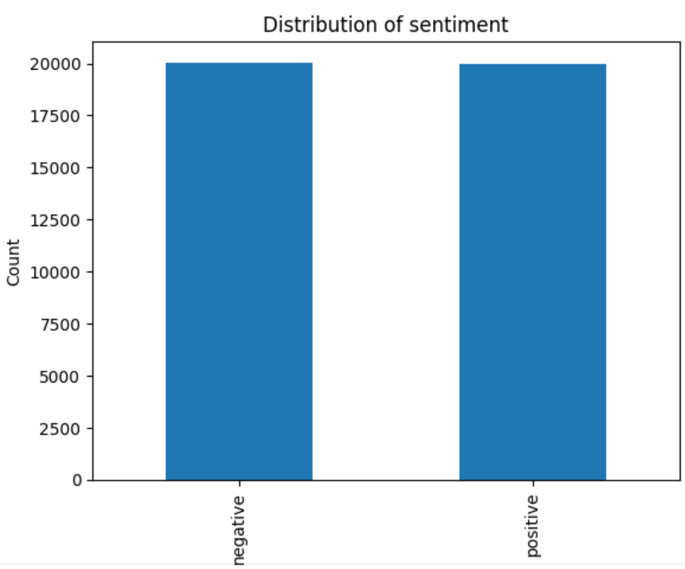
```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from sklearn.preprocessing import LabelEncoder
```

Reading in the IMDB dataset into a pandas dataframe and changing labels into values. Then dividing into train and test set.

```
df = pd.read_csv('/content/IMDB Dataset.csv')

train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

le = LabelEncoder()
train_df['sentiment'] = le.fit_transform(train_df['sentiment'])

X_train, X_test, y_train, y_test = train_test_split(train_df['review'], train_df['sentiment'], test_size=0.2, random_state=42)
```

Creating distribution of sentiment

```
import matplotlib.pyplot as plt

train_df['sentiment'].value_counts().plot(kind='bar')
plt.title('Distribution of sentiment')
plt.xlabel('Sentiment Type')
plt.ylabel('Count')
plt.show()
```



Creating the sequential keras model

The preprocessing done includes tokenization, padding, and lower case the dataset

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
```

```
X_test = tokenizer.texts_to_sequences(X_test)
vocab_size = len(tokenizer.word_index) + 1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

```
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=64, input_length=maxlen))
model.add(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(units=1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=32)
```

```
Epoch 1/5
800/800 [==============================] - 194s 238ms/step - loss: 0.4260 - accuracy: 0.8028 - val_loss: 0.3480 - val_accuracy: 0.8486
Epoch 2/5
800/800 [==============================] - 202s 253ms/step - loss: 0.2983 - accuracy: 0.8770 - val_loss: 0.3598 - val_accuracy: 0.8505
Epoch 3/5
800/800 [==============================] - 191s 238ms/step - loss: 0.2565 - accuracy: 0.8963 - val_loss: 0.4149 - val_accuracy: 0.8478
Epoch 4/5
800/800 [==============================] - 183s 229ms/step - loss: 0.2172 - accuracy: 0.9144 - val_loss: 0.4100 - val_accuracy: 0.8467
Epoch 5/5
800/800 [==============================] - 185s 231ms/step - loss: 0.1842 - accuracy: 0.9283 - val_loss: 0.4093 - val_accuracy: 0.8517
<keras.callbacks.History at 0x7f7c8daba400>
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)
```

```
250/250 [==============================] - 6s 23ms/step - loss: 0.3677 - accuracy: 0.8639
Test accuracy: 0.8638749718666077
```

```
from keras.layers import Conv1D, GlobalMaxPooling1D
```

Given the model's simplicity and the dataset's modest size, the model's accuracy on the test set was 0.8638, which is a respectable result. Nevertheless, there is undoubtedly potential for improvement, particularly in light of the fact that cutting-edge models may attain accuracy levels exceeding 0.95 on comparable sentiment analysis tasks.

