```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.18.0-py3-none-any.whl (510 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 4.6 MB/s eta 0:0(
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pacl
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-|
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.10/d:
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/di:
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 6.4 MB/s eta 0:0(
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa(
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/(
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
Collecting xxhash (from datasets)
  Downloading xxhash-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 7.4 MB/s eta 0:0(
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 5.6 MB/s eta 0:0(
Requirement already satisfied: fsspec[http]<=2024.2.0,>=2023.1.0 in /usr/loca
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/pytho
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa(
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-|
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/(
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10,
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.:
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/di:
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt|
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1(
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1(
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho|
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/di:
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pacl
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-2.18.0 dill-0.3.8 multiprocess-0.70.16 xxhash-
```

```
!pip install contractions
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 3.7 MB/s eta 0:0
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.7/110.7 kB 14.9 MB/s eta 0:
Installing collected packages: pyahocorasick, anyascii, textsearch, contracti
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.1.0
```

```
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from gensim.models import Word2Vec
import numpy as np
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import contractions
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

Loading and Displaying Data

```
datasets, ds_info = tfds.load('huggingface:banking77', split=['train', 'test'], w
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow_datasets/core/dataset_builc
  hf_names = hf_datasets.list_datasets()
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: Us
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tak
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
  warnings.warn(
```

Downloading readme: 100% 14.4k/14.4k [00:00<00:00, 672kB/s]

```
Downloading and preparing dataset Unknown size (download: Unknown size, genera
```

Downloading data: 100% 298k/298k [00:01<00:00, 150kB/s]

Downloading data: 100% 93.9k/93.9k [00:01<00:00, 61.0kB/s]

Generating train split: 100% 10003/10003 [00:00<00:00, 140891.39 examples/s]

Generating test split: 100% 3080/3080 [00:00<00:00, 113296.93 examples/s]

```
# Unpack the datasets
dataset_train, dataset_test = datasets

# Extract the label names using the metadata
label_names = ds_info.features['label'].names
```

```
for example in dataset_train.take(5):
    print(example)
```

```
{'label': <tf.Tensor: shape=(), dtype=int64, numpy=11>, 'text': <tf.Tensor: sh
{'label': <tf.Tensor: shape=(), dtype=int64, numpy=11>, 'text': <tf.Tensor: sh
{'label': <tf.Tensor: shape=(), dtype=int64, numpy=11>, 'text': <tf.Tensor: sh
{'label': <tf.Tensor: shape=(), dtype=int64, numpy=11>, 'text': <tf.Tensor: sh
{'label': <tf.Tensor: shape=(), dtype=int64, numpy=11>, 'text': <tf.Tensor: sh
```

```
for example in dataset_train.take(5):
    text = example['text'].numpy().decode('utf-8')
    label = int(example['label'].numpy())
    category = label_names[label]
    print(f'Text: {text}, Label: {label}, Category: {category}')
```

```
Text: I am still waiting on my card?, Label: 11, Category: card_arrival
Text: What can I do if my card still hasn't arrived after 2 weeks?, Label: 11,
Text: I have been waiting over a week. Is the card still coming?, Label: 11, C
Text: Can I track my card while it is in the process of delivery?, Label: 11,
Text: How do I know if I will get my card, or if it is lost?, Label: 11, Categ
```

```
for example in dataset_test.take(5):
    text = example['text'].numpy().decode('utf-8')
    label = int(example['label'].numpy())
    category = label_names[label]
    print(f'Text: {text}, Label: {label}, Category: {category}')
```

```
Text: How do I locate my card?, Label: 11, Category: card_arrival
Text: I still have not received my new card, I ordered over a week ago., Label
Text: I ordered a card but it has not arrived. Help please!, Label: 11, Catego
Text: Is there a way to know when my card will arrive?, Label: 11, Category: c
Text: My card has not arrived yet., Label: 11, Category: card_arrival
```

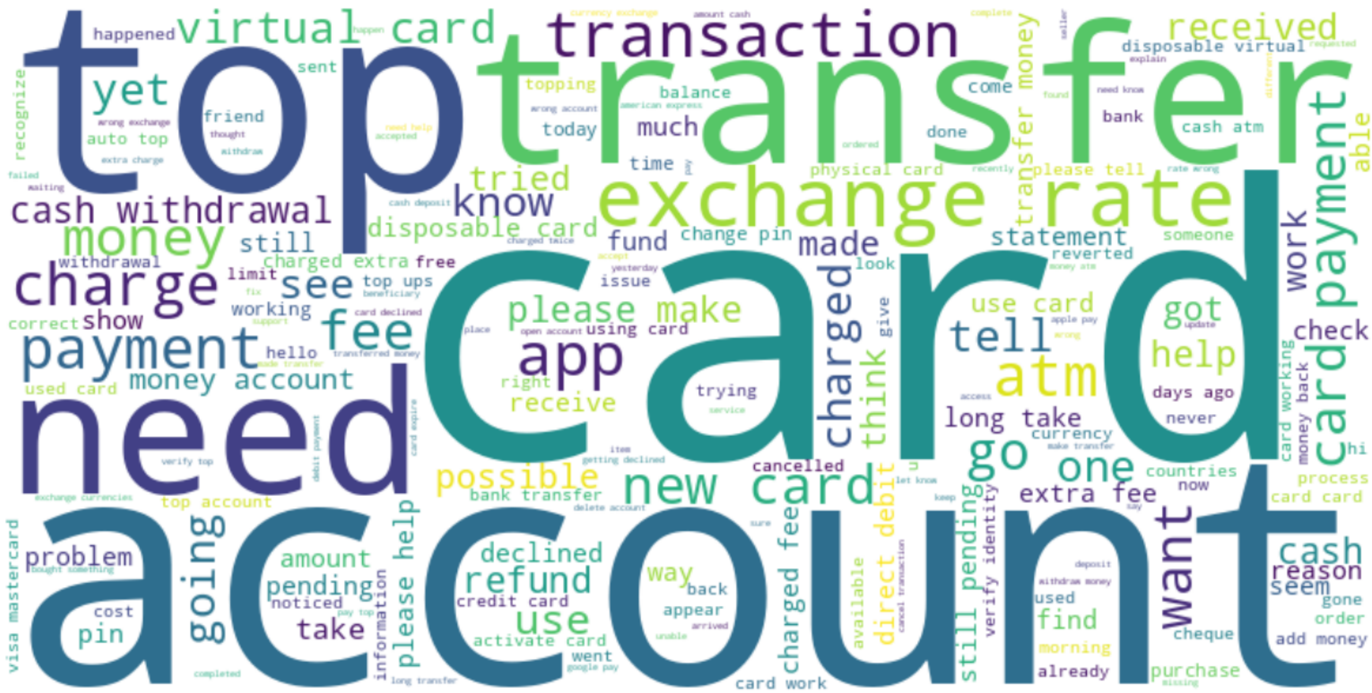
Exploratory Data Analysis

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

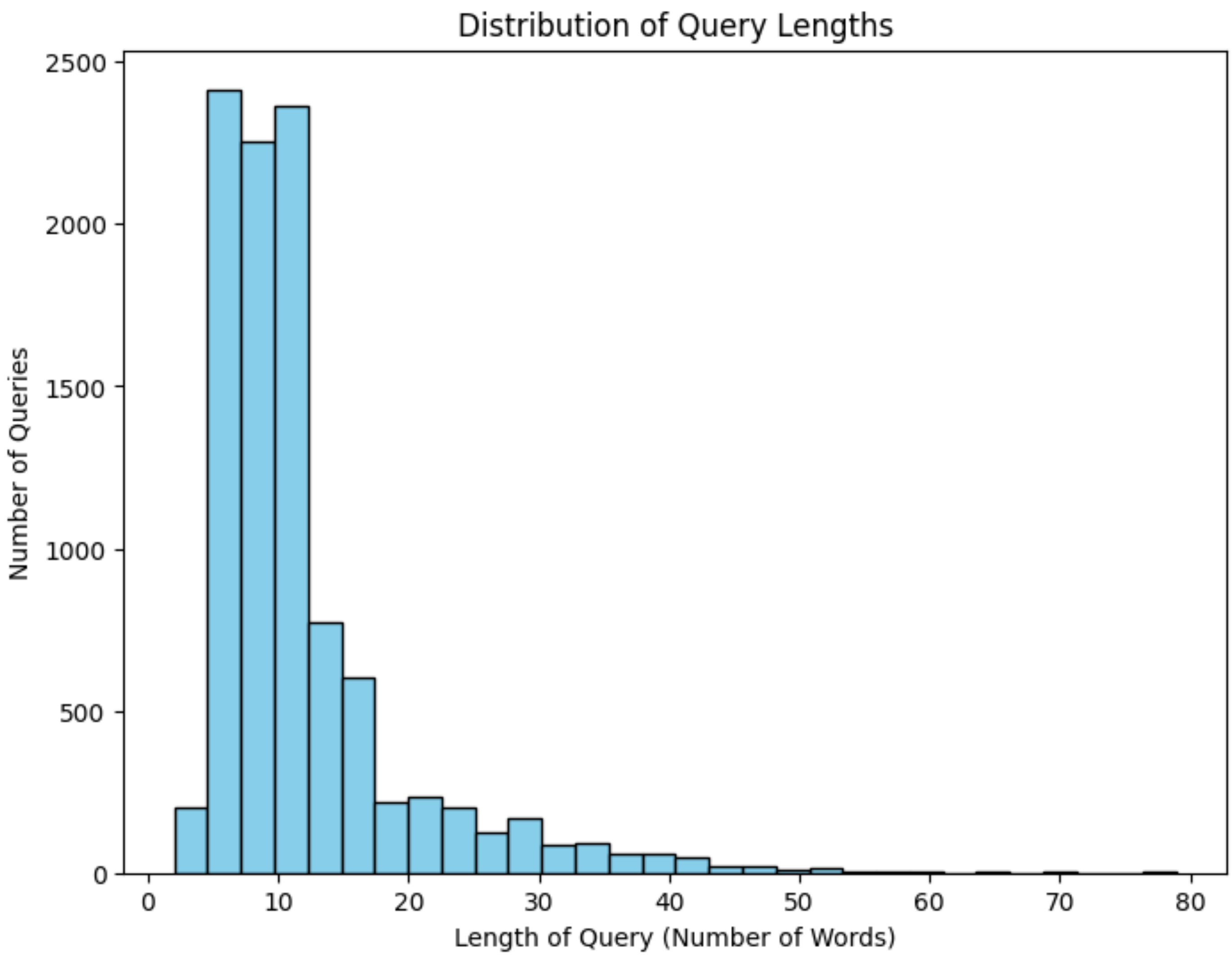
```
# Collect all labels from the dataset
labels = [int(example['label'].numpy()) for example in dataset_train]

# Count the occurrences of each label
label_counts = np.bincount(labels)
label_names = ds_info.features['label'].names

# Create a DataFrame with label numbers, names, and counts
label_data = {'Label Number': range(len(label_counts)),
              'Label Name': label_names,
              'Number of Examples': label_counts}

df = pd.DataFrame(label_data)

# Sort by 'Number of Examples' in descending order
df_sorted = df.sort_values('Number of Examples', ascending=False).reset_index(dro

# Display the DataFrame
print(df_sorted)
```

```
    Label Number                                  Label Name  \
0             15                    card_payment_fee_charged
1             28          direct_debit_payment_not_recognised
2              6  balance_not_updated_after_cheque_or_cash_deposit
3             75                wrong_amount_of_cash_received
4             19                      cash_withdrawal_charge
..           ...                                         ...
72            41                         lost_or_stolen_card
73            18                              card_swallowed
74            10                             card_acceptance
75            72                    virtual_card_not_working
76            23                     contactless_not_working

    Number of Examples
0                  187
1                  182
2                  181
3                  180
4                  177
..                 ...
72                  82
73                  61
74                  59
75                  41
76                  35

