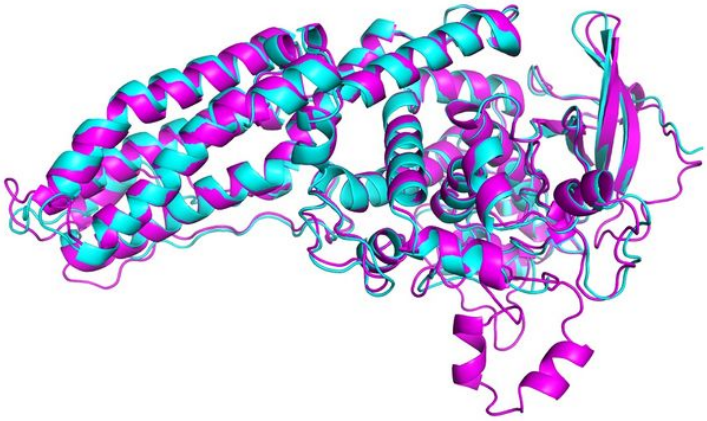
ESPCI PARIS | PSL

Deep sequence models for protein classification



École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris

Jean Guillaume Brasier
Cosme Mosneron
Pablo Mas
Axel Puyo

Prof. Alexandre Allauzen

Contents

1 Introduction and State of the Art **2**
1.1 Classification Problems . . . 2
1.2 Relevancy of LSTM Models . . . 2
1.3 CNNs for text classification . . . 3

2 Network operations **4**
2.1 Long Short-Term Memory (LSTM) Cells . . . 4
2.2 Concatenation of Two LSTMs: biLSTM cells . . . 5
2.3 1-D Convolutions over text . . . 6

3 Modeling and Results **7**
3.1 Input Dataset Pre-Processing . . . 7
3.2 Hyper-parameters . . . 8
3.3 Model Training and Testing . . . 8
3.3.1 1 Layer biLSTM Only . . . 10
3.3.2 4 Layer CNN Only . . . 10
3.3.3 biLSTM + CNN . . . 11
3.4 Classification Accuracy and Missed Predictions . . . 11

4 Discussion and Conclusion **14**

1 Introduction and State of the Art

Proteins are linear chains of amino acid residues that fold into specific 3D conformations as a result of the physical properties of their sequence. These structures, in turn, determine a wide array of protein functions, from binding specificity and catalytic activity to localization within the cell. Information about structure is vital for studying the mechanisms of these molecular machines in health and disease, and for developing new therapeutics. However, experimental structure determination is costly and atomic structures have only been determined for a tiny fraction of known proteins.

Methods for finding proteins with related structure directly from sequence are of considerable interest, but the problem is challenging, because sequence similarity and structural similarity are only loosely related, e.g. similar structural folds can be formed by diverse sequences. As a result, our ability to transfer knowledge between proteins with similar structures is limited. To address this problem, it is often sought to sequence the proteins and use computational tools in order to classify them, especially through the use of deep neural networks, able to infer this loose and complex relationship between sequence and structural similarities.

1.1 Classification Problems

Classification is a central topic in machine learning that has to do with teaching machines how to group together data by particular criteria. Classification is an important tool in today's world, where big data is used to make all kinds of decisions in government, economics, medicine, and more. Researchers have access to huge amounts of data, and classification is one tool that helps them to make sense of the data and find patterns.

In this report, we deal with a protein sequence classification problem, which is a fundamental task for the analysis and interpretation of biological sequential data in bioinformatics. In fact, biologists are often interested in identifying the family to which a lately sequenced protein belongs [2]. This makes it possible to study the evolution of this protein and to discover its biological functions. Since relevant information is encoded by character strings, other well-known classification techniques such as decision trees, naive Bayes, support vector machines or nearest neighbor do not prove very efficient at accomplishing this task. Thus, neural network assisted classification and prediction techniques are more widely used in this field.

Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Neural networks are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.

1.2 Relevancy of LSTM Models

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture which, unlike standard feed-forward neural networks, has feedback connections. It can not only

process single data points (such as images), but also entire sequences of data (such as speech or video). RNN and derivative networks are well-suited to classifying task. In fact, LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs, by allowing the back-propagating gradients to flow unchanged. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications, thus LSTM models are further well-equipped to deal with classification tasks.

