UNIVERSITY OF SOUTHAMPTON

Applying Saliency Map Analysis to CNNs on Protein Secondary Structure Prediction

by

Guillermo Romero Moreno

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Physical Sciences and Engineering

Electronics and Computer Science

August 2018

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

Master of Science

by Guillermo Romero Moreno

The relatively new field of deep learning is slowly being transferred to the biomedical field and pushing its state-of-the-art achievements. However, improvements in performance with deep learning machines come with the drawback of opaqueness, which is particularly negative in biomedical research. A few authors have already started applying deep network interpretability techniques on biological problems to overcome this issue, but none of them has approached problems like structural tagging, where every element in the sequence has a classification output.

The aim of this work is to develop interpretability techniques for deep networks that have been trained to solve the secondary-structure prediction problem as one of the best studied structural tagging problems. For doing so, a state-of-the-art-similar convolutional neural network is first trained and then saliency maps are applied to it. Since a saliency maps gets produced per position (instead of by sequence, as in previous studies), new ways of aggregating them for gathering meaningful insights have been construed. These preliminary techniques could be of double value: on one side, they may help biologists to get a better understanding on the underlying protein structural forming process; on the other, machine learning researchers can better understand their machines and spot their previously uncovered flaws.

Contents

1 Introduction **1**

2 Theory **3**
2.1 Deep Learning . . . 3
2.1.1 Convolutional Neural Networks . . . 4
2.1.2 Drawbacks of deep-learning . . . 5
2.2 Interpretability techniques . . . 5
2.2.1 Feature visualization . . . 6
2.2.2 Saliency maps (or how to assign importance scores) . . . 8
2.3 Deep Learning in biology . . . 9
2.3.1 CNNs on biological sequence data . . . 10
2.3.2 Interpretability techniques on biological sequence problems . . . 10
2.4 Protein Secondary Structure Prediction . . . 13
2.4.1 Input features . . . 13
2.4.2 Targets . . . 14
2.4.3 State of the art . . . 15

3 Methods **17**
3.1 Dataset . . . 17
3.2 Network of study . . . 18
3.3 Feature visualization . . . 19
3.4 Saliency maps . . . 19
3.4.1 Extracting information from saliency maps . . . 20
3.5 Open-source . . . 20

4 Results & Discussion **21**
4.1 Outlier analysis . . . 21
4.2 Feature visualization . . . 22
4.2.1 First layer filters . . . 22
4.3 Saliency maps on inputs . . . 22
4.3.1 Sequence-specific saliency maps . . . 23
4.3.2 Class-specific and pssm-specific saliency maps . . . 24
4.4 Final discussion . . . 27

5 Conclusions **29**

Bibliography **31**

Chapter 1

Introduction

Chapter 2

Theory

2.1 Deep Learning

The Deep Learning term was created in contrast to "shallow" Machine Learning. Typical machine learning algorithms involve processing the information from the input into a different *feature space* in which the data is represented in a more structured way. With deep learning, this process is repeated several times allowing for feature spaces with higher-level abstractions that have more power to learn complex relations in the input. These bigger networks were really hard to train at the beginning because of an increased number of parameters that significantly extend the computation training time, a higher risk to over-fit due to their increased power, and *vanishing gradients*, meaning that lower-layer parameters receive very weak signals from the error and could barely learn. Such problems have been recently overcome thanks to the use of Graphical Processing Units (GPUs) that boost computing speed, techniques such as *dropout* (Srivastava et al. (2014)) and *batch normalization* (Sergey Ioffe and Christian Szegedy (2015)) that add extra regularization, and the inclusion of Rectifying Linear Units (ReLUs) as non-linear functions. Since then, its use has been growing and it has been translated to many other research fields, with particularly high performance in image and speech recognition, language translation and natural language processing.

Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) can be found among the most popular deep networks. MLPs are composed by consecutive layers of fully-connected units that pass the information in one direction, and hence are called *feedforward*. RNNs differ from MLPs in that units are also allowed to have connections to themselves. Instead of having an input vector that produces an output in a *one-go* fashion, RNNs store information from previous inputs and are particularly well-suited for inputs that are sequentially

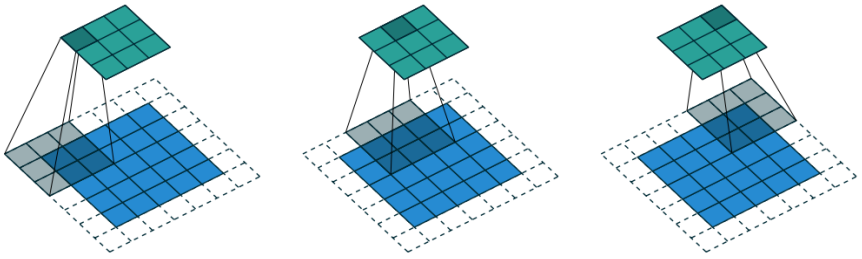

FIGURE 2.1: *Convolution operation with a 3x3 filter, padding of 1x1 and strides of 2x2.* The blue cells represent the input space, the white ones are the result of padding and the green cells conform the output space. The filter operation is performed onto the shaded input cells and produce the shaded output cells. Due to the strides, the filter jumps two positions while sliding. Figure reproduced from Dumoulin and Visin (2016).

correlated. CNNs slide a local filter throughout the input, allowing obtaining location-invariant information. A deeper look into this kind of network is shown in the following sub-section.

2.1.1 Convolutional Neural Networks

As mentioned above, CNNs (LeCun et al. (1998)) work by having filters that operate along some spatially correlated input. Filters are small patches that hold one weight per position. They perform element-wise multiplication with a patch from the input of the same size and add them up into a single output that is located at its centre. A non-linear function is then applied to the output space in order to obtain more expressive models. By performing this operation at all possible overlapping input patches, they produce a feature map that is somehow linked to the underlying input space. Since the same small filter is applied to the whole input space, they capture features or motifs that are location-independent. Moreover, just a small set of weights is used for a arbitrarily large input space, making them much more efficient and able to build deeper networks.

When defining a convolution operation, there are two extra parameters that need to be taken into account: *padding* and *strides*. Padding consists of augmenting the input space by adding margins with zero value. This is especially useful for preserving the size of the input into the feature space, which is done by adding a margin of half of the filter size. The strides define the size of the steps taken every time the filter is slid, determining the amount of overlapping in the produced feature map. Figure 2.1 shows a simple schematic of how a filter with padding and strides works.

A convolutional layer typically includes a few sets of filters with varying size, producing a set of feature maps. Convolutional layers stack onto each other, building more and more abstract concepts. They can also be interleaved by pooling layers that shrink the maps by applying operations to their inputs in a similar fashion as the filters. Typical pooling operations are *max-pool*, which takes the maximum value in a patch, or *average*

pooling, which averages over all units in the patch. The whole structure can be capped by a fully-connected MLP that performs a classification task, and all the weights trained by *backpropagation*.

CNNs have found an increased use in the last few years, thanks to advances in the techniques to build them that led to more sophisticated networks. They have had particular success for the tasks of image classification, object detection, text recognition, speech recognition, natural language processing, among others (Gu et al. (2017)).

2.1.2 Drawbacks of deep-learning

Despite the benefits of improved performance, deep learning still finds some resistance in the broader public due to varied reasons. A First, its higher complexity imposes a steeper learning curve. Second, the lack of mature set of tools and samples due to its novelty makes its access difficult, although there has been an increasing, active community that is swiftly covering these needs. Third, the high computational times are only partially relieved by GPUs. Last, deep networks are hard to interpret and obtaining information from the middle layers renders futile. Their *black box* behaviour hinders developers in both understanding where networks fail and why certain outputs are produced. This drawback has been of big concern lately, and the research community has switched the focus on developing *interpretability techniques* to overcome it.

2.2 Interpretability techniques

Due to the *black box* nature of non-linear deep networks, they cannot be easily interpreted by humans. This is a significant different from previous shallow, linear models, where a look at the weights can give you already an idea of what the network is doing and which inputs are more important. Unfortunately, linear models are limited in their power and cannot tackle complex problems, so in many situations we are forced to employ deep alternatives and sacrifice the interpretability benefits. For this reason, in recent years researchers have been developing interpretability techniques to increase our understanding of deep networks. Although still not as clear as with linear models, we can get some insights of what deep networks are learning and why they take their decisions. Two main groups of techniques have emerged in the literature: *feature visualization* and *saliency maps*. While the first group focuses on the transformations and latent spaces that are formed in the hidden feature layers, the second group tries to detect which parts of the input had more relevance when performing a classification.

2.2.1 Feature visualization

Feature visualization techniques aim to understand what is learned at each layer. A typical example happens with CNNs for image recognition. There was some intuition that the base layers would look for simple patters, like edges or dots, and successive layers would build more complex patterns (eyes, leaves) and end up with full objects at upper neurons. Visualizing first layers was a common practice by then, but it could only reach the basic edge detection level, which is not as useful, as it is a feature shared by most image recognition networks, so other methods of visualization arose.

Linear combination of filters

The most naive approach to represent higher-level filters is to stack them onto lower-level ones and get visualizations similar to the low-level filters, as performed by Lee et al. (2009), who built higher level filters from the filters of lower layers that hold highest weights. Despite its simplicity and unprecedented results for the time, this technique presents some problems. Erhan et al. (2009) point out that the method ignores non-linearities and that "it is not clear how to automatically choose the appropriate number of filters to keep at each layer."

