain of a graph, defined using the laplacian operator $\mathcal{L} = I - D^{-1/2}WD^{-1/2}$, where I is the identity matrix, W is a matrix of edge weights, and D is a diagonal matrix containing the degree of each vertex [76–78]. Each filter in a spectral convolution implies a weighting of each frequency in the spectral decomposition of the graph [79]. In physics, the Laplacian operator is used to model the process of diffusion. In a similar way, the graph Laplacian can be thought of as modeling diffusion along the edges of the graph.

Spatial approaches instead directly model the diffusion process through a transition matrix, or by defining operations in a localized neighborhood of a central vertex [77, 80]. In the latter, each neighborhood constitutes a receptive field where a convolution operation is performed (see Figure 4.1). Spatial convolution commonly involves a vector of weights and takes a weighted sum of neighbors, much like a discrete convolution on a grid can be viewed as taking a weighted sum of pixels within the receptive field. Spatial convolution is more directly analogous to grid based convolution as described in Chapter 3. There is a critical difference, however, in that receptive fields from different parts of a graph have no natural correspondence with one another.

4.2.2 Receptive Field Correspondence in Spatial Convolution

Grid receptive fields have defined positions, such as "upper left most" or "bottom middle". Therefore weights can be applied consistently across receptive fields such that each weight is always applied to the same position. With graphs, there is often no such correspondence between receptive fields, since vertices are inherently unordered and lack a specific position (recall their coordinate free nature). The only well defined position is the center, but the problem persists in the neighbors. See Figure 4.2. In fact, even the number of neighbors may vary from one receptive field

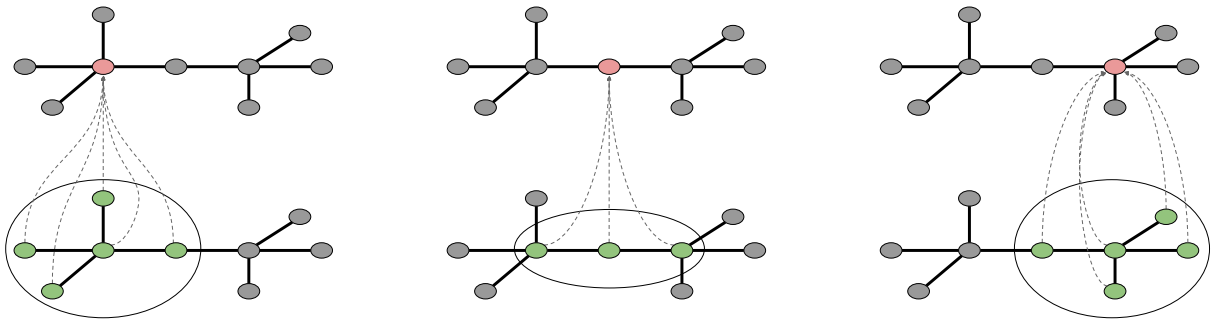


Figure 4.1: Receptive fields in a graph context, where each receptive field is defined around a central vertex. The result of convolution is applied to the central vertex in the receptive field.

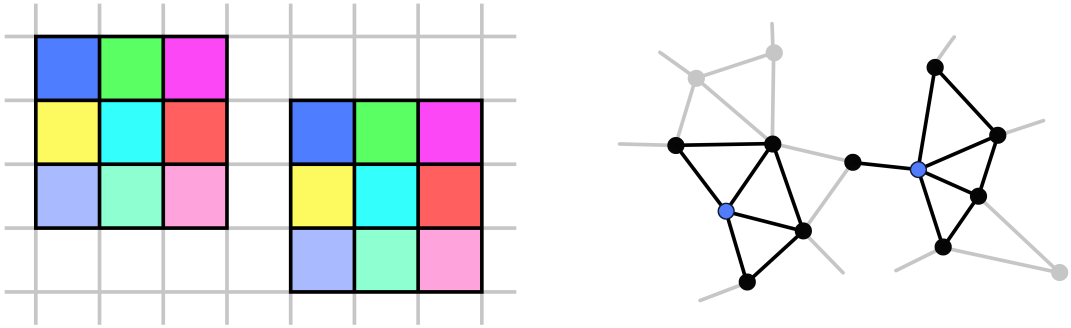


Figure 4.2: The difference between grid receptive fields and graph receptive fields with respect to correspondence. Grid receptive fields give rise to positions (denoted by color) which are consistent from one receptive field to another. Graphs have no positions other than the central vertex (in blue).

to another, depending on the definition of a local neighborhood. Hence the consistent application of weights across receptive fields is only possible after these issues have been addressed. This is typically done in one of two ways:

1. *Imposed ordering of neighbors.* This approach establishes a correspondence between two receptive fields by ordering the neighbors in each and associating neighbors that share a common position. Ordering methods are either based on vertex characteristics, like degree and betweenness centrality, or domain specific knowledge [81, 82]. They typically require the number of neighbors in a receptive field to remain the same. This approach allows filter weights which are applied to a particular position in the ordering, which, it is assumed,

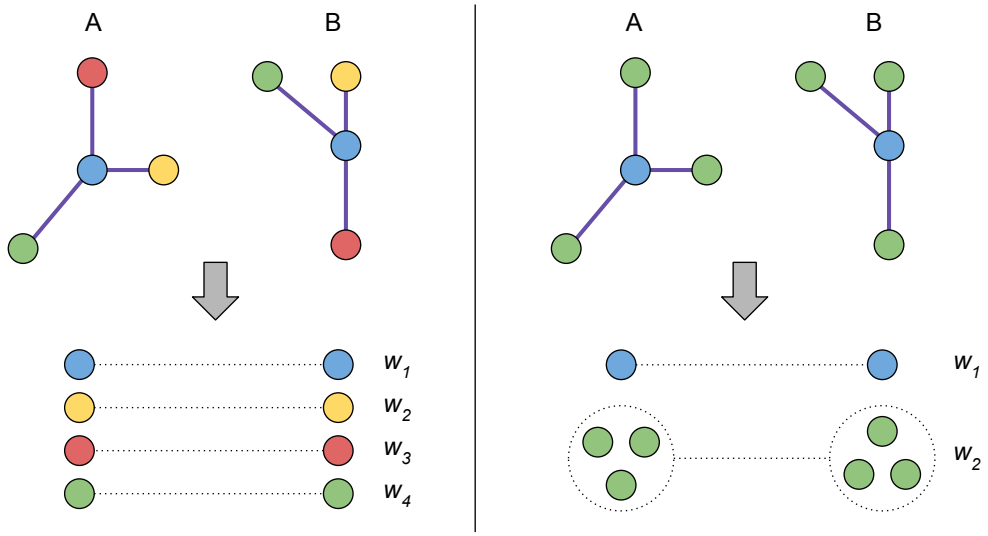


Figure 4.3: Two approaches of establishing correspondence between the neighbors of receptive fields A and B. Central vertices are shown in blue and neighbors in green. The central vertices always correspond with one another. Left: neighbors are ordered and placed into correspondence based on position. Unique weights (w_2–w_4) can then be applied to each position in the order. Right: neighbors are left unordered and treated identically. This requires that the same weights (w_2) be used for all neighbors.

has some significance across all receptive fields. If the imposed ordering is arbitrary, this approach has limited utility. Methods following this approach can be called *ordered*.

2. *Identical treatment of neighbors*. This approach ignores the need to establish a correspondence between receptive fields and instead treats all neighbors identically. Rather than apply different weights to neighbors depending on their position in an ordering, the same weights are applied to each neighbor. This allows for different sizes of receptive fields and avoids choosing an ordering method, but lacks the ability to treat each neighbor uniquely. Methods following this approach can be called *order-free*.

Figure 4.3 depicts both approaches. Examples of each were evaluated for this thesis. In addition, proposed convolutions were examined which attempt to incorporate the advantages of both ordered and order-free methods. Below is a description of each existing method followed by the proposed methods presented in this thesis.

4.2.3 Diffusion Based Method

As mentioned, spectral methods utilize the Laplacian operator, which is used to model diffusion processes. Although no spectral methods were examined in this thesis, a spatial diffusion method was. Atwood & Towsley [80] proposed a Diffusion Convolutional Neural Network (DCNN) which converts a graph's weight matrix to a transition matrix by normalizing its rows. The transition matrix P is raised to successive exponents, generating a power series, with each power corresponding to all walks of length equivalent to that power. For a maximum power of L, the activation of a vertex is:

$$h(x_i|W) = \sigma\bigg(\sum_{l=0}^{L}(W_{l\cdot} \odot P^l X)\bigg), \tag{4.1}$$

where W is a matrix of weights, $W_{l\cdot}$ is its l^{th} row, $\odot$ denotes broadcasted elementwise multiplication, and X is a matrix of all vertices, where each row is a vertex and each column a different feature. For each power l, all vertices that have a random walk of length l that end at the vertex of interest are summed together, weighted by the walk probability. This sum is then multiplied elementwise with a weight vector and summed with the result of all other powers to produce the overall signal. Unlike other spatial convolutions presented in this thesis, this method does not use a receptive field, instead relying on the similarity matrix to indicate proximity.

4.2.4 Ordered Method

Niepert, Ahmed, & Kutzkov [81] described an ordered graph convolution, *PATCHY-SAN*, which performs classification at the graph level. This method constructs local neighborhoods and orders neighboring vertices according to a *normalization* procedure. The ordered vertices serve as the receptive field for convolution at the central vertex, which is also in the ordering. If $(x_1, x_2, ..., x_k)$ is the ordered list of vertices, then convolution takes the form:

$$h(x_i|\{W_j\}, b) = \sigma\bigg(\frac{1}{k}\sum_{j=1}^{k}(W_j x_j) + b\bigg), \tag{4.2}$$

where W_j is the weight matrix for position j in the ordering. A natural ordering technique is one which places the central vertex first and orders its neighbors from least to greatest distance from the center. In this way, one weight matrix is used for all central vertices, another for all nearest neighbors, etc. The vertex ordering also imposes a lexicographic ordering on the neighborhood edges as well, so unique weights can be used to generate a signal from each edge. Adding this term to the convolution generates:

$$h(x_i|\{W_j\},\{W_{jk}\},b) = \sigma\left(\frac{1}{k}\sum_{j=1}^{k}(W_j x_j) + \frac{1}{k^2}\sum_{j=1}^{k-1}\sum_{l=j+1}^{k}(W_{jk}A_{jk}) + b\right), \tag{4.3}$$

where W_{jl} is the weight matrix associated with edge (j,l) in the lexicographic ordering, and A_{jk} is the feature vector associated with edge (j,k). Originally, Niepert, et al., convolved vertices and edges separately and combined them in a subsequent layer. Classification is then performed on the whole graph level. For interface prediction however, the edges and vertex signals are averaged together and applied to the central vertex so that repeated graph convolutions may be performed without losing the graph structure.

4.2.5 Order-Free Methods

One of the simplest forms of order-free graph convolution was proposed by Duvenaud & Maclaurin, et al. [82], which was used for generating molecular *fingerprints*. This method uses a single set of weights for all vertices in the receptive field, including the central vertex:

$$h(x_i|W,b) = \sigma\left(Wx_i + \frac{1}{|\mathcal{N}_i|}W\sum_{j\in\mathcal{N}_i}(x_j) + b\right), \tag{4.4}$$

where $\mathcal{N}_i$ is the index set of all neighbors of x_i. In the original formulation, no bias term was included, and the sum was not divided by the neighborhood size. However, when used for interface prediction, including the bias and normalization provided better overall results.

As mentioned, center vertices may be treated separately from the neighbors, even if all neighbors are treated identically. Schlichtkrull & Kipf [83] proposed such an approach called *Relational*

Graph Convolutional Networks, or R-GCN. Their methods were developed for use in knowledge bases, graph structures where the vertices are named entities and the edges capture the many binary relationships between the entities. To convolve a vertex, they take the neighborhood defined by each relation type and sum the signal from all the neighbors in that neighborhood. The resultant signal is the sum of signals from each relation type. For protein graphs, spatial proximity is the only method for determining neighborhoods that makes sense biologically, so there is only one neighborhood. The adapted version of this convolution for use in interface prediction is:

$$h(x_i|W^{\mathrm{C}}, W^{\mathrm{N}}, b) = \sigma\left(W^{\mathrm{C}}x_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}(W^{\mathrm{N}}x_j) + b\right), \tag{4.5}$$

where separate weight matrices, W^{C} and W^{N}, are used for the center and neighbors, respectively. In the original version, a different weight matrix was used for each of the many relation types. To tie learning across all relation types, all weight matrices were simply linear combinations of a set of learnable basis weight matrices:

$$\begin{aligned} W^{\mathrm{N}} &= \sum_b a_b^{\mathrm{N}} V_b \\ W^{\mathrm{C}} &= \sum_b a_b^{\mathrm{C}} V_b \end{aligned} \tag{4.6}$$

where $\{V_b\}$ is a set of basis matrices, and a^{C} and a^{N} are scalar weights that combine the basis matrices to create W^{C} and W^{N}, respectively. For interface prediction, R-GCNs were examined both with and without basis matrices.

