```
# Connect to Google Drive
# Upload the dataset to your Google drive so it can be loaded here

from google.colab import drive
drive.mount('/content/gdrive')
```

```
    Mounted at /content/gdrive
```

Loading the required libraries

```
#for data analysis and modeling
import tensorflow as tf
from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, Conv1D, M
from tensorflow.keras.preprocessing import text, sequence
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
#for text cleaning
import string
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
#for visualization
import matplotlib.pyplot as plt
```

Loading data and visualizing

```
true = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/fake-news/True.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
	FBI Russia probe helped by	WASHINGTON (Reuters) - Trump		December

```
fake = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/fake-news/Fake.csv')
fake.head()
```

	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
	Trump Is So Obsessed He Even Has	On Christmas day, Donald Trump		December

Combine fake and true dataframes

Create a new column 'truth' showing whethere the news is fake or real. Then, concatenate two datasets into one dataframe. We can choose to use either 'title' or 'text' column or concatenated 'title+text' for training. But, for the sake of processing time, we'll only use 'title'.

```
true['truth'] = 1
fake['truth'] = 0
df = pd.concat([true, fake], axis=0, ignore_index=True)
df.shape
```

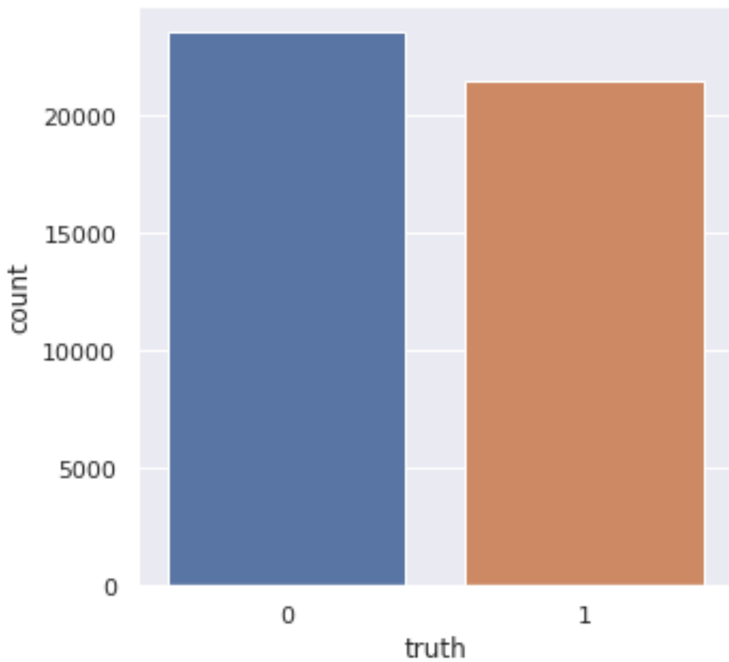
```
(44898, 5)
```

Data Exploration

```
import seaborn as sns

# checking for class imbalance
sns.set(rc={'figure.figsize':(5,5)})
sns.countplot(x='truth', data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f190a6fc750>
```



From the above, it is clear that the dataset is balanced for both fake and real news

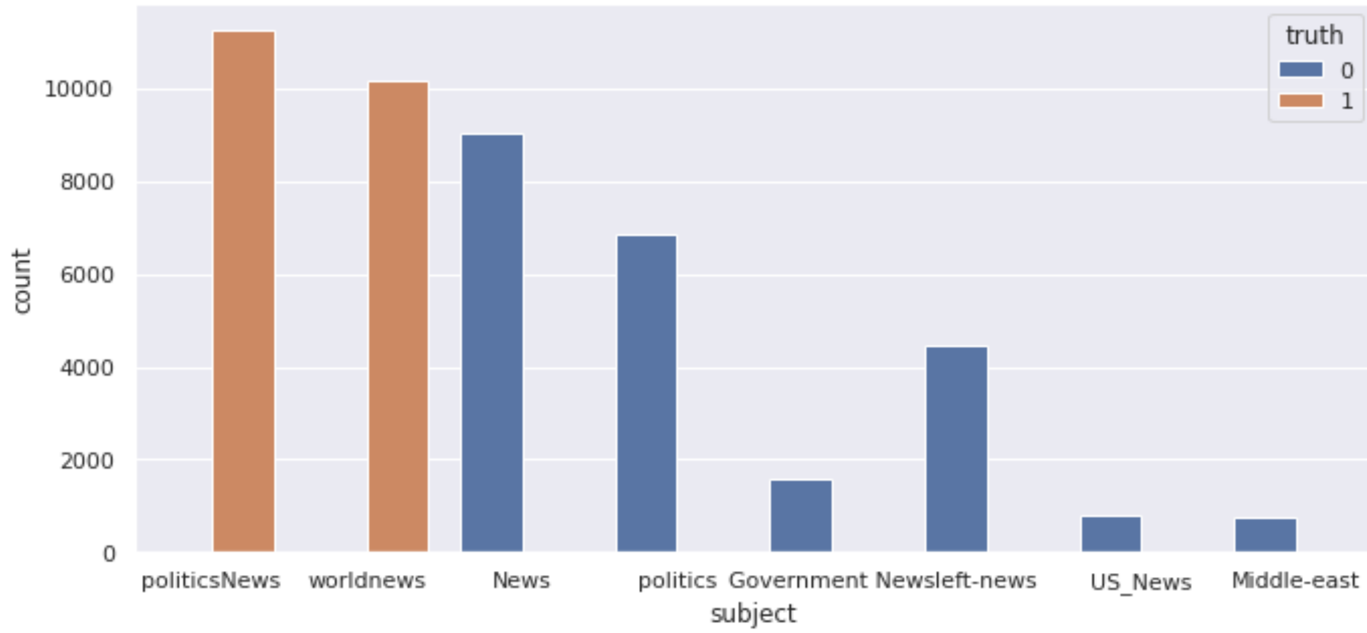
Checking for Relationship between features(subject) and labels(credibility)

Checking for relationship between news subject and news credibility

