# Image Recognition Sentiment Analysis

## Steps

1. Load the dataset
2. Clean Dataset
3. Encode Sentiments
4. Split Dataset
5. Tokenize and Pad/Truncate Reviews
6. Build Architecture/Model
7. Train and Test

```
import pandas as pd    # to load dataset

import numpy as np     # for mathematic equation

from nltk.corpus import stopwords   # to get collection of stopwords

from sklearn.model_selection import train_test_split       # for splitting
dataset

from tensorflow.keras.preprocessing.text import Tokenizer  # to encode text
 to int

from tensorflow.keras.preprocessing.sequence import pad_sequences   # to do
 padding or truncating

from tensorflow.keras.models import Sequential     # the model

from tensorflow.keras.layers import Embedding, LSTM, Dense # layers of the
architecture

from tensorflow.keras.callbacks import ModelCheckpoint   # save model

from tensorflow.keras.models import load_model   # load saved model

import re
```

<hr size=2 width="100%" align=center>

```
from google.colab import drive

drive.mount("/content/drive")
```

output *cancel*

```
Mounted at /content/drive
```

*Preview dataset*

```
data = pd.read_csv('/content/IMDB_Dataset (1).csv')

print(data)
```

output *cancel*

```
                                                  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
```

```
...                                                  ...        ...
49995  I thought this movie did a down right good job...  positive
49996  Bad plot, bad dialogue, bad acting, idiotic di...  negative
49997  I am a Catholic taught in parochial elementary...  negative
49998  I'm going to have to disagree with the previou...  negative
49999  No one expects the Star Trek movies to be high...  negative

[50000 rows x 2 columns]
```

## Load and Clean Dataset

In the original dataset, the reviews are still dirty. There are still html tags, numbers, uppercase, and punctuations. This will not be good for training, so in load_dataset() function, beside loading the dataset using pandas, I also pre-process the reviews by removing html tags, non alphabet (punctuations and numbers), stop words, and lower case all of the reviews.

## Encode Sentiments

In the same function, I also encode the sentiments into integers (0 and 1). Where 0 is for negative sentiments and 1 is for positive sentiments.

```
def load_dataset():

    df = pd.read_csv('/content/IMDB_Dataset (1).csv')

    x_data = df['review']       # Reviews/Input

    y_data = df['sentiment']    # Sentiment/Output


    # PRE-PROCESS REVIEW
    x_data = x_data.replace({'<.*?>': ''}, regex = True)          # remove 
html tag

    x_data = x_data.replace({'[^A-Za-
z]': ' '}, regex = True)     # remove non alphabet

    x_data = x_data.apply(lambda review: [w for w in review.split() if w no
t in english_stops])  # remove stop words

    x_data = x_data.apply(lambda review: [w.lower() for w in review])   # l
ower case



    # ENCODE SENTIMENT -> 0 & 1

    y_data = y_data.replace('positive', 1)

    y_data = y_data.replace('negative', 0)



    return x_data, y_data
```

```
x_data, y_data = load_dataset()



print('Reviews')

print(x_data, '\n')

print('Sentiment')

print(y_data)
```

output *cancel*

```
Reviews
0        [one, reviewers, mentioned, watching, oz, epis...
1        [a, wonderful, little, production, the, filmin...
2        [i, thought, wonderful, way, spend, time, hot,...
3        [basically, family, little, boy, jake, thinks,...
4        [petter, mattei, love, time, money, visually, ...
                               ...
49995    [i, thought, movie, right, good, job, it, crea...
49996    [bad, plot, bad, dialogue, bad, acting, idioti...
49997    [i, catholic, taught, parochial, elementary, s...
49998    [i, going, disagree, previous, comment, side, ...
49999    [no, one, expects, star, trek, movies, high, a...
Name: review, Length: 50000, dtype: object

Sentiment
0        1
1        1
2        1
3        0
4        1
        ..
49995    1
49996    0
49997    0
49998    0
49999    0
Name: sentiment, Length: 50000, dtype: int64
```

## Split Dataset

In this work, I decided to split the data into 80% of Training and 20% of Testing set using train_test_split method from Scikit-Learn. By using this method, it automatically shuffles the dataset. We need to shuffle the data because in the original dataset, the reviews and sentiments are in order, where they list positive reviews first and then negative reviews. By shuffling the data, it will be distributed equally in the model, so it will be more accurate for predictions.

