Classifying the PFam dataset

Lionel Cheng

May 2022

1 Dataset Analysis

1.1 Overview of the dataset

The PFam dataset Mistry et al. [2020] is a widely used resource for classifying protein sequences into families and domains. It is used by experimental biologists researching specific proteins, by structural biologists to identify new targets for structure determination, by computational biologists to organise sequences and by evolutionary biologists tracing the origins of proteins.

Deep learning has been applied on such datasets in Bileschi et al. [2022]. The purpose of the dataset in this report is to repose the PFam seed dataset as a multiclass classification machine learning task.

The task is: given the amino acid sequence of the protein domain, predict which class it belongs to. There are about 1 million training examples, and 18,000 output classes.

The dataset has been randomly splitted into a training, validation and test datasets. The first entries of the training datasets are shown in Tab. 1. For each entry corresponding to a protein domain, 5 properties are found: family_id, sequence_name, family_accesion, aligned_sequence and sequence.

- sequence_name is a unique identifier for the sequence
- family_id and family_accesion correspond to the labels of our dataset
- sequence correspond to the list of amino-acids of the sequence
- aligned_sequence contains a single sequence from the multiple sequence alignment generated using hidden Markov models [Finn et al., 2007]

The dataset has been splitted so that 80% is contained in the training dataset and 10% in the validation and test datasets respectively. The number of sequences, the number of unique families and the number of unique amino-acids can be found in Tab. 2 for each dataset. We see that the number of families and amino-acids is not the same depending on the dataset: this should be taken into account during training.

family_id	sequence_name	family_accession	aligned_sequence	sequence
Penicillinase_R	Q81U16_BACAN/8-123	PF03965.16	ISEAELEIMK	ISEAELEIMK
Rtt106	POB3_CANAL/362-454	PF08512.12	AGVPCSVKA.	AGVPCSVKAS
F-actin_cap_A	Q8I3I2_PLAF7/12-301	PF01267.17	IRHVLMNSPP	IRHVLMNSPP
HupF_HypC	O28902_ARCFU/1-65	PF01455.18	MCIAIPGR..	MCIAIPGRIE
DUF3794	R6BY75_9CLOT/189-271	PF12673.7	NIFHI..LWE	NIFHILWEDV

Table 1: Example of the first 5 entries of the training dataset. The aligned_sequence and sequence entries have been cropped to the first 10 amino-acids.

Partition	Number of sequences	Number of unique families	Number of amino acids
Train	1086741	17929	25
Valid	126171	13071	22
Test	126171	13071	24

Table 2: Description of the global properties of the three datasets.

1.2 Dataset metrics

The distribution of the 20 most common labels in the datasets is shown in Fig. 1 for the three datasets. We can confirm that the random split has been well performed since the proportion of each dataset is conserved across the shown families (this has also been checked across the other families).

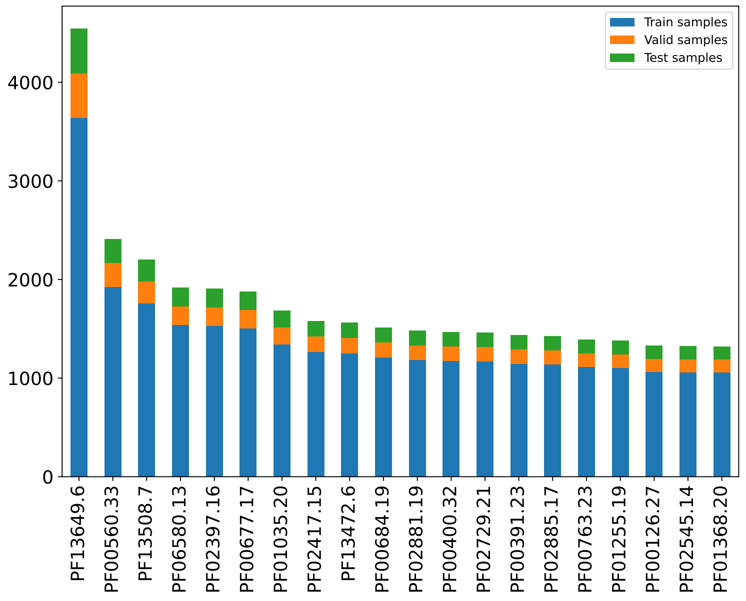


Figure 1: Distribution of the 20 most common labels.

We now turn to the length of the sequences which is not uniform across the datasets as shown in Fig. 2. The distributions are again close to each other and we can see that most of the sequences have a length around 100.

