Uniwersytet Jagielloński
Wydział Matematyki i Informatyki
INSTYTUT INFORMATYKI i MATEMATYKI KOMPUTEROWEJ
Studia stacjonarne

Nr indeksu: 1064077

Agnieszka Pocha

Convolutional Neural Networks for Drug Design

Opiekun pracy magisterskiej:
dr hab. Igor Podolak

Kraków 2015

Abstract

The goal of this work is to build a model that would classify ligands as active or inactive based on their fingerprints. Convolutional neural network are used. Two approaches are described. Two new (?) methods to include unlabeled samples in the learning procedure are presented.

Contents

Chapter 1

Introduction

There is a constant need for novel drugs to be designed, either because new diseases are found, old drugs proove to be ineffective, there are patents on current drugs, *etc.* This might be a lengthy and very costly process, even if high-throughput methods of *in vitro* (using cultures of cells) screening are used [8]. An alternative is the *virtual screening* (frequently called *in silico* screening, *i.e.* screening conducted within a computer chip). The approach is to check, using computer methods, if a small chemical molecule (a ligand) is *active* with a given protein, *i.e.* it either activates or prohibits its functioning, that is connected with some disease. Molecules found to be particularly likely to be active are then passed to wet-laboratory tests. This forms a pipeline filtering out most of possible molecules (the chemical space size is in the range of 10^{60}!) leaving just a few. In usual applications starting from a million of molecules, the virtual screening (together with some toxicity checking and drug-likeliness expert's methods) ends up in less than 10 molecules to test in a laboratory (*e.g.* [11]).

The virtual screening might be done in, basically, two ways [10, 9]

- target- (or structure- or protein-) based screening when the structure of a protein is known and methods of *docking* might be used to predict the activity,
- ligand-based screening if protein structure is *not* known, but there is a knowledge about some molecules that are active; then the task is to find molecules similar to those active (following a Johnson-Maggiora rule that similar compounds should have similar properties [9]).

1.1 The goal of this paper

Virtual screening involves checking great amounts of molecules in order to find those which have interesting features. In this paper we describe how convolutional neural networks can be used to approach this problem. Our goal was to

build a model that would and we proposed two approaches to tackle this problem. Moreover, we present new approach to make use of unlabeled examples during the learning process of convolutional neural networks.

1.2 Related work

During last few years research was conducted in the area on incorporating machine learning methods to address the problem of virtual screening [23, 24, 27]. Such methods as support vector machines [25], binary kernel disctimination [26] or neural networks [28] have been successfully used to address this problem. In this paper we show that using convolutional neural networks is also a promising approach.

1.3 Problem

The task to be solved here is a kind of target-based virtual screening. Protein-molecule interactions are given along with labels describing which of these pairs result in activation. The data is obtained with a docking approach.

First the molecules are docked in a protein and then the molecules' pharmacophore features are calculated. Based on the docking adjustment the score is calculated for each molecule. If the score is above a certain threshold, the complex is classified as active. There is also another threshold, below which the complex is classified as inactive. If the score for a molecule is in between these two thresholds, the molecule is classified as middle (or unclassified). The machine learning task is to predict which molecules activate (or prohibit) the protein.

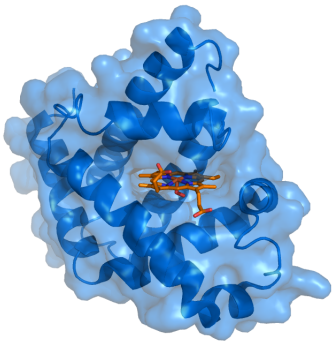

Figure 1.1: A molecule (blue) with a docked ligand(orange) 1.1.

Proteins are molecules built from *amino acid residues* forming a single *chain*. Some proteins have less than 100 residues while others might have even few thousands of them. The order of amino acids in the chain and their spatial arrangement carry a lot of information.

Our goal was to build a model that would classify the complex as *active* or *inactive* based on its *fingerprint*. Fingerprints are a widely used representation of molecules. There are many types of fingerprints designed. They vary in length and the type of information they carry. Some of them are designed for specific tasks. (citation needed). Conventionally, fingerprints represent a molecule as a vector. Each element of this vector describes whether a specific structure is present in the molecule or not, e.g. whether the molecule has a hydrophobic group or not.

Vectors are commonly used in machine learning as data representation because they can be stored in matrices which allows easy calculations. Even though representing a protein as a vector means loosing a lot of information about its topology, it enables effective computation and still provides us with reasonable results.

As already said, there are many different fingerprints designed. They vary in length and the features included. In this paper we used 2D-SIFt [7] fingerprints. We used convolutional neural networks because they are eligible to work with data which topological order carries information.

1.4 2D-SIFt fingerprints

In this work the 2D-SIFt fingerprints are used [12, **?**]. They are a composition of Structural Interaction Fingerprints (SIFt) [13] together with the pharmacophore model. SIFts code information from the point of view of the receptor conveying *how* a molecule interacts with the protein, not *what* interacts. A single block codes information in a nine element long binary vector:

$$[\text{any contact}, \text{backbone}, \text{sidechain}, \text{polar}, \text{hydrophobic},$$
$$\text{H-bond acceptor}, \text{H-bond donor}, \text{aromatic}, \text{charged}].$$

These are informations about the single points of contact between the target protein and the molecule.

Additionaly, there is a pharmacophore model of the molecule added. A pharmacophore is a set of sferic and electronic features that is necessary to ensure the optimal supramolecular binding with a protein (or another specific biological target structure) and to trigger (or to block) its biological response [16].

One can see pharmacophore features as a way to perform the virtual screening in a half way between protein- and ligand-based approach. The docking needs to be performed for plenty but not all of the molecules. At first, the pharmacophore features of the docked molecules are computed. Afterwards, a big data set of molecules with known pharmacophore features can be screened in order to find molecules most similar to those which were docked and classified as active. The protein itself is used only at the beginning during the process of docking. Later it might be dismissed as the whole method relies on pharmacophore features.

In 2D-SIFt each pharmacophore point is described using a six element vector:

$$[\text{H-bond donor}, \text{H-bond acceptor}, \text{hydrophobic}, \\ \text{negatively charged}, \text{positively charged}, \text{aromatic}].$$

These two models combined result in a 6×9 matrix for each possible point of contact. The model shall now be called 2D-SIFt [12]. For a given protein and given molecule this results in a string of n of 6×9 arrays glued together, where n describes the number of residues of a given protein. We can say, that this representation resembles and image, on which patterns shall be looked after. This explains why a convolutional network is used in this work.

1.5 How this paper is organised

At first, the convolutional neural networks are described. We present such concepts as convolution or pooling. The properties of convolutional neural networks are given.

Chapter 3 is the most important part of the paper. In this chapter we present the two approaches that we used to address the problem. We describe in detail the data on which we performed experiments and the preprocessing that we used to extract most interesting features from the data. A detailed explanation of the architecture that we used is given. We describe in detail both training methods that we created to tackle the problem of using unlabeled samples in the learning algorithm - the naive approach and the fancy approach.

Our results are described in the next chapter. Finally, in the last chapter we discuss possible ways to further develop our approach. At the end of this paper there is a short appendix which includes some less important concepts used in this work.

Chapter 2

Convolutional Neural Networks

In this chapter the Convolutional Neural Network (CNN) model shall be described.

The definition of a convolutional neural network is: a neural network that takes advantage of using the convolution operation. CNN might be seen as a network consisting of two parts - first part is the convolutional part and it is responsible for extracting the features from the data. The second part might be a softmax layer responsible for classifying samples based on features provided by the convolutional part. This schema is shown on figure 2.1.

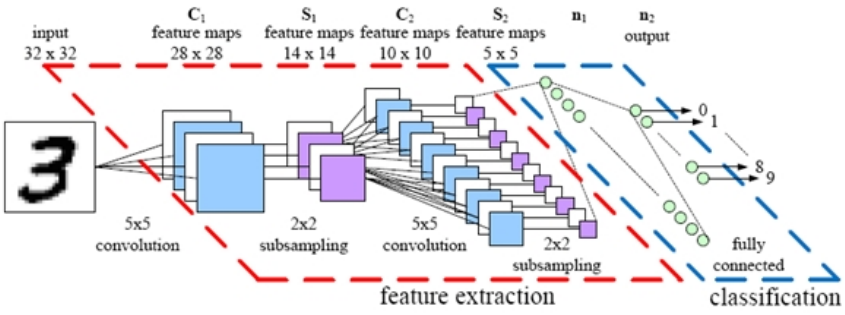

Figure 2.1: An example of convolutional neural network [17].

In this section we will show what motivation stands behind using convolutional neural networks, explain what is the convolution operation, and what is pooling. Types of activation functions that might be used in convolutional neural networks will be presented as well as the learning algorithm.

2.1 Motivation

Convolutional neural networks take advantage of data which topological order carries some information. Therefore, CNNs are often applied to image recognition or natural language processing problems and achieve state-of-the-art results [14, 15]. Since the the topological order of the data we are working on carries information (due to how the proteins are built), we expect that using convolutional neural networks might bring good results.

