The data used below called clickbait.csv includes about 30,000 entries of news headlines and a binary evaluation of wether or not it is clickbait.The model should predict if a headline is clickbait or not

1 = clickbait 0 = not clickbait

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
```

```
df = pd.read_csv('clickbait_data.csv')
df = df.sample(frac=1) #Shuffle data to make splitting data simpler since its sorted pre-shuffle
df.head()
```

	headline	clickbait
15581	Tell Us About Being In Debt In Your Twenties	1
11136	Reminder That Emma Stone Sang "Bitch" On A VH1...	1
3204	Here's What Happens When You Watch Too Many "S...	1
8652	Are You Netflix Or Are You Chill	1
2508	This Is What It's Like To Have Hypochondria	1

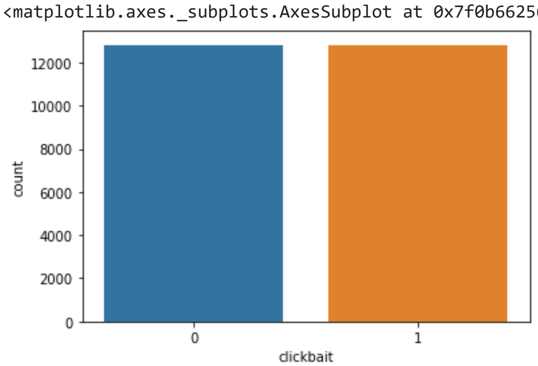
```
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
train.head()
```

```
train data size:  (25629, 2)
test data size:  (6371, 2)
```

	headline	clickbait
15581	Tell Us About Being In Debt In Your Twenties	1
11136	Reminder That Emma Stone Sang "Bitch" On A VH1...	1
3204	Here's What Happens When You Watch Too Many "S...	1
8652	Are You Netflix Or Are You Chill	1
2508	This Is What It's Like To Have Hypochondria	1

```
sns.countplot(x=train["clickbait"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b66256a90>
```



Data is evenly split between clickbait and not clicbait articles. The only target can be one of either options.

```
# set up X and Y
num_labels = 2
vocab_size = 20000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.headline)

x_train = tokenizer.texts_to_matrix(train.headline, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.headline, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.clickbait)
y_train = encoder.transform(train.clickbait)
y_test = encoder.transform(test.clickbait)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (25629, 20000) (25629,)
test shapes: (6371, 20000) (6371,)
test first five labels: [1 1 1 0 1]
```

```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

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

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=7,
                    verbose=1,
                    validation_split=0.1,
                    shuffle =True)
```

```
Epoch 1/7
231/231 [==============================] - 5s 19ms/step - loss: 0.2676 - accuracy: 0.9185 - val_loss: 0.0916 - val_accuracy: 0.9707
Epoch 2/7
231/231 [==============================] - 4s 18ms/step - loss: 0.0371 - accuracy: 0.9896 - val_loss: 0.0750 - val_accuracy: 0.9719
Epoch 3/7
231/231 [==============================] - 4s 17ms/step - loss: 0.0113 - accuracy: 0.9980 - val_loss: 0.0740 - val_accuracy: 0.9731
Epoch 4/7
231/231 [==============================] - 4s 18ms/step - loss: 0.0050 - accuracy: 0.9994 - val_loss: 0.0751 - val_accuracy: 0.9723
Epoch 5/7
231/231 [==============================] - 4s 17ms/step - loss: 0.0027 - accuracy: 0.9997 - val_loss: 0.0772 - val_accuracy: 0.9727
Epoch 6/7
231/231 [==============================] - 4s 17ms/step - loss: 0.0016 - accuracy: 0.9999 - val_loss: 0.0798 - val_accuracy: 0.9731
Epoch 7/7
231/231 [==============================] - 4s 17ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0820 - val_accuracy: 0.9727
```

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
64/64 [==============================] - 1s 8ms/step - loss: 0.0616 - accuracy: 0.9777
Accuracy:  0.9777114987373352
```

```
# get predictions so we can calculate more metrics
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
```

```
200/200 [==============================] - 1s 3ms/step
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
accuracy score:  0.9777115052582013
precision score:  0.9728971962616823
recall score:  0.9826935179358087
f1 score:  0.9777708202880401
```

Testing the model with the test data, the model had an accuracy of 97.7% qith 7 epochs. I determined that running it any further was unnecessary due to little increased gain. Additionally I needed to minimize memory waste since Colab would run out of RAM.

```
max_features = 100
maxlen = 50
batch_size = 32

modelCNN = models.Sequential()
modelCNN.add(layers.Embedding(max_features, 128, input_length=maxlen))
modelCNN.add(layers.Conv1D(32, 7, activation='relu'))
modelCNN.add(layers.MaxPooling1D(5))
modelCNN.add(layers.Conv1D(32, 7, activation='relu'))
modelCNN.add(layers.GlobalMaxPooling1D())
modelCNN.add(layers.Dense(1))


modelCNN.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use
  super(RMSprop, self).__init__(name, **kwargs)
```

```
history = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
161/161 [==============================] - 4s 28ms/step - loss: 1.3178e-04 - accuracy: 1.0000 - val_loss: 0.0482 - val_accuracy: 0.9863
Epoch 2/10
161/161 [==============================] - 6s 34ms/step - loss: 1.1571e-04 - accuracy: 1.0000 - val_loss: 0.0488 - val_accuracy: 0.9863
Epoch 3/10
161/161 [==============================] - 4s 23ms/step - loss: 1.0197e-04 - accuracy: 1.0000 - val_loss: 0.0493 - val_accuracy: 0.9863
Epoch 4/10
161/161 [==============================] - 4s 25ms/step - loss: 9.0186e-05 - accuracy: 1.0000 - val_loss: 0.0498 - val_accuracy: 0.9863
Epoch 5/10
161/161 [==============================] - 4s 25ms/step - loss: 8.0021e-05 - accuracy: 1.0000 - val_loss: 0.0502 - val_accuracy: 0.9863
Epoch 6/10
161/161 [==============================] - 4s 25ms/step - loss: 7.1188e-05 - accuracy: 1.0000 - val_loss: 0.0507 - val_accuracy: 0.9863
Epoch 7/10
161/161 [==============================] - 4s 23ms/step - loss: 6.3547e-05 - accuracy: 1.0000 - val_loss: 0.0512 - val_accuracy: 0.9863
Epoch 8/10
161/161 [==============================] - 5s 32ms/step - loss: 5.6776e-05 - accuracy: 1.0000 - val_loss: 0.0517 - val_accuracy: 0.9860
Epoch 9/10
161/161 [==============================] - 5s 30ms/step - loss: 5.0828e-05 - accuracy: 1.0000 - val_loss: 0.0521 - val_accuracy: 0.9860
Epoch 10/10
161/161 [==============================] - 5s 29ms/step - loss: 4.5593e-05 - accuracy: 1.0000 - val_loss: 0.0526 - val_accuracy: 0.9858
```

```
from sklearn.metrics import classification_report

predCNN = model.predict(x_test)
predCNN = [1.0 if p>= 0.5 else 0.0 for p in predCNN]
print(classification_report(y_test, predCNN))
```

```
200/200 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      3193
           1       0.97      0.98      0.98      3178

    accuracy                           0.98      6371
   macro avg       0.98      0.98      0.98      6371
weighted avg       0.98      0.98      0.98      6371
```

Trying a CNN architecture had slightly better the same accuracy of 98%. At this point this is about as high as it can get.

I changed the embedding features and tried it with the testing data, but I was unable to reach anything better than 98%. This involved changing the max_features, max length, batch size, number of dimensions, etc. Trying to fine tune everything did not seem to impact the accuracy much and I believe a 98% accuracy is more than adequate in predicting whether a headline is clickbait or not.

The main downside with performance was using up so much of the RAM avaialable that the model could not run twice without resetting. Regardless, it produced and effective model in predicting clickbait titles.

✓ 1s completed at 8:18 PM