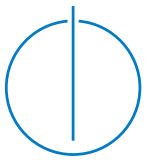
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Deep Learning for Recognition of Protein Domains from Contact Maps of *ab initio* Models

Benjamin Gallusser



DEPARTMENT OF INFORMATICS

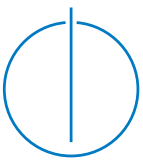
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Deep Learning for Recognition of Protein Domains from Contact Maps of *ab initio* Models

Deep Learning zur Erkennung von Proteindomänen aus Kontaktkarten von *ab-initio*-Modellen

Author:	Benjamin Gallusser
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	M.Sc. Vladimir Golkov
Submission Date:	September 15, 2016



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, September 15, 2016 Benjamin Gallusser

Acknowledgments

I am extraordinarily grateful to Vladimir Golkov for his truly excellent mentoring throughout this thesis. Not only his competent advice at all times, but also his enthusiasm and our ongoing discussions about new ideas were an essential contribution to the project. Moreover, I wish to thank Axel Fischer, Nicholas Hyman and Marcin Skwark for providing the used data and answering all of my questions about the biochemical background of this work. Finally, I would like to thank Andreas Gallusser and Georgi Dikov for critically reading the draft version of this thesis.

Abstract

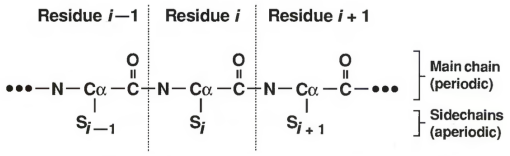
Protein structure prediction is considered one of the biggest challenges in Computational Biology. Current *de novo* approaches output many different low-level fold models for a protein sequence, which are clustered and then passed on to refinement methods. This work aims to introduce a new clustering method for the mentioned workflow. We represent the protein models as contact maps (matrices of pairwise contacts of amino acids). In our first approach, each model is classified according to the CATH database with a convolutional neural network. Due to variable input size and distributed features, our convolutional network uses 2D and 1D global pooling. Furthermore, we propose online batch selection with a priority queue to tackle class imbalance. We also introduce multi-level classification, which outperforms normal classification in terms of normal-classification loss. Most notably, the A-level loss is reduced by 60% if a network is trained with three-level classification loss. Finally, we reach a TOP1 classification accuracy of 86% and a ground-truth-in-TOP10 classification accuracy of 96% on an artificial dataset of perturbed CATH models. Our second approach uses the features extracted by a convolutional neural network for unsupervised clustering. Both approaches are discussed and evaluated in this thesis.

Contents

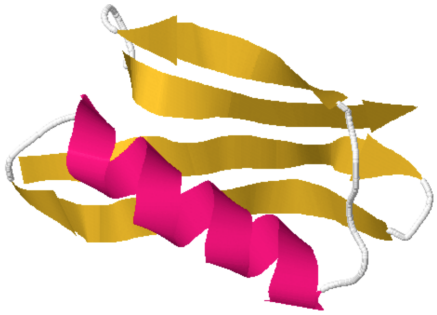
1 Introduction

1.1 Protein structure

Proteins are large biomolecules that perform a variety of different tasks within all living organisms. A protein consists of a linear sequence of amino acids ranging from only a few peptides up to several hundreds. They are linked to each other by peptide bonds and form a polypeptide chain consisting of a so-called backbone (or main chain) and side chains (see Figure 1.1a). The side chain of each amino acid residue is attached to its Cα-atom. The polypeptide chain is flexible



(a) The structural formula of a protein. By convention, polypeptide sequences are numbered from N-terminus (left) to C-terminus (right). Figure from [1].



(b) Example of a protein domain with a β-sheet consisting of four β-strands and one α-helix. It is the B1 immunoglobulin-binding domain of streptococcal protein G, PDB-ID: 1PGA [2].

Figure 1.1: Protein structure visualized

and can fold into many different so-called conformations. The most stable and energetically favorable of these conformations is called the native state or native structure.

Amino acids are the building blocks of proteins. The 20 naturally existing amino acids can be combined to a sequence of arbitrary length. What this means is that a protein of e.g. 200 amino acids could theoretically have 20^{200} different amino acid sequences. The native state of proteins is prone to small structural changes depending on temperature and solvent and it can also be irreversibly modified by e.g. heating. This is called protein denaturation. Nevertheless, under given conditions a protein always folds to a specific well-defined conformation. For describing

a protein, four distinct levels of protein structure have been defined:

- The amino acid sequence of a protein is called **primary structure**.
- **Secondary structure**: These are standard pieces of protein structure that most proteins are assembled from. The two most important secondary structure elements (SSEs) are α-helices and β-sheets. A total of eight different secondary structures have been defined in the Dictionary of Protein Secondary Structure (DSSP) [3].
  There are three different types of helices: 3_{10}-helices (three residues in one "loop" of the helix), α-helices (approx. four residues) and π-helices (approx. five residues), with α-helices being much more common than the other two types of helices. In schematic visual representations of proteins, helices are usually depicted (regardless of the type) either as cylinders or thick, colored spirals Figure 1.1b.
  β-sheets always consist of at least two neighboring β-strands, that is to say more or less spatially straight fractions of the protein sequence connected by at least two hydrogen bonds. If two β-strands are only connected by one hydrogen bond we speak of a β-bridge. β-strands are normally illustrated as thick arrows in schematic visual representations of proteins. Any SSE that does not match any of the above descriptions is called loop. This includes coils, turns and bends. They do not have a well defined spatial arrangement like helices and β-sheets do.
- The manner of how SSEs are combined to form one or several so-called protein domains is called **tertiary structure** or folding pattern. There is a vast amount of different protein folding patterns, the space of folding patterns is only slightly limited by some physical constraints. The tertiary structure highly influences the function of a protein. A protein can consist of only one domain, but can also contain multiple domains connected by linker structures.
- **Quaternary structure**: Proteins can aggregate together to form a so-called protein complex. These are multimers, meaning that they consist of multiple polypeptide chains. The interactions between proteins in a protein complex are usually non-covalent. Rarely, they can be covalent due to posttranslational modifications. The spatial arrangement of the individual subunits within the protein complex is called quaternary structure and is the highest level of protein structure abstraction.

1.2 Contact maps

A contact map M is a representation of a protein tertiary structure. For a protein of n residues, $M \in \mathbb{R}^{n \times n}$ and M symmetric. Each entry $m_{i,j} \in [0,1]$ specifies [1] whether the minimal distance λ between the Cα-atoms (see Figure 1.1a) and Cβ-atoms (C atom in the side chain closest to the Cα-atom) of residues i and j is less than a threshold, with

$$\lambda_{i,j} = \min\{\text{dist}(C\alpha_i, C\alpha_j), \text{dist}(C\alpha_i, C\beta_j), \text{dist}(C\beta_i, C\alpha_j), \text{dist}(C\beta_i, C\beta_j)\}. \tag{1.1}$$

In the used data set, a symmetrical matrix $(d_{kl})_{kl} = D \in \mathbb{R}^{20 \times 20}$ defines individual thresholds for each pair of two types of residues based on sizes of the side chains. These thresholds vary between 4.5 Å and 12.8 Å (1 angstrom equals 10^{-10} m). The value $m_{i,j}$ is 0 if $\lambda_{i,j}$ is 1 Å or more above the threshold, it is 1 if $\lambda_{i,j}$ is 1 Å or more below the threshold, and the values in between are interpolated linearly, yielding the overall formula:

$$m_{i,j} = \frac{\max(u - \max(\lambda, l), 0)}{2}, \tag{1.2}$$

$$u = d_{\text{type(i),type(j)}} + 1\mathring{A}, \tag{1.3}$$

$$l = d_{\text{type(i),type(j)}} - 1\mathring{A}, \tag{1.4}$$

with $m_{i,j} = 0$ representing a distance too large for physical interaction between the two residues and $m_{i,j} = 1$ representing a distance for likely physical interaction. Contacts of adjacent residues in the sequence are also depicted, resulting in a continuous band around the matrix diagonal. A contact map contains the following information about contacts within SSEs and SSE-SSE contacts:

- Helices: A broadening of the band around the diagonal indicates the existence of a helix.
- Anti-parallel β-strands appear as continuous cross-diagonals.
- Parallel β-strands are depicted as continuous strips parallel to the diagonal. They have to be clearly separated from the constant diagonal that represents the linear sequence contacts.
- Aligned helix-helix contacts and aligned helix-strand contacts can be identified by sparse cross-diagonals if anti-parallel and sparse strips parallel to the diagonal if parallel (see Figure 1.2). Idealized helix-helix and helix-strand contacts have a distinct sparsity pattern, but in real contact maps they are sometimes hard to distinguish.

[1] In literature, $m_{i,j}$ is often defined as a binary value, but in the used dataset it is continuous because this provides additional information.

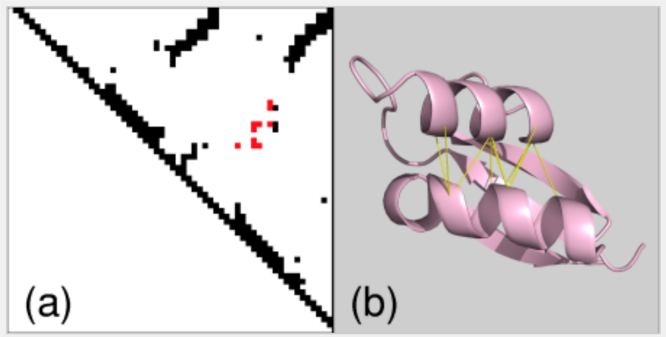


Figure 1.2: (a) The contact map of the ribosomal protein L30 (PDB-ID: 1BXY), the redundant lower triangle is not shown. The contacts shown in red form a sparse cross-diagonal indicating the existence of an antiparallel helix-helix contact. (b) The highlighted contacts are shown as yellow lines in the visualization, representing the hydrogen bonds between the residues. Figure from [4].

1.3 Protein structure prediction

Protein structure prediction is the task of predicting protein tertiary structure solely from its amino acid sequence. It is considered to be one of the most important challenges in Computational Biology and Bioinformatics. Every two years since 1994 the community-wide *Critical Assessment of protein Structure Prediction* (CASP) has been carried out. Research groups from all over the world submit their algorithms for specific (sub-)challenges and objectively evaluate them on new proteins whose structure has been resolved experimentally but not yet published. Reliably predicting protein tertiary structure is of great importance for a variety of applications such as drug design. The entire procedure can be split up into various sub-tasks, such as predicting SSEs from the sequence, assembly of SSEs to obtain possible tertiary structures etc. A lot of approaches use the properties of already resolved structures as prior knowledge. Two general approaches exist to predict protein folding:

> “
> - homology modelling - prediction of an unknown structure from the known structure of one or more related proteins;
> - fold recognition - identification of a folding pattern from a library that is predicted to resemble the folding pattern of target protein.
>
> *from [5]*
> ”

On the contrary, methods that seek to build three-dimensional protein models “from scratch”, i.e. based on physical principles rather than on previously resolved structures are called *ab initio*,

de novo or *a priori* protein modelling. Some authors use these terms under slightly different meanings depending on whether prior knowledge is included.