2.2 Computation Flow

As stated above, CNNs can be conceptually divided into two subnetworks. In this subsection it will be described how the data is processed within the convolutional subnetwork. Not much attention will be put to classifying subnetwork as there is a wide variety of possible approaches.

There are three elemental operations that are performed in each layer of the convolutional subnetwork. At first, the input is convoluted with a convolution matrix. The result of this operation becomes an input to the activation function, which output becomes the input to the pooling function. The output of the pooling function becomes the input to the next layer which might be another convolutional layer and the whole process will begin again. This process is schematically depicted on figure 2.2 and it can be also described by the following formula:

$$output = p(\sigma(c(input))),$$

where c is the convolution function, σ is the activation function, and p is the pooling function.

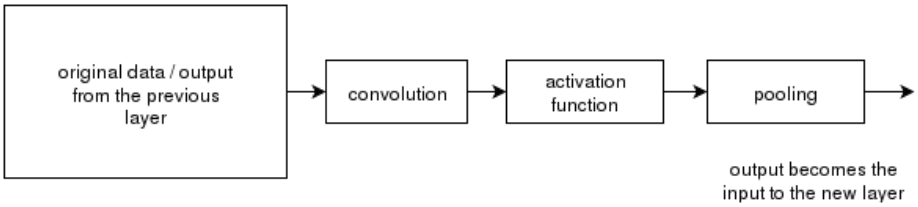

Figure 2.2: The three elemental operations performed in each convolutional layer.

One might also imagine three consecutive seperate layers: a convolutional layer, a classical layer that applies activation function and finally a pooling layer.

Each of these operations will be described in details in the following subsections.

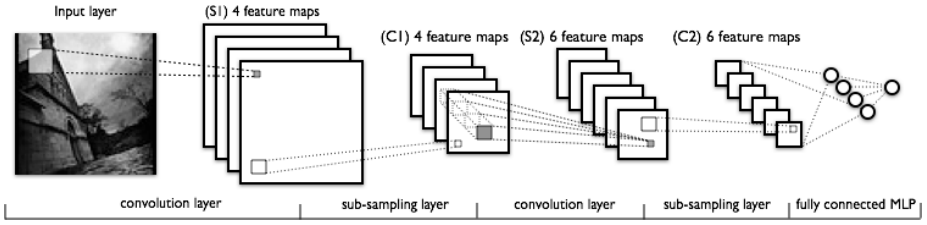

Figure 2.3: An example of a whole neural network with stages of convolution and polling marked [19].

2.2.1 Convolution

Convolution operation takes as operands two functions and returns a new function as a result. Mathematically, convolution is defined as:

$$c(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx,$$

where c is a function returned by convolution operation and t is a point in which function c is evaluated. c is defined as an integral over two other functions: f is often called the input, while g is often reffered to as a weighting function.

Convolution might be seen as a measure how well two functions fit. Function g is a pattern that moves along function f and it each place it measures how well the two functions fit. If the similarity is high, the returned value is high as well.

Convolution for neural networks

It is worth considering how this equation can be applied to neural networks. Since representing real values in the computers is rather troublesome, it is desirable to express this equation in a discrete form, which is shown below:

$$\overline{c}(t) = \sum_{x=-\infty}^{\infty} \overline{f}(x)\overline{g}(t-x)$$

If we want to apply this equation to neural networks we might want to think of $\overline{f}$ as a data sample. In such approach $\overline{g}$ would become a matrix that is moving around the data sample producing a single value in each place. From now on, we will call f an input, g a filter or a convolution matrix or a convolution window and c an output.

Usually, the convolution window is much smaller then the input and convolution is appplied multiple times. Each time a submatrix of input is multiplied by the convolution window. The result of this operation is a scalar, and a result of a whole process is a matrix. Each layer of neural network might include multiple filters and thus each layer might produce an output of higher dimensionality than the provided input - the additional dimension is produced due to using multiple filters.

It is worth noting, that there are different filters in each layer. Filters in the next layer work on the features extracted from the previous layers. Therefore,

the later the layer is in the neural network, the more complicated patterns it may discover.

Size and stride, spatial invariance

It is worth taking a closer glance at how convolution works in details. First, we have to define two important parameters of the convolution windows - their size and stride. The size is simply the size of the convolution matrix. The stride defines which part of the input will be convoluted next with respect to the part of the input that is currently being convoluted. This concept is shown in figure 2.4.

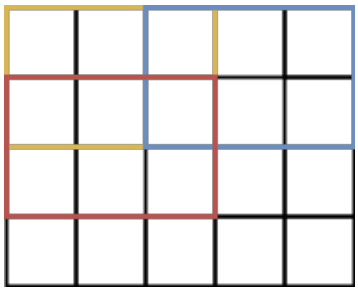

Figure 2.4: The figure shows three consecutive positions of a convolution window - yellow, blue and red. The input is of size 4×5, the convolution window has size 2×3 and stride 1×2, which means the window can move two elements to the right or one to the bottom.

Three consecutive positions of a convolution window are depicted. Each position defines which submatrix of the input will be multiplied by the convolution matrix. Each multiplication will produce a single value - these values when combined will produce an output matrix.

It should be noted that each time the convolution matrix is the same, only the part of the input that is being convoluted changes. Each filter is specialised in finding a specific pattern - because it is convoluted with many submatrices of the input it will find that pattern regardless of where the patter is on the input matrix. This property is called spatial invariance. Figure 2.5 shows this concept more clearly.

As stated above, in each layer usually there are plenty of convolution matrices - each is specialised in finding a specific pattern. Each pattern might be seen as a useful feature of the data and convolution provides us with some sort of map that shows where in the input this feature exists and where it does not. Therefore, it is often useful to see convolution windows as filters or feature extractors.

Data size reduction due to convolution operation

Let the input be an $I \times J$ matrix and the convolution filter a $K \times L$ matrix with a 1×1 stride. Therefore, the first submatrix to multiply by the convolution

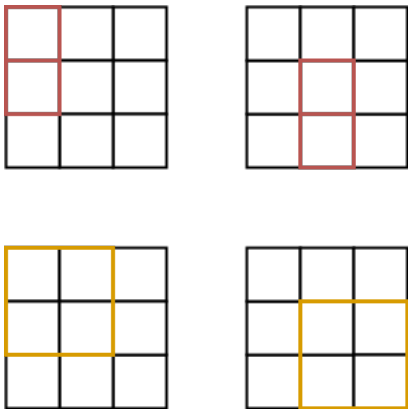

Figure 2.5: In the upper row two locations of a certain feature are shown in red. In the bottom row the location of a convolution window that will discover this feature is shown in yellow. As can be observed, by moving the convolution window it is possible to extract a certain feature regardless of its location.

filter will be $[1:K] \times [1:L]$ and it will return a single value. The last submatrix will be $[I-K+1:I] \times [J-L+1:J]$ and it will also produce a single value. As a result the output will be a $[I-K+1] \times [J-L+1]$ matrix. This process is shown on picture 2.6.

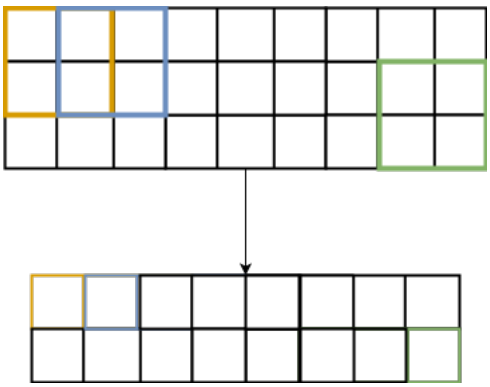

Figure 2.6: In this picture $I = 3, J = 9, K$ and $L = 2$. First submatrix and the output it produces are shown in yellow, second are shown in blue and the last are shown in green. The dimensionality reduction might be observed.

It can be observed that the values laying close to the edge of the input matrix will be underrepresented. The upper left corner of the input matrix will have influence on only one element of the output (the yellow one) while its right neighbour will already have influence on two elements of the output (the yellow and the blue one). The elements in the middle of the input matrix will influence four elements of the output. Such unequalities of the influence might be undesirable. There is a variety of ways to address this problem, e.g. zero-padding, but we will not cover them. More information about such methods can be found in [6].

2.2.2 Activation function

There are few possible activation functions that might be used in convolution neural networks [1].

- the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$
- the hyperbolic tangent activation function $tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- the rectifier linear function [2] $rect(x) = max(0, x)$

2.2.3 Pooling

Pooling is an operation, usually applied to a matrix, that takes as an input multiple values and returns a single value describing the input. Typical pooling functions are [1]:

- max pooling - the max value of input is returned
- average pooling - the average value of input is returned
- stochastic max pooling - one element is chosen from the input to become the result. Probability of choosing an element is proportional to its value. [5]

Pooling is defined not only by its type but also by its size and stride. The size of pooling defines how many values will be taken as an input - the bigger the size of pooling, the more information is accumulated in a single value. The pooling stride defines where will be the next submatrix with respect to its present location. Fig 2.7 illustrates this concept.