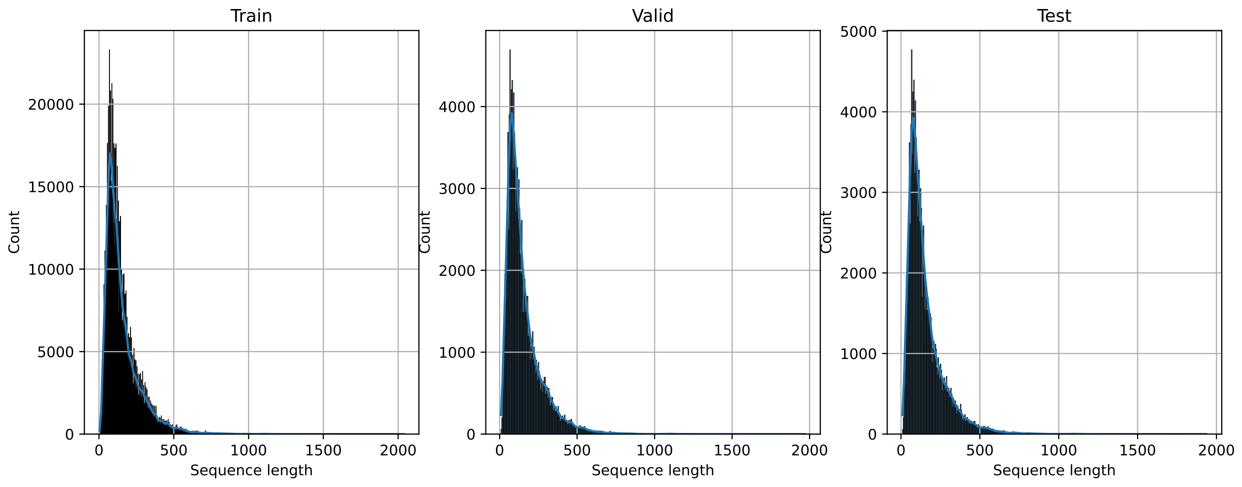


Figure 2: Distribution of the length of the sequences in each dataset.

Finally the distribution of the amino-acids in each dataset is shown in Fig. 3. As said earlier, there are 25, 22 and 24 amino acids in the train, validation and test datasets respectively and the 20 most common ones are the same across the three datasets: these are the 20 most common amino-acids in nature [Lopez and Mohiuddin, 2021]. The other amino acides present in the sequences (X, U, B, O, Z) are really rare and seem to not be really relevant for classification.

