A Spike protein sequences embedding approach for pattern recognition analysis

Pattern Recognition

Author: Manuela Carriero

Dated: February 21, 2022

Contents

Abstract

The aim of this study is to exploit machine learning methods to extract important features which characterize protein sequences of the novel Spike (S) protein, that is found on the outside of the SARS-CoV-2 virus particle and gives coronavirus viruses their crown-like appearance. In particular, we train a LSTM neural network to classify Spike protein sequences according to their host species. Although in a supervised machine learning task the goal is usually to train a model to make predictions on new data, in this analysis, the predictions can be just a means to an end. What we want is to get the embedding weights, that is the representation of proteins sequences of symbols as continuous vectors. Thus this approach is a sequence embedding that is learned as a part of a deep learning model classifier (built with Python using Keras libraries). Then through dimensionality reduction techniques (PCA, t-SNE and UMAP), we "embed" these vectors in a low dimensional space in order to visualize our data in a plot. In this new 2D space, some important features come out such as that sequences from bat hosts are near to humans and that human protein sequences arrange themselves in a more regular shape with respect to animals, which suggests a "more regular and less random" variability in Spike protein sequences from human host species.

1 Introduction

1.1 Neural networks and deep learning models

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain [3].

Human brain is full of neurons which are cells able to exchange signal that can vary in intensity. In biophysical terms, this signal is the action potential that is an electrical signal transmitted by a neural cell across the axon. In figure 1 we see the anatomical structure of a real neuron: incoming signals run through dendrides connecting to the body of the cell that then performs an integration of these signals and produces an output that will be provided to an other neuron connected through an axon to other dendrides and so on.

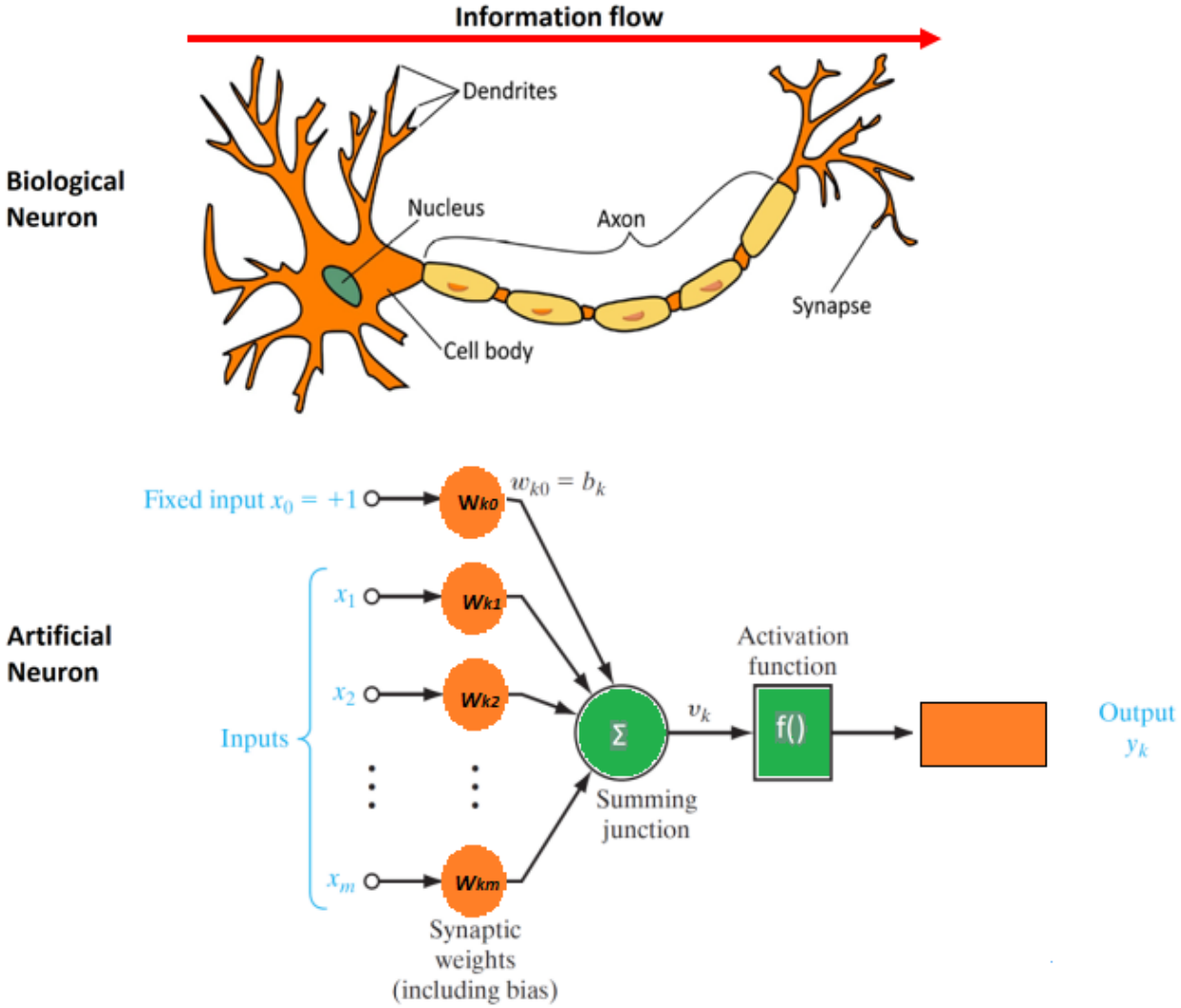

Figure 1: Sketch of biological neuron (above) and artificial neuron (below).

Figure 1 shows also the "artificial" neuron. It represents the fundamental information-processing unit of an artificial neural network. The model of a neuron is composed of three basic elements:

1. A set of **synapses**, or connecting links, characterized by **weights** or strength (i.e. the efficiency of the propagation of signals from one cell to another [41]), and a signal x_i at the **input** of synapse i connected to a neuron k. It is multiplied by the synaptic weight w_{ki} (k-*th* neuron, i-*th* input component).

2. An **adder** to sum input signals weighted by synaptic strengths (linear

integration).

3. An **activation function** for limiting the amplitude of the output of a neuron.

Thus, in few words, we can model, from a mathematical point of view, a neuron as a set of inputs which are processed with a set of weights into the computational unit of our neuron. This is the simple Rosenblatt Perceptron model whose output is a linear combination of the inputs x_i weighted by the weights w_{ki}:

$$\nu_k = \sum_{i=1}^{m} w_{ki} x_i \tag{1}$$

and then processed by an activation function:

$$y_k = f(\sum_{i=1}^{m} w_{ki} x_i) \tag{2}$$

In this way greater is the synaptic weight, greater is the output, and vice versa.

However, this model has matematical limits: since the output is a simple linear combination, such a neural network classifier would be able to solve only linearly separable problems with two classes. If we have two classes of data points linearly separable, we can train our simple perceptron model that learns from the pattern of data so that the output is a straight-line (whose parameters, slope and intercept, are given by the weights of the trained model) that divides the two classes. But no more. Thus, it can solve AND logical problems but it already fails with XOR types of problems.

From here the needing to build a more complex model with more than one neuron and so, combining together simple perceptron models in which the next one is fed by the input of the previous one as in this scheme in figure 2, it was born the multi-layer perceptron:

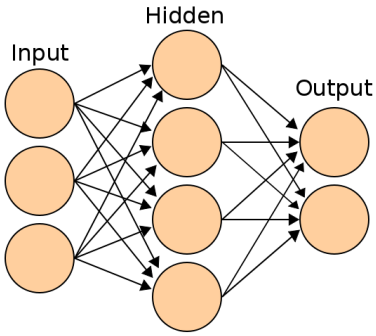

Figure 2: Sketch of a multi-layer perceptron (MLP) or that we can also call as Neural Network (NN).

Here the input information is processed through many layers. Thus, artificial neural networks have two main *hyperparameters* that control the architecture or topology of the network: the number of layers and the number of nodes in each hidden layer.There is no "a standard and accepted method" for choosing network configuration. In fact, we have to consider that neural network is a "black box" in the sense that while it can approximate any function, studying its structure will not give you any insights on the structure of the function being approximated. Moreover, from a traditional statistics viewpoint, a neural network is a non-identifiable model: given a dataset and network topology, there can be two neural networks with different weights but exactly the same result. This makes the analysis very hard [10]. However, typically all neural networks have:

- one *input layer* in which the data come in and the number of neurons comprising that layer is equal to the number of features;
- an *output layer* that will provide us the useful information: if the NN is a regressor, then the output layer has a single node; if the NN is a classifier, then it also has a single node unless softmax activation function is used in which case the output layer has one node per class label in your model.
- one or more *hidden layers* that will process the information before the output. How many hidden layers? As said before, if data are linearly separable, the hidden layer is not needed. Beyond that, the choice depends on the kind of problem: if a complex separating surface between two classes is needed, a more complex neural network with many layers is better. Instead about size of the hidden layer(s), there are some empirically-derived rules-of-thumb but trial and error process is the general method.

Figure 2 shows a Feedforward Neural Network (FFNN) that means there are links only from one layer to the next. In this work we will exploit an other type of network called Recurrent Neural Network. Before describing the one used in this analysis, let us discuss which are the main "ingredients" to build a neural network model.

1.1.1 How to build a Neural Network model

There are many different neural network models according to the choice made for each building block. In this study we use the following ones:

- As figure 1 shows, the neuron processes inputs that in the artificial neuron corresponds to the sum of inputs weighted by their weights. Then, as in real biological neurons the *action potential* governs how information flows within a neuron, here an ***activation function*** is applied to this output. As the name says, this function "activates" the output that means the weighted sum of the inputs is transformed so that the output is limited into a determined range of values. Moreover, it introduces *non-linearity* so that the Neural Network can successfully approximate any function . In fact, a Neural Network without an activation function is just a linear model. As for the whole architecture of a Neural Network, there is no universal rule for choosing an activation function for hidden layers.

  The output layer neural network of this project exploits the *softmax* activation function that is popularly used for multiclass classification problems. Here is the equation for the softmax activation function:

$$softmax(z_j) = \frac{\exp(z_j)}{\sum_j \exp(z_j)} \tag{3}$$

  The z represents the values from the neurons of the output layer. The exponential acts as the non-linear function. These values are divided by the sum of exponential values in order to normalize and then convert them into probabilities. In this way, the softmax function returns the probability of each class [13]. The node of the output layer that has highest probability value represents the class to which the data point belongs to.

- The **learning** strategy of our neural network is based on *backpropagation algorithm* that *updates all the weights (i.e. connections between neurons) in order to minimize an* ***error function*** *which characterizes the comparison between the network output and the ground truth*

in supervised models. The variation in time of the weights is given by:

$$\dot{w}_i = \frac{dE}{dw_i} \tag{4}$$

where E is the error function that is also refered to as *energy function* in analogy to a physical system in which forces act to minimize the energy. There are different *optimizers* to make the network evolve, i.e. algorithms used to change the attributes of a neural network such as weights and learning rate in order to reach the optimal solution and minimize the loss.

In this study we use the *Adam* as optimization algorithm that deals with the challenge of choosing the proper learning rate (in a classic gradient descent method, if it is too small leads to painfully slow convergence, if too large can hinder convergence) and of avoiding getting trapped in the suboptimal local minima. Its strategy is to accelarate the learning rate but it dampens oscillations towards local minimum. It combines the heuristics of two methods called Momentum and RMSProp.

As regards the error function instead, in our case of multi-class classification problem where softmax activation function is used, a proper choice is to use the *categorical cross-entropy* loss function (CCE) that is:

$$CCE = \sum_i p_i \cdot log(q_i) \tag{5}$$

where p_i are the "ground truth" probabilities and q_i are the estimated probabilities by the network. We can write this expression as:

$$\begin{aligned} CCE = S_p - K(p,q) = \sum_i p_i \cdot log(p_i) - \sum_i p_i \cdot log(\frac{p_i}{q_i}) = \\ = \sum_i p_i \cdot log(p_i) - \sum_i p_i \cdot log(p_i) + \sum_i p_i \cdot log(q_i) \end{aligned} \tag{6}$$

where S_p is the canonical entropy (i.e. uncertainty) of the distribution probability p of the training ground truth and $K(p,q)$ is the Kullback-Leibler divergence. The Kullback-Leibler divergence provides a measure of similarity between two statistical distributions and so, if S_p is a constant (since in our machine learning task, the true distribution p_i of the problem is given), minimizing CCE is equivalent to minimizing the Kullback-Leibler divergence. Thus, we can interpret the minimization of CCE as if we want to make the estimated distribution as close as possible to the correct one.

Finally, let us briefly describe three *hyperparameters* that should be tuned in order to obtain a model with optimal performance:

- the *batch size*: computing the gradient on the whole dataset is computationally very complex so nowadays the choice is to devide the dataset into batches, that is the number of samples to work through before updating the internal model *parameters* (i.e. weights). Thus the bigger the batches, the more samples are averaged and the more accurate will be the estimate of the gradient; however, small batch size helps for a more robust learning because it provides a high variance in the estimate of the gradient, so it reduces the risk of over-fitting or in getting stuck in a so called local minimum of the energy function. The batch size decides also the speed of the learning since small batch size implies that the network learns slowly. On the other hand, higher batch size requires higher computer memory storage availability. In our analysis, we set the batch size to 128 that is a trade off among all these aspects, hence the model weights will be updates after each batch of 128 samples (CPU and GPU memory architecture usually organizes the memory in power of 2 and so choosing mini-batch size of powers of 2 helps to speed up the fetch of data to memory).

- *number of epochs*: it defines the number of times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. This means that: if we have 1280 data, a batch size equal to 128 and 100 epochs, there are 10 batches. Each epoch will involve 10 updates to the model and with 100 epochs the model will pass through the whole dataset 100 times, that is a total of 100 batches during the entire training process [4].

- *dropout*: since there can be problems of overfitting so that the network learns the input data without being able to generalize or also the problem to get stuck in local minima of the energy function that implies a not optimal learning of the network, there are some *regularization* procedures to overcome these problems, one of which is perturbing the architecture of the network (dropout). This method consists in removing a subset of nodes during training, randomly choosing the removed units at each training epoch. The reason is that it was observed that while training a network, after overfitting, the weights for some of neurons increases and cause the network to be dependant on them. By exploiting dropout, we are not dependant on any node anymore due to it is possible to drop it while training.Dropout makes the neural

network more like an ensemble model, that is, just as a random forest is averaging together the results of many individual decision trees, you can see a neural network trained using dropout as averaging together the results of many individual neural networks .For example, if you set dropout rate to 0.1, then for each iteration within each epoch, each node in that layer has a 10% probability of being dropped from the neural network. This makes the network more robust and allows weights to explore a wider parameter space.

1.1.2 Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network

We will use a neural network trained to classify protein sequences in order to obtain a dense vector representation of them so that we can meaningfully represent them in an embedding space. We can consider protein sequences as sequences of letters, in particular there are 20 unique symbols which are the amino acids (22 if we consider also two rare amino acids).

When it comes to sequential or time series data (where the ordering has a meaning), traditional feedforward networks cannot be used for learning and prediction. Ordinary feed forward neural networks are only meant for data points, which are independent of each other. If we have data in a sequence such that one data point depends upon the previous data point, we need to modify the neural network to incorporate the dependencies between these data points. Recurrent neural networks, or RNNs for short, are a variant of the conventional feedforward artificial neural networks that can deal with sequential data and can be trained to hold the knowledge about the past. RNNs have the concept of "memory" that helps them store the states or information of previous inputs to generate the next output of the sequence [5]. A RNN is designed to mimic the human way of processing sequences: we consider the entire sentence when forming a response instead of words by themselves.

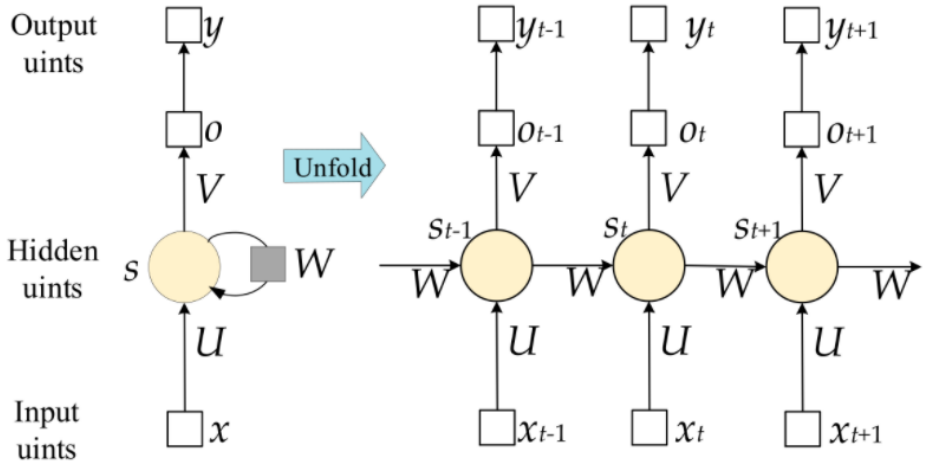

Figure 3: Sketch of a Recurrent Neural Network. Compressed representation (left), unfolded network (right). Image from https://www.mdpi.com/1996-1073/12/13/2538.

The sketch of a traditional Feed Forward Neural Network could be simply made by an input layer, a hidden layer and an output layer. Figure 3 shows a Recurrent Neural Network that has the same components of a Feed Forward Neural Network but, in addition, with a *recurrent loop* that can pass previous information forward. This information is the *hidden state* and it is a representation of previous inputs

We can "unfold" the network, that is we can interpret this single loop with just a neuron, as a network with a set of neurons that receives inputs sequentially as time-series data. These inputs could be words in a sentence or, as in our case study, amino acids of a protein sequence.

The final output (for example, in the above figure, O_{t+1} at time $t+1$) has encoded all information from all the inputs in the previous steps and it is finally passed to a feed forward layer. The last feedforward layer, which computes the final output for the kth time step, is just like an ordinary layer of a traditional feedforward network.

In order to really understand, let us consider an example that is how a chatbot [36] works to *classify* intentions. If a user input the question "What time is it ?", the question will be broken into individual words. Then, since the RNN works sequentially, a word at time is fed to the network as in figure:

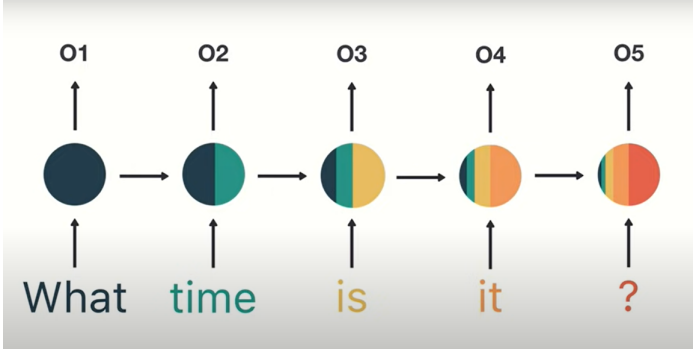

Figure 4: Sketch of an unfolded Recurrent Neural Network that processes the sentence "What time is it ?". Image from [37].

The first step is to feed "what" into the RNN. The RNN encode "what" and produces an output. Then, the next step is to feed the word "time" and the hidden state from the previous step. At this point, the RNN has information of both words "what" and "time". This is repeated untill the final step when the final output $O5$ is passed to a feed forward layer to classify in intent:

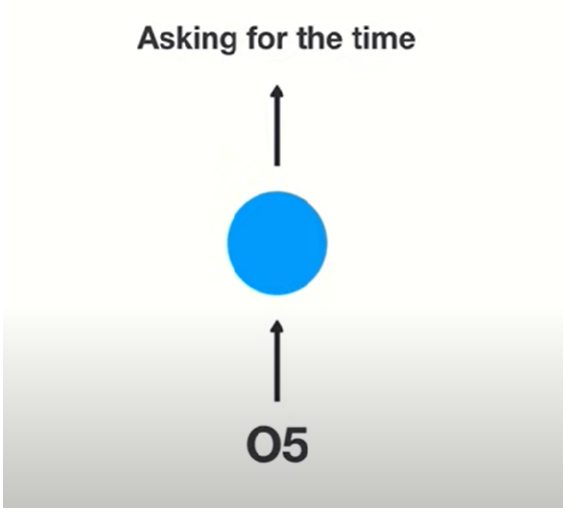

Figure 5: Sketch of final feed forward layer of a Recurrent Neural Network that processes the RNN output to classify in intent the sentence "What time is it ?". Image from [37].

However, there is a relevant shortcoming in this kind of architecture that is highlighted by the distribution of colors in the hidden states in figure 4: as said, the feedback loop in RNNs lets them maintain information in

"memory" over time, but it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because RNNs use backpropagation algorithm to update model parameters (weights) as well as the FFNN do and causes what is known as "memory decay": if RNNs combine actual input x_t with preceding input x_{t-1}, the net will keep track of previous inputs (memory) with geometrically decaying attention. In the previous example of intent classification of the question "What time is it ?", this problem would mean that the words "what" and "time" are not considered when trying to predict the user's intention and the network has to make its best guest with "is" and "it" that is something of ambigous also for us as humans. This is true also in our case in which we want to classify protein sequences according to the host species and, moreover, this would also mean that dependencies between the first and last amino acid of a protein sequence will poorly be captured by the network to perform its classification task.

A technique particularly used to solve this problem in RNNs is the *long short-term memory* (LSTM) network. LSTMs are again inspired by the long-term and short-term memory capabilities in real biological neurons, i.e. the capacity to store a small amount of information and keep it readily available for a short period of time (short-term memory) and to store events over an extended period (long-term memory).

