Part A: Neural Machine Translation [30 points]

1. Task 1: Implementing the encoder [5 marks].

Code showed as below:

```
    """
    Task 1 encoder

    Start
    """
    # The train encoder
    # (a.) Create two randomly initialized embedding lookups, one for the source, another for the target.
    print('Task 1(a): Creating the embedding lookups...')
    embeddings_source = Embedding(input_dim=self.vocab_source_size, output_dim=self.embedding_size, mask_zero=True)(source_words)
    embeddings_target = Embedding(input_dim=self.vocab_target_size, output_dim=self.embedding_size, mask_zero=True)(target_words)

    # (b.) Look up the embeddings for source words and for target words. Apply dropout each encoded input
    print('\nTask 1(b): Looking up source and target words...')
    source_words_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_source)
    target_words_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_target)

    # (c.) An encoder LSTM() with return sequences set to True
    print('\nTask 1(c): Creating an encoder')
    encoder_outputs, encoder_state_h, encoder_state_c =  LSTM(self.hidden_size, return_sequences=True, return_state=True, recurrent_dropout=self.hidden_dropout_rate)
(source_words_embeddings)
    """
    End Task 1
    """
```

a. This step initializes two embedding lookups, one for the source language and another for the target language, using the Embedding layer. The Embedding layer transforms each word in the vocabulary into a dense vector of a specified dimension (output_dim), capturing semantic relationships between words. The input_dim parameter specifies the size of the vocabulary, and mask_zero=True indicates that zeros in the input should be treated as padding values to be ignored, which is useful for handling sequences of varying lengths.

b. Look up the embeddings for source words and target words, followed by the application of Dropout. Dropout is a regularization technique that helps prevent overfitting by randomly dropping a fraction of the output units during training (self.embedding_dropout_rate specifies the dropout rate). This randomness helps ensure that the model does not become overly dependent on any one feature.

c. An LSTM layer is utilized to create the encoder. LSTM (Long Short-Term Memory) networks are a type of recurrent neural network (RNN) capable of capturing long-distance dependencies in sequence data. `return_sequences=True` is configured to return the hidden states for all time steps, `return_state=True` is configured to return the final time step's hidden and cell states, which can be used to initialize the decoder's initial state. `self.hidden_size` specifies the number of neurons in the LSTM layer.

2. Task 2: Implementing the decoder and the inference loop [12.5 marks].

Code showed as below:

```
"""
Task 2 decoder for inference

Start
"""
# Task 2 (a.) Get the decoded outputs
print('\n Putting together the decoder states')
# get the inititial states for the decoder, decoder_states
# decoder states are the hidden and cell states from the training stage
decoder_states =  [decoder_state_input_h, decoder_state_input_c]
# use decoder states as input to the decoder lstm to get the decoder outputs, h, and c for test time inference
decoder_outputs_test,decoder_state_output_h, decoder_state_output_c = decoder_lstm(target_words_embeddings, initial_state=decoder_states)

# Task 2 (b.) Add attention if attention
if self.use_attention:
    decoder_outputs_test = decoder_attention([encoder_outputs_input, decoder_outputs_test])

# Task 2 (c.) pass the decoder_outputs_test (with or without attention) to the decoder dense layer
decoder_outputs_test = decoder_dense(decoder_outputs_test)

"""
End Task 2
"""
```

a. Setting decoder_states to obtain the initial state of the decoder helps the decoder generate output sequences that are relevant to the input sequence. Using decoder states as input to the decoder LSTM to get the decoder outputs, h, and c for test time inference means that each decoding step is updated based on the output and state of the previous time step.
b. Passing the encoder outputs (encoder_outputs_input) and decoder outputs (decoder_outputs_test) as inputs to the attention layer (decoder_attention) to generate weighted decoder outputs. The attention mechanism allows the model to dynamically focus on different parts of the input sequence while generating each target word, thereby improving the quality of translation or text generation.
c. Passing the decoder_outputs_test (with or without attention) to the decoder dense layer. Thus, the model is able to select the word with the highest probability as the output for each time step.

