In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import re
import nltk
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from sklearn.metrics import accuracy_score
```

In [2]:

```
nltk.download('omw-1.4')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\13486\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\13486\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\13486\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\13486\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\13486\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
# for reproducibility
seed0=1337
np.random.seed(seed0)
tf.keras.utils.set_random_seed(1)
tf.random.set_seed(seed0)
pd.set_option('display.max_colwidth', None)
```

In [4]:
```
df = pd.read_csv("Reviews_10000.csv")
df.head(10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	531660	B005K4Q34S	A3EHFQ2ZL4ALA	J. Stallworth	2	

In [5]:
```
df_processed = df[['Text','Score']]
df_processed.head()
```

Out[5]:

	Text	Score
0	I just tried this product for the first time this morning and was eager to do so. Boy, was I disappointed and disgusted! I tried with all my might to get through a fourth of the cup but I simply could not do it. This so-called cappuccino tasted greasy and overly sweet at the same time. I did not even know that was possible for a caffeine product. I would have rather had gas station cappuccino than this garbage. I wish there were a way for me to return the 22 K-Cups that are left but, sadly, they will just end up in the trash. What a waste of my hard-earned money.	1
1	The description of this item is utterly misleading. The Web page lists the shipping weight at 1 lb, yet what one receives for $21 is 7 ounces of cookies. This is misrepresentation and a ripoff. The merchant did not respond to my complaint.	1
2	I was so disappointed that I threw them all out. I don't know why it tastes so good at the stadium...is it because it stands out a long time?	1
3	This is my fist batch from amazon of the salmon and chicken formula. Purchase 3 cases from another pet food store and our cat loved it. When we opened this new one from Amazon, it didn't smell the same as the last batch, this one smelled like rotten fish and the last one smelled good. We have tried to give this new one to our cat and he refused to eat it (when he would beg for the other one and finish it promptly). I inspected both a can from this new batch and a can from the old batch and everything is the same including all ingredients. The only difference I could find was the older batch had a 2014 date on the bottom printed a little differently from the 2015 best by date the the new one....? I don't understand why this product has such inconsistent quality or maybe Newman's is not making the same quality product as they used to. Too bad since I would like to find a organic cat food that is not super expensive cause I'm not going to buy this one again.	1
4	when i received this product it was already out of date. the product already 2 month that passed its shelf life. Its shame to sell out of date products.	1

In [6]:
```
df_processed[df_processed['Score']==1].count()
```

Out[6]:
```
Text     5000
Score    5000
dtype: int64
```

In [7]:

```
# Help funciton to classify based on score
def score_converter(score):
    if score <= 2:
        return 'unsatisfied'
    elif score>=4:
        return 'satisfied'

# Helper function to clean the text
def remove_tags(string):
    result =re.sub(r'<br\s*/?>', '', string)
    result = re.sub('https://.*','',result)   #remove URLs
    result = re.sub('[^a-zA-Z0-9 ]', '', result)    #remove non-alphanumeric characte
    result = result.lower()
    return result

df_processed['Category'] = df_processed['Score'].apply(score_converter)
df_processed['Text'] = df_processed['Text'].apply(remove_tags)

df_processed = df_processed.sample(frac = 1)
```

```
C:\Users\13486\AppData\Local\Temp\ipykernel_34800\3730055545.py:18: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_processed['Category'] = df_processed['Score'].apply(score_converter)
C:\Users\13486\AppData\Local\Temp\ipykernel_34800\3730055545.py:19: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_processed['Text'] = df_processed['Text'].apply(remove_tags)
```

In [8]:

```
# remove the stop word to increase model efficiency

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
df_processed['Text'] = df_processed['Text'].apply(lambda x: ' '.join([word for word
```

In [9]:

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import pos_tag
from nltk.tokenize import word_tokenize


def get_wordnet_pos(treebank_tag):
    """Converts treebank tags to WordNet tags."""
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    # Tokenize the text into words
    words = word_tokenize(text)
    # Get part-of-speech tags for each word
    pos_tags = pos_tag(words)

    lemmatized_words = []
    for word, tag in pos_tags:
        # Convert part-of-speech tag to a format recognized by WordNetLemmatizer
        wntag = get_wordnet_pos(tag)
        if wntag is None:
            # If the tag is not recognized, keep the word as is
            lemmatized_words.append(word)
        else:
            # Lemmatize the word with the appropriate part of speech tag
            lemmatized_words.append(lemmatizer.lemmatize(word, pos=wntag))

