Neural Networks for Subcellular Localization Prediction

MASTER THESIS

by

Alejandro Fontal

submitted to obtain the degree of

MASTER OF SCIENCE (M.SC.)

at

WAGENINGEN UNIVERSITY & RESEARCH

Course of Studies

BIOINFORMATICS

First supervisor: Dr. Aalt-Jan VAN DIJK
Wageningen University & Research

Second supervisor: Prof. Dr. Ir. Dick DE RIDDER
Wageningen University & Research

Wageningen, December 2017

Contact details: Alejandro FONTAL
Dijkgraaf 4, 12C9
67008PG Wageningen
alejandro.fontal@wur.nl

Dr. Aalt-Jan VAN DIJK
Wageningen University & Research
Mathematical and Statistical Methods - Biometris
107/W4.Aa.033
Droevendaalsesteeg 1
6708PB Wageningen
aaltjan.vandijk@wur.nl

Prof. Dr. Ir. Dick DE RIDDER
Wageningen University & Research
Department of Plant Sciences - Bioinformatics Subdivision
107/W1.Bc.054
Droevendaalsesteeg 1
6708PB Wageningen
dick.deridder@wur.nl

Contents

List of Figures

List of Tables

List of Abbreviations

NN	Neural Network
FC	Fully Connected
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
TL	True Labels
PL	Predicted Labels
TP	True Positives
FP	False Positives
TN	True Negatives
FN	False Negatives
CH	Chloroplast
LS	Lysosomal
NC	Nuclear
MT	Mitochondrial
ER	Endoplasmic Reticulum
VC	Vacuolar
PX	Peroxisomal
GG	Golgi
EC	Extracellular
PM	Plasma Membrane
CY	Cytoplasmic

Chapter 1

Introduction

The continuously decreasing cost of sequencing genomes is promoting the generation of vast amounts of sequence data. In consequence, the need to develop automated methods of analysis for this kind of data is on the rise. A first step towards inferring a protein's function is to know its subcellular location. Being able to reliably do so from just the amino acid sequence would be a powerful method to automatically label unannotated sequence data. As such, protein subcellular location prediction has received a great deal of attention from the bioinformatics field [27, 28]. Several methods have been developed to tackle the issue. These can be classified in three main groups of approaches:

- Prediction by recognizing the actual sorting signals i.e. signal peptides [26] or transmembrane α-helices [24]. Using knowledge about the sorting signals, these methods look for them in the sequences. The main disadvantage is that they can only detect those signals that have been previously annotated. In consequence, they do not work for compartments whose sorting signals are yet to be discovered.

- Prediction by sequence homology. When trying to predict the function of an unknown protein, the standard procedure is to look for homologue sequences and then infer that they share functional annotations. Likewise, it is expected that closely related proteins stay in the same subcellular compartment [25]. This has been exploited in a variety of tools [6, 22].

- Prediction based on global properties of the proteins, such as their amino acid composition and features derived from them. The main advantage to using these kind of approaches is that they can be used for poorly annotated sequences or subcellular compartments for which the sorting signals are unknown. Most developed tools of this type use machine learning based methods [31, 13].

Machine learning is a field inside the discipline of artificial intelligence based on the concept that machines should be able to learn, adapt, and improve through experience. Machine learning methodologies have been widely used in several domains of bioinformatics [18]. Deep learning [19, 32], a branch of machine learning, consists of approaches that use deep neural networks in order to transform a series of inputs/features into the desired outputs. This is done via propagation of the data through a series of layers that represent different transformations of the original inputs. Deep learning approaches have proved to almost always be able to perform at the same or better level as state-of-the-art methods on specific tasks. One of the main advantages of neural networks is that they act as universal approximators: given any relationship between some input features and an output, there is a neural network that can capture it [12]. The fact that such a network exists, though, does not mean that it is easily achievable to unveil its architecture, nor its proper hyperparameters. That is one

of the main drawbacks and points of criticism that neural networks and deep learning receive: while their ability and potential to predict complex and abstract relationships among features is extremely powerful, the time to fine tune its hyperparameters and architecture make it a tedious and non-trivial task. Another issue that is specially relevant in the context of bioinformatics is the need for large amounts of labeled training data, which is typically difficult to find and expensive to generate. Given the complexity and non-linearity of deep learning algorithms, they usually require larger amounts of training data to fully exploit their predictive capabilities compared to simpler machine learning approaches.

The use of deep learning in bioinformatics [23], while not as extended as in the fields of natural language processing, image processing or speech recognition, has been successfully implemented in areas such as genomics, proteomics, or biomedical signal processing. Several efforts for using combinations of neural network architectures [40, 21] in order to predict protein function have been made with good results.

Neural networks look like a promising method to predict the subcellular location of proteins based on just their amino acid sequence. The ability to capture and learn patterns in the data ought to make them detect the sorting signals without the need to guide them to do so. Hence, a well-enough built and trained neural network model should be able to combine the best of the methods that include the sorting signals data in their predictions while only using the raw sequence data. Some work with neural networks has already been successfully applied for this purpose [37, 21].

The main barrier that prevents deep learning approaches to be fully implemented and more widely used in most fields is the fact that they act like *black boxes*, achieving great performances but doing so in an indecipherable way. While this might not be a major issue in some cases, it becomes crucial in applications where interpretability is essential. In the biomedical field, for instance, it is hard for a doctor to diagnose a patient based on some model that, even if apparently performs greatly, he cannot interpret nor fully understand. In the context of subcellular location prediction, an interpretable model would be one that not only has a good prediction performance, but that provides insights about the learned features and could potentially lead to the discovery of new sorting signals. As such, there is a growing interest in developing methods to open the *black box* that neural networks represent in many fields [34, 30, 29] and specifically in bioinformatics [36, 17].

The aim of this project is to (i) apply deep learning to predict protein subcellular localization and (ii) "open the black box" i.e. use the resulting models to learn which properties of sequences influence subcellular localization. In order to do so, several neural networks comprising different architectures have been built and trained. In an attempt to extract meaningful conclusions from the trained models, methods have been either created or adapted so as to interpret them.

Chapter 2

Materials & Methods

2.1 Framework

The whole project has been written in Python 2.7, with the code available in: https://github.com/AlFontal/Thesis.

The TensorFlow [1] library has been extensively used both for building, training and evaluating the NN models.

2.2 Dataset

The dataset used is the MultiLoc [11] Dataset. This dataset was obtained by extracting all animal, fungal and plant protein sequences from the SWISS-PROT database release 42, using the keywords *Metazoa, Fungi* or *Viridiplantae* in the organism classification field. These proteins were assigned to 1 of 11 possible subcellular localizations based on the annotation in the CC (comments) field.

Plant proteins can be localized in the chloroplast, cytoplasm, endoplasmic reticulum, extracellular space, Golgi apparatus, mitochondrion, nucleus, peroxisome, plasma membrane and vacuole. Fungal cells share the same subcellular localizations as plant cells, except that they lack the chloroplast. Finally, animal cells share all localizations with fungal cells, but have lysosomes instead of vacuoles.

TABLE 2.1: Number of sequences for each localization

Localization	Complete	Reduced
CH	730	449
CY	2768	1411
ER	328	198
EC	1124	843
GG	244	150
LS	164	103
MC	872	510
NC	1040	837
PX	278	157
PM	2115	1238
VC	98	63
Total	9761	5959

A total of 9761 sequences were extracted, with no restrictions on the level of homology at this point. In order to train prediction models, using datasets containing way

too high similarity will lead to recognition of nearly identical sequences. This could therefore inflate the accuracy estimations, disrupting the proper optimization of the model via the loss function. Consequently, a homology-reduced dataset was created by removing proteins from the original dataset until it contained no sequences with a pairwise similarity >80% using the ClustalW [41] algorithm. The reduced dataset ended up containing 5959 sequences. The specific class distribution can be seen in Table 2.1.

2.3 Data Processing

2.3.1 Sequence Length

In order to make the sequences a valid input for the TensorFlow models, they all need to be of the same length. The specific sequence length was left as one hyperparameter more of the models, which could be tuned to the specific needs of each of them. However, in order to trim the sequences longer than the desired length and pad the shorter ones, a position in the sequence needed to be chosen. Since it is known that most sorting signals are located near the C- and N- terminus of the protein sequences [7], it was decided to either pad or trim the sequences in the middle. For a desired sequence length of n and a sequence length of l, the process would be the following:

- If $l > n$, the sequence is trimmed in the middle, only keeping $n/2$ amino acids from each of the ends of the sequence.
- If $l < n$, the sequence gets added $n - l$ X's at position $l/2$.

The X's are defined in order to contain no signal.

2.3.2 Sequence to Tensor Transformation

The encoding of the sequences into tensors was done using variations of the methods used in [40] and [37]. The standard input of the models was a 3D tensor of the following shape: `[minibatch_size, sequence_length, aa_features]`.

`minibatch_size` is the amount of sequences introduced in the network at each training step and is considered a hyperparameter. `sequence_length` is the length of the processed sequences. `aa_features` is the number of features which are encoded for each of the amino acids of the sequence. 3 variations were used:

- A one-hot vector with 20 components (one for each of the amino acids). In a one-hot vector all components are zero except the one identifying the amino acid in question.
- Amino acid properties. 6 components describing chemical properties of the amino acids: `charge`, `hydrophobicity` and the binary attributes `isPolar`, `isAromatic`, `hasHydroxyl` and `hasSulfur`.
- The values for each amino acid in the BLOSUM80 substitution matrix [9]. This accounts for 20 more features

X's are defined as a vector of `aa_features` zeros. Combinations of these three components were used in order to transform each amino acid position in the sequences into a vector. Each of the sequences can therefore be visualized as a matrix of `sequence_length` columns and `aa_features` rows.

2.3.3 Training and Test Sets

Before training every model, 80% of the sequences are assigned to the training set which is used to train the network and the 20% remaining sequences are used to test its performance. Given the huge class imbalance present in the dataset, this 80/20 proportion was not left totally randomized but constrained in order to keep the real balance of the classes in both sets. That is, the 80/20 division in the assigning is done in a class per class basis, thus keeping the same class distribution in train and test sets.

In the same way that overfitting can occur for the weights and biases of the network for the training set, the optimization of the hyperparameters of the network can overfit for the test set. In order to avoid this, a validation set which is not used to optimize the hyperparameters can be used so as to validate the actual performance of the model without any kind of bias. This methodology works greatly with large datasets, however, given that the used dataset was rather small, it was decided to not use a validation set. With the aim to avoid hyperparameter overfitting for a certain test set, though, randomization of the test/train selection is done before the training of every new model, thus ensuring no bias in this setting.

2.4 Neural Networks' Set-up

All the predictive models built consisted of variations of neural network architectures, in particular using fully-connected (FC), convolutional (CNN) and long short-term memory (LSTM) layers. Before explaining details about these architectures (sections 2.4.1 to 2.4.3) general characteristics of the neural networks are introduced.

Loss Function

All neural networks need to compute how well they perform. This value, called the *cost* or *loss* function, is the value that is optimized via gradient descent by tuning all the weights(W) and biases(B) in the network. The loss function chosen is the most commonly used in classification tasks, the cross entropy. The cross entropy can be considered a measure of the difference between two probability distributions.

Formally, if C denotes the number of classes, S the network score and N the mini-batch size:

$$S = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\hat{y}_{ij}(-\log y_{ij}) + (1-\hat{y})(-\log\,(1-y_{ij}))$$

Where $\hat{y}_{ij}$ is the experimental probability (so either a 0 or a 1) of the j_{th} class of the i_{th} sequence in the mini-batch while y_{ij} is the predicted probability of the j_{th} class of the i_{th} sequence in the mini-batch.

Softmax Transformation

The neural networks used for classification tasks apply a series of transformations to an input (X) which can be expressed as a highly non-linear function $f(X)$. This provides a vector P containing a probability for each of the classes in the model. If not forced, $f(X)$ will not output a vector of probabilities P but just an output vector O with values without any kind of constraint. In order to transform the output vector O into the probabilities vector P, the Softmax function is used. Assuming that O is

a one-dimensional vector with K values, the Softmax transformation consists of the following:

$$P = \text{Softmax}(O) \rightarrow P_j = \frac{\exp(O_j)}{\sum_{k=1}^{K} \exp(O_k)}$$

P can then be used to compute the cross entropy.

Optimizing Algorithm

The algorithm used to iteratively update the weights and biases of the network in order to minimize the loss function is the Adam optimizer [15]. It was chosen instead of the classical stochastic gradient descent (SGD)[20] given its computational efficiency and adaptability. While classical SGD maintains a single learning rate for all weight updates which does not change during training, Adam keeps a different learning rate for each of the parameter and specifically adapts it as the training advances. While an initial learning rate must be defined, given its adaptability of the algorithm it becomes drastically easier to fine tune this hyperparameter.

Dropout

In order to reduce the amount of overfitting, dropout [39] is used in certain layers during the training of the networks. Essentially, the use of dropout addresses the problem of overfitting by randomly generating thinned versions of the total network where certain nodes and its connections are "dropped", so not used, at each train step. Effectively, the complete network is in a certain way an averaged model of all the thinned networks used along the training steps. This introduces noise and reduces the capability of the network to adapt too much to the training data, thus decreasing the overfitting.

Activation Functions

All the units in a neural network layer need to use a certain activation function in order to transform their inputs into a reasonable output. Three different activation functions are used in the prediction models built in the project:

- Logistic (Sigmoid): $\sigma(x) = \dfrac{1}{1 + e^{-x}}$
- Hyperbolic Tangent: $tanh(x) = \dfrac{2}{1 + e^{-2x}} - 1$
- Leaky ReLU: $f(x) = \alpha x$ for $x < 0$; x for $x \geq 0$ $\quad (\alpha = 0.1)$

2.4.1 Fully Connected Network

The final fully connected Network, whose structure can be seen in Figure 2.1, is composed of the following architecture:

- The input consists of a sequence with a fixed length of 500 amino acids with one-hot encoding + properties as features, yielding 26 features for each one of the 500 amino acids.
- The input is fed to a first fully connected layer of 200 units with a Leaky Relu with $\alpha = 0.1$ as activation function. This layer used 40% dropout during training.
- The output of the first fully connected layer is fed to a second fully connected layer of 11 units and Leaky ReLu with $\alpha = 0.1$ as activation function. This layer did not use dropout.
- The output of the second fully connected layer is transformed via Softmax to a vector of probabilities for each of the classes.

