University of New Hampshire Graduate School

Data Science and Analytics

Data 897: Self-Designed Analytics Lab

Jan Michael Austria

Prof. Prashant Mittal

Title: Classification of Protein Class from Protein Sequence and Sequence Generation of Viral Proteins

May 4, 2020

Introduction

Proteins are large biomolecules, or macromolecules, consisting of one or more long chains of amino acid residues. Proteins perform a vast array of functions within organisms, including catalysing metabolic reactions, DNA replication, responding to stimuli, providing structure to cells, and organisms, and transporting molecules from one location to another. Proteins differ from one another primarily in their sequence of amino acids, which is dictated by the nucleotide sequence of their genes, and which usually results in protein folding into a specific 3D structure that determines its activity.[1]

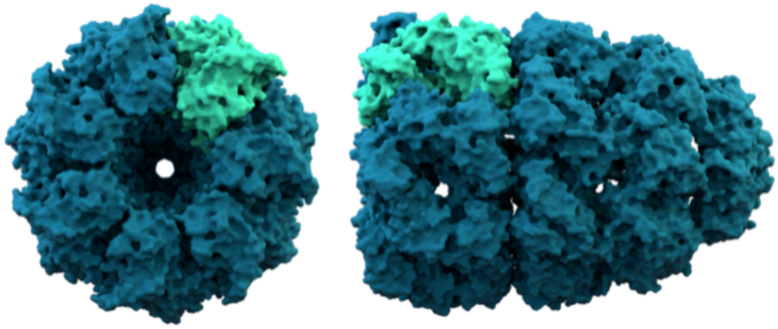


Figure 1. The crystal structure of the chaperonin, a huge protein complex. A single protein subunit is highlighted. Chaperonins assist protein folding.

A linear chain of a protein is composed of amino acids residues and is called a polypeptide. A protein contains at least one long polypeptide. Short polypeptides, containing less than 20–30 residues, are rarely considered to be proteins and are commonly called peptides, or sometimes oligopeptides. The individual amino acid residues are bonded together by peptide bonds and adjacent amino acid residues.[1]

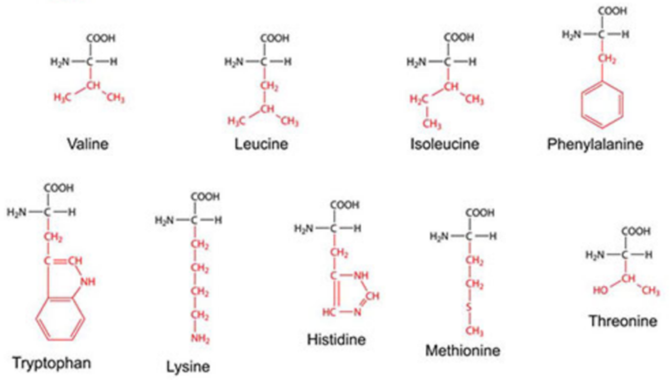


Figure 2. The chemical structure of the essential amino acids

The sequence of amino acid residues in a protein is defined by the sequence of a gene, which is encoded in the genetic code. In general, the genetic code specifies 20 standard amino acids; but in certain organisms the genetic code can include selenocysteine and—in certain archaea—pyrrolysine. Shortly after or even during synthesis, the residues in a protein are often chemically modified by post-translational modification, which alters the physical and chemical properties, folding, stability, activity, and ultimately, the function of the proteins. Some proteins have non-peptide groups attached, which can be called prosthetic groups or cofactors.

Proteins can also work together to achieve a particular function, and they often associate to form stable protein complexes.[1]

A	Alanine	M	Methionine
C	Cisteine	N	Asparagine
D	Aspartic Acid	P	Proline
E	Glutamic Acid	Q	Glutamine
F	Phenylalanine	R	Arginine
G	Glycine	S	Serine
H	Histidine	T	Threonine
I	Isoleucine	V	Valine
K	Lysine	W	Tryptophan
L	Leucine	Y	Tyrosine

Figure 3. The abbreviations for the 20 most common amino acids. There are more, but for the most part most proteins are made up of these ones.

Once formed, proteins only exist for a certain period and are then degraded and recycled by the cell's machinery through the process of protein turnover. A protein's lifespan is measured in terms of its half-life and covers a wide range. They can exist for minutes or years with an average lifespan of 1–2 days in mammalian cells. Abnormal or misfolded proteins are degraded more rapidly either due to being targeted for destruction or due to being unstable. Proteins also have structural or mechanical functions, such as actin and myosin in muscle and the proteins in the cytoskeleton, which form a system of scaffolding that maintains cell shape. Other proteins are important in cell signaling, immune responses, cell adhesion, and the cell cycle. In animals, proteins are needed in the diet to provide the essential amino acids that cannot be synthesized. Digestion breaks the proteins down for use in the metabolism.[1]

A succinct ‘cartoon’ portraying the amino acid sequence and its protein structure is shown below:

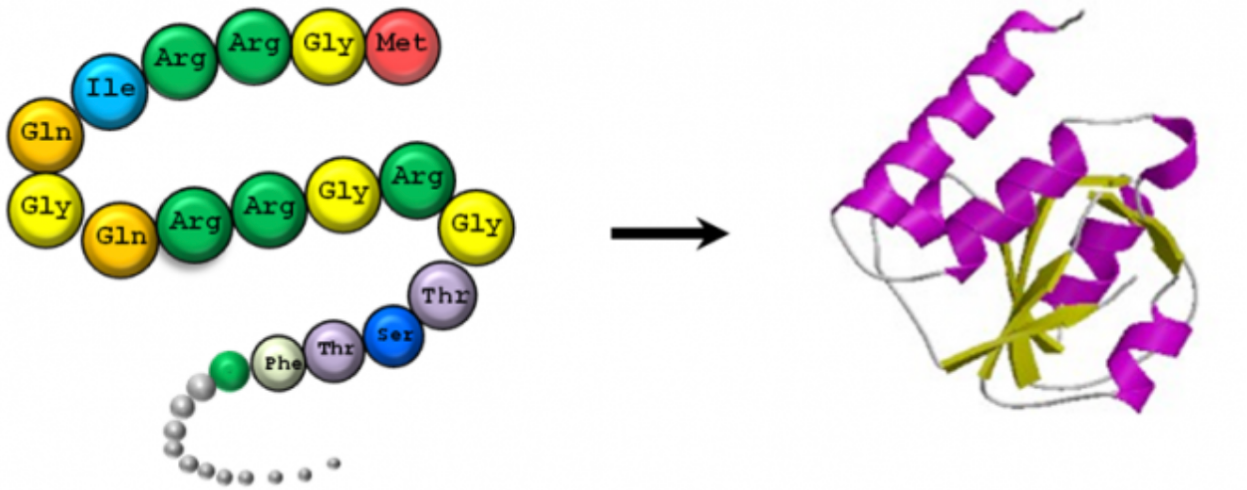


Figure 4. Proteins consist of one or more polypeptides. A polypeptide is a chain of amino acids. The polypeptide chains fold into their final three-dimensional structure to constitute a functional protein.

t is important to note, that the sequence of amino acids plays a role in the folding and formation of a protein. For example, the sequence A-C-D-E-F (using the abbreviation nomenclature) is very different from the reverse sequence F-E-D-C-A. Although, the content in both sequences remains the same, they may fold differently within the cell cytoplasm or may function differently.

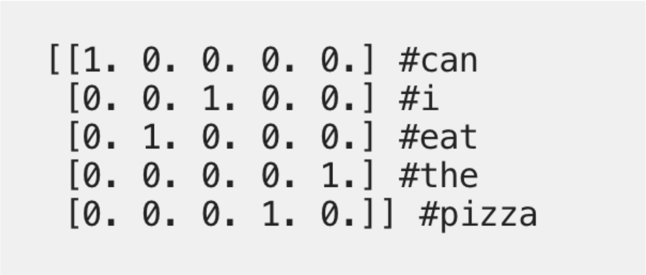
In this interpretation of a protein, we can ascribe certain methodologies from natural language processing. We can begin to ask questions like: is it possible to classify proteins based on sequence only? Can we predict the next amino acid given the history of amino acids before it? Another even more exciting question we can ask is if we can create novel proteins based on the sequences of similar proteins? And if so, how similar are the new proteins are to real ones.

A Brief Introduction into Natural Language Processing for Machine Learning Methods

This section will serve as a preliminary for seeing how language models can be applied to the amino acid representation of proteins. A more in-depth review of such methods has been left to the reader.

Given the sentence, "The dog jumped over the", we can create an n-gram model of size three and step size one, to get the following n-grams "The dog jumped", "dog jumped over", "jumped over the". Each of these vectors can be one hot encoded and turned into a matrix (or left as vector) with dimensions 'length of the window' by 'length of the vocabulary'. The length of the vocabulary is usually the length of all unique tokens in each n-gram sample. The i,j position in the matrix would be filled with a zero or one, where a one indicates the position index of the token in the dictionary. For example, if the word 'dog' was the second word in the vocabulary, then the row vector in the matrix for the first n-gram sample would be [0,1,0,0].

In traditional NLP (natural language processing) methods, a one hot encoded n-gram vector could be used to calculate the counts of each word in a sentence (or smallest denominational representation i.e. sentence, paragraph, etc.). The counts would then be used to calculate the proportion of tokens present across all n-gram samples. The count matrix, affectionately called the document term matrix or term frequency matrix in machine learning literature, can also be used to calculate what is called the inverse document frequency matrix. A longer description of such matrices is left to the reader, but such methodologies were not implemented in the analysis of proteins. Below is a snippet explaining one hot representation.



```
[[1. 0. 0. 0. 0.] #can
 [0. 0. 1. 0. 0.] #i
 [0. 1. 0. 0. 0.] #eat
 [0. 0. 0. 0. 1.] #the
 [0. 0. 0. 1. 0.]] #pizza
```

Figure 5. One hot representation using the sentence "can eat i pizza the". Note that the order does not make sense lexically, but the content is still preserved.

From this NLP approach, we can think of proteins as being made up of a bunch of amino acids, where we can treat each amino acid as word, and the length of the row vector would be equal to the number of all unique amino acids present, and a 1 in the position indicating the index corresponding to the amino acid. This allows us to use proteins in various language models creating classification and sequence predictions. More details of how protein sequence was transformed will be explained later on in the paper, but for now an introduction illustrates how one can use a language model. Let's see if we can learn the language of proteins!

Protein Dataset

The data used for analysis in this study came from the Protein Data Bank archive. Not all proteins from the data bank were used in this study. A total of 346,3245 protein chains were randomly pulled in bulk from their query generator, https://www.rcsb.org/. The data is publicly available and typically used by researchers in bioinformatics and computational biology. In addition, they also have DNA and RNA sequences for all proteins for which there is an already defined amino acid sequence. The website also had the ability to show the secondary, tertiary, and quaternary structures for all proteins that have been sequenced.

Protein Dataset – Exploratory Data Analysis

Before going into depth on some of the experiments performed with proteins. A thorough EDA (exploratory data analysis) of that data was performed. The following below are snippets using python's pandas package to observe the column names and amount of missing values.

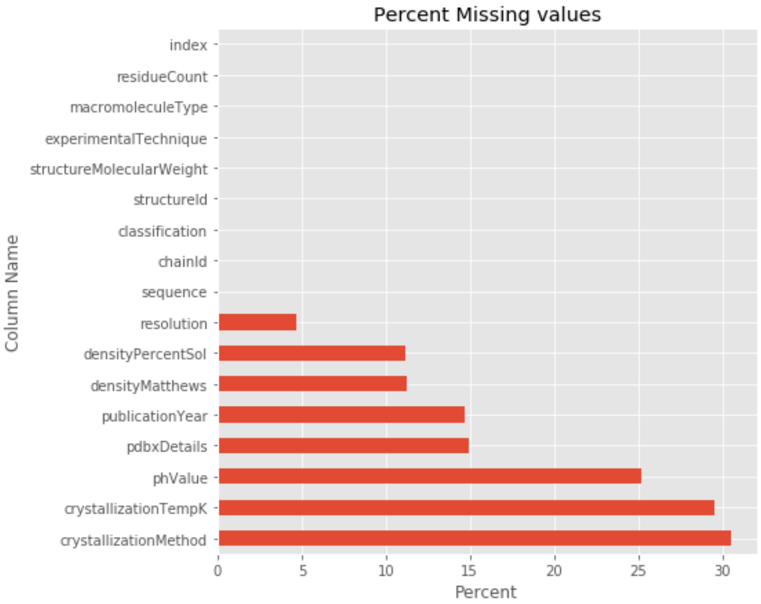


Figure 6. Missing values in the protein dataset. The columns 'sequence' and 'residueCount' have no missing values. These will be the only two columns needed for the analysis in this experiment. For the classification models, the target label will be 'classification' column. This has no missing values, which means all proteins can be used! The other columns 'resolution', 'density', and 'crystallization' are other experimental properties of biomolecules. Although these contain information unique to biomolecules, they will not be used for the remaining parts of the experiments.

We also looked at the number of unique protein classes in the dataset. The first part of the project involves creating a classification model to predict the type of protein from its sequence. If there are a small number of classes, for examples 10 to 30 classes, then computation times would not be much of an issue. However, in the case that there are thousands of classes, then computation times would become more of an issue. In the protein dataset, there are a total of 4468 unique protein classes! The top 5 classes of proteins with the number of unique elements in the dataset are hydrolases, transferases, oxidoreductases, immune system proteins, and lyases. With counts respectively being 46336, 36424, 34322, 15615, and 11682. The image below shows the counts for the top 25 classes of proteins.

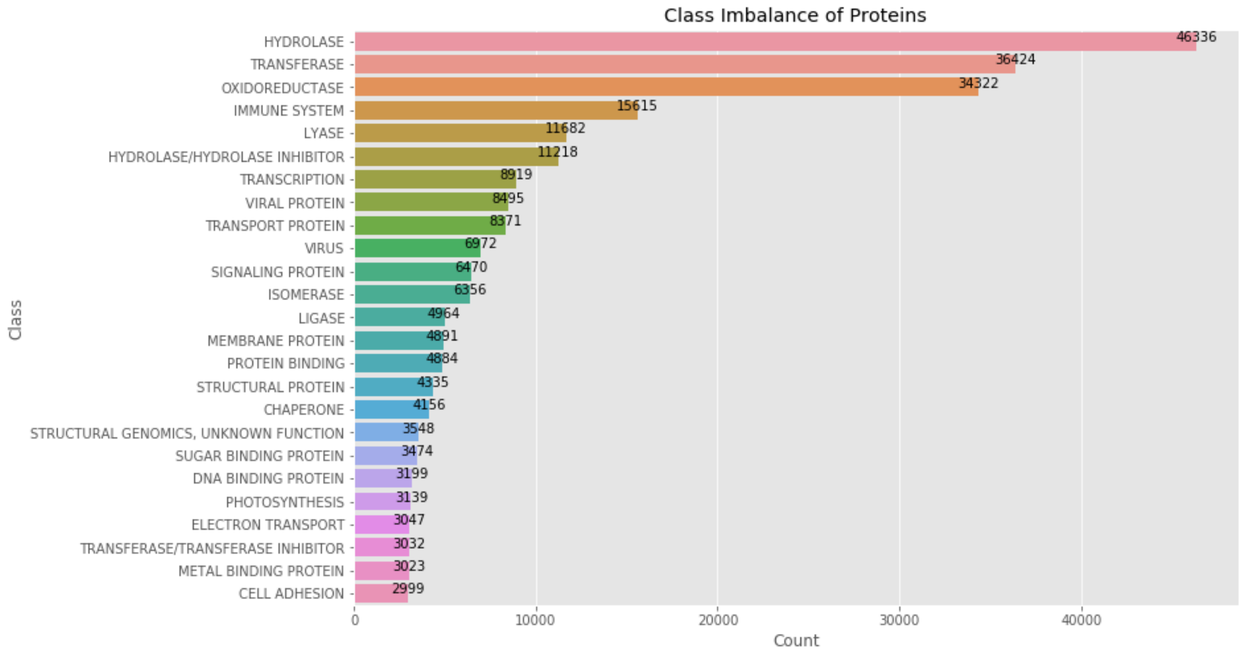


Figure 7. Class Imbalance of top 25 proteins in the dataset

In the dataset, we also had visibility of the year a protein was sequenced. We wanted to see the number of proteins sequenced as a function of the year. The plot is insightful and shows the clear exponential growth of the number of proteins sequenced from 1970 to now due the rise in computing power available to researchers.

