In [1]:
```
# import libraries
import requests
import pandas as pd
import time
```

In [2]:
```
# Keys
API_KEY = "AIzaSyCx0t_YpxdF7vFi5Jy1h1_M5tmGL20jGt4" #Refer Documentation for getting API_K
CHANNEL_ID = "UCQxyNpIris-BbR-EebLDHAw" #Refer Documentation for getting CHANNEL ID
```

In [3]:
```
# Make a dummy API call
response = requests.get('https://api.github.com').json()
```

In [4]:
```
response
```

Out[4]:

```
{'current_user_url': 'https://api.github.com/user',
 'current_user_authorizations_html_url': 'https://github.com/settings/connections/applica
tions{/client_id}',
 'authorizations_url': 'https://api.github.com/authorizations',
 'code_search_url': 'https://api.github.com/search/code?q={query}{&page,per_page,sort,ord
er}',
 'commit_search_url': 'https://api.github.com/search/commits?q={query}{&page,per_page,sor
t,order}',
 'emails_url': 'https://api.github.com/user/emails',
 'emojis_url': 'https://api.github.com/emojis',
 'events_url': 'https://api.github.com/events',
 'feeds_url': 'https://api.github.com/feeds',
 'followers_url': 'https://api.github.com/user/followers',
 'following_url': 'https://api.github.com/user/following{/target}',
 'gists_url': 'https://api.github.com/gists{/gist_id}',
 'hub_url': 'https://api.github.com/hub',
 'issue_search_url': 'https://api.github.com/search/issues?q={query}{&page,per_page,sort,
order}',
 'issues_url': 'https://api.github.com/issues',
 'keys_url': 'https://api.github.com/user/keys',
 'label_search_url': 'https://api.github.com/search/labels?q={query}&repository_id={repos
itory_id}{&page,per_page}',
 'notifications_url': 'https://api.github.com/notifications',
 'organization_url': 'https://api.github.com/orgs/{org}',
 'organization_repositories_url': 'https://api.github.com/orgs/{org}/repos{?type,page,per
_page,sort}',
 'organization_teams_url': 'https://api.github.com/orgs/{org}/teams',
 'public_gists_url': 'https://api.github.com/gists/public',
 'rate_limit_url': 'https://api.github.com/rate_limit',
 'repository_url': 'https://api.github.com/repos/{owner}/{repo}',
 'repository_search_url': 'https://api.github.com/search/repositories?q={query}{&page,per
_page,sort,order}',
 'current_user_repositories_url': 'https://api.github.com/user/repos{?type,page,per_page,
sort}',
 'starred_url': 'https://api.github.com/user/starred{/owner}{/repo}',
 'starred_gists_url': 'https://api.github.com/gists/starred',
 'topic_search_url': 'https://api.github.com/search/topics?q={query}{&page,per_page}',
 'user_url': 'https://api.github.com/users/{user}',
 'user_organizations_url': 'https://api.github.com/user/orgs',
 'user_repositories_url': 'https://api.github.com/users/{user}/repos{?type,page,per_page,
sort}',
 'user_search_url': 'https://api.github.com/search/users?q={query}{&page,per_page,sort,or
der}'}
```

In [548]:
```
import os
import pandas as pd
from googleapiclient.discovery import build

# Define your API key
api_key = 'AIzaSyCx0t_YpxdF7vFi5Jy1h1_M5tmGL20jGt4'

# Define the video ID of the YouTube video you want to retrieve comments from
video_id = 'bk-nQ7HF6k4'

# Define the path for the CSV file
csv_path = 'The_Dangers_Of_AI.csv'

# Initialize the YouTube Data API client
youtube = build('youtube', 'v3', developerKey=api_key)

# Function to get video comments
def get_video_comments(youtube, **kwargs):
    comments = []
    results = youtube.commentThreads().list(**kwargs).execute()

    while results:
        for item in results['items']:
            comment = item['snippet']['topLevelComment']['snippet']
            reply_count = item['snippet']['totalReplyCount']
            comments.append({
                'comment_id': item['id'],
                'comment': comment['textDisplay'],
                'author_url': comment['authorChannelUrl'],
                'author_name': comment['authorDisplayName'],
                'reply_count': reply_count,
                'like_count': comment['likeCount'],
                'date': comment['publishedAt'],
                'vidid': comment['videoId'],
                'total_reply_counts': reply_count
            })

        if 'nextPageToken' in results:
            kwargs['pageToken'] = results['nextPageToken']
            results = youtube.commentThreads().list(**kwargs).execute()
        else:
            break

    return comments

# Retrieve comments and save to CSV
comments = get_video_comments(youtube, part='snippet', videoId=video_id, textFormat='plain

if comments:
    # Create a DataFrame from the comments
    df = pd.DataFrame(comments)

    # Save to CSV
    df.to_csv(csv_path, index=False)
    print(f'Comments saved to {csv_path}')
else:
    print('No comments found.')
```

```
Comments saved to The_Dangers_Of_AI.csv
```

In [550]:
```
import pandas as pd

comments = pd.read_csv('The_Dangers_Of_AI.csv')
comments.shape
```

Out[550]:
```
(22577, 9)
```

In [564]:
```
comments['comment']
```

Out[564]:
```
0        Mo is back, and this is honestly a conversatio...
1        AI can indeed become self-aware, but it will n...
2        Just seeing this.  Very much appreciate the in...
3        This guy is the poster child for being smart d...
4                   I was AI once but now I AM a real boy!
                               ...
22572                            They truly are delusional
22573                          It is beyond an Emergency 😳
22574    Thank you for this video. A very needed conver...
22575    Disgusting, trying to destroy the traditional ...
22576                                                 Dang
Name: comment, Length: 22577, dtype: object
```

In [577]:
```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from numpy import array

from keras.preprocessing.text import one_hot, Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Activation, Dropout, Dense
from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM
from sklearn.model_selection import train_test_split
```

In [558]:
```
# Remove all the html tags from the comments
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    '''Removes HTML tags: replaces anything between opening and closing <> with empty spac

    return TAG_RE.sub('', text)
```

In [570]:
```
# This code is used to precess the text remove unwanted characters and stop words.
def preprocess_text(sen):
    '''Cleans text data up, leaving only 2 or more char long non-stepwords composed of A-Z
    in lowercase'''

    if isinstance(sen, str):
        sentence = sen.lower()
    else:
    # Handle the case where sen is not a string (e.g., float, None, etc.)
        sentence = str(sen).lower()

    # Remove html tags
    sentence = remove_tags(sentence)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)  # When we remove apostrophe from

    # Remove multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)  # Next, we remove all the single characters

    # Remove Stopwords
    pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s*')
    sentence = pattern.sub('', sentence)

    return sentence
```

In [571]:
```
# This code snippit applies sentiment analysis using VEDAR and save to a new csv file.
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer

# Download NLTK resources
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('vader_lexicon')


comments['processed_comment'] = comments['comment'].apply(preprocess_text)

# Perform sentiment analysis
sia = SentimentIntensityAnalyzer()
comments['sentiment_score'] = comments['processed_comment'].apply(lambda x: sia.polarity_s

# Categorize sentiments
comments['sentiment'] = comments['sentiment_score'].apply(lambda x: 'positive' if x > 0 el

# Save the processed DataFrame to a CSV file
csv_file_path = 'The_Dangers_Of_AI_with_sentiment.csv'
comments.to_csv(csv_file_path, index=False)

print(f'Data has been saved to {csv_file_path}')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!

Data has been saved to The_Dangers_Of_AI_with_sentiment.csv
```

In [574]:
```
comments_with_sentiment_df = pd.read_csv('The_Dangers_Of_AI_with_sentiment.csv')
comments_with_sentiment_df.head()
```