[77 rows x 3 columns]
```

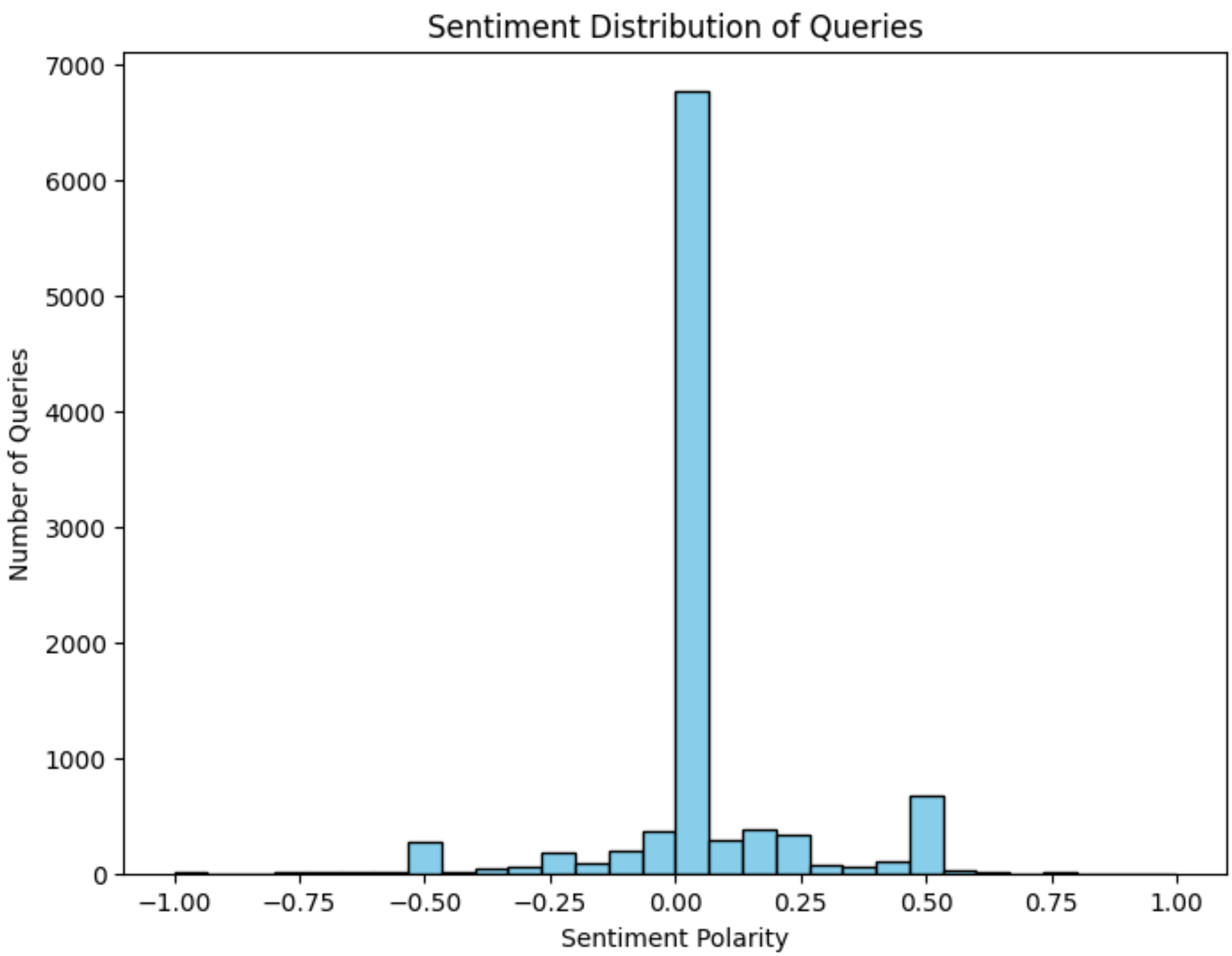
```
from wordcloud import WordCloud
# Extract texts from the dataset and remove stop words
stop_words = set(stopwords.words('english'))
texts = [' '.join(word for word in example['text'].numpy().decode('utf-8').lower(
         for example in dataset_train]

# Join all texts to create a combined string
combined_texts = ' '.join(texts)

# Create and generate a word cloud image
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(c

# Display the word cloud using matplotlib
plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Hide the axes
plt.show()
```


top
transfer
card
account
need
exchange rate
transaction
virtual card
received
payment
charge
money
yet
cash withdrawal
fee
app
know
tried
atm
charged
new card
go
one
want
tell
got
help
please make
refund
declined
use
going
problem
amount
pending
take
possible
cash
reason
seem
find
still
see
made
fund
statement
work
card payment
disposable card
think
direct debit
extra fee
long take
check
transfer money
much
today
balance
able
topping
friend
auto top
happened
sent
recognize
change pin
still pending
charged fee
please help
verify identity
information
credit card
noticed
activate card
appear
back
pin
cost
visa mastercard
money account
receive
hello
working
show
correct
withdrawal
limit
top ups
charged extra
free
physical card
please tell
transfer pending
come
done
bank
cash atm
disposable virtual
someone
reverted
use card
days ago
never
currency
cancelled
bank transfer
countries
now
process
card card
used
gone
order
cheque
add money
morning
already
purchase
card work
went
top account
used card
way
available
give
time
issue
look
using card
trying
right
card working
money back
hi

```
# Query Length Analysis
text_lengths = [len(example['text'].numpy().decode('utf-8').split()) for example

plt.figure(figsize=(8, 6))
plt.hist(text_lengths, bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Query Lengths')
plt.xlabel('Length of Query (Number of Words)')
plt.ylabel('Number of Queries')
plt.show()
```

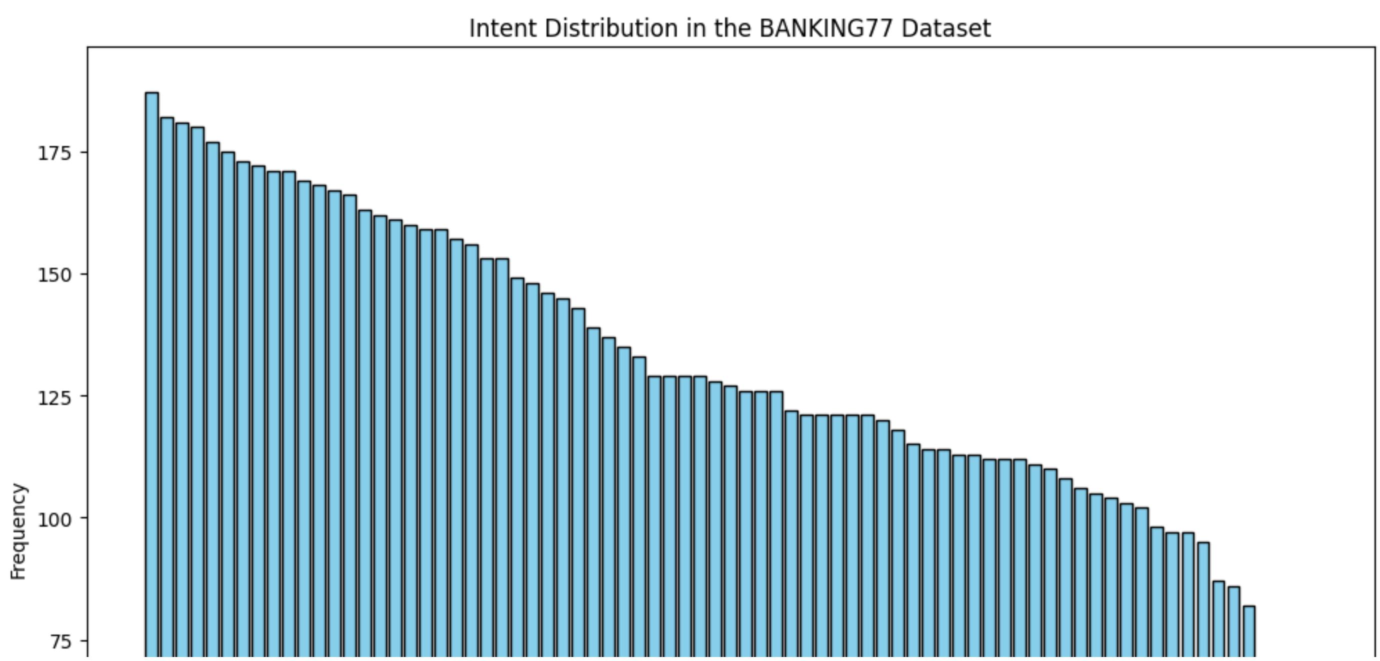


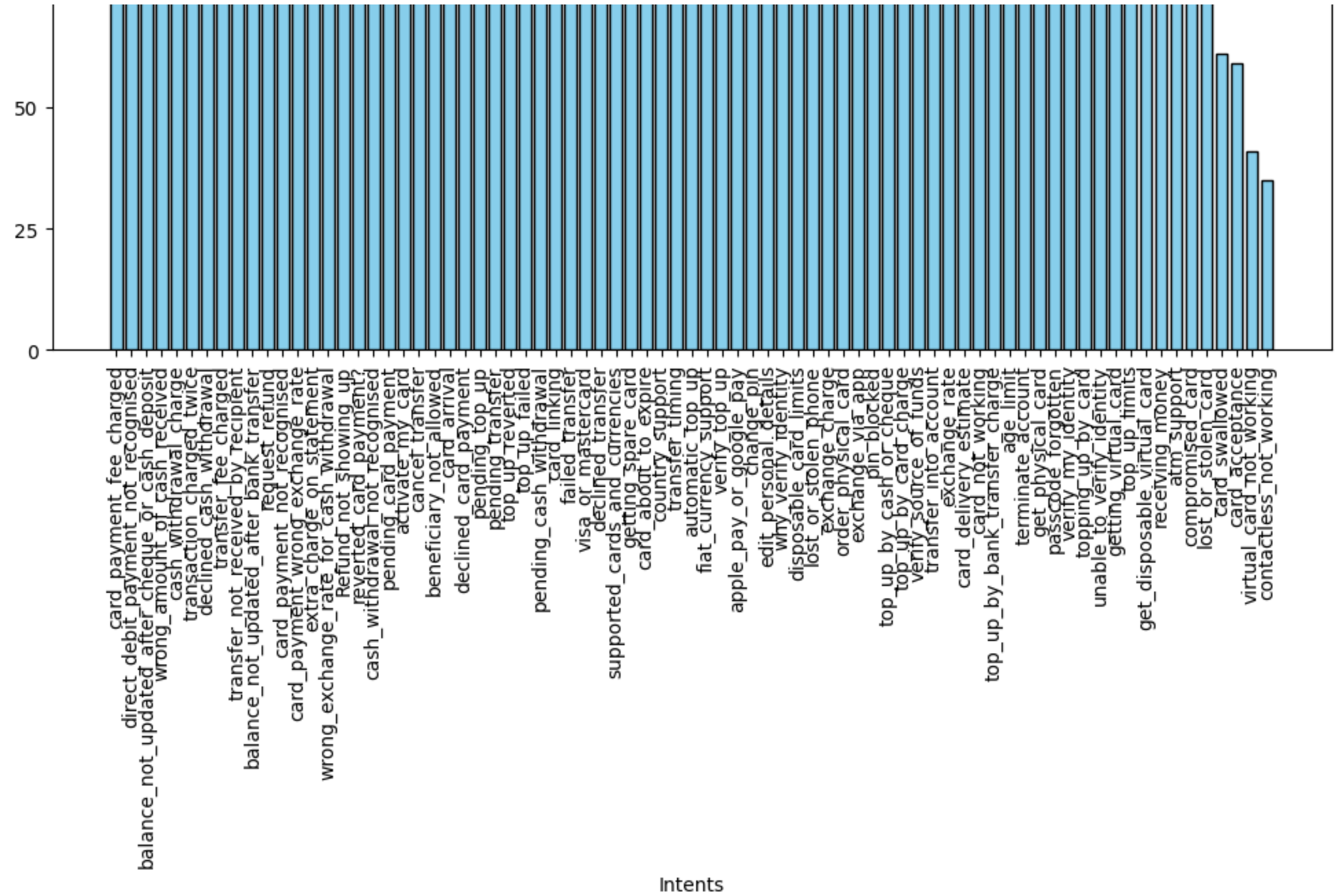
```
from textblob import TextBlob