The initial learning rate was 0.02 and the mini-batch size was 500.
The output from each of the fully connected layers is calculated by the following equations:

$$\text{Out}(\text{fc}_{1,i}) = \text{LeakyReLu}\left(\sum_{j=1}^{13000} x_j \cdot W_{1,i,j} + b_{1,i}\right) \quad i \text{ in } [1, 200]$$

$$O_i = \text{LeakyReLu}\left(\sum_{j=1}^{200} \text{Out}(\text{fc}_{1,j}) \cdot W_{2,i,j} + b_{2,i}\right) \quad i \text{ in } [1, 11]$$

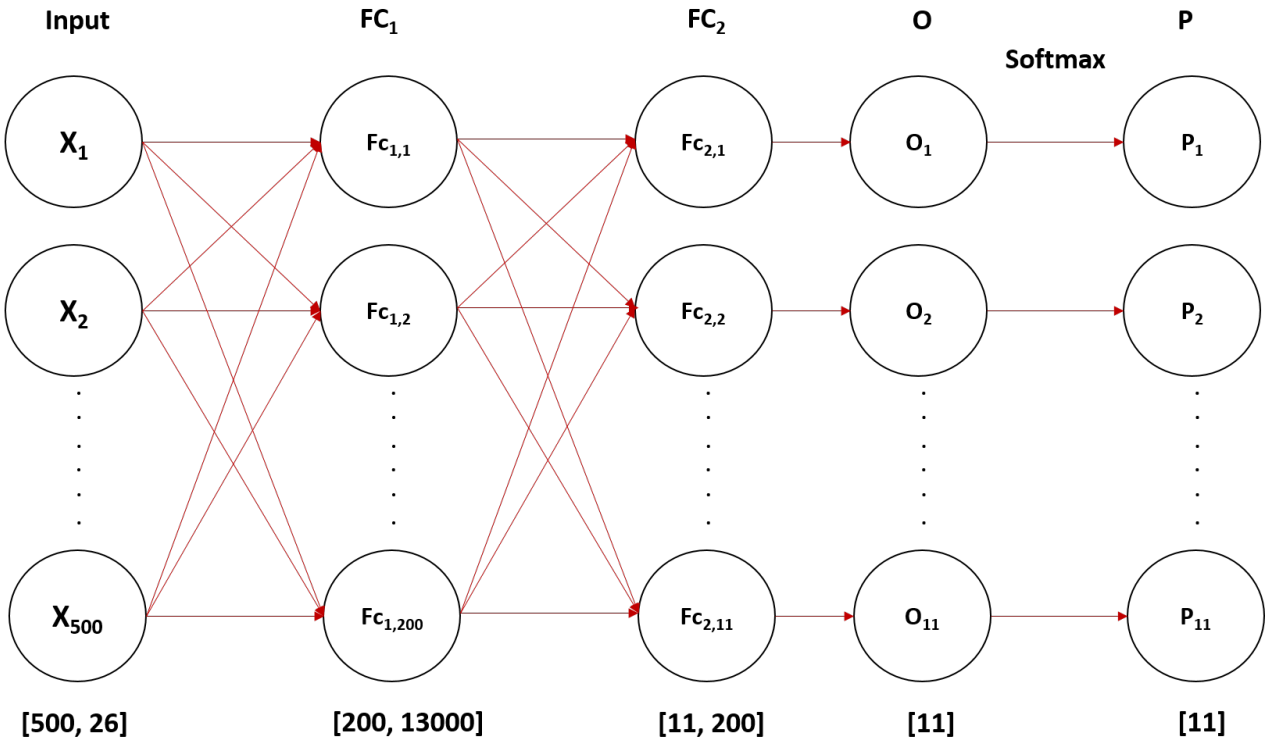


FIGURE 2.1: Structure of the final FC model

2.4.2 Convolutional Network

The final convolutional neural network, whose structure can be seen in Figure 2.2, is composed of the following architecture:

- The input consists of a sequence with a fixed length of 300 amino acids with one-hot encoding + properties as features, yielding 26 features for each one of the 300 amino acids.
- The first layer is a convolution layer consisting of 75 units or filters. 25 of these filters are of shape `[5, 26]`, 25 of shape `[10, 26]` and 25 of shape `[15, 26]`. All of them with a stride of 1 and "SAME" padding. Leaky ReLu with $\alpha = 0.1$ is used as the activation function.
- The second layer is a Max Pooling layer which only outputs the maximum value on a window of 5 positions along the tensors. The stride used is 5 and "SAME" padding. This reduces by a factor of 5 the dimensionality of the output tensors.
- The outputs of the pooling layer are concatenated and fed to a fully connected layer with 100 units with a Leaky Relu with $\alpha = 0.1$ as activation function. This layer used 40% dropout during training.
- The output of the first fully connected layer is fed to a second fully connected layer of 11 units and Leaky Relu with $\alpha = 0.1$ as activation function. This layer also used 40% dropout during training.
- The output of the second fully connected layer is transformed via Softmax to a vector of probabilities for each of the classes.

The initial learning rate was 0.02 and the mini-batch size 500.

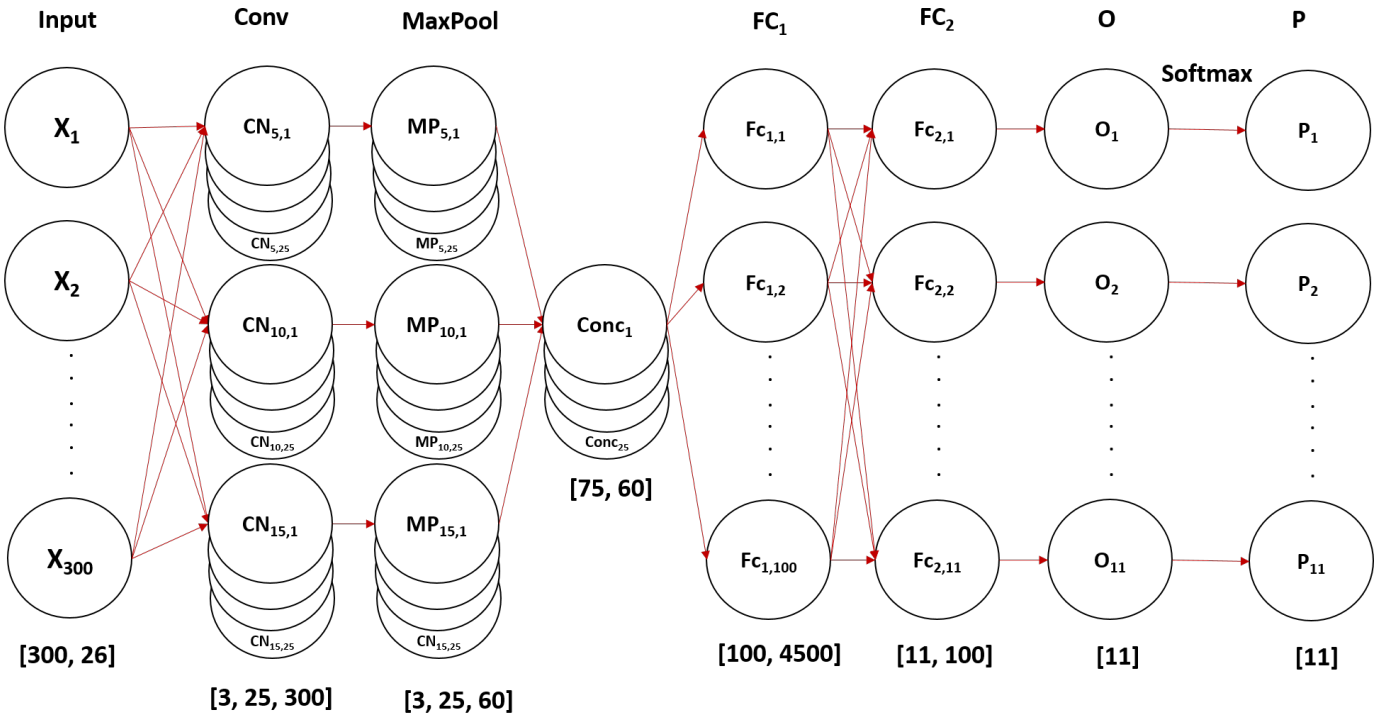


FIGURE 2.2: Structure of the final CNN model

2.4.3 Long Short-term Memory Network

The final model built was a bidirectional LSTM [10] network. The basic cells which make up the LSTM layers consist of the following structure:

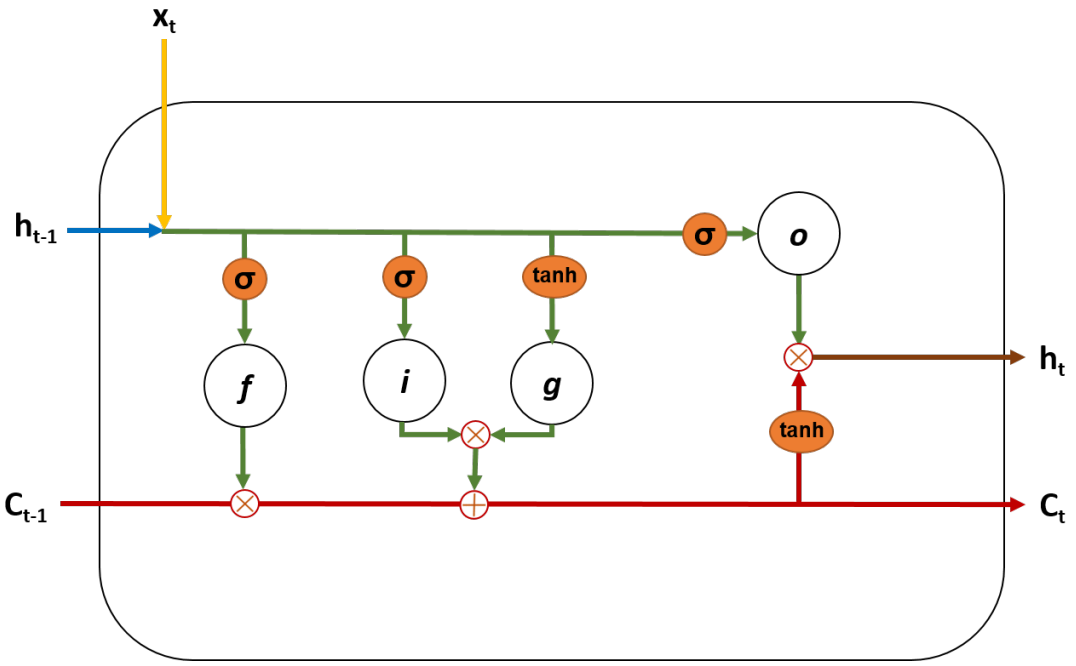


FIGURE 2.3: Structure of LSTM cells

As seen in Figure 2.3, each LSTM cell receives 3 inputs (x_t, C_{t-1} and h_{t-1}) and yields 2 outputs (C_t and h_t). C is the long term memory cell and is responsible for keeping the relevant information along the whole sequence. h is the short-term output of each cell, and in the implemented model, the values of h in the last cell of each LSTM layer are the outputs fed to the deeper layers. x is the input corresponding to the current step.

At each LSTM cell, the output of the previous cells h_{t-1} and the information from the step t, x_t are combined and go through the *forget gate 'f'*, the *input gate 'i'*, the *input modulation gate 'g'* and the *output gate 'o'*. First, the output from the *forget gate* is applied to the *memory cell 'C'* coming from the previous cell, C_{t-1}, deleting certain information from it. After that, new information from combining the input and input modulation gates is added to it. After this modification, the *memory cell* is ready to be outputted to the following cell. A combination of the *memory cell* at this point and the *output gate 'o'* will yield the short-term output $'h'_t$ for this cell. The equations that control each one of these gates are the following:

$$i_t = \sigma(x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i)$$

$$f_t = \sigma(x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f)$$

$$g_t = \tanh(x_t \cdot W_{xg} + h_{t-1} \cdot W_{hg} + b_g)$$

$$o_t = \sigma(x_t \cdot W_{xo} + h_{t-1} \cdot W_{ho} + b_o)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

($\odot$: Elementwise multiplication σ : Sigmoid activation function)

The fully connected layers used the same units as the ones from the previous models. The architecture of the network consists of the following layers:(Figure 2.4):

- The input consists of a sequence with a fixed length of 750 amino acids with one-hot encoding + properties as features, yielding 26 features for each one of the 750 amino acids.
- The first layer is the forward LSTM layer, which takes the input in the N $\rightarrow$ C direction. The input is divided in 750 steps (each amino acid is taken as one step), each of them consecutively fed to the 750 unrolled LSTM cells. This layer consists of 100 units, which means that a total of 750 x 100 = 75000 unrolled LSTM cells are used.
- In parallel, the input goes through a backwards LSTM layer equal to the forward one but taking the input sequence in the C $\rightarrow$ N direction.
- The outputs of the last cell two LSTM layers are concatenated and fed to a fully connected layer with 250 units with a Leaky Relu with $\alpha = 0.1$ as activation function. This layer used 20% dropout during training.
- The output of the first fully connected layer is fed to a second fully connected layer of 11 units and Leaky Relu with $\alpha = 0.1$ as activation function. This layer also used 20% dropout during training.
- The output of the second fully connected layer is transformed via Softmax to a vector of probabilities for each of the classes.

The initial learning rate was 0.02 and the mini-batch size 500.

