NANDHA ENGINEERING COLLEGE, AUTONOMOUS, ERODE -52
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT -2

ACADEMIC YEAR: 2024-2025

Register No : **22cs085 , 22cs086**

Name : **SANTHOSH. S , SATHISHKUMAR. M**

COURSE CODE & NAME : **22CSX01 & DEEP LEARNING**

CLASS / SEM : **III- B.E(CSE) / V**

TEAM – 14

Topics	Marks
How would you adapt a CNN, typically used for images, to handle text-based sentiment analysis for social media posts?	

Student signature **FacultySignature**

ASSIGNMENT -2

How would you adapt a CNN, typically used for images, to handle text-based sentiment analysis for social media posts?

CODE:

```
import pandas as pd
import numpy as np
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Step 1: Load the dataset
data = pd.read_csv('/kaggle/input/tweets/Tweets.csv')  # Adjust this path accordingly

# Preprocess the data
data['text'] = data['text'].str.replace(r'http\S+', '', regex=True)  # Remove URLs
```

```
data['text'] = data['text'].str.replace(r'[^a-zA-Z\s]', '', regex=True)  # Remove special characters

label_encoder = LabelEncoder()
data['sentiment'] = label_encoder.fit_transform(data['airline_sentiment'])

# Tokenize and pad the text data
max_sequence_length = 100
vocab_size = 10000
embedding_dim = 100

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(data['text'])
X = tokenizer.texts_to_sequences(data['text'])
X = pad_sequences(X, maxlen=max_sequence_length)
y = data['sentiment'].values

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the improved CNN-RNN model
```

```
model = models.Sequential()

# 1. Embedding layer to learn word representations
model.add(layers.Embedding(input_dim=vocab_size,
output_dim=embedding_dim))

# 2. Convolutional layer to capture local patterns
model.add(layers.Conv1D(filters=128, kernel_size=5, activation='relu'))
model.add(layers.MaxPooling1D(pool_size=2))
model.add(layers.Dropout(0.3))  # Dropout for regularization

# 3. LSTM or GRU layer to capture sequence information
model.add(layers.Bidirectional(layers.LSTM(64,
return_sequences=True)))
model.add(layers.Dropout(0.3))

# 4. Additional Convolutional Layer (optional for deeper feature
extraction)
model.add(layers.Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(layers.GlobalMaxPooling1D())

# 5. Dense layer with L2 regularization
```

```
model.add(layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))

model.add(layers.Dropout(0.3))


# 6. Output layer with softmax activation for classification

model.add(layers.Dense(3, activation='softmax'))


# Compile the model

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])


# Set up early stopping to avoid overfitting

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)


# Train the model

history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test), callbacks=[early_stopping])


# Evaluate the model

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy:.4f}")
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Example new tweets
new_tweets = [
    "I love this airline! Great service and friendly staff.",
    "The flight was delayed and I missed my connection.",
    "It was an average experience, nothing special.",
```

