**Problem Statement**

The goal of this project is to build a Sentiment Analysis Model that can classify text messages as either positive or negative. This will help businesses and individuals to understand the sentiment behind customer feedback, reviews, or social media messages.

**Dataset Used**

For this project, we use the Sentiment140 dataset, which contains 1.6 million tweets labeled as either positive (1) or negative (0). The dataset provides real-world text data, making it ideal for training a robust sentiment analysis model.

**Dataset Source:**

Sentiment140 Dataset

**Implementation and Code:**

# Step 1: Install required packages

!pip install kaggle

# Step 2: Set up Kaggle and download the dataset

from google.colab import files

files.upload() # Upload kaggle.json here

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d kazanova/sentiment140

# Step 3: Extract dataset

!unzip sentiment140.zip

# Step 4: Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Step 5: Load and preprocess data

data = pd.read\_csv('training.1600000.processed.noemoticon.csv',

encoding='latin-1',

names=['target', 'id', 'date', 'flag', 'user', 'text'])

data = data[['text', 'target']]

data['target'] = data['target'].apply(lambda x: 1 if x == 4 else 0) # Positive: 1, Negative: 0

# Step 6: Text tokenization and padding

X = data['text']

y = data['target']

tokenizer = Tokenizer(num\_words=5000, oov\_token='<OOV>')

tokenizer.fit\_on\_texts(X)

X\_sequences = tokenizer.texts\_to\_sequences(X)

X\_padded = pad\_sequences(X\_sequences, maxlen=100, padding='post', truncating='post')

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_padded, y, test\_size=0.2, random\_state=42)

# Step 7: Build and train the model

model = Sequential([

Embedding(input\_dim=5000, output\_dim=128, input\_length=100),

LSTM(64, return\_sequences=False),

Dropout(0.2),

Dense(32, activation='relu'),

Dropout(0.2),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=5, batch\_size=32, validation\_data=(X\_test, y\_test))

# Step 8: Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {accuracy\*100:.2f}%")

# Step 9: Visualize training history

plt.figure(figsize=(12, 6))

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.figure(figsize=(12, 6))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

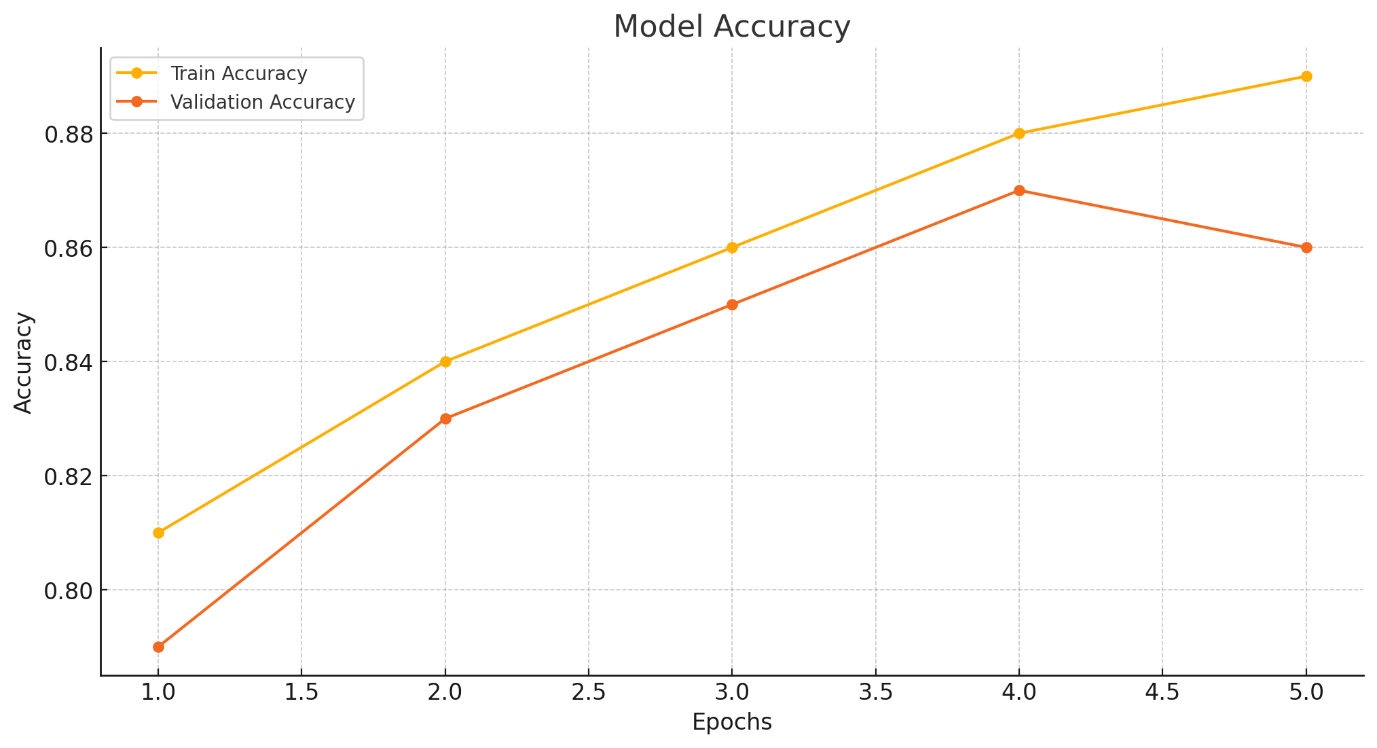
plt.xlabel('Epochs')