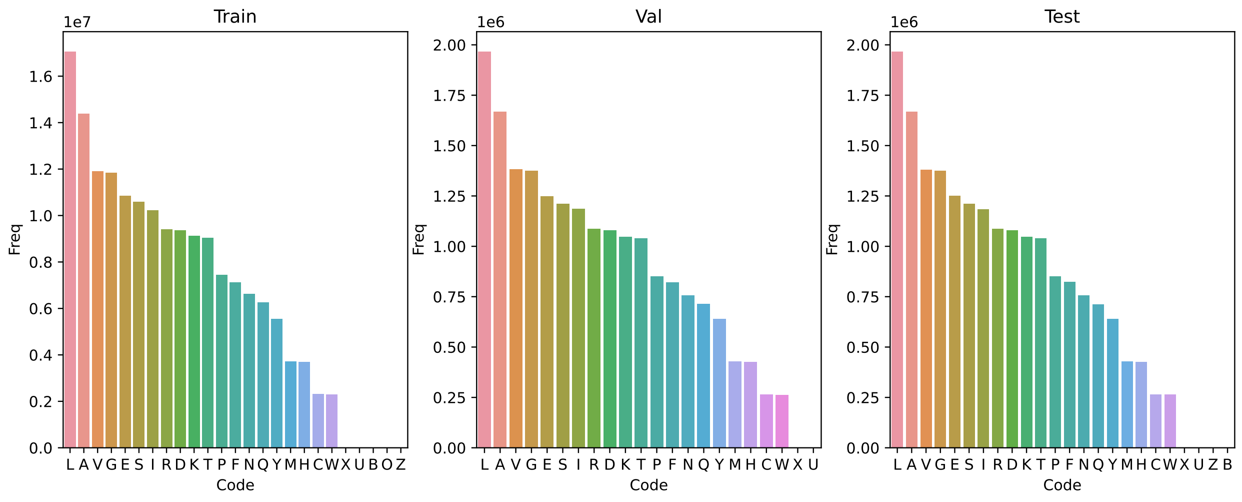


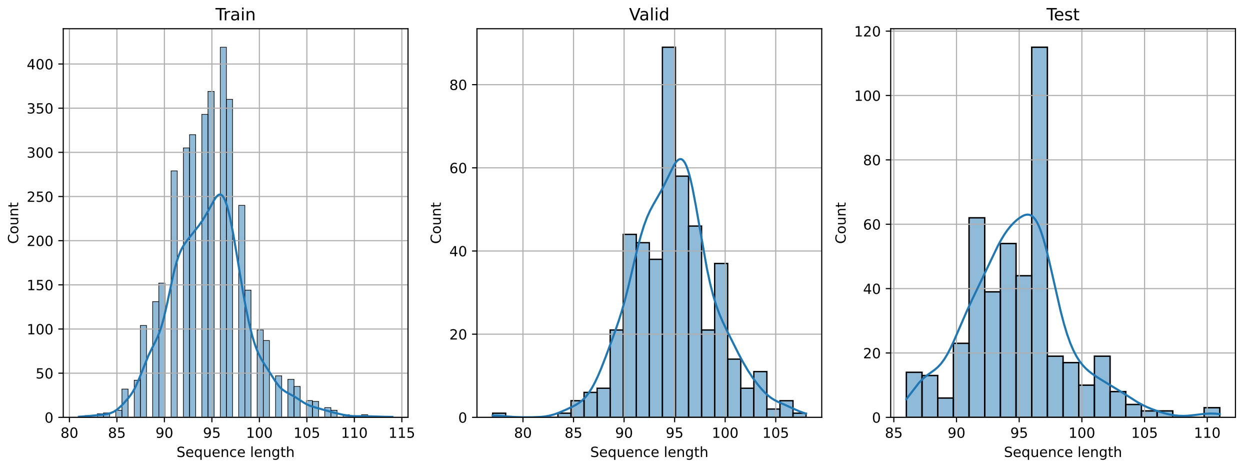
Figure 3: Distribution of the amino acids in each dataset.

1.3 Dataset families

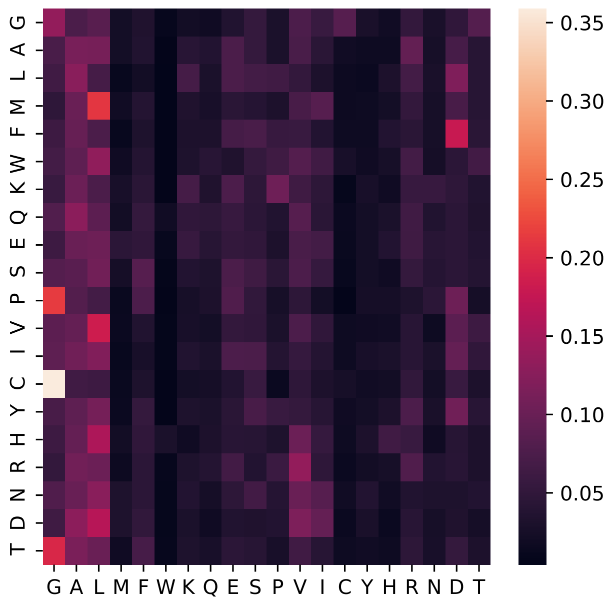
This section focuses on families of the dataset and tries to understand what is unique in each family. These properties should be, one way or another, be understood by the neural network. We choose two families from the 20 most common families in the datasets in Fig. 1:

the PF13649.6 and PF00677.17 families. For each one the sequence length distribution and the transition matrix heatmap are plotted in Figs. 4 and 5. The transition matrix should be read as follows: for each row (associated to a letter) each column correspond to the probability that the next letter is one of the column. Hence for the PF13649.6 family the C amino-acid is mostly followed by the G amino-acid in Fig. 4b.

For each family we can see there is a certain coherence in terms of sequence length for each family: all sequences of a family have similar sequence lengths for the most-common labels. The transition matrix can be considered as a unique footprint of each family.