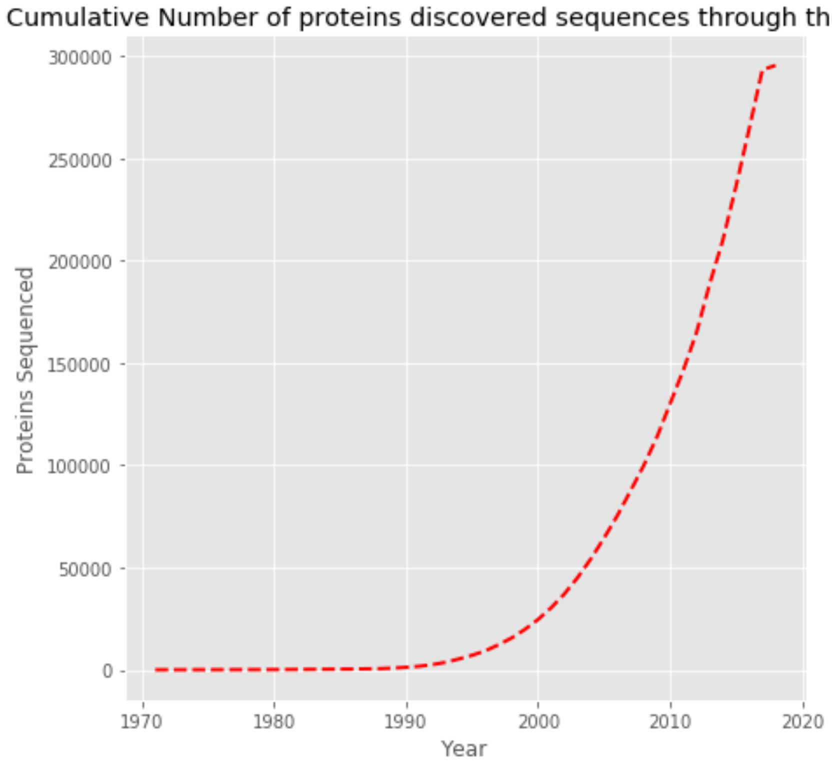


Figure 8. The number of proteins sequenced throughout the years.

We wanted to see the distribution of sequence length for all protein chains in the dataset. Most of the protein sequence lengths are between 0 and 1000 amino acids along, with a mean length of 266 amino acids and a median sequence length of 231 amino acids. There are a few outliers whose longest protein chains are in the range of 300,000 to 1,500,000 amino acids long. Interestingly enough, the proteins with the longest chains belong to a family of viruses relating to the coronaviruses! An interesting thought would be to think of protein complexity and function as a function of sequence length (i.e, proteins with longer chains are more complex). The fact that viruses have longer amino acid sequences and are some of the most complex and largest of proteins gives merit to this assumption. Although a further analysis would need to be completed before concluding this theory.

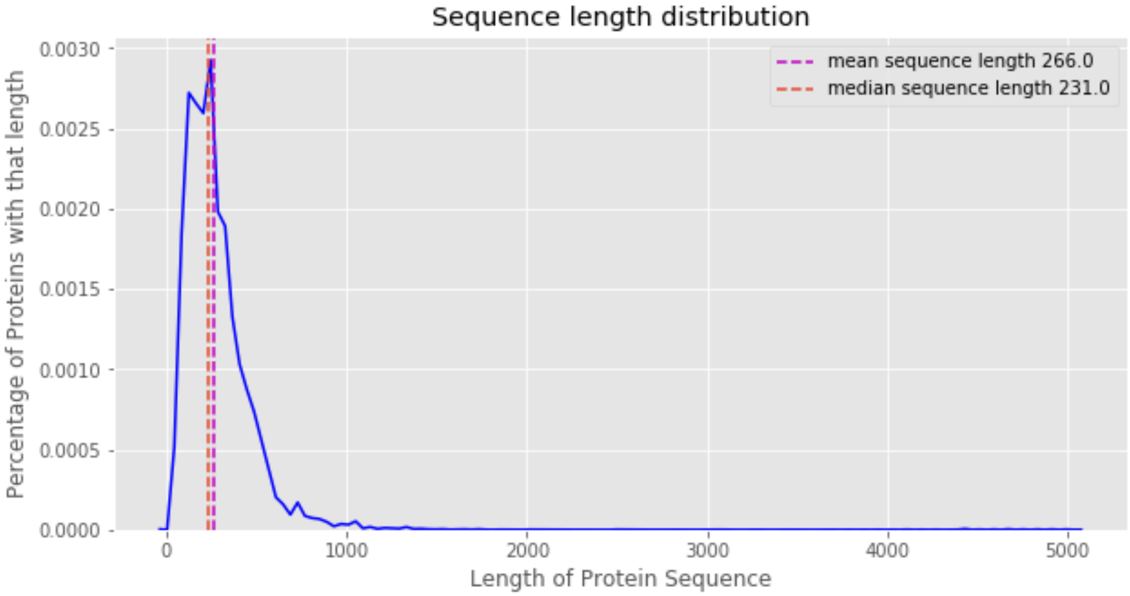


Figure 9. Distribution of protein sequence lengths for all protein chains.

We also decided to take a look at the distribution of sequence length across multiple classes. Figure 10 shows boxplots for the top 25 protein classes and the distribution of their sequence lengths on a log base 2 scale. Log base 2 was chosen because the numbers work out nicely and the scaling is not too spread across. It is interesting to see that the two protein classes with the largest average sequence length are belong to the family of viruses and photosynthesis proteins. The numbers on the right-hand side of the plot indicate the average sequence lengths across proteins.

This method, however, takes into account looking at the total number of amino acids in a single chain, and not the whole protein. In a later section, we will talk about the chain to protein structure disparity. For now, we will only bring it into light. For some of the proteins in this dataset, there are multiple chains that exists. If we examine the chains separately, that is, each protein being made up of n chains and each n'th chain is independent of the other n-1 chains. Then we can get the same distributions as in Figure 10. We can perform an ANOVA test on the different distributions to see if there is a statistically significant difference among the mean log base 2 lengths for the top 25 proteins. Running the ANOVA test, we obtain an F statistic of 3289 and a p-value of 0.0. The p-value is below the significance threshold and we can reject the null hypothesis that two or more groups have the same mean. And there is indeed a difference between the groups.

Figure 11 shows distributions across the proteins but per chain instead of per protein. Instead of using a typical box plot, we use a boxen plot to capture some of the mass within each distribution. Again, we use the log base 2 for the value of the sequence length. We can also perform another ANOVA test to see if the there is a difference in one distribution from the rest. Doing so we obtain an F statistic of 1436 and a p value of 0.0. Suggesting that there is a significant difference in two or more of the distributions from the rest.

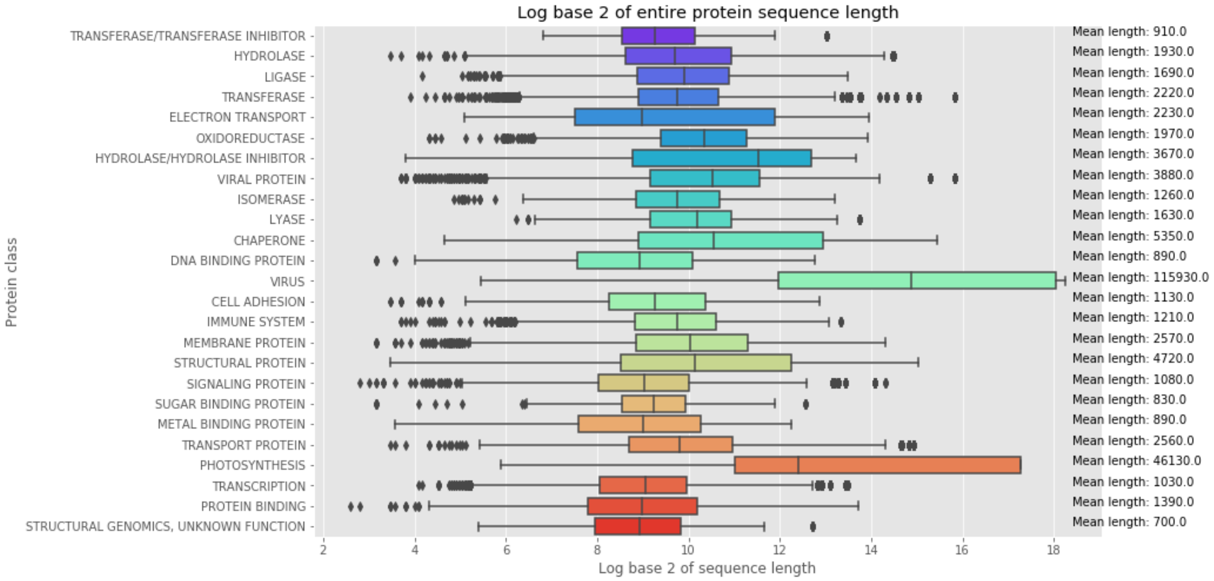


Figure 10. Log base 2 distributions of sequence length across the top 25 proteins by count.

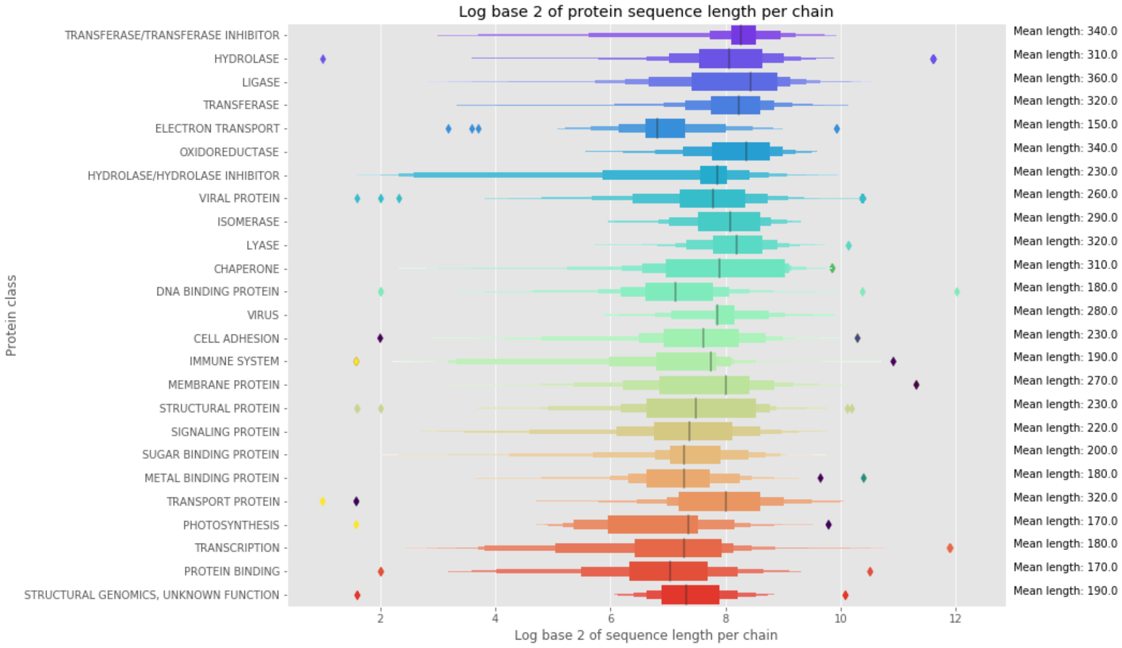


Figure 11. Log base 2 of protein sequence length per chain across all top 25 proteins by count.

Protein Dataset – Amino Acid Counts

For this part of the experiment we were interested in seeing the percentage of amino acid content across all proteins and within a certain class of proteins. We were able to obtain the raw counts of each amino acid for all proteins in the data set, as well as the proportions of each amino acids.

In Figure 12, we see that the most common amino acid by counts are Leucine, Alanine, Glycine, Valine, and Glutamic acid. The bottom few amino acids by count include Cysteine, Tryptophan, Selenocysteine, Glutamic Acid, and Pyrrolysine. It is interesting to note that there are only a few amino acids that have either Selenocysteine, Glutamic Acid, and Pyrrolysine. In the biological literature, these amino acids have already been identified as being rare amino acids.

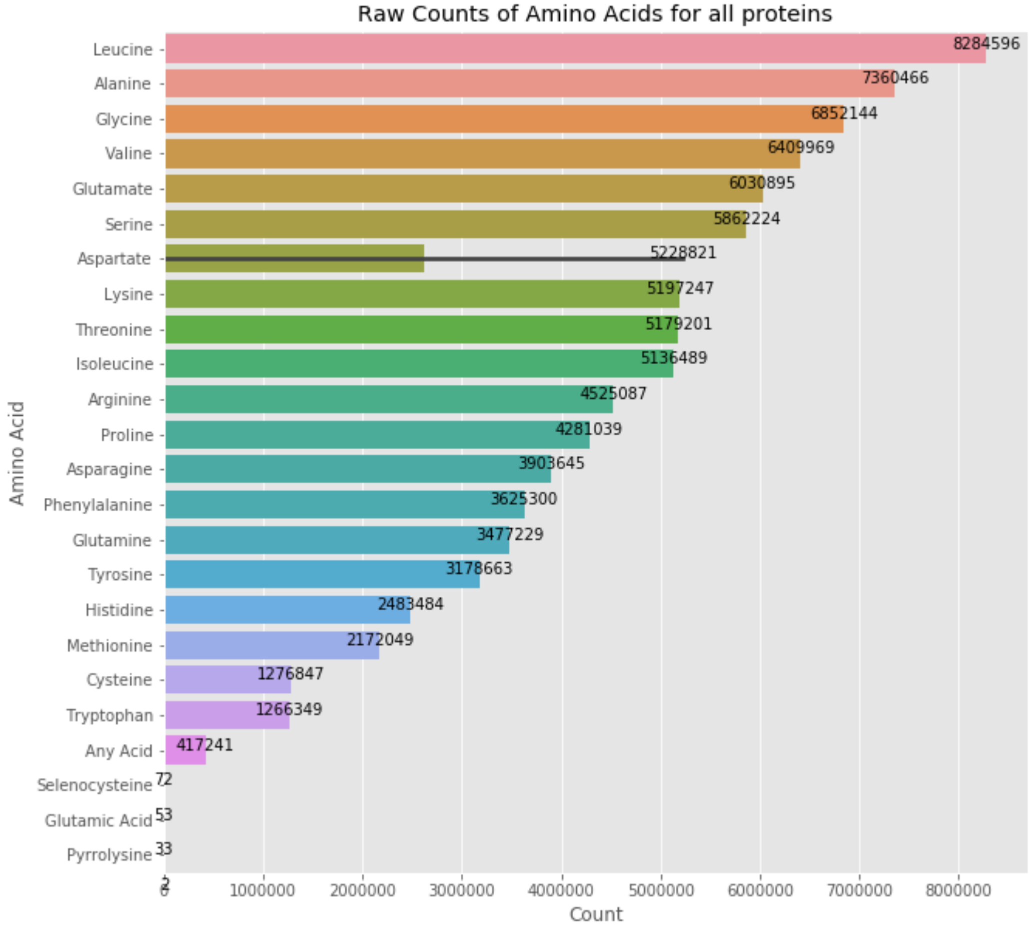


Figure 12. Raw counts of amino acids for proteins in dataset

In Figure 13, we show the percentage of amino acids across all proteins in the dataset. Or more precisely:

$$\frac{p_i}{\sum_i^N p_i}$$

Where p_i is the i'th amino acid. We do this for all amino acids in each unique protein and obtain the following percentages.