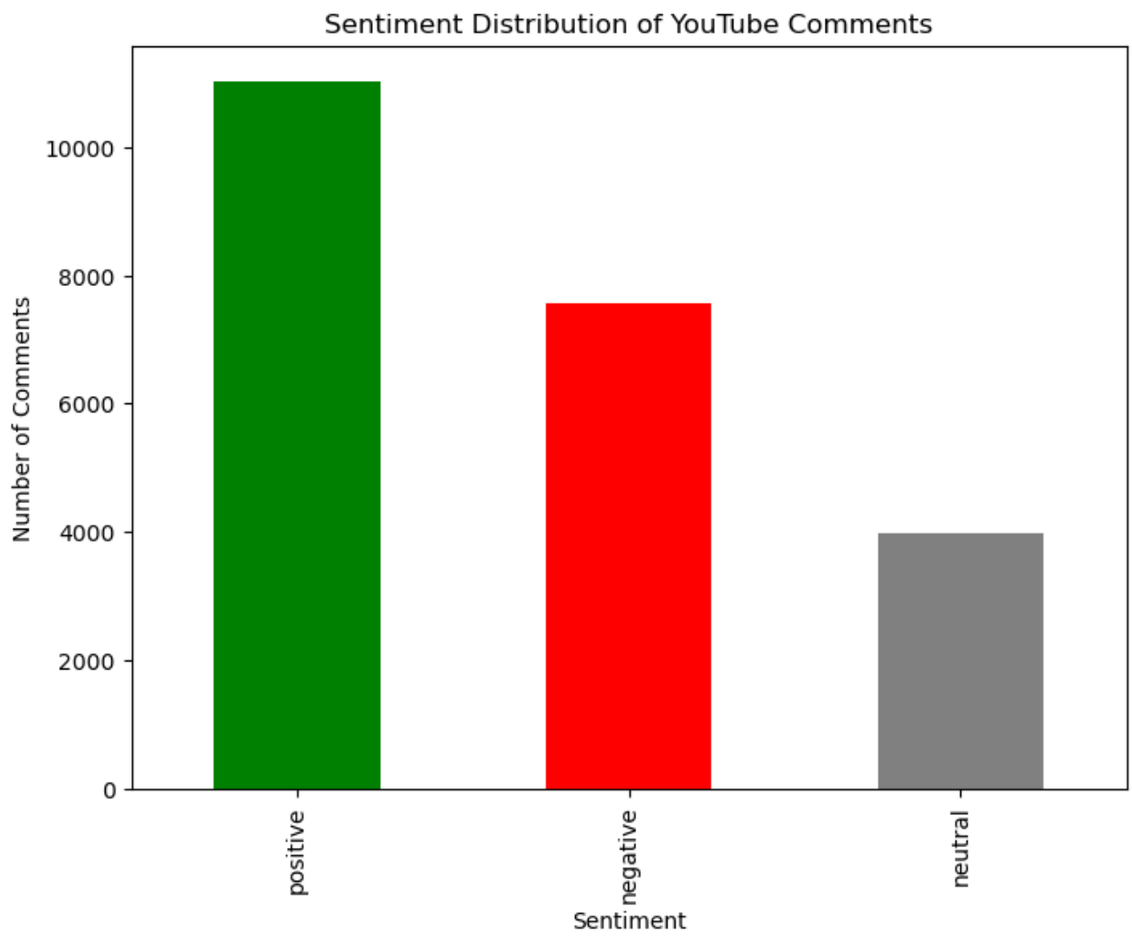
Out[574]:

	comment_id	comment	author_url	author_
0	UgxRNRfy58W4Z5XzyNh4AaABAg	Mo is back, and this is honestly a conversatio...	http://www.youtube.com/channel/UCGq-a57w-aPwyi...	@TheDiaryOf
1	Ugz3_UytFedYlrJkcg14AaABAg	AI can indeed become self-aware, but it will n...	http://www.youtube.com/channel/UCO8wv8cMVvRBeq...	@buckfucker
2	Ugy_pG9P7q935ZZE4uJ4AaABAg	Just seeing this. Very much appreciate the in...	http://www.youtube.com/channel/UC2EVnHHC-S7lmn...	@lisabenc
3	UgwYRh_G6dGD78djlRN4AaABAg	This guy is the poster child for being smart d...	http://www.youtube.com/channel/UCiyiT2kMm2V2mY...	@mrstati
4	Ugwdmv6axlLvL8qGyph4AaABAg	I was AI once but now I AM a real boy!	http://www.youtube.com/channel/UCNZJ7TzCidgXYU...	@blahbla

In [575]:
```
# Distribution of Sentiments

sentiment_distribution = comments_with_sentiment_df['sentiment'].value_counts()
print(sentiment_distribution)
```

```
positive    11027
negative     7565
neutral      3985
Name: sentiment, dtype: int64
```

In [576]:
```
# Sentiment Distribution Bar Chart

import matplotlib.pyplot as plt

sentiment_distribution = comments_with_sentiment_df['sentiment'].value_counts()

# Plotting the sentiment distribution
plt.figure(figsize=(8, 6))
sentiment_distribution.plot(kind='bar', color=['green', 'red', 'gray'])
plt.title('Sentiment Distribution of YouTube Comments')
plt.xlabel('Sentiment')
plt.ylabel('Number of Comments')
plt.show()
```



In [578]:
```
#Check Na values
na_count = comments_with_sentiment_df['processed_comment'].isna().sum()

print(f'Number of NaN values in column: {na_count}')
```

```
Number of NaN values in column: 21
```

In [579]:
```
data = comments_with_sentiment_df[['sentiment', 'processed_comment']]

# Pre-process the data and check for any NaN values -
# as anything other that english language would not have been processed and it would give
# Remove those comments from the list.
data_cleaned = data.dropna(subset=['processed_comment'])
data_cleaned.shape
```

Out[579]:
```
(22556, 2)
```

In [581]:
```
# Distribution of Sentiments

sentiment_distribution = data_cleaned['sentiment'].value_counts()
print(sentiment_distribution)
```

```
positive    11027
negative     7565
neutral      3964
Name: sentiment, dtype: int64
```

In [643]:
```
# Remove the columns which has neutral values. asuming that will not be beneficial for dec
# Converting it to a Binary Classification Problem.

data_cleaned = data_cleaned[data_cleaned['sentiment'] != 'neutral']
data_cleaned
```

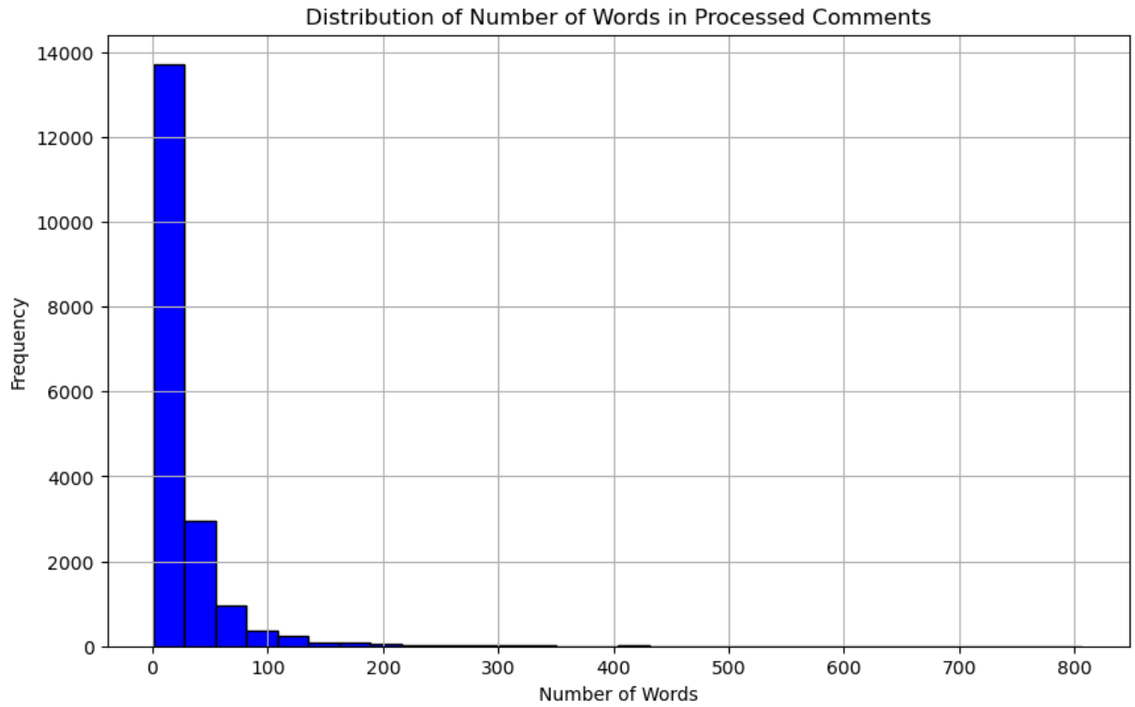
Out[643]:

	sentiment	processed_comment
0	positive	mo back honestly conversation missed please sh...
1	negative	ai indeed become self aware emotional sure rat...
2	positive	seeing much appreciate intelligence careful ha...
3	positive	guy poster child smart make wise
6	positive	sounds like ai
...	...	...
22570	positive	glad see mo gawdat back
22572	positive	truly delusional
22573	negative	beyond emergency
22574	positive	thank video needed conversation indeed
22575	negative	disgusting trying destroy traditional family