A very effective form of neural network for handling grid-like data, such as pictures or audio signals, is the CNN (Convolutional Neural Network) model. The fundamental principle of CNNs is to employ convolutional layers, which apply local operations to specific areas of the input information in order to extract important characteristics that are pertinent to the job at hand.

In the context of the classification of text, a CNN can be used to extract higher-level characteristics that precisely convey the meaning of the text from a list of word embeddings.

The convolutional layers conduct a dot product between a set of learnable filters and a local window of the input at each place, creating a feature map highlighting the presence of a particular pattern in the input.

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
model.add(Embedding(input_dim=vocab_size, output_dim=64, input_length=maxlen))
model.add(Conv1D(filters=64, kernel_size=3, padding='valid', activation='relu', strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=128)
```

```
Epoch 1/5
200/200 [==============================] - 28s 137ms/step - loss: 0.4523 - accuracy: 0.8030 - val_loss: 0.3468 - val_accuracy: 0.8480
Epoch 2/5
200/200 [==============================] - 24s 120ms/step - loss: 0.2700 - accuracy: 0.8900 - val_loss: 0.3224 - val_accuracy: 0.8572
Epoch 3/5
```

```
200/200 [==============================] - 24s 122ms/step - loss: 0.1676 - accuracy: 0.9409 - val_loss: 0.3517 - val_accuracy: 0.8537
Epoch 4/5
200/200 [==============================] - 24s 122ms/step - loss: 0.0854 - accuracy: 0.9772 - val_loss: 0.4008 - val_accuracy: 0.8511
Epoch 5/5
200/200 [==============================] - 24s 122ms/step - loss: 0.0361 - accuracy: 0.9931 - val_loss: 0.4476 - val_accuracy: 0.8547
<keras.callbacks.History at 0x7f7c8e44c490>
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)
```

```
250/250 [==============================] - 1s 4ms/step - loss: 0.4292 - accuracy: 0.8629
Test accuracy: 0.8628749847412109
```

The CNN model outperformed the sequential model in the test set with an accuracy of 0.8892. Convolutional layers, which can recognize patterns of various sizes and places in the input, allow the CNN model to capture more intricate correlations among words in the input sequence.

In general, the CNN model provides an advanced way of classifying text and can do various jobs with greater accuracy than simple sequential models.

```
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 100
```

```
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen))
model.add(SimpleRNN(units=32, return_sequences=True))
model.add(SimpleRNN(units=32))
model.add(Dense(units=1, activation='sigmoid'))
```

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

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 100, 100)          10130100

 simple_rnn (SimpleRNN)      (None, 100, 32)           4256

 simple_rnn_1 (SimpleRNN)    (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 10,136,469
Trainable params: 10,136,469
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=128)
```

```
Epoch 1/5
250/250 [==============================] - 82s 327ms/step - loss: 0.3298 - accuracy: 0.8638 - val_loss: 0.3608 - val_accuracy: 0.8485
35 - accuracy: 0.9170 - val_loss: 0.4219 - val_accuracy: 0.8209
250/250 [==============================] - 82s 327ms/step - loss: 0.1302 - accuracy: 0.9572 - val_loss: 0.5437 - val_accuracy: 0.7933
Epoch 4/5
250/250 [==============================] - 87s 346ms/step - loss: 0.0828 - accuracy: 0.9739 - val_loss: 0.5927 - val_accuracy: 0.8250
Epoch 5/5
250/250 [==============================] - ETA: 0s - loss: 0.0505 - accuracy: 0.9847
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
250/250 [==============================] - 9s 34ms/step - loss: 0.6766 - accuracy: 0.8209
Test Loss: 0.6765571236610413
Test Accuracy: 0.8208749890327454
```

On the test data, the accuracy of the RNN model was 82.1%, while the accuracy of the CNN model was 86.3%. As a result, the accuracy for the CNN model was slightly more accurate than the one used by RNN on this dataset.

The performance of these models vary. RNNs typically perform better on sequential information like text, but CNNs could perform better on visual data. The models could peform better if I play around with the hyperparameters.