DATA LOADING

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import nltk
import re
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Dense, Conv1D, MaxPooling1D, LSTM, Bidirectional,
                                     Embedding, Dropout, SpatialDropout1D, GlobalMaxPooling
from tensorflow.keras.regularizers import l2
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

data = pd.read_csv('/content/customer_service_sentiment - meaningful_customer_service_senti
```

DATA PROCESSING

```
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)

# preprocessing
data['processed_text'] = data['customer_message'].apply(preprocess_text)

#MAPPING sentiments
sentiment_mapping = {'Frustrated': 0, 'Neutral': 1, 'Satisfied': 2}
data['sentiment_label'] = data['sentiment_label'].map(sentiment_mapping)
```

TEXT EMBEDDINGS

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Bag of Words (BoW)
bow_vectorizer = CountVectorizer(max_features=5000)
X_train_bow = bow_vectorizer.fit_transform(train_data['processed_text']).toarray()
X_val_bow = bow_vectorizer.transform(val_data['processed_text']).toarray()
X_test_bow = bow_vectorizer.transform(test_data['processed_text']).toarray()

# TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(train_data['processed_text']).toarray()
X_val_tfidf = tfidf_vectorizer.transform(val_data['processed_text']).toarray()
X_test_tfidf = tfidf_vectorizer.transform(test_data['processed_text']).toarray()

print("Embedding completed for BoW and TF-IDF.")
```

MODEL TRAINING WITH DIFFERENT ARCHITECTURES

COMPARATIVE ANALYSIS OF MODELS

```
cnn_model.save('best_cnn_model.h5')
print("Best CNN model saved to best_cnn_model.h5")
```

