Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Name: Nushrat Tarmin Meem

Roll: **1907083**

Year: 4th

Semester: 2nd

Course: BME 4241

Assignment Topic: Protein 2D Structure Prediction Using Deep Learning

Date of Submission: 30/05/2025

Introduction:

Proteins are fundamental macromolecules responsible for nearly every biological process in living organisms. Their function is directly determined by their three-dimensional (3D) structure, which in turn is influenced by the protein's primary structure- the linear sequence of amino acids. However, predicting the full 3D structure of a protein directly from its sequence remains a challenging and computationally intensive task. To bridge this gap, protein 2D (secondary) structure prediction plays a critical intermediary role.

Protein secondary structure refers to the local spatial arrangement of amino acid residues within a protein, primarily characterized by recurring patterns such as alpha-helices (α-helices), beta-sheets (β-sheets) and random coils. These structural elements form through hydrogen bonding patterns between the backbone atoms of the amino acid chain and represent an essential level of structural organization before the complete 3D shape is formed.

Accurately predicting these secondary structural elements offers several key advantages:

- **Functional Insight:** Secondary structure helps in understanding the biological function of unknown proteins and their potential interaction sites.
- **Structure Modeling:** It provides constraints and guidance for tertiary (3D) structure prediction algorithms.
- **Experimental Support:** Computational predictions can aid and validate results from experimental methods like X-ray crystallography and NMR spectroscopy, which are often expensive and time-consuming.
- **Biomedical Applications:** It is particularly important in fields like drug discovery where the identification of binding sites and active regions in a protein is necessary.

With the explosion of protein sequence data from databases such as NCBI and UniProt, there is a growing need for accurate and scalable computational models. Traditional statistical approaches are being outperformed by machine learning and deep learning techniques, which can learn complex sequence-structure relationships from large datasets. Modern neural network models- like convolutional neural networks (CNNs), recurrent neural networks (RNNs) and transformer-based architectures- are increasingly used for predicting secondary structures with higher accuracy.

Methodology:

This section outlines the step-by-step approach adopted for classifying protein sequences using deep learning techniques. The methodology includes data acquisition, preprocessing, sequence encoding, model architecture design, training strategy and evaluation.

- **Data Collection:**
  Protein sequences for this study were retrieved from the National Center for Biotechnology Information (NCBI) database in FASTA format. Each entry in the dataset corresponds to a distinct protein and consists of two primary components: a descriptive header and the amino acid sequence itself. The amino acid sequences represent the unique primary structures of the proteins, while the descriptive headers contain valuable metadata, such as protein names, functional information and source organism details. From these descriptive annotations, relevant labels were systematically extracted to serve as target classes for subsequent analysis. These labels typically denote the species of origin or broader taxonomic classifications (e.g., genus, family, or kingdom). By linking each sequence to its corresponding taxonomic label, the dataset enables both supervised learning approaches in bioinformatics and the study of sequence–taxonomy relationships. fasta_file = "/kaggle/input/ncbi-data/ncbi_dataset/ncbi_dataset/data/GCF_963681135.1/protein.faa"

```
import os
directory = "/kaggle/input/ncbi-data/ncbi_dataset/ncbi_dataset/data"
files = os.listdir(directory)
print(files)
```

```
['GCF_963681135.1', 'assembly_data_report.jsonl', 'data_summary.tsv', 'dataset_catalog.json']
```

```
✅Total Sequences Parsed from FASTA: 4943
✅Top 3 Labels:
1. Enterobacteriaceae (2338 sequences)
2. Escherichia coli (1294 sequences)
3. Escherichia (635 sequences)
✅Total Sequences After Filtering for Top 3 Labels: 4267
```