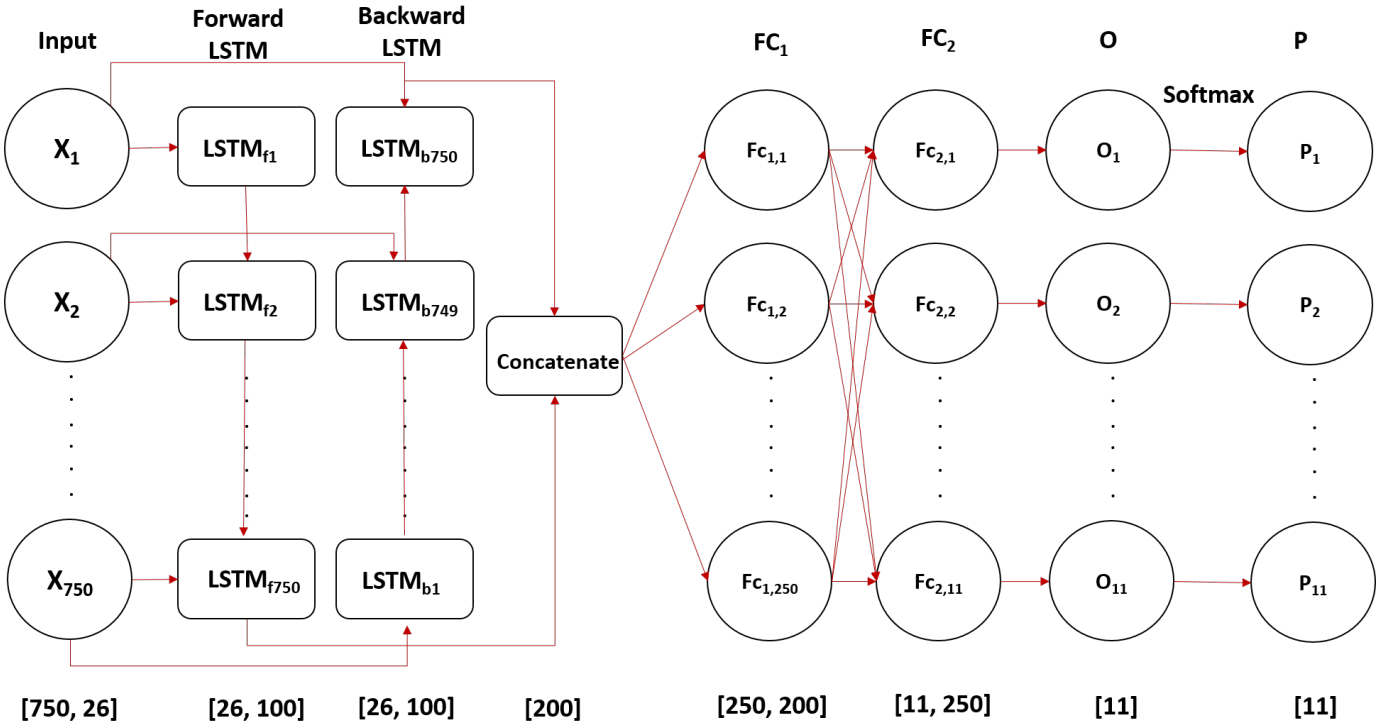


FIGURE 2.4: Structure of the LSTM network

2.5 Model Evaluation

2.5.1 Prediction Performance

In order to assess the performance of the prediction models, mainly 3 measures are evaluated:

- **Classification Accuracy**: $\dfrac{\text{Correct Predictions}}{\text{Total Predictions}}$
- **Class-Specific Precision**: $\dfrac{\text{TP}}{\text{TP + FP}}$
- **Class-Specific Recall**: $\dfrac{\text{TP}}{\text{TP + FN}}$

2.5.2 Convolutional Filters Representation

The networks whose filters were saved and plotted only used one-hot encoding so as to be able to represent all features as PSSM-logos. The visualization of the convolutional filters was based on the methodology used in [37]. The convolutional filters are represented as a matrix of `filter_width` columns and `aa_encoding` rows. So as to visualize the relative importance of each of the positions in the filter, each of them is rescaled in a way that the height of the highest column is 1. After this transformation, each filter can be visualized as a PSSM logo where the position importance is proportional to the height of the column and the height of each letter is proportional to the importance of the amino acid in that position. Seq2logo [42] is used in order to generate the PSSM-logo plots. The Seq2Logo default amino acid colour coding is used: negatively charged (DE) residues are red, polar uncharged residues (NQSGTY) are green, positively charged (RKH) residues are blue and the remaining are black.

2.5.3 Signal Masking Approach

This method was developed along the course of the project in order to evaluate the effect of removing the signal from certain regions of the input sequences. It is a way of assessing, for a trained model, which parts of the input sequence have a higher importance to determine the final classification. This assessment is performed in the following manner:

1. A sequence S is fed as input to a trained NN model. The output layer O (before the Softmax transformation), containing a vector of K values, one for each class, is extracted.

2. The vector O is transformed to a vector of probabilities P via Softmax with a scaling parameter w:

$$P_i = \frac{e^{O_i \cdot w}}{\sum_{k=1}^{K} e^{O_k \cdot w}} \qquad \text{with } w = \frac{1}{\max(O)}$$

   The scaling is performed in order to make the final layer of probabilities more variable and allow for comparison of even subtle changes.

3. The cross entropy of vector P with itself is calculated. Since P is identical to itself, $H(P, P) = H(P)$, so the entropy of P is calculated.

4. Sequence S is modified by substitution of a certain number of amino acid positions by X's, removing any signal from that part of the sequence. This is done for window sizes $ws = 5$ and $ws = 15$. If S_i is the amino acid on position i of sequence S with length l, a modified sequence S^* with a span from S_i^* to S_{i+ws}^* containing only X's is generated, with a total of $l - ws$ modified sequences per original sequence.

5. Each of the modified sequences S^* is fed to the trained NN model and their respective output vectors O^* are extracted and transformed into a vector of probabilities P^* via the same Softmax transformation with scaling weights used for the original sequence.

6. The cross entropy between P and each P^* is computed, and the difference between this value and the entropy of P is stored.

   $H(P, P^*) = H(P) + D_{KL}(P \parallel P^*)$, so by calculating this difference we are actually computing the Kullback-Leibler divergence (D_{KL}) [16] of P^* from P, which is a measure of divergence between two probability distributions. This value is calculated for each of the modified sequences S^*, providing a position-specific score profile.

This approach was used to assess the trained models by calculating an average score for all sequences belonging to each class. The result is a profile which can be used to evaluate the importance of certain regions of the sequences in each of the classes in each of the models.

2.5.4 Class Optimization

This approach, first developed for image classifying CNNs [35] and later adapted to genomic sequences (Deep Motif [17]), consists on the following:

Given a trained classification NN and a class of interest C_i, the objective is to obtain the input X which maximizes the classification score S_i for class C_i. In other words, the approach tries to find the optimal input for a certain class, thus providing valuable information about what features the NN has learned for that specific class.

In the case of the NN trained in this project, where the inputs are protein sequences encoded as one-hot vectors, the optimized inputs lose the one-hot vector encoding but can be forced to behave as a Position Probability Matrix (PPM) which can then be represented as a motif.

Formally, the following equation, where S_i is the score (defined as the unscaled values for class C_i pre Softmax transformation) and X is the input sequence, is optimized:

$$\arg\max S_i(X) + \lambda \parallel X \parallel_2^2$$

λ is the regularisation parameter. The L2-regularization term is introduced in order to minimize the number of significant amino acids per position in the optimized input sequence. The score used is the value of the pre-Softmax layer because the post-Softmax value can be maximized by just minimizing the scores of the other classes, and the objective is to get an ideal input for the desired class, not an input modified to score low in the remaining classes.

The optimization is done via Adam [15] changing only the values of the input sequence. Initial values are all set as $1/20$ to give equal probability to each amino acid in each position. $\lambda = 0.00005$. At the end of each optimization step, the values are forced to stay within a realistic range:

- Negative values are converted to 0.
- The values for each position are standardized in order to make the total sum equal to 1.

After a certain number of optimization steps, the final sequence is recovered and represented as a motif logo using Seq2Logo [42].

Chapter 3

Results & Discussion

The results of this project can be divided in two main phases or sections:

1. The building and fine tuning of neural networks to predict subcellular localization of proteins.
2. The analysis of the trained models, in an attempt to open the *black box* that neural networks are in regards to its interpretability.

3.1 Subcellular Localization Prediction Models

Throughout the length of the project, over 500 models with different sets of hyperparameters and NN architectures were trained. Apart from obtaining increasingly accurate models over time, comparing the results obtained among them already provides some insight regarding the behaviour of the learning process.

3.1.1 Input Features

While only the protein sequence is used to make the predictions, the input features and the exact transformation to a numeric tensor was not a trivial decision.

Sequence Length

Since the input tensors of TensorFlow models need to be of a fixed length, assessing the effect of the sequence length on the model's ability to predict the subcellular localization of the protein was a necessary step. First of all, observing the distribution of the length of the sequences in the dataset (Fig 3.1) allows us to infer the degree of the effect of the processing of the sequences. Ideally, we would like to keep as much information as possible without adding noise.

50% of the proteins in the dataset consist of < 393 amino acids and 80% of them consist of < 800 amino acids. Choosing a short sequence length would mean trimming and losing information from most sequences, while adding little to no noise. Choosing a long sequence length, on the other hand, would imply a barely null loss of information combined with the addition of a significant degree of noise. Longer sequences also translate into more complex models, with higher computational times for training and assessing tasks. The information contained in the sequences is most likely not equally distributed along the whole sequence. Hypothesizing that the most relevant information about the subcellular localization is located near the ends of the protein sequences [7] means that trimming the sequences in the middle would minimize the loss of information caused by this modification.

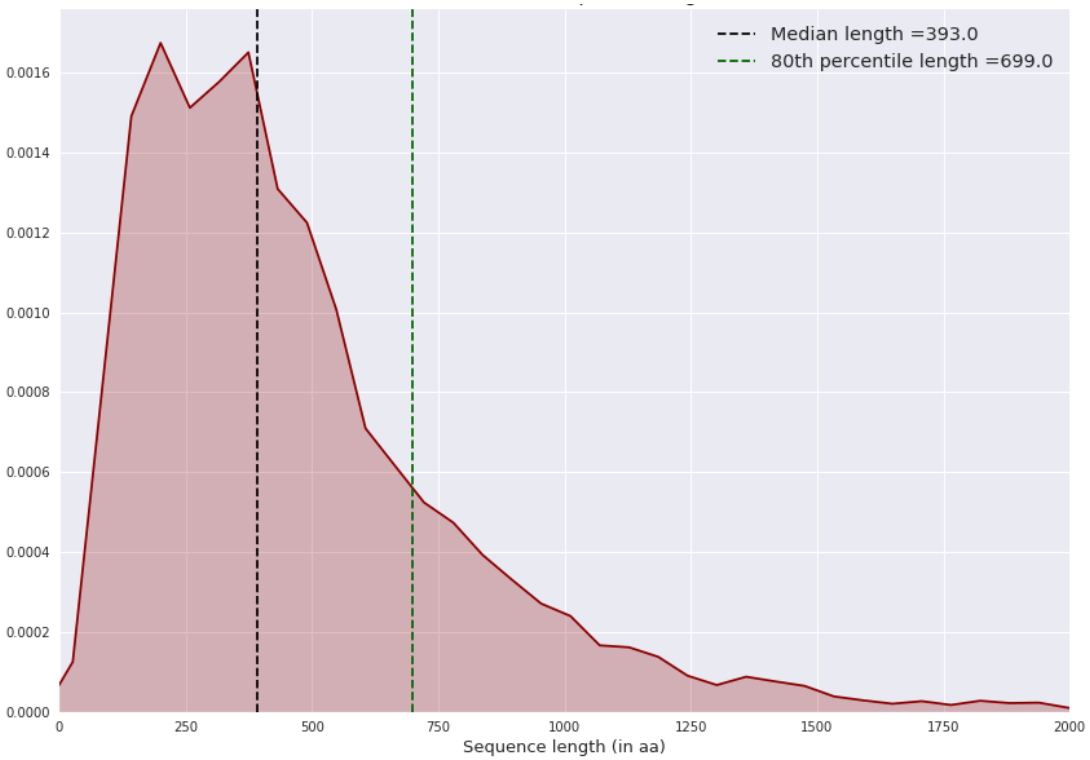


FIGURE 3.1: Distribution of length of sequences in MultiLoc Dataset

In order to assess the previous hypothesis, a series of models with different input sequence lengths were trained with the same hyper parameters: 2 fully connected layers with 100 and 11 units respectively, 40% dropout during training and input features containing 1-hot vector + properties. Two processing methods were used: the first keeps only the ends of the sequences, while the second keeps only the center of the sequences. The test accuracies obtained with each of the models for sequences of lengths from 2 to 100 amino acids are represented in Fig 3.2. The maximum test accuracies for both kinds of models were achieved at a sequence length of 100: 62.5% for those that kept the sequence ends compared to a 38% for those models that only kept the center of the sequences.

The hypothesis of the most relevant information in the sequence being located near the C- and N- terminal ends of the sequences seems to be supported by this results. The models which used the center of the sequences performed worse than the ones keeping the ends at every sequence length, barely increasing their performance with longer sequence lengths.

For the models which kept the ends of the sequences, the performance was drastically better: with a sequence length of 20 amino acids (that is, the first 10 and the last 10 of the whole sequence) the trained models were able to achieve a test accuracy of over 50%. Given that the largest class, *cytoplasm*, accounts for a 23.7% of the total sequences, the results obtained cannot be explained by the model classifying all sequences as cytoplasmic sequences. It is then rather surprising to find that a test accuracy of over 40% is already achieved with a sequence length of just 10 amino acids. Larger sequence lengths (up to 1000) were tested for this configuration, but no strong improvement was detected after sequences of 100 residues long. The best performing model achieved a 63.87% with a sequence length of 500 residues.

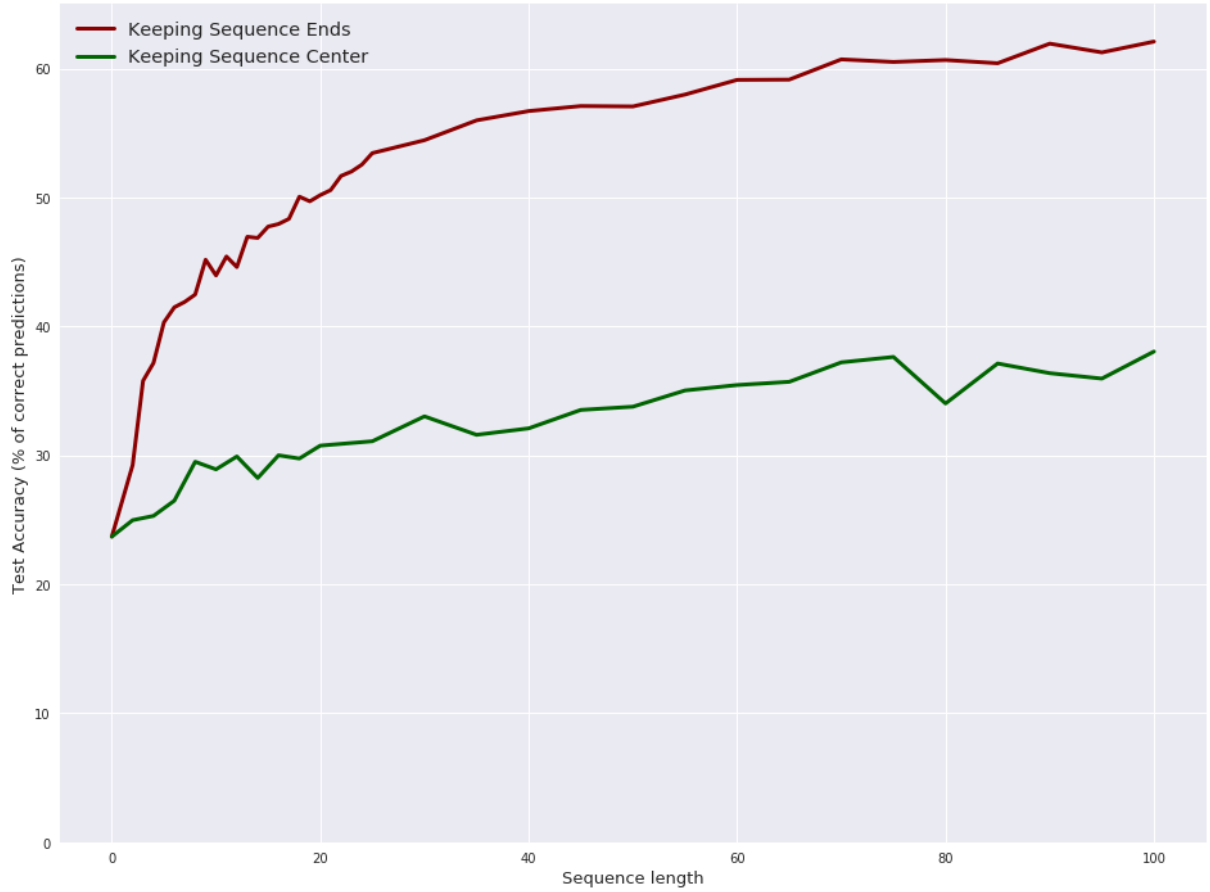


FIGURE 3.2: Test Accuracy obtained at different input sequence lengths

Although these results seem to indicate that most of the signal contained on the sequences are indeed located near the C- and N- terminal ends, the fact that a fully-connected neural network is unable to learn extra information from longer sequences does not mean that more complex networks containing convolutional or RNN layers will not be able to extract valuable information from other regions.