The above order-free methods do not incorporate edge information. Schütt, Arbabzadah, Chmiela, Müller, & Tkatchenko [84] proposed a version called *Deep Tensor Neural Networks* (DTNN) which creates a signal for the neighbor vertices as well as the edges which connect them to the central vertex:

$$h(x_i|W, W^{\mathrm{N}}, W^{\mathrm{E}}, b^{\mathrm{N}}, b^{\mathrm{E}}) = x_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\sigma\left[W\left((W^{\mathrm{N}}x_j + b^{\mathrm{N}}) \odot (W^{\mathrm{E}}A_{ij} + b^{\mathrm{E}})\right)\right], \tag{4.7}$$

where $\odot$ denotes the elementwise product. In this formulation, W^{N} and W^{E} transform vertices and edges, respectively, into a common space, and W transforms their combination to have the same dimensionality as the input. This convolution does not transform the input by any weight matrix, rather it can be viewed as updating the representation at x_i using information from its neighbors. This restricts representations to always have the same dimensionality, which is not generally required in convolutions. Again, the normalization is omitted from the original formulation, but consistently improves performance when performing interface prediction. Nevertheless, this method uniquely incorporates edge information, so that even while all neighbors use the same weight matrix, the information on their edges is used to differentiate them. This concept is carried forward into the convolution methods proposed below.

4.2.6 Proposed Order-Free Methods

Like DTNN, these methods incorporate edge information to help differentiate neighbors, but also avoid imposing an arbitrary ordering on the neighbors in a receptive field. This is accomplished by incorporating information from the edges between each neighbor and the central vertex. Here are two variants of graph convolution which differ only in how the edge information is incorporated, denoted *Sum Coupling* and *Product Coupling*. For a central vertex i on the graph and a local neighborhood of vertices $\mathcal{N}_i$, the output of Sum Coupling graph convolution is:

$$h(x_i|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{E}}, b) = \sigma\left(W^{\mathrm{C}}x_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}(W^{\mathrm{N}}x_j + W^{\mathrm{E}}A_{ij}) + b\right), \tag{4.8}$$

where x_i is the feature vector associated with vertex i, W^{C}, W^{N} and W^{E} are weight matrices, and b is a vector of biases. Intuitively, this calculates an activation for the central vertex, each neighbor vertex, and each edge between a neighbor and the central vertex separately. It is the inclusion of edge activations that allows each neighbor to be distinguished from the others on the basis of its

relationship to the central vertex. This variant is called Sum Coupling because the neighbor vertex and edge activations are added together. Because of this, the direct association between a neighbor and its edge is lost. A variant which maintains the association is Product Coupling, for which the output is:

$$h(x_i|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{E}}, b) = \sigma\left(W^{\mathrm{C}}x_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}(W^{\mathrm{N}}x_j \odot W^{\mathrm{E}}A_{ij}) + b\right), \tag{4.9}$$

where $\odot$ denotes the elementwise product between two vectors or matrices. This allows a neighbor's influence on the overall activation to be modulated by its relationship to the central vertex. For protein graphs, this means neighboring residues will contribute more or less to the overall activation, depending on their distance from and relative orientation to the central vertex, with the precise modulation determined by the edge activations.

In relation to prior methods, Sum Coupling is most similar to Fingerprint and R-GCN convolutions. Unlike Fingerprint convolutions, it uses separate weights for central and neighbor vertices. Unlike R-GCN convolutions, it does not use basis functions which tie together all weight matrices. Also, Fingerprint and R-GCN convolutions do not generate signals from the edges. Product Coupling is similar to DTNN convolutions in the way vertex and edge information is coupled together, but unlike DTNN it calculates a signal for the central vertex and allows the number of channels/features to change layer by layer.

The receptive fields are always defined around a central vertex, so the results of convolution can be applied to that vertex. This retains the graph structure after each convolution, so convolutional layers are stackable, just like convolutions on grids.

A note on receptive fields: protein graphs are complete and embedded in a metric space, so we can define a receptive field using a fixed number of closest neighbors to the central vertex. A receptive field can also be defined using a threshold $\delta > 0$ such that all vertices closer to the central vertex than the threshold are included in the receptive field. All neighbors in a receptive field are guaranteed to share an edge with the central vertex, allowing the application of equations (4.8) and

(4.9). For incomplete graphs, a receptive field can be defined as all vertices within k hops of the central vertex. If $k = 1$, both Sum and Product Coupling can directly be applied. If $k > 1$, then Product Coupling can't be directly applied to neighbors more than 1 hop away from the center vertex, since they share no edge with the center. Though there are ways to deal with this issue, they are not the focus of this thesis.

Lastly, to assess the benefit that incorporating information from neighboring residues has on classification performance, the *No Convolution* variant is defined as:

$$h(x_i|W^{\mathrm{C}}, b) = \sigma\bigg(W^{\mathrm{C}}x_i + b\bigg), \tag{4.10}$$

which excludes all neighbors. Note the similarity to Equation (3.1), except that in this case the output is just for a single vertex rather than for the entire layer.

4.3 Pairwise Neural Network Architecture

These graph convolution operations allow the detection of local patterns on a single graph, and produce a new representation at each vertex. Partner specific protein interaction, however, requires classifying pairs of residues in different proteins (vertices in different graphs), which is equivalent to making predictions on vertices in the product graph. Such predictions are made using a pairwise neural network architecture.

A pairwise architecture consists first of two identical convolutional modules, each responsible for generating the representation for one of the proteins in the pair. A key requirement for the pairwise architecture is symmetry, since the prediction for a pair of residues should be the same irrespective of which leg is used for which protein. To ensure symmetry in the convolution layers, weights are shared between layers in the different modules. The merge layer then combines the vertex representations from one graph with the vertex representations from the other into pairs. To maintain symmetry, this merge process should also be symmetric. For example, the elementwise sum, elementwise product, and outer-product are all commutative and therefore produce symmetric output. Another option is to combine pairs asymmetrically (e.g. concatenate

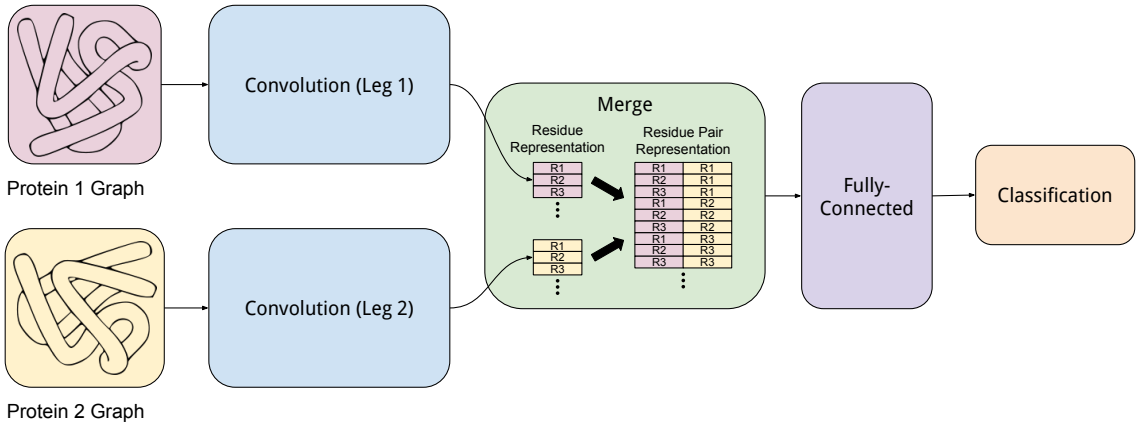


Figure 4.4: A pairwise neural network architecture that takes two protein graphs as input. Each leg contains one or more convolutional layers. The resultant graphs are then merged to create representations of residue pairs. After more fully connected layers, a final classification is performed for each pair.

the two representations together), but then average the predictions from each ordering of the pair. Finally, the combined representation for each pair of residues is passed through a number of fully connected layers. The data are represented as pairs of residues at this point. Theoretically, graph convolution could be performed at this stage as well, this time in the product graph. However, the computational and memory requirements of doing so prove prohibitive, since the number of convolutions and the number of neighbors in each convolution increase quadratically in the graph size. Hence the work in this thesis performs no convolution after merging. The final layer has a single output for each pair indicating the prediction for that pair. See Figure 4.4 for a graphical depiction.

This output is converted to a binary class prediction vector using a softmax function,

$$\text{softmax}(x) = \left[\frac{e^{-x}}{e^{x}+e^{-x}}, \frac{e^{x}}{e^{x}+e^{-x}}\right], \tag{4.11}$$

the elements of which can be interpreted as class probabilities for the negative (non-interfacial) and positive (interfacial) classes, respectively. This output is compared to a one-hot label vector indicating whether $\big([0,1]\big)$ or not$\big([1,0]\big)$ the pair constitute part of the true interface.

This chapter has presented protein graphs, graph convolution operations, and pairwise neural network architectures, all of which are components in this thesis' approach to partner specific protein interface prediction. Chapter 5 describes the experiments that were performed.

Chapter 5

Experimental Setup

All of the spatial graph convolution methods described in Chapter 4 were evaluated and compared to the existing state of the art method, PAIRpred [1]. This chapter outlines the training, validation, and testing data used for the evaluation, as well as the experimental procedure and relevant metrics.

5.1 Data

Each model was trained, validated, and tested using complexes from the Docking Benchmark Dataset (DBD) version 5.0 [85], a collection of 3D crystalline structures of transient complexes in both the bound and unbound formation. DBD is non-redundant in the sense that no two proteins in one complex are in the same Structural Classification of Proteins (SCOP) [86] families as two proteins in another complex, respectively. This ensures the data set is appropriate for testing, since no complex in the test set will be too similar to a complex in the training set. Each protein is classified into either *rigid body*, *medium difficulty*, or *difficult*, based on the degree of conformational change upon forming the interface. This is quantified using the root mean squared deviation of the α-carbons after unbound and bound structures are superimposed on one another. Training and validation were performed using the 175 complexes that were present in version 4.0 of DBD, whereas the 55 new complexes added in DBD v5.0 were reserved for testing. Training and validation sets were created using a 80%/20% partition on the DBD v4.0 complexes, such that the difficulty class proportions in each partition were roughly equivalent. There were 140 training complexes and 35 validation complexes. Table 5.1 indicates the number of positive and negative examples used in each dataset.

3D structures are contained in Protein Data Bank (PDB) files, which contain atomic coordinates for each amino acid in a complex. True interfaces were determined using the bound formations, where a pair of amino acid residues is assumed part of the interface if they are within 6 Å of each

Table 5.1: Number of complexes and examples (residue pairs) for each phase of experimentation. Negative examples were down sampled to a ratio of 10:1 with positive examples when training, but testing included all examples. During model selection, three complexes were excluded from the training and validation sets due to a software bug and missing features. These complexes were eventually included for the testing phase.

Phase	Data Set	Complexes	Positive examples	Negative examples
Model Selection	Train	138	12,632 (9.1%)	126,320 (90.9%)
	Validation	34	3,042 (0.2%)	1,868,270 (99.8%)
Testing	Train + Validation	175	16,004 (9.1%)	160,040 (90.9%)
	Test	55	4,871 (0.1%)	4,953,446 (99.9%)

other, consistent with prior work [1, 20, 87]. This definition permits a residue to have multiple partners in the interface. As mentioned, two partners in the interface may not formally interact via bonding, but the do need to be electrochemically and structurally compatible with each other. This justifies using proximity as to define the interface rather than focusing on strictly interacting proteins.

5.1.1 Vertex and Edge Features

Prediction must be made using the unbound formation of each protein, since the bound formation is not known a priori. Hence unbound PDB files were used to generate a graph representation of each protein, with vertices indicating residues and edges indicating the relationships between residues. Features were then computed for each vertex and edge using sequence and structure information from the protein. Vertex features pertain to the degree of residue conservation, accessible surface area, depth within the protein, and protrusion. Edge features capture the distance between two residues and their relative orientation as well. Raw distances are passed through a Gaussian function centered at zero with a standard deviation chosen during model selection. More detail on each feature can be found in Appendix A.

5.1.2 Handling Missing Data

In some cases, features could not be calculated for a residue, resulting in missing values. For 13 residues, the atomic coordinates of the central α-carbon were not present in the PDB files, so

some features could not be calculated (see Appendix A for more detail). Most features rely on third party software to calculate the features from the PDB data. For the proteins with DBD codes 2B42 and 3R9A, features related to the protrusion index could not be calculated for the receptor due to an unknown third party software fault. These complexes were omitted during model selection but re-introduced with imputed values before testing. Two other complexes, 1NW9 and 1PPE, were also excluded during model selection due to a software bug. This bug was eventually fixed, so these were also included during testing. During model selection, missing values were imputed using the protein mean, whereas for testing, the global median (within the training, validation, or test set) was used. In total there were 138 training and 34 validation complexes during model selection, and there were 175 training and 55 test complexes during testing.