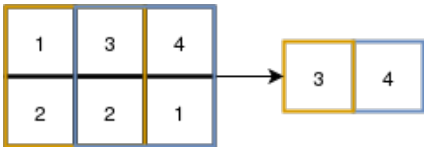

Figure 2.7: The result of applying the max pooling of size 2×2 with stride 1×1 to the input. Yellow square is the first pooling window, blue square is the second one. The values of the output are coloured accordingly.

Thanks to pooling the network is not affected by small changes of the patterns location. If only the pattern location is not in another window, the output of the network will not change at all. And even if the location of pattern has changed enough for it to be in another window, then the pooling will make this difference in distance smaller by being applied to submatrices not to single values.

When pooling is applied on the edges of the input same problems might be encountered as with the convolutions, namely some values will be underrepresented. It is worth noting that the output of the pooling is off smaller size than the input. To address these problems same techniques might be applied.

2.2.4 Summary

In the convolutional subnetwork each layer applies three operations to the input, namely: convolution, which provides us with spatial invariance, activation function and pooling. As a result of this processing, the features are extracted from the data. These features are then used by the classifying subnetwork.

Convolution provides us with spatial invariance and is responsible for extracting features. Each layer of the convolutional part of the network has its own filters. These filters take advantage of the features that were already extracted in the previous layers. Therefore, the later in the network, the more complicated pattern the filter may recognise. It is worth noting that the patterns recognised by the filter are local - the filters are smaller than the input and find patterns that are also smaller than the input, which may appear few times in each sample.

Pooling provides us with resistance to small changes in patterns location.

2.3 Learning Algorithm

One might assume that, due to using convolution and pooling, the learning algorithm will also need to undergo a lot of changes. Luckily, it is enough to compute the partial derivatives for both these functions and include them in the learning algorithm which then will be ready to use.

The learning algorithm proceeds the same way as any other learning algorithm for neural networks. At first the forward propagation is used to obtain the activation of the network for the data samples. Knowing the activation values it is possible to calculate the error in the activation function. This error is then used during backpropagation, when the updates to the network are calculated based on the error and the derivatives of the functions used for calculating the activation. The whole procedure is repeated until the performance of the network is satisfactory - the error in activation function in acceptably small.

2.4 Computation properties of convolutional neural networks

Convolutional neural networks have properties that make the computation fast, namely *parameter sharing* and *sparse interactions* [6]. Both these properties are connected to using convolution.

Parameter sharing means that the same weight is used multiple times during calculating the activation for a sample. Convolution matrices are much smaller than the input and are moved around it, so each time the same set of parameters is used to compute the output. Thanks to that, there is no need to have multiple sets of parameters that would discover the same feature in multiple places of the input - it is enough to have one filter that moves around that input. Because of that, the number of parameters is greatly reduced, and therefore the learning process speeds up.

In a typical neural network each element of input has influence on each element of the output which means that the interactions are dense. In convolutional neural networks the situation is different - the filters are much smaller than the input and discover local patterns, what leads to sparse interactions. This property again causes a reduction in the number of parameters in the model and makes the learning process much faster.

Chapter 3

The Model

In this chapter we will present the approach that have been used to tackle the problem of classification the provided data. First, the data itself will be described in detail. Afterwards, we will present both approach that we investigated during our experiments. The detailed description of the experiments will be given.

3.1 Goal

The goal of this work was to build a model that will well perform the task of classification of the provided data. To complete this task multiple obstacles had to be overcomed, i.e. small data size, missing labels, a big number of hyperparameters that had to be adjusted.

3.2 Data

The dataset describes interactions between a single protein and multiple ligands. One might choose to see the dataset as a four dimensional matrix with axes dimensions described by: the number of ligands, length of the protein, 6 standard pharmacophore features of ligand and 9 types of interactions with amino acid[7]. One data sample can be seen as a 3-dimensional matrix that describes how a protein bounds with a specific ligand. The 3 dimensions are: the length of the protein (number of its residues), 6 standard pharmacophore features and 9 types of interactions with amino acid. A single data sample is presented on figure 3.1.

The 6 pharmacophore features are: hydrogen bond acceptor, hydrogen bond donor, hydrophobic, negatively charged group, positively charged group, aromatic. The 9 types of interactions with amino acid are: any, with a backbone, interaction with sidechain, polar, hydrophobic, hydrogen bond acceptor, hydrogen bond donor, charged interaction, aromatic.

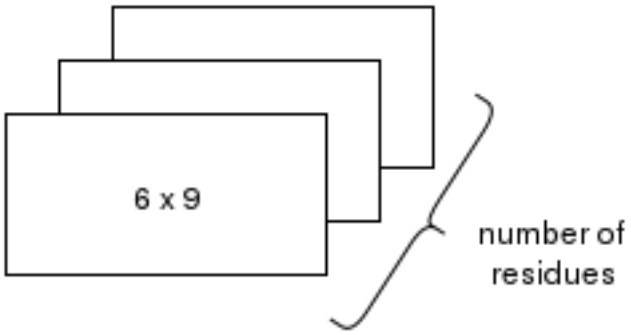

Figure 3.1: A single data sample.

Even though it might be intuitive to look at this data as if it were 4-dimensional, it was stored in the memory in a 3-dimensional form by placing the 6 x 9 matrices adjacent to each other. If we had a protein with only three residues, name them A, B and C, a single sample would look like on figure 3.2.

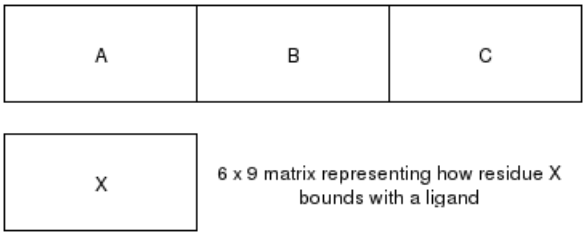

Figure 3.2: Data stored in the memory

The values constituting the dataset are discrete numbers in range 0 to 9. They describe how many interactions of a specific kind there is in a certain residue. The data was very sparse - more than 99% of all values were zeros.

The dataset was stored in 3 files - each file contained samples of only one type: active, inactive, middle (not labled). Out of 5844 samples 2655 were labeled as active, 1945 was labeled as inactive and there were also 1244 unlabeled examples.

3.2.1 Data preprocessing

In order to extract most promising features, the data has been preprocessed. Our goal was to enable the model building such patterns that could detect whether a bound of a specific kind was present in both adjacent residues. We expect that such approach might lead to discovering of interesting correlations.

To achieve such a form of the dataset that would enable this approach, the dataset was extended in the following way:

1. three copies of the dataset have been created.
2. Each copy was put just below the previous one.

3. Each copy was shifted in such a way that going from top to bottom and from left to right would preserve the order of the residues. The shift forced us to either complement each row with zeros or to cut off the residues that would stick out.

4. We decided not to cut off any residues in order to have each of them the same number of times in the dataset - this way no residue will be underrepresented.

As a result each sample had 18 instead of 6 rows and 18 columns more.

The schema of this approach is shown on figure 3.3 which depicts the simple example of a protein with only three residues. It can be observed that a convolution window broader than 9 would be able to detect whether an interaction of some type is present in both adjacent residues while convolution window higher than 6 would be able to detect if two adjacent residues have same pharmacophore features.

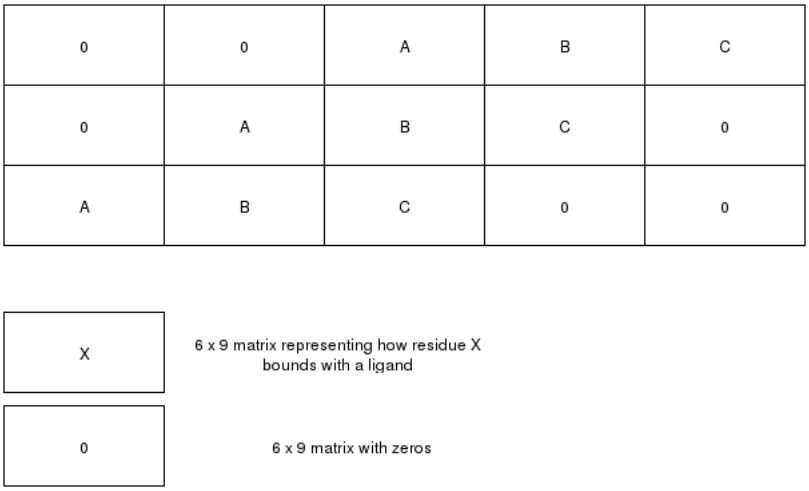

Figure 3.3: Data after preprocessing.

3.3 The Architecture

In this section we will describe the type of architecture which we used for experiments. All the models we have trained were convolutional neural networks with one or two convolutional rectified linear layers (those are layers that use as activation function the rectifier linear function, see 2.2.2). Each layer had 16 or 32 output channels that correspond to number of filters created in each layer. The number of layers have been chosen in such a way that the learning process will not take too much time and the data size will not be reduced too much. Rectifier activation function was used because of its eligible properties [1].