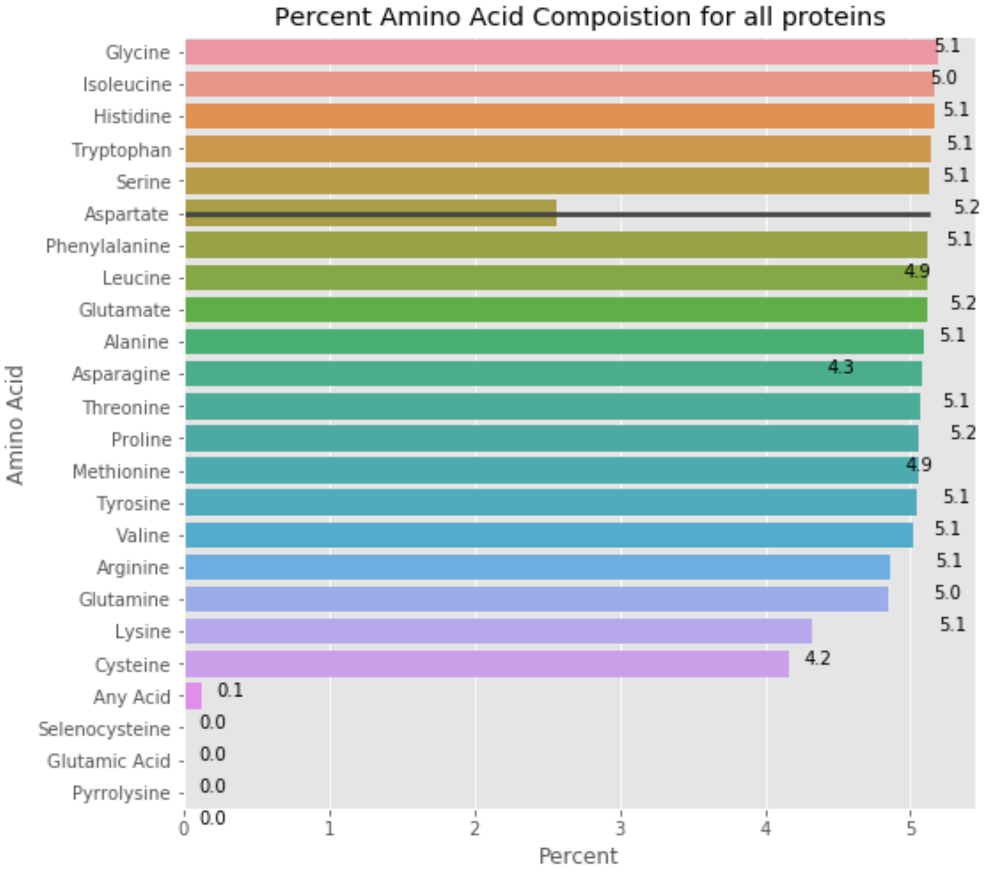
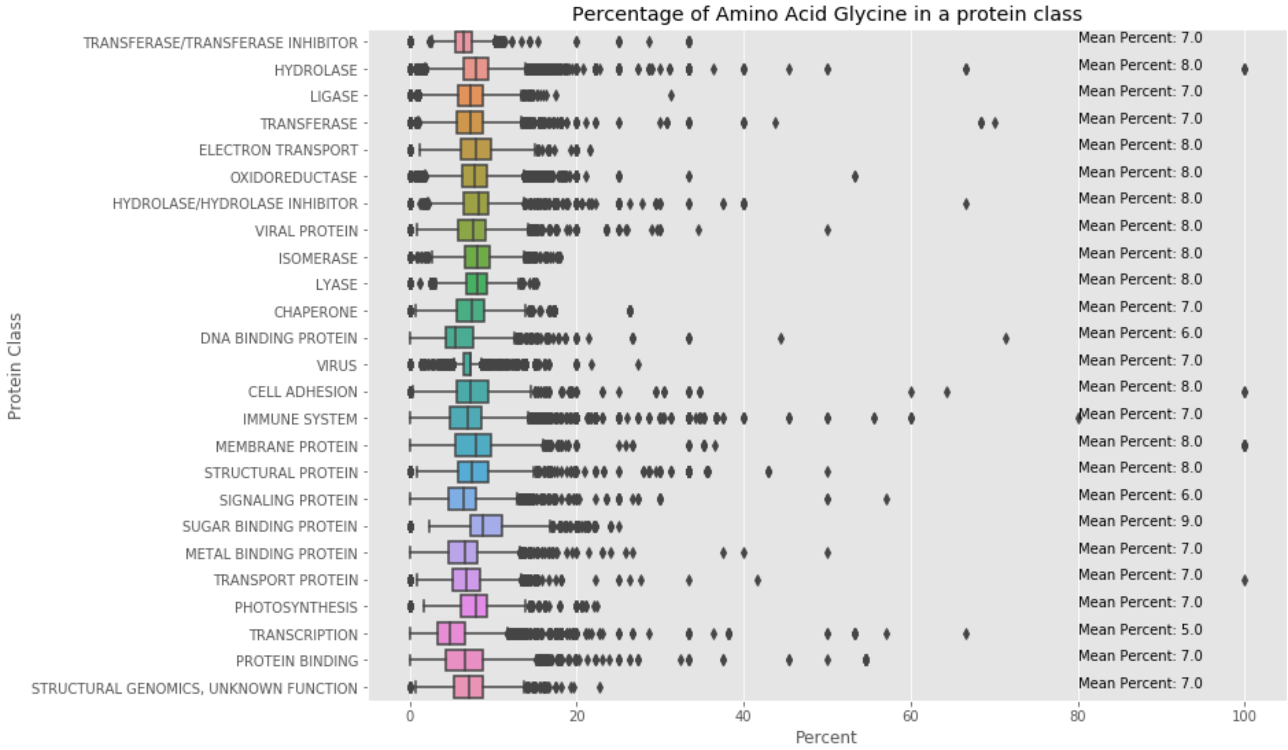


Figure 13. Percentage of Amino Acids across all proteins.

We also wanted to see if the percent compositions of amino acids were different across some of the protein classes. An exhaustive list of all amino acids was used, but for brevity we only include the percentages of the amino acid Glycine across the top 25 proteins. We can also perform a chi-sqaured test of independence. We assume that the distribution of amino acids in a protein is independent of the protein class, and given that there are 27 unique values of amino acids in the dataset, we would expect that the proportions of amino acids in each protein should be about 1/27 or 0.037 times the length of the protein chain. We set up the two-way contingency table and perform a chi-squared test (see attached code) and obtain a chi squared test statistic of 96045231, with 9004320 degrees of freedom. The p-value in this test is driven down to nearly zero, indicating that the proportion of amino acids within a protein chain is not independent. Note, the high chi-squared test statistic is most likely due to the large number of degrees of freedom in the sample size! There are more than 300000 different proteins!

Figure 14 also shows that there is not much of a difference in the percentage of Glycine in the chains of proteins. The range of mean percent of Glycine for the top 25 proteins by count is between 6%-9% and looking at the distributions it would seems as they do not appear too different across these proteins.

Figure 14. Percent Glycine across top 25 proteins by count.



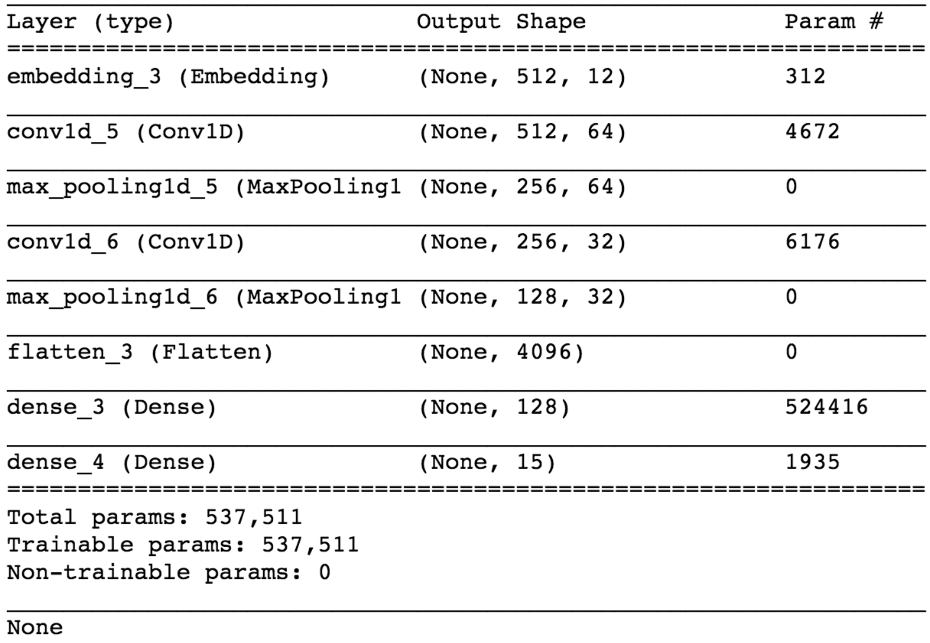
Classification Modeling - Setup

In order to use the sequence data, we need to transform the sequences into a form that a deep learning model would like. Also, to help with computation time we limit the choice of classes to the top 15 protein classes. For the classification modeling part of the experiment we include only these classes of proteins: Ribosome, Hydrolase, Transferase, Oxidoreductase, Immune System, Lyase, Hydrolase Inhibitor, Viruses, Transcription Proteins, Viral Proteins, Transport Proteins, Isomerase, Signaling Protein, Ribosome/Antibiotic, and Ligase. These will be our target predictions of our Y vector. To make these usable for a deep learning model, we one hot-encode these target labels. For example, if a sample is of the protein class 'Ribosome' and 'Ribosome' is the first class, then the vector would be [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0].

For the sequence data, we encode each amino acid with a unique integer from 1 to the total number of amino acids. Due to the length of some amino acid chains, we chose to truncate after a certain number of sequences - 512 in this case. For any protein chain less than 512 amino acids, we pad the sequence with zeros. This is to make sure that each amino acid chain from a protein has equal length when used as input in our classification model

Classification Modeling – First Model

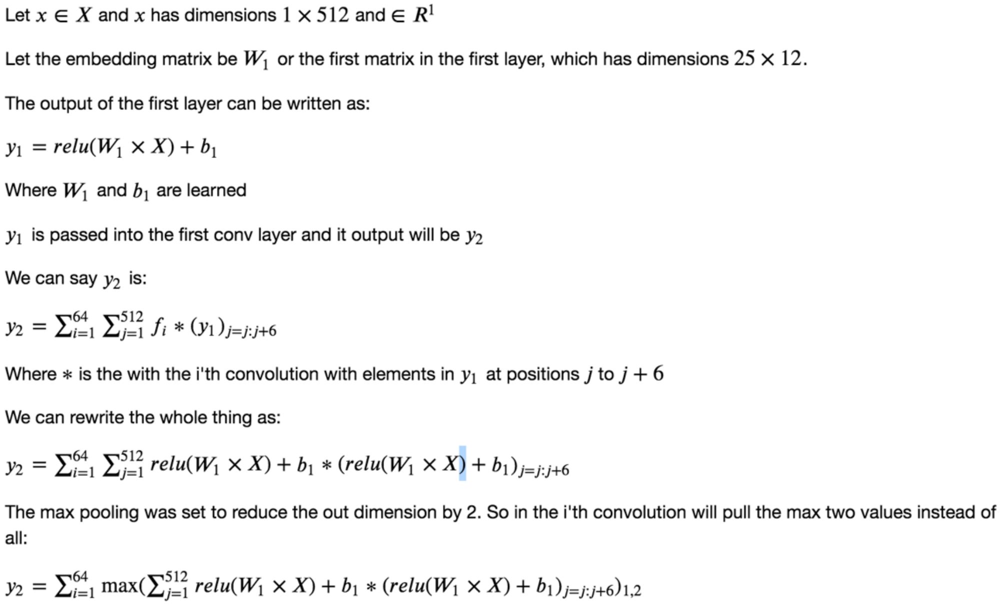
As a first pass, we wanted to test the validity of applying NLP techniques and models to amino acids. For our first model, we chose to combine an embedding layer followed by a 1D convolution and max pooling (repeated twice). Followed by a densely connected layer of 128 units. We use a kernel size of 6, apply padding to keep the input and output dimensions the same after a layer and use the relu activation function. The summary model summary was created using keras and the summary of the layers is shown in Figure 15.



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 512, 12)           312
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 512, 64)           4672
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 256, 64)           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 256, 32)           6176
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 128, 32)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               524416
_________________________________________________________________
dense_4 (Dense)              (None, 15)                1935
=================================================================
Total params: 537,511
Trainable params: 537,511
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 15. Model summary for 1D conv classification model

Our input sample will be a 1 dimensional vector of length 512. Going into the embedding layer, keras will automatically turn this vector into a one hot matrix, where the dimensions will be 512 X 25, since there are 25 unique amino acids in this dataset. The transition matrix will be a 25 X 12 matrix to reduce dimensions. This matrix will be learned by the network through backpropagation and our choice of optimization techniques. This input is that then passed into the 1D conv layer, with kernel size 6, and 64 filters. Max pooling is done and is then passed into the next convolutional layer. Keeping track of the first few matrix operations in the first two layers we can write it out formally as:



Let $x \in X$ and x has dimensions 1×512 and $\in R^1$

Let the embedding matrix be W_1 or the first matrix in the first layer, which has dimensions 25×12.

The output of the first layer can be written as:

$$y_1 = relu(W_1 \times X) + b_1$$

Where W_1 and b_1 are learned

y_1 is passed into the first conv layer and it output will be y_2

We can say y_2 is:

$$y_2 = \sum_{i=1}^{64} \sum_{j=1}^{512} f_i * (y_1)_{j=j:j+6}$$

Where $*$ is the with the i'th convolution with elements in y_1 at positions j to $j+6$

We can rewrite the whole thing as:

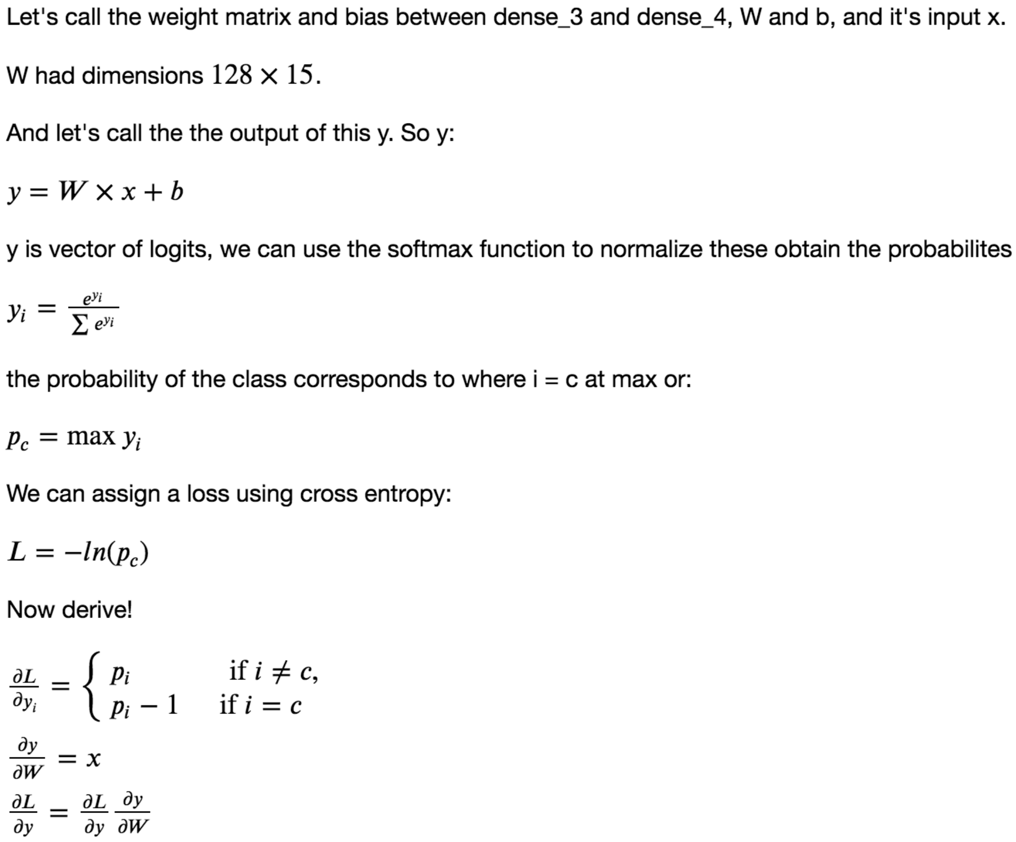
$$y_2 = \sum_{i=1}^{64} \sum_{j=1}^{512} relu(W_1 \times X) + b_1 * (relu(W_1 \times X) + b_1)_{j=j:j+6}$$

The max pooling was set to reduce the out dimension by 2. So in the i'th convolution will pull the max two values instead of all:

$$y_2 = \sum_{i=1}^{64} \max(\sum_{j=1}^{512} relu(W_1 \times X) + b_1 * (relu(W_1 \times X) + b_1)_{j=j:j+6})_{1,2}$$

With an abuse of notation, we can write it out seeing all the operations. y_2 would then be the input into the next layer of this network, which is just the same as the first – except with a smaller matrix!

Backpropagation from the output layer is little tricky, but we can see how the gradients are derived and flow using the last two dense layers. We can write.



Let's call the weight matrix and bias between dense_3 and dense_4, W and b, and it's input x.

W had dimensions 128×15.

And let's call the the output of this y. So y:

$$y = W \times x + b$$

y is vector of logits, we can use the softmax function to normalize these obtain the probabilites

$$y_i = \frac{e^{y_i}}{\sum e^{y_i}}$$

the probability of the class corresponds to where i = c at max or:

$$p_c = \max y_i$$

We can assign a loss using cross entropy:

$$L = -ln(p_c)$$

Now derive!

$$\frac{\partial L}{\partial y_i} = \begin{cases} p_i & \text{if } i \neq c, \\ p_i - 1 & \text{if } i = c \end{cases}$$

$$\frac{\partial y}{\partial W} = x$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial W}$$

Although the derivations are beyond the scope of this experiment, it is helpful to understand how the forward pass math works out and where the gradient flows from in backpropagation. To update the weight matrices, we 'Adam' as our optimizer with a learning rate of 0.001. The derivations for the gradient update are left out of discussion as they are easier to visualize then back propagation.

We run the model for 40 epochs and obtain the following plots loss and accuracy plots. The loss curves show that in 40 iterations of training, we were able to drive the loss to nearly zero for the training set, however the loss on the validation set stops decreasing after epoch 5. In fact, it is evident that we might have lost model performance on the validation set after epoch 5. The gap in the loss curves are indicative of overfitting to the training set.

The accuracy curves show that we were able to get an accuracy of 91% at epoch 40 for the validation set, and around 95% for the training set. Again, this is a clear sign overfitting. Regularization methods such as drop out or L2 norm were not used in this experiment, but it might have helped the accuracy on the validation set, and not affect loss or the accuracy on the training set. We also examined exactly where the model classified incorrectly using a confusion matrix.