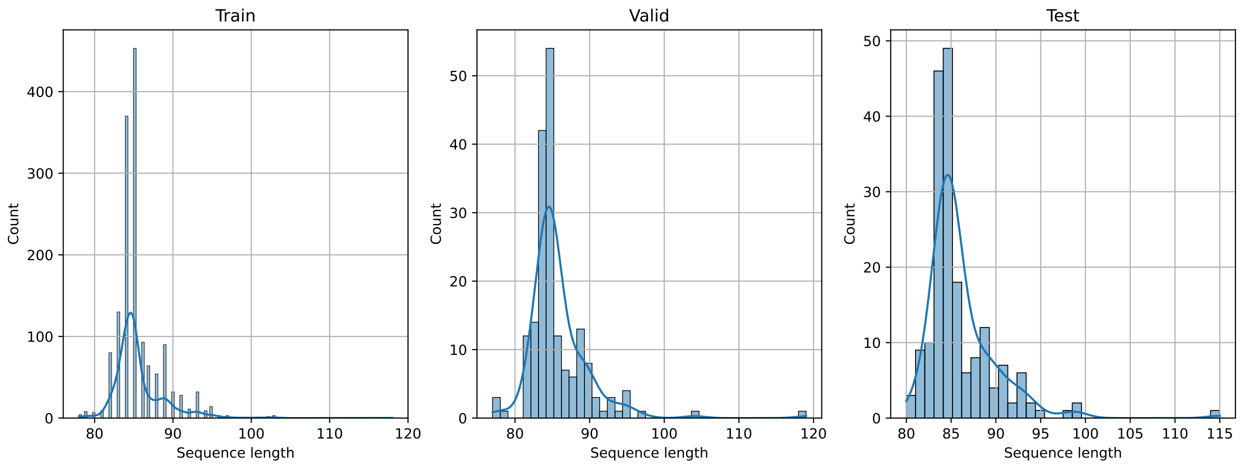


(a) Length distribution

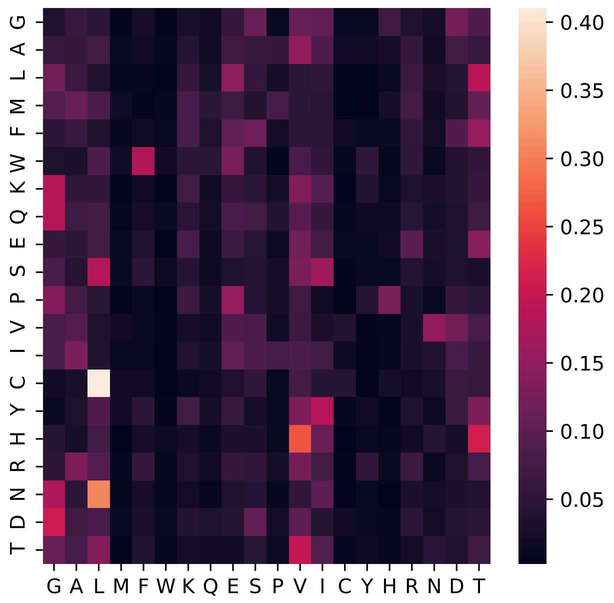


(b) Transition Matrix Heatmap in train dataset

Figure 4: Properties of the PF13649.6 family.



(a) Length distribution



(b) Transition Matrix Heatmap in train dataset

Figure 5: Properties of the PF00677.17 family.

2 Method Explanation

The method used to tackle this classification problem is now explained. The whole code is written using the PyTorch library. First the dataset is reduced and encoded to homogeneize entries and make it suitable for training. Then the neural network architecture is discussed as well as the loss and optimizer used.

2.1 Dataset reduction and encoding

From the dataset analysis performed above, we use 3 parameters to reduce the dataset: the number of common labels n_{mc}, the maximum length of the sequences $l_{\max}$ and the number of amino acids classes n_{aa}.

The number of common labels n_{mc} allows to decrease the complexity of the classification problem so that it is easy to test that everything is working properly before scaling up. We choose the number of most common labels in the **training** dataset and filter the three datasets to include only the sequences whose labels are among this set. The labels are each assigned an integer to create a unique identifier for each label.

As shown in Fig. 3, only 20 amino-acids are really meaningful in this dataset. We create a letter to integer (1 to 20) correspondance to encode each of these 20 amino-acid and set all the other amino acids to 0.

The length of the sequences in the datasets is not constant (Fig.2) and it would be easier to have constant length sequences. To do so we set the maximum length of the sequences to $l_{\max}$ by cutting the sequences or adding padding zeros to the encoded sequences.

Finally the labels are each assigned an integer

All these treatments are done in the `ProteinDataset` class in the `dataset.py` file of the `proteinclass` library.