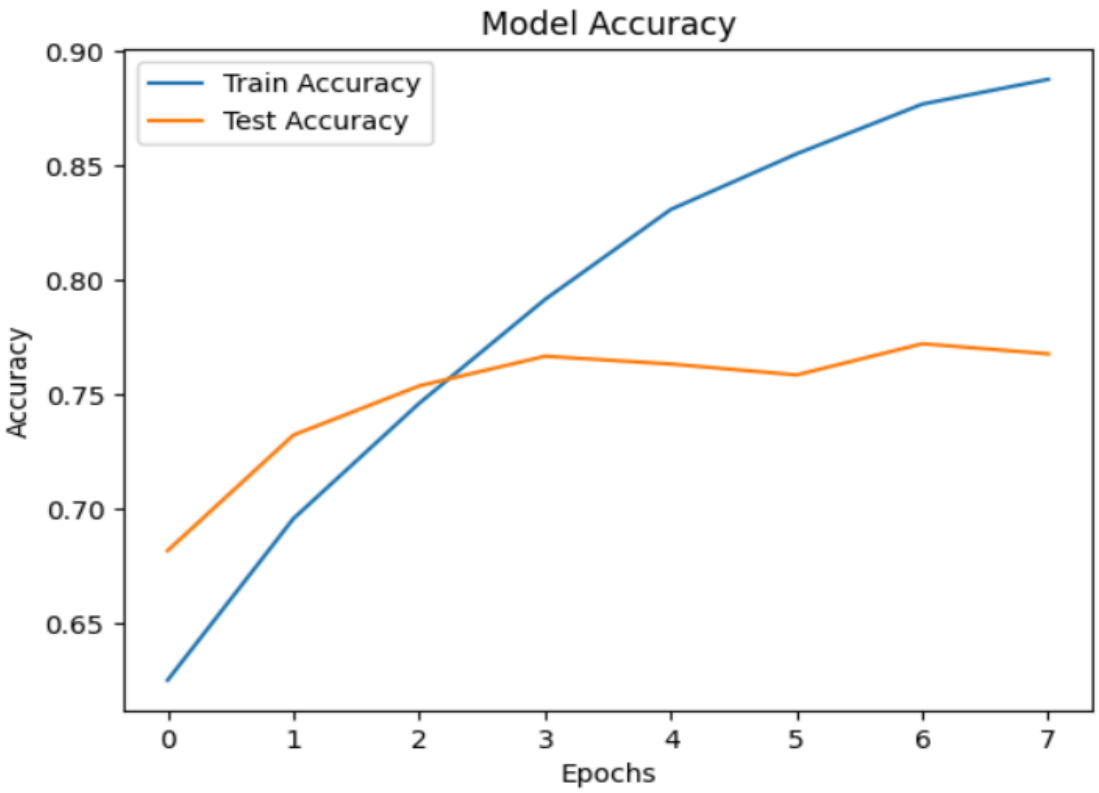
```
    "i am sad because i lost my bag",
    "Hurray, we won the UCL cup"
]

# Step 1: Preprocess the new tweets (remove URLs and special characters)
new_tweets = [tweet.replace(r'http\S+', '').replace(r'[^a-zA-Z\s]', '') for tweet in new_tweets]

# Step 2: Tokenize and pad the new tweets
new_sequences = tokenizer.texts_to_sequences(new_tweets)
new_padded = pad_sequences(new_sequences, maxlen=max_sequence_length)

# Step 3: Make predictions
predictions = model.predict(new_padded)

# Step 4: Interpret the predictions
predicted_labels = np.argmax(predictions, axis=1)
predicted_sentiments = label_encoder.inverse_transform(predicted_labels)

# Display the results
```

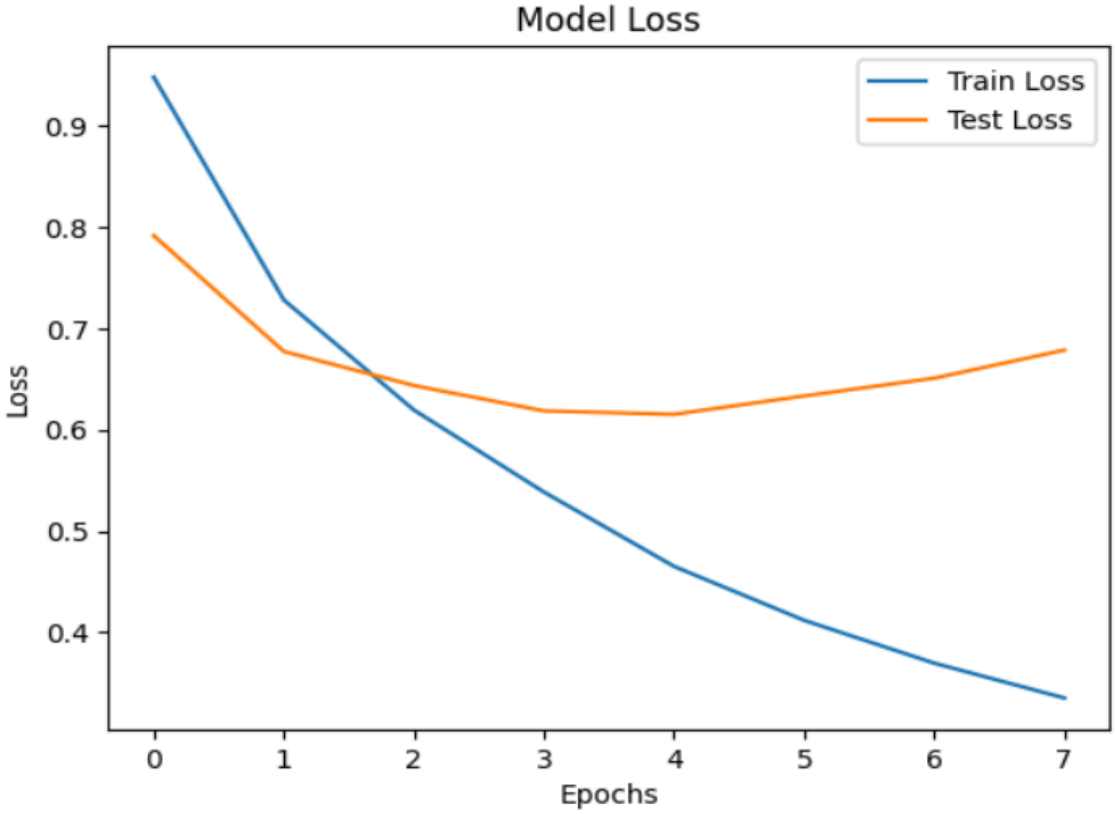
```
for tweet, sentiment in zip(new_tweets, predicted_sentiments):
    print(f"Tweet: {tweet}")
    print(f"Predicted Sentiment: {sentiment}")
    print()
```

