

# **TITLE: TEXT CLASSIFICATION WITH TENSORFLOW**

## **Abstract**

This document presents a machine learning approach for sentiment analysis using the IMDB movie reviews dataset. The task involves classifying movie reviews as positive or negative based on their text content. We utilize a Recurrent Neural Network (RNN) architecture with Long Short-Term Memory (LSTM) units for this purpose. The model employs an embedding layer to convert words into dense vectors, followed by LSTM layers to capture temporal dependencies in the sequences. The model is trained and evaluated on the IMDB dataset, demonstrating effective sentiment classification performance.

## **Objective**

The primary objective of this study is to build and evaluate a sentiment analysis model using the IMDB dataset. The specific goals include:

1. Preprocessing and preparing text data for model training.
2. Building a Sequential model with embedding and LSTM layers for sentiment analysis.
3. Training the model on the IMDB dataset and evaluating its performance.

## Introduction

Sentiment analysis is a critical application in natural language processing (NLP) that aims to determine the sentiment conveyed in text data. In this study, we focus on classifying movie reviews from the IMDB dataset into positive or negative sentiments. The IMDB dataset is a widely used benchmark dataset for sentiment analysis tasks. Our approach leverages the power of Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, to capture the sequential nature of text data and make accurate sentiment predictions.

## Methodology

### 1. Data Loading and Preparation:

- The IMDB dataset is loaded with a focus on the top 10,000 most frequent words.
- The dataset is split into training and test sets.
- Sequences of text data are padded to ensure uniform length for model input.

### 2. Model Architecture:

- **Embedding Layer:** Converts integer-encoded words into dense vectors of fixed size.
- **LSTM Layer:** Captures temporal dependencies and patterns in the sequences.
- **Dense Layer:** Produces the final classification output using a sigmoid activation function.

### 3. Model Training:

- The model is compiled with binary cross-entropy loss and the Adam optimizer.
- It is trained for 15 epochs with a batch size of 32, using both training and validation data.

### 4. Evaluation:

- The model's performance is assessed on the test set using accuracy and loss metrics.

## Code

```
import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.models import Sequential

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


# Set parameters

max_features = 10000

maxlen = 500

batch_size = 32


# Load the IMDB dataset

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)


# Pad sequences to a fixed length

x_train = sequence.pad_sequences(x_train, maxlen=maxlen)

x_test = sequence.pad_sequences(x_test, maxlen=maxlen)


# Build the model

model = Sequential()

model.add(Embedding(max_features, 128))

model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
```

```
model.add(Dense(1, activation='sigmoid'))
```

```
# Compile the model
```

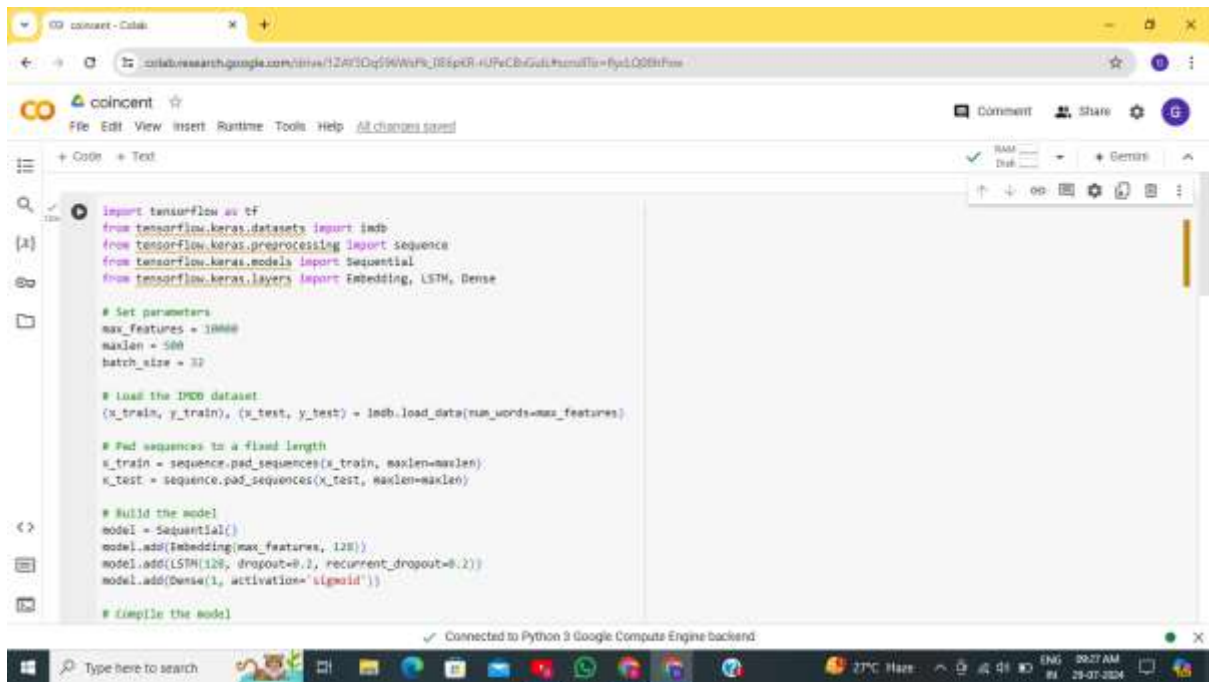
```
model.compile(loss='binary_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