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_si
ze = 0.2)



print('Train Set')

print(x_train, '\n')
print(x_test, '\n')

print('Test Set')

print(y_train, '\n')

print(y_test)
```
output *cancel*

```
Train Set
49988    [when, i, first, tuned, morning, news, i, thou...
14805    [i, thought, maybe, film, boasted, cast, inclu...
17188    [a, weeks, ago, german, broadcaster, sat, adve...
160      [mystery, men, got, the, stupidest, film, i, e...
12029    [based, fact, story, teenager, named, homer, h...
                                ...
10025    [recipe, one, worst, movies, time, male, villa...
38146    [i, recommend, christmas, movie, worthless, pi...
28300    [to, surprise, i, really, enjoyed, disney, lat...
35034    [after, mob, boss, vic, moretti, late, great, ...
42327    [directed, jacques, tourneur, cat, people, out...
Name: review, Length: 40000, dtype: object

12941    [critics, need, review, class, quality, movie,...
34120    [computing, can, anything, boring, sitting, fr...
39933    [during, kurt, weill, celebration, brooklyn, w...
35153    [this, movie, difficult, watch, stan, looked, ...
47426    [no, redeeming, features, film, rubbish, its, ...
                                ...
26205    [yep, lots, shouting, screaming, cheering, arg...
653      [a, group, friends, break, middle, nowhere, on...
336      [elvira, mistress, dark, one, fav, movies, eve...
36961    [this, one, julie, greatest, tributes, music, ...
46654    [dear, mr, seitzman, or, whomever, i, may, hol...
Name: review, Length: 10000, dtype: object

Test Set
49988    0
14805    0
17188    0
160      1
12029    1
        ..
10025    0
38146    0
28300    1
35034    0
42327    1
Name: sentiment, Length: 40000, dtype: int64

12941    1
34120    0
39933    1
35153    0
47426    0
        ..
26205    0
653      1
336      1
36961    1
46654    0
```

```
Name: sentiment, Length: 10000, dtype: int64
```
*Function for getting the maximum review length, by calculating the mean of all the reviews length (using **numpy.mean**)*

```
def get_max_length():

    review_length = []

    for review in x_train:

        review_length.append(len(review))



    return int(np.ceil(np.mean(review_length)))
```
<hr size=2 width="100%" align=center>

## Tokenize and Pad/Truncate Reviews

A Neural Network only accepts numeric data, so we need to encode the reviews. I use tensorflow.keras.preprocessing.text.Tokenizer to encode the reviews into integers, where each unique word is automatically indexed (using fit_on_texts method) based on x_train.
x_train and x_test is converted into integers using texts_to_sequences method.

Each reviews has a different length, so we need to add padding (by adding 0) or truncating the words to the same length (in this case, it is the mean of all reviews length) using tensorflow.keras.preprocessing.sequence.pad_sequences.

post, pad or truncate the words in the back of a sentence
pre, pad or truncate the words in front of a sentence

```
# ENCODE REVIEW

token = Tokenizer(lower=False)    # no need lower, because already lowered
the data in load_data()

token.fit_on_texts(x_train)

x_train = token.texts_to_sequences(x_train)

x_test = token.texts_to_sequences(x_test)



max_length = get_max_length()



x_train = pad_sequences(x_train, maxlen=max_length, padding='post', truncat
ing='post')
```

```
x_test = pad_sequences(x_test, maxlen=max_length, padding='post', truncatin
g='post')



total_words = len(token.word_index) + 1   # add 1 because of 0 padding



print('Encoded X Train\n', x_train, '\n')

print('Encoded X Test\n', x_test, '\n')

print('Maximum review length: ', max_length)
```
output *cancel*

```
Encoded X Train
 [[  172     1    23 ...  1015    94  2042]
 [    1   104   181 ...     0     0     0]
 [   39  2227   502 ...  2131     2     3]
 ...
 [  284   741     1 ...     0     0     0]
 [  301  2732  1328 ... 10486     1   771]
 [  428  7794 15792 ...   634   248 15723]]

