SPORTS ANALYSIS

Project Proposal

Course: CSCE 5290 Natural Language Processing, Spring 2022

Instructor: *Dr. Sayed Khushal Shah (sayed.shah@unt.edu)*

Member:

Sreeja Bellamkonda *(sreejabellamkonda@my.unt.edu)*

Github Link:

https://github.com/SreejaBellamkonda/NLP--Sport-Analysis.git

INTRODUCTION

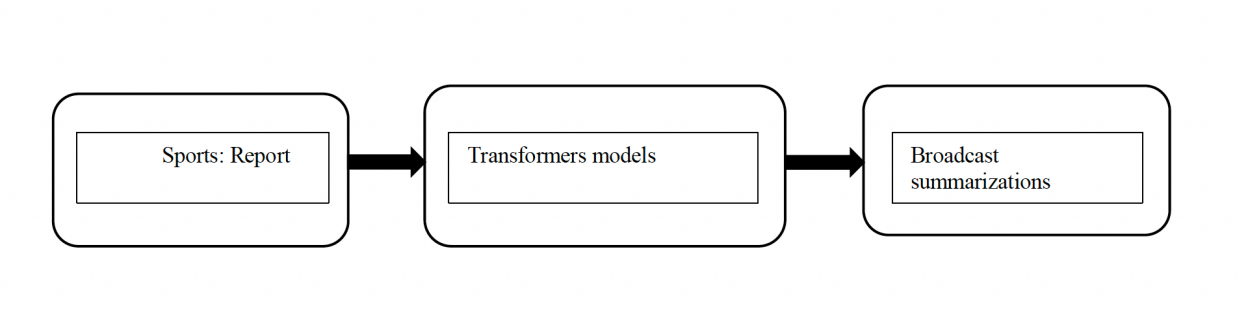


Fig: Model Diagram

Every day, tens of thousands of athletic activities take place. Despite their pattern form, most of the sports news (results of sports tournaments) is written by hand. We want to see if it's feasible to produce news based on the broadcast - a series of remarks that characterize the game in real-time in this project. This problem is classified as a summarization problem, and it is solved by using the Natural Language Processing Models.

Approaches that are both extractive and abstract We'll start with extractive models and then go on to using the Bidirectional Encoder Representations from Transformers (BERT) as an encoder and text augmentation using a thesaurus. Other types of encoders, on the other hand, do not exhibit considerable gains.

The basic idea of our project is to build a model that can generate summarizations of sports commentary.

BACKGROUND

Related work:

1. SPORTS COMMENTARY SUMMARIZATION

This article is written by the Anirudh vyas. The main goal of this project is to generate these types of summarized notes of cricket commentaries and connect this to the statistics. Anyways the results are written on notes, or the patterns can be seen in the handwritten summaries. To change this to auto summarization he used the Natural Language Processing and Machine learning techniques. He attempted summarization extraction where using Natural Language processing is the biggest task itself. As input, he took the sports commentary live and changed it into a script that is in the text format for the news generation. Analyzing, He also took the news documents with their corresponding commentaries. After processing it gives the output concerning the input. The output gives the subset of the text in the form of the summarization forming in the sentences from the commentary. Each sentence is selected so that it can cover all the information and should match the commentary accordingly.

