SC4001 Neural Network & Deep Learning Group Project

Name	Matriculation Number	Peer Review / Contribution	Signature
Brandon Jang Jin Tian	U2220936G	Deep Learning for Protein Structure Prediction (33.33%)	[illegible]
Chung Zhi Xuan	U2220300H	Deep Learning for Protein Structure Prediction (33.33%)	[illegible]
Yau Jun Hao	U2220332F	Sentiment Analysis (33.33%)	[illegible]

1. Introduction

This study explores advanced approaches in sentiment analysis and protein structure prediction through deep learning methodologies. Initially, a compact transformer model, DistilGPT-2, was fine-tuned on movie review data to perform sentiment classification, achieving an accuracy of 88.4%. Afterwards, the model was adapted to predict star ratings, yielding a test accuracy of 51.8% initially, which improved to 75.5% after consolidating classes. A comparative analysis with DistilBERT and TinyBERT confirmed the effectiveness of DistilBERT for this task. Further research that addresses the limitations of small training data were carried out to conclude section 1. Further, protein secondary structure prediction from amino acid sequences was investigated using CNN, LSTM, and Seq2Seq architectures. While these approaches yielded moderate success in simpler Q3 classification, they struggled with the more complex Q8 classification due to limited context-awareness. To address this, a Transformer-based model leveraging self-attention mechanisms was employed to capture both short- and long-range dependencies. This comparative analysis underscores the strengths of Transformer models in capturing complex dependencies, marking a significant advance in both sentiment analysis and bioinformatics applications.

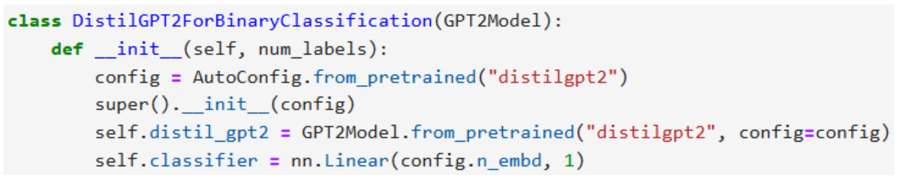
Table of Contents

1. Introduction..1
2. Sentiment Analysis..3
2.1. Base Model Training..3
2.2. Transfer Learning..3
2.3. Comparative Analysis Between Transformers..4
2.4. Training with insufficient data..4
2.4.1. TinyBert..5
2.4.2. DistilBert..5
3. Deep Learning for Protein Structure Prediction..5
3.1. Objective..5
3.2. Review of Existing Techniques..6
3.3. Data Preprocessing..6
3.4. Baseline Architectures..6
3.4.1. CNN Layer and LSTM Layer..6
3.5. Architecture Refinement and Improvement..7
3.5.1. Novel Architecture..7
3.5.1.1. Sequence-to-Sequence (Seq2Seq)..7
3.5.1.2. Transformer..9
3.5.2. Features Representation..10
3.5.3. Comparative Analysis..10
3.6. Results Analysis..11
References..13

2. Sentiment Analysis

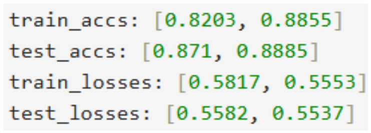
2.1. Base Model Training

We trained a transformer model to classify the sentiment of text using a dataset of movie reviews. The dataset consists of reviews labelled with either a positive or negative sentiment. For the model, we chose **DistilGPT2**, a more compact version of GPT2 provided by HuggingFace (Wolf et al.,2020). While GPT2 excelled in generating and predicting the next word (Xu, H. et al, 2021), we modified it by adding a classifier layer for sentiment classification. The model is pre-trained on a large corpus and is now fine-tuned using the movie reviews dataset.



```
class DistilGPT2ForBinaryClassification(GPT2Model):
    def __init__(self, num_labels):
        config = AutoConfig.from_pretrained("distilgpt2")
        super().__init__(config)
        self.distil_gpt2 = GPT2Model.from_pretrained("distilgpt2", config=config)
        self.classifier = nn.Linear(config.n_embd, 1)
```

Figure 1: DistilGPT2 Model For Binary Classification



```
train_accs: [0.8203, 0.8855]
test_accs: [0.871, 0.8885]
train_losses: [0.5817, 0.5553]
test_losses: [0.5582, 0.5537]
```

Figure 2(a): Train, Test Accuracies & Losses

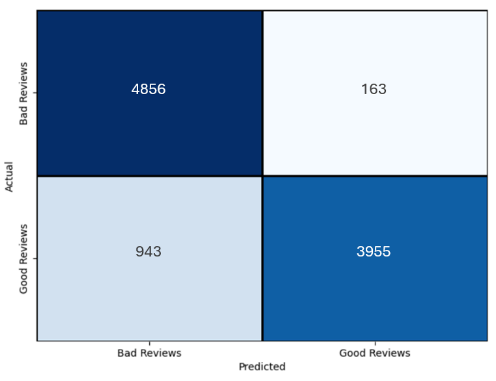
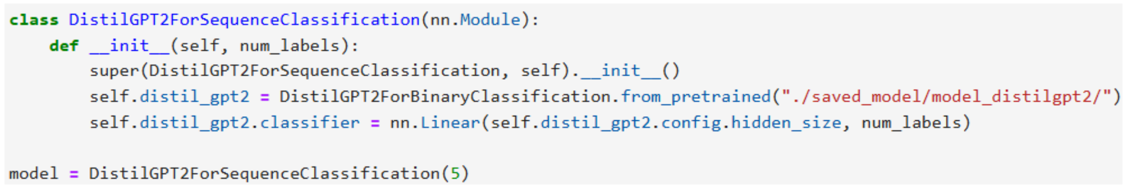


Figure 2(b): Confusion Matrix of Base Model

After training for 2 epochs, the model achieved test accuracies of 88.9%.

2.2. Transfer Learning

The text sentiments are correlated with star ratings (Sameh, A.,2020). Therefore, we utilised domain adaptation to modify a trained binary classifier model into a multiclass classifier to predict star ratings. The target domain is a dataset of restaurant customer reviews, where the star rating is the target variable. Since the dataset is imbalanced, we selected a subset of the original dataset, ensuring a constant number of rows for each representative class.



```
class DistilGPT2ForSequenceClassification(nn.Module):
    def __init__(self, num_labels):
        super(DistilGPT2ForSequenceClassification, self).__init__()
        self.distil_gpt2 = DistilGPT2ForBinaryClassification.from_pretrained("./saved_model/model_distilgpt2/")
        self.distil_gpt2.classifier = nn.Linear(self.distil_gpt2.config.hidden_size, num_labels)

model = DistilGPT2ForSequenceClassification(5)
```

Figure 3: DistilGPT2 Model For Multiclass classification

Our initial approach classifies the star ratings into the full set of five labels. However, the results were not as expected.

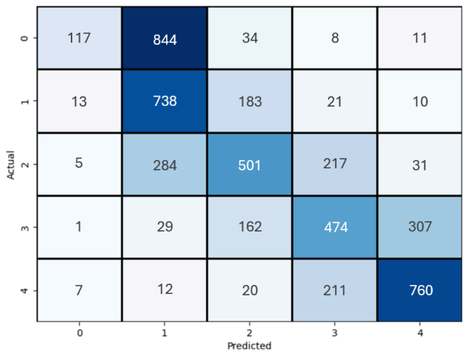


Figure 4(a): Confusion Matrix for five labels (before)

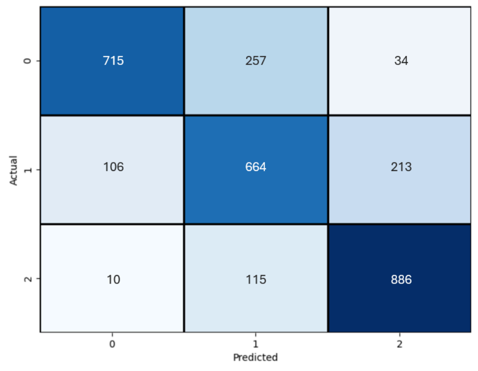
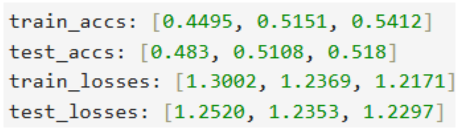
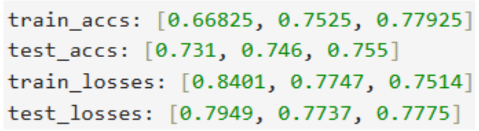


Figure 5(a): Confusion Matrix for three labels (after)



```
train_accs: [0.4495, 0.5151, 0.5412]
test_accs: [0.483, 0.5108, 0.518]
train_losses: [1.3002, 1.2369, 1.2171]
test_losses: [1.2520, 1.2353, 1.2297]
```

Figure 4(b): Train, Test Accuracies & Losses



```
train_accs: [0.66825, 0.7525, 0.77925]
test_accs: [0.731, 0.746, 0.755]
train_losses: [0.8401, 0.7747, 0.7514]
test_losses: [0.7949, 0.7737, 0.7775]
```

Figure 5(b): Train, Test Accuracies & Losses

As shown in Figure 4(b), the test accuracy did not improve and remains at 51.8% after 3 epochs. Most 1-star ratings were misclassified as 2-star ratings. Other ratings tend to be misclassified as either one grade higher or one grade lower. This is likely because the distinction between star ratings is not as clear-cut as sentiment, which is classified as either positive, negative, or neutral.