This network is not a simple recurrent network but it has a very complex structure which is made of different blocks (known as "memory blocks" or "memory cells") that can choose how far to go to keep track and to keep memory through learning. They substitute each one of the elements of the recursive loop seen in figure 3. Thus, in this way, there is not only the memory decay for events far in time characterizing short-term memory but they can memorize events far in the time or in a sequence in order to allow past information to be reinjected at a later time, selectively choosing the size of the long-term memory with this memory cell structure:

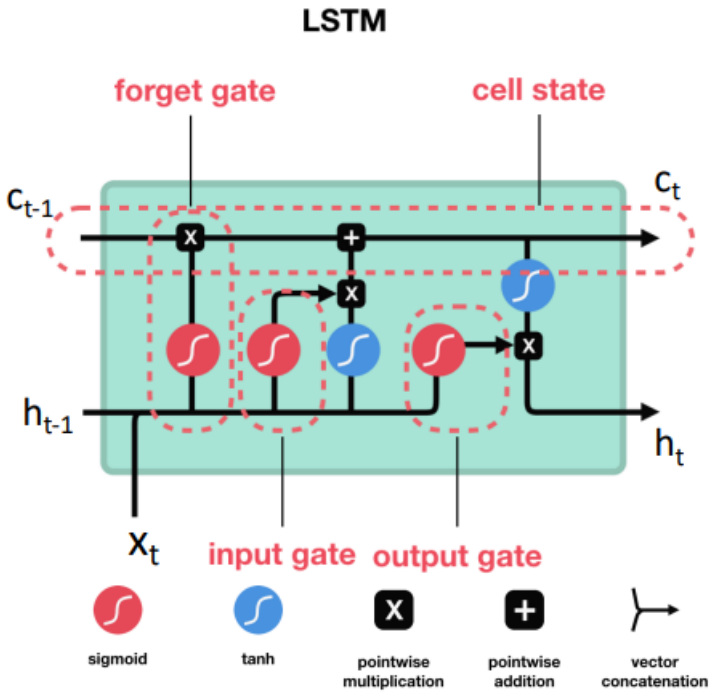

Figure 6: Sketch of a LSTM memory cell. Image from [1].

Each one of these memory blocks contains three multiplicative units: the input, the output and forget gates. Each cell receive "new" input and input from previous cells. Gates operate these inputs:

- *Forget gate*: receives new input x_t and previous hidden state h_{t-1}, that is information coming from previous cells, and decides what is not relevant;
- *Input gate*: receives new input x_t and previous cell output c_{t-1} and combines them;
- Input and forget gate are combined to produce the cell's output to be passed c_t;
- *Output gate*: receives new input x_t and previous cell output; it combines them to generate next hidden state to be passed h_t.

Thus each cell is based on the mechanism of addition and forgetting: they integrate the information from the previous cells with the new input and gives the possibility to decide which is the important information to keep. In other words,they are both passing some global information important for all the

time series and encoding new information that is depending on each local time step, adding or forgetting some information that is passing from the beginning to the end of the cell. So, for example, in text analysis it allows to consider previous words of the sentence that are important to understand the meaning of the current word, differently from a simple RNN that considers only the latest words of the series and with memory decay the others.

1.2 Machine learning for "protein language" processing

Applying machine learning to biological sequences is not an easy or straightforward task [15]. We have to understand a functional way to deal with biological sequences both in the pre-processing and processing steps. In this study we train a RNN with LSTM architecture neural network to classify protein sequences according to the host species they belong to. This is because in this "unknown language of life" there could be important correlations , for example, between distant aminoacids in the protein sequence similar to what happens for words in a sentence. More precisely, a bi-directional LSTM (biLSTM) is really suitable for this case study because, differently from every natural human language, for protein sequences there is not one direction we are interested in but also the opposite direction could encode important information. So bi-directional LSTM combines two LSTMs working in the two opposite sequence directions and integrating the information coming from the opposite sides.

The computational tools that we will use and that we have just described (RNN and LSTM) belong to the family of Natural Language Processing (NLP) methods that is a field of deep and machine learning concerned with automated text and language analysis. Proteins are a natural fit to many NLP methods as we can deduce by the following *analogies* between them and human language: proteins' *alphabet* consists of 20 common amino acids (excluding unconventional and rare amino acids) and they can be represented as strings of amino acids letters; like natural language, naturally evolved proteins are typically composed of reused modular elements which are *motifs* and *domains* that constitutes the basic functional building blocks of proteins similarly to words, phrases and sentences; from information-theory perspective, the protein's information (e.g. its structure or function) is contained within its sequence as well as the meaning of a sentence is contained in its sequence of words. If we change the position of words in a sentence, we could change the meaning of the sentence; if we change amino acids positions we could change structure and function of proteins [19].

Given these similarities in shape and substance, it seems natural to apply natural language processing methods to protein sequences giving rise to what we could call a "protein language processing" method. As well as in NLP models, there are two important factors we need to consider: tokenization and embedding.

1.2.1 Tokenization

Computational text analysis requires tokenization, i.e. splitting the text into individual *tokens*, which are the *atomic units of information in a chosen language representation* [19]. Tokenizing is a pre-processing step in NLP tasks that can be independent from NN (e.g. based on statistical properties, letters frequencies, correlation or joint probability of appearance of certain letters).

Its purpose is to translate for example sentences (i.e. sequences of words, for instance) or protein sequences (i.e. sequences of letters) into a vector so that it can be used as input to a neural network.

We can decide which are the tokens in the sentence. NLP models typically use words as tokens, although some approaches use individual characters. Individual-character tokens offer greater flexibility, especially for out-of-vocabulary or misspelled words, or for languages without clear separation between words such as Mandarin.

In proteins, the simplest and most common tokenization method is to regard individual aminoacids as character-level tokens. Since proteins do not have a well-defined vocabulary of words, word-level tokenization is not a well-defined option in the case of proteins. In fact, besides the previously mentioned similarities with human language, there is also this important difference: while most human languages include uniform punctuation and stop words, with clearly separable structures such as words, sentences and paragraphs, with proteins we do not always know whether a sequence of amino acids is part of a functional unit (e.g. a domain). There is no clear analogy between the building blocks of language and those of proteins and considering protein domain as being equivalent to words is often misleading. Furthermore, protein functional units often overlap. As a result, while natural languages have a well-defined *vocabulary* (with $\sim$ million words in English), proteins lack a clear vocabulary.

Keras deep learning library in Python provides the Tokenizer class for preparing text documents for deep learning. The Tokenizer must be constructed and then fit on either raw text documents or integer encoded text docu-

ments [6]. Below we report an example of Python code in order to better understand:

```
#Import module
from keras.preprocessing.text import Tokenizer
# define a document
text = ['The cat sat on the mat']
# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the document
tokenizer.fit_on_texts(text)
```

`fit_on_texts`: it will create a dictionary so that every word is assigned to a unique integer value. So lower integer means more frequent word (often the first few are stop words because they appear a lot) [21]. In this case:

```
print('word_index : ',tokenizer.word_index)
```

It will print:

```
word_index :  {'the': 1, 'cat': 2, 'sat': 3, 'on': 4,
↪  'mat': 5}
```

Then we need:

```
encoded_doc=tokenizer.texts_to_sequences(text)
```

`texts_to_sequences`: it transforms each sentence (i.e. sequence of strings) in a sequence of integers. So it basically takes each word in the text and replaces it with its corresponding integer value from the `word_index` dictionary. It returns a two dimensional array that in this case is:

```
[[1, 2, 3, 4, 1, 5]]
```

This procedure can be applied on more sentences (that is more sequences of words) and it will return a 2D array with several inner arrays corresponding to each sequence. This will be our case in which we have several protein sequences of amino acids.

We will apply this to protein sequences and, in that case, the single AAs are chosen as tokens. In that case, the `texts_to_sequences` will return a 2D array in which each inner array represents a "tokenized" protein sequence.

1.2.2 Embedding

An embedding is a mapping of a discrete (categorical) variable to a vector of continuous numbers [16]. It is the class of approaches for representing words, documents or (in our case) amino acids, protein sequences using a dense vector representation. It is an improvement over the traditional bag-of-word model encoding schemes where large sparse vectors (comprised mostly of zero values) were used to represent each word that often results in an unrealistic amount of computational resource requirement and often ignores the informative relations between them (each vector is orthogonal to all the other ones). Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used as well as the position of the amino acid within the vector space is learned from sequences and is based on the amino acids that surround it when it appears. The position of a word or amino acid in the learned vector space is referred to as its embedding.

There are several methods of learning word embeddings from text. In this study, we will use the embedding that is learned as part of a deep learning model. In particular, Keras provides an *Embedding layer*, that is defined as *the first hidden layer of a neural network* trained on text data. The embedding layer requires that the input data be integer encoded (so that each word is represented by a unique integer) so that it can convert positive integer representations of words into words embeddings.

For example, if we consider the sentence "the cat sat on the mat", after tokenization, a unique integer number is assigned to each word of the sentence: $[1, 2, 3, 4, 1, 5]$

Now imagine we want to train a network whose first layer is an embedding layer. In this case, we could initialize it as follows:

```
Embedding(5,2,input_length=6)
```

The first argument is *the number of distinct words or symbols in the training set*, the second argument indicates *the size of the embedding vectors* in which words or symbols will be embedded and the *input_length* argument determines the size of each input sequence.

Once the network has been trained, for example, to classify several sentences, we get the weights of the embedding layer, i.e. the output of the embedding layer that will be (in this example) 5 vectors of 2-dimensions each and each

one reprents a word in the sentence.

The embeddings form the parameters (weights) of the network which are adjusted to minimize loss on the task [16]. Indeed, the weights for the embedding are randomly initialized (just like any other layer). During training, they are gradually adjusted via backpropagation. Once trained, the learned word embeddings will roughly encode similarities between words (as they were learned for the specific problem the model is trained on) [38]. Basically the embedding layer comes up with a relation of the inputs in another dimension, whether it is in 2 dimensions or even higher. Indeed, our neural network captures underlying structure of the inputs (our sentences) *in order to make, for example, classification* and puts relation between words in our vocabulary into a higher dimension (let us say 2) by optimization. For instance, the frequency of each word appearing with another word from our vocabulary could be one of many underlying structures that neural network can capture [22]. And many researchers who have struggled with the problem of semantics have come to the conclusion that the meaning of words is closely connected to the statistics of word usage [39]. So, by training a neural network whose task is sentiment review classification and words are chosen as tokens, words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space [2].

Similarly in the language of life, if we train a neural network for protein sequence host species classification task with amino acids chosen as tokens, amino acids play the role of words which are encoded as real-valued vectors in a high dimensional space and similarity between amino acids, in terms of frequency and relative positions in the sequence (i.e. potentially their "meaning", as in case of words in a sentence) translates to closeness in the vector space.

How to choose the dimension of the embedding space that defines the dimensionality in which we map the categorical variables? It is common to see word embeddings that are 8-dimensional (for small datasets), up to 1024-dimensions when working with large datasets. A higher dimensional embedding can capture fine-grained relationships between words, but takes more data to learn [40]. Anyway, generally the key factors for deciding on the optimal embedding dimension are mainly related to the availability of computing resources: in most cases, embedding dimension is chosen empirically, by trial and error and if there is no difference in results and one can halve dimensions, it is better to do so [11]. Jeremy Howard provides a general rule of thumb about the number of embedding dimensions: embedding size =

min(50, number of categories/2). However, literature shows that embedding dimensions of size 50 produces the most accurate results [12]. In this work, we choose for each amino acid an embedding dimension equal to 10 that obeys to the Jeremy's rule of thumb and, at the same time, fits with our computational resources.

Furthermore, we point out that, differently from FFNs, RNNs would be suitable to process sequential data of different size since they process inputs continuously. However, Keras prefers inputs to be vectorized and all inputs to have the same length so in our analysis, we will truncate amino acid sequences to a fixed size due to low computational power available for this study (see section 2, **Software and Methods**) by setting a value of the *input_length* argument in the embedding layer.

In text analysis, it is often useful to initialize the embedding layer with weights learned by *Pretrained Word Embeddings*. The most common are Word2Vec and GloVe that are *unsupervised* learning algorithms to learn vector representations of words: they turn text into a numerical form that deep neural networks can understand and they are able to capture the full meaning of words in the resulting embedding while the embedding layer in a supervised network might not learn such semantically-rich and general representation. Thus, Pretrained Word Embeddings are trained on large datasets (an operation that can take not hours but months), saved, and then used for solving tasks such as classification. That is why pretrained word embeddings are a form of Transfer Learning [24]. In our protein sequences analysis we will exploit only the embedding provided by the randomly initialized neural network embedding layer but initializing the embedding layer with weights learned by Pretrained Embedding and then refine them on a classification task could be useful also in this field and in this kind of problem (i.e. extracting embedding vectors to represent protein sequences). One example of pretrained embedding, in this case, is the method proposed by Michael Heinzinger et al. in the research article *Modeling aspects of the language of life through transfer-learning protein sequences* [23]. Here they introduced the *unsupervised* SeqVec embeddings showing that this transfer learning method can be used to capture biochemical or biophysical properties of protein sequences from large unlabeled sequence databases.

So, we will map all the 22 amino acids to 10-dimensional vectors using neural network embeddings and then, for each protein sequence, we will average these vectors to obtain a 10-dimensional vector representing all the sequence. As in natural language average embedding gives an idea of the sentence topic, here in protein language it gives an idea of protein sequence structure (i.e.

"topic") based on the frequency and position of amino acids. Next, we reduce all the resulting 10 dimensional vectors sequences to 2 dimensions using dimensionality reduction methods (PCA, t-SNE and UMAP) that can be considered also themselves embedding methods. This allows us to visualize the learned embeddings to show which categories are similar to each other, that is the great advantage provided by embeddings and something that could not be achieved by one-hot encoding type of representation of categorical variables.

1.3 Dimensionality reduction (embedding) methods

The goal of dimensionality reduction techniques is, given multivariate data, to reduce the number of variables without reducing the information content. The hypothesis behind is that much of data is noise thus many variables do not contain useful information for our analysis. So in this sense, as mentioned in the previous section, even here we can talk about finding an optimal *embedding, that in this case is a mapping of data from a high dimensional space into a lower-dimensional space*. One of the final purposes of this is to transform the input space into a 2 or 3 dimensional space so that we are able to visualize our data in a plot. The dimensionality reduction methods used in this project are the following:

- *Principal Component Analysis* (PCA): it *reduces the dimensionality while conserving as much as possible the variation present in the data*. Practically, it makes a traslation to shift the data into the origin of axis; it rotates the data and in the new rotated space it identifies the highest eigenvalues of the covariance matrix and projects the data on the corresponding eigenvectors, that are the directions of maximum variation of the data.

  So we can understand that if we have, for example, two groups of data that can be linearly separated along one direction, this technique works perfectly by reducing the initial variables to a new set of variables called the *Principal Components* (PCs), that are linear combination of the initial variables, which most account for the spreading of the data. This helps for clustering investigations because very often data have larger variance along the direction that separates the clusters.

- *Stochastic Neighbour Embedding* (SNE): this approach is based on a network description of our space: we can consider the nodes as our samples and the links (the relation between all the samples) as the mutual distance. In this case, the algorithm *reduces the dimensionality*

with the aim to reproduce local neighborhood structure of the initial variable space X in a lower-dimensional space Y. It imposes that two nodes (vectors), that are close in the original space, must be close also in the new space. It works in this way: the sample distance is calculated as the conditional probability $p_{j|i}$ of finding a neighbor j from i in X space with the hypothesis that neighborhoods for each *i-th* point are distributed as a Gaussian distribution. In the new low-dimensional Y space, this probability is $q_{j|i}$, i.e. the same function but with the distance calculated with the new variables. The objective is to maximize the similarity between p and q distributions minimizing the distance between two distributions that is measured using the *Kullback-Leibler divergence*:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{7}$$

C can be interpreted as cost function but it has also the functional shape of an entropy just like the case of neural networks described in the previous section 1.1.1. C is minimum, that is null, when $p_{j|i} = q_{j|i}$ and so when the two probabilities are equal.

A parameter that is not known and that one can choose is the standard deviation σ of the neighborhood Gaussian distribution. One can do this by fixing the value of *perplexity* given by:

$$Perp(P_i) = 2^{H(P_i)}, \quad H(P_i) = -\sum_i p_{j|i} \log_2 p_{j|i} \tag{8}$$

So fixing the perplexity value means fixing σ. And this means also fixing *the same volume* around each point of the space. Hence, the aim is to preserve the local neighborhood for each element with respect to a certain number of neighbors in the original space that depends on this chosen volume.

In our analysis we will use a SNE variant that is t-SNE. The differences are the following: t-SNE considers a symmetric distribution $p_{ij} = p_{ji}$ and this produces a simpler gradient for cost function minimization; moreover, it considers the neighborhoods distribution as the Student's T distribution that has wider tails than the Gaussian, that means that t-SNE preserves neighborhood over wider area.

- *Uniform Manifold Approximation and Projection* (UMAP): One of the limits of SNE and t-SNE is that it can join some neighborhoods into a

unique manifold even in cases in which it is wrong. Thus this algorithm tries to overcome this problem by fixing *for each node* which are the nearest *k-neighbors*, so that these can be different from node to node. Here the choice of σ is given by the following equation:

$$\sum_{j=1}^{k} \exp \frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i} = log_2(k) \tag{9}$$

Given k nearest neighbors, given the mutual distances between x_i and x_j, d(x_i,x_j), we can solve this system of equations and calculate σ for each node. Even in this case, equation 9 has an entropy functional shape where the logarithm of the microstates of the system is represented by the logarithm of the number of neighbors.

In our analysis, we will use these methods to reduce the dimension of the embedding space in which our protein sequences data lie.

1.4 Clustering

We will analyse data using a clustering method called DBScan, that belongs to the family of unsupervised machine learning methods, after the application of a supervised neural network. In fact, while in the first part of the analysis we exploit a neural network that knows the truth about data (i.e. labels in classification task) and we train it to recognize those which are the data labels, with clustering we look for intrinsical structures in the data, without having a priori information to train the model. *What we want to know is if there is some intrinsic structure characterizing the data.*We want to assess if the data are spontaneously divided, stratified into groups.

DBScan algorithm does not require to choose the number of clusters (as in case of an other popular algorithm that is K-Means) but they are automatically estimated by the algorithm. In this case, there are two important parameters: *neighborhood radius* ϵ and *minimum number of samples* that each cluster must have.

The algorithm starts from one random sample and calculates how many other samples fall within its neighborhood ϵ and generates a cluster if in the neighborhood radius there are at least the minimum number of samples. Then (differently from K-Means), if points have no minimum number of neighbours, they are considered as noise without altering the cluster structure. Due to the fact that the result may vary according to the starting point, it is a stochastic algorithm and we understand well that what affects the formation of cluster and their size is the local density of samples in the space. The

two parameters define formally what we mean when we say dense. Higher minimum number of samples or lower ϵ indicate higher density necessary to form a cluster [29].

Therefore, DBSCAN views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to K-Means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

While the number of samples parameter primarily controls how tolerant the algorithm is towards noise (on noisy and large data sets it may be desirable to increase this parameter), the parameter ϵ is *crucial to choose appropriately for the data set* and it controls the local neighborhood of the points. When chosen too small, most data will not be clustered at all (and labeled as -1 for "noise"). When chosen too large, it causes close clusters to be merged into one cluster, and eventually the entire data set to be returned as a single cluster.

The optimal embedding of initial samples in a new metric space can allow clustering with even simple methods (like K-means) to recognize important patterns. In our case, we will use Neural Network as "embedder" to have a dense vector representation of protein sequences (that at the beginning are, essentially a sequence of symbols not treatable for quantitative analysis) and then we will apply "other types of embedders" (the previously described dimensionality reduction techniques) on these N-dimensional vectors (in our case, N=10 as mentioned in section 1.2.2). Thus the choice of these embedders and their parameters, in each step of our analysis, is a crucial point to be able to extract information from the amino acids sequences data.

2 Software and methods

In this analysis, we study the characteristics of Spike protein sequence in SARS-CoV-2 virus [9] that has the molecular function to bind to a receptor on the host cell surface, initiating the infection. We exploit their numerical representation through weights vectors extracted from the embedding layer of a LSTM network, using it as a classifier. The analysis is done on a laptop with Processor characteristics: Intel(R) Celeron(R) N4000 CPU @ 1.10GHz

1.10 GHz. Because of the limitations of the laptop characteristics in terms of CPU or lack of availability of modern GPU in this project, we do not use a bidirectional LSTM that would require much more time to train.

The same type of protein in a virus can have very different amino acids sequences, even within the same species, so our data are in FASTA format file that contains the protein sequences of Spike protein for different host species. They are avaible in Github: `https://github.com/brianhie/viral-mutation`. We will focus on the classification of these protein sequences according to their host species because it is a good starting point to understand if our Keras Neural Network works in capturing the meaningful patterns from raw sequences. In particular, the model developed by Brian Hie et al. [30] shows that the human sequences are very similar to those of bat and pangolin sequences and so we would like to reproduce at least these results, even though using an other model. To run our model we use a Python program with a set of software dependencies, in particular, we install *TensorFlow 2.0+* library, which includes the *Keras* deep learning framework. This and all the code built to obtain the results are reported in the Jupyter Notebook provided.

Hence, these are the tools that we will use to analyze these novel Spike protein sequences. As explained in section 1.2, in this study we will consider single amino acids as tokens and as if they were words of a sentence, but it can help to keep in mind that the language of life is typically characterized by the so called motifs.

A protein sequence motif, or pattern, can be defined as a region (a subsequence) of amino acid sequence that has a specific structure and that is important for protein function. Protein sequence motifs are signatures of protein families and can often be used as tools for the prediction of protein function and as a base of protein classification [32]. Therefore a protein sequence motif is found in similar proteins and change of a motif changes the corresponding biological function. One well known motif of Spike protein is the so called receptor binding motif/domain (RBD), that is believed to be the major target to block viral entry since the viral spike protein binds the host receptor angiotensin-converting enzyme 2 (ACE2) via the RBD. Thus, it is for this reason that protein sequence similarites among different host species points towards the leap from animals to humans rather than the hypothesis of an engineered virus as regards the coronavirus outbreak origin.

Hence, in terms of language of life, if our neural network is able to capture the underlying structure of the inputs (protein sequences), it means

that it will capture motifs (and so patterns) that are more conserved than the background sites. Moreover, generally in nature, the same species have more similar protein sequences between each other with respect to protein sequences of different species.