1.4 Machine Learning and Convolutional Neural Networks

Machine Learning is an area of Artificial Intelligence that gained a lot of attention due to the need of processing big amounts of data and due to the availability of sufficiently fast hardware to handle the demanding calculations of state-of-the-art algorithms. Machine Learning can be roughly divided into several types of learning, most notably supervised and unsupervised. Supervised learning aims to do a regression or classification task after the model has been trained with labeled test data whereas the goal of unsupervised learning is to discover hidden structures in a big amount of unlabeled data.

One particularly popular supervised learning technique in Machine Learning is called Artificial Neural Networks (ANN). ANNs are inspired by the biological architecture and functionality of the brain. While the first general models have been defined in the 1940s, the predecessors of today's dominating ANNs came into use with the application of backpropagation to multi-layer neural networks proposed by P. Werbos [6] in 1974. Multiple layers of neurons are able to transform the feature space nonlinearly in a way such that a linear classification can be performed at the end of the network.

In image recognition, a specific kind of ANN called Convolutional Neural Network (CNN) is currently outperforming all other approaches. The basic idea of one of its layers is that a small-sized filter is shifted along the input space performing translation-covariant transformations. Although CNNs have been proposed as early as the 1980s, they only came to success in recent years with different CNN architectures performing extraordinarily well at the annual ImageNet Large Scale Visual Recognition Competition (ILSVRC) [7, 8]. As these networks generally consist of more layers than the ones used before, they form an important part of the current *Deep Learning* research field. After the sweeping success of [7] in 2012, CNNs were applied to a large number of Computational Biology problems (a summary of those can be found in [9]).

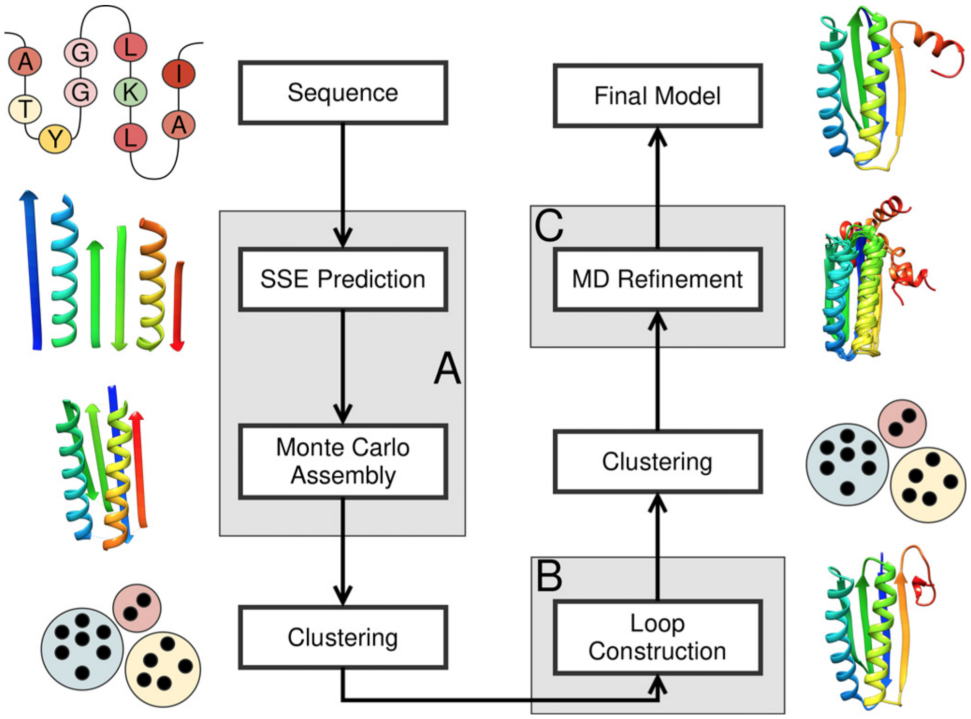


Figure 1.3: The pipeline of the modular BCL::Fold-based protein structure prediction. Figure taken from [10].

.

1.5 BCL::Fold-based protein structure prediction

BCL::Fold [11] is a low-resolution *de novo* protein structure prediction algorithm developed at Vanderbilt University in Nashville, TN. Folding predictions were submitted to CASP for the first time in 2010 (CASP9). BCL::Fold is part of the BioChemical Library [2], a software package with applications for numerous areas of biological research.

The pipeline of an entire BCL::Fold-based protein structure prediction is shown in Figure 1.3. The first module, which is the actual BCL::Fold algorithm, creates a pool of SSEs (helices and β-strands) based on the protein sequence using prior knowledge. These are afterwards assembled to different 3D-models without loop regions using simulated annealing (Monte Carlo assembly algorithm). The knowledge-based fitness function is called BCL::Score [12] and estimates the free energy of each conformation. The created conformations work with predefined idealized angles, which limits the search space of the Monte Carlo algorithm. On the downside, this leads

[2] `http://www.meilerlab.org/index.php/bclcommons/show/b_apps_id/1`, available free of charge for academic users

to a severe simplification of the SSE elements, such as only using very straight β-strands, even though they are often bent in reality.

The obtained pool of conformations without loop regions is clustered to reduce the amount of models to 10 to 50 representatives, which are subsequently fed into the second module. This module adds loops to the conformations and conducts a high-resolution refinement on them, based on ROSETTA [13].

After another iteration of clustering, the representatives of each cluster are passed on to the third module, which does Molecular Dynamics simulations for each of them and returns five final model suggestions as the result of the pipeline.

1.6 Research goals and approaches

The performance of the whole pipeline was critically evaluated by the developers in [10] and necessary improvements were defined. Finding good representatives of the models created by BCL::Fold was identified as the weakest step in the pipeline. This work aims to improve this step by using Deep Learning in order to find representatives to be passed on to the loop construction and side-chain placement algorithm based on ROSETTA (module B in Figure 1.3).

Two approaches are presented in this work. In one approach, the models produced by BCL::Fold are classified using the hierarchical protein structure classification system of CATH [14] (see section 3.1) with a supervised CNN. These models are presented to the CNN in the form of contact maps. This approach can be considered to be part of a *de novo* protein folding approach even though it uses prior knowledge about resolved protein structures as part of the training set. In the other approach, an unsupervised clustering algorithm is applied to deep features extracted by CNNs from contact maps of models produced by BCL::Fold.

2 Related work

2.1 Current clustering approach

The original approach used by BCL::Fold-based protein structure prediction for clustering the models without loop regions was presented in [15]. It is a standard hierarchical agglomerative clustering method, which can be visualized with a dendrogram. It uses a pairwise distance matrix of the samples as input, the distance metric being flexible. The root-mean-square deviation (RMSD) of Cα-atoms was used as the distance metric for the original algorithm of CASP9. The mean inner distance of the clusters was approximately 10 Å. The centers of the 10 to 20 largest clusters were chosen for further processing. On top of that, the best scoring conformation in each cluster was passed on too, as well as the five models evaluated as best by BCL::Score.

For CASP11, the clustering algorithm was changed to a k-medoids algorithm, $k \in \{10, 11, \ldots, 50\}$, but the significant loss in model accuracy during the clustering step remained. The distance metric was changed as well for CASP11 to the $\text{RMSD100} = \frac{\text{RMSD}}{\ln \sqrt{L/100}}$, where L is the number of residues in the protein.

2.2 Other fold clustering/prediction approaches

Gupta et al. presented a different approach of fold prediction in 2005 using contact maps [16]. They propose a linear combination of five custom-designed parameters to determine the similarity between two contact maps. Thus, it is possible to compare an *a priori* contact map to the contact maps of known structures and thereby choose the best-matching known fold as the prediction. Two of the five criteria are based on graph theory, the other three on physical properties of the protein domain. The results of this method are promising on the relatively small dataset that the authors used (24 domains) but have not been reproduced on a bigger dataset to date.

Another clustering approach that is used in protein structure prediction and is similar to the one described above in section 2.1 is Gray's et al. work [17] on protein-protein docking structure prediction. They successfully apply hierarchical clustering to the approximately 200 best decoys per target using a knowledge-based scoring function. The maximum inner difference of a cluster is limited to a fixed value, resulting in a certain amount of clusters. The representative of each cluster is its best-scoring decoy. The representatives of the clusters with most members are chosen as final predictions.

Yet another method that clusters protein conformations [18] applies dimensionality reduction to the pairwise distance matrix of Cα-atoms of each protein and uses the obtained features for an adapted *self-organizing map* clustering [19].

3 The dataset

3.1 CATH

CATH [14] is one of the most important databases for protein structure classification. The most important characteristic of CATH is its multi-level semi-automatic hierarchical classification systems of protein domains based on both structural similarity and homology (descent from a common ancestor). The protein database SCOP and its successor SCOP2 possess a similar classification system using slightly different criteria and terminology. The main difference is that the SCOP classification is done largely manually. Other mentionable databases that contain more detailed information are e.g. the Protein Data Bank (PDB) and UniProt, but they do not have a hierarchical classification system as the previously mentioned databases do.

The entries in CATH are taken from PDB. The PDB protein structures are first cut into individual domains and only afterwards hierarchically classified, so CATH does not contain entire proteins, but rather single protein domains with an annotation to which protein(s) they belong.

The first four levels are called Class, Architecture, Topology and Homologous superfamily and their initials form the name CATH. The definition of these four levels is the following:

> “
> - Class - structures are classified according to their secondary structure composition (mostly alpha, mostly beta, mixed alpha/beta or few secondary structures).
> - Architecture - structures are classified according to their overall shape as determined by the orientations of the secondary structures in 3D space but ignoring the connectivity between them.
> - Topology (fold family) - structures are grouped into fold groups at this level depending on both the overall shape and connectivity of the secondary structures.
> - Homologous superfamily - this level groups together protein domains which are thought to share a common ancestor [...].
>
> *from [20]*
>
> ”

CATH (Version 4.0, based on the PDB release from 01/01/2015) contains approximately 235.000 solved protein domain structures. CATH separately classifies sequentially identical domains that originate e.g. from domain repeats in proteins or in homooligomers. Furthermore, evolutionarily fully conserved domains that occur in different proteins are classified independently in CATH, as well as multiple structures of the same protein. We use a dataset that avoids redundancy and hence contains 63.737 sequentially unique domains. These are spread out over all of the four Classes, all of the 40 Architectures, a subset of 1375 Topologies and a subset of 2728 Homologous superfamilies. We will refer to this dataset as the **CATH dataset** in the rest of this thesis. Further characteristics of the dataset are shown in Figure 3.2. It was necessary to set an upper limit for the domain length to avoid difficulties with GPU memory size. Considering Figure 3.2a, a length of 600 amino acids was chosen as the limit. A lower limit is also required, but it depends on the specific architecture of the CNN. It will be evaluated in section 6.2.