2.2 Neural network architecture

After dataset reduction and encoding the input matrix $\mathbf{X}$ of a neural network and the target labels $\mathbf{y}$ have the following shape:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \dots & \dots & x_{1,l_{\max}} \\ \vdots & \dots & \dots & \vdots \\ x_{b_s,1} & \dots & \dots & x_{b_s,l_{\max}} \end{bmatrix} \in [\![0, 20]\!]^{b_s \times l_{\max}}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{b_s} \end{bmatrix} \in [\![0, n_{\text{mc}} - 1]\!]^{b_s} \tag{1}$$

where b_s is the batch-size. Each row in this matrix corresponds to an encoded sequence that has been padded to $l_{\max}$ length. At this point the input matrix could be encodded in two ways in the first layer of the neural network: by using a one-hot encoding or a word embedding. The size of the vocabulary (in our case the number of amino-acids) being relatively small (21), there is no curse of dimensionality using the one-hot encoding. We expect certain amino-acids to be more closely related than others and a one-hot encoding can not capture that as it imposes orthogonality between all the amino-acids.

We thus begin our networks with an embedding layer (`nn.Embedding` in the code) where the embedding dimension is n_e. The simplest network (`SimpleProteinClass` class in `model.py` file of the `proteinclass` library) is followed by a fully-connected (FC) layer so that the network is depicted in Fig. 6.

$$\begin{tikzcd}
\boxed{\mathbf{X} \in [\![0, 20]\!]^{b_s \times l_{\max}}} \arrow[r, "\text{Embedding}"] & \boxed{\mathbf{X}_h \in \mathbb{R}^{b_s \times l_{\max} \times n_e}} \arrow[r, "\text{FC layer}"] & \boxed{\mathbf{Z} \in \mathbb{R}^{b_s \times n_{\mathrm{mc}}}}
\end{tikzcd}$$

Figure 6: `SimpleProteinClass` network.

This very simple network has the caveat of having a lot of parameters since the fully convolutional layers directly flattens the embedding matrix. To reduce the memory footprint of the model, we design two other models which add another layer after the embedding and before the FC layer: the `RNNProteinClass` which add one or more Recurent Neural Network (RNN) layers and `LSTMProteinClass` which add one or more Long Short Term Memory (LSTM) layers. This decreases greatly the number of parameters although it is more computationaly expensive.

$$\begin{tikzcd}
\boxed{\mathbf{X} \in [\![0, 20]\!]^{b_s \times l_{\max}}} \arrow[r, "\text{Embedding}"] & \boxed{\mathbf{X}_{h1} \in \mathbb{R}^{b_s \times l_{\max} \times n_e}} \arrow[r, "\text{RNN or LSTM layer}"] & \boxed{\mathbf{X}_{h2} \in \mathbb{R}^{b_s \times h_s}} \arrow[d, "\text{FC layer}"'] \\
 & & \boxed{\mathbf{Z} \in \mathbb{R}^{b_s \times n_{\mathrm{mc}}}}
\end{tikzcd}$$

Figure 7: `RNNProteinClass` and `LSTMProteinClass` networks.

2.3 Loss and optimizer

We use a simple stochastic gradient optimizer [Ruder, 2016] for the optimizer and the loss is the classical cross-entropy loss for categorical data [Bishop, 2006, Chap. 4]:

$$l(\mathbf{Z}, \mathbf{y}) = \frac{1}{b_s} \sum_{b=1}^{b_s} -\log \frac{\exp(Z_{b,y_b})}{\sum_{n=1}^{n_{\mathrm{mc}}} \exp(Z_{b,n})} \tag{2}$$

where $\mathbf{Z}$ is the output after the FC layer in Fig. 6.

3 Experiment Description

To be able to easily debug and understand the network behavior on the dataset we vary the n_{mc} parameter, *i.e.* the number of most-common labels in the training dataset used to filter the datasets. The small set will consist of only the 200 most common labels and will be used to test the different networks architectures and tune the hyperparameters. The medium set contains 5000 labels.

4 Result Analysis

4.1 Small set

We start by tuning the optimizer parameters of the different network architectures on the 200 labels dataset. The training set contains now 156 275 entries, the validation and test 19 426 entries. We vary two parameters that control the optimization: the learning rate and the maximum norm of gradient (above which it is clipped).

The `SimpleProteinClass` is first tested by setting the maximum norm of gradient to 5 and varying the fixed learning rate from 0.001 to 1.0. The embedding size of the first layer is set to 40 so that the network contains 801 040 trainable parameters. The training and validation loss/accuracy are shown in Fig. 8: the optimal value of the learning rate for this case is between 0.1 and 1.0.

The training of the `LSTMProteinClass` is longer than the `SimpleProteinClass` for all learning rates studied but it manages to get a rather good accuracy for $l_r = 0.1$ as shown in Fig. 9.