The possible convolution window's sizes were chosen in a way that would enable the model to find filters catching correlation between same types of interaction

in two adjacent residues, what was described in section 3.2.1. The convolution windows were of size $(width, height) \in \{6, 8, 10, 12\} \times \{4, 5, 6, 7, 8\}$.The convolution window's strides were of size $(width, height) \in \{2, 4, 6\} \times \{2, 3\}$.

If both convolution window size and stride had big values in the first layer, it could happen that the data size in the second layer would be too small to satisfy the conditions above. In such cases the convolution window's size and stride in the second layer were reduced in such a way that the convolution window's size would always be smaller than the data size. Moreover, the convolution window's stride would always be smaller than the convolution window and, if only it was possible (i.e. all dimensions of the convolution window were smaller then the corresponding dimensions of the data), small enough to enable existence of at least two "windows" in each dimension.

The shape of pooling windows was (1, 1), (2, 1) or (2, 2) and smaller by at least one than the data size in each dimension so moving the pooling window was always possible. Pooling stride was always equal to or smaller by half than the pooling window in each dimension. Max pooling was used.

The last layer of the network was a softmax layer with two neurons. It was used to classify the sample based on features extracted by the convolutional part of the network.

3.3.1 Finding best architecture

Due to many hyperparameters, there exist many models that fulfill our architecture restrictions and therefore it is not possible to train and measure the performance of all possible architectures. To find the best one we used the tree of Parzen estimators algorithm (see appendix A) provided by a Python library - Hyperopt [4] and let it sample 20 models.

Each architecture that was tested was chosen by hyperopt module. Based on the performance of the already tested architectures hyperopt was choosing another one. Each architecture was passed from hyperopt to the function responsible for measuring the performance of the model. We will call this function an objective function.

Objective function for hyperopt

Each time the objective function created five models of a given architecture and then trained and measured the performance of each. Cross validation procedure was used to obtain different training data for each model. Validation and test set included only labeled examples because classifying unlabeled examples would not be possible. All the unlabeled samples were added only to the training set. The cross validation proceeded as follows: the active and inactive samples were split into five parts of even size. One part became the validation set, one part the test set and the other three parts along with all the unlabeled examples became the training set. The whole procedure was repeated five times.

Each time after training the model its performance on testing data was measured and stored. At the end the mean value of scores of all five models was returned

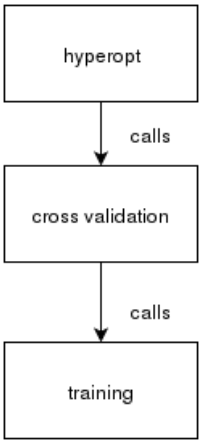

Figure 3.4: The control flow of the experiments.

to the hyperopt module. The score used for measuring the performance of the model was receiver operating characteristic (ROC) with Youden's J statistic. We will cover this topic in details later in this section. Algorithm 1 shows pseudo code for the cross validation procedure.

Algorithm 1 Cross validation

```
1:  procedure OBJECTIVE_FUNC(sample, data_labeled, data_unlabeled)
2:
3:      scores ← empty list
4:
5:      for k ∈ [0...4] do
6:          train_set, validation_set, test_set ← split(data_labeled, k)
7:          train_set ← train_set + data_unlabeled
8:          model ← build_model(sample)
9:          model ← train(model, train_set, validation_set)
10:         score ← measure_performance(model, test_set)
11:         scores.append(score)
12:
13:     return mean(scores)
```

Training the model

The model provided by the hyperopt module was built five times and every time it was trained on another part of the data which was obtained using the cross validation procedure. After each epoch of training the optimal threshold was calculated on the validation set. All samples, for which activation value was above the threshold, were then classified as active samples and those, for which activation value was below the threshold, were classified as inactive. Keep in mind, that there were no unlabeled examples in the validation set.

In order to compute the optimal threshold, we used the receiver operating char-

acteristic (ROC) - a method that is widely used in such problems [29, 30] (see appendix A for more information about ROC). To measure the quality of the threshold the Youden's J statistic [3] (see appendix A for more information) was used. Afterwards, the model's performance on validation data was measured. The score was the Youden's score. If the performance for this model was best until this point of time, the whole model (i.e. all its parameters along with the computed threshold) was dumped to hard drive for later reference. As a result, at the end of the learning process the model's best version was remembered and could be read in order to measure its performance on the testing set and append its score to the list in the cross validation procedure. The threshold calculated during the learning phase was used. The score to measure the performance of the model was the Youden's score.

The details of the learning algorithm are covered in section 3.4.

3.4 The Learning Algorithm

The provided data included unlabeled examples. Two approaches that would enable using these examples to training the model were tested.

3.4.1 Naïve Approach

Training set was constructed in the following way: all examples were included in the training set two times - the labeled samples were included with their label and the unlabeled examples were included once with positive label, and once with negative label. This way the impact on classification of unlabeled data was minimised while the unalabeled data could have been used to improve parameters in the convolutional part of the model.

Example:

If there were the following samples: [A, B, C] along with the following labels: [act, inact, unlabeled] then the training set would look like this: [A, A, B, B, C, C] and the labels would be [act, act, inact, inact, act, inact].

For this approach the stochastic gradient descent algorithm included in Pylearn2 Python library was used [20].

3.4.2 Fancy Approach

In this approach each sample was included in the dataset only once. In order to train the model, a variation of stochastic gradient descent algorithm was written. This enabled using unlabeled examples during the learning process. The SGD implementation provided in Pylearn2 was used as a base [20]. Major changes were introduced in the training function in such a way that the unlabeled examples were used to adjust the parameters of the convolutional part of

the model only and had no impact on the classification part. The pseudo-code of this algorithm can be found below as Algorithm 2.

Algorithm 2 Learning

```
1:  procedure TRAIN(sample, label)
2:      if sample is unclassified then
3:          parameters_on_enter ← current_parameters
4:
5:          SGD(sample, inactive)
6:          diff_vec_1 ← current_parameters − parameters_on_enter
7:          current_parameters ← parameters_on_enter
8:
9:          SGD(sample, active)
10:         diff_vec_2 ← current_parameters − parameters_on_enter
11:         current_parameters ← parameters_on_enter
12:
13:         update_vector = new vector of length same to difference vectors
14:         for el1, el2, up_el ∈ zip(diff_vec_1, diff_vec_2, update_vec) do
15:             if sign(el1) == sign(el2) then
16:                 up_el ← combination_function(el1, el2)
17:             else
18:                 up_el ← 0
19:
20:         for up_el ∈ update_vec do
21:             if up_el is responsible for updating the classification part then
22:                 up_el ← 0
23:
24:         current_parameters ← parameters_on_enter + update_vector
25:     else
26:         SGD(sample, label)
27:
```

For labeled samples the learning process was performed with no changes. When the sample was unlabeled the network parameters were stored and then the sample was presented to the network as if it was labeled as inactive. During this process the network parameters were updated. The difference in the network parameters was stored and old parameters were restored. Afterwards, the sample was presented to the network again - this time as an active sample. The procedure was the same as before. After calculating the difference and restoring the old parameters the two vectors of differences were compared to produce the final vector of updates.

The final vector had the following properties:

1. the elements responsible for updating the classification part of the network were all zeros, therefore the unlabeled examples had only impact on learning parameter of the convolutional subnetwork and did not influence the classification part of the network.

2. the elements responsible for updating the convolutional part of the model

were calculated in the following way:

- if the corresponding elements of the two vectors had the opposite sign, then the corresponding element in the final vector was zero. As a result, the unlabeled samples were used by the network to learn only these filters that were useful for classifying samples of both classes
- if the corresponding elements in both vectors had the same sign, then the corresponding element in the final vector was calculated using the values of the two elements. The final value could be:
  - minimum by absolute value of the two elements
  - maximum by absolute value of the two elements
  - mean of the two elements
  - softmax mean of the two elements, i.e. having $x, y \in \mathbb{R}$ the softmax mean σ is equal to $x \cdot \frac{e^x}{e^x+e^y} + y \cdot \frac{e^y}{e^x+e^y}$.

    Remark

    It can be observed that $\frac{e^x}{e^x+e^y} \in [0, 1]$ for any x, $y \in \mathbb{R}$ and that $\frac{e^x}{e^x+e^y} + \frac{e^y}{e^x+e^y} = 1$, therefore $\sigma = x \cdot \frac{e^x}{e^x+e^y} + y \cdot \frac{e^y}{e^x+e^y}$ is a convex combination of x and y, so σ will be between x and y.

Please, see the example below to understand this concept more clearly.

Concluding, the update vector had zeros in part responsible for classification. If two corresponding values in the vectors of differences had opposite sign, then the corresponding value of the update vector was zero. All other elements were calculated using one of the combination functions.

