Praca Magisterska

Agnieszka Pocha

September 12, 2015

Abstract

The goal of this work is to... ...drug design... This is achieved by applying (deep?) convolutional neural networks to the problem.

Contents

Chapter 1

Introduction

1.1 related work, że to ma oparcie w czymś i o czym jest praca

Chapter 2

The Problem and the Data

2.1 Terminology

One of major areas of study nowadays is **drug design**. It is a quickly developing field, facing challenging problems, such as conducting experiments which are very expensive and extremely time consuming. Therefore, computer modelling is of vital importance. Artificial intellingence and machine learning have been successfuly incorporated to this field. (citation needed?). One of the most common problems in drug design is telling whether **protein** and **ligand** will together produce an **active** or **inactive** compund.

Proteins are molecules built from **amino acid residues** forming a single **chain**. The number of residues defines the length of the chain. Proteins are present in all leaving organisms.

ligand, protein-ligand docking, receptor, donor, active, not active, pharmacophore, interactions, active non/inactive protein

2.2 Fingerprints

One of the first questions that have to be answered before *any modelling task can be started* is: how will I represent my data? 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. It might seem that a natural representation of a protein will be a graph containing extra information about how the nodes are arranged in space. Unfortunately, graph algorithms are computationally expensive and it is nowadays not possible to use them (na czym się opieram?). Therefore, *we* need another representation, that will carry as much information as possible and *be computationally effective*. *For this reason* many types of fingerprints have been designed and they meet the criterions metioned.

Conventionally, fingerprints represent a *protein/molecule* as a binary(?) vector. Each element of this vector *tells* whether a specific structure is present in the *protein/molecule* or not, e.g. whether the *protein/molecule* has a .... (oprzeć się na jakiejś publikacji). Vectors are widely used in machine learning as data representation as they can be *compilled* into matrices which allows to *easy computation, easy proofs, happy algebra* «We need a simpler representation - such on which we can quickly apply many function, such that is well designed for computers, for modern algorithms» Even though representing a protein as a vector means loosing a lot of information about it, 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. Some of them are designed for specific tasks. (citation needed). In this work 2D-SIFt will be used.

2.3 Problem

drug design, innovative data representation from [2], more details, what exactly am I trying to achieve? *Deep* Convolutional neural networks will be applied to the problem.

2.4 Datasets

The data consists of multiple datasets, each describing reactions between a single protein and multiple ligands. Each dataset consists of four 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[2]. One data sample can be seen as a 3-dimensional matrix that describes how a single ligand bounds with a specific protein. 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. Most of the data samples are labeled as active or nonactive, the rest is unlabeled. A single data sample is presented on figure 2.1.

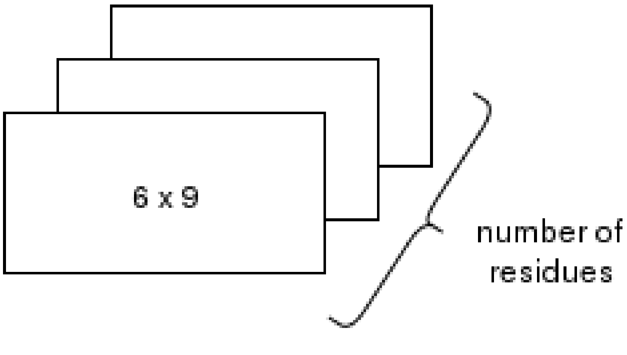

Figure 2.1: A single data sample.

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.

The values constituting the dataset are discrete, namely: 0 to 9. 0 means there is no interaction of specific kind. 1 and 2? As stated above the labels are reperesented as ??? (not active), ??? (active), ?? (no information).

2.5 Sparsity

check if the data is sparse, it yes then state that it is and explain why

2.6 Data representation

Each data sample is represented as a vector of $r * 6 * 9$ length, where r is the length of the protein. Data samples constitute a dataset. Each dataset describes a reaction/bonding between a certain protein and the ligand.

why was this particular fingerprint representation chosen

Chapter 3

The Model or Deep Convolutional Neural Networks

3.1 Deep Neural Networks

DNN

3.2 Convolutional Neural Networks

The simplest definition of convolutional neural networks is probably: neural networks that take adventage of using the convolution operation. Usually the CNN is *conceptually* divided into two subnetworks: first subnetwork is *built from* convolutional layers and is responsible for feature extraction, the second one is a classical neural network, e.g. multilayer perceptron. Its aim is to *poprawnie* classify the examples taking as input the features extracted by the previous subnetwork.