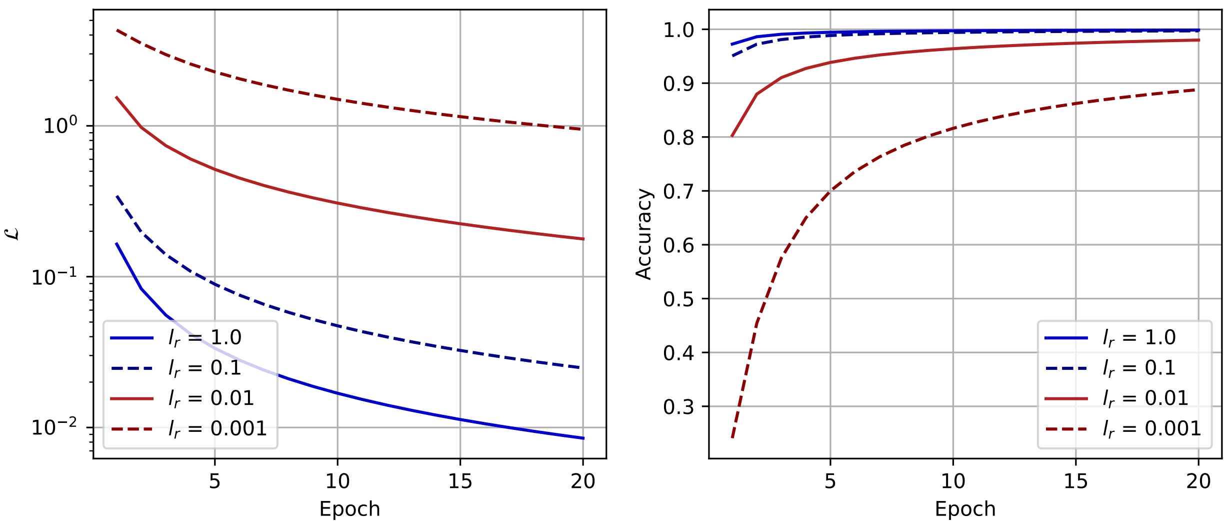
Training of the RNN network proved to be unsuccesful. Hyperparameter tuning should be more thoroughly explored for improvement. The three best networks of each architecture are compared in Fig. 10.

4.2 Longer sets

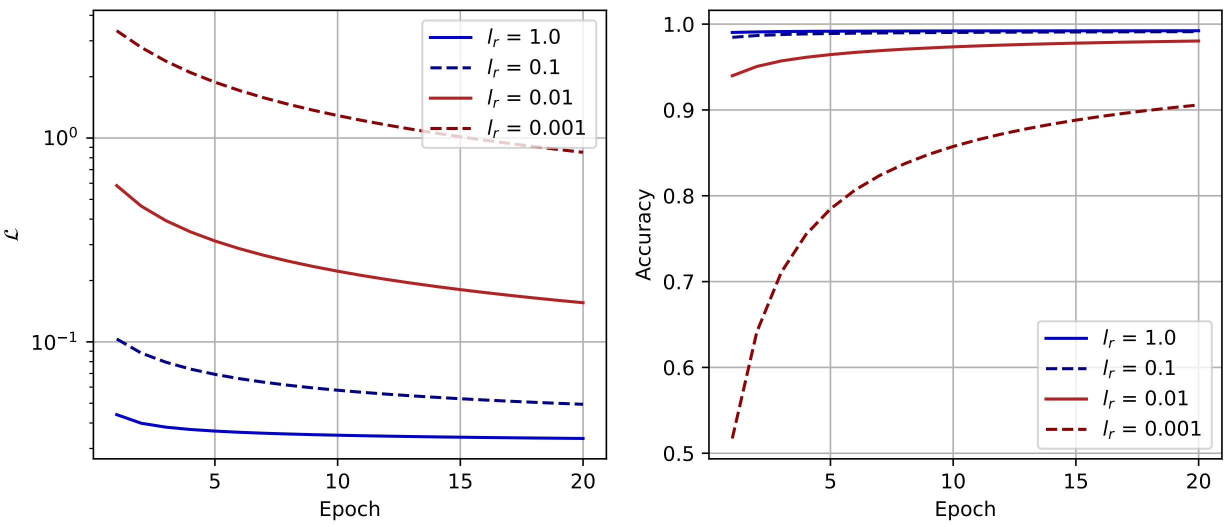
We increase the number of included labels in the sets by taking into account 5000 labels which brings the data sets closer to reality (18 000 labels in the original data set). The `SimpleProteinClass` with the optimal hyper-parameters found earlier work well on these datasets as well. Increasing further the number of labels to 15 000 yield good results although training should be extended to more than 20 epochs to really get the best accuracy as shown in Fig. 11. The network start to get really big with respectively 20 005 840 and 60 015 840 parameters for the 5000 and 15 000 label networks respectively.