Model BLEU score on test set: 4.97:

A BLEU score of 4.97 indicates that while the model can probably capture some of the language's structure, it struggles to produce translations that closely match the reference translations.

3. Adding attention [12.5 marks].

Code showed as below:

```
"""
Task 3 attention

Start
"""

# Transpose decoder_outputs
decoder_outputs_transposed = K.permute_dimensions(decoder_outputs, (0, 2, 1))

# Compute luong_score
luong_score = K.batch_dot(encoder_outputs, decoder_outputs_transposed)

# Apply softmax
attention_weights = K.softmax(luong_score, axis=1)

# Expand dimensions for element-wise multiplication
attention_weights_expanded = K.expand_dims(attention_weights, axis=-1)
encoder_outputs_expanded = K.expand_dims(encoder_outputs, axis=2) # shape is [batch_size, max_source_sent_len, 1, hidden_size]

# Compute encoder_vector
weighted_encoder_outputs = encoder_outputs_expanded * attention_weights_expanded
encoder_vector = K.sum(weighted_encoder_outputs, axis=1)


"""
End Task 3
"""
```

The Luong attention score essentially measures the similarity between each encoder state and the decoder state, resulting in a matrix of scores that represent how much attention should be paid to each encoder state at each decoder timestep. The weighted encoder outputs are calculated by performing element-wise multiplication of the expanded encoder outputs with the expanded attention weights, effectively focusing on the most relevant parts of the input sequence for each step of the decoding process. The result is then summed across the encoder sequence length axis, resulting in a single vector for each sample in the batch.

Model BLEU score on test set: 8.56:

An improvement to a BLEU score of 8.56 with the addition of an attention mechanism indicates a significant enhancement in the quality of the translations. Attention mechanisms allow the model to focus on different parts of the input sequence when predicting each word of the output sequence, effectively addressing the problem of long-range dependencies and improving the context understanding.

Part B: Using the Pre-trained BERT models [15 marks]

1. Task 1: Data pre-processing [4 marks].

The code for dataset setup and conversion to index and mask sequences:

```
# your code goes here
train_reviews=[item[0] for item in train]
train_aspects=[item[1] for item in train]
dev_reviews=[item[0] for item in dev]
dev_aspects=[item[1] for item in dev]
test_reviews=[item[0] for item in test]
test_aspects=[item[1] for item in test]

# Tokenization for Training Dataset
x_train_review_int, x_train_review_masks, _ = tokenize(train_reviews, tokenizer)
x_train_aspect_int, x_train_aspect_masks, _ = tokenize(train_aspects, tokenizer)

# Tokenization for Development (Validation) Dataset
x_dev_review_int, x_dev_review_masks, _ = tokenize(dev_reviews, tokenizer)
x_dev_aspect_int, x_dev_aspect_masks, _ = tokenize(dev_aspects, tokenizer)

# Tokenization for Testing Dataset
x_test_review_int, x_test_review_masks, _ = tokenize(test_reviews, tokenizer)
x_test_aspect_int, x_test_aspect_masks, _ = tokenize(test_aspects, tokenizer)
```

the code with correct use of separator tokens between text and aspect:

```
# Tips:
# 1) Use the special token [SEP] to concatenate sentences and aspect words
# 2) Make sure they are paded/truncated to a max length

# your code goes here
def combineSentAndAspect(sentences, aspect_words, tokenizer, max_length=128):
    combined_texts = [sentence + " [SEP] " + aspect for sentence, aspect in zip(sentences, aspect_words)]
    input_ids, attention_masks = [], []

    for text in combined_texts:
        inputs = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=max_length,
            padding='max_length',
            truncation=True,
            return_attention_mask=True
        )
        input_ids.append(inputs['input_ids'])
        attention_masks.append(inputs['attention_mask'])

    return np.array(input_ids), np.array(attention_masks)


x_train_int, x_train_masks = combineSentAndAspect(train_reviews, train_aspects, tokenizer)
x_dev_int, x_dev_masks = combineSentAndAspect(dev_reviews, dev_aspects, tokenizer)
x_test_int, x_test_masks = combineSentAndAspect(test_reviews, test_aspects, tokenizer)
```