    # Return the lemmatized words as a single string
    return ' '.join(lemmatized_words)
df_processed['Text'] = df_processed['Text'].apply(lemmatize_text)
```

In [10]:

```
# df_processed = pd.read_csv("processed.csv")
df_processed  = df_processed.dropna()
```

In [11]:

```
df_processed["Category"].unique()
```

Out[11]:

```
array(['satisfied', 'unsatisfied'], dtype=object)
```

In [12]:

```
s = 0.0
for i in df_processed['Text']:
    word_list = i.split()
    s = s + len(word_list)
print("Average length of each review : ",s/df_processed.shape[0])
pos = 0
neg = 0
for i in range(df_processed.shape[0]):

    if df_processed.iloc[i]['Category'] == 'satisfied':
        pos = pos + 1
    elif df_processed.iloc[i]['Category'] == 'unsatisfied':
        neg+=1
print("Percentage of reviews with positive sentiment is "+str(pos/df_processed.shape[
print("Percentage of reviews with negative sentiment is "+str(neg/df_processed.shape[
```

```
Average length of each review :  43.06955
Percentage of reviews with positive sentiment is 50.0%
Percentage of reviews with negative sentiment is 50.0%
```

In [13]:

```
reviews = df_processed['Text'].values
labels = df_processed['Category'].values
encoder = LabelEncoder()
encoded_labels = encoder.fit_transform(labels)

train_sentences, test_sentences, train_labels, test_labels = train_test_split(reviews
```

In [14]:

```
def plot_acc(his,title):
    plt.plot(his.history['accuracy'], label='Training Accuracy')
    plt.plot(his.history['val_accuracy'], label='Validation Accuracy')
    # add label and tile
    plt.title(title+' Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend()
    plt.show()
```

In [15]:

```
# Hyperparameters of the model
vocab_size = 6000
oov_tok = ''
embedding_dim = 100
max_length = 200
padding_type='post'
trunc_type='post'
# tokenize sentences
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index
# convert train dataset to sequence and pad sequences
train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_padded = pad_sequences(train_sequences, padding='post', maxlen=max_length)
# convert Test dataset to sequence and pad sequences
test_sequences = tokenizer.texts_to_sequences(test_sentences)
test_padded = pad_sequences(test_sequences, padding='post', maxlen=max_length)
```

In [16]:

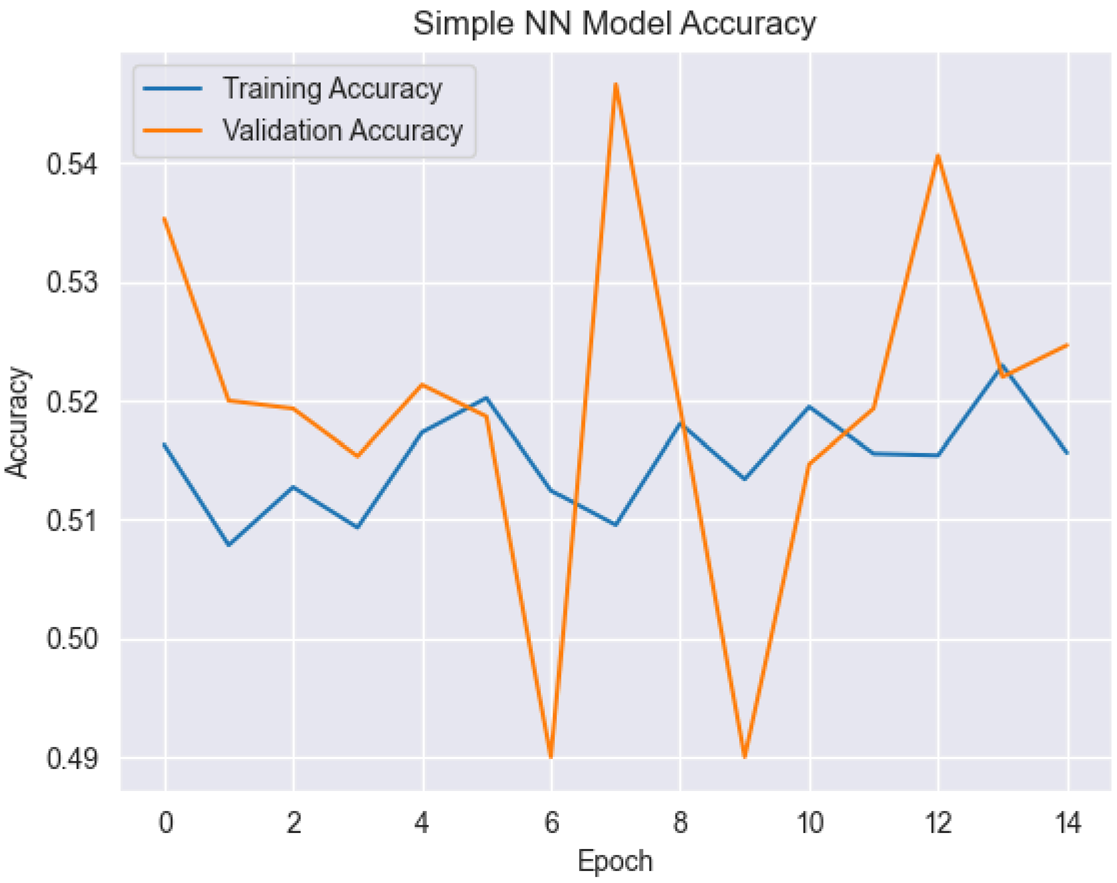
```
# Construct simple NN network
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(3000, 100, input_length=max_length))
model.add(tf.keras.layers.GlobalAveragePooling1D())
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))


optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

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


history = model.fit(train_padded, train_labels,
                    epochs=15, verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 2s 3ms/step - loss: 0.6928 - accuracy:
0.5163 - val_loss: 0.6927 - val_accuracy: 0.5353
Epoch 2/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6931 - accuracy:
0.5079 - val_loss: 0.6922 - val_accuracy: 0.5200
Epoch 3/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6930 - accuracy:
0.5127 - val_loss: 0.6920 - val_accuracy: 0.5193
Epoch 4/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6929 - accuracy:
0.5093 - val_loss: 0.6918 - val_accuracy: 0.5153
Epoch 5/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6925 - accuracy:
0.5173 - val_loss: 0.6919 - val_accuracy: 0.5213
Epoch 6/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6924 - accuracy:
0.5202 - val_loss: 0.6916 - val_accuracy: 0.5187
Epoch 7/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6927 - accuracy:
0.5124 - val_loss: 0.6931 - val_accuracy: 0.4900
Epoch 8/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6925 - accuracy:
0.5096 - val_loss: 0.6920 - val_accuracy: 0.5467
Epoch 9/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6923 - accuracy:
0.5181 - val_loss: 0.6926 - val_accuracy: 0.5193
Epoch 10/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6924 - accuracy:
0.5134 - val_loss: 0.6939 - val_accuracy: 0.4900
Epoch 11/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6921 - accuracy:
0.5195 - val_loss: 0.6917 - val_accuracy: 0.5147
Epoch 12/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6924 - accuracy:
0.5156 - val_loss: 0.6912 - val_accuracy: 0.5193
Epoch 13/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6922 - accuracy:
0.5154 - val_loss: 0.6916 - val_accuracy: 0.5407
Epoch 14/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6922 - accuracy:
0.5230 - val_loss: 0.6911 - val_accuracy: 0.5220
Epoch 15/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6925 - accuracy:
0.5156 - val_loss: 0.6910 - val_accuracy: 0.5247
```

In [17]:

```
prediction = model.predict(test_padded)
class_predictions = np.where(prediction > 0.5, 1, 0)

accuracy = accuracy_score(test_labels, class_predictions)
print(f"Model accuracy: {accuracy * 100:.2f}%")