# Sentiment analysis on the texts
sentiments = [TextBlob(example['text'].numpy().decode('utf-8')).sentiment.polarit

plt.figure(figsize=(8, 6))
plt.hist(sentiments, bins=30, color='skyblue', edgecolor='black')
plt.title('Sentiment Distribution of Queries')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Number of Queries')
plt.show()
```



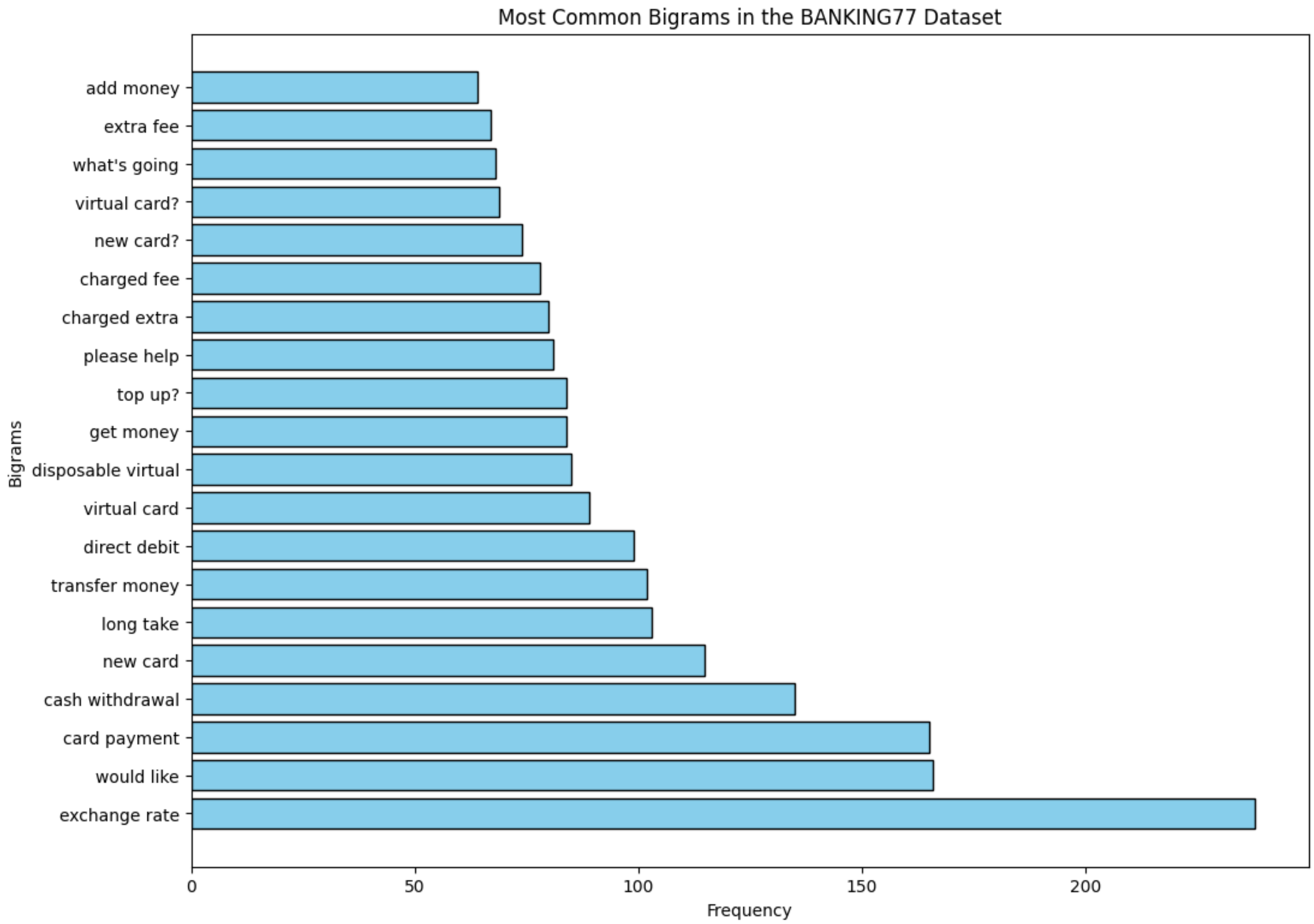

```
# Extract intent labels from the dataset
```

```
intent_labels = [int(example['label'].numpy()) for example in dataset_train]

from collections import Counter
import matplotlib.pyplot as plt

# Count frequencies of each label
intent_freq = Counter(intent_labels)

# Get label names using dataset metadata
label_names = ds_info.features['label'].names

# Map label indices to label names
labeled_intents = [label_names[label] for label in intent_labels]

# Count frequencies of each named intent
named_intent_freq = Counter(labeled_intents)

# Extracting data for plotting
intents, intent_counts = zip(*named_intent_freq.most_common())

# Plotting the distribution of intents
plt.figure(figsize=(12, 9))
plt.bar(intents, intent_counts, color='skyblue', edgecolor='black')
plt.title('Intent Distribution in the BANKING77 Dataset')
plt.xticks(rotation=90)
plt.xlabel('Intents')
plt.ylabel('Frequency')
plt.show()
```

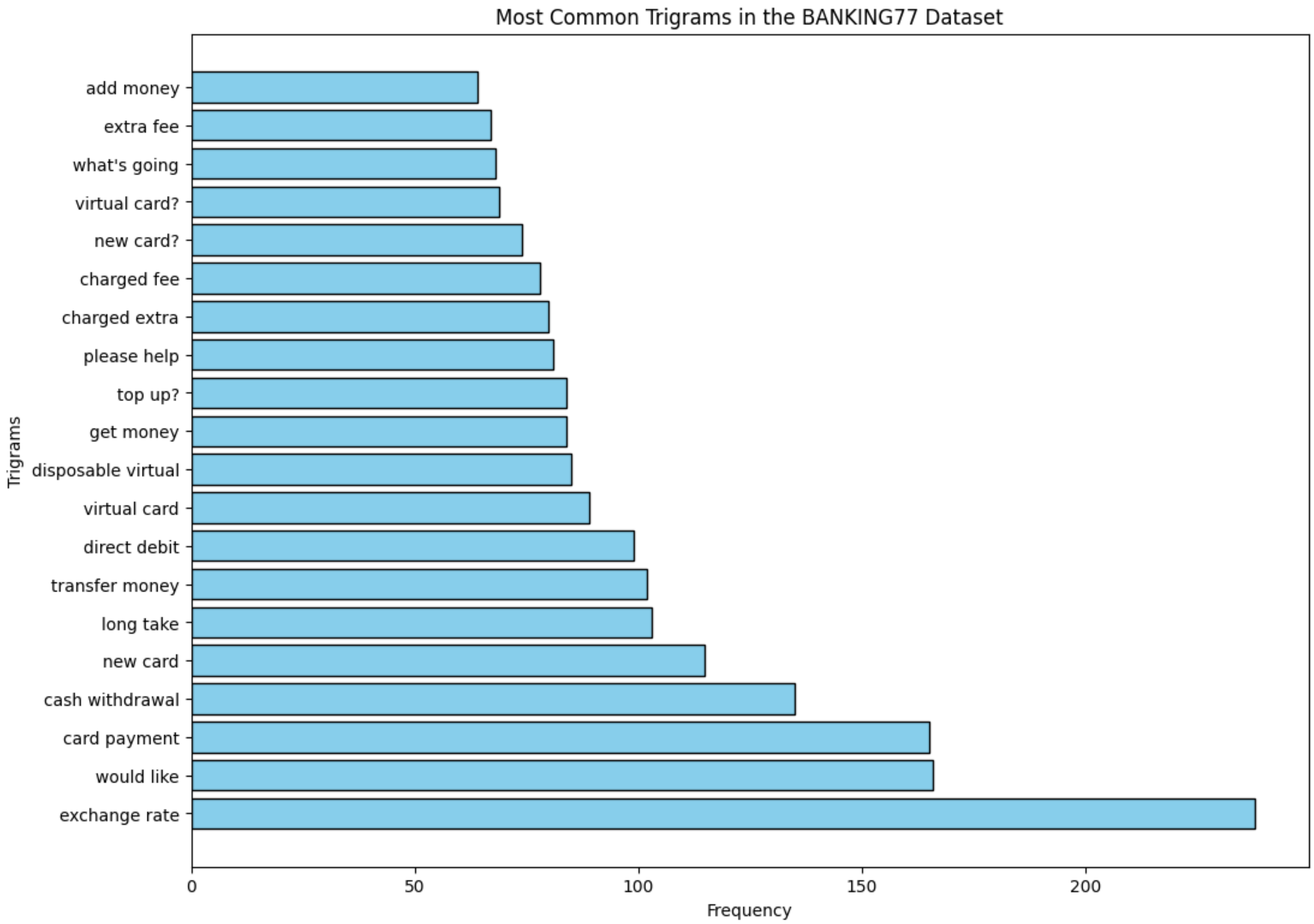



50
25
0
card_payment_fee_charged
direct_debit_payment_not_recognised
balance_not_updated_after_cheque_or_cash_deposit
wrong_amount_of_cash_received
cash_withdrawal_charge
transaction_charged_twice
declined_cash_withdrawal
transfer_fee_charged
transfer_not_received_by_recipient
balance_not_updated_after_bank_transfer
request_refund
card_payment_not_recognised
card_payment_wrong_exchange_rate
extra_charge_on_statement
wrong_exchange_rate_for_cash_withdrawal
Refund_not_showing_up
reverted_card_payment?
cash_withdrawal_not_recognised
pending_card_payment
activate_my_card
cancel_transfer
beneficiary_not_allowed
card_arrival
declined_card_payment
pending_top_up
pending_transfer
top_up_reverted
top_up_failed
pending_cash_withdrawal
card_linking
failed_transfer
visa_or_mastercard
declined_transfer
supported_cards_and_currencies
getting_spare_card
card_about_to_expire
country_support
transfer_timing
automatic_top_up
fiat_currency_support
verify_top_up
apple_pay_or_google_pay
change_pin
edit_personal_details
why_verify_identity
disposable_card_limits
lost_or_stolen_phone
exchange_charge
order_physical_card
exchange_via_app
pin_blocked
top_up_by_cash_or_cheque
top_up_by_card_charge
verify_source_of_funds
transfer_into_account
exchange_rate
card_delivery_estimate
card_not_working
top_up_by_bank_transfer_charge
age_limit
terminate_account
get_physical_card
passcode_forgotten
verify_my_identity
topping_up_by_card
unable_to_verify_identity
getting_virtual_card
top_up_limits
get_disposable_virtual_card
receiving_money
atm_support
compromised_card
lost_or_stolen_card
card_swallowed
card_acceptance
virtual_card_not_working
contactless_not_working
Intents

```
from nltk import ngrams

def extract_ngrams(data, num):
    n_grams = ngrams(data, num)
    return [' '.join(grams) for grams in n_grams]

# Extract bigrams from texts
bigrams = Counter()
for example in dataset_train:
    text = example['text'].numpy().decode('utf-8').lower().split()
    filtered_text = [word for word in text if word not in stop_words]
    bigrams.update(extract_ngrams(filtered_text, 2))

most_common_bigrams = bigrams.most_common(20)
bigram_words, bigram_counts = zip(*most_common_bigrams)

plt.figure(figsize=(12, 9))
plt.barh(bigram_words, bigram_counts, color='skyblue', edgecolor='black')
plt.title('Most Common Bigrams in the BANKING77 Dataset')
plt.xlabel('Frequency')
plt.ylabel('Bigrams')
plt.show()
```


Most Common Bigrams in the BANKING77 Dataset
add money
extra fee
what's going
virtual card?
new card?
charged fee
charged extra
please help
top up?
get money
disposable virtual
virtual card
direct debit
transfer money
long take
new card
cash withdrawal
card payment
would like
exchange rate
Bigrams
0
50
100
150
200
Frequency

```
def extract_ngrams(data, num):
    n_grams = ngrams(data, num)
    return [' '.join(grams) for grams in n_grams]

# Extract bigrams from texts
trigrams = Counter()
for example in dataset_train:
    text = example['text'].numpy().decode('utf-8').lower().split()
    filtered_text = [word for word in text if word not in stop_words]
    trigrams.update(extract_ngrams(filtered_text, 3))

most_common_trigrams = trigrams.most_common(20)
trigram_words, trigram_counts = zip(*most_common_trigrams)

plt.figure(figsize=(12, 9))
plt.barh(bigram_words, bigram_counts, color='skyblue', edgecolor='black')
plt.title('Most Common Trigrams in the BANKING77 Dataset')
plt.xlabel('Frequency')
plt.ylabel('Trigrams')
plt.show()
```


Most Common Trigrams in the BANKING77 Dataset
add money
extra fee
what's going
virtual card?
new card?
charged fee
charged extra
please help
top up?
get money
disposable virtual
virtual card
direct debit
transfer money
long take
new card
cash withdrawal
card payment
would like
exchange rate
Trigrams
0
50
100
150
200
Frequency

```
# Clear session
tf.keras.backend.clear_session()
```

Data Preprocessing

```
def load_and_shuffle_data(shuffle_train=False):
  dataset_train, dataset_test = tfds.load('huggingface:banking77', split=['train'
  if shuffle_train:
      dataset_train = dataset_train.shuffle(buffer_size=10000, seed=42)
  train_data = list(dataset_train)
  test_data = list(dataset_test)
  sentences_train = [example['text'].numpy().decode('utf-8') for example in train
  sentences_test = [example['text'].numpy().decode('utf-8') for example in test_d
  labels_train = [example['label'].numpy() for example in train_data]
  labels_test = [example['label'].numpy() for example in test_data]
  return sentences_train, sentences_test, labels_train, labels_test
```

```
def preprocess_sentences_withcontractions(sentences):
    stop_words = set(stopwords.words('english'))  # Set of English stop words
    punctuation_table = str.maketrans('', '', string.punctuation)  # Mapping tabl
    lemmatizer = WordNetLemmatizer()  # NLTK Word lemmatizer

    processed_sentences = []
    for sentence in sentences:
        sentence = contractions.fix(sentence)
        # Create space around punctuation and remove double spaces
        sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)
        sentence = re.sub(r'[" "]+', " ", sentence)