To address this, we modified the star rating column by combining 1-star and 2-star ratings into one class, and 4-star and 5-star ratings into another. As a result, we now have a total of 3 classes for classification. To ensure there is no imbalance between the classes, we included double the number of training samples for the 3-star ratings compared to the other classes.

We now experience significantly less misclassification, with a test accuracy of 0.755 as shown in Figure 5(b). The entire process took 22 minutes per epoch with 15,000 rows of samples. This is notably faster than the base model, which used 50,000 rows of samples and took 32 minutes per epoch, yet achieved slightly lower accuracy.

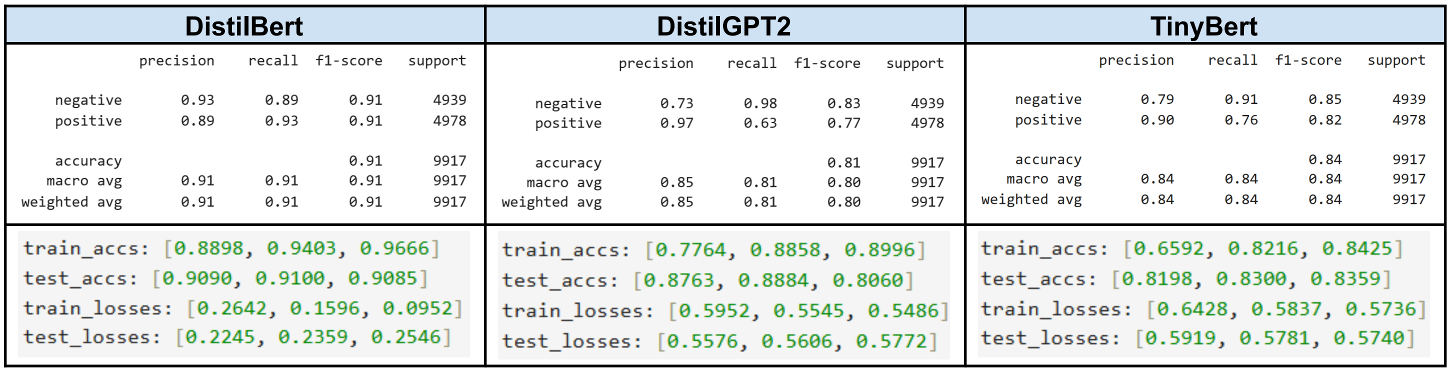
2.3. Comparative Analysis Between Transformers

In this section, we have selected several transformer models to perform the same tasks. We kept the hyperparameters consistent across all models to ensure a fair comparison. Three models were chosen: **DistilBert** and **DistilGPT2** as medium-sized models, and **TinyBert** as a small model.

The main distinction between DistilBert and DistilGPT2 lies in their prediction mechanisms: DistilBert predicts words by considering both the left and right context, while DistilGPT2 predicts words based on the preceding words in a left-to-right manner (Xu, H. et al, 2021). As a result, DistilBert excels at understanding context more effectively.

Hyperparameters	
Learning Rate	2e-5
Max Length	256
Batch Size	32
Epochs	3
Optimizer	AdamW

	DistilBert	**DistilGPT2**	**TinyBert**
No. of parameters	66955010	81913345	4386049
Time per epoch/min	30.6	32.4	1.5
Test accuracies	0.909	0.806	0.836
Highest Test accuracies	0.910	0.888	0.836
Overfits	True	True	False



DistilBert

```
              precision    recall  f1-score   support

    negative       0.93      0.89      0.91      4939
    positive       0.89      0.93      0.91      4978

    accuracy                           0.91      9917
   macro avg       0.91      0.91      0.91      9917
weighted avg       0.91      0.91      0.91      9917
```

```
train_accs: [0.8898, 0.9403, 0.9666]
test_accs: [0.9090, 0.9100, 0.9085]
train_losses: [0.2642, 0.1596, 0.0952]
test_losses: [0.2245, 0.2359, 0.2546]
```

DistilGPT2

```
              precision    recall  f1-score   support

    negative       0.73      0.98      0.83      4939
    positive       0.97      0.63      0.77      4978

    accuracy                           0.81      9917
   macro avg       0.85      0.81      0.80      9917
weighted avg       0.85      0.81      0.80      9917
```

```
train_accs: [0.7764, 0.8858, 0.8996]
test_accs: [0.8763, 0.8884, 0.8060]
train_losses: [0.5952, 0.5545, 0.5486]
test_losses: [0.5576, 0.5606, 0.5772]
```

TinyBert

```
              precision    recall  f1-score   support

    negative       0.79      0.91      0.85      4939
    positive       0.90      0.76      0.82      4978

    accuracy                           0.84      9917
   macro avg       0.84      0.84      0.84      9917
weighted avg       0.84      0.84      0.84      9917
```

```
train_accs: [0.6592, 0.8216, 0.8425]
test_accs: [0.8198, 0.8300, 0.8359]
train_losses: [0.6428, 0.5837, 0.5736]
test_losses: [0.5919, 0.5781, 0.5740]
```

Based on the classification report, DistilBert has the highest F1-score among all three models. However, if the purpose of the model is to detect spam, DistilGPT2 actually performs better, as it has a higher recall than the DistilBert model. TinyBERT provides a trade-off between accuracy and training time, as it achieves relatively lower accuracy but significantly reduces training time.

2.4. Training with insufficient data

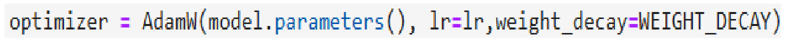
Training a deep learning model on a small dataset can be challenging, as these models typically require large amounts of data to effectively learn meaningful patterns. To address this limitation, several techniques can be employed. One of the most widely used approaches is transfer learning, which is discussed in Section 1.2. However, for the purposes of this task, we assumed that transfer learning is not a feasible solution.

Model complexity plays a significant role when training on small datasets, and using a simpler model is often more suitable for such tasks (Brigato & Iocchi, 2020).

2.4.1. TinyBert

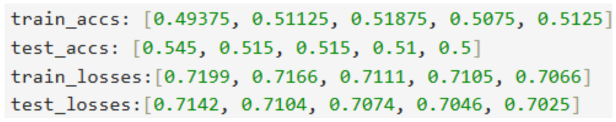
Our first choice was TinyBert. However, the model tended to classify all inputs into a single class. To address this, we implemented several mitigation strategies. Aside from experimenting with more complex models, we also applied various regularisation techniques, such as introducing weight decay to the optimiser.

Hyperparameters	
Learning Rate	1e-6
Max Length	256
Batch Size	8
Weight Decay	1e-4
Epochs	5



```
optimizer = AdamW(model.parameters(), lr=lr,weight_decay=WEIGHT_DECAY)
```

Figure 6: AdamW Optimiser with Weight Decay



```
train_accs: [0.49375, 0.51125, 0.51875, 0.5075, 0.5125]
test_accs: [0.545, 0.515, 0.515, 0.51, 0.5]
train_losses:[0.7199, 0.7166, 0.7111, 0.7105, 0.7066]
test_losses:[0.7142, 0.7104, 0.7074, 0.7046, 0.7025]
```

Figure 7(b): *Train, Test Accuracies & Losses*

Despite these efforts, our model still underperformed, achieving only a test accuracy of 0.5. This suggests that the chosen model may be too simple to capture the underlying patterns. Therefore, we have decided to explore alternative models that may be better suited for the task.