3 Results and discussion

The first problem to deal with in this study is the class-imbalance problem. There are four classes of host species on which we are interested to focus our study (human, bat, pangolin and other animals) but these are unbalanced due to practical reasons. The minor class is pangolin that counts only 10 Spike protein sequences, in fact there are eight species of pangolins, four of which are found in Asia and they're listed by the IUCN as critically endangered, while the other four are African species and are listed as vulnerable [25]. The human class represents the majority class as we can expect since it is the most numerous species and also the species more "available" to be sequenced . We can visualize the classes distribution with the following bar plot:

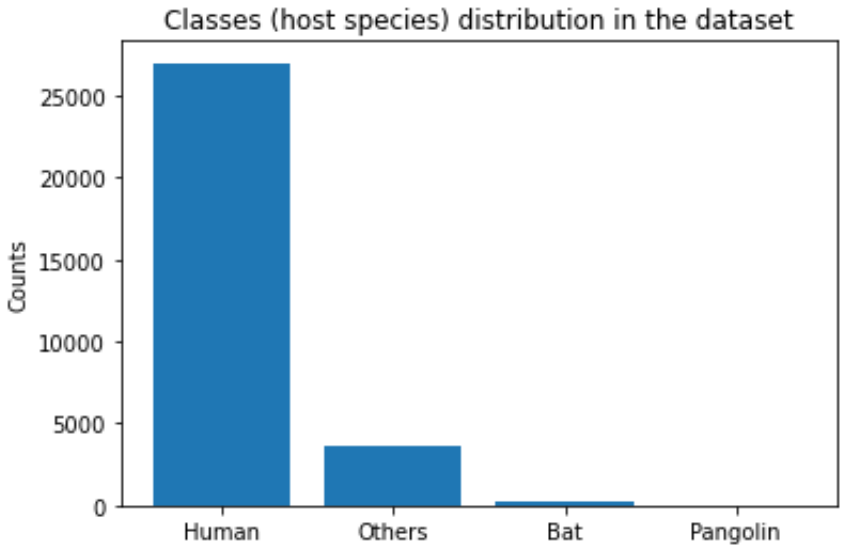

Figure 7: The bar chart shows the distribution of classes, i.e. host species, in our dataset. In particular there are: 26987 human protein sequences, 260 bat protein sequences, 10 pangolin protein sequences and 3596 other animals protein sequences.

Thus we wonder if a Neural Network classifier, that is trained to classify these data as they are, could give reliable results as protein host species

classifier but also if the vectors weights used for dense vector representation of protein sequences are reliable, i.e. if they are really representative of the relative protein sequences.

In general association among trained weights and class-imbalance is not that easy to establish and it is an open research. The survey paper by Justin M. Johnson and Taghi M. Khoshgoftaar [26] points out that despite recent advances in deep learning, along with its increasing popularity, very little empirical work in the area of deep learning with class imbalance exists.

Anand et al. [27] explored the effects of class imbalance on the backpropagation algorithm in shallow neural networks in the 1990's. The authors show that in class imbalanced scenarios, the length of the minority class's gradient component is much smaller than the length of the majority class's gradient component. In other words, the majority class is essentially dominating the net gradient that is responsible for updating the model's weights. This reduces the error of the majority group very quickly during early iterations, but often increases the error of the minority group and causes the network to get stuck in a slow convergence mode.

In our analysis we investigate the relation between class-imbalance and the protein sequences embedding space built following the procedure explained in section *Embedding* 1.2.2. We could study also how this embedding changes varying the neural network architecture and varying many hyperparameters but, in this project, we focus first of all on the data and in particular we try three configurations:

- In case we train the neural network to classify the amount of data of the original dataset, i.e. with *imbalanced classes*;
- In case we train the neural network to classify data with 260 human protein sequences, 260 bat protein sequences and 260 protein sequences of other animals, i.e. *undersampling method* with number of data of each class equal to the minority class counts;
- We divide the most abundant classes among humans, bats and other animals in N subsets each one with 260 data, that is the amount of data of the minority class (bats), and we train our neural network classifier sequentially on all these subsets, in particular on a N-th subset of a majority class (so on a subset of the human category and on a subset of the other animals category) but on all of the data from the rare class.

Before exploring these steps, we need to check which is the sequence length (that is the number of amino acids) distribution of our protein sequences

because, as explained previously, even though a recurrent neural network is in principle made to analyze input data with variable size (in this case protein sequences with different lengths), in practice we need to use a fixed size because Keras embedding layer accepts sequences in this way (it is common to input sequences with the same size to make the code more efficient, both in the sense of memory and computing time). Thus we need the sequence length distribution in order to choose the input sequence length. In order to not increase too much the training time, we will have to choose a length that is lower than the maximum length for most of them, so we need to have an idea of how much data we are cutting out.

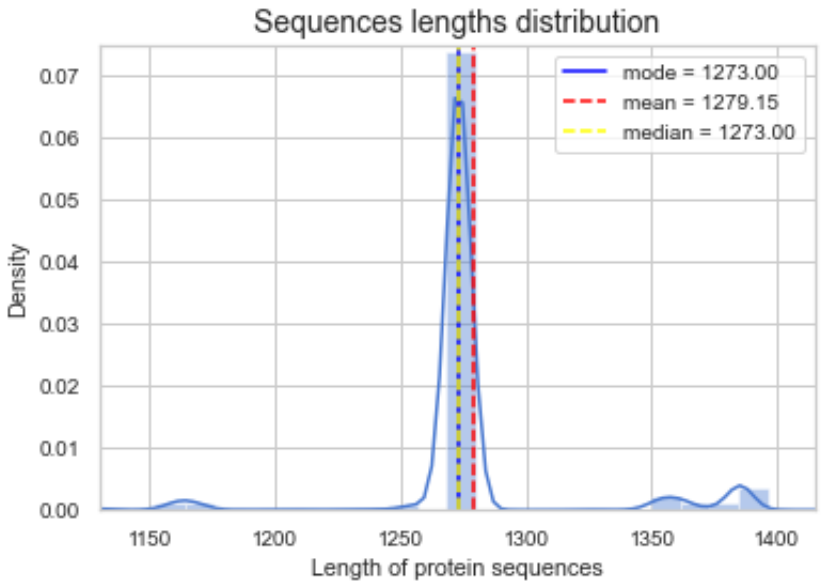

Figure 8: Distribution of protein sequences lengths with median value (yellow dashed line) equal to 1273 that coincides with the mode (solid blue line). The mean value (red dashed line) is instead equal to 1279.15. The shadows represent a histogram with bin size determined automatically with a reference rule.

The sequence lengths distribution has a sharp peak corresponding to the value equal to 1273 that means that it is the most abundant class of sequence lengths, while the maximum sequence length (that is computed with Python using the `max()` function) is 1582. Hence, we could consider the maximum sequence length as the length of all input sequences and pad[1] all the sequences that are shorter than this. However, in the first case in which we

[1] Add the zeros at the end of the sequence to make the samples in the same size.

train the neural network on all the protein sequences, it is unfeasible to use such long protein sequence length in terms of training time. Therefore, we will use this length only when we will consider the undersampling approach.

Moreover, we have to point out that we have set an embedding dimension equal to 10, thus each protein embedding vector has 10 dimensions. We need to embedd these 10 dimensional vectors into 2 dimensions through dimensionality reduction methods. We will use three different approaches: PCA, UMAP and t-SNE that have been described in section 1.3. These methods allow to exploit different characteristics of our data: the first one reduces the dimensionality conserving as much as possible the variation present in the data; the second ones reduce the dimensionality with the aim to reproduce local neighborhood structure of the initial 10-dimensional space. Thus, let us discuss the results in details.

As reported in the Jupyter Notebook, we train a neural network whose main components and hyperparameters are: a LSTM layer with 64 LSTM cells, a dense hidden layer with 64 neurons and Dropout layers between the Embedding and LSTM layers and the LSTM and Dense output layers (with rate parameter respectively equal to 0.25 and 0.3). The number of epochs is 15. The first method explored considers 350 amino acids as maximum length of protein sequences with neural network trained on all data, i.e. 26987 human, 260 bat, 10 pangolin and 3596 other animals host species. In this configuration, we choose a sequence length equal to 350 because many papers consider 300-500 length sequences [31]. This means that any protein sequence greater than that length will be truncated[2] and any protein sequence shorter than that length will be padded. Thus, we understand well that this method, besides the problem of imbalanced classes, carries also the problem of removing potentially useful information from the training set. But this sequence length allows the model to see all of the available data (even though only the first part) with the hope that the considered subsequence has some important "fingerprint" that characterize the species.

[2]When a sequence exceeds the maximum number of amino acids drop the last amino acids in the sequence.

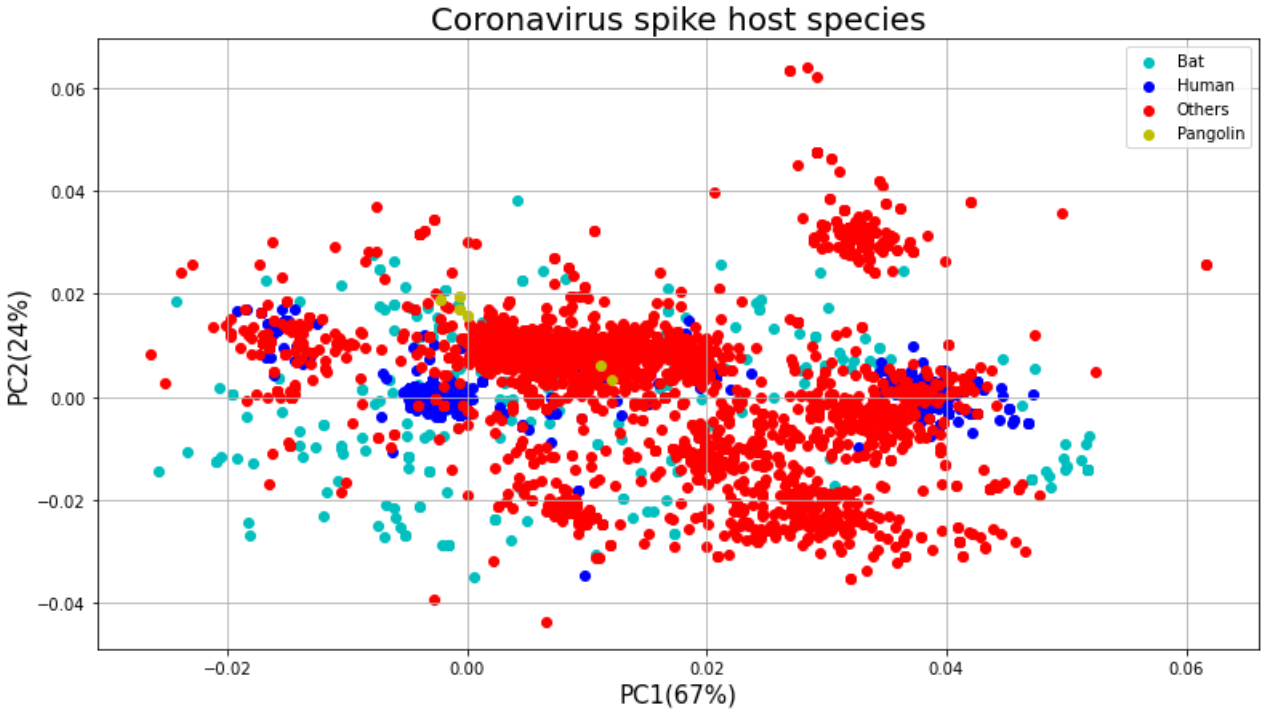

Figure 9: Principal component analysis (PCA) plot of Spike protein sequences considering all data, i.e. 26987 human protein sequences, 260 bat protein sequences, 10 pangolin protein sequences and 3596 other animals protein sequences. The first two principal components (PCs) are plotted and coloured according to host species. PCA was applied on protein sequences embedding 10-dimensional vectors. Percentage of variation accounted for by each principal component is shown in brackets with the axis label.

Figure 10 shows the principal component analysis (PCA) scatterplot that compares the samples based on the first two projections. We can notice that the data are mixed thus it is not particularly informative so either the PCA has failed or the neural network model built is failing or the first 350 AAs do not contain particular patterns that can distinguish the species. However, we can notice that the first principal component PC1 accounts for well 67% and in particular we can look at the scree plot:

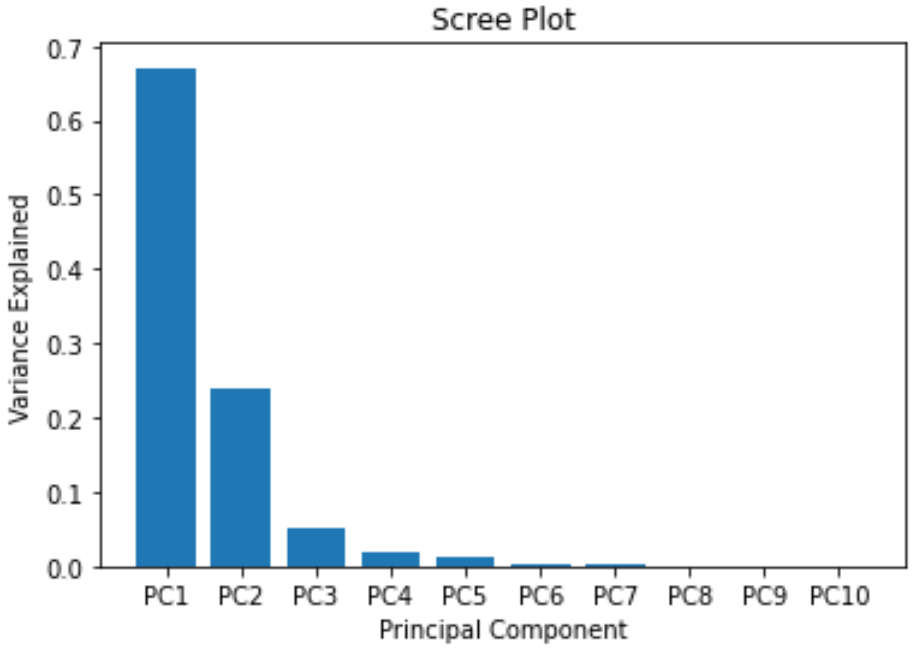

Figure 10: The scree plot shows the amount of variation accounted for by each principal component: the first principal component explains 67% of the total variation in the dataset; the second explains 24% of the total variation; the third explains 5% of the total variation and then lower and lower values with the sum that is 100%.

We have a numer of principal components equal to the number of dimensions of the space but the ones that account for the vast majority of the variation are the first two that respectively account for 67% and 24% of variation. Even though they are characterized by a very large difference in value, the first two eigenvalues are really relevant. Thus a 2D graph using just PC1 and PC2 would be a good approximation of the 10-dimensional space since it would account for the 91% of the variation in the data. In fact, we can consider that it is common practice to use just 70% as predefined percentage of total variance explained to decide how many PCs should be retained (although the requirements of graphical representation often lead to the use of just the first two or three PCs)[34].

Thus the cause of this apparently not informative PCA plot could be attributed to the imbalanced classes: for example PC1, that is on the horizontal axis, accounts for 67% of the total variance thus along the x-axis of the PCA plot we should notice a gap between the groups of data or at least that these groups are linearly separable even with some degree of overlapping. We can

notice in fact some clusters but they seem to heavily overlap: maybe if we would have had more data from pangolin class, bat class and other animals class, which are all significantly smaller in quantity than humans, we could have seen that the direction along the first component separates the colors and so the host species.

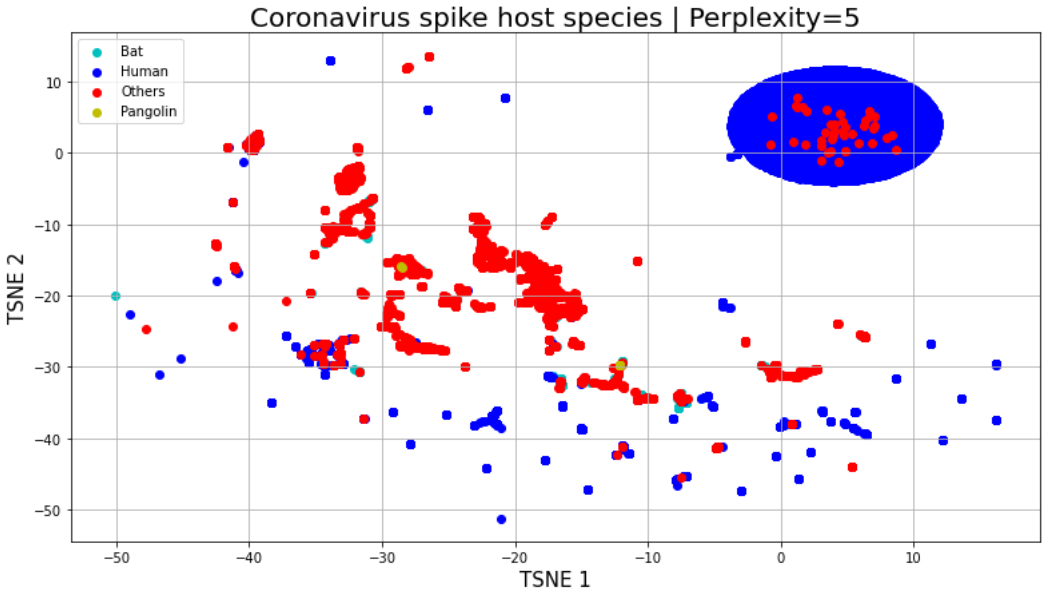

Figure 11: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=5.

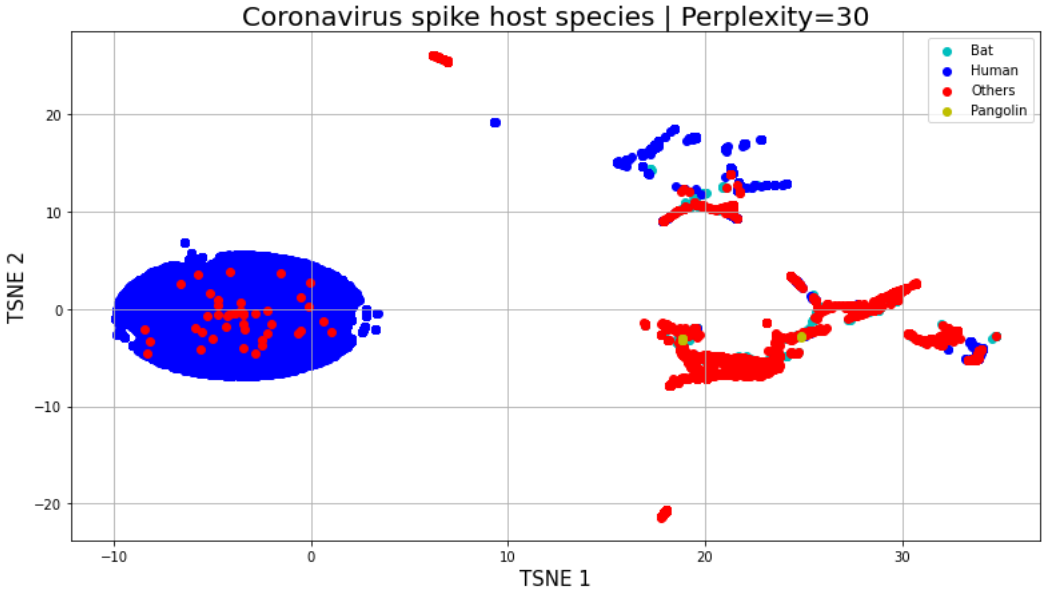

Figure 12: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=30.

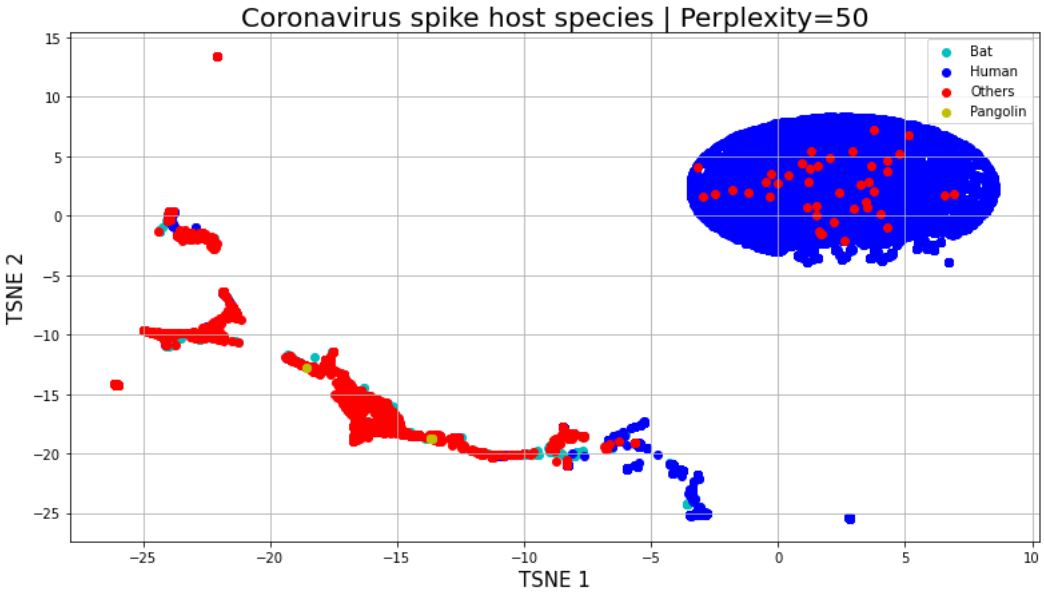

Figure 13: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=50.

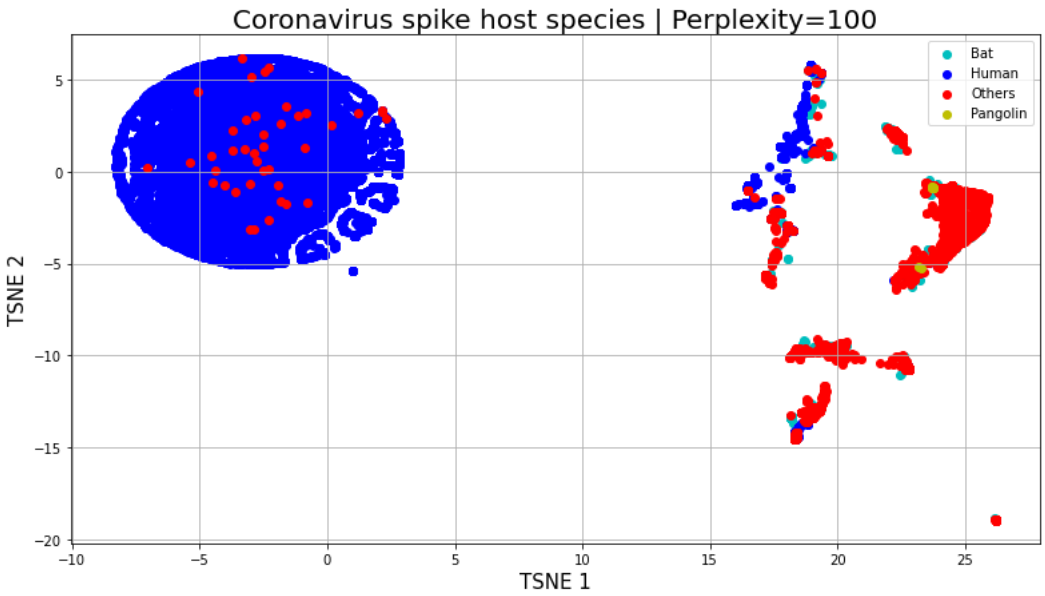

Figure 14: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100.

