CS6002
Data Mining

Classification of Family Domain of Amino Acid Sequences using CNN – LSTM Architecture

Steven F. Gilbert
2018103071

Gokul S
2018103026

Table of Contents

1 Introduction

1.1 Overview

As diseases become more and more resistant and adaptive, protein classification is crucial to identify and create cures or vaccines for the diseases. Proteins interact with other macromolecules, playing a central role in many biological processes. Investigating protein function often involves structural studies or biochemical studies, which require time consuming efforts. So, classifying a protein in a virus or such can help researchers to determine the basis of the disease, which can in turn elucidate methods to prevent and treat the disease. Other approaches to classify proteins using neural networks have been proposed, but we take a different approach by using an LSTM for the main purpose of identification. LSTMs have an advantage over RNNs in that they do not suffer from the vanishing gradient problem. LSTMs are very good at processing long sequences without losing any gradient value. Coupled with this a CNN to deep-extract the features present in the protein sequence. The dataset used here is based on real-time data of researchers, who conduct various tests on proteins and the results are archived together as a whole dataset.

1.2 Problem Statement

Protein classification is an important part of modern biotechnology as proteins are the macromolecules which play a major role in the biological processes of a cell. From DNA sequences we also know the amino acid sequences of proteins, which are the fundamental molecules that perform most biological functions. The study of proteins are crucial as different and new diseases can be studied from the proteins they are made up of and new drugs can be synthesized based on the proteins to come up with a cure. The functionality of a protein is thus encoded in the amino acid sequence and understanding the sequence-function relationship is a major challenge in bioinformatics. Protein families are defined to group together proteins that share similar functions. Moreover, classifying a type of protein can bring insight to its functional properties.

All the modules of the project have been coded and the model has been trained and tested, along with the calculation of performance metrics. As far as fine tuning is concerned, small changes have been updated to the final model to improve accuracy.

1.3 Dataset

The PFAM database is a collection of families of protein domains.[1] Proteins comprise of multiple functional regions known as domains. Different domains can in varying combinations can produce a diverse category of proteins. In this dataset posted by Maxwell Bileschi et al on Kaggle called the seed random split[2,3]. This data set was provided by Maxwell Bileschi et al by taking the highly curated protein families and splitting each family with at least 10 seed sequences. This means that only all the protein sequences available in this dataset correspond to a single domain. The dataset consists of the sequences of amino acids along with other metadata such as sequence name, aligned_sequence which contains a single sequence from the multiple sequence alignment, family_id and family_accession. Family accession is used here to identify the different families within the domain and is of the form PFxxxxx.y, where xxxxx is the family accession, and y is the version number.

