Case study report

Study of the PFAM seed random split dataset

Emilien Vimont
Research Engineer - Technical Test

Version of the 24/09/2022

Executive Summary

The goal of the experiments carried out is to build a protein classifier. Each protein in the PFAM dataset will be assigned its corresponding Pfam family (i.e. protein family). Pfam is a dataset of protein families that includes their annotations and multiple sequence alignments generated using hidden Markov models. The PFAM seed random split dataset, was released in March 2021 and contains 18,000 families.

Protein sequence classification has widely utilized the technologies of text categorization from natural language processing. Benefiting from deep learning models and word embedding methods, text categorization has achieved great progress, which also brings opportunities for improving the performance of protein classification tasks. Nonetheless, the tokenization process, the choice of the dictionary used to perform those models and the length of the sentences studied are different than the ones encountered in classic NLP problems.

Protein sequence classification is thus a problem relying more on NLP techniques than on more classic approach. Deep Learning and attention-based architectures have already shown their superiority for classifying this type of data. The sequence alignment argument for the sequence was not used for training the classifier, only the sequence was considered. The baseline model chosen for this case study is thus a classical LSTM. This LSTM [1] will then be compared to a more advanced architecture which is the one described in Very Deep Convolutional architecture for text classification [2]. At this stage, those methods already provided significantly good results regarding our metrics. However, in order to try state-of-the-art methods on this specific dataset, attempts to use ProtTrans [3](BERT-based model for protein sequence classification) embeddings have been made to train a small neural network in order to illustrate the performance of this model regarding one experiment.

One must bear in mind, that the limitation of computer resources is the main obstacle to have the best classifier. Several steps were therefore taken in order to establish the most relevant model with regard to resource constraints.

Table des matières

1 Structure of the study

The purpose of this report is to simply summarize the steps that have been taken and to report on the performance obtained. More details are available in the different scripts developed.

You can find all the Python scripts developed in the src section. *train.py* is the script to run to train the different implemented models. The *config.py* file gathers the parameters necessary for the proper functioning of the *train.py* script, the change of these values leads to the change of the performances of the different models. These parameters have not been tuned during this study. *engine.py* groups the evaluation and training functions of the PyTorch model. A class corresponding to the studied database is also available in the script *dataset.py*. Finally, the script *createfolds.py* aims to separate the input database into 5 different folds.

The input folder gathers all the input data provided to the different algorithms. *mutli-class.pkl* files correspond to the dictionnary indexing the protein families chosen for creating the different datasets. Each dataset is identified by the name of the dataset concerned followed by the number of families studied in this dataset. Some databases have the attribute *fold* meaning that these databases are used to be evaluated according to the K-Fold methodology.

The models folder gathers some models that have been trained as classifiers.

These databases are initially obtained through the notebook *exploration.ipynb* in the notebook section of the project. This notebook was used to perform the dataset analysis step and also to select the studied families. A Keras implementation of the Very Deep CNN architecture [2] is trained and evaluated in this notebook. If the user does not have access to GPU, one can execute the notebook *kfoldevaluation.ipynb* on Colab for using the Colab GPU.

2 Dataset analysis

The notebook *exploration.ipynb* describes the analysis related to the dataset. The key take-aways from this analysis are the following :

— The format of the data and the environment of Google Colab limit the number of classes that can be chosen (despite having tried to reduce the size of the data in order to optimise the management of the ram of colab)
— There are too many classes (17k), even for such a large database which justifies the desire to select only a limited number of families.
— There is an imbalance on the repartition of the different families across the different sets (training, validation,test) which has been resolved using an imbalance metric. The families selected for training and evaluating the models are those which respect this imbalance metric.
— The number of families studied are : 176, 570 and 1422
— The distribution of the length of the families selected in the final dataset follows a Poisson distribution with lambda = 100. The maximum sequence length is thus set to 100 in the first case and 150 in the other case.
— Dictionaries mapping the amino acids to an index and families to an index are output. Another output is the concatenation of the train,validation and test datasets.