Sequence Preprocessing

Three options were tested in order to transform the amino acid sequences into suitable tensors to be used in TensorFlow. The main idea was to use one-hot encoding to represent each amino acidic position in the sequence, but inspired by [37, 40], the inclusion of more features such as chemical properties of the amino acids and the BLOSUM80 substitution matrix values was also considered.

Table 3.1 shows the average test accuracies obtained with each of the different combination of sequence transformations after multiple runs of trained NN models (Input Sequence length of 500, 2 fully connected layers, with 400/300/200/100 and 11 units respectively, and with 0/20%/40% dropout).

	Properties (6)	**One-Hot (20)**	**One-Hot + Properties (26)**	**One-Hot + Properties + BLOSUM80 (46)**
Average Test Accuracy	50.9%	57.2%	62.5%	56.4%

TABLE 3.1: Average test accuracy for different input types
Between parentheses: Number of features per position

The best performing models were those which had an input consisting of a one-hot vector of the amino acids plus the chemical properties (62.5% average test accuracy). The amino acids one-hot vector alone was the second best input type at a 57.2% average test accuracy. The addition of the BLOSUM80 substitution matrix values did not seem to provide any useful information to the networks, making them perform even worse (56.4% average test accuracy) than the one-hot vector alone, probably due to the effects of high redundancy of the features.

The conclusion derived from these results was to use the one-hot vector + properties input for further improving the prediction models based on the consistent improved performance shown compared to the other options. However, models built simply with a one-hot vector for the amino acid alphabet, albeit potentially less accurate, provide a higher degree of interpretability, and were therefore also used in further experiments with the objective of unveiling the *black box* (Section 3.2).

3.1.2 Prediction Performance

The main objective of this section is to compare the performance of the different neural network architectures used. In order to do so, the metrics used will be the global test accuracies and the class-specific precision and recall. It is important to note that comparing performance among different neural network models can be rather unfair. A mediocre model which has undergone an extensive and successful hyperparameter optimization process could outperform a better suited model with less finely-tuned hyperparameters. As such, it is wise to be careful at stating that one model is better than the other given that, when differences are small, it is rather hard to distinguish whether the improvement in performance is originated in an actually better suited model or just a better tuned model.

The hyperparameter optimization process can be rather time consuming, more so when the architecture of the model is considered a hyperparameter by itself. In consequence, and since the objective was not just to get the best performing model but also to develop some methods to interpret it, this process was done more exhaustively for the simpler models (the fully-connected networks) than for the later ones. It is therefore quite likely that the performances of the LSTM and convolutional models could be further improved if they underwent a more extensive hyperparameter search.

Fully Connected Networks (FC)

The FC networks, given their simplicity in comparison to the other type of networks, allowed for a longer fine-tuning of its parameters given their short training time. While this is an advantage towards training, the fact that it does not take into account the sequential nature of the inputs also means that, often, the FC networks cannot perform at the level the other architectures can.

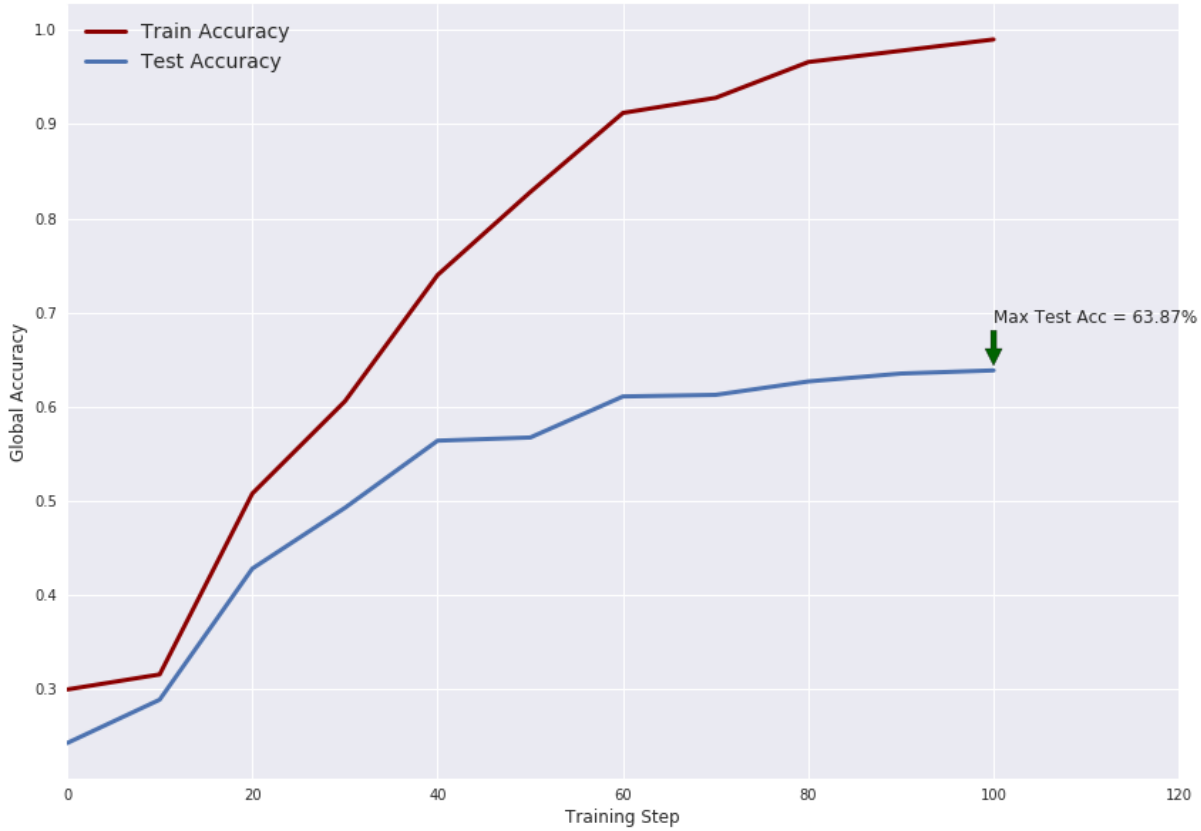


FIGURE 3.3: Accuracy progression during FC model training

For the best performing FC model, as Figure 3.3 depicts, the training process only took about 100 steps ($\approx$ 9 epochs) until the training accuracy increased over 98%. The maximum test accuracy, which was of 63.87%, was achieved at the last training step. There is a clear divergence between the training ant test accuracies that is visible from the 20th step and only increases until the end of the training process.

TABLE 3.2: Confusion matrix for fully connected model

TL / PL	**CH**	**LS**	**NC**	**MT**	**ER**	**VC**	**PX**	**GG**	**EC**	**PM**	**CY**	**Recall**
CH	**69**	0	0	10	0	0	0	0	0	5	6	76.7%
LS	1	**5**	1	1	1	0	0	0	2	10	0	23.8%
NC	2	0	**79**	0	0	0	0	0	1	7	78	47.3%
MT	16	1	9	**47**	0	0	0	0	2	3	24	46.1%
ER	0	1	1	2	**19**	0	0	1	6	7	3	47.5%
VC	1	0	0	0	1	**1**	0	0	2	5	3	7.7%
PX	2	0	4	1	0	0	**1**	0	1	6	16	3.2%
GG	1	0	2	1	0	0	0	**13**	3	7	3	43.3%
EC	0	0	5	3	0	0	0	1	**126**	23	11	74.6%
PM	3	0	7	4	3	0	0	2	14	**192**	23	77.4%
CY	7	0	41	7	0	0	1	0	4	12	**210**	74.5%
Precision	67.6%	71.4%	53.0%	61.8%	79.2%	100.0%	50.0%	76.5%	78.3%	69.3%	55.7%	**63.9%**

The classes that this model seems to be able to distinguish the best are *chloroplast*, *extracellular*, *plasma membrane* and *cytoplasm*, all of them with recalls between 70 and 80%. There is a second group that consists of the classes *nuclear*, *mitochondrial*, *endoplasmic reticulum* and *Golgi*, which the model classifies with recalls between 40 and 50%. Roughly 50% of the *nuclear* sequences are classified as *cytoplasmic*, while $\approx$ 20%

of the *cytoplasmic* sequences are classified as *nuclear*. Finally, the model specially struggles at classifying sequences from the smaller labels: *lysosome*, *vacuole* and *peroxisome*, which have recalls of 23.8%, 7.7% and 3.2%, respectively.

Convolutional Neural Networks (CNNs)

CNNs, given that they take into account the spatial relationships of the data, seem like a potentially powerful tool to evaluate protein sequences.

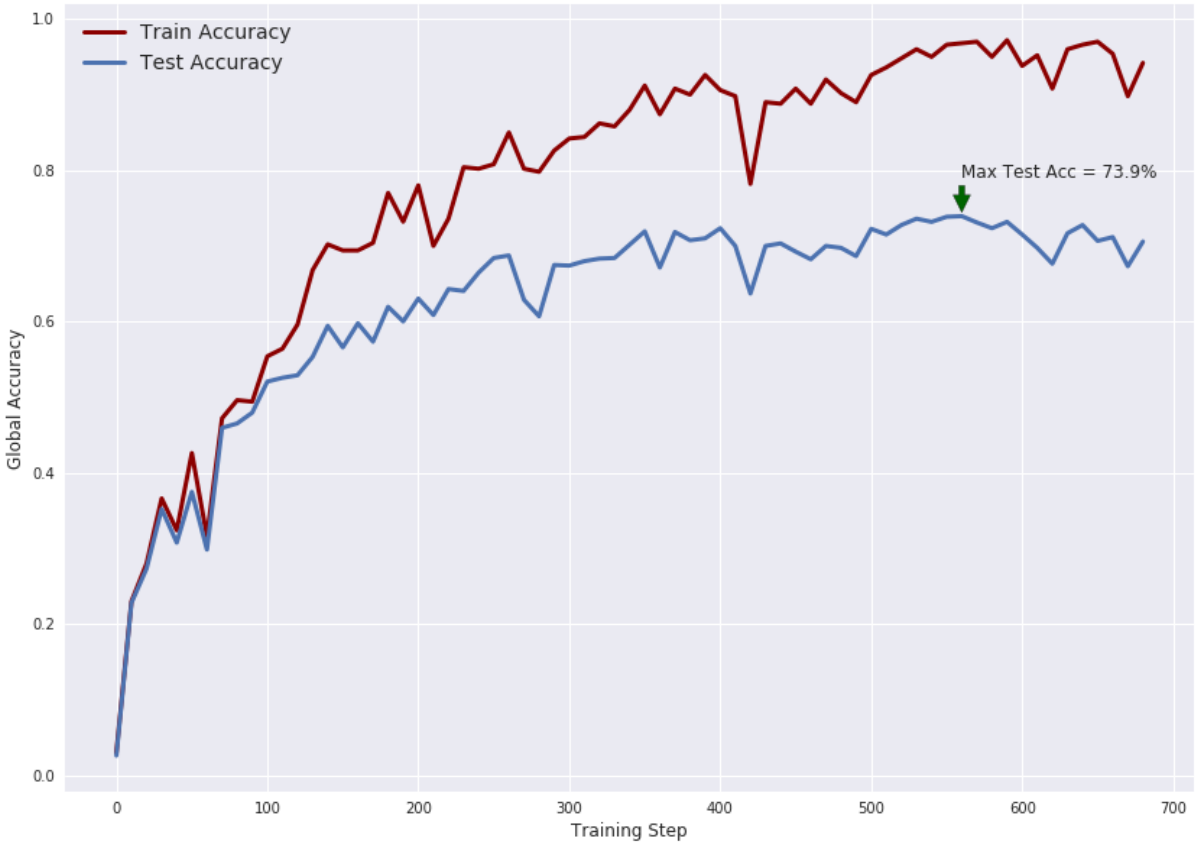


FIGURE 3.4: Accuracy progression during CNN model training

The CNN model needed the longest to train, achieving its best test accuracy, 73.9%, at around step 550 ($\approx$ 46 epochs). The training and test accuracies are on par until around step 100, at 50% accuracy, when the divergence between train and test accuracies starts. After that, both accuracies increase but at a different rate.

TABLE 3.3: Confusion matrix for CNN model

	CH	**LS**	**NC**	**MC**	**ER**	**VC**	**PX**	**GG**	**EC**	**PM**	**CY**	**Recall**
CH	**74**	0	1	3	0	0	0	0	0	1	11	82.2%
LS	0	**1**	0	2	0	0	1	1	7	9	0	4.8%
NC	2	0	**95**	3	0	0	1	1	0	4	61	56.9%
MC	4	0	3	**85**	0	0	0	0	0	1	9	83.3%
ER	0	0	0	0	**20**	0	0	3	5	10	2	50.0%
VC	1	0	1	1	0	**1**	0	1	5	2	1	7.7%
PX	0	0	5	0	0	0	**3**	2	1	1	19	9.7%
GG	1	0	0	0	1	0	0	**22**	2	4	0	73.3%
EC	0	0	2	2	1	0	0	3	**133**	22	6	78.7%
PM	2	0	2	3	0	0	0	2	8	**218**	13	87.9%
CY	6	0	35	3	1	0	1	0	2	4	**230**	81.6%
Precision	82.2%	100.0%	66.0%	83.3%	87.0%	100.0%	50.0%	62.9%	81.6%	79.0%	65.3%	**73.9%**

The CNN model performs better with the *chloroplastic*, *mitochondrial*, *plasma membrane* and *cytoplasmic* sequences, all of them classified with recalls >80%. However, it is barely able to detect *lysosomal*, *vacuolar* and *peroxisomal* sequences, all of which are classified with recalls <10%.