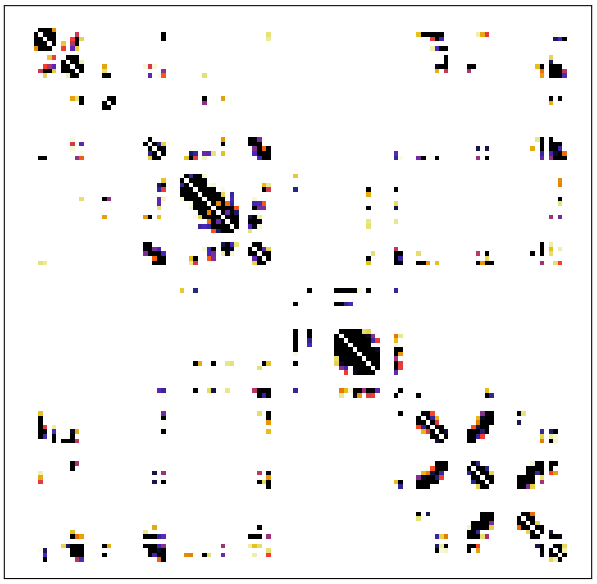


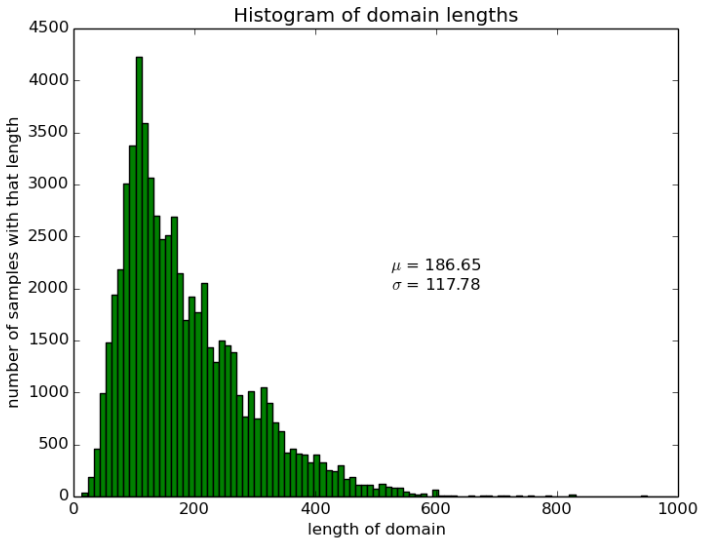
Figure 3.1: The contact map of Putative dihydrolipoamide dehydrogenase from Colwellia psychrerythraea 34H (PDB-ID: 3IC9). The color map ranges from white for $c_{i,j} = 0$ to black for $c_{i,j} = 1$. The horizontal and vertical blank areas are loop regions which have been zeroed.

3.2 Preprocessing CATH data

The contact maps of the domains were created with the algorithm explained in section 1.2. As the domains in CATH are fully solved protein structures, the contact maps also contain information about the contacts in loop sections. The output data of BCL::Fold does not contain any information about loop regions, that is to say that in the contact maps, any row and column belonging to a loop region is left blank. In order to train on a realistic dataset, the loop regions of the CATH contact maps are zeroed when loading the data (see Figure 3.1). This concerns all residues labeled as C, T or S by DSSP.

3.3 Generation of protein conformations with BCL

Structure decoys is a term for possible conformations of proteins created by a protein structure prediction algorithm. So BCL::Fold creates a set of structure decoys for an unsolved protein structure. Most of these structures roughly correspond to already known tertiary structures. To simulate such a set of decoys, a dataset of perturbed models of known structures is used. The perturbed tertiary structure models of domains are created using the BCL application BCL::perturb. It takes the atomic coordinates of solved and classified domains and applies small modifications like rotations to some residues, still working with idealized angles in the backbone. The obtained models do not contain information about the structure of loops. The dataset contains 30358 perturbed models of CATH classes 1, 2 and 3 that cover 487 CATH topologies. We will refer to this dataset as the **perturbation dataset** in the rest of this thesis.



(a) Histogram of domain lengths in the CATH dataset

(b) Number of domains per class (T-level) in the CATH dataset

Figure 3.2: CATH dataset

4 Fold prediction

4.1 The deep network

4.1.1 Basic characteristics of the network

A lot of different problem-specific architectures have been presented in the literature. However, some techniques have been identified to work well for a wide range of tasks, so this work also makes use of them. A state-of-the-art network that won the ILSVRC 2014 classification task was presented by Simonyan and Zisserman [8]. Its main characteristics, which are also used in this work, are the following:

- **3x3 convolutions** as universal filter size. If a bigger receptive field is desired, 3x3 convolutional layers can be stacked upon each other, leading to faster calculation and an increased number of non-linear transformations compared to larger filter sizes. A convolutional layer[1] performs a mapping

$$f : \mathbb{R}^{M \times N \times K} \to \mathbb{R}^{M \times N \times K'}, \qquad \mathbf{x} \mapsto \mathbf{y} \tag{4.1}$$

  with $\mathbf{x}$ being a real-valued array of $M \times N$ pixels and K channels [21]. The output of the convolutional layer is

$$y_{ijk'} = g(b_{k'} + \sum_{\hat{i}\hat{j}k} w_{\hat{i}\hat{j}kk'} x_{i+\hat{i},j+\hat{j},k}). \tag{4.2}$$

  where $\mathbf{W} = (w_{ijkk'})_{ijkk'}$ is a weight array and $\mathbf{b}$ the biases. The weights $\mathbf{W}$ and the biases $\mathbf{b}$ are the parameters of the mapping in Equation 4.1 learned by the backpropagation algorithm. Note that the filters are three-dimensional, in the sense that they operate on a map $\mathbf{x}$ with K channels. Furthermore, there are K' such filters, generating a map $\mathbf{y}$ with K' channels. The sum of the convolution and the bias term are passed to a non-linear activation function

$$g : \mathbb{R}^{M \times N \times K'} \to \mathbb{R}^{M \times N \times K'}, \qquad \mathbf{x} \mapsto \mathbf{y}, \tag{4.3}$$

  which is the rectified linear unit (ReLU) in our case, defined element-wise as

$$y_{ijk} = \max\{0, x_{ijk}\}. \tag{4.4}$$

[1] We assume implicit padding with zeros such that input and output have the same size (apart from channels).

- **2x2 max-poolings** with stride $= 2$ in order to spatially group features and decrease the dimensionality of the feature space, therefore substantially shortening training time and hard-wiring invariances to certain spatial transformations. A pooling operates on a single feature channel, merging spatially close features into one. Max-pooling is defined as

$$y_{ijk} = \max\{y_{i'j'k} : i \leq i' < i + p, \quad j \leq j' < j + p\}, \tag{4.5}$$

  where p is the size of the pooling filter. The value stride $= 2$ means that only even-valued i,j,k are calculated and passed on.

- **Fully connected (FC) layers** after the convolutional layers allow a combination of all extracted features. A fully connected layer is a function

$$h : \mathbb{R}^K \to \mathbb{R}^{K'}, \qquad \mathbf{x} \mapsto \mathbf{y}, \tag{4.6}$$

  where K is the number of neurons in the previous layer, K' the number of neurons in the current layer. The output of the new layer is

$$y_{k'} = g(b_{k'} + \sum_k w_{k'k} x_k) = g(\mathbf{b} + \mathbf{W}\mathbf{x}), \tag{4.7}$$

  where $\mathbf{W}$ are the weights, $\mathbf{b}$ the biases and g is a non-linear activation function (ReLU in hidden layers).

- The output layer is a FC layer with **softmax** nonlinearity with the number of neurons equal to the C possible classes of the classification task. The softmax output of a neuron i is defined as

$$\varphi(\mathbf{x})_i = \frac{e^{\mathbf{a}_i}}{\sum_{c=1}^{C} e^{\mathbf{a}_c}}, \tag{4.8}$$

  where $a = \mathbf{b} + \mathbf{W}\mathbf{x}$ is the neuron activation before applying the nonlinearity. The softmax output values can be interpreted as a "probability estimate" because

$$\sum_i (\varphi(\mathbf{x}))_i = 1. \tag{4.9}$$

  For classification, the softmax outputs are transformed to a sparse vector of equal size, where the element at the position of the maximum in the softmax output vector is set to 1 and all other elements are set to 0. Alternatively, the classes of the N highest values in the softmax output can be defined as a set of TOPN predictions, with their softmax outputs as a weighting.

The complete objective function that is to be minimized with backpropagation is $E(\{\mathbf{W}_l\}_l, \{\mathbf{b}_l\}_l)$, where l enumerates the layers. It is a composition of the above presented functions with a

so-called loss function as the last function of the composition (see subsection 4.2.1). The weights of each layer are initialized as an orthogonal matrix as proposed in [22]. For controlling the weight updates of error backpropagation we use a popular adaptive learning rate method called *Adam*, introduced in 2014 by Kingma and Ba [23]. It estimates the moments of the gradients with exponential moving average. The principal reason for selecting Adam is its robustness in many applications. Finally, we choose *Dropout* [24] with dropout rate $p = 0.5$ as a regularization technique to avoid overfitting.

4.1.2 Handling input of variable size

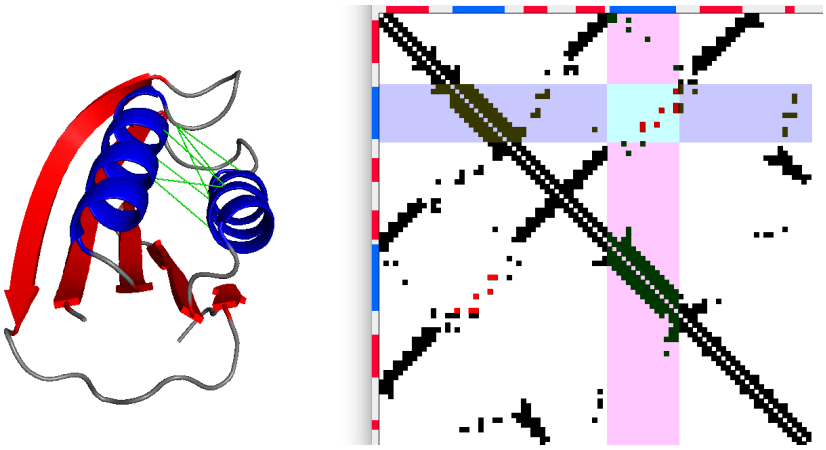
As described in section 3.1, the range of protein lengths is large. This leads to input matrices of very different sizes, but they all have to be handled by the same neural network and also lead to meaningful weight updates in the training process. What would be difficult to realize with a fully connected network is easy to do with a CNN. The filters of the convolutional layers are shifted over the whole size of the input space independent of its size. The filters of the max-pooling layers are shifted over the entire input space as well. The transition from a variable-size feature space to a softmax output with fixed size is realized by a global pooling layer. This reduces the feature map of size $M \times N \times K$ to size $1 \times 1 \times K$, hence each feature channel is transformed to a scalar (see subsection 4.1.3). So the number of features becomes independent of the input size. These features in turn are subsequently passed on to fully connected layers and finally to a softmax nonlinearity that leads to a representation suitable for the classification task.