18592 rows × 2 columns

In [583]:
```
# Check for word count in the comments to decide on maxlength parameter for Neural Networks
# Copy and process for analysis.
check_df = data_cleaned.copy()
check_df['num_words'] = check_df['processed_comment'].apply(lambda x: len(str(x).split()))

# Calculate mean and std
mean_num_words = check_df['num_words'].mean()
std_num_words = check_df['num_words'].std()

# Display results
print(f"Mean number of words: {mean_num_words:.2f}")
print(f"Standard deviation of words: {std_num_words:.2f}")

# Plotting a histogram
plt.figure(figsize=(10, 6))
plt.hist(check_df['num_words'], bins=30, color='blue', edgecolor='black')
plt.title('Distribution of Number of Words in Processed Comments')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
Mean number of words: 25.91
Standard deviation of words: 42.37
```



In [584]:
```
# Assuming 'check_df' contains the processed data including sentiment information
long_comments_df = check_df[check_df['num_words'] > 200]

# Display the resulting DataFrame
long_comments_df
```

Out[584]:

	sentiment	processed_comment	num_words
141	positive	love mo read books note topic podcast psycholo...	237
209	negative	interesting hear far rabbit hole google manage...	218
332	positive	people amazed squeezy toy aware ai potential o...	201
399	negative	live short road next freeway numbers match fm ...	305
558	positive	key takeaways quick navigation speaker emphas...	592
...	...	...	...
20750	negative	okay someone please respond desperate advice o...	487
21453	positive	great podcast intrigued ai also think hustle f...	404
21681	positive	panoramic landscape human endeavor exists epoc...	439
21988	positive	yes little bit terrifying australia must stran...	259
22316	positive	question define best interest answer say gover...	395

155 rows × 3 columns

Preparing embedding laye

In [ ]:
```
```

In [585]:
```
# Split the dataset into X and y variables.
# Map the sentiments - positive = 1 and negative = 0 and save to 'y' variable.
X = data_cleaned['processed_comment']
y = data_cleaned['sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

In [586]:
```
y
```

Out[586]:
```
array([1, 0, 1, ..., 0, 1, 0])
```

In [587]:
```
# The train set will be used to train our deep learning models
# while test set will be used to evaluate how well our model performs

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [588]:
```
X_train
```

Out[588]:
```
20268    creator designed first ai however taught creat...
19590    fabulous talk hope us come together kind conve...
16185    drake tracks manipulated human act like entire...
14239    written dna one take away one thing stop coron...
12293    saddens hink agi last thing humanity ever inve...
                               ...
13752    maybe one else figured already late chat gpt i...
14581    emotions chemical gland excreted hints brain h...
6547     thank much podcast gentlemen gotten nuclear ag...
1036               people give nature guy anti christ evil
19204    greatest danger humanity military industrial c...
Name: processed_comment, Length: 14873, dtype: object
```

In [589]:
```
# Embedding layer expects the words to be in numeric form
# Using Tokenizer function from keras.preprocessing.text library
# Method fit_on_text trains the tokenizer
# Method texts_to_sequences converts sentences to their numeric form

word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)

X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
```

In [590]:
```
# Adding 1 to store dimensions for words for which no pretrained word embeddings exist

vocab_length = len(word_tokenizer.word_index) + 1

vocab_length
```

Out[590]: 25393

In [591]:
```
# Padding all reviews to fixed length 300

maxlen = 300

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

In [592]:
```
# Load GloVe word embeddings and create an Embeddings Dictionary

from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
glove_file = open('a2_glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

In [596]:
```
# Create Embedding Matrix having 100 columns
# Containing 100-dimensional GloVe word embeddings for all words in our corpus.

embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

In [594]:
```
embedding_matrix.shape
```

Out[594]: (25393, 100)

Model Training

Simple Neural Network

In [607]:
```
# Neural Network architecture

snn_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=ma

snn_model.add(embedding_layer)

snn_model.add(Flatten())
snn_model.add(Dense(1, activation='sigmoid'))
```

In [608]:
```
# Model compiling

snn_model.compile(optimizer=keras.optimizers.Adam(0.01), loss='binary_crossentropy', metri

print(snn_model.summary())
```

```
Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_49 (Embedding)    (None, 300, 100)          2539300

 flatten_7 (Flatten)         (None, 30000)             0

 dense_46 (Dense)            (None, 1)                 30001

=================================================================
Total params: 2569301 (9.80 MB)
Trainable params: 30001 (117.19 KB)
Non-trainable params: 2539300 (9.69 MB)
_________________________________________________________________
None
```

In [609]:
```
# Model training

snn_model_history = snn_model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, 
```

```
Epoch 1/20
93/93 [==============================] - 1s 7ms/step - loss: 0.6505 - acc: 0.7186 - val_l
oss: 0.7264 - val_acc: 0.7234
Epoch 2/20
93/93 [==============================] - 1s 6ms/step - loss: 0.4402 - acc: 0.8080 - val_l
oss: 0.7478 - val_acc: 0.7338
Epoch 3/20
93/93 [==============================] - 1s 6ms/step - loss: 0.3733 - acc: 0.8328 - val_l
oss: 0.8207 - val_acc: 0.7355
Epoch 4/20
93/93 [==============================] - 1s 6ms/step - loss: 0.3398 - acc: 0.8520 - val_l
oss: 0.8144 - val_acc: 0.7445
Epoch 5/20
93/93 [==============================] - 1s 6ms/step - loss: 0.3187 - acc: 0.8586 - val_l
oss: 0.8378 - val_acc: 0.7388
Epoch 6/20
93/93 [==============================] - 1s 6ms/step - loss: 0.3050 - acc: 0.8667 - val_l
oss: 0.8550 - val_acc: 0.7398
Epoch 7/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2986 - acc: 0.8724 - val_l
oss: 0.9184 - val_acc: 0.7355
Epoch 8/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2869 - acc: 0.8773 - val_l
oss: 0.9251 - val_acc: 0.7368
Epoch 9/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2834 - acc: 0.8799 - val_l
oss: 0.9351 - val_acc: 0.7355
Epoch 10/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2793 - acc: 0.8813 - val_l
oss: 0.9787 - val_acc: 0.7318
Epoch 11/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2746 - acc: 0.8822 - val_l
oss: 0.9933 - val_acc: 0.7348
Epoch 12/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2718 - acc: 0.8848 - val_l
oss: 1.1521 - val_acc: 0.7207
Epoch 13/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2652 - acc: 0.8859 - val_l
oss: 1.1066 - val_acc: 0.7304
Epoch 14/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2642 - acc: 0.8861 - val_l
oss: 1.0966 - val_acc: 0.7261
Epoch 15/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2627 - acc: 0.8877 - val_l
oss: 1.1164 - val_acc: 0.7308
Epoch 16/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2549 - acc: 0.8929 - val_l
oss: 1.1659 - val_acc: 0.7264
Epoch 17/20
93/93 [==============================] - 1s 7ms/step - loss: 0.2536 - acc: 0.8930 - val_l
oss: 1.1738 - val_acc: 0.7284
Epoch 18/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2523 - acc: 0.8951 - val_l
oss: 1.1989 - val_acc: 0.7267
Epoch 19/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2486 - acc: 0.8940 - val_l
oss: 1.2245 - val_acc: 0.7334
Epoch 20/20
93/93 [==============================] - 1s 6ms/step - loss: 0.2458 - acc: 0.8954 - val_l
oss: 1.2733 - val_acc: 0.7267
```

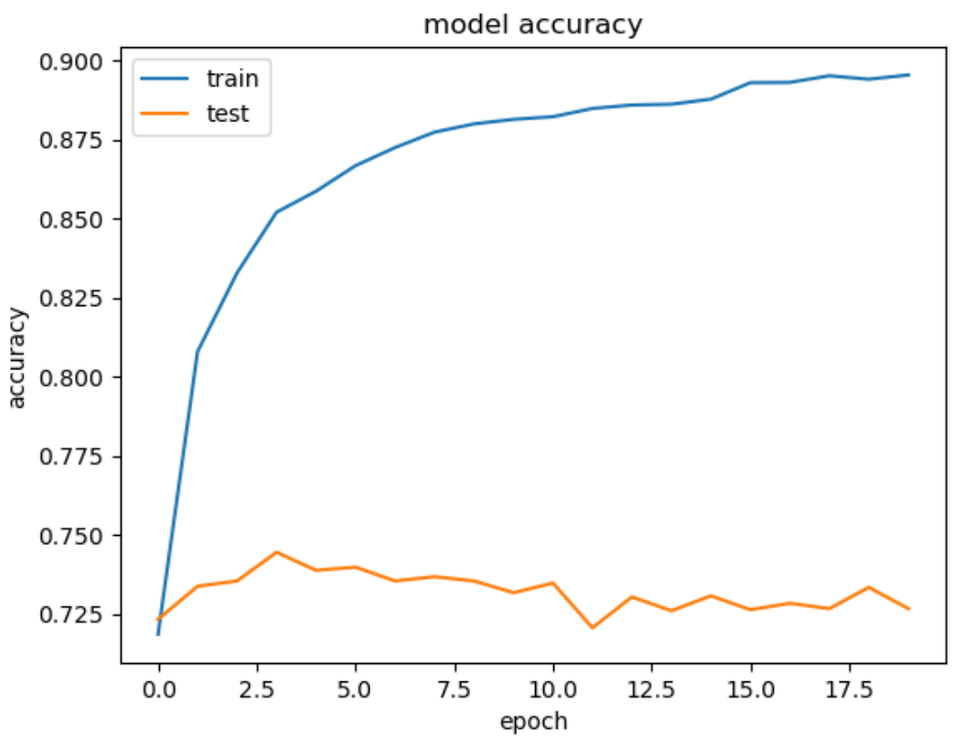
In [610]:
```
# Predictions on the Test Set