In particular, Bidirectional LSTM, or biLSTM tend to outperform vanilla LSTMs for this reason: biLSTM is a sequence processing model that consists of two LSTMs, one taking the input in a forward direction, and the other in a backwards direction. biLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm (e.g. knowing what characters immediately follow and precede the current one), thus better inferring the properties of protein sequences in our case *(see Chapter 2)*.

1.3 CNNs for text classification

Convolutional Neural Networks (or CNN) were initially developped for image processing tasks where they have acheived breakthrough results in classification and object detection. CNNs use mainly two operations in sequence to extract features from images, **convolutions** and **pooling**. The output of this sequence of operations is then typically connected to a fully connected layer which is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Recently CNNs have gained recognition in the Natural Language Processing (NLP) community. Texts like protein sequences are 1-dimensional arrays that can be analysed and classified by CNNs. These models are generally more cost effective in terms of computation time compared to LSTMs. An example of 1-d convolutions for sentence classification is illustrated in figure 1.

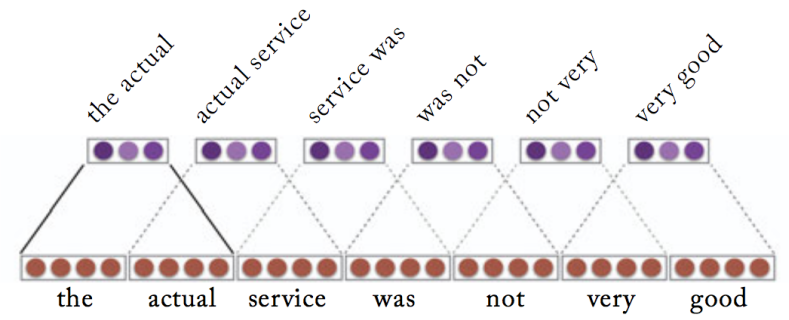


Figure 1: Example of a sentence convolution with kernel size k=2 and dimensional output l=3.

Proteins contain regions of sequences which are responsible for their macroscopic structure. Identifying these regions which can be 20 to 40 amino acids long is key to correctly classifying such proteins. 1-D convolutions with an adapted kernel size over these sequence regions are able to extract the underlying features that uniquely characterise a protein.

2 Network operations

2.1 Long Short-Term Memory (LSTM) Cells

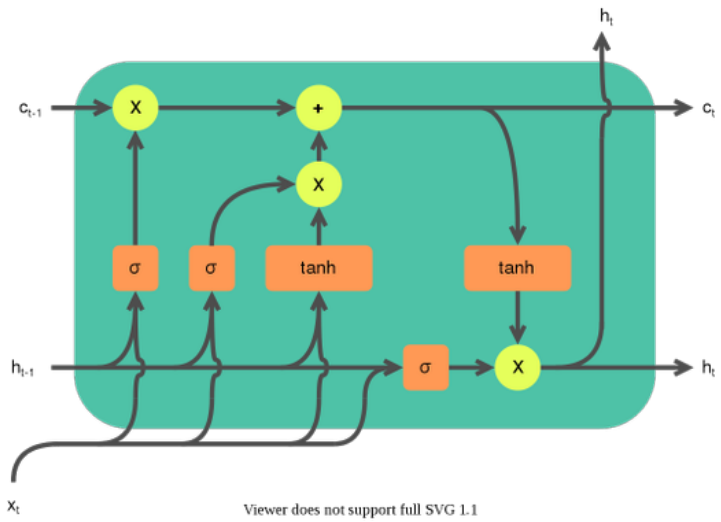


Figure 2: Schematic of a LSTM cell

In order to elucidate the equations describing the functioning of a LSTM cell, let us first consider the following notations:

Notations

Let the superscripts d, h respectively denote the number of input features and of hidden units, we will refer to the following notations:

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit.
- $f_t \in \mathbb{R}^h$: forget gate's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector (output vector of the LSTM unit)
- $\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training.