```
import seaborn as sns

# checking for relationship between credibility and subject
sns.set(rc={'figure.figsize':(11,5)})
sns.countplot(x='subject', data=df, hue='truth')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1909a94d50>
```



- From the plot above, it is clear that real news are only centered around politicNews and worldnews subject areas, while fake news are centered around the other subject areas.
- This indicates that the subject area can help determine if news is fake or real.

Text Cleaning

We need to clean the text first. If you start searching on the text cleaning domain, you realize there are many different techniques. But you may need just a few methods for the purpose of your NLP task.

I looked up the following resources for the text cleaning that I used in this notebook:
https://machinelearningmastery.com/clean-text-machine-learning-python/
https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/

Here are 5 steps that give decent text cleaning result for this task:

1. Replace contractions

In English, a contraction is a word or phrase that has been shortened by dropping one or more letters, such as “I’m” instead of “I am”. We can either split the contractions (“I’m” to “ I ”+” ’m ”) or convert them to their full format (“I’m ” to “I am”). In my experience the latter works better as it's harder to find a word embedding for sub-words like “ ‘m “.

2. Removing punctuation

We want the sentences without punctuations like commas, brackets, etc. Python has a constant called string.punctuation that provides a list of punctuation characters. We’ll use this list to clean our text from punctuations.

3. Splitting into words

In order to remove stopwords, we first need to split the text into words. We do this with word_tokenize function by NLTK. This function splits the text based on white space and punctuation.

4. Removing stopwords

Stopwords are common words like “the”, “and”, … which don’t add much value to the meaning of the text. NLTK has a list of these words that can be imported and used to remove them from the text.

5. Removing leftover punctuations

I noticed after all this cleaning, there were still some words like “…but” with dots in them. I added this last step to clean them up.

Normalizing by case is also common practice. But, since we are using keras tokenizer later, we can skip this step as tokenizer does this step by default. There are other preprocessing techniques of text like Stemming, and Lemmatization. However, in the realm of deep learning NLP they are not necessary anymore.

```
nltk.download('punkt')
nltk.download('stopwords')

def clean_text(txt):
    """
    cleans the input text in the following steps
```

```
    1- replace contractions
    2- removing punctuation
    3- spliting into words
    4- removing stopwords
    5- removing leftover punctuations
    """
    contraction_dict = {"ain't": "is not", "aren't": "are not","can't": "cannot", '
    def _get_contractions(contraction_dict):
        contraction_re = re.compile('(%s)' % '|'.join(contraction_dict.keys()))
        return contraction_dict, contraction_re

    def replace_contractions(text):
        contractions, contractions_re = _get_contractions(contraction_dict)
        def replace(match):
            return contractions[match.group(0)]
        return contractions_re.sub(replace, text)

    # replace contractions
    txt = replace_contractions(txt)

    #remove punctuations
    txt  = "".join([char for char in txt if char not in string.punctuation])
    txt = re.sub('[0-9]+', '', txt)

    # split into words
    words = word_tokenize(txt)

    # remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]

    # removing leftover punctuations
    words = [word for word in words if word.isalpha()]

    cleaned_text = ' '.join(words)
    return cleaned_text

df['data_cleaned'] = df['title'].apply(lambda txt: clean_text(txt))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
df['data_cleaned']
```

```
0        As US budget fight looms Republicans flip fisc...
1        US military accept transgender recruits Monday...
2          Senior US Republican senator Let Mr Mueller job
3        FBI Russia probe helped Australian diplomat ti...
4        Trump wants Postal Service charge much Amazon ...
                                ...
44893    McPain John McCain Furious That Iran Treated U...
44894    JUSTICE Yahoo Settles Email Privacy Classactio...
44895    Sunnistan US Allied Safe Zone Plan Take Territ...
44896    How Blow Million Al Jazeera America Finally Ca...
44897    US Navy Sailors Held Iranian Military Signs Ne...
Name: data_cleaned, Length: 44898, dtype: object
```

Prepare train and test datasets

Use the usual train_test_split by sklearn to split the data.

```
xtrain, xtest, ytrain, ytest = train_test_split(df['data_cleaned'], df['truth'], sh
# find the length of the largest sentence in training data
max_len = xtrain.apply(lambda x: len(x)).max()
print(f'Max number of words in a text in training data: {max_len}')
```

```
Max number of words in a text in training data: 223
```

Tokenize the input training sentences

In most of the NLP tasks, we need to represent each word in the text with an integer value (index) before feeding it to any model. In this way, the text will be converted to a sequence of integer values. One of the ways of doing that is with Keras. Keras provides an API for tokenizing the text. Tokenizer in Keras finds the frequency of each unique word and sort them based on their frequency. It then assigns an integer value starting from 1 to each word from the top. You can see the index mapping dictionary by reading tokenizer.word_index.

There are two distinct steps in tokenizing in this way:

1. fit_on_texts: We'll fit the tokenizer on our training data to create the word indices

2. texts_to_sequences: using the word index dictionary from step above, we take this step to transform both train and test data.

Here, we set num_words to a limit such as 10000 words. num_words is a parameter that defines the maximum number of words to keep, based on the word frequency. Keras actually keeps (num_words-1) words. We can leave the num_words to 'None' and tokenizer will pick all the words in the vacabulary.

Padding and truncating the input training sequences

All your input sequences to the model need to have the same length. In order to achieve that, we can use a function that pads the short sequences with zeros (options are 'pre' or 'post' which pads either before or after each sequence). It also truncates any sequence that is longer than a predefined parameter "maxlen". Truncating also has the option of 'pre' or 'post' which either truncates at the beginning or at the end of the sequences.