{obrazek}

In this section I will give motivation that stands behind using convolutional neural networks, *explain/define* what is the convolution operation, and give a detailed explanation of CNNs and its properties. Finally, I will describe what problems might arise while learning a CNN model and how to prevent them. The learning algorithm for CNNs will be given.

3.2.1 Motivation

Convolutional neural networks are *mostly* applied to image recognition, video analysis and natural language processing problems.
This attempts *are often succesful/often give better results than (any other)

models*.

3.2.2 Convolution

Convolution operation takes as *operands* two functions (dziedzina? zbiór wartości?) and return a new function (dziedzina? zbiór wartości?) as a result. Mathematically, convolution is defined as:

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

In the equation above we can see c - a function returned by convolution operation, that is evaluated in point t. c is defined as an integral over two other functions. f is often called an input, while g is often reffered to as a kernel.

It is often useful to see kernels as feature extractors. why kernels can be seen as feature extractors?

3.2.3 Computation Flow

As stated above, CNNs can be conceptually divided into two subnetworks. In this subsection I will describe how the signal is processed within the convolutional subnetwork. I will not *dig into* the classifying subnetwork as any neural network that can be used to classify objects might be used. Many such networks exist*s* and they are well described in the literature.

In each layer of the convolutional subnetwork there are three elemental operations are performed. Firstly, the input is convoluted with a kernel matrix. The result of this operation is an input to the activation function. If the input is a matrix *(and it usually/always is a matrix)* each element forms a single input to the activation function *(a może da się inaczej?)*. Finally, the pooling is applied to the result.

może też wzorek? pooling(sigmoid(convolution(input)))

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

In the following subsections I will describe in details how each of this operation exactly works. *Szczególna uwaga zostanie zwrócona na to, jak zmieniają się wymiary danych na każdym etapie*

Convolution for neural networks

The convolution operation was defined as:

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

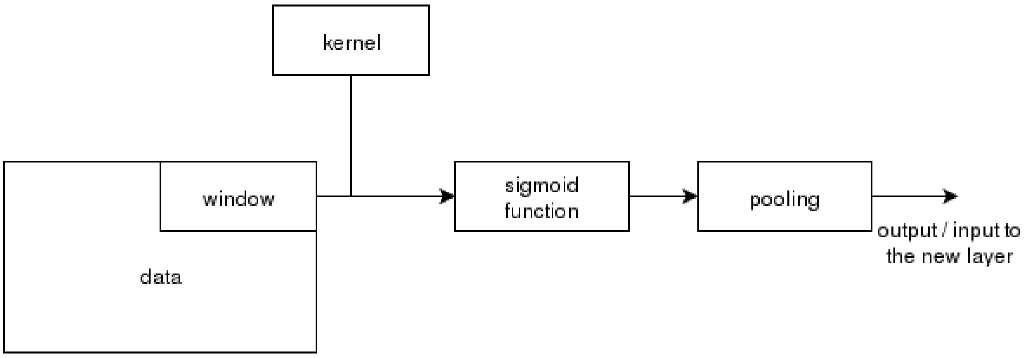

Figure 3.1: The three elemental operations performed in each layer.

It is worth considering how this equation can be applied to neural networks. Real values cannot *?* be represented in *computer calculations* therefore using an integral is *not possible*. It is desirable to have the equation (*number*) in discrete form.

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

From now on *we/I* will call f an input, g a kernel and c na output.

Usually, the kernel that is used is much smaller then the input and convolution is appplied multiple times. Each time kernel is convoluted with a submatrix of input. The result of this operation is a scalar, and a result of a whole process is a matrix.

As stated above, each kernel might be seen as a filter extracting a single feature. Usually, there is need to extract multiple features from the image. As a result, in each layer many convolutional kernels are used. It is worth noting, that kernels are different in each layer. Kernels in the next layer work on the features extracted from the previous layer. *It might be shown* (citation needed), that the features from the next layer are constructed of features from the previous layer. Therefore, in each layer features are more and more complicated.