Figure 2 translates to the following mathematical equations with the notations introduced above:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \tanh(W_{\tilde{c}} x_t + U_{\tilde{c}} h_{t-1} + b_{\tilde{c}}) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
h_t &= o_t \cdot \tanh(c_t)
\end{aligned}
$$

where σ is the sigmoid activation function, $c_0 = h_0 = 0$ and $\cdot$ denotes the Hadamard product (element-wise multiplication). The subscript t indicates the time step.

Knowing that $W_t = W_{t-1} + l_r \nabla$, where l_r is the **learning rate** and ∇ denotes the **gradient**, traditional RNNs tend to be subject to the **vanishing gradient** effect, in which a small gradient will cause weights not to be updated, and thus, memory to be lost. LSTM cells free themselves from this issue through the use **sigmoid activations** in forget, input and output update gates (values between zero and one: the closer to zero means "forget" these values, the closer to one means "keep" this information).

2.2 Concatenation of Two LSTMs: biLSTM cells

Nonetheless, unidirectional LSTM networks only traverse sequential character string data in one direction, and thus concatenating two LSTMs parsing the input data in opposite directions allows for memorizing and recognizing patterns in both directions, without loss of physical meaning, since protein sequences can be read both ways –thus achieving better predictions at the cost of reaching equilibrium slower. In this sense, a biLSTM network preserves information from the "future", and using the two hidden states combined you are able in any point in time to preserve information from both "past and future", **thus better inferring and understanding context**.

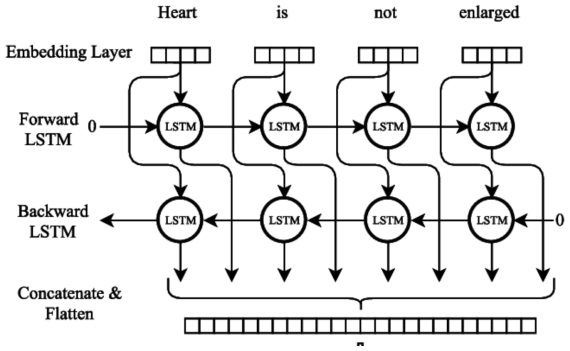


Figure 3: Example schematic of a biLSTM network

2.3 1-D Convolutions over text

Given a sequence of characters $c_{1:n} = c_1, ..., c_n$ where each is associated with an embedding vector of dimension d. A 1-D convolution of width k is the result of moving a sliding-window of size k over the sentence, and applying the same **convolution filter** or **kernel** to each window in the sequence, i.e., a dot-product between the concatenation of the embedding vectors in a given window and a weight vector u, which is then often followed by a non-linear activation function g.

Considering a window of characters $w_1, ..., w_n$, the concatenated vector of the i^{th} window is then:

$$x_i = [w_1, ..., w_n] \in \mathbb{R}^{k \times d}$$

The kernel filters are applied to each window, resulting is the scalar values r_i, each for each window i:

$$r_i = g(x_i \cdot U + b) \in \mathbb{R}^l$$

with: $x_i \in \mathbb{R}^{k \times d}$, $U \in \mathbb{R}^{k \cdot d \times l}$ and $b \in \mathbb{R}^l$.

The 1-D convolution mechanism is detailed in *Figure 4*.

1D convolutions are followed by 1D pooling operations. They combine vectors resulting from different convolution windows into a single l-dimensional vector. In our case we use a 1D *max* pool which takes the maximum value in a frame for each vector. This new vector should have extracted the most relevant features of the sequence.