4.1.3 Multiple global pooling functions

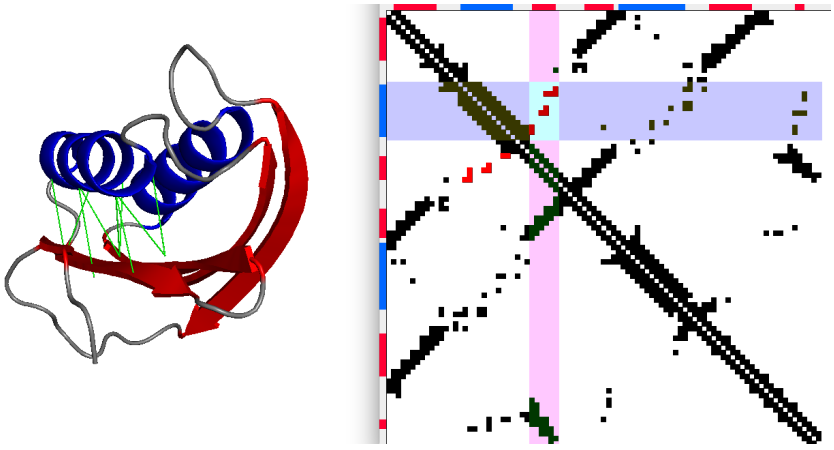
Global pooling can be done with different pooling functions such as max, sum or mean. We chose to use both max-pooling and sum-pooling in our networks. We apply both poolings to the same input space and forward the concatenated results to the next layer of the network. Max-pooling is the the most common global pooling that is used in literature. It extracts the largest value of each channel. Sum-pooling is particularly useful if the network is designated to count certain recurring features. In our case, e.g. counting helices and strands is an indicator for the predicted fold. The counts greatly diminish the set of possible folds for a sequence, as all possible folds that do not exhibit the number of helices and strands extracted by the network can be immediately ruled out. Other more sophisticated elements that can be counted in contact maps among others are different types of SSE-SSE contacts, possibly split up by their length and their arrangement.

4.1.4 One-dimensional global pooling

A characteristic property of CNNs is that the detection of features is based on local neighborhoods. In the shallow layers of the networks, where low-level features are being detected, only



(a) (left) The visualization of acylphosphatase (PDB-ID: 2HLT) with two helices (blue) and five strands (red). SSE-SSE contacts are shown as green lines.
(right) The corresponding contact map. The contacts in the intersection of the two wide bars form a sparse cross-diagonal indicating the existence of an antiparallel helix-helix contact.



(b) (left) The visualization of acylphosphatase (different angle as in Figure 4.1a).
(right) The corresponding contact map. The contacts in the intersection of the two wide bars again form a sparse cross-diagonal indicating the existence of an antiparallel helix-strand contact.

Figure 4.1: Difference between helix-helix and helix-strand contacts in contact maps

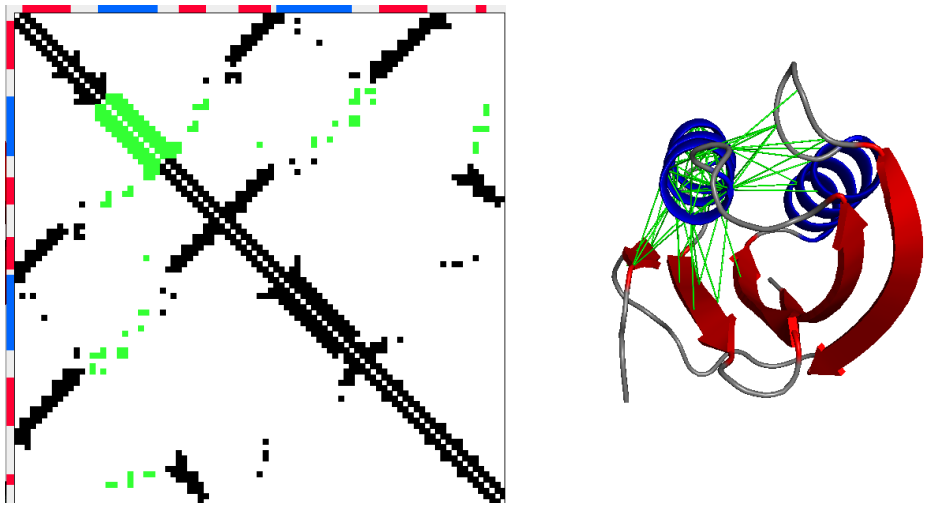


Figure 4.2: (left) The contact map of acylphosphatase (PDB-ID: 2HLT). A group of amino acids that form a helix was chosen, with all its contacts highlighted in green. This helix has three SSE-SSE contacts, to two β-strands and another helix. This would be a distributed feature which can be detected by 1D global pooling.
(right) The corresponding visualization with two helices (blue), five strands (red) and the observed contacts (green).

features that are spatially close can be discovered. Only in deeper layers where the receptive window is containing bigger parts of the initial input image can high-level features that are defined by multiple spatially distributed areas be detected. We will call such features *distributed features*. To detect these distributed features we propose one-dimensional global pooling (see Figure 4.2). Both one-dimensional sum-pooling and one-dimensional max-pooling are used and their results are concatenated. After that pooling, multiple one-dimensional convolutional layers are applied to the feature space. This step assures that the network is able to learn features distributed over entire columns of data, which it would not be able to do with the common two-dimensional convolutional architecture.

4.1.5 Multi-branch network

We want to include the one-dimensional global pooling into a standard CNN as described in subsection 4.1.1. However, we also want to keep using the features from the two-dimensional feature space to do classification. The approach chosen for this is a CNN with multiple independent branches (see Figure 6.2). In the beginning, a common track applies several convolutional layers to the input space. After that, the network is split into two branches: one that starts with a global one-dimensional pooling followed by numerous one-dimensional convolutions

and poolings and one with the standard two-dimensional CNN architecture. The two branches are merged together again after each of them has gone through a global pooling. Thereafter, they are passed on to a fully connected layer. We tried different numbers of layers before the network gets split up, the results can be found in section 6.2.

4.2 The training procedure

4.2.1 Three-level classification loss

The CATH class hierarchy has four main levels (see section 1.2). A classification task could be carried out on either of these four levels, depending on the desired precision. The required classification levels in this work are C, A and T, so the three lower levels of CATH. The homologous superfamily (H) is not of importance for this task as we want to group conformations by structure, not by common ancestors. Furthermore, the number of different homologous superfamilies in CATH is only about twice the number of topologies, so on average there are two superfamilies per topology. Therefore, including the H-level would not lead to a much finer classification.

Instead of training a separate network for each task, a single network can be used by splitting it up after the second last layer. Each final layer consists of the respective number of neurons for its classification level C, A and T. The loss is calculated as a linear combination of each final layer loss. At the moment all levels are weighted equally.

As it is common practice for softmax classification tasks, the categorical cross-entropy was chosen as the loss function for our CNN in each of the three final layers. For targets $\mathbf{t}$ and predictions $\mathbf{p}$ of one sample it is defined as

$$L = -\sum_i t_i \log(p_i). \tag{4.10}$$

4.2.2 Priority Queue

The CATH database with its pronounced class imbalance is far from an ideal training set for a neural network. A method to improve the training procedure is not using simple iterative training, but instead online batch selection. The goal of this approach is to stronger weigh the small classes in the training loss sum. For our approach, we use a priority queue out of which the m (mini-batch size) first elements form the current mini-batch that the network is trained on. The priority ω of each sample is initialized randomly, but it is assured that each sample is used once before the prioritizing starts to take effect. After training on the mini-batch, each sample has to be reinserted into the priority queue with an updated priority. The obvious choice is to base the new priority upon the loss value. However, the used categorical cross-entropy loss (see Equation 4.10) only considers the softmax value of the ground truth class. Only the

summand of the ground truth class g with $t_g = 1$ influences the loss, because $t_c = 0 \quad \forall c \neq g$. Instead of the loss, here we use the rank r of the ground truth class within the softmax-vector because we want the correct output to be within the top few predictions. This is combined with a measurement of the temporal progress of the training procedure to make sure each sample is used from time to time. Instead of directly relying on run time which is prone to many unforeseen influences, using the overall number of already trained samples s was chosen. As a result, the priority ω is computed as

$$\omega = r \cdot f(s) \tag{4.11}$$

with f being a parameter function to set the influence of the temporal progress on the new priority.

4.3 Programming environment and used tools

The whole project was implemented in Python. The main reason for that is the use of the Deep Learning library *Lasagne* [25]. *Lasagne* is an open-source lightweight library initiated in 2014 to build and train deep neural networks in *Theano* [26]. *Theano* in turn is an open-source Python library that allows the user to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It has been mainly developed and maintained by the machine learning institute at the Université de Montréal since 2007. It supports compiling identical code on either CPU or GPU, with the latter being substantially faster. In order to run *Theano* code on an Nvidia GPU [2] it is converted to CUDA code using Nvidia's neural network library cuDNN. The structure of *Theano* and therefore *Lasagne* as well is similar to dataflow graphs, which are created to represent mathematical expressions. The elements of a graph are Variable nodes, which are usually arrays, and Apply nodes, which define mathematical operations to be executed upon said arrays. During compilation, the graphs are first optimized, second CUDA or C++ code is generated and compiled and finally a callable object is returned. The training of the neural networks was done on a Nvidia GeForce GTX Titan X GPU with 12 GB of memory.

[2] Support of arbitrary OpenCL GPU is still in beta stage

5 Fold clustering

With the fold prediction method presented in chapter 4, clustering with predefined cluster properties can be done. The properties of each cluster will correspond to the properties of the predicted CATH topology for each sample. There are several downsides to this, such as a non-variable number of clusters and classifying possibly newly discovered folds as already known folds. There are regular clustering approaches that are more flexible and do not have these disadvantages. We will consider them in sections 5.1-5.3.

Clustering can be based on either a pairwise similarity matrix between samples or on a feature vector for each sample. In the latter approach, these features form a multi-dimensional space where each sample can clearly be associated to one particular point in that space. Clusters are formed by samples that are in some way close in feature space, depending on how the clustering algorithm is defined.

5.1 Clustering using deep features from classification

The network used for the classification task classifies based on features that are extracted from the original data. These features are learned during training and therefore represent characteristic properties of the contact maps. They are generally more adapted to perform clustering than features that were chosen manually based on data analysis and prior knowledge. Global descriptors based on deep features are used in state-of-the-art image retrieval approaches [27].

To use the features that are extracted by the supervised deep network, a few final layers of the trained network can be chopped off and the values of the output neurons can be used as features for a clustering algorithm. However, this cutting-off usually results in a lot of output neurons and therefore too many features to be handled by a regular clustering algorithm. A dimensionality reduction, e.g. with Principal Component Analysis (PCA) is the obvious choice. After the dimensionality reduction, a regular clustering algorithm such as k-means++ [28] or DBSCAN [29] can be applied.

5.2 Clustering using deep autoencoder features

Features representing the input data even better can be extracted by a deep autoencoder. The concept of an autoencoder has been introduced in [30]. It is a feedforward neural network

with a bottleneck after half of its layers. The second half is usually symmetrical to the first one and is designed to reconstruct the original input data from the features in the bottleneck. This assures that the features in the bottleneck suffer only a minimal information loss compared to the original representation. The training loss of the network is calculated by comparing the input neurons with the output neurons, e.g. with MSE. The updates of the weights are carried out with backpropagation as in any feedforward neural network. The features extracted from the bottleneck are a compact high-level representation of the data and, as in section 5.1, can be passed on to a common clustering algorithm.