5 Conclusion

A method for classifying the PFam dataset has been carried out in this document. The dataset analysis shows a certain repartition of length and amino-acids which leads to a first reduction of the dataset to homogeneize it. Restricting to only the 20 most common amino-acids the sequences, an embedding has been chosen over a one-hot encoding for each sequence. Three networks have been tested throughout this document and the simplest one has proven to yield the best results. More hyperparameter tuning is necessary for the other models to get better accuracy and could be carried out in the future.



(a) Training dataset



(b) Validation dataset

Figure 8: Loss and accuracy.

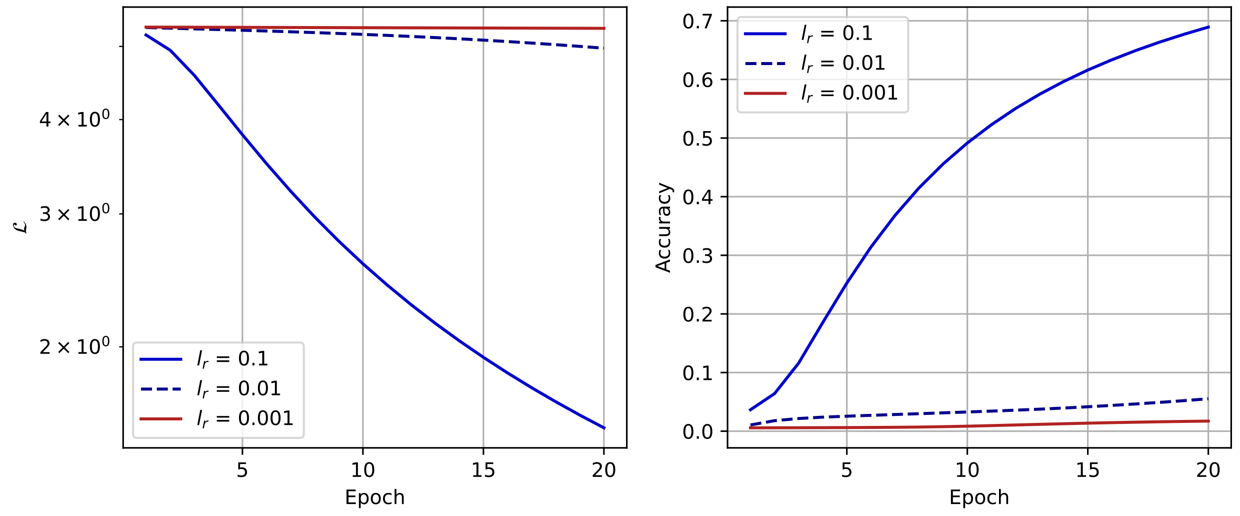


Figure 9: Loss and accuracy for training dataset.

