JMLR: Workshop and Conference Proceedings 46:45–56, 2015 ECML 2014

Signal Correlation Prediction Using Convolutional Neural Networks

Lukasz Romaszko LUKASZ.ROMASZKO@GMAIL.COM
University of Amsterdam

Editors: Demian Battaglia, Isabelle Guyon, Vincent Lemaire, Jordi Soriano

Abstract

This paper focuses on analysing multiple time series relationships such as correlations between them. We develop a solution for the Connectiomics contest dataset of fluorescence imaging of neural activity recordings – the aim is reconstruction of the wiring between brain neurons. The model is implemented to achieve high evaluation score. Our model took the fourth place in this contest. The performance is similar to the other leading solutions, thus we showed that deep learning methods for time series processing are comparable to the other approaches and have wide opportunities for further improvement. We discuss a range of methods and code optimisations applied for the convolutional neural network for the time series domain.

Keywords: deep learning, convolutional neural network, multiple time series, classification, correlation, connectome

1. Introduction

We implement a model involving a deep learning approach suitable for multiple time series processing (Bengio, 2009). The developed model is evaluated on the dataset of fluorescence imaging of neural activity recordings for connection prediction, prepared for the machine learning contest hosted on the kaggle.com platform. The contest, entitled *Connectomics*, was organized by ChaLearn[1] in spring 2014. Understanding the exact brain structure is crucial for research on brain functioning and its learning capabilities. Therefore the aim of the Connectomics is reconstruction of the wiring between brain neurons, which is achieved by estimating the correlations of all pairs of cells (i.e. neurons in a real brain) in the provided network.

2. Dataset and evaluation

The Brain dataset of neural network recordings comes from a simulator (Stetter et al., 2012). The organisers provided several networks, the most important are: four Normal networks for training, Validation and Test for prediction, each composed of $N = 1000$ cells. Another type of networks were 6 Small networks, composed of 100 cells. They required 100 times lower number of predictions, therefore they were good for fast verification purposes. The length of the activity recordings in all the networks is 180 000 frames, which were generated by one hour simulation with a 20 ms frame rate. This frame rate is very low,

1. http://www.chalearn.org/

which makes this task a real challenge. The ground-truth answer is a binary matrix of ones and zeros. The graph is sparse – the edges constitute 1% of all possible connections. The prediction is a matrix of N^2 values between 0 and 1, indicating a confidence that there is a connection between given cells. For this binary classification task the standard Area Under the Receiver Operating Characteristic (ROC) curve scoring measure was used (denoted by AUC). ROC is a graphical plot that illustrates the performance of a binary classifier, when applying all possible thresholds. It compares the order of the predicted confidence with the expected one. It is worth emphasising that the AUC score ranges from 0.5 (random prediction) to 1.0 (ideal prediction). In this contest, for the specifics of the Normal networks structure, which are sparse (connections are 1% of all possible edges), the score of 0.8, is not satisfactory. The score of 0.9, even close to 1.0, would still mean that after applying an optimal threshold there would be more false positive than true positive connections

The most important for connection detection is communication, which means that the recorded signals have spikes at the same time. It is worth emphasising that the baseline Cross-correlation solution is using only this observation. In all of the described approaches we assume that an activity recording is a discrete derivative of the original values of the fluorescence signal of a recorded cell. Therefore, a single cell's activity value at the time frame i is: $recording_i = FluorescenceRecording_{i+1} - FluorescenceRecording_i$. Thus, the value of recording (discrete derivative) equal to 0 means no change in the activity level of the given cell. In the provided dataset the activity recording values range from around -0.2 (a drop) to 1.0 (a spike). The absolute value of a drop is much lower than the one of a spike, since the substance can quickly increase its brightness when provided an input, but then substance brightness decays slowly.

3. CNN Model

Before proceeding to developing a complex CNN model, we tested a simpler, but a very promising solution. The learning method used in the solution is straightforward: each input is encoded into a finite space, with very similar inputs encoded to the same value. The method remembers the number of positive and negative examples of that particular encoded value. Therefore it is a simple pattern recogniser, a mock of a CNN.