2. TEXT SUMMARIZATION FOR TAMIL ONLINE SPORTS NEWS USING NLP

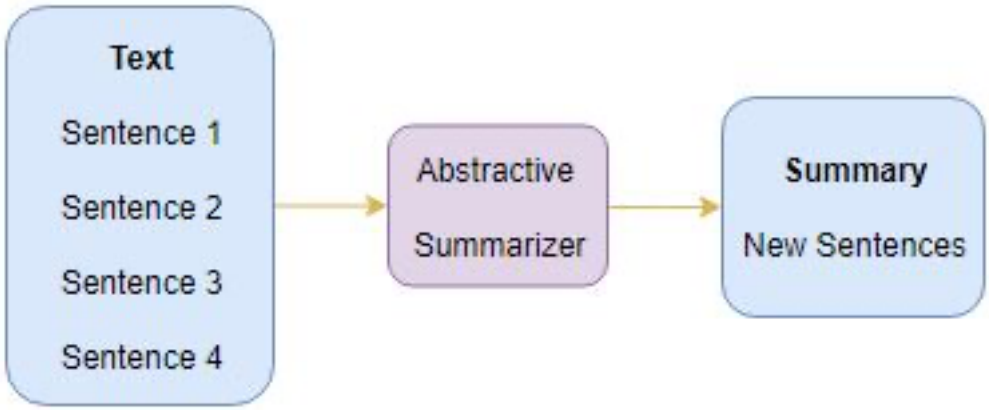
This article is published by the Thevatheepan Priyadharshan; Sagara Sumathipala. They said that text summarization is a critical challenge in natural language comprehension and information retrieval. People have paid greater attention to automatic text summary these days since it saves time in decision-making processes, even in everyday life. Deep learning methods are currently receiving more attention than traditional approaches. The major goal of this project is to offer an approach to handle the problem of summarizing Tamil sports news by automatically creating

extractive summaries from news data using Natural Language Processing (NLP) and a generic stochastic artificial neural network. Features such as sentence position, paragraph position, number of named entities, word frequency, inverted document frequency, and number of numerals are used. The feature matrix for each phrase is constructed, and the Restricted Boltzmann Machine is utilized to enhance those features while improving accuracy without losing the fundamental meaning of the text. The ROUGE tool package is used to analyze the recall, accuracy, and F-measure for the summary generated by both the human experts and the algorithm.

DATASET

Description of the data set:

The data set is collected from the internet where we have two types of data sets raw_dataset and the reference_summaries. In raw_dataset we have the 31 files which contain the commentaries and the refrence_summaries have the 31 text files.



Design Features:

```
def expand_contractions(text, contraction_map=contractions_dict):
    # Using regex for getting all contracted words
    contractions_keys = '|'.join(contraction_map.keys())
    contractions_pattern = re.compile(f'({contractions_keys})', flags=re.DOTALL)
                           (parameter) contraction: Any
    def expand_match(contraction):
        # Getting entire matched sub-string
        match = contraction.group(0)
        expanded_contraction = contraction_map.get(match)
        if not expand_contractions:
            print(match)
            return match
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text


expand_contractions("y'all can't expand contractions i'd think")
```

```
'you all can not expand contractions id think'
```

The above figure shows that the using the regex method for all the contracted words. Here the data is collected and read the sentences by using the clean text and removing the punctuations we are also using the lowercase conversion. Then remove stop words from the given dataset. In the Data collection, we have taken the dataset from the internet and cleaned the text and all the processing is done before the implementation.

ANALYSIS OF DATA

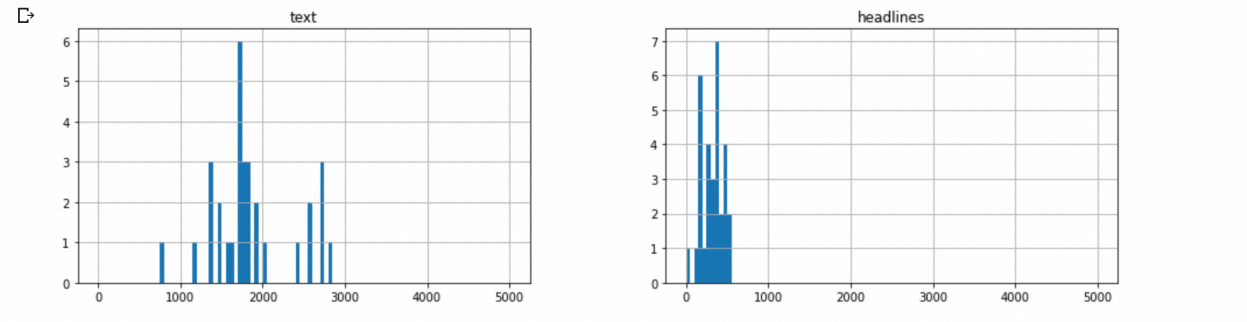
Data Preprocessing:

As we told before we are using the text classification methods using the Natural Language Processing.

