Assignment 4 - NLP Disaster Tweets Kaggle Mini Project

Description

This project is a binary text classification. The goal of this project is to develop a recurring nueral network model that can accurately identify Tweets whose content relates to a real disaster from those that do not. The data is provided by the Kaggle Natural Language Processing with Disaster Tweets Competition and located at https://www.kaggle.com/c/nlp-getting-started/overview.

Natural Language Processing (NLP)

NLP is a machine learning technique that seeks to understand and make sense of the humnan language in its different forms, text, speech, etc. NLP processes a series of words, text or images and attempts to analyze the input to produce the desired output.

The challenge in NLP lies in capturing the intended meaning of the input, given that the same input can have different meanings, i.e. different outputs. The trick lies in captuing the context of the input so the the correct output can be derived.

Data Summary

The data consists of training and test data. The train.csv file contains the training data comprised of an id, keyword, location, Tweet text and ground truth labels. There are 7613 rows in the training data. The test.csv file contains the test data comprised of an id, keyword, location, Tweet text, however it does not include a label. There are 3263 rows in the test data. The sample_submission.csv contains the ids of the test Tweets and sample ground truth labels. The labeles are to be replaced with test results and submitted for assessment of the model.

In [1]:
```
#!pip install pandas emoji
#!pip install nltk
```

In [1]:
```
#Set Page Width to 100%
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [2]:
```
#Load Required Resources

import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import math
from sklearn import metrics
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.preprocessing import text, sequence
from tensorflow.keras.layers import TextVectorization
from tensorflow.keras import models
from tensorflow.python.keras.layers import LSTM, Dense, Conv2D, MaxPooling2D, Dropout,
from tensorflow.python.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Bidirectional
from keras.optimizers import Adam
from tensorflow.keras.layers import*
```

In [3]:

```python
## Import Data
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv ")

print(train_df.head(), '\n')
print(train_df.info(), '\n')
print('Train Shape: ', train_df.shape, '\n')
print(test_df.head(), '\n')
print(test_df.info())
print('Test Shape: ', test_df.shape, '\n')
```

```
   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M...
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or...
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as ...

   target
0       1
1       1
2       1
3       1
4       1

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
None

Train Shape:  (7613, 5)

   id keyword location                                               text
0   0     NaN      NaN                 Just happened a terrible car crash
1   2     NaN      NaN  Heard about #earthquake is different cities, s...
2   3     NaN      NaN  there is a forest fire at spot pond, geese are...
3   9     NaN      NaN           Apocalypse lighting. #Spokane #wildfires
4  11     NaN      NaN      Typhoon Soudelor kills 28 in China and Taiwan

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        3263 non-null   int64
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
None
Test Shape:  (3263, 4)
```

EDA

EDA will be performed as follows:

1. Remove unnecessary columns (keyword and location)
2. Check for NaNs and Nulls in remaining columns

```
3. Understand distributions of data sets
4. Cleanse test strings
    - Remove Hyperlinks
    - Remove Punctuation
    - Remove Stop Words
    - Convert to all lower case
```

Columns keyword and location are irrelevent to the analysis and therefore removed from the data sets.

Drop keyword and location

In [4]:
```
# drop key and location

train_df = train_df.drop(['keyword', 'location'], axis=1)
test_df = test_df.drop(['keyword', 'location'], axis=1)

print(train_df.head(), '\n')
print(test_df.head())
```

```
   id                                               text  target
0   1  Our Deeds are the Reason of this #earthquake M...       1
1   4             Forest fire near La Ronge Sask. Canada       1
2   5  All residents asked to 'shelter in place' are ...       1
3   6  13,000 people receive #wildfires evacuation or...       1
4   7  Just got sent this photo from Ruby #Alaska as ...       1

   id                                               text
0   0                 Just happened a terrible car crash
1   2  Heard about #earthquake is different cities, s...
2   3  there is a forest fire at spot pond, geese are...
3   9            Apocalypse lighting. #Spokane #wildfires
4  11      Typhoon Soudelor kills 28 in China and Taiwan
```

Check for Nulls

In [5]:
```
#Check for NaNs and Nulls
print('Train id NaNs / Null Count: ', train_df['id'].isna().sum(), '\n')
print('Test id NaNs / Null Count: ', train_df['id'].isna().sum(), '\n')
print('Train text NaNs / Null Count: ', train_df['text'].isna().sum(), '\n')
print('Train text NaNs / Null Count: ', train_df['text'].isna().sum(), '\n')
```

```
Train id NaNs / Null Count:  0

Test id NaNs / Null Count:  0

Train text NaNs / Null Count:  0

Train text NaNs / Null Count:  0
```

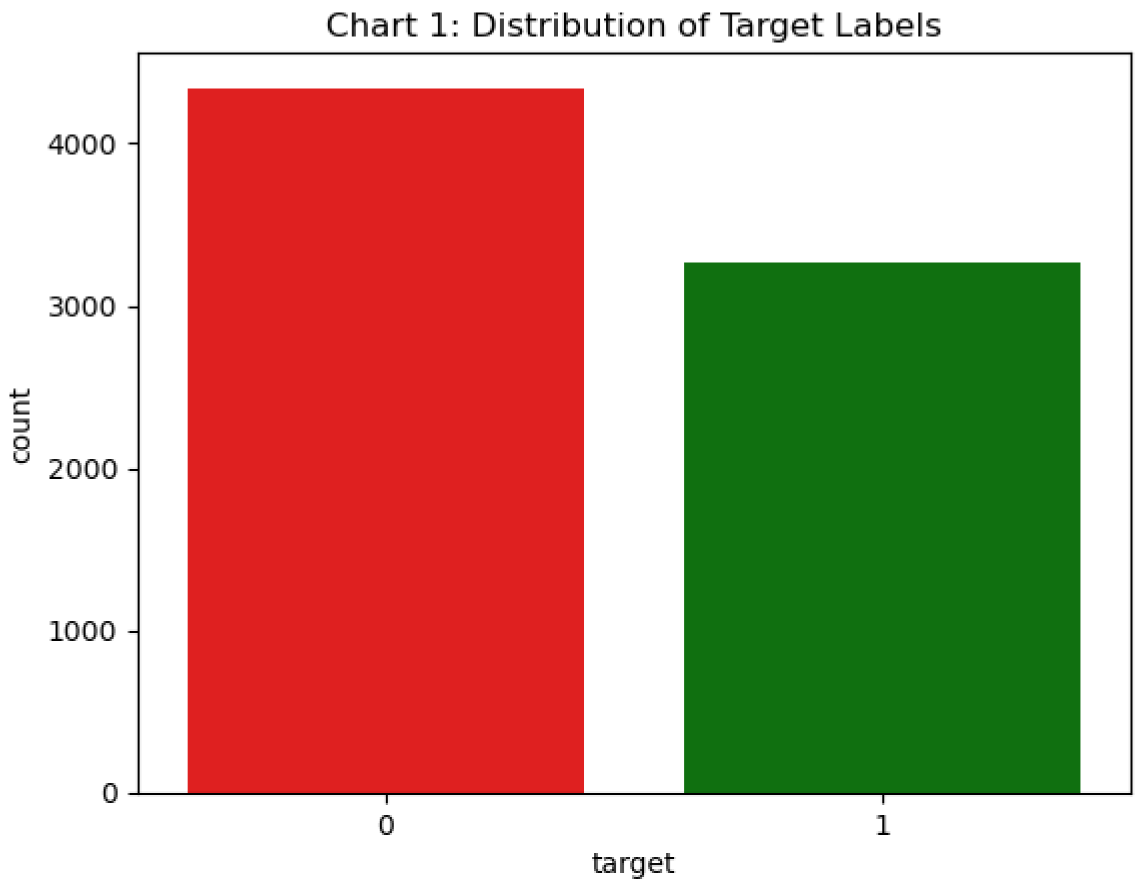
Label Distributions

In [6]:
```
plt.figure()
sns.countplot(data=train_df, x='target', palette=['#ff0000',"#008000"])
plt.title('Chart 1: Distribution of Target Labels')
```

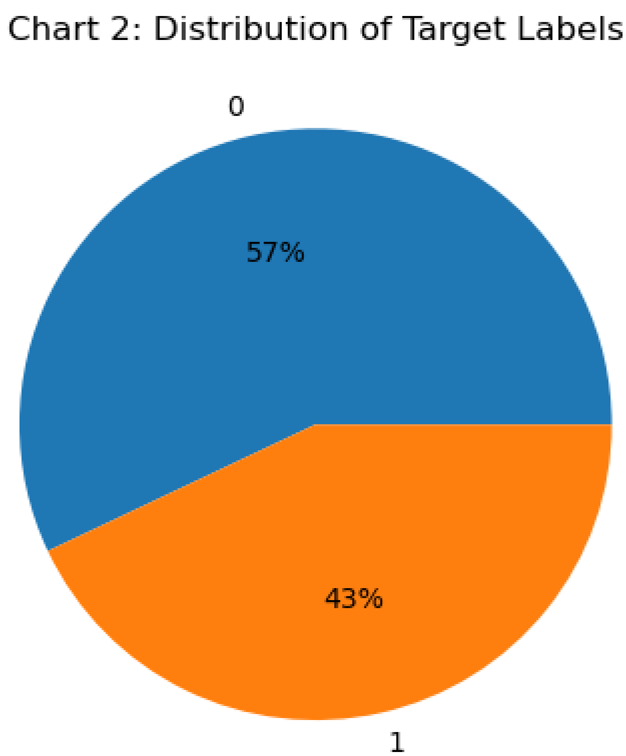
```
plt.show()

target_0 = train_df["target"].value_counts()[0]
target_1 = train_df["target"].value_counts()[1]
print('0 Label: ', target_0)
print('1 Label: ', target_1)

plt.figure()
plt.pie([target_0, target_1], labels=[0,1], autopct='%.0f%%')
plt.title('Chart 2: Distribution of Target Labels')
plt.show()
```




```
0 Label:  4342
1 Label:  3271
```



Cleanse Data

Remove Hyperlinks

In [6]:
```
import re

print('Before')
print(train_df['text'][31])
print(test_df['text'][32])

train_df['text'] = train_df['text'].apply(lambda x:  re.sub(r'https?:\/\/\S+', '', x))
test_df['text'] = test_df['text'].apply(lambda x:  re.sub(r'https?:\/\/\S+', '', x))

print('After')
print(train_df['text'][31])
print(test_df['text'][32])
```

```
Before
@bbcmtd Wholesale Markets ablaze http://t.co/lHYXEOHY6C
#3: Car Recorder ZeroEdgeå¨ Dual-lens Car Camera Vehicle Traffic/Driving History/Acci
dent Camcorder  Large Re... http://t.co/kKFaSJv6Cj
After
@bbcmtd Wholesale Markets ablaze
#3: Car Recorder ZeroEdgeå¨ Dual-lens Car Camera Vehicle Traffic/Driving History/Acci
dent Camcorder  Large Re...
```

Remove Punctuation

In [7]:
```
import warnings
warnings.filterwarnings('ignore')

train_df['text'] = train_df['text'].str.replace(r'[^\w\s]+', '')
```

```
test_df['text'] = test_df['text'].str.replace(r'[^\w\s]+', '')
print(train_df.head(), '/n')
print(test_df.head())
```

```
   id                                               text  target
0   1  Our Deeds are the Reason of this earthquake Ma...       1
1   4             Forest fire near La Ronge Sask Canada       1
2   5  All residents asked to shelter in place are be...       1
3   6  13000 people receive wildfires evacuation orde...       1
4   7  Just got sent this photo from Ruby Alaska as s...       1 /n
   id                                               text
0   0                 Just happened a terrible car crash
1   2  Heard about earthquake is different cities sta...
2   3  there is a forest fire at spot pond geese are ...
3   9             Apocalypse lighting Spokane wildfires
4  11      Typhoon Soudelor kills 28 in China and Taiwan
```

Remove Stop Words

In [9]:

```
import nltk
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
train_df['text'] = train_df['text'].apply(lambda x: ' '.join([word for word in x.split
test_df['text'] = test_df['text'].apply(lambda x: ' '.join([word for word in x.split(
print(train_df.head(), '/n')
print(test_df.head())
```

```
   id                                               text  target
0   1   Our Deeds Reason earthquake May ALLAH Forgive us       1
1   4             Forest fire near La Ronge Sask Canada       1
2   5  All residents asked shelter place notified off...       1
3   6  13000 people receive wildfires evacuation orde...       1
4   7  Just got sent photo Ruby Alaska smoke wildfire...       1 /n
   id                                               text
0   0                     Just happened terrible car crash
1   2  Heard earthquake different cities stay safe ev...
2   3  forest fire spot pond geese fleeing across str...
3   9             Apocalypse lighting Spokane wildfires
4  11                Typhoon Soudelor kills 28 China Taiwan
```

Convert to all lower case

In [10]:

```
print('Before')
print(train_df['text'][31])
print(test_df['text'][32])

train_df['text'] = train_df['text'].apply(lambda x:  x.lower())
test_df['text'] = test_df['text'].apply(lambda x:  x.lower())

print('After')
print(train_df['text'][31])
print(test_df['text'][32])
```

```
Before
bbcmtd Wholesale Markets ablaze
3 Car Recorder ZeroEdgeå Duallens Car Camera Vehicle TrafficDriving HistoryAccident C
amcorder Large Re
After
bbcmtd wholesale markets ablaze
3 car recorder zeroedgeå duallens car camera vehicle trafficdriving historyaccident c
amcorder large re
```

Models

Model Architecture - Long-Short Term Memory (LSTM)

The LSTM model is built around the recurring neural networks (RNN) architecture. RNNs are neural networks that are capable of analyzing sequential or time series data. The LSTM is based on a network of gates and feedback loops between nodes and layers. These gates and feedback loops result in the ability of the network to maintain information over time, where the time frames vary.

The ability to maintain both short- and long-term information allows the network to contextualize the information. The contextualization characteristic brings the network closer to analyzing information as a human would. This makes LSTM models especially adept at analyzing sequential data such a speech or text recognition.

We will use the LSTM model to analyze and categorize texts in this project. The goal of the LSTM is to properly characterize texts as disaster related or not. The LSTM model will use training and learned context within the texts to characterize each text.

The basic LSTM first initializes the text sequence through an encoder. The encoder transforms the character text strings in to linear arrays based on encoder parameters. The encoded sequences then pass through various LSTM layers and dense layers to generate a characterization of the text.

This project will compare two LTSM model architectures, based on accuracy, to see which architecture provides the better performance. The first model will be a basic LSTM model with and embedding(encoding) layer, a bi-directional LSTM layer, followed by a dense layer with ReLU activation. Then pass to a dense layer of one that feeds the characterization layer with sigmoid activation for characterization purposes.

The second model will bracket the LSTM layer with a spatial dropout and dropout layer. The purpose of the addition of the dropout layers is to assess the impact on potential overfitting in the base model.

Transfer learning will not be deployed as to get a better feel for how well the base models learn and characterize from raw inputs.

Various epoch counts were reviewed with 15 being the sweet spot for observing the model differences, while maintaining a reasonable run time.

Simple LSTM Model

In [15]:

```
#Split the training data
x_training_set, x_validation_set, target_train, target_validation= train_test_split(tr


max_length_text = train_df.text.map(len).max()
print('Maximum Text Length: ', max_length_text)
dict_size = 15000
embedding_size = 64
#Set Up tokenizer

tokenizer = text.Tokenizer(num_words = dict_size)
tokenizer.fit_on_texts(x_training_set)

word_index = tokenizer.word_index

sequence_x_training_set = tokenizer.texts_to_sequences(x_training_set)
sequence_x_validation_set = tokenizer.texts_to_sequences(x_validation_set)
sequence_x_test_set = tokenizer.texts_to_sequences(test_df['text'].values)

sequence_x_training_set_padded = sequence.pad_sequences(sequence_x_training_set, maxle
sequence_x_validation_set_padded = sequence.pad_sequences(sequence_x_validation_set, m
sequence_x_test_set_padded = sequence.pad_sequences(sequence_x_test_set, maxlen=max_le

print('Training Shape: ', sequence_x_training_set_padded.shape)
print('Sample: ', sequence_x_training_set_padded[0])

#Hyperparameter Tuning variables
dropout_rate = .2
recurrent_dropout_rate = .2
num_epochs = 15

model = Sequential()

model.add(Embedding(dict_size, embedding_size, input_length=max_length_text))
model.add(Bidirectional(tf.keras.layers.LSTM(64, dropout = dropout_rate, recurrent_dro
model.add(Dense(embedding_size, activation='relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

input_shape = sequence_x_training_set_padded.shape
model.build(input_shape)

model.summary()

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

history = model.fit(sequence_x_training_set_padded, np.asarray(target_train), epochs=n
```

```
Maximum Text Length:  139
Training Shape:  (5329, 139)
Sample:  [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0 3519 1057 5398  968  969  239  373 3520 2660]
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_19 (ModuleWra (5329, 139, 64)           960000
_________________________________________________________________
module_wrapper_20 (ModuleWra (5329, 128)               66048
_________________________________________________________________
module_wrapper_21 (ModuleWra (5329, 64)                8256
_________________________________________________________________
module_wrapper_22 (ModuleWra (5329, 1)                 65
_________________________________________________________________
module_wrapper_23 (ModuleWra (5329, 1)                 0
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
167/167 - 36s - loss: 0.5635 - accuracy: 0.6999 - val_loss: 0.2911 - val_accuracy: 0.
8951
Epoch 2/15
167/167 - 31s - loss: 0.2897 - accuracy: 0.8853 - val_loss: 0.1469 - val_accuracy: 0.
9527
Epoch 3/15
167/167 - 29s - loss: 0.1557 - accuracy: 0.9454 - val_loss: 0.0904 - val_accuracy: 0.
9779
Epoch 4/15
167/167 - 33s - loss: 0.0963 - accuracy: 0.9696 - val_loss: 0.0549 - val_accuracy: 0.
9831
Epoch 5/15
167/167 - 34s - loss: 0.0717 - accuracy: 0.9773 - val_loss: 0.0446 - val_accuracy: 0.
9840
Epoch 6/15
167/167 - 35s - loss: 0.0585 - accuracy: 0.9801 - val_loss: 0.0372 - val_accuracy: 0.
9854
Epoch 7/15
167/167 - 39s - loss: 0.0460 - accuracy: 0.9818 - val_loss: 0.0307 - val_accuracy: 0.
9859
Epoch 8/15
167/167 - 37s - loss: 0.0410 - accuracy: 0.9797 - val_loss: 0.0262 - val_accuracy: 0.
9884
Epoch 9/15
167/167 - 38s - loss: 0.0380 - accuracy: 0.9820 - val_loss: 0.0249 - val_accuracy: 0.
9872
Epoch 10/15
167/167 - 47s - loss: 0.0296 - accuracy: 0.9837 - val_loss: 0.0237 - val_accuracy: 0.
9863
```

```
Epoch 11/15
167/167 - 36s - loss: 0.0288 - accuracy: 0.9840 - val_loss: 0.0239 - val_accuracy: 0.
9869
Epoch 12/15
167/167 - 33s - loss: 0.0344 - accuracy: 0.9829 - val_loss: 0.0239 - val_accuracy: 0.
9876
Epoch 13/15
167/167 - 33s - loss: 0.0290 - accuracy: 0.9837 - val_loss: 0.0230 - val_accuracy: 0.
9874
Epoch 14/15
167/167 - 34s - loss: 0.0254 - accuracy: 0.9850 - val_loss: 0.0220 - val_accuracy: 0.
9886
Epoch 15/15
167/167 - 33s - loss: 0.0268 - accuracy: 0.9842 - val_loss: 0.0250 - val_accuracy: 0.
9859
```

Complex LSTM Model

Increase the complexity of the previous model by adding a Spatial Dropout layer and a Dropout layer around the LSTM layer. The addtional layers are intended to help provide increased accurracy while attempting to protect agaist overfitting.

In [16]:

```
model_complex = Sequential()

model_complex.add(Embedding(dict_size, embedding_size, input_length=max_length_text))
model_complex.add(SpatialDropout1D(0.5))  ## addtional layer
model_complex.add(Bidirectional(tf.keras.layers.LSTM(64, dropout = dropout_rate, recur
model_complex.add(Dropout(0.2)) ## addtional layer
model_complex.add(Dense(embedding_size, activation='relu'))
model_complex.add(Dense(1))
model_complex.add(Activation('sigmoid'))

input_shape = sequence_x_training_set_padded.shape
model_complex.build(input_shape)

model_complex.summary()

model_complex.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

comlex_history = model_complex.fit(sequence_x_training_set_padded, np.asarray(target_t
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
module_wrapper_24 (ModuleWra (5329, 139, 64)           960000    
_________________________________________________________________
module_wrapper_25 (ModuleWra (5329, 139, 64)           0         
_________________________________________________________________
module_wrapper_26 (ModuleWra (5329, 128)               66048     
_________________________________________________________________
module_wrapper_27 (ModuleWra (5329, 128)               0         
_________________________________________________________________
module_wrapper_28 (ModuleWra (5329, 64)                8256      
_________________________________________________________________
module_wrapper_29 (ModuleWra (5329, 1)                 65        
_________________________________________________________________
module_wrapper_30 (ModuleWra (5329, 1)                 0         
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
167/167 - 41s - loss: 0.6004 - accuracy: 0.6622 - val_loss: 0.3589 - val_accuracy: 0.
8525
Epoch 2/15
167/167 - 32s - loss: 0.3621 - accuracy: 0.8497 - val_loss: 0.2182 - val_accuracy: 0.
9266
Epoch 3/15
167/167 - 30s - loss: 0.2379 - accuracy: 0.9099 - val_loss: 0.1356 - val_accuracy: 0.
9557
Epoch 4/15
167/167 - 31s - loss: 0.1604 - accuracy: 0.9428 - val_loss: 0.0869 - val_accuracy: 0.
9704
Epoch 5/15
167/167 - 43s - loss: 0.1201 - accuracy: 0.9587 - val_loss: 0.0715 - val_accuracy: 0.
9775
Epoch 6/15
167/167 - 36s - loss: 0.0953 - accuracy: 0.9696 - val_loss: 0.0563 - val_accuracy: 0.
9829
Epoch 7/15
167/167 - 36s - loss: 0.0855 - accuracy: 0.9741 - val_loss: 0.0500 - val_accuracy: 0.
9852
Epoch 8/15
167/167 - 37s - loss: 0.0738 - accuracy: 0.9771 - val_loss: 0.0459 - val_accuracy: 0.
9846
Epoch 9/15
167/167 - 42s - loss: 0.0708 - accuracy: 0.9777 - val_loss: 0.0410 - val_accuracy: 0.
9859
Epoch 10/15
167/167 - 40s - loss: 0.0710 - accuracy: 0.9775 - val_loss: 0.0506 - val_accuracy: 0.
9831
Epoch 11/15
167/167 - 59s - loss: 0.0648 - accuracy: 0.9775 - val_loss: 0.0353 - val_accuracy: 0.
9865
Epoch 12/15
167/167 - 56s - loss: 0.0558 - accuracy: 0.9822 - val_loss: 0.0322 - val_accuracy: 0.
9869
Epoch 13/15
167/167 - 41s - loss: 0.0532 - accuracy: 0.9814 - val_loss: 0.0302 - val_accuracy: 0.
```

```
9876
Epoch 14/15
167/167 - 40s - loss: 0.0552 - accuracy: 0.9810 - val_loss: 0.0318 - val_accuracy: 0.
9869
Epoch 15/15
167/167 - 40s - loss: 0.0476 - accuracy: 0.9822 - val_loss: 0.0277 - val_accuracy: 0.
9872
```

Model Comparison

In [20]:

```
#Simple Model
model_accuracy = history.history['accuracy']
model_val_acc = history.history['val_accuracy']
model_loss = history.history['loss']
model_val_loss = history.history['val_loss']

epochs_range = range(num_epochs)

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, model_accuracy, label='Training Accuracy')
plt.plot(epochs_range, model_val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Plo1 1: Simple Model Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, model_loss, label='Training Loss')
plt.plot(epochs_range, model_val_loss, label='Validation Loss')
plt.legend(loc='upper center')
plt.title('Plot 2: Simple Model Loss')
plt.show()

#complex Model
model_complex_accuracy = comlex_history.history['accuracy']
model_complex_val_acc = comlex_history.history['val_accuracy']
model_complex_loss = comlex_history.history['loss']
model_complex_val_loss = comlex_history.history['val_loss']


plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, model_complex_accuracy, label='Training Accuracy')
plt.plot(epochs_range, model_complex_val_acc, label='Validation Accuracy')
plt.legend(loc='lower center')
plt.title('Plot 3: Complex Model Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, model_complex_loss, label='Training Loss')
plt.plot(epochs_range, model_complex_val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Plot 4: Complex Model Loss')
plt.show()
```

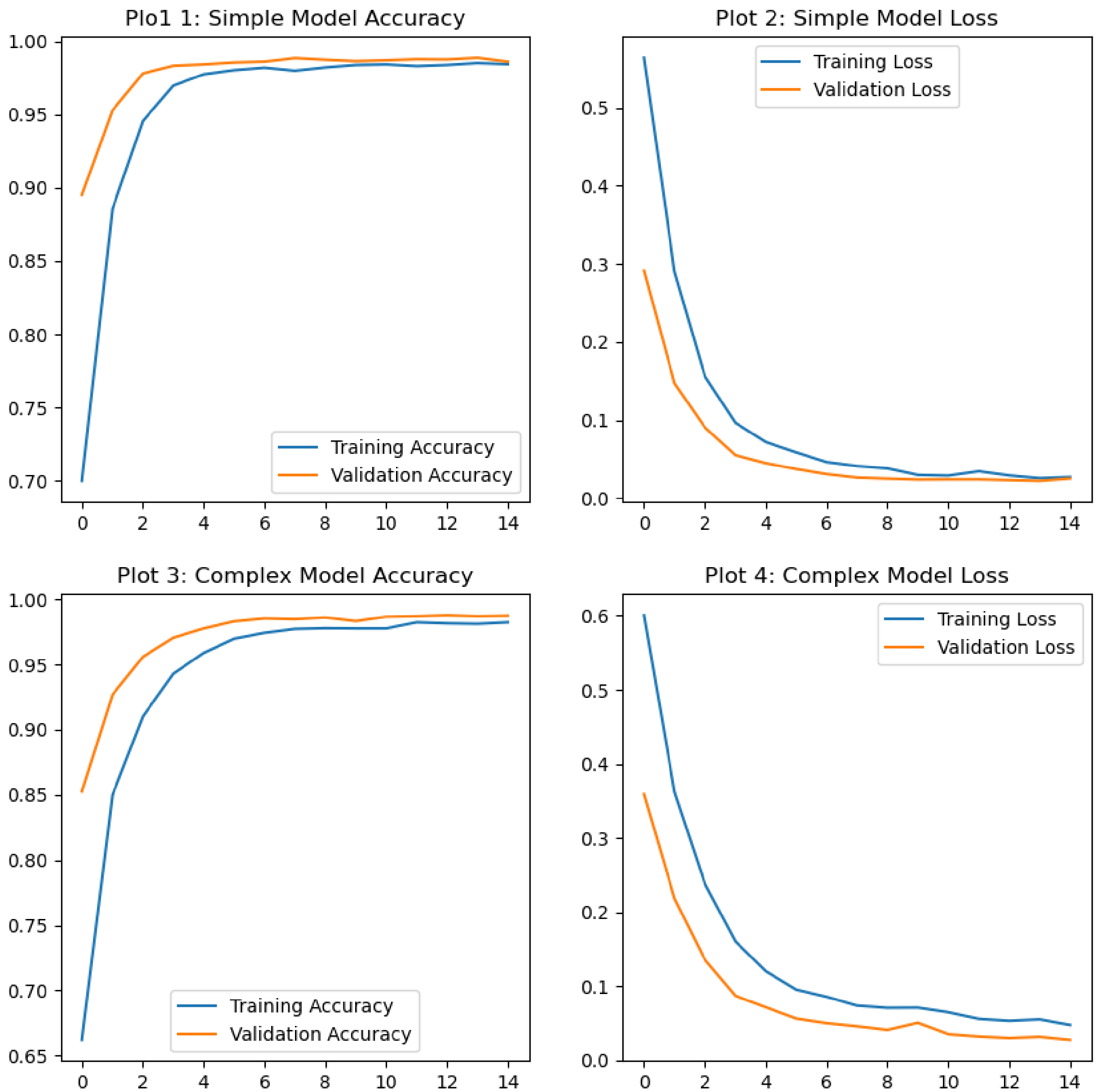


Table 1: Model Complexity vs Accuracy

Model	Accuracy	Validation Accuracy
Simple	.9842	.9889
Complex	.9722	.9872

Plots 1 thru 4 provide a view of the training accuracy and loss metrics between the simple and complex LSTM models. Based on the charts and table, increasing the complexity by adding the additional dropout layers negatively impacted the accuracy. The more complex model showed ~1.5% degradation.

However, when looking closely at the plots at around the 8th epoch, the simple model's accuracy and loss begin to show signs of slight cavitation. Additionally, by the 15th epoch the loss and accuracy of the base model are essentially identical.

Whereas the complex model's accuracy and loss continue to converge with the gap between the two eventually leveling out around the 10th epoch.

Taking these two observations together it appears that the simple model could be showing the initial signs of overfitting, while the complex model does not. The implication being that adding the additional dropout layers aided in protecting the complex model from potential overfitting.

Hyperparameter Tuning

Number of Epochs will be varied and the impact on accruacy assessed.

In [58]:

```
epoch_count = [5, 10, 15, 20]
history = []

for e in epoch_count:

    num_epochs = e

    tune_model = Sequential()

    tune_model.add(Embedding(dict_size, embedding_size, input_length=max_length_text))
    tune_model.add(Bidirectional(tf.keras.layers.LSTM(64, dropout = dropout_rate, recu
    tune_model.add(Dense(embedding_size, activation='relu'))
    tune_model.add(Dense(1))
    tune_model.add(Activation('sigmoid'))

    input_shape = sequence_x_training_set_padded.shape
    tune_model.build(input_shape)

    tune_model.summary()

    tune_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac

    tune_history = tune_model.fit(sequence_x_training_set_padded, np.asarray(target_tr

    print('Eochs: ', e)
    print('History: ', tune_history.history)
    history.append(tune_history.history)
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_132 (ModuleWr (5329, 139, 64)           960000
_________________________________________________________________
module_wrapper_133 (ModuleWr (5329, 128)               66048
_________________________________________________________________
module_wrapper_134 (ModuleWr (5329, 64)                8256
_________________________________________________________________
module_wrapper_135 (ModuleWr (5329, 1)                 65
_________________________________________________________________
module_wrapper_136 (ModuleWr (5329, 1)                 0
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
167/167 - 34s - loss: 0.5614 - accuracy: 0.7013 - val_loss: 0.3061 - val_accuracy: 0.
8790
Epoch 2/5
167/167 - 36s - loss: 0.2865 - accuracy: 0.8853 - val_loss: 0.1298 - val_accuracy: 0.
9568
Epoch 3/5
167/167 - 40s - loss: 0.1491 - accuracy: 0.9446 - val_loss: 0.0735 - val_accuracy: 0.
9780
Epoch 4/5
167/167 - 39s - loss: 0.0946 - accuracy: 0.9700 - val_loss: 0.0562 - val_accuracy: 0.
9829
Epoch 5/5
167/167 - 39s - loss: 0.0710 - accuracy: 0.9758 - val_loss: 0.0431 - val_accuracy: 0.
9844
Eochs:  5
History:  {'loss': [0.5614140033721924, 0.2864967882633209, 0.149135023355484, 0.0946
434885263443, 0.07098724693059921], 'accuracy': [0.7012572884559631, 0.88534432649612
43, 0.9446425437927246, 0.9699755907058716, 0.9757928252220154], 'val_loss': [0.30607
131123542786, 0.12984530627727509, 0.07354626059532166, 0.05617747828364372, 0.043070
36101818085], 'val_accuracy': [0.8789641857147217, 0.956839919090271, 0.9780446887016
296, 0.9829236268997192, 0.9844248294830322]}
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_137 (ModuleWr (5329, 139, 64)           960000
_________________________________________________________________
module_wrapper_138 (ModuleWr (5329, 128)               66048
_________________________________________________________________
module_wrapper_139 (ModuleWr (5329, 64)                8256
_________________________________________________________________
module_wrapper_140 (ModuleWr (5329, 1)                 65
_________________________________________________________________
module_wrapper_141 (ModuleWr (5329, 1)                 0
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
167/167 - 48s - loss: 0.5585 - accuracy: 0.7142 - val_loss: 0.3037 - val_accuracy: 0.
```

```
8989
Epoch 2/10
167/167 - 43s - loss: 0.2814 - accuracy: 0.8913 - val_loss: 0.1523 - val_accuracy: 0.
9606
Epoch 3/10
167/167 - 41s - loss: 0.1514 - accuracy: 0.9458 - val_loss: 0.0770 - val_accuracy: 0.
9758
Epoch 4/10
167/167 - 41s - loss: 0.0971 - accuracy: 0.9698 - val_loss: 0.0629 - val_accuracy: 0.
9820
Epoch 5/10
167/167 - 41s - loss: 0.0776 - accuracy: 0.9737 - val_loss: 0.0554 - val_accuracy: 0.
9848
Epoch 6/10
167/167 - 42s - loss: 0.0624 - accuracy: 0.9794 - val_loss: 0.0371 - val_accuracy: 0.
9863
Epoch 7/10
167/167 - 48s - loss: 0.0463 - accuracy: 0.9820 - val_loss: 0.0315 - val_accuracy: 0.
9861
Epoch 8/10
167/167 - 48s - loss: 0.0359 - accuracy: 0.9827 - val_loss: 0.0266 - val_accuracy: 0.
9867
Epoch 9/10
167/167 - 42s - loss: 0.0432 - accuracy: 0.9807 - val_loss: 0.0264 - val_accuracy: 0.
9869
Epoch 10/10
167/167 - 41s - loss: 0.0362 - accuracy: 0.9810 - val_loss: 0.0271 - val_accuracy: 0.
9865
Eochs:  10
History:  {'loss': [0.5584985017776489, 0.2814079821109772, 0.15138214826583862, 0.09
710437804460526, 0.07755817472934723, 0.06239072978496515, 0.04626854136586189, 0.03
5866398364305496, 0.04322785511612892, 0.03619629144668579], 'accuracy': [0.714205265
045166, 0.891349196434021, 0.945768415927887, 0.9697879552841187, 0.9737286567687988,
0.9793582558631897, 0.9819853901863098, 0.9827359914779663, 0.9806718230247498, 0.981
0470938682556], 'val_loss': [0.3037181794643402, 0.15231981873512268, 0.0770305618643
7607, 0.062869131565094, 0.055403560400009155, 0.03707709535956383, 0.031542345881462
1, 0.026636997237801552, 0.026371002197265625, 0.02713766135275364], 'val_accuracy':
[0.8988553285598755, 0.9605929851531982, 0.9757928252220154, 0.9819853901863098, 0.98
48001599311829, 0.9863013625144958, 0.9861137270927429, 0.9866766929626465, 0.9868643
283843994, 0.9864889979362488]}
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_142 (ModuleWr (5329, 139, 64)           960000
_________________________________________________________________
module_wrapper_143 (ModuleWr (5329, 128)               66048
_________________________________________________________________
module_wrapper_144 (ModuleWr (5329, 64)                8256
_________________________________________________________________
module_wrapper_145 (ModuleWr (5329, 1)                 65
_________________________________________________________________
module_wrapper_146 (ModuleWr (5329, 1)                 0
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
167/167 - 54s - loss: 0.5589 - accuracy: 0.7020 - val_loss: 0.3012 - val_accuracy: 0.
```

```
8923
Epoch 2/15
167/167 - 52s - loss: 0.2850 - accuracy: 0.8840 - val_loss: 0.1485 - val_accuracy: 0.
9557
Epoch 3/15
167/167 - 47s - loss: 0.1496 - accuracy: 0.9463 - val_loss: 0.0842 - val_accuracy: 0.
9773
Epoch 4/15
167/167 - 50s - loss: 0.0984 - accuracy: 0.9670 - val_loss: 0.0555 - val_accuracy: 0.
9833
Epoch 5/15
167/167 - 48s - loss: 0.0778 - accuracy: 0.9741 - val_loss: 0.0445 - val_accuracy: 0.
9850
Epoch 6/15
167/167 - 46s - loss: 0.0584 - accuracy: 0.9801 - val_loss: 0.0414 - val_accuracy: 0.
9835
Epoch 7/15
167/167 - 54s - loss: 0.0517 - accuracy: 0.9803 - val_loss: 0.0379 - val_accuracy: 0.
9848
Epoch 8/15
167/167 - 56s - loss: 0.0417 - accuracy: 0.9824 - val_loss: 0.0309 - val_accuracy: 0.
9856
Epoch 9/15
167/167 - 65s - loss: 0.0366 - accuracy: 0.9839 - val_loss: 0.0293 - val_accuracy: 0.
9863
Epoch 10/15
167/167 - 58s - loss: 0.0359 - accuracy: 0.9833 - val_loss: 0.0409 - val_accuracy: 0.
9859
Epoch 11/15
167/167 - 61s - loss: 0.0437 - accuracy: 0.9797 - val_loss: 0.0251 - val_accuracy: 0.
9869
Epoch 12/15
167/167 - 62s - loss: 0.0310 - accuracy: 0.9840 - val_loss: 0.0254 - val_accuracy: 0.
9865
Epoch 13/15
167/167 - 57s - loss: 0.0283 - accuracy: 0.9837 - val_loss: 0.0229 - val_accuracy: 0.
9874
Epoch 14/15
167/167 - 57s - loss: 0.0269 - accuracy: 0.9842 - val_loss: 0.0230 - val_accuracy: 0.
9876
Epoch 15/15
167/167 - 50s - loss: 0.0248 - accuracy: 0.9846 - val_loss: 0.0212 - val_accuracy: 0.
9884
Eochs:  15
History:  {'loss': [0.5588950514793396, 0.28500956296920776, 0.14960886538028717, 0.0
9843225032091141, 0.07783649116754532, 0.058436907827854156, 0.05170290917158127, 0.0
41731905192136765, 0.0365636833012104, 0.03591294586658478, 0.043717872351408005, 0.0
310295931994915, 0.028256408870220184, 0.026917271316051483, 0.02480809949338436], 'a
ccuracy': [0.7020078897476196, 0.8840307593345642, 0.9463313817977905, 0.966973185539
2456, 0.9741039872169495, 0.9801088571548462, 0.9802964925765991, 0.9823606610298157,
0.9838618636131287, 0.9832989573478699, 0.9797335267066956, 0.9840495586395264, 0.983
6742281913757, 0.9842371940612793, 0.9846125245094299], 'val_loss': [0.30117154121398
926, 0.14850321412086487, 0.0841822698712349, 0.05548781156539917, 0.0445405803620815
3, 0.041394900530576706, 0.03793704882264137, 0.03088011033833027, 0.0293108187615871
43, 0.04090180993080139, 0.025051996111869812, 0.02536335214972496, 0.022923635318875
313, 0.022966450080275536, 0.021172823384404182], 'val_accuracy': [0.892287492752075
2, 0.9557140469551086, 0.9772940278053284, 0.9832989573478699, 0.9849877953529358, 0.
9834865927696228, 0.9848001599311829, 0.9855507612228394, 0.9863013625144958, 0.98592
609167099, 0.9868643283843994, 0.9864889979362488, 0.987427294254303, 0.9876149296760
559, 0.9883655309677124]}
```

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_147 (ModuleWr (5329, 139, 64)           960000
_________________________________________________________________
module_wrapper_148 (ModuleWr (5329, 128)               66048
_________________________________________________________________
module_wrapper_149 (ModuleWr (5329, 64)                8256
_________________________________________________________________
module_wrapper_150 (ModuleWr (5329, 1)                 65
_________________________________________________________________
module_wrapper_151 (ModuleWr (5329, 1)                 0
=================================================================
Total params: 1,034,369
Trainable params: 1,034,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
167/167 - 54s - loss: 0.5591 - accuracy: 0.7050 - val_loss: 0.3000 - val_accuracy: 0.
8801
Epoch 2/20
167/167 - 50s - loss: 0.2854 - accuracy: 0.8874 - val_loss: 0.1411 - val_accuracy: 0.
9583
Epoch 3/20
167/167 - 48s - loss: 0.1523 - accuracy: 0.9439 - val_loss: 0.0894 - val_accuracy: 0.
9762
Epoch 4/20
167/167 - 47s - loss: 0.0937 - accuracy: 0.9675 - val_loss: 0.0549 - val_accuracy: 0.
9831
Epoch 5/20
167/167 - 49s - loss: 0.0731 - accuracy: 0.9762 - val_loss: 0.0531 - val_accuracy: 0.
9810
Epoch 6/20
167/167 - 49s - loss: 0.0584 - accuracy: 0.9788 - val_loss: 0.0375 - val_accuracy: 0.
9863
Epoch 7/20
167/167 - 45s - loss: 0.0477 - accuracy: 0.9809 - val_loss: 0.0351 - val_accuracy: 0.
9846
Epoch 8/20
167/167 - 46s - loss: 0.0412 - accuracy: 0.9822 - val_loss: 0.0337 - val_accuracy: 0.
9857
Epoch 9/20
167/167 - 49s - loss: 0.0344 - accuracy: 0.9833 - val_loss: 0.0260 - val_accuracy: 0.
9876
Epoch 10/20
167/167 - 43s - loss: 0.0322 - accuracy: 0.9833 - val_loss: 0.0258 - val_accuracy: 0.
9878
Epoch 11/20
167/167 - 43s - loss: 0.0291 - accuracy: 0.9840 - val_loss: 0.0248 - val_accuracy: 0.
9876
Epoch 12/20
167/167 - 44s - loss: 0.0308 - accuracy: 0.9835 - val_loss: 0.0231 - val_accuracy: 0.
9874
Epoch 13/20
167/167 - 49s - loss: 0.0275 - accuracy: 0.9848 - val_loss: 0.0224 - val_accuracy: 0.
9874
Epoch 14/20
167/167 - 46s - loss: 0.0283 - accuracy: 0.9854 - val_loss: 0.0216 - val_accuracy: 0.
9882
```

```
Epoch 15/20
167/167 - 45s - loss: 0.0331 - accuracy: 0.9844 - val_loss: 0.0222 - val_accuracy: 0.
9880
Epoch 16/20
167/167 - 45s - loss: 0.0253 - accuracy: 0.9856 - val_loss: 0.0214 - val_accuracy: 0.
9880
Epoch 17/20
167/167 - 46s - loss: 0.0243 - accuracy: 0.9861 - val_loss: 0.0236 - val_accuracy: 0.
9852
Epoch 18/20
167/167 - 45s - loss: 0.0249 - accuracy: 0.9856 - val_loss: 0.0217 - val_accuracy: 0.
9882
Epoch 19/20
167/167 - 45s - loss: 0.0244 - accuracy: 0.9865 - val_loss: 0.0221 - val_accuracy: 0.
9880
Epoch 20/20
167/167 - 46s - loss: 0.0245 - accuracy: 0.9859 - val_loss: 0.0225 - val_accuracy: 0.
9882
Eochs:  20
History:  {'loss': [0.5590558052062988, 0.2853848934173584, 0.15233176946640015, 0.09
367526322603226, 0.07306363433599472, 0.058415140956640244, 0.04771655052900314, 0.04
124179482460022, 0.03442216292023659, 0.03222259506583214, 0.02914738468825817, 0.030
76910600066185, 0.027469146996736526, 0.0282643623650074, 0.03310258314013481, 0.0252
63847783207893, 0.02430000901222229, 0.024867689236998558, 0.024443116039037704, 0.02
4477217346429825], 'accuracy': [0.7050102949142456, 0.8874084949493408, 0.94389188289
64233, 0.9675361514091492, 0.976168155670166, 0.9787952899932861, 0.9808594584465027,
0.9821730256080627, 0.9832989573478699, 0.9832989573478699, 0.9840495586395264, 0.983
4865927696228, 0.9848001599311829, 0.9853631258010864, 0.9844248294830322, 0.98555076
12228394, 0.9861137270927429, 0.9855507612228394, 0.9864889979362488, 0.9859260916709
9], 'val_loss': [0.2999962568283081, 0.14114342629909515, 0.08939231932163239, 0.0549
1739884018898, 0.05307019501924515, 0.03753018379211426, 0.035120248794555664, 0.0337
00522035360336, 0.02599528059363365, 0.025758124887943268, 0.024822894483804703, 0.02
3097651079297066, 0.02237818017601967, 0.02163519896566868, 0.022177396342158318, 0.0
2141791209578514, 0.023563040420413017, 0.02166137658059597, 0.022070568054914474, 0.
022538933902978897], 'val_accuracy': [0.880090057849884, 0.9583411812782288, 0.976168
155670166, 0.9831112623214722, 0.9810470938682556, 0.9863013625144958, 0.984612524509
4299, 0.9857383966445923, 0.9876149296760559, 0.9878025650978088, 0.9876149296760559,
0.987427294254303, 0.987427294254303, 0.9881778955459595, 0.9879902601242065, 0.98799
02601242065, 0.9851754307746887, 0.9881778955459595, 0.9879902601242065, 0.9881778955
459595]}
```

Hyperparameter Tuning Results

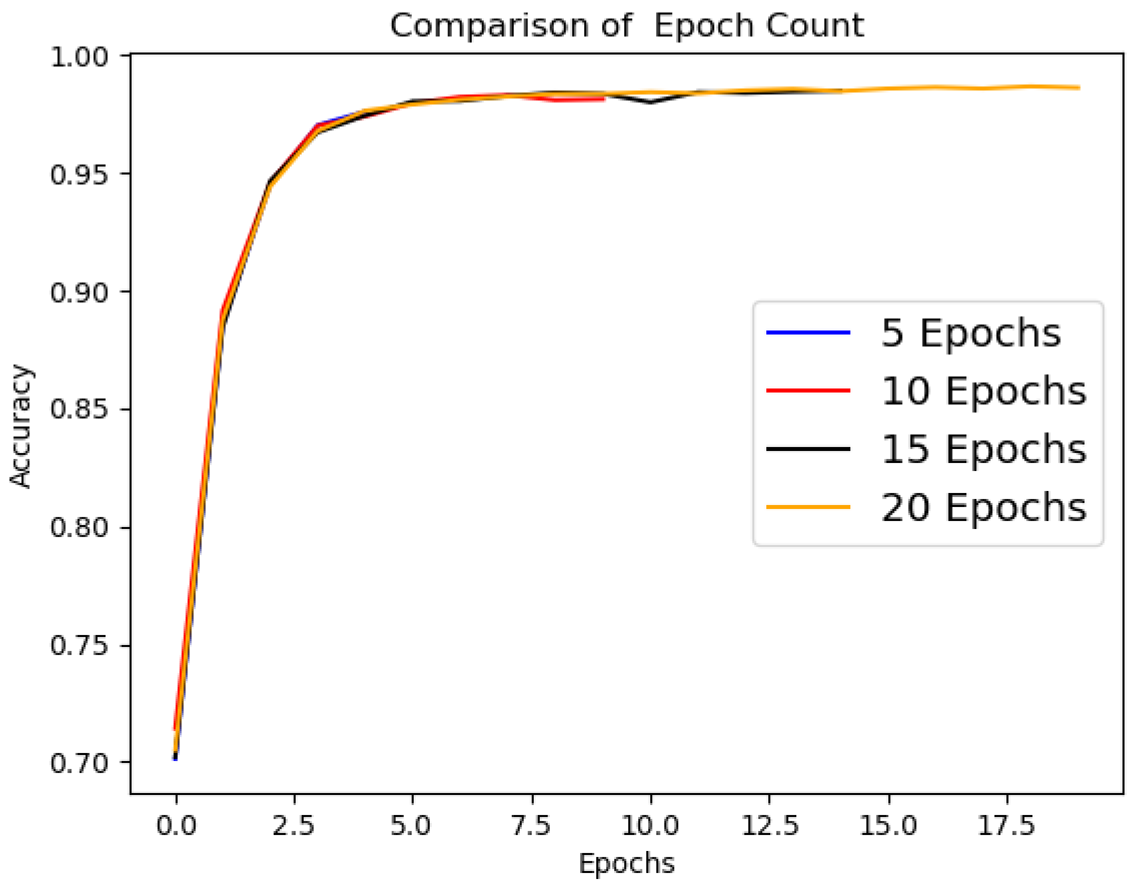
In [68]:

```python
e1 = [11, 10, 12, 14, 16, 19, 17, 14, 18, 17]
x_e1 = range(0,5)
x_e2 = range(0,10)
x_e3 = range(0,15)
x_e4 = range(0,20)

fig, ax = plt.subplots()
ax.plot(x_e1, history[0]['accuracy'], label='5 Epochs', color='blue')
ax.plot(x_e2, history[1]['accuracy'], label='10 Epochs', color = 'red')
ax.plot(x_e3, history[2]['accuracy'], label='15 Epochs', color='black')
ax.plot(x_e4, history[3]['accuracy'], label='20 Epochs', color = 'orange')

legend = ax.legend(loc='center right', fontsize='x-large')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.title('Plot 5: Comparison of  Epoch Count')
plt.show()
```



Accuracy Comparison

Table 2: Epoch Count vs Accuracy

Epoch Count	Accuracy	Validation Accuracy
5	.9758	.9844
10	.9810	.9865
15	.9846	.9884
20	.9859	.9882

As can be seen in Plot 5 and Table 2, increased epoch count provided a marginal impact to accuracy. Increasing the epoch count from 5 to 20 realized a 1% increase in accuracy from 97.58% to 98.59% respectively. Plot 5 also shows that the accuracy climb rate was not impacted by the number of epochs, all four cases climbed in accuracy at essentially the same rate.

Given the margin improvement in test and validation accuracy, going with the 10-15 epochs is the best tradeoff given computational resources and model accuracy.

In [21]:
```
#Get test set predictions
target_test = model.predict(sequence_x_test_set_padded, verbose = 1)
target_test
print('Test Shape: ', target_test.shape, '\n')
```

```
102/102 [==============================] - 4s 19ms/step
Test Shape:  (3263, 1)
```

In [22]:

```
#Generate submission file
test_submission = np.where(target_test <= 0.5, 0, 1)

final_submission = np.transpose(test_submission)[0]
final_submission = pd.DataFrame()
final_submission['id'] = test_df['id']
final_submission['target'] = test_submission
print(final_submission.head())

final_submission.to_csv('submission.csv', index=False)
```

```
   id  target
0   0       1
1   2       1
2   3       1
3   9       1
4  11       1
```

Conclusion

Model Comparison

Over 15 epochs, comparisons of the simple and complex models showed marginal accuracy and loss differences. The simple model performed slightly better with its final accuracy 1.2% better at 98.42%, then the complex model at 97.22%.

However, as outlined above, the simple model began to show the initial signs of potential overfitting at around the 10th epoch, while the complex model did not. Based on the analysis and discussion above, the conclusion is that the addition of the drop out layers to the complex model proved to be beneficial. Given the marginal difference in performance and the overfitting protection on the complex model, the complex model is deemed the better of the two.

Hyperparamter Tuning

Hyperparameter tuning, varying the number of epochs, provided some insight into how epoch count impacted the model's performance. The total difference in performance across the 4 different epoch counts resulted in a maximum accuracy difference of 1.01% between 5 and 20 epochs. The middle two epoch counts (10 and15) resulted in only a .36% difference in accuracy. When considering the essentially identical accuracy rate climb between models and the minimal accuracy differences, 10 epochs is deemed more than sufficient in getting good accuracy, while balancing computation effort.

Test Submission

Best model submission was 78.85%. Based on the much better performance of the training and validation models one can conclude that overfitting existed or that the test data was reasonably different from the training data set. Most likely a combination of both.

References

Remove Stop Words from Text in DataFrame Column, https://www.datasnips.com/58/remove-stop-words-from-text-in-dataframe-column/

How to Remove URLs from Text in Python, https://bobbyhadz.com/blog/python-remove-url-from-text