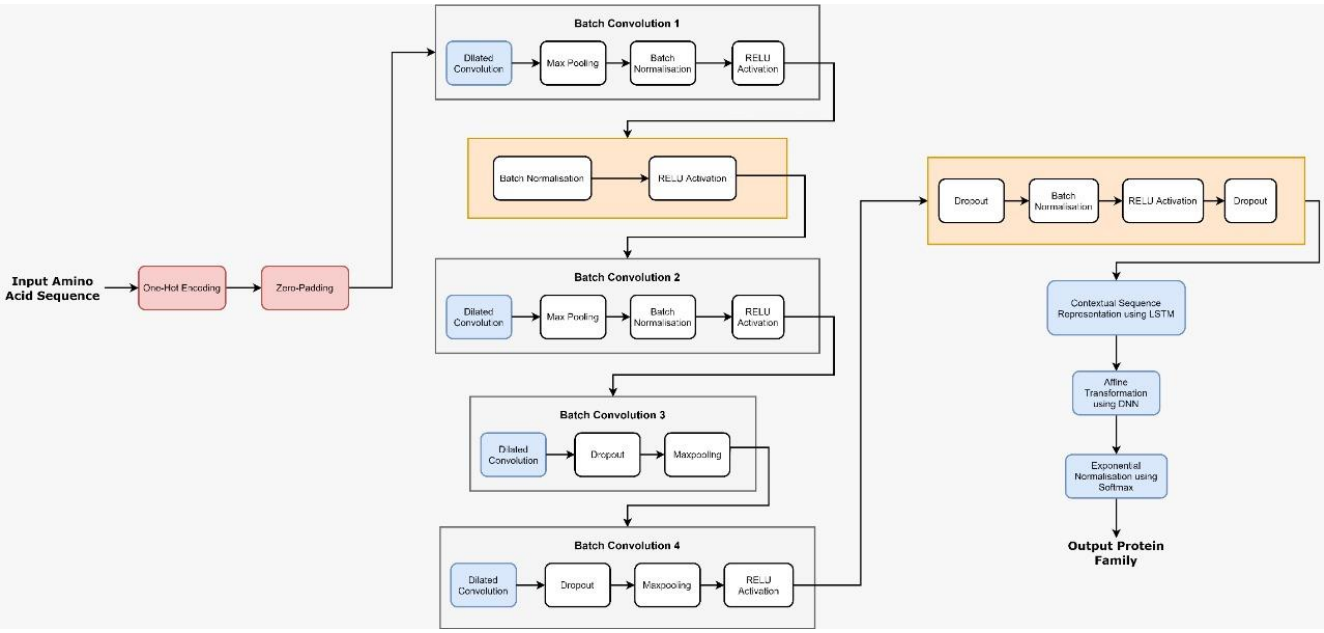
This dataset consists of over 1 million sequences with 18,000 unique families.

Label	**Data**
sequence	HWLQMRDSMNTYNNMVNRCFATCI RSFQEKKVNAEEMDCTKRCVTKFVG YSQRVALRFAE
family_accession	PF02953.15
sequence_name	C5K6N5_PERM5/28-87
aligned_sequence	....HWLQMRDSMNTYNNMVNRCFAT CI.....RS.F....QEKKVNAEE.....MDCT ....KRCVTKFVGYSQRVALRFAE
family_id	zf-Tim10_DDP

2 Model Summary

```
==================================================================================================
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)           [(None, 100, 21)]    0           []

conv1d_1 (Conv1D)              (None, 100, 32)      704         ['input_1[0][0]']

max_pooling1d_6 (MaxPooling1D)  (None, 50, 32)      0           ['conv1d_1[0][0]']

batch_normalization_1 (BatchNo  (None, 50, 32)      128         ['max_pooling1d_6[0][0]']
rmalization)

activation_1 (Activation)      (None, 50, 32)       0           ['batch_normalization_1[0][0]']

batch_normalization_2 (BatchNo  (None, 50, 32)      128         ['activation_1[0][0]']
rmalization)

activation_2 (Activation)      (None, 50, 32)       0           ['batch_normalization_2[0][0]']

conv1d_3 (Conv1D)              (None, 50, 128)      4224        ['activation_2[0][0]']

batch_normalization_3 (BatchNo  (None, 50, 128)     512         ['conv1d_3[0][0]']
rmalization)

activation_3 (Activation)      (None, 50, 128)      0           ['batch_normalization_3[0][0]']

conv1d_4 (Conv1D)              (None, 50, 128)      16512       ['activation_3[0][0]']

conv1d_2 (Conv1D)              (None, 50, 128)      4224        ['activation_1[0][0]']

d3 (Dropout)                   (None, 50, 128)      0           ['conv1d_4[0][0]']

d7 (Dropout)                   (None, 50, 128)      0           ['conv1d_2[0][0]']

max_pooling1d_7 (MaxPooling1D)  (None, 25, 128)     0           ['d3[0][0]']

max_pooling1d_8 (MaxPooling1D)  (None, 25, 128)     0           ['d7[0][0]']

add_2 (Add)                    (None, 25, 128)      0           ['max_pooling1d_7[0][0]',
                                                                 'max_pooling1d_8[0][0]']

activation_4 (Activation)      (None, 25, 128)      0           ['add_2[0][0]']

dropout_2 (Dropout)            (None, 25, 128)      0           ['activation_4[0][0]']

batch_normalization_4 (BatchNo  (None, 25, 128)     512         ['dropout_2[0][0]']
rmalization)

activation_5 (Activation)      (None, 25, 128)      0           ['batch_normalization_4[0][0]']

dropout_1 (Dropout)            (None, 25, 128)      0           ['activation_5[0][0]']

lstm_1 (LSTM)                  (None, 256)          394240      ['dropout_1[0][0]']

flatten_1 (Flatten)            (None, 256)          0           ['lstm_1[0][0]']

fc5832 (Dense)                 (None, 5832)         1498824     ['flatten_1[0][0]']

activation_6 (Activation)      (None, 5832)         0           ['fc5832[0][0]']

==================================================================================================
Total params: 1,920,008
```