- First, collect the dataset which has the CSV and the text format.
- Then they run the dataset. We use the regex method for getting all the contracted words in the dataset files.

- Then convert them to lowercase sentences and expand the sentences.
- After the conversion removes the punctuations from the sentences of the text. And remove the numbers from the text.
- Then remove the stop words from the text.
- After all the classifications clean the text for good performance.
- Then finally save the cleaned dataset into the drive.

GRAPH MODEL:



In the above figure, we can see that the count of the text and the headlines are plotted. The first belongs to the text and the second graph belongs to the headlines. We also checked how many rows and columns are there. We can also read the maximum text and the summary length by the data preprocessing form.

We also printed the text which is the most used word in the text and the summary. The below two figures are the two files in which the first figure is that it shows the words most used in the headlines and the second figure is the most common words used in the text file.

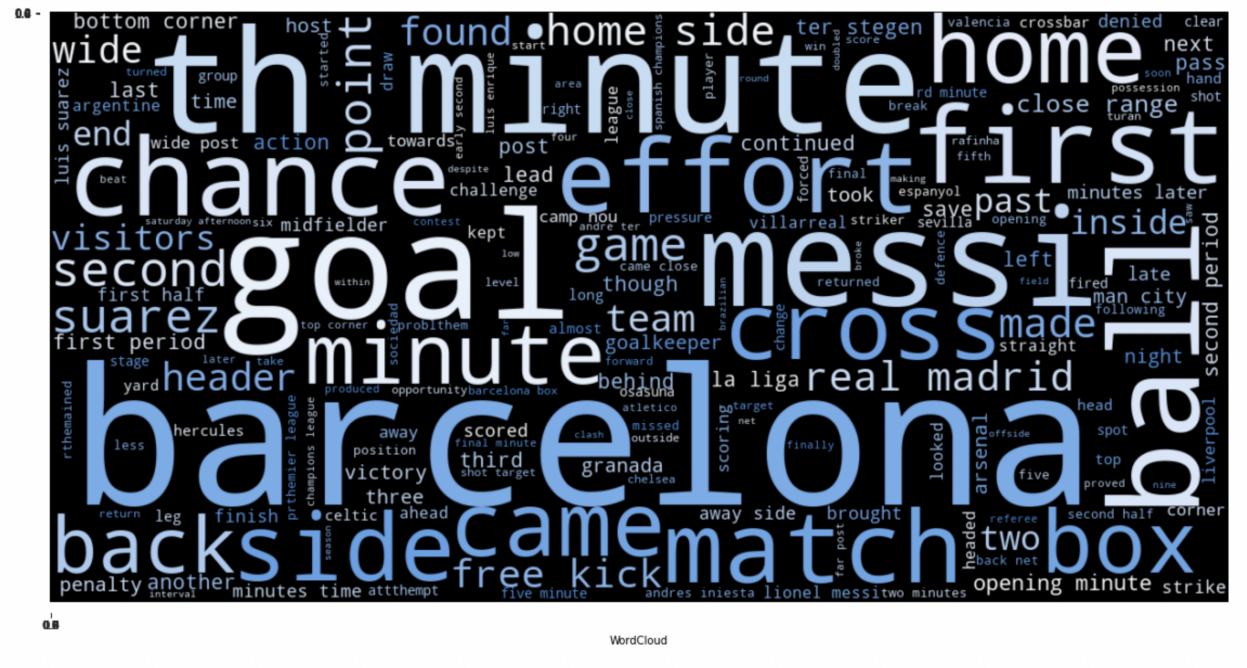


Fig: Most commonly used words of Headlines

Fig: Most commonly used words of Text

IMPLEMENTATION

In the implementation, we are using the three different modelings they are :