Activation maximization

Activation maximization techniques were first coined by Erhan et al. (2009) and hold a different approach than previous methods, instead of looking at filters they take the visualizations to the input space. The goal is to find inputs that maximize the activation of the specific neuron (set of neurons) that is inspected. The simplest way of doing so would be finding which samples from the training or test set achieve this and try to figure out what they have in common. This simple idea may give us some intuition about the unit, but it is barely as good with networks that are trained in other tasks than image recognition, as is may be much harder to spot the similarities. Besides, it is not straightforward how to obtain the key common features of the images (apart from intuition) and we do not even know how many samples should be gathered per unit.

Erhan et al. (2009) came out with a more principled way of finding maximally activating samples outside of the dataset. They proposed to artificially produce the samples by starting with random noise and tweaking the input space in the direction that increases the activation via *gradient ascent*, while keeping the norm constant. Since most neural networks learn by backpropagation, they are already differentiable and the calculation of the gradients can be recycled for this technique. Formally, we can define the optimization problem as

$$x^* = arg \max_{x \ s.t. ||x||=\rho} h_{ij}(\theta, x) \ , \tag{2.1}$$

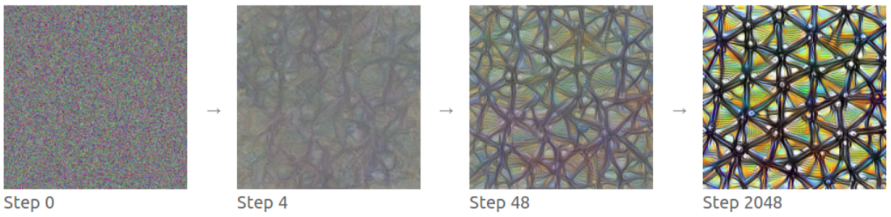

FIGURE 2.2: *Activation maximization method.* The process starts with random noise and performs gradient ascent to find the input sample that maximizes the activation of a unit (or a channel, in this case). Figure reproduced from Olah et al. (2017).

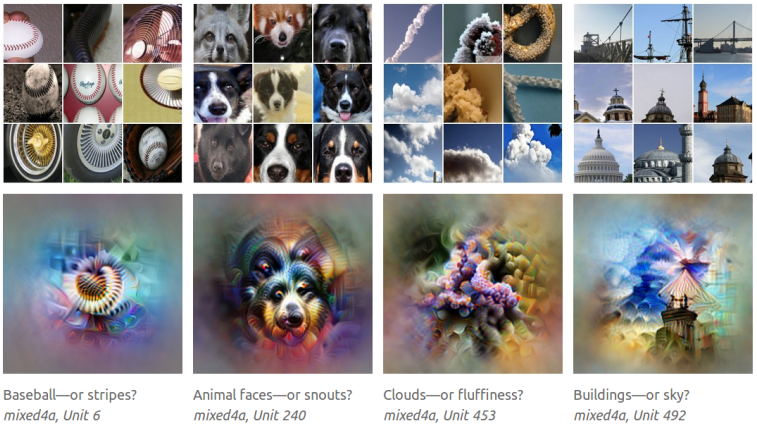

FIGURE 2.3: *Maximally activating samples from training/test set (above) and activation maximization outcome (below) for four different units.* A single activation maximization local minimum does not capture all the variety of the neuron, so a multi-faceted set of them should be sought. Figure reproduced from Olah et al. (2017).

where x is an input vector, h_{ij} the activation unit we want to visualize, and x^* the maximally activating sample. A visual instance of this process can be seen in Figure 2.2.

This technique brings the benefit of keeping the most relevant bits that are captured by the unit and discard all non-relevant information. It can also be easily applied to group of neurons, such as channels, layers or other arbitrary collections, and it can be applied at different stages of the training process in order to visualize how the learning evolves. However, it also has some drawbacks: the optimization function is non-convex and the algorithm can arrive to different local minima, so special techniques need to be added to find all the relevant ones (Nguyen et al. (2016); Olah et al. (2017)); the gradient ascent requires setting a stopping criterion and a learning rate and the process can produce highly unrealistic samples. This last issue could be potentially mitigated by the introduction of duly prior constrains, such as "neighboring pixels needing to be correlated" in natural images (Mordvintsev et al. (2015)). Montavon et al. (2018) materialize the inclusion of this prior knowledge in an *expert term* in Equation 2.1 and argue that its determination is crucial in the resulting sample, and suggest seeking slightly under-fitting expert terms.

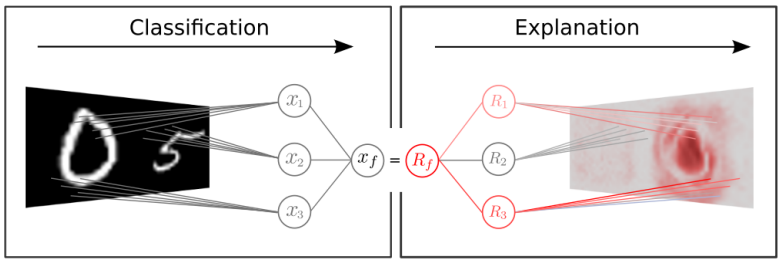

FIGURE 2.4: *Basic sample of a saliency map.* Here, the network is detecting the digit zero (on the left) and the pixels that contributed to the classification are highlighted (on the right). Image reproduced form Montavon et al. (2017).

2.2.2 Saliency maps (or how to assign importance scores)

Saliency maps are a series of techniques that are developed in order to reveal which parts of an input sample are mainly responsible for the output decision that the network makes. They can be a powerful tool both for spotting spurious rules learned by the system (in cases where we can easily judge whether a result is wrong) and learning more about the underlying processes (when we cannot explain the reason of a result, as it is common in biology). An example of a saliency map visualization can be seen in Figure 2.4.

The most simple approach to achieve this goal is performing a sensitivity analysis, i.e. making modifications of the input sample and register how variations in each input (or combinations of them) affect the output of the system. These techniques can be applied for other units different from the post-softmax to give ideas of intermediate layers as well. The set of techniques that perform it can be grouped by the name *perturbation-based approaches* and they all face the same problem: when the input space is moderately big, exploring a decent amount of variations and combinations is intractable, or conspicuously computationally expensive at the least. A second concern relates to its inability to capture the importance of neurons that are already saturated. One example of such methods specifically applied to CNNs are DeConvNets (Zeiler and Fergus (2014)), a machine that assigns scores to certain patches of the input image based on their contribution.

A second series of approaches that mostly overcome the computational problems are based in back-propagation. The most intuitive idea behind them is that the gradient of an input already tells you its sensitivity (how much the output will change from a small change in the input) and it can be calculated on a single back-prop operation instead of performing a profligate number of feed-forward passings with small variations in the inputs (Shrikumar et al. (2017)). It can be thought as a linear approximation of the function around the sample point x_0 by applying first-order Taylor expansion, as

introduced by Simonyan et al. (2014):

$$f(x) \approx wx + b\,,$$
$$w = \left.\frac{\partial f}{\partial x}\right|_{x_0}\,. \tag{2.2}$$

Some other similar methods were developed, with their main differences residing in the treatment of ReLUs in the back-propagation process, such as *guided backpropagation* (Springenberg et al. (2014)). However, they do not give solid explanations for assigning importance since they do not hold basic principles such as conservation (all the importance scores at any layer should sum up to the output score). This principle was later introduced by Bach et al. (2015) with the *Layer-wise Relevance Propagation* (LRP) method, although it has been later discussed that the rules it follows are mainly heuristic and more principled ways of choosing the rules have been suggested (Montavon et al. (2017)). It was later proven that these rules were basically equivalent to having element-wise multiplication of the original saliency maps by the input itself (Shrikumar et al. (2016)). Finally, a last wave of methods propose enhanced method by including a reference point and hence accomplishing a proper Taylor approximation. Main examples are *integrated gradients* (Sundararajan et al. (2017)), Deep Taylor Decomposition (Montavon et al. (2017)), and DeepLIFT (Shrikumar et al. (2017)). They overcome problems of previous methods such as saturation or discontinuities in the gradients.

2.3 Deep Learning in biology

Recent advances in measuring techniques have produced vast amounts of biological data, which usually is highly dimensional and reflects incredibly complex underlying processes. Such settings demand more powerful models, and deep learning has found there a natural field in which to be applied. Among the most prominent application domains in which it has been successfully implemented are images (bioimaging, medical imaging), brain/-body machine interfaces (*BMI*s) and *omics* (Mahmud et al. (2018)). The omics term includes research of sequence data at the cellular level, namely the DNA (*genomics*), RNA (*transcriptomics*), proteins (*proteomics*), and their interaction (*multiomics*). Sequential data is particularly interesting for deep learning, since it is one of the kind of problems in which it excels, particularly with the application of RNNs and CNNs. They have been widely employed for structural annotation of the DNA chain (Jones et al. (2017)), protein classification (Min et al. (2017)), splicing code (Mamoshina et al. (2016)) and transcription factors (Ching et al. (2017)), among others.