Example

Let $[+2, +5, +1, -3, +5, +7]$ and $[-2, +3, -1, -7, -7, +7]$ be the vectors of differences, elements 1 to 4 were responsible for updating the convolutional part, elements 5 and 6 were responsible for updating the classifying part of the network and the combination function used was minimum, then the final vector would be $[0, 3, 0, -3, 0, 0]$. Elements 5 and 6 are zeros because they are responsible for updating the classification part of the network. Elements 1 and 3 are zeros because the corresponding values in two vectors have opposite signs. Elements 2 and 4 are minimums by absolute value of the two corresponding values. This example is ilustrated in figure 3.5.

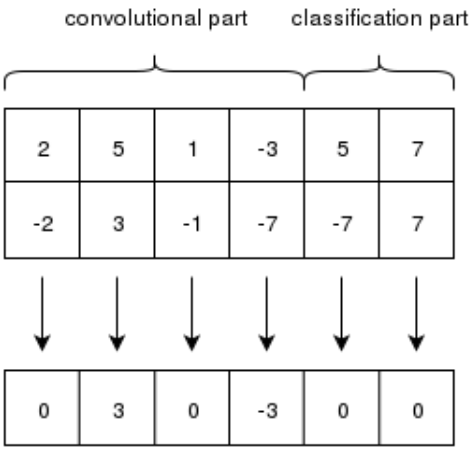

Figure 3.5: Example of the min combining function.

Chapter 4

Results

In this chapter the results we obtained during the experiments shall be described.

4.1 Hyperopt

The figure 4.1 presents the mean score for each architecture chosen by hyperopt module in each iteration for all the combination functions we investigated. As already said, the mean value has been obtained by averaging the performance of a given architecture trained on five different train sets in cross validation procedure. The score used for measuring the performance of the model was $1 - \frac{TP}{TP+FN} - \frac{FP}{FP+FN}$. The goal of the hyperopt module was to minimise that score.

As can be seen, these scores do not reduce their values with consecutive iterations of hyperopt. It suggests that 20 iterations might not be enough for hyperopt to scan the search space with so many parameters well. In the future experiments it is desirable to let the hyperopt sample much more models.

It can be observed that the plots for models that were using the minimum or the mean combination function and the plot for models that were using the softmax combination function are very different. The mean scores of models that used the softmax combination function have much lower variance and, in general, higher values than the mean scores of model that used the minimum or the mean combination function. This suggests that the combination function that is used has a strong influence on the learning process of the model. It is of vital importance to closely investigate how each combination function affects the learning process of the model.

4.2 Architecture

After close investigation it turned out that each architecture that had a poor mean score in cross validation procedure had only one convolutional layer. This

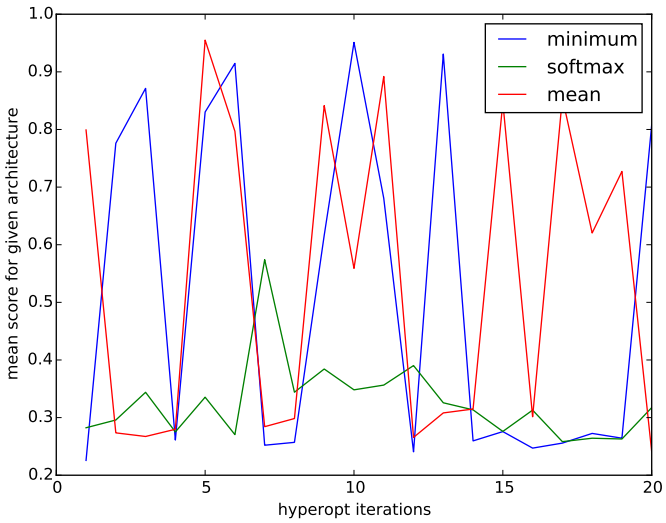

Figure 4.1: Mean score for given architecture in each hyperopt iteration.

was true for each combination function. This is depicted on figures 4.2, 4.3 and 4.4. It suggests that one convolutional layer is not enough to perform the classification on this data with good results. It shows that it will be a good approach to use more convolutional layers in the future. Moreover, it suggests that the patterns in this data are rather complicated.

As already said, each architecture sampled by the hyperopt module was trained five times during the cross validation procedure. It is interesting to see how big were the differences between the best scores of these five models. This data is presented on figure 4.5. As one can see, the variance of performance of architectures that used the minimum or the mean combination function is much bigger than the variance of performance of architectures that used softmax as combination functions. It is another argument that the combination function influences the learning procedure strongly. Due to the big variance it can be considered in the future to split the dataset into more parts in the cross validation procedure.

4.3 Learning process

We have chosen the best architecture using specific combination function and plotted its learning process for all models created during the cross validation procedure. These plots are presented on figures 4.8, 4.7 and 4.6. It can be seen that the initial performances of the networks were rather poor and their values had big variance. This is due to the fact that the initial parameters of the model were chosen randomly. It can be observed that there is not any strong correlation between the initial parameters of the network and its later performance.

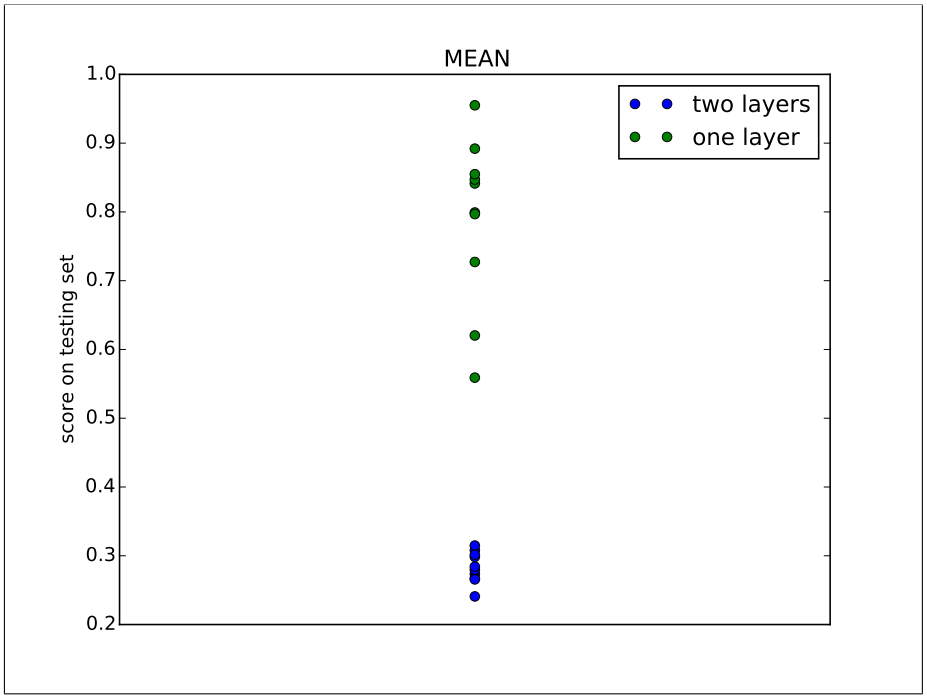

Figure 4.2: Score of the network and number of convolutional layers.

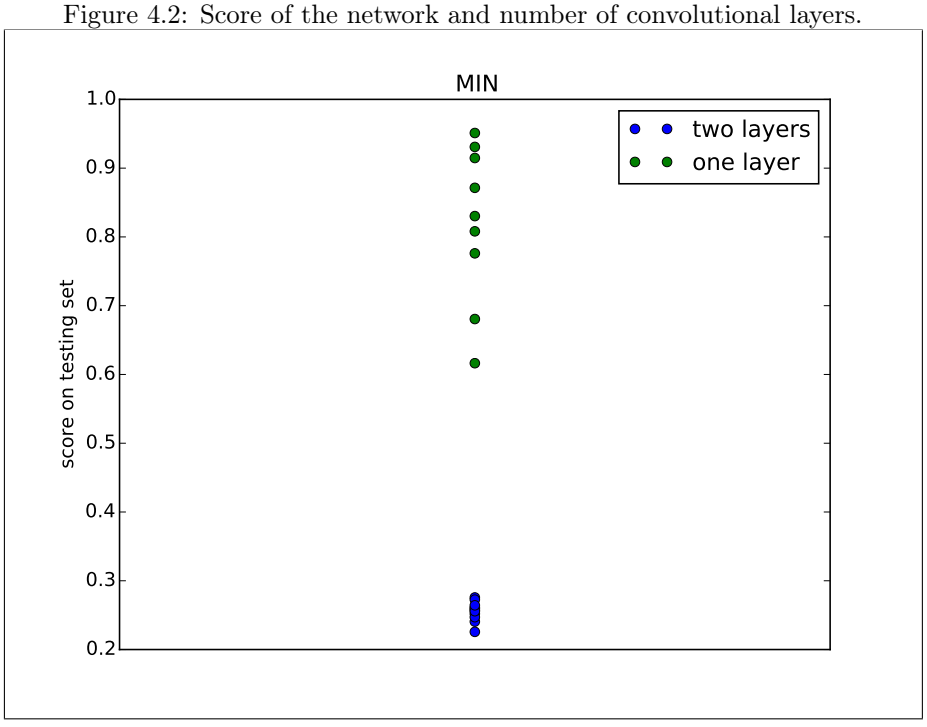

Figure 4.3: Score of the network and number of convolutional layers.