```
# Train the model
```

```
model.fit(x_train, y_train,  
          batch_size=batch_size,  
          epochs=15, # Adjust as needed  
          validation_data=(x_test, y_test))
```

```
# Evaluate the model
```

```
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)  
print('Test score:', score)  
print('Test accuracy:', acc)
```



```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

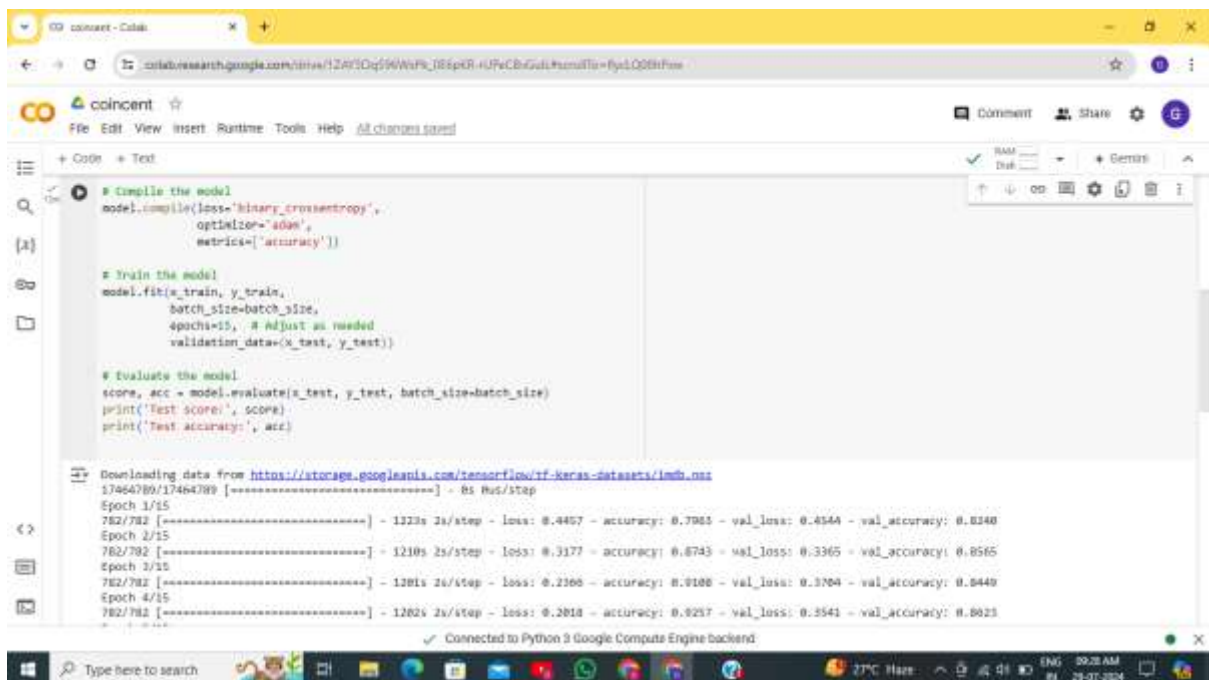
# Set parameters
max_features = 10000
maxlen = 500
batch_size = 32

# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to a fixed length
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Build the model
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
```



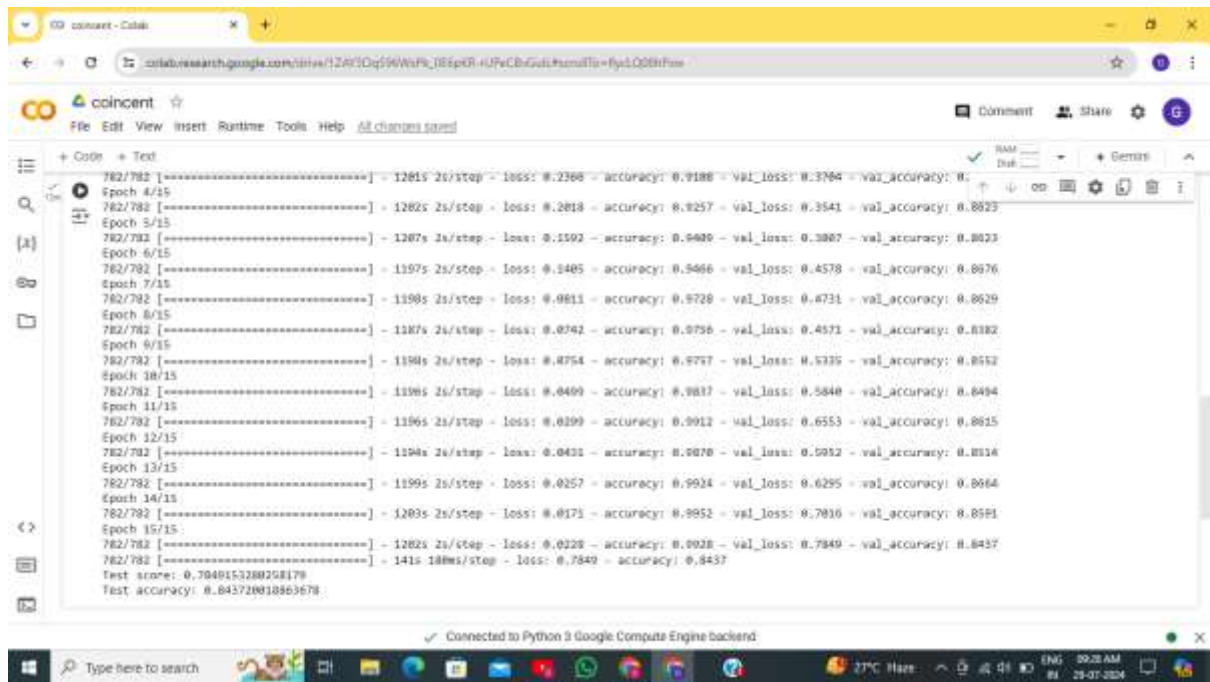
```
# Compile the model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=15, # Adjust as needed
        validation_data=(x_test, y_test))

# Evaluate the model
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Test score:', score)
print('Test accuracy:', acc)
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz>  
17464780/17464789 [=====] - 85 B/s/step

Epoch	Step	Loss	Accuracy	Val Loss	Val Accuracy
1	1223s	0.4457	0.7965	0.4544	0.8240
2	1210s	0.3177	0.8743	0.3365	0.8565
3	1201s	0.2366	0.9108	0.3704	0.8449
4	1202s	0.2818	0.9257	0.3541	0.8623



```
782/782 [=====] - 1281s 2s/step - loss: 0.2386 - accuracy: 0.9188 - val_loss: 0.3764 - val_accuracy: 0.8823
Epoch 4/15
782/782 [=====] - 1282s 2s/step - loss: 0.2818 - accuracy: 0.9257 - val_loss: 0.3541 - val_accuracy: 0.8823
Epoch 5/15
782/782 [=====] - 1287s 2s/step - loss: 0.1592 - accuracy: 0.9489 - val_loss: 0.3887 - val_accuracy: 0.8823
Epoch 6/15
782/782 [=====] - 1197s 2s/step - loss: 0.1485 - accuracy: 0.9466 - val_loss: 0.4578 - val_accuracy: 0.8876
Epoch 7/15
782/782 [=====] - 1198s 2s/step - loss: 0.0811 - accuracy: 0.9728 - val_loss: 0.4731 - val_accuracy: 0.8829
Epoch 8/15
782/782 [=====] - 1187s 2s/step - loss: 0.0742 - accuracy: 0.9756 - val_loss: 0.4571 - val_accuracy: 0.8882
Epoch 9/15
782/782 [=====] - 1198s 2s/step - loss: 0.0754 - accuracy: 0.9777 - val_loss: 0.5335 - val_accuracy: 0.8552
Epoch 10/15
782/782 [=====] - 1196s 2s/step - loss: 0.0409 - accuracy: 0.9817 - val_loss: 0.5840 - val_accuracy: 0.8484
Epoch 11/15
782/782 [=====] - 1186s 2s/step - loss: 0.0299 - accuracy: 0.9912 - val_loss: 0.6553 - val_accuracy: 0.8615
Epoch 12/15
782/782 [=====] - 1194s 2s/step - loss: 0.0432 - accuracy: 0.9870 - val_loss: 0.5952 - val_accuracy: 0.8818
Epoch 13/15
782/782 [=====] - 1199s 2s/step - loss: 0.0257 - accuracy: 0.9924 - val_loss: 0.6295 - val_accuracy: 0.8664
Epoch 14/15
782/782 [=====] - 1203s 2s/step - loss: 0.0171 - accuracy: 0.9952 - val_loss: 0.7016 - val_accuracy: 0.8581
Epoch 15/15
782/782 [=====] - 1262s 2s/step - loss: 0.0228 - accuracy: 0.9928 - val_loss: 0.7849 - val_accuracy: 0.8437
Test score: 0.7949153280268179
Test accuracy: 0.843718818863678
```

## Conclusion

The sentiment analysis model based on LSTM networks achieved notable results on the IMDB dataset. The embedding layer effectively converts word indices into dense vectors, while the LSTM layer captures the sequential dependencies in the text data. The model demonstrated robust performance in classifying movie reviews into positive or negative sentiments. Future work may involve experimenting with different architectures or hyperparameters to further enhance model performance.