```
from google.colab import files
import io
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
import numpy as np
```

```
# Step 1: Upload Dataset
uploaded = files.upload()
file_name = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[file_name]))

# Display first few rows
print(df.head())
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving IMDB Dataset - IMDB Dataset.csv to IMDB Dataset - IMDB Dataset.csv
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive
```

```
max_words = 10000
max_len = 500
```

```
df['sentiment_encoded'] = df['sentiment'].map({'positive': 1, 'negative': 0})
```

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['review'])
sequences = tokenizer.texts_to_sequences(df['review'])
data = pad_sequences(sequences, maxlen=max_len)
labels = np.array(df['sentiment_encoded'])
```

```
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

```
model = Sequential()
model.add(Embedding(max_words, 128, input_length=max_len))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just
  warnings.warn(
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train, epochs=15, batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 54s 104ms/step - accuracy: 0.6978 - loss: 0.5196 - val_accuracy: 0.8823 - val_loss: 0.2766
Epoch 2/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 84s 108ms/step - accuracy: 0.9668 - loss: 0.1007 - val_accuracy: 0.8627 - val_loss: 0.3906
Epoch 3/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 78s 100ms/step - accuracy: 0.9971 - loss: 0.0128 - val_accuracy: 0.8695 - val_loss: 0.5335
Epoch 4/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 50s 101ms/step - accuracy: 0.9998 - loss: 0.0019 - val_accuracy: 0.8692 - val_loss: 0.6231
Epoch 5/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 82s 102ms/step - accuracy: 1.0000 - loss: 1.5994e-04 - val_accuracy: 0.8709 - val_loss: 0.6629
Epoch 6/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 50s 99ms/step - accuracy: 1.0000 - loss: 5.4014e-05 - val_accuracy: 0.8720 - val_loss: 0.6883
Epoch 7/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 83s 100ms/step - accuracy: 1.0000 - loss: 3.2248e-05 - val_accuracy: 0.8717 - val_loss: 0.7104
Epoch 8/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 84s 104ms/step - accuracy: 1.0000 - loss: 2.0553e-05 - val_accuracy: 0.8724 - val_loss: 0.7287
Epoch 9/15
```

```
500/500 ━━━━━━━━━━━━━━━━━━━━ 53s 105ms/step - accuracy: 1.0000 - loss: 1.4938e-05 - val_accuracy: 0.8724 - val_loss: 0.7473
Epoch 10/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 80s 101ms/step - accuracy: 1.0000 - loss: 9.5247e-06 - val_accuracy: 0.8727 - val_loss: 0.7642
Epoch 11/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 81s 99ms/step - accuracy: 1.0000 - loss: 7.6442e-06 - val_accuracy: 0.8729 - val_loss: 0.7812
Epoch 12/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 83s 102ms/step - accuracy: 1.0000 - loss: 5.5654e-06 - val_accuracy: 0.8730 - val_loss: 0.7970
Epoch 13/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 53s 106ms/step - accuracy: 1.0000 - loss: 4.1142e-06 - val_accuracy: 0.8731 - val_loss: 0.8133
Epoch 14/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 80s 102ms/step - accuracy: 1.0000 - loss: 2.9965e-06 - val_accuracy: 0.8733 - val_loss: 0.8297
Epoch 15/15
500/500 ━━━━━━━━━━━━━━━━━━━━ 83s 103ms/step - accuracy: 1.0000 - loss: 2.2291e-06 - val_accuracy: 0.8733 - val_loss: 0.8453
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {accuracy}")
print(f"Test loss: {loss}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.8767 - loss: 0.7598
Test accuracy: 0.8791999816894531
Test loss: 0.7705792188644409
```

```
predictions = model.predict(x_test)
predictions = np.where(predictions > 0.5, 1, 0)
decoded_predictions = np.where(predictions == 1, 'positive', 'negative')
print(decoded_predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
[['positive']
 ['positive']
 ['negative']
 ...
 ['positive']
 ['negative']
 ['positive']]
```

```
def predict_review(new_review):
  new_review_sequence = tokenizer.texts_to_sequences([new_review])
  new_review_padded = pad_sequences(new_review_sequence, maxlen=max_len)
  prediction = model.predict(new_review_padded)
  prediction_binary = np.where(prediction > 0.5, 1, 0)
  decoded_prediction = np.where(prediction_binary == 1, 'positive', 'negative')
  return decoded_prediction[0][0]  # Return the string prediction
```

```
user_review = input("Enter a review: ")
predicted_sentiment = predict_review(user_review)
print(f"Predicted sentiment: {predicted_sentiment}")
```

```
Enter a review: The movie was a very awesome. The choise of actor was also very good
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Predicted sentiment: positive
```

```
```