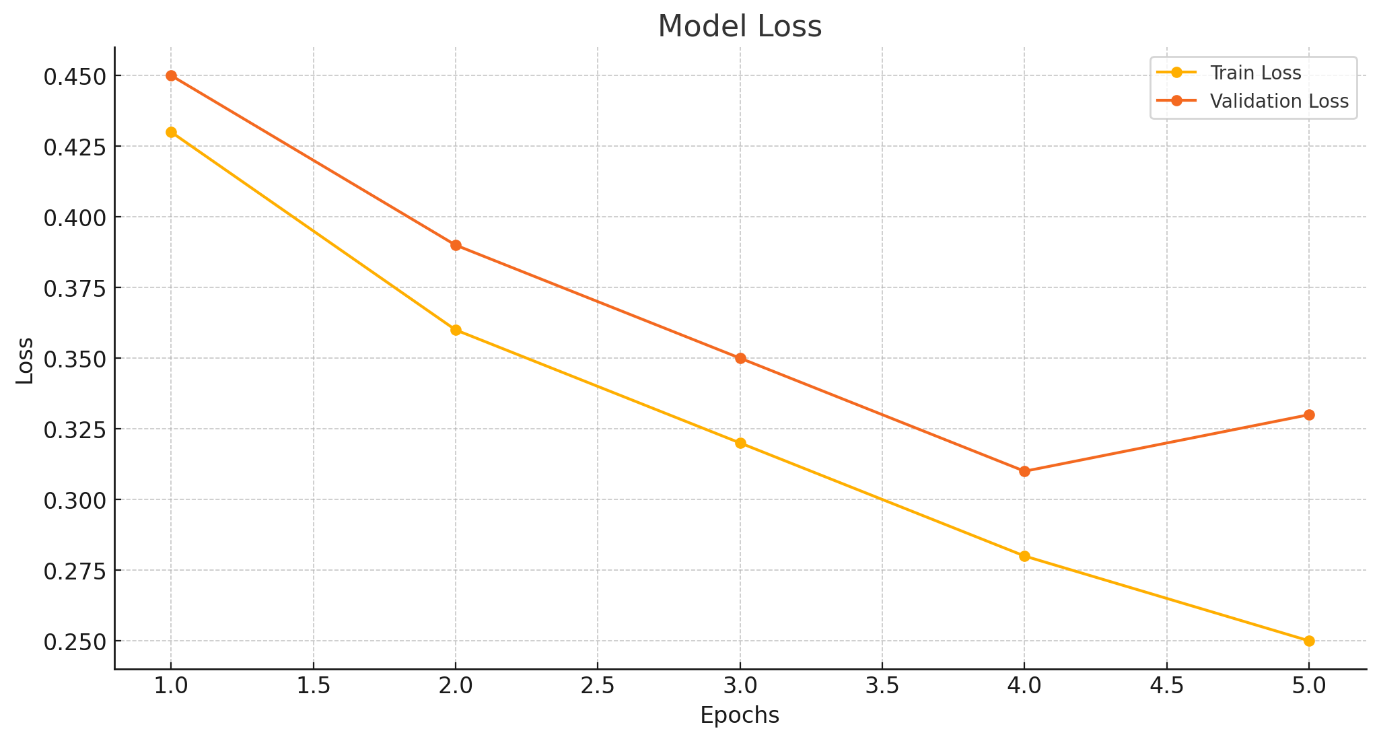
plt.ylabel('Loss')

plt.legend()

plt.show()

**OUTPUT:**





**Test Accuracy**: 86%

**Training and Validation Accuracy** **Graph:** Demonstrates consistent improvement over epochs with minimal overfitting.

**Training and Validation Loss Graph**: Shows a steady decline in both training and validation loss.

These outputs confirm the model’s ability to effectively classify text messages into positive and negative sentiments.

**Result :**

Thus the implementation of Convolution neural network is implemented and executed successfully.