2. Task 2: Basic classifiers using BERT: Model 1 and Model 2 [3 marks].

Mode11: loss: 6.7893 - accuracy: 0.2994

Model2: loss: 0.6328 - accuracy: 0.4543

a. Model 1 has a much higher loss compared to Model 2. This indicates that Model 1's predictions are, on average, further away from the actual labels. Model 2's lower loss suggests it is making predictions closer to the actual labels.

b. These differences suggest that Model 2 is better suited for the task at hand, possibly due to a more appropriate model architecture, better tuning, more relevant features being used. Model 1's high loss and low accuracy could point to issues such as overfitting to the training data, underfitting due to a too simple model, or not being trained sufficiently.

3. Task 3: Advanced classifier using BERT: Model 3 [8 marks].

Model3: loss: 0.7231 - accuracy: 0.2994

Model 3 has a loss of 0.7231, which is higher than Model 2 but significantly lower than Model 1, placing it in the middle in terms of predictive accuracy. Both Model 1 and Model 3 have the same accuracy (0.2994), which is significantly lower than Model 2. This suggests that nearly 30% of their predictions are correct. Model 3 shows that while it has learned to some extent, it might still be lacking in certain aspects that prevent it from achieving performance comparable to Model 2. It might benefit from further optimization or adjustments in its approach.

LSTM model:

```
def create_lstm_on_bert():
    input_ids_in = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype=tf.int32, name='input_ids')
    input_masks_in = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype=tf.int32, name='input_masks')

    bert_embeddings = get_BERT_layer()
    embedded_sent = bert_embeddings(input_ids_in, attention_mask=input_masks_in)[0]

    # Adding an LSTM layer on top of the BERT embedding
    lstm_layer = LSTM(units=100)(embedded_sent)

    label = Dense(3, activation='softmax', kernel_initializer='glorot_uniform')(lstm_layer)

    return Model(inputs=[input_ids_in, input_masks_in], outputs=label, name='Model3_LSTM')


use_tpu = True
if use_tpu:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.TPUStrategy(tpu)
    with strategy.scope():
      model3 = create_lstm_on_bert()
      optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
      model3.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

else:
    model3 = create_lstm_on_bert()
    model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

This model can be used for NLP tasks such as sentiment analysis and topic classification, where BERT is utilized to extract textual features, and LSTM is employed to capture the sequential dependencies between these features. The final dense layer serves to make the ultimate classification decision based on the output from the LSTM layer.

Part C: Social Media Processing [20 marks]

1. **Build and evaluate two regression models for humour rating prediction (a) without special pre-processing applied and (b) with special pre- processing applied to the training data (Tasks 1 and 2 in the script). [3 marks]**

   a. output of the model.summary() ：

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_token (InputLayer)    [(None, 128)]             0         []

 masked_token (InputLayer)   [(None, 128)]             0         []

 tf_distil_bert_model (TFDi  TFBaseModelOutput(last_hid  6636288   ['input_token[0][0]',
 stilBertModel)              den_state=(None, 128, 768)  0          'masked_token[0][0]']
                             , hidden_states=None, atte
                             ntions=None)

 global_average_pooling1d_m  (None, 768)               0         ['tf_distil_bert_model[0][0]']
 asked (GlobalAveragePoolin
 g1DMasked)

 dense (Dense)               (None, 16)                12304     ['global_average_pooling1d_mas
                                                                 ked[0][0]']

 dense_1 (Dense)             (None, 1)                 17        ['dense[0][0]']

=================================================================================================
Total params: 66375201 (253.20 MB)
Trainable params: 66375201 (253.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________________________
```

   b. the mean squared error (MSE) scores over the test set for both models.