5.3 Active clustering with learned similarity metric

The possibility of clustering with a pairwise similarity matrix without feature-space coordinates was already mentioned above. A function to compare images can be learned by a CNN as proposed in [31]. CNN architectures that are most suitable to learn a similarity function incorporate some form of input combination or parallelism. In [31], a 2-channel architecture is found to work best to compare image patches of a fixed size. Two image patches are joint together as if they were two channels and are fed to the first layer of a CNN. If working with variable size images, *siamese networks* are the state of the art for a comparison task [32].

Computing the entire similarity matrix is of complexity $\mathcal{O}(n^2)$, which can be very computationally expensive for large datasets or complex distance metrics. So-called *active clustering* based on an incomplete similarity matrix has been an intensively researched area, with one of the first approaches presented in 1998 [33]. Many papers elaborate on *active spectral clustering*, as *spectral clustering* is a successful and stable method for clustering with a similarity matrix. A recent example of spectral clustering, which works with uncertainty about similarities and could be adapted to clustering protein decoys based on their contact maps, can be found in [34].

5.4 Clustering quality loss function

The goal of this work is to achieve a problem-specific appropriate clustering of the protein models. A metric that measures the goodness of clustering can be directly used as a loss function instead of the current categorical cross-entropy loss to train a CNN. This will lead to the extraction of features that are particularly important for the clustering quality.

6 Experiments and results

Each of the networks is trained with 80% of the samples of the CATH dataset, while the remaining 20% are used as the test set. Splitting is not consistent in these experiments. The number of epochs is set to 20 as no improvements of the loss of the test set (see Figure 6.1) could be observed with the chosen initial learning rate of 10^{-5}. The aforementioned configurations are used in all further experiments unless stated otherwise.

6.1 The effect of branches with 1D global pooling

We compare a network with a 1D global pooling branch (1DGP) to a network with identical architecture except for the missing 1D global pooling branch.The details of the network architecture with 1DGP are shown in Figure 6.2. The results of both networks are shown in table Table 6.1. A pronounced difference between the two networks is not notable.

	with 1DGP	without 1DGP
training time per sample	0.069 s	0.060 s
ground truth in TOP10 predictions on T-level	96.4%	96.8%
ground truth in TOP1 predictions on T-level	90.4%	90.9%

Table 6.1: Performance of a network with 1D global pooling compared to a similar network without the 1D global pooling, on the test set.

6.2 Comparing different architectures

Based on the concepts presented in section 4.1, we compare three different architectures. They all share the same configurations for the final segment of the network, which ranges from the layer that merges all features of the different branches until the output layer. The distinct characteristics are:

- **Network A**: 6 convolutional layers, 3 max-pooling layers in the 2D branch. One additional branch with 1D global pooling (see Figure 6.2).

- **Network B**: 8 convolutional layers, 4 max-pooling layers in the 2D branch. One additional branch with 1D global pooling (see Figure 6.3).
- **Network C**: 6 convolutional layers, 3 max-pooling layers in the 2D branch. Two additional branches with 1D global pooling that fork at different layers (see Figure 6.4).

The performance of the three different architectures is shown in Table 6.2.

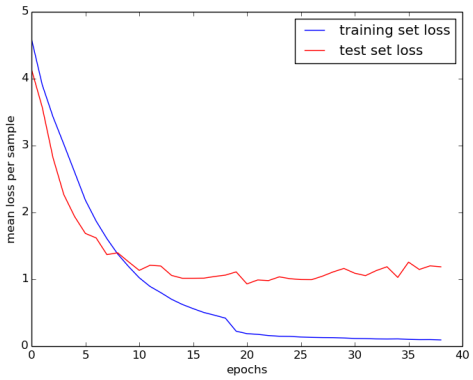


Figure 6.1: Mean loss of a network with 1DGP for training and test set (network A). The learning rate was multiplied by 0.1 after 20 epochs. This led to a notable decrease of the training loss, but not of the test loss. This indicates that the test loss has already converged after 20 epochs or even earlier.

	Network A	Network B	Network C
training time per sample	0.069 s	0.079 s	0.076 s
ground truth in TOP10 predictions on T-level	96.4%	97.0%	96.0%
ground truth in TOP1 predictions on T-level	90.4%	92.7%	90.4%

Table 6.2: Performance of different network architectures on the test set. Network B outperforms the other two proposed architectures in TOP10 predictions.

Implementation details: A lower limit for the length of the domains was chosen due to difficulties with the max-pooling layers. In default mode of the used CNN library *Lasagne* (see section 4.3), our 2×2 max-pooling layers do not work on an input of size 1×1, which we can obtain depending on input size and the number of max-pooling layers. This current constraint can probably be overcome by setting the option `ignore_border=True` in the MaxPool2DLayer. Note that the currently excluded models pose less than 0.5% of the dataset even for our largest network C with 5 max-pooling layers.

<table>
<tr><td colspan="6">input layer</td><td>200×200×1</td></tr>
<tr><td colspan="6">conv 3×3 - 64</td><td>200×200×64</td></tr>
<tr><td colspan="6">maxpool 2×2</td><td>100×100×64</td></tr>
<tr><td colspan="6">conv 3×3 - 128</td><td>100×100×128</td></tr>
<tr><td colspan="6">conv 3×3 - 256</td><td>100×100×256</td></tr>
<tr><td>1D-global-maxpool</td><td>100×1×256</td><td>1D-global-sumpool</td><td>100×1×256</td><td colspan="2">maxpool 2×2</td><td>50×50×256</td></tr>
<tr><td colspan="3">merge</td><td>100×1×512</td><td colspan="2">conv 3×3 - 256</td><td>50×50×256</td></tr>
<tr><td colspan="3">1D-conv3 - 256</td><td>100×1×256</td><td colspan="2">maxpool 2×2</td><td>25×25×256</td></tr>
<tr><td colspan="3">1D-conv3 - 256</td><td>100×1×256</td><td colspan="2">conv 3×3 - 512</td><td>25×25×512</td></tr>
<tr><td colspan="3">1D-conv3 - 256</td><td>100×1×256</td><td colspan="2" rowspan="4">conv 3×3 - 512</td><td rowspan="4">25×25×512</td></tr>
<tr><td colspan="3">1D-maxpool2</td><td>50×1×256</td></tr>
<tr><td colspan="3">1D-conv3 - 256</td><td>50×1×256</td></tr>
<tr><td colspan="3">1D-conv3 - 256</td><td>50×1×256</td></tr>
<tr><td>global-maxpool</td><td>1×1×256</td><td>global-sumpool</td><td>1×1×256</td><td>global-maxpool</td><td>1×1×512</td><td>global-sumpool</td><td>1×1×512</td></tr>
<tr><td colspan="8">merge - 1536</td></tr>
<tr><td colspan="8">fully connected - 4096</td></tr>
<tr><td colspan="8">fully connected - 2048</td></tr>
<tr><td colspan="2">softmax - 4</td><td colspan="2">softmax - 40</td><td colspan="4">softmax - 1375</td></tr>
<tr><td colspan="2">C-level</td><td colspan="2">A-level</td><td colspan="4">T-level</td></tr>
</table>

Figure 6.2: Architecture of network A, with one 1D-branch. The array size for an input domain of typical size 200×200 is depicted in blue for each layer.

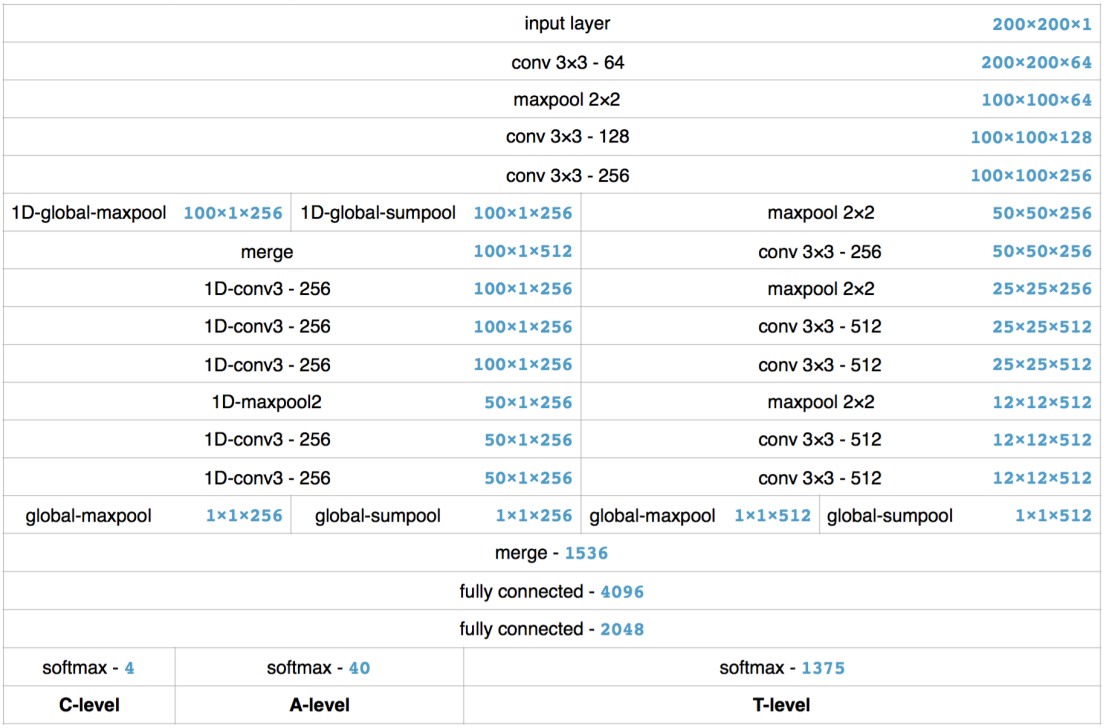


Figure 6.3: Architecture of network B, with one 1D-branch.

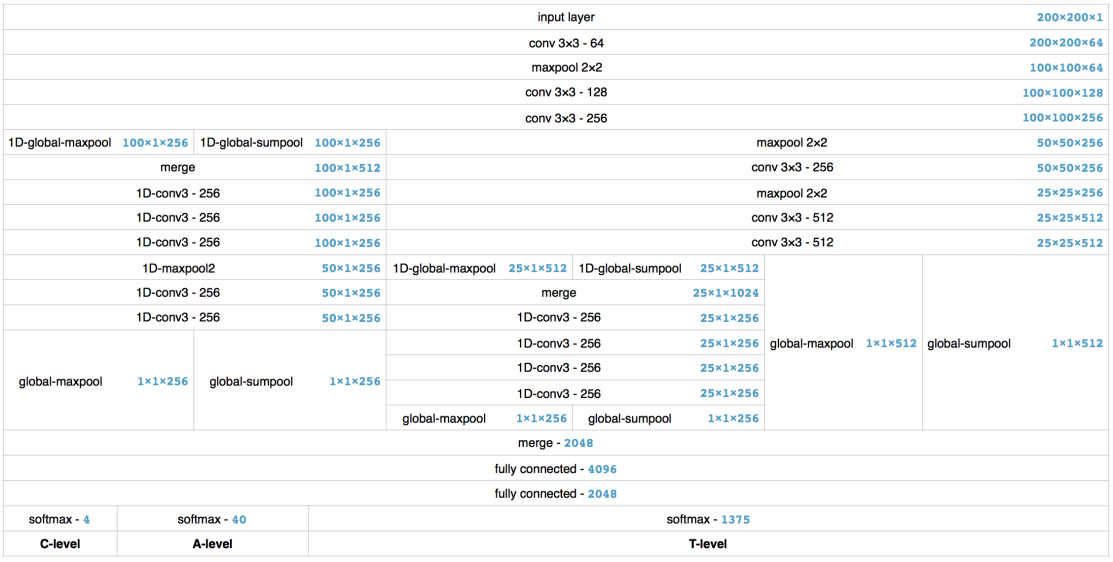


Figure 6.4: Architecture of network C, with two 1D-branches.