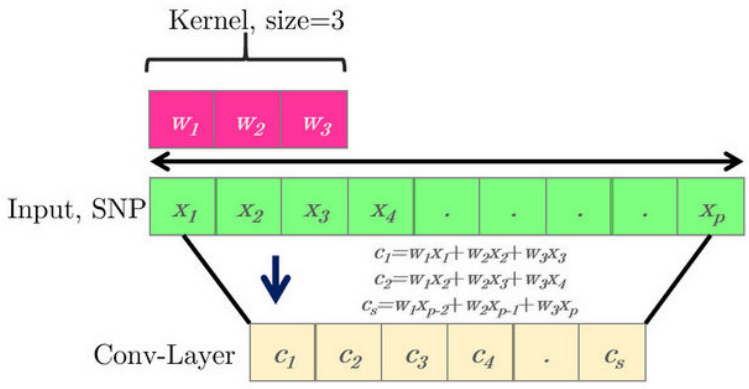


Figure 4: 1D convolution example

3 Modeling and Results

3.1 Input Dataset Pre-Processing

Our input dataset is a protein dataset retrieved from Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank (PDB), at `https://www.kaggle.com/shahir/protein-data-set`.

This dataset has identified each polypeptide by the following columns: `structureId, classification, experimentalTechnique, macromoleculeType, residueCount` alongside other information not relevant to our classification problem. For simplicity and computational time's sake, we seek to pre-process our input dataset in order to shorten it and only keep information relevant to our classification task:

- Keep proteins only : `macromoleculeType == 'protein'`
- Maximum chain length : `residueCount` $\leq$ `2000`
- Removing NANs : `data = pandas.dropna(data)`
- Restructuring the columns to keep relevant information only : `data.columns = ['structureId', 'classification', 'residueCount', 'sequence']`

Additionally, we seek to keep only the 10 most represented family types by instance count (figure 5). The remaining dataset is then sorted, its labels binarized and encoded using `sklearn`'s `LabelBinarizer, OneHotEncoder`, and finally tokenized from character strings to numbers using `keras.tokenizer` for ease-of-training purposes. It is also relevant to pad sequences with `residueCount` strictly less than 2000 with a neutral value (0), as we are working with `pyTorch` which acts on tensors, to make use of the full parallelization capabilities while retaining most information. Each tokenized protein sequence is embedded before being passed into the model. This a first step towards feature extraction. Through this learned representation of amino acids, similar sub-sequences will have similar embedding values.

HYDROLASE	9704
TRANSFERASE	7277
OXIDOREDUCTASE	5544
IMMUNE SYSTEM	2333
LYASE	2177
TRANSCRIPTION	1660
TRANSPORT PROTEIN	1608
HYDROLASE/HYDROLASE INHIBITOR	1457
SIGNALING PROTEIN	1286
TRANSFERASE/TRANSFERASE INHIBITOR	1242

Figure 5: 10 most represented protein classes by instance count

Finally, we opt for a **80/10/10% train-val-test split** of the input dataset to keep a maximum amount of data for training the complex biLSTM model. For each dataset the available amount of unique proteins stands at:

- Train: 27773
- Validation: 3086
- Test: 3429

3.2 Hyper-parameters

Hyperparameters were fine-tuned in order to attain the highest possible test and validation accuracy, while minimizing test and validation loss.

- Optimizer: Adam
- Loss function: cross-entropy
- Learning rate: 0.001
- Number of epochs: 20
- Batch size: 64
- Embedding dimension: 11
- Scheduler γ: 0.1

To optimise the learning rate, the `StepLR` scheduler was used. It decays the learning rate of each parameter group by γ every epoch.

```
%% Hyper-parameters
sequence_length = 2000 % maximum length of sequence
embedding_size = 11
hidden_size = 512
num_classes = 10 % number of different proteins
lr = 0.001 % learning rate
batch_size = 64
n_epochs = 20
%% biLSTM Parameters
num_layers = 1 % number of layers for BiLTSM model
%% CNN Parameters
n_filters = 100
filter_sizes = [3, 10, 20, 40]
```

3.3 Model Training and Testing

The results of the training and testing of our model are organized as follows:

- firstly, we tried using a simple biLSTM layer with a rectified linear unit and dropout (with 30% dropout rate) feeding two fully-connected layers and a log softmax layer for output classification with nodes for each class (six nodes at least for Level 1 protein class prediction).
- secondly, we attempted using a 4-layer convolutional neural network (CNN) with 1D convolution layers and 1D max-pooling. Kernel sizes are set to 3, 10, 20, 40 respectively of 100 filters each, and aim to correctly identify sequence regions of various length which are responsible for a protein's macroscopic structure.
- finally, we concatenated these two individual models and observed a significant performance increase when it comes to the task of protein class prediction. The final model architecture is detailed in figure 6.