5.2 Experimental Procedure

Experimentation was conducted in two phases: model selection, and final testing. The model selection phase concerned the selection of model *hyper-parameters*. For this phase, pairwise graph convolutional networks were trained using the training set and evaluated on the validation set. Once model selection was complete, the final set of hyper-parameters was used to train and test the most promising hyperparameter configurations and compare them. For this phase, the model was trained on the combined training and validation sets and tested on the test set. More detail about each phase follows.

5.2.1 Model Selection

Hyper-parameters comprise all model specifications besides the parameters themselves, such as the number and types of layers, number of units/filters in each layer, nonlinear activation function, loss function, and training procedure. Hyper-parameters are typically chosen either by some automated process, or by manual exploration.

Methods for automated hyper-parameter selection include randomized search, grid search, or sequential, model based optimization (SMBO) such as a Gaussian Process Optimization or Tree

Structured Parzen Estimator Optimization [88]. The advantage of these approaches is their ability to automatically search a hyper-parameter space without human intervention. SMBO approaches are able to estimate the performance "response surface" in hyper-parameter space in order to efficiently find the optimum value. The disadvantage of these approaches is they require a large number of iterations in order to converge to optimal parameters if the number of hyperparameters is large and the search space for each is large. Random or grid search are no better. For this thesis, some experiments were performed with SMBO algorithms, but none produced hyper-parameters which outperformed results found by human exploration. Therefore automated approaches were abandoned in favor of a manual search.

Several hyper-parameters were explored manually by training and validating networks under different settings of the hyper-parameters. This exploration occurred in a one-hyper-parameter-at-a-time fashion, alleviating the search space problem encountered with the automated algorithms, but also limiting the ability to explore interactions between different hyper-parameters. In some cases, the choice of hyperparameter affected training time and model convergence rate, but had little impact on overall model performance (e.g. the nonlinear activation function, training algorithm, the number of convolutional filters). In other cases, few values had to be searched before a local optimum was found (e.g. the number of dense layers after merging, minibatch size, optimization strategy, residue distance Gaussian function standard deviation). Because this manual optimization was iterative, it is impossible to summarize how each alernative choice of hyperparameter would change the results from the set of hyperparameters ultimately chosen. However, some trends were observed consistently throughout testing and are worth describing in more detail. This is done below.

As the name implies, deep networks typically perform best with more than one layer. This is indeed the case for pairwise graph convolutional networks, although the benefit began to diminish beyond 4 layers. To illustrate this trend, multiple depths were used during the validation and testing phases. Typically convolutional neural networks begin with a small receptive field size which grows in successive convolutional layers. For this problem, networks performed better if

all layers had the same relatively large receptive field, around 21. Significantly larger receptive fields would begin to encompass entire proteins, since the smallest protein in the data set has 29 residues. During testing, receptive field sizes of 11 and 21 were used to enable comparison. For image convolution, it is not uncommon for the number of filters in the first convolutional layer to be roughly an order of magnitude more than the number of input channels, and to increase in subsequent layers. In contrast to images which may have only 1 or 3 channels, protein graphs have 70 features for each vertex, so an analogous increase in the number of filters would quickly exceed the memory resources of a GPU. Nevertheless, a larger number of filters proved beneficial, so all graph convolutions started with 256 filters and increased to a size of 512 in the last convolutional layer (if the network had more than one). Smaller numbers of filters were explored but did not perform as well. Larger numbers of filters began to exceed the memory capacity of the GPU's used for training.

Most model selection was performed on the proposed graph convolutions and No Convolution. Because most of the other convolution methods are not appreciably different, an abridged model selection process was performed for them, simply to check whether the same hyperparameters could reasonably be used. This was indeed the case. The exception was DCNN, where the Gaussian standard deviation for residue distances was observed to have a significant impact on model performance. For this method, rather than using a value of $\sigma = 18$, much smaller values of 2Å and 4Å were chosen for the testing phase. Given that the diffusion is occurring on a complete graph, the smaller values allow only local information to propagate during each hop. The other graph convolutional methods do not have this same problem, since they are restricted to a fixed size receptive field. Diffusion convolutions do not downsample the input, so it is possible to stack multiple convolutions on top of one another. However, additional diffusion convolutional layers did not improve performance, so only results from a single layer were tested.

5.2.2 Model Testing Setup

Final model testing was performed for all variants of convolution and No Convolution. For each variant, receptive field sizes of 11 and 21 were tested (the central vertex plus 10 or 20 nearest neighbors, respectively), with one, two, three, and four graph convolutional layers. In total, 16 combinations were tested, except DCNN as already noted. All variants followed the same training scheme: The number of filters for networks with one convolutional layer was 256, Likewise for two, three, and four convolutional layers the number of filters was (256, 512), (256, 256, 512), and (256, 256, 512, 512) respectively. For DTNN, the number of filters matched the number of input features, 70. This was required, because Equation (4.7) does not apply a weight matrix to x_i. After merging examples via concatenation, dense layers of 256 and one units were applied, with the latter producing the model output. Except for the last layer, rectified linear units (ReLU) as the nonlinear activation function, except for DTNN for the following reason. Note that activation functions are being applied to each neighbor individually in Equation (4.7), which is different from the other methods. Since ReLU is non-negative, it would force the layer output to be greater or equal to the input. Therefore, tanh is used instead to allow for both positive and negative adjustments to x_i.

Loss was computed using weighted cross entropy, a measure of the dissimilarity between two probability distributions. Since this is a binary classification problem, the cross entropy for a residue pair ξ_i is defined as:

$$\mathbb{H}(\xi_i, y_i|\Theta) = -r\ y_{i0}\ log\big(f_0(\xi_i|\Theta)\big) - y_{i1}\ log\big(f_1(\xi_i|\Theta)\big), \tag{5.1}$$

where $f = (f_0, f_1)$ is the softmax network output, interpreted as class probabilities, $y_i = (y_{i0}, y_{i1})$ is the one-hot class label, and r is the ratio of positive to negative examples, which ensures that the set of positive examples and the set of negative examples contribute equally to the loss function. The overall loss function is then:

$$\mathbb{L}(\Theta|\{\xi_i, y_i\}) = \sum_i \mathbb{H}(\xi_i, y_i|\Theta), \tag{5.2}$$

which sums the cross entropy from each residue pair. This *class balanced* version of cross entropy is similar to the *median frequency* approach taken by Eigen & Fergus [89], except that it differs by a multiplicative constant.

All bias weights were initialized to zero. All non-bias weights were initialized by drawing from a uniform distribution between $-\tau_0$ and τ_0, where $\tau_0 = \frac{1}{\sqrt{n_{in}}}$ and n_{in} is the number of inputs to the layer. This is similar to the initialization proposed by He, Zhang, Ren & Sun [90], except they differ by the small multiplicative constant, $\sqrt{6}$. Stochastic gradient descent was performed with a learning rate of 0.1, mini-batch size of 128 pairs, for 80 epochs. Dropout with probability $p = 0.5$ was performed during training.

The networks were implemented in TensorFlow [91] v1.0.1 and trained using a single NVIDIA GTX 980 or GTX TITAN X GPU. Training time varied from roughly 7 to 220 minutes depending on convolution method, implementation, network size, GPU card, and computer resource availability. Most methods took less than 60 minutes, with the exceptions being 4 layer PATCHY-SAN networks and R-GCNs that include basis functions.

The graph convolution variants were compared to PAIRpred, an existing state of the art partner specific interface prediction algorithm based on an SVM with custom symmetric pairwise kernels, proposed by Minhas et al. [1]. PAIRpred was run with the same features and data that were used in this work for the purpose of comparison. Grid search with five-fold cross validation was used to select the optimal kernel and soft margin parameters.

5.3 Metrics

While measures like accuracy, precision, recall, specificity, and F-score are common for many classification problems, they are dependent on the choice of a particular classification threshold. A metric more appropriate for interface prediction would be the area under the receiver operating characteristic curve (AUC), because it does not rely on a particular choice of threshold. Instead, it summarizes the true positive rate across all false positive rates, indicating threshold independent performance of the classifier.

In addition, in the case of an extreme class imbalance, the AUC is misleading, since the movement of a single positive example can heavily influence the true positive rate compared to the effect that a single negative example has on the false positive rate. Even with a high AUC, a classifier may rank a number of negative examples above the first positive example, when ranking by predicted likelihood of being part of the interface. In this case, it is also useful to measure how far down the ranked scores one has to go before encountering a true positive. A metric which captures this directly is the rank of the first positive prediction (RFPP), where the rank is computed after ordering pairs by classifier score. Ideally, the top ranking prediction is truly part of the interface, in which case RFPP is 1.

Interface prediction occurs at the complex level, so AUC and RFPP should be calculated with respect to each complex. The performance of a classifier on a set of complexes can then be summarized using median AUC and RFPP across the complexes. These metrics closely mimic those used in the PAIRpred paper [1], except that median AUC is taken instead of average.

Chapter 6

Results

This chapter first explores the behavior of Sum and Product Coupling. Then it compares to PAIRpred and the other variants of graph convolution. Some convolution filters are then visualized, along with network output for one trained network. Finally, it summarizes the effect of protein size on network training time.

6.1 Sum and Product Coupling Performance

Table 6.1 shows the results of experiments involving Sum Coupling and Product Coupling, as well as No Convolution. Each number displayed represents a single experiment (not the average of repeated experiments), so note that results may change if the experiments are run again with different initial weights. To evaluate the effect of weight initialization on performance, 10 replications were run for Sum Coupling, two layers, and a receptive field of size 21. The standard deviation of the runs was 0.006, small compared to some of the differences observed in the results. Since these replications were only performed for one method and set of conditions, it is possible that other methods are more sensitive to initial conditions. Running repeated replications of each condition could explore this. Because of the large number of methods/experiments run and the training time for each network, this was not performed as part of this thesis.

Comparing Sum and Product Coupling to No Convolution reveals that convolution is beneficial, since the convolution layers consistently perform better, regardless of the number of layers. In other words, incorporating information from neighboring residues helps indicate whether a residue is part of an interface, which is consistent with the biological properties of interfaces. It's also clear that a receptive field of size 21 is generally better than 11. Interestingly, this value is near the median number of interface residues for a single protein, which is 27. When using convolution, performance improves with network depth up to a point, then either decreases or remains relatively constant. In Sum Coupling, this maximum occurs in the second layer, whereas in Product

Table 6.1: Median area under the receiver operating characteristic curve (AUC) across all complexes in the test set for two variants of graph convolution, Sum Coupling and Product Coupling, as well as No Convolution. Results are shown for two different sizes of receptive field, 11 and 21, for different numbers of convolutional layers before the pairwise merge operation. Bold faced values indicate best performance for each method.

Method	Receptive Field Size	Layers Before Merge 1	2	3	4
No Convolution	N/A	**0.815**	0.812	0.800	0.811
Sum Coupling	11	0.868	0.889	0.882	0.884
	21	0.875	**0.903**	0.880	0.890
Product Coupling	11	0.856	0.869	0.885	0.868
	21	0.863	0.876	0.896	**0.899**

Table 6.2: Median rank of the first positive prediction (RFPP) across all complexes in the test set for two variants of graph convolution, Sum Coupling and Product Coupling, as well as No Convolution. Results are shown for two different sizes of receptive field, 11 and 21, for different numbers of convolutional layers before the pairwise merge operation. Bold faced values indicate best performance for each method (lower is better).

Method	Receptive Field Size	Layers Before Merge 1	2	3	4
No Convolution	N/A	**48**	55	53	66
Sum Coupling	11	32	28	70	86
	21	**26**	37	56	63
Product Coupling	11	30	46	26	51
	21	26	**25**	36	37

Coupling, this maximum occurs in the fourth layer. Though not shown in the table, Product Coupling indeed fails to improve when using 5 or 6 layers. In contrast, networks without convolution are best with only one pre-merge layer. This suggests that depth alone does not improve performance, but when convolution is performed, a useful hierarchical representation is learned. Other applications of deep learning have seen this same trend of increasing and decreasing performance. The commonly given explanations for such behavior include insufficient training data, a vanishing gradient, and a difficulty with optimization [90].