        # Tokenize the sentence into words
        words = word_tokenize(sentence)

        # Convert each word to lowercase, remove stopwords and punctuation
        words = [word.lower() for word in words ]

        # Remove non-alphanumeric characters and empty strings
        words = [re.sub(r'\W+', '', word) for word in words if word.isalnum()]

        # Lemmatize each word
        #words = [lemmatizer.lemmatize(word) for word in words]

        # Append the list of cleaned words to the processed_sentences list
        processed_sentences.append(words)

    return processed_sentences
```

```
def preprocess_sentences_withoutpunctuations_lemmatized(sentences):
    stop_words = set(stopwords.words('english'))  # Set of English stop words
    punctuation_table = str.maketrans('', '', string.punctuation)  # Mapping tabl
    lemmatizer = WordNetLemmatizer()  # NLTK Word lemmatizer

    processed_sentences = []
    for sentence in sentences:
        # Create space around punctuation and remove double spaces
        sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)
        sentence = re.sub(r'[" "]+', " ", sentence)

        # Tokenize the sentence into words
        words = word_tokenize(sentence)

        # Convert each word to lowercase, remove stopwords and punctuation
        words = [word.lower() for word in words if word not in stop_words and wor

        # Remove non-alphanumeric characters and empty strings
        words = [re.sub(r'\W+', '', word) for word in words if word.isalnum()]

        # Lemmatize each word
        words = [lemmatizer.lemmatize(word) for word in words]

        # Append the list of cleaned words to the processed_sentences list
        processed_sentences.append(words)

    return processed_sentences
```

```
def preprocess_sentences_withpunctuations_lemmatized(sentences):
    stop_words = set(stopwords.words('english'))  # Set of English stop words
    punctuation_table = str.maketrans('', '', string.punctuation)  # Mapping tabl
    lemmatizer = WordNetLemmatizer()  # NLTK Word lemmatizer

    processed_sentences = []
    for sentence in sentences:
        # Create space around punctuation and remove double spaces
        sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)
        sentence = re.sub(r'[" "]+', " ", sentence)

        # Tokenize the sentence into words
        words = word_tokenize(sentence)

        # Convert each word to lowercase, remove stopwords
        words = [word.lower() for word in words]

        # Remove non-alphanumeric characters and empty strings
        words = [re.sub(r'\W+', '', word) for word in words if word.isalnum()]

        # Lemmatize each word
        words = [lemmatizer.lemmatize(word) for word in words]

        # Append the list of cleaned words to the processed_sentences list
        processed_sentences.append(words)

    return processed_sentences
```

```
def preprocess_sentences_withoutpunctuations_notlemmatized(sentences):
    stop_words = set(stopwords.words('english'))  # Set of English stop words
    punctuation_table = str.maketrans('', '', string.punctuation)  # Mapping tabl
    lemmatizer = WordNetLemmatizer()  # NLTK Word lemmatizer

    processed_sentences = []
    for sentence in sentences:
        # Create space around punctuation and remove double spaces
        sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)
        sentence = re.sub(r'[" "]+', " ", sentence)

        # Tokenize the sentence into words
        words = word_tokenize(sentence)

        # Convert each word to lowercase, remove stopwords and punctuation
        words = [word.lower() for word in words if word not in stop_words and wor

        # Remove non-alphanumeric characters and empty strings
        words = [re.sub(r'\W+', '', word) for word in words if word.isalnum()]

        # Lemmatize each word
        #words = [lemmatizer.lemmatize(word) for word in words]

        # Append the list of cleaned words to the processed_sentences list
        processed_sentences.append(words)

    return processed_sentences
```

```
def preprocess_sentences_withpunctuations_notlemmatized(sentences):
    stop_words = set(stopwords.words('english'))  # Set of English stop words
    punctuation_table = str.maketrans('', '', string.punctuation)  # Mapping tabl
    lemmatizer = WordNetLemmatizer()  # NLTK Word lemmatizer

    processed_sentences = []
    for sentence in sentences:
        # Create space around punctuation and remove double spaces
        sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)
        sentence = re.sub(r'[" "]+', " ", sentence)

        # Tokenize the sentence into words
        words = word_tokenize(sentence)

        # Convert each word to lowercase, remove stopwords
        words = [word.lower() for word in words]

        # Remove non-alphanumeric characters and empty strings
        words = [re.sub(r'\W+', '', word) for word in words if word.isalnum()]

        # Lemmatize each word
        #words = [lemmatizer.lemmatize(word) for word in words]

        # Append the list of cleaned words to the processed_sentences list
        processed_sentences.append(words)

    return processed_sentences
```

```
def create_word_index_and_train_word2vec(sentences, word_embedding_dim):
    word2vec_model = Word2Vec(sentences, vector_size=word_embedding_dim, window=1
    word2vec_model.build_vocab([["<UNK>"]], update=True)
    word_index = {word: idx + 1 for idx, word in enumerate(word2vec_model.wv.inde
    word_index['<UNK>'] = 0  # Assign index 0 to <UNK>
    return word_index, word2vec_model

def convert_words_to_ids(sentences, word_index):
    return [[word_index.get(word, word_index['<UNK>']) for word in sentence] for

def pad_sequences_to_max_length(sequences, max_length=None):
    if max_length is None:
        max_length = max(len(seq) for seq in sequences)
    return pad_sequences(sequences, maxlen=max_length, padding='post', truncating

def create_embedding_matrix(word_index, word2vec_model, embedding_dim):
    max_features = len(word_index) + 1  # <UNK> uses 0 index
    embedding_matrix = np.zeros((max_features, embedding_dim))
    for word, i in word_index.items():
        if word in word2vec_model.wv and i > 0:  # Ensure i > 0 to skip <UNK>
            embedding_matrix[i] = word2vec_model.wv[word]
    return embedding_matrix
```

```
def create_word_index_and_train_word2vec_handlingoov(sentences, word_embedding_di
    sentences = [['<UNK>']] + sentences
    word2vec_model = Word2Vec(sentences, vector_size=word_embedding_dim, window=2
    word_index = {word: idx + 1 for idx, word in enumerate(word2vec_model.wv.inde
    word_index['<UNK>'] = 0 if initialize_unk_zero else word2vec_model.wv.key_to_
    return word_index, word2vec_model

def convert_words_to_ids(sentences, word_index):
    return [[word_index.get(word, word_index['<UNK>']) for word in sentence] for

def pad_sequences_to_max_length(sequences, max_length=None):
    if max_length is None:
        max_length = max(len(seq) for seq in sequences)
    return pad_sequences(sequences, maxlen=max_length, padding='post', truncating

def create_embedding_matrix_handlingoov(word_index, word2vec_model, embedding_dim
    max_index = max(word_index.values())
    embedding_matrix = np.zeros((max_index+1, embedding_dim))
    for word, i in word_index.items():
        if word in word2vec_model.wv:
            embedding_matrix[i] = word2vec_model.wv[word]
    return embedding_matrix
```

LSTM model without punctuations and stopwords - Lemmatized

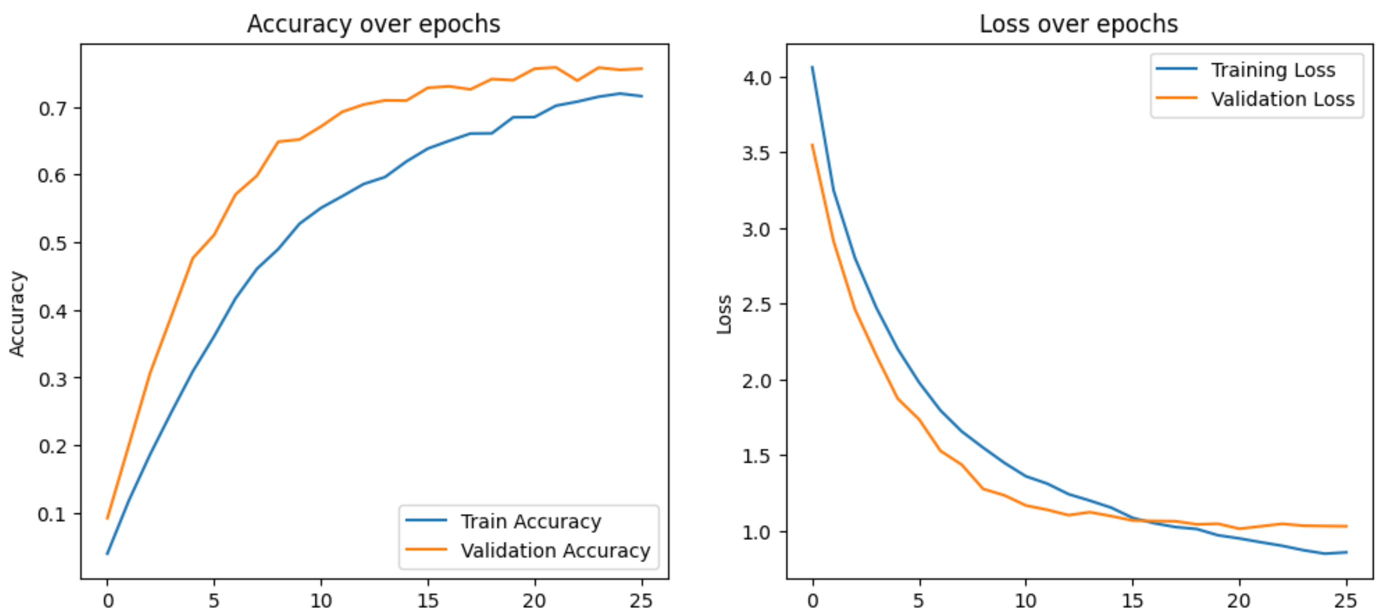
```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withoutpunctuations_lemmatized(s
tokenized_sentences_test = preprocess_sentences_withoutpunctuations_lemmatized(se
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)
```

```
# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_ma
    LSTM(64, return_sequences=True),  # More units and returning sequences for st
    Dropout(0.3),
    LSTM(32),  # Additional LSTM layer
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_l

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

```
print(f"Test Loss: {test_loss:.4f}")

# Optionally, visualize training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 39s 97ms/step - loss: 4.0606 - accu
Epoch 2/30
  1/313 [..............................] - ETA: 37s - loss: 3.6581 - accuracy:
  saving_api.save_model(
313/313 [==============================] - 26s 84ms/step - loss: 3.2476 - accu
Epoch 3/30
313/313 [==============================] - 28s 90ms/step - loss: 2.8010 - accu
Epoch 4/30
313/313 [==============================] - 26s 83ms/step - loss: 2.4722 - accu
Epoch 5/30
313/313 [==============================] - 26s 83ms/step - loss: 2.1999 - accu
Epoch 6/30
313/313 [==============================] - 26s 83ms/step - loss: 1.9791 - accu
Epoch 7/30
313/313 [==============================] - 26s 83ms/step - loss: 1.7939 - accu
Epoch 8/30
313/313 [==============================] - 26s 83ms/step - loss: 1.6545 - accu
Epoch 9/30
313/313 [==============================] - 26s 82ms/step - loss: 1.5477 - accu
Epoch 10/30
313/313 [==============================] - 26s 82ms/step - loss: 1.4466 - accu
Epoch 11/30
313/313 [==============================] - 26s 82ms/step - loss: 1.3584 - accu
Epoch 12/30
```

```
313/313 [==============================] - 26s 84ms/step - loss: 1.3099 - accu
Epoch 13/30
313/313 [==============================] - 26s 82ms/step - loss: 1.2401 - accu
Epoch 14/30
313/313 [==============================] - 25s 81ms/step - loss: 1.1973 - accu
Epoch 15/30
313/313 [==============================] - 26s 82ms/step - loss: 1.1508 - accu
Epoch 16/30
313/313 [==============================] - 25s 80ms/step - loss: 1.0834 - accu
Epoch 17/30
313/313 [==============================] - 25s 81ms/step - loss: 1.0484 - accu
Epoch 18/30
313/313 [==============================] - 36s 115ms/step - loss: 1.0229 - acc
Epoch 19/30
313/313 [==============================] - 27s 88ms/step - loss: 1.0099 - accu
Epoch 20/30
313/313 [==============================] - 26s 82ms/step - loss: 0.9685 - accu
Epoch 21/30
313/313 [==============================] - 26s 82ms/step - loss: 0.9475 - accu
Epoch 22/30
313/313 [==============================] - 26s 84ms/step - loss: 0.9226 - accu
Epoch 23/30
313/313 [==============================] - 26s 82ms/step - loss: 0.8991 - accu
Epoch 24/30
313/313 [==============================] - 26s 82ms/step - loss: 0.8697 - accu
Epoch 25/30
313/313 [==============================] - 26s 82ms/step - loss: 0.8470 - accu
Epoch 26/30
313/313 [==============================] - 26s 82ms/step - loss: 0.8556 - accu
Epoch 26: early stopping
97/97 [==============================] - 4s 14ms/step - loss: 1.0116 - accurac
Test Accuracy: 75.62%
Test Loss: 1.0116
```