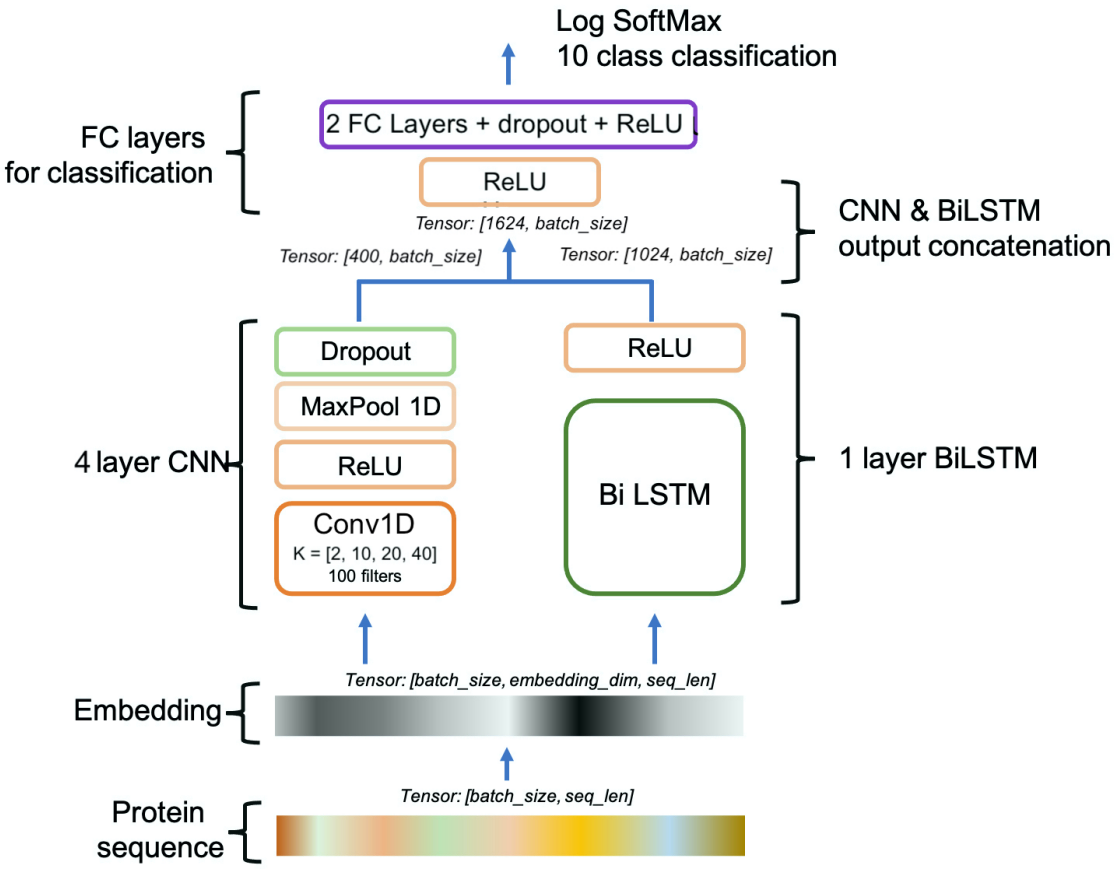


Figure 6: Final model: concatenated 1-layer biLSTM + 4-layer CNN. Each 1D convolution layer is activated using ReLU and pooled with 1D MaxPool. CNN kernel sizes for each layer: [3, 10, 20, 40] of 100 filters each. Fully connected layers sizes: [512, 256].

All models are trained until convergence for 20 epochs on a Tesla P100 16GB GPU available through Google Colab.