MODEL TRAINING AND COMAPRATIVE ANALYSIS

```
dropout_rate = 0.65
learning_rate = 0.0002
l2_factor = 1e-4
batch_size = 32

# ==============================
# CNN Model
# ==============================
cnn_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128),
    SpatialDropout1D(0.5),  # Slightly reduced dropout on embeddings
    Conv1D(filters=32, kernel_size=5, activation='relu', kernel_regularizer=l2(l2_factor)),
    tf.keras.layers.BatchNormalization(),
    GlobalMaxPooling1D(),
    Dropout(dropout_rate),
    Dense(16, activation='relu', kernel_regularizer=l2(l2_factor)),
    tf.keras.layers.BatchNormalization(),
    Dropout(dropout_rate),
    Dense(3, activation='softmax')
])
```

```
cnn_model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate), me


# ================================
# LSTM Model
# ================================
lstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128),
    SpatialDropout1D(0.8),
    LSTM(32, return_sequences=True, dropout=dropout_rate, recurrent_dropout=0.2, kernel_reg
    tf.keras.layers.BatchNormalization(),
    LSTM(16, dropout=dropout_rate, kernel_regularizer=l2(1e-3)),
    tf.keras.layers.BatchNormalization(),
    Dense(16, activation='relu', kernel_regularizer=l2(1e-3)),
    Dropout(dropout_rate),
    Dense(3, activation='softmax')
])
lstm_model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate), m

# ================================
# Revised BiLSTM Model
# ================================
bilstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128),
    SpatialDropout1D(0.8),
    Bidirectional(LSTM(32, return_sequences=True, dropout=dropout_rate, recurrent_dropout=0
    tf.keras.layers.BatchNormalization(),
    Bidirectional(LSTM(16, dropout=dropout_rate, kernel_regularizer=l2(1e-3))),
    tf.keras.layers.BatchNormalization(),
    Dense(16, activation='relu', kernel_regularizer=l2(1e-3)),
    Dropout(dropout_rate),
    Dense(3, activation='softmax')
])
bilstm_model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate),

# ============================
# Train Models with individual epoch values
# ============================
print("Training CNN Model...")
history_cnn = cnn_model.fit(X_train, y_train,
                            validation_data=(X_val, y_val),
                            epochs=epochs_cnn,
                            batch_size=batch_size,
                            callbacks=[early_stop, reduce_lr],
                            verbose=1)

print("Training LSTM Model...")
history_lstm = lstm_model.fit(X_train, y_train,
                              validation_data=(X_val, y_val),
                              epochs=epochs_lstm,
                              batch_size=batch_size,
                              callbacks=[early_stop, reduce_lr],
                              verbose=1)

print("Training BiLSTM Model...")
history_bilstm = bilstm_model.fit(X_train, y_train,
                                  validation_data=(X_val, y_val),
                                  epochs=epochs_bilstm,
                                  batch_size=batch_size,
                                  callbacks=[early_stop, reduce_lr],
```

```
                                        verbose=1)


def evaluate_model(model, X, y, name):
    loss, acc = model.evaluate(X, y, verbose=0)
    print(f"{name} Model Accuracy on test data: {acc:.4f}")

print("\nEvaluating models on test data:")
evaluate_model(cnn_model, X_test, y_test, "CNN")
evaluate_model(lstm_model, X_test, y_test, "LSTM")
evaluate_model(bilstm_model, X_test, y_test, "BiLSTM")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Training CNN Model...
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.3305 - loss: 1.7905 -
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3878 - loss: 1.5052 -
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4350 - loss: 1.3354 -
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4882 - loss: 1.1513 -
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5072 - loss: 1.0413 -
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5554 - loss: 1.0045 -
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5755 - loss: 0.8875 -
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6047 - loss: 0.8518 -
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6658 - loss: 0.7693 -
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6685 - loss: 0.7221 -
Restoring model weights from the end of the best epoch: 10.
Training LSTM Model...
Epoch 1/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 31s 247ms/step - accuracy: 0.3481 - loss: 1.718
Epoch 2/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 249ms/step - accuracy: 0.3558 - loss: 1.621
Epoch 3/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 39s 233ms/step - accuracy: 0.3630 - loss: 1.488
Epoch 4/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 42s 240ms/step - accuracy: 0.3559 - loss: 1.443
Epoch 5/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 42s 250ms/step - accuracy: 0.3593 - loss: 1.431
Epoch 6/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 247ms/step - accuracy: 0.3753 - loss: 1.373
Epoch 7/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 40s 233ms/step - accuracy: 0.3962 - loss: 1.307
Epoch 8/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 236ms/step - accuracy: 0.3928 - loss: 1.295
Epoch 9/12
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 250ms/step - accuracy: 0.3990 - loss: 1.282(
Epoch 10/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 25s 248ms/step - accuracy: 0.4164 - loss: 1.2619
Epoch 11/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 249ms/step - accuracy: 0.4261 - loss: 1.2356
Epoch 12/12
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 252ms/step - accuracy: 0.4695 - loss: 1.2003
Restoring model weights from the end of the best epoch: 12.
Training BiLSTM Model...
Epoch 1/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 54s 433ms/step - accuracy: 0.3302 - loss: 2.0759
Epoch 2/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 42s 424ms/step - accuracy: 0.3318 - loss: 1.9273
Epoch 3/15
```

SAVING THE BEST MODEL

```
cnn_model.save('best_cnn_model.h5')
print("Best CNN model saved to best_cnn_model.h5")
```

```
#  predict sentiment for new text data
def predict_sentiment(text, model, tokenizer, max_length=100):
    # Preprocess the text using your existing function
    processed_text = preprocess_text(text)

    # Convert text to sequence and pad it
    sequence = tokenizer.texts_to_sequences([processed_text])
    padded_sequence = pad_sequences(sequence, maxlen=max_length)


    prediction = model.predict(padded_sequence)
    predicted_class = np.argmax(prediction, axis=1)[0]

    #mapping numeric prediction back to sentiment label
    sentiment_mapping = {0: 'Frustrated', 1: 'Neutral', 2: 'Satisfied'}
    predicted_sentiment = sentiment_mapping[predicted_class]

    return predicted_sentiment, prediction

:
new_text = "I am really happy with your service, everything was excellent."
sentiment, probabilities = predict_sentiment(new_text, cnn_model, tokenizer, max_length)
print("Predicted sentiment:", sentiment)
print("Prediction probabilities:", probabilities)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
Predicted sentiment: Satisfied
Prediction probabilities: [[0.24302487 0.2500682  0.50690687]]
```