Figures 11, 12, 13 and 14 show t-SNE 2D scatterplots for different values of perplexity. We can notice firstly that the result of this dimensionality reduction method is very different from what PCA provided with figure 10. In this case, protein sequences are arranged in some clouds and there is some grouping according to the species. In particular, we considered four perplexity values since very often different values of perplexity result in significantly different results and it has been observed a tendency towards clearer shapes as the perplexity value increases in case of datasets with respectively two concentric circles and the S-curve topologies [33]. In our case we notice a more "compact" grouping of data with perplexity= 100 while instead decreasing its value the sparsity of data increases. In fact, the larger the perplexity, the more non-local information will be retained in the dimensionality reduction result, consequently we observe that the clusters found by t-SNE will become consistently more well-defined with the increase of perplexity. However we notice an exception in the human class: human sequences make a big cluster with ellipse shape and this regularity is present in all the t-SNE plots. We notice also that there are sequences of some humans that differ from this behavior and are far from this ellipse and close to other animals. Finally we see that the human blue ellipse has some sparse red points overlapping on it.

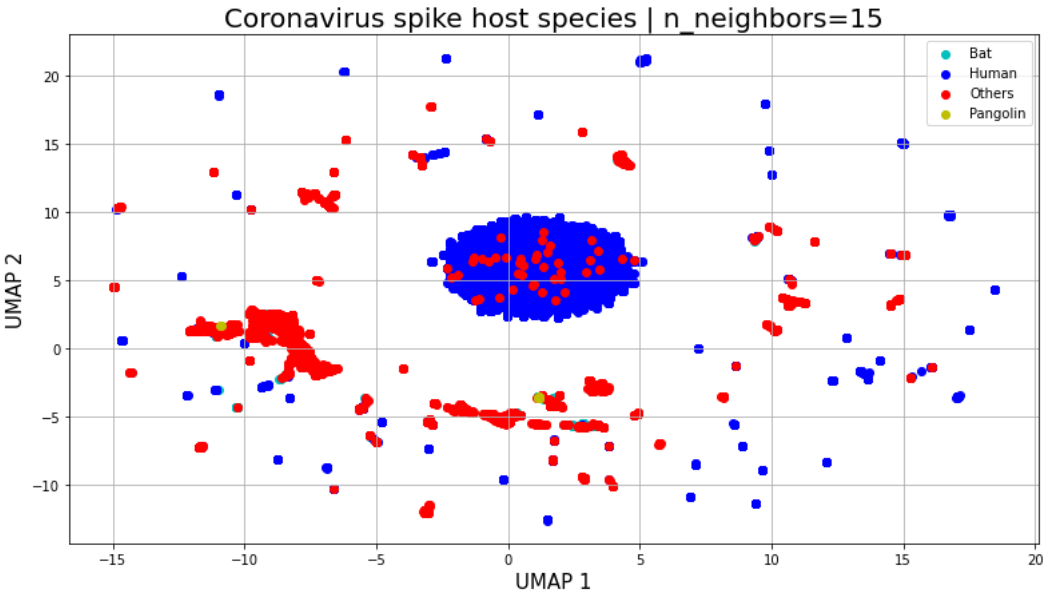

Figure 15: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors=15.

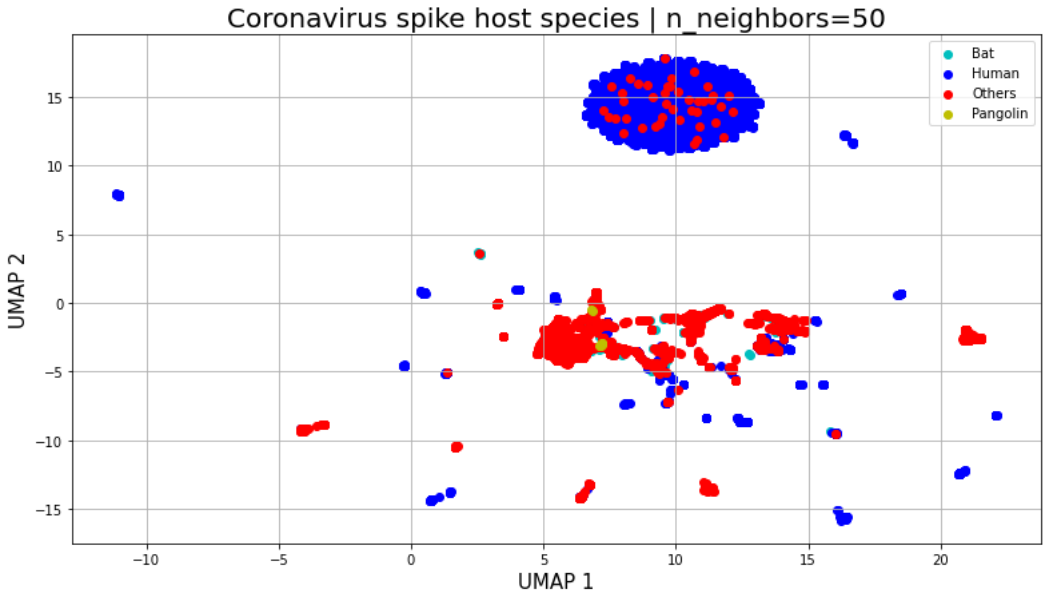

Figure 16: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors=50.

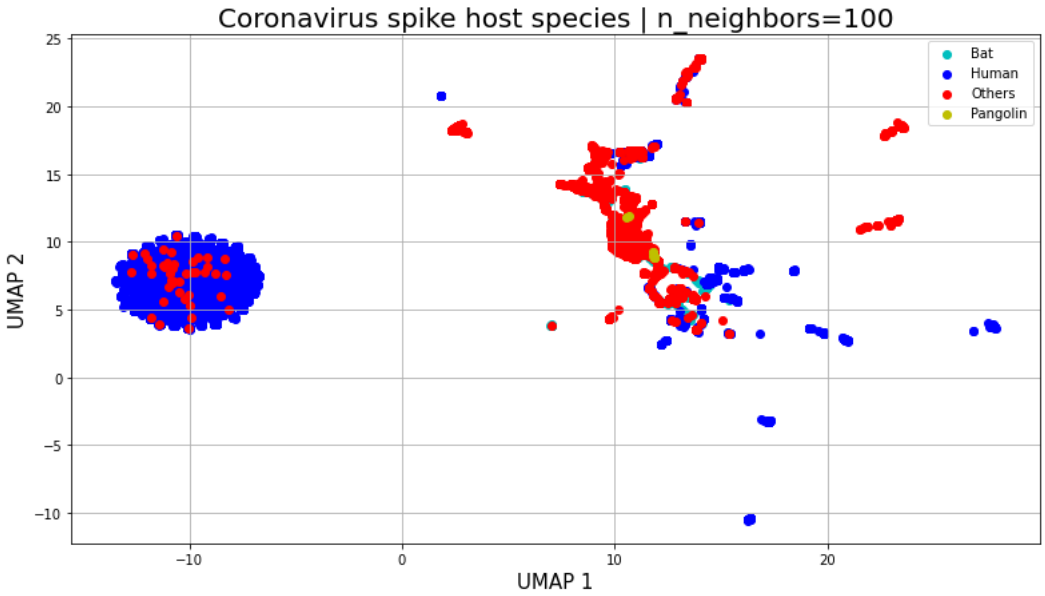

Figure 17: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors=100.

The last dimensionality reduction method used is UMAP that as t-SNE aims to conserve as much as possible the local structure of the original space but in this case the k nearest neighbors are different from node to node. UMAP has several hyperparameters that can have a significant impact on the resulting embedding. In this analysis we have considered different values of a parameter that is the number of neighbors. This parameter controls how UMAP balances local versus global structure in the data. It does this by constraining the size of the local neighborhood UMAP will look at when attempting to learn the manifold structure of the data. This means that low values of number of neighbors will force UMAP to concentrate on very local structure (potentially to the detriment of the big picture), while large values will push UMAP to look at larger neighborhoods of each point when estimating the manifold structure of the data, losing fine detail structure for the sake of getting the broader of the data [35]. Hence, while t-SNE preserves local structure in the data, UMAP claims to preserve both local and most of the global structure in the data. Figures 15, 16 and 17 show UMAP 2D scatterplots for different values of the parameter.

As in case of t-SNE, we can notice that there is still the ellipse made mainly by human species sequences and a second goup of data that is more sparse. As the number of neighbors increases, this second group of data reduces its

sparsity, similarly to what happens in t-SNE increasing the perplexity value parameter. In fact, increasing this value, UMAP manages to see more of the overall structure of the data, gluing more components together, and better coverying the broader structure of the data.

Thus from this first configuration, it comes out that the Neural Network has been able to capture some general structure of the amino acids sequences and methods that preserve the local structures (t-SNE and UMAP) are preferred to PCA that aims to preserve the variation. This suggests that data in the high dimensional space are not linearly separable but they may lie in a more complex structure. Also t-SNE and UMAP do not show a perfect and clear result but they seem to reveal some macroscopic properties and clusterings of the data such as that human sequences are mainly ordered to form a regular shape that is an ellipse while animals are separated from these and do not arrange in a clear shape. There are also some human sequences that mix with this second group and also there are some "islands" in which bat, pangolin and humans are close together but they are very close also to the red points that represent other animals, so for the moment this property that we wanted to reproduce does not emerge very well. However, besides the Neural Network characteristics and capabilities, we have always to keep in mind that we are considering only the first 350 amino acids of Spike protein sequences, so it is also for this reason that we do not see clear and clean clusters.

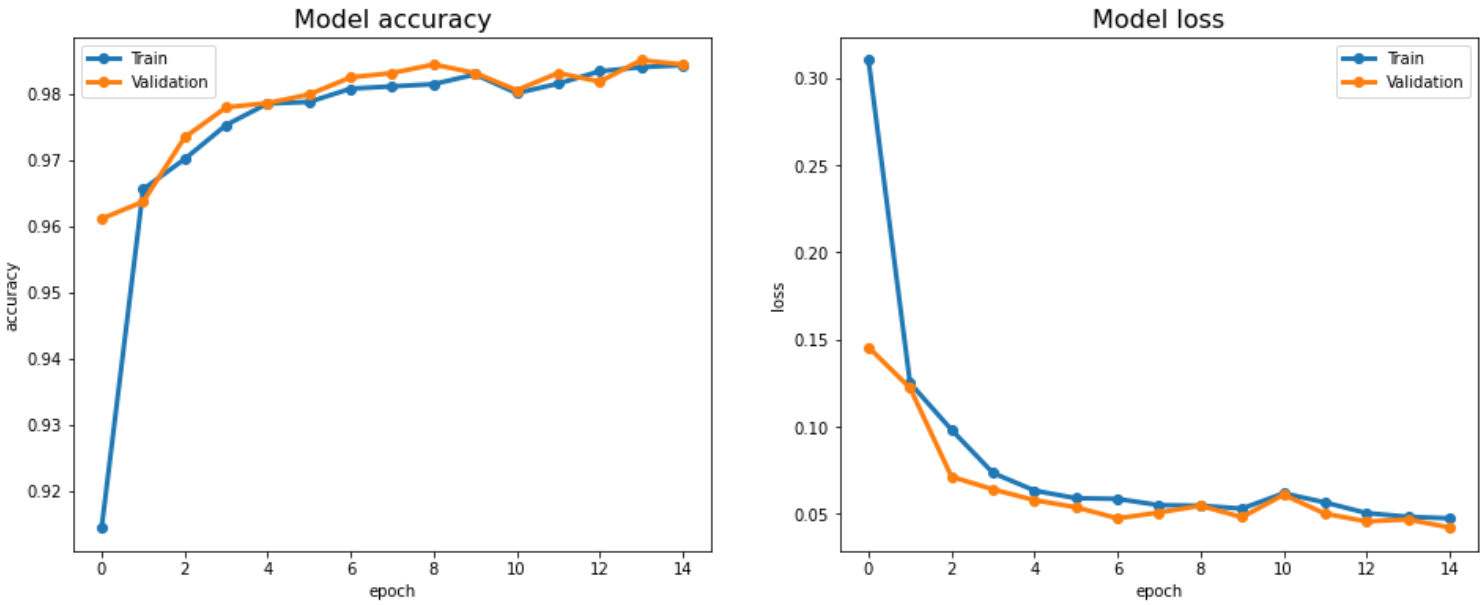

Figure 18: History plots of the first training explored: on the left hand side there is the trend of the model accuracy on training data (blue) and validation set (yellow); on the right hand side there is the trend of the model loss on training data (blue) and validation set (yellow).

Figure 18 shows the model accuracy and model loss history during the training of the first configuration explored. Each epoch lasts around $\sim 175s$ so we have run only 15 epochs. The trend that we observe is a little bit not as expected: the model accuracy for validation set is generally higher than the training one while the model loss for the validation set is lower than the training set and instead we would expect the opposite behavior. This may be due to different reasons such as: validation set consists of "easier" examples than the training set or imbalanced data in validation set. It could be due also to the presence of regularization method that is Dropout because this introduces some noise during training , but while evaluating the model, the model does not use regularization, hence no noise and validation accuracy is a bit better due to the improved generalization produced by Dropout. Now let us check if this problem persists in case of classification with balanced classes. Anyway, from this trend in figure 18 we can say that the network is learning since the training loss decreases with epochs to a point of stability and has a small gap with the validation loss. This indicates a situation of good fit.

The second case discussed is the one in which we apply a simple *undersam-*

pling method decreasing the number of majority classes to the number of data of the minority class. Before doing it, we shuffle the amino acids sequences because sequences which are near in the FASTA file seem to be more similar one to each other but we want to check if the neural network is able to provide a better dense vector representation due to the balancing of classes and not due to a lucky choice or very similar sequences in each group of data.

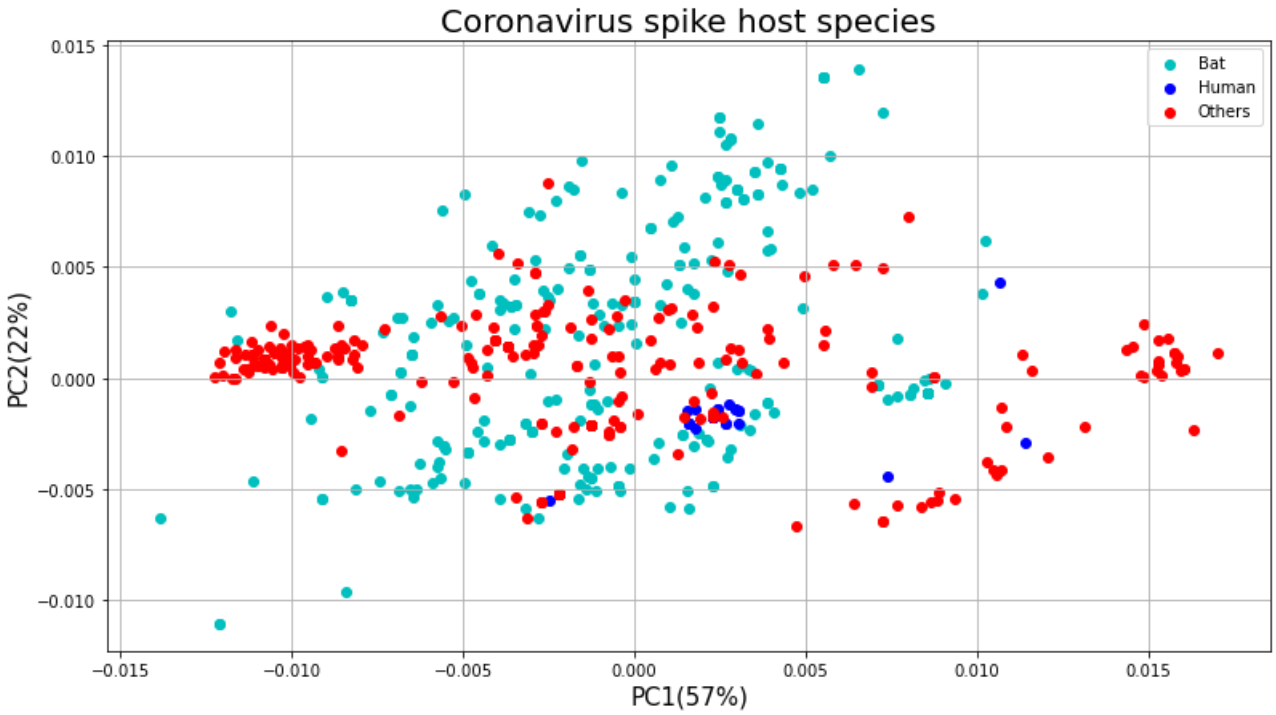

Figure 19: Principal component analysis (PCA) plot of Spike protein sequences with a dataset for classification made by 260 human protein sequences, 260 bat protein sequences, 260 other animals protein sequences. PCA was applied on protein sequences embedding vectors. Percentage of variation accounted for by each component is shown in brackets with the axis label.

Figure 19 shows PCA scatterplot of Spike amino acids sequences for this balanced classes configuration. We can notice that here the situation is even worse because the percentage of variation explained by each principal component is lower and this is reflected into a plot that, again, does not show a clear separation between data belonging to different classes.

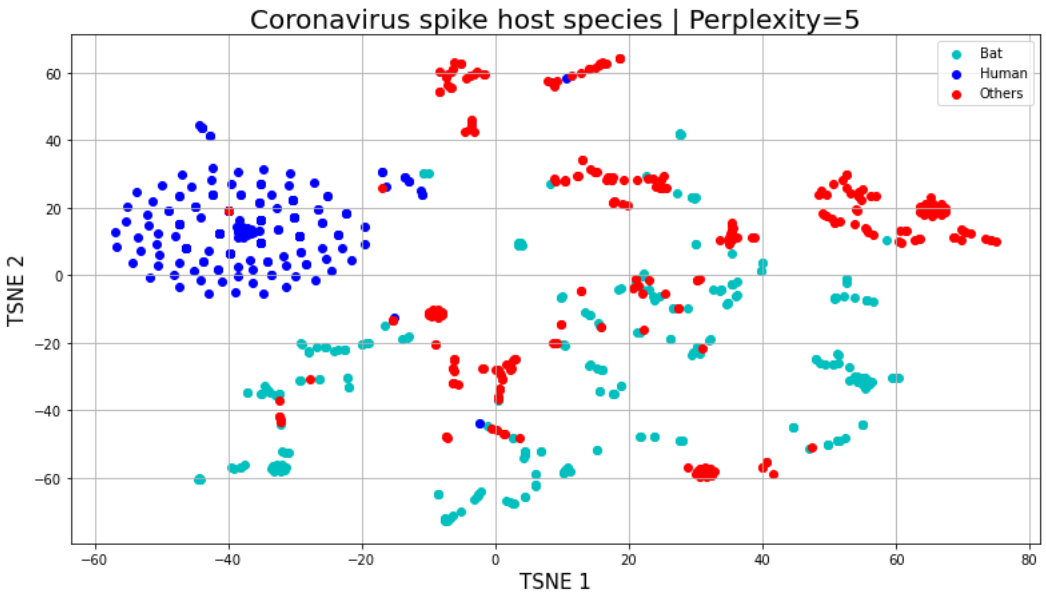

Figure 20: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=5 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

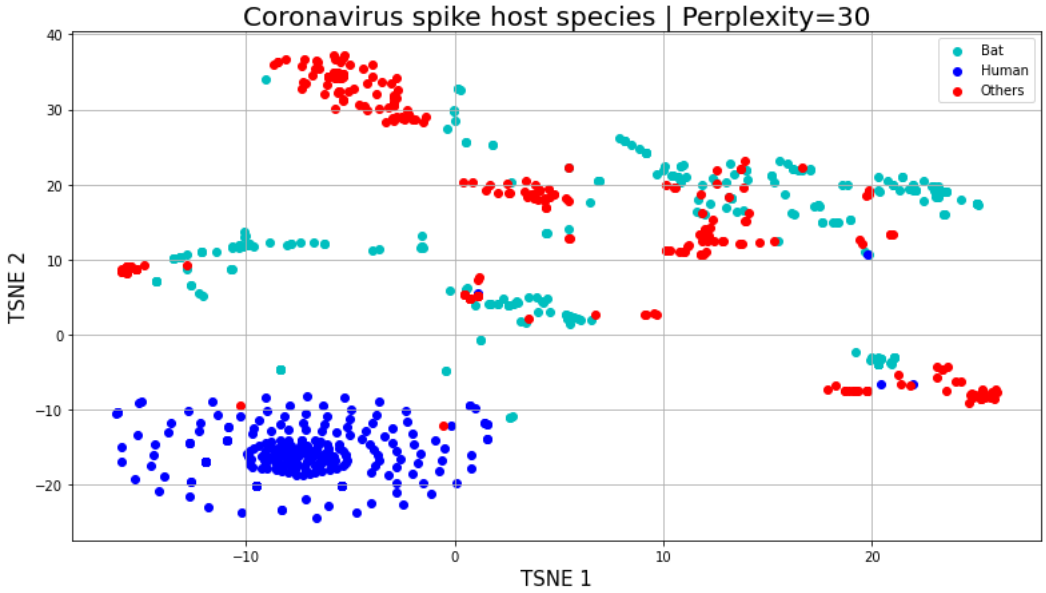

Figure 21: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=30 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

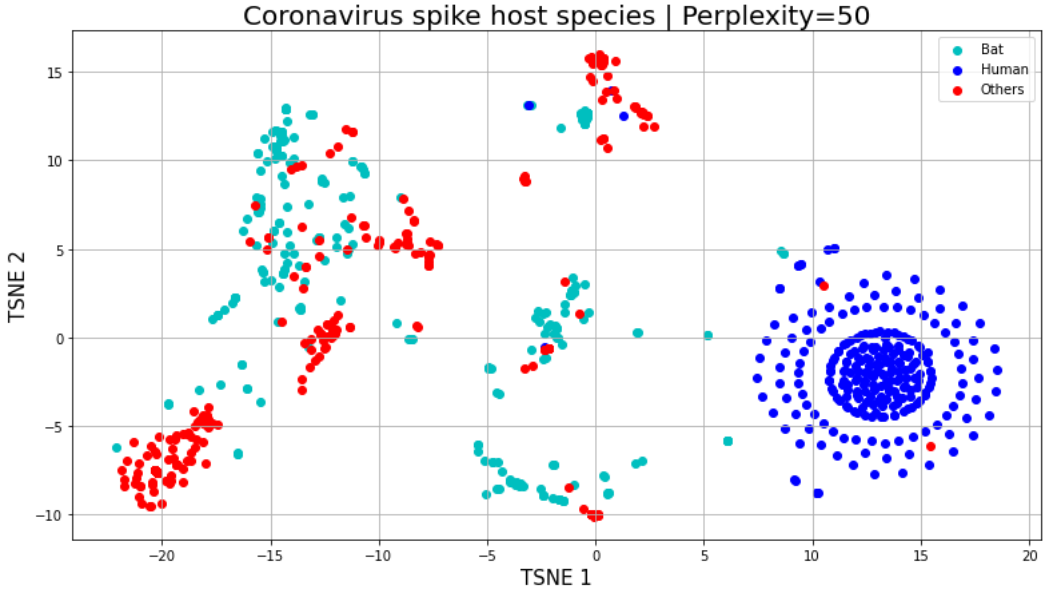

Figure 22: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=50 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