3.3.1 1 Layer biLSTM Only

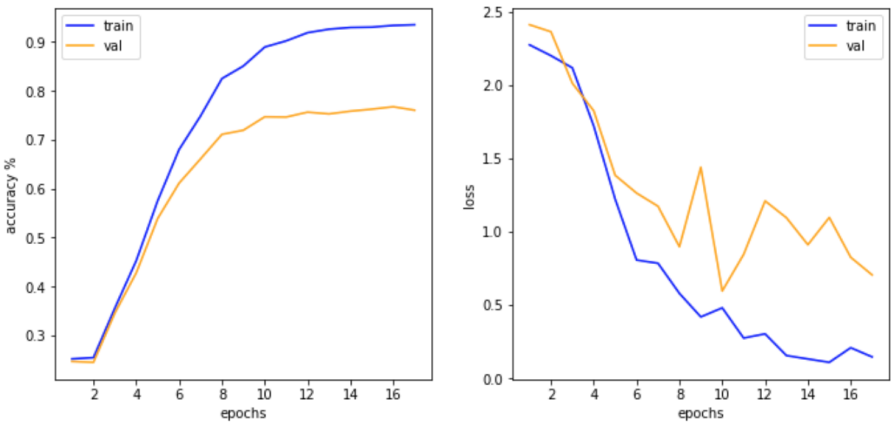


Figure 7: biLSTM-only: validation/training accuracy and loss vs. epoch plots

	Accuracy	Loss
Training	92%	0.2
Validation	73%	0.6

3.3.2 4 Layer CNN Only

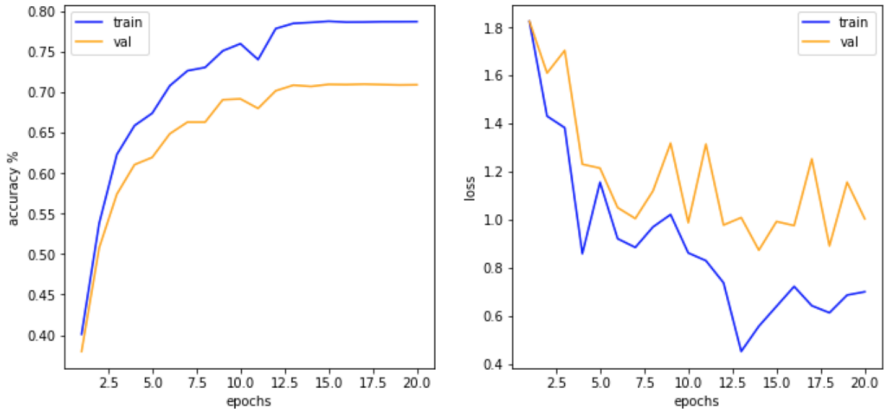


Figure 8: CNN-only: validation/training accuracy and loss vs. epoch plots

	Accuracy	Loss
Training	79%	0.7
Validation	70%	1.05

These simple models yield only 70-75% accuracy on the validation set are are subject to extreme overfitting especially for the BiLSTM. However these models are not fully optimized and the incentive is to concatenate their output to harness both the advantages of each method (namely computational performance advantage and easier to infer features, and context recognition for the CNN and biLSTM respectively).

3.3.3 biLSTM + CNN

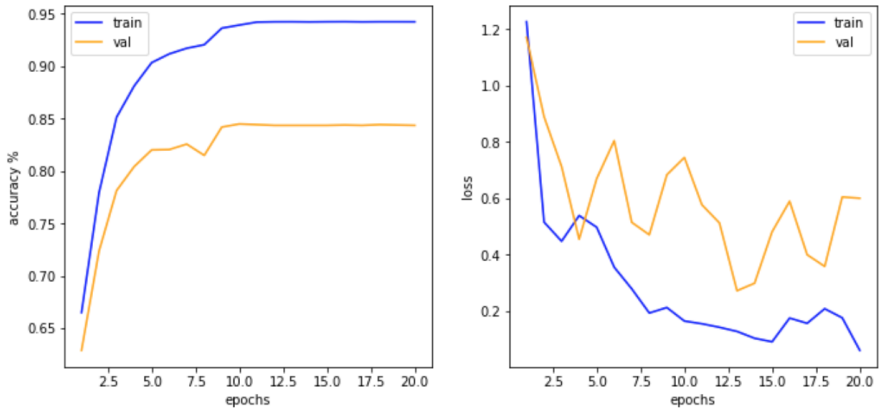


Figure 9: Concatenated biLSTM + CNN final model: validation/training accuracy and loss vs. epoch plots

