Part 1

A1.
How do specific aspects of customer service mentioned in Yelp reviews impact overall sentiment towards a business?

A2.
The objectives of the data analysis are centered around extracting actionable insights from Yelp reviews to enhance customer service in businesses. The primary goal is to identify and categorize key aspects of customer service, such as friendliness, speed of service, cleanliness, and product quality, that are frequently mentioned in reviews. This understanding will allow the focus to be on relevant topics during sentiment analysis. Another crucial objective is to develop and fine-tune a neural network model capable of accurately classifying the sentiment of Yelp reviews as positive, negative, or neutral. The classification will provide a baseline understanding of overall customer satisfaction and the emotional tone of the reviews.

Furthermore, the analysis aims to correlate the identified aspects of customer service with the sentiment of the reviews, determining how each aspect influences overall customer sentiment. By linking specific service aspects to sentiment, businesses can pinpoint their strengths and weaknesses. The goal is to generate clear, actionable recommendations for business improvements based on this aspect-sentiment analysis. These insights will enable businesses to make targeted enhancements in the areas that most impact customer satisfaction.

A3.
For performing a text classification task on Yelp reviews and producing useful predictions on text sequence, a Recurrent Neural Network (RNN) is highly suitable. RNN's are designed to handle sequential data and are particularly effective for natural processing language (NLP) tasks, such as text classification. RNNs can maintain context over sequences, capturing dependencies between words in a review that might be separated by many other words, which is crucial for understanding sentiment expressed in reviews. They are adept at processing sequential data, ensuring that the order of words is preserved, and their meaning is accurately interpreted.

The implementation of an RNN model involves several steps. First, the Yelp reviews need to be tokenized and converted into numerical sequences, with sequences padded to ensure uniform input length. An embedding layer is used to convert the tokenized words into dense vectors that capture semantic meaning, and pre-trained embeddings like Word2Vec can enhance performance. The core of the model is the RNN layer, which processes the embedded sequences and captures context and dependencies. After the RNN layer, a dense layer followed by an output layer with a suitable activation function, such as softmax classification, is added to generate the sentiment prediction.

The model is trained on the labeled Yelp review dataset, optimizing for metrics such as accuracy. Evaluation on a validation set and fine-tuning of hyperparameters and model architecture are essential to improve performance. Tools and libraries like Keras or TensorFlow simplify the implementation of RNN models, which NLP libraries like NLTK assist with text preprocessing and tokenization. By leveraging an RNN, the analysis can effectively capture the complex patterns in Yelp reviews, enabling accurate sentiment predictions and providing valuable insights for business improvements.

In text citations:
(“Sentiment Analysis”, n.d.)
(“Introduction to the course”, n.d.)

Part 2

B1.

Presence of unusual characters:

To ensure the quality and relevance of the text data, an analysis of unusual characters was conducted. This step is crucial for understanding the nature of the text and determining if preprocessing steps are necessary to handle special characters, emojis, or non-English characters. Upon initial inspection, a total of 1601 unusual characters were identified, including commas, periods, exclamation marks, apostrophes, asterisks, dashes, money signs, colon, and forward slashes. This combination suggests a diverse range of textual elements that may impact subsequent analysis or natural processing tasks. Addressing these characters through appropriate preprocessing techniques will be essential to maintain data integrity and improve the accuracy of the text analysis.

Vocabulary size:

Understanding the vocabulary size is essential for optimizing the design of the embedding layer within the model. With a vocabulary size of 2354 unique words identified in the dataset, it becomes imperative to tailor the embedding layer to efficiently capture and represent the semantic and nuances of these words. This foundational information ensures that the model can effectively process and interpret the diverse vocabulary present in the dataset, thereby enhancing its ability to perform accurate natural language processing tasks. By leveraging this knowledge, the model’s performance can be maximized through strategic configuration of the embedding layer and accommodation of the specific linguistic characteristics of the data.

Proposed word embedding length:

Given the relatively small vocabulary size of 2354 words in the Yelp dataset, a common practice is to set word embeddings between 100 to 300 dimensions. This range strikes a balance between adequately capturing semantic relationships and maintaining computational efficiency. Research suggests that embedding lengths between 100 and 300 dimensions are effective; longer vectors may not provide significant additional information, while shorter ones might not capture semantics robustly. This choice aims to ensure that the model captures semantic nuances effectively without unnecessary computational overhead.