plot_acc(history,"Simple NN")
```

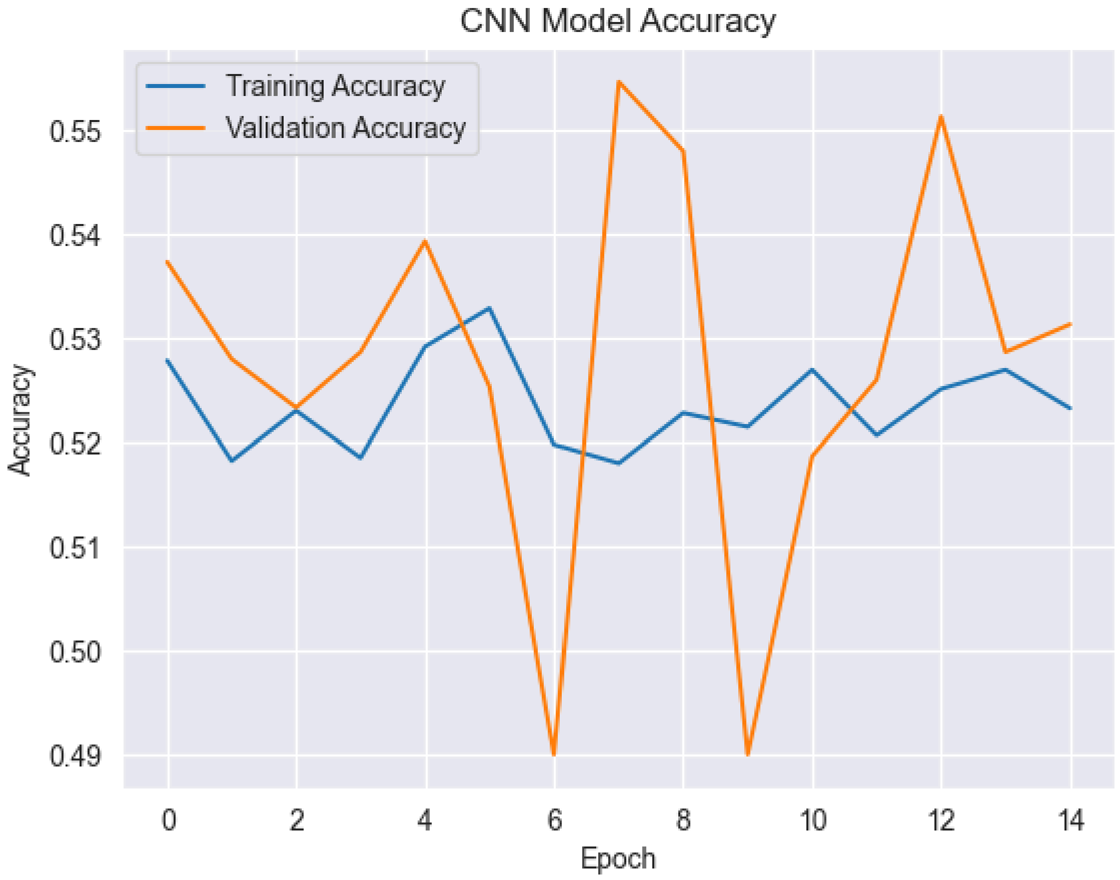
```
157/157 [==============================] - 0s 681us/step
Model accuracy: 51.22%
```



In [18]:

```


model_cnn = tf.keras.Sequential()
model_cnn.add(tf.keras.layers.Embedding(input_dim=3000, output_dim=100, input_length=
model_cnn.add(tf.keras.layers.Conv1D(filters=128, kernel_size=5, activation='relu'))
model_cnn.add(tf.keras.layers.GlobalMaxPooling1D())
model_cnn.add(tf.keras.layers.Dense(10, activation='relu'))
model_cnn.add(tf.keras.layers.Dense(1, activation='sigmoid'))


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

history = model.fit(train_padded, train_labels,
                    epochs=15, verbose=1,
                    validation_split=0.1)




prediction = model_cnn.predict(test_padded)
class_predictions = np.where(prediction > 0.5, 1, 0)

accuracy = accuracy_score(test_labels, class_predictions)
print(f"Model accuracy: {accuracy * 100:.2f}%")