	Accuracy	Loss
Training	94%	0.17
Validation	85%	0.29

The validation accuracy plateaus at 0.85 whilst the training accuracy is around 0.94. This model, just like the isolated CNN and BiLSTM, is thus subject to overfitting. Dropout between the fully-connected layers helps to tame it, yet it is not sufficient. This is reflected in the validation loss which fails to stabilize after 20 epochs. Nonetheless, the combined model's final validation accuracy remains very respectable (0.85). The minimum validation loss is obtained on epoch 13, the corresponding weights will be used as the final model weights for testing.

3.4 Classification Accuracy and Missed Predictions

Having confirmed that our concatenated biLSTM + CNN network does indeed outperform each singular component by a significant margin, we test the model and use the following metrics to quantify its performance (figure 10):

- **precision**: also called *positive predictive value*, it is the fraction of relevant instances among the retrieved instances.
- **recall**: also known as *sensitivity*, it is the fraction of relevant instances that were retrieved.

- **F1-score**: also known as Sørensen–Dice coefficient or Dice similarity coefficient, it is the *harmonic mean* of the precision and recall.

The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero.

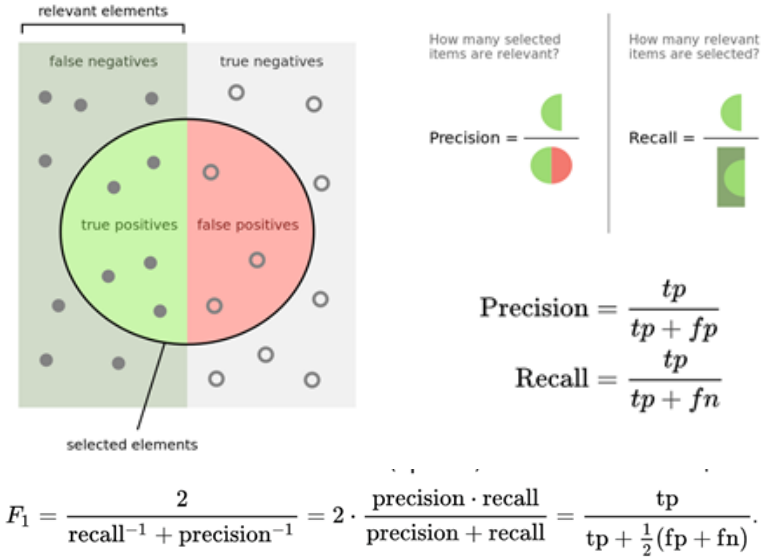


Figure 10: Precision, recall and F1-score as a function of true positives (tp), false positives (fp) and false negatives (fn).

Figure 11 reports the accuracy table of our model's predictions for each of the 10 considered protein families. The model fairs well on the test set with 83% accuracy. It manages to correctly classify most of the proteins (F1-score ≥ 0.8). However *Transferase* and *Signaling protein* are misclassified (F1-score ≈ 0.5). Because we have the true labels for our test set predictions we can plot a confusion matrix to see how certain classes are misclassified. The confusion matrix for our 10 protein families is given in figure 12.

The ideal confusion matrix would be a diagonal matrix. This means that each class is correctly labeled. In our case two protein families are frequently misclassified. *Transferase* is commonly misclassified as *hydrolase*, and *signaling protein* is commonly misclassified as *hydrolase inhibitor*. However the reciprocal is not true.

	precision	recall	f1-score	support
HYDROLASE	0.84	0.89	0.86	993
TRANSFERASE	0.60	0.46	0.52	137
OXIDOREDUCTASE	0.95	0.91	0.93	209
IMMUNE SYSTEM	0.90	0.81	0.86	218
LYASE	0.91	0.93	0.92	545
TRANSCRIPTION	0.70	0.63	0.66	114
TRANSPORT PROTEIN	0.81	0.79	0.80	173
HYDROLASE/HYDROLASE INHIBITOR	0.78	0.85	0.82	722
SIGNALING PROTEIN	0.70	0.48	0.57	130
TRANSFERASE/TRANSFERASE INHIBITOR	0.85	0.77	0.81	151
accuracy			0.83	3392
macro avg	0.80	0.75	0.77	3392
weighted avg	0.83	0.83	0.83	3392