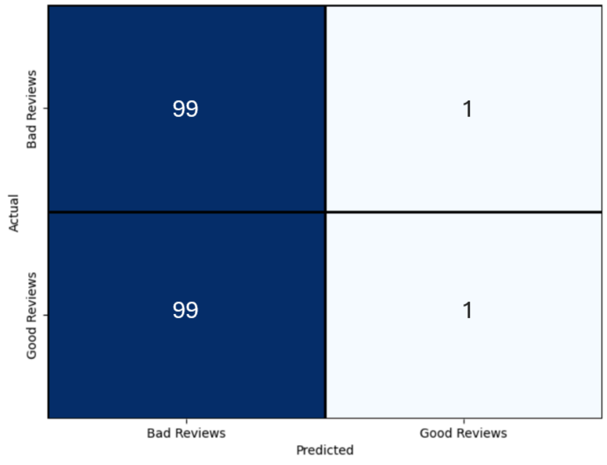


Figure 7(a): Confusion Matrix for TinyBert Model

2.4.2. DistilBert

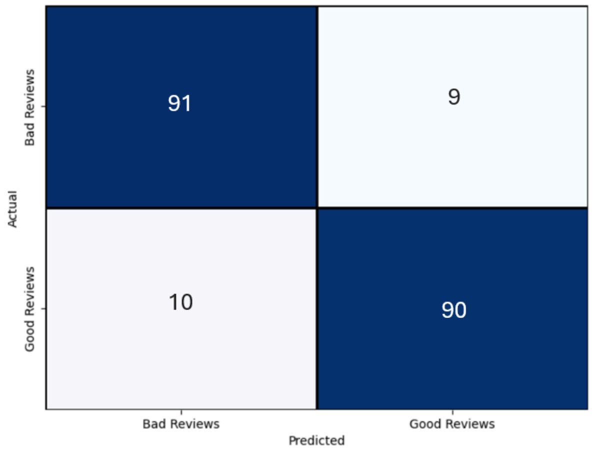
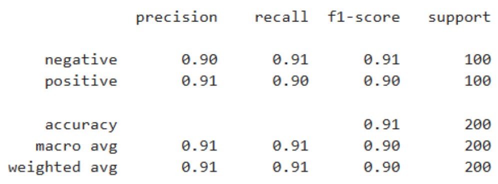


Figure 8(a): Confusion Matrix for DistilBert Model



```
              precision    recall  f1-score   support

    negative       0.90      0.91      0.91       100
    positive       0.91      0.90      0.90       100

    accuracy                           0.91       200
   macro avg       0.91      0.91      0.90       200
weighted avg       0.91      0.91      0.90       200
```

Figure 8(b): Classification Report

Our second model, DistilBert, performs exceptionally well on this dataset, even without additional tuning. This suggests that the model is neither too complex to overfit nor too simple to underfit, which is especially important for small datasets that are sensitive to model complexity.

3. Deep Learning for Protein Structure Prediction

3.1. Objective

The primary goal of this project is to predict the secondary structure of proteins, represented in Q3 and Q8 formats, from their primary amino acid sequences. This approach seeks to reduce the need for extensive lab work by using deep learning techniques, such as RNN, CNN and transformers to capture both short- and long-range dependencies within protein sequences.

3.2. Review of Existing Techniques

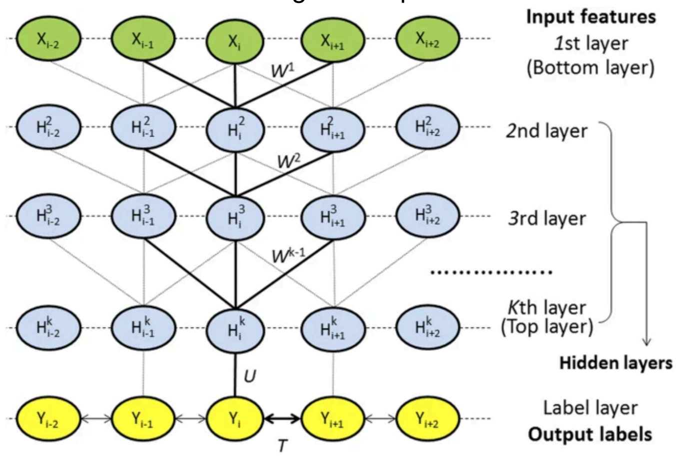
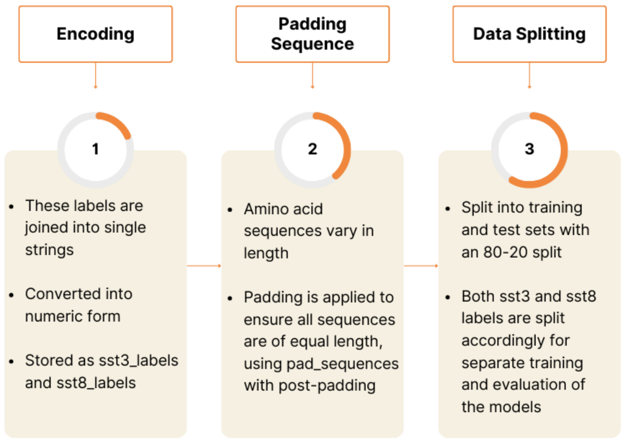


Figure 9: DeepCNF Architecture

In the field of protein secondary structure prediction, various techniques have been employed, including:

- **Traditional methods:** Protein secondary structure can be determined from the 3D coordinates of its atoms, which are obtained when the protein's structure is resolved through X-ray crystallography or nuclear magnetic resonance (NMR) spectroscopy (Secondary Structure Analysis - Creative Proteomics, 2024).
- **Deep learning models:** Models like CNNs (Convolutional Neural Networks) and LSTMs (Long Short-Term Memory networks) have shown promising results due to their ability to learn complex patterns and dependencies in data. For example, the DeepCNF (Deep Convolutional Neural Fields) model can achieve approximately 84% accuracy for Q3 classification and around 72% accuracy for Q8 classification (Wang et al., 2016).

3.3. Data Preprocessing

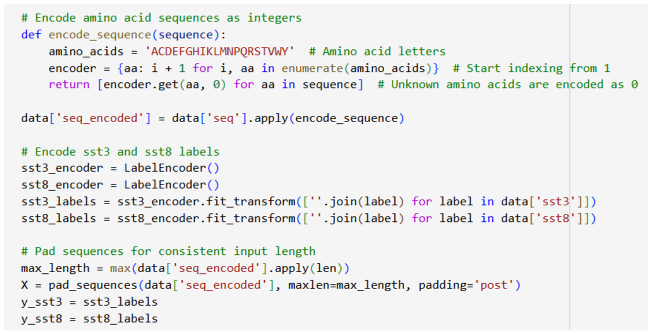


- These labels are joined into single strings
- Converted into numeric form
- Stored as sst3_labels and sst8_labels

- Amino acid sequences vary in length
- Padding is applied to ensure all sequences are of equal length, using pad_sequences with post-padding

- Split into training and test sets with an 80-20 split
- Both sst3 and sst8 labels are split accordingly for separate training and evaluation of the models

Figure 10: Data Preprocessing Steps



```
# Encode amino acid sequences as integers
def encode_sequence(sequence):
    amino_acids = 'ACDEFGHIKLMNPQRSTVWY'  # Amino acid letters
    encoder = {aa: i + 1 for i, aa in enumerate(amino_acids)}  # Start indexing from 1
    return [encoder.get(aa, 0) for aa in sequence]  # Unknown amino acids are encoded as 0

data['seq_encoded'] = data['seq'].apply(encode_sequence)

# Encode sst3 and sst8 labels
sst3_encoder = LabelEncoder()
sst8_encoder = LabelEncoder()
sst3_labels = sst3_encoder.fit_transform([''.join(label) for label in data['sst3']])
sst8_labels = sst8_encoder.fit_transform([''.join(label) for label in data['sst8']])

# Pad sequences for consistent input length
max_length = max(data['seq_encoded'].apply(len))
X = pad_sequences(data['seq_encoded'], maxlen=max_length, padding='post')
y_sst3 = sst3_labels
y_sst8 = sst8_labels
```

Figure 11: Encoding and Padding Code

As shown in Figure 10, the preprocessing pipeline begins by encoding amino acid sequences numerically, transforming secondary structure labels into integers, padding sequences to ensure consistent input lengths, and splitting the data into training and test sets. Each amino acid is represented by a unique integer from 1 to 20, with 0 reserved for unknown amino acids. The function *encode_sequence* performs this encoding, mapping amino acids in the sequence to integers based on a dictionary, *encoder*. The encoded sequences are stored in a new column, *seq_encoded*.