plot_acc(history,"CNN")
```

```
Epoch 1/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6915 - accuracy:
0.5279 - val_loss: 0.6913 - val_accuracy: 0.5373
Epoch 2/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6920 - accuracy:
0.5182 - val_loss: 0.6910 - val_accuracy: 0.5280
Epoch 3/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6921 - accuracy:
0.5230 - val_loss: 0.6909 - val_accuracy: 0.5233
Epoch 4/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6920 - accuracy:
0.5185 - val_loss: 0.6909 - val_accuracy: 0.5287
Epoch 5/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6917 - accuracy:
0.5292 - val_loss: 0.6910 - val_accuracy: 0.5393
Epoch 6/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6915 - accuracy:
0.5329 - val_loss: 0.6907 - val_accuracy: 0.5253
Epoch 7/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6919 - accuracy:
0.5198 - val_loss: 0.6924 - val_accuracy: 0.4900
Epoch 8/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6916 - accuracy:
0.5180 - val_loss: 0.6911 - val_accuracy: 0.5547
Epoch 9/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6914 - accuracy:
0.5228 - val_loss: 0.6916 - val_accuracy: 0.5480
Epoch 10/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6914 - accuracy:
0.5215 - val_loss: 0.6930 - val_accuracy: 0.4900
Epoch 11/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6912 - accuracy:
0.5270 - val_loss: 0.6908 - val_accuracy: 0.5187
Epoch 12/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6915 - accuracy:
0.5207 - val_loss: 0.6902 - val_accuracy: 0.5260
Epoch 13/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6912 - accuracy:
0.5251 - val_loss: 0.6904 - val_accuracy: 0.5513
Epoch 14/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6911 - accuracy:
0.5270 - val_loss: 0.6899 - val_accuracy: 0.5287
Epoch 15/15
422/422 [==============================] - 1s 3ms/step - loss: 0.6916 - accuracy:
0.5233 - val_loss: 0.6898 - val_accuracy: 0.5313
157/157 [==============================] - 0s 887us/step
Model accuracy: 50.64%
```



In [19]:

```
#attention layer

from tensorflow.keras.layers import Layer
import tensorflow.keras.backend as K

class Attention(Layer):
    def __init__(self, return_sequences=True):
        self.return_sequences = return_sequences
        super(Attention, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(name="att_weight", shape=(input_shape[-1], 1),
                                 initializer="normal")
        self.b = self.add_weight(name="att_bias", shape=(input_shape[1], 1),
                                 initializer="zeros")

        super(Attention, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        a = K.softmax(e, axis=1)
        output = x * a

        if self.return_sequences:
            return output

        return K.sum(output, axis=1)
```

In [20]:

```
# model initialization
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    Attention(return_sequences=False),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
# compile model
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.SGD(learning_rate=0.03,momentum=0.8) ,
              metrics=['accuracy'])
# model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 200, 100)          600000    
                                                                 
 bidirectional (Bidirectiona  (None, 200, 128)         84480     
 l)                                                              
                                                                 
 attention (Attention)       (None, 128)               328       
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_3 (Dense)             (None, 32)                4128      
                                                                 
 dense_4 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 688,969
Trainable params: 688,969
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

```
history = model.fit(train_padded, train_labels,
                    epochs=70, verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/70
422/422 [==============================] - 12s 24ms/step - loss: 0.6934 - accura
cy: 0.5113 - val_loss: 0.6930 - val_accuracy: 0.5200
Epoch 2/70
422/422 [==============================] - 10s 23ms/step - loss: 0.6938 - accura
cy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.5133
Epoch 3/70
422/422 [==============================] - 10s 24ms/step - loss: 0.6938 - accura
cy: 0.5019 - val_loss: 0.6931 - val_accuracy: 0.5107
Epoch 4/70
422/422 [==============================] - 10s 23ms/step - loss: 0.6936 - accura
cy: 0.5006 - val_loss: 0.6930 - val_accuracy: 0.5207
Epoch 5/70
422/422 [==============================] - 9s 23ms/step - loss: 0.6935 - accurac
y: 0.5004 - val_loss: 0.6929 - val_accuracy: 0.5420
Epoch 6/70
422/422 [==============================] - 10s 24ms/step - loss: 0.6933 - accura
cy: 0.4995 - val_loss: 0.6926 - val_accuracy: 0.5120
Epoch 7/70
```

In [22]:

```
prediction = model.predict(test_padded)
prediction
```

```
157/157 [==============================] - 2s 8ms/step
```