3.1. The first solution: Basic Approach

The idea behind the method of connection prediction within a pair is to take fragments of activity recordings (two fragments of recording covering the same period of time, each from one cell) in order to compute the probability that cells with these exact fragments are connected. To have a limited number of possible inputs, values of the recordings were discretised into 4 integer values, representing a drop, no change, a small increase and a spike. The threshold of a spike and increase (0.3, 0.1) was close to the Cross-correlation baseline threshold of a spike (0.2), the three thresholds splitting the values into 4 intervals for discretisation were as follows: -0.05, 0.1, 0.3. The length of one fragment was set to 6, to not exceed the memory allocation limits. Thus the recordings were represented by 4 different possible values, in total 2 fragments, each fragment of length 6 (total input length equals $2 \cdot 6 = 12$). This led to 4^{12} combinations, which is around 17 million different possible inputs. The input values were encoded into 64-bit integer, stored in two 17-million

elements arrays, representing their `#pos` and `#neg` counts. The code was converted to C using Cython library[2], a Python framework for direct translation of a Python code (with previously assigned types to variables) to C. This decreased execution time by more than one order of magnitude.

A probability of a pair being connected, based only by its two 6-length signal values, was defined by the number of positive occurrences in the training dataset (number of cases that such a pair was connected), `#pos`, and the negative number of this particular case, `#neg`. The confidence was then straightforwardly $\frac{\texttt{\#pos}}{\texttt{\#pos}+\texttt{\#neg}}$ for the given case of activity recording. The final confidence for a given directed connection was calculated as an average of the above confidences through all possible time shifts. We tested the performance of the Basic Approach on 6 Small networks of 100 cells by comparing the result to the accuracy of cross-correlation. In Figure 1 we present a comparison of both methods, showing that the Basic Approach outperforms Cross-correlation baseline provided by the organisers. Our Basic Approach is better, since it takes into account longer patterns, instead of basing the prediction on pairs of single values. The Basic Approach is better on average by 3.25 percentage point.

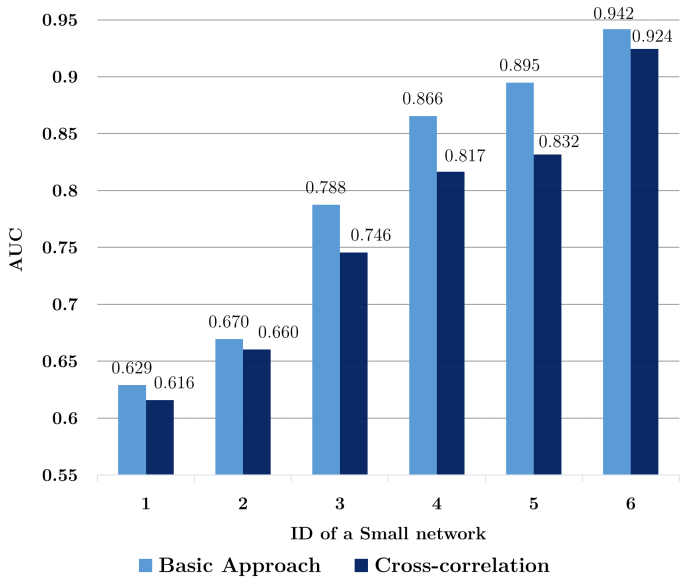

Figure 1: Basic Approach compared to Cross-correlation

The method is simple, however the implementation of optimisations for Cython took significant amount of time. Despite its simplicity, on the Normal networks the presented Basic Approach received 90.0% AUC score. It outperforms the baselines (88.3% and 89.3%). Eventually, together with frame filtering keeping periods of a high activity of the whole network, which we describe later, the Basic Approach obtained a score of 90.6%, and would be classified in top 20% of all the submitted solutions.

2. http://www.cython.org/

3.2. Background on convolutional neural networks

In this section we will present a brief discussion of basics of convolutional neural networks. They constitute the core of our solution, so we describe their structure properties in detail. Finally, we present Max-pooling, which is a technique that helps to improve learning.