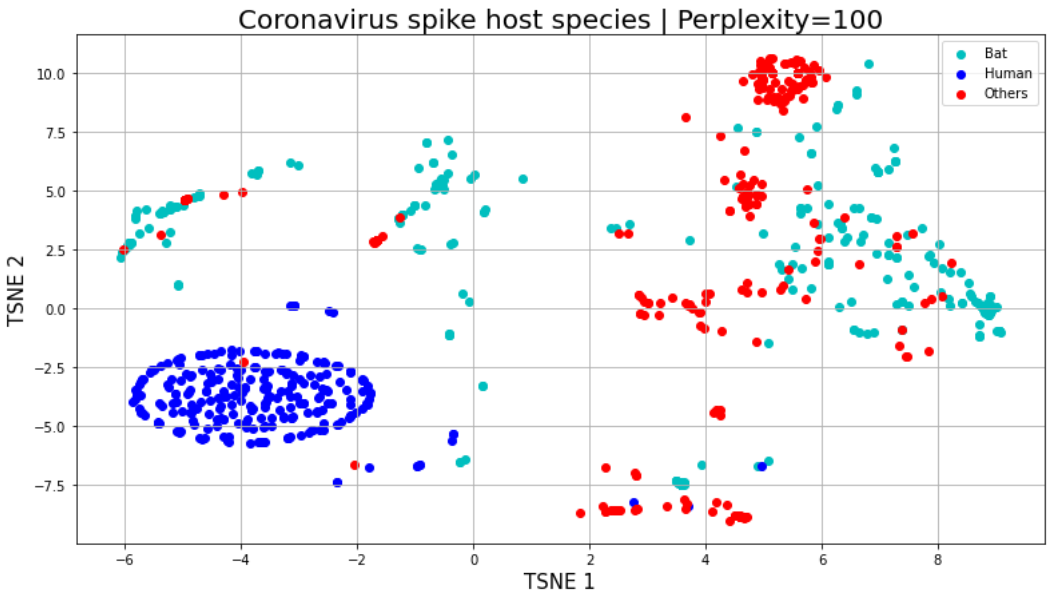

Figure 23: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

Figures 20, 21, 22 and 23 show t-SNE plot in this balanced classes case for five values of perplexity: 5, 30, 50 and 100 as before. The first thing that we can notice is that humans still arrange themselves as a regular shape: for values of perplexity equal to 5, 30 and 100 this shape is still an ellipse shape, while for perplexity equal to 50 it has more circular shape with lower density of points at the edges and much higher density in the centre (so like a Gaussian cloud). Thus this could be a real property of human protein sequences considered as average embedding of the relative amino acids dense vectors, since it is conserved also in this case in which we have decreased the number of humans data points and we have "leveled the playing field" among the three categories. Moreover, generally sequences are spontaneously divided into clusters associated to host species already when the value of perplexity is equal to 5, while instead in case of imbalanced classes the same dimensionality reduction method for perplexity equal to 5, i.e. see figure 11, the clouds of classes species were much more mixed. Only with perplexity equal to 100 (figure 14) the situation became more ordered. However also in this case there is still some degree of overlapping. Without knowing the labels, we could have distinguished only the human cloud as separated cluster, that is always more regular and far with respect to the other species but nothing

more.

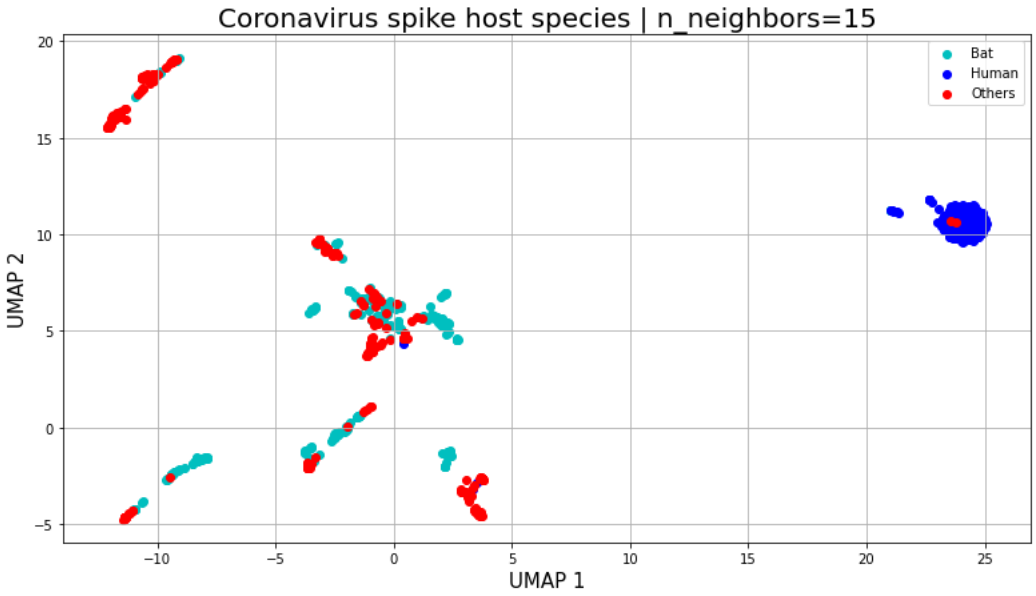

Figure 24: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 15 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

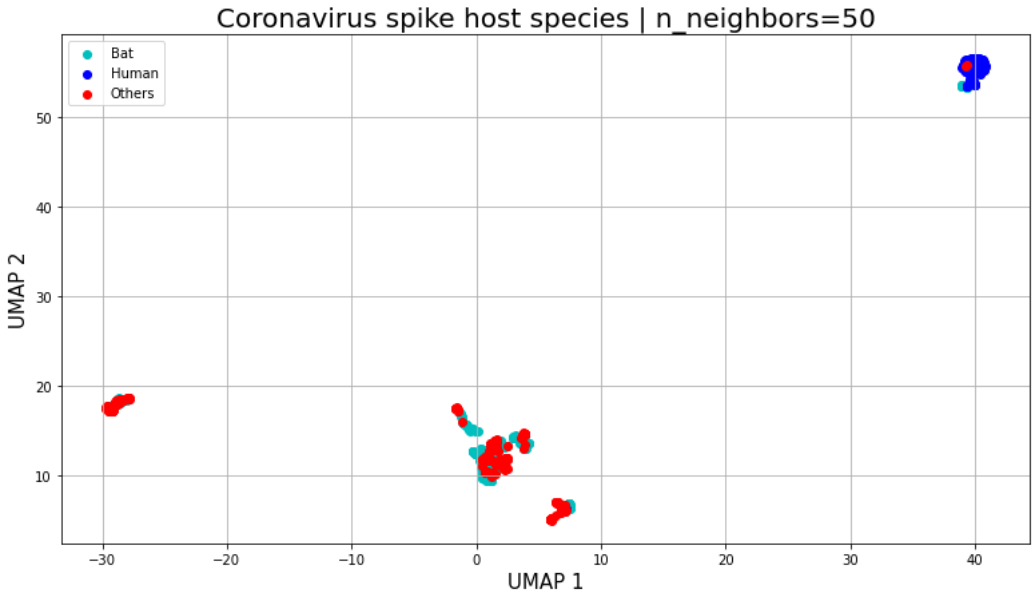

Figure 25: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 50 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

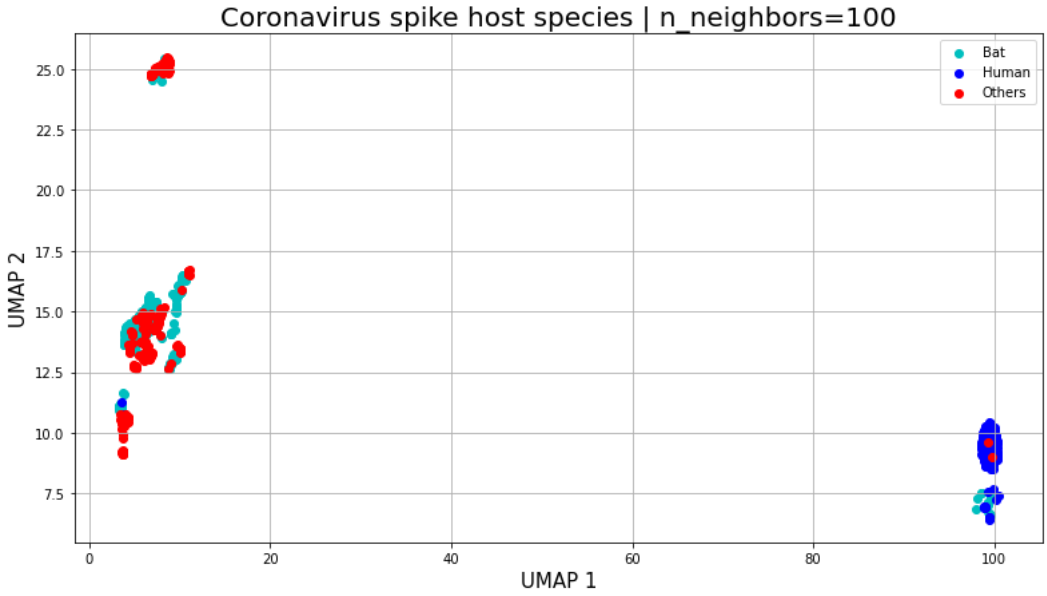

Figure 26: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

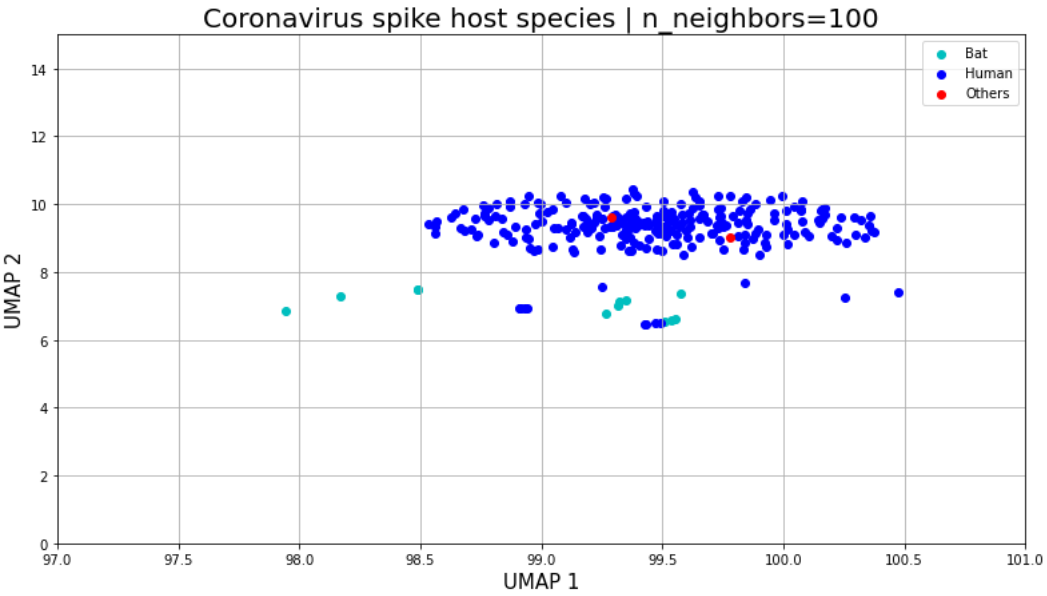

Figure 27: Down right corner detail of UMAP plot in figure 26.

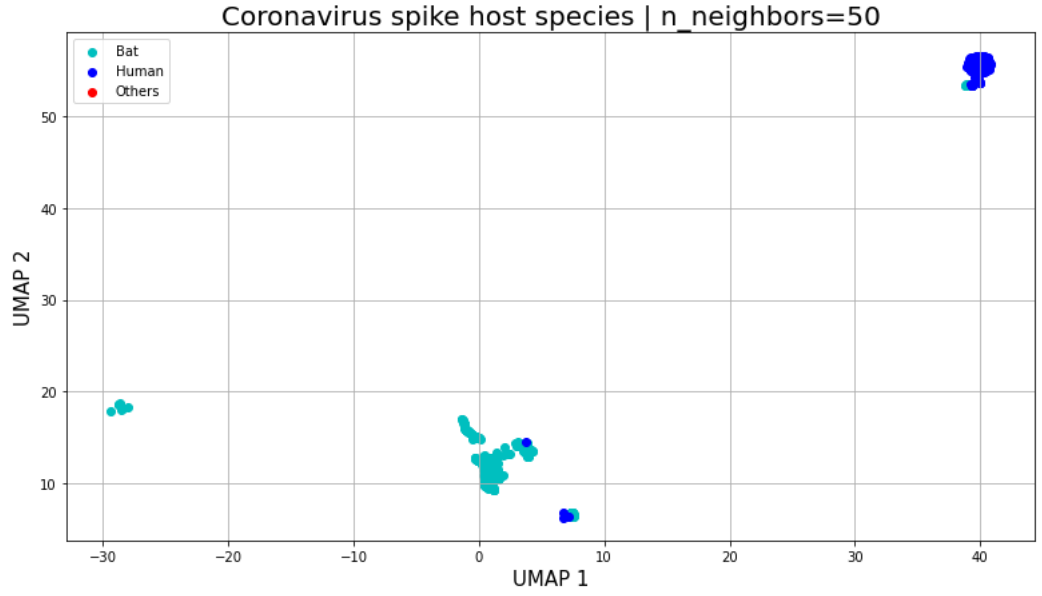

Figure 28: UMAP plot of figure 25 showing only humans and bats host species.

Figures 24, 25 and 26 show the two components plots of UMAP dimensionality reduction method changing the number of neighbors. With UMAP,

host species make even more "tidy" and compact groups and the distinction between humans and animals is much more net. Furthermore, figure 27 and 28 show two details: first the fact that humans and bats start to show effectively more evident similarities (in figure 27 the most of "strangers" that lie near the human ellipse big island is from bat sequences) and secondly that also here, as in t-SNE, there is still some overlapping between species in particular between humans and other animals labeled in red (for example in figure 27 we still notice some red points in the core of the blue regular human shape as it was for imbalanced classes case).

Thus one natural strategy that we apply, in order to understand if those red points are effectively equal to human blue species, is to consider as length of the input sequences the maximum length of the sequences that is 1852.

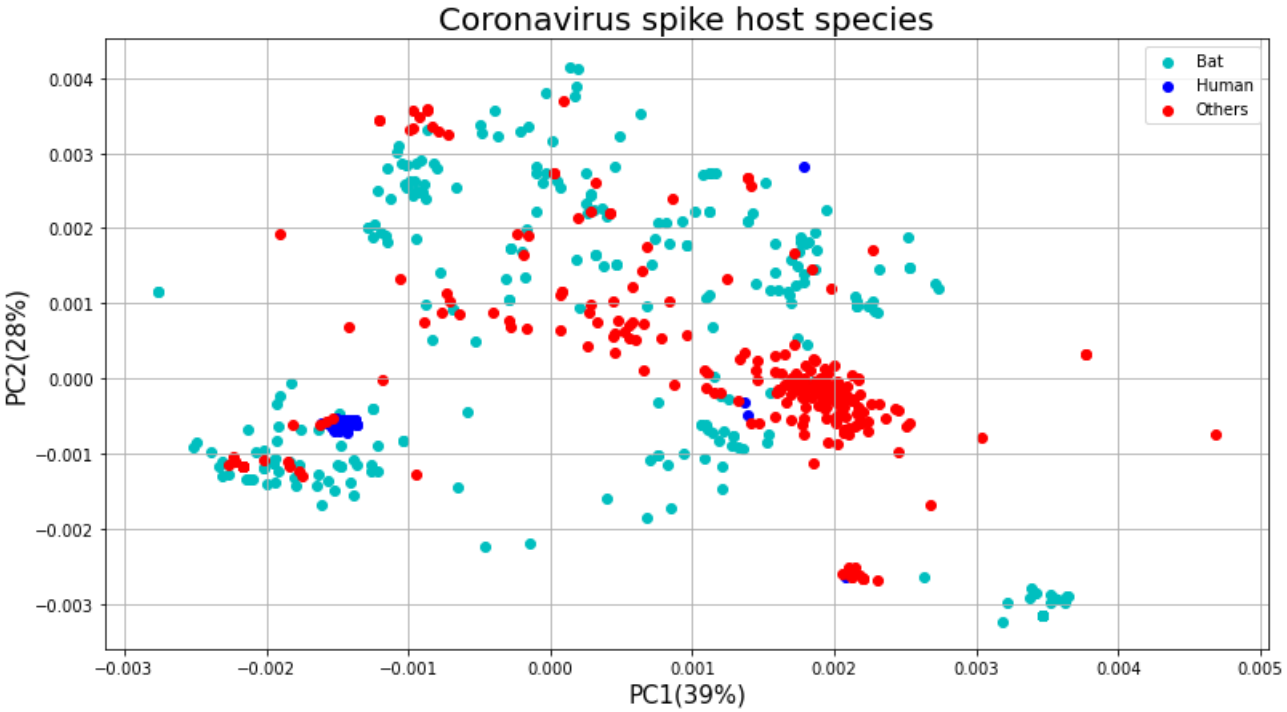

Figure 29: Principal component analysis (PCA) plot of Spike protein sequences with a dataset for classification made by 260 human protein sequences, 260 bat protein sequences, 260 other animals protein sequences. PCA was applied on protein sequences embedding vectors considering as length of input sequences the maximum length of the sequences. Percentage of variation accounted for by each component is shown in brackets with the axis label.

Figure 29 shows PCA projections plots in case we consider the maximum length of the sequences as input sequence length. The situation with respect to the previous cases becomes even worse: percentage of variation accounted for by each components is even lower. This could mean that in the initial 10 dimensional space data are arranged in more complex structures (no linearly separable) than the cases in which we considered only the first 350 amino acids of the sequences.

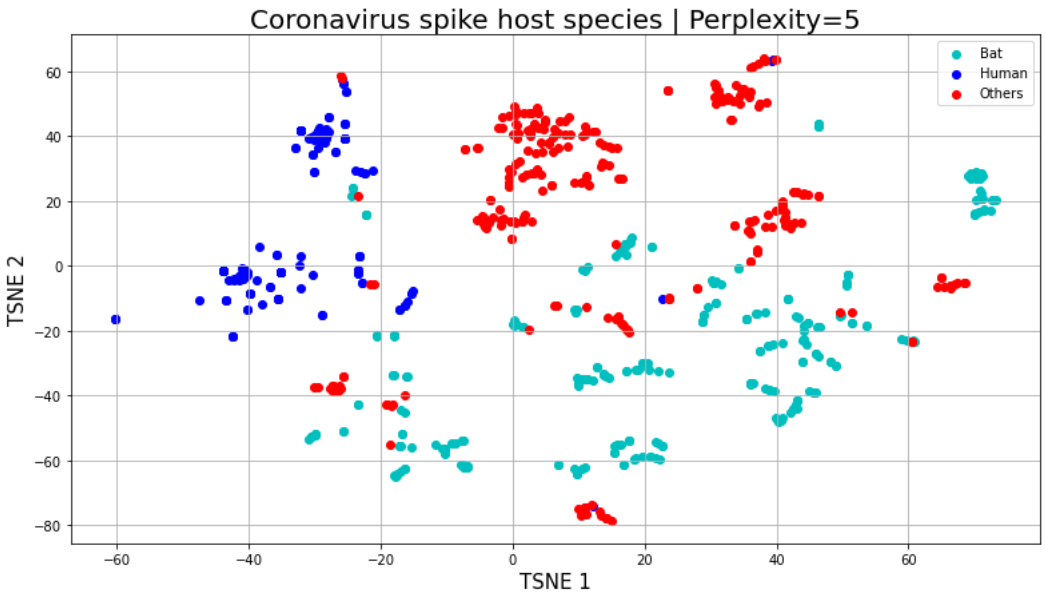

Figure 30: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=5 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

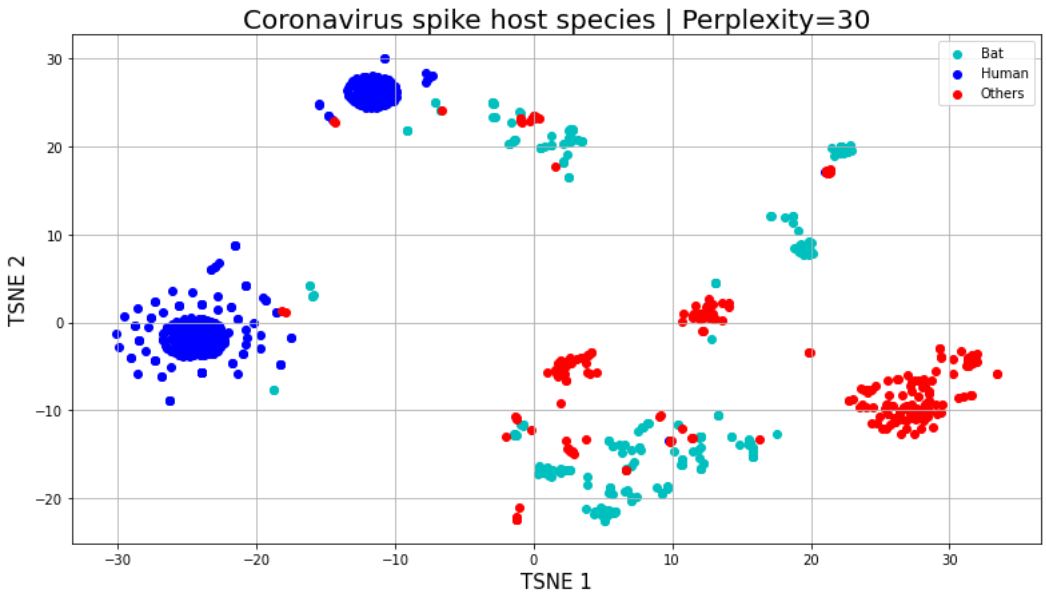

Figure 31: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=30 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