Table 6.2 parallels Table 6.1 but shows RFPP instead of AUC. As before, Product Coupling does better than No Convolution, regardless of the number of layers. Sum Coupling, is only better

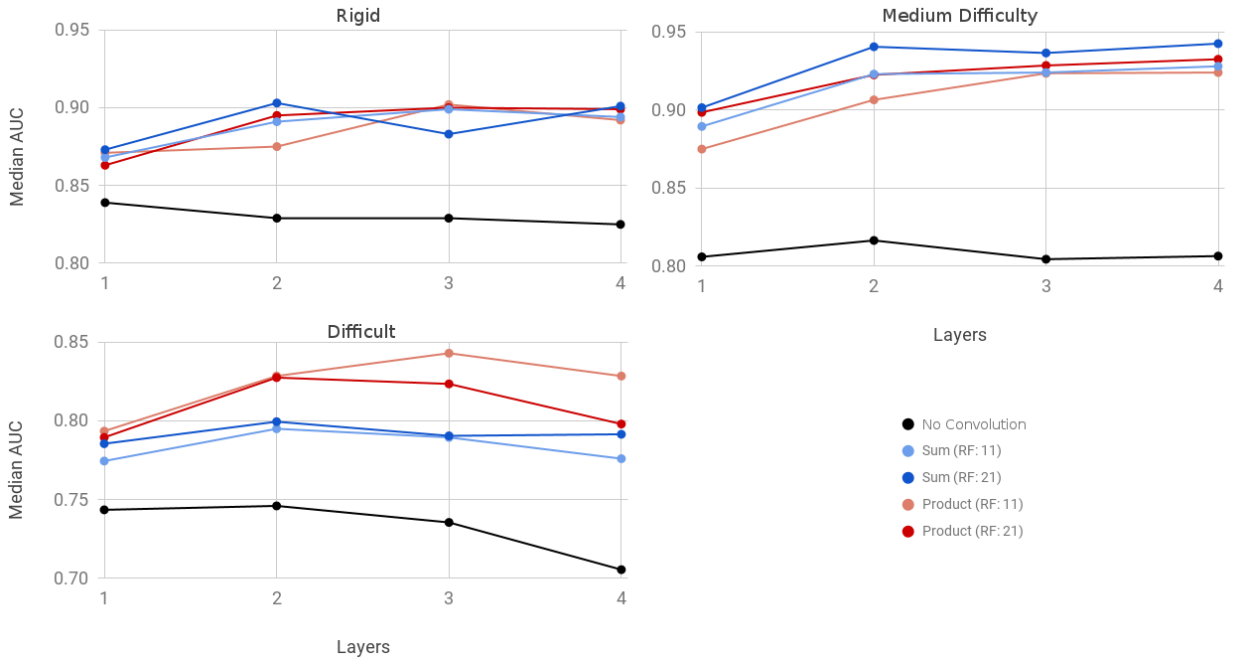


Figure 6.1: Median area under the receiver operating characteristic curve (AUC) across all complexes in the test set, separated by complex class. Sum and Product Coupling are shown for two receptive field sizes each (11 and 21), as well as No Convolution, for 1-4 pre-merge layers. Product Coupling performs better for difficult complexes, but worse overall because ther are far more rigid and medium difficulty complexes.

than No Convolution in some cases. In this case, however, the best performance for Sum and Product Coupling is seen for fewer layers than AUC. This difference is not surprising, considering the cross-entropy loss function leads to optimization of performance on *all* pairs, not just the top scoring ones. In other words, RFPP is not being explicitly optimized in this problem, whereas AUC is more closely related to the quantity being optimized.

To understand the behavior of each method in more detail, we can separate performance by the difficulty class of the test proteins. Figure 6.1 shows performance for rigid, medium difficulty, and difficult classes, with 33, 16, and 6 complexes respectively. Here it appears that Sum and Product Coupling are closely matched for rigid and medium difficulty. A slight difference is seen for the difficult complexes, where Product Coupling appears to be performing better for two and three layers, but with so few complexes it's unclear if this is a true trend.

For another picture of performance across difficulty classes, we can examine a histogram of AUCs, as shown in Figure 6.2. These AUCs are heavily skewed, justifying the choice of median

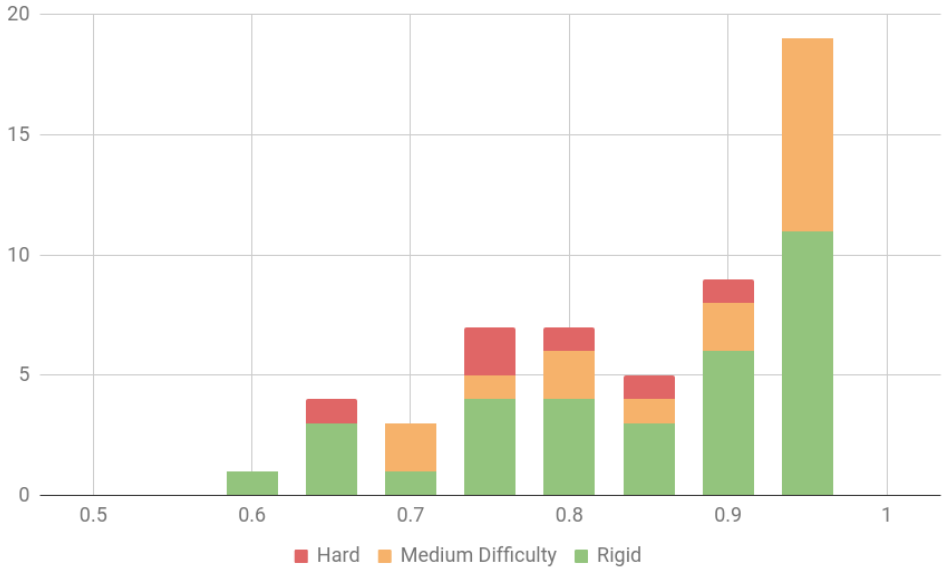


Figure 6.2: Histogram of area under the receiver operating characteristic curve (AUC) for complexes in the test set, colored by difficulty class. Scores are from Sum Coupling with two layers and receptive field size 21, which had the highest median AUC of all methods.

as a summary measure. Surprisingly, there is no clear divide between rigid, medium difficulty, and difficult classes. In fact, the worst AUC is a rigid complex, and one difficult complex achieves AUC above 0.9. It appears that the distinguishing characteristic between classes is the number of complexes that achieve above 0.95 AUC. These "trivial" complexes are most frequent in the rigid class, less so in the medium difficulty class, and absent for the difficult class. This suggests that the networks are effectively coping with some complexes that under conformational change.

RFPP can also be separated by difficulty class. From figure 6.3 we can observe some heterogeneus behavior across methods, but there are some trends worth noting. For each class, there appears to be a favored number of layers where a dip (improvement) in RFPP is observed. This optimal depth appears to increase with difficulty class, suggesting that harder complexes require more layers to achieve best performance. Therefore an ensemble approach with varying depths may perform well on complexes of any difficulty.

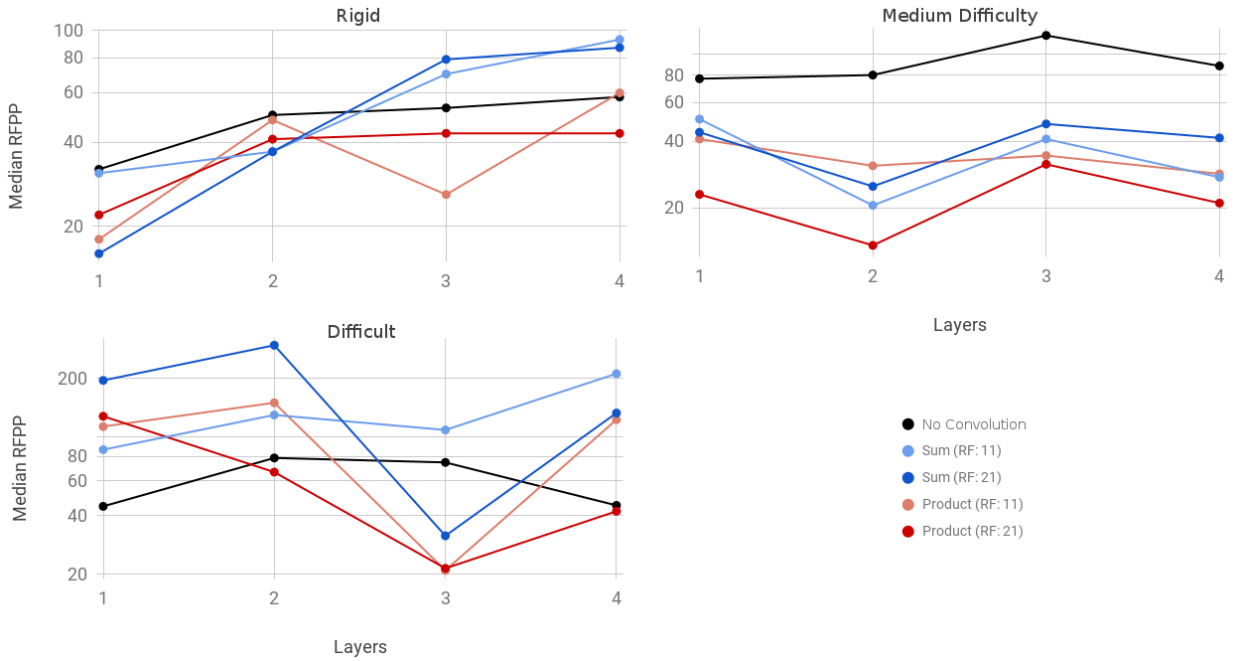


Figure 6.3: Median rank of the first positive prediction (RFPP) across all complexes in the test set, separated by difficulty class. Vertical axes are log scaled. Sum and Product Coupling are shown for two receptive field sizes each (11 and 21), as well as No Convolution, for 1-4 pre-merge layers. Lower RFPP is better. Best performance on rigid complexes is achieved with just one layer for all networks. As difficulty increases, so does the number of layers needed to achieve best results.

Table 6.3: Comparison of proposed convolutions with existing classification methods. Bold values indicate best performance for each method. For reference, retraining Sum Coupling ten times with a different random seed yields a standard deviation of 0.006, and other methods are believed to behave similarly. *PAIRpred is an SVM-based approach, so the result is not really associated with a layer number.

Method	Variant	Layers			
		1	2	3	4
No Convolution	N/A	**0.815**	0.812	0.800	0.811
Sum Coupling	RF $= 11$	0.868	0.889	0.882	0.884
	RF $= 21$	0.875	**0.903**	0.880	0.890
Product Coupling	RF $= 11$	0.856	0.869	0.885	0.868
	RF $= 21$	0.863	0.876	0.896	**0.899**
PAIRpred [1]	N/A	(**0.863**)*	-	-	-
PATCHY-SAN [81]	RF $= 11$	0.862	0.867	0.883	0.891
	RF $= 21$	0.850	0.875	**0.897**	0.886
Fingerprint [82]	RF $= 11$	0.857	0.850	0.863	0.833
	RF $= 21$	0.861	0.867	0.881	**0.891**
R-GCN [83]	No basis fns, RF $= 11$	0.862	0.871	0.886	0.893
	No basis fns, RF $= 21$	0.876	**0.901**	0.892	0.897
	2 basis fns, RF $= 11$	0.851	0.872	0.779	-
	2 basis fns, RF $= 21$	0.873	0.804	0.539	-
	5 basis fns, RF $= 11$	0.870	0.747	0.748	-
	5 basis fns, RF $= 21$	0.867	0.900	0.709	-
DTNN [84]	RF $= 11$	0.853	0.872	0.878	0.861
	RF $= 21$	0.862	0.880	0.873	**0.885**
DCNN [80]	2 hops, $\sigma = 2$Å	0.782	-	-	-
	2 hops, $\sigma = 4$Å	0.801	-	-	-
	5 hops, $\sigma = 2$Å	**0.838**	-	-	-
	5 hops, $\sigma = 4$Å	0.819	-	-	-

6.2 Other Methods

Table 6.3 gives the median AUC of the various existing methods discussed. PAIRpred, the state of the art pairwise interface predictor, establishes the baseline. Interestingly, most of the graph convolution based methods exceed this, under the right set of hyperparameters. DCNN is the exception, showing worse performance than PAIRpred in all cases examined. This is perhaps unsurprising, since it differs considerably from the other convolution methods, which are very similar to one another.

The order imposing PATCHY-SAN performs quite well, suggesting that the chosen ordering of neighbors by proximity to the central vertex was a good one. Its best AUC of 0.897 is within one standard deviation (0.006) of the best overall performance, which was 0.903 from Sum Coupling. Therefore it is difficult to say whether order-free or ordered methods perform better for this problem. Only one ordered method was examined, leaving the possibility that others may do better. Because of the unique weights used for each neighbor, ordered methods have significantly more parameters than order-free methods, making them more susceptible to overfitting, though that is not observed in these results. This method also uses information from all edges in the receptive field, whereas the other methods are only concerned with edges connecting neighbors to the central vertex.