3.2.1. Convolutional neural networks

A commonly known, standard feed-forward fully connected Neural Network (NN) is a computational model composed of several layers, where each layer has several neurons (units). An input to a particular unit are outputs of all the units in the previous layer (or input data for the first layer). The unit output is a single linear regression, to which output value a specific activation function is applied. Convolutional neural network (CNN) is a type of NN where the input variables are related spatially to each other. In a standard NN, a permutation (constant for the whole computation) of input variables does not change the final accuracy of the NN, since the model treats them equivalently. However, to detect for example patterns or objects in images, the input (e.g. pixels order in an image) cannot be permuted, since that will lose spatial dependencies. To take into account very important spatial positions, CNNs were developed. Not only they are able to detect general spatial dependencies, but also are capable of specific patterns recognition. Shared weights, representing different patterns, improve the convergence by reducing significantly the number of parameters. CNN recognise small patterns at each layer, generalising them (detecting higher order, more complex patterns) in subsequent layers. Usually a small filter of a particular pattern is applied with all possible shifts to an image. This means that to compute an output (called a feature map) a layer uses the same filter, defined by its weights. This allows detection of various patterns and keeps the number of weights to be learnt very low.

3.2.2. Convolutional layer details

A convolutional layer belongs to a CNN and is determined in particular by the number of filters and their shapes it can learn to recognise patterns. It takes as input fragments of feature maps produced in the previous layer. Usually, layers are fully-connected, which means that next layer filters over all the maps from the previous layer. The fragment size is defined as a 2-dimensional shape. Thus a layer of filter shape $[\texttt{\#maps} \times 2 \times 3]$ takes as input 2 by 3 fragments from each map in the previous layer, and produces a single value of a new feature map. The number of parameters (weight and bias for each edge) to be learnt is then $2 \cdot \texttt{\#maps} \cdot 2 \cdot 3$. The feature map dimension depends on the input dimension as well as on the filter shape. The filter is applied to all the possible shifts in the previous map, thus the new dimension is the input dimension decreased by the filter dimension. For instance, for an input $[50 \times 50]$ and filter shape $[2 \times 10]$, the new feature map is $[49 \times 41]$. When we say that convolutional layers has N units, it means that each of these N units is generating a 2-dimensional feature map, resulting in N 2-dimensional feature maps passed as the output.

3.2.3. Max-pooling

To improve learning, the concept of max-pooling Boureau et al. (2010) has been developed. To decrease the size of an obtained feature map, it is divided into rectangles (pools), in image recognition usually into squares $[k \times k]$ where $k \in \{2, 3, 4, 5\}$. Only the highest activation value of a unit, i.e. a maximal value within the respective rectangle is preserved. It allows to reduce the size of intermediate representation by k^2 in a single layer. Max-pooling keeps information about the highest matching to the given pattern among the units within a respective pool, so the information whether a certain pattern was detected in a given small region is persisted, which is the most important in pattern recognition. While learning, the errors are only propagated to the position of the maximally activated unit.

3.3. Introduction to CNN Filter and CNN Model

The Basic Approach method had a great number of significant disadvantages due to significant information loss, therefore we decided to use CNN to detect patterns, which are often the best state-of-the-art models for patterns recognition (Ciresan et al., 2012). We based our initial CNN structure on the network called Lenet5, which is a CNN designed for digit recognition by LeCun (LeCun et al., 1998) . That network achieves an error rate below 0.9% in the MNIST dataset. We adapted an implementation of Lenet5 for our task of time series processing. Our CNN is composed of five layers: three convolutional layers, then one fully connected standard layer and Softmax. The Softmax function is numerically stable and is commonly used for the classification. It maps n-length input into c classes, the output are probabilities of each class. In our case the Softmax layer has two outputs, one for negative and one for positive class. In this paper by $[N \times M]$ we denote a 2-dimensional matrix. It has N rows, each of a length M. $[K \times N \times M]$ denotes K 2-dimensional matrices. A *Unit* is the another name for a neuron in a CNN.