3 Model Architecture



4 Model Parameters

Parameter	**Value**
Epochs	50
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Cross Entropy
Batch Size	256

4.1 Layers used

1) Dilated CNN 1D (Filters = (64,256), Kernel = 5, Activation Function = ‘relu’)
2) Dropout (CNN = (0.3,0.5), Dense = 0.5)
3) MaxPooling (Pool size = 3)
4) LSTM (Units = 100)
5) Dense (Filters = 128, Kernel Regularizer = ‘l2’, Activation Function = ‘relu’)

6) Output (Filters = 10, Activation Function = ‘softmax’)

Early stopping has been added to prevent the model from overfitting beyond a point. It is set to monitor the ‘validation loss’ to minimum. The patience is set at 10 epochs.

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',verbose=1,patience = 10,
                           mode='min', restore_best_weights=True)
```

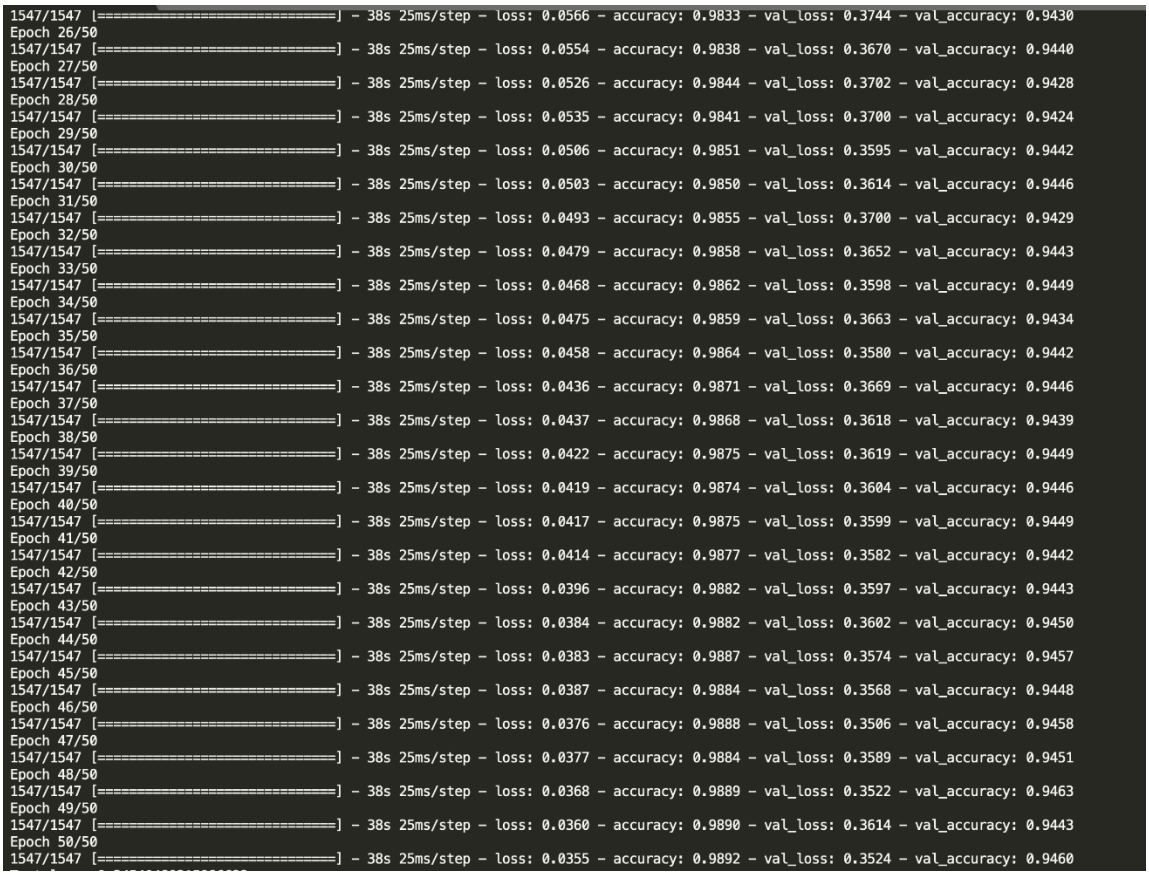
5 Training

5.1 Model Compilation