Long Short Term Memory Neural Networks (LSTM)

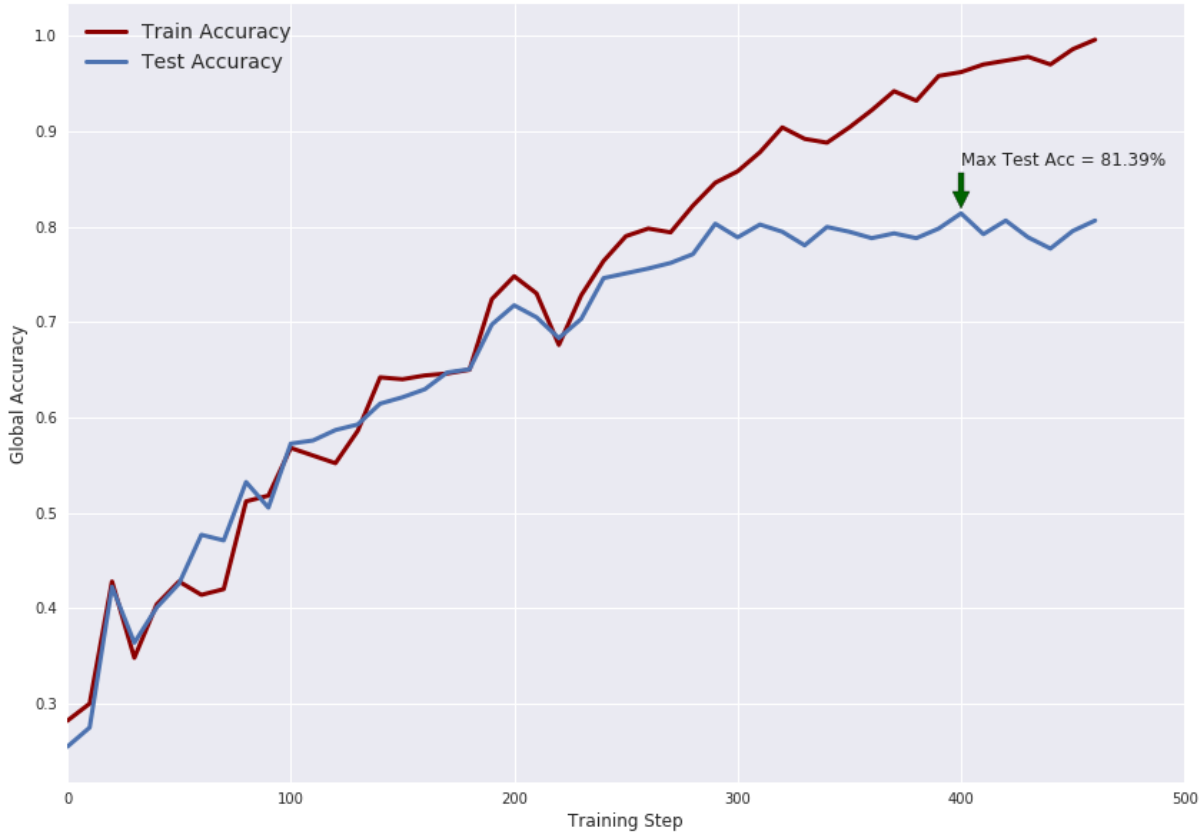


FIGURE 3.5: Accuracy progression during biLSTM model training

The ability of LSTM networks to capture long term relationships in the data while treating it like a sequence looks like the ideal way to recognize the signals present in a protein sequence. Taking into account that the input data is just the the primary sequence, with no information of the tridimensional structure of the proteins, LSTM's

faculty to recognize distantly connected patterns in sequences could fill in the gaps that the data is not providing.

For the LSTM model, as Figure 3.5 depicts, train and test accuracies appear to increase at the same rate during training until the step 300, where the test accuracy hits the 80% threshold and barely improves at all after that. After that, the model overfits for the training data, achieving a training accuracy of >98% at step 460, where the training is stopped. The maximum test accuracy recorded is of **81.4%**, at step 400 ($\approx$ 33 epochs).

TABLE 3.4: Confusion matrix for bidirectional LSTM model

TL / PL	**CH**	**LS**	**NC**	**MT**	**ER**	**VC**	**PX**	**GG**	**EC**	**PM**	**CY**	**Recall**
CH	**82**	0	1	5	0	0	0	0	0	0	2	91.1%
LS	0	**9**	0	0	0	1	0	2	5	4	0	42.9%
NC	2	0	**119**	1	0	0	0	0	3	2	40	71.3%
MT	3	0	1	**92**	0	0	1	0	0	0	5	90.2%
ER	0	1	0	1	**25**	1	0	2	7	3	0	62.5%
VC	0	0	0	0	2	**1**	0	1	6	3	0	7.7%
PX	2	0	3	1	0	0	**15**	0	0	1	9	48.4%
GG	0	0	0	0	2	0	1	**20**	2	5	0	66.7%
EC	0	9	0	0	1	0	0	1	**153**	4	1	90.5%
PM	2	0	0	0	2	1	0	4	6	**230**	3	92.7%
CY	4	1	43	5	0	0	3	0	0	1	**225**	79.8%
Precision	86.3%	45.0%	71.3%	87.6%	78.1%	25.0%	75.0%	66.7%	84.1%	90.9%	78.9%	**81.4%**

This model excels at classifying *chloroplastic*, *mitochondrial*, *extracellular* and *plasma membrane* sequences: all of these classes have a recall over 90%. In the case of *nuclear* and *cytoplasmic* sequences, both large classes, they have recalls over 70% and the model seems to find it hard to distinguish among the two of them, since over 20% of the *nuclear* sequences are classified as *cytoplasmic* and vice versa.

Models Comparison

After seeing the results of the best performing models of the different architectures, it is quite clear that the LSTM model is the better at the task of accurately predicting the subcellular localization of the proteins. The reason why it performs better, though, is not a trivial matter.

In table 3.5 a dissected comparison of the class-specific precision and recall results is shown.

TABLE 3.5: Comparison of precision, recall and accuracy results (%)

Class	**Precision FC**	**Recall FC**	**Precision CNN**	**Recall CNN**	**Precision LSTM**	**Recall LSTM**
CH	67.6	76.7	82.2	82.2	86.3	91.1
LS	71.4	23.8	100	4.8	45.0	42.9
NC	53.0	47.3	66.0	56.9	71.3	71.3
MC	61.8	46.1	83.3	83.3	87.6	90.2
ER	79.2	47.5	87.0	50.0	78.1	62.5
VC	100	7.7	100	7.7	25.0	7.7
PX	50	3.2	50.0	9.7	75.0	48.4
GG	76.5	43.3	62.9	73.3	66.7	66.7
EC	78.3	74.6	81.6	78.7	84.1	90.5
PM	69.3	77.4	79.0	87.9	90.9	92.7
CY	55.7	74.5	65.3	81.6	78.9	79.8
Accuracy	**63.9**		**73.9**		**81.4**	

While there is a clear progression in the accuracy of the networks in the FC → CNN → LSTM direction, the class-specific performance does not follow such a clear pattern:

- The *lysosomal* class appears to be hardly detectable by the CNN model, where its has just a 4.8% recall with a 100% precision. The FC model performs better, with a 23.8% recall, and the LSTM tops it with a 42.9%. These results seem to indicate that the signal present in *lysosomal* sequences [38] is lost during the convolution step.

- None of the models is able to properly predict the location of *vacuolar* sequences. The fact that only 63 out of the total 5959 sequences belong to that class is probably the main reason why this happens. Increasing the class size would most likely help to improve the performance.

- Both the FC and the CNN model are unable to classify the *peroxisomal* proteins, with recalls of 3.2% and 9.7%, respectively, and with a 50% precision. The LSTM model, however, drastically improves the performance in this area, achieving a 48.4% recall with an even better precision of 75%. These results look rather counter-intuitive at first glance, since it is known that one of the peroxisomal targeting signals is usually located at the carboxy terminus of proteins and consists of no more than 12 amino acids [8]. Theoretically, CNNs should be able to detect this signals with ease, but it has not been the case with our model. An explanation on why the LSTM shows such a drastic improvement in performance for this class could be the fact that a motif present in peroxisomal membrane proteins, the *mPTS* motif, might consist of discontinuous subdomains [33] that could be better captured by the LSTM model due its ability to extract information from non-consecutive signals.

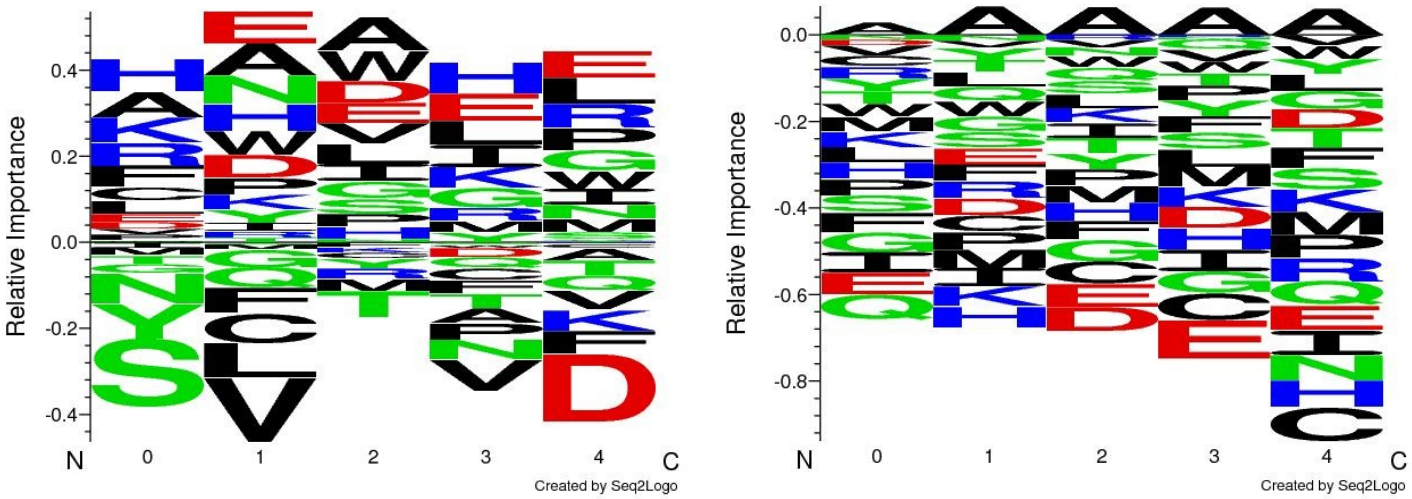
- All in all, the performance in the rest of the classes improves steadily in the FC → CNN → LSTM direction, except the *Golgi* and *cytoplasm* classes, which are slightly better classified in the CNN model. The fact that CNNs are specially useful to extract short and consecutive signals might give them the edge over the rest for these classes.

3.2 Opening the *Black Box*

3.2.1 Convolutional Filters

PSSM-logo representations for the learned filters of all CNN models trained with one-hot encoding as input were generated, which resulted in a high amount of plots. The examples shown here belong to the best-performing CNN model which only used one-hot encoding as input. All the hyperparameters were exactly the same as the ones for the CNN model described in section 2.4.2 but without including the amino acid properties in the encoding. The performance of this model was slightly lower, achieving a 70.5% test accuracy.

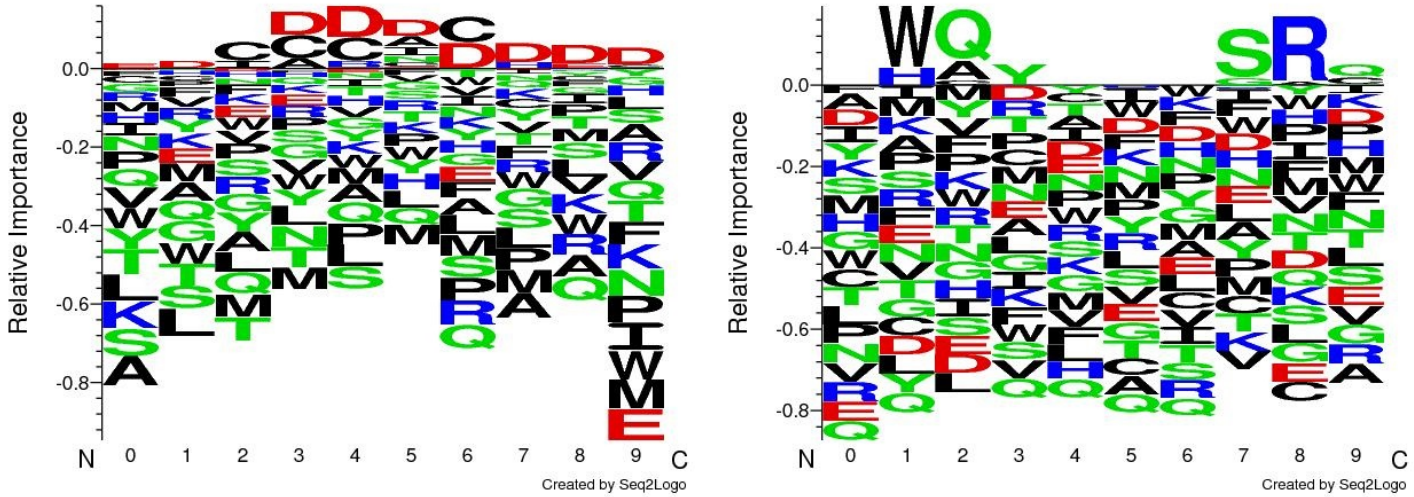
A total of 75 filter representations were generated for this model. The majority of them seem to not only search for certain amino acids but also negatively select for a big number of residues. The following examples illustrate the nature of the generated representations:



(A) (B)

FIGURE 3.6: Examples of learned convolutional filters of width 5

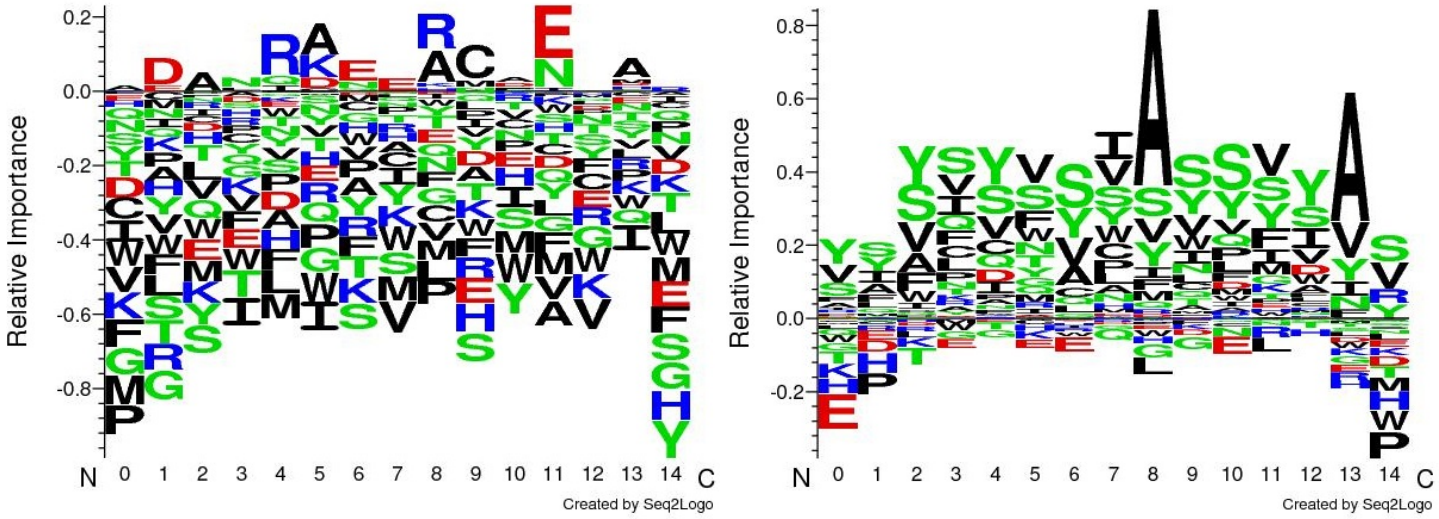
The filter in Figure 3.6a seems to detect charged residues (mainly red and blue amino acids appear in the positive side) except in the last position where it strongly negatively selects for aspartic acid. The filter in Figure 3.6b, however, seems to only search for stretches of alanine while it negatively selects most other residues.