Encoded X Test
 [[ 1338   261   617 ...     0     0     0]
 [89930  1359   141 ...     0     0     0]
 [ 2336  4212 50546 ...     0     0     0]
 ...
 [ 4781  4119   360 ...     0     0     0]
 [    8     5  1987 ...     0     0     0]
 [ 2979   337 46655 ...  2177   227    18]]

Maximum review length:  130
```

---

---

## Build Architecture/Model

Embedding Layer: in simple terms, it creates word vectors of each word in the word_index and group words that are related or have similar meaning by analyzing other words around them.

LSTM Layer: to make a decision to keep or throw away data by considering the current input, previous output, and previous memory. There are some important components in LSTM.

- Forget Gate, decides information is to be kept or thrown away
- Input Gate, updates cell state by passing previous output and current input into sigmoid activation function
- Cell State, calculate new cell state, it is multiplied by forget vector (drop value if multiplied by a near 0), add it with the output from input gate to update the cell state value.
- Ouput Gate, decides the next hidden state and used for predictions
- Dense Layer: compute the input with the weight matrix and bias (optional), and using an activation function. I use Sigmoid activation function for this work because the output is only 0 or 1.

- The optimizer is Adam and the loss function is Binary Crossentropy because again the output is only 0 and 1, which is a binary number.

```
# ARCHITECTURE

EMBED_DIM = 32

LSTM_OUT = 64


model = Sequential()

model.add(Embedding(total_words, EMBED_DIM, input_length = max_length))

model.add(LSTM(LSTM_OUT))

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

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


print(model.summary())
```

output *cancel*

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 130, 32)           2956032   

 lstm (LSTM)                 (None, 64)                24832     

 dense (Dense)               (None, 1)                 65        

=================================================================
Total params: 2980929 (11.37 MB)
Trainable params: 2980929 (11.37 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

---

---

## Training

For training, it is simple. We only need to fit our x_train (input) and y_train (output/label) data. For this training, I use a mini-batch learning method with a batch_size of 128 and 5 epochs.Also, I added a callback called checkpoint to save the model locally for every epoch if its accuracy improved from the previous epoch.

```
checkpoint = ModelCheckpoint(

    'models/LSTM.h5',

    monitor='accuracy',

    save_best_only=True,

    verbose=1

)
```

<hr size=2 width="100%" align=center>

```
model.fit(x_train, y_train, batch_size = 128, epochs = 30, callbacks=[check
point])
```

output *cancel*

```
Epoch 1/30
313/313 [==============================] - ETA: 0s - loss: 0.4799 -
accuracy: 0.7372
Epoch 1: accuracy improved from -inf to 0.73722, saving model to
models/LSTM.h5
313/313 [==============================] - 62s 174ms/step - loss: 0.4799 -
accuracy: 0.7372
Epoch 2/30
  1/313 [..............................] - ETA: 4s - loss: 0.2247 -
accuracy: 0.9297
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the
native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
313/313 [==============================] - ETA: 0s - loss: 0.2209 -
accuracy: 0.9215
Epoch 2: accuracy improved from 0.73722 to 0.92150, saving model to
models/LSTM.h5
313/313 [==============================] - 34s 109ms/step - loss: 0.2209 -
accuracy: 0.9215
Epoch 3/30
313/313 [==============================] - ETA: 0s - loss: 0.1300 -
accuracy: 0.9602
Epoch 3: accuracy improved from 0.92150 to 0.96022, saving model to
models/LSTM.h5
313/313 [==============================] - 26s 82ms/step - loss: 0.1300 -
accuracy: 0.9602
Epoch 4/30
313/313 [==============================] - ETA: 0s - loss: 0.0834 -
accuracy: 0.9767
Epoch 4: accuracy improved from 0.96022 to 0.97672, saving model to
models/LSTM.h5
313/313 [==============================] - 18s 57ms/step - loss: 0.0834 -
accuracy: 0.9767
Epoch 5/30
313/313 [==============================] - ETA: 0s - loss: 0.0573 -
accuracy: 0.9848
Epoch 5: accuracy improved from 0.97672 to 0.98480, saving model to
models/LSTM.h5
313/313 [==============================] - 11s 34ms/step - loss: 0.0573 -
accuracy: 0.9848
```