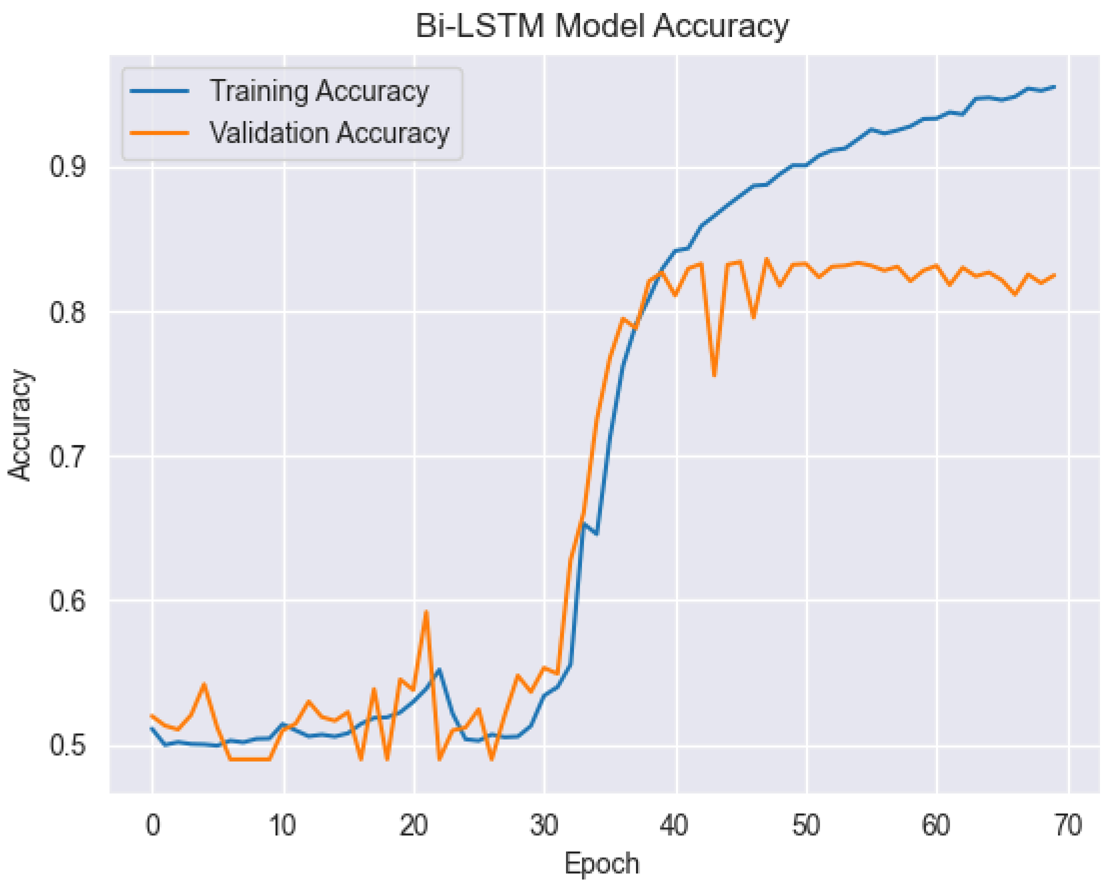
Out[22]:

```
array([[0.00310564],
       [0.9997496 ],
       [0.01250503],
       ...,
       [0.99858105],
       [0.96049744],
       [0.99969983]], dtype=float32)
```

In [23]:

```
class_predictions = np.where(prediction > 0.5, 1, 0)

accuracy = accuracy_score(test_labels, class_predictions)
print(f"Model accuracy: {accuracy * 100:.2f}%")

plot_acc(history, "Bi-LSTM")
```

```
Model accuracy: 82.16%
```



In [24]:

```
from tensorflow.keras import regularizers
from tensorflow.keras.layers import LeakyReLU
```

In [25]:

```
drop_out=0.2
activation=LeakyReLU(alpha = 0.01)
regularizer=regularizers.l2(2e-4)
```