- **Data Preprocessing:**

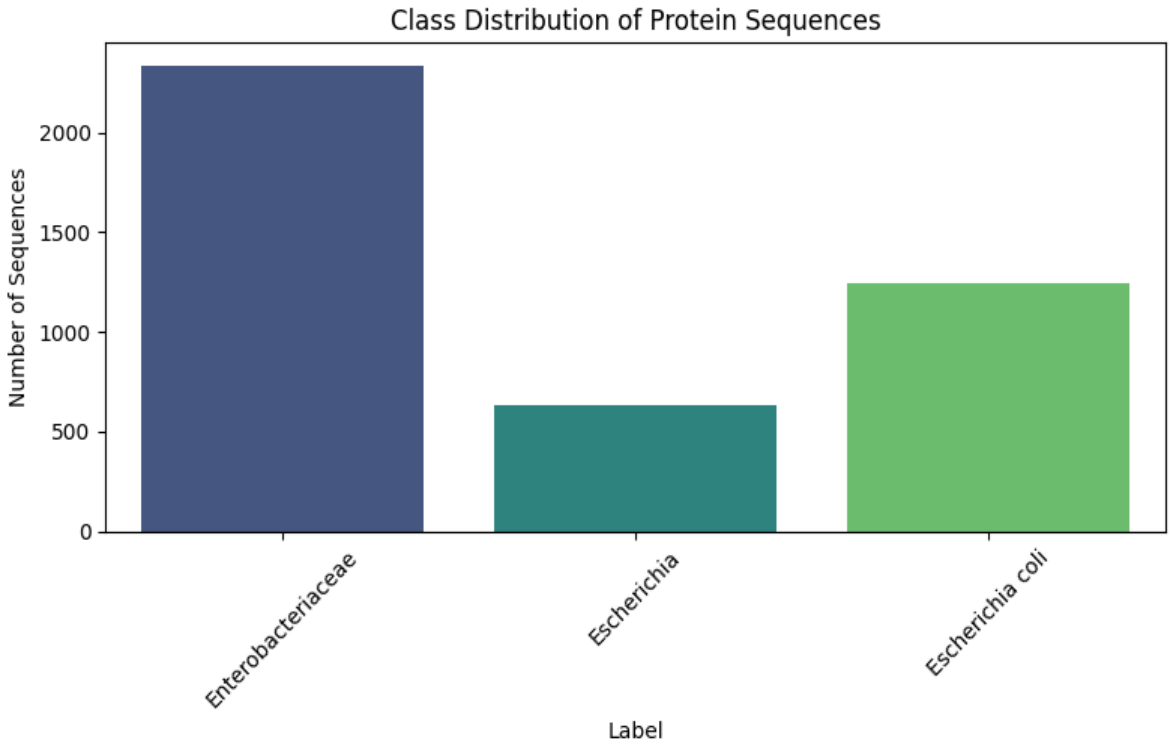
The collected protein sequences underwent a structured preprocessing pipeline to ensure data quality and suitability for downstream analysis. This process involved extracting biologically meaningful labels, filtering and cleaning sequences to remove inconsistencies and converting amino acid residues into numerical representations for computational modeling. Preprocessing includes:

- ✓ **Label Extraction**: Descriptive metadata from each FASTA header was parsed to extract biologically relevant labels (e.g., species names within square brackets).
- ✓ **Filtering**: Only the top three most frequent labels were retained to ensure class balance and reduce noise.
- ✓ **Sequence Cleaning**: Sequences containing uncommon amino acids or with lengths less than 50 residues were discarded.
- ✓ **Tokenization**: Each amino acid was mapped to an integer index using a predefined vocabulary of 20 standard amino acids (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y).

```
✅Amino Acid Vocabulary Size (Excluding Padding): 20
✅Total Vocabulary Size (Including Padding Token 0): 21
✅Example Tokenized Sequence (First One First 50 Tokens):
[11, 1, 12, 10, 16, 6, 20, 12, 5, 1, 20, 10, 3, 4, 14, 17, 9, 15, 11, 8, 15,
15, 1, 8, 10, 9, 1, 18, 1, 8, 13, 6, 20, 14, 18, 13, 5, 6, 6, 15, 4, 11, 13,
11, 13, 20, 6, 19, 6, 17]
✅Sequences with ≥50 Amino Acids: 4206
✅Average Sequence Length: 336.50
✅Max Sequence Length: 3515

✅Input shape (X): (4206, 500)  → (samples, padded_sequence_length)
✅Output shape (Y): (4206, 3)  → (samples, number_of_classes)
✅Encoded Label Classes:
0: Enterobacteriaceae
1: Escherichia
2: Escherichia coli
✅Example Padded Sequence (first 20 tokens):
[11  1 12 10 16  6 20 12  5  1 20 10  3  4 14 17  9 15 11  8]
✅Corresponding One-Hot Label Vector:
[1. 0. 0.]
📊Class Distribution in One-Hot Format:
Enterobacteriaceae: 2332 samples
Escherichia: 634 samples
Escherichia coli: 1240 samples
```

- **Bar Chart of Class Distribution (One-Hot Encoded Labels):**



- **Integer Encoding:** Each protein sequence was converted into a list of integers using the amino acid-to-index mapping.
- **Padding/Truncation:** Sequences were padded or truncated to a uniform length of 500 residues to ensure consistent input dimensions for the neural network.
- **Categorical Encoding:** The class labels were transformed into one-hot encoded vectors for use in multi-class classification.