6.3 Validation of the multi-level classification loss

6.3.1 Consistency between the levels

To validate that the designed three-level loss and that the associated three final layers lead to feasible results, it can be investigated whether the levels predict consistent CATH classes. As an example, it would be a bad result if the network predicted class 2 at the C-level, but class 3.10 at the A-level. Two predictions, one of a coarse classification level h and one of a fine classification level l, are considered consistent if

$$\sum_i (s_{l,c,i}) - s_{h,c} < \alpha \tag{6.1}$$

where $s_{h,c}$ is the softmax value for one class c of the coarse level, $s_{l,c,i}$ is the softmax value of the fine level that belongs to the specified class c and α is a custom threshold. Consistencies of predictions between different CATH levels are evaluated, namely C-to-A consistency, C-to-T consistency and A-to-T consistency. The results are shown in Table 6.3. If we consider all softmax outputs of an output vector, a lot of them are usually close to zero and are considered as consistent with the respective target output, even if the value of the ground truth class is corrupted. Only observing the consistency of the ground truth class leads to more meaningful results and on level A-T, to a drop from 98.7% to 81.3% consistency. The joint distribution of the predictions of pairs of levels is shown in Figure 6.5. It considers all possible output classes for each sample which explains why there is a strong concentration of samples at both $(0, 0)$ and $(1, 1)$ in each of the joint histograms. In Figure 6.5a, each sample leads to four entries as there are four distinct possible classes (C-level). In Figure 6.5b respectively, there are 40 possible architectures (A-level) and therefore 40 entries per sample.

Inconsistent predictions that are far away from the diagonal are very few compared to the consistent predictions (Table 6.3).

In Figure 6.6 only the predictions for the ground truth classes of the respective coarse level are depicted, which leads to the desired concentration around $(1, 1)$ and a few false negatives at $(0, 0)$.

	C-A	C-T	A-T
ground truth	0.948	0.899	0.813
all classes	0.981	0.958	0.987

Table 6.3: Three-level loss consistencies of the test set as defined in Equation 6.1 with $\alpha = 0.02$

6.3.2 Correlation between the levels

Another metric to measure the performance of the three-level classification loss is the Pearson correlation coefficient ϱ. It is a measure of the linear correlation between two variables X and Y, in our case the class predictions of two levels. Its value is between 1 and -1 inclusive, where 1 means total positive correlation, 0 means no correlation and -1 means total negative correlation. It is defined as

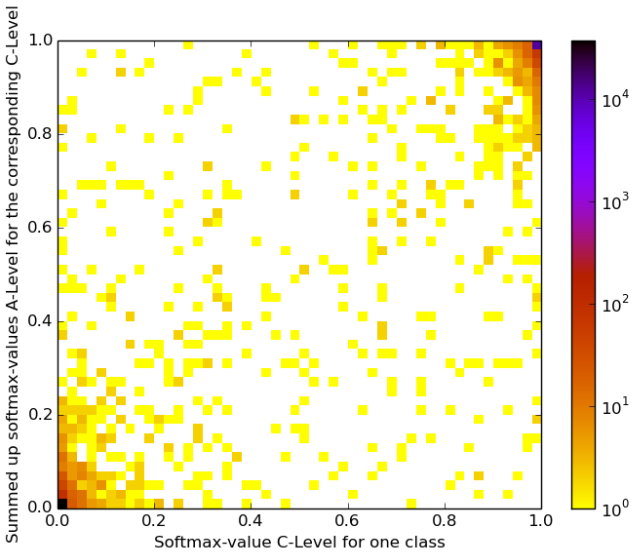
$$\varrho(x,y) := \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.2}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the sample means.

Table 6.4 indicates strong correlation between all three classification levels. However, if only the outputs of the ground truth classes of the coarse level are considered, the correlation between C-level and T-level is remarkably lower.

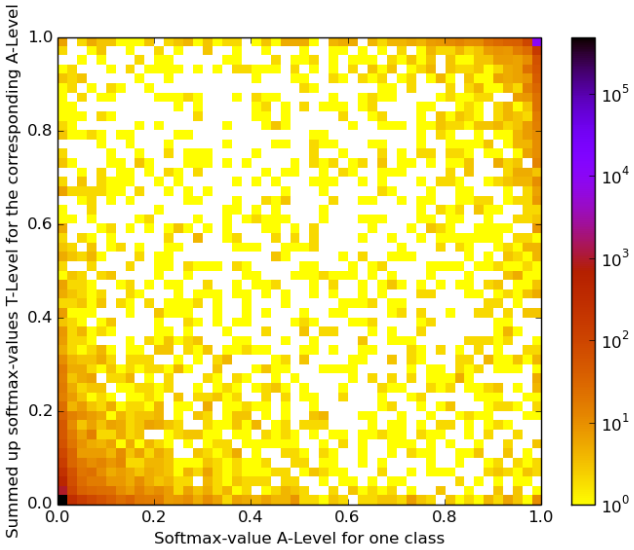
	C-A	C-T	A-T
ground truth	0.916	0.784	0.853
all classes	0.997	0.992	0.983

Table 6.4: Three-level classification correlations ϱ of the test set



$$\varrho = 0.997$$

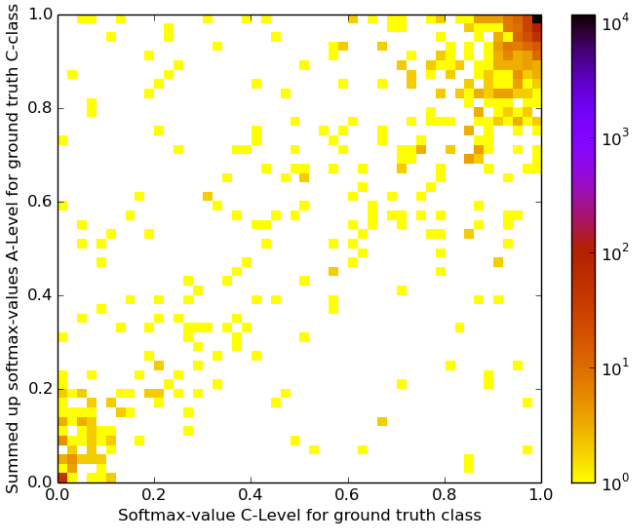
(a) Joint histogram of predictions at C-level and A-level (both mapped to C-level).



$$\varrho = 0.983$$

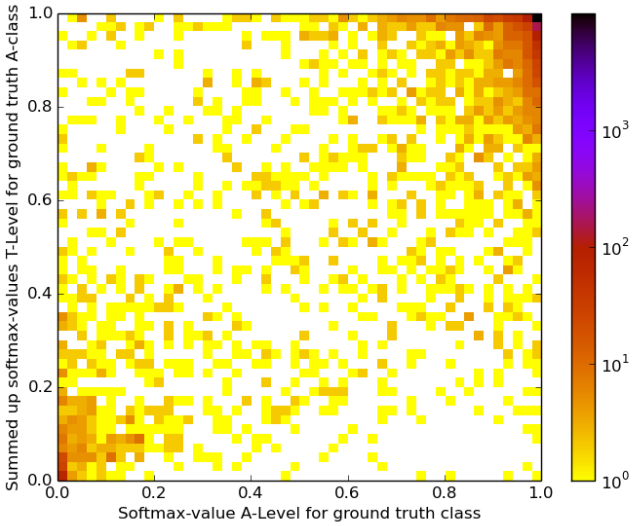
(b) Joint histogram of predictions at A-level and T-level (both mapped to A-level).

Figure 6.5: Classification consistency of the test set (∼12500 samples)



$$\varrho = 0.916$$

(a) Joint histogram of predictions for ground truth C-class at C-level and A-level (both mapped to C-level).



$$\varrho = 0.853$$

(b) Joint histogram of predictions for ground truth A-class (architecture) at A-level and T-level (both mapped to A-level).

Figure 6.6: Ground truth classification consistency of the test set (∼12500 samples)

6.3.3 Comparison of different loss definitions

Using a multi-level classification method can not only be useful for this task, but can be applied to a wide range of classification problems. In this section we want to evaluate how different setups of loss calculations affect the quality of the classifications on different levels. The results are displayed in Table 6.5. The values in parenthesis were not calculated directly from the output of the network as this is not possible in some cases. Instead, as an example, if the network is only trained on A-level, the softmax loss for C-Level does not exist. For all approaches to be comparable, we infer losses on coarser levels than the one being trained on by summing up the losses of the finer level according to the CATH hierarchy (see Figure 6.7).

Architecture	φ("1.10") = 0.5	φ("1.20")=0.2	φ("1.25")=0.1	φ("2.10")=0.05	φ("2.30")=0.1	φ("2.40")=0.05
Class	⟹ φ("1")=0.8			⟹ φ("2")=0.2		

Figure 6.7: Example of interpreting the softmax loss φ of the A-level for the C-level

Table 6.5 clearly shows that the proposed three-level classification loss improves the performance of the CNN. Interestingly, optimization of fine-level loss even outperforms optimization of coarse-level loss in terms of coarse-level loss, i.e. beats coarse-level loss at its own game. This is the case for both C-level and A-level as coarse levels.

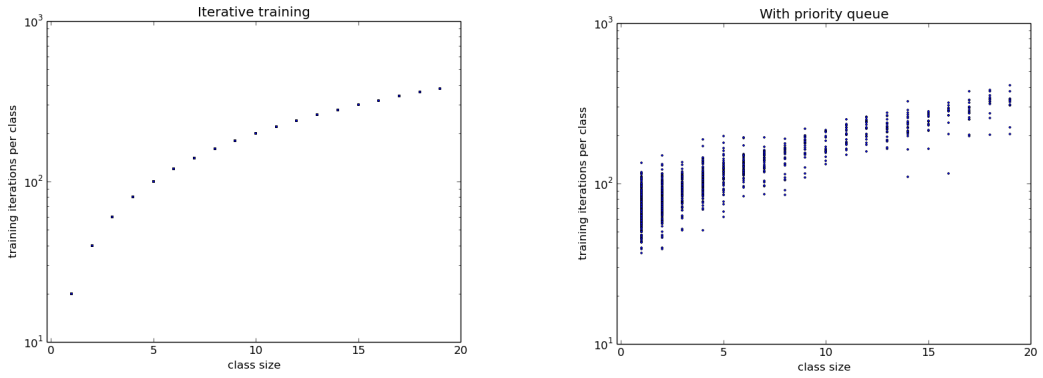
	C mean loss	A mean loss	T mean loss
C	*0.104*	-	-
A	(0.105)	*0.437*	-
T	(**0.053**)	(0.231)	*0.786*
C,A	0.086	0.452	-
A,T	(0.057)	0.210	0.648
C,A,T	0.054	**0.182**	**0.616**

Table 6.5: Mean softmax loss per sample on C-, A- and T-level in the test set (columns) for different loss configurations in training (rows). The results of the standard single losses for each level are italicized. Inferred losses are shown in parenthesis.