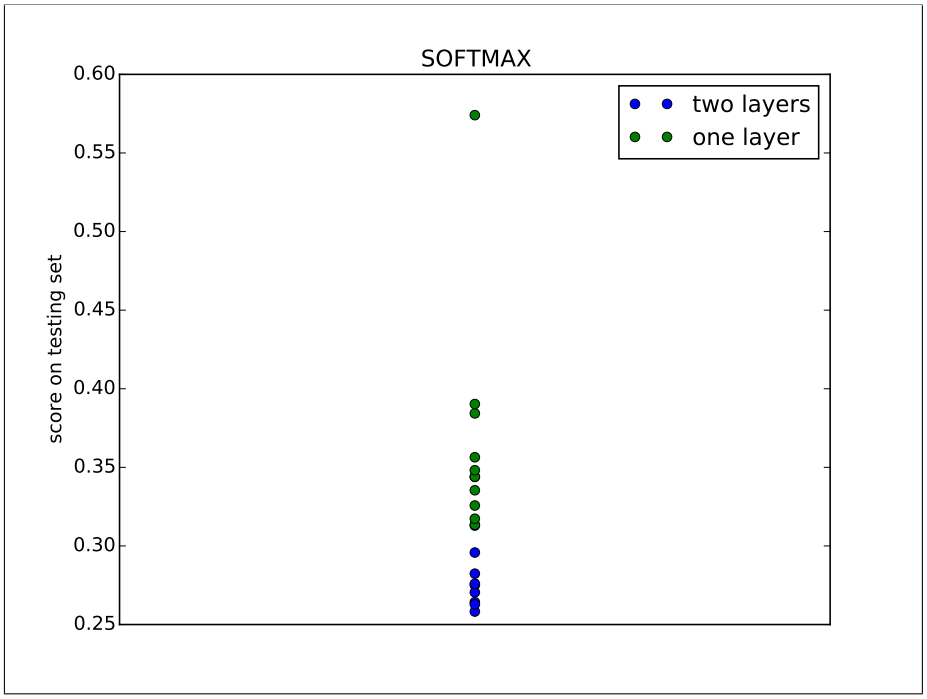

Figure 4.4: Score of the network and number of convolutional layers.

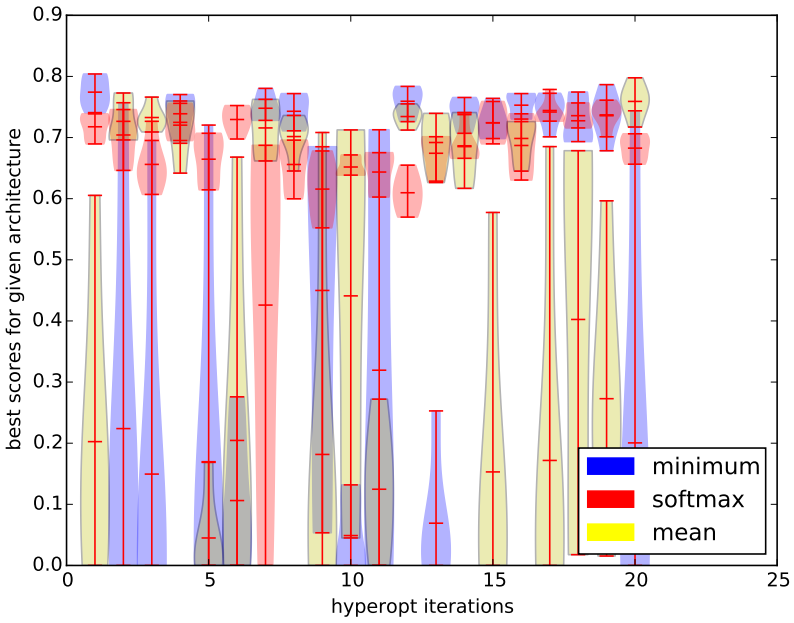

Figure 4.5: Best scores for given architecture in each hyperopt iteration.

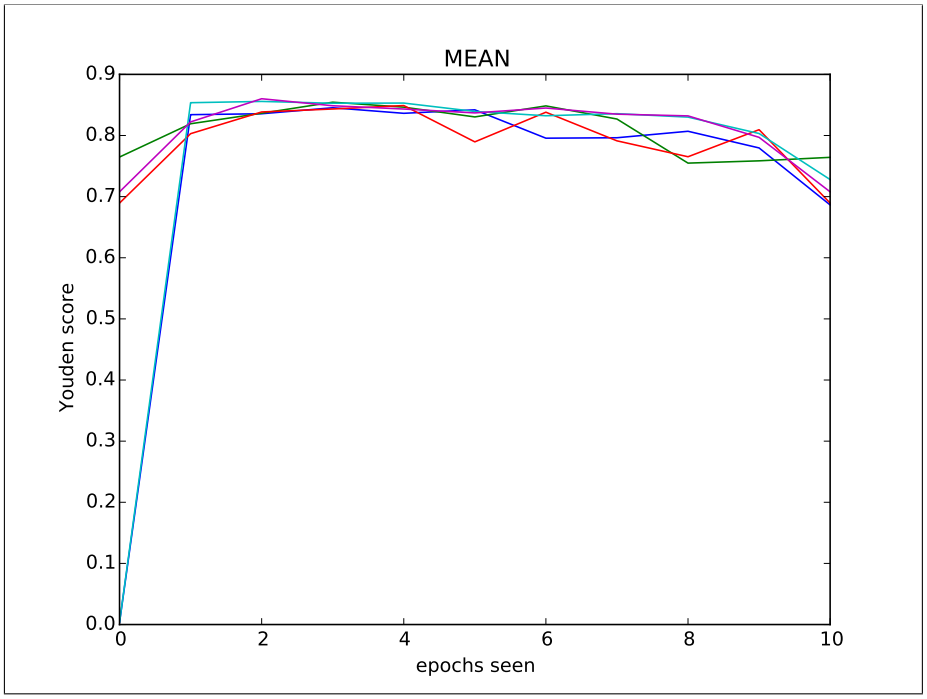

Figure 4.6: Performance of the model with respect to epochs seen.

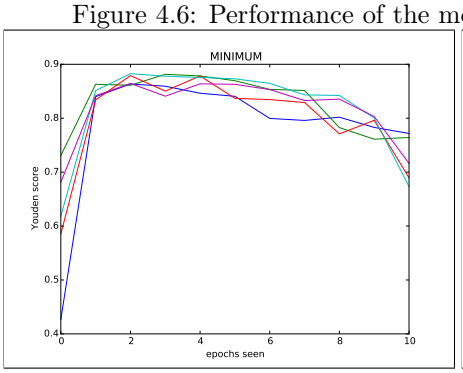

Figure 4.7: Performance of the model with respect to epochs seen.

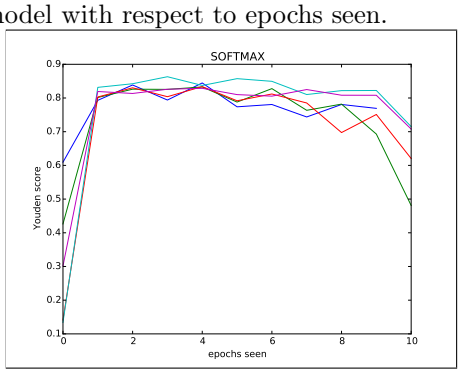

Figure 4.8: Performance of the model with respect to epochs seen.

One can be notice that the biggest improvement in performance was done during the first epoch of the learning process, afterwards the networks were reaching a plateau and finally, their performance was dropping probably due to the overfitting.

One can notice that the performance of architecture that used mean as the combination function was more stable and started dropping later than the performance of architectures that used minimum or softmax as the comination function. It suggests that architectures using mean as combination function might be more resistant to overfitting but it need further investigation.

On figures 4.10, 4.9 and 4.11 there is presented how many models reached its

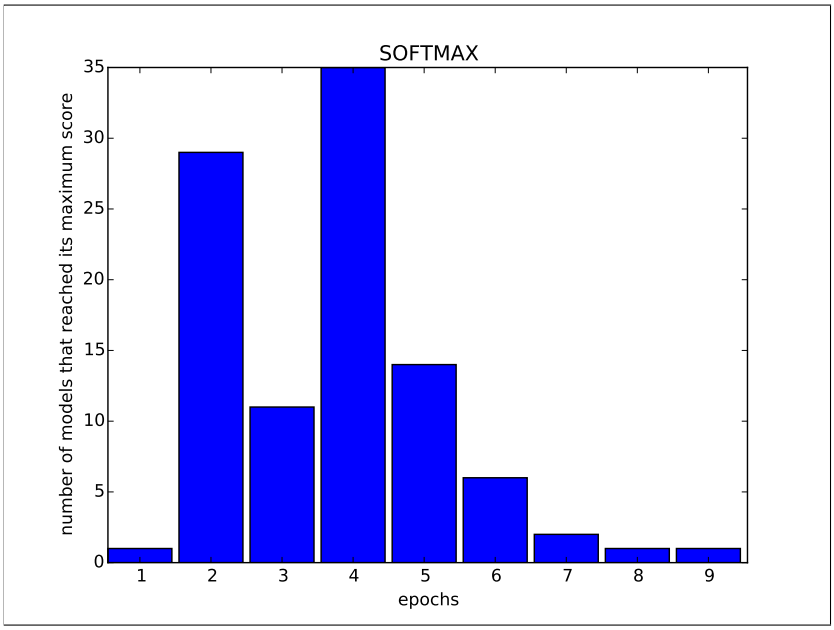

Figure 4.9: Number of models that reached its best score in each epoch.