(A) (B)

FIGURE 3.7: Examples of learned convolutional filters of width 10

In the case of the filters of width 10, the negative selection was even more prominent, with most residues falling in the negative side and very few residues on the positive side. The filter in Figure 3.7a captures stretches of aspartic acid with cystein in the middle positions. The filter in Figure 3.7b is does not capture stretches of residues but searches for certain amino acids in specific positions, it basically captures a motif with the defining sequence 'x[WH][QA]YxxxSRQ'.



(A) (B)

FIGURE 3.8: Examples of learned convolutional filters of width 15

The learned convolutional filters of width 15 are rather diverse. Some of them are of similar nature as the ones of width 10 (Figure 3.8a), finding a specific pattern of residues for certain positions. Others, not so much. For instance, the example in Figure 3.8b captures stretches of non-charged residues with alanine in positions 8 and 13.

All in all, this method of representing the learned features of convolutional layers applied to protein (or DNA) sequence data looks like a perfect way to portray the inherently complicated parameters of the convolution steps. A follow-up work which could shed some light into this results would be to search the motifs generated in motif specialized databases to see if they represent already known protein motifs. If that were the case, it would possibly validate the use of this method as a motif-discovery tool.

3.2.2 Signal Masking Approach

The results of the signal masking approach for both the CNN and LSTM models show not only differences among the classes in each model, but also between the models. Visualizing the impact that removing a part of the input signal has on the final layer of the neural network helps to estimate each of this regions' importance.

Convolutional Neural Network

In Figure 3.9, the results for the CNN model are shown. The first aspect to note is the scale of the divergence between the different classes. *Lysosomal* sequences showed about an order of magnitude higher KL divergence than the other classes. *Mitochondrial* and vacuolar *sequences* displayed maximal divergences around 0.15 while the rest stayed between 0.035 and 0.07.

All the classes showed maximum divergence when the signal from the N-terminus was removed, indicating that probably the model picks the signal from the signal peptides located in that region and its classification capability is strongly affected when it is removed. The removal of the C-terminal region also displayed a strong effect in most classes, although not as strong as that of the N-terminal one.

Nuclear, *cytoplasmic* and *peroxisomal* sequences have a practically identical pattern of divergence among the whole sequence, which is to be expected given the fact that the CNN model had a hard time distinguishing this classes among themselves (most *peroxisomal* sequences were classified as either *nuclear* or *cytoplasmic* too). The three classes show a slight increase in divergence when removing the regions about 100 positions before the C-terminal end, which could indicate the presence of some important sorting signal. Cytoplasmic and nuclear proteins are not secreted and thus lack N-terminal signal peptides. It is then rather surprising to find that they still show the maximum divergence in this region. The reason behind this could lie in the fact that signals specific to *nuclear/cytoplasmic* sequences might also happen to be located near the N-terminal end.

Golgi and *plasma* membrane sequences showed the smallest effect: Both of them are groups of transmembrane proteins known to possess signal anchors further away from the N-terminus [11]. This could mean that they possess a less localized and more distributed sorting signal which would be less affected by a local disruption of 5 or 15 amino acids. The pattern and scale of the effect shown by *lysosomal* sequences might be the most difficult to explain. As shown in 3.1.2, the CNN model was the worst at classifying *lysosomal* sequences, with a 4.8% recall. The classification of these sequences was probably so unstable that changes at any region of the sequence have a strong effect in the final layer.

The way the sequences were processed, padding with X's the middle of the shorter sequences, also makes the comparison unfair, since the average effect of the central regions is brought down by those sequences padded with X's for which there is no effect. A better design of this experiment would not take into account the regions pre-filled with X's when calculating the averages and would allow for a fairer comparison.

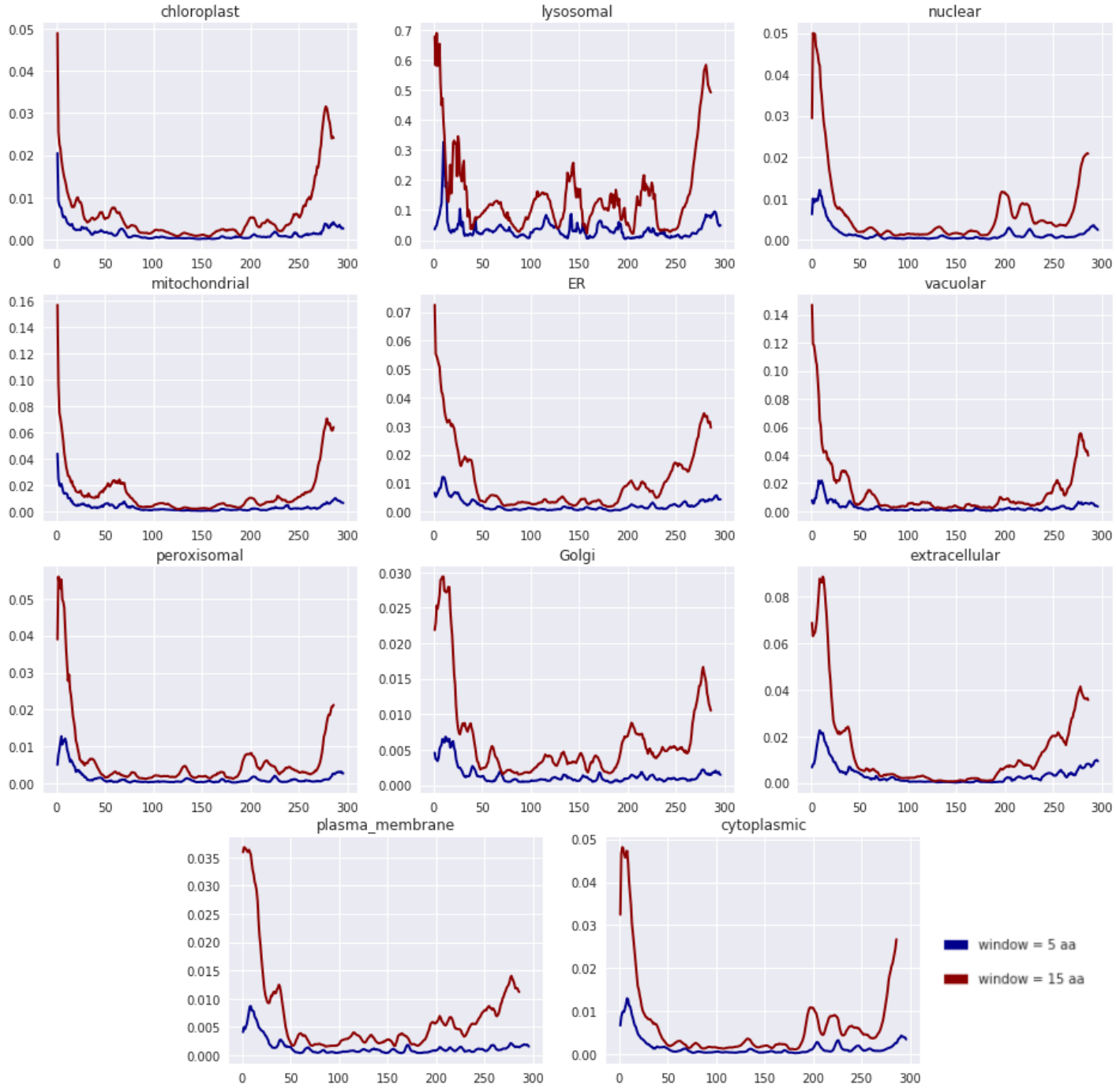


FIGURE 3.9: KL divergence between original and modified sequences at each position for each class in the CNN model

Bidirectional LSTM Neural Network

The results for the biLSTM model can be seen in Figure 3.10. Given the longer input sequence length (750 aa) and the fact that the only visible effects were concentrated on the ends of the sequences, only 125 amino acids of the N-terminal end and 125 of the C-terminal end are shown to enhance the visualization.

The first noticeable detail is the magnitude of the KL divergence in some of the classes. The *lysosomal* class is again the one showing the highest effect, reaching almost a KL divergence of 12 at its maximal point. The pattern that it displays along the sequence, though, is now way less erratic than in the CNN. *Extracellular*, *Golgi* and *ER* sequences also show strong effects, reaching KL divergence values ranging from 5 to 10. All of these, together with the *vacuolar* sequences, belong to the secretory pathway and contain N-terminal signal peptides, strong signals that the forward LSTM layer probably picks at the first time steps and propagates until the end. The way the information is propagated in the LSTM layers might be responsible for the increase in KL divergence when removing the signal of the N-terminal ends. It is also specially remarkable the strength of the divergence in the C-terminal end of the ER sequences, indicating a relatively important presence of sorting signals in the region.

The *lysosomal* sequences improved from a 4.8% test accuracy in the CNN to a 42.9% in the biLSTM model. That means that some of the signals that were not picked by the CNN are being captured in this biLSTM model. *Lysosomal* proteins are known to contain sorting signals in their cytosolic domains [5], but the results obtained here suggest that only those in the N-terminal end are being picked and the cytosolic domains are not being recognized by the network, which would explain the 57.1% error rate for this class.

Another striking feature is the divergence pattern shown by the *peroxisomal* sequences. It is the sole class with a higher divergence when removing the signal from the C-terminal end than when doing so in the N-terminal end. Whilst the CNN did not show this feature, it only classified *peroxisomal* sequences with a 9.7% accuracy, so it barely learned any feature from this class. The biLSTM model, however, achieves a 48.4%. The results obtained here suggest that mainly the targeting signals located at the C-terminal end of proteins [8] mentioned in 3.1.2 are the ones being learned by the biLSTM and responsible for its increase in accuracy.

In general, all the classes for this model showed minimal to null KL divergence when removing the signal of every region but the C- and N-terminal ends. This could mean that either the directional nature of the LSTM layers confers way more importance to the beginning and the end of the sequences or that the increase in accuracy compared to the CNN model comes from the increase in signal recognition from the ends, which makes the whole prediction more resilient to changes in other regions.

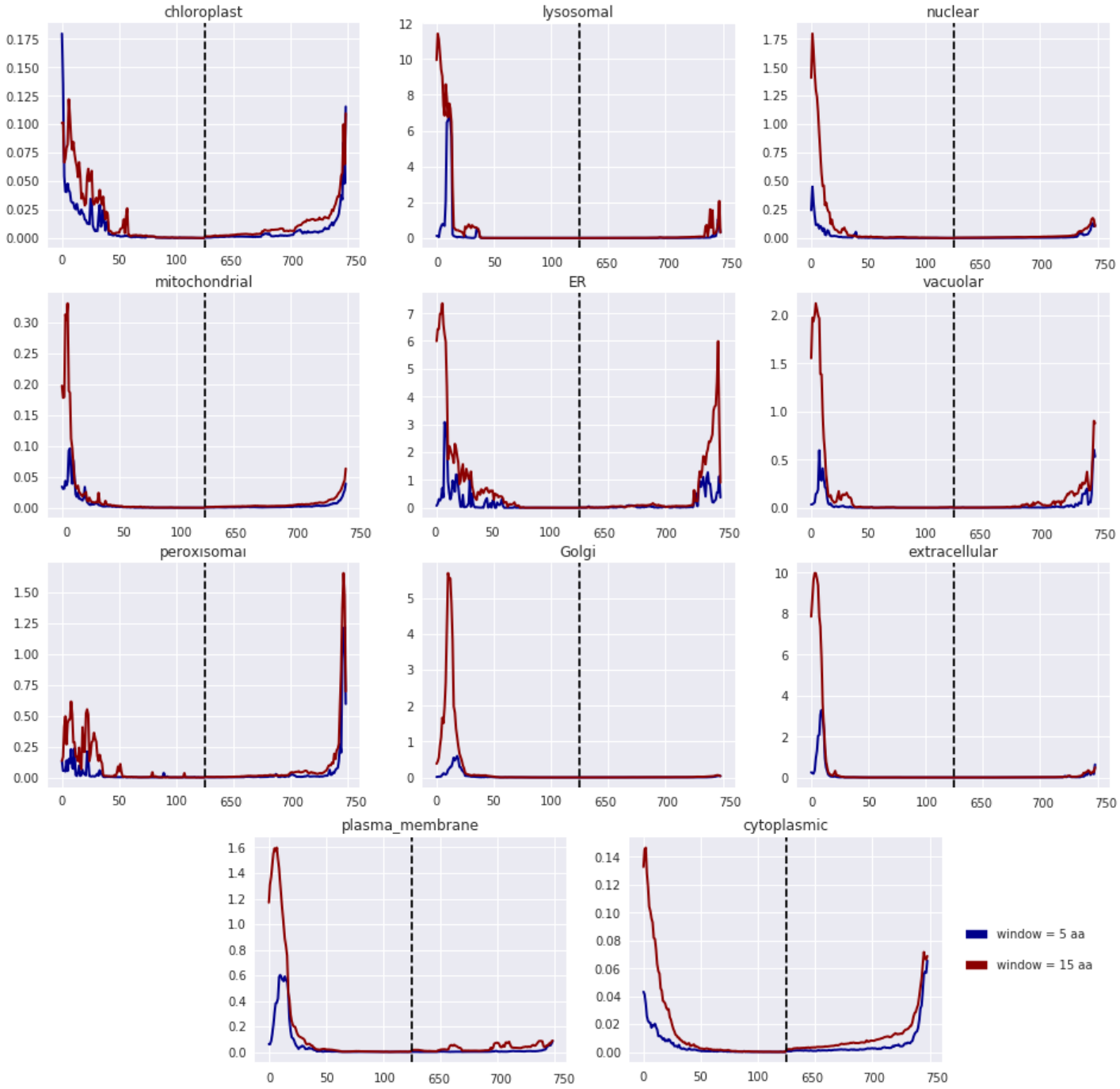


FIGURE 3.10: KL divergence between original and modified sequences at each position for each class in the LSTM model

3.2.3 Class Optimization

The class optimization approach was attempted for both the CNN and biLSTM models. The optimization of the input of the CNN model, however, did not appear to converge for any of the classes even after >5000 optimization steps. For that reason, only the results of the biLSTM model are shown and discussed.