```
from tensorflow.keras.optimizers import Adam
opt = Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['acc'])
```

5.2 Model Fitting

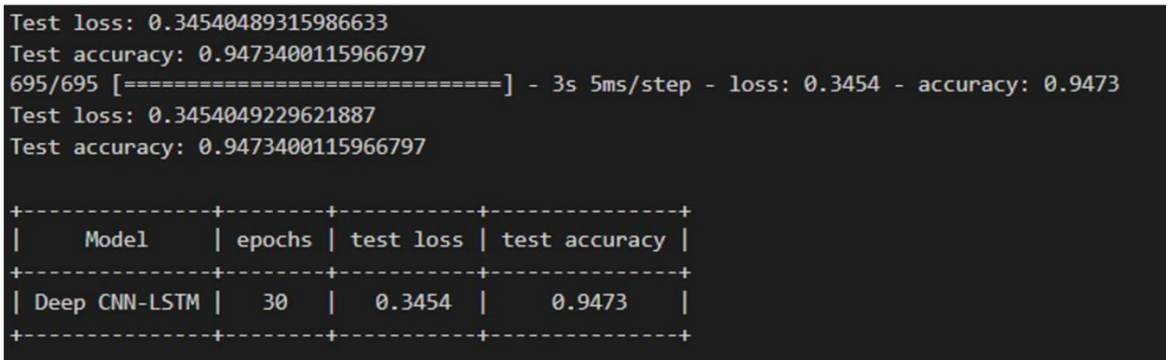
```
Epoch 1/50
2022-01-03 13:41:07.770377: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8100
1547/1547 [==============================] - 42s 25ms/step - loss: 3.2222 - accuracy: 0.5456 - val_loss: 1.5441 - val_accuracy: 0.8468
Epoch 2/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.9284 - accuracy: 0.8300 - val_loss: 0.6729 - val_accuracy: 0.9008
Epoch 3/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.5524 - accuracy: 0.8894 - val_loss: 0.5044 - val_accuracy: 0.9170
Epoch 4/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.3835 - accuracy: 0.9170 - val_loss: 0.4397 - val_accuracy: 0.9246
Epoch 5/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.2864 - accuracy: 0.9341 - val_loss: 0.4133 - val_accuracy:
0.9288
Epoch 6/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.2300 - accuracy: 0.9440 - val_loss: 0.3993 - val_accuracy: 0.9333
Epoch 7/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1875 - accuracy: 0.9528 - val_loss: 0.3879 - val_accuracy: 0.9345
Epoch 8/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1609 - accuracy: 0.9581 - val_loss: 0.3842 - val_accuracy: 0.9366
Epoch 9/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1435 - accuracy: 0.9620 - val_loss: 0.3853 - val_accuracy: 0.9360
Epoch 10/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1289 - accuracy: 0.9654 - val_loss: 0.3810 - val_accuracy: 0.9373