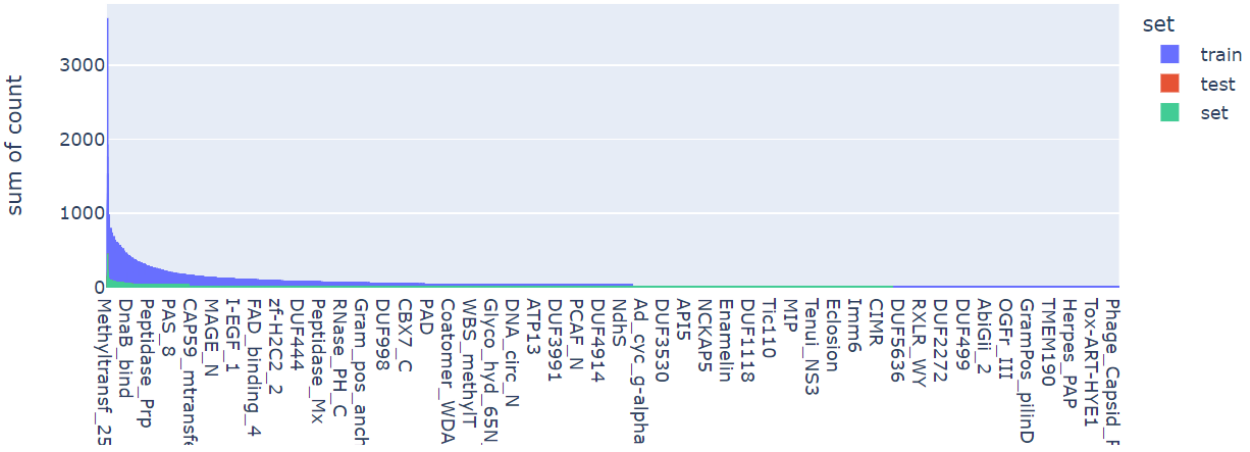


FIGURE 1 – Distribution of the samples per protein families in the dataset. The dataset is imbalanced

3 Method explanation

The classification problem is a multi-class problem with a high number of classes. The model performances will be evaluated by looking at the accuracy, recall, precision and loss obtained after training the model on different folds. The loss used will be the Cross Entropy which is more adapted to multi-class problems.

3.1 Model evaluation

Two evaluation methods are used to evaluate the model.

3.1.1 Classic evaluation

The first is to rely on the balanced dataset obtained in the *exploration.ipynb* notebook. The training set is used for training, the validation set for validation and the test set for measuring the performance of the model. These sets have been reworked in order to balance the representation of the classes, the representation of the different classes is supposed to be "optimal". **One should remember that this process of modifying the test set involves high scores for the metrics we studied which is not representative of the reality. A more proper way to study these metrics would have been not to remove samples from the test set.**

3.1.2 Stratified K-Fold evaluation

A second approach presented in the *train.py* script consists in concatenating the different data sets established in the *exploration.ipynb* notebook. A stratified K-Fold in 5 different folds is applied to this new data set, the model is then trained on the first 4 folds and evaluated on the last one. There is no test set here.

The final metrics are then the average of the previous evaluation metrics applied on the different folds (validation sets). The aim of this approach is to evaluate the performance of the model on the whole dataset and not to limit the evaluation to the test validation datasets provided.

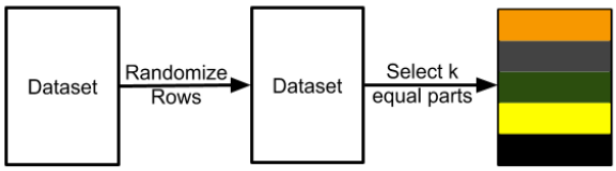


FIGURE 2 – K-Fold cross validation.