2.3.1 CNNs on biological sequence data

As it has been mentioned in Section 2.1.1, CNNs are mostly useful on data which is spatially correlated, which is the case of sequential data. Some reasons for this is that they can accept input sequences of arbitrary size and they are able to capture motifs in them irrespectively of where the are located (Jurtz et al. (2017)). While applying convolutions to 2D input images is a well-known task, it is not as common to apply to one-dimensional data. It is nevertheless fairly intuitive. Filters cover the whole depth (channels) of the data and have variable width depending on the number of neighbouring positions that we want them to capture. Filters slide only in the direction of the sequence, with possibilities of different strides and padding, as in normal 2D convolution. Many simultaneous convolutions operations can be performed at any layer with varying parameters in order to obtain a set of feature maps (sequences) that will serve as input for the next layer. While signals as inputs contain real-value data, categorical sequences (such as DNA or amino-acid chains) are presented in their one-hot version, defining the depth of protein chains as 21 (different possible amino-acids) and four for DNA and RNA (the four bases). Some extra depth can be added with additional known features of the element, such as solvent accessibility or *pssm* values in the case of amino-acids.

There are two main ways in which to apply convolutions to sequence data depending on the problem: structural annotation or sequence classification. The first kind of problems requires the network to assign a specific label or value to each of the positions in the input sequence and therefore the network has to maintain the input length throughout the convolution layers, which can be done with half padding and strides of one. The second problem produces a single output (classification) for a whole sequence. Doing so requires a shrinking process that cannot be achieved by normal means of convolution and pooling operations, since they have fixed reduction sizes and input sequences may have varying lengths. The final step reduction can be achieved by a global max-pooling that retrieves the highest value of the sequence feature map at a single channel, therefore producing a fixed-size vector that can be fed to a soft-max layer (Jurtz et al. (2017)).

2.3.2 Interpretability techniques on biological sequence problems

The bio-medical field is one with special pressures into having interpretable systems. On one hand, the costs of false positive-negatives is expensive when it comes to diagnosis, and finding spurious rule is of particular importance. On the other hand, inspecting what networks are learning can provide with valuable information about the underlying biological processes. Unlike with image classification processes, sequence properties relies sheerly on the arrange of the limited number of possible elements, and being able to find which motifs lead to certain outputs is of particular relevance. Apart from this, having non-interpretable models can impinge the trust of experts in the biomedical field and

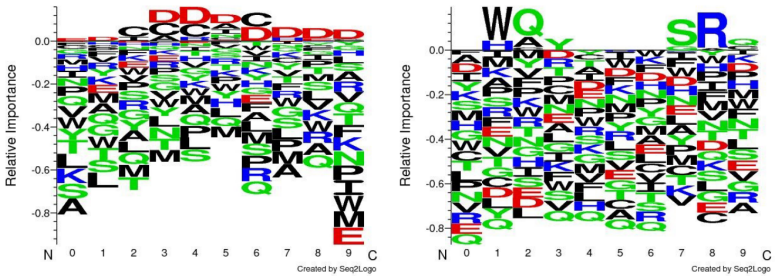

FIGURE 2.5: *Two first-layer filters for a protein sequence problem shown as Seq2Logo.* The x axis represents the span of the filters, width 10 in both filters presented here. While the left filter captures the appearance of two specific amino-acids anywhere around, the right one is more restrictive in terms of the position the must apper at. Figure reproduced from Fontal (2017).

deter them for using the more advanced deep learning versions since they are not able to understand them.

Feature visualization methods

As discussed in Section 2.2.1, a first approximation to visualize hidden features is to inspect the filters of the first layer of a CNN. This gives you a first idea of which basic features the network is building, but is unable to go further than that, so it remains relevant only for shallow structures. When it comes to biological sequential data, an efficient way to convey the information that is also familiar for biology experts are the *Seq2Logo* charts (Thomsen and Nielsen (2012)). 1D convolutional filters can be represented in this way by pairing up their values with the amino-acid of the one-hot row they cover. Two samples of such representations can be seen in Figure 2.5.

Some works have gone further by applying activation maximization techniques as well. Fontal (2017) attempted to apply it for protein sub-cellular localization prediction by applying gradient ascent on the pre-softmax units with a L2 regularization parameter. Lanchantin et al. (2016) applied it to TF binding-site classification of genetic sequences with the same method (applied to post-softmax) and found that 30% of them could be found in known motif databases. As another means of visualizing hidden layer features, Lanchantin et al. (2016) also included temporal output scores for the RNNs models.

Saliency maps

As a simplified version of saliency maps, Alipanahi et al. (2015) and Quang and Xie (2016) spotted the segment in the input genetic sequence that had highest activation of the first-layer filters and compared them to known motifs. This approach only makes sense as long as the network has a single layer, but cannot be applied to deeper networks.

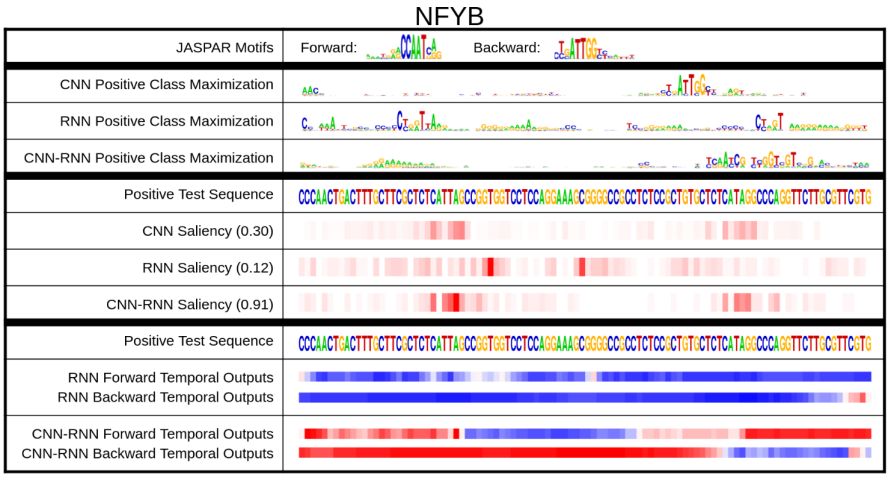

FIGURE 2.6: *Deep Motif dashboard including varied interpretability techniques for TF Binding Site classification.* This figure includes activation maximization methods (top), saliency maps (middle) and temporal outputs (bottom) for CNNs, RNNs, and a combination of both. Figure reproduced from Lanchantin et al. (2016).

Alipanahi et al. (2015) and Zhou and Troyanskaya (2015) showed how genetic mutations affected the output, which is a natural way of performing perturbation-based approaches for the field. Umarov and Solovyev (2017); Fontal (2017) substituted small windows of the sequence —by random genetic code and the general amino-acid X, respectively— and assessed the differences in the output along the sequence by sliding these windows. Kelley et al. (2016) introduced in the centre of DNA sequences known motifs. All of these can be categorized in the *perturbation-based approach* group, therefore need high computational times and cannot be exhaustive enough: how many window sizes and up to which range should be tried to localize the influence accurately?.

Gradient-based approaches have barely been translated to the biological field yet. Lanchantin et al. (2016) include saliency maps with the form of gradient * input for TF binding sites classification, extracted the size 9 window with highest score from each saliency map and compared them with a database of known motifs, matching almost half of the motifs thus produced. Figure 2.6 shows a sample of their saliency maps along with other interpretability methods.

Shrikumar et al. (2017) developed the new back-propagation based technique *DeepLIFT* and simulated a motif detection task within genomic sequence to prove their effectiveness. Finnegan and Song (2017) utilised Markov chain Monte Carlo methods to withdraw samples from the maximum entropy distribution and assessed importance score by looking at the variance of the samples at each position. This technique was applied for previously trained DNA-protein binding CNN and proved to be better than DeepLIFT.

All these methods address classification problems where there is a single output (classification task) for the whole sequence, but, to the best of my knowledge, interpretability techniques have not been applied to structural tagging problems.

2.4 Protein Secondary Structure Prediction

Secondary structure prediction is a long-time studied problem that spans more than 50 years already (Pauling et al. (1951)). It has gone through many different stages as mathematical and computational methods evolved, being recently benefited from deep learning as well, to the point of approaching the theoretical accuracy limit (Heffernan et al. (2017))[1]. Due to its long history, it has also been suggested that it effectively works as a problem benchmark for new techniques dealing with sequence data.

Predicting the secondary structure of a protein purely from its sequence can be seen as a middle step for tackling the much-harder problem of predicting 3D structure. This last problem is of great importance in the biomedical field since the protein structures is the principal indicator of their function, and obtaining this information can be of great use at drug design, disease treatment or early diagnosis, among others. However, due to the molecular scale, these structures cannot be easily measured, and any attempt to do so remains costly. A more feasible alternative is utilising computational tools to predict protein structures from amino-acid sequences alone, as these can be easily obtained from DNA sequencing and they are widely available on heavily annotated public databases (Dill and MacCallum (2012)). In fact, from the nearly 85 millions proteins whose sequences are known, we only know the structure of about 130 thousands (Hattori et al. (2017)).