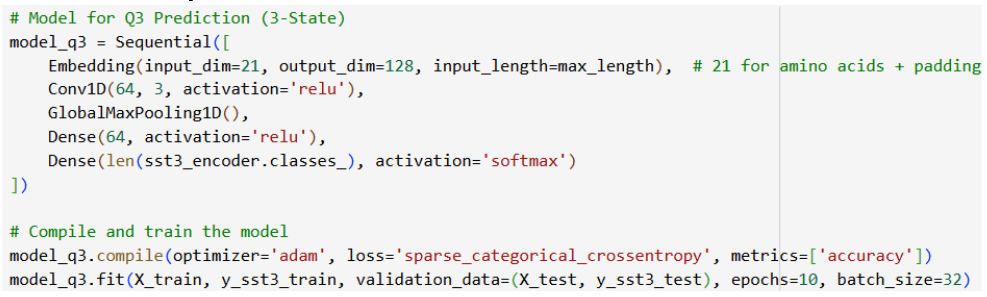
We then use the *LabelEncoder* class to encode the secondary structural labels *sst3* (three-state (Q3) secondary structure) and *sst8* (eight-state (Q8) secondary structure labels) into integer values.

Since amino acid sequences vary in length, padding is applied to ensure all sequences are of equal length. This is done by using *pad_sequences* with post-padding, resulting in a consistent input matrix *X*, where each sequence has the maximum length found in the dataset.

Finally, the data is then split into training and test sets with an 80-20 split, ensuring that both sst3 and sst8 labels are split accordingly for separate training and evaluation of the models.

3.4. Baseline Architectures

3.4.1. CNN Layer and LSTM Layer

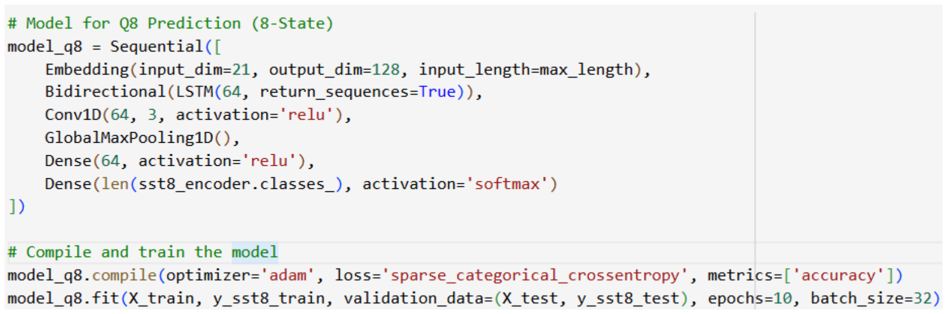


```
# Model for Q3 Prediction (3-State)
model_q3 = Sequential([
    Embedding(input_dim=21, output_dim=128, input_length=max_length),  # 21 for amino acids + padding
    Conv1D(64, 3, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dense(len(sst3_encoder.classes_), activation='softmax')
])

# Compile and train the model
model_q3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_q3.fit(X_train, y_sst3_train, validation_data=(X_test, y_sst3_test), epochs=10, batch_size=32)
```

Figure 12: CNN Model

CNN Architecture

Embedding Layer	Convert integer-encoded sequences into dense vector representations.
1D Convolutional Layer	64 filters and a kernel size of 3, capturing local patterns in the sequence.
Global Max Pooling	To downsample the feature maps.
Two Dense Layers	With the final layer outputting a probability distribution over the possible *sst3* labels using a softmax activation function.



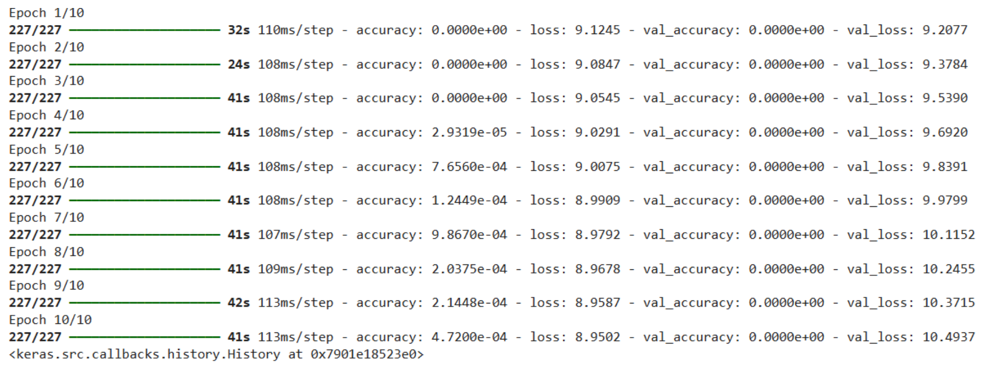
```
# Model for Q8 Prediction (8-State)
model_q8 = Sequential([
    Embedding(input_dim=21, output_dim=128, input_length=max_length),
    Bidirectional(LSTM(64, return_sequences=True)),
    Conv1D(64, 3, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dense(len(sst8_encoder.classes_), activation='softmax')
])

# Compile and train the model
model_q8.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_q8.fit(X_train, y_sst8_train, validation_data=(X_test, y_sst8_test), epochs=10, batch_size=32)
```

Figure 13: CNN with LSTM Layer

LSTM Architecture

Embedding Layer	Convert the sequences to dense vector representations.
Bidirectional LSTM Layer	Capture sequential dependencies in both directions along the sequence.
1D Convolutional Layer	64 filters and a kernel size of 3, capturing local patterns in the sequence.
Output Layer	Uses softmax activation to produce probabilities for the *sst8* labels.

Both models are compiled with the Adam optimizer and sparse categorical cross-entropy loss and trained for 10 epochs on the sst3 and sst8 labels.



```
Epoch 1/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 32s 110ms/step - accuracy: 0.0000e+00 - loss: 9.1245 - val_accuracy: 0.0000e+00 - val_loss: 9.2077
Epoch 2/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.0000e+00 - loss: 9.0847 - val_accuracy: 0.0000e+00 - val_loss: 9.3784
Epoch 3/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 108ms/step - accuracy: 0.0000e+00 - loss: 9.0545 - val_accuracy: 0.0000e+00 - val_loss: 9.5390
Epoch 4/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 108ms/step - accuracy: 2.9319e-05 - loss: 9.0291 - val_accuracy: 0.0000e+00 - val_loss: 9.6920
Epoch 5/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 108ms/step - accuracy: 7.6560e-04 - loss: 9.0075 - val_accuracy: 0.0000e+00 - val_loss: 9.8391
Epoch 6/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 108ms/step - accuracy: 1.2449e-04 - loss: 8.9909 - val_accuracy: 0.0000e+00 - val_loss: 9.9799
Epoch 7/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 107ms/step - accuracy: 9.8670e-04 - loss: 8.9792 - val_accuracy: 0.0000e+00 - val_loss: 10.1152
Epoch 8/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 109ms/step - accuracy: 2.0375e-04 - loss: 8.9678 - val_accuracy: 0.0000e+00 - val_loss: 10.2455
Epoch 9/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 42s 113ms/step - accuracy: 2.1448e-04 - loss: 8.9587 - val_accuracy: 0.0000e+00 - val_loss: 10.3715
Epoch 10/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 41s 113ms/step - accuracy: 4.7200e-04 - loss: 8.9502 - val_accuracy: 0.0000e+00 - val_loss: 10.4937
<keras.src.callbacks.history.History at 0x7901e18523e0>
```

Figure 14: Model Output Training for Q8

Our model's accuracy remains near zero across all epochs, which indicates that it is unable to make correct predictions for the Q8 classification task. Both training and validation accuracy show no improvement, staying at 0% accuracy, which is a critical issue. The loss values for both training and validation also do not exhibit a substantial decrease, remaining high throughout the training. This suggests that our current model is not learning effectively from the data. This shows that the current architecture **<u>might not be complex enough</u>** to capture the details required for 8-state classification, which is inherently more challenging than 3-state classification. In order to improve the performance, we will proceed to do an architecture refinement and improvement incorporating novel architecture and feature representation.