R-GCN did best without basis matricess. The original intent for basis matricess was to allow for weight sharing across several relation types. This is less relevant for these protein graphs since only a single neighborhood is being considered per central vertex, so in a sense there is only one relation type. When using basis matricess, larger networks (starting at three layers) drop significantly in performance compared to other methods. At four layers, networks consistently classified all residue pairs with the same score, suggesting all filters had become zero valued (note that AUC cannot be calculated with all scores are identical). Note that AUC cannot be calculated when all scores are identical. This is possible when using ReLU nonlinearities, since the output and gradient vanishes for signals less than zero. It's possible that a different initialization scheme or a different nonlinearity would remedy this problem. Regardless, the use of basis matricess in

this context is somewhat unjustified because these graphs lack the numerous relation types that prompted use of basis matricess in the first place [83]. Nevertheless, the two layer, five basis matrices version performed similarly to the best results observed. But testing with 8 basis matrices did not show improvement. Without basis matricess, R-GCN is equivalent to Sum Coupling if edge information is excluded. The fact that R-GCN without basis matricess performs similarly to Sum Coupling suggests that the added edge information in Sum Coupling is not very useful.

Fingerprint is arguably the simplest form of graph convolution since it uses the same weight matrix for both central and neighbor vertices. This simpler mode achieves its best performance at higher depths than other, more complex convolutions. Increasing the number of layers to five or six exhibits no substantive improvement.

Whereas PATCHY-SAN, Fingerprint, and R-GCN are most similar to Sum Coupling, DTNN allows for a multiplicative coupling between neighbors and edges, similar to Product Coupling. Like Product Coupling, best performance is achieved at a greater depth than the other methods, but does not continue to improve for five or six layers. This method is notably worse than the three just mentioned. One potentially limiting aspect of this method is that the number of channels is necessarily unchanged.

For DCNN, five hops performs better than two hops, presumably for the same reason that a larger receptive field is better in the other convolution methods: information is able to propagate further across the graph. The Gaussian standard deviation determines the range of diffusion, where smaller values limit diffusion to a localized neighborhood for each hop, and larger values allow diffusion across longer distances. In the two hop DCNN, the larger standard deviation allows more diffusion to occur across the graph, compensating for the limited number of hops. This explains the overall better performance for $\sigma = 4$. In contrast, the five hop DCNN performed better for $\sigma = 2$, suggesting that the larger number of hops allows sufficient information propagation, eliminating the need for diffusion across greater distances for each hop.

Note: since multiple versions of each method were run against the test set, it is tempting to select the best performance in each method for comparison. This action essentially uses the test

set to perform further model selection, and doing so limits the generalizability of the results. It is more appropriate to examine the trends within a method, and to compare methods under identical experimental conditions (e.g. number of layers), if possible. It is evident, however, that Sum and Product Coupling outperform PAIRpred under most hyperparameter settings examined. This may indicate that the proposed convolutional methods truly generalize better than the existing SVM based approach.

6.3 Filter Visualization

It is often difficult to understand the behavior of a model by simply looking at the overall performance. For convolutional neural networks on images, intuition can be gained through a variety of methods, including visualizing filters directly, tracking a filter's activation as different regions of the image are occluded, and identifying which images maximally activate each filter. To understand the filters being learned in this pairwise neural network architecture, network scores and filter activations were mapped to a protein complex heatmap. Figure 6.4 presents color maps on the complex 3HI6 [5] which depict the true interface, maximum predicted scores for each residue, and the activation of two filters from last convolutional layer, for the best performing method (Sum Coupling with two layers and receptive field size of 21). Scores are partner specific, so they are shown for a single residue after taking the maximum over all potential partners. This partner independent visualization matches well with the true interface. In contrast, convolutional filters occur at the residue level and can be visualized directly. The two filters shown illustrate learned features which are useful for interface prediction. Specifically, one activates only for buried residues (indicating an *unlikeliness* to participate in an interface), and the other activates only for residues near the true interface.

6.4 Training Time

Figure 6.5 shows a log-log plot of training time as a function of the number of residue pairs in a given complex. The relationship is roughly linear in this space, indicating a power-law relationship.

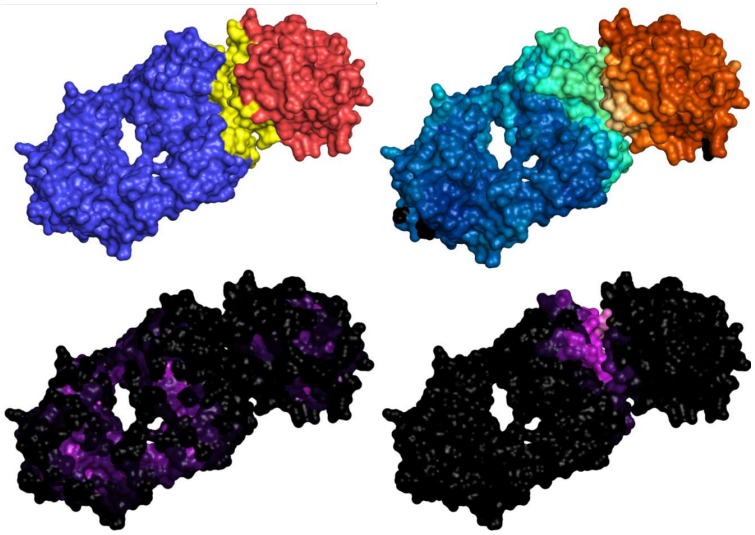


Figure 6.4: PyMOL [4] visualizations of the best performing test complex (3HI6 [5]). Upper left: Ligand (red) and receptor (blue), along with the true interface (yellow). Upper right: Visualization of predicted scores, where brighter colors (cyan and orange) are higher scores and darker colors (blue and red) are lower scores. Since scores are for pairs of residues, we take the max score over all partners in the opposing protein. Bottom row: Activations of two filters in the second convolutional layer, where brighter colors indicate greater activation and black indicates activation of zero. Lower left: Filter which provides higher activations for buried residues, a useful screening criterion for interface detection. Lower right: Filter which gives high activations for residues near the interface of this complex. Image credit: Basir Shariat and Alex Fout.

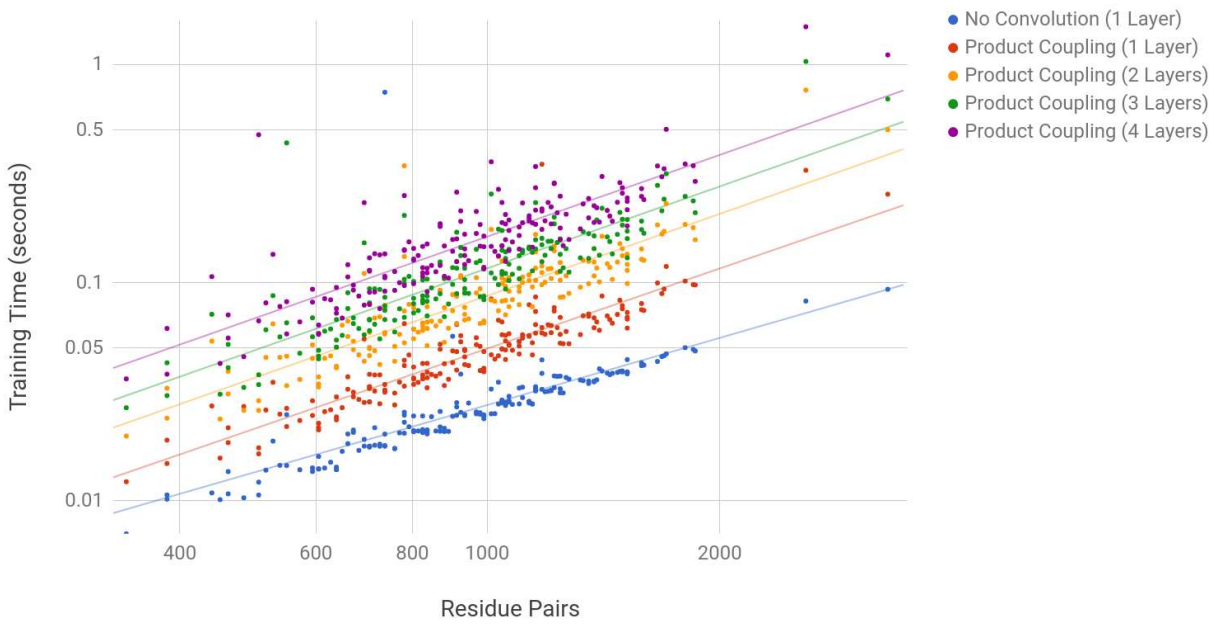


Figure 6.5: Training time as a function of number of residue pairs, for a single No Convolution layer, as well as four depths of Product Coupling, receptive field size 21. Linearity in this log-log plot indicates a power law relationship. In this case, the relationship is near linear, with powers of 1.02, 1.22, 1.25, 1.25, and 1.24 respectively for No Convolution 1 layer, and Product Coupling for 1, 2, 3, and 4 layers. Other convolution methods have similar power law relationships.

Fitting lines to the data shows that the power is slightly greater than 1.0 in all cases, showing that training time increases near linearly in the number of pairs as expected. This is due to the pairwise nature of the problem. For problems on a single graph, the training time would likely scale near linearly in the number of vertices. Deeper networks take longer to train, and summing over neighbors (as in Product Coupling), takes longer than just using the central vertex (as in No Convolution). These trends are as expected.

Chapter 7

Possible Future Work

The work in this thesis is pertinent to emerging methods in machine learning as well as current problems in structural bioinformatics. As such, extensions of this work can be made in several distict veins, some focused on model formulation and training procedure, others on possible applications outside of interface prediction. Several of these extensions are documented below, categorized as either extensions of method or extensions of application.

7.1 Extensions of Method

7.1.1 Double Coupling and Ensemble Approaches

As discussed in Chapter 5, Sum Coupling outperforms Product Coupling on rigid and medium difficulty complexes, whereas Product Coupling is better for difficult complexes. Therefore it would be reasonable to incorporate both sum and Product Coupling into a single convolution operation:

$$h_i(x|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{E}}, b) = \sigma\left(W^{\mathrm{C}}x_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}(W^{\mathrm{N}}x_j)\odot(W^{\mathrm{E}}A_{ij}) + W^{\mathrm{N}}x_j + W^{\mathrm{E}}A_{ij} + b\right), \quad (7.1)$$

where here the same weight matrices are used in both the Sum Coupling and Product Coupling components to help prevent overfitting. Alternately, separate weight matrices could be used to increase the model's expressive power.

Recall that concerning the RFPP metric, the optimal network depth increased with complex difficulty, where more layers were needed for complexes of greater difficulty. This suggests that an ensemble model with multiple networks of varying depth could perform better than each individual network. Alternative ensemble approaches like *boosting* or *bagging* may also prove beneficial. Boosting trains a sequence of "weak" models, with each added model being trained by giving

more importance (loss weight) to examples misclassified by the existing set of models. For interface prediction, this weighting can be performed on a per-complex basis, using RFPP or AUC as an indication of performance on each complex, or on a per-residue-pair basis, using the cross entropy loss of that specific training example. The final prediction is a weighted average of the weak models' predictions, with the weights determined by the validation error of each respective model (with higher weight given to models with lower error). Bagging creates multiple datasets by sampling with replacement from the original data set, and trains a different model on each sampled dataset. This sampling can be performed at the complex level or the residue pair level. Again, the final prediction is a weighted average of the weak models' predictions, weighted according to validation error.

7.1.2 RFPP Optimization

Predicting an entire interface is a more difficult task than predicting only a few interacting residues in that problem. It could be that training a classifier in the former may hinder its performance in the latter. An alternative approach would be to just focus on generating a few predictions of high quality (i.e. confidence), which means optimizing performance on just the top positive predictions. The existing model uses a loss function which incorporates every training example, not just the top positive prediction, hence the model is optimized for all examples. The new approach implies treating RFPP as the primary metric and optimizing it directly. An alternate loss function which includes the loss from only the highest scoring positive example could be used. Hence when the loss is minimized, the performance of the highest performing positive example will be maximized. Here a sum over all examples has been replaced by a max function, which is not differentiable, but differentiable variants of the loss function could be constructed to enable gradient descent based learning. This method is currently being investigated by another student in Professor Ben-Hur's research group.

7.1.3 Additional Data Sources

The success of deep learning methods has been attributed to large volumes of data and deep architectures [60]. For interface prediction, despite the large volume of recorded protein structures, precious few complexes have been labeled in bound and unbound forms for use in model training and testing. This dependence on small curated subsets of the available proteins has potentially limited the full leveraging of deep learning methods, however some opportunities exist for enlarging the training and testing data.

The Docking Benchmark Dataset was conceived as a method to evaluate docking methods, and correspondingly was carefully constructed to be non-redundant with respect to SCOP families, in order to give a fair evaluation. However for training purposes, it may be useful to include redundant proteins simply to provide the model with more training data. The Critical Assessment of PRediction of Interactions (CAPRI) is an annual competition aimed at evaluating protein-protein docking methods [31], and could also be used to help train or evaluate partner-specific protein interface prediction methods.