Let the input be a $I1$x$I2$ matrix and the kernel a $K1$x$K2$ matrix with a 1x1 stride. Therefore the first submatrix to convolute with a kernel will be *$[1{:}K1]x[1{:}K2]$* and it will return a *single value/scalar*. The last submatrix will be *$[I1{-}K1{+}1{:}I1]$x$[I2_k2{+}1{:}I2]$* and it will produce a *single value/scalar* as well. As a result the output will be a $(I1-K1+1)$x$(I2-K2+1)$ matrix.

{obrazek} ilustrujący problem z kernelem i brzegami. Niech na nim będzie, first, second, (może third) i last submatrix

Itcan be observed that the values laying close to the edge of the input matrix will be underrepresented and consequently the output matrix will be of smaller size than the input matrix. There is a variety of ways to address this problem, e.g. zero-pad.

Opisać jakie rozwiązanie zostało użyte przez nas.

Activation function

różne możliwe typy

Pooling

Pooling is an operation that takes as an input multiple values and returns *the statistic(s) of these values*. It is usually applied on a matrix. It takes as *an* input the submatrices and returns a single value as a result. 3 most common *(citation needed?)* types of pooling are:

- max pooling - the max value is returned
- average pooling - the average value is returned
- weighted average pooling - the weighted average is returned. Weights are usually *(citation needed?)* defined by the distance from *what?*

types: Bengio, 181, pierwszy pełny akapit, pooling shape and stride → boost computational efficiency.

{obrazek} jak wygląda max pooling

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 single value. The pooling stride defines where will be the next submatrix with respect to its previous location. Fig *???* illustrates this concept.

{obrazek} pooling size and stride

The same problems might be encountered with pooling on the edges as with the convolutions, namely the output of the layer will be off smaller size than the input unless some techniques avoiding it will *are/will be* applied.

The kernel is smaller than the input. It is moved around the picture. It will find feature everywhere - we need this własność spatial invariance.

{obrazek} dlaczego daje nam invariance

Summary

During the convolutional subnetwork flow the features are extracted from the data. These features are then used by the classifying part of the network. *Convolution/pooling* provide us *with* spatial invariance and *this other awesome thing*. In the convolutional subnetwork each layer applies three operations to the input, namely: convolution, activation function and pooling.

$$output = pooling(activation\_function(convolution(input, kernel)))$$

Convolution and pooling might decrease the size of the input. This might be avoided by zero-pad or other methods. Several types of pooling and activation functions have been provided.

3.2.4 implementationally awesome things

Fast computation

(spatial invariance) → temu nie musimy też mieć osobnych macierzy na feature w każdym miejscu - oszczędzamy pamięć na parametry (i efektywnośc, bo im więcej paramterów, tym wolniej się uczymy). Small kernel = litlle parameters.

sparse interactions

because kernel is smaller then data so it not kazdy z kazdym but some with some (sparse) → computational boost, kernel is small and moved around input - less parameters, instead of a big matrix we store a small one that runs over the data

{obrazek}

parameter sharing

Connected to the fact thet we move the convolution kernel around

{obrazek} a moze nie?

equivariant representations

equivariance - property of *what?* meaning that if the input changes that output changes the same way. $f(g(x)) = g(f(x))$ Intuition about it: detecting feature in a particuler place - feature elsewhere - we find it elsewhere. To what types of transformation is convolution equivariant and to which transformations it isn‘t?

3.2.5 Learning Algorithm

A *zmieniona wersja* of backpropagation algorithm has been provided to *include the changes that must be applied because of* the convolution operation and avoid the diminishing gradient flow. In this subsection the *zmienona wersja* of backpropagation is given and the problem of diminishing gradient flow is *addressed*.

The problems with a classical backpropagation

diminishing gradient flow, niedouczanie się, przeuczanie się, obczaić co o tym mówił Larochelle, on to chyba jednak mówił o głębokich. Wtedy to i tak napisać i przerzucić do głębokich.

Diminishing gradient flow

co to jest, skąd się bierze, można się wesprzeć wykładami Larochelle, on poleca dużo paperów zawsze.

Backpropagation

See (Goodfellow, 2010) from Bengio

3.2.6 Extensions

dropout/dropconnect method, activation functions for dropout, other things from the Dutch paper

3.3 Why was this model chosen

... Having said that kernels might be used as feature extractors it's worth considering what kinds of features might be discovered in the provided data. ...