score_snn = snn_model.evaluate(X_test, y_test, verbose=1)
```

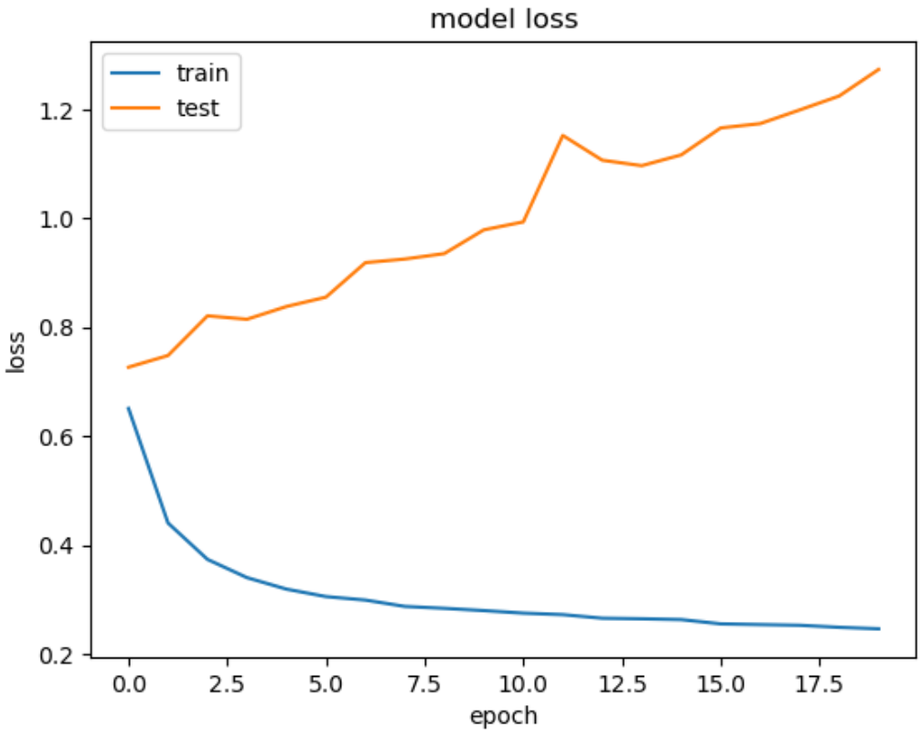
```
117/117 [==============================] - 0s 2ms/step - loss: 1.1083 - acc: 0.7403
```

In [611]:
```
# Model Performance

print("Test Score:", score_snn[0])
print("Test Accuracy:", score_snn[1])
```

```
Test Score: 1.1083227396011353
Test Accuracy: 0.7402527332305908
```

In [612]:
```
# Model Performance Charts

import matplotlib.pyplot as plt

plt.plot(snn_model_history.history['acc'])
plt.plot(snn_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(snn_model_history.history['loss'])
plt.plot(snn_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```



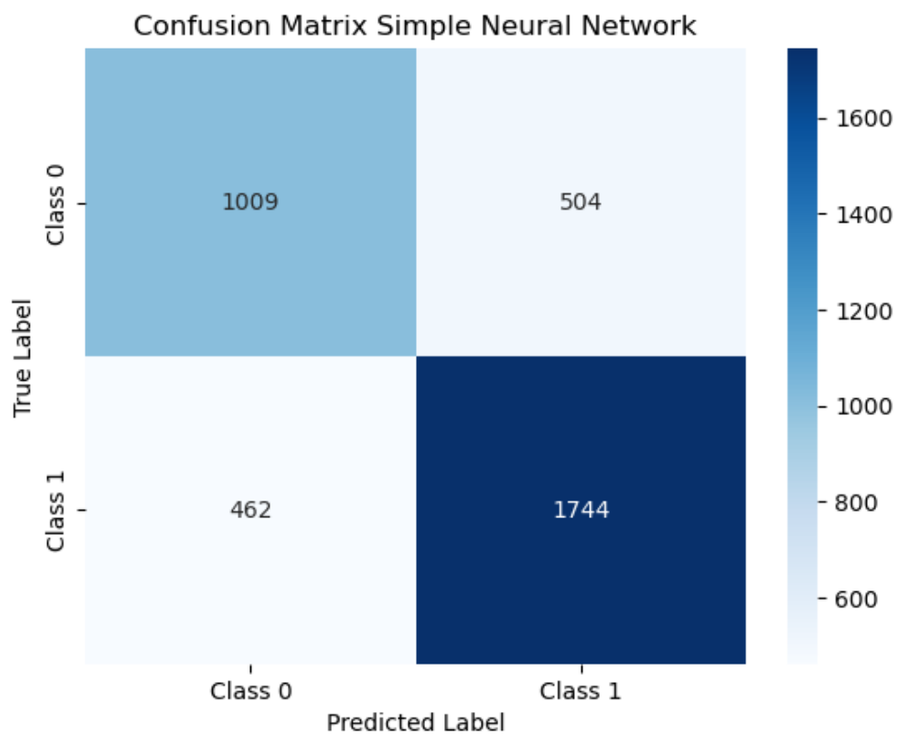


In [613]:
```
# Make Predictions
predictions_snn = snn_model.predict(X_test)

# Compare Predictions with Actual Values
comparison_df_snn = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_snn.flatten()
comparison_df_snn.shape
```

```
117/117 [==============================] - 0s 2ms/step
```

Out[613]:
```
(3719, 2)
```

In [614]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Convert probabilities to class labels (0 or 1) based on a threshold (e.g., 0.5)
threshold = 0.5
predicted_labels_snn = np.where(predictions_snn > threshold, 1, 0)

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_snn)

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_snn)

# Plot confusion matrix with labels and colors
labels = ['Class 0', 'Class 1']
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabel
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix Simple Neural Network')
plt.show()

# Classification report.
print("\nClassification Report Simple Neural Network:")
print(classification_report(y_test, predicted_labels_snn))
```




```
Classification Report Simple Neural Network:
              precision    recall  f1-score   support

           0       0.69      0.67      0.68      1513
           1       0.78      0.79      0.78      2206

    accuracy                           0.74      3719
   macro avg       0.73      0.73      0.73      3719
weighted avg       0.74      0.74      0.74      3719
```

Convolutional Neural Network

In [615]:
```
from keras.layers import Conv1D
```

In [616]:
```
# Neural Network architecture

cnn_model = Sequential()

embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=ma
cnn_model.add(embedding_layer)

cnn_model.add(Conv1D(128, 5, activation='relu'))
cnn_model.add(GlobalMaxPooling1D())
cnn_model.add(Dense(1, activation='sigmoid'))
```

In [617]:
```
# Model compiling

cnn_model.compile(optimizer=keras.optimizers.Adam(0.01), loss='binary_crossentropy', metri
print(cnn_model.summary())
```