Implementation details: The loss interpretation of fine level losses as coarse level losses as shown in Figure 6.7 led to numerical problems in our implementation. For an unknown reason, some softmax outputs were found to be zero. This can lead to $\log(0) = -\infty$ in the categorical cross-entropy loss function. As we sum the losses of the finer level, we multiply some losses with zero, leading to $-\infty \cdot 0$ which by lack of definition leads to a NaN value (not a number). To avoid this problem, we add the float value with minimal exponent to each softmax output to assure that $\log(0)$ is never calculated.

6.4 Validation of the priority queue

The parameter function f from Equation 4.11 is set to the identity function. We compare training with iterative batches and training with the proposed priority queue on network A. The desired effect of samples in small classes being trained more often is shown in Figure 6.8. Note that the classes with 1 to 5 samples form about half of the 1375 classes (T-level) of the CATH dataset (see Figure 3.2b), i.e. the use of priority queue affects a substantial part of the CATH hierarchy (but of course only a small subset of samples). Furthermore, the use of a priority queue also aims to improve the rank of the ground truth class in the softmax outputs for each sample. The obtained improvement is shown in Figure 6.9. The function f in Equation 4.11 determines the impact



(a) Each class is visited proportionally according to its size, e.g. classes of size 1 receive 20 training iterations, classes of size 2 receive 40 iterations etc.

(b) Variable number of training iterations for classes of equal size depending on the performance of each individual sample.

Figure 6.8: Comparison of training iterations with iterative batches vs. online batch selection with priority queue (20 training epochs), sorted by class size for classes up to size 20. Note that the small classes of size 5 or less clearly receive more training iterations.

of the temporal progress of the training procedure on the assignment of a new priority. As the balancing effect of the used identity function was still not leading to an evenly distributed training over all CATH topologies, we tried to weaken the influence of temporal progress. The goal was to train samples with a low-ranked ground truth more often, as a correlation of small classes and poorly performing classes was detected. Two promising approaches that were tried were $f(x) = \log(x)$ and $f(x) = \sqrt{x}$. However, both approaches led to overfitting of small classes. The quantitative results are a lot worse than the results that are based on a linear f and are therefore not presented in this work.

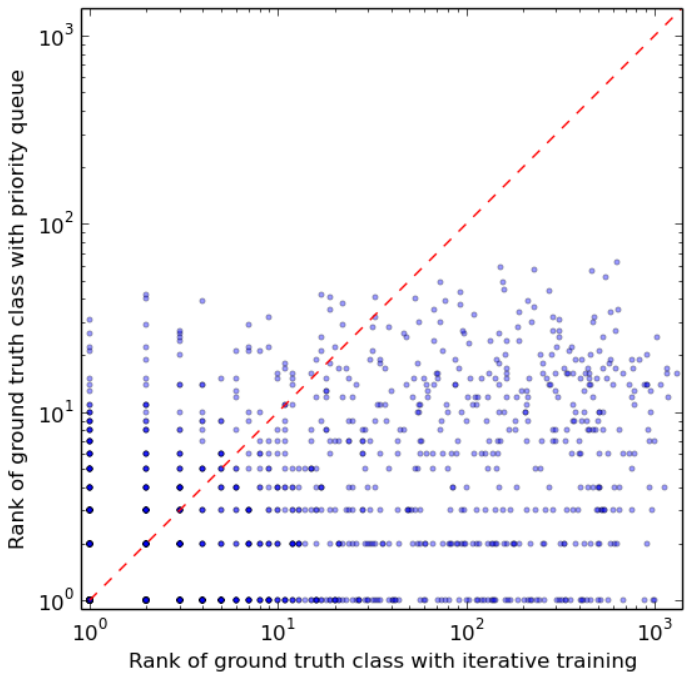


Figure 6.9: Comparison of the rank of the ground truth class in the softmax outputs for training with iterative batches (x-axis) versus training with priority queue (y-axis) for each sample. All samples below the red line represent an improvement if the priority queue is used (the best rank is 1). The color intensity of the entries is proportional to the number of samples at the respective coordinates. However, the use of the priority queue does not considerably improve the employed TOP1 and TOP10 metrics on the test set due to the class imbalance of the dataset. The improved classes are mostly small and thus have only a small influence on the overall TOP10 and TOP1 performance.

6.5 Comparing the use of distance matrices versus contact maps

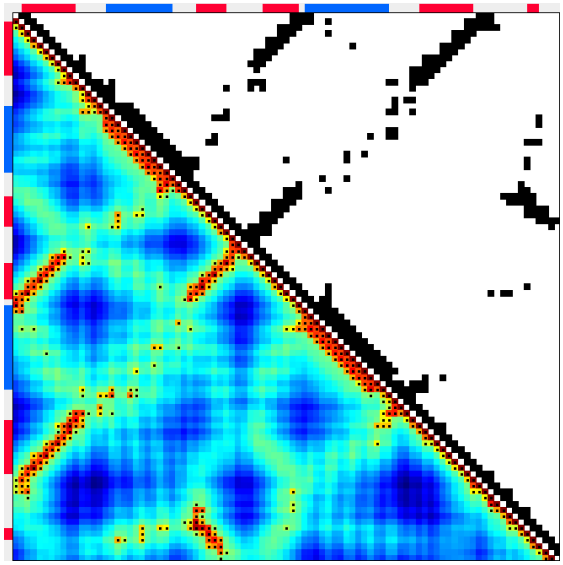


Figure 6.10: Comparison of the distance matrix (lower triangle) and the contact map (upper triangle) of acylphosphatase (PDB-ID: 2HLT). In the distance matrix, the color scale ranges from red for close amino acids to blue for distant amino acids. While the contact map contains a lot of uniformly blank areas, the distance matrix has shading in these same areas. The distance matrix contains more information about the arrangement of distant SSEs.

With the algorithm presented in section 1.2 it is also possible to create a pairwise distance matrix (DM) for the residues. It has the same dimension as the corresponding contact map (CM), but contains more information about the protein (see Figure 6.10). It is possible for the deep network to learn how to infer spatial arrangement of SSEs that are far away from each other and do not have physical contacts. A difficulty is that as CATH only classifies single domains, some distance matrices (and also the contact maps) contain large empty sections. The reason is that a domain is a spatially separated unit of a protein but the amino acids do not need to be sequentially connected. For example, if the domain contains amino acids 1-10 and 60-100, all $\hat{m}_{i,j}$ in its distance matrix (and also in its contact map) with $10 < i < 50$ or $10 < j < 50$ are empty (by convention zero). Whereas in contact maps, these zeros do not pose a problem, a zero in a distance matrix would mean that the two residues are spatially equal. The approach for solving this problem is creating a second input channel, namely a mask with binary values that indicates which rows and columns are a valid part of the distance matrix and which are not. This mask has the same dimension as the distance matrix and is used as a second input

channel along with the distance matrix.

We compare the performance of using distance matrices and using contact maps as inputs for the CNN. Moreover, we also combine both CMs, DMs and the mask to a total of three channels. The results are shown in Table 6.6. Both the individual use of distance matrices and the combination with contact maps raise the TOP1 predictions by about 2%, while the TOP10 predictions slightly improve as well. The training time per sample stays roughly the same, as adding input channels has only an effect on the calculations of the first layer.

	CM	DM	CM & DM
ground truth in TOP10 predictions on T-level	96.4%	97.3%	97.2%
ground truth in TOP1 predictions on T-level	90.4%	92.6%	92.1%

Table 6.6: Performance comparison of different input data (on network A).

6.6 Classifying perturbed CATH models

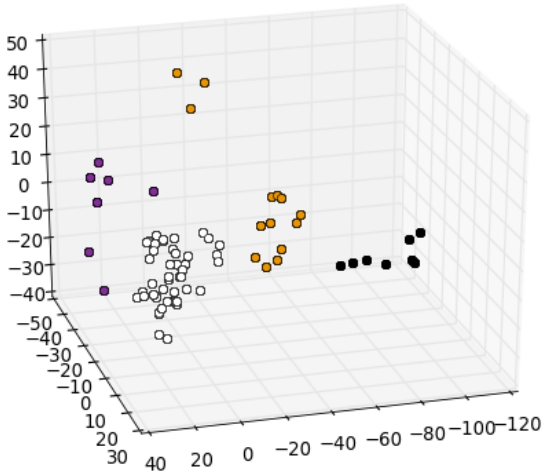
The perturbation dataset (see section 3.3) shall simulate outputs of the BCL::Fold algorithm. In order to test the performance of the developed approach, the entire CATH dataset is used as a training set for network A. The trained network is used to do fold prediction on the perturbation dataset. As each conformation in the perturbation dataset is similar to the CATH conformation it originates from, the dataset is labeled and can therefore be used to evaluate the performance of our classification approach. The results are shown in Table 6.7. The prediction accuracy for the ground truth topology being in the TOP10 predictions with the perturbation dataset (Table 6.7, right column) is comparable to the prediction accuracy of a test set that consists of unperturbed contact maps (see section 6.2). However, the accuracy for predicting the TOP1 topology is lower (by about 4%).

	training	fold prediction
ground truth in TOP1 predictions on C-level	99.5%	97.6%
ground truth in TOP1 predictions on A-level	97.9%	92.7%
ground truth in TOP1 predictions on T-level	93.6%	85.9%
ground truth in TOP10 predictions on T-level	98.3%	96.1%

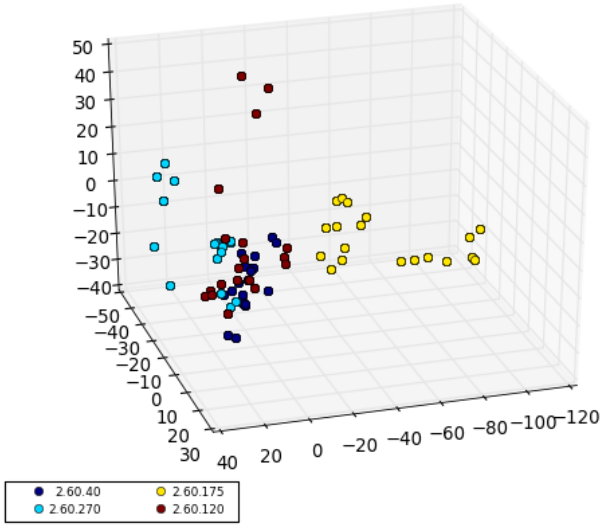
Table 6.7: Results for training on the CATH dataset (left column) and fold prediction of the perturbation dataset (right column).