- **Model Development:** To classify protein sequences, a deep learning model was developed using the Keras API in TensorFlow. The architecture begins with an Embedding layer, which transforms tokenized amino acid sequences into dense vector representations suitable for learning. This is followed by a 1D Convolutional layer that captures local sequence motifs and patterns indicative of structural or functional properties. To improve training stability and prevent overfitting, Batch Normalization and Dropout layers are strategically used. A Bidirectional GRU (Gated Recurrent Unit) layer is included to capture long-range dependencies in both forward and reverse directions, which is crucial for understanding biological sequence context.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(8, 500, 64)	1,344
conv1d_2 (Conv1D)	(8, 500, 128)	41,088
batch_normalization_4 (BatchNormalization)	(8, 500, 128)	512
dropout_4 (Dropout)	(8, 500, 128)	0
bidirectional_2 (Bidirectional)	(8, 128)	74,496
batch_normalization_5 (BatchNormalization)	(8, 128)	512
dropout_5 (Dropout)	(8, 128)	0
dense_2 (Dense)	(8, 3)	387

Total params: 353,995 (1.35 MB)

Trainable params: 117,827 (460.26 KB)

Non-trainable params: 512 (2.00 KB)

Optimizer params: 235,656 (920.54 KB)

Finally, a Dense output layer with softmax activation classifies each sequence into one of the top three classes. The model is compiled using the Adam optimizer and categorical cross-entropy loss function and it tracks accuracy as the primary evaluation metric. Additionally, a custom callback (WeightTracker) was implemented to save model weights layer-wise at the end of each training epoch, facilitating visualization and analysis of learning behavior.

- **Model Training:** The model was trained on the preprocessed protein sequence data using a supervised learning approach. To optimize performance and prevent overfitting, early stopping was employed via the Early Stopping callback, which monitors the validation loss during training. If the validation loss does not improve for 15 consecutive epochs, training is halted early and the best weights are automatically restored. The training process runs for a maximum of 15 epochs with a batch size of 8, balancing training time and gradient stability. Additionally, the custom Weight Tracker callback was integrated to save the weights of each layer at the end of every epoch, enabling further analysis and visualization of how the model's internal representations evolve during training. The training history, including accuracy and loss trends, is stored for later evaluation.

```
X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.3, random_state=42)
X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5, random_state=42)

print("✅Dataset Split Summary:")
print(f"🧠Training set:     {X_train.shape[0]} samples")
print(f"🧪Validation set:   {X_val.shape[0]} samples")
print(f"🔬Test set:         {X_test.shape[0]} samples")
```

```
✅Dataset Split Summary:
🧠Training set:     2944 samples
🧪Validation set:   631 samples
🔬Test set:         631 samples
```

early_stop = Early_Stopping (monitor = 'val_loss', patience = 15, restore_best_weights = True)

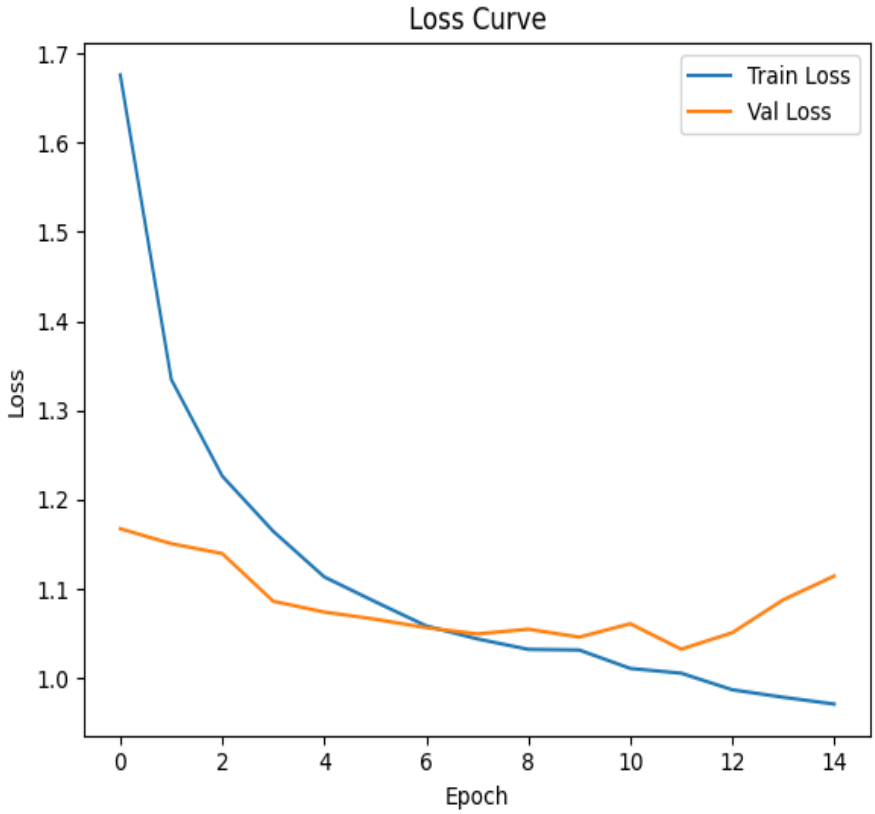
weight_tracker = WeightTracker()
history = model.fit(
X_train, Y_train,
validation_data = (X_val, Y_val),
epochs = 15,
batch_size = 8,
callbacks = [early_stop, weight_tracker]
)