3.5. Architecture Refinement and Improvement

3.5.1. Novel Architecture

3.5.1.1. Sequence-to-Sequence (Seq2Seq)

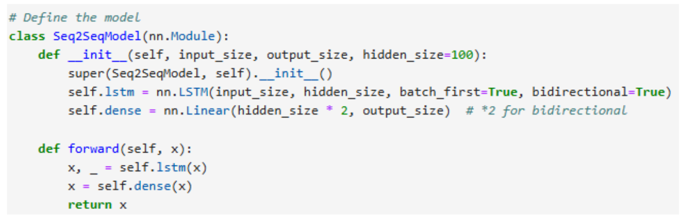
To address the challenge of predicting secondary protein structure, we developed a Seq2Seq architecture aimed at learning amino acid dependencies for accurate secondary structure classification in both Q3 and Q8 formats. This model is a special class of RNN which leverages the LSTM network's capacity to handle sequential data and capture long-term dependencies, which is essential given the complexities of protein sequences.

Additional Preprocessing Step Employed

One-Hot Encoding	The target labels (*sst3* and *sst8*) are converted to categorical format, which allows for multi-class classification across multiple positions within the sequence.

Bidirectional LSTM Layer	The core of the architecture is a bidirectional LSTM with a hidden size of 100 units, allowing the model to process information from both directions in the sequence. This setup improves the ability to capture context by considering both previous and upcoming amino acids.
Dense Output Layer	After processing the sequence with the LSTM, the model passes the output through a dense layer to produce predictions for each time step. The dense layer outputs a probability distribution over possible secondary structure labels (*sst3* or *sst8*) for each amino acid, aiding the model in learning structural class probabilities at each position within the sequence.

Seq2Seq Model Architecture



```python
# Define the model
class Seq2SeqModel(nn.Module):
    def __init__(self, input_size, output_size, hidden_size=100):
        super(Seq2SeqModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True, bidirectional=True)
        self.dense = nn.Linear(hidden_size * 2, output_size)  # *2 for bidirectional

    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.dense(x)
        return x
```

Figure 15: Seq2Seq Model

Training and Validation Accuracy

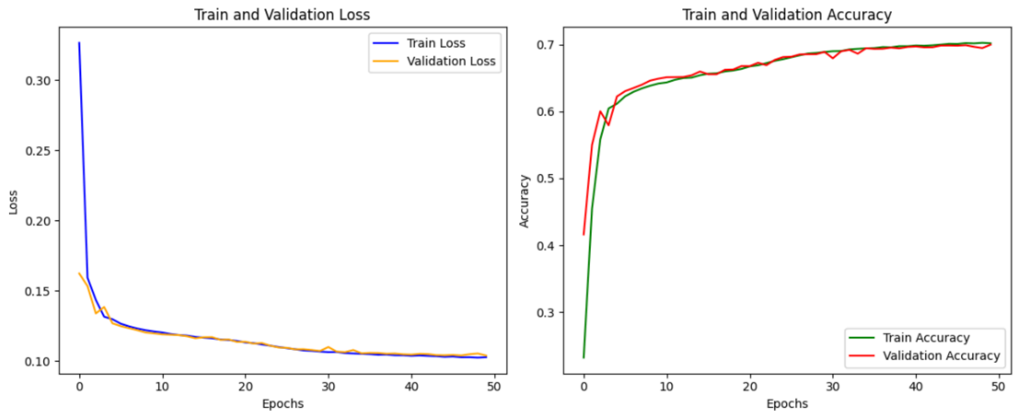
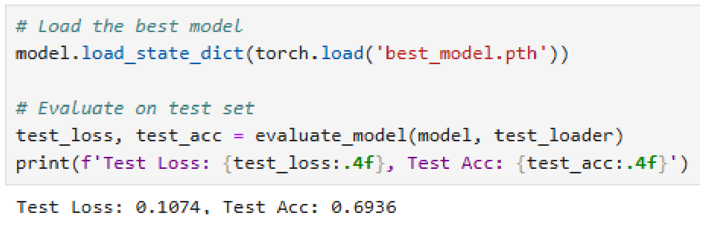


Figure 16: Train and Validation Loss and Accuracy for Q3 Structure Prediction

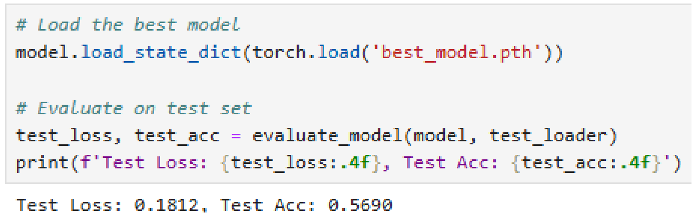
Results



```python
# Load the best model
model.load_state_dict(torch.load('best_model.pth'))

# Evaluate on test set
test_loss, test_acc = evaluate_model(model, test_loader)
print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.4f}')
```

```
Test Loss: 0.1074, Test Acc: 0.6936
```

Figure 17: Seq2Seq Q3 Test Accuracy

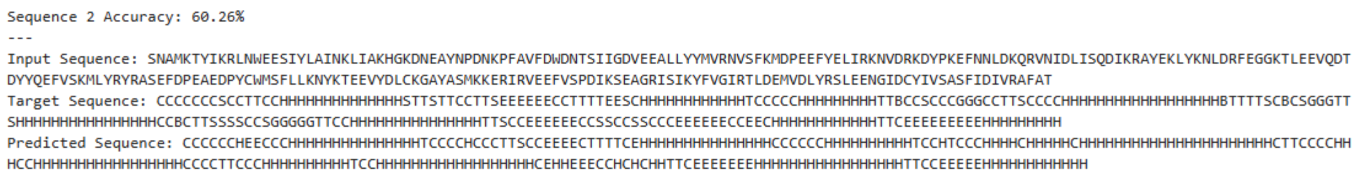


```python
# Load the best model
model.load_state_dict(torch.load('best_model.pth'))

# Evaluate on test set
test_loss, test_acc = evaluate_model(model, test_loader)
print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.4f}')
```

```
Test Loss: 0.1812, Test Acc: 0.5690
```

Figure 18: Seq2Seq Q8 Test Accuracy

The Seq2Seq model is able to obtain an overall test accuracy of 69.36% and 56.90% for predicting the Q3 structure and Q8 structure respectively. An example prediction for Q8 is shown below.



```
Sequence 2 Accuracy: 60.26%
---
Input Sequence: SNAMKTYIKRLNWEESIYLAINKLIAKHGKDNEAYNPDNKPFAVFDWDNTSIIGDVEEALLYYMVRNVSFKMDPEEFYELIRKNVDRKDYPKEFNNLDKQRVNIDLISQDIKRAYEKLYKNLDRFEGGKTLEEVQDT
DYYQEFVSKMLYRYRASEFDPEAEDPYCWMSFLLKNYKTEEVYDLCKGAYASMKKERIRVEEFVSPDIKSEAGRISIKYFVGIRTLDEMVDLYRSLEENGIDCYIVSASFIDIVRAFAT
Target Sequence: CCCCCCCSCCTTCCHHHHHHHHHHHHHHHSTTSTTCCTTSEEEEEECCTTTTEESCHHHHHHHHHHHTCCCCCHHHHHHHHHHTTBCCSCCCGGGCCTTSCCCCHHHHHHHHHHHHHHHHHBTTTTSCBCSGGGTT
SHHHHHHHHHHHHHHHHCCBCTTSSSSCCSGGGGGTTCCHHHHHHHHHHHHHHHTTSCCEEEEEECCSSCCSSCCCEEEEEECCEECHHHHHHHHHHHHTTCEEEEEEEEEHHHHHHHHH
Predicted Sequence: CCCCCCHEECCCHHHHHHHHHHHHHHHTCCCCHCCCTTSCCEEEECTTTTCEHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHTCCHTCCCHHHHCHHHHHCHHHHHHHHHHHHHHHHHHHHHHCTTCCCCHH
HCCHHHHHHHHHHHHHHHHHCCCCTTCCCHHHHHHHHHHTCCHHHHHHHHHHHHHHHHHHCEHHEEECCHCHCHHTTCEEEEEEEHHHHHHHHHHHHHHHHTTCCEEEEEHHHHHHHHHHHH
```