3.2 Selecting the number of classes studied

As mentioned in the section concerning the dataset analysis, a number of families have to be selected. The model is therefore evaluated to distinguish the best represented classes within the database (which is debatable : see in the *exploration.ipynb* notebook). The best represented classes are those families with an imbalance score > 0.125.
The imbalance score is described as follows :

$$imbalance_{test} = \frac{size_{test}}{size_{training}} \tag{1}$$

If the balance is respected, the imbalance score should be 0.125. Therefore, the classes that do not reach this score are not retained. This filter could be reduced to 0.12 but 0.125 was chosen so as to be sure the families are well balanced across the sets.

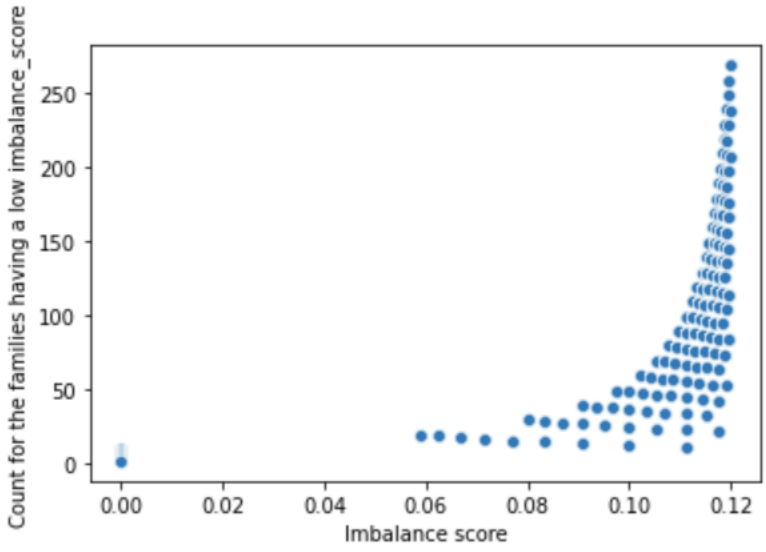


FIGURE 3 – This figure represents the distribution of the numbers of examples per families that have an imbalanced score inferior to 12 in the dataset. This shows that having a low imbalance score is correlated to a low number of examples representing the family in the dataset.

Once the protein families meeting this requirement have been selected, we also choose to select only those families that appear at least 5, 25 and 150 times in the training dataset (**which leads to the study of 1422, 736 and 176 protein families respectively**). Evaluating the model on classes appearing less than 5 times in the database would be counterproductive, although means can be put in place if these classes are of interest, these means can be described in section "Ideas to explore". The table 1 shows information about the datasets studied.

Name	Nb. of families	Training set size	Validation and Test size	Minimum number of samples for a family
PFAM 176	176	58,240	7,280	150
PFAM 736	736	94,912	11,864	25
PFAM 1442	1442	104,480	13,060	5

TABLE 1 – Information about the three databases constructed for different numbers of selected families.

3.3 Architectures studied

In order to assess the difficulty of the task regarding the choices of families made, a simple LSTM is chosen as a baseline for the task, the architecture has indeed been proven to be robust for text classification tasks [1]. This model is one of the first to introduce the long-term memory to a recurrent neural network with the introduction of cells, ,input, output and forget gates.

The performances of the model will then be compared to the model described in the paper Very Deep Convolutional Networks for Text Classification [2] with a depth of 17 convolutional layers. This architecture is now a quite old model relying on character-based input. The character-based aspect seemed interesting if ones wants to change the dictionary from amino-acids to sequence of amino-acids, this process has however not been done in this study.

The implementation of the two architectures is done using PyTorch in the *lstm.py* and *vdcnn.py* files. A Keras implementation is also available for the VDCNN (Very Deep Convolutional Network) architecture in the *exploration.ipynb* notebook. Finally, an implementation of a classifier based on the ProtTrans [3] embeddings has been done in the *prottrans.py* based on the work from a user on Kaggle. The embeddings selected are T5 embeddings.

3.4 Pre-processing and tokenization

The same pre-processing steps were followed for the LSTM and the VDCNN. The sequences were first put in lowercase and then each amino acid was placed in a dictionary where it was associated to a specific index. These indices were then used to create a One-Hot-Encoding of each sequence. A padding of length 100 is then applied to all the sequences. The value of 100 is chosen with respect to the size distribution of the different protein families studied in the final database. The impact of this value on the performance of the model would however be interesting to study. At the input of the model, a sequence has coordinates of dimensions 100 x | dictionary size |.