7.1.4 Unsupervised Pretraining

Another approach to the problem of limited labeled complexes is to utilize the vast quantity (>125,000) of recorded protein structures via unsupervised pre-training. Hinton and Salakhutdinov (2006) [92] proposed a greedy layer-wise training algorithm for a particular form of neural network called an *autoencoder*. Generally speaking, an autoencoder is simply a network trained to reproduce the input at the output under certain constraints [93]. These constraints, such as undercompleteness, regularization, and sparsity, prevent the network from simply learning an identity mapping of the input, and instead promote the learning of a compressed representation of the input (i.e. an "encoding"). The portion of the network which compresses the input is called the encoder, while the portion which reconstructs the input from the encoding is called the decoder. The encoding half of the autoencoder can be used independently of the decoding half to create representations of the input that are often useful in supervised tasks like classification [92, 94].

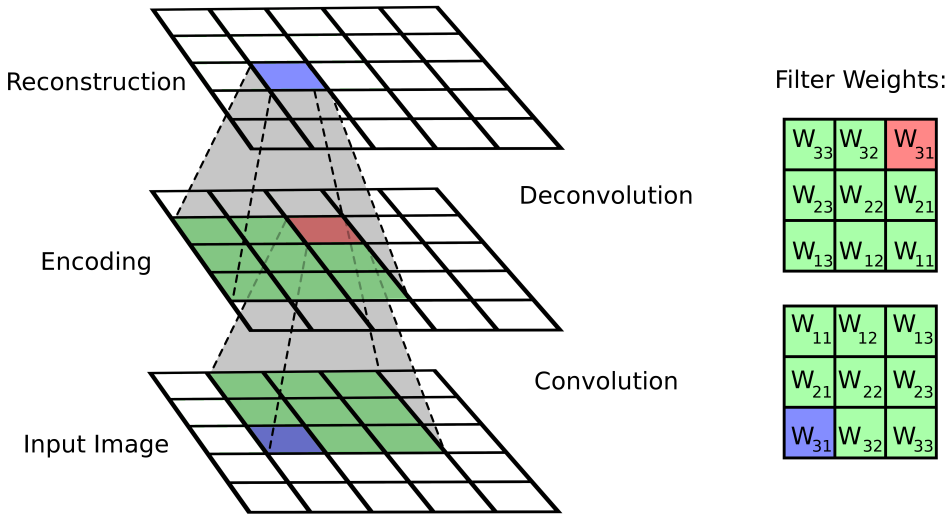


Figure 7.1: Application of convolution, then deconvolution on a single channel input image. The relationship between the purple pixel in the input/reconstruction and the red pixel in the encoding is captured by the appropriate weight in the filters (lower left for convolution and upper right for deconvolution). By reflecting the convolution filter horizontally and vertically for use in deconvolution, the same weight (W_{31}) is used for this relationship during both encoding and decoding.

Masci et al. (2011) [65] proposed convolutional autoencoders (CAEs) which perform convolution to encode an image and *deconvolution* to decode the image. There is also a corresponding *unpooling* operation which upsamples images which were pooled during the encoding step. Deconvolution is simply a convolution operation performed on the result of an encoding convolution, with weights being tied between convolution and deconvolution layers. For k filters of size $m \times m$ and c channels, the corresponding deconvolution would consist of c filters of size $m \times m$ and k channels, where the weights have been reflected in both spatial dimensions (e.g. the weights for the lower left pixel of the receptive field are now applied to the upper right pixel). The spatial reflection allows a particular weight to carry a specific meaning, namely it characterizes the relationship between a particular pixel in an image and a pixel in its encoding. To illustrate this, Figure 7.1 shows a deconvolution for a single filter and single channel. CAEs are trained in the same greedy, layerwise fashion as conventional autoencoders.

The concept of deconvolution can be applied to graph convolutions as well. Note that when reflecting image convolution filters, the center weight remains in the same position. In the same way, graph deconvolutions retain the same weights for the central vertex. Furthermore, since all

neighbors use the same weights, there is no spatial reflection necessary. Deconvolution simply consists of transposing the center and neighbor weight matrices so that channels become filters and filters become channels. For Sum Coupling, deconvolution becomes:

$$\hat{x_i}(h_i|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{E}*}, b^*) = \sigma\bigg((W^{\mathrm{C}})'h_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\left[(W^{\mathrm{N}})'h_j + W^{\mathrm{E}*}A_{ij}\right] + b^*\bigg), \tag{7.2}$$

where $\hat{x}$ denotes the reconstruction of x after deconvolving, h_i is the convolved representation at vertex i, $(W^{\mathrm{C}})'$ is the transpose of W^{C}, likewise for $(W^{\mathrm{N}})'$ and W^{N}, and both $W^{\mathrm{E}*}$ and b^* are weight matrices with unshared weights. Note that graph convolutions generate representations on vertices of the graph, not the edges, so the non-encoded edge information (A_{ij}) must be used when deconvolving, and the weight matrix $W^{\mathrm{E}*}$ cannot be tied to the encoding matrix W^{E}.

Alternately, edge representations could be created in a separate convolution operation, where an edge's receptive field is simply the incident vertices:

$$h_{ij}(A_{ij}|W^{\mathrm{EE}}, W^{\mathrm{EV}}, b_e) = \sigma\bigg(W^{\mathrm{EE}}A_{ij} + \frac{1}{2}\big(W^{\mathrm{EV}}x_i + W^{\mathrm{EV}}x_j\big) + b_{\mathrm{E}}\bigg), \tag{7.3}$$

where h_{ij} is the representation of edge (i, j) , W^{EE} is the weight matrix associated with the edge, W^{EV} is the weight matrix associated with the incident vertices, and b_{E} is the bias. In this case, vertex deconvolution can use the encoded edge representations and associated weights. Equation (7.2) then becomes:

$$\hat{x_i}(h_i|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{EV}}, b) = \sigma\bigg((W^{\mathrm{C}})'h_i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\left[(W^{\mathrm{N}})'h_j + (W^{\mathrm{EV}})'h_{ij}\right] + b^*\bigg), \tag{7.4}$$

where $(W^{\mathrm{EV}})'$ is the transpose of W^{EF}. Edges can also be deconvolved using the edge and vertex representations:

$$\hat{A_{ij}}(h_{ij}|W^{\text{EE}}, W^{\text{E}}, b) = \sigma\bigg((W^{\text{EE}})'h_{ij} + \frac{1}{2}\big[(W^{\text{E}})'h_i + (W^{\text{E}})'h_j\big] + b^*_{\text{E}}\bigg). \tag{7.5}$$

This added weight sharing and symmetry between convolution and deconvolution operations may allow training of deeper networks which recognize more sophisticated structures.

7.1.5 Simplicial Complex Convolution

Graphs are only one way to model structural relationships between entities. A simplicial complex is a generalization of a graph in which higher order simplices (triangles, tetrahedra, etc), in addition to edges, describe relationships between groups of vertices. For example, just as an edge indicates a relationship between two vertices, a triangle (or 2-simplex) indicates the relationship between three vertices. For example, both the Čech and Vietoris-Rips definitions can be used to construct a simplicial complex from a set of points in an underlying metric space:

Definition 7.1.1. *Given a point set X in some metric space and a number $\epsilon > 0$, the Čech complex $C_\epsilon^{\check{\text{C}}}$ is the simplicial complex whose simplices are formed as follows. For each subset $S \subset X$ of points, form an $(\epsilon/2)$-ball around each point in S, and include S as a simplex (of dimension $|S|$) if there is a common point contained in all of the balls in S.*

Definition 7.1.2. *Given a point set X in some metric space and a number $\epsilon > 0$, the Vietoris-Rips complex C_ϵ^{VR} is the simplicial complex whose simplices are formed as follows. For each subset $S \subset X$ of points, form an $(\epsilon/2)$-ball around each point in S, and include S as a simplex (of dimension $|S|$) if for each pair of balls in S there is a common point contained in both balls.*

In both definitions, simplices are included in the simplicial complex based on some notion of proximity. This means simplicial complexes are naturally equipped to identify clusters of vertices within a neighborhood, something which can be useful for interface prediction, where local regions of residues may constitute portions of an interface. Though the number of possible convolution operations on simplicial complexes are many, one example is provided here as an illustration. In this formulation, rather than sum the signal from all neighbors in the receptive field, each simplex $S_k, k \in \{1, 2, ...K\}$ (not counting vertices and edges, which are the lowest order simplexes) in

the neighborhood is summed individually and the maximum simplex signal is added to the central signal:

$$h_i(x|W^{\mathrm{C}}, W^{\mathrm{N}}, W^{\mathrm{E1}}, W^{\mathrm{E2}}, b) = \sigma\left(W^{\mathrm{C}}x_i + \max_k \left[\frac{1}{|S_k|}\sum_{j\in\mathcal{S}_k}(W^{\mathrm{N}}x_j + W^{\mathrm{E1}}A_{ij}) + \frac{1}{|S_k|^2}\sum_{j,l\in\mathcal{S}_k}(W^{\mathrm{E2}}A_{jl})\right] + b\right). \tag{7.6}$$

This formulation also includes *secondary* edges (A_{jl}), those between different neighbors in the same simplex. It is possible that neighbors and edges occur in more than one simplex in the neighborhood, but that doesn't pose a problem because of the normalization and max function. The use of simplicial complexes in defining convolution operations is currently being investigated by a student in Professor Ben-Hur's research group.

7.2 Extensions of Application

Interface prediction is not the only active area of protein research, and so the modeling of proteins as graphs and use of graph convolution could potentially be used for other protein problems as well. A common characteristic of docking methods is that they create a high number of putative 3D bound structures [44]. Graph convolutional neural networks could also be used to evaluate putative docking solutions and rank them in order of likelihood. For example, the interface derived from a docking solution could be compared to the interface predicted by a trained pairwise graph convolutional neural network and ranked according to similarity with the network prediction. Similarly, much research has been performed in protein modeling, which seeks to predict a protein's secondary and tertiary structures from sequence alone [95]. Though graph convolutions cannot be used to create protein folds *de novo*, they can be used to evaluate a set of putative folds and likewise rank them. In both of these applications, scoring is being performed not at the individual vertex level, but at the graph level. These could take advantage of a global pooling method like

taking the maximum or average across all vertices, or developing a fingerprint similar to work by Duvenaud & Maclaurin, et al. [82].

Any application involving structured data that do not fit naturally into a grid is a potential candidate for graph convolutional approaches. Aside from proteins, other biological and chemical data often meet this criterion. In particular, Quantitative Structure-Activity Relationship (QSAR) and Quantitative Structure-Property Relationship (QSPR) both attempt to predict chemical behavior and properties of proposed chemical structures in order to identify structures with certain desired properties. Identified structures become candidates for laboratory synthesis and experimentation, a much more laborious and time consuming process. This is the focus of the Fingerprint convolution from Duvenaud & Maclaurin et al. [82]. Studies in QSAR and QSPR have long been considered "unquestionably of great importance in modern chemistry and biochemistry" [96]. Like proteins, chemical structures can be modeled as a graph and used to train graph convolutional networks to predict the properties of interest. This is made possible by the vast and growing databases of chemical structures and their corresponding structures [97, 98].

Beyond the biochemical realm, graph structured data abound. Knowledge bases are collections of structured data in the form of triples indicating *(subject, predicate, object)*, where *subject* and *object* are named entities and *predicate* indicates the relationship between them, and pairs indicating *(subject, attribute)*, where *attribute* is a type or property label for *subject*. These data can easily be thought of graphs, where entities are vertices, each type of predicate is one of many different types of edge features, and each attribute is one of many different types of vertex features. Knowledge bases emerge in online information repositories such as YAGO [99], DBPedia [100], and Wikidata [101], and are used in the Google search engine [102]. Due to their large size, knowledge graphs are often under-annotated, where vertex relationships and properties are missing. A graph convolutional neural network can be used to predict properties or relationships on graphs, essentially performing automatic inference on the graph. This is the chief problem of interest for Schichtkrull & Kipf [83].

Bibliography

[1] Fayyaz ul Amir Afsar Minhas, Brian J. Geiss, and Asa Ben-Hur. PAIRpred: Partner-specific prediction of interacting residues from sequence and structure. *Proteins: Structure, Function, and Bioinformatics*, 82(7):1142–1155, 2014.

[2] 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.

[3] Alexandre Faille, Sabine Gavalda, Nawel Slama, Christian Lherbet, Laurent Maveyraud, Valérie Guillet, Françoise Laval, Annaïk Quémard, Lionel Mourey, and Jean-Denis Pedelacq. Insights into substrate modification by dehydratases from type i polyketide synthases. *Journal of Molecular Biology*, 429(10):1554–1569, 2017.

[4] Schrödinger, LLC. The PyMOL molecular graphics system, version 1.8. November 2015.

[5] Hongmin Zhang, Jin-huan Liu, Wei Yang, Timothy Springer, Motomu Shimaoka, and Jiahuai Wang. Structural basis of activation-dependent binding of ligand-mimetic antibody al-57 to integrin lfa-1. *Proceedings of the National Academy of Sciences*, 106(43):18345–18350, 2009.