Epoch 11/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1165 - accuracy: 0.9682 - val_loss: 0.3835 - val_accuracy: 0.9373
Epoch 12/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.1077 - accuracy: 0.9701 - val_loss: 0.3775 - val_accuracy: 0.9389
Epoch 13/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0994 - accuracy: 0.9723 - val_loss: 0.3782 - val_accuracy: 0.9388
Epoch 14/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0950 - accuracy: 0.9733 - val_loss: 0.3694 - val_accuracy: 0.9404
Epoch 15/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0872 - accuracy: 0.9755 - val_loss: 0.3738 - val_accuracy: 0.9405
Epoch 16/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0827 - accuracy: 0.9763 - val_loss: 0.3747 - val_accuracy: 0.9402
Epoch 17/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0780 - accuracy: 0.9776 - val_loss: 0.3782 - val_accuracy: 0.9397
Epoch 18/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0739 - accuracy: 0.9787 - val_loss: 0.3754 - val_accuracy: 0.9407
Epoch 19/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0708 - accuracy: 0.9797 - val_loss: 0.3767 - val_accuracy: 0.9424
Epoch 20/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0691 - accuracy: 0.9801 - val_loss: 0.3749 - val_accuracy: 0.9417
Epoch 21/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0660 - accuracy: 0.9807 - val_loss: 0.3727 - val_accuracy: 0.9425
Epoch 22/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0648 - accuracy: 0.9813 - val_loss: 0.3647 - val_accuracy: 0.9431
Epoch 23/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0614 - accuracy: 0.9819 - val_loss: 0.3696 - val_accuracy: 0.9429
Epoch 24/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0599 - accuracy: 0.9826 - val_loss: 0.3726 - val_accuracy: 0.9429
Epoch 25/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0566 - accuracy: 0.9833 - val_loss: 0.3744 - val_accuracy: 0.9430
Epoch 26/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0554 - accuracy: 0.9838 - val_loss: 0.3670 - val_accuracy: 0.9440
Epoch 27/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0526 - accuracy: 0.9844 - val_loss: 0.3702 - val_accuracy: 0.9428
Epoch 28/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0535 - accuracy: 0.9841 - val_loss: 0.3700 - val_accuracy: 0.9424
Epoch 29/50
1547/1547 [==============================] - 38s 25ms/step - loss: 0.0506 - accuracy: 0.9851 - val_loss: 0.3595 - val_accuracy: 0.9442
Epoch 30/50
```



Model stopped at 50 epochs and restored to the best weights with the minimum validation loss.

6 Testing

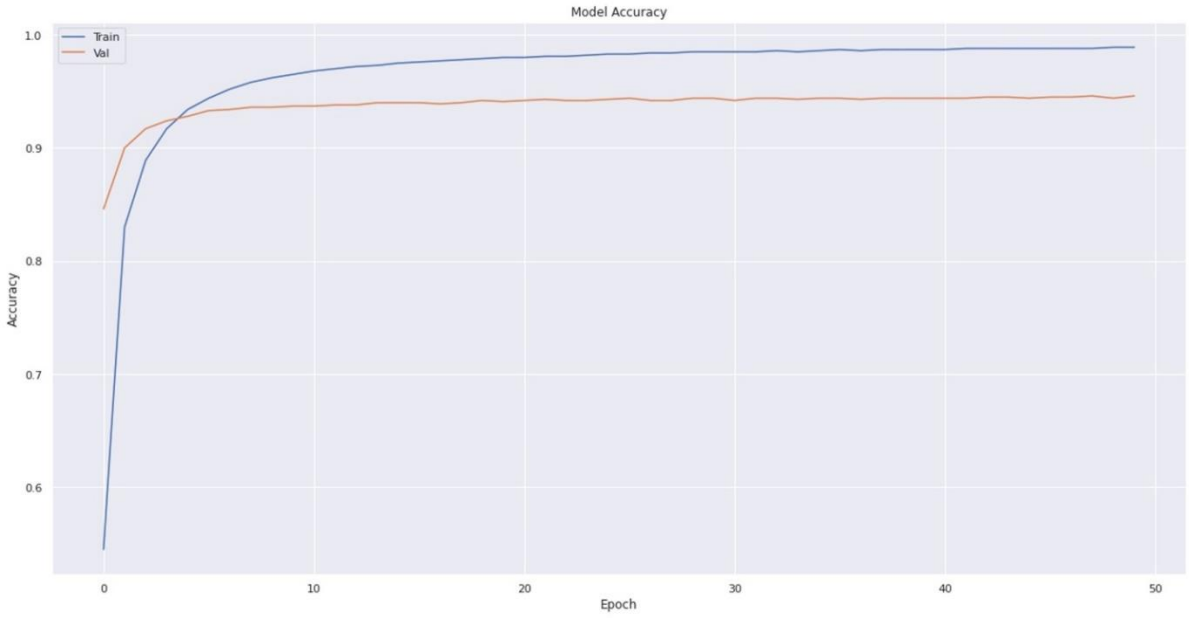
The model is tested with the test data with the same batch size of 256.



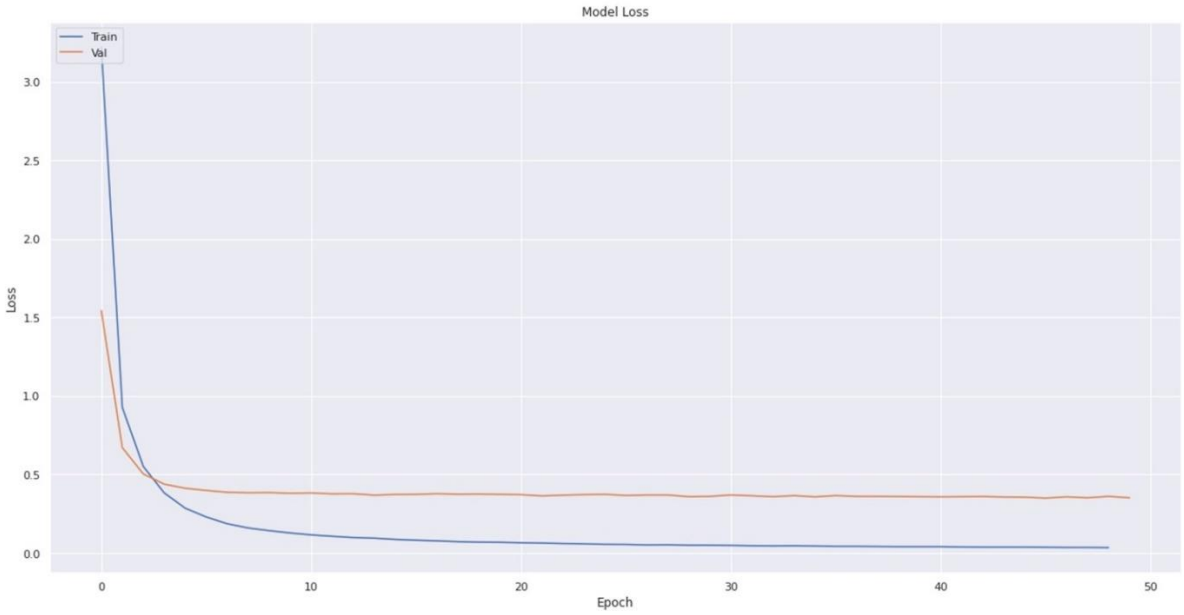
The model has 94.73% accuracy and loss of 0.3454

7 Graphs

7.1 Accuracy Curve



7.2 Loss Curve

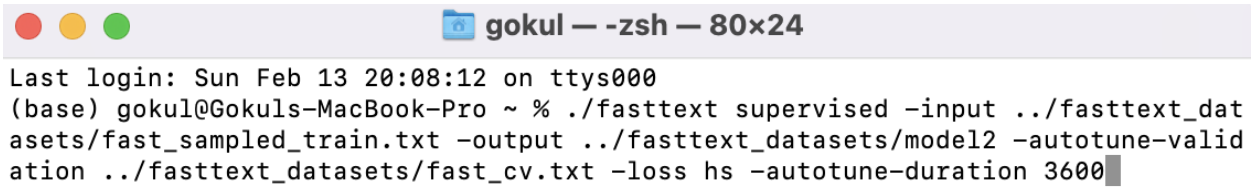


9 Classification Report

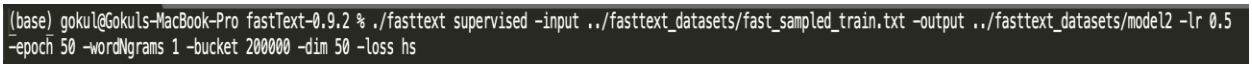
```
Test loss: 0.34540489315986633
Test accuracy: 0.9473400115966797
695/695 [==============================] - 3s 5ms/step - loss: 0.3454 - accuracy: 0.9473
Test loss: 0.3454049229621887
Test accuracy: 0.9473400115966797