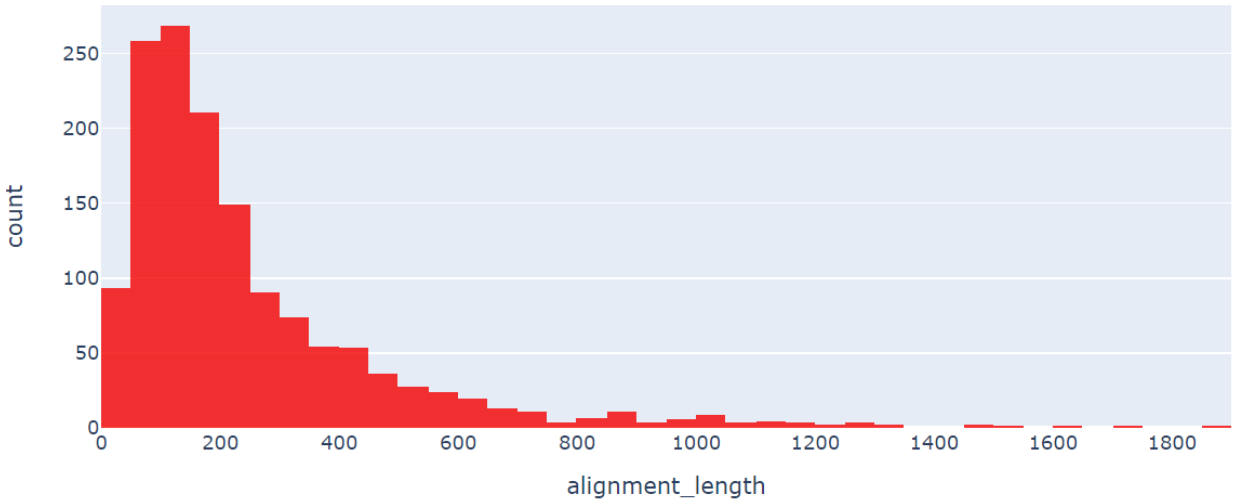


FIGURE 4 – Distribution of the length for protein families in the dataset taking 176 families. The distribution seems to follow a Poisson distribution with $= 100$.

4 Experiment Description

The experiments conducted consist in training and evaluating the two architectures presented above with different dataset content. The simple execution of the notebook *exploration.ipynb* allows to train the VDCNN model implemented in Keras and to obtain the evaluation metrics related to it according to the classic evaluation method. In order to take into account the volume disproportion between the different classes, data are re-balanced by putting less weights on the majority class instances. Specifically, the weight is inversely proportional to class frequency.

The models are evaluated based on the final cross-entropy loss obtained, the accuracy, the recall and the precision. A confusion matrix is displayed to know the protein families for which the accuracy is low (even if it is not a proper metric but a visualization). These metrics are obtained for the two evaluations described before. The table 2 provides results obtained for the two architectures with different number of families studied for the classic evaluation process.

The table 3 provides the results for the K-Fold evaluation method. Since the performance of the models was already satisfactory, no hyperparameters tuning was performed. The learning rate is therefore set to 0.001 using an Adam optimizer which is an adaptive optimizer combining the advantages of many of these optimizers. The models are each trained on 10 epochs. The train batch size was set to 256 and the valid batch size to 128.

5 Results analysis

The results obtained for the different classifiers trained can be observed on the table 2 for the classic evaluation and on the table 3 for the K-Fold evaluation.