[6] Eric D. Scheef and J. Lynn Fink. Fundamentals of protein structure. In Philip E. Bourne and Helge Weissig, editors, *Structural Bioinformatics*, chapter 2, pages 15–39. Wiley-Liss, Inc., Hoboken, New Jersey, 2003.

[7] Eric B. Fauman, Andrew L. Hopkins, and Colin R. Groom. Structural bioinformatics in drug discovery. In Philip E. Bourne and Helge Weissig, editors, *Structural Bioinformatics*, chapter 23, pages 477–497. Wiley-Liss, Inc., Hoboken, New Jersey, 2003.

[8] Russ B. Altman and Jonathan M. Dugan. Defining bioinformatics and structural bioinformatics. In Philip E. Bourne and Helge Weissig, editors, *Structural Bioinformatics*, chapter 1, pages 3–14. Wiley-Liss, Inc., Hoboken, New Jersey, 2003.

[9] Aleksandar Bijelic and Annette Rompel. Ten good reasons for the use of the tellurium-centered anderson-evans polyoxotungstate in protein crystallography. *Accounts of Chemical Research*, 50(6):1441–1448, 2017. PMID: 28562014.

[10] Andrea Ilari and Carmelinda Savino. A practical approach to protein crystallography. In Philip E. Bourne and Helge Weissig, editors, *Bioinformatics, Volume I: Data, Sequence Analysis, and Evolution*, chapter 3, pages 47–78. Springer Science+Business Media, New York, New York, 2 edition, 2017.

[11] Hong-Wei Wang and Jia-Wei Wang. How cryo-electron microscopy and x-ray crystallography complement each other. *Protein Science*, 26(1):32–39, 2017.

[12] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235, 2000.

[13] Vishnu Priyanka Reddy Chichili, Veerendra Kumar, and J Sivaraman. A protocol to retain weakly interacting protein complexes for structural studies using an appropriate linker. *Protocol Exchange*, 10, 2013.

[14] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[15] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, May 2010.

[16] Changhui Yan, Feihong Wu, Robert L Jernigan, Drena Dobbs, and Vasant Honavar. Characterization of protein–protein interfaces. *The protein journal*, 27(1):59–70, 2008.

[17] Susan Jones and Janet M Thornton. Principles of protein-protein interactions. *Proceedings of the National Academy of Sciences*, 93(1):13–20, 1996.

[18] James R. Perkins, Ilhem Diboun, Benoit H. Dessailly, Jon G. Lees, and Christine Orengo. Transient protein-protein interactions: Structural, functional, and network properties. *Structure*, 18(10):1233 – 1243, 2010.

[19] Yanay Ofran and Burkhard Rost. Analysing six types of protein-protein interfaces. *Journal of Molecular Biology*, 325(2):377 – 387, 2003.

[20] Shandar Ahmad and Kenji Mizuguchi. Partner-aware prediction of interacting residues in protein-protein complexes from sequence data. *PLOS ONE*, 6(12):1–11, 12 2011.

[21] Rong Chen, Li Li, and Zhiping Weng. Zdock: An initial-stage protein-docking algorithm. *Proteins:Structure, Function, and Bioinformatics*, 51(1):80–87, 2003.

[22] G.C.P van Zundert, J.P.G.L.M. Rodrigues, M. Trellet, C. Schmitz, P.L. Kastritis, E. Karaca, A.S.J. Melquiond, M. van Dijk, S.J. de Vries, and A.M.J.J. Bonvin. The haddock2.2 webserver: User-friendly integrative modeling of biomolecular complexes. *J. Mol. Biol.*, 428:720–725, 2016.

[23] Iakes Ezkurdia, Lisa Bartoli, Piero Fariselli, Rita Casadio, Alfonso Valencia, and Michael L. Tress. Progress and challenges in predicting protein-protein interaction sites. *Briefings in Bioinformatics*, 10(3):233–246, 2009.

[24] Nurcan Tuncbag, Attila Gursoy, and Ozlem Keskin. Prediction of protein-protein interactions: unifying evolution and structure at protein interfaces. *Physical Biology*, 8(3):035006, 2011.

[25] R. Esmaielbeiki, K. Krawczyk, B. Knapp, J.-C. Nebel, and C. M. Deane. Progress and challenges in predicting protein interfaces. *Briefings in Bioinformatics*, (January):1–15, 2015.

[26] Joël Janin. Protein-protein recognition. *Progress in biophysics and molecular biology*, 64(2-3):145–166, 1995.

[27] Olivier Lichtarge, Henry R Bourne, and Fred E Cohen. An evolutionary trace method defines binding surfaces common to protein families. *Journal of molecular biology*, 257(2):342–358, 1996.

[28] Florencio Pazos, Manuela Helmer-Citterich, Gabriele Ausiello, and Alfonso Valencia. Correlated mutations contain information about protein-protein interaction. *Journal of molecular biology*, 271(4):511–523, 1997.

[29] Xavier Gallet, Benoit Charloteaux, Annick Thomas, and Robert Brasseur. A fast method to predict protein interaction sites from sequences. *Journal of molecular biology*, 302(4):917–926, 2000.

[30] Susan Jones and Janet M Thornton. Prediction of protein-protein interaction sites using patch analysis. *Journal of molecular biology*, 272(1):133–143, 1997.

[31] Joël Janin, Kim Henrick, John Moult, Lynn Ten Eyck, Michael JE Sternberg, Sandor Vajda, Ilya Vakser, and Shoshana J Wodak. Capri: a critical assessment of predicted interactions. *Proteins: Structure, Function, and Bioinformatics*, 52(1):2–9, 2003.

[32] Rong Chen and Zhiping Weng. Docking unbound proteins using shape complementarity, desolvation, and electrostatics. *Proteins: Structure, Function, and Bioinformatics*, 47(3):281–294, 2002.

[33] Asako Koike and Toshihisa Takagi. Prediction of protein–protein interaction sites using support vector machines. *Protein Engineering Design and Selection*, 17(2):165–173, 2004.

[34] James R Bradford and David R Westhead. Improved prediction of protein–protein binding sites using a support vector machines approach. *Bioinformatics*, 21(8):1487–1494, 2004.

[35] Huan-Xiang Zhou and Yibing Shan. Prediction of protein interaction sites from sequence profile and residue neighbor list. *Proteins: Structure, Function, and Bioinformatics*, 44(3):336–343, 2001.

[36] Huiling Chen and Huan-Xiang Zhou. Prediction of interface residues in protein–protein complexes by a consensus neural network method: test against nmr data. *Proteins: Structure, Function, and Bioinformatics*, 61(1):21–35, 2005.

[37] James R. Bradford, Chris J. Needham, Andrew J. Bulpitt, and David R. Westhead. Insights into protein-protein interfaces using a bayesian network prediction method. *Journal of Molecular Biology*, 362(2):365 – 386, 2006.

[38] Torben Friedrich, Birgit Pils, Thomas Dandekar, Jörg Schultz, and Tobias Müller. Modelling interaction sites in protein domains with interaction profile hidden markov models. *Bioinformatics*, 22(23):2851–2857, 2006.

[39] Sean R. Eddy. Profile hidden markov models. *Bioinformatics (Oxford, England)*, 14(9):755–763, 1998.

[40] Ming-Hui Li, Lei Lin, Xiao-Long Wang, and Tao Liu. Protein–protein interaction site prediction based on conditional random fields, 2007.

[41] Huan-Xiang Zhou and Sanbo Qin. Interaction-site prediction for protein complexes: a critical assessment. *Bioinformatics*, 23(17):2203–2209, 2007.

[42] Howook Hwang, Brian Pierce, Julian Mintseris, Joël Janin, and Zhiping Weng. Protein–protein docking benchmark version 3.0. *Proteins: Structure, Function, and Bioinformatics*, 73(3):705–709, 2008.

[43] Howook Hwang, Thom Vreven, Joël Janin, and Zhiping Weng. Protein–protein docking benchmark version 4.0. *Proteins: Structure, Function, and Bioinformatics*, 78(15):3111–3114, 2010.

[44] Joël Janin. Docking predictions of protein-protein interactions and their assessment: the capri experiment. In *Identification of Ligand Binding Site and Protein-Protein Interaction Area*, pages 87–104. Springer, 2013.

[45] Santiago Ramón y Cajal and Léon Azoulay. *Histologie du système nerveux de l'homme & des vertébrés*. Consejo superior de investigaciones cientificas, Instituto Ramon Y Cajal, 1955.

[46] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[47] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[48] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[49] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[50] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[51] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[52] Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[53] Elijah Polak and Gerard Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, 3(16):35–43, 1969.

[54] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[55] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[56] R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, Sep 1993.

[57] PJ Dalianis, SG Tzafestas, and G Anthopoulos. A study of the generalization capability versus training in backpropagation neural networks. In *Systems, Man and Cybernetics, 1993.'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on*, volume 4, pages 485–490. IEEE, 1993.

[58] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in neural information processing systems*, pages 630–637, 1990.

[59] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[61] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[62] Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pages 2791–2799, 2015.

[63] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[64] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[65] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59, 2011.

[66] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[67] 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.

[68] 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.

[69] T Ciodaro, D Deva, JM De Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. In *Journal of physics: conference series*, volume 368, page 012030. IOP Publishing, 2012.

[70] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[71] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[72] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[73] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[74] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

[75] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

[76] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

[77] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015.

[78] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[79] Stéphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2009.

[80] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[81] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016.

[82] David K. Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015.

[83] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017.

[84] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.

[85] Thom Vreven, Iain H Moal, Anna Vangone, Brian G Pierce, Panagiotis L Kastritis, Mieczyslaw Torchala, Raphael Chaleil, Brian Jiménez-García, Paul A Bates, Juan Fernandez-Recio, et al. Updates to the integrated protein–protein interaction benchmarks: docking benchmark version 5 and affinity benchmark version 2. *Journal of molecular biology*, 427(19):3031–3041, 2015.

[86] Alexey G Murzin, Steven E Brenner, Tim Hubbard, and Cyrus Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, 1995.

[87] Yanay Ofran and Burkhard Rost. Isis: interaction sites identified from sequence. *Bioinformatics*, 23(2):e13–e16, 2007.

[88] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[89] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[91] 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. Software available from tensorflow.org.

[92] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[93] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[94] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[95] Torsten Schwede. Protein modeling: what happened to the "protein structure gap"? *Structure*, 21(9):1531–1540, 2013.

[96] Mati Karelson, Victor S Lobanov, and Alan R Katritzky. Quantum-chemical descriptors in qsar/qspr studies. *Chemical reviews*, 96(3):1027–1044, 1996.

[97] Marius Olah, Ramona Rad, Liliana Ostopovici, Alina Bora, Nicoleta Hadaruga, Dan Hadaruga, Ramona Moldovan, Adriana Fulias, Maria Mractc, and Tudor I Oprea. Wombat and wombat-pk: bioactivity databases for lead and drug discovery. *Chemical Biology: From Small Molecules to Systems Biology and Drug Design, Volume 1-3*, pages 760–786, 2008.

[98] Richard Judson, Ann Richard, David Dix, Keith Houck, Fathi Elloumi, Matthew Martin, Tommy Cathey, Thomas R Transue, Richard Spencer, and Maritja Wolf. Actor-aggregated computational toxicology resource. *Toxicology and applied pharmacology*, 233(1):7–13, 2008.

[99] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th Biennial Conference on Innovative Data Systems Research*. CIDR Conference, 2014.

[100] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[101] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[102] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 2012.

[103] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[104] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[105] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[106] Matthias Heinig and Dmitrij Frishman. Stride: a web server for secondary structure assignment from known atomic coordinates of proteins. *Nucleic acids research*, 32(suppl_2):W500–W502, 2004.

[107] Frank Eisenhaber, Philip Lijnzaad, Patrick Argos, Chris Sander, and Michael Scharf. The double cubic lattice method: efficient approaches to numerical integration of surface area and volume and to dot surface contouring of molecular assemblies. *Journal of Computational Chemistry*, 16(3):273–284, 1995.

[108] Michel F Sanner, Arthur J Olson, and Jean-Claude Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[109] Josip Mihel, Mile Šikić, Sanja Tomić, Branko Jeren, and Kristian Vlahoviček. Psaia–protein structure and interaction analyzer. *BMC structural biology*, 8(1):21, 2008.

[110] Jack Kyte and Russell F Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology*, 157(1):105–132, 1982.

Appendix A

Features

The raw data associated with protein structures are found in PDB files, which contain the protein sequence of amino acids and the atomic coordinates of each atom in each amino acid. When performing interface prediction, a number of derived features have been found useful, which are either calculated directly from the PDB data, or from other sources such as sequence databases. Here, the term *features* describes a vector of values associated with either a vertex or edge in a graph, with vertices corresponding to amino acid residues, and edges the relationships between two residues. The term *feature* then is just a single element of the vector. Some methods described below generate more than one feature, hence they are grouped into appropriate *categories*. This appendix describes each feature category in detail, including the number of features generated, external tools, and a description of the feature. In addition to the tools specified for each feature category, extensive use was made of Biopython [103] to parse PDB files into convenient data structures.