6.7 Clustering with deep features, PCA and k-means++

We evaluate the clustering approach presented in section 5.1 on a small exemplary dataset of 400 perturbed CATH domains, with 100 samples each from topologies 2.60.40, 2.60.120, 2.60.175 and 2.60.270. PCA is applied to reduce the 1536 deep features of *network A* (see section 6.2) to a three-dimensional space in order to visualize the results. The chosen clustering algorithm is k-means++ with $k = 4$. The results are shown in Figure 6.11. Although some parts of clusters are detected correctly, others are mixed up in the 3D deep feature space and therefore k-means++ is not able to clearly separate the clusters. Increasing the number of components of the PCA to as far as 50 does not improve the results.



(a) Clustering results for k-means++, $k = 4$, on deep features reduced to 3D space by PCA



(b) Ground truth data for Figure 6.11a.

Figure 6.11: Comparison of clustering results with ground truth data

7 Conclusions and future work

7.1 Performance of the CNN-based classification

Using distance matrices instead of contact maps raises the TOP1 prediction accuracy by 2% because distance matrices contain more information about the protein. Out of the three final network architectures, network B was found to achieve the best results for prediction with contact maps of the native (unperturbed) CATH dataset, namely 97% ground-truth-in-TOP10 classification accuracy and 93% TOP1 predictions on T-level. In our final experiment on the artificial dataset of perturbed CATH models, we reach a ground-truth-in-TOP10 classification accuracy of 96% and are able to predict the TOP1 topology for 86% of the test set, using network A.

The priority queue achieves a more class-balanced training, but is not able to entirely overcome the huge class imbalance of the CATH dataset, as training the samples of small classes even more frequently led to overfitting.

The three-level classification loss on levels C, A and T performs better in terms of individual level losses than any other loss that is based on a subset of the three. The most outstanding result is reached for A-level mean loss: While with single A-level classification it amounts to 0.44, it is down to 0.18 with three-level classification. This means that multi-level loss outperforms single-level loss in terms of single-level loss, i.e. beats it at its own game. This result may have far-reaching consequences for classification methods.

Both the priority queue and the multi-level loss classification clearly improve the performance of the CNN. However, the reached ground-truth-in-TOP10 and TOP1 accuracies do not reflect that. In case of the priority queue, this is due to the reason that the affected number of classes is high, but those are only the smallest classes and therefore only few samples are improved. For the multi-level classification loss the reason is that the current loss function (categorical cross-entropy loss) does not consider the rank of the ground truth class in the softmax output vector, which we quantify with the TOP10 and TOP1 metrics. By definition it only considers the output value of the ground truth class. If the ground-truth-in-TOP10 percentage is defined to be the most important in the future, a loss function based upon it should be used and it would most probably improve the classification accuracy.

7.2 Next steps

Although the supervised classification approach has performed well on a simulated dataset, a validation on real *ab initio* models produced by BCL::Fold has not been done yet. As the output conformations of BCL::Fold do not have ground truth CATH-labels, this would require evaluating the performance of the whole pipeline for BCL::Fold-based protein folding prediction with our new approach replacing the current one. An easier profound evaluation could possibly be done using CATH models that are more severely perturbed. To further improve the performance of the classification, the class imbalance of the training set should be diminished. This can be done by adding perturbed models to small classes of the training set.

7.3 Future work on fold clustering

Our unsupervised clustering approach still produces poor results at this stage. It can be further developed in the future to obtain a competitive unsupervised clustering. Unsupervised learning approaches can overcome the constraint of the CNN-based supervised clustering, namely the predefined cluster characteristics given by the CATH domains. The three additional unsupervised clustering methods that are proposed in this work, a deep autoencoder, active clustering with a learned similarity metric, and using a custom clustering metric as loss function of a CNN, can be validated in future work.

List of Figures

List of Tables

Bibliography

[1] A. Lesk, *Introduction to protein architecture: The structural biology of proteins*. Oxford University Press, 2001, ISBN: 9780198504740.

[2] I. Sillitoe, T. Lewis, A. Cuff, S. Das, P. Ashford, N. Dawson, N. Furnham, R. Laskowski, D. Lee, J. Lees, S. Lehtinen, R. Studer, J. Thornton, and C. Orengo. (2016). CATH Domain 1pgaA00, [Online]. Available: `http://www.cathdb.info/version/latest/domain/1pgaA00/structure` (visited on 08/17/2016).

[3] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, vol. 22, no. 12, pp. 2577–2637, 1983, ISSN: 1097-0282. DOI: `10.1002/bip.360221211`.

[4] C. Vehlow, H. Stehr, M. Winkelmann, J. M. Duarte, L. Petzold, J. Dinse, and M. Lappe. (2016). CMView - Protein contact map visualization and analysis, [Online]. Available: `http://www.bioinformatics.org/cmview/tutorial.html` (visited on 08/17/2016).

[5] A. Lesk, *Introduction to protein science: Architecture, function, and genomics*. Oxford University Press, 2016, ISBN: 9780198716846.

[6] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," PhD thesis, Harvard University, Cambridge, MA, 1974.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[9] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, "Deep learning for computational biology," *Molecular systems biology*, vol. 12, no. 7, 2016. DOI: `10.15252/msb.20156651`. eprint: `http://msb.embopress.org/content/12/7/878.full.pdf`.

[10] A. W. Fischer, S. Heinze, D. K. Putnam, B. Li, J. C. Pino, Y. Xia, C. F. Lopez, and J. Meiler, "CASP11 – an evaluation of a modular BCL::Fold-based protein structure prediction pipeline," *PLoS ONE*, vol. 11, no. 4, pp. 1–19, Apr. 2016. DOI: 10.1371/journal.pone.0152517.

[11] M. Karakaş, N. Woetzel, R. Staritzbichler, N. Alexander, B. E. Weiner, and J. Meiler, "BCL::Fold - *de novo* prediction of complex and large protein topologies by assembly of secondary structure elements," *PLoS ONE*, vol. 7, no. 11, pp. 1–20, Nov. 2012. DOI: 10.1371/journal.pone.0049240.

[12] N. Woetzel, M. Karakaş, R. Staritzbichler, R. Müller, B. E. Weiner, and J. Meiler, "BCL::Score - knowledge based energy potentials for ranking protein models represented by idealized secondary structure elements," *PLoS ONE*, vol. 7, no. 11, pp. 1–17, Nov. 2012. DOI: 10.1371/journal.pone.0049242.

[13] A. Leaver-Fay, M. Tyka, S. M. Lewis, O. F. Lange, J. Thompson, R. Jacak, K. W. Kaufman, P. D. Renfrew, C. A. Smith, W. Sheffler, I. W. Davis, S. Cooper, A. Treuille, D. J. Mandell, F. Richter, Y.-E. A. Ban, S. J. Fleishman, J. E. Corn, D. E. Kim, S. Lyskov, M. Berrondo, S. Mentzer, Z. Popović, J. J. Havranek, J. Karanicolas, R. Das, J. Meiler, T. Kortemme, J. J. Gray, B. Kuhlman, D. Baker, and P. Bradley, "Chapter nineteen - Rosetta3: An object-oriented software suite for the simulation and design of macromolecules," in *Computer methods, part C*, ser. Methods in Enzymology, M. L. Johnson and L. Brand, Eds., vol. 487, Academic Press, 2011, pp. 545–574. DOI: http://dx.doi.org/10.1016/B978-0-12-381270-4.00019-6.

[14] I. Sillitoe, T. E. Lewis, A. Cuff, S. Das, P. Ashford, N. L. Dawson, N. Furnham, R. A. Laskowski, D. Lee, J. G. Lees, S. Lehtinen, R. A. Studer, J. Thornton, and C. A. Orengo, "CATH: Comprehensive structural and functional annotations for genome sequences," *Nucleic Acids Research*, vol. 43, no. D1, pp. D376–D381, 2015. DOI: 10.1093/nar/gku947. eprint: http://nar.oxfordjournals.org/content/43/D1/D376.full.pdf+html.

[15] N. Alexander, N. Woetzel, and J. Meiler, "Bcl::Cluster: A method for clustering biological molecules coupled with visualization in the pymol molecular graphics system," in *Computational Advances in Bio and Medical Sciences (ICCABS), 2011 IEEE 1st International Conference on*, Feb. 2011, pp. 13–18. DOI: 10.1109/ICCABS.2011.5729867.

[16] N. Gupta, N. Mangal, and S. Biswas, "Evolution and similarity evaluation of protein structures in contact map space," *Proteins: Structure, function, and bioinformatics*, vol. 59, no. 2, pp. 196–204, 2005, ISSN: 1097-0134. DOI: 10.1002/prot.20415.

[17] J. J. Gray, S. Moughon, C. Wang, O. Schueler-Furman, B. Kuhlman, C. A. Rohl, and D. Baker, "Protein–protein docking with simultaneous optimization of rigid-body displacement

and side-chain conformations," *Journal of molecular biology*, vol. 331, no. 1, pp. 281–299, 2003.

[18] G. Bouvier, N. Duclert-Savatier, N. Desdouits, D. Meziane-Cherif, A. Blondel, P. Courvalin, M. Nilges, and T. E. Malliavin, "Functional motions modulating VanA ligand binding unraveled by self-organizing maps," *Journal of chemical information and modeling*, vol. 54, no. 1, pp. 289–301, 2014.

[19] T. Kohonen and P. Somervuo, "Self-organizing maps of symbol strings," *Neurocomputing*, vol. 21, no. 1, pp. 19–30, 1998.

[20] I. Sillitoe, T. Lewis, A. Cuff, S. Das, P. Ashford, N. Dawson, N. Furnham, R. Laskowski, D. Lee, J. Lees, S. Lehtinen, R. Studer, J. Thornton, and C. Orengo. (2016). What is CATH? [Online]. Available: `http://www.cathdb.info/wiki/doku/?id=faq` (visited on 08/20/2016).

[21] A. Vedaldi and A. Zisserman. (2016). VGG convolutional neural networks practical, [Online]. Available: `http://www.robots.ox.ac.uk/~vgg/practicals/cnn/#part1-1` (visited on 09/04/2016).

[22] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *CoRR*, vol. abs/1312.6120, 2013.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] E. Battenberg, S. Dieleman, D. Nouri, E. Olson, A. van den Oord, C. Raffel, J. Schlüter, and S. K. Sønderby. (2016). Lasagne - lightweight library to build and train neural networks in theano, [Online]. Available: `https://github.com/Lasagne/Lasagne` (visited on 08/18/2016).

[26] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *Arxiv e-prints*, vol. abs/1605.02688, May 2016.

[27] A. Babenko and V. S. Lempitsky, "Aggregating deep convolutional features for image retrieval," *CoRR*, vol. abs/1510.07493, 2015.

[28] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.

[30] Y. Le Cun, "Modèles connexionnistes de l'apprentissage," PhD thesis, Paris 6, 1987.

[31] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4353–4361.

[32] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[33] T. Hofmann and J. M. Buhmann, "Active data clustering," *Advances in neural information processing systems*, pp. 528–534, 1998.

[34] F. L. Wauthier, N. Jojic, and M. I. Jordan, "Active spectral clustering via iterative uncertainty reduction," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2012, pp. 1339–1347.