2.4.1 Input features

The input to the system is an amino-acid chain of length l. Since amino-acids can be of any of 21 categories[2], the most natural way is to treat them as one-hot vectors, forming an input matrix of 21xl. Other features of the amino-acids —such as solving accessibility or other physio-chemical properties (Fauchère et al. (1988))— could also be added to the end of the vector, adding thus extra-information that could be potentially important for making the classification. An typical approach involves inspecting the sequence by taking *n-grams* (i.e. fragments of size n) and classify the middle amino-acid with classical machine learning algorithms such as support-vector machines (SVMs). This has been the approach also held by initial neural networks applied to the problem. With the arrival of CNNs and RNNs, the whole input matrix is fed to the system, as they allow for inputs of varying lengths.

A key turning point for prediction performance appeared when Position Specific Substitution Matrices ($PSSM$) were added as extra features. These values are originated from

[1] Since there is 12% structure homology between proteins (Rost (2001)).

[2] There are 20 standard types while amino-acid X is commonly used for representing all the unknown or non-standard ones (Zhou et al. (2018)). Other ways of handling non-standard amino-acids are also possible (Fang et al. (2017b)).

sequence alignment with *PSI-BLAST* (Altschul et al. (1997)) and express the probability of finding sequences that have a specific amino-acid substituted by another. Each amino-acid of the sequence has then 21 extra features that represent how likely are each of the 21 types of amino-acids found on similar sequences, which gives valuable information about mutations that do not change the structure of the protein (and hence its function) and are then kept by evolution. In order to make most of this information with machine learning algorithms, it is usually filtered through a sigmoid function that the values between 0 and 1 (Jones (1999)), although it can be further normalized to zero mean and one standard deviation (Busia and Jaitly (2017)).

The combination of one-hot encoded (sparse) along with *pssm* (dense) features brings some problems for typical machine learning algorithms: the dense part carries more information, so the weights associated to this part of the input vector learn much faster. Li and Yu (2016) proposed having the one-hot side of the input embedded into a denser representation through an MLP —as it is common in natural language processing (Mesnil et al. (2015))—, but reported only a small marginal performance improvement in doing that (0.05% Q8 accuracy).

2.4.2 Targets

Broadly, there are three kinds of local structures based on the hydrogen bonds formed between the amino-acids: α-*helix*, β-*sheet*, and *coil*, which includes everything else. These were proposed by Pauling et al. (1951) when the protein secondary-structure problem was first defined and have been the targets mostly used in the secondary structure prediction problem and it is generally referred as *Q3*.

A finer classification schema developed by Kabsch and Sander (1983), the *Dictionary of Secondary Structure of Proteins (DSSP)*, focuses in two smaller sub-units: *turns* and *bridges*. Repeating turns would form helices and consecutive bridges create *ladders*, which in turn form sheets. Depending on how many of them you can find together, different kinds of helices or strands could form, leading to a finer classification that comprehends eight classes in total: three kinds of helices, two sheets, and three coils. Having a more minute classification increases the complexity of the problem and has therefore been the focus of recent, more powerful deep learning approaches. This classification schema is commonly referred as *Q8* and an algorithm for its classification was also developed by Kabsch and Sander (1983). A detailed list of the classes can be seen in Table 2.4.2.

The natural way to format them for prediction is their transformation in one-hot vector form, with a softmax function as the last layer of the network as a means to provide with posterior probabilities, $\mathrm{p}(Q|x)$. Networks are typically trained using cross-entropy between targets and predictions and the most commonly used performance score is

Q3 grouping		**Q8 grouping**		**Explanation**
α-helix	H	α-helix	H	Helix with 4 turns
		3_{10}-helix	G	Smaller helix with 3 turns
		pi-helix	I	Bigger helix with 5 turns. Not as common.
β-sheet	B	β-bridge	B	Isolated β-bridge
		β-strand	E	Participates in β-ladders
Coil	C	Turn	T	Turns that do not reach the minimum to be a helix
		Bend	S	Curved piece
		Loop	L	Sometimes also categorized simply as coil (C)

TABLE 2.1: *Targets for the secondary structure prediction problem.* The simpler way of 3-class grouping was further expanded into eight classes by Kabsch and Sander (1983) in their Dictionary of Secondary Structure or Proteins (DSSP).

accuracy, as the percentage of residues correctly classified. Some other performance measures include per-class precision and recall (Wang et al. (2016)), ROC Area Under the Curve (Hattori et al. (2017)), Matthews correlation coefficient (Fang et al. (2017b)), Person's Correlation Coefficient (Jurtz et al. (2017)), or Segment of OVerlap (Wang et al. (2016)).

2.4.3 State of the art

Secondary structure prediction problem has a long history, with first approaches mainly rooted in statistics (Chou and Fasman (1974)). Major breakthroughs can be found with the first implementations of neural networks (Qian and Sejnowski (1988)) and the inclusion of *pssm* values as inputs (Rost and Sander (1993)). A new generation of deep learning networks started recently (Zhou and Troyanskaya (2014)), introducing new major improvements while being more flexible in merging information from local and more distant contexts. Earlier approaches typically used the n-grams or sliding window approach. CNNs, although with a similar principle, have a rather pyramidal structure, being able to increase the span of their action (see Figure 2.7). RNNs are able to capture even longer interactions. Table 2.4.3 collects the best results achieved for Q8 in the last few years. All results are measured on the CB513 benchmark dataset first introduced by Zhou and Troyanskaya (2014). All state-of-the-art algorithms are based on deep learning networks, either in the form of CNNs, RNNs, or both combined.

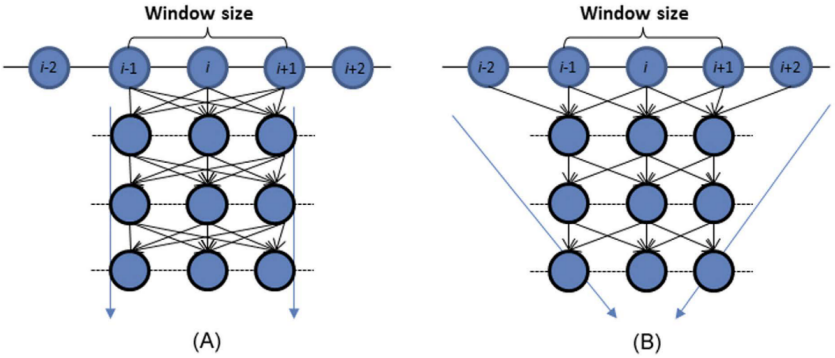

FIGURE 2.7: *Improvements of CNNs over fixed-window sliding.* While for MLP are not as flexible when at capturing information from different contexts (left), CNNs are able to merge local information at the lower layers with longer-range info at upper layers (Busia and Jaitly (2017)). Furthermore, increasing the depth expands the window size without adding much complexity to the model. Figure reproduced from Wang et al. (2016).

Names	Q8	Architecture	Date
Zhou and Troyanskaya (2014)	66.4%	**CGSN**	2014
Magnan and Baldi (2014)	66.5%	SSPro (BRNN)	2014
Sønderby and Winther (2014)	67.45%	LSTM	2014
Wang et al. (2016)	68.3%	**DeepCNF**	Jan 2016
Li and Yu (2016)	69.7%	**DCRNN**	Apr 2016
Lin et al. (2016)	68.4%	**MUST-CNN**	May 2016
Busia and Jaitly (2017)	**71.4%**	**Deep3I+conditioning**	Feb 2017
Hattori et al. (2017)	68.0%	DBLSTM	May 2017
Jurtz et al. (2017)	70.2%	**LSTM+CNN**	Aug 2017
Johansen et al. (2017)	70.9%	BGRU-CRF	Aug 2017
Fang et al. (2017a)	**71.1%**	**DeepNRN+Deep3I**	Nov 2017
Zhou et al. (2018)	70.3%	**CNN+highway**	Jan 2018
Fang et al. (2017b)	70.63%	**MUFold-SS (Deep3I)**	Feb 2018

TABLE 2.2: *State-of-the-art Q8 results on benchmark CB513.* All networks include some sort of deep learning architecture. Architectures in bold indicate some sort of CNN were included. Best Q8 results are marked in bold.

Chapter 3

Methods

Most of the computation work is performed on the *IRIDIS High Performance Computing Facility*, with help of associated support services at the university. They make use of NVIDIA® GeForce GTX 1080 Ti Graphical Processing Units (*GPU*s). The use of the GPUs has been possible thanks to the CUDA 9.0 toolkit[1].

3.1 Dataset

For the purpose of this project, the database produced and made public by Zhou and Troyanskaya (2014) is used[2]. This database has been taken as the flagship benchmark since its release and making use of it allows a fair comparisons with most state-of-the-art algorithms[3]. This dataset includes two sub-sets (training and test) of proteins that come from different sources in order to ensure that the test set is composed by completely new samples. For this same reason, they filtered the training set, stripping out every sequence that held 25% or more similarity with any protein from the test set. The training set was obtained from PISCES server (Wang and Dunbrack (2003))[4] from a date before January 2014, which was in time culled from the Protein Data Bank (PDB) (Berman et al. (2003)). It has an original size of 6133 proteins and 5534 after the filtering. The test set comes from the CB513 dataset Cuff and Barton (1999) and includes 514 sequences[5].