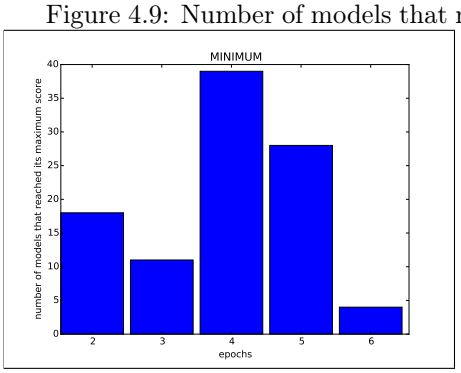

Figure 4.10: Number of models that reached its best score in each epoch.

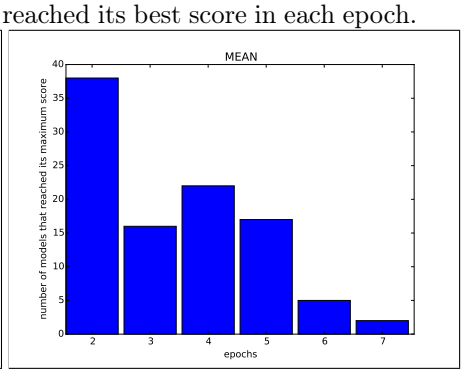

Figure 4.11: Number of models that reached its best score in each epoch.

best performance in each epoch. Regardless of the combination function that the network uses, it usually reaches it best performance between second and fifth epoch. Again there is a significant difference between models that were using different combination fucntions. Those that used mean tend to reach its best performance faster. Those models that use softmax as combination function usually reach its best performance between second and fifth epoch but it might happen that it would happen later.

On figures 4.13, 4.12 and 4.14 there is presented how many models finished learning after specific epoch. One can see that models that used minimum or mean as activation function were usuall finishing learning after either ninth or tenth epoch. Models that as activation function used softmax tend to finish learning after ninth epoch.

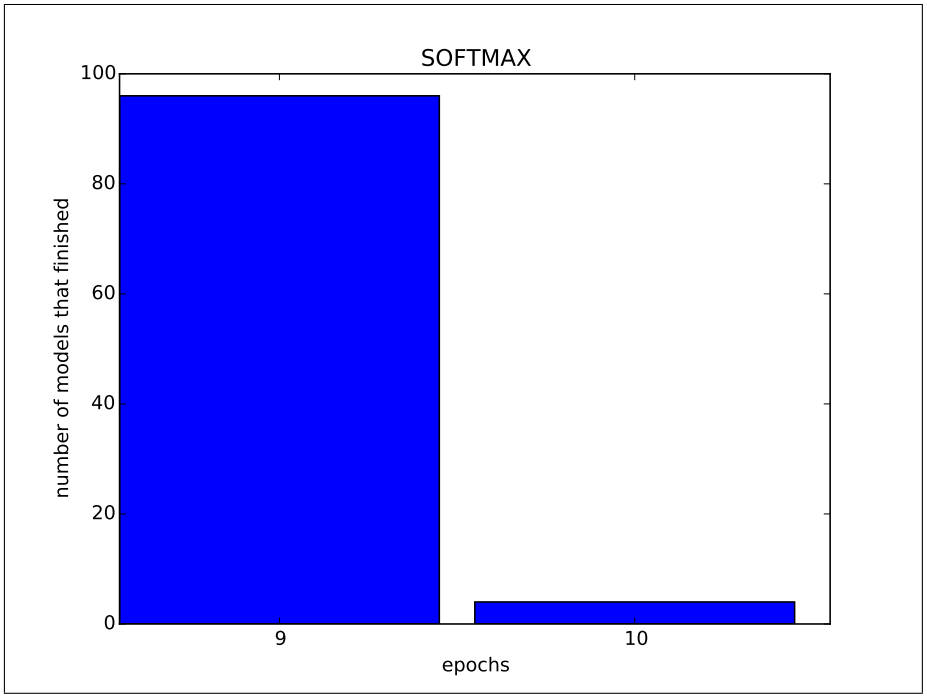

Figure 4.12: Number of models that reached its best score in each epoch.

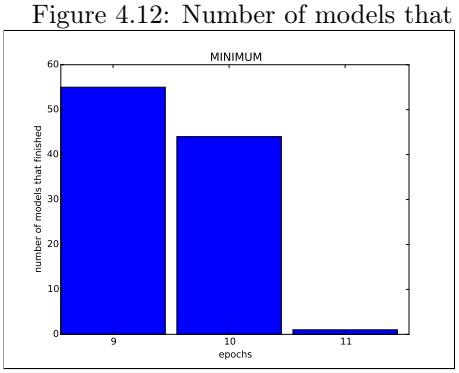

Figure 4.13: Number of models that reached its best score in each epoch.

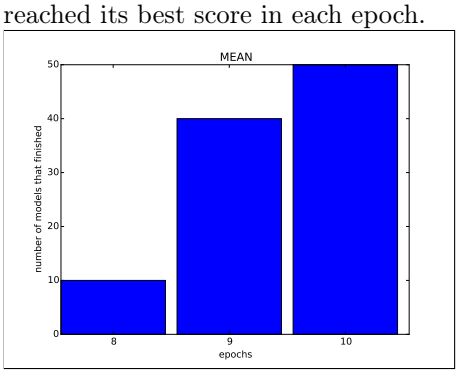

Figure 4.14: Number of models that reached its best score in each epoch.

Chapter 5

Discussion

5.1 napisać co się udało

5.2 Future work

There are still many ways to improve and further investigate the proposed method.

All experiments should be conducted on a bigger set of data which is important for relevance of the results.

The emphasis should be put on investigating how the combination functions affect the networks ability to learn. There is a need to further look for new possible combination functions.

As the experiments have shown, there is a need to enlarge the set of architectures that are being explored - the number of possible convolutional layers should be increased.

Further, it is interesting to test other methods of evaluation the model and explore new methods of finding the optimal threshold for classification. In particular, we would like to investigate the approach which would enable finding two thresholds. Note that the model that created the dataset also used two thresholds so the new approach will lead to a greater resemblance (see section 1.3 for more information). For evaluation the model such metrics as enrichment factor [30, 31] or BEDROC [31] can be used.

The two-threshold model will leave some samples unlabeled. It is of vital importance to explore what error functions can be used in such approach.

Moreover, tt is important to investigate if representing the data in the memory in a 4-dimensional instead of 3-dimensional form can improve the performance of the model (see 3.2 for details). Using another form of representing the data in the memory might lead to discovering other patterns which can be more relevant for this data.

During the experiments we realised that creating three copies of the data during preprocessing (see 3.2 for details) might be redundant. We want to inspect if having only two copies would cause a big difference in performance. Reducing the size of data will also lead to shorter time of learning pahse.

It very important to note, that this work is only the first attempt to aprroach this problem. In reality, the goal is not only to classify molecules as active or inactive but also provide a ranking of those which are most likely to be active. This is because conducting wet-laboratory tests is very expensive and it is desirable to choose only few molecules to test but such that will likely give expected results. Testing all the molecules which are classified as active is not possible. In the future a model that provides not only classification but also ranking shall be proposed.

Since the docking is a stochastic process that is not fully repetitive, it is a common procedure to dock the same molecule multiple times and each time calculate score for this docking. In the data provided each pair protein-molecule was provided only once with its class (active, inactive, middle). In the future it is recommended to use a dataset in which all docking trials are included with the score for each (not the class).

Finally, there exist some techniques for deep and convolutional neural networks that might further improve the performance of our model, i.e dropout and dropconnect methods [1].

bedroc, enrichment factor, bo problem farmakologiczny jest inny

Appendices

Appendix A

Dictionary

The tree of Parzen estimators algorithm or *the tree-structured Parzen estimator* is a method of finding the best hyperparameters of the network. The algorithm runs sequentially - a model is build and its performance is measured. Based on this knowledge another model is chosen and and the whole procedure is repeated.

Receiver Operating Characteristic (ROC) is a plot that measures ratio of false positive rate (FPR) to true positive rate (TPR). $FPR = \frac{FP}{FP+TN}$, where FP is the number of false positive examples and TN is the number of true negative examples. $TPR = \frac{TP}{TP+FN}$, where TP is the number of true positive examples and FN is the number of false negative examples. The example plot is presented on figure A.1.