   without special pre-processing:
   Test loss: 0.014797660522162914
   Test MSE: 0.014505450613796711

   with special pre- processing:
   Test loss: 0.012011639773845673
   Test MSE: 0.011655214242637157

   c. the histograms of the predicted and true humour ratings for the first model

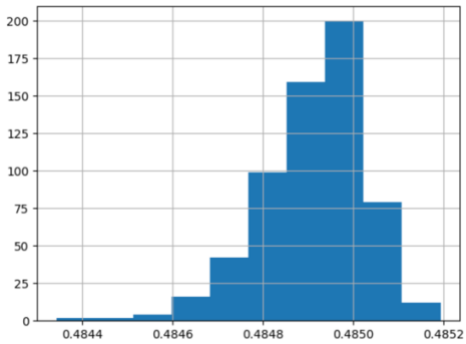
Predicted Histogram: The predicted values are concentrated in a very narrow range around 0.4848. True Histogram: The true values have a wider distribution and span from approximately 0.1 to 0.7.

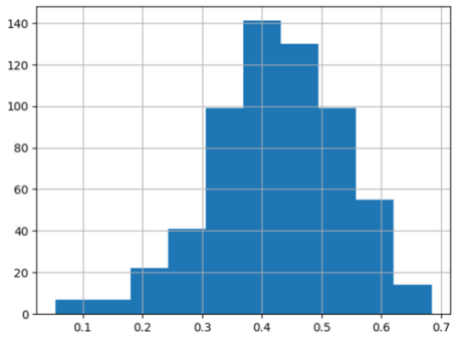
The model's predictions are not aligning well with the actual distribution of the true values. There's a clear discrepancy in the variability and the central tendency (mean) of the distributions. The model's narrow range of predicted values suggests that it is not effectively learning from the data.

The difference in the spread of the distributions might also mean that the predicted scores are too conservative and not sensitive enough to the features that contribute to higher or lower humour ratings.

Without special pre-processing, it seems that the model is struggling to accurately predict the humour ratings, which might require a review of the model's structure, the features being used, and the pre-processing steps that might be necessary to improve its predictions.

predicted: **true:**





d. The **differences in performance** between the two models (a and b) and discuss why they occur:

The results indicate that the model with special pre-processing achieved lower test loss and lower mean squared error (MSE) on the test set compared to the model without special pre-processing. This suggests that the model with special pre-processing performs better in predicting humour ratings.

The reason for the difference in performance between the two models lies in the impact of special pre-processing techniques on the training data. Special pre-processing may help reduce noise, remove unnecessary information, and enhance useful features, thereby improving the predictive capability of the model. This could result in the model with special pre-processing achieving lower loss and higher accuracy on the test set.

2. **Augment the training data twice for the regression model (Task 1 above) with the two following methods (a) synonym replacement from Wordnet and (b) deletion of random words (Task 3). [3 marks].**

   a. training epochs (with loss values), as well as the MSE scores over the test set for both models.

      augment_with_synonyms:

      Test loss: 0.012889405712485313
      Test MSE: 0.012569302693009377

      deletion of random words:

      Test loss: 0.015859611332416534
      Test MSE: 0.015396928414702415

      augment_with_synonyms and deletion of random words:

      Test loss: 0.024885181337594986
      Test MSE: 0.024347852915525436

   b. Describe the differences in performance between the models (a) and (b) and hypothesise why they occur.

      The model augmented with synonyms outperforms the one with random word deletion in terms of both test loss and test MSE. This suggests that synonym augmentation preserves the semantic integrity of the text better than random deletion.

      Effectiveness of Synonym Replacement: Synonym replacement likely maintains the overall meaning of the sentences while introducing sufficient variability in the training data, aiding the model in learning more robust representations.