OUTPUT:

```
Epoch 1/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 29s 63ms/step - accuracy: 0.6211 - loss: 1.0059 - val_accuracy: 0.6817 - val_loss: 0.7914
Epoch 2/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 23s 63ms/step - accuracy: 0.6808 - loss: 0.7563 - val_accuracy: 0.7322 - val_loss: 0.6772
Epoch 3/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 23s 62ms/step - accuracy: 0.7406 - loss: 0.6207 - val_accuracy: 0.7538 - val_loss: 0.6435
Epoch 4/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 22s 61ms/step - accuracy: 0.7860 - loss: 0.5413 - val_accuracy: 0.7667 - val_loss: 0.6185
Epoch 5/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 24s 65ms/step - accuracy: 0.8308 - loss: 0.4654 - val_accuracy: 0.7633 - val_loss: 0.6151
Epoch 6/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 40s 62ms/step - accuracy: 0.8590 - loss: 0.4038 - val_accuracy: 0.7585 - val_loss: 0.6332
Epoch 7/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 23s 63ms/step - accuracy: 0.8791 - loss: 0.3666 - val_accuracy: 0.7722 - val_loss: 0.6508
Epoch 8/20
366/366 ━━━━━━━━━━━━━━━━━━━━ 41s 62ms/step - accuracy: 0.8918 - loss: 0.3310 - val_accuracy: 0.7678 - val_loss: 0.6785
92/92 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.7581 - loss: 0.6256
Test Accuracy: 0.7633
```

```
Tweet: I love this airline! Great service and friendly staff.
Predicted Sentiment: positive

Tweet: The flight was delayed and I missed my connection.
Predicted Sentiment: negative

Tweet: It was an average experience, nothing special.
Predicted Sentiment: negative

Tweet: i am sad because i lost my bag
Predicted Sentiment: negative

Tweet: Hurray, we won the UCL cup
Predicted Sentiment: positive
```


Model Accuracy
Train Accuracy
Test Accuracy
Accuracy
Epochs
0.90
0.85
0.80
0.75
0.70
0.65
0
1
2
3
4
5
6
7


Model Loss
Train Loss
Test Loss
Loss
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0
1
2
3
4
5
6
7

1. Libraries and Modules:

- **TensorFlow and Keras**:
    - tensorflow and tensorflow.keras: TensorFlow is the main framework, and Keras is its high-level API for building and training models.
    - Sequential: Used to create a linear stack of layers for the neural network.
    - Embedding, LSTM, Bidirectional, Dense, Dropout: Keras layers used for building the model.