```
Model: "sequential_50"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_50 (Embedding)    (None, 300, 100)          2539300

 conv1d_20 (Conv1D)          (None, 296, 128)          64128

 global_max_pooling1d_19 (G  (None, 128)               0
 lobalMaxPooling1D)

 dense_47 (Dense)            (None, 1)                 129

=================================================================
Total params: 2603557 (9.93 MB)
Trainable params: 64257 (251.00 KB)
Non-trainable params: 2539300 (9.69 MB)
_________________________________________________________________
None
```

In [618]:
```
# Model training

cnn_model_history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,
```

```
Epoch 1/20
93/93 [==============================] - 7s 68ms/step - loss: 0.6382 - acc: 0.7126 - val_
loss: 0.4920 - val_acc: 0.7671
Epoch 2/20
93/93 [==============================] - 6s 68ms/step - loss: 0.3836 - acc: 0.8280 - val_
loss: 0.4925 - val_acc: 0.7775
Epoch 3/20
93/93 [==============================] - 6s 67ms/step - loss: 0.3068 - acc: 0.8686 - val_
loss: 0.4387 - val_acc: 0.8128
Epoch 4/20
93/93 [==============================] - 6s 62ms/step - loss: 0.2169 - acc: 0.9174 - val_
loss: 0.5087 - val_acc: 0.8003
Epoch 5/20
93/93 [==============================] - 6s 61ms/step - loss: 0.1432 - acc: 0.9522 - val_
loss: 0.6384 - val_acc: 0.7805
Epoch 6/20
93/93 [==============================] - 6s 63ms/step - loss: 0.0974 - acc: 0.9680 - val_
loss: 0.6182 - val_acc: 0.8013
Epoch 7/20
93/93 [==============================] - 6s 62ms/step - loss: 0.0693 - acc: 0.9790 - val_
loss: 0.6949 - val_acc: 0.7960
Epoch 8/20
93/93 [==============================] - 6s 64ms/step - loss: 0.0355 - acc: 0.9926 - val_
loss: 0.7485 - val_acc: 0.8047
Epoch 9/20
93/93 [==============================] - 6s 61ms/step - loss: 0.0174 - acc: 0.9975 - val_
loss: 0.7859 - val_acc: 0.8040
Epoch 10/20
93/93 [==============================] - 6s 61ms/step - loss: 0.0106 - acc: 0.9987 - val_
loss: 0.8363 - val_acc: 0.8064
Epoch 11/20
93/93 [==============================] - 6s 62ms/step - loss: 0.0064 - acc: 0.9992 - val_
loss: 0.8598 - val_acc: 0.8047
Epoch 12/20
93/93 [==============================] - 6s 63ms/step - loss: 0.0051 - acc: 0.9994 - val_
loss: 0.8935 - val_acc: 0.8067
Epoch 13/20
93/93 [==============================] - 7s 74ms/step - loss: 0.0039 - acc: 0.9996 - val_
loss: 0.9032 - val_acc: 0.8077
Epoch 14/20
93/93 [==============================] - 6s 63ms/step - loss: 0.0023 - acc: 0.9998 - val_
loss: 0.9313 - val_acc: 0.8091
Epoch 15/20
93/93 [==============================] - 6s 61ms/step - loss: 0.0019 - acc: 0.9998 - val_
loss: 0.9418 - val_acc: 0.8084
Epoch 16/20
93/93 [==============================] - 6s 61ms/step - loss: 0.0016 - acc: 0.9999 - val_
loss: 0.9752 - val_acc: 0.8027
Epoch 17/20
93/93 [==============================] - 6s 63ms/step - loss: 9.8045e-04 - acc: 1.0000 -
val_loss: 0.9779 - val_acc: 0.8077
Epoch 18/20
93/93 [==============================] - 6s 64ms/step - loss: 7.9844e-04 - acc: 1.0000 -
val_loss: 0.9886 - val_acc: 0.8061
Epoch 19/20
93/93 [==============================] - 6s 63ms/step - loss: 7.1589e-04 - acc: 1.0000 -
val_loss: 0.9974 - val_acc: 0.8064
Epoch 20/20
93/93 [==============================] - 6s 64ms/step - loss: 6.2594e-04 - acc: 1.0000 -
val_loss: 1.0093 - val_acc: 0.8061
```

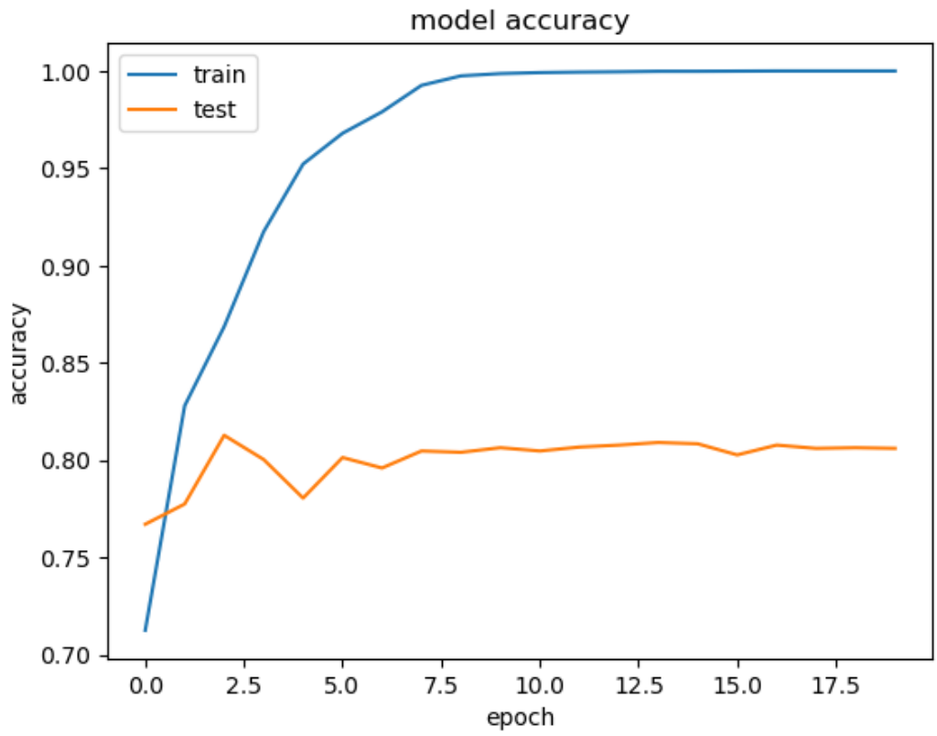
In [619]:
```
# Predictions on the Test Set

score_cnn = cnn_model.evaluate(X_test, y_test, verbose=1)
```

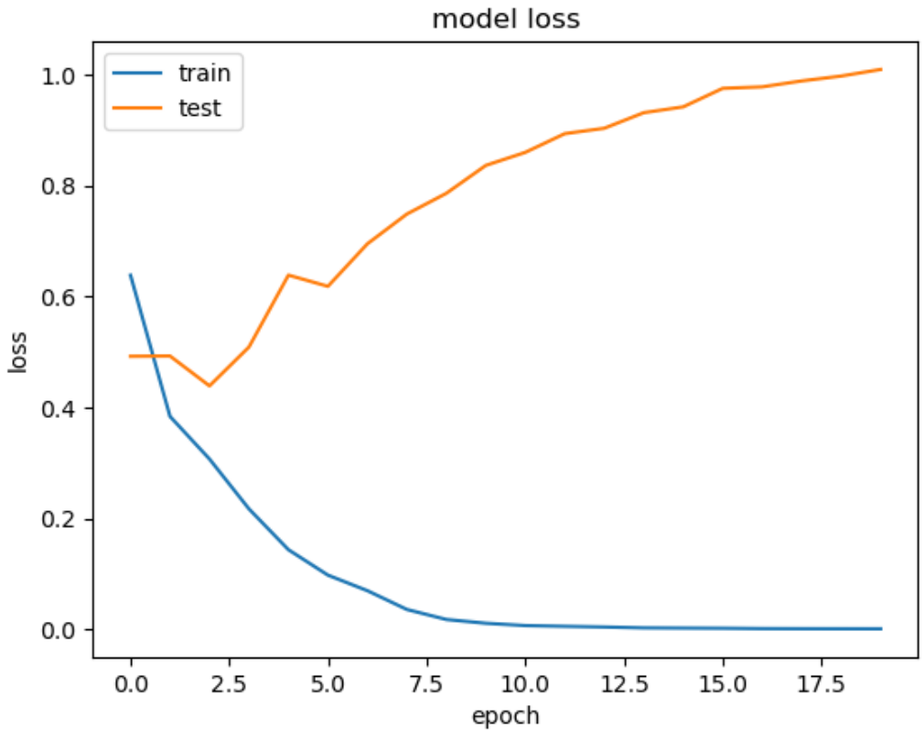
```
117/117 [==============================] - 1s 6ms/step - loss: 0.9419 - acc: 0.8032
```

In [620]:
```
# Model Performance

print("Test Score:", score_cnn[0])
print("Test Accuracy:", score_cnn[1])
```

```
Test Score: 0.9418678283691406
Test Accuracy: 0.8031728863716125
```

In [621]:
```
# Model Performance Charts

import matplotlib.pyplot as plt

plt.plot(cnn_model_history.history['acc'])
plt.plot(cnn_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(cnn_model_history.history['loss'])
plt.plot(cnn_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```