Epochs Epochs

LSTM model with punctuations and stopwords - Lemmatized

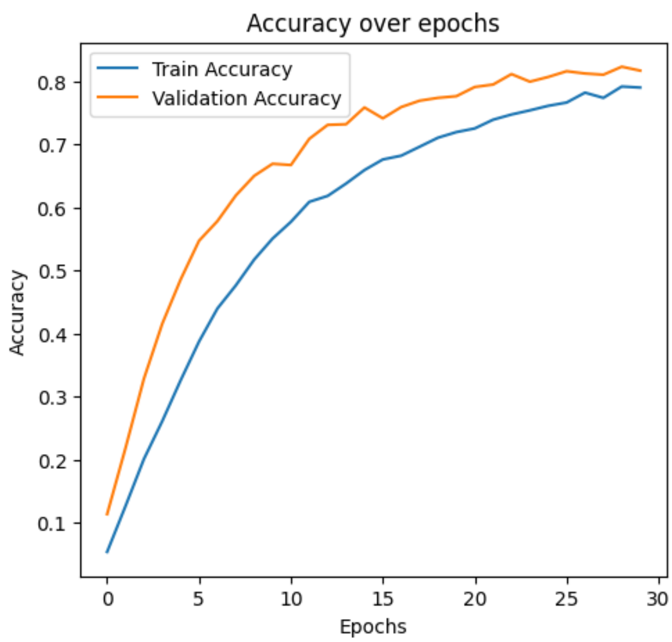
```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_data
tokenized_sentences_train = preprocess_sentences_withpunctuations_lemmatized(sent
tokenized_sentences_test = preprocess_sentences_withpunctuations_lemmatized(sente
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_m
    LSTM(64, return_sequences=True),  # More units and returning sequences for st
    Dropout(0.3),
    LSTM(32),  # Additional LSTM layer
```

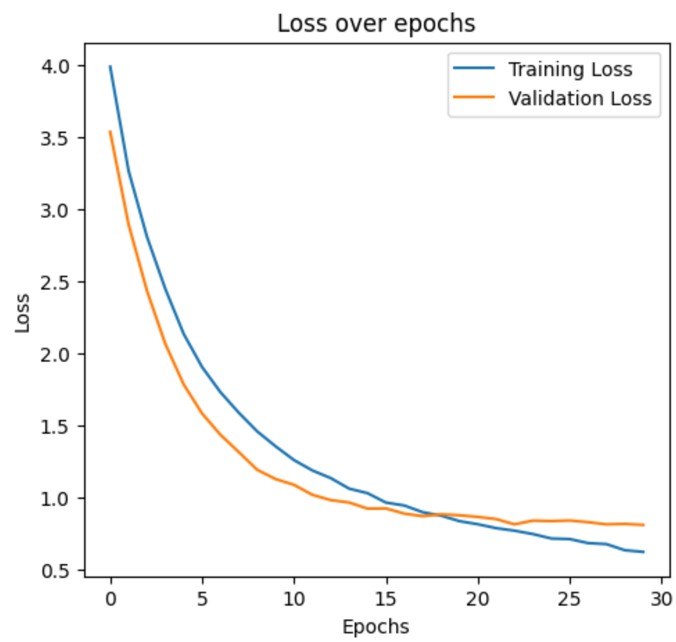
```
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model2.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Optionally, visualize training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 51s 133ms/step - loss: 3.9839 - acc
Epoch 2/30
313/313 [==============================] - 39s 124ms/step - loss: 3.2625 - acc
Epoch 3/30
313/313 [==============================] - 39s 123ms/step - loss: 2.8055 - acc
Epoch 4/30
313/313 [==============================] - 39s 124ms/step - loss: 2.4431 - acc
Epoch 5/30
313/313 [==============================] - 39s 125ms/step - loss: 2.1311 - acc
Epoch 6/30
313/313 [==============================] - 39s 124ms/step - loss: 1.9019 - acc
Epoch 7/30
313/313 [==============================] - 39s 125ms/step - loss: 1.7275 - acc
Epoch 8/30
313/313 [==============================] - 39s 125ms/step - loss: 1.5848 - acc
Epoch 9/30
313/313 [==============================] - 39s 125ms/step - loss: 1.4556 - acc
Epoch 10/30
313/313 [==============================] - 39s 124ms/step - loss: 1.3522 - acc
Epoch 11/30
313/313 [==============================] - 40s 127ms/step - loss: 1.2569 - acc
Epoch 12/30
313/313 [==============================] - 39s 125ms/step - loss: 1.1850 - acc
Epoch 13/30
313/313 [==============================] - 40s 127ms/step - loss: 1.1319 - acc
Epoch 14/30
313/313 [==============================] - 39s 126ms/step - loss: 1.0590 - acc
Epoch 15/30
313/313 [==============================] - 39s 124ms/step - loss: 1.0275 - acc
Epoch 16/30
313/313 [==============================] - 39s 124ms/step - loss: 0.9624 - acc
Epoch 17/30
313/313 [==============================] - 39s 125ms/step - loss: 0.9420 - acc
Epoch 18/30
313/313 [==============================] - 39s 125ms/step - loss: 0.8956 - acc
Epoch 19/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8720 - acc
```

```
Epoch 20/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8336 - acc
Epoch 21/30
313/313 [==============================] - 39s 125ms/step - loss: 0.8121 - acc
Epoch 22/30
313/313 [==============================] - 39s 124ms/step - loss: 0.7850 - acc
Epoch 23/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7674 - acc
Epoch 24/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7439 - acc
Epoch 25/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7133 - acc
Epoch 26/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7095 - acc
Epoch 27/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6814 - acc
Epoch 28/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6743 - acc
Epoch 29/30
313/313 [==============================] - 39s 124ms/step - loss: 0.6318 - acc
Epoch 30/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6204 - acc
97/97 [==============================] - 5s 23ms/step - loss: 0.8076 - accurac
Test Accuracy: 81.69%
Test Loss: 0.8076
```





LSTM model without punctuations and stopwords - NotLemmatized

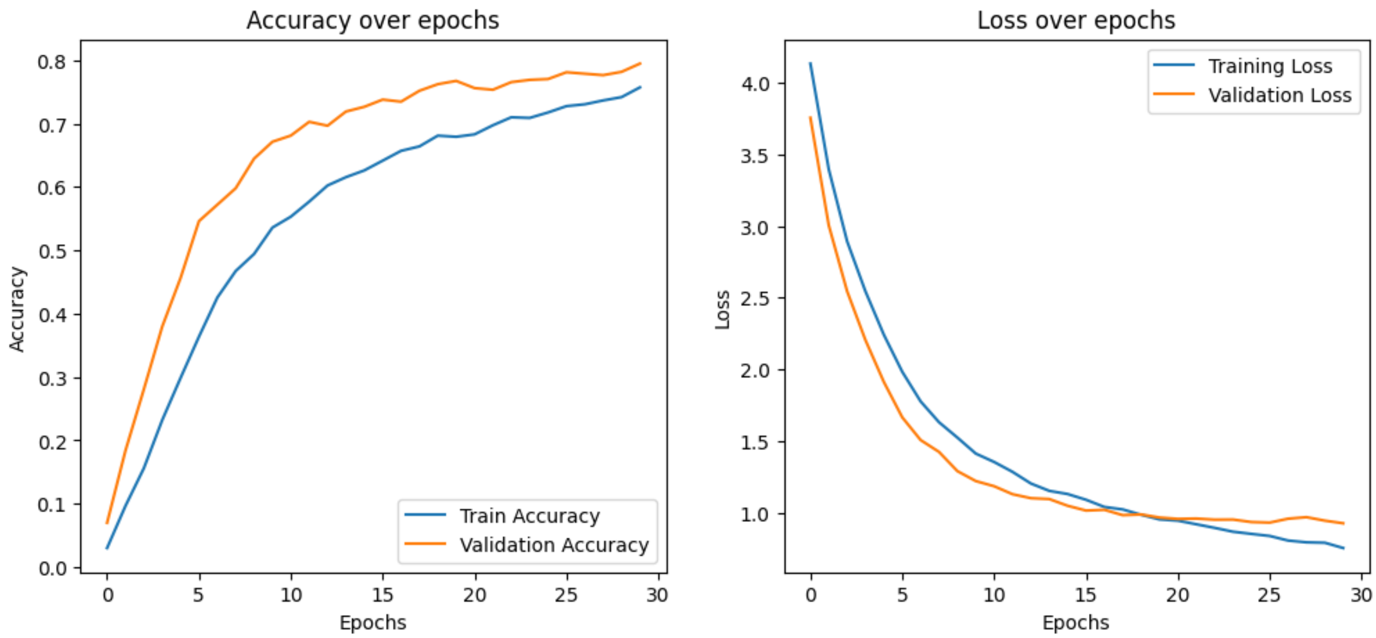
```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withoutpunctuations_notlemmatize
tokenized_sentences_test = preprocess_sentences_withoutpunctuations_notlemmatized
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_m
    LSTM(64, return_sequences=True),  # More units and returning sequences for st
    Dropout(0.3),
    LSTM(32),  # Additional LSTM layer
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
```

```
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model3.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Optionally, visualize training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 39s 98ms/step - loss: 4.1320 - accu
Epoch 2/30
313/313 [==============================] - 27s 87ms/step - loss: 3.3948 - accu
Epoch 3/30
313/313 [==============================] - 26s 84ms/step - loss: 2.8922 - accu
Epoch 4/30
313/313 [==============================] - 27s 87ms/step - loss: 2.5425 - accu
Epoch 5/30
313/313 [==============================] - 26s 84ms/step - loss: 2.2406 - accu
Epoch 6/30
313/313 [==============================] - 28s 88ms/step - loss: 1.9830 - accu
Epoch 7/30
313/313 [==============================] - 26s 85ms/step - loss: 1.7764 - accu
Epoch 8/30
313/313 [==============================] - 26s 85ms/step - loss: 1.6312 - accu
Epoch 9/30
313/313 [==============================] - 26s 84ms/step - loss: 1.5242 - accu
Epoch 10/30
313/313 [==============================] - 26s 84ms/step - loss: 1.4127 - accu
Epoch 11/30
313/313 [==============================] - 26s 84ms/step - loss: 1.3540 - accu
Epoch 12/30
313/313 [==============================] - 26s 84ms/step - loss: 1.2852 - accu
Epoch 13/30
313/313 [==============================] - 26s 84ms/step - loss: 1.2040 - accu
Epoch 14/30
313/313 [==============================] - 26s 84ms/step - loss: 1.1526 - accu
Epoch 15/30
313/313 [==============================] - 26s 84ms/step - loss: 1.1305 - accu
Epoch 16/30
313/313 [==============================] - 26s 84ms/step - loss: 1.0907 - accu
Epoch 17/30
313/313 [==============================] - 26s 84ms/step - loss: 1.0404 - accu
Epoch 18/30
313/313 [==============================] - 26s 84ms/step - loss: 1.0232 - accu
Epoch 19/30
313/313 [==============================] - 26s 84ms/step - loss: 0.9858 - accu
Epoch 20/30
313/313 [==============================] - 26s 84ms/step - loss: 0.9535 - accu
Epoch 21/30
313/313 [==============================] - 26s 84ms/step - loss: 0.9449 - accu
Epoch 22/30
313/313 [==============================] - 26s 84ms/step - loss: 0.9197 - accu
Epoch 23/30
313/313 [==============================] - 26s 84ms/step - loss: 0.8946 - accu
Epoch 24/30
313/313 [==============================] - 26s 84ms/step - loss: 0.8678 - accu
```

```
Epoch 25/30
313/313 [==============================] - 26s 84ms/step - loss: 0.8525 - accu
Epoch 26/30
313/313 [==============================] - 28s 88ms/step - loss: 0.8381 - accu
Epoch 27/30
313/313 [==============================] - 27s 86ms/step - loss: 0.8061 - accu
Epoch 28/30
313/313 [==============================] - 28s 88ms/step - loss: 0.7938 - accu
Epoch 29/30
313/313 [==============================] - 26s 84ms/step - loss: 0.7909 - accu
Epoch 30/30
313/313 [==============================] - 26s 84ms/step - loss: 0.7539 - accu
97/97 [==============================] - 4s 14ms/step - loss: 0.9264 - accurac
Test Accuracy: 79.48%
Test Loss: 0.9264
```