Protein sequences can be up to 700 amino-acids long (with an average of about 214) and have already been pre-processed by Zhou and Troyanskaya (2014), with one-hot encoded inputs for the 21 one types of amino-acid along with a 21-long vector of the *pssm* values

[1] https://developer.nvidia.com/cuda-toolkit
[2] It can be accessed at https://www.princeton.edu/~7Ejzthree/datasets/ICML2014/
[3] See section **??**
[4] http://dunbrack.fccc.edu/Guoli/PISCES_OptionPage.php
[5] Originally 513, but the last one was split in two since it was larger than the 700 amino-acids limit.

and their one-hot encoded secondary structure Q8 classes. The appearance frequencies of the eight classes in both datasets is shown in Figure 3.1.

A sub-set of 256 proteins were taken out of the training set and used for validation, leaving 5278 proteins for the training. This splitting is common in previous papers (Zhou and Troyanskaya (2014); Sønderby and Winther (2014); Busia and Jaitly (2017); Jurtz et al. (2017); Hattori et al. (2017)).

FREQUENCY DISTRIBUTION

Freq. Train (%)	Freq. Test (%)		Class
34.535	30.85	H	(α-helix)
21.781	21.25	E	(β-strand)
19.185	21.14	L	(loop)
11.284	11.81	T	(β-turn)
8.258	9.81	S	(bend)
3.911	3.69	G	(3_{10}-helix)
1.029	1.39	B	(β-bridge)
0.018	0.03	I	(π-helix)

TABLE 3.1: Target frequencies on the CB6133 (left) and the CB513 (right). Table reproduced from Hattori et al. (2017).

3.2 Network of study

The interpretability methods could be applied to any state-of-the-art network, since these would provide the most refined information about the secondary structure problem. However, in order to reach peak accuracies, these models include incredibly huge models that are not handy to work on (especially when it comes to computational time). For this reason, a simpler network is produced and employed for this analysis, with the hope that it will not lose generality over more complex networks.

The network that is used in this study has been built and trained using the open-source code developed by Jurtz et al. (2017)[6] on the *Lasagne* framework (Dieleman et al. (2015)). While most training configurations are preserved (cross-entropy with L2 regularization as error function, batch normalization, uniform glorot initialization, mini-batches of size 64, RMSprop rule updates, gradient clipping), the network architecture itself is completely rebuilt. The weights for the final model are the ones from the epoch that has the best performance on the validation set.

The network is composed by three successive convolutional networks and a dense layer on top. Each of the convolutional layers contains three sets of filters of size 3, 5 and 7, respectively, with 16 filters per size. There are skip connections that by-pass every convolutional network in the same fashion as *ResNet*(He et al. (2015)), so the dense layer gets the concatenation of the raw input along with the outputs from the first, second and third layer altogether. The dense layer has 200 neurons and is connected to a *soft-max*

[6] Accesible at https://github.com/vanessajurtz/lasagne4bio.

layer that gives the output (secondary structure prediction). The convolution operation is carried out with padding at each end of the sequence so as to preserve the sequences length throughout the convolution operations and produce in the final step one label per position.

The total window size for this network is 19, meaning that for making a single secondary structure classification the network obtains information from 9 adjacent positions at each side. Although some authors claim that the network should capture long interactions between amino-acids (Li and Yu (2016); Lin et al. (2016); Hattori et al. (2017); Heffernan et al. (2017)), it has been proven that most relevant information comes from the local environment (Busia and Jaitly (2017)), so the limited window size should not impinge the model. This fact also supports the idea that the skip connections are beneficial since they avoid the most local information to be "washed out" in upper layers (Busia and Jaitly (2017)).

3.3 Feature visualization

First order filters. They only show the first layer of feature extraction. Optimization maximization. Bound to give unreal data. Not credible priors.

3.4 Saliency maps

Saliency maps have been calculated by the common technique of computing the gradient of the output with respect to the inputs and multiplying it by the value of the input itself (Shrikumar et al. (2016)). A major difference between this work and most previous papers that make use of saliency maps is that they perform many-to-one classification (one output class per sequence / image), here the classification task is many-to-many (each position of the sequence is assigned a class), producing many saliency maps for a single sequence. More specifically, each single position produces a saliency map that contains the influence of the 42-size input (21 amino-acids plus 21 *pssm* scores) onto the 8-size soft-max output. This information covers the 19 positions surrounding the one being predicted, ending up with a saliency map with total dimension 8x42x19. They are computed using Theano framework (The Theano Development Team et al. (2016)) since it allows automatic differentiation of symbolical expressions and the use of GPUs.

3.4.1 Extracting information from saliency maps

The presence of overlapping saliency maps allows for different ways of aggregating them and extracting meaningful information. In order to obtain information for a specific sequence, the overlapping saliency maps of such sequence could just be added up, resulting in a single, long *sequence-specific* saliency map of size 8x42xl.

By changing the focus to general information of the network behaviour, the sheer addition of all saliency maps produces a single 8x42x19 map with an average behaviour. From this map we could extract information about the general behaviour around a certain class (*class-specific* saliency map) or a certain amino-acid (*pssm-specific* saliency map). If a more fine-grained inspection is desired, clustering techniques can be used and every cluster create their independent 8x42x19 representative map by addition of all their components.

3.5 Open-source

As it is the common practice in both the bioinformatics and machine learning research communities, all the code from this project has been released as open-source in the web platform GitHub and can be accessed through the url `https://github.com/Juillermo/lasagne4bio`, with the hope that the tools here developed can be of use for future research.

Chapter 4

Results & Discussion

I have trained the network described in section 3.2 for 400 epochs with the same parameters as the ones used by Jurtz et al. (2017); i.e., gradient clipping at 20, regularization term $\lambda = 10^{-4}$, learning rate varying from $\mu = 0.025$ down to 10^{-4} throughout the epochs. The resulting network reaches an accuracy of 67.7% on the test set, which is not far from the 71% of the state-of-the-art (see section **??**). I believe that the techniques of analysis here presented and the conclusions withdrawn from them can be transferred to current state-of-the-art methods without losing validity.

4.1 Outlier analysis

In order to analyse the performance space a bit better, the average accuracy per sequence has been calculated and it has been plotted in Figure 4.1 with respect to the sequence length. The distribution exhibits the typical funnel shape that one could expect from processes with random variables forming groups of different sizes: the bigger the groups, the smaller the variance. The funnel ceases to shrink at length about 400, so it would be particularly interesting to understand why the network is classifying worse (60% and below) some of the sequences above that length.

If we observe the colour scheme of the figure, we can understand right away that sequences rich in α-helix are generally better predicted than β-sheets and coils. An explanation could be that while α-helix sizes are up to five positions away, which is inside the window size, β-sheets interact with amino-aids further away in the sequence, which is not possible to be captured with the window of the network, of lateral size of nine.

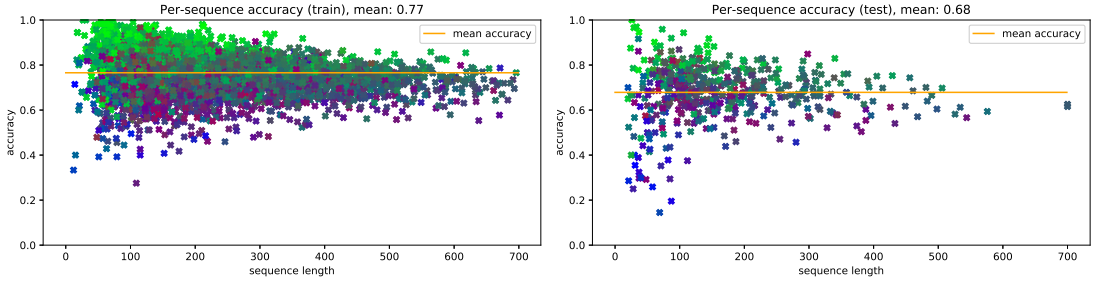

FIGURE 4.1: *The mean accuracy per sequence by sequence length.* The 5504 sequences of the training set are shown on the left and the 514 of the test set on the right. Each point represents a single protein, and its colour corresponds to the amount of β-sheets (red), α-helices (green), and coils (blue) it has. A purple point, for instance, would predominantly have β-sheets and coils.

4.2 Feature visualization

4.2.1 First layer filters

Saliency maps on layers?

4.3 Saliency maps on inputs

Before going through the analysis, it is worth recalling that the saliency map outputs has dimensions 8x42x19, for each position at each sequence, corresponding to the 8 classes (outputs), the 42 inputs and and the total window size of 19 (as explained in Section 3.4).

Analysis on amino-acids and *pssm*

When looking at typical secondary-structure prediction algorithm, there is one point that may raise some suspicion: the inclusion of half of the inputs as one-hot encoded (amino-acids) and the other half as dense vectors (*pssm*). One could thing that this discrepancy may strongly favour the information coming from the dense part, since the weights associated to it will learn much faster in a typical gradient descent learning schema.