```
Epoch 6/30
313/313 [==============================] - ETA: 0s - loss: 0.0465 -
accuracy: 0.9881
Epoch 6: accuracy improved from 0.98480 to 0.98813, saving model to
models/LSTM.h5
313/313 [==============================] - 11s 36ms/step - loss: 0.0465 -
accuracy: 0.9881
Epoch 7/30
313/313 [==============================] - ETA: 0s - loss: 0.0401 -
accuracy: 0.9898
Epoch 7: accuracy improved from 0.98813 to 0.98978, saving model to
models/LSTM.h5
313/313 [==============================] - 9s 30ms/step - loss: 0.0401 -
accuracy: 0.9898
Epoch 8/30
313/313 [==============================] - ETA: 0s - loss: 0.0391 -
accuracy: 0.9901
Epoch 8: accuracy improved from 0.98978 to 0.99013, saving model to
models/LSTM.h5
313/313 [==============================] - 8s 26ms/step - loss: 0.0391 -
accuracy: 0.9901
Epoch 9/30
313/313 [==============================] - ETA: 0s - loss: 0.0343 -
accuracy: 0.9915
Epoch 9: accuracy improved from 0.99013 to 0.99145, saving model to
models/LSTM.h5
313/313 [==============================] - 6s 20ms/step - loss: 0.0343 -
accuracy: 0.9915
Epoch 10/30
313/313 [==============================] - ETA: 0s - loss: 0.0294 -
accuracy: 0.9929
Epoch 10: accuracy improved from 0.99145 to 0.99285, saving model to
models/LSTM.h5
313/313 [==============================] - 7s 23ms/step - loss: 0.0294 -
accuracy: 0.9929
Epoch 11/30
313/313 [==============================] - ETA: 0s - loss: 0.0240 -
accuracy: 0.9944
Epoch 11: accuracy improved from 0.99285 to 0.99437, saving model to
models/LSTM.h5
313/313 [==============================] - 6s 21ms/step - loss: 0.0240 -
accuracy: 0.9944
Epoch 12/30
313/313 [==============================] - ETA: 0s - loss: 0.0256 -
accuracy: 0.9937
Epoch 12: accuracy did not improve from 0.99437
313/313 [==============================] - 6s 18ms/step - loss: 0.0256 -
accuracy: 0.9937
Epoch 13/30
313/313 [==============================] - ETA: 0s - loss: 0.0190 -
accuracy: 0.9957
Epoch 13: accuracy improved from 0.99437 to 0.99568, saving model to
models/LSTM.h5
313/313 [==============================] - 5s 15ms/step - loss: 0.0190 -
accuracy: 0.9957
Epoch 14/30
313/313 [==============================] - ETA: 0s - loss: 0.0261 -
accuracy: 0.9934
Epoch 14: accuracy did not improve from 0.99568
313/313 [==============================] - 6s 19ms/step - loss: 0.0261 -
accuracy: 0.9934
```

```
Epoch 15/30
313/313 [==============================] - ETA: 0s - loss: 0.0345 -
accuracy: 0.9920
Epoch 15: accuracy did not improve from 0.99568
313/313 [==============================] - 6s 20ms/step - loss: 0.0345 -
accuracy: 0.9920
Epoch 16/30
313/313 [==============================] - ETA: 0s - loss: 0.0362 -
accuracy: 0.9914
Epoch 16: accuracy did not improve from 0.99568
313/313 [==============================] - 4s 13ms/step - loss: 0.0362 -
accuracy: 0.9914
Epoch 17/30
313/313 [==============================] - ETA: 0s - loss: 0.0168 -
accuracy: 0.9962
Epoch 17: accuracy improved from 0.99568 to 0.99620, saving model to
models/LSTM.h5
313/313 [==============================] - 4s 12ms/step - loss: 0.0168 -
accuracy: 0.9962
Epoch 18/30
313/313 [==============================] - ETA: 0s - loss: 0.0142 -
accuracy: 0.9969
Epoch 18: accuracy improved from 0.99620 to 0.99690, saving model to
models/LSTM.h5
313/313 [==============================] - 6s 19ms/step - loss: 0.0142 -
accuracy: 0.9969
Epoch 19/30
313/313 [==============================] - ETA: 0s - loss: 0.0144 -
accuracy: 0.9966
Epoch 19: accuracy did not improve from 0.99690
313/313 [==============================] - 4s 13ms/step - loss: 0.0144 -
accuracy: 0.9966
Epoch 20/30
313/313 [==============================] - ETA: 0s - loss: 0.0118 -
accuracy: 0.9975
Epoch 20: accuracy improved from 0.99690 to 0.99748, saving model to
models/LSTM.h5
313/313 [==============================] - 5s 15ms/step - loss: 0.0118 -
accuracy: 0.9975
Epoch 21/30
313/313 [==============================] - ETA: 0s - loss: 0.0302 -
accuracy: 0.9930
Epoch 21: accuracy did not improve from 0.99748
313/313 [==============================] - 5s 15ms/step - loss: 0.0302 -
accuracy: 0.9930
Epoch 22/30
310/313 [============================>.] - ETA: 0s - loss: 0.0259 -
accuracy: 0.9943
Epoch 22: accuracy did not improve from 0.99748
313/313 [==============================] - 3s 11ms/step - loss: 0.0259 -
accuracy: 0.9943
Epoch 23/30
313/313 [==============================] - ETA: 0s - loss: 0.0106 -
accuracy: 0.9980
Epoch 23: accuracy improved from 0.99748 to 0.99797, saving model to
models/LSTM.h5
313/313 [==============================] - 3s 10ms/step - loss: 0.0106 -
accuracy: 0.9980
Epoch 24/30
313/313 [==============================] - ETA: 0s - loss: 0.0227 -
accuracy: 0.9936
```