      Impact of Random Word Deletion: Random deletion of words introduces noise into the training data. On one hand, it can force the model to learn from incomplete information, potentially making it more robust. On the other hand, excessive or poorly chosen deletions might remove critical information from the text, hindering the model's ability to learn effective representations.

3. **One way to improve the performance is to perform ensembling of three regressors. Try this (Task 4) and see if it improves the performance of the model from Task 1. Use non-augmented data. [6 marks]**

   a. training epochs (with loss values), as well as the MSE scores over the test set

Ensemble Test MSE: 0.0126

b. Describe the **difference** in performance to the model from Task 1.

Comparing the MSE values from Task 1 and Task 4, the ensembled model from Task 4 has a lower MSE (0.0126) than the model from Task 1 (0.014505450613796711). This suggests that the ensemble method improved the model's performance. Ensemble methods typically combine the predictions from multiple regression models to improve the overall prediction accuracy, as they can balance out the errors from individual models.

4. Build and evaluate the two following regression models using 50% of the original training data: (a) multi-task learning simultaneously with the second regression task of offense rating and (b) single-task regressor (re-build the model from Task 1). [8 marks]

a. output of the model.summary()

Layer (type)	Output Shape	Param #	Connected to
input_token (InputLayer)	[(None, 128)]	0	[]
masked_token (InputLayer)	[(None, 128)]	0	[]
tf_distil_bert_model_4 (TF DistilBertModel)	TFBaseModelOutput(last_hidden_state=(None, 128, 768), hidden_states=None, attentions=None)	66362880	['input_token[0][0]', 'masked_token[0][0]']
global_average_pooling1d_masked_4 (GlobalAveragePooling1DMasked)	(None, 768)	0	['tf_distil_bert_model_4[0][0]']
dense_8 (Dense)	(None, 16)	12304	['global_average_pooling1d_masked_4[0][0]']
out_reg1 (Dense)	(None, 1)	17	['dense_8[0][0]']
out_reg2 (Dense)	(None, 1)	17	['dense_8[0][0]']

Total params: 66375218 (253.20 MB)
Trainable params: 66375218 (253.20 MB)
Non-trainable params: 0 (0.00 Byte)

b. training epochs (with loss values), as well as the MSE scores over the test set for both models.

Model a:

Test loss: 0.059403643012046814

Test MSE: 0.018911944702267647

Model b:

Test loss: 0.014797660522162914

Test MSE: 0.014505450613796711

c. Describe the differences in performance between the two models (a and b) and discuss why they occur.

Model (b) shows better performance on its single task of humor rating prediction than Model (a). Multi-task learning models like Model (a) are inherently more complex because they are learning to predict multiple outputs simultaneously. This complexity can make the training process more challenging, especially if the tasks are quite different from each other or if one task is much harder than the other.

Model (b) might be better tuned to its single task, potentially allowing it to generalize better with the limited data it was trained on. Meanwhile, Model (a) has to generalize across two tasks, which can be more difficult, leading to higher loss and MSE.

Part D: Natural Language Generation [20 marks]

1. **Build and evaluate greedy and beam search algorithms from scratch (Tasks 1 and 2 in the script). [10 marks]**
    a. The output of both greedy and beam search algorithms (hypotheses and log probabilities)

Greedy search:

Hypotheses:

```
--------------------------------------------------------------------------------------------------------------
The cat slept on the floor, and the cat was still asleep.

"I was just trying to get her to sleep," she said. "I was trying to get her to sleep, but she was still asleep."

The cat
```

Log probability: -62.585667

beam_search:

Hypotheses:

```
--------------------------------------------------------------------------------------------------------------
The cat slept on the floor of the house.

"It was like a nightmare," she said.

"It was like a nightmare. It was like a nightmare. It was like a nightmare. It was like a nightmare. It
```

Log probability: -46.06695018656319

    b. Describe the differences in the generated hypotheses and their log probabilities and discuss why they occur.