```
max_words = 10000
tokenizer = text.Tokenizer(num_words = max_words)
# create the vocabulary by fitting on x_train text
tokenizer.fit_on_texts(xtrain)
# generate the sequence of tokens
xtrain_seq = tokenizer.texts_to_sequences(xtrain)
xtest_seq = tokenizer.texts_to_sequences(xtest)

# pad the sequences
xtrain_pad = sequence.pad_sequences(xtrain_seq, maxlen=max_len)
xtest_pad = sequence.pad_sequences(xtest_seq, maxlen=max_len)
word_index = tokenizer.word_index

print('text example:', xtrain[0])
print('sequence of indices(before padding):', xtrain_seq[0])
print('sequence of indices(after padding):', xtrain_pad[0])
```

```
text example: As US budget fight looms Republicans flip fiscal script
sequence of indices(before padding): [152, 42, 2127, 743, 1377]
sequence of indices(after padding): [   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    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  152   42 2127  743 1377]
```

Word embedding using pre-trained GloVe vectors

Now that we have tokenized the text, we use GloVe pretrained vectors. Word embeddings is a way to represent words with similar meaning to have a similar representation. Using word embedding through GloVe, we can have a decent performance with models with even relatively small label training sets.

You can download GloVe pre-trained word vectors from the link below. There are different sizes of vocabulary and embedding dimension available.

https://nlp.stanford.edu/projects/glove/

Load the GloVe vectors

```
embedding_vectors = {}
with open('/content/gdrive/My Drive/Colab Notebooks/fake-news/glove.6B.100d.txt','r
    for row in file:
        values = row.split(' ')
        word = values[0]
        weights = np.asarray([float(val) for val in values[1:]])
        embedding_vectors[word] = weights
print(f"Size of vocabulary in GloVe: {len(embedding_vectors)}")
```

```
    Size of vocabulary in GloVe: 400000
```

Create an embedding matrix with the GloVe vectors

The embedding matrix has a shape of (vocabulary length, embedding dimension).

Note tht vacab_len is equal to max_words if we set that limit when tokenizing. Otherwise, vocab_len is equal to lenght of all words in word_index+1.

Each row of the embedding matrix belongs to one word in the vocabulary (derived from xtrain) and it contains the weights of embedding vector of that word.

In the code below, we initialze the embedding matrix with zeros. If a word in our word_index is not found in the embedding vectors from GloVe.

The weight of that word remains as zero. Below, you can see print out of the some example words that are out of vacabulary (OOV).

```
#initialize the embedding_matrix with zeros
emb_dim = 100
if max_words is not None:
    vocab_len = max_words
else:
    vocab_len = len(word_index)+1
embedding_matrix = np.zeros((vocab_len, emb_dim))
oov_count = 0
oov_words = []
for word, idx in word_index.items():
    if idx < vocab_len:
        embedding_vector = embedding_vectors.get(word)
        if embedding_vector is not None:
            embedding_matrix[idx] = embedding_vector
        else:
            oov_count += 1
            oov_words.append(word)
#print some of the out of vocabulary words
print(f'Some out of valubulary words: {oov_words[0:5]}')
```

```
Some out of valubulary words: ['brexit', 'antitrump', 'antifa', 'syrias', 're
```

```
print(f'{oov_count} out of {vocab_len} words were OOV.')
```

```
361 out of 10000 words were OOV.
```

Modeling

Now, we can create a model and pre-train the embedding layer with the embedding matrix we just created based on GloVe vectors. You saw the model overview above. The first layer is Embedding. Embedding layer by Keras is a flexible layer that can be used also without any pre-trained weights. In that case, the Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. In this exercise, we will set the weights of the Embedding layer to the embedding matrix from GloVe pre-trained vectors.This is a tranfer learning.

Another parameter in Embedding layer is "trainable" which can be set to True in case you want to fine-tune the word embedding or if you don't want the embedding weights to be updated you can set it to False. Here, we set it to False.

After the Embedding layer, we have a layer of LSTM or GRU and then a Dropout layer for regularization.

Then we have a Dense layer with Sigmoid activation which transforms the output of previous layers to 0 or 1 (real or fake).

LSTM

Let's start with LSTM models.

```
lstm_model = Sequential()
lstm_model.add(Embedding(vocab_len, emb_dim, trainable = False, weights=[embedding_
lstm_model.add(LSTM(128, return_sequences=False))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(1, activation = 'sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
print(lstm_model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         1000000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 1,117,377
Trainable params: 117,377
Non-trainable params: 1,000,000
_________________________________________________________________
None
```

```
batch_size = 256
epochs  = 10
history = lstm_model.fit(xtrain_pad, np.asarray(ytrain), validation_data=(xtest_pac
```