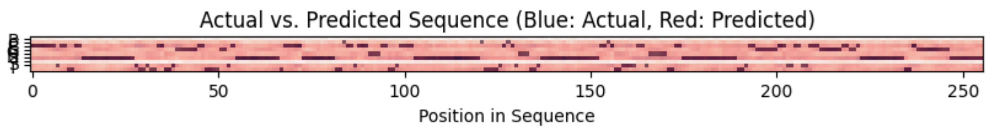


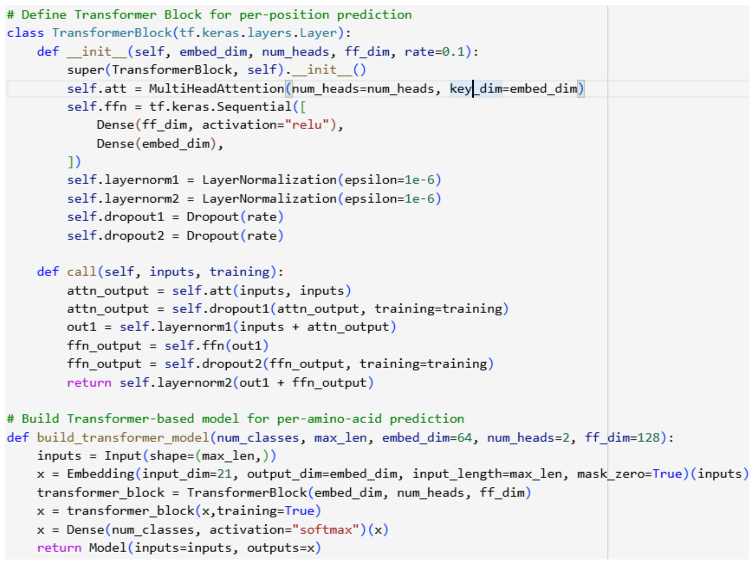
Figure 19: Example Prediction of a Q8 Sequence

This outcome suggests that the bidirectional LSTM effectively leverages contextual information, improving the model's accuracy over baseline methods. However, initial results showed that simple architectures like LSTM may still face limitations in capturing the full complexity required for Q8 secondary structure prediction. For improved performance, we will explore moving to a Transformer-based model to model both short- and long-range dependencies.

3.5.1.2. Transformer

Transformer Block Architecture

The Transformer model provides a highly flexible approach to handle sequential data, particularly in natural language processing. Unlike traditional RNNs or CNNs, the Transformer uses a self-attention mechanism to manage both short-range and long-range dependencies. This is essential in tasks like protein structure prediction, where amino acids' properties depend on distant interactions within the sequence.



```python
# Define Transformer Block for per-position prediction
class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)

# Build Transformer-based model for per-amino-acid prediction
def build_transformer_model(num_classes, max_len, embed_dim=64, num_heads=2, ff_dim=128):
    inputs = Input(shape=(max_len,))
    x = Embedding(input_dim=21, output_dim=embed_dim, input_length=max_len, mask_zero=True)(inputs)
    transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
    x = transformer_block(x,training=True)
    x = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=inputs, outputs=x)
```

Figure 20: Transformer Model

Multi-Head Self-Attention	Computes attention scores for each position in the sequence, enabling our model to focus on important dependencies between amino acids. Each attention head learns unique patterns, providing diverse perspectives on the relationships within the sequence.
Feed-Forward Network (FFN)	Applied after the self-attention layer, enabling complex transformations to the attention output.
Residual Connections and Layer Normalisation	These components enhance gradient flow and stabilise training. Residual connections prevent issues like vanishing or exploding gradients, while layer normalisation ensures stable and efficient learning from long sequences.

Novel Transformer Architecture for Protein Prediction

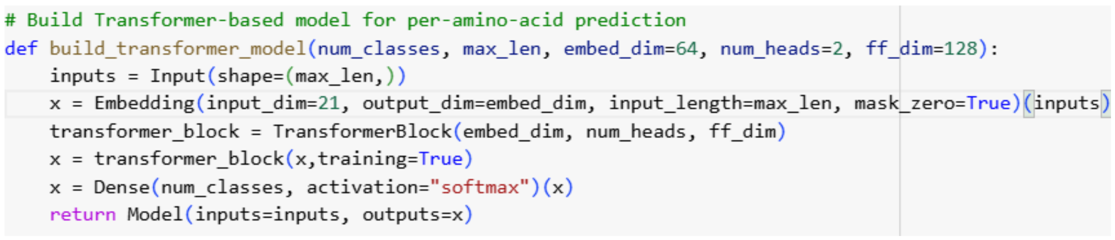
To adapt the standard Transformer to protein secondary structure prediction, we introduced specific adjustments aimed to effectively capture the nuanced dependencies within protein sequences.

Self-Attention Mechanism	This mechanism enables the model to assess interactions between distant amino acids, allowing each "head" to focus on different sequence regions, which helps capture multi-level relationships without sequential processing constraints. For example, when predicting the structure of an amino acid at position *i*, the self-attention mechanism can examine amino acids at distant positions *j* and *k* if they provide relevant structural context. Unlike RNNs, which process sequences step-by-step, the self-attention mechanism evaluates all positions simultaneously, enabling faster computation and overcoming issues like vanishing gradients.
FFN	Following self-attention, the FFN adds non-linear transformations that enhance the model's understanding of intricate patterns within the sequence, aiding in accurate structural predictions.
Residual Connections and Layer Normalisation	These architectural elements improve training stability and help in preserving important features across layers, enhancing our model's ability to learn from deep representations without losing critical information.
Output Layer for	Designed to predict each amino acid's secondary structure (Q3 or Q8) independently,

Per-Position Predictions	enabling our model to handle protein sequences of varying lengths.

3.5.2. Features Representation

We also incorporated feature representation to transform the input amino acid sequences into dense, informative vectors that capture the biochemical properties and relationships between amino acids.



```
# Build Transformer-based model for per-amino-acid prediction
def build_transformer_model(num_classes, max_len, embed_dim=64, num_heads=2, ff_dim=128):
    inputs = Input(shape=(max_len,))
    x = Embedding(input_dim=21, output_dim=embed_dim, input_length=max_len, mask_zero=True)(inputs)
    transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
    x = transformer_block(x,training=True)
    x = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=inputs, outputs=x)
```

Figure 21: Feature Representation Code

In Figure 21, each amino acid in the sequence is passed through an embedding layer, *Embedding*, where *input_dim=21* accounts for 20 amino acids plus a padding token, and *output_dim* (e.g., 64) represents the dimension of the learned embeddings.

Embedding Layer for Amino Acids	Each amino acid is mapped to a dense vector of specified dimensions (e.g., 64). Unlike one-hot encoding, where each amino acid is represented by a binary vector with minimal information, embeddings allow our model to represent amino acids in a continuous space. In this space, similar amino acids can cluster together, allowing the model to recognize patterns and interactions relevant to secondary structure prediction. Through training, our model learns relationships between amino acids based on their positions within sequences, allowing the embeddings to represent not just individual amino acids but also their roles in different structural contexts.
Self-Attention as a Contextual Feature Extractor	The self-attention mechanism effectively represents how each amino acid is influenced by others across the sequence. Long-range dependencies are important in protein structure prediction, as interactions between distant amino acids can be crucial for determining secondary structure. The attention mechanism assigns weights that represent the importance of each amino acid relative to others in the sequence. These weights serve as interpretive features that illustrate how the model identifies structural dependencies within a protein sequence.

By combining embeddings with self-attention, the model can learn intricate sequence relationships, greatly improving its ability to predict secondary structures based on both local and long-range amino acid interactions.

3.5.3. Comparative Analysis

Our initial experiments employed simpler models such as CNNs, LSTMs, and Seq2Seq architectures. While CNNs are effective in capturing local patterns, they struggle with the long-range dependencies critical for accurate secondary structure prediction. LSTMs can process sequential data effectively but are limited in depth and often face vanishing gradient issues, particularly with longer sequences. These limitations highlighted the need for a more robust and flexible architecture. Using a Transformer-based model with detailed feature representation, including learned embeddings and self-attention, has shown to be highly effective for predicting protein structures. Our model not only addresses the limitations of simpler approaches but also enhances our understanding of the amino acids that contribute to structural determination.This approach marks a major step forward in bioinformatics, as it aligns well with the natural complexity of protein folding and structure, providing a robust and scalable solution for similar challenges.