+---------------+--------+-----------+---------------+
|     Model     | epochs | test loss | test accuracy |
+---------------+--------+-----------+---------------+
| Deep CNN-LSTM |   30   |   0.3454  |     0.9473    |
+---------------+--------+-----------+---------------+

Classification Report
              precision    recall  f1-score   support

        5809       0.00      0.00      0.00         1
        5810       0.00      0.00      0.00         1
        5811       1.00      1.00      1.00         1
        5812       0.00      0.00      0.00         1
        5813       1.00      1.00      1.00         1
        5814       1.00      1.00      1.00         1
        5815       0.00      0.00      0.00         1
        5816       1.00      1.00      1.00         1
        5817       1.00      1.00      1.00         1
        5818       0.00      0.00      0.00         1
        5819       1.00      1.00      1.00         1
        5820       0.00      0.00      0.00         1
        5821       1.00      1.00      1.00         1
        5822       1.00      1.00      1.00         1
        5823       0.00      0.00      0.00         1
        5824       1.00      1.00      1.00         1
        5825       0.00      0.00      0.00         1
        5826       0.00      0.00      0.00         1
        5827       0.00      0.00      0.00         1
        5828       1.00      1.00      1.00         1
        5829       1.00      1.00      1.00         1
        5830       0.00      0.00      0.00         1
        5831       0.00      0.00      0.00         1