Saliency maps can be used to prove whether this hypothesis is right by inspecting which of the input groups is being most decisive in the classification process. For doing so, each saliency map is divided into two groups of $8x21x19$, one corresponding to the values for one-hot amino-acids and the other for the *pssm* inputs, and all the values inside each group are added up in absolute value to a single **saliency score**. Thus, each position of each sequence will have two scores, one for the amino-acids and one for the *pssm*, and

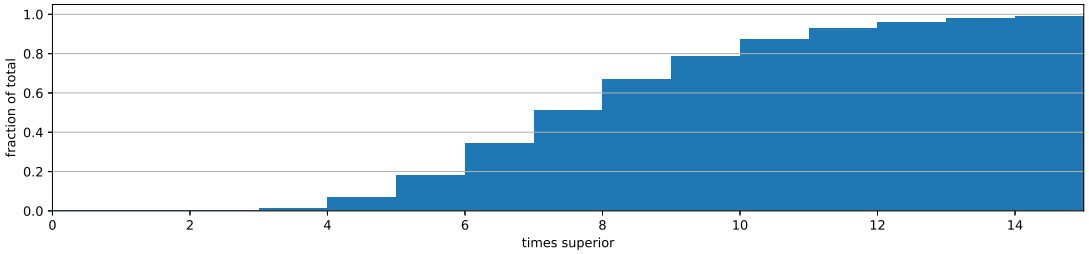

FIGURE 4.2: *Pssm saliency superiority.* This figure shows a cumulative histogram of the number of positions whose *pssm* saliency score was higher than the one-hot score. The x axis represent the superiority ratio, i.e. how many times bigger was the score.

its comparison will give us which part of the inputs is more decisive and quantifies this difference. Only a single case out of the close to 1.3 millions positions showed a higher one-hot than *pssm* score, and the different only accounted for 1% of their value. The rest of the scores were in favour of the *pssm* saliency scores, with the distribution shown Figure 4.2. From the figure we can know that almost the totality of the *pssm* scores were at least four times bigger than the one-hot ones, and therefore we can solidly confirm that the influence of the one-hot data on the input is minor and its omission would not cause much loss in the performance of the network. Consequently, further results focus only on the effect of the *pssm* on classification.

4.3.1 Sequence-specific saliency maps

This section will present the sort of analysis that can be done for a specific sequence. This can be specially useful for analysing the sequences whose accuracy was remarkably lower (outliers). Each sequence has a number of saliency maps equal to its length, l, and they can either be inspected one by one or be overlapped through the sequence, obtaining a single 8x42xl **sequence map**.

Figure 4.3 shows consecutive saliency maps belonging to a single class. The patterns they present are generally preserved (note the differences in scales), just being shifted by one position on the window axis. This suggests that the algorithm is not differentiating that much between specific amino-acid positions, but rather looking for them to be in the vicinity. For this reason, we can consider that overlapping them in a single sequence map does preserve most of the information, as it is shown in Figure 4.4. This sort of map reveals which amino-acids of the vicinity are mostly responsible for a prediction in a particular position. For instance, the predictions of class E on the left side of the shown fragment have to do with the presence of amino-acids A, V and L, while the failure to predict class H on position 109 is likely related to the presence of amino-acid D at position 110. Note that these saliency values come from the aggregation of all the 42 inputs and not only from the one-hot encoded side, which explains why positions 105 and 106 have different values in spite of having the same amino-acid (D). It is specially

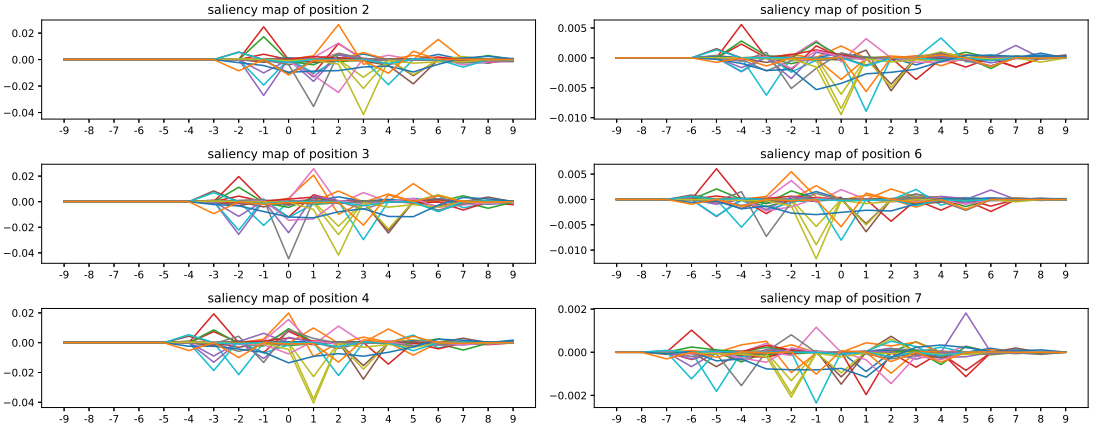

FIGURE 4.3: *Saliency maps for consecutive positions in a sequence.* In every sub-figure there are 42 lines of the 42 inputs, corresponding to their saliency values for class *H*.

important to take this into account since we have discussed before that one-hot amino-acid information is not as decisive for making the predictions.

Another issue worth noting is a non-realistic inconsistency among predictions: the algorithm assigns labels individually per-position and does not take into account the labels it is providing at the sides. This leads to situations such as having individual *G*s, when these only come at least in groups of three. Different approaches in the state-of-the-art for dealing with this are adding a *struct2struct* machine on top of the predictions Rost and Sander (1993); Fang et al. (2017b), iterative processes that learn from previous step's prediction Heffernan et al. (2017), or probabilistic methods that convey information about adjacent positions, like *Conditional random fields* Wang et al. (2016) or *next-step conditioning* Busia and Jaitly (2017).

For a deeper look into the whole *pssm* spectrum, we would need to narrow the scope down to a single class, as it is done in Figure 4.5. This figure reveals that is indeed common that the real amino-acid in the sequence is not among the most influential ones from the *pssm* for making the decision. Note that the pssm values of the amino-acids not always need to have contributions of the same sign (such as *P* or *D* in the figure), revealing that the network is capturing something more than pure presence: location and combinations of amino-acids are also important.

4.3.2 Class-specific and pssm-specific saliency maps

Instead of looking at the saliency information along single sequences, aggregating the information from all the 1.3 millions saliency maps could give us some broader information of what the network has learned. Again, the high dimensionality of the data to explore requires us to seek for meaningful ways to aggregate and represent the data.

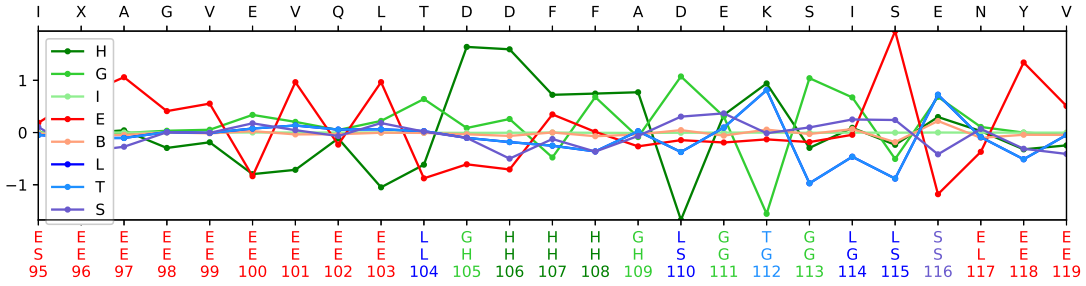

FIGURE 4.4: *Fragment of a sequence map aggregated by input.* Each line corresponds to the aggregated saliency values of one of the eight output classes. The upper x axis displays the amino-acids in the sequence. The labels at lower x axis contains three sets of labels in this order: the predictions, the true values, and the positions in the sequence. The code of colours of the labels is the same as the lines and is set according to the class of the predictions.

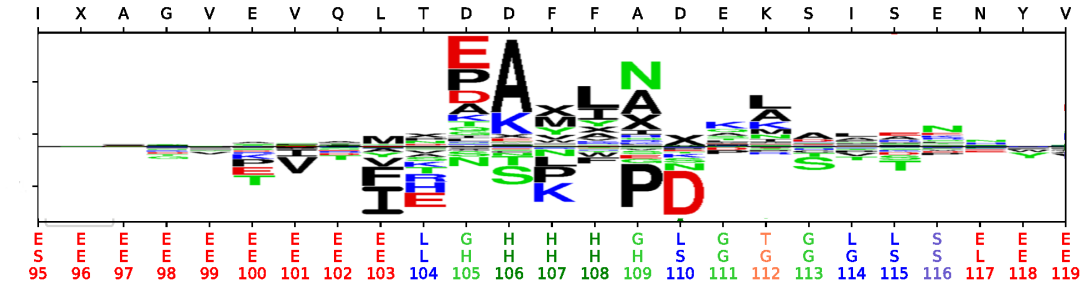

FIGURE 4.5: *Fragment of sequence map for class H in SeqLogo form.* The saliency of the *pssm* values is shown here. The frame and labels of the figure is the same as in Figure 4.4. The image has been generated by *SeqLogo* Thomsen and Nielsen (2012).

Sheer addition

A simple way to aggregate the saliency maps would be adding them up with element-wise addition, thus composing a single saliency map of size 8x42x19. This mega-saliency map should convey rough information of the general features of the network. Since there is no a sense of "left" or "right" in the sequences, the aggregation should be *side-invariant* and therefore an extra *mirroring* operation is applied to the aggregation in order to make the aggregated saliency map symmetric along the centre of the window.