Model	Test accuracy	Test loss	Test precision	Test recall
VDCNN for PFAM 176	0.99	0.05	0.99	0.98
VDCNN for PFAM 736	0.97	0.18	0.98	0.96
VDCNN for PFAM 1442	0.96	0.21	0.98	0.96
VDCNN for PFAM 176 with class weight	0.99	0.05	0.99	0.99
VDCNN for PFAM 736 with class weight	0.97	0.18	0.98	0.97
VDCNN for PFAM 1442 with class weight	0.96	0.21	0.98	0.96
VDCNN for PFAM 1442 with Gloriot initilization, Dropout and batch normalization	0.96	0.20	0.98	0.95
LSTM for PFAM 176	0.90	0.89	0.90	0.88
LSTM for PFAM 736	0.87	1.08	0.88	0.85
LSTM for PFAM 1442	0.84	1.21	0.83	0.78

TABLE 2 – Classic evaluation of the models after 10 epochs.

It appears that both architectures provide very satisfactory results for 176, 747 and 1442 families. A study of an even higher number of families would have allowed to measure more precisely the limits of each model. It is noteworthy to notice that the performances of the VDCNN are however superior to those of the LSTM for the two first datasets.

However, the K-Fold evaluation method shows that the performance of the model tends to drop particularly (loss of 10 precent on each of the metrics approximately) with a greater number of families studied. This is due to the fact that the number of copies for some families is between 5 and 10 for this dataset, which is extremely low. The addition of a class weight allows a slight increase in performance for the VDCNN. The figure 6 shows that the few percent of accuracy missing from the model is due to the failure in predicting a significant number of classes. It is noticeable that all the classes seem to be well classified on the figure 6a corresponding to the study of 176 families. A straight line on the confusion matrix indicates that each class is well predicted. Figure 6c, on the other hand, corresponds to the

study of 1442 families, and shows that the diagonal of the confusion matrix is no longer as clear-cut as before and that the model is therefore less accurate for many classes. If the metrics of the table show little change, it finally appears that the classes added to PFAM 1442 are responsible for the drop in precision and recall.

A modification of the batch size for the training from 256 to 16 and of the validation batch size from 128 to 8 led to a significant improvement for the results of the LSTM for 176 families. The table 3 does not show it but it evolved from an average score of 0.87 for the accuracy to a score of 0.98 which is in line with the results proposed in [4] for convolutional architectures but on a different task. However, the conditions of the experiment set up show a clear superiority of the VDCNN on all the metrics studied and on both evaluation methods for 176 families. However, the LSTM has more constant performances and has even slightly better performances when it comes to 1422 families studied. [2] indeed restrained the study of the Very Deep Convolutional architecture to 14 classes at most which may explain this drop in performances. The training and inference time is however slightly longer for the VDCNN.

Model	Mean accuracy	Mean precision	Mean recall
VDCNN for PFAM 176	0.95	0.95	0.94
VDCNN for PFAM 736	0.88	0.85	0.82
VDCNN for PFAM 1442	0.85	0.75	0.70
LSTM for PFAM 176	0.89	0.89	0.87
LSTM for PFAM 736	0.87	0.86	0.80
LSTM for PFAM 1442	0.85	0.79	0.71

TABLE 3 – K-Fold evaluation of the models after 10 epochs for each fold. Each metric studied is averaged over the 5 folds that served as validation folds.