A.1 Vertex Features

A.1.1 Windowed Position Specific Scoring Matrix (PSSM)

Number of features: 20

External tools: PSI-BLAST [104]

Description: The PSSM is a set of features constructed from protein sequence alone, without any structural information. It captures the relative abundance of each type of amino acid in proteins which share a similar sequence in a window around the amino acid of interest. Proteins with similar sequences to the search window are identified through PSI-BLAST, a position specific, iterative version of BLAST, the Basic Local Alignment and Search Tool [105]. The similar sequences are then used to generate a summary in a window around the residue of interest. For each window

position and each residue type, a score is computed by comparing the observed counts of that residue type in that window position to the expected count generated by a null model. The score is then $\log(Q_i/P_i)$ where Q_i is the fraction of sequences with that residue type in that position, and P_i is the expected fraction based on data-dependent pseudocounts (more detail can be found in Altschul, et al. [105]. For this thesis, only the scores at the residue of interest were considered (a window size of one), so there are 20 features, one for each amino acid type.

A.1.2 Relative Accessible Surface Area (rASA)

Number of features: 1

External tools: STRIDE [106]

Description: Relative Accessible Surface Area, or *Solvent Accessible Area*, reflects the fraction of a residue that is exposed to a potential solvent. This is computed by sliding a spherical probe of radius of 1.4Å (approximating the radius of a water molecule [107]) over the van der Waals surface of the protein near the residue of interest. The Area generated by the center of the probe as it is in contact with the residue is taken to be the accessible surface area. This is divided by the maximum possible accessible surface area to achieve a relative measure.

A.1.3 Residue Depth

Number of features: 2

External tools: MSMS [108]

Description: Residue Depth is the minimum distance from the residue to the surface of the protein, as calculated by MSMS. The distance is calculated in two ways: by taking the average distance of all non-hydrogen atoms in the residue to the surface, and by taking the distance of the α-carbon to the surface.

A.1.4 Protrusion Index

Number of features: 6

External tools: PSAIA [109]

Description: The protrusion index indicates the degree to which a sphere (radius 10Å) centered at a non-hydrogen atom is not filled with other atoms. V_{int} is the volume occupied by atoms, which is calculated as the number of atoms times the mean atomic volume found in proteins. V_{ext} is the difference between the total sphere volume and V_{int}. The protrusion index is defined as V_{ext}/V_{int}. This value is calculated for each non-hydrogen atom in the residue, then the following summary statistics are taken:

- *Total mean*: The average over all non-hydrogen atoms in the residue
- *Total standard deviation*: The standard deviation over all non-hydrogen atoms in the residue
- *Side chain mean*: The average over just non-hydrogen atoms in the residue side chain
- *Side chain standard deviation*: The standard deviation over just non-hydrogen atoms in the residue side chain
- *maximum*: The maximum over all non-hydrogen atoms
- *minimum*: The minimum over all non-hydrogen atoms

All of these features are normalized to values between 0 and 1 on a per-protein basis. Normalizing across proteins was avoided so that normalizations of existing proteins in the training set would not change if additional proteins are added later.

A.1.5 Hydrophobicity

Number of features: 1

External tools: PSAIA [109]

Description: Hydrophobicity indicates the tendency for an amino acid to avoid water. PSAIA calculates this value base on the scale defined by Kyte & Doolittle [110]. Hydrophobicity values were normalized between 0 and 1 on a per-protein basis.

A.1.6 Half Sphere Amino Acid Composition

Number of features: 40

External tools: None

Description: These features capture the residue composition in a neighborhood of the residue of interest. The neighborhood is defined as all residues with at least one atom closer than 8Å to at least one atom in the residue of interest. An array of counts is created which stores the number of each type of residue in the neighborhood, which is then divided by the total count of all types. This array is calculated for two regions, separated by the null space of the vector that is the average of the unit vectors from the α-carbon to the side chain atoms.

A.2 Edge Features

A.2.1 Average Atomic Distance

Number of features: 1

External tools: None

Description: This feature is computed by taking the average of the distance between any two atoms of the two residues of interest. The raw distance is then passed through a Gaussian function:

$$f(x) = e^{-x^2/\sigma^2}, \tag{A.1}$$

where σ is chosen by model selection, so that this feature has a higher value for residues that are closer together.

A.2.2 $CC_\alpha O$ Angle

Number of features: 1

External tools: None

Description: This feature captures the relative orientation of two residues as defined by the normals to their adjacent peptide planes. Let X_{C_α}, X_C, and X_O be the positions vectors of the α-carbon,

carboxyl-group carbon, and carboxyl-group oxygen, respectively. The normal is then defined as:

$$\hat{N} = \frac{(X_O - X_C) \times (X_{C_\alpha} - X_C)}{||(X_O - X_C) \times (X_{C_\alpha} - X_C)||}, \tag{A.2}$$

where $\times$ denotes the cross product, and $||\cdot||$ the L^2 norm. The angle between residues i and j is then:

$$\text{C C}_\alpha\text{O} = \cos^{-1}\left(\frac{\hat{N}_i \cdot \hat{N}_j}{||\hat{N}_i||\ ||\hat{N}_j||}\right) \tag{A.3}$$

Finally, the angle is normalized to be between 0 and 1 by dividing by 2π.

A.3 Missing Data

In Section 5.1.2 it was noted that 13 residues were not provided with an α-carbon in the corresponding PDB file. These residues were determined to be partial records at the end of a protein chain, where only a single atom was listed. This made calculation of Half Sphere Amino Acid Composition, Residue Depth, and $CC_\alpha O$ Angle impossible. Upon inspection, these partial records were deemed to be legitimate residues and so were retained, with the missing features imputed according to the procedure described in Chapter 5.

Appendix B

Software Implementation

The main software used in this thesis was written in Python version 2.7.13. Rather than writing several individual scripts which each ran their own experiments, the software was written as a unified framework which would allow consistent use of data, rapid experimentation, and reuse of critical experimental code. This experiment briefly describes some of the notable elements of the software.

B.1 Experiment Specification Files

Experiments are specified using the YAML data format, a human readable data serialization language. To run an experiment, the user calls the main experiment running script and passes the specification file name as an argument. The specification file fully defines the experiment(s) being run, to include the data format, model architecture, and training/testing procedure. This has a number of advantages. First, it ensures traceability from experimental conditions to results. A copy of the experiment file is placed in the same directory of the experiment results, ensuring that any results from past experiments can be understood and interpreted appropriately. Second, it allows rapid iteration of experimentation, since running a modified version of an experiment is requires a simple modification of the specification file. Third, it allows concurrent researchers to review each others' work without having lengthy discussions about the precise experimental settings used. For illustration, the contents of an example YAML file are shown below (for Sum Coupling, receptive field size 21, for one, two , three, and four layers):

```
type: 'experimentset'
common:
  type: 'traintest'
  args:
    test_frequency: 80
```

```
  vis_frequency: 80
  test_before_train: True
  num_epochs: 80
  minibatch_size: 128
  test_batch_size: 2000
  checkpoint_frequency: 80
data:
  dataset: "dbd5"
  train_list: ["train_list_w_2B42.yml", "validate_list.yml"]
  test_list: "test_list_w_3R9A.yml"
  n_train: -1
  n_test: -1
  train_ratio: 10
  test_ratio: -1
  nans: "global_median"
  features:
  - "WINDOWED_POSITION_SPECIFIC_SCORING_MATRIX"
  - "RELATIVE_ACCESSIBLE_SURFACE_AREA"
  - "RESIDUE_DEPTH"
  - "PROTRUSION_INDEX"
  - "HALF_SPHERE_EXPOSURE"
  - "RES_DIST_AVE"
  #- "RES_DIST_STD"
  - "ANGLE_CCAO"
  #- "ANGLE_HSAA_COS"
  feature_args:
    profile_params:
      window_size: 1
      blast-database: "/s/<host>/a/tmp/blastdb/db"
```

```
      distance_matrix:
        preprocess: ["gaussian_kernel", 18]
      angle_matrix:
        preprocess: "divide_pi"
    representation: "pairwise_data_per_protein_with_hood_indices"
  metrics:
    - "learning_rate"
    - "params"
    - "loss_train"
    - "loss_test"
    - "roc_train"
    - "roc_test"
    - "rfpp_train"
    - "rfpp_test"
  model:
    name: "PWClassifier"
    layer_args: {init: "he", coupling: "sum", bias_init: "zero",
        nonlin: "relu", dropout_keep_prob: 0.5}
    loss: "tf_weighted_crossent"
    loss_args:
      pn_ratio: 0.1
    optimizer: "tf_sgd"
    optimizer_args:
      learning_rate: 0.1
experiments:
- - "1_layer_rf20"
  - data:
      nhood_size: 20
    model:
```

```
      layers:
      - ["star_v_conv", {filters: 256}]
      - ["merge_gather_swap_paired_examples",
             {mode: ["concat"]}, ["merge"]]
      - ["dense", {out_dims: 512}]
      - ["dense", {out_dims: 1, nonlin: "linear"}]
      - ["combine_swapped"]
- - "2_layer_rf20"
  - data:
      nhood_size: 20
    model:
      layers:
      - ["star_v_conv", {filters: 256}]
      - ["star_v_conv", {filters: 512}]
      - ["merge_gather_swap_paired_examples",
             {mode: ["concat"]}, ["merge"]]
      - ["dense", {out_dims: 512}]
      - ["dense", {out_dims: 1, nonlin: "linear"}]
      - ["combine_swapped"]
- - "3_layer_rf20"
  - data:
      nhood_size: 20
    model:
      layers:
      - ["star_v_conv", {filters: 256}]
      - ["star_v_conv", {filters: 256}]
      - ["star_v_conv", {filters: 512}]
      - ["merge_gather_swap_paired_examples",
             {mode: ["concat"]}, ["merge"]]
```

```
      - ["dense", {out_dims: 512}]
      - ["dense", {out_dims: 1, nonlin: "linear"}]
      - ["combine_swapped"]
- - "4_layer_rf20"
  - data:
      nhood_size: 20
    model:
      layers:
      - ["star_v_conv", {filters: 256}]
      - ["star_v_conv", {filters: 256}]
      - ["star_v_conv", {filters: 512}]
      - ["star_v_conv", {filters: 512}]
      - ["merge_gather_swap_paired_examples",
            {mode: ["concat"]}, ["merge"]]
      - ["dense", {out_dims: 512}]
      - ["dense", {out_dims: 1, nonlin: "linear"}]
      - ["combine_swapped"]
```

B.2 Convolution Functions

Because all convolution approaches used a similar neural network architecture, each layer type was implemented as a function, and the relevant function is named in the experiment specification file (for example, "star_v_conv" in the YAML code above). Each layer function would receive as inputs the outputs of the previous layer(s), and any number of keyword arguments specific to that layer. The layer would define any TensorFlow operations on the inputs that produce the layer's output. Below is an example function implementing a convolutional layer, which is a simplified version of "star_v_conv" mentioned above.

```
def star_v_conv(layer_input, filters, init, bias_init, nonlin,
                coupling, dropout_keep_prob, bias, **kwargs):
```

```
# get inputs
vertices, edges, nh_indices = layer_input
v_shape = vertices.get_shape()
e_shape = edges.get_shape()
nh_size = nh_indices.get_shape()[1].value

# create weights
Wvc = tf.Variable(initializer(init, (v_shape[1].value, filters)))
bv = tf.Variable(initializer(bias_init, (filters,)), name="bv")
Wvn = tf.Variable(initializer(init, (v_shape[1].value, filters)))
We = tf.Variable(initializer(init, (e_shape[2].value, filters)))

# create central vertex signals
Zc = tf.matmul(vertices, Wvc, name="Zc")

# create neighbor signals
e_We = tf.tensordot(edges, We, axes=[[2], [0]], name="e_We")
v_Wvn = tf.matmul(vertices, Wvn, name="v_Wvn")
if coupling == "sum":
    Zn = tf.divide(
         tf.reduce_sum(tf.gather(v_Wvn, nh_indices), 1) \\
         + tf.reduce_sum(e_We, 1), \\
         nh_size)
elif coupling == "product":
    Zn = tf.divide(
         tf.reduce_sum(tf.gather(v_Wvn, nh_indices) \\
         * e_We, axis=1), \\
         nh_size)
```

```
    # signal
    sig = Zn + Zc
    if bias:
        sig += bv

    # pass through the nonlinearity
    z = tf.reshape(nonlin(sig), tf.constant([-1, filters]))

    # dropout
    z = tf.nn.dropout(z, dropout_keep_prob)
    return z
```