The learning was performed using 1.2 million of examples, 96% of which were used to learn and 4% to evaluate accuracy of the network during learning. From now on we assume that training is performed using one of the Normal brain network recordings (network: *Normal1*). The number of positive and negative examples is equal, even in the network they are in the ratio 1 to 100. Moreover, all pairs were included the same number of times in their class – for example, if there are 10 000 connected pairs, and we want 30 000 examples in the training dataset, we create 3 examples composed of signals of each pair, starting at different time. The time-shift, i.e. a frame which fragment starts at, for each example was chosen uniformly at random. This condition is important since if all pairs started in some selected frame, e.g. in the first, the network would overfit to match positions of spikes. Learning was performed by optimisation of all CNN weights by Gradient Descent algorithm (LeCun et al., 1998). Below, we refer to a trained CNN as to the *CNN Filter*, since it predicts a probability based on a partial input. The CNN Model is a whole solution of data pre-processing, training and prediction. Within it, the CNN Filter is applied to a pair of fragments of recordings, alike in the Basic Approach. Figure 2 presents the simplified workflow of our solution. In the CNN Model, a final confidence prediction for a particular pair is an average probability, assessed by applying the CNN Filter to different periods of recordings, shifted each time by a constant number of frames, covering the whole input. A single *pass* is an evaluation of all the pairs starting in a given time frame. We perform a

few passes for each pair. The number of passes could by arbitrary, but at least should cover whole input. For the provided networks the minimal number was 5 (input after filtering of length 1500, 330 CNN Filter input). However, assessing the input in equal shifts would cause that some parts of the input would be assessed twice, and other parts only once. Increasing the number of passes improves the accuracy and makes the number of assessments more uniform, finally we perform 14 passes, since we tested that was comparable with the higher number of passes.

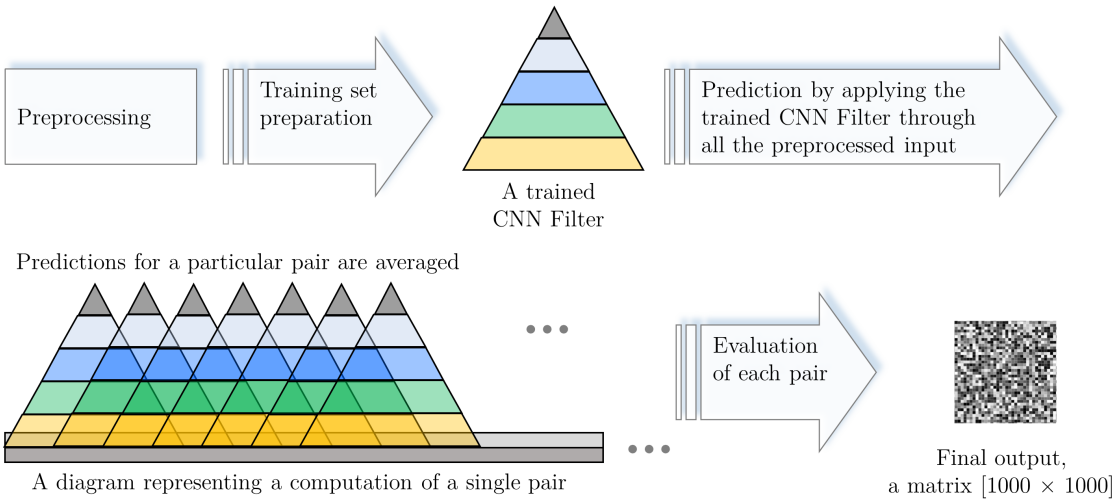

Figure 2: CNN Model - simplified workflow

3.4. CNN Filter key time series processing methods

3.4.1. Spatial representation

The initial idea was to incorporate time information by providing fragments in a two-dimensional matrix ($[2 \times length]$). To take into consideration these time dependencies while learning, filters in the first layer cover both recordings simultaneously. The detected patterns are then combined in the next, higher level layers. In addition to the two rows representing a pair of signals, a third row was computed. It was computed during the pre-processing, based on the whole network data and each time added to the two rows as an additional feature. This row includes overall (the whole) network activity increase in a respective frame. The overall activity is defined as: the total change in the network activity, which equals to the sum of activity recordings of all cells in a respective frame. This allows the CNN filter to learn the network different behaviour depending on its activity level. This led to a major improvement in accuracy, since it significantly enhances distinguishing different states of a network. We tested that applying two filters of height 2, and subsequently again a filter of height 2 in the next layer gave much better result, than just a filter of height 3 in the first layer. CNN was able to learn simple patterns and then predict more accurately by learning their combinations in the second layer. Also, better results were achieved when the second layer had more units (number of combinations) than when more units were set in the first layer. Moreover, the third row of network activity is adjacent only to one of

the cell signals, which occurred that was sufficient, since the network already had that data provided to infer dependencies.