In [26]:

```
model_cb = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Conv1D(filters=128, kernel_size=8,
                           strides=1,
                           activation=activation,
                           padding='causal'),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    Attention(return_sequences=False),
    tf.keras.layers.Dense(32, activation=activation, kernel_regularizer = regularizer
    tf.keras.layers.Dense(1, activation='sigmoid')
])
# compile model
model_cb.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
              metrics=['accuracy'])
# model summary
model_cb.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 200, 100)          600000

 conv1d_1 (Conv1D)           (None, 200, 128)          102528

 bidirectional_1 (Bidirectio  (None, 200, 128)         98816
 nal)

 attention_1 (Attention)     (None, 128)               328

 dense_5 (Dense)             (None, 32)                4128

 dense_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 805,833
Trainable params: 805,833
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

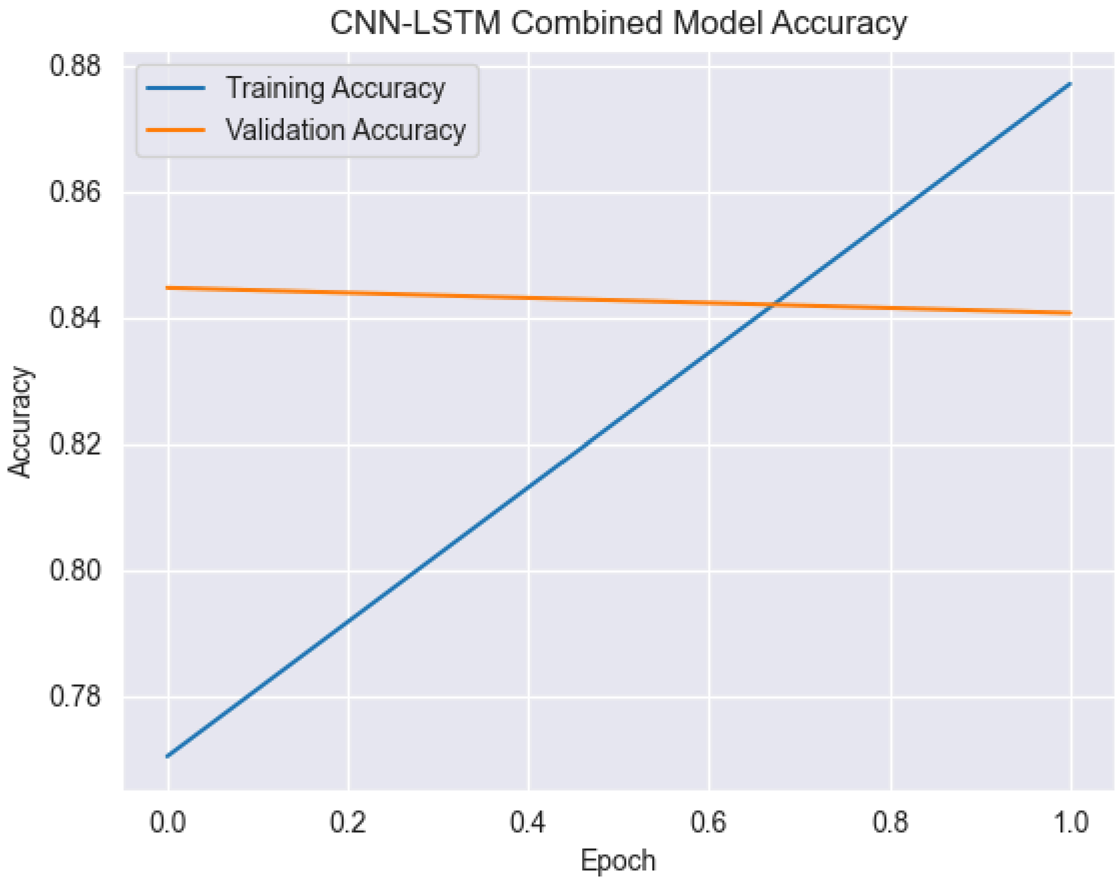
```
history = model_cb.fit(train_padded, train_labels,
                    epochs=2, verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/2
422/422 [==============================] - 14s 29ms/step - loss: 0.4697 - accurac
y: 0.7704 - val_loss: 0.3848 - val_accuracy: 0.8447
Epoch 2/2
422/422 [==============================] - 12s 28ms/step - loss: 0.3051 - accurac
y: 0.8770 - val_loss: 0.3669 - val_accuracy: 0.8407
```

```
In [28]: prediction = model_cb.predict(test_padded)
         class_predictions = np.where(prediction > 0.5, 1, 0)

         accuracy = accuracy_score(test_labels, class_predictions)
         print(f"Model accuracy: {accuracy * 100:.2f}%")
```

```
157/157 [==============================] - 2s 9ms/step
Model accuracy: 84.30%
```

```
In [29]: plot_acc(history,"CNN-LSTM Combined")
```



```
In [29]:
```