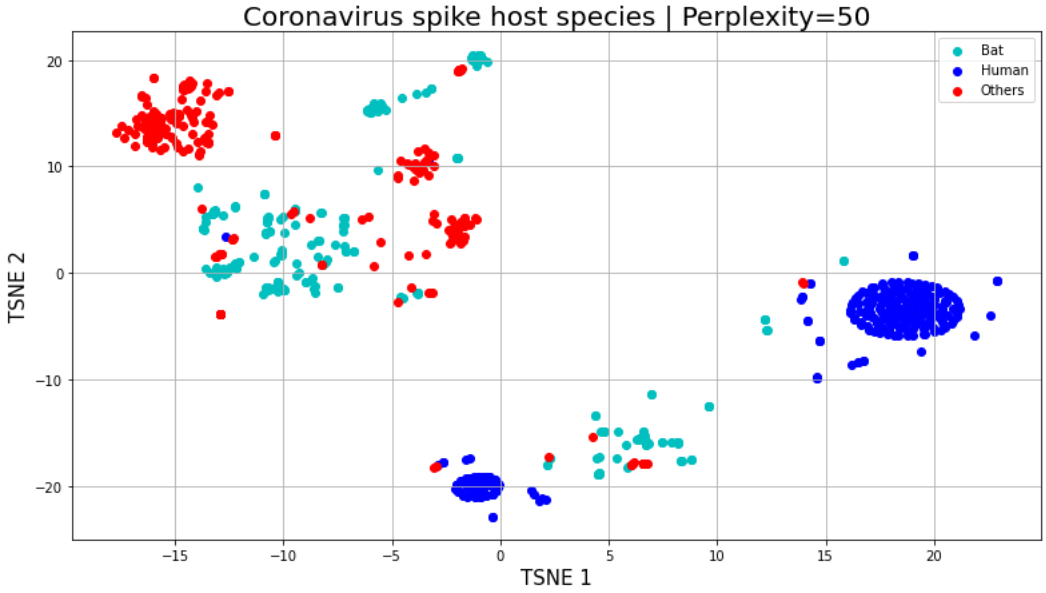

Figure 32: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=50 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

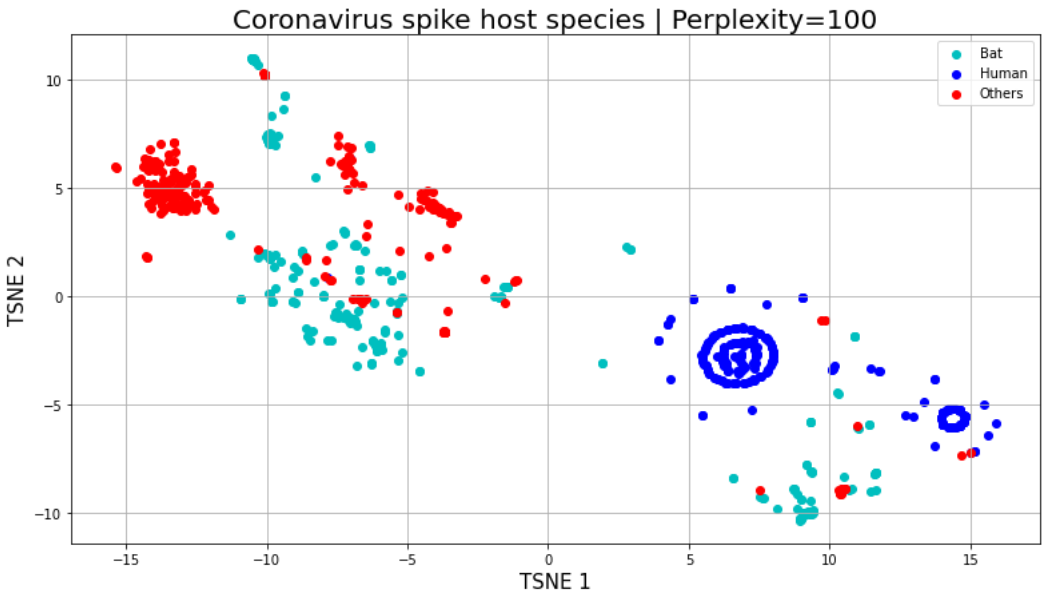

Figure 33: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

The scenario is once again much more different in case of t-SNE: human host species now make clearly two regular shapes (circle or ellipse) with data surrounding this shape as "satellites" human sequences. And, moreover, now human clouds have "removed" red points from their core islands of sequences meaning that there were some characteristics and meaningful patterns of amino acids only after the first 350 amino acids sequences that now the neural network could catch.

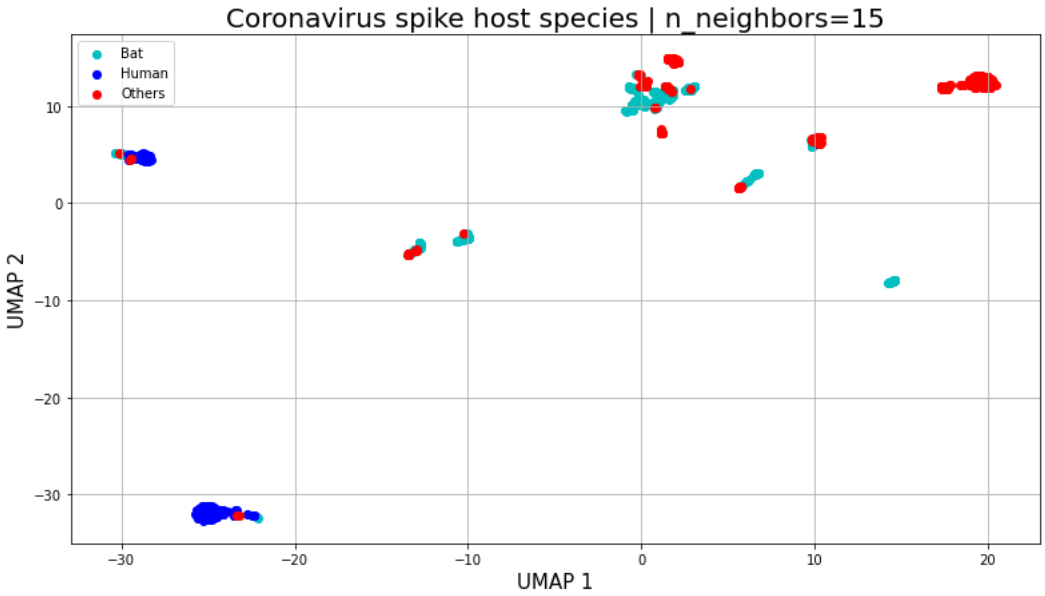

Figure 34: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 15 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

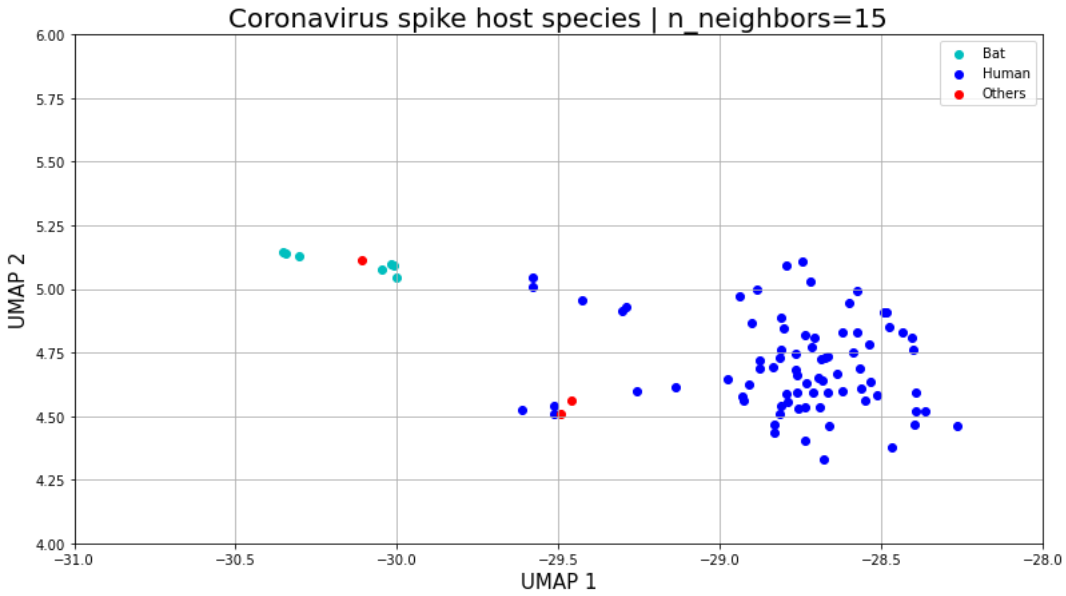

Figure 35: Up left corner detail of UMAP plot in figure 34.

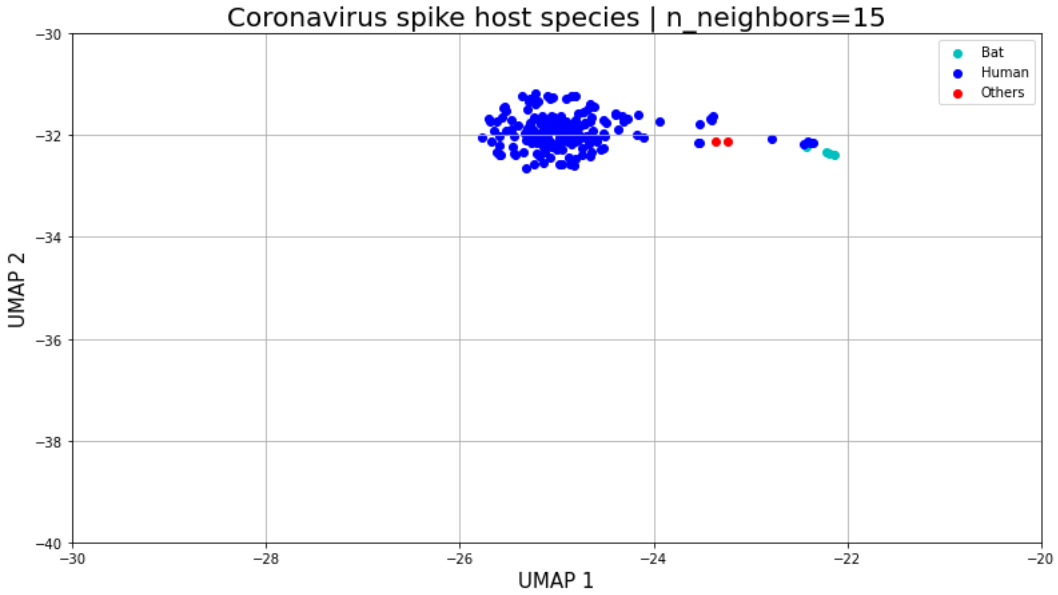

Figure 36: Down left corner detail of UMAP plot in figure 34.

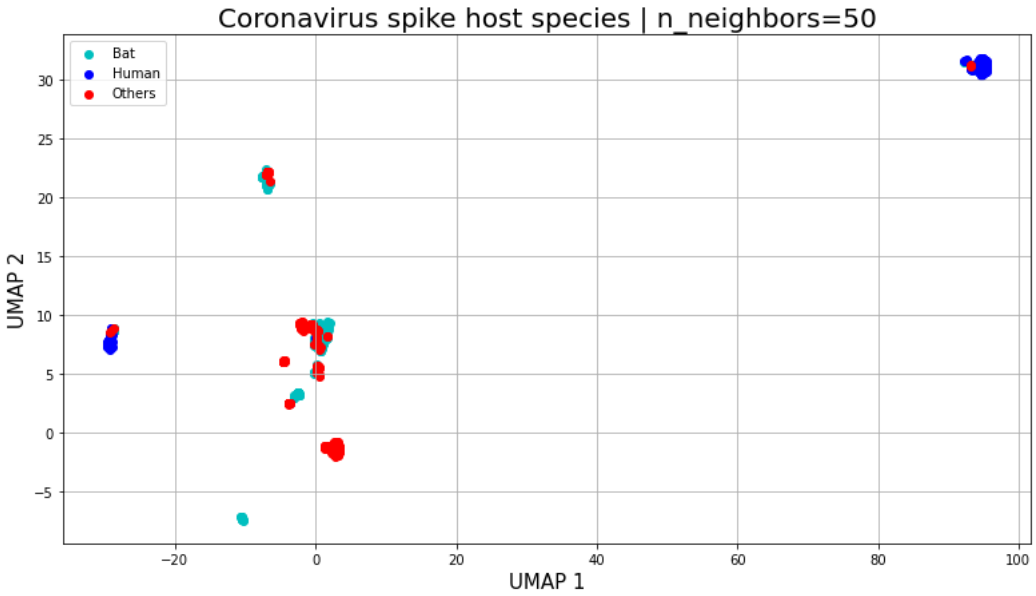

Figure 37: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 50 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

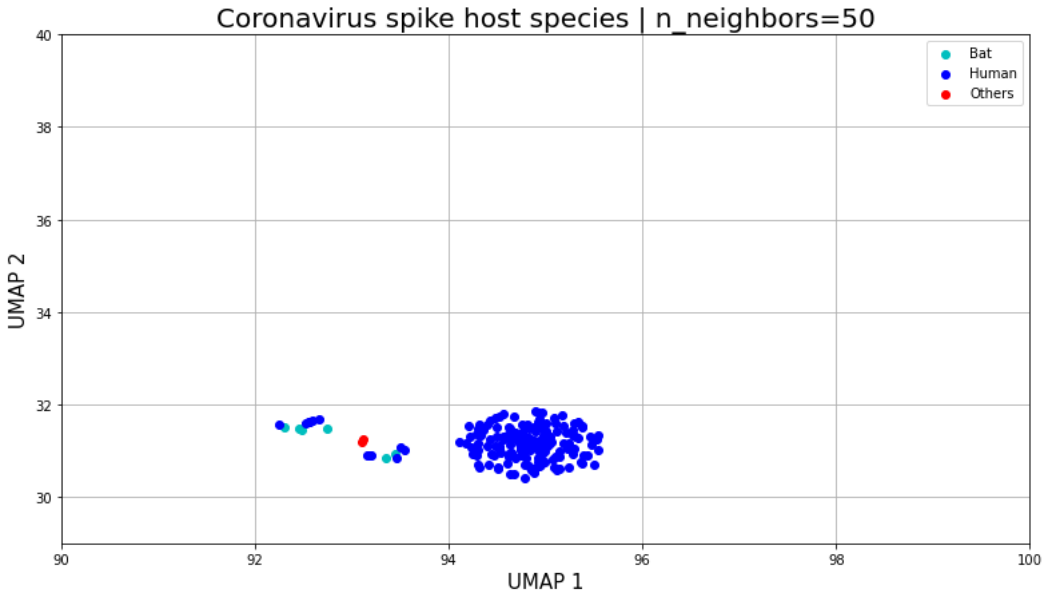

Figure 38: Up right corner detail of UMAP plot in figure 37.

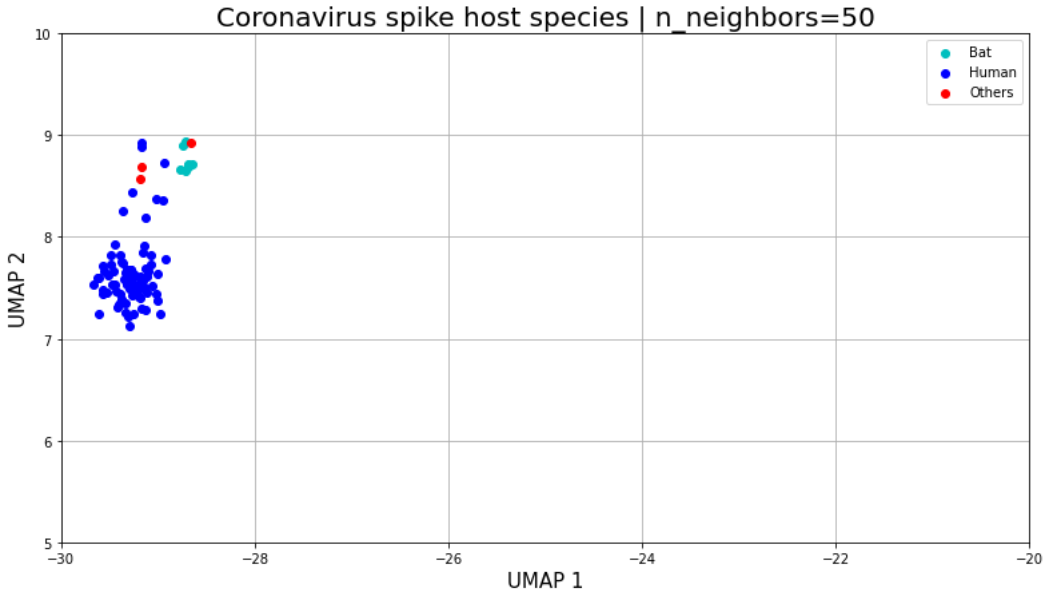

Figure 39: Left hand side detail of UMAP plot in figure 37.

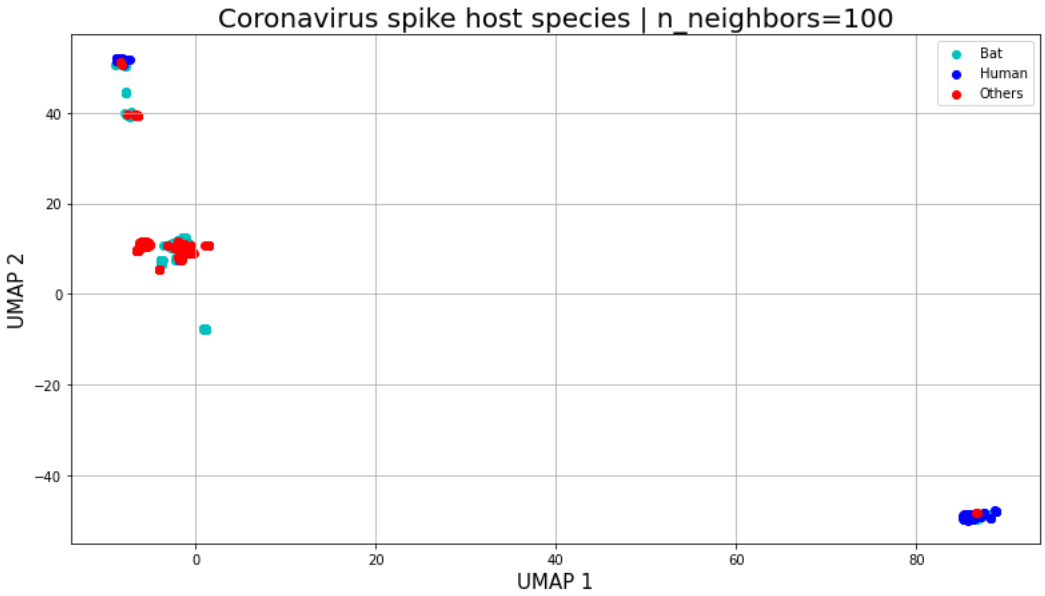

Figure 40: UMAP plot visualizing Spike protein sequences with the first two UMAP components as the axes of the plot and with number of neighbors equal to 100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species and considering the maximum length of amino acids sequences as input sequence length.

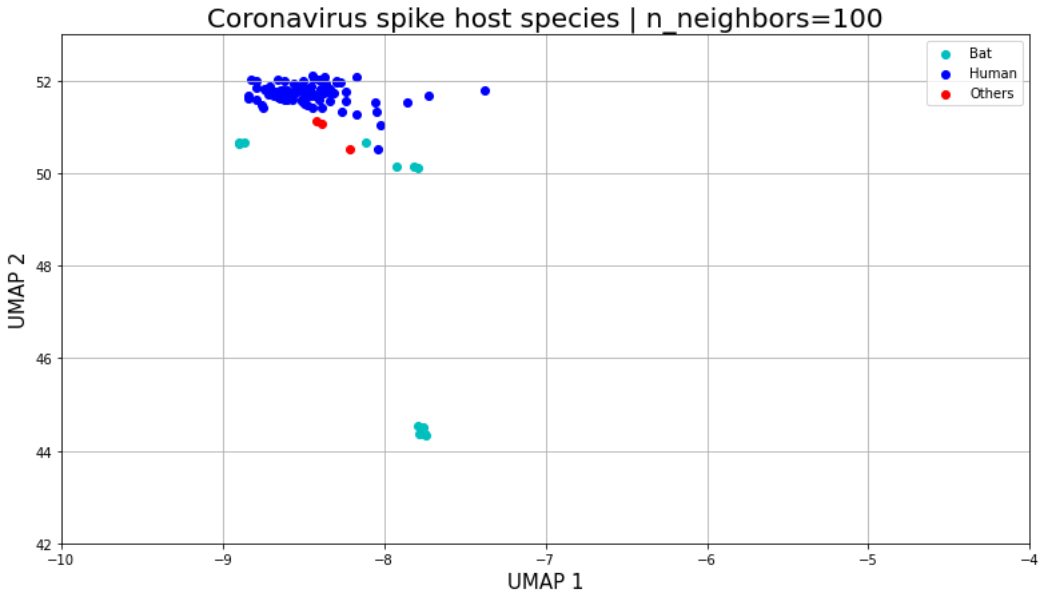

Figure 41: Up left corner detail of UMAP plot in figure 40.

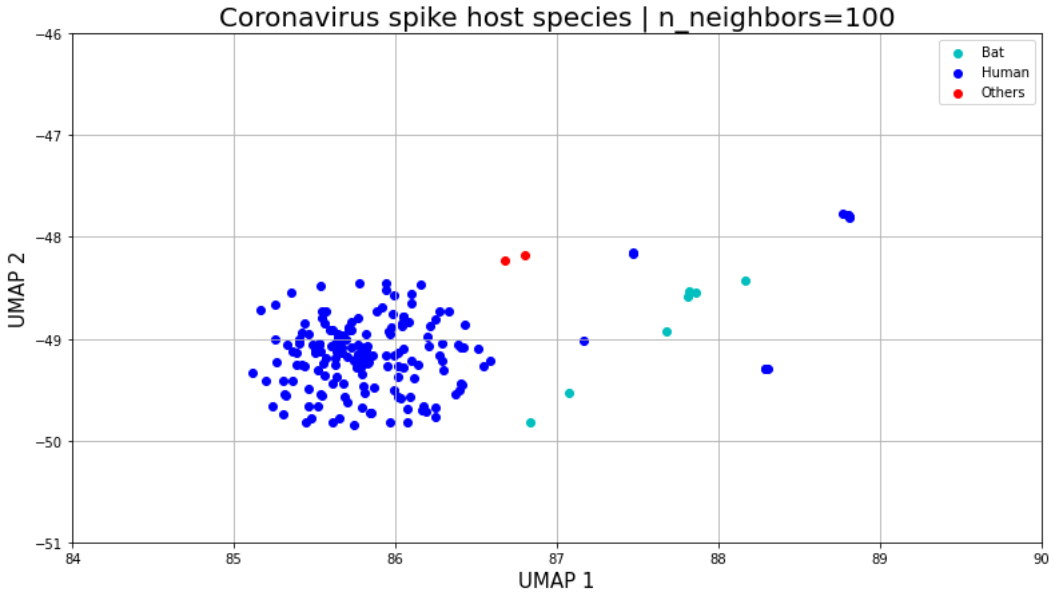

Figure 42: Down right corner detail of UMAP plot in figure 40.