Seq2Seq model with just LSTMs

In this model, we have both the encoder and the decoder which is having the only LSTM

```
# =====================
#  Encoder
# =====================
encoder_input = Input(shape=(max_text_len, ))

# encoder embedding layer
encoder_embedding = Embedding(
    x_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
    trainable=False
)(encoder_input)

# encoder lstm 1
encoder_lstm1 = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.4
)
encoder_output1, state_h1, state_c1 = encoder_lstm1(encoder_embedding)

# encoder lstm 2
encoder_lstm2 = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.4
)
encoder_output, *encoder_final_states = encoder_lstm2(encoder_output1)
```

Fig: Encoder

The above shown is the encoder LSTM. LSTM is nothing but the room which has the information to control the sequence of where data comes into, which stores, and the network is built.

```
# ----------------------
#  Decoder
# ======================

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=True
)
decoder_embedding = decoder_embedding_layer(decoder_input)

# decoder lstm 1
decoder_lstm = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.4
)
decoder_output, *decoder_final_states = decoder_lstm(
    decoder_embedding, initial_state=encoder_final_states
)

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)
```

Fig: Decoder

The above shown is the decoder which is where the text is encoded and then decodes the text where the text is converted to the headlines in the form of the summaries. This way the three models are used with the encoding and the decoding method.

```
#  Model
# =====================
model = Model([encoder_input, decoder_input], decoder_output)
model.summary()

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
        'decoder': decoder_final_states
    },
    'layers': {
        'decoder': {
            'embedding': decoder_embedding_layer,
            'last_decoder_lstm': decoder_lstm,
            'dense': decoder_dense
        }
    }
```

The above figure shows the model which is having both the encoder and the decoder output.

Seq2Seq model with Bidirectional LSTMs

```
# =====================
#  Encoder
# =====================
encoder_input = Input(shape=(max_text_len, ))

# encoder embedding layer
encoder_embedding = Embedding(
    x_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
    trainable=False,
    name='encoder_embedding'
)(encoder_input)

# encoder lstm1
encoder_bi_lstm1 = Bidirectional(
    LSTM(
        return_sequences: Any
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_1'
    ),
    name='encoder_bidirectional_lstm_1'
)
encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 = encoder_bi_lstm1(
    encoder_embedding
)
encoder_bi_lstm1_output = [
    encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
]

# encoder lstm 2
```

Fig: Encoder

The above figure shows the bidirectional LSTM which has both the encoder and the decoder the figure describes the encoder with the bidirectional LSTM which means it will have the information from both sides may be from the backward or the forward side.

```
# =====================
#  Decoder
# =====================

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

decoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.2,
        name='decoder_lstm_1'
    ),
    name='decoder_bidirectional_lstm_1'
)
decoder_output, *decoder_final_states = decoder_bi_lstm(
    decoder_embedding, initial_state=encoder_final_states
    # decoder_embedding, initial_state=encoder_final_states[:2]
)  # taking only the forward states
```

Fig: Decoder

The above figure shows the decoder of the bidirectional LSTM which can send the information from both sides.

```
# =====================
#  Model
# =====================
model = Model([encoder_input, decoder_input], decoder_output, name='seq2seq_model_with_bidirectional_lstm')
model.summary()

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
        'decoder': decoder_final_states
    },
    'layers': {
        'decoder': {
            'embedding': decoder_embedding_layer,
            'last_decoder_lstm': decoder_bi_lstm,
            'dense': decoder_dense
        }
    }
```

The above figure shows the model output where it provides the encoder and the decoder output with the cross-entropy method.

Seq2Seq model with hybrid architecture.