To start with, Figure 4.6 shows the aggregated class-specific saliency maps for the three main classes and the 21 *pssm* values. There are a few details to notice here. First, higher saliency values concentrate around the centre of the window, meaning that most relevant information is located around 5 positions from the predicted one. Second, most *pssm* values preserve the same sign along the window of a class, although a few exceptions also occur. Third, as it could be expected, class L generally presents inverse patterns with respect to classes H and E. This makes sense since coils appear wherever there are no α-helices or β-strands, they are complementary. Lastly, each class has a "feature" *pssm* value that is their best indicator: A for class H; V for E; and P for L.

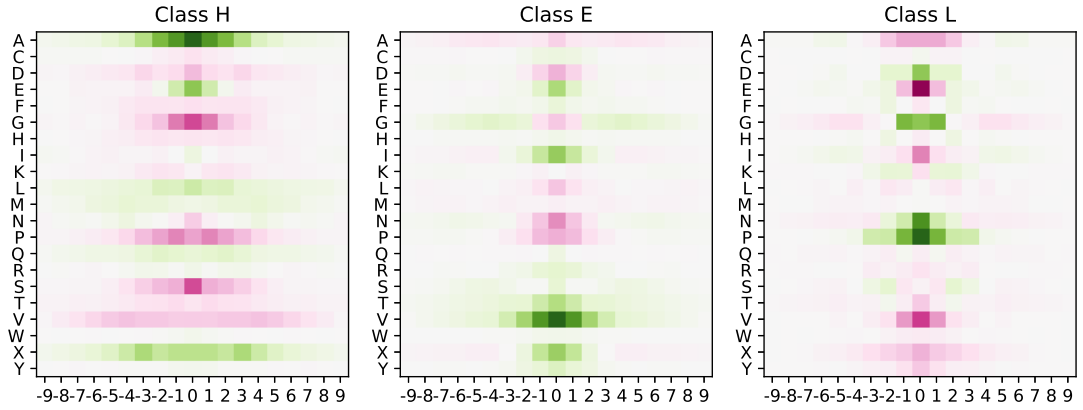

FIGURE 4.6: *Per-class aggregated saliency maps of the main three classes along the window.* Only the saliency values of *pssm* are shown here. Green means positive saliency values and purple means negative.

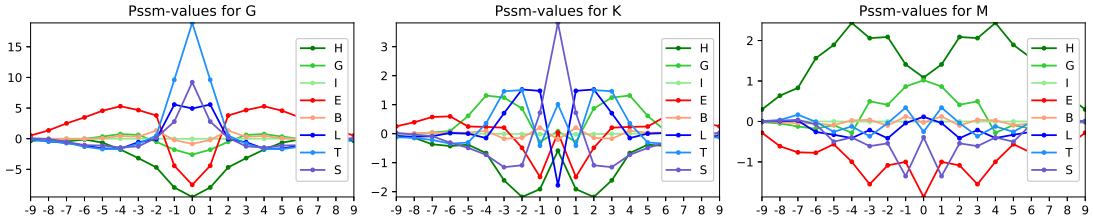

FIGURE 4.7: *Per-class aggregated saliency map of amino-acids G, K, and M, in this order.* *Y* axis is in 1000s. Note the change in scale, particularly for the left-most plot (amino-acid *G*).

Figure 4.7 shows the same information but from a different perspective: as it is *pssm-specific*, it focuses on how a single *pssm*'s position affects the classification on the eight classes. These plots show important effects of the position, as all of them make significant shifts at the centre of the window. *Pssm* value *G* strongly favours coils at its spot, but foresees the presence of β-strands a few positions away. *Pssm* value *K* has a somehow similar behaviour, although with a more intricate ladder of classes as you move further away from the centre. *Pssm* value *M* is generally favouring α-helices, although 3_{10}-helices gain equal importance at the very centre. Note the difference in scales, which indicates that *pssm*-values of *G* are in any case more influential than the other two.

We can use the aggregated saliency maps to investigate on the span of the class influence, i.e. how much of the further away positions the network uses to predict the class. For performing this analysis the addition of the saliency maps must be done in absolute value and then aggregated by the *pssm* dimension as well. Figure 4.8 reveals the results for this operation. α-helices (H) and β-bridges (B) seem to have the highest side influence, although β-strand's E influence gets more weight for further positions. Most classes show a quasi-linear descend on influence from one to five-positions away and some tail afterwards. Coils are the most centred classes.

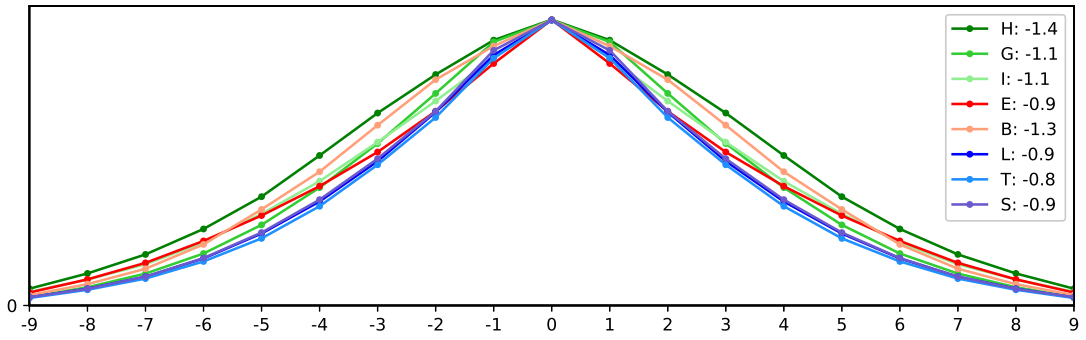

FIGURE 4.8: *Classes overall influence along the window size.* All lines are normalized to have the same hight , as the analysis focuses on the shapes rather than the values. Labels include a measurement of the *kurtosis* of the distributions.

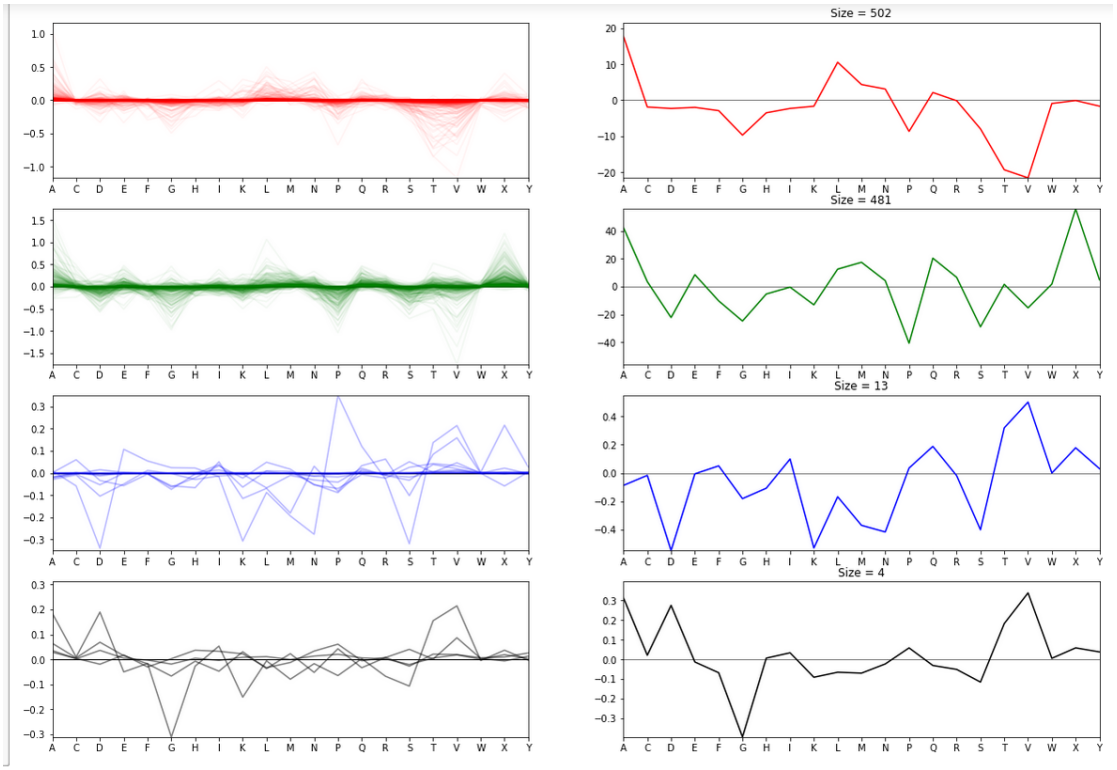

FIGURE 4.9:

Clustering techniques

aggregate all individual saliency maps

4.4 Final discussion

- Problem with Q8 as a not-so-accurate schema. Better with angles.
- Shouldn't types of errors have different costs? It is as bad to miss a Q8 prediction that was in the same Q3 prediction class

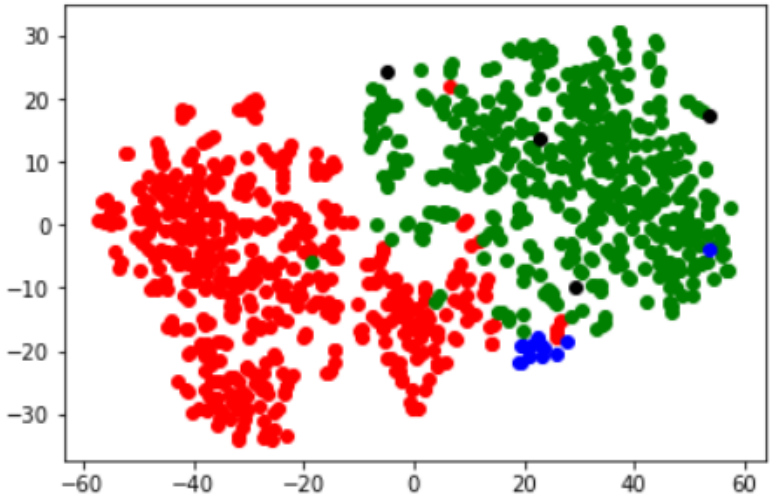

FIGURE 4.10:

Chapter 5

Conclusions

Bibliography

Babak Alipanahi, Andrew Delong, Matthew T. Weirauch, and Brendan J. Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, 2015. ISSN 15461696.

Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, 1997. ISSN 03051048.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 2015. ISSN 19326203.

Helen Berman, Kim Henrick, and Haruki Nakamura. Announcing the worldwide Protein Data Bank, 2003. ISSN 10728368.

Akosua Busia and Navdeep Jaitly. Next-Step Conditioned Deep Convolutional Neural Networks Improve Protein Secondary Structure Prediction. *arXiv:1702.03865v1*, 2017.

Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Wei Xie, Gail L. Rosen, Benjamin J. Lengerich, Johnny Israeli, Jack Lanchantin, Stephen Woloszynek, Anne E. Carpenter, Avanti Shrikumar, Jinbo Xu, Evan M. Cofer, David J. Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K. Wiley, Marwin H. S. Segler, Anthony Gitter, and Casey S. Greene. *Opportunities And Obstacles For Deep Learning In Biology And Medicine*. 2017. ISBN 0000000305396.

Peter Y. Chou and Gerald D. Fasman. Prediction of Protein Conformation. *Biochemistry*, 1974. ISSN 15204995.

J a Cuff and G J Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins*, 1999. ISSN 0887-3585.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De

Fauw, Michael Heilman, Diogo149, Brian McFee, Hendrik Weideman, Takacsg84, Peterderivaz, Jon, Instagibbs, Dr Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release., 2015.

Ken A. Dill and Justin L. MacCallum. The protein-folding problem, 50 years on, 2012. ISSN 10959203.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. mar 2016.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Bernoulli*, 2009.

C. Fang, Y. Shang, and D. Xu. A new deep neighbor residual network for protein secondary structure prediction. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 66–71, Nov 2017a.

Chao Fang, Yi Shang, and Dong Xu. MUFold-SS: Protein Secondary Structure Prediction Using Deep Inception-Inside-Inception Networks. sep 2017b.

Jean-Luc Fauchère, Marvin Charton, Lemont B Kier, Arie Verloop, and Vladimir Pliska. Amino acid side chain parameters for correlation studies in biology and pharmacology. *International Journal of Peptide and Protein Research*, 1988. ISSN 0367-8377.

Alex Finnegan and Jun S. Song. Maximum entropy methods for extracting the learned features of deep neural networks. *PLoS Computational Biology*, 2017. ISSN 15537358.

Alejandro Fontal. *Neural Networks for Subcellular Localization Prediction*. PhD thesis, Wageningen University & Research, 2017.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent Advances in Convolutional Neural Networks. *Arxiv*, 2017. ISSN 16641078.

Leandro Takeshi Hattori, Cesar Manuel Vargas Benitez, and Heitor Silverio Lopes. A deep bidirectional long short-term memory approach applied to the protein secondary structure prediction problem. In *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6, 2017. ISBN 978-1-5386-3734-0.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. ResNet. *arXiv preprint arXiv:1512.03385v1*, 2015. ISSN 1664-1078.

Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017. ISSN 14602059.

Alexander Rosenberg Johansen, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Deep Recurrent Conditional Random Field Network for Protein Secondary Prediction. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics - ACM-BCB '17*, 2017. ISBN 9781450347228.

D. T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.*, 1999. ISSN 0022-2836.

William Jones, Kaur Alasoo, Dmytro Fishman, and Leopold Parts. Computational biology: deep learning. *Emerging Topics in Life Sciences*, 1(3):257–274, 2017. ISSN 2397-8554.

Vanessa Isabell Jurtz, Alexander Rosenberg Johansen, Morten Nielsen, Jose Juan Almagro Armenteros, Henrik Nielsen, Casper Kaae Sønderby, Ole Winther, and Søren Kaae Sønderby. An introduction to deep learning on biological sequence data: Examples and solutions. *Bioinformatics*, 33(22):3685–3690, 2017. ISSN 14602059.

Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogenbonded and geometrical features. *Biopolymers*, 22 (12):2577–2637, 1983. ISSN 10970282.

David R. Kelley, Jasper Snoek, and John L. Rinn. Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, 2016. ISSN 15495469.

Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep Motif Dashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks. 2016. ISSN 23356936.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. ISSN 00189219.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 2009. ISBN 9781605585161.

Zhen Li and Yizhou Yu. Protein Secondary Structure Prediction Using Cascaded Convolutional and Recurrent Neural Networks. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016. ISBN 978-1-57735-770-4.

Zeming Lin, Jack Lanchantin, and Yanjun Qi. MUST-CNN: A Multilayer Shift-and-Stitch Deep Convolutional Architecture for Sequence-based Protein Structure Prediction. may 2016.

Christophe N. Magnan and Pierre Baldi. SSpro/ACCpro 5: Almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning and structural similarity. *Bioinformatics*, 2014. ISSN 14602059.

Mufti Mahmud, Mohammed Shamim Kaiser, Amir Hussain, and Stefano Vassanelli. Applications of Deep Learning and Reinforcement Learning to Biological Data, 2018. ISSN 21622388.

Polina Mamoshina, Armando Vieira, Evgeny Putin, and Alex Zhavoronkov. Applications of Deep Learning in Biomedicine, 2016. ISSN 15438392.

Gregoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2015. ISSN 2329-9290.

Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics, 2017. ISSN 14774054.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus Robert Müller. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 2017. ISSN 00313203.

Grégoire Montavon, Wojciech Samek, and Klaus Robert Müller. Methods for interpreting and understanding deep neural networks, 2018. ISSN 10512004.

Alexander Mordvintsev, Michael Tyka, and Christopher Olah. Inceptionism: Going deeper into neural networks, google research blog, 2015.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. feb 2016.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2017. ISSN 2476-0757.

L. Pauling, R. B. Corey, and H. R. Branson. The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain. *Proceedings of the National Academy of Sciences*, 1951. ISSN 0027-8424.

Ning Qian and Terrence J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 1988. ISSN 00222836.

Daniel Quang and Xiaohui Xie. DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, 2016. ISSN 13624962.

Burkhard Rost. Review: Protein secondary structure prediction continues to rise, 2001. ISSN 10478477.

Burkhard Rost and Chris Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 1993. ISSN 00222836.

Google Sergey Ioffe and Google Christian Szegedy. Batch Normalization. *Icml*, 2015. ISSN 10282092.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. apr 2017.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. *arXiv:1605.01713 [cs]*, 2016. ISSN 1938-7228.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv.org*, 2014.

Søren Kaae Sønderby and Ole Winther. Protein Secondary Structure Prediction with Long Short Term Memory Networks. *arXiv:1412.7828*, 2014.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. dec 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014. ISSN 15337928.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. 2017. ISSN 1938-7228.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon

Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, may 2016.

Martin Christen Frolund Thomsen and Morten Nielsen. Seq2Logo: A method for construction and visualization of amino acid binding motifs and sequence profiles including sequence weighting, pseudo counts and two-sided representation of amino acid enrichment and depletion. *Nucleic Acids Research*, 40(W1), 2012. ISSN 03051048.

Ramzan Kh Umarov and Victor V. Solovyev. Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLoS ONE*, 12(2), 2017. ISSN 19326203.

Guoli Wang and Roland L. Dunbrack. PISCES: A protein sequence culling server. *Bioinformatics*, 2003. ISSN 13674803.

Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu. Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields. *Scientific Reports*, 6, 2016. ISSN 20452322.

Zhiyong Wang, Feng Zhao, Jian Peng, and Jinbo Xu. Protein 8-class secondary structure prediction using conditional neural fields. *Proteomics*, 2011. ISSN 16159853.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014. ISBN 9783319105895.

Jian Zhou and Olga G. Troyanskaya. Deep Supervised and Convolutional Generative Stochastic Network for Protein Secondary Structure Prediction. 2014.

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 12(10):931–934, 2015. ISSN 15487105.

Jiyun Zhou, Hongpeng Wang, Zhishan Zhao, Ruifeng Xu, and Qin Lu. CNNH_PSS: Protein 8-class secondary structure prediction by convolutional neural network with highway. *BMC Bioinformatics*, 2018. ISSN 14712105.