From figures 34, 37 and 40 and the ones that show the relative details, we notice that also UMAP for different values of perplexity has captured this double-regular-clouds-structure of human protein sequences, differently from the 350 AAs case in which there was one regular ellipse shape and then other blue points scattered in the space. Moreover also here the red points,

that usually were overlapping on the blue human cloud, drift away and more and more clearly we notice that a consistent number of bat sequences are "menacingly" near to the human Spike protein sequences.

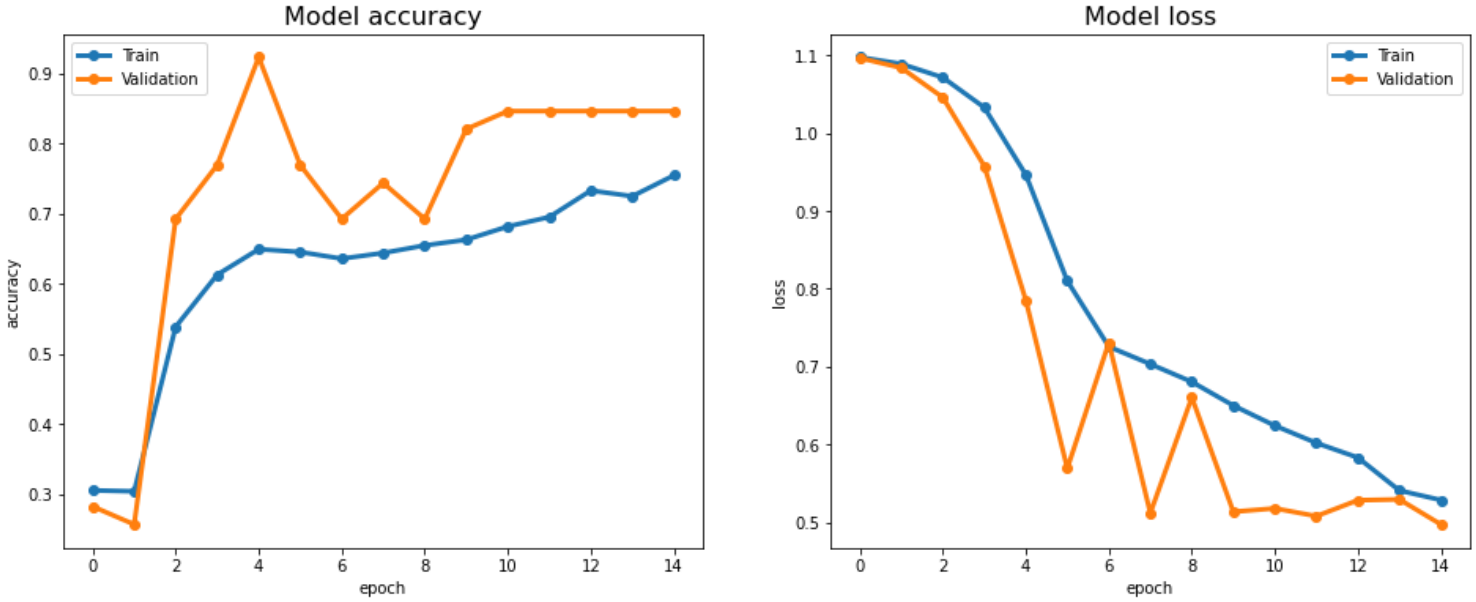

Figure 43: History plots of the second configuration explored, i.e. training the classifier on 260 human protein sequences, 260 bat protein sequences and 260 other animals protein sequences with input sequence length equal to 350 AAs: on the left hand side there is the trend of the model accuracy on training data (blue) and validation set (yellow); on the right hand side there is the trend of the model loss on training data (blue) and validation set (yellow).

Figure 43 shows the history in balanced classes case with considering 350 AAs as input sequence length. We notice that the validation and training accuracy are generally lower (on the other side the loss is higher) than the imbalanced classes case shown in figure 18, that could be an effect of the balancing, but the validation accuracy is still higher than the training accuracy, as well as the validation loss is still lower than the training loss. Furthermore, the validation loss in this case shows a noisier behavior. This could be due to the balancing of data and consequently decreasing of data and so validation dataset has too few examples as compared to the training dataset, hence it does not provide sufficient information to evaluate the ability of the model to generalize. Or, also in this case, the lower validation loss curve with respect to the training may be due to the regularization dropouts inserted in the

model. In this study, we use dropout because Recurrent Neural Networks like LSTM generally have the problem of overfitting, but a further analysis could be to consider the same architecture of the neural network without dropouts inserted and see how the history changes and also how the dense vector representation of protein sequences changes. Moreover, we notice that there is no sign of overfitting in both the reported histories (that is a point in which the validation accuracy starts keeps constant and the validation loss starts to increase with the training loss that starts to decrease and the training accuracy increases). Thus an other investigation could be to increase the number of epochs up to this point. However, for this study we do not consider these configurations but we try a last approach that again "plays" with data, that is to divide the most abundant classes (among humans, bats and other animals) in $N = 8$ subsets each one with 260 data, that is the amount of data of the minority class (bats), and we train our neural network classifier sequentially on a $N - th$ subset of the majority classes but on all of the data from the rare class. In other words, we train the neural network sequentially 8 times on different 260 human host species data, on different 260 other animals host species data (that are the majority classes) and the same 260 bat species data (that is the minority class). For this step we show the dense vector representation of Spike protein sequences using only t-SNE as dimensionality reduction method. These are reported below:

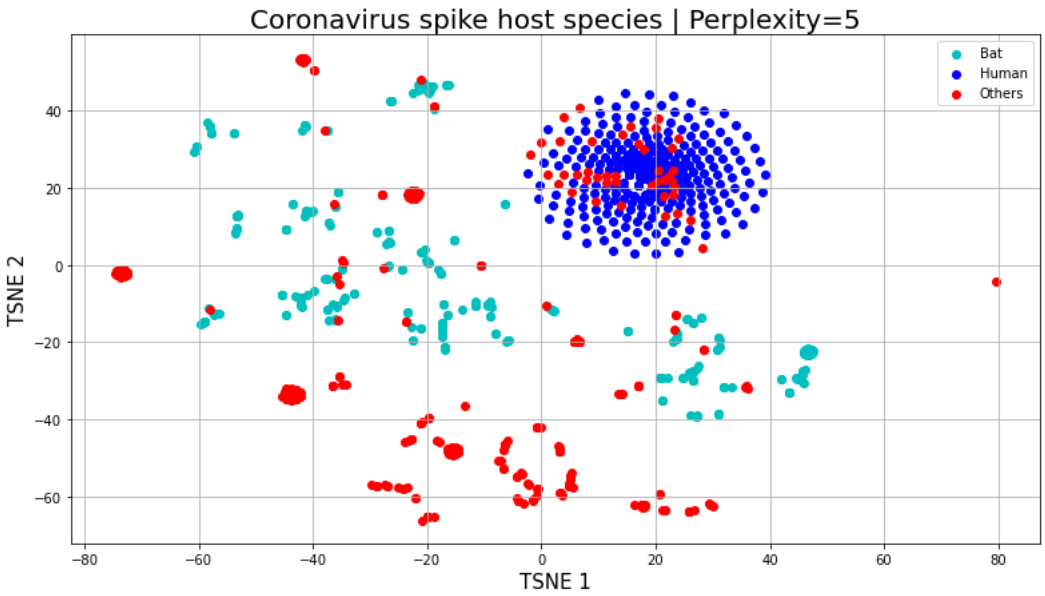

Figure 44: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=5 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species but using the third method.

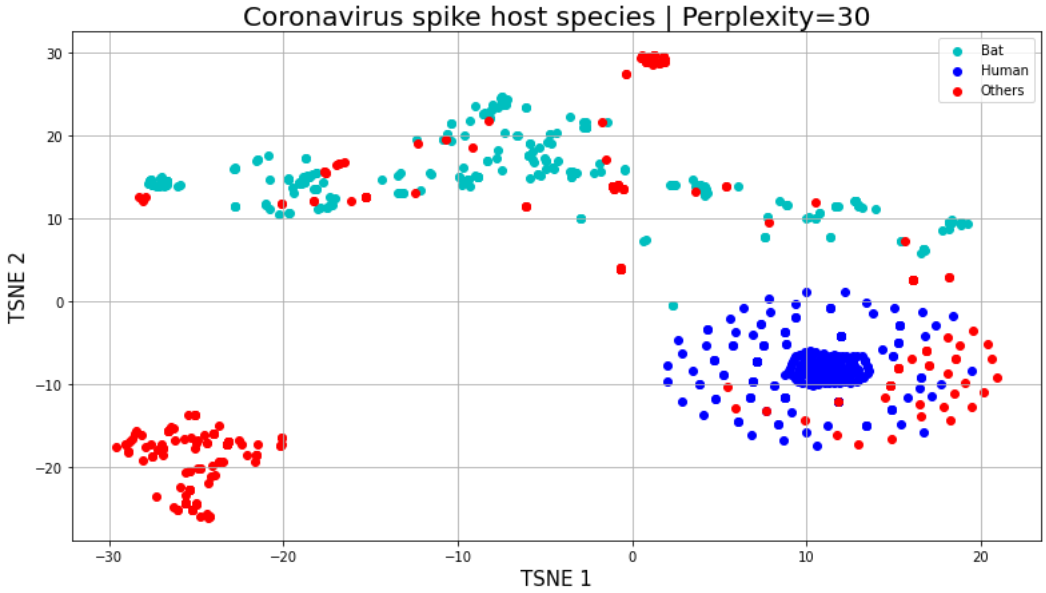

Figure 45: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=30 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species but using the third method.

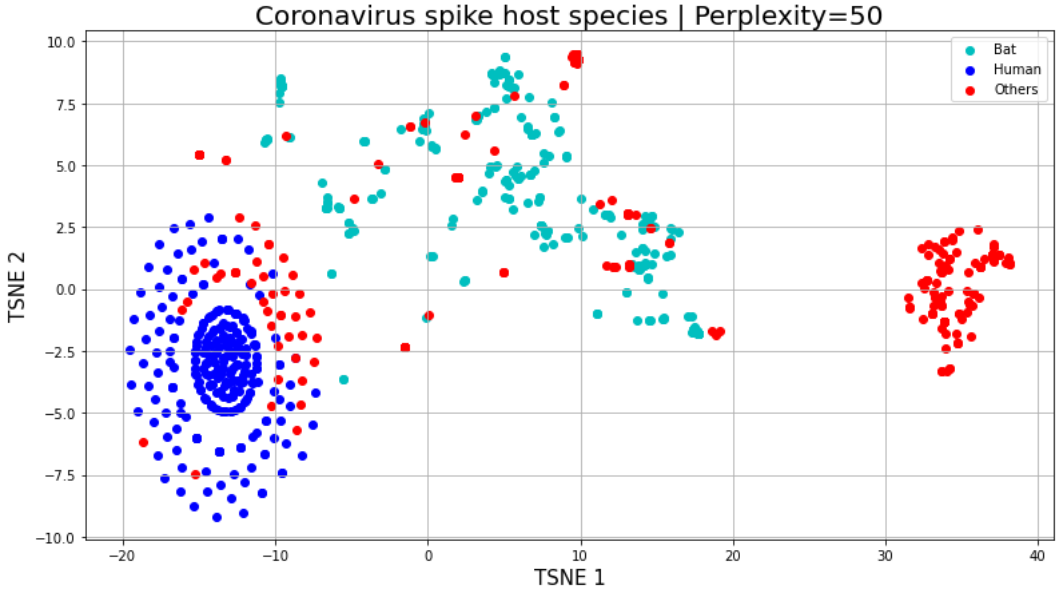

Figure 46: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=50 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species but using the third method.

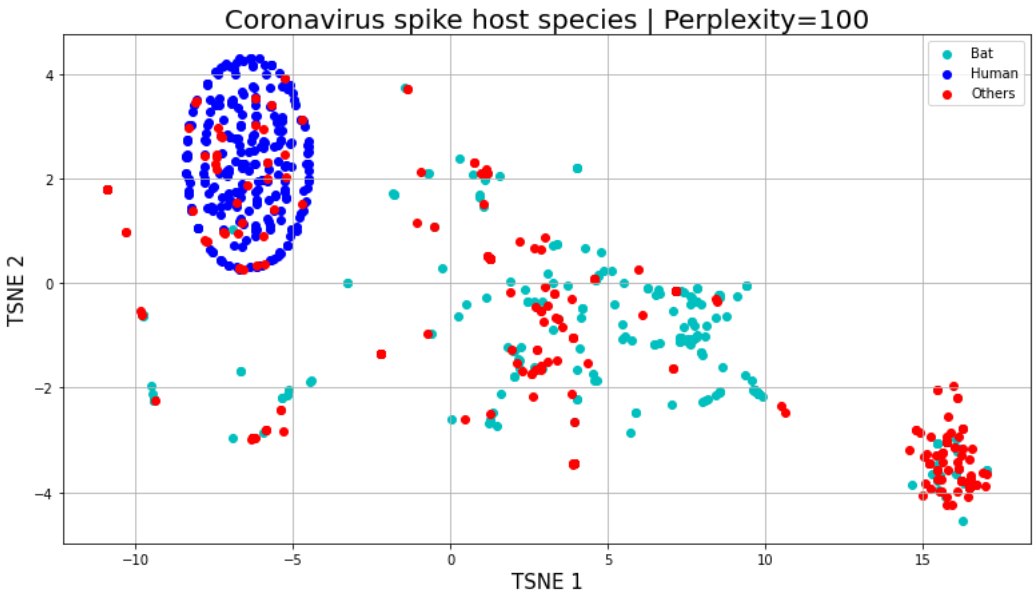

Figure 47: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species but using the third method.

From figure 44, 45, 46 and 47 we can notice that this third method has caused *all* the human sequences to cluster together in the usual regular ellipse shape. We can define this result as robust since this characteristics is true for each running of the t-SNE algorithm with different values of perplexity. On the other hand, looking at t-SNE plots in figures 20, 21, 22 and 23, we notice that in that case of simple undersampling method in which we also used 350 AAs as input length, different human blue points were scattered in the space and so were far from the principal "human blue island". And we have to point out that this can not be due to just a "lucky" choice of data (that means one could argue that we have chosen a subset of human data with sequences that were more similar between each other and that have amino acids structure such that they arrange in that way) because these last figures represent t-SNE plots of the same 260 human protein sequences, 260 bat protein sequences and other animals protein sequences of the previous method but, of course, now they are represented by weights vectors of this last neural network model that has been trained to classify 8 times 260 human protein sequences, 260 bat protein sequences and other animals protein sequences each time changing the majority classes (humans and other animals) and keeping the same data of the minority class (bats). This allows the

minority group to contribute more to the gradient and still allows the model to see more available data in the majority group that with simple undersampling would be lost. There are still red points from other animals sequences that are overlapping with the humans cluster, but, taking into account that the red points moved away from the blue cluster once increased the input sequence length to the maximum length in the previous approach that we proved, this could be due to the fact that important information in the sequences (in order for the network to be able to distinguish them according to the species) are written only after the first 350 AAs and so they are clustered together with humans because that first part is similar to humans.

Furthermore, we notice that these t-SNE plots of this third method show a more clear separation between the 3 classes used in our multi-class classification models with respect to the simple undersampling method. We can notice it with our eyes (our human brain is a powerful pattern recognizer) but we could quantify this using DBSCAN clustering method and comparing the different values of Silhouette Score (SS) parameter. In particular in this analysis we choose to consider t-SNE plots for perplexity value equal to 100 shown in figure 47 and figure 23. Moreover, we choose to keep fixed the minimum number of samples needed to make a cluster to 3 and to check the trend of Silhouette Score as function of the neighborhood radius. We do this for both t-SNE plots. For the one obtained using the simple undersampling method, the Silhouette Score is the following:

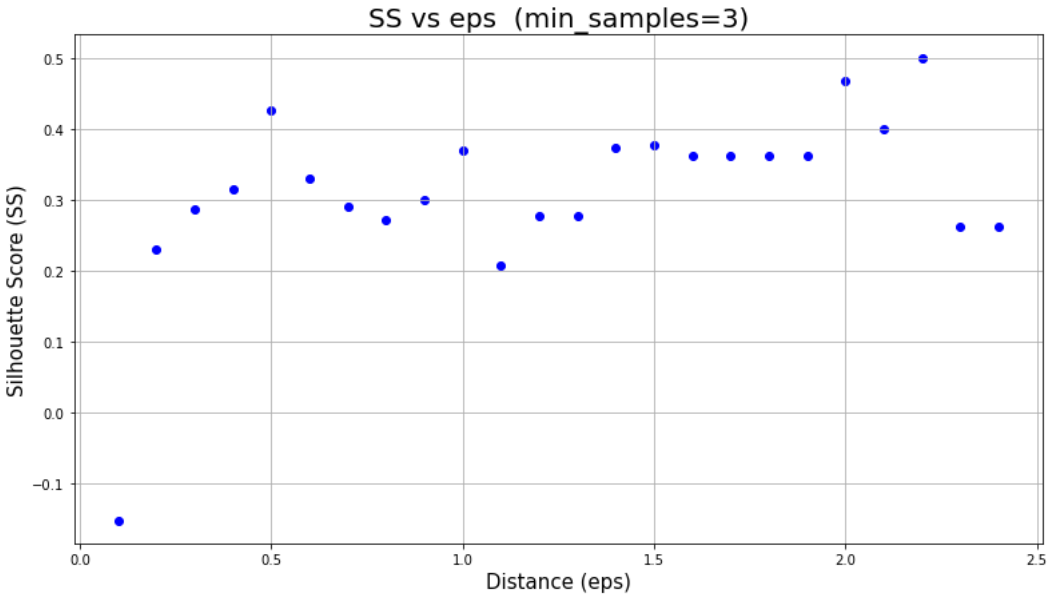

Figure 48: Silhouette Score trend as function of neighborhood distance in case of t-SNE plot (for perplexity=100) obtained with the first simple undersampling method.

Keeping fixed the minimum number of samples to 3 and varying the distance from 0.1 to 3 with step equal to 0.1, we obtain a trend like in figure 48 that has different local maxima and local minima. By setting the radius value to 2.2, that is the x-value corresponding to the maximum y-value (i.e. maximum Silhouette Score), we obtain the following clustering result:

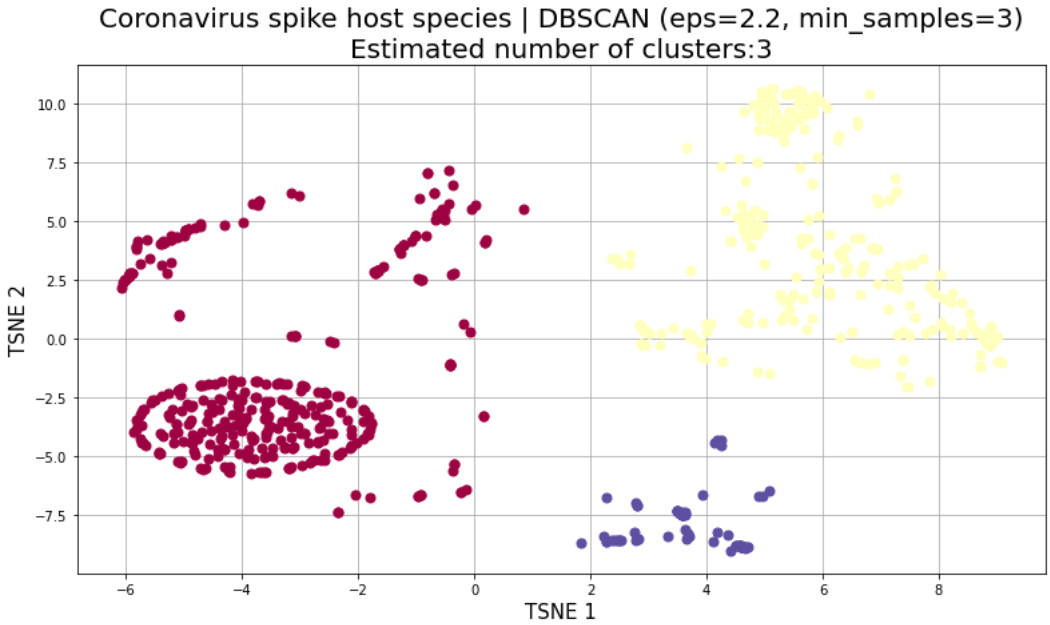

Figure 49: t-SNE plot (perplexity=100) with data labeled according to DBScan clustering algorithm in case of simple undersampling method, with number of samples=3, radius=2.2.

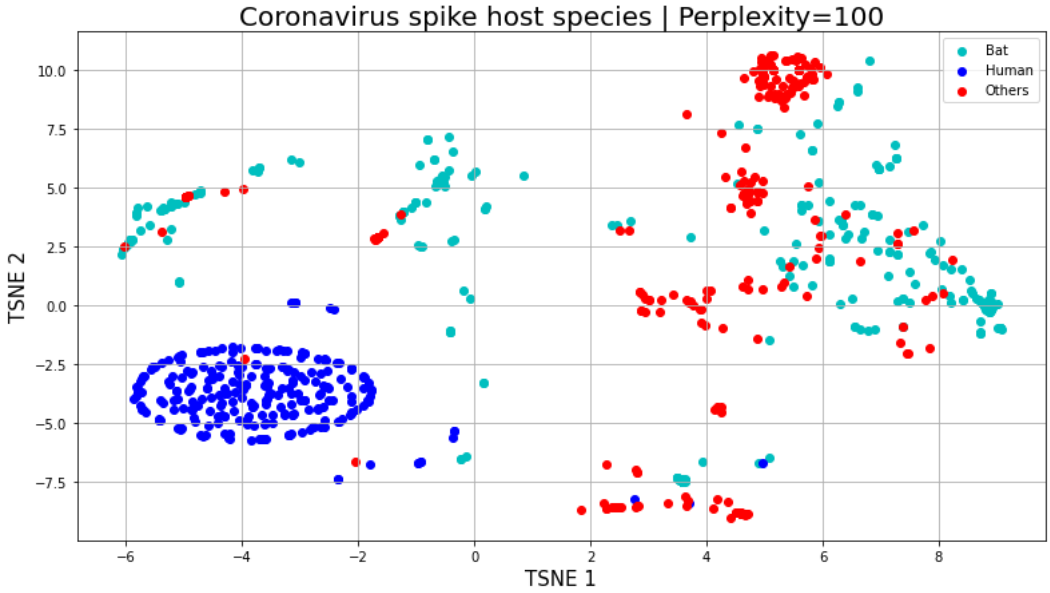

Figure 50: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species.