The third model is the hybrid architecture where the encoder has the bidirectional LSTM, and the decoder just has the LSTM.

```
# =====================
#  Encoder
# =====================
encoder_input = Input(shape=(max_text_len, ))

# encoder embedding layer
encoder_embedding = Embedding(
    x_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
    trainable=False,
    name='encoder_embedding'
)(encoder_input)

# encoder lstm1
encoder_bi_lstm1 = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_1'
    ),
    name='encoder_bidirectional_lstm_1'
)
encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 = encoder_bi_lstm1(
    encoder_embedding
)
encoder_bi_lstm1_output = [
    encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
]

# encoder lstm 2
```

Fig: Encoder

The above figure shows the encoder which acts as the bidirectional LSTM which gives the input from both sides.

```
#  Decoder
# =====================

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

decoder_lstm = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.2,
    name='decoder_lstm_1'
)
decoder_output, *decoder_final_states = decoder_lstm(
    decoder_embedding, initial_state=encoder_final_states[:2]
)  # taking only the forward states

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)
```

Fig: Decoder

The above figure shows the decoder which uses just the LSTM. Which it gets and gives the output from the direction and stores the information in a room.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 2800)]       0           []

 embedding (Embedding)          (None, 2800, 300)    1480500     ['input_1[0][0]']

 input_2 (InputLayer)           [(None, None)]       0           []

 lstm (LSTM)                    [(None, 2800, 240),  519360      ['embedding[0][0]']
                                 (None, 240),
                                 (None, 240)]

 embedding_1 (Embedding)        (None, None, 300)    603300      ['input_2[0][0]']

 lstm_1 (LSTM)                  [(None, 2800, 240),  461760      ['lstm[0][0]']
                                 (None, 240),
                                 (None, 240)]

 lstm_2 (LSTM)                  [(None, None, 240),  519360      ['embedding_1[0][0]',
                                 (None, 240),                     'lstm_1[0][1]',
                                 (None, 240)]                     'lstm_1[0][2]']

 time_distributed (TimeDistribu  (None, None, 2011)  484651      ['lstm_2[0][0]']
 ted)

==================================================================================================
Total params: 4,068,931
Trainable params: 2,588,431
Non-trainable params: 1,480,500
__________________________________________________________________________________________________
```

Fig: Model Output

The above figure shows the output of the model where it shows the param, trained params and non-trained params.