```
Epoch 1/10
141/141 [==============================] - 177s 1s/step - loss: 0.2441 - accu
Epoch 2/10
141/141 [==============================] - 174s 1s/step - loss: 0.0886 - accu
Epoch 3/10
141/141 [==============================] - 172s 1s/step - loss: 0.0751 - accu
Epoch 4/10
141/141 [==============================] - 173s 1s/step - loss: 0.0623 - accu
Epoch 5/10
141/141 [==============================] - 173s 1s/step - loss: 0.0524 - accu
Epoch 6/10
141/141 [==============================] - 173s 1s/step - loss: 0.0407 - accu
Epoch 7/10
141/141 [==============================] - 172s 1s/step - loss: 0.0379 - accu
Epoch 8/10
141/141 [==============================] - 171s 1s/step - loss: 0.0311 - accu
Epoch 9/10
141/141 [==============================] - 173s 1s/step - loss: 0.0271 - accu
Epoch 10/10
141/141 [==============================] - 173s 1s/step - loss: 0.0194 - accu
```

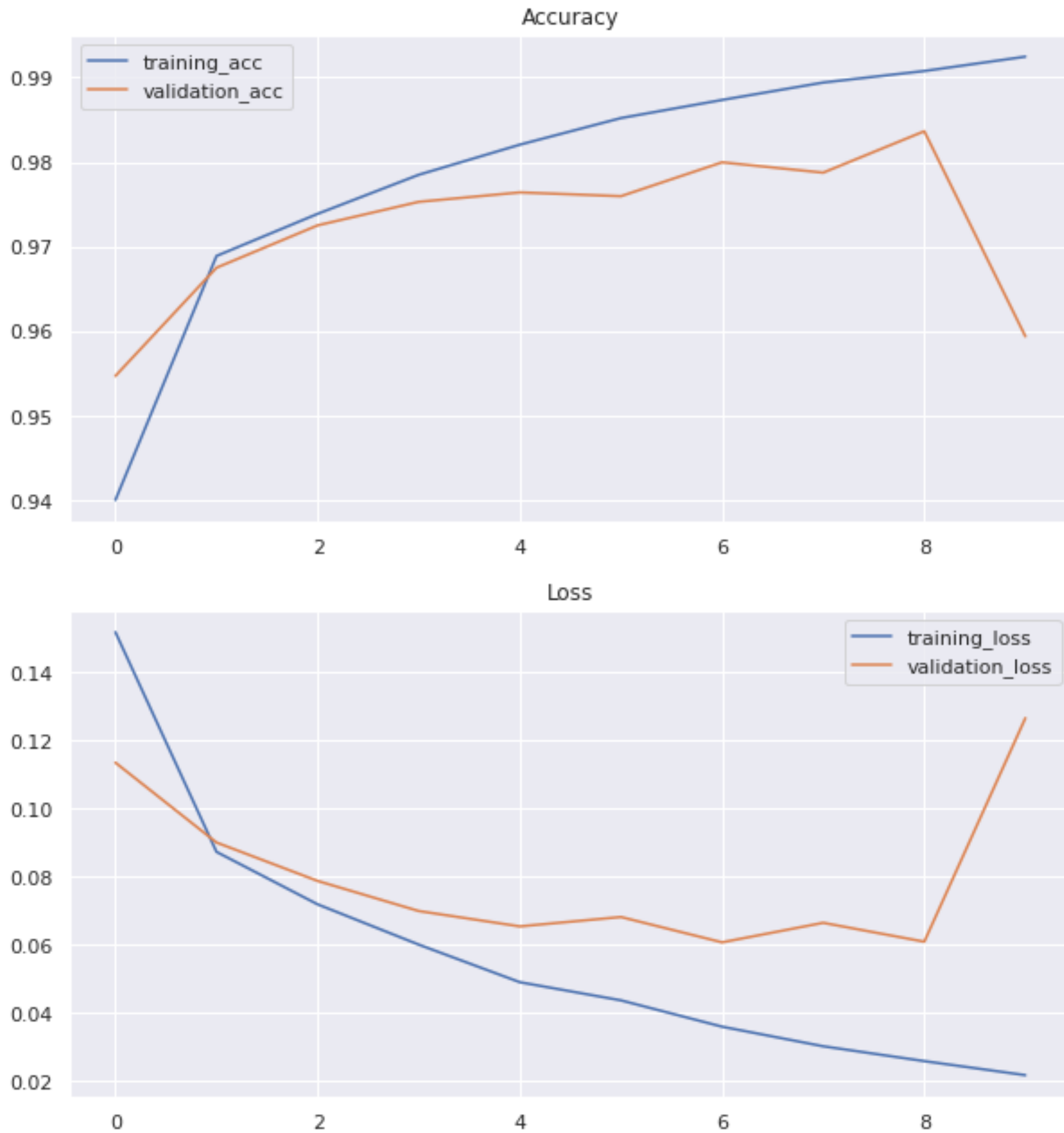
LSTM - Evaluation

Let's find the accuracy of training and testing dataset below:

```
#plot accuracy & loss
plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['accuracy'])
plt.plot(range(epochs), history.history['val_accuracy'])
plt.legend(['training_acc', 'validation_acc'])
plt.title('Accuracy')

plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['loss'])
plt.plot(range(epochs), history.history['val_loss'])
plt.legend(['training_loss', 'validation_loss'])
plt.title('Loss')
```

```
Text(0.5, 1.0, 'Loss')
```




```
train_lstm_results = lstm_model.evaluate(xtrain_pad, np.asarray(ytrain), verbose=0,
test_lstm_results = lstm_model.evaluate(xtest_pad, np.asarray(ytest), verbose=0, ba
print(f'Train accuracy: {train_lstm_results[1]*100:0.2f}')
print(f'Test accuracy: {test_lstm_results[1]*100:0.2f}')
```

```
Train accuracy: 96.93
Test accuracy: 95.94
```

```
y_pred=lstm_model.predict_classes(xtest_pad)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequent:
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(np.asarray(ytest),y_pred)
```