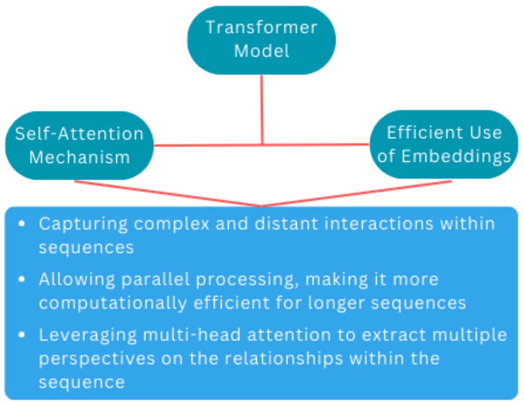
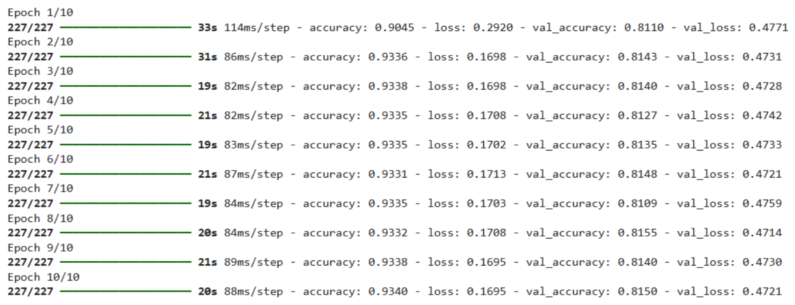

Transformer Model
Self-Attention Mechanism
Efficient Use of Embeddings
Capturing complex and distant interactions within sequences
Allowing parallel processing, making it more computationally efficient for longer sequences
Leveraging multi-head attention to extract multiple perspectives on the relationships within the sequence

Figure 22: The Importance of Advanced Architecture and Feature Representation



```
Epoch 1/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 33s 114ms/step - accuracy: 0.9045 - loss: 0.2920 - val_accuracy: 0.8110 - val_loss: 0.4771
Epoch 2/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 31s 86ms/step - accuracy: 0.9336 - loss: 0.1698 - val_accuracy: 0.8143 - val_loss: 0.4731
Epoch 3/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 19s 82ms/step - accuracy: 0.9338 - loss: 0.1698 - val_accuracy: 0.8140 - val_loss: 0.4728
Epoch 4/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 21s 82ms/step - accuracy: 0.9335 - loss: 0.1708 - val_accuracy: 0.8127 - val_loss: 0.4742
Epoch 5/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 19s 83ms/step - accuracy: 0.9335 - loss: 0.1702 - val_accuracy: 0.8135 - val_loss: 0.4733
Epoch 6/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 21s 87ms/step - accuracy: 0.9331 - loss: 0.1713 - val_accuracy: 0.8148 - val_loss: 0.4721
Epoch 7/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 19s 84ms/step - accuracy: 0.9335 - loss: 0.1703 - val_accuracy: 0.8109 - val_loss: 0.4759
Epoch 8/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9332 - loss: 0.1708 - val_accuracy: 0.8155 - val_loss: 0.4714
Epoch 9/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 21s 89ms/step - accuracy: 0.9338 - loss: 0.1695 - val_accuracy: 0.8140 - val_loss: 0.4730
Epoch 10/10
227/227 ━━━━━━━━━━━━━━━━━━━━ 20s 88ms/step - accuracy: 0.9340 - loss: 0.1695 - val_accuracy: 0.8150 - val_loss: 0.4721
```

Figure 23: Transformer Model Training Output

3.6. Results Analysis

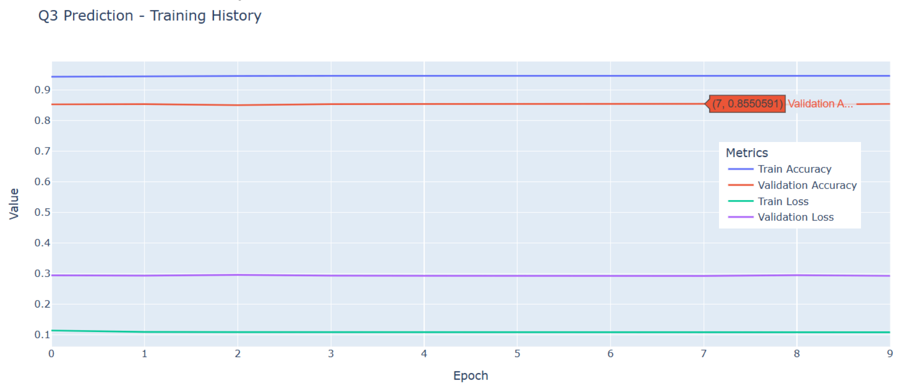


Figure 24: Training and Validation Accuracy and Loss Across Epochs for Q3 Prediction

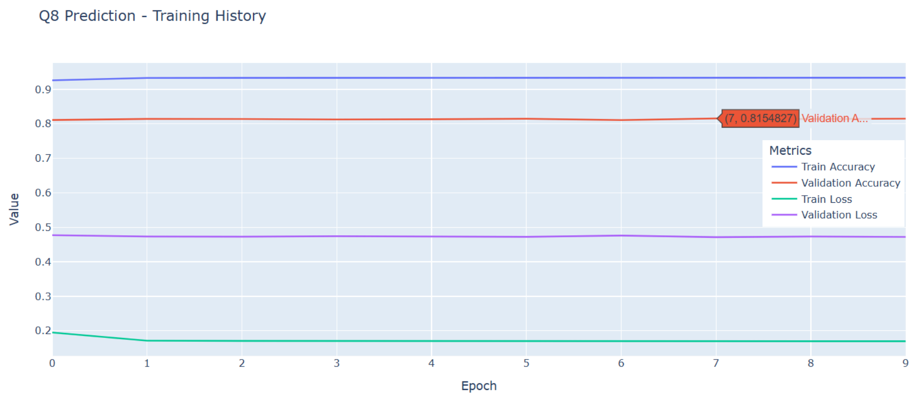


Figure 25: Training and Validation Accuracy and Loss Across Epochs for Q8 Prediction

From Figure 24, both training and validation accuracies reach high values, with little difference between them, suggesting that the model is fitting well to the Q3 task. The loss curves for both training and validation data remain stable, with minimal divergence, indicating that the model is neither overfitting nor underfitting significantly on Q3 predictions. This stability reflects the model's ability to generalise well in the simpler Q3 task, where there are fewer classes to distinguish, making it less challenging than Q8.

From Figure 25, we observe that the training accuracy reaches a high level early in the training, while validation accuracy stabilises at a slightly lower level, suggesting potential overfitting. The validation loss remains relatively stable throughout the training process. These patterns indicate that the model might be capturing patterns in the training data that do not generalise as well to unseen data.

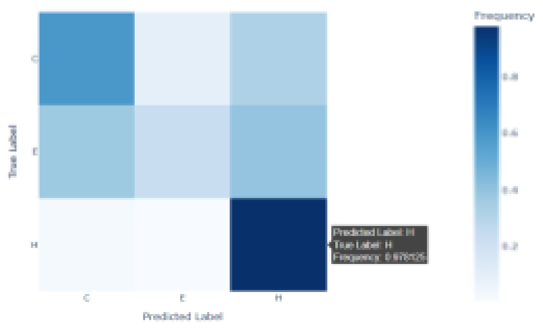


Figure 26: Confusion Matrix for Q3

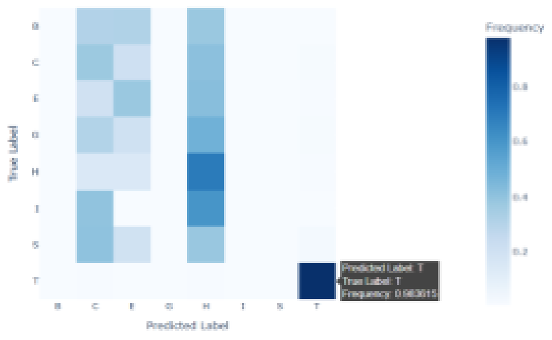


Figure 27: Confusion Matrix for Q8

As shown in Figure 26, the Q3 matrix shows a strong diagonal trend, especially for the H class, which has the highest frequency of accurate predictions. This indicates that the model performs well for the helical structure (H), which is generally more consistent and easier to predict. Whereas for Q8, as shown in Figure 27, certain classes, particularly T and H, show high frequencies along the diagonal, indicating accurate predictions for these structures.