```
Epoch 1/50
1/1 [==============================] - 24s 24s/step - loss: 7.6078 - accuracy: 0.2007 - val_loss: 7.5475 - val_accuracy: 0.5598 - lr: 0.0010
Epoch 2/50
1/1 [==============================] - 2s 2s/step - loss: 7.5366 - accuracy: 0.3626 - val_loss: 7.2683 - val_accuracy: 0.5591 - lr: 0.0010
Epoch 3/50
1/1 [==============================] - 2s 2s/step - loss: 7.2534 - accuracy: 0.3622 - val_loss: 6.2741 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 4/50
1/1 [==============================] - 2s 2s/step - loss: 6.5081 - accuracy: 0.3642 - val_loss: 5.2097 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 5/50
1/1 [==============================] - 2s 2s/step - loss: 5.8058 - accuracy: 0.3583 - val_loss: 4.5721 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 6/50
1/1 [==============================] - 2s 2s/step - loss: 5.3695 - accuracy: 0.3583 - val_loss: 4.0282 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 7/50
1/1 [==============================] - 2s 2s/step - loss: 5.0629 - accuracy: 0.3583 - val_loss: 3.7206 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 8/50
1/1 [==============================] - 2s 2s/step - loss: 4.8576 - accuracy: 0.3583 - val_loss: 3.6606 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 9/50
1/1 [==============================] - 2s 2s/step - loss: 4.7955 - accuracy: 0.3583 - val_loss: 3.5961 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 10/50
1/1 [==============================] - 2s 2s/step - loss: 4.9481 - accuracy: 0.3584 - val_loss: 3.6098 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 11/50
1/1 [==============================] - 2s 2s/step - loss: 4.7344 - accuracy: 0.3585 - val_loss: 3.3769 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 12/50
1/1 [==============================] - 2s 2s/step - loss: 4.5800 - accuracy: 0.3584 - val_loss: 3.3094 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 13/50
1/1 [==============================] - 2s 2s/step - loss: 4.5143 - accuracy: 0.3587 - val_loss: 3.2302 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 14/50