```
array([[4669,   53],
       [ 312, 3946]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(np.asarray(ytest),y_pred)
```

```
0.9593541202672605
```

GRU

```
emb_dim = embedding_matrix.shape[1]
gru_model = Sequential()
gru_model.add(Embedding(vocab_len, emb_dim, trainable = False, weights=[embedding_m
gru_model.add(GRU(128, return_sequences=False))
gru_model.add(Dropout(0.5))
gru_model.add(Dense(1, activation = 'sigmoid'))
gru_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
print(gru_model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         1000000
_________________________________________________________________
gru (GRU)                    (None, 128)               88320
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,088,449
Trainable params: 88,449
Non-trainable params: 1,000,000
_________________________________________________________________
None
```

```
batch_size = 256
epochs  = 10
history = gru_model.fit(xtrain_pad, np.asarray(ytrain), validation_data=(xtest_pad,
```

```
Epoch 1/10
141/141 [==============================] - 149s 1s/step - loss: 0.2919 - accu
Epoch 2/10
141/141 [==============================] - 147s 1s/step - loss: 0.0865 - accu
Epoch 3/10
141/141 [==============================] - 147s 1s/step - loss: 0.0680 - accu
Epoch 4/10
141/141 [==============================] - 147s 1s/step - loss: 0.0550 - accu
Epoch 5/10
141/141 [==============================] - 147s 1s/step - loss: 0.0449 - accu
Epoch 6/10
141/141 [==============================] - 148s 1s/step - loss: 0.0371 - accu
Epoch 7/10
141/141 [==============================] - 148s 1s/step - loss: 0.0310 - accu
Epoch 8/10
141/141 [==============================] - 149s 1s/step - loss: 0.0261 - accu
Epoch 9/10
141/141 [==============================] - 147s 1s/step - loss: 0.0201 - accu
Epoch 10/10
141/141 [==============================] - 150s 1s/step - loss: 0.0168 - accu
```

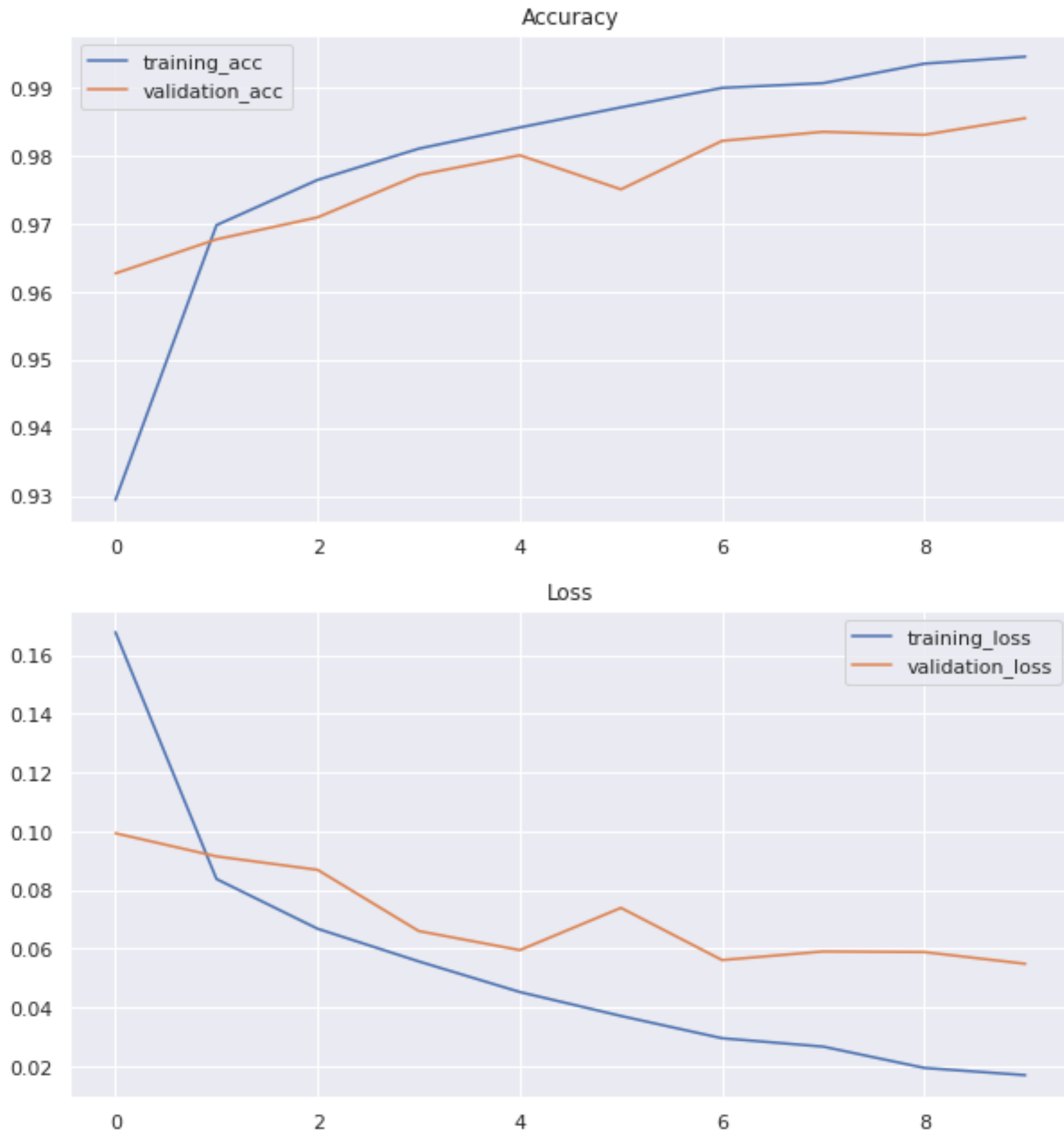
GRU Evaluation

Let's find the accuracy of training and testing dataset with GRU model below:

```
#plot accuracy & loss
plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['accuracy'])
plt.plot(range(epochs), history.history['val_accuracy'])
plt.legend(['training_acc', 'validation_acc'])
plt.title('Accuracy')

plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['loss'])
plt.plot(range(epochs), history.history['val_loss'])
plt.legend(['training_loss', 'validation_loss'])
plt.title('Loss')
```

```
Text(0.5, 1.0, 'Loss')
```




```
train_gru_results = gru_model.evaluate(xtrain_pad, np.asarray(ytrain), verbose=0, b
test_gru_results = gru_model.evaluate(xtest_pad, np.asarray(ytest), verbose=0, batc
print(f'Train accuracy: {train_gru_results[1]*100:0.2f}')
print(f'Test accuracy: {test_gru_results[1]*100:0.2f}')
```

```
Train accuracy: 99.72
Test accuracy: 98.55
```

```
y_pred=gru_model.predict_classes(xtest_pad)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequent:
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(np.asarray(ytest),y_pred)
```

```
array([[4641,   81],
       [  49, 4209]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(np.asarray(ytest),y_pred)
```

```
0.9855233853006682
```

Convolution

```
emb_dim = embedding_matrix.shape[1]
con_model = Sequential()
con_model.add(Embedding(vocab_len, emb_dim, trainable = False, weights=[embedding_m
con_model.add(Dropout(0.2))
con_model.add(Conv1D(128,5,activation='relu'))
con_model.add(MaxPooling1D(pool_size=4))
con_model.add(LSTM(20, return_sequences=True))
con_model.add(LSTM(20))
con_model.add(Dropout(0.2))
con_model.add(Dense(512))
con_model.add(Dropout(0.3))
con_model.add(Dense(256))
con_model.add(Dense(1, activation = 'sigmoid'))
con_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
print(con_model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 100)         1000000
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 100)         0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 128)         64128
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, None, 128)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 20)          11920
_________________________________________________________________
lstm_2 (LSTM)                (None, 20)                3280
_________________________________________________________________
dropout_3 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               10752
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 1,221,665
Trainable params: 221,665
Non-trainable params: 1,000,000
_________________________________________________________________
None
```

```
batch_size = 256
epochs  = 10
history = con_model.fit(xtrain_pad, np.asarray(ytrain), validation_data=(xtest_pad,
```

```
Epoch 1/10
141/141 [==============================] - 80s 540ms/step - loss: 0.2894 - acc
Epoch 2/10
141/141 [==============================] - 76s 536ms/step - loss: 0.1184 - acc
Epoch 3/10
141/141 [==============================] - 76s 539ms/step - loss: 0.0990 - acc
Epoch 4/10
141/141 [==============================] - 77s 546ms/step - loss: 0.0694 - acc
Epoch 5/10
141/141 [==============================] - 76s 539ms/step - loss: 0.0546 - acc
Epoch 6/10
141/141 [==============================] - 76s 537ms/step - loss: 0.0532 - acc
Epoch 7/10
141/141 [==============================] - 75s 534ms/step - loss: 0.0450 - acc
Epoch 8/10
141/141 [==============================] - 76s 539ms/step - loss: 0.0365 - acc
Epoch 9/10
141/141 [==============================] - 78s 550ms/step - loss: 0.0359 - acc
Epoch 10/10
141/141 [==============================] - 76s 541ms/step - loss: 0.0311 - acc
```

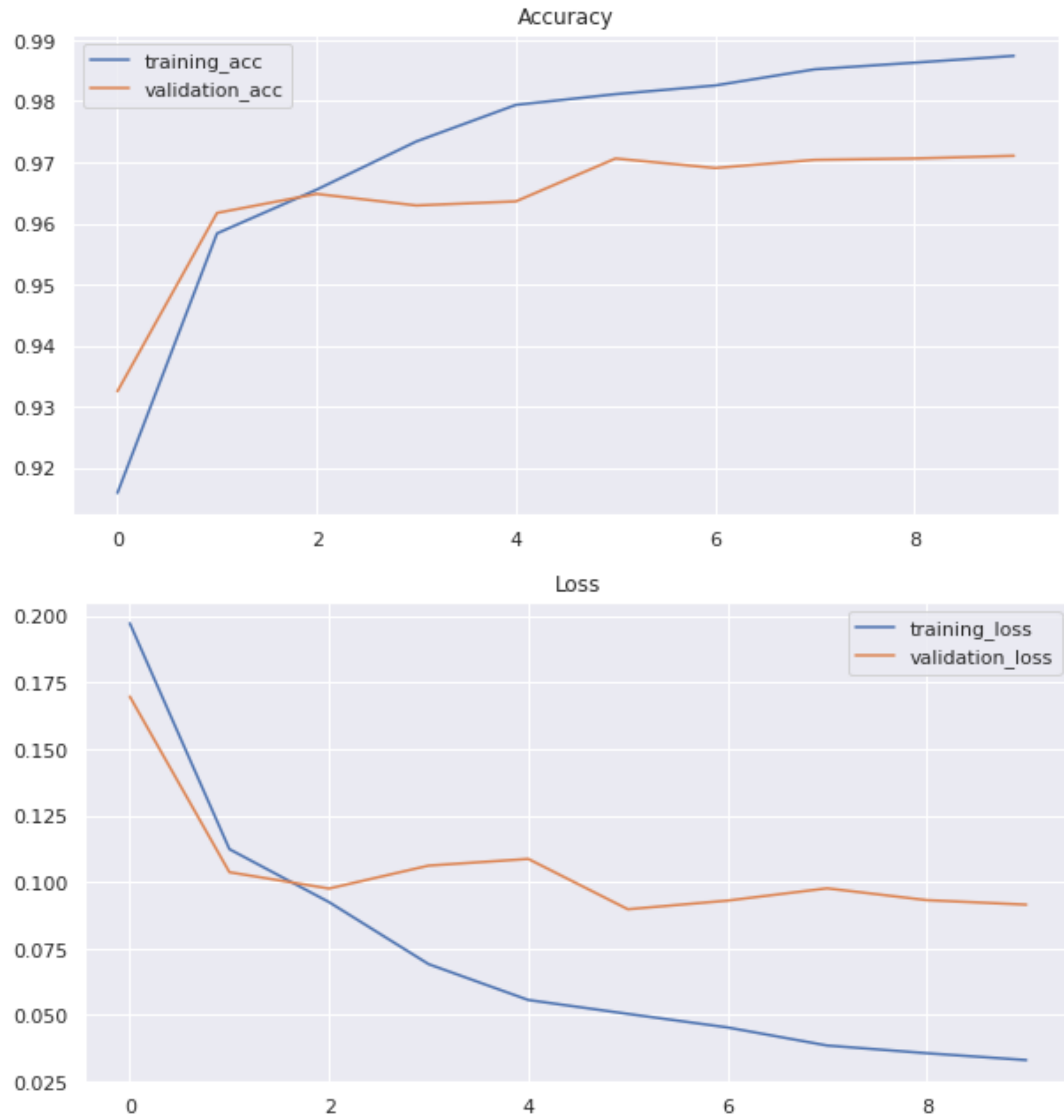
Convolution Model Evaluation

Let's find the accuracy of training and testing dataset with GRU model below:

```
#plot accuracy & loss
plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['accuracy'])
plt.plot(range(epochs), history.history['val_accuracy'])
plt.legend(['training_acc', 'validation_acc'])
plt.title('Accuracy')

plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['loss'])
plt.plot(range(epochs), history.history['val_loss'])
plt.legend(['training_loss', 'validation_loss'])
plt.title('Loss')
```

```
Text(0.5, 1.0, 'Loss')
```




```
train_con_results = con_model.evaluate(xtrain_pad, np.asarray(ytrain), verbose=0, b
test_con_results = con_model.evaluate(xtest_pad, np.asarray(ytest), verbose=0, batc
print(f'Train accuracy: {train_con_results[1]*100:0.2f}')
print(f'Test accuracy: {test_con_results[1]*100:0.2f}')
```

```
Train accuracy: 99.59
Test accuracy: 97.10
```

```
y_pred=con_model.predict_classes(xtest_pad)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequent:
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(np.asarray(ytest),y_pred)
```

```
array([[4573,  149],
       [ 111, 4147]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(np.asarray(ytest),y_pred)
```

```
0.9710467706013363
```

Global Average Pooling

```
emb_dim = embedding_matrix.shape[1]
nn_model = Sequential()
nn_model.add(Embedding(vocab_len, emb_dim, trainable = False, weights=[embedding_ma
nn_model.add(Dropout(0.5))
nn_model.add(GlobalAveragePooling1D())
nn_model.add(Dropout(0.5))
nn_model.add(Dense(16, activation='relu'))
nn_model.add(Dense(1))
nn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']
print(nn_model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 100)         1000000
_________________________________________________________________
dropout_5 (Dropout)          (None, None, 100)         0
_________________________________________________________________
global_average_pooling1d (Gl (None, 100)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 16)                1616
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,001,633
Trainable params: 1,633
Non-trainable params: 1,000,000
_________________________________________________________________
None
```

```
batch_size = 256
epochs  = 10
history = nn_model.fit(xtrain_pad, np.asarray(ytrain), validation_data=(xtest_pad,