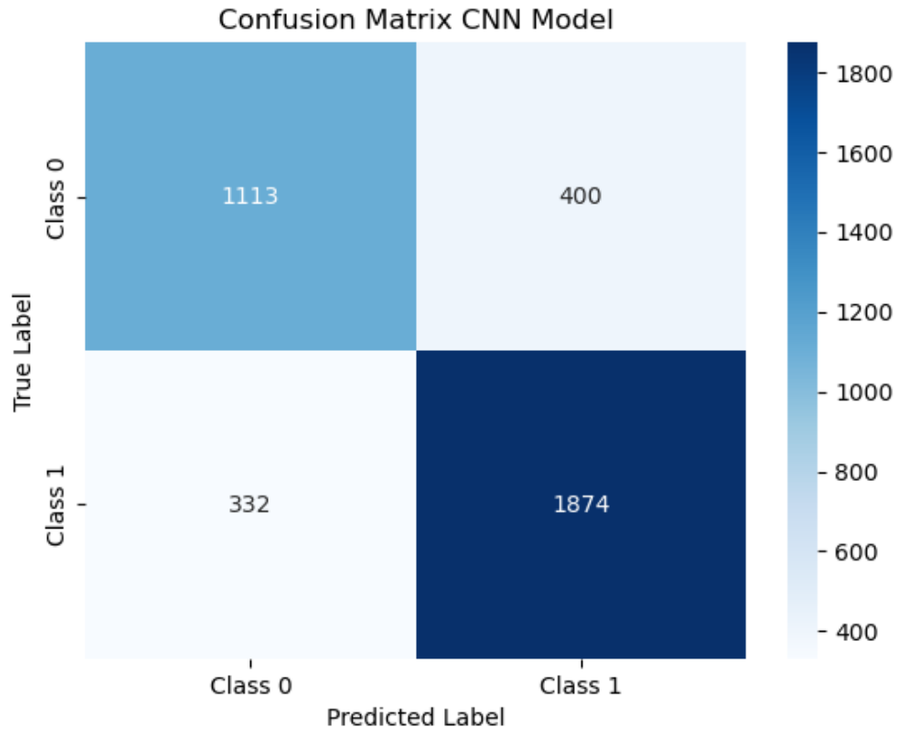
In [622]:
```
# Make Predictions
predictions_cnn = cnn_model.predict(X_test)

# Compare Predictions with Actual Values
comparison_df_cnn = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_cnn.flatten()
comparison_df_cnn.head()
```

```
117/117 [==============================] - 1s 5ms/step
```

Out[622]:

	Actual	Predicted
0	1	9.999568e-01
1	1	9.999973e-01
2	0	1.955808e-01
3	1	9.999996e-01
4	0	3.673070e-10

In [623]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Convert probabilities to class labels (0 or 1) based on a threshold (e.g., 0.5)
threshold = 0.5
predicted_labels_cnn = np.where(predictions_cnn > threshold, 1, 0)

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_cnn)

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_cnn)

# Plot confusion matrix with labels and colors
labels = ['Class 0', 'Class 1']
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabel
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix CNN Model')
plt.show()

# Classification report.
print("\nClassification Report CNN Model:")
print(classification_report(y_test, predicted_labels_cnn))
```




```
Classification Report CNN Model:
              precision    recall  f1-score   support

           0       0.77      0.74      0.75      1513
           1       0.82      0.85      0.84      2206

    accuracy                           0.80      3719
   macro avg       0.80      0.79      0.79      3719
weighted avg       0.80      0.80      0.80      3719
```

In [ ]:
```
```

Recurrent Neural Network (LSTM)

In [624]:
```
from keras.layers import LSTM
```

In [625]:
```
# Neural Network architecture

lstm_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=ma

lstm_model.add(embedding_layer)
lstm_model.add(LSTM(128))
lstm_model.add(Dense(1, activation='sigmoid'))
```

In [626]:
```
# Model compiling

lstm_model.compile(optimizer=keras.optimizers.Adam(0.01), loss='binary_crossentropy', metr
print(lstm_model.summary())
```

```
Model: "sequential_51"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_51 (Embedding)    (None, 300, 100)          2539300

 lstm_26 (LSTM)              (None, 128)               117248

 dense_48 (Dense)            (None, 1)                 129

=================================================================
Total params: 2656677 (10.13 MB)
Trainable params: 117377 (458.50 KB)
Non-trainable params: 2539300 (9.69 MB)
_________________________________________________________________
None
```

In [627]:
```
# Model Training

lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=20, verbose =
```

```
Epoch 1/20
93/93 [==============================] - 57s 604ms/step - loss: 0.6809 - acc: 0.5942 - va
l_loss: 0.6805 - val_acc: 0.5785
Epoch 2/20
93/93 [==============================] - 64s 689ms/step - loss: 0.6732 - acc: 0.5967 - va
l_loss: 0.6811 - val_acc: 0.5785
Epoch 3/20
93/93 [==============================] - 69s 740ms/step - loss: 0.6730 - acc: 0.5970 - va
l_loss: 0.6813 - val_acc: 0.5785
Epoch 4/20
93/93 [==============================] - 72s 775ms/step - loss: 0.6729 - acc: 0.5970 - va
l_loss: 0.6816 - val_acc: 0.5785
Epoch 5/20
93/93 [==============================] - 76s 816ms/step - loss: 0.6728 - acc: 0.5970 - va
l_loss: 0.6804 - val_acc: 0.5788
Epoch 6/20
93/93 [==============================] - 71s 759ms/step - loss: 0.6730 - acc: 0.5970 - va
l_loss: 0.6827 - val_acc: 0.5785
Epoch 7/20
93/93 [==============================] - 71s 759ms/step - loss: 0.6727 - acc: 0.5970 - va
l_loss: 0.6805 - val_acc: 0.5785
Epoch 8/20
93/93 [==============================] - 71s 761ms/step - loss: 0.6838 - acc: 0.5864 - va
l_loss: 0.6836 - val_acc: 0.5785
Epoch 9/20
93/93 [==============================] - 70s 755ms/step - loss: 0.6883 - acc: 0.5794 - va
l_loss: 0.6821 - val_acc: 0.5785
Epoch 10/20
93/93 [==============================] - 115s 1s/step - loss: 0.6780 - acc: 0.5881 - val_
loss: 0.6825 - val_acc: 0.5785
Epoch 11/20
93/93 [==============================] - 69s 747ms/step - loss: 0.6760 - acc: 0.5967 - va
l_loss: 0.6839 - val_acc: 0.5785
Epoch 12/20
93/93 [==============================] - 69s 747ms/step - loss: 0.6737 - acc: 0.5967 - va
l_loss: 0.6820 - val_acc: 0.5785
Epoch 13/20
93/93 [==============================] - 69s 745ms/step - loss: 0.6735 - acc: 0.5967 - va
l_loss: 0.6812 - val_acc: 0.5785
Epoch 14/20
93/93 [==============================] - 69s 747ms/step - loss: 0.6742 - acc: 0.5967 - va
l_loss: 0.6811 - val_acc: 0.5785
Epoch 15/20
93/93 [==============================] - 69s 747ms/step - loss: 0.6331 - acc: 0.6461 - va
l_loss: 0.5940 - val_acc: 0.7116
Epoch 16/20
93/93 [==============================] - 70s 753ms/step - loss: 0.4990 - acc: 0.7655 - va
l_loss: 0.4676 - val_acc: 0.7906
Epoch 17/20
93/93 [==============================] - 589s 6s/step - loss: 0.4281 - acc: 0.8165 - val_
loss: 0.4333 - val_acc: 0.8077
Epoch 18/20
93/93 [==============================] - 69s 745ms/step - loss: 0.3537 - acc: 0.8537 - va
l_loss: 0.4359 - val_acc: 0.8165
Epoch 19/20
93/93 [==============================] - 69s 738ms/step - loss: 0.3443 - acc: 0.8580 - va
l_loss: 0.4100 - val_acc: 0.8262
Epoch 20/20
93/93 [==============================] - 69s 747ms/step - loss: 0.2864 - acc: 0.8880 - va
l_loss: 0.4395 - val_acc: 0.8259
```

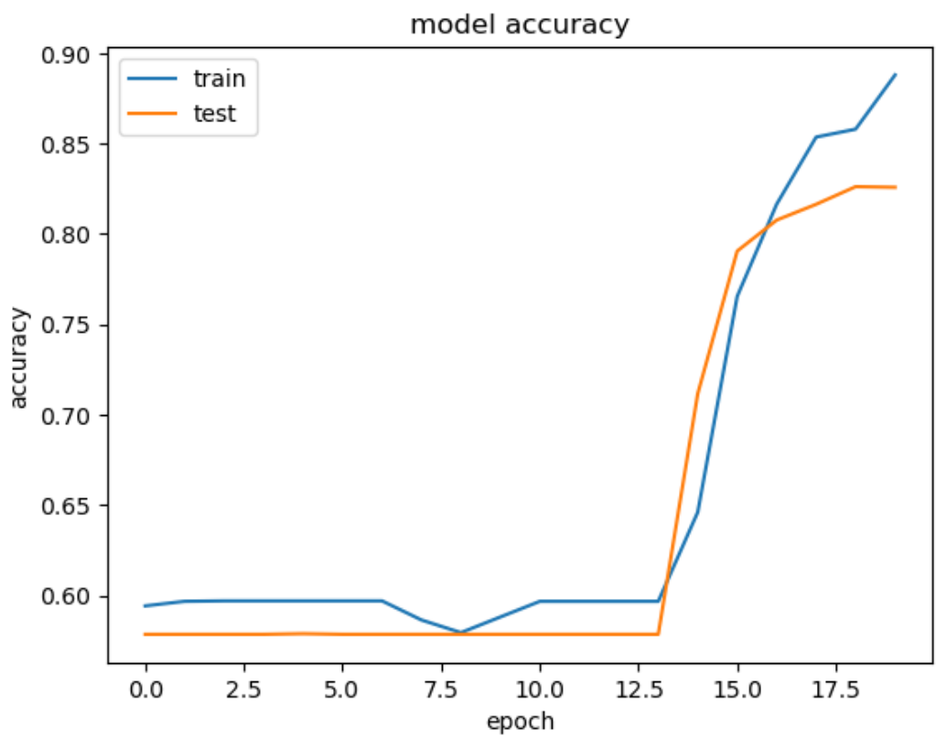
In [628]:
```
# Predictions on the Test Set