Figure 49 shows the result of DBScan clustering algorithm for parameters set to have the maximum value of Silhouette Score (SS=0.501) according to the trend in figure 48. In figure 50 we show again the t-SNE plot with the right labels. Lastly, we have to point out that figure 48 does not show all the values of SS for the radius ranging from 0.1 to 3 but only from 0.1 to 2.4. This is because for those removed values, $2.4 < radius < 3$, DBScan algorithm is able to find only one cluster and so it can not compute a Silhouette Score value.

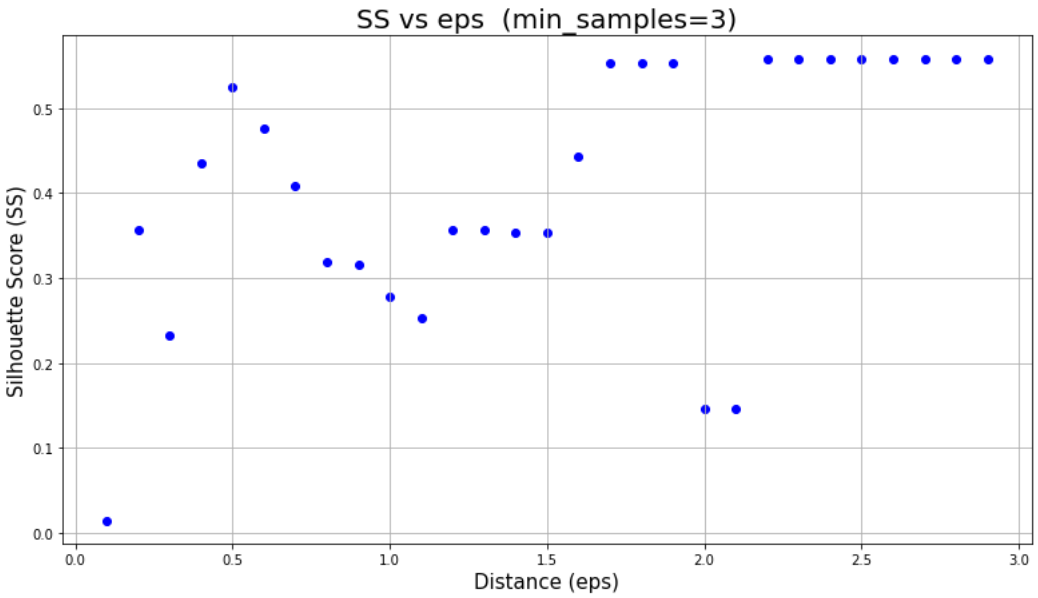

Figure 51: Silhouette Score trend as function of neighborhood distance in case of t-SNE plot (for perplexity=100) obtained with the third method.

Figure 51 shows the trend of Silhouette Score for the data obtained with the third method. Also in this case we fix the number of samples equal to 3 and we vary the value of the neighborhood radius always in the range $0 < radius < 3$. The Silhouette Score trend has different local minima and maximum, but differently from before, firstly there are not negative values of Silhoutte Score, that generally indicates that a sample has been assigned to the wrong cluster, as a different cluster is more similar and so suggesting a situation of more overlapping clustering and disordered situation; secondly in this case we find all the values that we expected because there are no situations in which the algorithm falls into seeing only a unique cluster. Also this last point is a sign of better situation from the point of view of "clusterizability". In this case, instead, what we find from a certain value on $(2.1 < radius < 3)$ is a flat region, that is the SS value keeps its value

constant and this value corresponds to the maximum Silhouette Score among all the values computed.

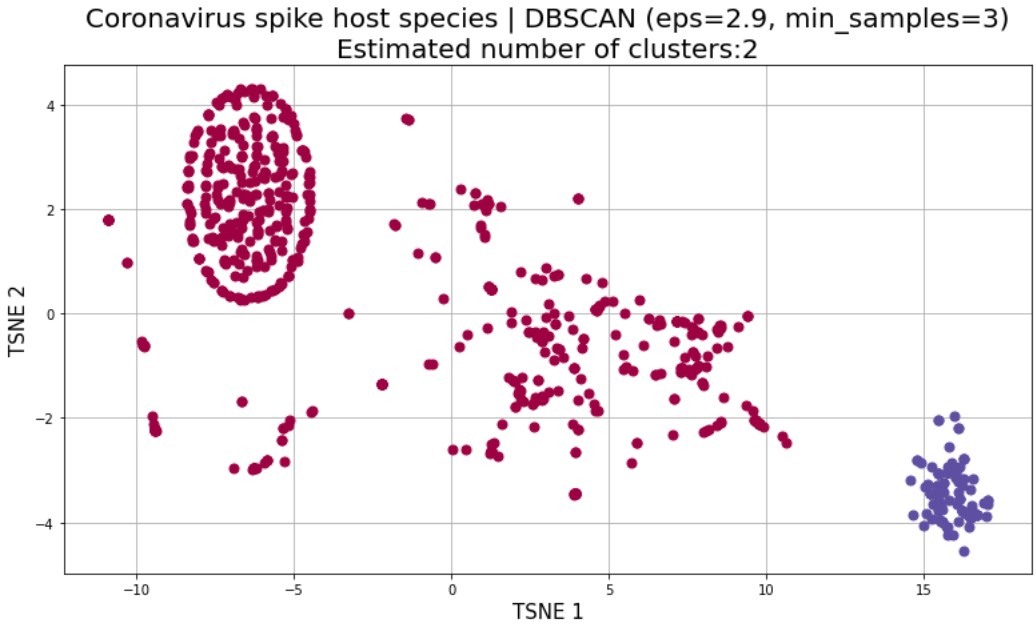

Figure 52: t-SNE plot (perplexity=100) with data obtained with the third method and labeled according to DBScan clustering algorithm, with number of samples=3, radius=0.5.

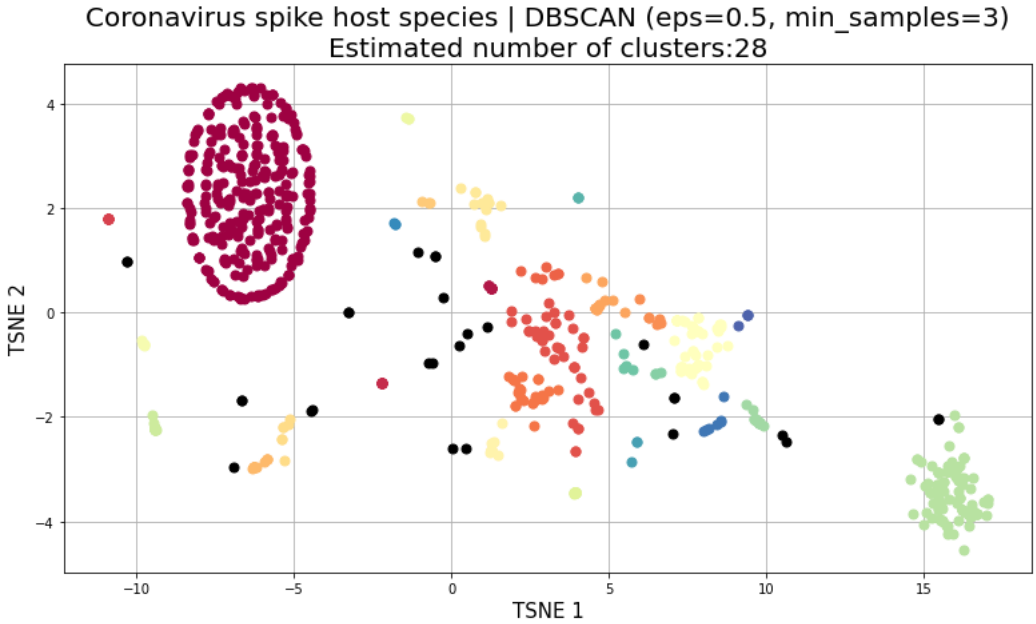

Figure 53: t-SNE plot (perplexity=100) with data obtained with the third method and labeled according to DBScan clustering algorithm, with number of samples=3, radius=2.9. The black points indicate the estimated noise points.

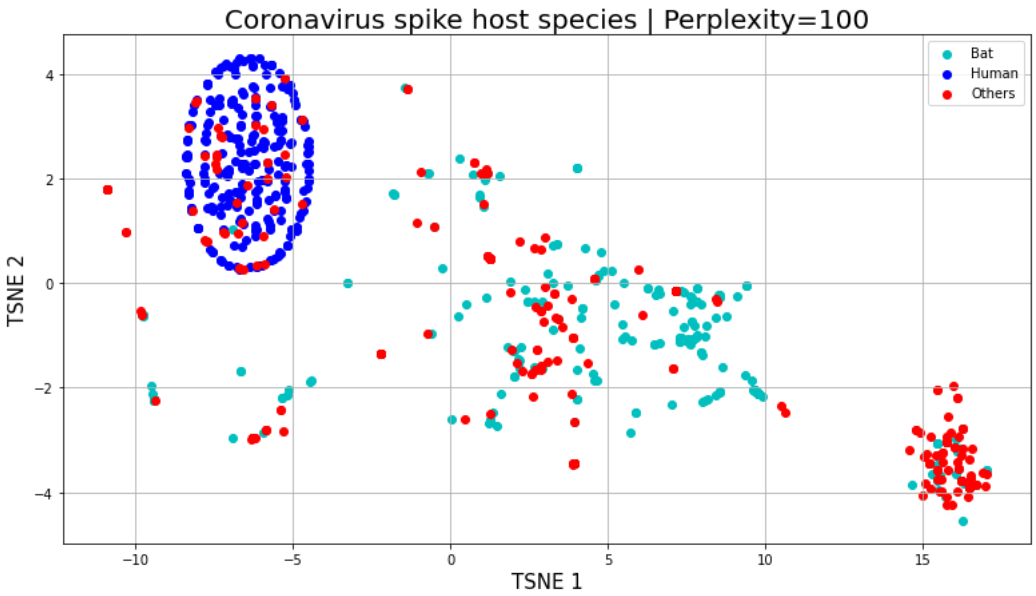

Figure 54: t-SNE plot visualizing Spike protein sequences with the first two t-SNE components as the axes of the plot and with perplexity parameter=100 in case of balanced data configuration, i.e. 260 human host species, 260 bat host species and 260 other animals host species but using the third method.

Figure 52 shows the result of DBScan algorithm using the neighborhood radius that gives the maximum value of Silhouette Score among all the computed values (that is SS=0.56 for radius=2.9). Using this it detects only two clusters that is not really truthfull if we consider the real labels that we show again in figure 54. Setting instead the radius to a local maxima (SS=0.53 for radius=0.5), the DBScan labels are more close to the reality, with two main clusters (human ellipse detected with dark red labels and other animals cluster on the down right hand corner detected with green labels) and many other small islands of clusters in the middle. Because of the decreasing of the radius, the algorithm has detected also 20 data points as noise that are shown in black colour.

4 Conclusions

Galileo once observed that the book of nature is written in mathematical characters. Proteins could be written in words of which we can find a numerical embedding.

In this project, we have tried a numerical embedding provided by machine

learning methods and the first understanding of this work is that studying the language of life, with such tools, is a process full of choices, modelling and trials. It could be an infinite project but we summarize our main findings.

In the framework in which we consider a protein as if it was a sentence of natural human language and its amino acids as words, i.e. tokens, and we average the amino acids embeddings (provided by an embedding layer of a LSTM neural network trained to classify protein sequences according to their host species) to represent sequences, we have tried three methods to deal with the imbalanced classes problem present in the Spike protein sequences host species categories: training the neural network on imbalanced data, training the neural network on balanced data (simple undersampling method) and, finally, a more "creative" method that trains the neural network classifier sequentially on a N-th subset of the majority classes but on all of the data from the rare class. In these approaches, while PCA method always fails in providing an informative plot with its two principal components, t-SNE and UMAP plots always show that protein sequences coming from the same animals tend to be more similar within each other. So this is a validation that this embedding is meaningful at least in terms of animal host species. In particular, human protein sequences always arrange in a very regular shape while animals data are always more irregularly scattered in the space. But among the three mentioned methods used, the "winner" is the more creative third method, in which really all human protein sequences are gathered in a regular shape while in the other methods some human host species mix with the animals sparse clouds; moreover, in this third method, all the categories make clearer clusters as perceived by our human brain and as assessed by the DBScan clustering algorithm that in the t-SNE plot (with perplexity=100) of the undersampling method tends to find just a unique cluster while this does not happen for the same t-SNE plot of the third method, suggesting a noisier situation and less defined clusters in the simple undersampling method plot.

One other parameter that has been exploited is the length of the sequence. Undersampling method has the drawback to cut out some data but, having fewer data, it has also the advantage that we can train the model on longer sequences without increasing too much the computational time. Indeed with this configuration we finally found that the red data points, that overlap on the blue human main cluster in all the three methods that consider 350 AAs as maximum input length, do not overlap anymore with the human elipse cluster.

Hence, from a biological point of view, this means that important differences

between human sequences and animals are after the first 350 AAs. But we have been able to show also that some bats Spike protein sequences are particularly similar to humans and that human Spike protein sequences vary following a more particular regularity with respect to the sequences from animal host species. Thus this "series of embedding methods", each one by trying to minimize an energy function, let us to transform sequences of amino acids symbols in numerical vectors that embed as much as possible information about those that originally were sequences of strings and, in fact, in this little project made without the use of modern GPUs, such approach has already revealed interesting characteristics of the novel Spike protein sequences.

Appendix A

Quality of classification in supervised analysis

We need to quantify the goodness of our model as classifier in supervised analysis. Supervised analysis is based on *error minimization* because we know in advance the correct answer and so the aim is to minimize the difference between the model outcome and the true outcome. Typically metrics to keep track of during the model training process, *in order to judge* the performance of the model, is the *accuracy*. This has not to be confused with the loss function that is *minimized by the model* using algorithms such as *Adam* and that is used to update the internal parameters, as mentioned in section 1.1.1. In Keras in Python, for example, metric is the model performance parameter that one can see while the model is judging itself on the validation set after each epoch of training. The accuracy is the ratio between all correct classifications vs all data:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

where $TP = TruePositives$, $TF = TrueNegatives$, $FP = FalsePositives$ and, finally, $FN = FalseNegatives$. This is not the best metrics for all the situations such as in case of imbalanced classes (in a binary classification problem, if the method correctly classifies only the majority class it will have a high accuracy although it is not a good classifier). There are metrics that are more suitable to assess the performance of the model, such as the Confusion Matrix (i.e. a 2x2 matrix made by total number of TPs, TNs in the diagonal and the elements off diagonal that are FPs and FNs) if the task is classification.

In any case, minimizing an error is not necessarily the best thing to do but care must be taken also to not overfit the data, that is *the model learns the dataset specificities and not a generalized learning of data structure*. We run this risk, for example, when there are few data and a too complex model to analyse them. Thus analysis methods must be evaluated also for their *ability to describe data*.

Thus, using all data for training is not a good estimate of the generalization performance because it could have undergone overfitting. If we want to build a solid model we have to follow the specific protocol of splitting data into two sets: one for **training** and one for **test**.

During each epoch, the model will be trained on samples in the training

set (this is the only dataset on which the weights are updated during back-propagation) but will not be trained on samples in the test set. Instead, the model will only be evaluating itself on each sample in the test set. Validation set is only used for tuning hyper-parameters to change the way of training to make the model eligible for working well on unseen data during training and no back-propagation occurs on test set and hence no direct learning from it. The purpose of doing this for us is to be able to judge how well our model can generalize, that is how well our model is able to predict on data that are not seen while being trained. Hence, having a validation set provides great insight into whether your model is overfitting or not. This can be interpreted by comparing the accuracy and loss from the training samples with the validation accuracy and validation loss from the validation samples. For example, if the accuracy is high, but the validation accuracy is lagging way behind, this is good indication that our model is overfitting.

This procedure of dataset splitting into train and test set should be modified when the data are few (e.g. tens/hundreds): in this case a strategy is to re-use the data both for train and test and for such reason this method is called **K-fold cross-validation**. K-fold cross validation divides the dataset into K parts: K-1 parts are used for training and the remaining part for the test. By looping throughout all the folds, in each single realization there is a non-overlapping train and test set but in the end all the data are used both for training and testing. This method is much more powerful than a complete hold-out.

Appendix B

Quality of clustering in unsupervised analysis

We need to quantify the goodness of clustering and we choose the number of clusters that maximize this goodness. In clustering, generally the aims are to have a good separation between clusters, a high level of homogeneity within the cluster (underlying "true" classes, such as biological species, may cause homogeneous distributional shapes) and rapresentative centroid of the whole cluster. For stochastic algorithms such as K-Means and DBSCAN one should also check if the results are stable when applying the algorithm different times with different random initial points but also with different parameters so that we can talk about robust results. If the clusters are very clearly defined, the results will not vary (we will always obtain the same labels); if instead the situation is very unstable the point for one cluster could differ significantly.

The points that are at the boundaries will tend to be assigned each time to a different cluster while the core samples will be those that will have always the same label.

A typical measure with which we can quantify the goodness of clustering is the *Silhouette Score* (SS). It can be calculated with the following formula for the *i-th* cluster:

$$SS_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{11}$$

For 2 classes: the SS is the difference between the *between* cluster distance b_i and the *within* cluster distance a_i (for N classes it is applied one-versus-closest cluster) . The between cluster distance is the *minimum* distance between samples in one cluster and samples in the other; the within cluster distance is the average distance between all the elements of the cluster. The ratio between the distance $b_i - a_i$ and the maximum between the two values allows that $-1 < \text{SS} < +1$. When $a_i = 0$, $\text{SS} = 1$, thus the more SS is close to 1, $\text{SS} \sim 1$, the better is the clustering.

However, quality measure can rely also on simply the separation between clusters measuring the *minimum distance between clusters* or the *distance between centroids* exploiting respectively the properties at the boundaries of the clusters or the global picture of the clusters.

References

[1] D. Remondini, G. Castellani, F. Durazzi, N. Curti, teaching material of *Pattern Recognition* course, University of Bologna.

[2] Jason Brownlee PhD, Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras

[3] Jason Brownlee PhD, What is Deep Learning ?

[4] Jason Brownlee PhD, Difference Between a Batch and an Epoch in a Neural Network.

[5] Jason Brownlee PhD, An Introduction To Recurrent Neural Networks And The Math That Powers Them.

[6] Jason Brownlee PhD, How to Prepare Text Data for Deep Learning with Keras

[7] Jason Brownlee PhD, How to Use Word Embedding Layers for Deep Learning with Keras

[8] S surface glycoprotein [Severe acute respiratory syndrome coronavirus 2] url: https://www.ncbi.nlm.nih.gov/gene/43740568

[9] Spike glycoprotein https://www.uniprot.org/uniprot/P0DTC2

[10] Why are neural networks described as black-box models? https://stats.stackexchange.com

[11] What is the preferred ratio between the vocabulary size and embedding dimension? https://stackoverflow.com

[12] How to use embedding layer and other feature columns together in a network using Keras? https://mmuratarat.github.io

[13] Introduction to Softmax for Neural Network. https://www.analyticsvidhya.com

[14] How to choose an activation function for the hidden layers? https://ai.stackexchange.com

[15] How to process bio-sequences for use in Data Science. https://towardsdatascience.com

[16] Neural Network Embeddings Explained. https://towardsdatascience.com

[17] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin.

[18] Intro to optimization in deep learning: Momentum, RMSProp and Adam https://blog.paperspace.com

[19] Dan Ofer, Nadav Brandes, Michal Linial, *The language of proteins: NLP, machine learning & protein sequences*.

[20] Recurrent Neural Networks by Example in Python. https://towardsdatascience.com

[21] What does Keras Tokenizer method exactly do? https://stackoverflow.com

[22] How does Keras 'Embedding' layer work?
https://stats.stackexchange.com

[23] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes and Burkhard Rost, *Modeling aspects of the language of life through transfer-learning protein sequences.*

[24] An Essential Guide to Pretrained Word Embeddings for NLP Practitioners. https://www.analyticsvidhya.com

[25] Pangolins https://www.nationalgeographic.com

[26] Justin M. Johnson and Taghi M. Khoshgoftaar, *Survey on deep learning with class imbalance.*

[27] R. Anand, K.G. Mehrotra, C.K. Mohan and S. Ranka, *An improved algorithm for neural network classification of imbalanced training sets.*

[28] Choosing the right Hyperparameters for a simple LSTM using Keras. https://towardsdatascience.com

[29] 2.3.7 DBSCAN https://scikit-learn.org

[30] Brian Hie, Ellen D. Zhong, Bonnie Berger and Bryan Bryson, *Learning the language of viral evolution and escape.*

[31] Deep Protein Sequence family Classification https://www.kaggle.com

[32] Peer Borkab, Eugene VKooninc, *Protein sequence motifs*, Current Opinion in Structural Biology, Volume 6, Issue 3, June 1996, Pages 366-37.

[33] t-SNE: The effect of various perplexity values on the shape https://scikit-learn.org

[34] Ian T. Jolliffe and Jorge Cadima, *Principal component analysis: a review and recent developments.*

[35] Basic UMAP Parameters https://umap-learn.readthedocs.io

[36] Chat bot `https://it.wikipedia.org/wiki/Chat_bot`

[37] Illustrated Guide to Recurrent Neural Networks: Understanding the Intuition https://www.youtube.com

[38] Word embeddings https://www.tensorflow.org

[39] Peter David Turney and Patrick Pantel, *From Frequency to Meaning: Vector Space Models of Semantics.*

[40] 18 Word embeddings https://ambarishg.github.io

[41] Synaptic Strength https://www.sciencedirect.com