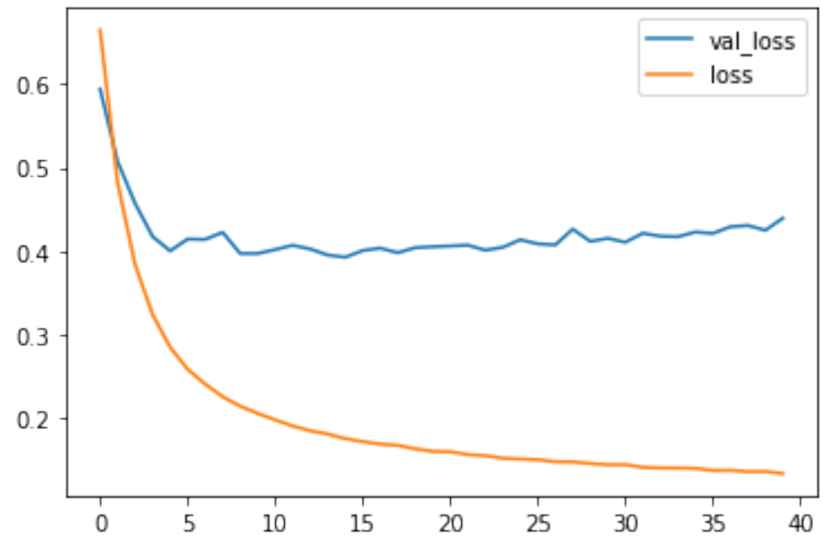


Figure 16. Loss plots for training and validation sets

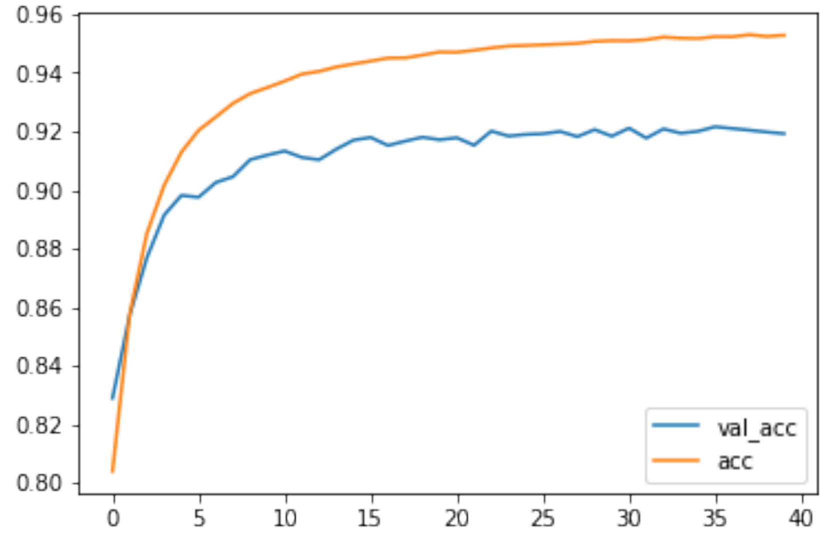
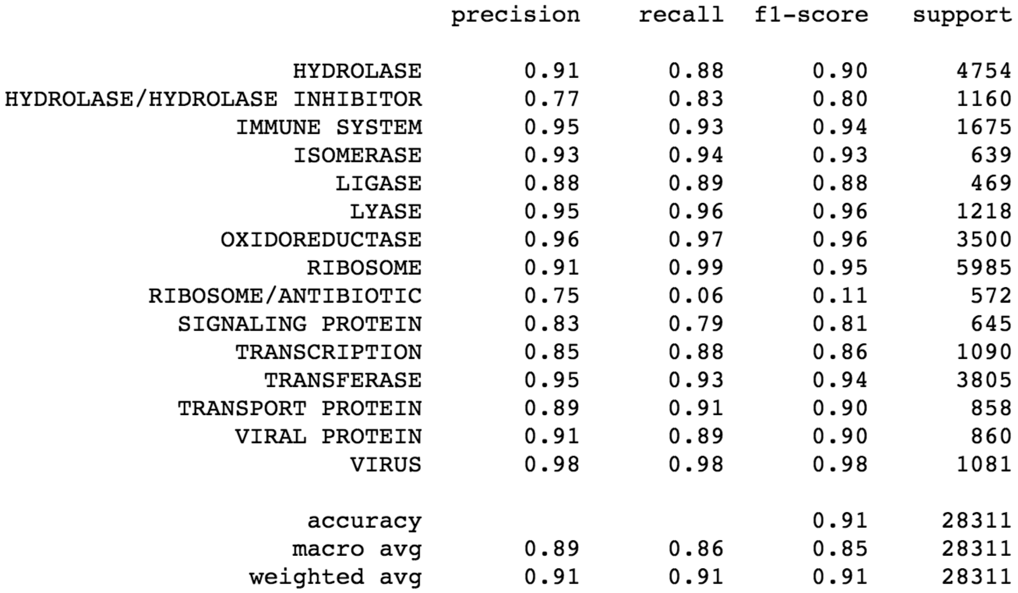


Figure 17. Accuracy Plots for training and validation sets

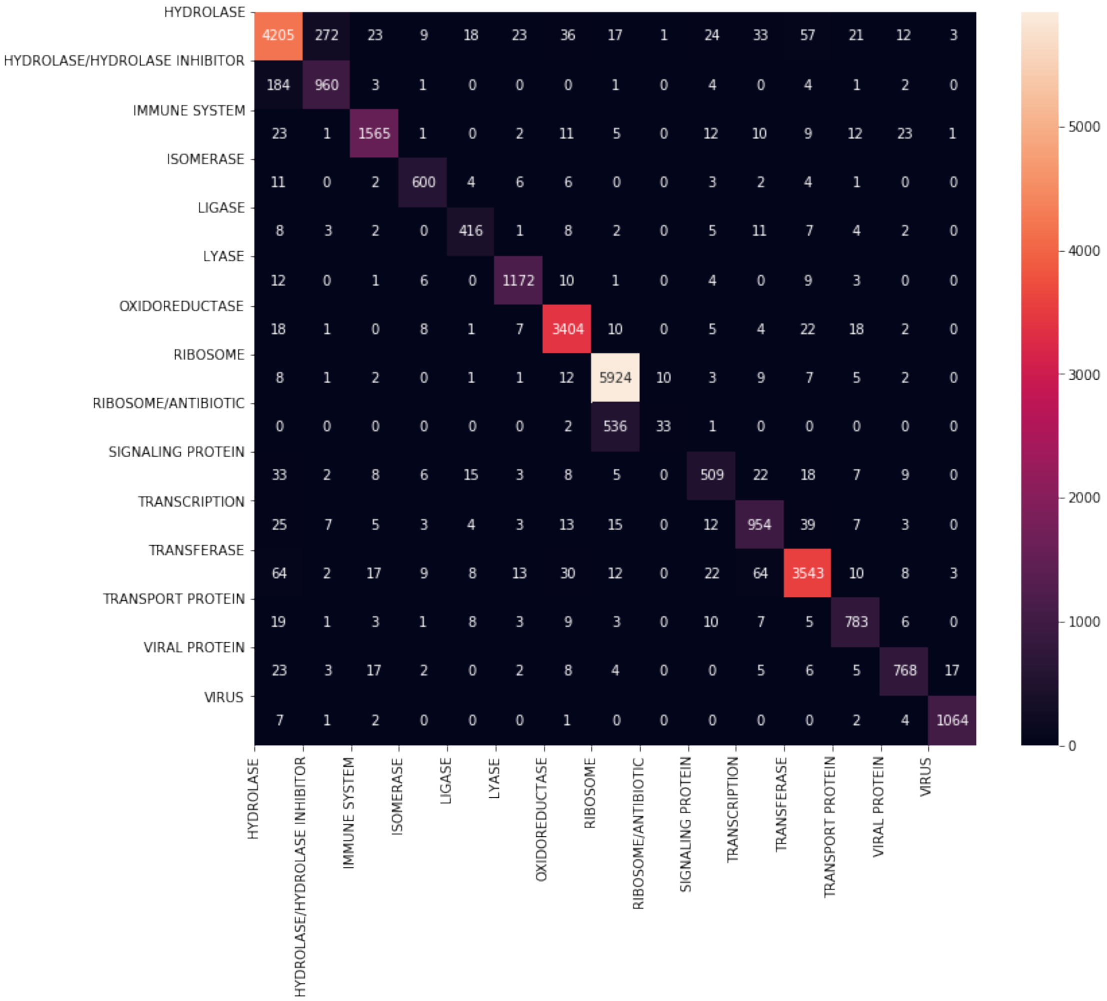


```
                                precision    recall  f1-score   support

                      HYDROLASE      0.91      0.88      0.90      4754
  HYDROLASE/HYDROLASE INHIBITOR      0.77      0.83      0.80      1160
                  IMMUNE SYSTEM      0.95      0.93      0.94      1675
                      ISOMERASE      0.93      0.94      0.93       639
                         LIGASE      0.88      0.89      0.88       469
                          LYASE      0.95      0.96      0.96      1218
                 OXIDOREDUCTASE      0.96      0.97      0.96      3500
                       RIBOSOME      0.91      0.99      0.95      5985
            RIBOSOME/ANTIBIOTIC      0.75      0.06      0.11       572
              SIGNALING PROTEIN      0.83      0.79      0.81       645
                  TRANSCRIPTION      0.85      0.88      0.86      1090
                    TRANSFERASE      0.95      0.93      0.94      3805
              TRANSPORT PROTEIN      0.89      0.91      0.90       858
                  VIRAL PROTEIN      0.91      0.89      0.90       860
                          VIRUS      0.98      0.98      0.98      1081

                       accuracy                          0.91     28311
                      macro avg      0.89      0.86      0.85     28311
                   weighted avg      0.91      0.91      0.91     28311
```

Figure 18. Classification report for first classification model.



	HYDROLASE	HYDROLASE/HYDROLASE INHIBITOR	IMMUNE SYSTEM	ISOMERASE	LIGASE	LYASE	OXIDOREDUCTASE	RIBOSOME	RIBOSOME/ANTIBIOTIC	SIGNALING PROTEIN	TRANSCRIPTION	TRANSFERASE	TRANSPORT PROTEIN	VIRAL PROTEIN	VIRUS
HYDROLASE	4205	272	23	9	18	23	36	17	1	24	33	57	21	12	3
HYDROLASE/HYDROLASE INHIBITOR	184	960	3	1	0	0	0	1	0	4	0	4	1	2	0
IMMUNE SYSTEM	23	1	1565	1	0	2	11	5	0	12	10	9	12	23	1
ISOMERASE	11	0	2	600	4	6	6	0	0	3	2	4	1	0	0
LIGASE	8	3	2	0	416	1	8	2	0	5	11	7	4	2	0
LYASE	12	0	1	6	0	1172	10	1	0	4	0	9	3	0	0
OXIDOREDUCTASE	18	1	0	8	1	7	3404	10	0	5	4	22	18	2	0
RIBOSOME	8	1	2	0	1	1	12	5924	10	3	9	7	5	2	0
RIBOSOME/ANTIBIOTIC	0	0	0	0	0	0	2	536	33	1	0	0	0	0	0
SIGNALING PROTEIN	33	2	8	6	15	3	8	5	0	509	22	18	7	9	0
TRANSCRIPTION	25	7	5	3	4	3	13	15	0	12	954	39	7	3	0
TRANSFERASE	64	2	17	9	8	13	30	12	0	22	64	3543	10	8	3
TRANSPORT PROTEIN	19	1	3	1	8	3	9	3	0	10	7	5	783	6	0
VIRAL PROTEIN	23	3	17	2	0	2	8	4	0	0	5	6	5	768	17
VIRUS	7	1	2	0	0	0	1	0	0	0	0	0	2	4	1064

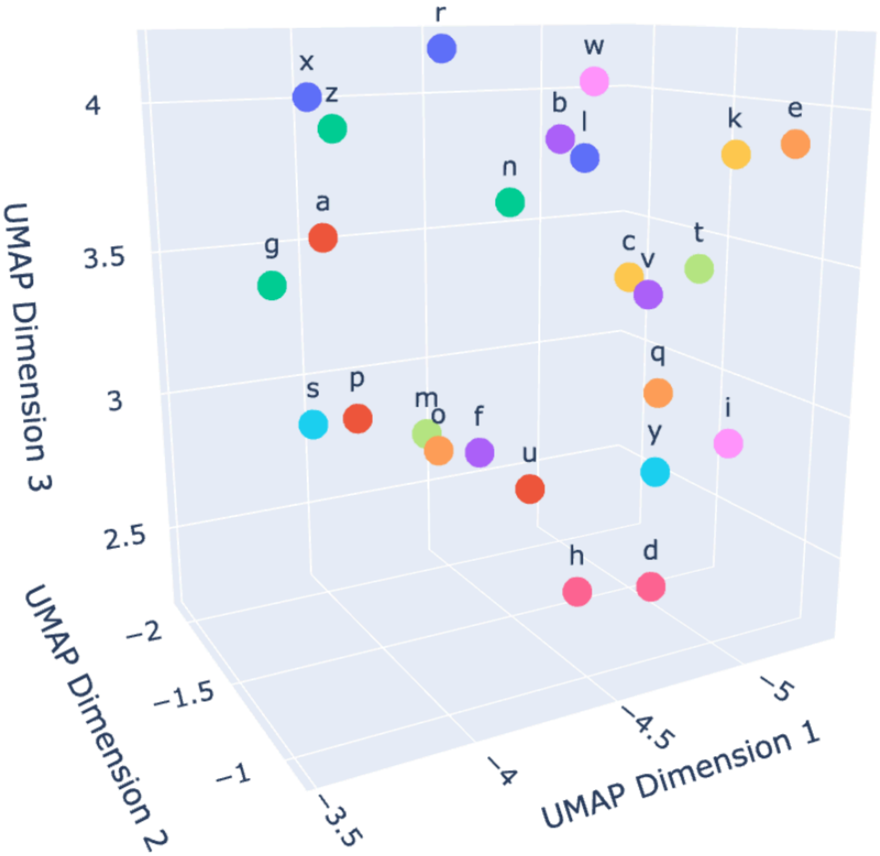
Figure 19. Confusion matrix for model. The X axis is predicted, and Y axis is the actual.

Figures 18 and 19 were generated using a hold-out test set (not the validation set). Looking at the confusion matrix, most of the elements lie along the diagonal, indicating that the model is classifying proteins correctly. The classification report shows that precision for all groups, except for two of them are between 83% to 91%. The same thing goes for recall. The interesting thing to notice are with the two classes 'Ribosome/Antibiotic' and 'Ribosome'. It looks if the model had a hard time distinguishing between these two classes. For 536 predicted samples as 'Ribosome' all 536 samples should have classified as 'Ribosome/Antibiotic'. Most likely these two should have been grouped together as one class before classifying. But it was interesting to see what the model classified them as the same group!

Protein Classification – Examining the Embedding Matrix

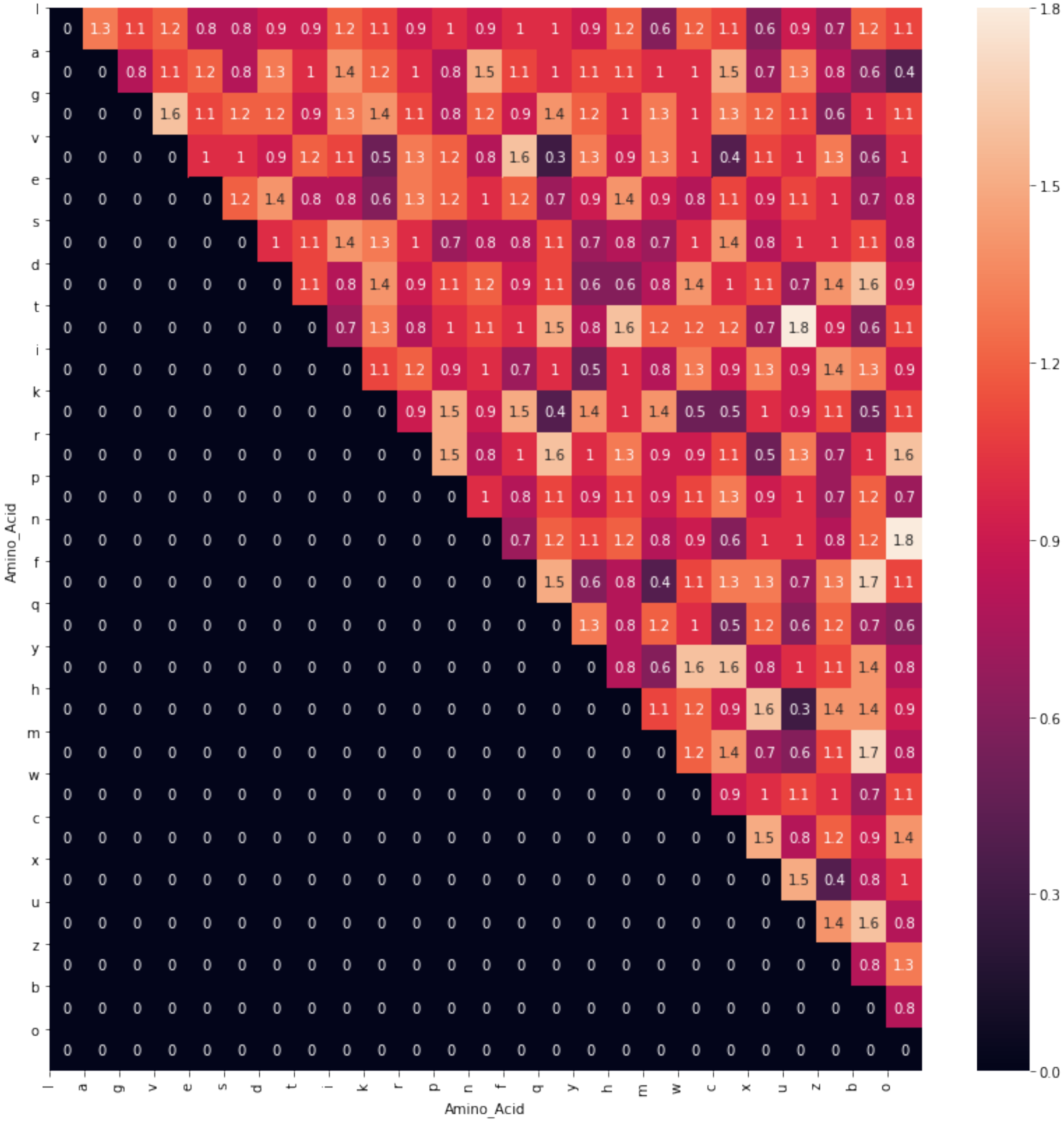
From the learned embedding matrix, we can do two things. We can perform some kind of dimension reduction technique followed by clustering to see what amino acids are most similar to each other. Or we can find similarity scores across for each of the vectors from the embedding matrix to see which two proteins are similar to one another.