3.4.2. Filtering

The brain time series exhibit two regimes: high activity, when an external input was given to the network, and the low activity, when signal amplitudes are low, mixed with the noise. During short periods of high activity, cells activity is several times higher than normally, which enhances detection of interdependencies. The method of filtering performed during pre-processing consist in keeping only the fragments of high activity, above a particular threshold. The cut fragments were glued together in their appearing order. This method reduces the frames number, in our case, to only around 1% of the initial number of frames in the provided dataset. The network overall activity is defined as total increase in cells activities, which equals to the sum of activity recordings of all cells. The threshold value of 20 (which means that the average of activity recordings in that frame is higher than 0.02) was selected based on local validation. This parameter had to be chosen purely by local validation, moreover, increasing the threshold value preserves even more active time frames (increases accuracy) but the input data amount is smaller (decreases accuracy). Lowering this parameters changes the above properties in the opposite way, which to a certain extent reduce themselves and the accuracy is similar. Therefore the exact value was not that important, the most important was preserving only the most active fragments, i.e. around 1% of the total input. Without filtering, the communication is very rare, and CNN would not be able to learn properly (since in many positive examples there would be no communication in the non-filtered recordings).

3.4.3. Proper activation functions

One of the important decisions related to CNN was setting proper activation functions in the units. We started with default hyperbolic tangent (tanh, $\lambda x.\frac{e^{2x}-1}{e^{2x}+1}$) activation function, afterwards tested other ones. Finally, the most common one, tanh activation function is used in first convolutional layer, while Rectified Linear Unit (ReLU, $\lambda x.\max(0,x)$) in the next two convolutional layers. The latter activation function is often suggested as closer to biological behaviour Nair and Hinton (2010). Since positive indications of correlation should be additive it occurred that using ReLU improved the result. Indeed, ReLU's additive behaviour can improve the positive patterns detection. Suppose that there is a pattern which indicates a correlation, suppose also that the fragment is this pattern's negative, thus obtains a very low, negative score. However, this is not interesting since it does not match the pattern. Therefore, we do not want to allow an input from the previous layer, which may be accidental and it is not an implication of a lower correlation, to reduce the correlation confidence detected in other filters. ReLU units allow to disregard values which were not positive, therefore a particular unit value is computed based on a weighted sum of the positive scores in the previous layer.

3.4.4. Improvement of the CNN Filter

The data was normalised (mean 0, standard deviation 1), what increased learning speed. The same normalisation factors, saved during pre-processing training dataset, were used when normalising test networks.

Afterwards, we substantially improved the network structure. Firstly, we did not apply max-pooling in the first two layers, since the frequency was low compared to cells communication speed. We wanted to keep all the available information, which was already in very low resolution. However, we used max pooling in the last convolutional layer, with a rather high information loss, of length 10 ($[1 \times 10]$). It allowed to select most interesting cases, i.e. to update the weights based on the correlations indication, which had a much higher probability of occurring in the 10-length span. Max-pooling should allow to update the network parameters according to those true correlation indicators. Since the errors are only propagated to the position of the maximally activated unit, it is highly probable that in a wider span there will exist a strong indication of a communication between cells, especially since the data is composed of a high activity periods. Gradient Descent (hereafter GD) will calculate the derivative only based on the maximal ones, thus a possible correlation indicators, on the contrary of performing it based on each input value and regardless its relevance.

Another method applied to improve learning, was momentum Polyak (1964). GD performs steps based on the derivative which has a dimension of a number of parameters and updates each respective parameter independently. With the momentum, GD algorithm updates are based mainly on the previous one, refreshed according to the current gradient. This method allows avoiding local extrema, since a few wrong gradients will not affect the final step direction. We could explain the momentum by a physical movement: when using the momentum, GD steps over time act like a velocity, whilst gradient updates like an acceleration.