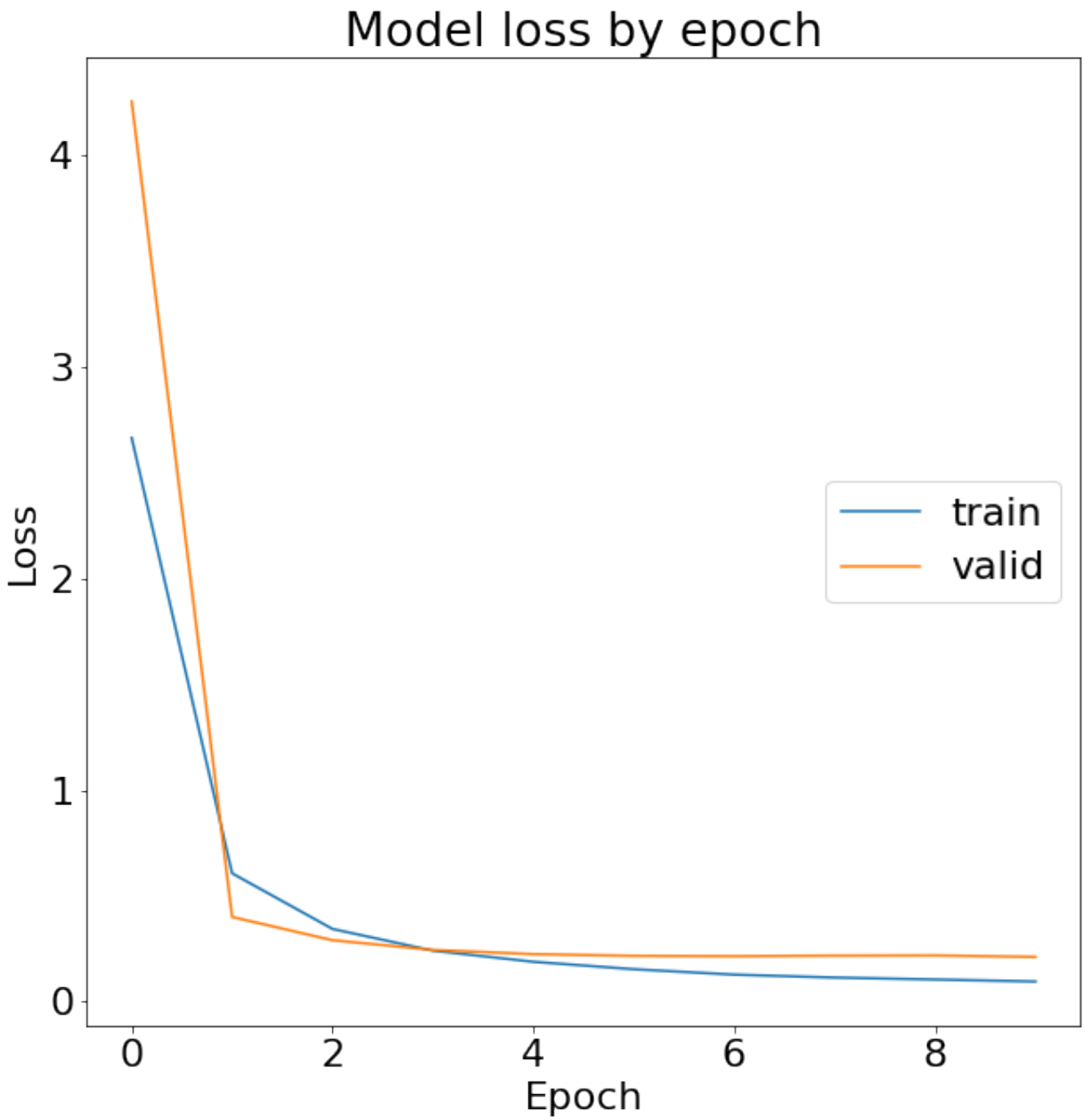
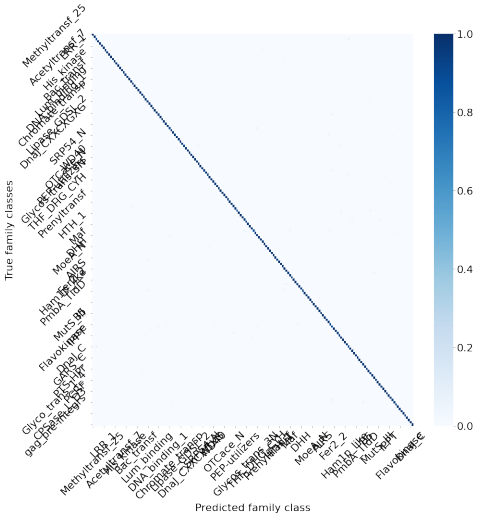
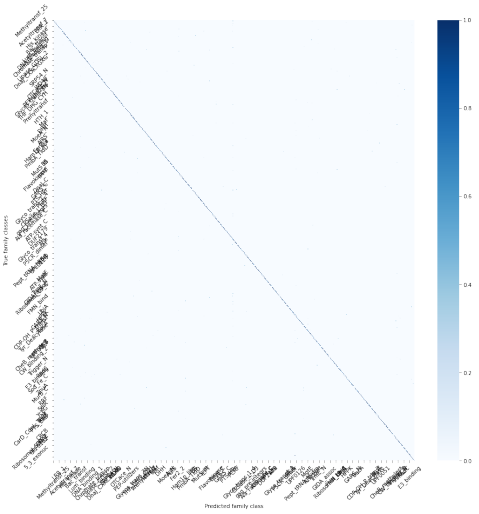