We try the first method and perform UMAP on the embedding matrix. The scores from UMAP are mapped in three dimensions. Amino acids are labeled with their one letter abbreviation and amino acids that are close to each other in the space would have some sort of



similarity among them. Since UMAP was performed on the embedding matrix, and the embedding matrix serves as a type of transition matrix for the input into the classification model. We can interpret the nearness of these points as two amino acids being similar to one another.

We can create a similarity score for each two-way comparison of amino acids to create a similarity matrix. In this example, we use the cosine similarity. The image below shows the similarity matrix for each amino acid. For this example, we use the full embedding matrix to have access to the row vectors for each amino acid.



	l	a	g	v	e	s	d	t	i	k	r	p	n	f	q	y	h	m	w	c	x	u	z	b	o
l	0	1.3	1.1	1.2	0.8	0.8	0.9	0.9	1.2	1.1	0.9	1	0.9	1	1	0.9	1.2	0.6	1.2	1.1	0.6	0.9	0.7	1.2	1.1
a	0	0	0.8	1.1	1.2	0.8	1.3	1	1.4	1.2	1	0.8	1.5	1.1	1	1.1	1.1	1	1	1.5	0.7	1.3	0.8	0.6	0.4
g	0	0	0	1.6	1.1	1.2	1.2	0.9	1.3	1.4	1.1	0.8	1.2	0.9	1.4	1.2	1	1.3	1	1.3	1.2	1.1	0.6	1	1.1
v	0	0	0	0	1	1	0.9	1.2	1.1	0.5	1.3	1.2	0.8	1.6	0.3	1.3	0.9	1.3	1	0.4	1.1	1	1.3	0.6	1
e	0	0	0	0	0	1.2	1.4	0.8	0.8	0.6	1.3	1.2	1	1.2	0.7	0.9	1.4	0.9	0.8	1.1	0.9	1.1	1	0.7	0.8
s	0	0	0	0	0	0	1	1.1	1.4	1.3	1	0.7	0.8	0.8	1.1	0.7	0.8	0.7	1	1.4	0.8	1	1	1.1	0.8
d	0	0	0	0	0	0	0	1.1	0.8	1.4	0.9	1.1	1.2	0.9	1.1	0.6	0.6	0.8	1.4	1	1.1	0.7	1.4	1.6	0.9
t	0	0	0	0	0	0	0	0	0.7	1.3	0.8	1	1.1	1	1.5	0.8	1.6	1.2	1.2	1.2	0.7	1.8	0.9	0.6	1.1
i	0	0	0	0	0	0	0	0	0	1.1	1.2	0.9	1	0.7	1	0.5	1	0.8	1.3	0.9	1.3	0.9	1.4	1.3	0.9
k	0	0	0	0	0	0	0	0	0	0	0.9	1.5	0.9	1.5	0.4	1.4	1	1.4	0.5	0.5	1	0.9	1.1	0.5	1.1
r	0	0	0	0	0	0	0	0	0	0	0	1.5	0.8	1	1.6	1	1.3	0.9	0.9	1.1	0.5	1.3	0.7	1	1.6
p	0	0	0	0	0	0	0	0	0	0	0	0	1	0.8	1.1	0.9	1.1	0.9	1.1	1.3	0.9	1	0.7	1.2	0.7
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	1.2	1.1	1.2	0.8	0.9	0.6	1	1	0.8	1.2	1.8
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	0.6	0.8	0.4	1.1	1.3	1.3	0.7	1.3	1.7	1.1
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.3	0.8	1.2	1	0.5	1.2	0.6	1.2	0.7	0.6
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8	0.6	1.6	1.6	0.8	1	1.1	1.4	0.8
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.1	1.2	0.9	1.6	0.3	1.4	1.4	0.9
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.2	1.4	0.7	0.6	1.1	1.7	0.8
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9	1	1.1	1	0.7	1.1
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	0.8	1.2	0.9	1.4
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	0.4	0.8	1
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.4	1.6	0.8
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8	1.3
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20. Similarity Matrix for amino acids given their one letter abbreviation. The I,j value in this matrix corresponds to the cosine similarity between two amino acids.

The above matrix corresponds the cosine similarity:

$$1 - \frac{u \cdot v}{||u||_2 ||v||_2}$$

The smaller this number is, the more similar two amino acids are. Looking along the diagonal of the matrix, the trace is equal to 0 and each element along the diagonal is zero. This makes sense. Picking just a few amino acid pairs that are somewhat similar we have the following: Alanine and Pyrrolysine with a similarity score of 0.4, Valine and Glutamine have a similarity score of 0.3, and Valine and Cysteine have a similarity of score of 0.3. Looking at some of the computational biology literature, we found that some of our similarity scores actually matched up with known matrices![3]

Protein Classification – Other Model Architectures

Since the amount of time to train one model takes a significant amount of time (the training time for 40 epochs in our first model was about 10 hours), we try experimenting with other model architectures instead of varying the hyperparameters. In particular, we tried four different models. For Model 1, we used a similar structure as we did in our first attempt. The first layer contains an embedding layer, then followed by three layers with each layer having 1D convolution followed by a max pooling layer. The remaining two layers are fully connected dense layers.

Model 2 uses the same embedding layer, but the output is passed into an LSTM layer with 12 hidden unites, which is then passed into another fully connected layer.

Model 3 is the same as model 2 but uses a GRU unit instead of the LSTM unit.

Model 4 uses a bidirectional LSTM layer.

Their structures are outlined in the table below.

Model No.	Description	Number of layers	Number of Parameters
1	Embedding Layer, 1D Conv with 64 filters, max pooling, 1D Conv with 32 layers, max pooling, 1D Conv with 16 filters, max pooling, fully connected layer 128 units, fully connected layer 15 units	10	151,071
2	Embedding Layer, 20-unit LSTM, fully connected layer 15 units	3	3,691

3	Embedding Layer, 32-unit GRU, 15-unit fully connected layer	3	5,615
4	Embedding layer, 64 unit, 15-unit fully connected layer	3	10,799

We leave the inner workings of LSTM and GRU for the reader. We wanted to see if a sequential model learns better or as good as a convolutional network. Again, we created the X samples and Y target labels as we did the previous model and obtain the following training accuracy and loss plot for each of the four models.

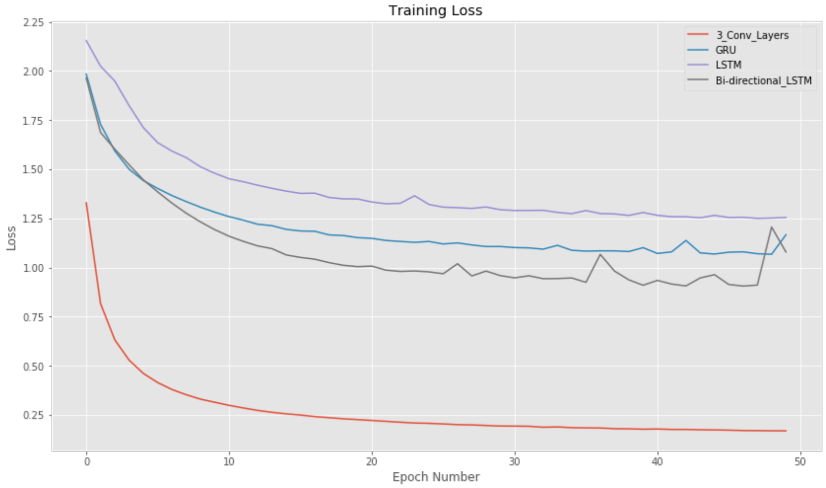


Figure 21. Training Loss for each of the four classification models.

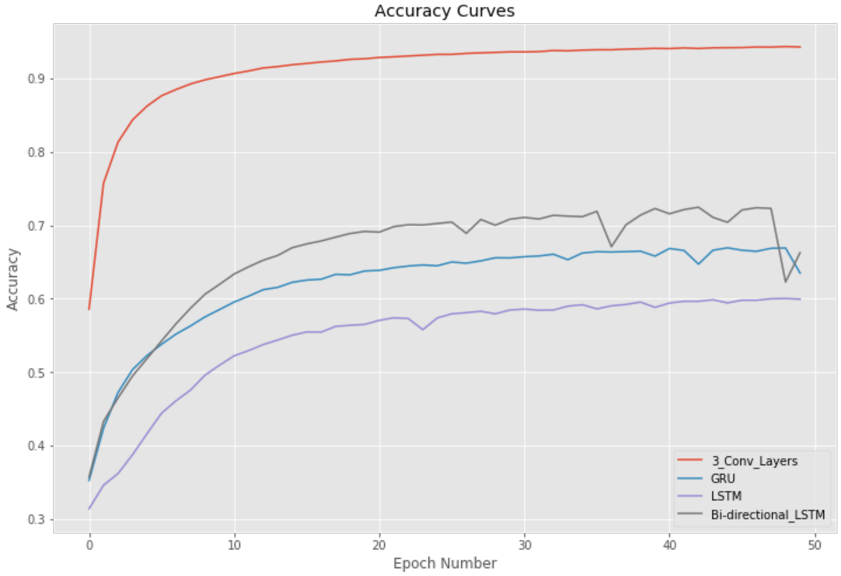


Figure 22. Training accuracy curves for each of the four models.

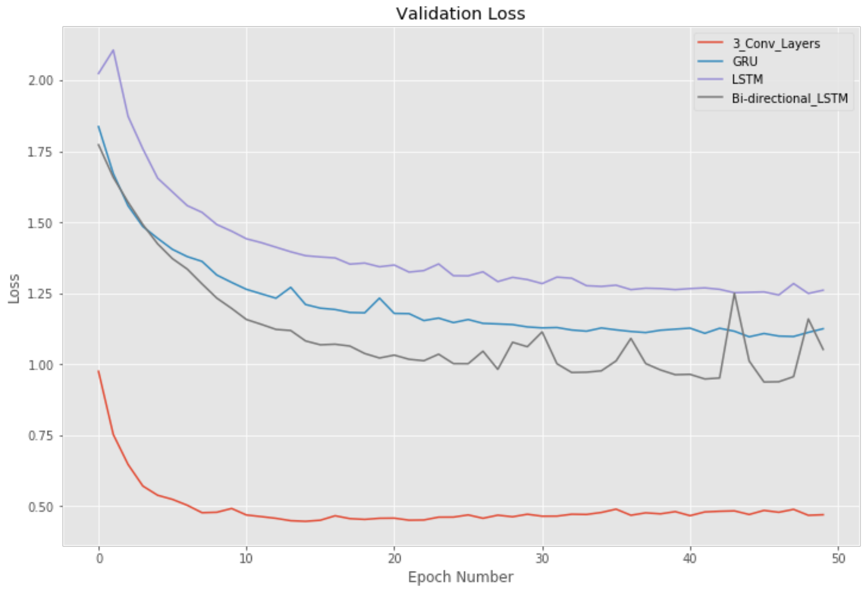


Figure 23. Validation loss curves for each of the four models

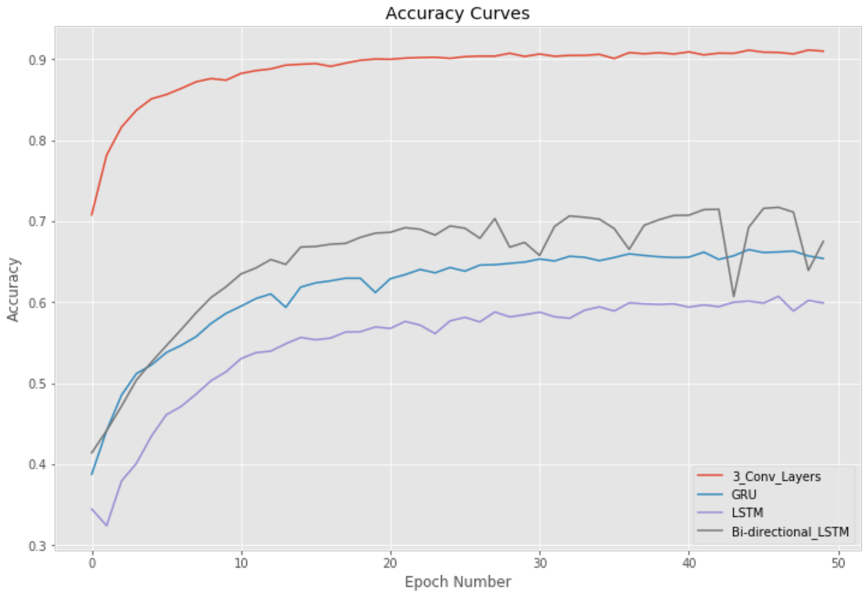


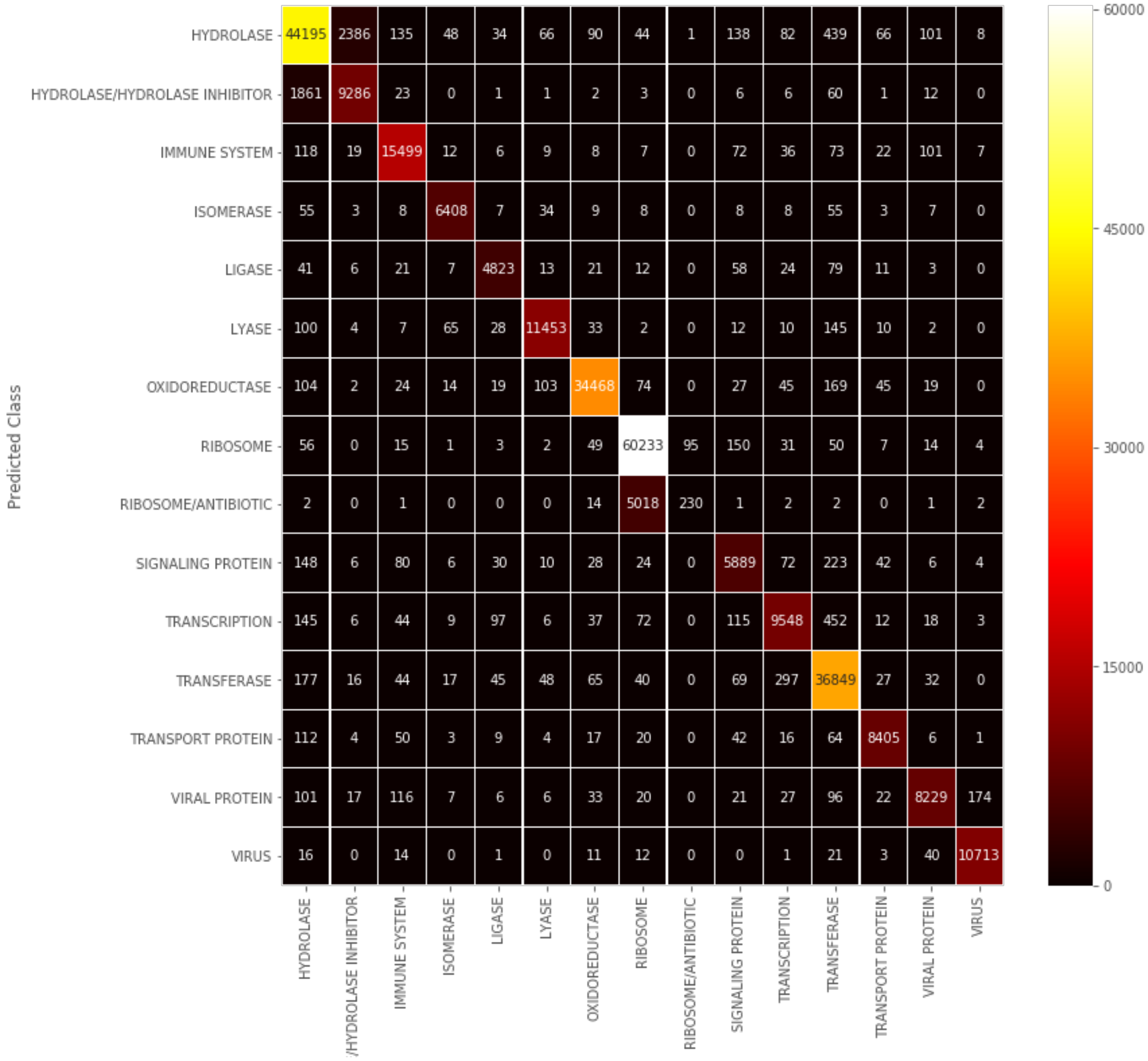
Figure 24. Validation accuracy curves for each of the four models

In all four of the metric curves, Model 1 (3_Conv_Layers), outperforms any of the of sequential learning models. Coming right out of the gate at the first epoch, Model 1 already has a higher training and validation accuracy at the first epoch, and a lower training and validation loss at the first epoch than any of the three models. Overfitting is clearly evident in all these plots as we quickly start to learn the dataset in the first few epochs. For these models, no regularization was used. The highest accuracy on the validation set was about 92% for the three-layer convolutional layer model and the loss was around 0.5. Even if were to continue training for a longer number of epochs, performance would not get much better.

Another important thing we noticed was that of the three sequential learning models, the bi-directional models performed the best. Achieving a lower loss and a higher accuracy at the end of the training periods. However, for the bi-directional LSTM, there is more oscillatory behavior towards the end of training (i.e. there is more variance in the values for accuracy and loss). The results seem to indicate that a sequential learning model is not as strong as a convolutional model in terms of predicting whether a chain of amino acids belongs to a certain protein class. This gives credence to the assumption that the ordering of amino acids does not matter as much compared to the overall content of the amino acids in a chain at predicting whether a chain belongs to a certain class.

Protein Classification – Other Model Architectures - Confusion Matrices and Reporting

In this section we evaluate the efficacy of each of these models by showing their respective confusion matrices to see where exactly the models are making mistakes. Figure 25 show the confusion matrix for model 1, the three convolutional layer model. When creating this



	HYDROLASE	/HYDROLASE INHIBITOR	IMMUNE SYSTEM	ISOMERASE	LIGASE	LYASE	OXIDOREDUCTASE	RIBOSOME	RIBOSOME/ANTIBIOTIC	SIGNALING PROTEIN	TRANSCRIPTION	TRANSFERASE	TRANSPORT PROTEIN	VIRAL PROTEIN	VIRUS
HYDROLASE	44195	2386	135	48	34	66	90	44	1	138	82	439	66	101	8
HYDROLASE/HYDROLASE INHIBITOR	1861	9286	23	0	1	1	2	3	0	6	6	60	1	12	0
IMMUNE SYSTEM	118	19	15499	12	6	9	8	7	0	72	36	73	22	101	7
ISOMERASE	55	3	8	6408	7	34	9	8	0	8	8	55	3	7	0
LIGASE	41	6	21	7	4823	13	21	12	0	58	24	79	11	3	0
LYASE	100	4	7	65	28	11453	33	2	0	12	10	145	10	2	0
OXIDOREDUCTASE	104	2	24	14	19	103	34468	74	0	27	45	169	45	19	0
RIBOSOME	56	0	15	1	3	2	49	60233	95	150	31	50	7	14	4
RIBOSOME/ANTIBIOTIC	2	0	1	0	0	0	14	5018	230	1	2	2	0	1	2
SIGNALING PROTEIN	148	6	80	6	30	10	28	24	0	5889	72	223	42	6	4
TRANSCRIPTION	145	6	44	9	97	6	37	72	0	115	9548	452	12	18	3
TRANSFERASE	177	16	44	17	45	48	65	40	0	69	297	36849	27	32	0
TRANSPORT PROTEIN	112	4	50	3	9	4	17	20	0	42	16	64	8405	6	1
VIRAL PROTEIN	101	17	116	7	6	6	33	20	0	21	27	96	22	8229	174
VIRUS	16	0	14	0	1	0	11	12	0	0	1	21	3	40	10713

Figure 25. Confusion matrix for Model 1.

confusion matrix, we used the union of the training data and the test data. There were not enough test samples for the 15 groups of proteins if we only used the test set.

Looking along the diagonal for this confusion matrix, we see that most of the counts lie along here. For the classes Hydrolase and Hydrolase/Hydrolase Inhibitor, we see that the model had a hard time differentiating between the two (in the upper left corner). There are few blocks in the off diagonal for these two groups. For example, 1861 samples were classified as Hydrolase, but those samples should have been classified at Hydrolase/Inhibitor instead. We see the similar pattern with Ribosome and Ribosome/Antibiotic. There were 5018 samples

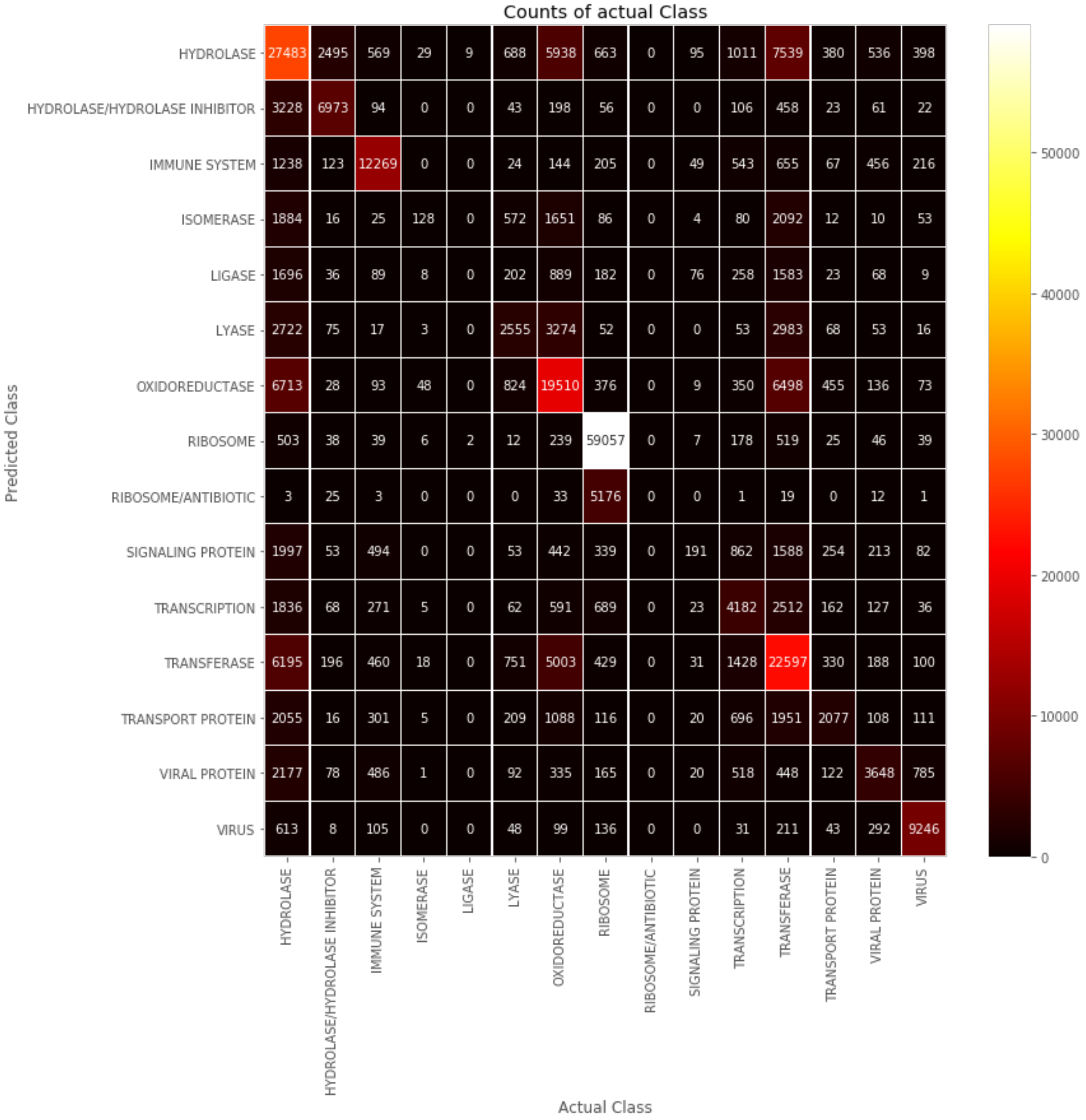
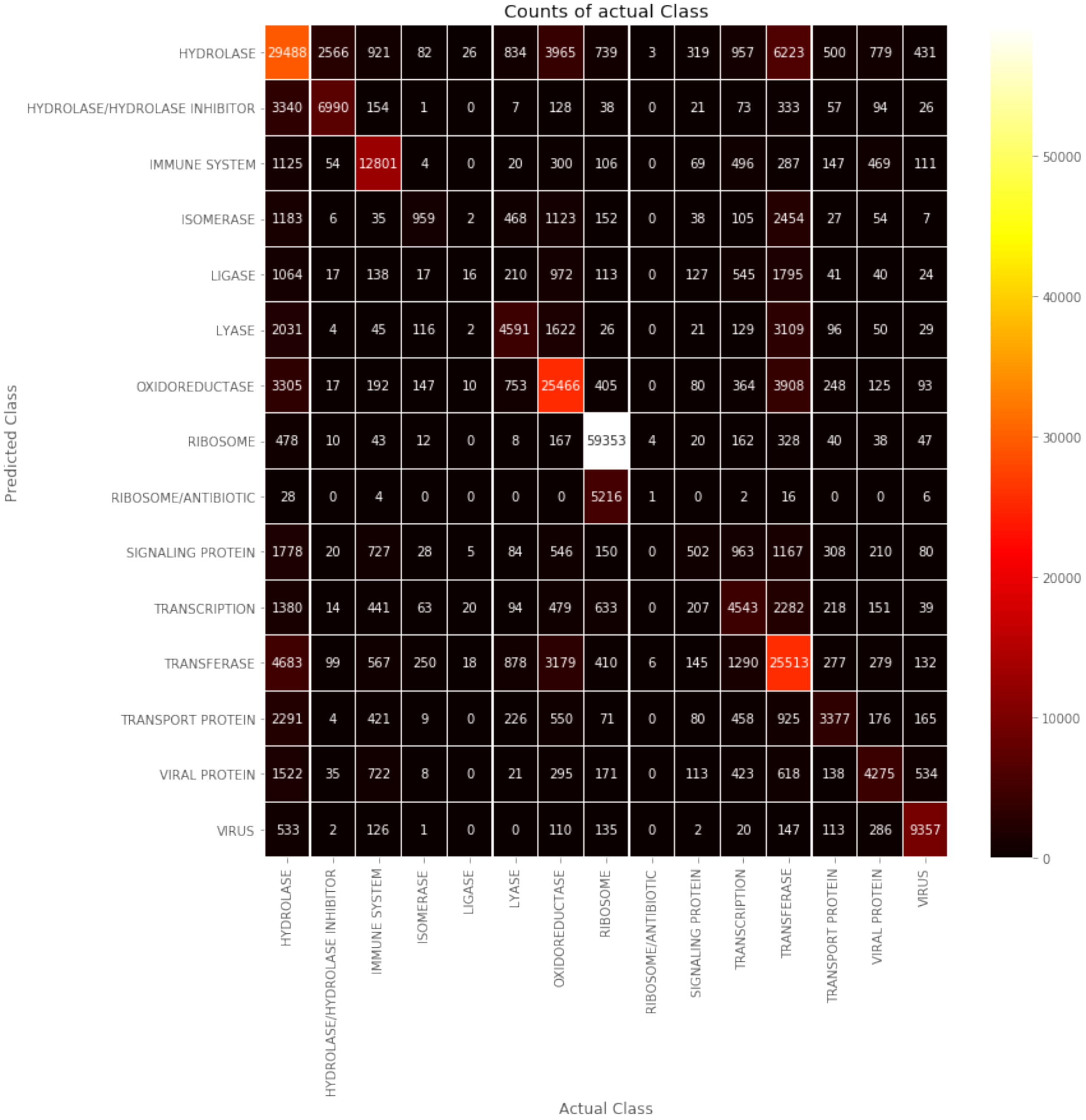


Figure 26. Confusion Matrix for Model 2

classified as Ribosome/Antibiotic but should haven classified at Ribosome. It might be likely that these two classes should be classified as the same.

We then go on to Model 2, in Figure 26. Looking along the diagonal, we see that there are not as many entries present as compared to Model 1. The only entries along the diagonal that have a large number of counts compared to the other entries, are entries corresponding to the protein classes Hydrolase, Hydrolase Inhibitor, Immune System, Oxidoreductase, Ribosome, Transferase, and Virus. It was interesting to see that a sequential model was not able to identify some of the protein classes that a convolutional model coul.

Below is the confusion matrix for Model 3.

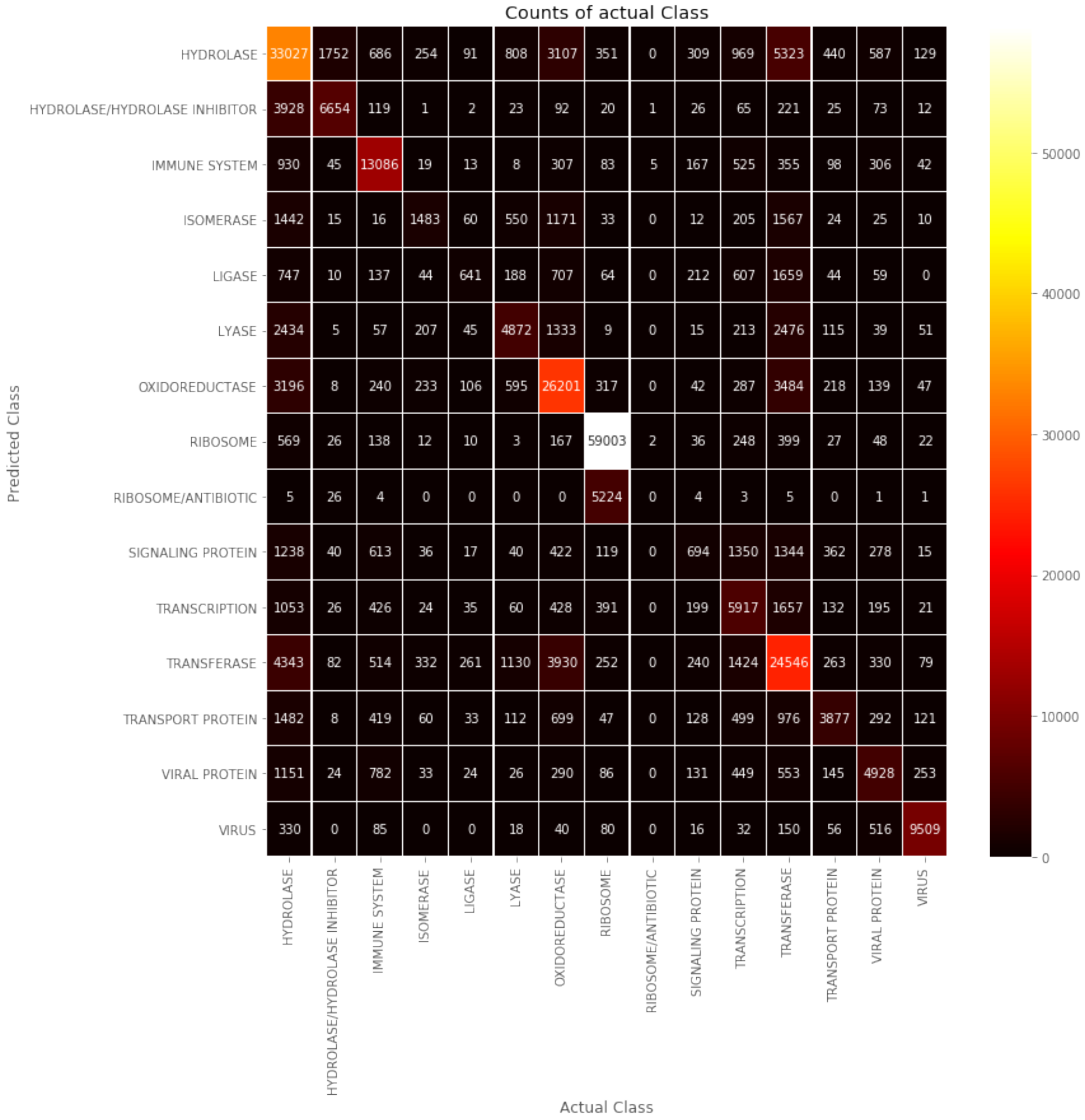


	HYDROLASE	HYDROLASE/HYDROLASE INHIBITOR	IMMUNE SYSTEM	ISOMERASE	LIGASE	LYASE	OXIDOREDUCTASE	RIBOSOME	RIBOSOME/ANTIBIOTIC	SIGNALING PROTEIN	TRANSCRIPTION	TRANSFERASE	TRANSPORT PROTEIN	VIRAL PROTEIN	VIRUS
HYDROLASE	29488	2566	921	82	26	834	3965	739	3	319	957	6223	500	779	431
HYDROLASE/HYDROLASE INHIBITOR	3340	6990	154	1	0	7	128	38	0	21	73	333	57	94	26
IMMUNE SYSTEM	1125	54	12801	4	0	20	300	106	0	69	496	287	147	469	111
ISOMERASE	1183	6	35	959	2	468	1123	152	0	38	105	2454	27	54	7
LIGASE	1064	17	138	17	16	210	972	113	0	127	545	1795	41	40	24
LYASE	2031	4	45	116	2	4591	1622	26	0	21	129	3109	96	50	29
OXIDOREDUCTASE	3305	17	192	147	10	753	25466	405	0	80	364	3908	248	125	93
RIBOSOME	478	10	43	12	0	8	167	59353	4	20	162	328	40	38	47
RIBOSOME/ANTIBIOTIC	28	0	4	0	0	0	0	5216	1	0	2	16	0	0	6
SIGNALING PROTEIN	1778	20	727	28	5	84	546	150	0	502	963	1167	308	210	80
TRANSCRIPTION	1380	14	441	63	20	94	479	633	0	207	4543	2282	218	151	39
TRANSFERASE	4683	99	567	250	18	878	3179	410	6	145	1290	25513	277	279	132
TRANSPORT PROTEIN	2291	4	421	9	0	226	550	71	0	80	458	925	3377	176	165
VIRAL PROTEIN	1522	35	722	8	0	21	295	171	0	113	423	618	138	4275	534
VIRUS	533	2	126	1	0	0	110	135	0	2	20	147	113	286	9357

Figure 27. Confusion matrix for Model 3

The confusion matrix for Model 3 suffers the same symptoms as for the confusion matrix for Model 2.

We now proceed to look at the confusion matrix for Model 4.



Predicted Class \ Actual Class	HYDROLASE	HYDROLASE/HYDROLASE INHIBITOR	IMMUNE SYSTEM	ISOMERASE	LIGASE	LYASE	OXIDOREDUCTASE	RIBOSOME	RIBOSOME/ANTIBIOTIC	SIGNALING PROTEIN	TRANSCRIPTION	TRANSFERASE	TRANSPORT PROTEIN	VIRAL PROTEIN	VIRUS
HYDROLASE	33027	1752	686	254	91	808	3107	351	0	309	969	5323	440	587	129
HYDROLASE/HYDROLASE INHIBITOR	3928	6654	119	1	2	23	92	20	1	26	65	221	25	73	12
IMMUNE SYSTEM	930	45	13086	19	13	8	307	83	5	167	525	355	98	306	42
ISOMERASE	1442	15	16	1483	60	550	1171	33	0	12	205	1567	24	25	10
LIGASE	747	10	137	44	641	188	707	64	0	212	607	1659	44	59	0
LYASE	2434	5	57	207	45	4872	1333	9	0	15	213	2476	115	39	51
OXIDOREDUCTASE	3196	8	240	233	106	595	26201	317	0	42	287	3484	218	139	47
RIBOSOME	569	26	138	12	10	3	167	59003	2	36	248	399	27	48	22
RIBOSOME/ANTIBIOTIC	5	26	4	0	0	0	0	5224	0	4	3	5	0	1	1
SIGNALING PROTEIN	1238	40	613	36	17	40	422	119	0	694	1350	1344	362	278	15
TRANSCRIPTION	1053	26	426	24	35	60	428	391	0	199	5917	1657	132	195	21
TRANSFERASE	4343	82	514	332	261	1130	3930	252	0	240	1424	24546	263	330	79
TRANSPORT PROTEIN	1482	8	419	60	33	112	699	47	0	128	499	976	3877	292	121
VIRAL PROTEIN	1151	24	782	33	24	26	290	86	0	131	449	553	145	4928	253
VIRUS	330	0	85	0	0	18	40	80	0	16	32	150	56	516	9509

Figure 28. Confusion Matrix for Model 4

Again, like the confusion matrices for Model 2 and Model 3, the confusion matrix for Model 4 has the same problems. It is worth noting that the groups Hydrolase and Ribosome could be merged together with Hydrolase Inhibitor and Ribosome/Antibiotic. It appears that all the models have difficulty trying to distinguish between a protein in the class Hydrolase and

Hydrolase/Hydrolase Inhibitor. We can finally show the accuracies for each of these models for each protein class in a table shown below.

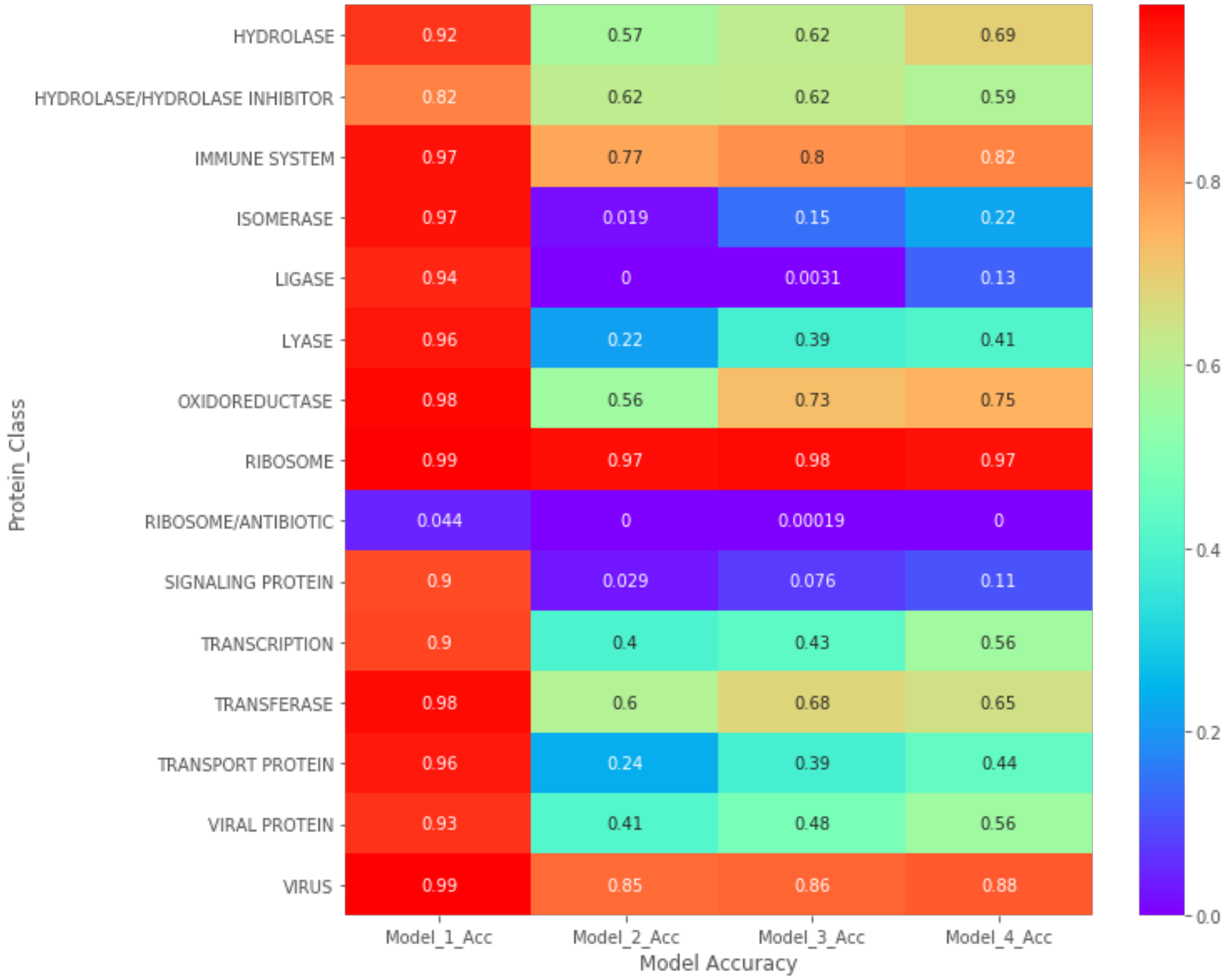


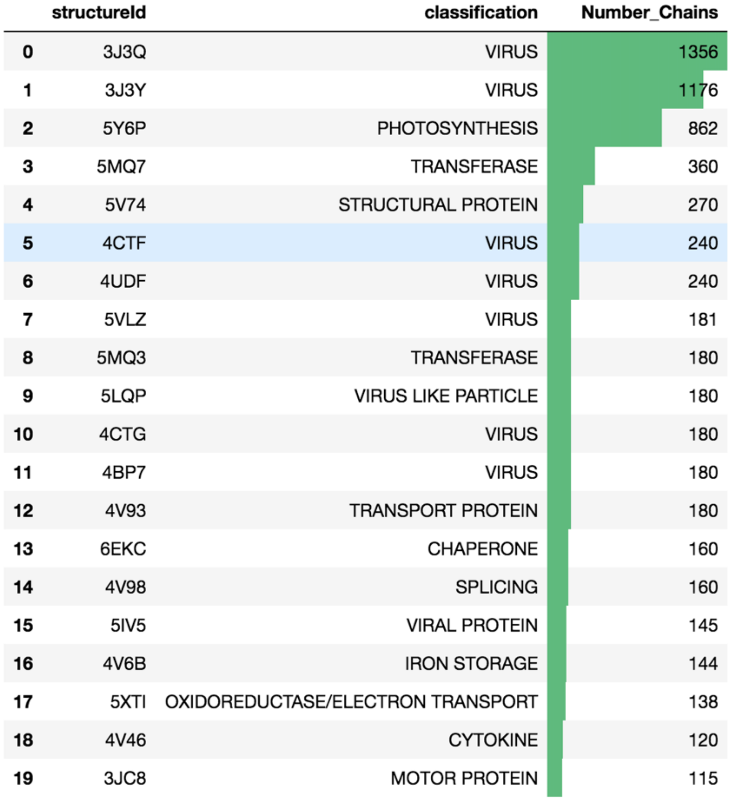
Figure 29. Model Accuracies for protein classes

If we take accuracies into consideration, then Model 1, three convolutional layers, clearly beats the other three models. However, it would appear that the other three sequential models classify the protein class Ribosomes just as all as a convolutional model. The accuracies for Ribosomes for each model are 0.99, 0.97, 0.98, and 0.97 respectively. The protein class Virus also seems to be classified pretty well.

For all four models, the accuracy was very low. We now go on to the final section of the paper where we attempt to predict the next amino acid given a sequence of amino acids. In particular we will try to see if we can create viruses!

Viral Protein Sequence Prediction

If we revisit the classification modeling scheme, what we were doing was taking a 'chain' of a protein and trying to predict what class of proteins that chain is in. However, proteins are actually made up of multiple chains. Some virus proteins are made up of 1300 different amino acid chains and some hydrolase proteins are simple one chain proteins. The original dataset contained over 300,000 different protein chains, but really there are only about 123,000 unique proteins. Below is a table showing the top 20 or so proteins by their unique structure ID from PDB, their classification, and the number of chains (ordered by this).



	structureId	classification	Number_Chains
0	3J3Q	VIRUS	1356
1	3J3Y	VIRUS	1176
2	5Y6P	PHOTOSYNTHESIS	862
3	5MQ7	TRANSFERASE	360
4	5V74	STRUCTURAL PROTEIN	270
5	4CTF	VIRUS	240
6	4UDF	VIRUS	240
7	5VLZ	VIRUS	181
8	5MQ3	TRANSFERASE	180
9	5LQP	VIRUS LIKE PARTICLE	180
10	4CTG	VIRUS	180
11	4BP7	VIRUS	180
12	4V93	TRANSPORT PROTEIN	180
13	6EKC	CHAPERONE	160
14	4V98	SPLICING	160
15	5IV5	VIRAL PROTEIN	145
16	4V6B	IRON STORAGE	144
17	5XTI	OXIDOREDUCTASE/ELECTRON TRANSPORT	138
18	4V46	CYTOKINE	120
19	3JC8	MOTOR PROTEIN	115

Figure 30. Number of chains for unique protein structure

Our idea is simple, what if we could create a new protein if we trained a sequential model on a single class of proteins? We would pick all the proteins in a certain class. Since there are multiple chains for a specific protein, we can concatenate each chain in a unique protein together but demarcated with a special character. This allows the representation to have a semblance of chains (i.e. consider each chain as a sentence, and each sentence can be combined together to make a paragraph). Now, with just one protein, we can again concatenate the concatenated chains with another special character different from the first. Now the concatenated chain has semblance of chains, and a semblance of what proteins are different.

This is best seen with an example. Say for example that we have two proteins. Let's call them A and B. Protein A is made up of 3 unique chains: "AAAA", "LLLL", "WWW". We can concatenate this with a special character, let's use '+'. We can now represent Protein A as 'AAAA+LLLL+WWWW'. Now, let's examine Protein B, which is made up of two chains: "DDDD", and “YYYY”. We can do the same thing as we did with A and get “DDDD+YYYY”. Since A and B are in the same protein class, we can concatenate this even further by using “|”, to separate A from B. We now have the final sequence as “AAAA+LLLL+WWWW| DDDD+YYYY’. We replicate for all proteins in a class and use the final sequence to train a sequential model (LSTM). The idea is similar to training an LSTM on a bunch of Harry Potter Books and have it create new sequences that are hopefully Harry Potter themed!

This is exactly what do. We subset the data for all protein classes related to viruses. We find that there are 3595 unique protein chains that have some sort of viral function. We join each chain with a “+”, for each unique protein we join the concatenated sequences with a “|”. This give us a string with more than 6000000 characters in it! This is more than enough to train a language model.

We one hot each character as we did for the previous models mentioned and create a rolling window of size 30 (i.e. we take 30 amino acids at a time, and then have the model predict the next amino acid after). The model we decided on was a GRU with 130 hidden units. This took a very long time to train, even with a GPU. Ten epochs of training took approximately 24 hours. We let the model train for about two weeks. The loss and accuracy curves are shown below.

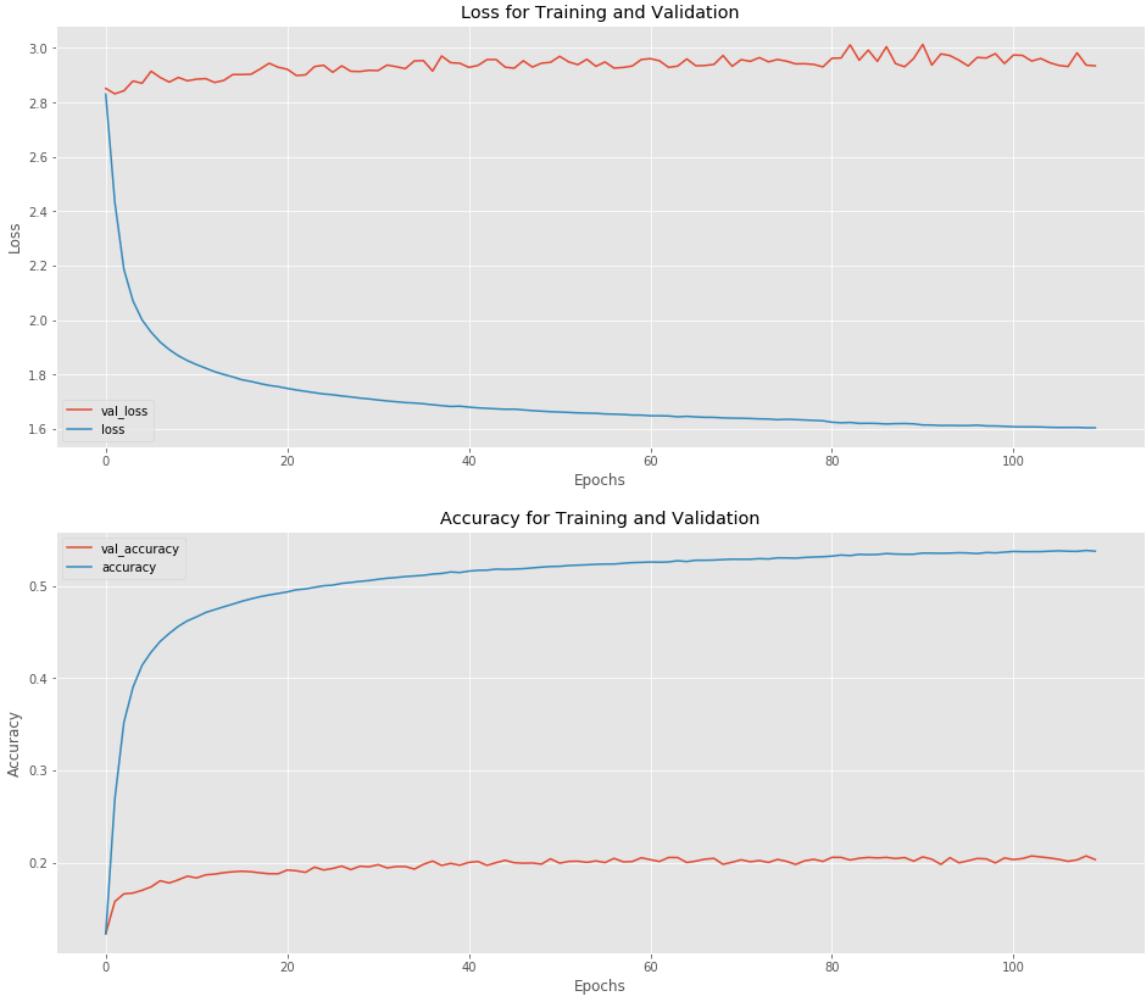


Figure 31. Loss and Accuracy curves for GRU virus model

The model clearly performs pitifully at predicting the next amino acid given a sequence of 30 amino acids. However, it is worth noting that we only trained for 100 epochs in the period of two weeks. Recurrent models usually require large iterations of training, with the most successful models reaching more appropriate accuracies around 500 epochs of training[4]. It is also worth noting that the models are clearly overfitting, but this is expected. If the recurrent models are only being used to evaluate a generated sequence, then loss and accuracy metrics are not the ways we want to evaluate the model. In the next part of the paper, we will use the model to create a new amino acid sequence.

Protein Viral Sequence Creation

First, we will see how well our model can recreate a viral protein that it was trained on. In our dataset, the longest viral protein belongs to '3J3Q', which has a sequence length of 314591 amino acids. Below are the first few hundred amino acids in this sequence:

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSALSEGATPQDLNTMLNTVGGHQAAMQLKETI
NEEAAEWDRLHPVHAGPIEPGQMREPRGSDIAGTTSTLQEQIGWMTHNPPIPVGEIYKRWIILGLNKIVRMYSP
TSILDIRQGPKEPFRDYVDRFYKTLRAEQASQEVKNWMTETLLVQNANPDCKTILKALGPAATLEEMMTACQGV
GGPGHKARVL|PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSALSEGATPQDLNTMLNTVGGHQ
AAMQMLKETINEEAAEWDRLHPVHAGPIEPGQMREPRGSDIAGTTSTLQEQIGWMTHNPPIPVGEIYKRWIILG
LNKIVRMYSPTSILDIRQGPKEPFRDYVDRFYKTLRAEQASQEVKNWMTETLLVQNANPDCKTILKALGPAATL
EEMMTACQGVGGPGHKARVL|PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSALSEGATPQDLN
TMLNTVGGHQAAMQMLKETINEEAAEWDRLHPVHAGPIEPGQMREPRGSDIAGTTSTLQEQIGWMTHNPPIPVG
EIYKRWIILGLNKIVRMYSPTSILDIRQGPKEPFRDYVDRFYKTLRAEQASQEVKNWMTETLLVQNANPDCKTI
LKALGPAATLEEMMTACQGVGGPGHKARVL|PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSAL
SEGATPQDLNTMLNTVGGHQAAMQMLKETINEEAAEWDRLHPVHAGPIEPGQMREPRGSDIAGTTSTLQEQIGW
MTHNPPIPVGEIYKRWIILGLNKIVRMYSPTSILDIRQGPKEPFRDYVDRFYKTLRAEQASQEVKNWMTETLLV
QNANPDCKTILKALGPAATLEEMMTACQGVGGPGHKARVL|PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFS
PEVIPMFSALSEGATPQDLNTMLNTVGGHQAAMQMLKETINEEAAEWDRLHPVHAGPIEPGQMREPRGSDIAGT
TSTLQEQIGWMTHNPPIPVGEIYKRWIILGLNKIVRMYSPTSILDIRQGPKEPFRDYVDRFYKTLRAEQASQEV
KNWMTETLLVQNANPDCKTILKALGPAATLEEMMTACQGVGGPGHKARVL|PIVQNLQGQMVHQAISPRTLNAW
VKVVEEKAFSPEVIPMF'
```

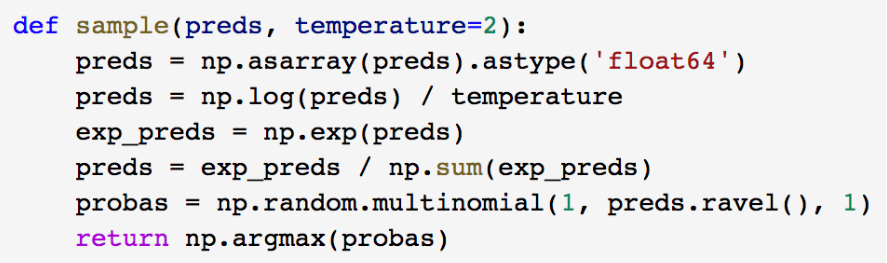
We can create a seed of the first 30 amino acids in the sequence and see how well our model does at re-creating this sequence. Doing so we get the following result:

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKSSDISNNNKKKKKYKYKKRRLLKRRRRIARRERAEMMDQYEEE
KEKEEEEEKEKKKKKKKLKEVVRGKEPNDVVVYVIVKRHSKKKTKKKKLKKHKKGKAPAQQGIDSAVYASESLV
IFFIIIIIDIEKKKRDANNAMMHPKGKKKKIVKLYSKKTKQVKKKTSSDTSKKEQRRKRKQKKKLLINKSRFVG
GKQKKKKKKKTTDDRDQDDDDNDLDYDKTFFTTTATIIIEAEEEEQEEQQEEEQEEPAGKLIAGSGTISVIEFD
KKKKKKKKNKQQRRGGGGGLGKKKKKIKVKRRRKRRRRRQNSRQIQRQKQIRAADISSSKQKSQKKSSKKSSAA
SSSSSAIKEEAAALAEEEEERGGVPVQYGDADKTKKLFFAAAANILIIIIINANNNDEQRRMRKIIIGIYISEK
STAFGGGIIGGIYMTDAIINNACMMEMEERRRLRRRVRRRRRRQQRYYRILKAEEEESAAKADAKLKKKKKKKK
KEEYEARFRMHEIEEDDEGDNQQIIIIIIIIIIISANQLTTFFRRLTFNIPMGNGGGISFFDDNNFKSSSKSKV
VNQKSQSPGKNSAKILISSAGARQGGGGDGLTTTFFKGSDNSQQTVSDDDDDDGDDDDVRDGLGLDITLQSSSA
QYYNAKKNNDNLQNALFFDDQQSDIAADDDDDKDRDAAAAAKRGPSIIFFIICFSKDDYSAAAAAAAAAAAAAA
MAEQREYPYIIKEIGTIIIVINIGLLGLINNDISYFFENKKKKKKKRKDKKKTKKVPKLLHQKIISGGKAAAAL
VKAAIAASSYGSGAAAAPAQQYSYAQYQNPVVKPVIIVFIDAADDDDDDDDDDDDDDLQPSYSDDDDSDEEEER
FKQIIIVIRDDDDTDDKKKLQDDKRKDGRSSSSSSSSSLFAQSSSIIPSTTTKNNGSGGSSSSSGSSSSSIVVQ
MQIIAAGLLRRDGRRLSYSEAKRNNYERRTKFFFCVIVRKRIIIRLIRRLPLIIILDGAEKSRAYYRRRRDERK
```

```
IFDKVVKDDDDNFVMMMMRYPHKIIIIIKKGKASNQQIIGNIIIITTLLSSTTTMIIIIAIIIIIGIGVGNNNN
NNNNNNNNNNYMAAQCWRKSDAIIISGYMMMMMMMLMSSKSSASSRSTSPSSSSSVDGSSNDVIYAQLLLFKLG
DQDNQDDDDDDDDNDD'
```

There are a couple of things to notice about the recreated sequence. The model likes to predict long sequences of the same amino acid (a few have been highlighted in yellow for ease). In the original sequence, none of these repeated sequences were present. In some repeated amino acid sequences, it looks like there is some semblance of alliterations (highlighted in red). Although there are only a few, the model was able to generate these patterns on only a fraction of the training data (we had the model generate a sequence length of 1200, but it was trained on more than 5 million characters! Also, the model was unable to generate chains, which would have been separated by "|" and whole proteins, which would have been separated by "+". This seems a little hand wavy, but if we had a model with more hidden states, trained for longer, and trained on more sequence data, who knows what the model might create.

Now comes the really cool part. The way our recurrent model predicts the next amino acid from a given sequence is based on the probability distribution over all possible amino acids. In this case, we have the 20 amino acids, and the special characters "|" and "+" to indicate chains and unique proteins. The next predicted amino acid is then the one with the highest normalized probability. Before having the model make a prediction, we can take the vector probabilities in the final layer, take the natural log of each element, and divide each element by a parameter called, temperature. The value of the temperature is any value greater than 0. The higher the temperature, the smaller those probabilities become, and the more random an amino acid prediction is. We then exponentiate and then normalize once again. We then draw randomly from this distribution and obtain a new vector of probabilities. We then take the amino acid or special character with the highest probability from this new vector. Written in python code we have:



```python
def sample(preds, temperature=2):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds.ravel(), 1)
    return np.argmax(probas)
```

Figure 32. Temperature function

We showcase a few generated sequences for different temperature values. For a temperature value of 0.5:

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKSSSSSSSSQKKKQQQQERQKRLGRPVVVIVDDYYKKKKGDDDD
DDDDGNEKRIYVKKKKKKRKKKKKYVKKRKKKKKVLLAEPEWFDNINPPIKLPLLKKKTGPQQQQSSASASSSF
SSSSSSSSISSSSGSALSGGSRSSSSSSSSVVAAAAAAASSSSSSSSCVSYYRRRRRRRRRRRRRRRRRRGIAD
GAAAAQRTTSSTLTYRSSSYSKSQQSSSFFLIQNAANNNNNANNAADDDDRDGGTKNINNKALQKKKKKKKKKK
KKKKKKKSQIQRRKRRRRR|RSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
```

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSS'
```

Temperature value of 1.0

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKDSNNNNNNKKKKKHHGKVIGAIAIIIIILIRWRVVVVVVNSVGV
RGGGYKKGEQDKRKKKKMMIFAQIQLGNNYIEIEISEVVVYDDYVYVYERRYKKRRKLKERRRRRRRRHRH+HFA
SKSSISSSGGQSAASASSSSSSSSSSSSTAFIARGIIIIIIIYYADAYSANYLLEAFFTGTTTKNAFFVFRNVVR
RGIIQQNKKGDDIIALLNKSGGNGNNRRRTRQYQLFFKADADAAFFFFFFILKKAIAAIAPNAIRQIIKIKKGV
IAKDKNINKTLYKKLGLQNNNKNKNNNDNQESEFFKKFKFLFGRVRRRRRYRRKRKKKDDGRDAQQTALLLLLN
LFFLFIFKKKNKTKKKNSSPPSSSSQQAASASASSSILVQQQQEQREEKKKNERRRRRRRRRRRRRRRLLIIII
IIKQKIIKKLKEAKRKRKKKKRKRKKEKKKEKKQERK|RXSXXXXMXXXALMXXXXXXXXXXXXXXXXMWWC++
XXXXXXXXX+XX+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXMXDWXCCWWWFYRFCCCCCC
EYCYKERVYCVHHHKKQQAISKIAIQIIAAIAAAALIGRSSTITIIIIIIIIIINDLLFTSRSSSNQNKSSSSSS
SSSSSGSSSLLIDSDSTSASFFSSRRSSSSSSSSSSSSMSPPLFEETDEVLHNHGGVKKKKKKKKKKKKKKKKQI
ITTSSSIIDPMVAEYAYYYRRKKKKYKNYKGKGAAYYYVVVVVVVFQQDDIIDDKNDANDDNADPDDDDKAAALK
KKSKRRKVFGQADGGIIKAGGGGGPGKGKLDKKNNTISSFFSAVQAQNNANQNLNDDNNDDNNGRRPGKNNIVN
INTIKNISQQALLYYLYWGGEGEGGGGGEGGEEAGADLKKFKVVRQNIRKTIKNKNNKKKNKKNNPPAGLGGGG
KKFVSSSAGKQSIAGAPAAAASGSGSDDAIDSDQFFSSTIQLYSEEKKPQDVVKKDKEKDDDDDDDYYYRRRYY
LRRKKQQRKEEDEIEKEDGADTDKYYYYKYYLKKKYYYFFANFEFVFVQRRLMIYPKDDDSIIIIITIKKDDDQ
S|ADNAAAASSAAAAA'
```

Temperature value of 2.0

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKGKGDSVKNNNTNFRLVQMKKQKKQRIQIKKQEDPVKSIYKKGN
TVVGFVLLNQIKVTKEVIQKKKKNAQYNPDEADSPERETKPIKLLRKFFIIFVRSKQQGGANALGDGKGNNTKD
DGADQDDGDIFQSQQRSSTNGTGGGQAGILLNSYDFKQNVIDANADDMNNNNPRDGNDANSLLQNNNRKSTVFF
IIANQLGNKDQKNDTQDDDDLNADAQGRQSKMFFIIQTQISNIVQNNPQQNTQQKVSAQQQQDAFFELEDRQDQ
GGGARGVKRWGDGINRAKKKKYIKGTTPIKHYTTYYHVMSFADGDQQISAAASAAALNNKKKKDRQRPVGGTTY
FLDGALNIFNNNDNNDNVNLANLERRKENQRGKYKSWEQRGRRKKSGTMLLFIV|KEPDDDDDGVNQNSLRSSS
SLLNSVLVDSEYFFRQRKRRRTKPGQRATPIKKIAKNLVVDDNKTLKYGREYKQKKEYHKANKQLLIIKKKKAS
FVITKDDKTKQEQAGLRIKPNNINGQNNGGNYKKLFT+IGGAASAMALAGDSVNASSTNKGGNYEEVVAVFLNC
NVAVKLLRAIIIAKLCEPIIPHEGGSSEEGIAASIIYIAVIVVNVDNKKFFAKNGKRDVRTDQQDLLNNNNNDD
SDGDPDLYVYFLKAALGFCIQEKEMRRMKDRKVQRMVKRGYYNCEYKAKVLCYYIFRELYYQMNTKKPINDIII
ILGIKCMVCQMYLYQLYYQYYYKKEKGALKRRGDRRRFMIIEYSGYVDCVNRQRPYYVYKKKKRREQKIKKKKY
EITIIILI+GGTLFGTAEASENALRKINXISVVVSVVXLIIIIVVRXGVVVVVEDVVVGWVRIKGRRMRMIMVYL
CRNVYEYIAKKRRDKYQGKKYIGSGRYYRTRKTLQQLYYQKKSSANILPVSGYLLTADLLLRQKEFKVGKGLQT
LIADDIDKNNGIKKKSNNRRGVVAIQVQQAQPQVGAAGVVLVGKGTKKLGGGQPQGLQFPPGLLGLDGNNRYPS
LVTVKDDAQLVQICDLYDVVELTFIDDTFEGRRQVNKNRYPKLENNPNNSKSQVQLKLIFIAIDAIITAQELGE
ARSKGPNNQGGAGKKTKNGTGVGILGGGDLLLYYKNNNMMYNWAKGNYNLYRYLYFYYYYNKNYSLDQGTPIED
FLVTKHISSASIDIII'
```

Temperature value of 3:

```
'PIVQNLQGQMVHQAISPRTLNAWVKVVEEKDGNNITNTTKTK+RGVVLMKVVLVVLIASLCMRASQSSLGLGG
IISGQVQDGRTMRYAIRYGTLGRDCRSSSNLLEYYLWVYVDRRMVVVVVMIRRTRSSKKQKQHDNGIAGQGLYQ
QQYQQQF|SPSLDSSDDTDTTEAAGMSFSDLSTSSLDNDKDYVDNVYLIRNNSDVIGETKKKRKNKAKKWDRPK
YTYKTNFLGLLLLMHINSFDNKDTWKVKQNKDNKDRTKRADQQEKELRIKKQYSEKWKTIHAKKQESKLKTAIL
ELLYPNGSTDTDIFLRTVISRGKARRDVWAINPDQPKPDGGNPKARMRKLTINKIFNNNNNISNNKISKVSNST
PQPAPAPTDTKIMVSQYQYQYKLASESTELYKQREEDRNYPRIKYHLLFTLILLDLKKTRLGIIQTAVNADKDQ
NKSTGYLA+DIAQRALIAGKALQDSVLFFIAKNFYLGTLNNADNAINNNNPDQEDEIYAMIMEMIEEYKLVVSR
EKKKWEDGKGEADTYKNKNPNQLFKEIKTLSKPNSLQVALIMQNQQRQPQQGQTDVQYQDSQEKKLFNQKSCLK
LLALIVRMQKVPLYREADIRIEWYKFKKDGTNKY+GLLGGIGXYSXIKCRRIAK+XGPXVSSDXQKVSTLGXKV
XRVGETVVRVVVFV+ILLVPVIIAAIAGISENKNVGRRIQKLKVSGVIADARRQRILYAPKSYGASISIEXKYF
RIVECN+DKWVNRSNREKINKXRKIRVGKKKLEKXKFMCWVRPKDGE+HVXCAIEIVEKVMKVVIENIALVXTT
DWLGGMKRKVKDAEKLKXKNCQTEKSIYGSGFFPSRSVVRIAGY|FSAAALDDDDGDASAADDMGAANYEDXXD
XEWIRCKWWPSVVVVRWRWNKILVEMYGMVSAVADATGGANPKSKGQYRFNMFIHLSNQTIEDIFLIDIISVCK
KQQGSIIQAKVPRKLTYGQDNSLKKQLQRYELKTYLKFYSLVVKKILKRN|MKDNSKNNPLTSSRFXNMXXKVI
QSQIDMX+VIVVYQVAMMMLLVRAFVDNAANMVYSSANMLCIRSNRMRAGAATGALSAMQIADKLQFRMSYKAQ
QGVLPKG|DSSTASAQDNASDSANFEAEAKAANNLSSAIQDGTSPQGKKSSVELRILNKQPEFFNKGNKSSLKT
PSDAHQDDTSSTVILC'
```

The generated sequences for each temperature value are clearly different from one another and are clearly different from the original sequence. The generated sequence with a temperature of 0.5 starts to look ok after a few runs, but after a while it only predicts an S, Serine. At a temperature value of 1, the model generates a few chains! It even created multiple proteins. At temperatures 2 and 3, we get even more diversity. From these generated sequences, we could find similarities to other proteins in the PDB via BLAST (Basic Local Alignment Search Tool). It would be interesting to see if these generated sequences actually had homologies to other proteins in the PDB!

Conclusion

It would seem that Natural Language approaches to understanding amino acid sequences and proteins has merits. Convolutional neural networks seem to do well at classifying and protein class given its chain of amino acids. They also perform better than recurrent models like LSTM's and GRU's. We applied an embedding technique to learn amino acid sequence similarities in way that was similar to Google's Word2Vec and Stanford's GloVe word vectors. And we were able to apply recurrent neural network models to generate new sequences of amino acids to create new proteins. Although it is very unlikely that these generated sequences would fold to some kind of structure, it does open doors to the fields of protein engineering and proteomics. The methods shown here are more of a thought experiment and proof of concept.

Work Cited

1. https://en.wikipedia.org/wiki/Protein
2. https://www.biorxiv.org/content/10.1101/614313v1.full
3. http://map-catalog-gis3015.blogspot.com/2014/04/similarity-matrix-amino-acid.html
4. http://karpathy.github.io/2015/05/21/rnn-effectiveness/