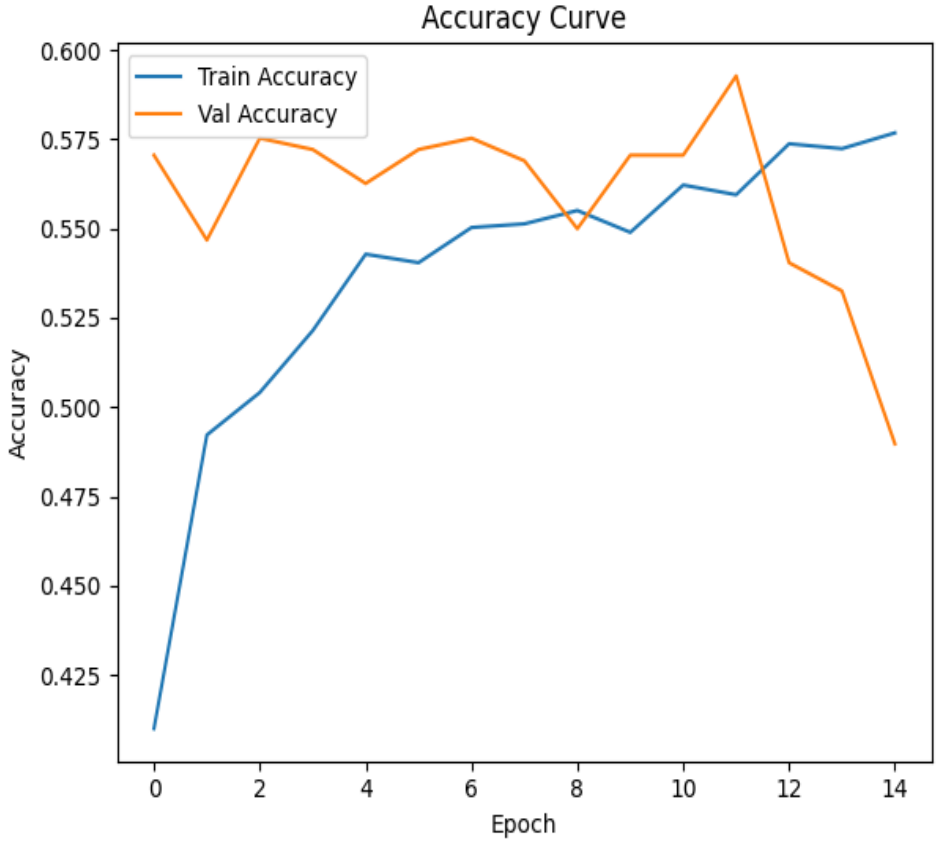
Result:

The model's training performance was evaluated using loss and accuracy curves, providing insights into its learning behavior and generalization capability over epochs. The loss curve illustrates how well the model minimizes prediction errors, while the accuracy curve reflects its classification performance on both training and validation dataset.

1. Loss Curve:

- **Train Loss (Blue line):** Decreases steadily across epochs, showing that the model is successfully learning patterns from the training data.
- **Validation Loss (Orange line):** Initially decreases, indicating improved generalization, but begins to fluctuate and rise slightly after epoch 9.


Loss Curve
Train Loss
Val Loss
Loss
Epoch
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
2
4
6
8
10
12
14


Accuracy Curve
Train Accuracy
Val Accuracy
Accuracy
Epoch
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
0
2
4
6
8
10
12
14

- **Interpretation:**

    - ✓ The increasing validation loss while training loss continues to drop is a sign of overfitting. The model starts memorizing training data instead of learning generalizable features.
    - ✓ The use of early stopping was beneficial here, as training stopped around epoch 14 before the validation loss worsened too much.