The log probability for the hypotheses generated by greedy search is lower (-62.585667), indicating a lower confidence level in the generated text. The log probability for the hypotheses generated by beam search is higher (-46.06695018656319), indicating a higher confidence level in the generated text.

Greedy search selects the most probable token at each step without considering future consequences. As a result, it may quickly settle into a locally optimal solution but might miss globally better solutions. Beam search considers multiple hypotheses (or beams) at each step and keeps track of the most probable ones. This allows beam search to explore a larger portion of the search space and potentially find more coherent and contextually relevant sequences.

2. **Build and evaluate the beam search algorithm with a repetition penalty (Task 3). [8 marks].**
    a. The output of the beam search with the repetition penalty (hypothesis and log probability)

    Hypotheses:

    ```
    The cat slept on the floor of the house.

    "It was like a nightmare," she said of the ordeal. "It was like, 'Oh my God, I can't believe this is happening.'"

    She said the cat was
    ```

    Log probability: -60.200512200273806

    b. Describe the differences between the hypothesis generated in Task 3 to the hypothesis generated by the beam search in Task 2:

    The first hypothesis generated without a repetition penalty repeats the phrase "It was like a nightmare." multiple times. This is a common issue in sequence generation without penalties, where the model finds a locally coherent phrase and keeps repeating it because each repetition is likely given the previous one, thus falling into a loop. The log-probability of this sequence is -46.06695131392917, which suggests that despite the repetition, the model sees this sequence as relatively likely, given the search constraints.

    The second hypothesis, generated with a beam search algorithm that includes a repetition penalty, provides a more diverse and extended text. The repetition penalty discourages the model from repeating the same phrase, leading to a more coherent and natural-sounding text. However, the log-probability is lower at -60.200512200273806.

3. **Experiment and assess sampling results in terms of coherence and diversity (Task 4). [2 marks]**
    a. The outputs of the four sampling methods for the specified prompt

    Random Sampling output:

    The cat slept on the floor of the tent, she noticed that no one saw her step back in the hallway, when she walked in. It didn't help that this is when I realized that not one of that room's inhabitants has a cat.

    High temperature output:

    The cat slept on the roof (a house cat being carried up from on down); that her belly crawled. When first taken, the crescent head is found to lie quite still inside the frame. It did at first belong more as it seemed as

    Low temperature output:

    The cat slept on the floor, and the dog slept on the floor.

    The cat was a little bit older than the dog, but she was still very young. She was a little bit older than the dog, but she was still very

    Top p output:

    The cat slept on the floors to relieve stress. Before bed, he rolled for his head, looking a lot more bluish, then going back down.

    b. The differences between the obtained hypotheses in terms of coherence and diversity.

    Random Sampling output:

    The hypotheses contain disparate content, involving multiple unrelated topics, thus displaying poor coherence. And it includes some unexpected information, lacking overall coherence.

    High temperature output:

    The hypotheses are exaggerated and unrealistic, potentially containing incoherent details, thus exhibiting poor coherence. It demonstrates more diversity in content and description but lack overall coherence.

    Low temperature output:

The hypotheses are concise but lack innovation and diversity. It exhibits coherence with significant repetition in content.

Top p output:

The hypotheses are relatively coherent but may come across as too formal or stiff. They maintain stability in description but lack some innovation and unexpected content.

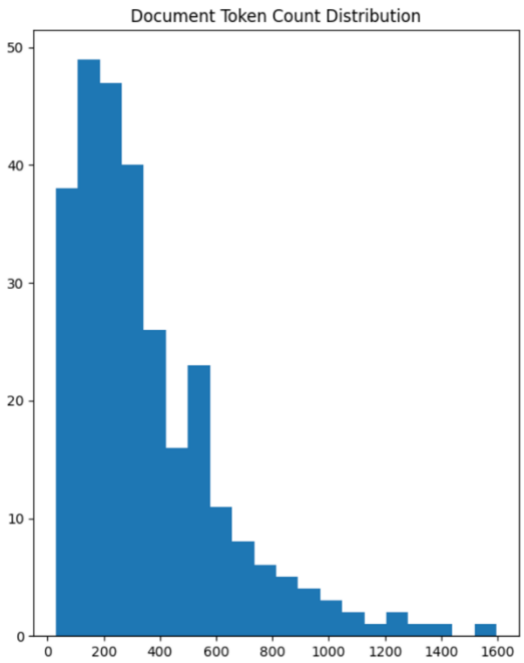
Part E: Summarisation and Data Generation [15 marks]