score_lstm = lstm_model.evaluate(X_test, y_test, verbose=1)
```

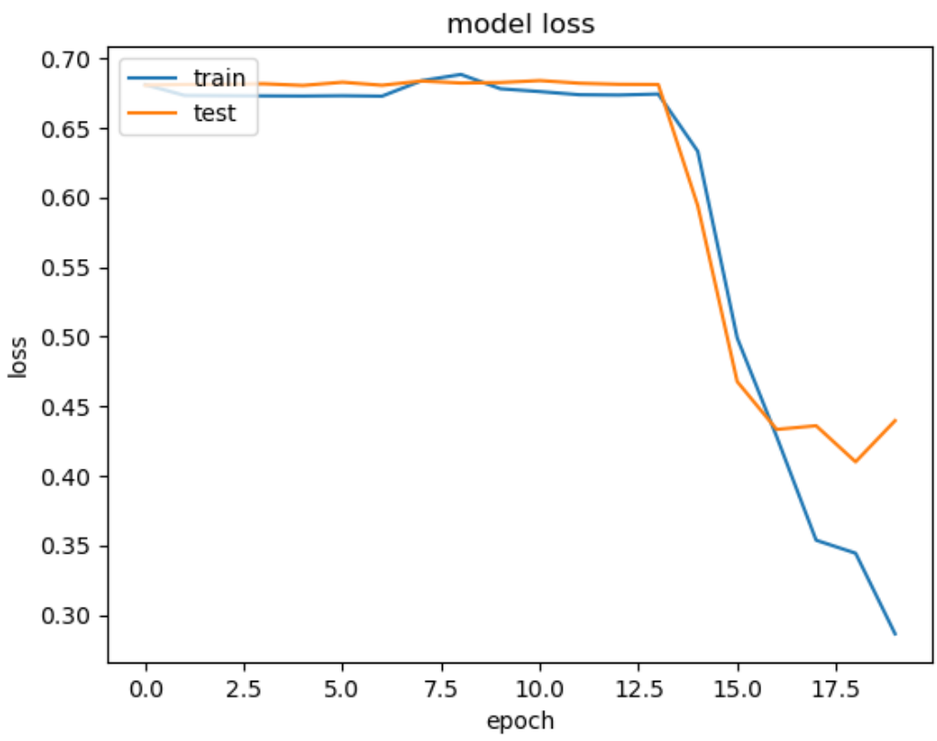
```
117/117 [==============================] - 7s 61ms/step - loss: 0.4052 - acc: 0.8354
```

In [629]:
```
# Model Performance

print("Test Score:", score_lstm[0])
print("Test Accuracy:", score_lstm[1])
```

```
Test Score: 0.4052123427391052
Test Accuracy: 0.8354396224021912
```

In [630]:
```
# Model Performance Charts

import matplotlib.pyplot as plt

plt.plot(lstm_model_history.history['acc'])
plt.plot(lstm_model_history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(lstm_model_history.history['loss'])
plt.plot(lstm_model_history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```





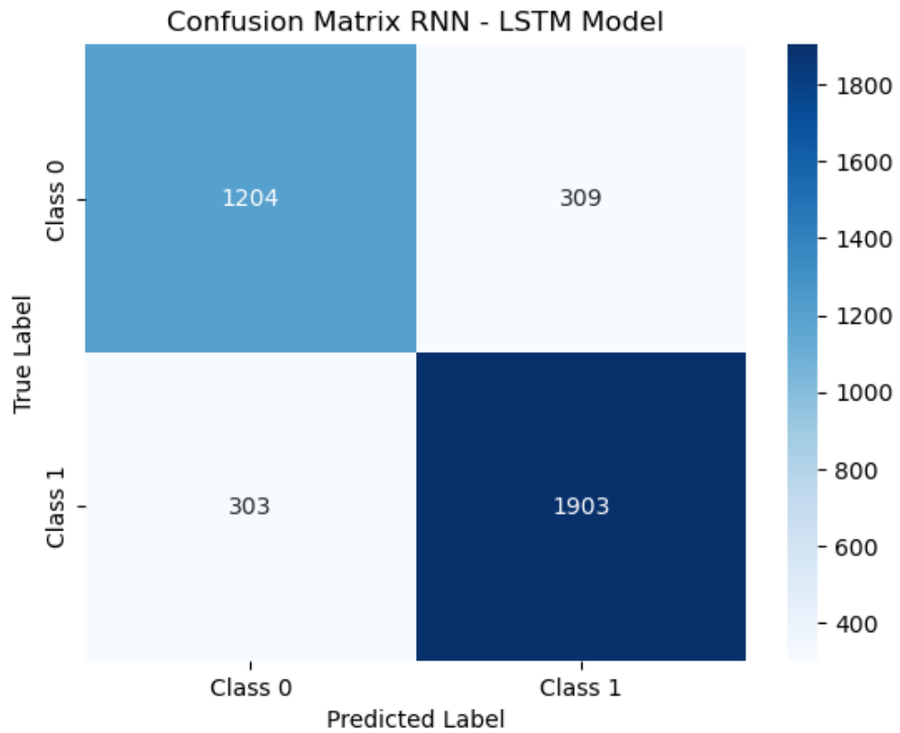
In [631]:
```
# Make Predictions
predictions_lstm = lstm_model.predict(X_test)

# Compare Predictions with Actual Values
comparison_df_lstm = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_lstm.flatten
comparison_df_lstm.head()
```

```
117/117 [==============================] - 7s 62ms/step
```

Out[631]:

	Actual	Predicted
0	1	0.970707
1	1	0.962432
2	0	0.969395
3	1	0.962105
4	0	0.038007

In [632]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Convert probabilities to class labels (0 or 1) based on a threshold (e.g., 0.5)
threshold = 0.5
predicted_labels_lstm = np.where(predictions_lstm > threshold, 1, 0)

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_lstm)

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels_lstm)

# Plot confusion matrix with labels and colors
labels = ['Class 0', 'Class 1']
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabel
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix RNN - LSTM Model')
plt.show()

# Classification report.
print("\nClassification Report RNN - LSTM Model:")
print(classification_report(y_test, predicted_labels_lstm))
```