2. Accuracy Curve:

- **Train Accuracy (Blue line):** Increases consistently, showing effective learning from training data.
- **Validation Accuracy (Orange line):** Improves up to epoch 11, even exceeding training accuracy at some points (which can happen due to small batch variance), but then starts to decline.
- **Interpretation:**

    - ✓ The peak validation accuracy followed by a decline reinforces the overfitting observation.
    - ✓ The model reached its best validation accuracy around epoch 11, which was likely the point where generalization was at its peak.

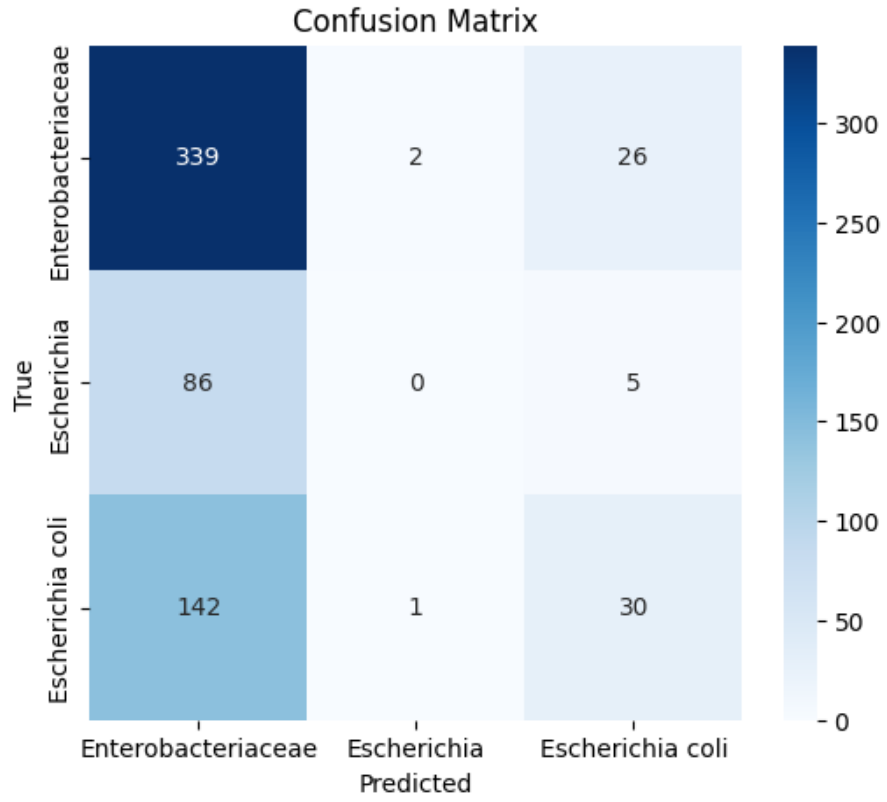
The confusion matrix summarizes the classification performance of the protein sequence model across three classes: Enterobacteriaceae, Escherichia and Escherichia coli. Each row represents the true label, while each column represents the predicted label.

Correct Predictions (Diagonal):

- Enterobacteriaceae: 339 correctly predicted.
- Escherichia coli: 30 correctly predicted.
- Escherichia: 0 correct predictions – very poor performance for this class.

Misclassifications:

- Escherichia samples were mostly misclassified as Enterobacteriaceae (86 out of 91).
- Escherichia coli also had 142 misclassified as Enterobacteriaceae.



✅ Classification Report:

	precision	recall	f1-score	support
Enterobacteriaceae	0.5979	0.9237	0.7259	367
Escherichia	0.0000	0.0000	0.0000	91
Escherichia coli	0.4918	0.1734	0.2564	173
accuracy			0.5848	631
macro avg	0.3632	0.3657	0.3274	631
weighted avg	0.4826	0.5848	0.4925	631

✅ Test Accuracy: 0.5848

The model heavily favors predicting the Enterobacteriaceae class. This classification report provides a detailed evaluation of the model's performance on the test dataset across three classes: Enterobacteriaceae, Escherichia, and Escherichia coli. The metrics used are precision, recall, and F1-score, each giving insight into different aspects of model behavior.

Enterobacteriaceae:

The model performs best on this class with high recall (0.92) meaning it captures most of the actual Enterobacteriaceae samples. However, precision is moderate (0.60) suggesting some false positives.

Escherichia:
Performance is very poor- the model failed to correctly identify any Escherichia samples, resulting in 0.0 precision, recall, and F1-score. This indicates the model completely ignored this class.