LSTM model with punctuations and stopwords - NonLemmatized

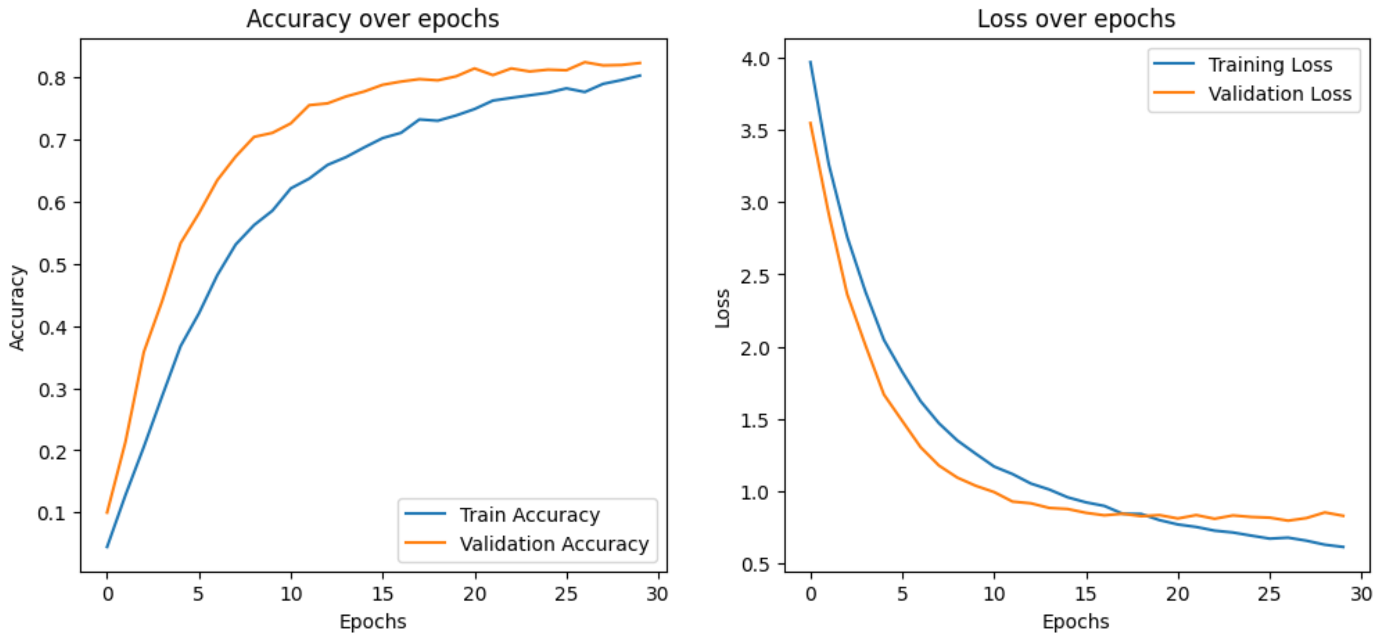
```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withpunctuations_notlemmatized(s
tokenized_sentences_test = preprocess_sentences_withpunctuations_notlemmatized(se
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_m
    LSTM(64, return_sequences=True),  # More units and returning sequences for st
    Dropout(0.3),
    LSTM(32),  # Additional LSTM layer
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
```

```
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model4.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Optionally, visualize training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
```

```
313/313 [==============================] - 52s 141ms/step - loss: 3.9667 - acc
Epoch 2/30
313/313 [==============================] - 39s 124ms/step - loss: 3.2610 - acc
Epoch 3/30
313/313 [==============================] - 39s 124ms/step - loss: 2.7614 - acc
Epoch 4/30
313/313 [==============================] - 39s 124ms/step - loss: 2.3789 - acc
Epoch 5/30
313/313 [==============================] - 39s 124ms/step - loss: 2.0476 - acc
Epoch 6/30
313/313 [==============================] - 39s 123ms/step - loss: 1.8263 - acc
Epoch 7/30
313/313 [==============================] - 39s 123ms/step - loss: 1.6234 - acc
Epoch 8/30
313/313 [==============================] - 41s 131ms/step - loss: 1.4706 - acc
Epoch 9/30
313/313 [==============================] - 39s 124ms/step - loss: 1.3519 - acc
Epoch 10/30
313/313 [==============================] - 39s 126ms/step - loss: 1.2604 - acc
Epoch 11/30
313/313 [==============================] - 40s 126ms/step - loss: 1.1720 - acc
Epoch 12/30
313/313 [==============================] - 39s 126ms/step - loss: 1.1202 - acc
Epoch 13/30
313/313 [==============================] - 39s 126ms/step - loss: 1.0545 - acc
Epoch 14/30
313/313 [==============================] - 39s 124ms/step - loss: 1.0132 - acc
Epoch 15/30
313/313 [==============================] - 39s 125ms/step - loss: 0.9595 - acc
Epoch 16/30
313/313 [==============================] - 40s 127ms/step - loss: 0.9245 - acc
Epoch 17/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8998 - acc
Epoch 18/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8456 - acc
Epoch 19/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8445 - acc
Epoch 20/30
313/313 [==============================] - 39s 124ms/step - loss: 0.8017 - acc
Epoch 21/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7717 - acc
Epoch 22/30
313/313 [==============================] - 39s 125ms/step - loss: 0.7545 - acc
Epoch 23/30
313/313 [==============================] - 39s 124ms/step - loss: 0.7290 - acc
Epoch 24/30
313/313 [==============================] - 39s 124ms/step - loss: 0.7162 - acc
Epoch 25/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6944 - acc
Epoch 26/30
313/313 [==============================] - 39s 126ms/step - loss: 0.6739 - acc
```

```
Epoch 27/30
313/313 [==============================] - 39s 126ms/step - loss: 0.6806 - acc
Epoch 28/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6595 - acc
Epoch 29/30
313/313 [==============================] - 39s 124ms/step - loss: 0.6319 - acc
Epoch 30/30
313/313 [==============================] - 39s 125ms/step - loss: 0.6162 - acc
97/97 [==============================] - 7s 26ms/step - loss: 0.7981 - accurac
Test Accuracy: 82.40%
Test Loss: 0.7981
```




```python
from sklearn.metrics import classification_report
best_model = tf.keras.models.load_model('best_model.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)

# Assuming ds_info is available and has a feature descriptor for labels
label_names = ds_info.features['label'].names
# Predicting the probabilities
predictions = best_model.predict(padded_test)
```

```
# Converting probabilities to class indices
predicted_classes = np.argmax(predictions, axis=1)

# Extract true classes from one-hot encoded test labels
true_classes = np.argmax(labels_test_one_hot, axis=1)

# Generate the classification report
report = classification_report(true_classes, predicted_classes, target_names=labe

# Print the classification report
print(report)
```

```
97/97 [==============================] - 8s 24ms/step - loss: 0.7981 - accurac
97/97 [==============================] - 5s 24ms/step
                                                 precision    recall  f1-scor

                                activate_my_card       0.95      0.95      0.9
                                       age_limit       0.93      0.97      0.9
                         apple_pay_or_google_pay       0.98      1.00      0.9
                                     atm_support       0.97      0.95      0.9
                                automatic_top_up       0.80      0.80      0.8
         balance_not_updated_after_bank_transfer       0.57      0.70      0.6
balance_not_updated_after_cheque_or_cash_deposit       0.83      0.72      0.7
                         beneficiary_not_allowed       0.70      0.78      0.7
                                 cancel_transfer       0.89      0.85      0.8
                           card_about_to_expire       0.95      0.95      0.9
                                 card_acceptance       0.71      0.60      0.6
                                    card_arrival       0.78      0.80      0.7
                          card_delivery_estimate       0.74      0.80      0.7
                                    card_linking       0.73      0.90      0.8
                                card_not_working       0.72      0.72      0.7
                        card_payment_fee_charged       0.82      0.80      0.8
                     card_payment_not_recognised       0.67      0.78      0.7
                card_payment_wrong_exchange_rate       0.83      0.88      0.8
                                  card_swallowed       0.97      0.82      0.8
                          cash_withdrawal_charge       0.95      0.90      0.9
                  cash_withdrawal_not_recognised       0.82      0.90      0.8
                                      change_pin       0.94      0.82      0.8
                               compromised_card       0.77      0.60      0.6
                         contactless_not_working       0.97      0.88      0.9
                                 country_support       0.91      0.97      0.9
                           declined_card_payment       0.69      0.78      0.7
                        declined_cash_withdrawal       0.82      0.93      0.8
                               declined_transfer       0.88      0.70      0.7
             direct_debit_payment_not_recognised       0.86      0.80      0.8
                          disposable_card_limits       0.79      0.75      0.7
                           edit_personal_details       0.87      0.97      0.9
```

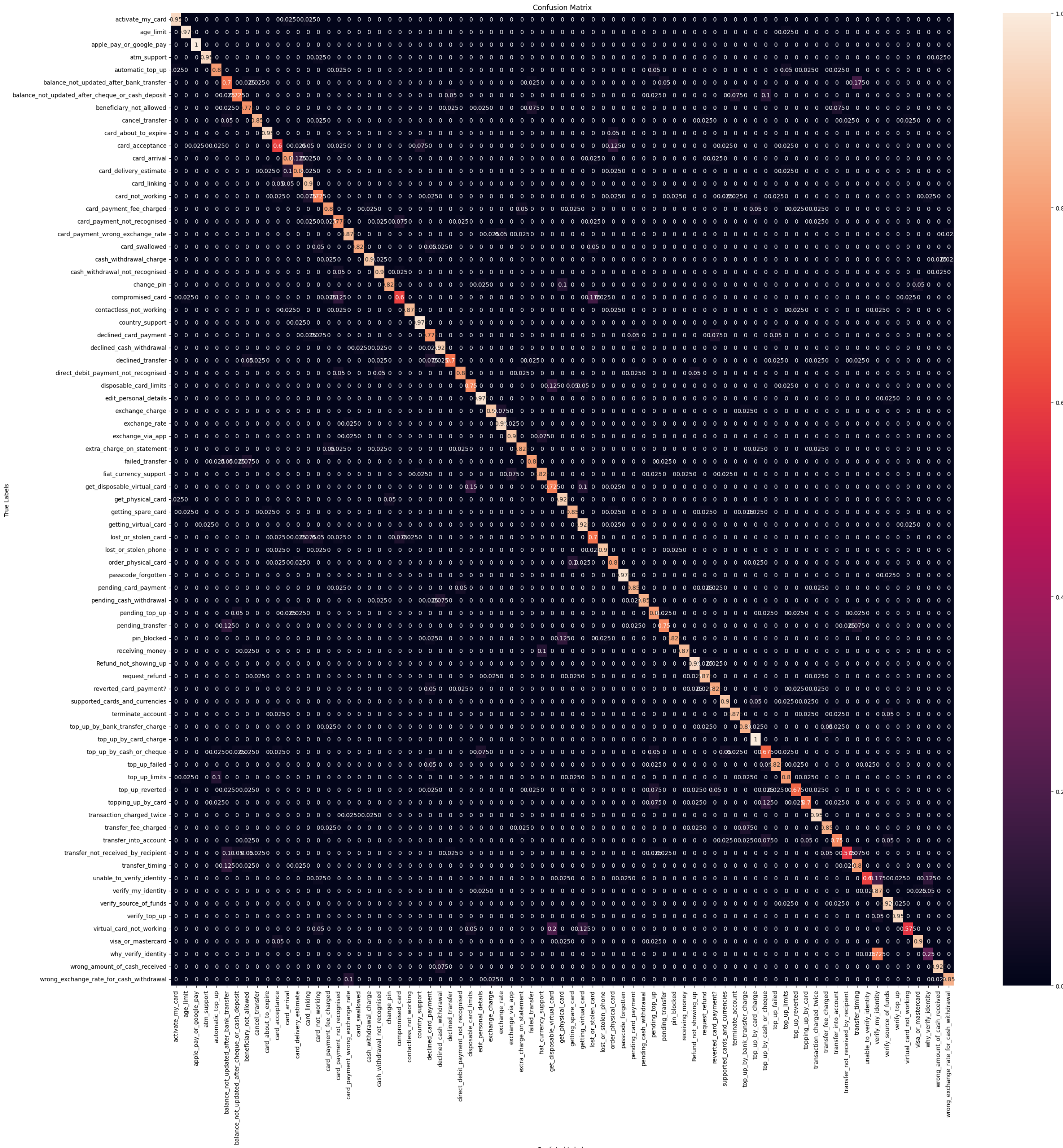
```
              exchange_charge       0.92      0.90      0.9
                exchange_rate       0.88      0.95      0.9
             exchange_via_app       0.90      0.90      0.9
    extra_charge_on_statement       0.87      0.82      0.8
              failed_transfer       0.84      0.80      0.8
        fiat_currency_support       0.82      0.82      0.8
get_disposable_virtual_card         0.69      0.72      0.7
            get_physical_card       0.77      0.93      0.8
           getting_spare_card       0.79      0.85      0.8
         getting_virtual_card       0.76      0.93      0.8
          lost_or_stolen_card       0.70      0.70      0.7
         lost_or_stolen_phone       0.95      0.90      0.9
          order_physical_card       0.71      0.80      0.7
           passcode_forgotten       0.97      0.97      0.9
         pending_card_payment       0.87      0.85      0.8
      pending_cash_withdrawal       1.00      0.85      0.9
               pending_top_up       0.68      0.80      0.7
             pending_transfer       0.86      0.75      0.8
                  pin_blocked       0.94      0.82      0.8
              receiving_money       0.97      0.88      0.9
        Refund_not_showing_up       0.84      0.95      0.8
               request_refund       0.85      0.88      0.8
      reverted_card_payment?        0.80      0.82      0.8
supported_cards_and_currencies      0.86      0.90      0.9
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
# Predicting the probabilities
predictions = best_model.predict(padded_test)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(labels_test_one_hot, axis=1)
# Compute the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)
# Optional: Normalize the confusion matrix by the number of samples in each class
#cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(30, 30))  # You can adjust the size to fit your needs
sns.heatmap(cm_normalized, annot=True, fmt='g',  xticklabels=label_names, yticklа
plt.title('Confusion Matrix')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
plt.xticks(rotation=90)
plt.yticks(rotation=0)  # Keeping the labels horizontal
```

```
plt.show()
```

```
97/97 [==============================] - 6s 60ms/step
```



```
Start coding or generate with AI.
```