Statistical justification for the chosen maximum sequence length:

Based on the analysis of review lengths, the approach to selecting the maximum sequence length involves examining the distribution across the dataset. The histogram of review lengths appears right skewed and trails off around the 25/26 review length mark, it suggests that most reviews are shorter in length, with fewer reviews extending beyond this point. In statistical terms, a right-skewed distribution suggests that shorter reviews dominate, tapering off towards longer reviews on the right side of the distribution. By focusing on metrics such as the 95th percentile of review lengths, the goal is to encompass a significant majority of the data while maintaining a balance between the performance of our models and computational efficiency. Therefore, the chosen maximum sequence length of 26 should aim to optimize model training on the dataset.

B2.

The tokenization process serves several crucial goals in natural language processing (NLP). Primarily, its purpose is to break down raw text into smaller units called tokens, which are usually words or sub words. This step is fundamental as it transforms unstructured text data into a format that machine learning algorithms can efficiently process and analyze.

During tokenization, normalization of text is often implemented to ensure consistency and improve the quality of tokenization results. This normalization typically includes steps such as converting text to lowercase, removing punctuation marks, handling special characters or symbols, and possibly stemming or lemmatizing words to reduce them to their base or root forms.

In practical terms, various libraries and packages are commonly used to perform these tasks. For instance, the 'nltk' (Natural Language Toolkit) library provides functions for tokenization, text normalization, and other preprocessing tasks. By employing tokenization and text normalization techniques like these, NLP models can effectively handle text data, improve accuracy in tasks such as sentiment analysis, and facilitate more robust and interpretable results.

Code Generate and Packages used to normalize text during the tokenization process:

```
#normalize text during the tokenization process

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

# Ensure necessary resources are downloaded
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')



# Function to tokenize and normalize text
def tokenize_normalize(text):
    # Tokenization
    tokens = word_tokenize(text)
    # Normalization
    tokens = [token.lower() for token in tokens]
    tokens = [token for token in tokens if token not in
string.punctuation]
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens

# Apply tokenization and normalization to the 'cleaned_text' column
cleaned_yelp_df['normalized_tokens'] =
cleaned_yelp_df['cleaned_text'].apply(tokenize_normalize)

# Print the DataFrame to verify the results
print(cleaned_yelp_df[['cleaned_text', 'normalized_tokens']])
```

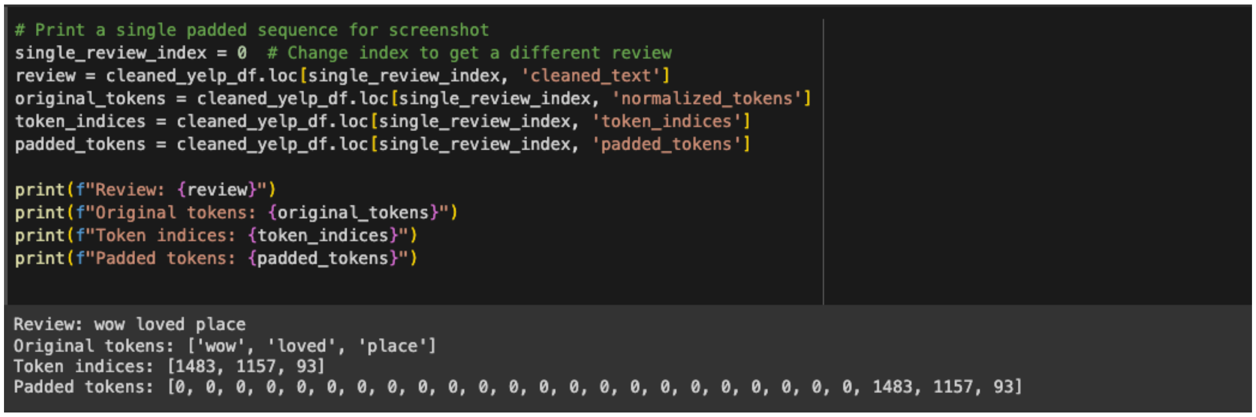
B3.

The padding process in natural language processing (NLP) is essential for standardizing the length of text sequences, ensuring that all sequences in a batch have the same length. This standardization is necessary for efficient processing by models, especially for operations like matrix multiplication in neural networks. Padding can occur either before or after the text sequence. Pre-padding, where padding tokens are added at the beginning of the sequence, is commonly used, particularly with models like Long Short-Term Memory (LSTM), which is a type of recurrent neural network (RNN). This approach helps in maintaining the sequential structure of the input data and is effective for tasks requiring LSTM's capability to capture long-term dependencies. In contrast, post-padding, which adds padding tokens at the end of the sequence, is less common but may be preferred for certain models or tasks. For this assignment, pre-padding will be implemented.

To implement padding, the 'pad_sequences' function from the 'keras.preprocessing.sequence' module is often used. By running a simple script, sequences can be padded to a specified maximum length allowing for uniform input sizes. For example, sequences can be padded to a maximum length of eight, with pre-padding being the method demonstrated. This ensures that all sequences, regardless of their original length, conform to the same size, making them suitable for batch processing in neural networks.

In summary, pre-padding is a common approach for standardizing sequence lengths. Implementing this technique allows for uniform input sizes, facilitating efficient processing by NLP models.

Screenshot of a single padded sequence:



```
# Print a single padded sequence for screenshot
single_review_index = 0  # Change index to get a different review
review = cleaned_yelp_df.loc[single_review_index, 'cleaned_text']
original_tokens = cleaned_yelp_df.loc[single_review_index, 'normalized_tokens']
token_indices = cleaned_yelp_df.loc[single_review_index, 'token_indices']
padded_tokens = cleaned_yelp_df.loc[single_review_index, 'padded_tokens']

print(f"Review: {review}")
print(f"Original tokens: {original_tokens}")
print(f"Token indices: {token_indices}")
print(f"Padded tokens: {padded_tokens}")
```

```
Review: wow loved place
Original tokens: ['wow', 'loved', 'place']
Token indices: [1483, 1157, 93]
Padded tokens: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1483, 1157, 93]
```

B4.

The sentiments will be classified into three distinct categories: positive, neutral, and negative. The choice of these categories is based on the need to capture a broad spectrum of sentiments which allows for a nuanced understanding of the sentiment expressed in the text. This ternary classification helps in differentiating between positive, neutral, and negative sentiments, which can be particularly useful in applications such as customer feedback analysis, social media monitoring, and product reviews.

To implement the ternary sentiment classification, a neural network with a multi-class classification approach will be utilized. The final dense layer of our neural network will have three output neurons, each corresponding to one of the sentiment categories: positive, neutral, and negative.

For the activation function of this final layer, the softmax function will be used. The softmax activation function is ideal for multi-class classification problems because it converts the raw output scores of the neurons into a probability distribution over the predicted output classes. Each output will represent the probability that the input text belongs to the respective sentiment category. By using the softmax activation function in the final layer, the model will output a probability distribution over the three sentiment categories, enabling accurate classification of input text into positive, neutral, or negative categories.

In summary, the sentiment analysis task will utilize three sentiment categories: positive, neutral, and negative. The final dense layer of the neural network will use the softmax activation function to produce a probability distribution over these categories, facilitating precise and interpretable multi-class sentiment classification.

B5.

Data Preparation Steps:

1. Load the raw dataset: Import the dataset into Google Colab for further processing.

2. Exploratory Data Analysis:
    - Presence of unusual characters: Check and handle emojis, non-English characters. Clean and preprocess text appropriately to remove these characters.
    - Vocabulary size: Calculate the number of unique tokens to determine the size of the vocabulary for embedding.
    - Proposed word embedding length: Typically set between 100 to 300 dimensions; 100 dimensions were chosen due to the small vocabulary size.
    - Statistical justification for maximum sequence length: Determine the appropriate sequence length using the 95th percentile of sequence lengths to minimize padding.
3. Data cleaning:
    - Remove noise: Eliminate unnecessary characters, HTML tags, and special symbols from the text.
    - Lowercase conversion: Convert all text to lowercase to ensure uniformity.
    - Tokenization: Split text into individual tokens (words or sub words).
    - Remove stop words: Optionally remove common words that may not add significant meaning to the text.
    - Lemmatization/Stemming: Reduce words to their base or root form to normalize the text.
4. Labeling: Assign sentiment categories to the text.
    - Label the text data with the appropriate sentiment categories (positive, neutral, negative).
5. Text vectorization: Convert text to numerical format using embeddings.
    - Tokenization process: Convert text into a format suitable for model input, such as sequences of integers.
    - Pre-padding process: Standardize sequence length to ensure uniform input size for the model. Padding occurs prior to the text sequence.
6. Data Splitting: Divide data into training (80%), validation (10%), and test sets (10%) using industry-standard ratios.

B6.

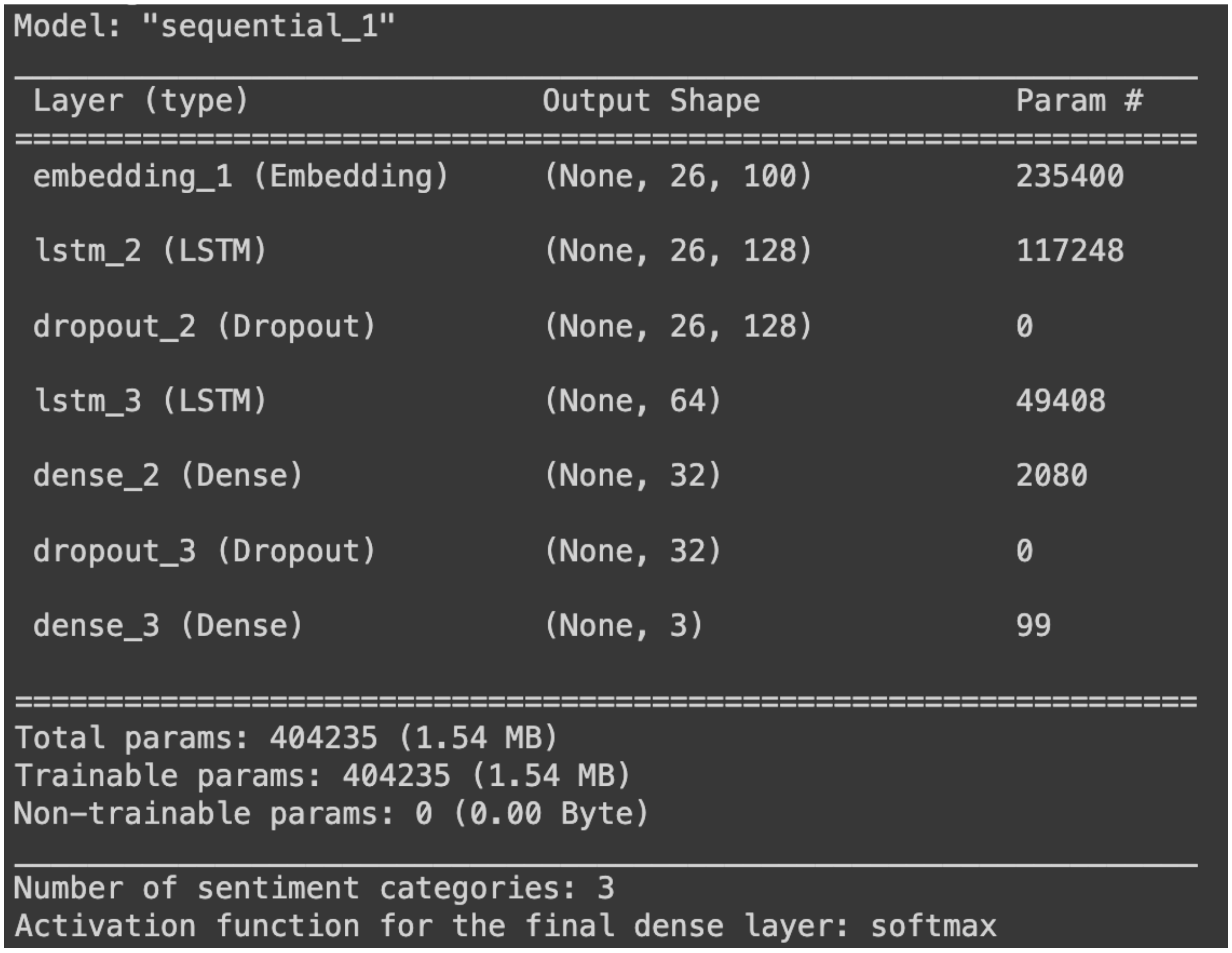
Please see a copy of the prepared data set called 'prepared_training_data_1.csv'

In text citations:
("Regular expression basics", n.d.)
("Tokenization", n.d.)
("Text cleaning basics", n.d.)
("Preparing text for modeling", n.d.)
("Classification modeling", n.d.)
("Sentiment Analysis", n.d.)
("Introduction to language models", n.d.)
("Multi-class classification models", n.d.)
(Pinecone, n.d.)
(Deepgram, n.d.)
(Data Science Stack Exchange, n.d.)

Part 3

C1.

Output of the model summary:


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 26, 100)           235400

 lstm_2 (LSTM)               (None, 26, 128)           117248

 dropout_2 (Dropout)         (None, 26, 128)           0

 lstm_3 (LSTM)               (None, 64)                49408

 dense_2 (Dense)             (None, 32)                2080

 dropout_3 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 3)                 99

=================================================================
Total params: 404235 (1.54 MB)
Trainable params: 404235 (1.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Number of sentiment categories: 3
Activation function for the final dense layer: softmax
```

C2.

The neural network model used for sentiment analysis is structured with a total of 7 layers, each serving specific roles in processing and classifying input data. The layers include:

- 1 Embedding layer: Converts input tokens into dense vectors of size 100 facilitating a meaningful representation of text data. This layer has 235,400 parameters.
- 2 LSTM layers:

    - Lstm_2: The first LSTM layer outputs sequences of 128 dimensions and has 117,248 parameters.
    - lstm_3: The second LSTM layer reduces the sequence to a single vector per sample with 64 dimensions and has 49,408 parameters.
- 2 Dropout layers:
    - dropout_2: Used after the first LSTM layer to prevent overfitting by randomly dropping units during training. This layer does not add any trainable parameters.
    - dropout_3: Used after the first dense layer to enhance regularization, also without adding trainable parameters.
- 2 Dense layers:
    - dense_2: The first dense layer has 32 units for intermediate feature extraction and contains 2,080 parameters.
    - dense_3: The final dense layer has 3 units for multi-class classification, using softmax activation to output the probabilities of each sentiment category with 99 parameters.

The model is parameterized with a total of 404,235 parameters, encompassing both trainable parameters adjusted during training to optimize performance across the sentiment categories. This structure balances feature extraction and regularization, making it robust and efficient for sentiment classification tasks.

C3.

Justification of hyperparameters:

Activation Functions:

Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns in data. Common choices include Rectified Linear Unit (relu) for hidden layers and softmax for multi-class classification tasks like sentiment analysis. Relu is chosen for intermediate (Dense and LSTM layers) because it addresses the vanishing gradient problem, promotes faster convergence by allowing the network to learn more robust features, and is computationally efficient. Softmax is used in the final output layer (Dense layer with softmax activation) for multi-class classification. Softmax converts raw output scores into probability distributions over predicted classes. This is suitable for determining sentiment categories with mutually exclusive classes (e.g., positive, neutral, negative).

Number of Nodes per Layer:

The number of nodes in each layer determines the model's capacity to learn representations at different levels of abstraction. Too few nodes may lead to underfitting, while too many may lead to overfitting. The Embedding layer's output dimension is chosen empirically based on the vocabulary size to represent dense words embeddings efficiently. 128 and 64 nodes were chosen for the first and second LSTM layers, respectively, based on the complexity of the sequential data. 32 nodes in the intermediate Dense layer were selected to extract higher-level features from LSTM outputs, balancing model complexity and computational efficiency.

Loss function:

The choice of loss function dictates how the model measures the difference between predicted and actual outputs during training. It directly affects model learning behavior and convergence. The categorical crossentropy loss function is suitable for multi-class classification tasks where each example belongs to one class. It measures the dissimilarity between the predicted probability distribution and the true distribution of the labels.

Optimizer:

The optimizer updates model parameters based on computed gradients during backpropagation. It influences training speed, convergence quality, and robustness to local minima. The Adam optimizer was chosen for its adaptive learning rate properties, which dynamically adjusts learning rates for each parameter.

Stopping criteria:

Early stopping is employed to prevent overfitting by terminating the training process when the model's performance on a validation dataset ceases to improve. This criterion ensures that the model does not continue to learn from noise in the training data, which can lead to a decrease in its ability to generalize new, unseen data. Specifically, early stopping monitors the validation loss during training and halts further training when it no longer decreases. The implementation of a patience parameter allows for minor fluctuations in validation loss, thereby ensuring that the training process is not prematurely stopped due to temporary variations. In this case, training stopped after 14 epochs, indicating that the optimal performance was achieved without unnecessary additional epochs, thus enhancing the model's efficiency and effectiveness.

Evaluation metric:

Evaluation metric are critical for assessing model performance and guiding decisions on model selection and optimization. They provide insights into how well the model performs on unseen data, ensuring its reliability and effectiveness in real-world applications. In this context, accuracy was chosen as the evaluation metric. Accuracy measures the proportion of correctly classified instances out of the total instances, offering a straightforward and comprehensive measure of overall model performance. This is suitable for multi-class classification tasks like sentiment analysis, where the goal is to correctly categorize each input into one of several sentiment classes. The best validation accuracy achieved by the model was 96%, indicating a high level of precision in classifying sentiments, thereby confirming the model's effectiveness and robustness in predicting unseen data accurately.

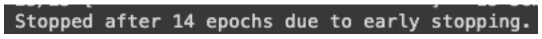
In text citations:
("Classification modeling", n.d.)
("Sentiment Analysis", n.d.)
("Introduction to language models", n.d.)
("Vanishing and exploring gradients", n.d.)
("Introduction to TensorFlow in Python", n.d.)

Part 4

D1.

Stopping criteria in training a neural network are crucial to prevent overfitting and ensure the model generalizes well on unseen data. Common stopping criteria include setting a maximum number of epochs and using early stopping based on the validation loss or another evaluation metric. The number of epochs determines how many times the learning algorithm will work through the entire training dataset. Too few epochs might result in underfitting, while too many can lead to overfitting. This technique monitors the performance of the model on a validation set and stops training when the performance stops improving, which helps prevent overfitting.

Screenshot of the final training epoch:



```
Stopped after 14 epochs due to early stopping.
```

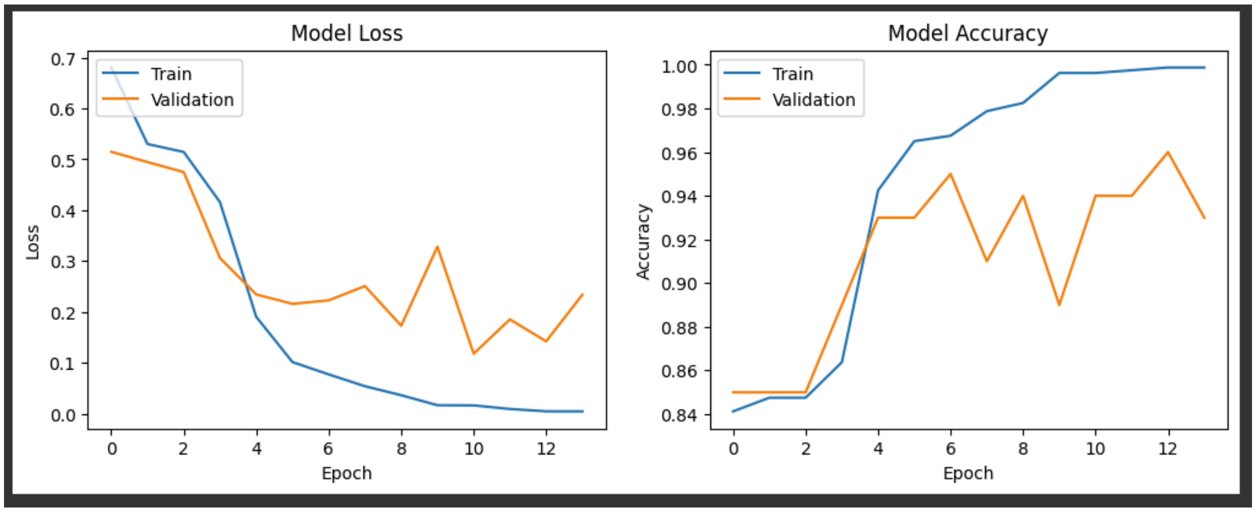
D2.

The model achieved a best validation accuracy of 96%, indicating excellent performance on the validation dataset. This high validation accuracy suggests that the model was able to generalize well to data it had not seen during training, performing with high precision and recall. To further assess the model's fitness, it was also evaluated on a separate test set, confirming its robustness and reliability.

To mitigate overfitting, several strategies were employed: early stopping, dropout layers, and validation set usage. As previously mentioned, early stopping was used to cease training when the validation loss stopped improving. This ensured the model did not overfit the training data by learning noise and details specific to the training set. Dropout regularization was incorporated into the model architecture, where a fraction of neurons was randomly ignored during each training epoch. This technique helped in preventing the model from becoming overly reliant on specific paths within the neural network, thereby promoting generalization. A portion of the training data was set aside as a validation set. The model's performance on this validation set was continuously monitored to ensure it was not overfitting the training data.

D3.

The training process was visualized through line graphs depicting the loss and accuracy over epochs for both the training and validation datasets. The graphs showed a consistent decrease in loss and increase in accuracy up to the point where early stopping was triggered, indicating the model had reached optimal performance without overfitting.



D4.

The predictive accuracy of the trained network, as assessed by the validation accuracy of 96%, demonstrates the model's ability to accurately predict outcomes on new, unseen data. This metric, along with the actions taken to prevent overfitting, underscores the model's efficacy and reliability for the given task.

In text citations:
("Introduction to RNN inside Keras", n.d.)
("Introduction to TensorFlow in Python", n.d.)

Part 5

E.

Code used to save the trained network within the neural network:

```
#save the trained network within the neural network

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define early stopping and model checkpoint callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True,
monitor='val_loss')
```

```
# Train the model with the callbacks
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=100,
                    callbacks=[early_stopping, model_checkpoint])

# Save the final model (optional)
model.save('final_model.h5')
```

F.

The neural network designed for sentiment analysis on Yelp reviews is structured with a focus on capturing the sequential nature of text data and accurately classifying the sentiment into three categories: positive, neutral, and negative. The network leverages several key layers and techniques to achieve this, including embedding layers, LSTM layers, dropout layers, and dense layers.

Network Architecture

The architecture of the neural network consists of the following layers:

1. Embedding layer
    - Functionality: Converts input tokens (words) into dense vectors of a specified dimension. This layer captures the semantic relationships between words providing a meaningful numerical representation of the text data.
    - Impact: By transforming words into dense vectors, the embedding layer allows the model to understand and leverage the semantic nuances of the text. This is crucial for accurate sentiment classification.
2. LSTM layers
    - Functionality: There are two LSTM layers in the network.
        - The first LSTM layer outputs sequences of 128 dimensions.
        - The second LSTM layer reduces the sequence to a single vector per sample with 64 dimensions.
    - Impact: LSTM layers are designed to capture long-term dependencies in sequential data. They help the model maintain context over the length of the review, which is vital for understanding the sentiment expressed, especially when key sentiment-bearing words are separated by other words in the review.
3. Dropout layers
    - Functionality: Applied after the LSTM layers and the first dense layer to prevent overfitting. Dropout randomly drops units during training, which helps in regularizing the model.
    - Impact: Dropout layers improve the generalization of the model by preventing it from becoming overly reliant on specific features or patterns in the training data. This leads to better performance on unseen data.
4. Dense Layers
    - Functionality: Two dense layers are included in the network.
        - The first dense layer has 32 units for intermediate feature extraction.

        - The final dense layer has 3 units corresponding to the three sentiment categories, using the softmax activation function.
    - Impact: Dense layers enable the model to learn higher-level features from the LSTM outputs. The final dense layer with softmax activation outputs a probability distribution over the sentiment categories, allowing for clear and interpretable classification results.

Justification of Hyperparameters

1. Activation functions:
    - Relu is used in intermediate layers to introduce non-linearity, addressing the vanishing gradient problem, and promoting faster convergence.
    - Softmax is used in in the final layer for multi-class classification, converting the raw output scores into probabilities over the sentiment categories.
2. Number of Nodes per Layer:
    - The embedding layer's output dimension was chosen based on the vocabulary size to represent word embedding efficiently.
    - The LSTM layers' dimensions (128 and 64) were selected to balance the model's capacity to learn from sequential data and computational efficiency.
    - The intermediate dense layer with 32 units extracts higher-level features without overfitting.
3. Loss function:
    - Categorical cross entropy was chosen for its suitability in multi-class classification tasks, measuring the dissimilarity between predicted and label distributions.
4. Optimizer:
    - Adam was used for its adaptive learning rate properties, which adjust the learning rate for each parameter dynamically, promoting efficient convergence.

Model Performance and Stopping Criteria

1. Early Stopping: The model used early stopping based on validation loss to prevent overfitting. Training stops when the validation loss no longer decreases, ensuring the model does not learn noise from the training data.
2. Dropout Regularization: Helps in reducing overfitting by randomly dropping units during training.
3. Evaluation Metric: Accuracy was chosen as the primary evaluation metric, providing a clear measure of overall model performance in classifying sentiment correctly.

Final Model Summary

1. Number of Parameters: The model is parameterized with a total of 404,235 parameters, encompassing both trainable and non-trainable parameters.
2. Performance: Achieved a best validation accuracy of 96%, indicating the model's robustness and effectiveness in sentiment classification.

In conclusion, the neural network's architecture, combining embedding layer, LSTM layers, dropout layers and dense layers, is designed to handle the complexities of sequential text data and accurately classify sentiment. The chosen hyperparameters and training techniques ensure the model is both effective and efficient, proving valuable insights into customer sentiment based on Yelp reviews. This structured approach allows businesses to pinpoint areas for improvement and enhance customer satisfaction.

G.

Based on neural network's impressive performance in classifying sentiment from Yelp reviews, achieving a high validation accuracy of 96%, several strategic recommendations can be made to leverage and enhance this model effectively. Firstly, the model should be deployed and integrated into existing systems or platforms to utilize its sentiment analysis capabilities. Developing an API endpoint would facilitate easy access for other applications to send text data and receive sentiment predictions.

To ensure consistent performance, continuous monitoring tools should be implemented to track key metrics such as accuracy, precision, recall, and F1-score. Additionally, regular updates and retraining with new data will help the model stay current with evolving the training dataset and experimenting with other neural network architectures, could potentially improve performance.

Gathering user feedback on the model's predictions and implementing mechanisms for interactive learning will aid in iterative improvements. Additionally, regularly analyzing the model's predictions to detect and address biases ensures ethical use and fair treatment of all inputs. Transparency about the model's workings and data usage can build trust and promote ethical use.

Staying updated with the latest developments in natural processing language and neural network architectures is essential for maintaining a competitive edge. Collaborating with academic institutions or industry partners can enhance the model further and explore new applications of sentiment analysis. The high accuracy of the neural network in sentiment classification signifies a well-performing model ready for deployment. By following this course of action, the model can be effectively integrated into practical applications, continuously improved, and leveraged to derive meaningful insights from customer feedback.

In text citations:
("Sentiment Analysis", n.d.)
("Vanishing and exploring gradients", n.d.)
("Introduction to TensorFlow in Python", n.d.)

<u>Part 6</u>

Citations for code:

DataCamp. (n.d.). Regular expression basics [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/true-fundamentals?ex=1

DataCamp. (n.d.). Tokenization [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/true-fundamentals?ex=4

DataCamp. (n.d.). Text cleaning basics [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/true-fundamentals?ex=7

DataCamp. (n.d.). Preparing text for modeling [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/applications-classification-and-topic-modeling?ex=1

DataCamp. (n.d.). Classification modeling [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/applications-classification-and-topic-modeling?ex=4

DataCamp. (n.d.). Sentiment analysis [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/advanced-techniques?ex=1

DataCamp. (n.d.). Vanishing and exploring gradients [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/rnn-architecture?ex=1

DataCamp. (n.d.). Multi-class classification models [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/multi-class-classification?ex=7

Pinecone. (n.d.). *Softmax activation*. Pinecone. Retrieved June 24, 2024, from https://www.pinecone.io/learn/softmax-activation/

Deepgram. (n.d.). *Tokenization*. Deepgram AI Glossary. Retrieved June 24, 2024, from https://deepgram.com/ai-glossary/tokenization
Data Science Stack Exchange. (n.d.). Ratio between embedded vector dimensions and vocabulary size. Retrieved June 24, 2024, from https://datascience.stackexchange.com/questions/31109/ratio-between-embedded-vector-dimensions-and-vocabulary-size

Introduction to TensorFlow in Python. (n.d.). *Data Science for Business*. Retrieved June 24, 2024, from https://campus.datacamp.com/pdf/web/viewer.html?file=https://projector-video-pdf-converter.datacamp.com/15108/chapter3.pdf#page=20

Citations for content:

DataCamp. (n.d.). Introduction to the course [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/recurrent-neural-networks-and-keras?ex=1

DataCamp. (n.d.). Introduction to language models [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/recurrent-neural-networks-and-keras?ex=5

DataCamp. (n.d.). Classification modeling [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/applications-classification-and-topic-modeling?ex=4

DataCamp. (n.d.). Sentiment analysis [Video file]. Retrieved from https://campus.datacamp.com/courses/introduction-to-natural-language-processing-in-r/advanced-techniques?ex=1

DataCamp. (n.d.). Introduction to RNN inside Keras [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/recurrent-neural-networks-and-keras?ex=9

DataCamp. (n.d.). Vanishing and exploring gradients [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/rnn-architecture?ex=1

DataCamp. (n.d.). Multi-class classification models [Video file]. Retrieved from https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-with-keras/multi-class-classification?ex=7

Pinecone. (n.d.). *Softmax activation*. Pinecone. Retrieved June 24, 2024, from https://www.pinecone.io/learn/softmax-activation/

Deepgram. (n.d.). *Tokenization*. Deepgram AI Glossary. Retrieved June 24, 2024, from https://deepgram.com/ai-glossary/tokenization
Data Science Stack Exchange. (n.d.). Ratio between embedded vector dimensions and vocabulary size. Retrieved June 24, 2024, from https://datascience.stackexchange.com/questions/31109/ratio-between-embedded-vector-dimensions-and-vocabulary-size

DataCamp. (n.d.). *Data Science for Business*. Retrieved June 24, 2024, from https://campus.datacamp.com/pdf/web/viewer.html?file=https://projector-video-pdf-converter.datacamp.com/15108/chapter3.pdf#page=20