Chapter 4

The Model

4.1 Goal

The goal of this work was building 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, the untrue distribution of the data.

4.2 Approaches to handle the unlabeled data

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

4.2.1 Naive Approach

Training set was constructed in the following way: all examples were included in the training set two times: the labeled samples were labeled correctly, the unlabeled examples were once labeled as active samples, and once labeled as inactive samples. This way the impact on classification of unlabeled data was minimised but the unalabeled data could been used to improve parameters in the convolutional part of the model.

Example:

if we had the following examples: [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].

The validation set included only labeled samples. The stochastic gradient descent algorithm included in pylearn2 package (include version) was used in this approach.

4.2.2 Fancy Approach

In training set all the samples were included only once along with their proper labels. Once again the validation set included only labeled data. The learning algorithm was *changed* in such a way that the unlabeled examples were used to adjust the parameters of the convolutional part of the model but had no impact on the classification part.

«GOTO: The Learning Algorithm»

4.3 Data

The data consists of multiple datasets, each describing reactions between a single protein and multiple ligands. Each dataset consists of four 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[2]. One data sample can be seen as a 3-dimensional matrix that describes how a single ligand bounds with a specific protein. 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 4.1.

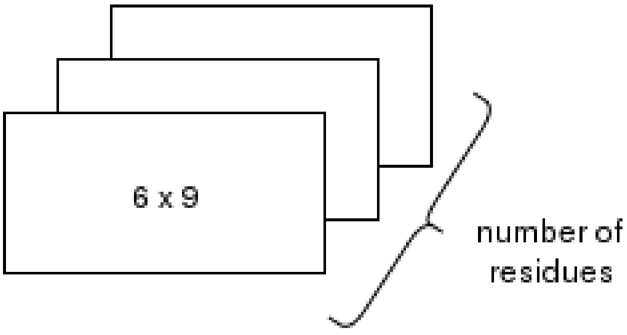

Figure 4.1: A single data sample.

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.

The values constituting the dataset are discrete, namely: 0 to 9. 0 means there is no interaction of specific kind. 1 and 2? As stated above the labels are reperesented as ??? (not active), ??? (active), ?? (no information). The data was very sparse - more than 99% of all values were zeros.

Out of 5844 samples 2655 were labeled as active, 1945 was labeled as inactive and there were also 1244 unlabeled examples.

«DATA EXTENDING»

Remark:

Unreal distribution of labels In reality there are more nonactive *samples* than active, in the data provided the ratio was 1/2.

4.4 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. Each layer had 16 or 32 output channels. 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 good properties.

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 $(width, height) \in \{2, 4, 6\} \times \{2, 3\}$, always smaller than the convolutional window and if it only was possible (i.e. all dimensions of the convolutional window were smaller then the corresponding dimensions of the data) small enough to be able to have at least two "windows" in each dimension. The convolution windows width size was due to the size of the window describing binding of a certain aminoacids. We wanted the network to be able to find patterns describin connection between ... of two following aminoacids, so both ... had to be in one convolutional window.

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 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.

4.5 Hyperparameters

Due to many hyperparameters there are many models that fulfill our architecture restrictions so it is not possible to train and measure the performance of all of them. To perform this task we used the Tree of Parzen Estimators algorithm provided by hyperopt and let it sample «HOW MANY» models.

The objective function choosen to measure the performance of the algorithm created five models of a given architecture, trained and measured the performance of each. Cross validation procedure was used to obtain different training data for each model. The unlabeled samples were present only in the training set. Validation and test set included only labeled examples as classifying unlabeled examples would not be possible. The measure used to determine the model's preformance was F1Score because «WHY?». The detailed description of the objective function used in hyperopt is in pesudo code presented as Algorithm 1.

Algorithm 1 Learning

```
procedure OBJECTIVE_FUNC(sample, data_labeled, data_unlabeled)

    scores ← empty list

    for $k \in$ range(0,K) do
        train_set, validation_set, test_set ← split(data_labeled, k)
        train_set ← train_set + data_unlabeled
        model ← build_model(sample)
        model ← train(model)
        score ← measure_performance(model)
        scores.append(score)

    return mean(scores)
```

4.6 The Learning Algorithm

For learning a variation of stochastic gradient descent algorithm was written. It enabled using unlabeled examples during the learning process. It used the SGD implementation provided in pylearn2 (version) and changed the learning phase in case of getting unlabeled example. The pseudo code of this algorithm can be found below as Algorithm 2.

For labeled samples the learning process was performed with no changes. When the saple 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. The the sample was presented to 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 to final vector of updates.

The final vector had the following properties: the elements responsible for updating the classification part of the network were all zeroes, therefore the unlabeled examples had no impact on training the classification part of the model. 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

Algorithm 2 Learning

```
procedure TRAIN(sample, label)
    if sample is unclassified then
        parameters_on_enter ← current_parameters

        SGD(sample, inactive)
        diff_vec_1 ← current_parameters − parameters_on_enter
        current_parameters ← parameters_on_load

        SGD(sample, active)
        diff_vec_2 ← current_parameters − parameters_on_enter
        current_parameters ← parameters_on_load

        update_vector = new vector of length same to difference vectors
        for el1, el2, up_el ∈ zip(diff_vec_1, diff_vec_2, update_vec) do
            if sign(el1) == sign(el2) then
                up_el ← combination_function(el1, el2)
            else
                up_el ← 0

        for up_el ∈ update_vec do
            if up_el is responsible for updating the classification part then
                up_el ← 0

        current_parameters ← parameters_on_enter + update_vector
    else
        SGD(sample, label)

```

final vector was zero. ecause of that the unlabeled samples were helping 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.

Finally all parameters corresponding to the classifying part of the network were zereod, therefore the unlabeled examples had only impact on learning filters of the network and did not influence the classification part of the network.

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

Example

If the vectors of differences were: $[2.5, 1, -3, 5, 7]$ and $[-2, 3, -1, -7, -7, 7]$, 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 in ilustrated in figure 4.2.

4.7 Experiments

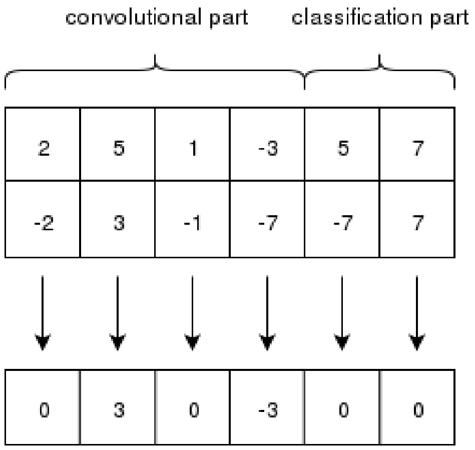

Figure 4.2: Example of the min combining function

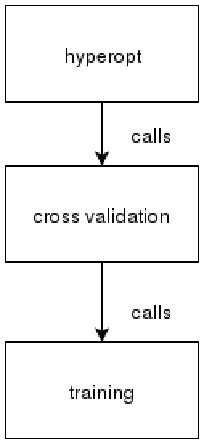

Figure 4.3: The control flow of the experiments.

Chapter 5

What can be improved

5.1 Everything...

...can be improved.

Bibliography

[1] Yoshua Bengio, Ian J. Goodfellow, Aaron Courville - *Deep Learning*

[2] Stefan Mordalski, Igor Podolak, Andrzej J. Bojarski - *2D SIFt - a matrix of ligand-receptor interactions*

Chapter 6

Irrelevant

6.1 zero-pad methods in detail

The easiest way is to let these values stay underrepresented (in MATLAB *citation?* this methodology is called valid), another one is to enlarge the input matrix by adding zeros *at the edges* - this is called zero-pad. One can either add enough zeros for each element of the original matrix to be convolutet exactly the same number of times (in MATLAB *citation?* this methodology is called full) or take only enough zeros for the output matrix to have the same size as the input matrix (in MATLAB *citation?* this methodology is called same).

{obrazek} ilustrujący te przykłady

One can question *legitimacy* of such approach. Adding zeros invites new information into the matrix and might cause additional noise. Instead of adding zeros one might try to change a matrix into torus or instead of zeros use the values that already are present in the original matrix. The added values might be symmetrical *lustrzane odbicie.*

{obrazek} ilustrujący te przykłady

6.2 pooling

is pooling subsampling and if yes then why is polling subsampling?