#	Layer type	Units	A.F.	Max-pool.	Filter shape	Outgoing dimensions
0	Input	–	–	–	–	(input) $[3 \times 330]$
1	Convolutional	18	tanh	None	$[1 \times 2 \times 5]$	$[18 \times 2 \times 326]$
2	Convolutional	40	ReLU	None	$[18 \times 2 \times 5]$	$[40 \times 1 \times 322]$
3	Convolutional	15	ReLU	$[1 \times 10]$	$[40 \times 1 \times 1]$	$[15 \times 1 \times 32]$
4	Standard	100	tanh	–	–	[100]
5	Softmax	–	–	–	–	(output, 2 classes) [2]

Table 1: CNN Filter details

A.F. denotes activation function. Outgoing dimensions is passed to the next layer. Layer 3 has last outgoing dimension equal to 32 due to max pooling, i.e. 322 div 10 = 32.

3.5. CNN Filter structure

After presenting all the aforementioned crucial CNN features, we can describe the exact setup of the CNN structure. The number of frames passed to the input is 330. This number

was selected based on local validation and the available memory limit, to keep the input length and dataset size high but balanced. The input length, changed within a range of one hundred is almost not influencing the final result, since decreasing (slightly) the length by a factor p (decreases accuracy) would allow a p times larger dataset size for training (increases accuracy). This length of 330 covers around 20% of total simulation time after frame-filtering. The first two rows are cell signals, the third row is overall network activity, which results in an input of 3 by 330 dimension. The network parameters were chosen to get the highest score when tested on Normal networks. Finally, the subsequent network layers are presented in Table 1. Please note that number of units in convolutional layers equals their feature maps number.