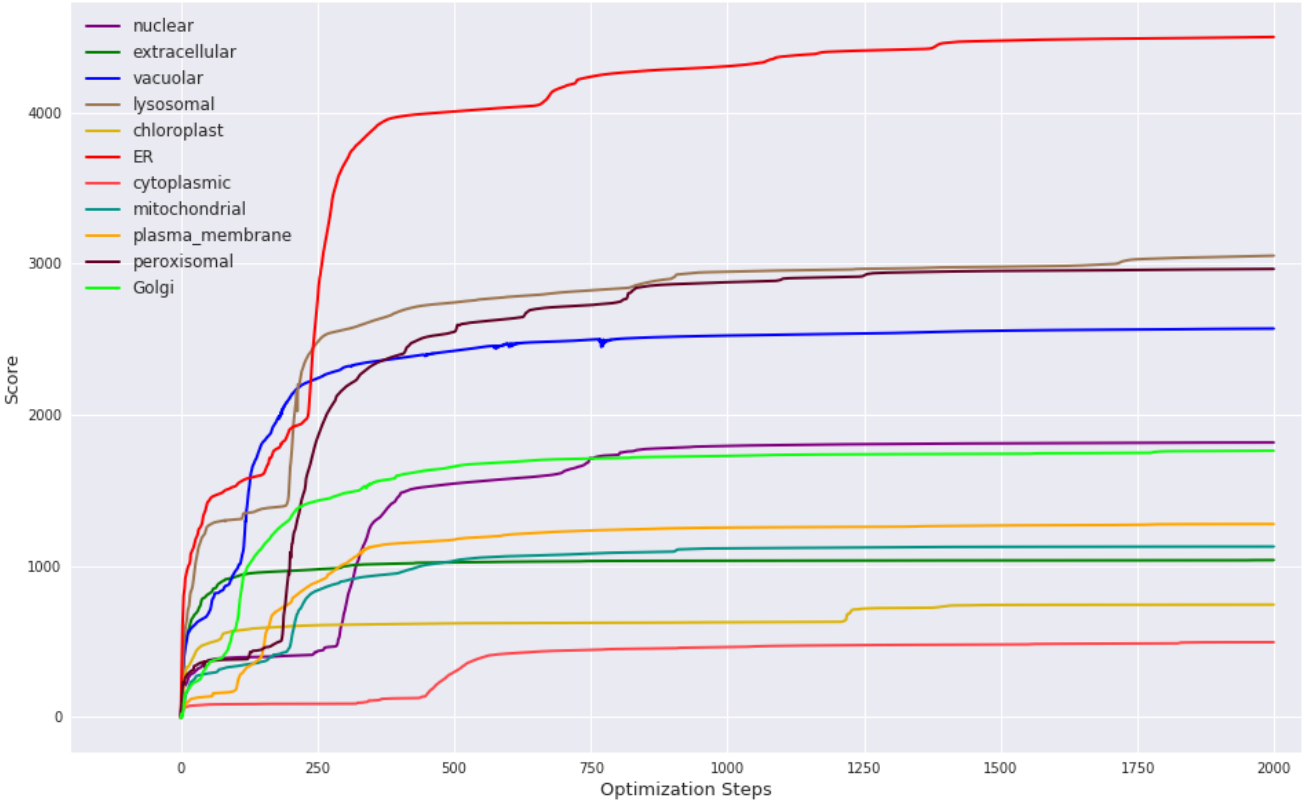


FIGURE 3.11: Score progress during optimization in biLSTM model

After 2000 optimization steps, all classes appear to have nearly converged at scores ranging from 500 (*cytoplasmic*) to 4500 (*endoplasmic reticulum*) (see Figure 3.11). There is no way to tell whether the process is stuck at local maxima or these are actually the optimal sequences for each of the classes. With the objective of assessing how well this optimized inputs rank when comparing them with the scores of the real sequences, the score distributions for the sequences of each of the classes are shown in Figure 3.12. Each class follows a rather different distribution of scores, but none of the real sequences has a score higher than 20 in any class. The optimized input sequences score 20 to 200 times better than the best-scoring real sequences. Of course, the fact that the real sequences are limited to contain only one-hot vectors hinders their ability to obtain the high scores that the optimized inputs, which can distribute their values with way less constraints, achieve. It was also checked with success whether the score of the optimal sequences for the other classes was smaller than the score they had for their own. Given that this was not an optimization criterion, it was necessary to add this step in order to be sure that the increase in score was localized in the desired class and not just a consequence of an increase of the order of magnitude of the overall scores.

Nonetheless, the obtained optimized inputs can be interpreted as a representation of

the features that the model has learned for each of the classes. If certain positions in the sequences strongly point towards a specific residue/group of residues it is likely that they are important towards the classification of the sequence to its class. As such, this method could potentially lead to the discovery of new sorting signal motifs.

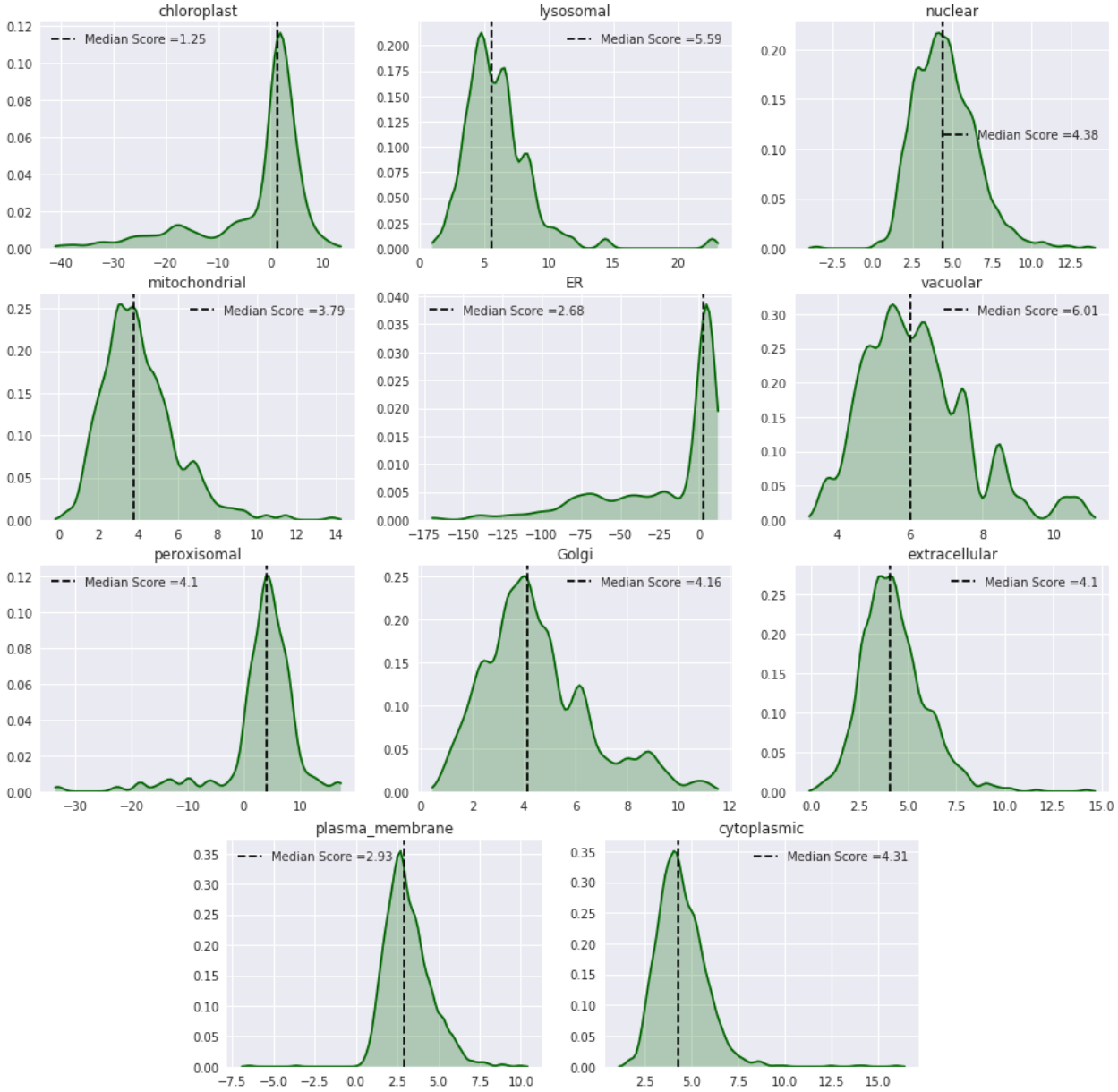


FIGURE 3.12: Score distribution among classes for real sequences

An analysis and a motif search for the obtained optimized inputs has not been performed given the time constraints of the project, but it would definitely be an interesting line of further research. Nevertheless, the representation of the results and some interpretation of them can already be done.

In Figures 3.13 and 3.14, the C-terminal end of the optimized input for the *peroxisomal* class and the N-terminal end of the optimized input for the *extracellular* class can be seen. The results suggest that sequences ending with stretches rich in phenylalanine or glycine will score highly for the peroxisomal class. A leucine in position 736

also seems to be specially important. Similar conclusions can be extracted from the N-terminal end of the *extracellular* input: A valine in first position and lysines/arginines along the following positions will guide the model to predict the input sequence as belonging to the extracellular class.

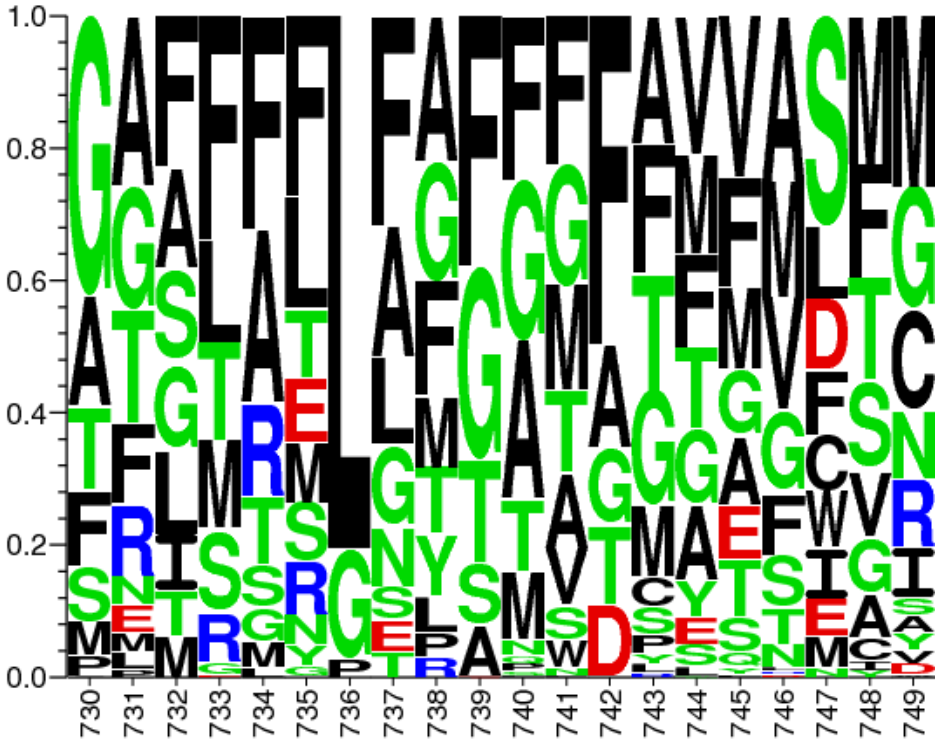


FIGURE 3.13: C-terminal end of the optimized peroxisomal input

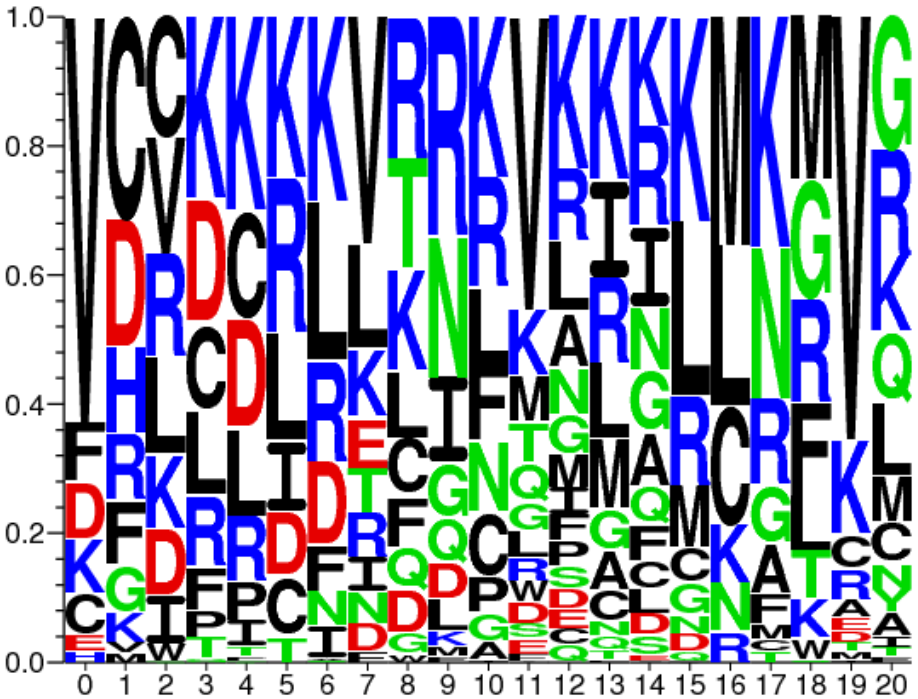


FIGURE 3.14: N-terminal end of the optimized extracellular input

Of course, these are just two examples of the many that could be shown, and further validation of the actual findings would be required to extract any meaningful conclusion.

3.3 General Discussion and Conclusion

Models training and performance

Comparing the performance achieved by the models trained with state-of-the-art methods is rather tricky given the fact that most employ their own metrics to assess performance and use different datasets to do so too. Also, comparing this method, which only uses protein sequence data as input, with methods that include multiple sources of metadata or homology data is quite unfair.

When comparing with methods only using the same kind of input data, the biLSTM model, with 81.4% accuracy performs better than SVM-based MultiLoc [11] (75% accuracy) but is still far from the Convolutional biLSTM ensemble model trained in [37] (90.2% accuracy). With similar datasets, but adding metadata as input too, Multiloc2 [4] improves to a $\sim$89% accuracy. SherLoc2 [6] goes even further and uses 7 subclassifiers which look up different sources to build the input features, achieving a striking 93% classification accuracy.