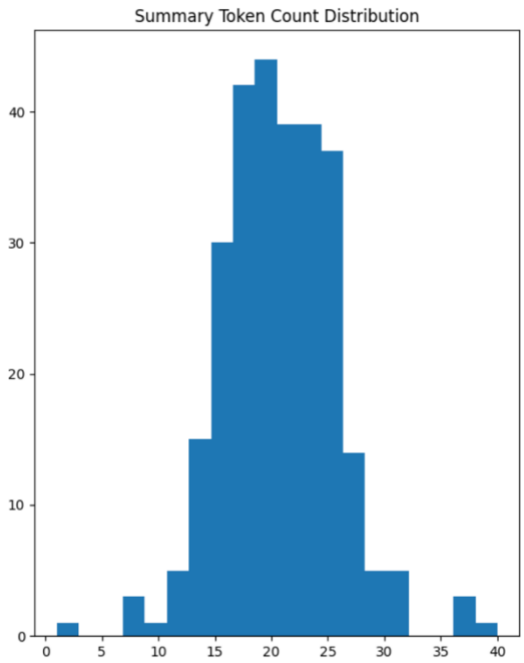
1. Analyse the XSum summarisation dataset (Task 1). [2 marks]

a. the mean count and standard deviation for input and target documents for the indicated dataset.

```
train Dataset:
Source Mean: 378.7349823321555, Source Std: 318.30072484710377
Target Mean: 21.20259128386337, Target Std: 5.2927439161150645

validation Dataset:
Source Mean: 353.90845070422534, Source Std: 267.7680545833393
Target Mean: 20.70774647887324, Target Std: 5.143963698090834
```





b. Comment on the lengths of input and target texts:

The mean length of the input texts is much longer than the mean length of the target. The substantial difference in length between source and target texts indicates that the model needs to effectively distill the most important information from a potentially large and complex source text into a concise and coherent summary. The lower standard deviation in target lengths compared to source lengths suggests that the summaries are expected to be relatively uniform in size, despite the varying lengths of the source texts.

2. Fine-tune and evaluate T5 for the XSum summarisation task (Task 2) [4 marks].

a. The final loss and validation ROUGE after one epoch of fine-tuning:

loss: 3.3958, rouge1: 17.8857, rouge2: 3.2372, rougeL: 14.3596, rougeLsum: 14.3986.

b. outputs of the fine-tuned and not fine-tuned T5 models for the sample input document, as well as their ROUGE scores, and the efficiency of fine-tuning taking those results into consideration:

outputs of the fine-tuned:

```
{'rouge1': AggregateScore(low=Score(precision=0.30434782608695654, recall=0.1794871794871795, fmeasure=0.22580645161290322),
mid=Score(precision=0.30434782608695654, recall=0.1794871794871795, fmeasure=0.22580645161290322), high=Score(precision=0.30434782608695654,
recall=0.1794871794871795, fmeasure=0.22580645161290322)),
 'rouge2': AggregateScore(low=Score(precision=0.18181818181818182, recall=0.10526315789473684, fmeasure=0.13333333333333333),
mid=Score(precision=0.18181818181818182, recall=0.10526315789473684, fmeasure=0.13333333333333333), high=Score(precision=0.18181818181818182,
recall=0.10526315789473684, fmeasure=0.13333333333333333)),
 'rougeL': AggregateScore(low=Score(precision=0.21739130434782608, recall=0.1282051282051282, fmeasure=0.16129032258064516),
mid=Score(precision=0.21739130434782608, recall=0.1282051282051282, fmeasure=0.16129032258064516), high=Score(precision=0.21739130434782608,
recall=0.1282051282051282, fmeasure=0.16129032258064516)),
 'rougeLsum': AggregateScore(low=Score(precision=0.21739130434782608, recall=0.1282051282051282, fmeasure=0.16129032258064516),
mid=Score(precision=0.21739130434782608, recall=0.1282051282051282, fmeasure=0.16129032258064516), high=Score(precision=0.21739130434782608,
recall=0.1282051282051282, fmeasure=0.16129032258064516))}
```

outputs of not fine-tuned:

```
{'rouge1': AggregateScore(low=Score(precision=0.19444444444444445, recall=0.1794871794871795, fmeasure=0.18666666666666665),
mid=Score(precision=0.19444444444444445, recall=0.1794871794871795, fmeasure=0.18666666666666665), high=Score(precision=0.19444444444444445,
recall=0.1794871794871795, fmeasure=0.18666666666666665)),
 'rouge2': AggregateScore(low=Score(precision=0.05714285714285714, recall=0.05263157894736842, fmeasure=0.05479452054794521),
mid=Score(precision=0.05714285714285714, recall=0.05263157894736842, fmeasure=0.05479452054794521), high=Score(precision=0.05714285714285714,
recall=0.05263157894736842, fmeasure=0.05479452054794521)),
 'rougeL': AggregateScore(low=Score(precision=0.08333333333333333, recall=0.07692307692307693, fmeasure=0.08), mid=Score(precision=0.08333333333333333,
recall=0.07692307692307693, fmeasure=0.08), high=Score(precision=0.08333333333333333, recall=0.07692307692307693, fmeasure=0.08)),
 'rougeLsum': AggregateScore(low=Score(precision=0.08333333333333333, recall=0.07692307692307693, fmeasure=0.08), mid=Score(precision=0.08333333333333333,
recall=0.07692307692307693, fmeasure=0.08), high=Score(precision=0.08333333333333333, recall=0.07692307692307693, fmeasure=0.08))}
```

For the not fine-tuned model, the ROUGE scores are lower. These results indicate that fine-tuning has significantly improved the model's performance in generating summaries that are closer to the target summaries:

The ROUGE-1 score, which measures the overlap of unigrams between the generated and target summaries, shows an increase in both precision and the F-measure, suggesting that the fine-tuned model is more capable of capturing the most important words.

The ROUGE-2 score, measuring the bigram overlap, also sees improvements, indicating better capture of phrase-level coherence and context in the generated summaries.

3. **Build and evaluate a T5-based model for data generation using key- words (Task 3). [9 marks]**

a. the final loss after five epochs of training: **loss:3.2446**

```
Epoch 1/5
106/106 [==============================] - 391s 3s/step - loss: 3.9563
Epoch 2/5
106/106 [==============================] - 349s 3s/step - loss: 3.5842
Epoch 3/5
106/106 [==============================] - 360s 3s/step - loss: 3.4281
Epoch 4/5
106/106 [==============================] - 375s 4s/step - loss: 3.3359
Epoch 5/5
106/106 [==============================] - 377s 4s/step - loss: 3.2446
<keras.src.callbacks.History at 0x7bf136111270>
```

b. the output of your model for the keywords provided in the script:

Hawick has been unable to restore its commercial premises in the area of Hawick, which is a multi-agency neglected and neglected business.

c. the efficiency of the data generation procedure.

The approach of selecting a random set of up to 10 unique words from each document summary as input aims to reduce the complexity of the input while still attempting to capture the essence of the document. However, This method might lead to a loss of important contextual information. The training process over 5 epochs shows a gradual decrease in loss, indicating that the model is learning from the data. The use of a custom data generation procedure, along with TensorFlow's dataset handling, seems to be effective in this context.