Figure 11: biLSTM + CNN model prediction accuracy for each of the 10 considered protein classes

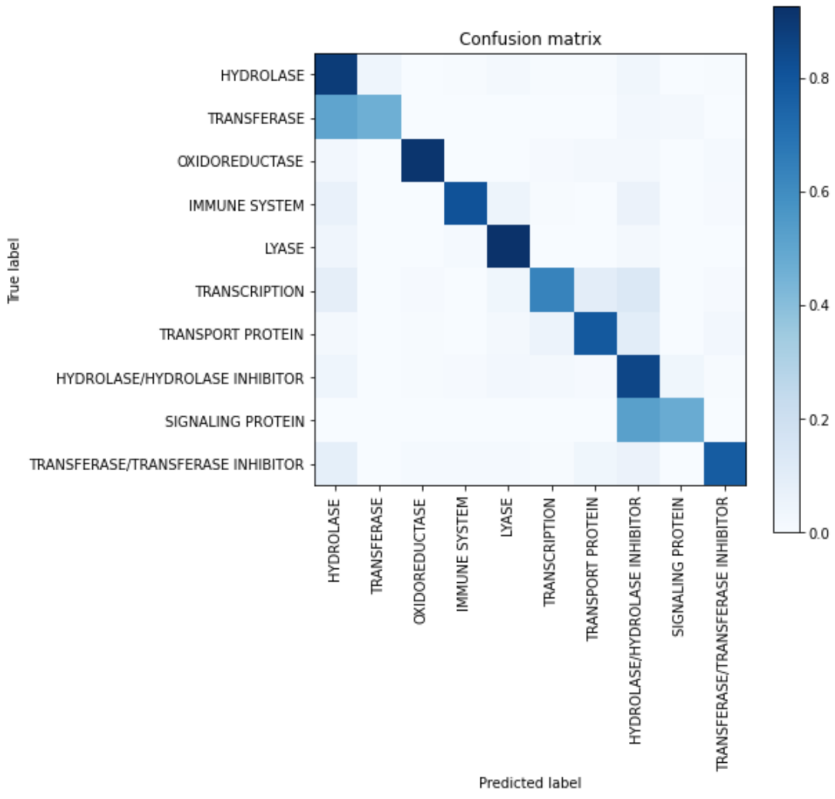


Figure 12: 10 class confusion matrix

4 Discussion and Conclusion

Overall, our BiLSTM + CNN model reaches an accuracy of 85% for our validation dataset. These are encouraging results, Strodthoff et al. (2020) obtained an accuracy score comprised between 0.84 and 0.97 depending on the dataset they used. The baseline model they compared to had an accuracy of 0.74 to 0.95. Our model is within both intervals, and there are still some ways to improve it.

The efficiency of the prediction is influenced by the protein class. This might be due to biological reasons and a bias in our model. Families such as Lyase or Oxidoreductase proteins display high F1-scores. Some of those high accuracy classes are characterized by a low diversity in terms of sequence. As an example, lyases proteins are known for having a lower number of identified catalytic sites (Porter et al., 2004). On the other hand, some families, such as transferase proteins, are misclassified in favor of other classes, hydrolase proteins in this instance. Once again, the explanation could be biological. Some hydrolases and transferases are extremely close to one another. Light et al. (2017) showed that glycoside hydrolases and transglycosylases (two subclasses of hydrolases and transferases respectively) share the same catalytic sites and differ only by a few amino acids. As our initial dataset contains a higher number of hydrolases, the model could have overtrained on the hydrolases close to some transferases, explaining why the misclassification is significant for the later ones.

To partially solve this problem, a custom loss can be implemented in the future, where weights are added to each term according to the number of proteins in the corresponding class. To enhance the global performance, custom filters can be added to the CNN kernels. Also it is possible to explore other machine learning models such as random forests.