Hyperparameter Tuning

```
pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 2.5 MB/s eta 0:0(
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packag(
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa(
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pacl
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pytl
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1(
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1(
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

```
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from keras_tuner import RandomSearch

# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withpunctuations_notlemmatized(s
tokenized_sentences_test = preprocess_sentences_withpunctuations_notlemmatized(se
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

def build_model(hp):
    model = tf.keras.Sequential([
```

```
        Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embeddi
        LSTM(hp.Int('units_lstm1', min_value=32, max_value=128, step=32), return_
        Dropout(hp.Float('dropout_1', min_value=0.2, max_value=0.5, step=0.1)),
        LSTM(hp.Int('units_lstm2', min_value=32, max_value=128, step=32)),
        Dropout(hp.Float('dropout_2', min_value=0.2, max_value=0.5, step=0.1)),
        Dense(hp.Int('units_dense', min_value=32, max_value=128, step=32), activa
        Dropout(hp.Float('dropout_3', min_value=0.2, max_value=0.5, step=0.1)),
        Dense(num_classes, activation='softmax')
    ])

    # Use a learning rate schedule
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=hp.Float('initial_lr', min_value=1e-4, max_value=1e
        decay_steps=10000,
        decay_rate=0.9,
        staircase=True)

    model.compile(optimizer=Adam(learning_rate=lr_schedule),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

```
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,  # Set a small number for demonstration
    executions_per_trial=1,
    directory='my_dir',
    project_name='lstm_tuning'
)

# Early stopping to avoid overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,

# Start the search process
tuner.search(padded_train, labels_train_one_hot, epochs=30, validation_data=(padd
```

```
Trial 10 Complete [00h 13m 24s]
val_accuracy: 0.8123376369476318

Best val_accuracy So Far: 0.8451298475265503
Total elapsed time: 03h 16m 27s
```

```
import numpy as np
from tensorflow.keras.models import load_model
from keras_tuner import RandomSearch

# Assuming 'tuner' is your Keras Tuner object after running the tuning
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Best Model Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Best Model Test Loss: {test_loss:.4f}")

# Optionally, you can print the best hyperparameters
print("Best hyperparameters:")
for param, value in best_hyperparameters.values.items():
    print(f"{param}: {value}")
```

```
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
WARNING:tensorflow:Value in checkpoint could not be found in the restored obje
97/97 [==============================] - 11s 69ms/step - loss: 0.6739 - accura
Best Model Test Accuracy: 84.51%
Best Model Test Loss: 0.6739
Best hyperparameters:
units_lstm1: 128
dropout_1: 0.30000000000000004
units_lstm2: 64
dropout_2: 0.30000000000000004
units_dense: 128
dropout_3: 0.30000000000000004
initial_lr: 0.003982162357170975
```

```
tuner.results_summary()
```

```
Results summary
Results in my_dir/lstm_tuning
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 05 summary
Hyperparameters:
units_lstm1: 128
dropout_1: 0.30000000000000004
units_lstm2: 64
```

```
dropout_2: 0.30000000000000004
units_dense: 128
dropout_3: 0.30000000000000004
initial_lr: 0.003982162357170975
Score: 0.8451298475265503

Trial 02 summary
Hyperparameters:
units_lstm1: 96
dropout_1: 0.2
units_lstm2: 128
dropout_2: 0.2
units_dense: 96
dropout_3: 0.30000000000000004
initial_lr: 0.00694090003577775
Score: 0.8431817889213562

Trial 07 summary
Hyperparameters:
units_lstm1: 64
dropout_1: 0.30000000000000004
units_lstm2: 32
dropout_2: 0.30000000000000004
units_dense: 96
dropout_3: 0.30000000000000004
initial_lr: 0.0009756992661327715
Score: 0.8415584564208984

Trial 08 summary
Hyperparameters:
units_lstm1: 96
dropout_1: 0.30000000000000004
units_lstm2: 96
dropout_2: 0.30000000000000004
units_dense: 32
dropout_3: 0.30000000000000004
initial_lr: 0.0013369436103508122
Score: 0.8363636136054993

Trial 00 summary
Hyperparameters:
units_lstm1: 128
dropout_1: 0.30000000000000004
units_lstm2: 128
dropout_2: 0.2
units_dense: 128
dropout_3: 0.4
initial_lr: 0.000697200142912107
Score: 0.8350649476051331
```

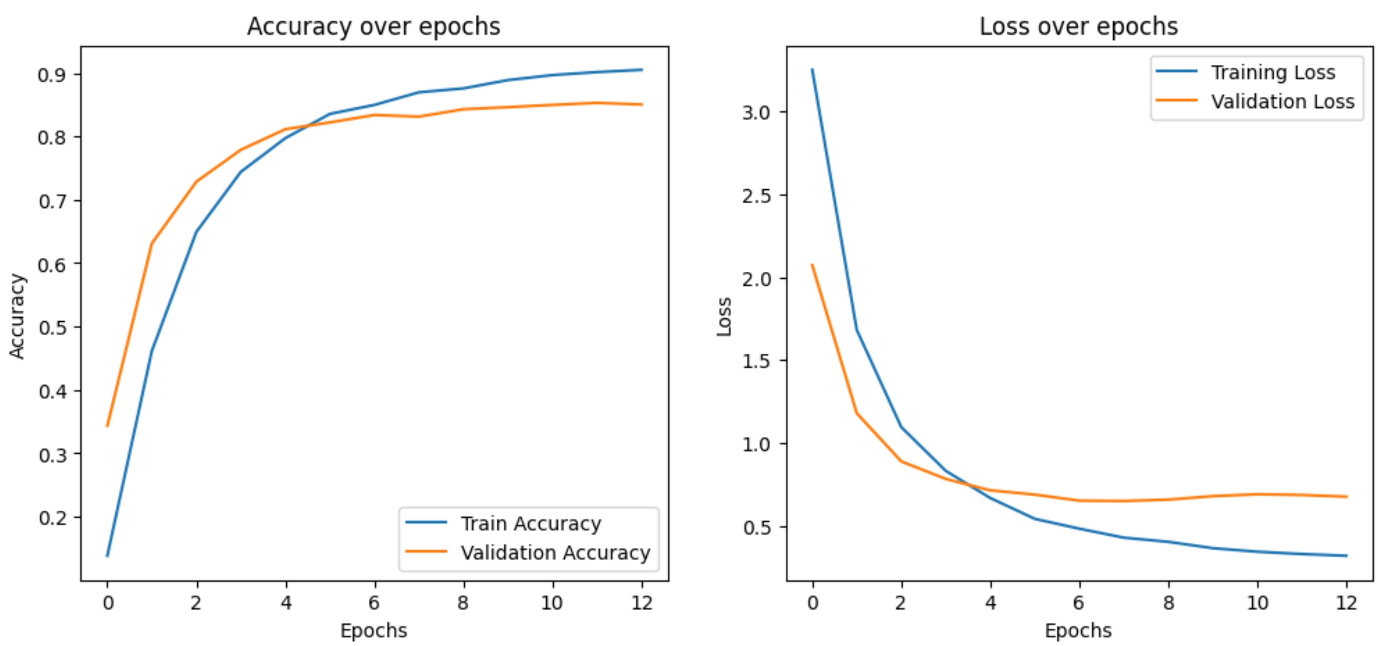
Model with updated Hyperparameters

```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withpunctuations_notlemmatized(s
tokenized_sentences_test = preprocess_sentences_withpunctuations_notlemmatized(se
word_index, word2vec_model = create_word_index_and_train_word2vec(tokenized_sente

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix(word_index, word2vec_model, 100)

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_m
    LSTM(128, return_sequences=True),  # More units and returning sequences for s
    Dropout(0.3),
    LSTM(64),  # Additional LSTM layer
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.004
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)
```

```
optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model4.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model4.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Optionally, visualize training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 89s 255ms/step - loss: 3.2525 - acc
Epoch 2/30
```

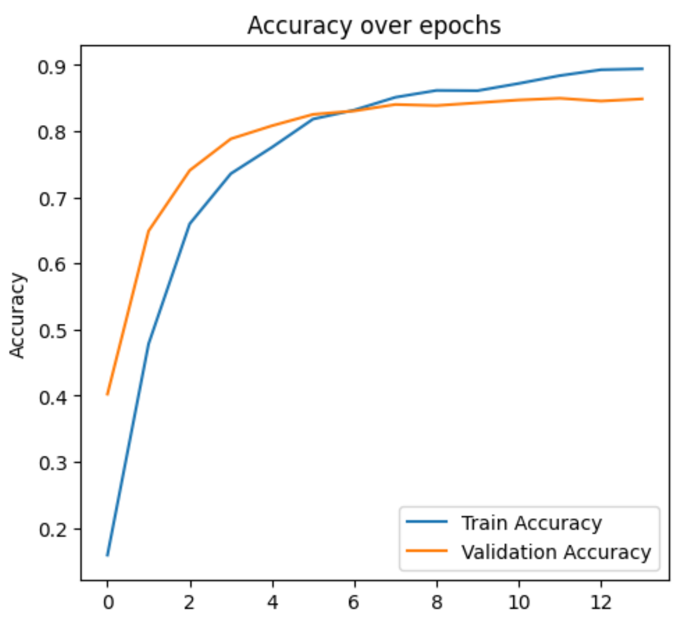
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: Use
  saving_api.save_model(
313/313 [==============================] - 74s 238ms/step - loss: 1.6825 - acc
Epoch 3/30
313/313 [==============================] - 74s 236ms/step - loss: 1.0957 - acc
Epoch 4/30
313/313 [==============================] - 93s 299ms/step - loss: 0.8313 - acc
Epoch 5/30
313/313 [==============================] - 78s 248ms/step - loss: 0.6674 - acc
Epoch 6/30
313/313 [==============================] - 79s 254ms/step - loss: 0.5416 - acc
Epoch 7/30
313/313 [==============================] - 74s 238ms/step - loss: 0.4822 - acc
Epoch 8/30
313/313 [==============================] - 82s 263ms/step - loss: 0.4281 - acc
Epoch 9/30
313/313 [==============================] - 78s 248ms/step - loss: 0.4029 - acc
Epoch 10/30
313/313 [==============================] - 75s 239ms/step - loss: 0.3645 - acc
Epoch 11/30
313/313 [==============================] - 73s 232ms/step - loss: 0.3432 - acc
Epoch 12/30
313/313 [==============================] - 74s 235ms/step - loss: 0.3291 - acc
Epoch 13/30
313/313 [==============================] - 73s 232ms/step - loss: 0.3189 - acc
Epoch 13: early stopping
97/97 [==============================] - 5s 24ms/step - loss: 0.7981 - accurac
Test Accuracy: 82.40%
Test Loss: 0.7981
```

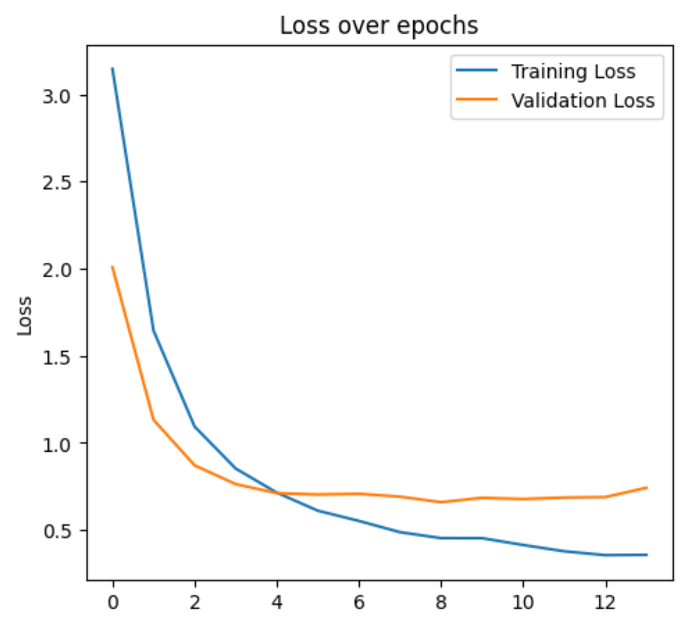