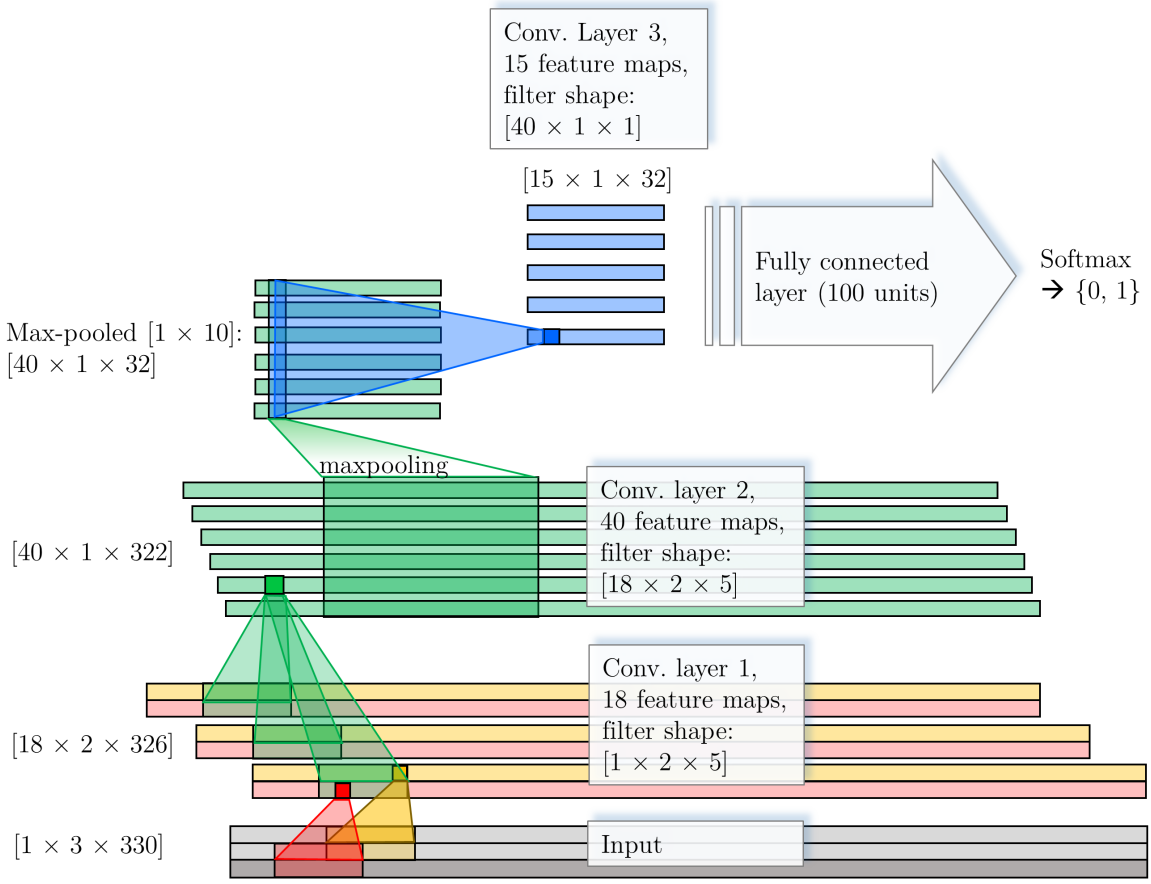

Figure 3: CNN structure

The final structure can be inferred from Figure 3, which also presents some extra visualisations, e.g. the small green square represents a single value in a single feature map, calculated by applying a filter of $[18 \times 2 \times 5]$ weights (which are this feature map's parameters to be learnt during training) to all of the feature maps in the previous layer. Regarding learning settings, the learning rate was set to 0.05, linearly decreasing to 0.01; momentum to 0.5, increasing to almost 1.0 in the last epoch.

To improve the final result we computed 8 separate outputs (Krogh and Vedelsby, 1995) using different seeds to the random number generator. These outputs where aggregated by averaging predicted values of each pair. It occurred that using two times more outputs for averaging increases the final score by a slightly more than 0.1 percentage point. Since one output was around 93.6%, two outputs gave around 93.7%, four 93.85% and finally eight the result close to 94.0%, which increased the initial score by almost 0.4 percentage point.

3.6. On the CNN Model implementation and development

Architecture and performance The code is written in Python, using the Numerical Python (NumPy) library for numerical operations. To allow parallel execution of the code on a graphic card's GPU (Graphics Processing Unit), we use Python library, Theano. Execution was performed on a fast graphic card nVidia K20 with 5 GB memory on board. A computation of one epoch, which processes over one million examples, lasts about 30 minutes. Number of epochs was set to 20, so it took around 10 hours for the training process. Total execution time for the whole workflow was 15 hours. Please note that in one CNN Model run, we evaluate four networks, each by performing 14 passes. This leads to a huge number of 56 million CNN Filter applications, each composed of thousands of computations.

Spatial representation optimisation An important major improvement regarding implementation is how to perform fast creation of pairs in Theano, without copying the values. The input dimension is $[3 \times 330]$, two signals and activity level row. One pass is composed of 1000 iterations (batches), where each batch is assessing 1000 pairs (input examples). Even if the model computation for a given pair is optimised, the significant time is wasted for preparing the input for CNN Filter by storing the input values directly. We solved this problem in the following way: instead of copying all triples of fragments, we stored the data in a form of three columns (tensors), each column with 1000 rows, each row of length 330. All cells signal values for that particular pass is denoted by `all_signals`. The second column is a tile of a cell's signal fragment, `current_cell_signal`. In one iteration we assess all connections incoming to that cell. The value of `current_cell_signal` is updated in each iteration. The third column is composed of `network_act`, network overall activity row, also tiled. `T` denotes `Theano.Tensor` module, `T.horizontal_stack()` concatenates given rows and `T.tile()` virtually copies (tiles) a row given number of times in two dimensions, in our case, in one dimension, 1000 times vertically. The result is an array $[1000 \times 330]$, each row with the same values. All three fragments are joined by:

```
T.horizontal_stack( all_signals,T.tile(current_cell_signal, (1000, 1)),
                    T.tile(network_act, (1000, 1))).
```

The above expression is a tensor $[1000 \times 990]$, each row of three concatenated signal rows. This tensor is passed to the predicting function, which executes CNN prediction for each row. Each row is reshaped during execution to match the $[3 \times 330]$ dimension. Thus during the whole pass, this improvement decreased the number of copied fragments in one pass by a factor of 1000 (from $2,001,000$ to 2001), which significantly decreased execution time of our 56 million CNN Filter evaluations in the CNN Model run.

4. Results

The error rate of the binary classification using our CNN Filter was around 12.5%. Figure 4 presents the error on validation dataset, evaluated during learning. It can be inferred that a comparable convergence results could be achieved at 7-th epoch, but we aimed to increase the accuracy as much as possible. Table 2 presents the score obtained by the CNN Model trained through a particular number of epochs. After 10-th epoch the accuracy increases very slowly, therefore we stop training after 20 epochs. There is no overfitting since the number of parameters is moderate compared to the 1.2 million examples in the training dataset due to shared weights. It would be not reasonable to run the training much longer, since better results could be achieved by an average of two predictions instead of an output generated by one model trained two times longer.

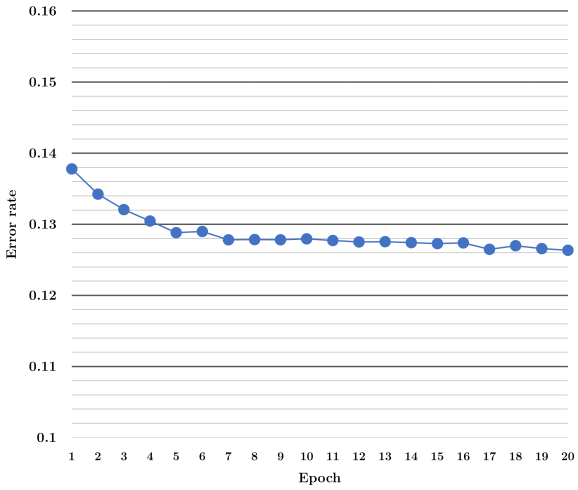

Figure 4: Error rate on a validation set through 20 epochs of learning

# of epochs	AUC score in %
1	92.5
2	93.0
4	93.3
10	93.5
20	93.6

Table 2: The AUC score obtained after training the model through a particular number of epochs.

Solution	AUC score in %
1st place (Team: AAAGV)	94.2
Our CNN Model (4th place, team: Lukasz 8000)	94.0
10th place	92.8
Our Basic Approach (with the filtering)	90.6
30th place	90.4
Baseline: GTE	89.3
Baseline: Cross-correlation	88.3

Table 3: Results – comparison of our solutions and top Contest solutions

The result of a single run of our CNN Model was 93.6%, and averaging the 8 outputs increased the accuracy from 93.6% to 94.0%. The score of 94.0% is therefore the final

Contest result. The final results of the contest are presented in Table 3. The CNN Model significantly outperforms the baselines: Cross-correlation (AUC score 88.3%) and Generalised Transfer Entropy (score 89.3%), as well as our Basic Approach (90.6%). Our Model took the fourth place in the Contest out of 144 teams, achieving accuracy comparable to the other top solutions, where the best solution achieved 94.2% AUC score. On the Validation network the solution took the third place. Since the differences between the results of the top solutions were marginal, we can expect that further exploration of deep learning methods can outperform signal processing techniques.

5. Conclusions

It is worth emphasising that we developed a pure deep learning solution. Incorporation of signal pre-processing methods into our approach might significantly improve its performance. We would get also a higher score, when more output were used for averaging. To conclude, the results are promising, especially because deep learning methods are often the best current state-of-the-art approaches in pattern recognition. Due to CNNs complexity, these models provide a wide range of possibilities of further enhancement and additional experiments. Since such models could outperform current methods in various domains of time series analysis, their more in-depth inspection is left for prospective research.

References

Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning, volume 2, iss. 1, pages 1-127*, 2009.

Y. Boureau, J. Ponce, and Y. Lecun. A theoretical analysis of feature pooling in visual recognition. *ICML 2010, page 111-118, Omnipress*, 2010.

D.C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR, Volume abs/1202.2745*, 2012.

A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning,. *Advances in Neural Information Processing Systems, volume 7, pages 231-238. MIT Press, 1995*, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86, pages 2278-2324*, 1998.

V. Nair and G.E. Hinton. Rectified linear units improve restricted boltzmann machines. *In Proceedings of ICML. 2010, pages 807-814.*, 2010.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics, 4(5):1-17*, 1964.

O. Stetter, D. Battaglia, J. Soriano, and T. Geisel. Model-free reconstruction of excitatory neuronal connectivity from calcium imaging signals. *PLOS Computational Biology, August 2012, Volume 8, Issue 8*, 2012.