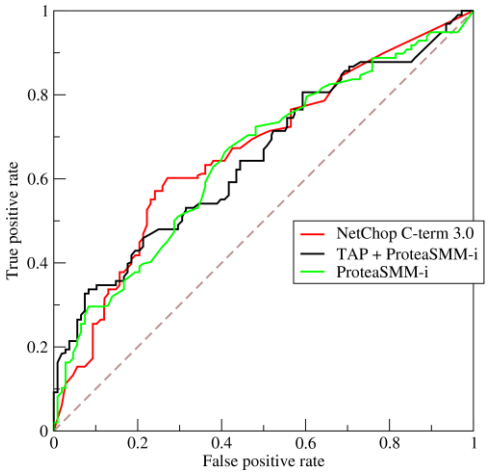

Figure A.1: Example of four ROC curves. The dashed line is the ROC curve of a random classifier. The other curves are ROC curves of three binary classifiers[21].

The ROC is used to measure the performance of a binary classifier. The bigger

the value of true positive rate and the smaller the value of the false positive rate, the better the model. Therefore, a classifier that would lay in point $(0, 1)$ is the best possible model - one that classifies all labels correctly. For each model that returns score for a sample we might decide what should be the threshold above which the model will classify samples as positive. All samples with the score below this threshold will be classified as negative. For a given model we might move this threshold along the axis and plot the performance of the model for each threshold. This way the ROC-curve will be created. There are several methods to score the points along the ROC curve in order to nominate the best threshold for a given model.

Youden's J statistic is one of the methods to score the points along the ROC curve. Mathematically it is described by the following equation:

$$J = sensitivity + specifity - 1,$$

where

$$sensitivity = TPR = \frac{TP}{TP+FN}$$
$$specifity = 1 - FPR = \frac{TN}{TN+FP}.$$

The intuition is that the farer the point is from the line which depicts the ROC curve of a random classifier, the better is the model which generated that point. Note, that in this context 'model' means all model's parameters *and* its threshold. Again, point $(0, 1)$ is the one that is the farthest away from the ROC curve of a binary classifier. On figure A.2 a sample ROC curve with best classifier with respect to Youden's J statistic is shown.

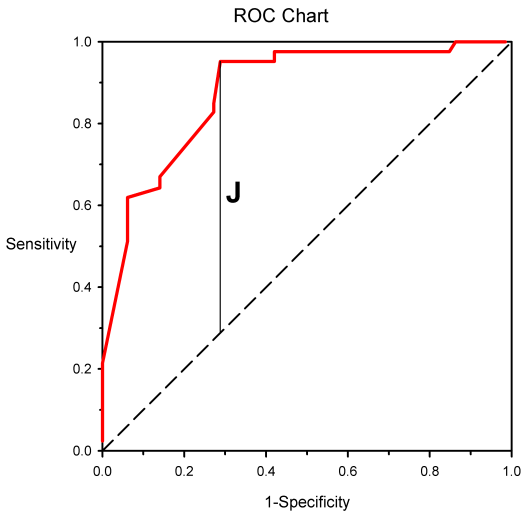

Figure A.2: A ROC curve of a sample classifier. The best point according to the Youden's J statistic is shown with its distance to the ROC curve of a random classifier [22].

Bibliography

[1] J. van Doorn, *Analysis of Deep Convolutional Neural Network Architectures*, 21th Twente Student Conference on IT, (2014).

[2] Y. Bengio, A. Bordes, X. Glorot, *Deep Sparse Rectifier Neural Networks*, International Conference on Artificial Intelligence and Statistics, (2011).

[3] W. J. Youden, *Index for rating diagnostic tests*, Cancer, 3.1, 32–35, (1950).

[4] J. Bergstra, D. Yamins, D. D. Cox, *Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms*, Proceedings of the 12th Python in Science Conference, 13–20, (2013).

[5] M. D. Zeiler, R. Fergus, *Stochastic pooling for regularization of deep convolutional neural networks*, arXiv preprint arXiv:1301.3557, (2013).

[6] Y. Bengio, I. Goodfellow, A. Courville, *Deep Learning*, MIT (w przygotowaniu), `www.iro.umontreal.ca/~bengioy/dlbook`.

[7] S. Mordalski, I. T. Podolak, A. J. Bojarski, *2D SIFt - a matrix of ligand-receptor interactions*, (w przygotowaniu).

[8] A. Breda, L. A. Basso, D. S. Santos, W. F. de Azevado Jr., *Virtual screening for drugs: score functions, docking, and drug design*, Current Computer-Aided Drug Design, 4, 265–272, (2008).

[9] A. Varnek, I. Baskin, *Machine learning methods in property prediction in cheminformatics: Quo Vadis?* J. of Chemical Information and Modelling, 52, 1413–1437, (2012).

[10] H. Geppert, M. Vogt, J. Bajorath, *Current trends in ligand-based virtual screening: molecular representations, data mining methods, new application areas, and performance evaluation*, J. of Chemical Information Modelling, 50, 205–216, (2010).

[11] P. Pethukov, M. Brunsteiner, R. Uddin, *Panthothenate synthetase: new inhibitors for tuberculosis target from a hierarchical virtual screening campaign*, ACS'2006.

[12] S. Mordalski, T. Kosciołek, K. Kristensen. I. Sylte, A. J. Bojarski, *Protein binding site analysis by means of structural interaction fingerprint patterns*, Bioorganic & Medicinal Chemistry Letters, 21, 6816–6819, (2011).

[13] J. Singh, Z. Deng, G. Narale, C. Chuaqui, *Structural Interaction Fingerprints: a new approach to organizing, mining, analyzing, and designing protein-small molecule complexes*, Chemical Biology Drug Design, 67, 5–12, (2006).

[14] A. Krizhevsky, I. Sutskever, G. E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 25, (2012).

[15] R. Collobert, J. Weston *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*, Proceedings of the 25th International Conference on Machine Learning, (2008).

[16] C. G. Wermuth, C. R. Ganellin, P. Lindberg, L. A. Mitscher, *Glossary of terms used in medicinal chemistry*, Pure and Applied Chemistry, Volume 70, Issue 5, 1129-–1143, (1998).

[17] *website of Parallel Architecture Research Eindhoven*, `http://parse.ele.tue.nl/education/cluster2`

[18] *Ligand (biochemistry) - Wikipedia article*, `https://en.wikipedia.org/wiki/Ligand_%28biochemistry%29`

[19] *Deep Learning - Convolutional Neural Networks (LeNet)*, `http://deeplearning.net/tutorial/lenet.html`

[20] - IJ. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, Y. Bengio, *"Pylearn2: a machine learning research library"*, arXiv preprint arXiv:1308.4214, (2013).

[21] *Receiver operating characteristic - Wikipedia article*, `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`

[22] *Youden's J Statistic - Wikipedia article*, `https://en.wikipedia.org/wiki/Youden%27s_J_statistic`

[23] B. Schölkopf, K. Tsuda, J. P. Vert, *Kernel Methods in Computational Biology*, Cambridge, MA: MIT Press, (2004).

[24] B. Chen, R. F. Harrison, G. Papadatos, P. Willett, D. J Wood, X. Q. Lewell, P. Greenidge, N. Stiefl, *Evaluation of machine-learning methods for ligand-based virtual screening*, Journal of Computer-Aided Molecular Design, Volume 21, Issue 1, 53–62 (2007).

[25] R. N. Jorissen, M. K. Gilson, *Virtual Screening of Molecular Databases Using a Support Vector Machine*, Journal of chemical information and modeling, Volume 45, Issue 3, 549–561, (2005).

[26] B. Chen, R. F. Harrison, K. Pasupa, P. Willett, D. J. Wilton, D. J Wood, X. Q. Lewell, *Virtual Screening Using Binary Kernel Discrimination: Effect of Noisy Training Data and the Optimization of Performance*, Journal of chemical information and modeling, Volume 46, Issue 2, 478–486, (2006).

[27] J.-P. Vert, L. Jacob, *Machine Learning for In Silico Virtual Screening and Chemical Genomics: New Strategies*, Combinatorial Chemistry & High Throughput Screening, Volume 11, Issue 8, 677-–685, (2008).

[28] L. Molnár, G. M. Keserű, *A Neural Network Based Virtual Screening of Cytochrome P450 3A4 Inhibitors*, Bioorganic & Medicinal Chemistry Letters, 12, 419–421, (2002).

[29] T. Unterthiner, A. Mayr, G. Klambauer, M. Steijaert, J. Wegner, H. Ceulemans, S. Hochreiter, *Deep Learning as an Opportunity in Virtual Screening*, Deep Learning and Representation Learning Workshop, NIPS, (2014).

[30] H. Geppert, M. Vogt, J. Bajorath, *Current Trends in Ligand-Based Virtual Screening: Molecular Representations, Data Mining Methods, New Application Areas, and Performance Evaluation*, Journal of chemical information and modeling, Volume 50, Issue 2, 205–2016, (2010).

[31] V. Venkatraman, P. R. Chakravarthy, D. Kihara *Application of 3D Zernike descriptors to shape-based ligand similarity searching* Journal of Cheminforminformatics, Volume 1, Issue 19, (2009)