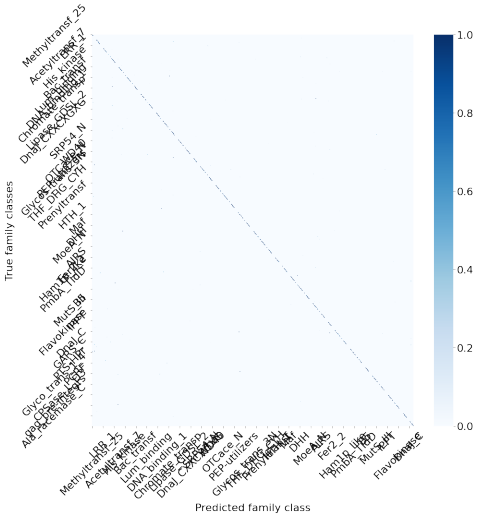
FIGURE 5 – Evolution of the loss for the ten epochs for the VDCNN architecture.



(a) Confusion matrix for a classic VDCNN trained on the PFAM 176 dataset.



(b) Confusion matrix for a classic VDCNN trained on the PFAM 746 dataset.



(c) Confusion matrix for a classic VDCNN trained on the PFAM 1422 dataset.

FIGURE 6 – Confusion matrixes illustrating that inequality in class prediction

6 Conclusion

6.1 Key takeaways

The main issue of the construction of this classifier was to manage the large number of classes and the limit of computer resources. Therefore, choices had to be made regarding the definition of a good classifier and its realization. In the absence of a business expert who could help in the choice of protein families to be used in a sequence classification task, it was chosen to focus on the classification of the most frequent and best distributed protein families in the dataset.

The experiments carried out show that beyond approximately 746 protein families, some classes have difficulty in being predicted in a completely correct way. A measure to quantify the accuracy of the classifier for frequent and rare protein families would have supported the hypothesis that the lowest ranked families turn out to be the least frequent. Great importance was also given to the imbalance score, however this has no real value in the K-Fold evaluation procedure since a Stratified K-Fold is applied to the whole database beforehand. A study more focused on the frequency of proteins within the database and not on the imbalance metric would have been more valuable in case this was the only evaluation method considered.

Finally, the two models presented still provide satisfactory results for the task initially set. The VDCNN seems to be better at handling a smaller number of families while the LSTM seems to be quite robust with respect to the global metrics presented.

6.2 Ideas to explore

The limitation of using Google Colab finally prevented the evaluation of the classifier trained on ProtTrans embeddings, the objective was to use these embeddings to train a logistic regression or another simpler architecture in order to compare the quality of these embeddings to the more classical models presented previously. This method would have been worth to try in order to classify the rarest protein families.

An interesting piece of information would also have been to know the categories of families that exist within the families presented. Indeed, if most of the families under-represented in the dataset are sub-families of more frequent families, it would have been valuable to develop a model that could determine large families and then use the weights of this model to re-train a model that would be more accurate.

Références

[1] cie. Klaus Greff Rupesh Kumar Srivastava. "LSTM : A Search Space Odyssey". In : (2015).

[2] cie. Alexis Conneau Holger Schwenk. "Very Deep Convolutional Networks for Text Classification". In : (2017).

[3] cie. Ahmed Elnaggar Michael Heinzinger. "ProtTrans : Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing". In : (2021).

[4] Castelli Kandel. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset". In : (2020).