1/1 [==============================] - 2s 2s/step - loss: 4.4598 - accuracy: 0.3586 - val_loss: 3.2152 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 15/50
1/1 [==============================] - 2s 2s/step - loss: 4.4355 - accuracy: 0.3608 - val_loss: 3.1675 - val_accuracy: 0.5578 - lr: 0.0010
Epoch 16/50
1/1 [==============================] - 2s 2s/step - loss: 4.4023 - accuracy: 0.3593 - val_loss: 3.1767 - val_accuracy: 0.5611 - lr: 0.0010
```

Fig: Training Model

The above is the training model. We trained the model accordingly

RESULTS

The below graph shows the profit and loss and accuracy of the LSTM Model. After some time, the disparity between its validation and training accuracy on the training set and evaluated on the testing set is virtually zero, indicating that the model has not succumbed to overfitting.

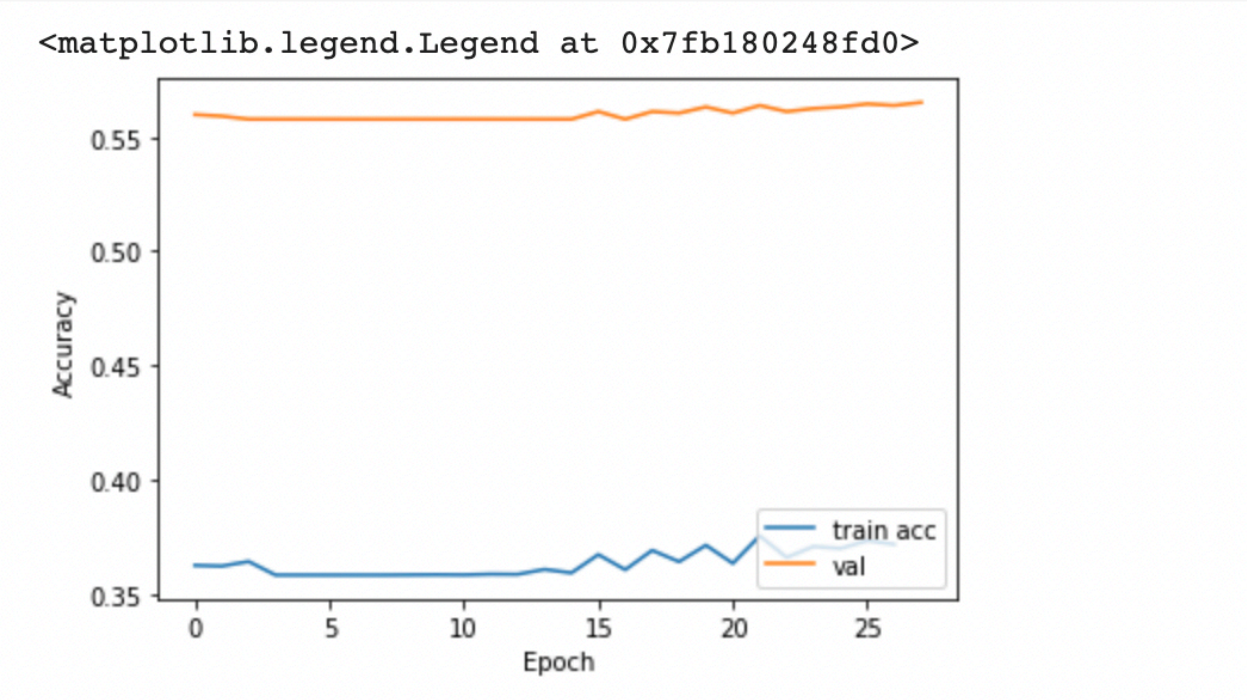


Fig: Accuracy

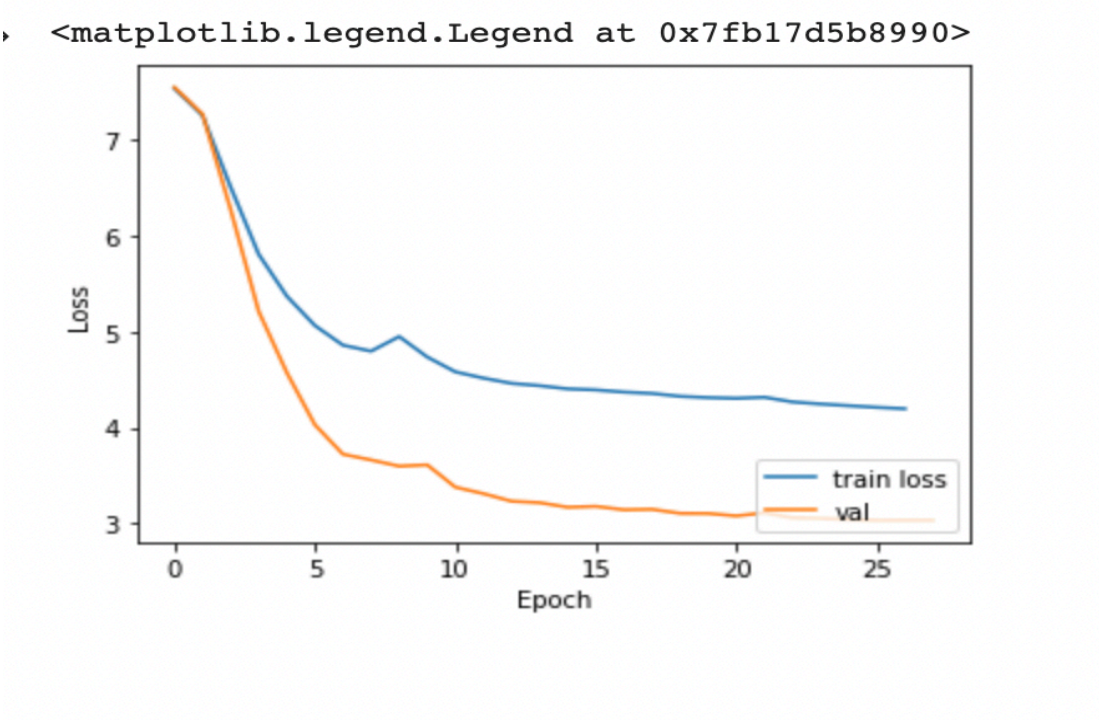


Fig: Loss

WORK TO DO

For increment 2 going to implement some other models and train the model which gives the accurate values and the graphs.

REFERENCES

1. Theanirudhvyas. “Theanirudhvyas/Sports-Commentary-Summarization: Creating News Articles from Cricket and Football Commentaries.” *GitHub*, https://github.com/theanirudhvyas/Sports-Commentary-Summarization.
2. Engdahl, Sylvia. “Blogs.” *Amazon*, Greenhaven Press/Gale, 2008, https://aws.amazon.com/blogs/machine-learning/enhance-sports-narratives-with-natural-language-generation-using-amazon-sagemaker/.
3. “Sports Analytics: Analyzing Cricket Commentary.” *Analytics Vidhya*, 25 June 2020, https://www.analyticsvidhya.com/blog/2020/02/sports-analytics-generating-actionable-insights-using-cricket-commentary/.
4. “Text Summarization for Tamil Online Sports News Using NLP.” *IEEE Xplore*, https://ieeexplore.ieee.org/document/8736154.
5. Theanirudhvyas. “Theanirudhvyas/Sports-Commentary-Summarization: Creating News Articles from Cricket and Football Commentaries.” *GitHub*, https://github.com/theanirudhvyas/Sports-Commentary-Summarization.