Escherichia coli:
Performance is weak with low recall (0.17), meaning most E. coli samples were misclassified, and modest precision (0.49).

Accuracy:
The model achieved an overall test accuracy of 0.5848, meaning it correctly classified about 58.48% of the test samples. This indicates that while the model performs better than random guessing, there is still substantial room for improvement. Further optimization of architecture, features or training strategy may help increase its predictive accuracy.

Macro Average:

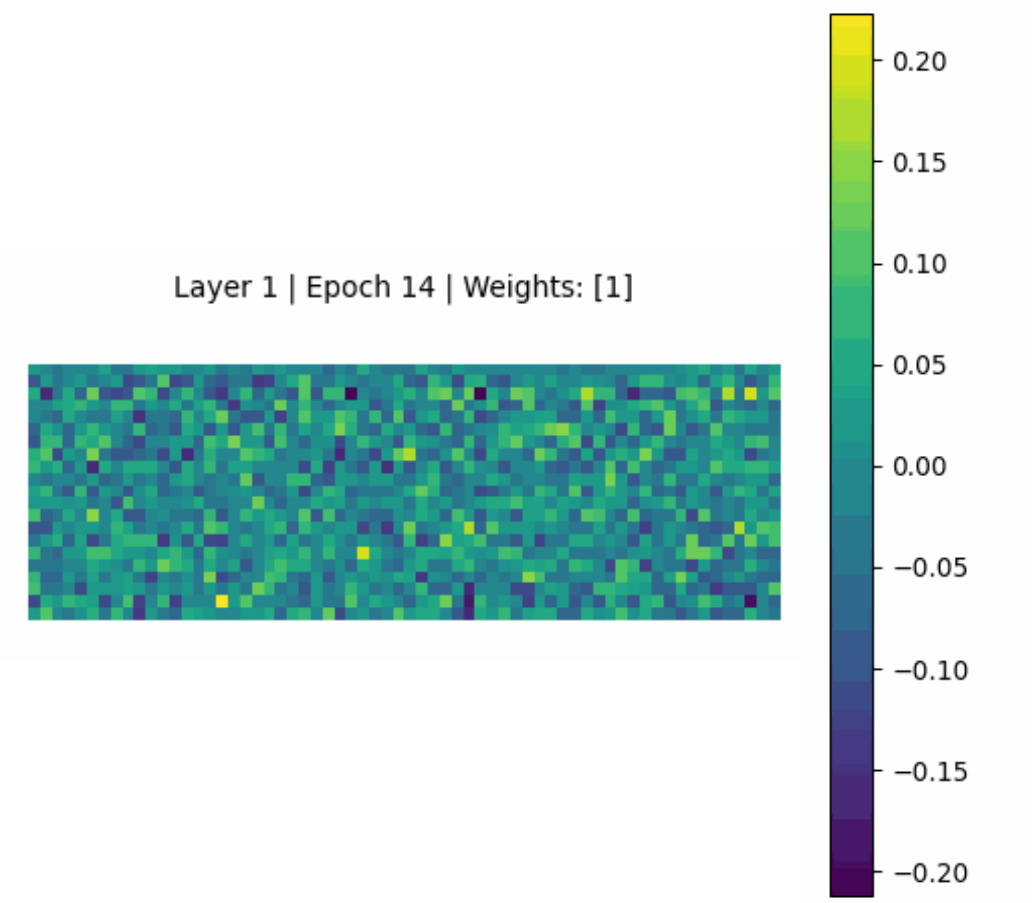
- Averages all class metrics equally, regardless of class size.
- Low scores (~0.36 precision/recall, 0.32 F1) show that performance is not balanced across classes.