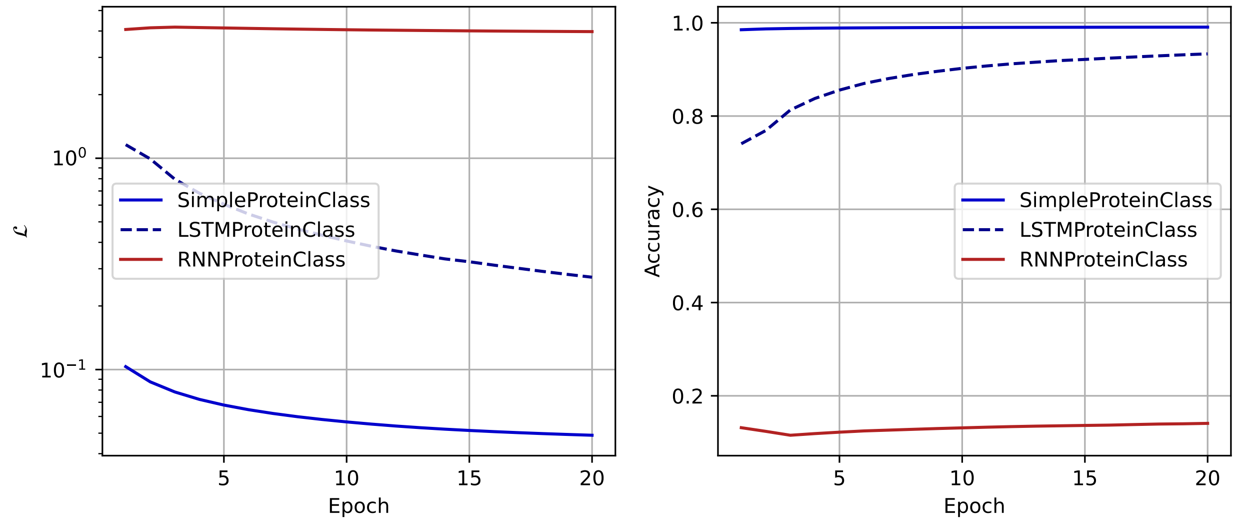


Figure 10: Loss and accuracy for the validation dataset using the 3 different architectures.

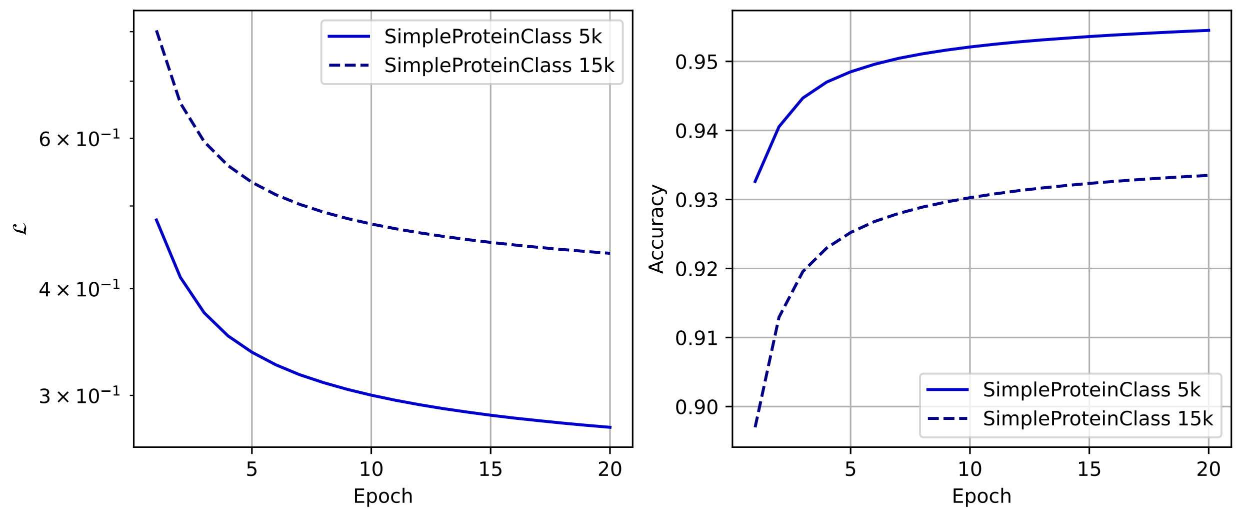


Figure 11: Loss and accuracy for the validation dataset using 5000 and 15 000 labels datasets.

References

Jaina Mistry, Sara Chuguransky, Lowri Williams, Matloob Qureshi, Gustavo A Salazar, Erik L L Sonnhammer, Silvio C E Tosatto, Lisanna Paladin, Shriya Raj, Lorna J Richardson, Robert D Finn, and Alex Bateman. Pfam: The protein families database in 2021. *Nucleic Acids Research*, 49(D1):D412–D419, oct 2020. doi: 10.1093/nar/gkaa913. URL https://doi.org/10.1093%2Fnar%2Fgkaa913.

Maxwell L. Bileschi, David Belanger, Drew H. Bryant, Theo Sanderson, Brandon Carter, D. Sculley, Alex Bateman, Mark A. DePristo, and Lucy J. Colwell. Using deep learning to annotate the protein universe. *Nature Biotechnology*, feb 2022. doi: 10.1038/s41587-021-01179-w. URL https://doi.org/10.1038%2Fs41587-021-01179-w.

R. D. Finn, J. Tate, J. Mistry, P. C. Coggill, S. J. Sammut, H.-R. Hotz, G. Ceric, K. Forslund, S. R. Eddy, E. L. L. Sonnhammer, and A. Bateman. The pfam protein families database. *Nucleic Acids Research*, 36(Database):D281–D288, dec 2007. doi: 10.1093/nar/gkm960. URL https://doi.org/10.1093%2Fnar%2Fgkm960.

MJ Lopez and SS. Mohiuddin. Biochemistry, essential amino acids. https://www.ncbi.nlm.nih.gov/books/NBK557845/, 2021.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.