With little fine tuning, the biLSTM model quickly achieved comparable performances at the classification task to state-of-the-art techniques, though still at some distance from the best. This indicates that with more hyperparameter search it could probably achieve even better accuracies. One thing that might have hampered the performance of the biLSTM models is the zero-padding (addition of X's) in the middle of the shorter sequences. Taking into account that most of the signal is located in the ends, fully connected networks benefit from this kind of padding, since they have position-specific weights and padding in the middle aligns the ends of the sequences in the same positions. CNNs do not directly have position-specific weights, but the structure of the sequence is maintained post-convolution and is then fed to fully connected layers whose parameters are again position-specific. With LSTM's, however, the flow of information is totally different. The LSTM layers treat the data on a sequential manner, and they share the weights of their four gates along all the steps, so they do not benefit from padding the sequences to have the ends aligned. In fact, having a gap without signal in the middle probably disrupts the sequential flow of the information. While the long-term memory information is kept, the short-term information is completely lost when it is propagated through the stretch of X's. Since only the output of the final step of both *forward* and *backward* LSTM layers is propagated, a sensible approach would be to pad the sequence with X's at the beginning for the input of the *forward* layer and at the end for the input of the *backward* layer.

One of the characteristics of the dataset used for training (as a consequence of the actual distribution of proteins in cells) is the huge class imbalance. The trained models are barely able to predict the extremely infrequent classes. This is more likely due to the scarce number of examples they are shown to learn in comparison to the more frequent classes, and the fact that in the way the loss function is defined, it is cheaper to misclassify infrequent classes than the frequent ones. This is a good procedure if the main interest is to achieve the highest possible global accuracy, but in that case we might as well remove the most infrequent classes and focus on the ones

well enough represented. If we take into account the fact that we would like to be able to extract some conclusions from the learnings of the model, increasing the weight that the small classes have during the training steps could be helpful. A modification of the cost function as the one used in [40], where the misclassification of infrequent classes has a higher cost than the misclassification of the more frequent ones, could be used. Manually assigning different costs to different kinds of errors, making misclassifications between more similar classes less costly, could also help guiding the optimization process in a better way. Some variation of curriculum learning [2] would also be an interesting feature to be tried. Roughly, curriculum learning is based on the idea of training the models on stages of increasing level of difficulty, making the network learn from the *easier* examples in the beginning and progressively add the more *difficult* examples to the training set. This is supposed to help speed up the convergence of training towards the global minimum (on convex criteria) or find better local minima (on non-convex criteria) while also having a regularizing effect. In the context of subcellular location prediction, the *easy* examples could be defined as those belonging to the classes with a previously recorded high success of classification whereas the *difficult* ones could be those belonging to the classes with a low success of classification. Another option would be to group the classes by similarity at the beginning (e.g. *secreted-intracellular*) and progressively separate them until the final 11.

In all cases of the three architectures, during training, the model ends up with an almost perfect accuracy for the training sets. Contrary to the expectation, though, this does not hinder the test performance, and when the test and train accuracies start to diverge, the test accuracies either stay the same or just keep getting better at a lower rate. While at first glance the differences in performance between the train and test sets seem to point towards a problem of overfitting, this appears not to be the case. Overfitting is usually defined as the phenomenon that occurs when models *memorize* the training set, achieving very good performance with it but losing its ability to generalize for any new data. The ability to generalize does not seem to be lost given the performance seen in the test sets, and therefore, what happens during the training of these models cannot be called overfitting. This means that, for some reason, the patterns specific to the training set that the model learns after the training performance starts to diverge do not harm the ability it has to generalize for new data.

Models interpretation

The challenging part of the project was to actually interpret and extract meaningful conclusions from the trained neural network models. The PSSM-logo representations of the learned filters are the most straightforward and realistic way to display the features learned by a convolution layer applied to sequence data. The fact that the convolution is done from left to right of the sequence implies that the output layers still keep the relationship to the original input amino acids, and filter representations of deeper convolution layers could still be done. A deeper convolutional network (with more convolution and pooling steps) could be represented and although the deeper levels would be harder to interpret they could shed some light about what are the more complex feature relationships that the network is able to learn.

Regarding the signal masking approach, several modifications could be applied. The first and more obvious is, as mentioned before, to not take into account the parts of the sequences which are already pre-filled with X's when calculating the average KL

divergence per position. Another feature to add would be to implement a way to actually weigh properly the changes in the output layer of the model. A slight change in the score for two classes might change the final decision of the model, and big changes among the scores of multiple classes might not influence the final decision of the model. The KL-divergence does not take this into account and it just measures the change of the whole probability distribution. An alternative way to measure the importance of each position of the input sequences would be to obtain the saliency maps [17, 35]. These are weighted maps of the influence of each position of a sequence in the final class score of a classification task. It has shown good results in image and DNA sequence data. On another note, variability of the KL-divergence for all sequences in a class has not been accounted for. It would be wise to take some measurement of variability for this at the class level, since it would provide useful information. For instance, specially high variability in the patterns shown by the sequences inside a class could be an indicator of different subgroups of sequences with distinct relevant regions.

The class optimization method is a great way to generate representations of what the network has learned from the examples used to train it. In some way, it shows what is its *idea* of each of the classes. Depending on the situation it could (i) help understand why some classes are being wrongly classified (if the optimal class input contradicts already validated knowledge about that class, for instance) or (ii) lead to the discovery of not yet known features of the classes. The latter usage is one of the main advantages and exciting characteristics of machine learning approaches, the ability to find new patterns in the data without (relatively) any guidance.

On a conceptual level, the three approaches used to attempt to *open the black box* work quite differently and at different levels:

- The representation of the convolutional filters is just a visual representation of the transformations that the data faces when it goes through a convolution step. It is possible to do this given the nature of the data (sequential) and of the transformation (convolution).

- The signal masking approach is an attempt to obtain an estimation of the importance of the input features. Feature selection [3] is rather straightforward with simpler machine learning algorithms like decision trees. It is useful both to select relevant input features and determine the importance of each of them. Deep learning algorithms lack an easy feature selection process, and that is the gap that approaches like the signal masking approach or the mentioned saliency maps try to fill. In this case, it was not used to select features but to determine the importance of each region of the sequences for the different classes.

- The class optimization is a portrayal of the general *idea* that the network has learned for each of the classes. An even better use for this approach in the context of subcellular location prediction would be to couple it with one of the feature importance methods. The class optimization provides information about what the optimal residue is for each position while the other methods yield the relative importance of each of the positions. Combining the two of them could render a faithful representation of what the network expects and looks at for each of the classes. The relevant stretches could then be used to search for known domains or motifs which could be linked to that specific class.

All in all, the application of deep learning approaches to biological sequence data is on the rise [14, 23], and the interest to find ways to decipher the *black box* that these represent will most certainly lead to several interesting attempts in the near future.

Bibliography

[1] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: ().

[2] Yoshua Bengio, umontrealca Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum Learning". In: *Proceedings of the 26th annual international conference on machine learning* (2009), pp. 41–48.

[3] Avrim L. Blum and Pat Langley. "Selection of relevant features and examples in machine learning". In: *Artificial Intelligence* 97.1-2 (Dec. 1997), pp. 245–271. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(97)00063-5.

[4] Torsten Blum, Sebastian Briesemeister, and Oliver Kohlbacher. "MultiLoc2: integrating phylogeny and Gene Ontology terms improves subcellular protein localization prediction". In: *BMC Bioinformatics* 10:274 (2009). DOI: 10.1186/1471-2105-10-274.

[5] Juan S. Bonifacino and Linton M. Traub. "Signals for Sorting of Transmembrane Proteins to Endosomes and Lysosomes". In: *Annual Review of Biochemistry* 72.1 (June 2003), pp. 395–447. ISSN: 0066-4154. DOI: 10.1146/annurev.biochem.72.121801.161800.

[6] Sebastian Briesemeister, Torsten Blum, Scott Brady, Yin Lam, Oliver Kohlbacher, and Hagit Shatkay. "SherLoc2: A High-Accuracy Hybrid Method for Predicting Subcellular Localization of Proteins". In: *Journal of Proteome Research* 8.11 (Nov. 2009), pp. 5363–5366. ISSN: 1535-3893. DOI: 10.1021/pr900665y.

[7] Olof Emanuelsson, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. "Locating proteins in the cell using TargetP, SignalP and related tools". In: *Nature Protocols* 2.4 (Apr. 2007), pp. 953–971. ISSN: 1754-2189. DOI: 10.1038/nprot.2007.131.

[8] S G Gould, G A Keller, and S Subramani. "Identification of a peroxisomal targeting signal at the carboxy terminus of firefly luciferase." In: *The Journal of cell biology* 105.6 Pt 2 (Dec. 1987), pp. 2923–31. ISSN: 0021-9525. DOI: 10.1083/JCB.105.6.2923.

[9] S Henikoff and J G Henikoff. "Amino acid substitution matrices from protein blocks." In: *Proceedings of the National Academy of Sciences of the United States of America* 89.22 (Nov. 1992), pp. 10915–9. ISSN: 0027-8424.

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[11] A. Hoglund, P. Donnes, T. Blum, H.-W. Adolph, and O. Kohlbacher. "MultiLoc: prediction of protein subcellular localization using N-terminal targeting sequences, sequence motifs and amino acid composition". In: *Bioinformatics* 22.10 (May 2006), pp. 1158–1165. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl002.

[12] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T.

[13] S Hua and Z Sun. "Support vector machine approach for protein subcellular localization prediction." In: *Bioinformatics (Oxford, England)* 17.8 (Aug. 2001), pp. 721–8. ISSN: 1367-4803.

[14] Vanessa Isabell Jurtz et al. "An introduction to deep learning on biological sequence data: examples and solutions". In: *Bioinformatics* 33.22 (Nov. 2017), pp. 3685–3690. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx531.

[15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014).

[16] S. Kullback and R. A. Leibler. *On Information and Sufficiency*. DOI: 10.2307/2236703.

[17] Jack Lanchantin, Ritambhara Singh, Zeming Lin, and Yanjun Qi. "Deep Motif: Visualizing Genomic Sequence Classifications". In: (May 2016).

[18] P. Larranaga et al. "Machine learning in bioinformatics". In: *Briefings in Bioinformatics* 7.1 (Feb. 2006), pp. 86–112. ISSN: 1467-5463. DOI: 10.1093/bib/bbk007.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

[20] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. "Efficient BackProp". In: 2012, pp. 9–48. DOI: 10.1007/978-3-642-35289-8{_}3.

[21] Xueliang Liu. "Deep Recurrent Neural Network for Protein Function Prediction from Sequence". In: (Jan. 2017).

[22] Z. Lu et al. "Predicting subcellular localization of proteins using machine-learned classifiers". In: *Bioinformatics* 20.4 (Mar. 2004), pp. 547–556. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg447.

[23] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. "Deep learning in bioinformatics". In: *Briefings in Bioinformatics* 18.5 (July 2016), bbw068. ISSN: 1467-5463. DOI: 10.1093/bib/bbw068.

[24] S Möller, M D Croning, and R Apweiler. "Evaluation of methods for the prediction of membrane spanning regions." In: *Bioinformatics (Oxford, England)* 17.7 (July 2001), pp. 646–53. ISSN: 1367-4803.

[25] Rajesh Nair and Burkhard Rost. "Sequence conserved for subcellular localization". In: *Protein Science* 11.12 (Apr. 2009), pp. 2836–2847. ISSN: 09618368. DOI: 10.1110/ps.0207402.

[26] H. Nielsen, J. Engelbrecht, S. Brunak, and G. von Heijne. "Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites". In: *Protein Engineering Design and Selection* 10.1 (Jan. 1997), pp. 1–6. ISSN: 1741-0126. DOI: 10.1093/protein/10.1.1.

[27] Henrik Nielsen. "Predicting Subcellular Localization of Proteins by Bioinformatic Algorithms". In: Springer, Cham, 2015, pp. 129–158. DOI: 10.1007/82{_}2015{_}5006.

[28] Henrik Nielsen. "Protein Sorting Prediction". In: Humana Press, New York, NY, 2017, pp. 23–57. DOI: 10.1007/978-1-4939-7033-9{_}2.

[29] Seong Joon Oh, Max Augustin, Bernt Schiele, and Mario Fritz. "Whitening Black-Box Neural Networks". In: (Nov. 2017).

[30] Julian D Olden and Donald A Jackson. "Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks". In: *Ecological Modelling* 154.1-2 (Aug. 2002), pp. 135–150. ISSN: 0304-3800. DOI: 10.1016/S0304-3800(02)00064-9.

[31] A Reinhardt and T Hubbard. "Using neural networks for prediction of the subcellular location of proteins." In: *Nucleic acids research* 26.9 (May 1998), pp. 2230–6. ISSN: 0305-1048.

[32] Juergen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: (Apr. 2014). DOI: 10.1016/j.neunet.2014.09.003.

[33] Nicole Schueller et al. "The peroxisomal receptor Pex19p forms a helical mPTS recognition domain." In: *The EMBO journal* 29.15 (Aug. 2010), pp. 2491–500. ISSN: 1460-2075. DOI: 10.1038/emboj.2010.115.

[34] Rudy Setiono, Wee Kheng Leow, and James Y. L. Thong. *Opening the neural network black box: an algorithm for extracting rules from function approximating artificial neural networks*. 2000.

[35] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: (Dec. 2013).

[36] Ritambhara Singh, Jack Lanchantin, Arshdeep Sekhon, and Yanjun Qi. "Attend and Predict: Understanding Gene Regulation by Selective Attention on Chromatin". In: (Aug. 2017).

[37] Søren Kaae Sønderby, Casper Kaae Sønderby, Henrik Nielsen, and Ole Winther. "Convolutional LSTM Networks for Subcellular Localization of Proteins". In: Springer, Cham, 2015, pp. 68–80. DOI: 10.1007/978-3-319-21233-3{_}6.

[38] "Sorting of lysosomal proteins". In: *Biochimica et Biophysica Acta (BBA) - Molecular Cell Research* 1793.4 (Apr. 2009), pp. 605–614. ISSN: 0167-4889. DOI: 10.1016/J.BBAMCR.2008.10.016.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[40] Balázs Szalkai and Vince Grolmusz. "Near perfect protein multi-label classification with deep neural networks". In: *Methods* (July 2017). ISSN: 10462023. DOI: 10.1016/j.ymeth.2017.06.034.

[41] Julie D. Thompson, Toby. J. Gibson, Des G. Higgins, Julie D. Thompson, Toby. J. Gibson, and Des G. Higgins. "Multiple Sequence Alignment Using ClustalW and ClustalX". In: *Current Protocols in Bioinformatics*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Aug. 2002, pp. 1–2. ISBN: 9780471250951. DOI: 10.1002/0471250953.bi0203s00.

[42] Martin Christen Frølund 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." In: *Nucleic acids research* 40.Web Server issue (July 2012), pp. 281–7. ISSN: 1362-4962. DOI: 10.1093/nar/gks469.