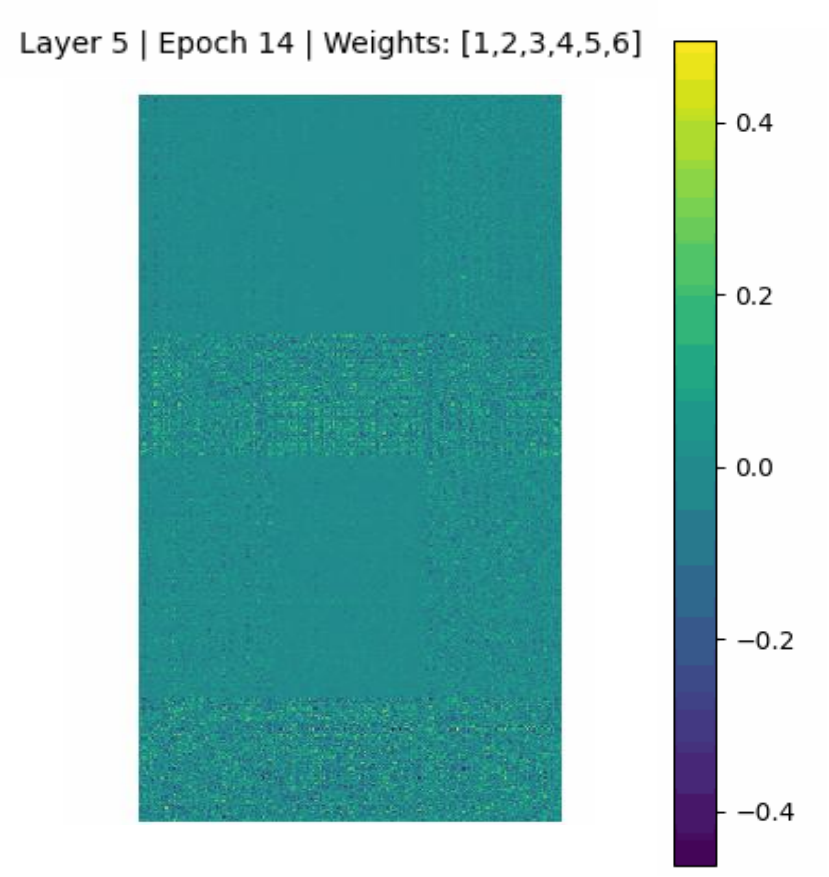
Weighted Average:

- Takes class imbalance into account.
- Slightly better (F1-score = 0.4925) due to dominance of the Enterobacteriaceae class.

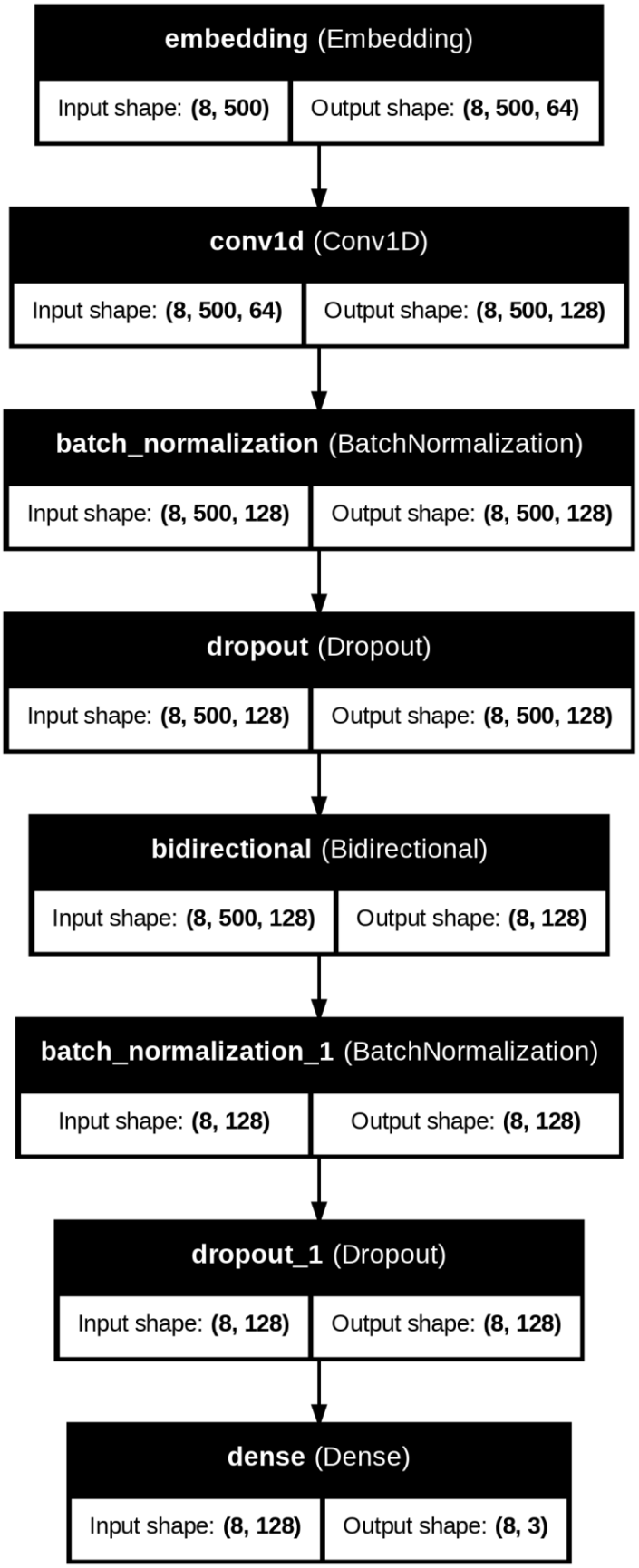
Sample Image from GIF as Example:

GIFs are generated that visually depict how the weights of each neural network layer evolve over training epochs. It reads .npy files containing saved weights, groups them by layer and epoch, and ensures each weight array is in a 2D format for visualization. For each epoch of a given layer, the script stacks all relevant weight arrays vertically and creates a heatmap using Matplotlib. These heatmaps, one per epoch, are saved as images and then compiled into animated GIFs using imageio, effectively illustrating the dynamic changes in layer weights throughout training.


Layer 1 | Epoch 14 | Weights: [1]
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20


Layer 5 | Epoch 14 | Weights: [1,2,3,4,5,6]
0.4
0.2
0.0
−0.2
−0.4

Model Architecture:



Discussion:

The classification results indicate that the model exhibits a significant bias towards the majority class, Enterobacteriaceae**,** which dominates the dataset. This is evident from its high recall (0.92) and F1-score (0.72) compared to the other classes. While the model successfully identifies most samples from this class, it fails to generalize well to the minority classes.

In particular, the model completely misclassifies all instances of the Escherichia class, yielding zero precision, recall, and F1-score. The performance on Escherichia coli is also suboptimal, with a low recall (0.17) and F1-score (0.25). These results strongly suggest that the model is unable to learn meaningful distinctions between the underrepresented classes, likely due to the imbalance in class distribution.

Such class imbalance is a common issue in biological datasets, where certain organisms or species may be overrepresented. Without addressing this imbalance, the model tends to overfit to the dominant class, ignoring the nuances of the others.

To mitigate these issues and improve classification performance, several strategies should be considered to gain better performance:

- **Use of Data balancing technique:** Such as oversampling the minority classes, under sampling the majority class, or employing synthetic data generation (e.g., SMOTE) could help the model receive a more balanced learning signal.
- **Incorporating class weights:** Integrating class weights into the loss function can help the model address class imbalance by penalizing errors in minority classes more heavily, thereby encouraging balanced learning across all categories.
- **Exploring more advanced architectures**: Utilizing more sophisticated architectures, such as attention mechanisms or transformer-based layers, may enhance the model's ability to capture long-range dependencies and complex patterns in protein sequences.
- **Feature Enrichment with Biological Insights:** Augmenting the input data with domain-specific biological features- such as physicochemical properties, conserved motifs, or predicted secondary structures- can provide the model with richer contextual information, improving its ability to distinguish between closely related classes.

To improve the model's generalizability and make it more suitable for practical biological applications, the following steps are proposed for future work:

- **Class Balancing**: Implement oversampling, undersampling, or data augmentation techniques to ensure a more uniform class distribution.
- **Class Weights in Training**: Assign higher weights to underrepresented classes in the loss function to address imbalance during learning.
- **Advanced Architectures**: Explore the use of transformers, attention mechanisms, or pretrained biosequence models (like ESM or ProtBert).
- **Biological Feature Integration**: Combine raw sequences with biologically relevant features, such as structural annotations, hydrophobicity, or evolutionary conservation.
- **Cross-validation**: Use stratified k-fold cross-validation to ensure the model is evaluated robustly across various class splits.
- **Larger and Diverse Datasets**: Consider expanding the dataset to include more balanced representations across species and genera.

By addressing these areas, future models can be made more accurate, balanced, and biologically meaningful, advancing the field of protein function and taxonomy prediction.

In summary, while the current model demonstrates promising performance on the majority class, it exhibits noticeable limitations when it comes to accurately predicting minority classes. This indicates that the model may be biased toward more frequently occurring categories, resulting in an imbalanced performance profile. Such imbalance can reduce the model's effectiveness and generalizability in real-world applications, where accurate classification across all protein types is crucial. Therefore, further improvements are needed—such as addressing data imbalance, refining model architecture, or applying targeted training strategies—to ensure more consistent and equitable performance across all categories. These enhancements would ultimately make the model more reliable, robust, and practical for comprehensive protein classification tasks.

Git Hub Project Link:

https://github.com/Nushrat-Tarmin-Meem/Protein-2D-Structure-Prediction-Using-Deep-Learning