- **Preprocessing**:
    - Tokenizer (from tensorflow.keras.preprocessing.text): Tokenizes and vectorizes text data into sequences.
    - pad_sequences (from tensorflow.keras.preprocessing.sequence): Pads sequences to ensure they have uniform length for model compatibility.
- **Data**:
    - imdb (from tensorflow.keras.datasets): Dataset loader for the IMDb movie reviews dataset, used for binary sentiment classification (positive or negative).
- **NumPy**:
    - numpy: Used for numerical operations (though not explicitly used in your code, it's imported and could be useful for data manipulation).

2. Dataset and Preprocessing:

- **IMDb Dataset**:
    - imdb.load_data: Loads the IMDb dataset with reviews as sequences of integers (word indices).
- **Padding**:
    - pad_sequences: Pads sequences of varying lengths to the same length (max_len), allowing the model to process them uniformly.

3. Model Architecture:

- **Embedding Layer**:
    - Embedding(input_dim=max_words, output_dim=128, input_length=max_len): Embeds integer sequences into dense vector representations (word embeddings).
- **Bidirectional LSTM Layer**:
    - Bidirectional(LSTM(64, return_sequences=False)): A bidirectional LSTM layer that processes input sequences in both forward and backward directions, capturing more context.
- **Dropout Layer**:
    - Dropout(0.5): Reduces overfitting by randomly setting a fraction of input units to zero during training.
- **Dense Layer**:
    - Dense(1, activation='sigmoid'): Fully connected layer with sigmoid activation, used for binary classification (outputs probability).

4. Model Compilation and Training:

- **Compilation**:
    - model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']): Compiles the model with binary cross-entropy loss (appropriate for binary classification) and Adam optimizer.
- **Training**:
    - model.fit: Trains the model on the training data with the specified number of epochs and batch size, validating on test data.

5. Evaluation:

- model.evaluate: Evaluates the trained model's performance on the test data.

6. Prediction on New Reviews:

- **New Data Preprocessing**:
    - Tokenizer: Creates a new tokenizer instance to tokenize new reviews.
    - tokenizer.texts_to_sequences: Converts the tokenized new reviews to sequences of integers.
    - pad_sequences: Pads these sequences to match the input length of the model.
- **Prediction**:
    - model.predict: Generates probability predictions for each new review.
- **Output Interpretation**:

- A loop interprets the predictions as "Positive" or "Negative" based on a threshold of 0.5 and displays the predicted sentiment along with probability.

Advantages of Using a Bidirectional RNN for Sentiment Analysis

1. Enhanced Context Understanding:
    - A Bidirectional RNN captures both preceding and following contexts for each word in the sequence. For sentiment analysis, this is valuable because a word's sentiment can depend on its surrounding words. For instance, in the sentence "I didn't love the product," the word "love" appears positive in isolation, but the preceding word "didn't" changes its sentiment to negative. A Bidirectional RNN captures this nuance by processing information in both directions.
2. Improved Performance on Long Sentences:
    - Sentiment analysis often involves customer reviews that may contain long sentences or multiple clauses. In such cases, the sentiment of a sentence can depend on information from both the beginning and the end of the sentence. A Bidirectional RNN is better equipped to handle these long dependencies as it can process information in both directions simultaneously, allowing it to recognize the overall sentiment more accurately.
3. Better Handling of Ambiguous Sentiments:

- Many customer reviews contain ambiguous or mixed sentiments where context is essential to determine the intended emotion. For example, in “The product is not bad,” the word “bad” typically indicates negative sentiment, but the presence of “not” reverses the sentiment. A Bidirectional RNN can capture these subtleties, whereas a unidirectional RNN might misinterpret the sentiment if it encounters the word "bad" before "not."

4. Improved Accuracy in Sentiment Analysis Tasks:
    - Due to the model’s ability to capture a fuller context of each word, Bidirectional RNNs generally achieve higher accuracy in sentiment analysis compared to unidirectional RNNs. Studies have shown that incorporating bidirectional layers can improve a model's performance on various NLP tasks, including sentiment analysis, by making better use of the available data.
5. Enhanced Generalization to Complex Sentence Structures:
    - Customer reviews often contain complex sentence structures, including conditional clauses, negations, and expressions that require an understanding of context from both ends of the sentence. A Bidirectional RNN can better generalize to these complex structures, making it more effective at capturing sentiment in diverse review formats.