```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_dat
tokenized_sentences_train = preprocess_sentences_withcontractions(sentences_train
tokenized_sentences_test = preprocess_sentences_withcontractions(sentences_test)
word_index, word2vec_model = create_word_index_and_train_word2vec_handlingoov(tok

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix_handlingoov(word_index, word2vec_model
max_index=max(word_index.values())

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim= max_index + 1, output_dim=250, weights=[embedding_matrix
    LSTM(128, return_sequences=True),  # More units and returning sequences for s
    Dropout(0.3),
    LSTM(64),  # Additional LSTM layer
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
initial_learning_rate = 0.004
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
```

```
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model5.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model5.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")


import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 106s 311ms/step - loss: 3.1470 - ac
Epoch 2/30
313/313 [==============================] - 97s 310ms/step - loss: 1.6447 - acc
Epoch 3/30
313/313 [==============================] - 95s 304ms/step - loss: 1.0944 - acc
Epoch 4/30
313/313 [==============================] - 96s 306ms/step - loss: 0.8534 - acc
Epoch 5/30
313/313 [==============================] - 92s 295ms/step - loss: 0.7151 - acc
Epoch 6/30
313/313 [==============================] - 93s 296ms/step - loss: 0.6115 - acc
Epoch 7/30
313/313 [==============================] - 92s 295ms/step - loss: 0.5527 - acc
Epoch 8/30
313/313 [==============================] - 109s 348ms/step - loss: 0.4882 - ac
Epoch 9/30
313/313 [==============================] - 93s 296ms/step - loss: 0.4540 - acc
Epoch 10/30
313/313 [==============================] - 92s 295ms/step - loss: 0.4536 - acc
Epoch 11/30
313/313 [==============================] - 95s 305ms/step - loss: 0.4147 - acc
Epoch 12/30
313/313 [==============================] - 109s 349ms/step - loss: 0.3784 - ac
Epoch 13/30
313/313 [==============================] - 96s 308ms/step - loss: 0.3563 - acc
Epoch 14/30
313/313 [==============================] - 96s 308ms/step - loss: 0.3577 - acc
Epoch 14: early stopping
97/97 [==============================] - 9s 64ms/step - loss: 0.6601 - accurac
Test Accuracy: 83.86%
Test Loss: 0.6601
```



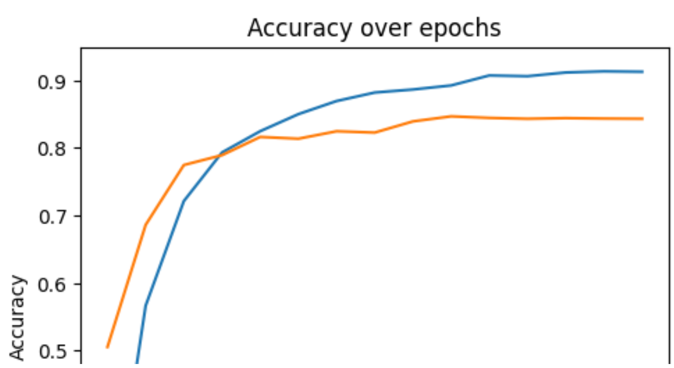


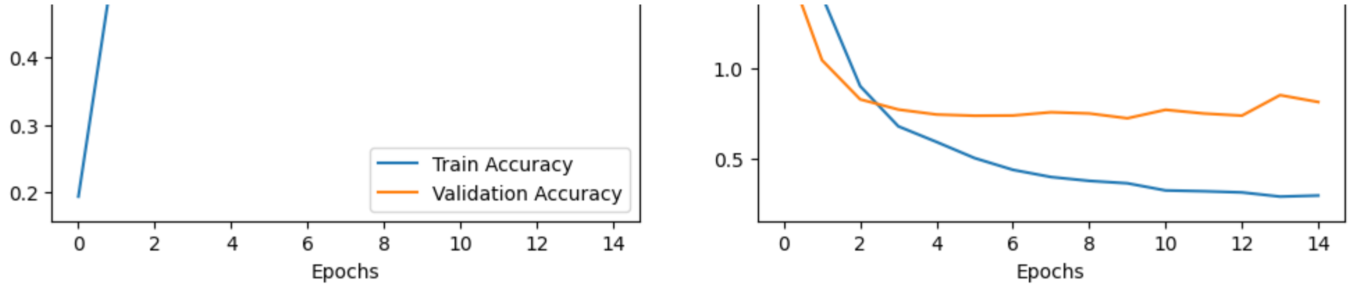
```
# Load and preprocess data
sentences_train, sentences_test, labels_train, labels_test = load_and_shuffle_data
tokenized_sentences_train = preprocess_sentences_withcontractions(sentences_train
tokenized_sentences_test = preprocess_sentences_withcontractions(sentences_test)
word_index, word2vec_model = create_word_index_and_train_word2vec_handlingoov(tok

# Convert sentences to IDs and pad sequences
id_sequences_train = convert_words_to_ids(tokenized_sentences_train, word_index)
id_sequences_test = convert_words_to_ids(tokenized_sentences_test, word_index)
padded_train = pad_sequences_to_max_length(id_sequences_train)
padded_test = pad_sequences_to_max_length(id_sequences_test)

# Create the embedding matrix
embedding_matrix = create_embedding_matrix_handlingoov(word_index, word2vec_model
max_index=max(word_index.values())

# One-hot encoding of labels
num_classes = 77
labels_train_one_hot = to_categorical(labels_train, num_classes=num_classes)
labels_test_one_hot = to_categorical(labels_test, num_classes=num_classes)

# Model definition with advanced layer configuration
model = tf.keras.Sequential([
    Embedding(input_dim= max_index + 1, output_dim=250, weights=[embedding_matrix
    Bidirectional(LSTM(128, return_sequences=True)),  # More units and returning
    Dropout(0.3),
    Bidirectional(LSTM(64)),  # Additional LSTM layer
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
], name="AdvancedLSTMModel")

# Setting up a variable learning rate
```

```
initial_learning_rate = 0.004
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.9,
    staircase=True)

optimizer = Adam(learning_rate=lr_schedule)

# Compile the model with the optimizer and learning rate schedule
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc

# Early stopping and model checkpointing
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

checkpoint = ModelCheckpoint('best_model7.h5', save_best_only=True, monitor='val_

# Train the model with early stopping and checkpointing
history = model.fit(padded_train, labels_train_one_hot, epochs=30,
                    validation_data=(padded_test, labels_test_one_hot),
                    callbacks=[early_stopping, checkpoint])

# Load the best model and evaluate its performance
best_model = tf.keras.models.load_model('best_model7.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")


import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/30
313/313 [==============================] - 207s 607ms/step - loss: 3.0021 - ac
Epoch 2/30
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: Use
  saving_api.save_model(
313/313 [==============================] - 219s 699ms/step - loss: 1.4060 - ac
Epoch 3/30
313/313 [==============================] - 193s 616ms/step - loss: 0.9017 - ac
Epoch 4/30
313/313 [==============================] - 189s 606ms/step - loss: 0.6772 - ac
Epoch 5/30
313/313 [==============================] - 181s 578ms/step - loss: 0.5901 - ac
Epoch 6/30
313/313 [==============================] - 189s 604ms/step - loss: 0.4998 - ac
Epoch 7/30
313/313 [==============================] - 188s 602ms/step - loss: 0.4345 - ac
Epoch 8/30
313/313 [==============================] - 188s 600ms/step - loss: 0.3943 - ac
Epoch 9/30
313/313 [==============================] - 189s 601ms/step - loss: 0.3728 - ac
Epoch 10/30
313/313 [==============================] - 190s 606ms/step - loss: 0.3593 - ac
Epoch 11/30
313/313 [==============================] - 190s 607ms/step - loss: 0.3193 - ac
Epoch 12/30
313/313 [==============================] - 194s 619ms/step - loss: 0.3148 - ac
Epoch 13/30
313/313 [==============================] - 190s 609ms/step - loss: 0.3083 - ac
Epoch 14/30
313/313 [==============================] - 187s 599ms/step - loss: 0.2850 - ac
Epoch 15/30
313/313 [==============================] - 187s 599ms/step - loss: 0.2909 - ac
Epoch 15: early stopping
97/97 [==============================] - 19s 133ms/step - loss: 0.7222 - accur
Test Accuracy: 84.68%
Test Loss: 0.7222
```






```
from sklearn.metrics import classification_report
best_model = tf.keras.models.load_model('best_model7.h5')
test_loss, test_accuracy = best_model.evaluate(padded_test, labels_test_one_hot)

# Assuming ds_info is available and has a feature descriptor for labels
label_names = ds_info.features['label'].names
# Predicting the probabilities
predictions = best_model.predict(padded_test)

# Converting probabilities to class indices
predicted_classes = np.argmax(predictions, axis=1)

# Extract true classes from one-hot encoded test labels
true_classes = np.argmax(labels_test_one_hot, axis=1)

# Generate the classification report
report = classification_report(true_classes, predicted_classes, target_names=labe

# Print the classification report
print(report)
```

```
97/97 [==============================] - 18s 133ms/step - loss: 0.7222 - accu
97/97 [==============================] - 18s 133ms/step
                                             precision    recall  f1-sco

                            activate_my_card       0.93      0.95      0.
                                   age_limit       0.95      0.93      0.
                     apple_pay_or_google_pay       0.98      1.00      0.
                                 atm_support       0.91      0.97      0.
                            automatic_top_up       0.92      0.82      0.
       balance_not_updated_after_bank_transfer       0.67      0.82      0.
balance_not_updated_after_cheque_or_cash_deposit       0.94      0.85      0.
                     beneficiary_not_allowed       0.86      0.75      0.
                             cancel_transfer       0.89      0.85      0.
```

card_about_to_expire	0.97	0.90	0.9
card_acceptance	0.93	0.70	0.8
card_arrival	0.84	0.90	0.8
card_delivery_estimate	0.71	0.75	0.7
card_linking	0.95	0.93	0.9
card_not_working	0.80	0.88	0.8
card_payment_fee_charged	0.80	0.80	0.8
card_payment_not_recognised	0.76	0.85	0.8
card_payment_wrong_exchange_rate	0.89	0.85	0.8
card_swallowed	0.96	0.68	0.7
cash_withdrawal_charge	0.88	0.90	0.8
cash_withdrawal_not_recognised	0.70	0.88	0.7
change_pin	0.90	0.90	0.9
compromised_card	0.77	0.60	0.6
contactless_not_working	1.00	0.80	0.8
country_support	0.90	0.93	0.9
declined_card_payment	0.74	0.93	0.8
declined_cash_withdrawal	0.80	0.93	0.8
declined_transfer	0.80	0.80	0.8
direct_debit_payment_not_recognised	0.89	0.80	0.8
disposable_card_limits	0.84	0.78	0.8
edit_personal_details	0.95	0.90	0.9
exchange_charge	0.95	0.88	0.9
exchange_rate	0.84	0.93	0.8
exchange_via_app	0.84	0.90	0.8
extra_charge_on_statement	0.81	0.88	0.8
failed_transfer	0.77	0.82	0.8
fiat_currency_support	0.90	0.70	0.7
get_disposable_virtual_card	0.85	0.82	0.8
get_physical_card	0.83	1.00	0.9
getting_spare_card	0.80	0.90	0.8
getting_virtual_card	0.97	0.95	0.9
lost_or_stolen_card	0.85	0.70	0.7
lost_or_stolen_phone	1.00	0.90	0.9
order_physical_card	0.79	0.78	0.7
passcode_forgotten	1.00	0.95	0.9
pending_card_payment	0.85	0.88	0.8
pending_cash_withdrawal	0.86	0.93	0.8
pending_top_up	0.67	0.88	0.7
pending_transfer	0.86	0.78	0.8
pin_blocked	0.97	0.75	0.8
receiving_money	0.84	0.93	0.8
Refund_not_showing_up	0.93	0.93	0.9
request_refund	0.90	0.88	0.8
reverted_card_payment?	0.87	0.82	0.8
supported_cards_and_currencies	0.88	0.88	0.8

```
text1 = ["Someone grabbed my card and ran away,now I dont have access to the card
predictclass(text1)
```

```
1/1 [==============================] - 0s 184ms/step
'lost_or_stolen_card'
```

```
def predictclass(text):

  new_sequences = convert_words_to_ids(preprocess_sentences_withcontractions(text
  new_padded = pad_sequences(new_sequences, maxlen=142, padding='post')
  predicted_class = np.argmax(best_model.predict(new_padded))
  return (label_names[predicted_class])
```