Protein Fold Classification using Deep Neural Networks

Aaron Ogle
M00634800
ogle417@live.missouristate.edu

***Abstract*—The protein folding problem is an important problem in biology that involves finding a protein's three dimensional shape from its amino acid sequence [2]. A protein's structure determines how it functions and this information can be used for drug discovery and protein based biotechnology [3]. Inspired by the work done in [4], we will be looking into a related problem that involves classifying protein folds by using their amino acid sequence. In our work, we will investigate different deep learning models to see which model is able to classify a protein fold with the highest accuracy. We will be analyzing a 1D Convolutional Neural Network, a Bidirectional Long Short-Term Memory Recurrent Neural Network, and a combined 1D Convolutional Neural Network and Bidirectional Long Short-Term Memory Recurrent Neural Network; we will analyze these different neural network models to see which one is able to achieve the highest classification accuracy.**

I. Introduction

Classifying proteins by their fold structure provides information about the protein that is useful for understanding how they function [1]. Recently there has been ground breaking work done by DeepMind [4] that utilizes deep learning methods to predict a protein's 3D structure from its amino acid sequence; with their work, the "Protein Folding Problem" that has been an open problem for decades is now considered by some to be basically solved [2].

From [11] we can see that all proteins consist of sequences of different amino acids. There are twenty different amino acids that exist and are represented by a letter, these amino acids and letters are in the table below [11]. Scientists are interested in being able to classify proteins and protein folds based off their amino acid sequence [1] and are also interested in predicting a protein's 3D structure based on the amino acid sequence as well [4].

The protein folding problem has been an open problem for decades with early attempts being made to predict protein structures using computers beginning in the 1980s and 1990s [2]. Since research on this problem has been ongoing for decades there are a large number of available data sets and databases for amino acid sequences, one of such databases being the Structural Classification of Proteins - extended Database (SCOPe) [7], [8] which will be used in this work.

Research in predicting a protein's 3D structure from its amino acid sequence is being pursued in earnest due to the belief that diseases such as Alzheimers are the result of miss-folded proteins [10]. New drugs to treat diseases resulting from miss-folded proteins could be created by studying the structure of polypeptides [10]. Methods for determining a protein's three dimensional structure have involved firing x-ray beams at crystallized proteins and having the light that is diffracted be translated into a protein's atomic coordinates [2], this however, can be expensive and time consuming which is why research into utilizing deep learning methods as an alternative are being made.

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

Amino Acids [11]

Inspired by the work done in [4], we will be conducting research on a related problem; instead of using deep learning methods to predict a protein's 3D structure from its amino acid sequence we will be utilizing deep learning methods to classify protein folds by their amino acid sequence. Although this problem is different from the one that DeepMind addressed in [4], we can see from [1], [3], [5], and [6] that the problem of classifying protein folds is still an active area of research.

This paper will be organized as follows: Section II will provide related work that is being done on the Protein Fold Problem along with work being made on Protein Fold Classification. Section III will provide information on Convolution Neural Networks and Bidirectional Long Short-Term Memory Recurrent Neural Networks since these will be the models that are used for our protein fold classification. Section IV will provide information on our experimental design and data cleaning process. Section V will contain the architectures and descriptions of our neural network models. Section VI will

contain details on how we train our models along with the accuracy results of our models. Section VII will contain our conclusion and plans for further research on this problem.

II. Related Work

The authors in [3] create a deep learning network method that predicts if a given query-template protein pair are a part to the same structural fold. Using a related data set to the one that we are using in our work, SCOP 1.75, the authors are able to create a deep learning network that has a correct recognition rate of 84.5%, 61.5%, and 33.6% for Top 1 predictions [3]. In their work, the authors apply Deep-learning Networks to the protein fold recognition problem by treating the problem as a binary classification problem, i.e. the authors approach the problem by having their model predict if two proteins are from the same protein fold [3].

The authors create three different models in their work, DN-Fold, DN-FoldS, and DN-FoldR, where DN-Fold predicts if two proteins are from the same fold, DN-FoldS is referred to as a single model method by the authors, and DN-FoldR is a single linear regression model [3]. The authors used 14 different model architectures and then averaged all of the predictions for the models and the output of the ensemble of networks was the final prediction result of their method DN-Fold [3]. With their architectures and averaging method the authors are able to obtain a 84.5% correct recognition rate for Top 1 predictions and 91.2% for Top 5 showing that their method is effective when using a binary classification based approach for protein folds [3].

In [5] the authors create a 1D Convolutional Neural Network named DeepSF that is able to classify protein sequences into one of 1195 known folds which the authors note is useful for fold recognition and for studying sequence structure relationships [5]. Using the SCOP 1.75 dataset the author's model is able to achieve a classification accuracy of 80.4% [5].

The author's CNN architecture consists of an input layer that accepts the features of proteins of differing sequence lengths, 10 hidden layers of convolutions where each layer applies 10 filters to the windows of previous layers so that it can generate different hidden features [5]. Thirty maximum values from the hidden values of each filter at the tenth hidden layer are selected using max pooling which are then joined together into one vector by a flattening layer, the features in this vector are then passed to a fully connected hidden layer of 500 nodes which are then sent to an output layer of 1195 nodes to predict the probability of each of the different 1195 folds [5]. The authors use the softmax activation function for their output layer and then use the rectified linear unit activation function in their other layers [5]. With this architecture the authors were able to achieve a classification accuracy of 80.4% for top 1 predictions and 93.7% for top 5 predictions [5].

In [10] the authors apply a Long Short-Term Memory (LSTM) network to the protein folding problem along with creating an encoding scheme for protein fold states [10]. The LSTM network architecture was created to address the problem of error signals flowing backward in time in recurrent neural networks tend to explode or vanish, i.e. this method was created to avoid the exploding and vanishing gradient problem that can be found with using a recurrent neural network [10].

LSTM networks have memory cells that are hidden units that can lead to the model "remembering" inputs for a long time which allows for the gradient to flow longer thus avoiding the vanishing gradient problem [10]. LSTM networks are composed of three gates; a forget gate, update gate, and output gate, where the forget gate decides how much of the data it has previously learned will be forgotten, how much of that data will be used in the next step of the model, and the update gate decides what information should be thrown away or what new information should be added [10].

With their method the authors analyze the results for their mean absolute error and found that the LSTM model with 800 neurons improved the quality of their predictions and achieved a smaller MAE than the LSTM models with 200 and 400 neurons [10]. Since the authors are using their model for the protein folding problem they are comparing the results of their models spherical coordinates predictions with the protein's actual 3D structure, doing this they found their best model was able to obtain a mean absolute error of 0.110 when attempting to predict a protein's 3D shape [10].

III. Convolutional Neural Networks and Bidirectional Long Short-Term Memory Networks

Convolutional Neural Networks (CNN) are neural networks that rely on convolutions instead of matrix multiplication in at least one of their layers [12]. The authors of [6] note that CNNs are useful for sequences due to being able to capture patterns that are in the sequence [6]. In CNNs, the network relies on kernels of weights that have been learned, also called convolutional filters, that slide across the sequence so that it can discover patterns in the data [6].

Convolutional layers learn local patterns instead of global patterns, and the patterns that they learn are invariant meaning that the network will be able to recognize a certain pattern anywhere in the data [9]. For 1D CNNs, the network will extract local 1D segments from the sequences and will be able to recognize local patterns from that sequence [9]. From [9], we can see that 1D convnets offer a faster method than a recurrent neural network (RNN) for "simple" tasks.

The idea behind using a CNN layer along with a Bidirectional LSTM layer in our work is that the CNN layer will be able to learn these local patterns in the data and be able to pass that information into the Bidirectional LSTM layer. In [9] the author notes that combining a 1D convnet as a preprocessing step can be useful when dealing with long sequences as the CNN will turn the long input sequence into a much shorter one with higher level features that can be passed as input to the RNN portion of the network [9].

Long short-term memory networks are a type of gated RNN that are effective with sequence models [12]. RNNs are neural networks that introduce iteration loops that allow for information to travel from the output of the network to the

input of the network as well [6]. A typical RNN will use its own output from a previous iteration along with input from the current iteration to compute output; it is for this reason that recurrent neural networks are useful for sequential data processing since they are able to summarize sequences [6].

RNNs suffer from the exploding and vanishing gradient problem, to address this, Long Short-Term Memory (LSTM) networks were introduced to overcome this problem [10]. LSTM architectures allow for the network to carry information across many time steps [9]. An example of this is provided in [9] where they compare the network to a conveyor belt running next to the sequence that is being processed, the information can jump over to the conveyor belt at any point in time, which then allows for it to be transported to a later time step where it can "jump off" when it is needed, i.e. the LSTM network saves information for later which prevents older signals from vanishing [9].

Bidirectional RNNs are able to process sequences from both directions of the sequence, these networks consist of two RNNs such as a LSTM layer, where each of them processes the sequence in different directions [9]. Essentially, one of the RNNs will go from start to end of the sequence and the other RNN will start from the end and go to the start; after this, the networks will merge their representations. By using a bidirectional approach the network can catch patterns that could be overlooked from a traditional RNN [9].

IV. Data Cleaning, Exploratory Data Analysis, and Experimental Design

For our work we will be utilizing the SCOPe 1.55 data set which consists of amino acid sequence data and their fold classes. We will specifically be looking into alpha proteins. Additional information on the Structural Classification of Proteins (SCOPe) data set can be found in [7] and [8].

The SCOPe 1.55 data set, when filtered to just the alpha proteins, consists of 138 different fold classes. In our original proposal, our goal was to classify the sequences into each of these 138 different classes but found this to be impractical; further discussion of this will be provided below.

The SCOPe 1.55 data set consists of protein data that looks like the below:

¿d1dlwa_ a.1.1.1 (A:) Truncated hemoglobin Ciliate (Paramecium caudatum) slfeqlggqaavqavtaqfyaniqadat-
vatffngidmpnqtnktaaflcaalggpnawt
grnlkevhanmgvsnaqfttvighlrsaltgagvaaalveqtvavaetvrgdvvtv
¿d1dlya_ a.1.1.1 (A:) Truncated hemoglobin Green alga (Chlamydomonas eugametos) slfaklggreaveaavdkfynki-
vadptvstyfsntdmkvqrskqfaflayalggasewk
gkdmrtahkdlvphlsdvhfqavarhlsdtltelgvppeditdamavvastrtevln-
mpqq
.
.
.
¿d1d4ca1 a.138.1.3 (A:1-102) Flavocytochrome c3 (respiratory fumarate reductase), N-terminal domain
Shewanella putrefaciens apevladfhgemggcdschvsdkggvt-
ndnlthengqcvschgdlkelaaaapkdkvsph
kshligeiactschkgheksvaycdachsfgfdmpfggkwer

Taking a.1.1.1 as an example, the "a" denotes "Class a, all alpha proteins", the a.1 represents the fold (Globin-like), the a.1.1 represents the Superfamily (Globin-like), the a.1.1.1 represents the Family (Truncated Hemoglobin), the Truncated Hemoglobin is the protein, and the Ciliate (Paramecium caudatum) is the species [7], [8]. From the data we thus have the below hierarchical structure:

1) Class
2) Fold
3) Superfamily
4) Family
5) Protein
6) Species

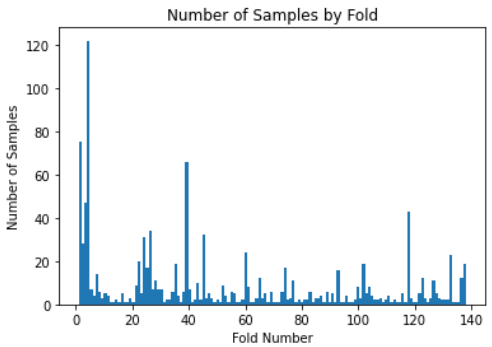
The information that we are interested in are the sequences themselves along with the coinciding fold class that they are a part of.

The first task that we had to overcome with this data set was to parse out the relevant information that we needed so that our deep learning model would be able to use it. Parsing the data involved different python methods such as split, partition, and find so that we could break the text into sequences and fold data. Additional data cleaning methods such as pop and delete were needed to be performed as well due to new line characters being brought in where they shouldn't be during the partitioning process.

Once the data had been parsed out we performed some exploratory data analysis to become more familiar with some of the statistical features of our data. While exploring the data originally, we found that the maximum length of a sequence was 904 characters, the minimum length was 29 characters, and the average sequence length was 151 characters.

After looking into this, we then looked into how many samples we had for each of the different fold classes. The quickest way to get an idea of this was to plot a histogram of the data to see the number of samples by the different fold classes. When doing this we obtain the below plot.

Fig. 1: Dataset Histogram



From this we can clearly see that some of our fold classes do not contain very many samples in our data set which will

make it incredibly difficulty for our models to learn enough information about those sequences. Exploring this further we found that around 40 of the fold classes only contained one amino acid sequence sample. After making this discovery we decided that instead of trying to classify the different amino acid sequences into the 138 different fold classes we would only investigate 10 different fold groups from which we have the most samples to work with.

Changing the problem to only look at the top 10 fold classes led to more data cleaning so that we could obtain a top 10 group of protein fold classes that we could work with. Upon further data exploration we found the below top 10 fold groups based off the number of sample sequences that we have in our SCOPe 1.55 data set, note that some of the names of the folds have been truncated:

Class Number and Fold	Number of Samples
4 - DNA/RNA-binding	122
1 - Globin-like	75
39 - EF Hand-like	66
3 - Cytochrome c	47
118 - alpha-alpha superhelix	43
26 - 4-helical cytokines	34
45 - Glutathione	32
24 - Four-helical	31
2 - Long alpha-hairpin	28
60 - SAM domain-like	24

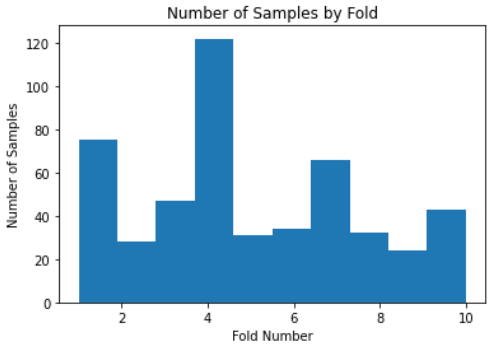
Due to condensing the data, we then relabelled the data so that it was in a more consistent format. We obtain the below labeling of our class labels after this:

Class Number Label	New Class Number Label
1	1
2	2
3	3
4	4
24	5
26	6
39	7
45	8
60	9
118	10

With this, we have a nicer set of data to work with than we had originally. From our condensed set of data our max sequence length is 904, minimum sequence length is 31, and the average sequence length is 131 characters. Since we have sequences of varying sizes, the next step in our data cleaning process is to put a max length to the sequences since our model will be unable to use sequences of varying sizes. We decide to truncate the sequences down to a max length of 200 characters.

Now that we have decided on a max length to use, we tokenize our data since our model will need to use numeric data types instead of character. We use the Tokenizer function

Fig. 2: Truncated Dataset Histogram



from the Keras library to tokenize our sequence data. Since some of our sequences have lengths that are less than 200 we will need to pad those sequences with zeros so that our model is able to work with sequences that are all length 200 - to do this we use Keras pad_sequences functionality.

For our experimental design, we need to have our data set split into groups for training and testing the models. Before we split the data, we first shuffle all of the sequences in our data set. The reasoning for the shuffling of the data is we have an ordered set of sequences so if we just split the data as is then we would be training our models on sequences that would not even exist in the test set, so we have to shuffle our data due to it originally being in an ordered structure.

Our dataset consists of a total of 503 sequences, we originally split the data into groups of 400 for training and 103 for testing which is close to a 80% and 20% split, respectively. Since we only have 503 sequences to work with, we wanted to give our models a few more samples to learn from so we increased the split to 420 samples for training and 83 samples for testing which results in a approximately 84% and 16% split, the reasoning for this is we noticed an increase in the accuracy when we give more samples to work with, we will discuss this further below in our experimental results section.

To measure our models, we will perform two sets of experiments. First, we will look into when we split the data into 80% for training and 20% for testing. We will run each of our models five times and record the model's accuracy. For our second set of experiments we will look into when we split the data into 84% for training and 16% for testing. We will record the same metrics as stated in the first experiment for this split as well.

V. Neural Network Architectures

For our CNN model, we use only one 1D conv layer with 128 filters and a kernel size (convolution window) of 20; the reasoning behind using 20 is due to having 20 characters in an amino acid sequence. The conv layer uses the ReLU activation function as well. After the conv layer, the resulting output is flattened and then passed to a fully connected layer that contains 1024 neurons using the ReLU activation function. After this, the signal is passed to the output layer where the softmax activation function is used.

For our Bidirectional LSTM model we use a similar architecture as that used in [10]. The network consists of a fully connected dense layer with 100 neurons that passes the signal to the Bidirectional LSTM layer which then has the output flattened and passed to another FC layer with 100 neurons that uses dropout with probability 20% - this output is then passed to another FC layer with 100 neurons that also uses dropout with probability 20%, and the signal is then passed to the output layer that uses the softmax activation function.

Our final model has a 1D conv layer with 128 filters and a kernel size of 20, this is the same as our first model. After this layer, the signal is then passed to a Bidirectional LSTM memory layer which then flattens the output and sends the result to a dense fully connected layer of 1024 neurons. This layer then passes the output to the final layer which uses the softmax activation function.

VI. Model Training, Validation, and Experimental Results

For training and validating our model we split the data that we have into training and testing sets as described above. For each of our models we use RMSprop as our optimizer as this was the one used in [10] and provides good results with the type of data we are working with. For our loss function, since we have multiple classes to work with, our models use the sparse categorical crossentropy loss function. In order to prevent overfitting, each of our models use the early stopping technique with a patience of 5 meaning that if the highest accuracy result is seen 5 times then the training is stopped. Each of our models use this early stopping technique and are also trained for 50 epochs.

Our experiments consist of a 80% and 20% split along with a 84% and 16% split for training and testing the data. We ran each of the models five times and recorded their accuracy for each run. The purpose of running the models multiple times is due to the data set being shuffled before it is split, by running it five different times we can see how the results changed based off the initial data set being shuffled before it is being split into groups for training and testing. The accuracy results for both the 80% and 20% split along with the results for the 84% and 16% split are provided in the tables below. We also provide plots for the accuracy and loss of our models that achieved the highest classification accuracy - these plots can be found after our references section.

80% and 20% Data Split			
Run	CNN	RNN	CNN/RNN
1	65.69	72.55	74.51
2	67.65	66.67	73.53
3	62.75	67.65	71.57
4	58.82	69.61	70.59
5	64.71	62.75	68.63

84% and 16% Data Split			
Run	CNN	RNN	CNN/RNN
1	71.95	74.39	78.05
2	65.85	65.85	74.39
3	62.2	73.17	68.29
4	64.63	69.51	65.85
5	64.63	59.76	74.39

From our results, we can see that when we split the data into a 84% and 16% split that we are able to obtain a higher classification accuracy and that our model that is able to classify with the highest accuracy is the combined architecture model.

We can infer from the fact that our model's accuracy increases when more data is given to them that each of our models would benefit from a larger data set instead of one that consists of only a little over 500 examples to work with in total. We can also see from our results that the rankings for the model goes from CNN being in last, RNN being in second, and then the combined architecture being in first. This shows that using a combined architecture allows for the conv layer to extract the most useful features of which it can then pass to the RNN layer portion which helps lead to higher classification accuracy with the data that we are working with.

VII. Conclusion

Being able to classify proteins by their amino acid sequence is a very important problem and we can see that deep learning is a very powerful method that can be used to approach this problem. In this work we first addressed the current work that is being done on the protein fold problem. We then attempted to work on a related problem which is to classify a protein fold by its corresponding amino acid sequence. In our work we built three different models and found that the model which combined a CNN and RNN architecture led to the highest classification accuracy.

For future research work we would like to work with a larger data set than the one that was used in this work to see if this helps increase the accuracy of the models. We would also like to look into how generative adversarial networks could be applied to the protein folding problem to see if these models might be able to help with creating either more data to work with or if they would be able to be used to classify the data itself directly.

References

[1] Sayeed, Suri Dipannita and Wolf, Jan and Koch, Ina and Song, Guang. (2022). Protein Fold Classification using Graph Neural Network and Protein Topology Graph. 10.1101/2022.08.10.503436.

[2] Callaway. (2020). "It will change everything": DeepMind's AI makes gigantic leap in solving protein structures. Nature (London), 588(7837), 203–204. https://doi.org/10.1038/d41586-020-03348-4

[3] Jo, T., Hou, J., Eickholt, J. et al. Improving Protein Fold Recognition by Deep Learning Networks. Sci Rep 5, 17573 (2015). https://doi.org/10.1038/srep17573

[4] Jumper, J., Evans, R., Pritzel, A. et al. Highly accurate protein structure prediction with AlphaFold. Nature 596, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2

[5] Jie Hou, Badri Adhikari, Jianlin Cheng, DeepSF: deep convolutional neural network for mapping protein sequences to folds, Bioinformatics, Volume 34, Issue 8, 15 April 2018, Pages 1295–1303, https://doi.org/10.1093/bioinformatics/btx780
[6] A. Villegas-Morcillo, A. M. Gomez, J. A. Morales-Cordovilla and V. Sanchez, "Protein Fold Recognition From Sequences Using Convolutional and Recurrent Neural Networks," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 18, no. 6, pp. 2848-2854, 1 Nov.-Dec. 2021, doi: 10.1109/TCBB.2020.3012732.
[7] Fox NK, Brenner SE, Chandonia JM. 2014. SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures. Nucleic Acids Research 42:D304-309. doi: 10.1093/nar/gkt1240.
[8] Chandonia JM, Guan L, Lin S, Yu C, Fox NK, Brenner SE. 2022. SCOPe: Improvements to the Structural Classification of Proteins—extended Database to facilitate Variant Interpretation and Machine Learning. Nucleic Acids Research 50:D553–559. doi: 10.1093/nar/gkab1054.
[9] Chollet, F. (2017). Deep learning with Python. Manning Publications.
[10] L. T. Hattori, C. M. V. Benitez, M. Gutoski, N. M. R. Aquino and H. S. Lopes, "A Novel Approach to Protein Folding Prediction based on Long Short-Term Memory Networks: A Preliminary Investigation and Analysis," 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489514.
[11] Smith, Y. (2019, February 26). Amino acids and protein sequences. News. Retrieved November 9, 2022, from https://www.news-medical.net/life-sciences/Amino-Acids-and-Protein-Sequences.aspx
[12] Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

VIII. Plots of Model Accuracy and Loss

Fig. 3: CNN Model Accuracy

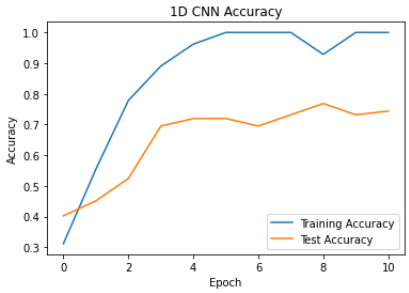


Fig. 4: CNN Model Loss

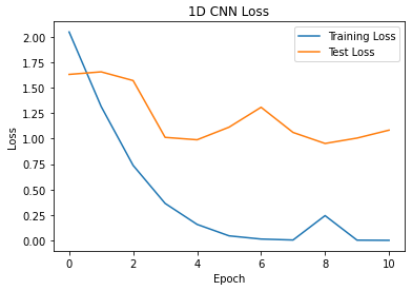


Fig. 5: RNN Model Accuracy

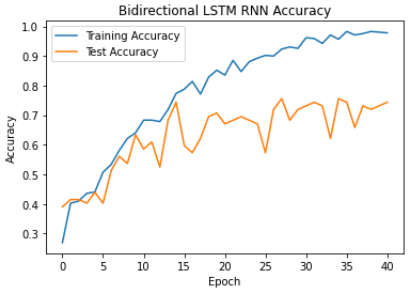


Fig. 6: RNN Model Loss

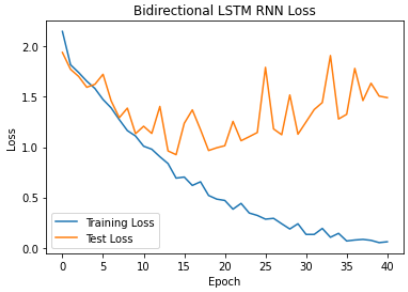


Fig. 7: CNN and RNN Model Accuracy

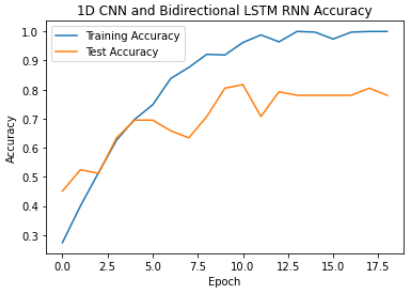


Fig. 8: CNN and RNN Model Loss