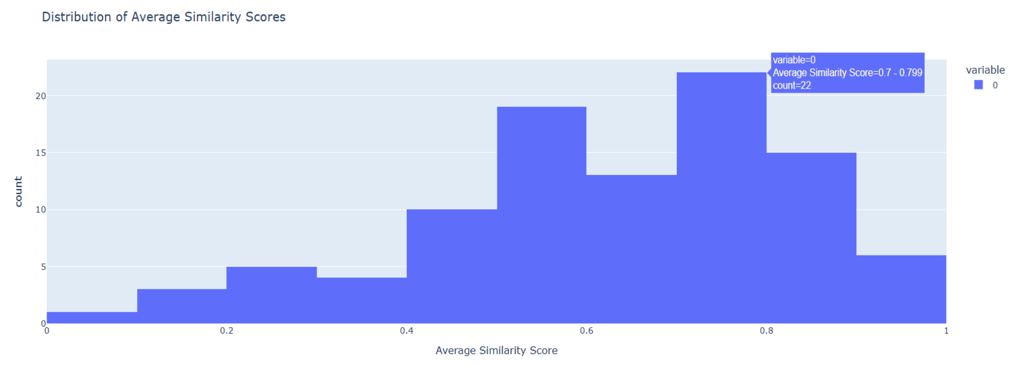


Figure 28: Distribution of Average Similarity Scores for the Predictions

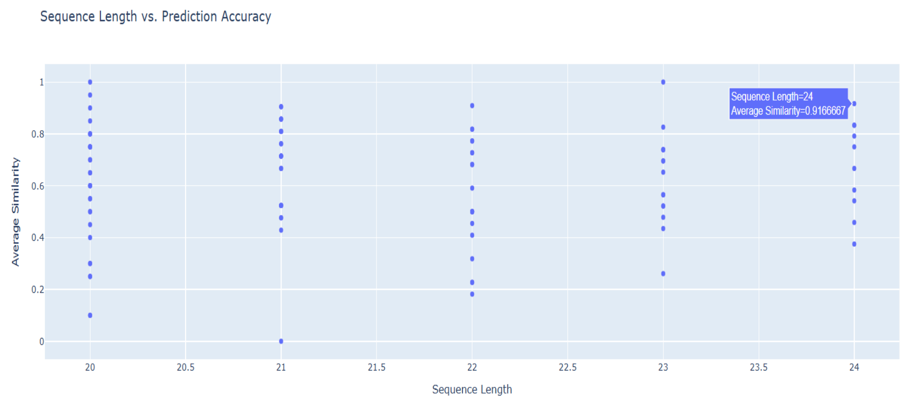


Figure 29: Relationship between Sequence Length and the Average Similarity Score

From Figure 28, we see that the similarity distribution is centred around mid-range values (0.4 to 0.7), with a significant number of sequences in the 0.6-0.8 range, showing moderate accuracy. A smaller number of sequences have similarity scores above 0.8, indicating high predictive accuracy for these sequences.

From Figure 29, we observe that most sequence lengths fall within a narrow range (around 20 to 24 amino acids). The average similarity varies widely within each sequence length group, suggesting that the model's performance is not strongly dependent on sequence length in this dataset. Some sequences achieve high similarity (close to 1.0), indicating accurate predictions, while others have low similarity, indicating more challenging sequences.

Predicting secondary structure (sst3 and sst8 values) from primary sequence (seq)

```
Sequence: KPSSPPEELKFQCGQKTLRPRFK
True Q3 Structure: CCCCCCCCCCCCCCCCCCCCCCC
Predicted Q3 Structure: CCCCCCCCCCCCCCCCCCCCCCC
Q3 Similarity: 100.00%
True Q8 Structure: CCCCCCCCCCCCTTCCCCCCCCC
Predicted Q8 Structure: CCCCCCCCCCCCCCCCCCCCCCC
Q8 Similarity: 91.30%
Average Similarity: 95.65%
```

```
Sequence: GPPPPPGPPPPPGPPPPPGL
True Q3 Structure: CCCCCCCCCCCCCCCCCCCC
Predicted Q3 Structure: CCCCCCCCCCCCCCCCCCCC
Q3 Similarity: 100.00%
True Q8 Structure: CCCCCSSCCCCCCCCCCCCC
Predicted Q8 Structure: CCCCCCCCCCCCCCCCCCCC
Q8 Similarity: 90.00%
Average Similarity: 95.00%
```

```
Sequence: MMAPANNPFGAPPAQVNNPF
True Q3 Structure: CCCCCCCCCCCCCCCCECCC
Predicted Q3 Structure: CCCCCCCCCCCCCCCCCCCC
Q3 Similarity: 95.00%
True Q8 Structure: CCCCCCCCCCCCCCCCBCCC
Predicted Q8 Structure: CCCCCCCCCCCCCCCCCCCC
Q8 Similarity: 95.00%
Average Similarity: 95.00%
```

```
Sequence: RRLRPRRPRLPRPRPRPRPRP
True Q3 Structure: CCCCCCCCEECCCCCCCCCCC
Predicted Q3 Structure: CCCCCCCCCCCCCCCCCCCCC
Q3 Similarity: 90.48%
True Q8 Structure: CCCCCCCCEECCCCCCCCCCC
Predicted Q8 Structure: CCCCCCCCCCCCCCCCCCCCC
Q8 Similarity: 90.48%
Average Similarity: 90.48%
```

```
Sequence: GEEEYQEMLENLREAEVKKNA
True Q3 Structure: CHHHHHHHHHHHHHHHHHHCC
Predicted Q3 Structure: CHHHHHHHHHHHHHHHHHHHH
Q3 Similarity: 90.48%
True Q8 Structure: CHHHHHHHHHHHHHHHHHHTC
Predicted Q8 Structure: HHHHHHHHHHHHHHHHHHHHH
Q8 Similarity: 85.71%
Average Similarity: 88.10%
```

References

Alfrandom. (2022). Protein secondary structure [Data set]. Kaggle. https://www.kaggle.com/datasets/alfrandom/protein-secondary-structure

Baek, M., DiMaio, F., Anishchenko, I., Dauparas, J., Ovchinnikov, S., Lee, G. R., ... & Baker, D. (2021). Accurate prediction of protein structures and interactions using a three-track neural network. Science, 373(6557), 871-876. https://doi.org/10.1126/science.abj8754

Brigato, L., & Iocchi, L. (2020, October 25). A close look at Deep Learning with small data. arXiv.org. https://arxiv.org/abs/2003.12843

Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., ... & Rost, B. (2021). ProtTrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. Scientific Reports, 11(1), 1-17. https://doi.org/10.1038/s41598-021-03819-2

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. International Conference on Learning Representations (ICLR). Retrieved from https://arxiv.org/abs/2001.04451

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., ... & Rives, A. (2023). Evolutionary-scale prediction of atomic level protein structure with a language model. Science, 379(6637), 1151-1157. https://doi.org/10.1126/science.ade2574

Plotly Technologies Inc. (2023). Plotly: Collaborative data science and data visualization. Retrieved from https://plotly.com

Rao, R. M., Meier, J., Sercu, T., Ovchinnikov, S., & Rives, A. (2021). Transformer protein language models are unsupervised structure learners. Proceedings of the International Conference on Learning Representations (ICLR). Retrieved from https://arxiv.org/abs/2003.03425

R. Hu, L. Rui, P. Zeng, L. Chen and X. Fan, "Text Sentiment Analysis: A Review," 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Chengdu, China, 2018, pp. 2283-2288, Text Sentiment Analysis: A Review | IEEE Conference Publication | IEEE Xplore

Sameh, A., Ozgur T, A comparative assessment of sentiment analysis and star ratings for consumer reviews, International Journal of Information Management, Volume 54, 2020, 102132, ISSN 0268-4012. https://doi.org/10.1016/j.ijinfomgt.2020.102132

Secondary Structure Analysis - Creative Proteomics. (2024). Creative-Proteomics.com. https://www.creative-proteomics.com/proteindrug/secondary-structure-analysis.htm

Wang, S., Peng, J., Ma, J. et al. Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields. Sci Rep 6, 18962 (2016). https://doi.org/10.1038/srep18962

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020, July 14). Huggingface’s transformers: State-of-the-art natural language processing. arXiv.org. https://arxiv.org/abs/1910.03771

Xu, H., et al. ,Pre-trained models: Past, present and future, AI Open, Volume 2, 2021, Pages 225-250, ISSN 2666-6510, https://doi.org/10.1016/j.aiopen.2021.08.002.

Zhang, Y., & Wallace, B. C. (2021). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(3), 789-801. https://doi.org/10.1109/TPAMI.2019.2913388