```
Epoch 24: accuracy did not improve from 0.99797
313/313 [==============================] - 5s 16ms/step - loss: 0.0227 -
accuracy: 0.9936
Epoch 25/30
313/313 [==============================] - ETA: 0s - loss: 0.0136 -
accuracy: 0.9970
Epoch 25: accuracy did not improve from 0.99797
313/313 [==============================] - 4s 13ms/step - loss: 0.0136 -
accuracy: 0.9970
Epoch 26/30
313/313 [==============================] - ETA: 0s - loss: 0.0068 -
accuracy: 0.9989
Epoch 26: accuracy improved from 0.99797 to 0.99888, saving model to
models/LSTM.h5
313/313 [==============================] - 4s 12ms/step - loss: 0.0068 -
accuracy: 0.9989
Epoch 27/30
313/313 [==============================] - ETA: 0s - loss: 0.0106 -
accuracy: 0.9980
Epoch 27: accuracy did not improve from 0.99888
313/313 [==============================] - 5s 15ms/step - loss: 0.0106 -
accuracy: 0.9980
Epoch 28/30
313/313 [==============================] - ETA: 0s - loss: 0.0180 -
accuracy: 0.9961
Epoch 28: accuracy did not improve from 0.99888
313/313 [==============================] - 4s 13ms/step - loss: 0.0180 -
accuracy: 0.9961
Epoch 29/30
313/313 [==============================] - ETA: 0s - loss: 0.0113 -
accuracy: 0.9977
Epoch 29: accuracy did not improve from 0.99888
313/313 [==============================] - 4s 12ms/step - loss: 0.0113 -
accuracy: 0.9977
Epoch 30/30
313/313 [==============================] - ETA: 0s - loss: 0.0077 -
accuracy: 0.9982
Epoch 30: accuracy did not improve from 0.99888
313/313 [==============================] - 5s 15ms/step - loss: 0.0077 -
accuracy: 0.9982
<keras.src.callbacks.History at 0x7f2d80a73b50>
```

## Testing

To evaluate the model, we need to predict the sentiment using our x_test data and comparing the predictions with y_test (expected output) data. Then, we calculate the accuracy of the model by dividing numbers of correct prediction with the total data.

```
predictions = model.predict(x_test)

y_pred = predictions.argmax(axis=-1)

# y_pred = model.predict_classes(x_test, batch_size = 128)



true = 0
```

```
for i, y in enumerate(y_test):

    if y == y_pred[i]:

        true += 1



print('Correct Prediction: {}'.format(true))

print('Wrong Prediction: {}'.format(len(y_pred) - true))

print('Accuracy: {}'.format(true/len(y_pred)*100))
```
output *cancel*

```
313/313 [==============================] - 2s 5ms/step
Correct Prediction: 4973
Wrong Prediction: 5027
Accuracy: 49.730000000000004
```