```
Classification Report RNN - LSTM Model:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80      1513
           1       0.86      0.86      0.86      2206

    accuracy                           0.84      3719
   macro avg       0.83      0.83      0.83      3719
weighted avg       0.84      0.84      0.84      3719
```

RNN - LSTM Model Implementation on new YouTube Video for evaluation.

In [360]:
```
import os
import pandas as pd
from googleapiclient.discovery import build

# Define your API key
api_key = 'AIzaSyCx0t_YpxdF7vFi5Jy1h1_M5tmGL20jGt4'

# Define the video ID of the YouTube video you want to retrieve comments from
video_id = '6ydFDwv-n8w'

# Define the path for the CSV file
csv_path = 'Dawn_of_the_AI_Wars.csv'

# Initialize the YouTube Data API client
youtube = build('youtube', 'v3', developerKey=api_key)

# Function to get video comments
def get_video_comments(youtube, **kwargs):
    comments = []
    results = youtube.commentThreads().list(**kwargs).execute()

    while results:
        for item in results['items']:
            comment = item['snippet']['topLevelComment']['snippet']
            reply_count = item['snippet']['totalReplyCount']
            comments.append({
                'comment_id': item['id'],
                'comment': comment['textDisplay'],
                'author_url': comment['authorChannelUrl'],
                'author_name': comment['authorDisplayName'],
                'reply_count': reply_count,
                'like_count': comment['likeCount'],
                'date': comment['publishedAt'],
                'vidid': comment['videoId'],
                'total_reply_counts': reply_count
            })

        if 'nextPageToken' in results:
            kwargs['pageToken'] = results['nextPageToken']
            results = youtube.commentThreads().list(**kwargs).execute()
        else:
            break

    return comments

# Retrieve comments and save to CSV
comments = get_video_comments(youtube, part='snippet', videoId=video_id, textFormat='plain

if comments:
    # Create a DataFrame from the comments
    df = pd.DataFrame(comments)

    # Save to CSV
    df.to_csv(csv_path, index=False)
    print(f'Comments saved to {csv_path}')
else:
    print('No comments found.')
```

```
Comments saved to Dawn_of_the_AI_Wars.csv
```

In [633]:
```
import pandas as pd

test_comments = pd.read_csv('Dawn_of_the_AI_Wars.csv')
test_comments.shape
```

Out[633]:
```
(358, 9)
```

In [634]:
```
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    '''Removes HTML tags: replaces anything between opening and closing <> with empty spac

    return TAG_RE.sub('', text)
```

In [635]:
```
def preprocess_text(sen):
    '''Cleans text data up, leaving only 2 or more char long non-stepwords composed of A-Z
    in lowercase'''

    sentence = sen.lower()

    # Remove html tags
    sentence = remove_tags(sentence)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)  # When we remove apostrophe from

    # Remove multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)  # Next, we remove all the single characters

    # Remove Stopwords
    pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s*')
    sentence = pattern.sub('', sentence)

    return sentence
```

In [445]:
```
from nltk.sentiment import SentimentIntensityAnalyzer

# Download NLTK resources
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('vader_lexicon')

test_comments['processed_comment'] = test_comments['comment'].apply(preprocess_text)

# Perform sentiment analysis
sia = SentimentIntensityAnalyzer()
test_comments['sentiment_score'] = test_comments['processed_comment'].apply(lambda x: sia.

# Categorize sentiments
test_comments['sentiment'] = test_comments['sentiment_score'].apply(lambda x: 'positive' i

# Save the processed DataFrame to a CSV file
csv_file_path = 'Dawn_of_the_AI_Wars_sentiment_analysis.csv'
comments_df.to_csv(csv_file_path, index=False)

print(f'Data has been saved to {csv_file_path}')
```

```
Data has been saved to Dawn_of_the_AI_Wars_sentiment_analysis.csv

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\bhave\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [636]:
```
sample_comments = pd.read_csv('Dawn_of_the_AI_Wars_sentiment_analysis.csv')
sample_comments.shape
```

Out[636]:
```
(486, 12)
```

In [637]:
```
# Remove the columns which has neutral values.

sample_comments = sample_comments[sample_comments['sentiment'] != 'neutral']
```

In [638]:
```
# Preprocess comment text with earlier defined preprocess_text function

unseen_comments = sample_comments['comment']

unseen_processed = []
for comment in unseen_comments:
    comment = preprocess_text(comment)
    unseen_processed.append(comment)

sample_comments['processed_comment'] = unseen_processed

sample_comments.shape
```

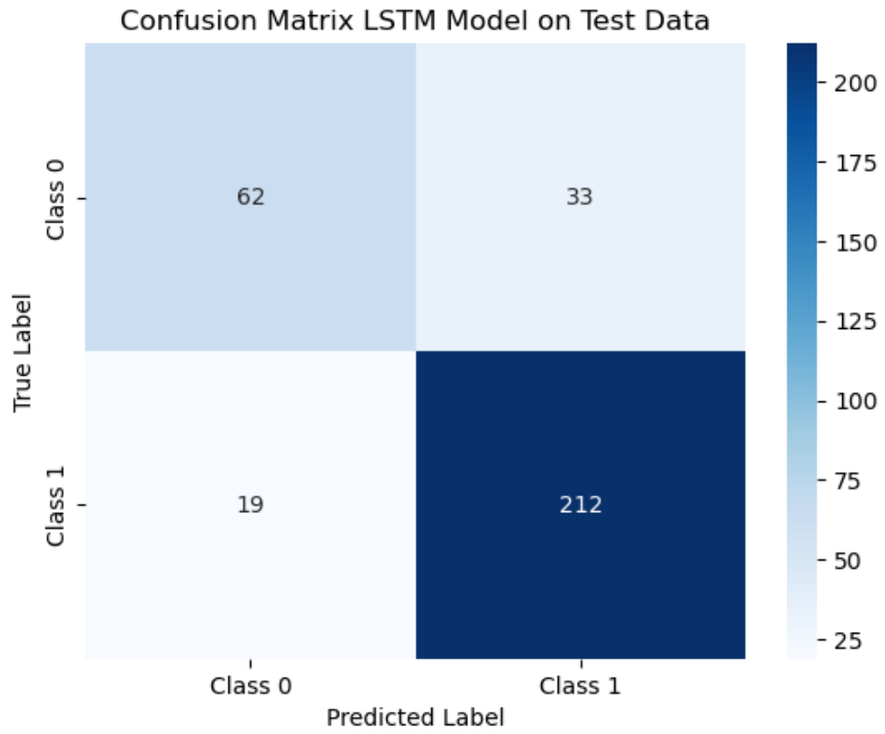
Out[638]: (326, 12)

In [639]:
```
# Tokenising instance with earlier trained tokeniser
unseen_tokenized = word_tokenizer.texts_to_sequences(unseen_processed)

# Pooling instance to have maxlength of 100 tokens
unseen_padded = pad_sequences(unseen_tokenized, padding='post', maxlen=maxlen)
```

In [640]:
```
# Passing tokenised instance to the LSTM model for predictions
unseen_sentiments = lstm_model.predict(unseen_padded)
```

```
11/11 [==============================] - 1s 58ms/step
```

In [642]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
threshold = 0.5
sample_comments['Predicted Sentiments'] = np.where(unseen_sentiments > threshold, 1, 0)
sample_comments['Predicted Sentiments'] = np.array(list(map(lambda x: "positive" if x== 1 (

df_prediction_sentiments = pd.DataFrame(sample_comments['Predicted Sentiments'], columns =
df_comment_id                = pd.DataFrame(sample_comments['comment_id'], columns = ['co
df_comment_text           = pd.DataFrame(sample_comments['comment'], columns = ['comment']
df_vedar_score           = pd.DataFrame(sample_comments['sentiment'], columns = ['sentimen


dfx=pd.concat([df_comment_id, df_comment_text, df_vedar_score, df_prediction_sentiments],

dfx.to_csv("./cnnModel_Unseen_Predictions.csv", sep=',', encoding='UTF-8')

dfx.head(20)

# Create confusion matrix
conf_matrix = confusion_matrix(sample_comments['sentiment'], sample_comments['Predicted Se

# Plot confusion matrix with labels and colors
labels = ['Class 0', 'Class 1']
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabel
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix LSTM Model on Test Data')
plt.show()

print("\nClassification Report LSTM Model on Test Data:")
print(classification_report(sample_comments['sentiment'], sample_comments['Predicted Senti
```




```
Classification Report LSTM Model on Test Data:
              precision    recall  f1-score   support

    negative       0.77      0.65      0.70        95
    positive       0.87      0.92      0.89       231

    accuracy                           0.84       326
   macro avg       0.82      0.79      0.80       326
weighted avg       0.84      0.84      0.84       326
```

In [ ]: