Machine Learning-based FakeNews Detection

1. Introduction

There are various newsworthy messages on social net media platforms, such as Facebook, Twitter, meanwhile, disseminating misleading information and fake news across social networks represent a big dilemma against researchers and social network service providers. It is meaningful to detect and classify fake news from the large amount of news on social media platforms to stop the propagation of rumor spreading. Machine learning as a tool is widely used to determine if a message or a news article is fake or not. In this paper, three machine learning models, including LSTM, RNN and CNN are used for fake news detection.

2. Dataset

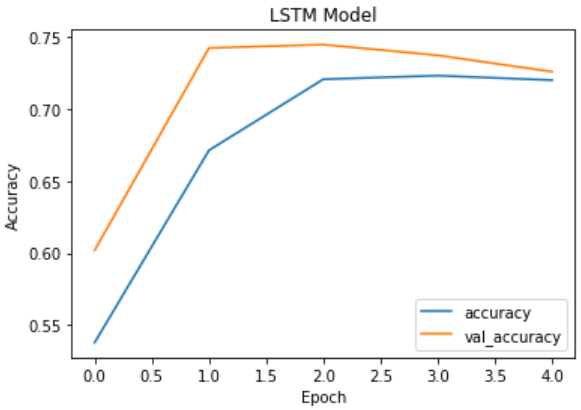
The Dataset for training and testing both were obtained from kaggle.
https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset?resource=download&select=True.csv (training data)

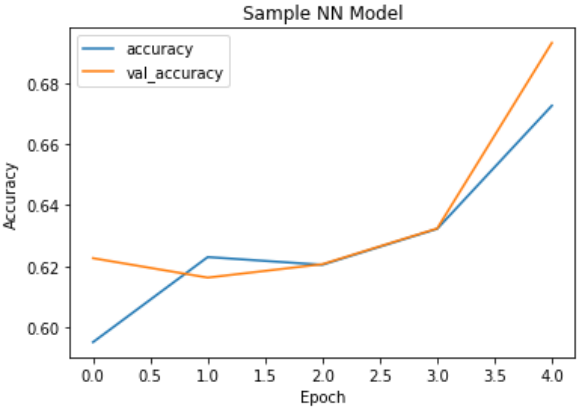
3. Methods

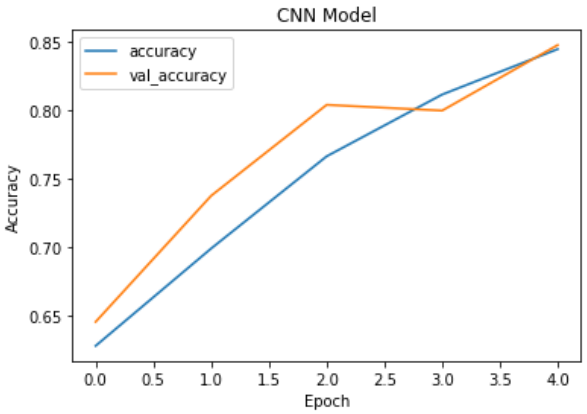
The dataset needs to be preprocessed by the NLP method, and be tokenized as the input to feed on the first layer of the training model.

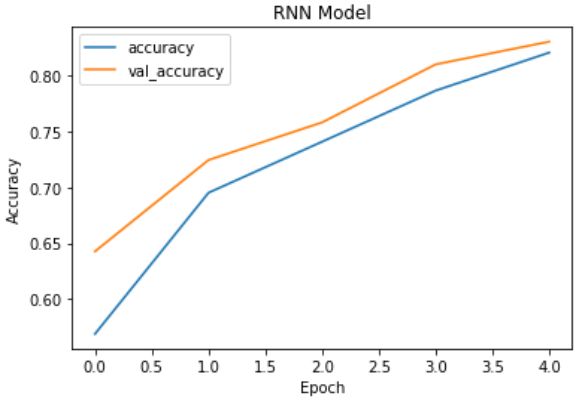
LSTM, RNN, CNN(1D and 2D) algorithms were applied to pre-train the classifiers, respectively. All models were trained with 5 epochs and 64 of batch size. The detection accuracy of the LSTM model reached up to 73.41% while recording of training history shows the training accuracy of the classifier was always lower than its validation accuracy. The detection accuracy of Sample NN model was 69.51%, and the training accuracy and validation accuracy of the classifier almost overlap. However, the training accuracy of CNN(2D) model reached up to 87.28% which was higher than the training accuracy obtained from LSTM model, and the validation accuracy of CNN model was slightly lower than the training accuracy (It took almost 3 hours to train CNN model which has a relatively high accuracy but high cost of training time). The next trained model was RNN model, which achieves a relatively high training accuracy of 85.13% and its slightly higher than the validation accuracy. Considering both the performance of each model and training time, I combined LSTM and CNN(1D) into a new model–LSTM_CNN model and trained the model, which improved training accuracy up to 2%~5% when compared with the LSTM model. Finally, a public dataset was used to test the feasibility of LSTM_CNN model and RNN model. The results were not ideal–both validation accuracy were only around 58%. The model still needs to be further refined in order to improve its feasibility.

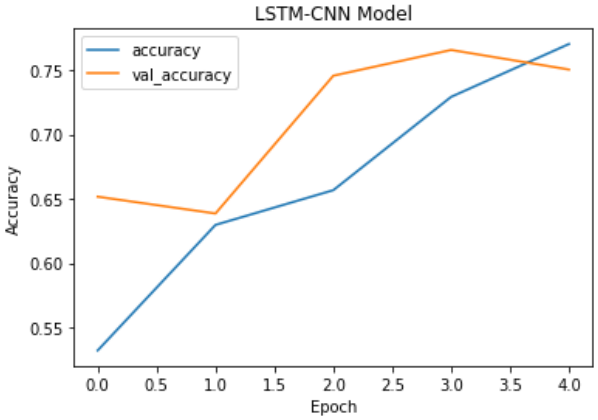
4. Results



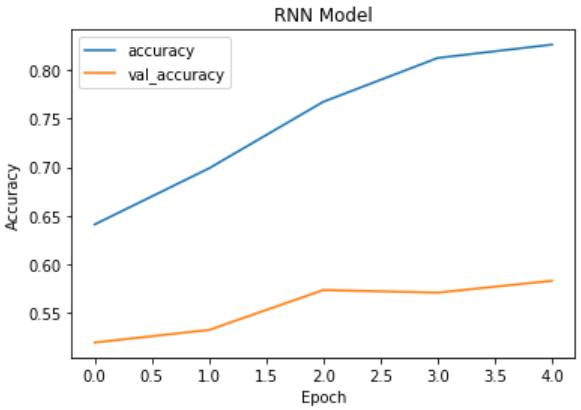


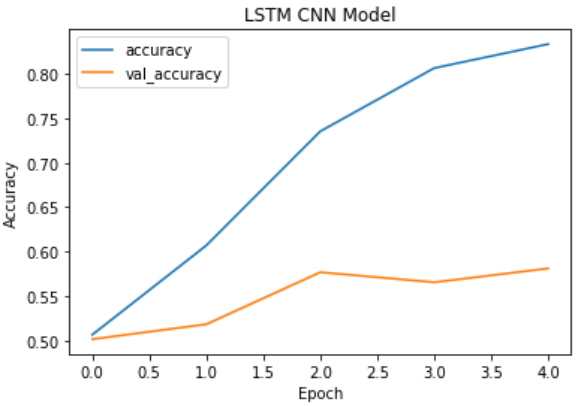






Test model using a public test dataset:





5. Coding

Upload dataset.

```
true = pd.read_csv('/content/gdrive/MyDrive/True.csv')
fake = pd.read_csv('/content/gdrive/MyDrive/Fake.csv')
```

```
true.head()
```

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

Extra the important columns of dataset.

```
df_pre = df[['title', 'text', 'label']]
df_pre['text'] = df['title'] + "" + df['text']
del df_pre['title']

df_pre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42625 entries, 0 to 42833
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    42625 non-null  object
 1   label   42625 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 999.0+ KB
```

Check the missing value.

```
null_df = df_pre.isnull().sum().sort_values(ascending = False)
percent = (df_pre.isnull().sum() / df_pre.isnull().count()).sort_values(ascending = False) * 100

null_df = pd.concat([null_df, percent], axis = 1, keys = ['Counts', '% Missing'])
null_df.head()
```

	Counts	% Missing
text	0	0.0
label	0	0.0

Split the dataset into training data and testing data.

```
x_train, x_test, y_train, y_test = train_test_split(df_pre['text'].to_numpy(), df_pre['label'].to_numpy(), test_size = 0.25)
len(x_train), len(x_test)
```

```
(31968, 10657)
```

Preprocess the dataset using the NLP method.

```
nltk.download('stopwords')
stop = set(stopwords.words('english'))
punctuation = list(string.punctuation)
stop.update(punctuation)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def strip_html(text):
  # get a BeautifulSoup object that follows standard structure
  soup = BeautifulSoup(text, "html.parser")
  return soup.get_text() # extract all the text from a page

# removing the square brackets
def remove_between_square_brackets(text):
  # '' replace patterns occured in left of text like \[[^]*\]
  return re.sub(r'\\[[^]*\]', '', text)

# removing URL's
def remove_urls(text):
  return re.sub(r'http\S+', '', text)

# removing the stopwords from text
def remove_stopwords(text):
  final_text = []
  for i in text.split():
    if i.strip().lower() not in stop:
      final_text.append(i.strip()) # strip(None) remove blank char in default
  return " ".join(final_text)

#removing punctuation from text
def remove_punctuation(text):
  return text.translate(str.maketrans('','',string.punctuation))
```

```
#removeing number
def remove_number(text):
  num = re.compile(r'\d+')
  return num.sub(r' num ', text)


# removing the noisy text
def denoise_text(text):
  text = strip_html(text)
  text = remove_between_square_brackets(text)
  text = remove_punctuation(text)
  text = remove_number(text)
  text = remove_stopwords(text)
  return text

# Apply function on review column
df_pre['text'] = df_pre['text'].apply(denoise_text)
```

```
import random
for i in range(20):
  print(df_pre['text'][random.randint(1,100)])
```

```
Spy chiefs pressure Congress renew expiring surveillance lawWASHINGTON Reuters leaders US intelligence community Thursday pressed Congress renew Natior
Factbox Trump Twitter Dec num Approval rating AmazonThe following statements posted verified Twitter accounts US President Donald Trump realDonaldTrump
US responds court fight illegal Indonesian immigrantsBOSTON Reuters US immigration officials sought block federal judge's order delaying efforts deport
Senior US Republican senator Let Mr Mueller jobWASHINGTON Reuters special counsel investigation links Russia President Trump's num election campaign cc
US military accept transgender recruits Monday PentagonWASHINGTON Reuters Transgender people allowed first time enlist US military starting Monday orde
US tax plan roils popular bet bond marketNEW YORK Reuters Passage longanticipated US tax overhaul upended bond market's favorite trade year yields long
Tax bills passthrough rule aid wealthy workers criticsWASHINGTON Reuters Wealthy business owners President Donald Trump stand gain provision Republican
Alabama official certify Senatorelect Jones today despite challenge CNNWASHINGTON Reuters Alabama Secretary State John Merrill said certify Democratic
Pence preside Senate tax bill vote office confirmsWASHINGTON Reuters US Vice President Mike Pence preside Senate's vote sweeping tax legislation office
House Democrats rally protect Special Counsel MuellerWASHINGTON Reuters Democrats US House Representatives rallied behind Special Counsel Robert Muelle
Factbox Trump Twitter Dec num Approval rating AmazonThe following statements posted verified Twitter accounts US President Donald Trump realDonaldTrump
Trump Twitter Dec num Tax cut Missile defense billThe following statements posted verified Twitter accounts US President Donald Trump realDonaldTrump F
Democrat Franken leave Senate January num WASHINGTON Reuters US Democratic Senator Al Franken earlier month announced plan resign following sexual misc
House panel asks Trump extop aide Bannon testify BloombergWASHINGTON Reuters Steve Bannon former top White House strategist former chief campaign aide
FBI Russia probe helped Australian diplomat tipoff NYTWASHINGTON Reuters Trump campaign adviser George Papadopoulos told Australian diplomat May num Ru
Senate begins debate final Republican tax billWASHINGTON Reuters Republicanled US Senate voted Tuesday begin debate sweeping tax legislation setting st
Banks healthcare service firms among winners US tax billNEW YORK Reuters Sweeping US tax legislation appears verge approval lifting prospects particula
Trump Twitter Dec num Hillary Clinton Tax Cut BillThe following statements posted verified Twitter accounts US President Donald Trump realDonaldTrump F
White House aide sees temporary funding fix childrens health programWASHINGTON Reuters shortterm fix fund Children's Health Insurance Program January
Green groups sue Trump administration delay methane ruleWASHINGTON Reuters coalition nearly num environmental Native American tribal groups sued Trump
```

Apply tokenizer to feed numerical data into the trained model.

```
max_voc_len = 10000
max_feature = 1 + round(sum([len(i.split()) for i in x_train]) / len(x_train))

max_feature
```

```
246
```

```
tokenizer = Tokenizer(num_words=max_voc_len)
tokenizer.fit_on_texts(texts = x_train)
X_train = tokenizer.texts_to_sequences(texts = x_train)
X_train = sequence.pad_sequences(X_train, maxlen=max_feature)
```

```
X_test = tokenizer.texts_to_sequences(texts = x_test)
X_test = sequence.pad_sequences(X_test, maxlen=max_feature)
```

Train LSTM model.

```
lstm_model = models.Sequential(name = 'LSTM_MODEL')
lstm_model.add(layers.Embedding(input_dim= max_voc_len, output_dim= 128, input_length=max_feature, trainable=False))
lstm_model.add(layers.LSTM(128, activation='relu', return_sequences=True, dropout=0.2, recurrent_dropout=0.2))
lstm_model.add(layers.LSTM(64, activation='relu', dropout=0.1, recurrent_dropout=0.1))
lstm_model.add(layers.Dense(32, activation='relu'))
lstm_model.add(layers.Dropout(rate=0.2))
lstm_model.add(layers.Dense(1, activation='sigmoid'))

adam_low_rate = optimizers.Adam(learning_rate=1.e-5)

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

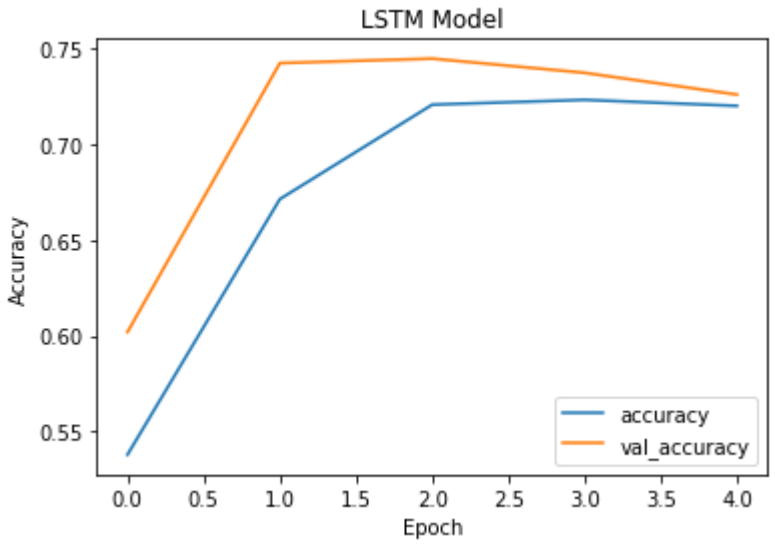
```
Model: "LSTM_MODEL"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 246, 128)          1280000

 lstm (LSTM)                 (None, 246, 128)          131584

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,463,105
Trainable params: 183,105
Non-trainable params: 1,280,000
_________________________________________________________________
```

```
lstm_model_history = lstm_model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=64)
```

```
Epoch 1/5
500/500 [==============================] - 730s 1s/step - loss: 0.6929 - accuracy: 0.5527 - val_loss: 0.6920 - val_accuracy: 0.7048
Epoch 2/5
500/500 [==============================] - 708s 1s/step - loss: 0.6908 - accuracy: 0.6778 - val_loss: 0.6884 - val_accuracy: 0.7231
Epoch 3/5
500/500 [==============================] - 683s 1s/step - loss: 0.6848 - accuracy: 0.7194 - val_loss: 0.6759 - val_accuracy: 0.7693
Epoch 4/5
500/500 [==============================] - 682s 1s/step - loss: 4354571.5000 - accuracy: 0.6846 - val_loss: 0.6468 - val_accuracy: 0.6721
Epoch 5/5
500/500 [==============================] - 698s 1s/step - loss: 0.6490 - accuracy: 0.6769 - val_loss: 0.6425 - val_accuracy: 0.7013
```

```
test_loss, test_acc = lstm_model.evaluate(X_train, y_train)
test_loss, test_acc
```

```
999/999 [==============================] - 122s 122ms/step - loss: 0.6757 - accuracy: 0.7341
(0.6756566762924194, 0.7340778112411499)
```



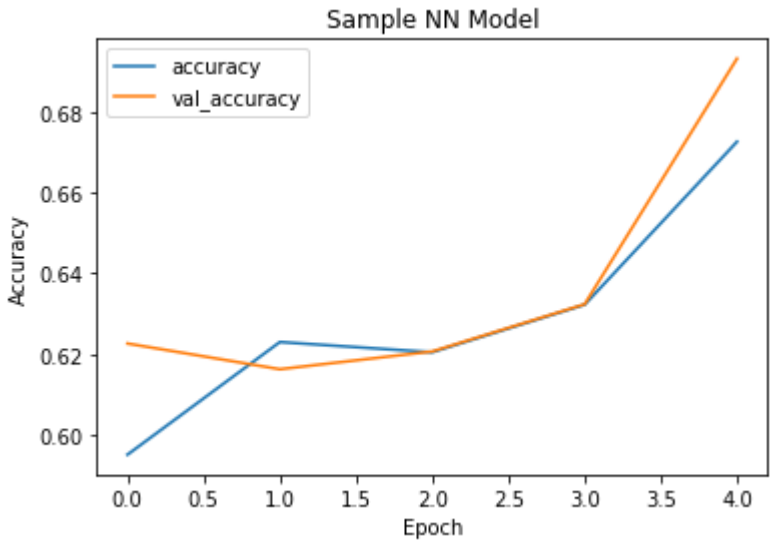
Train Sample NN model.

```
sample_nn_model = models.Sequential(name = 'Sample_NN_MODEL')
sample_nn_model.add(layers.Embedding(input_dim= max_voc_len, output_dim= 128, input_length=max_feature, trainable=False))
sample_nn_model.add(layers.Conv1D(128, 5, activation='relu'))
sample_nn_model.add(layers.MaxPooling1D(5))
sample_nn_model.add(layers.Conv1D(128, 5, activation='relu'))
sample_nn_model.add(layers.MaxPooling1D(5))
sample_nn_model.add(layers.Flatten())
sample_nn_model.add(layers.Dense(128, activation='relu'))
sample_nn_model.add(layers.Dense(64, activation='relu'))
sample_nn_model.add(layers.Dense(1, activation='sigmoid'))

adam_low_rate = optimizers.Adam(learning_rate=1.e-5)

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

```
Model: "Sample_NN_MODEL"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 246, 128)          1280000

 conv1d (Conv1D)             (None, 242, 128)          82048

 max_pooling1d (MaxPooling1D  (None, 48, 128)          0
 )

 conv1d_1 (Conv1D)           (None, 44, 128)           82048

 max_pooling1d_1 (MaxPooling  (None, 8, 128)           0
 1D)

 flatten (Flatten)           (None, 1024)              0

 dense_2 (Dense)             (None, 128)               131200

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,583,617
Trainable params: 303,617
Non-trainable params: 1,280,000
_________________________________________________________________
```

```
sample_nn_model_history = sample_nn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=64)
test_loss, test_acc = sample_nn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 95s 185ms/step - loss: 0.6894 - accuracy: 0.5950 - val_loss: 0.6839 - val_accuracy: 0.6226
Epoch 2/5
500/500 [==============================] - 90s 181ms/step - loss: 0.6778 - accuracy: 0.6230 - val_loss: 0.6706 - val_accuracy: 0.6162
Epoch 3/5
500/500 [==============================] - 92s 185ms/step - loss: 0.6637 - accuracy: 0.6204 - val_loss: 0.6564 - val_accuracy: 0.6206
Epoch 4/5
500/500 [==============================] - 94s 187ms/step - loss: 0.6474 - accuracy: 0.6322 - val_loss: 0.6372 - val_accuracy: 0.6324
Epoch 5/5
500/500 [==============================] - 94s 187ms/step - loss: 0.6178 - accuracy: 0.6727 - val_loss: 0.5946 - val_accuracy: 0.6933
999/999 [==============================] - 36s 36ms/step - loss: 0.5926 - accuracy: 0.6951
```



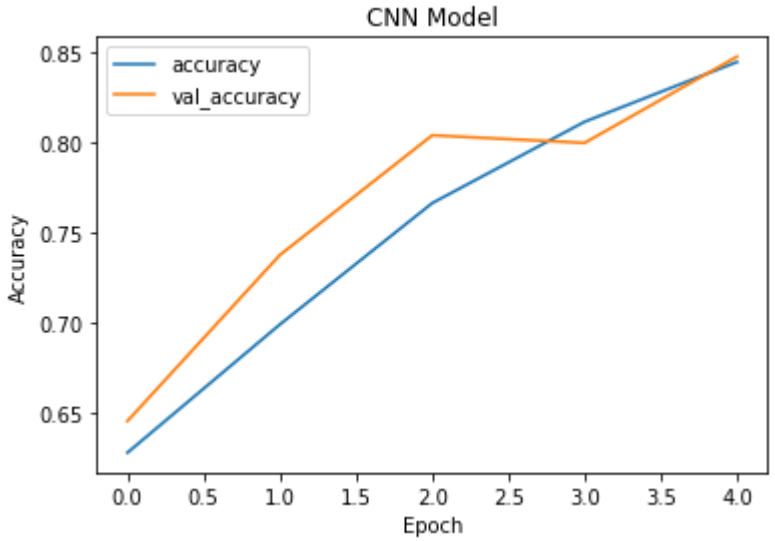
Train CNN Model.

```
cnn_model = models.Sequential(name = 'CNN_MODEL')
cnn_model.add(layers.Embedding(input_dim= max_voc_len, output_dim= 128, input_length=max_feature, trainable=False))
cnn_model.add(layers.Reshape((max_feature, 128, 1)))
cnn_model.add(layers.Conv2D(128, (3,3), activation='relu'))
cnn_model.add(layers.MaxPooling2D(2,2))
cnn_model.add(layers.Conv2D(128, (3,3), activation='relu'))
cnn_model.add(layers.MaxPooling2D(2,2))
cnn_model.add(layers.Flatten())
cnn_model.add(layers.Dense(128, activation='relu'))
cnn_model.add(layers.Dense(64, activation='relu'))
cnn_model.add(layers.Dense(1, activation='sigmoid'))

adam_low_rate = optimizers.Adam(learning_rate=1.e-5)

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

```
Model: "CNN_MODEL"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 246, 128)          1280000

 reshape_1 (Reshape)         (None, 246, 128, 1)       0

 conv2d_2 (Conv2D)           (None, 244, 126, 128)     1280

 max_pooling2d_2 (MaxPooling  (None, 122, 63, 128)     0
 2D)

 flatten_4 (Flatten)         (None, 983808)            0

 dense_19 (Dense)            (None, 128)               125927552

 dense_20 (Dense)            (None, 64)                8256

 dense_21 (Dense)            (None, 1)                 65

=================================================================
Total params: 127,217,153
Trainable params: 125,937,153
Non-trainable params: 1,280,000
_________________________________________________________________
```

```
cnn_model_history = cnn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=64)
test_loss, test_acc = cnn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 2215s 4s/step - loss: 0.6492 - accuracy: 0.6276 - val_loss: 0.6187 - val_accuracy: 0.6452
Epoch 2/5
500/500 [==============================] - 2287s 5s/step - loss: 0.5932 - accuracy: 0.6989 - val_loss: 0.5712 - val_accuracy: 0.7374
Epoch 3/5
500/500 [==============================] - 2480s 5s/step - loss: 0.5388 - accuracy: 0.7663 - val_loss: 0.5196 - val_accuracy: 0.8040
Epoch 4/5
500/500 [==============================] - 2653s 5s/step - loss: 0.4842 - accuracy: 0.8115 - val_loss: 0.4820 - val_accuracy: 0.7998
Epoch 5/5
500/500 [==============================] - 2273s 5s/step - loss: 0.4335 - accuracy: 0.8447 - val_loss: 0.4230 - val_accuracy: 0.8477
999/999 [==============================] - 448s 448ms/step - loss: 0.4010 - accuracy: 0.8728
```



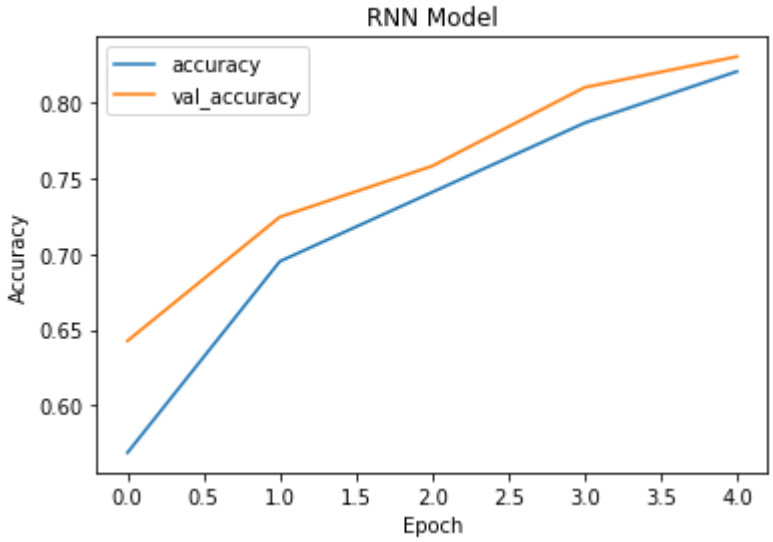
Train RNN Model

```
rnn_model = models.Sequential(name = 'RNN_MODEL')
rnn_model.add(layers.Embedding(input_dim= max_voc_len, output_dim= 128, input_length=max_feature, trainable=False))
rnn_model.add(layers.SimpleRNN(128, activation='relu'))
rnn_model.add(layers.Dense(128, activation='relu'))
rnn_model.add(layers.Dense(64, activation='relu'))
rnn_model.add(layers.Dense(32, activation='relu'))
rnn_model.add(layers.Dense(1, activation='sigmoid'))

adam_low_rate = optimizers.Adam(learning_rate=1.e-5)

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

```
Model: "RNN_MODEL"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 246, 128)          1280000

 simple_rnn (SimpleRNN)      (None, 128)               32896

 dense_8 (Dense)             (None, 128)               16512

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (Dense)            (None, 32)                2080

 dense_11 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,339,777
Trainable params: 59,777
Non-trainable params: 1,280,000
_________________________________________________________________
```

```
rnn_model_history = rnn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=64)
test_loss, test_acc = rnn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 78s 153ms/step - loss: 0.6913 - accuracy: 0.5688 - val_loss: 0.6881 - val_accuracy: 0.6427
Epoch 2/5
500/500 [==============================] - 73s 146ms/step - loss: 0.6779 - accuracy: 0.6953 - val_loss: 0.6550 - val_accuracy: 0.7246
Epoch 3/5
500/500 [==============================] - 74s 147ms/step - loss: 0.5941 - accuracy: 0.7410 - val_loss: 0.5365 - val_accuracy: 0.7583
Epoch 4/5
500/500 [==============================] - 75s 151ms/step - loss: 0.4849 - accuracy: 0.7868 - val_loss: 0.4288 - val_accuracy: 0.8103
Epoch 5/5
500/500 [==============================] - 75s 150ms/step - loss: 0.4163 - accuracy: 0.8208 - val_loss: 0.3941 - val_accuracy: 0.8306
999/999 [==============================] - 28s 28ms/step - loss: 0.3861 - accuracy: 0.8413
```



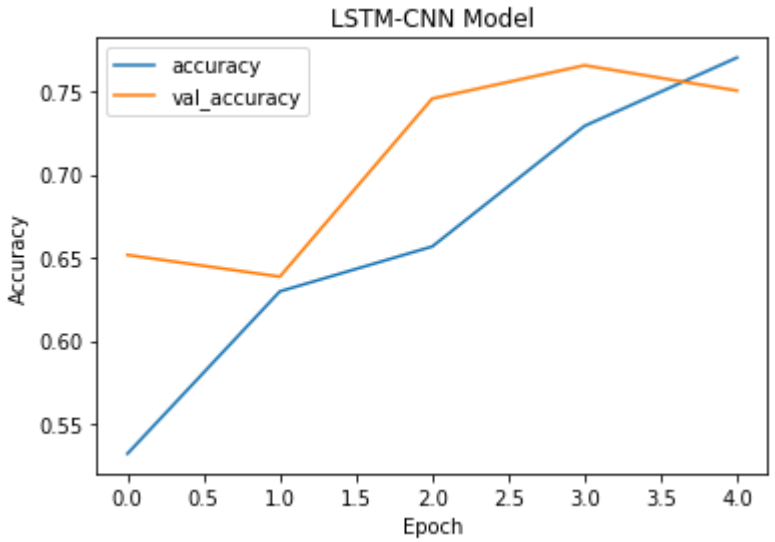
Train LSTM CNN Model.

```
lstm_cnn_model = models.Sequential(name = 'LSTM_CNN_MODEL')
lstm_cnn_model.add(layers.Embedding(input_dim= max_voc_len, output_dim= 128, input_length=max_feature, trainable=False))
lstm_cnn_model.add(Dropout(0.2))
lstm_cnn_model.add(layers.Conv1D(128, 5, activation='relu'))
lstm_cnn_model.add(layers.MaxPooling1D(3))
lstm_cnn_model.add(layers.Conv1D(64, 5, activation='relu'))
lstm_cnn_model.add(layers.MaxPooling1D(3))
lstm_cnn_model.add(layers.LSTM(128, activation='relu', dropout=0.2, recurrent_dropout=0.2))
lstm_cnn_model.add(layers.Flatten())
lstm_cnn_model.add(layers.Dense(128, activation='relu'))
lstm_cnn_model.add(layers.Dense(64, activation='relu'))
lstm_cnn_model.add(layers.Dense(32, activation='relu'))
lstm_cnn_model.add(layers.Dense(1, activation='sigmoid'))

adam_low_rate = optimizers.Adam(learning_rate=1.e-5)

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

```
Model: "LSTM_CNN_MODEL"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 246, 128)          1280000

 dropout_11 (Dropout)        (None, 246, 128)          0

 conv1d_4 (Conv1D)           (None, 242, 128)          82048

 max_pooling1d_4 (MaxPooling  (None, 80, 128)          0
 1D)

 conv1d_5 (Conv1D)           (None, 76, 64)            41024

 max_pooling1d_5 (MaxPooling  (None, 25, 64)           0
 1D)

 lstm_7 (LSTM)               (None, 128)               98816

 flatten_5 (Flatten)         (None, 128)               0

 dense_21 (Dense)            (None, 128)               16512

 dense_22 (Dense)            (None, 64)                8256

 dense_23 (Dense)            (None, 32)                2080

 dense_24 (Dense)            (None, 1)                 33
```

```
lstm_cnn_model_history = lstm_cnn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=64)
test_loss, test_acc = lstm_cnn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 172s 335ms/step - loss: 0.6927 - accuracy: 0.5322 - val_loss: 0.6920 - val_accuracy: 0.6516
Epoch 2/5
500/500 [==============================] - 144s 289ms/step - loss: 0.6892 - accuracy: 0.6297 - val_loss: 0.6843 - val_accuracy: 0.6385
Epoch 3/5
500/500 [==============================] - 143s 287ms/step - loss: 0.6545 - accuracy: 0.6567 - val_loss: 0.6360 - val_accuracy: 0.7455
Epoch 4/5
500/500 [==============================] - 143s 287ms/step - loss: 0.6128 - accuracy: 0.7291 - val_loss: 0.5936 - val_accuracy: 0.7655
Epoch 5/5
500/500 [==============================] - 143s 286ms/step - loss: 0.5088 - accuracy: 0.7701 - val_loss: 0.4973 - val_accuracy: 0.7503
999/999 [==============================] - 42s 42ms/step - loss: 0.4940 - accuracy: 0.7547
```



Upload a public dataset for testing feasibility of the model.

```
df_test = pd.read_csv('/content/gdrive/MyDrive/fake_or_real_news.csv')
df_test.head()
```

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

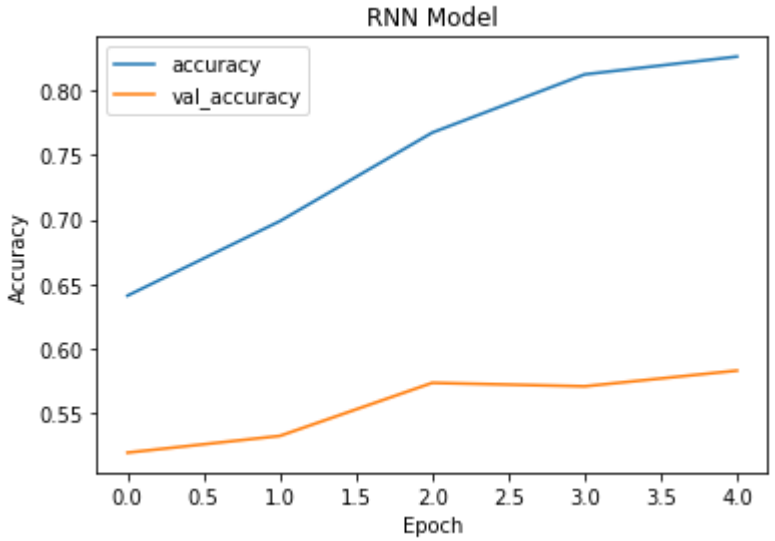
```
df_test['text'] = df_test['title'] + "" + df_test['text']
del df_test['title']

df_test['text'] = df_test['text'].apply(denoise_text)
```

```
#x_train, x_test, y_train, y_test = train_test_split(df_pre['text'].to_numpy(), df_pre['label'].to_numpy(), test_size = 0.25)

X_test2 = tokenizer.texts_to_sequences(texts = df_test['text'].to_numpy())
X_test2 = sequence.pad_sequences(X_test2, maxlen=max_feature)

y_test2 = df_test['label'].to_numpy()
```

Test model with a public dataset.

```
#test dataset on RNN Model
test_rnn = rnn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test2, y_test2), batch_size=64)
test_loss, test_acc = rnn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 88s 173ms/step - loss: 0.6893 - accuracy: 0.6409 - val_loss: 0.6924 - val_accuracy: 0.5189
Epoch 2/5
500/500 [==============================] - 75s 150ms/step - loss: 0.6693 - accuracy: 0.6988 - val_loss: 0.6916 - val_accuracy: 0.5319
Epoch 3/5
500/500 [==============================] - 75s 150ms/step - loss: 0.5392 - accuracy: 0.7676 - val_loss: 0.7127 - val_accuracy: 0.5731
Epoch 4/5
500/500 [==============================] - 74s 148ms/step - loss: 0.4352 - accuracy: 0.8128 - val_loss: 0.7337 - val_accuracy: 0.5704
Epoch 5/5
500/500 [==============================] - 84s 167ms/step - loss: 0.3992 - accuracy: 0.8266 - val_loss: 0.7509 - val_accuracy: 0.5826
999/999 [==============================] - 33s 32ms/step - loss: 0.3828 - accuracy: 0.8317
```



```
#test dataset on lstm_cnn model
test_lstm_cnn = lstm_cnn_model.fit(X_train, y_train, epochs=5, validation_data=(X_test2, y_test2), batch_size=64)
test_loss, test_acc = lstm_cnn_model.evaluate(X_train, y_train)
```

```
Epoch 1/5
500/500 [==============================] - 155s 296ms/step - loss: 0.6927 - accuracy: 0.5065 - val_loss: 0.6931 - val_accuracy: 0.5014
Epoch 2/5
500/500 [==============================] - 151s 303ms/step - loss: 0.6888 - accuracy: 0.6069 - val_loss: 0.6934 - val_accuracy: 0.5182
Epoch 3/5
500/500 [==============================] - 149s 298ms/step - loss: 0.6203 - accuracy: 0.7352 - val_loss: 0.7159 - val_accuracy: 0.5768
Epoch 4/5
500/500 [==============================] - 149s 299ms/step - loss: 0.4317 - accuracy: 0.8064 - val_loss: 0.7859 - val_accuracy: 0.5655
Epoch 5/5
500/500 [==============================] - 166s 331ms/step - loss: 0.3850 - accuracy: 0.8334 - val_loss: 0.7989 - val_accuracy: 0.5810
999/999 [==============================] - 48s 47ms/step - loss: 0.3530 - accuracy: 0.8764
```