    accuracy                           0.95     22218
   macro avg       0.92      0.91      0.91     22218
weighted avg       0.95      0.95      0.94     22218
```

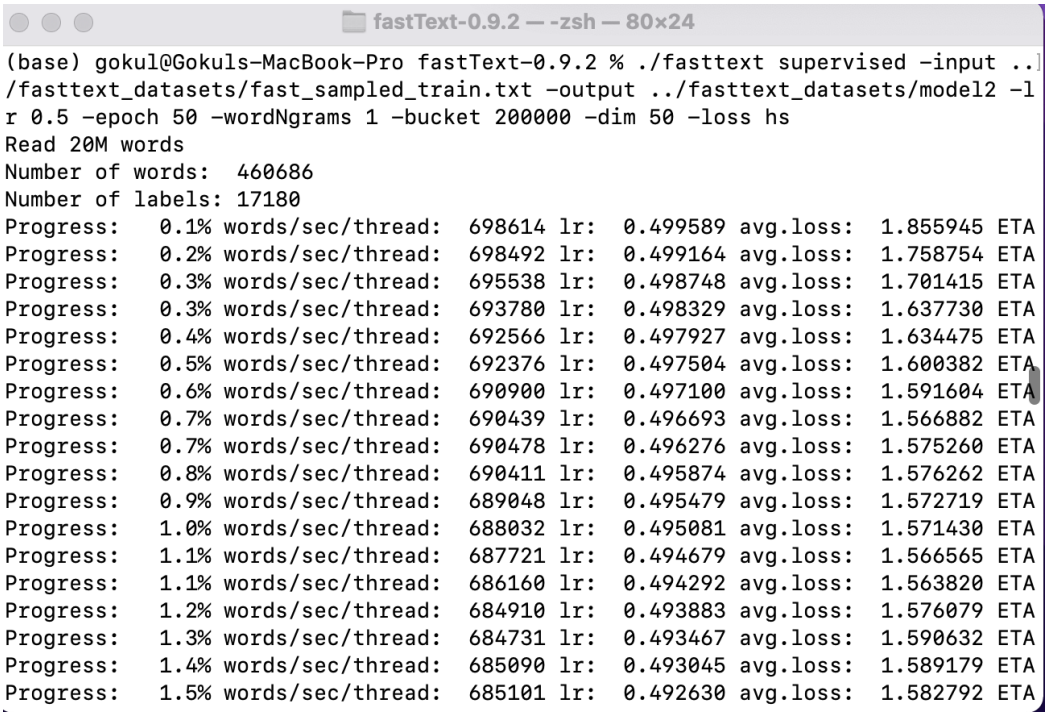
10 FastText Model:

10.1 Autotuning to find optimal model parameters:



10.2 Training:





10.3 Testing & Predictions:

11 Comparison

ProtCNN Model	FastText Model	Deep CNN-LSTM Model
Accuracy = 0.73 F1Score = 0.68	F1 Score = 0.9071	F1 Score = 0.94 Accuracy = 0.9473

11.1 Experimental Results

Taking a close look at the dataset, there are 17,929 unique family accessions.

After the preprocessing of our data, we took the dedicated train, test and validation datasets for further training. This is given by train – 500,000 and test – 25,000 and validation – 25,000 datapoints. The data is also shuffled so as to reduce variance and to make the model more general and to avoid any overfitting.

The hyperparameters used here are given below:

Parameter	**Value**
Epochs	50
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Cross Entropy
Batch Size	265

11.2 Results

The model took about 9 hours for preprocessing (cleaning, one-hot encoding & Count Vectorising) 4 hours to train with a training accuracy of 94.73% and loss of 0.3454 with validation accuracy of 94.64% and loss of 0.3524. Other metrics for performance such as Precision, Recall and F1 - score have been calculated. (The given metrics are calculated as **weighted average of all classes**).

Note: The time stated above is on an Nvidia Quadro P5000 16 GB RAM GPU from the Department of CSE, College of Engineering Guindy, Chennai, India
GPU Specifications:
Cuda – 11.2
CuDNN – 7.0.1

Classes	**Accuracy %**	**Precision**	**Recall**	**F1-score**
17,929	95.00	0.95	0.95	0.94

Model	**Classes**	**Error Rate %**	**Number of Errors**
ProtCNN by Maxwell Bileschi et al[3]	17,929	0.159	201
Blastp[3][4]	17,929	1.645	2087
RNN (LSTM)	100	0.63	219
Proposed Model	**17,929**	**0.52**	**180**

Our proposed model performs better than the BLASTP[4] model which is based on a Basic Local Alignment Search Tool Query[5] which has a much higher error rate than our proposed model.

We performed the same test on a LSTM model with the same parameters and the same dataset and that model resulted in a much higher error rate than the proposed model.

12 References

1. Base Paper:”Using Deep Learning to Annotate the Protein Universe”, https://www.biorxiv.org/content/10.1101/626507v4, Google AI, MIT CSAI Laboratory

2. http://pfam.xfam.org/

3. https://www.kaggle.com/googleai/pfam-seed-random-split

4. Bileschi, Maxwell L., David Belanger, Drew Bryant, Theo Sanderson, Brandon Carter, D. Sculley, Mark A. DePristo, and Lucy J. Colwell. "Using deep learning to annotate the protein universe." bioRxiv (2019): 626507.

5. https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE=Proteins

6. Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. Journal of molecular biology, 215(3):403–410, 1990.