    Epoch 1/10
    141/141 [==============================] - 10s 70ms/step - loss: 1.4828 - accu
    Epoch 2/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.7752 - accu
    Epoch 3/10
    141/141 [==============================] - 10s 68ms/step - loss: 0.6392 - accu
    Epoch 4/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.5080 - accu
    Epoch 5/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.3826 - accu
    Epoch 6/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.3141 - accu
    Epoch 7/10
    141/141 [==============================] - 10s 70ms/step - loss: 0.3098 - accu
    Epoch 8/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.3099 - accu
    Epoch 9/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.3192 - accu
    Epoch 10/10
    141/141 [==============================] - 10s 69ms/step - loss: 0.2998 - accu
```

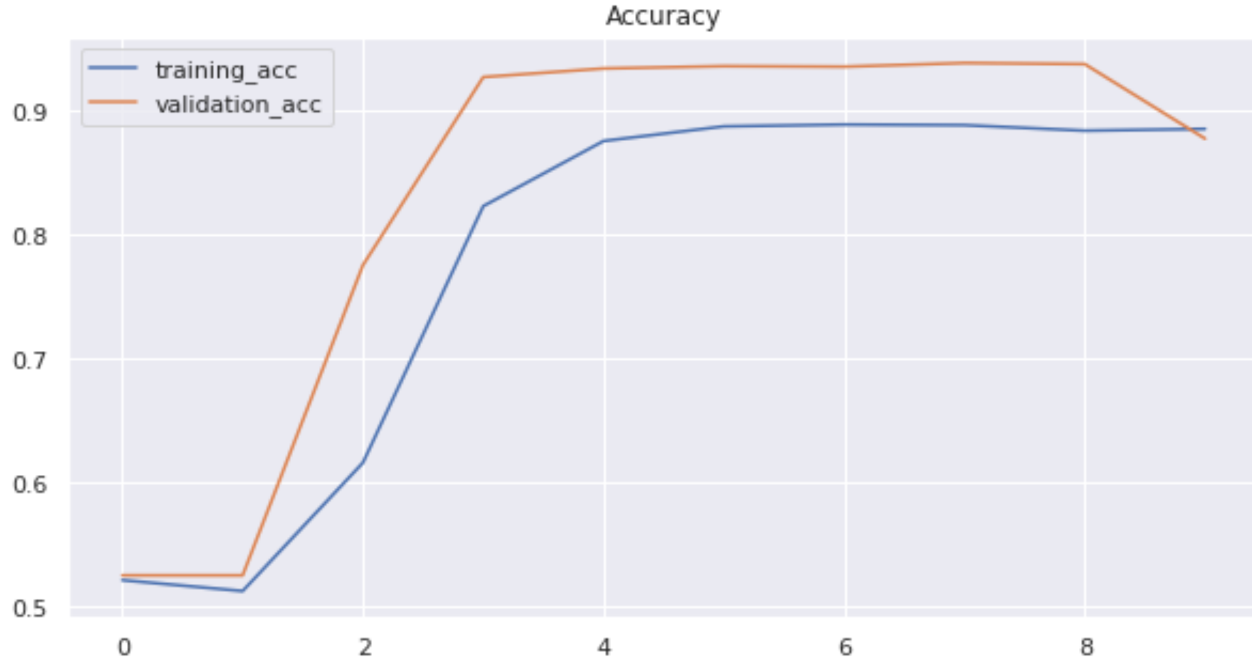
Global Average Pooling Evalution

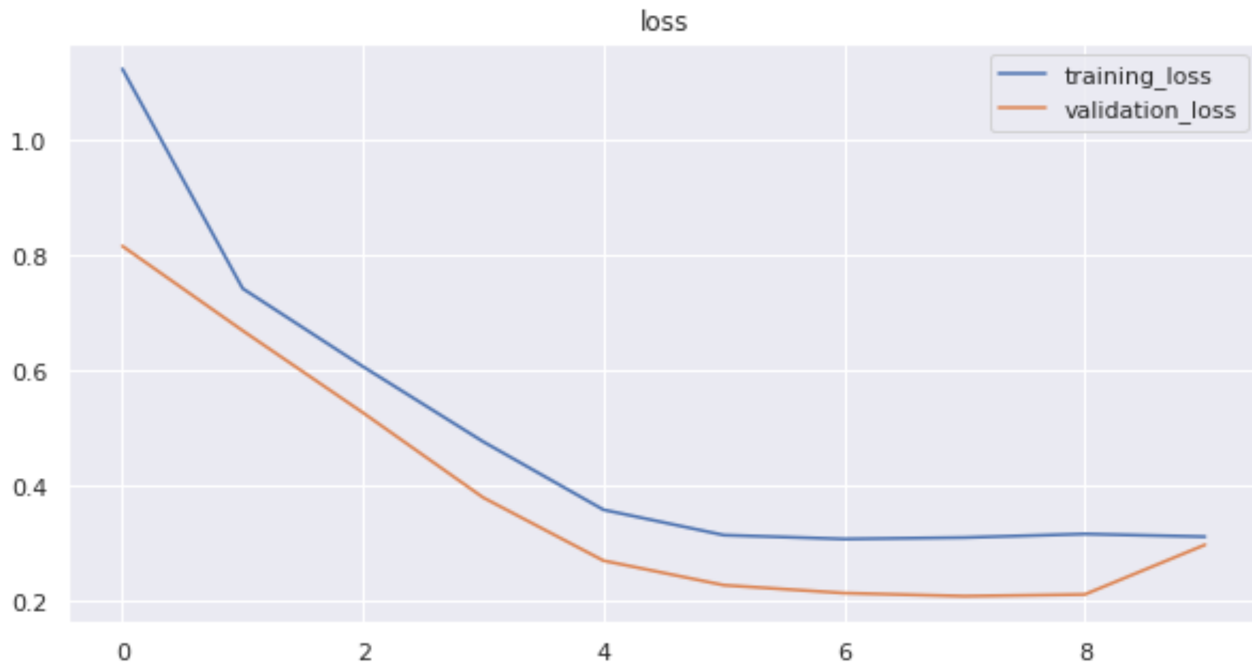
find the accuracy of training and testing dataset with this model below:

```
#plot accuracy & loss
plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['accuracy'])
plt.plot(range(epochs), history.history['val_accuracy'])
plt.legend(['training_acc', 'validation_acc'])
plt.title('Accuracy')

plt.figure(figsize=(10, 5))
plt.plot(range(epochs), history.history['loss'])
plt.plot(range(epochs), history.history['val_loss'])
plt.legend(['training_loss', 'validation_loss'])
plt.title('loss')
```

```
Text(0.5, 1.0, 'loss')
```






```
train_nn_results = nn_model.evaluate(xtrain_pad, np.asarray(ytrain), verbose=0, bat
test_nn_results = nn_model.evaluate(xtest_pad, np.asarray(ytest), verbose=0, batch_
print(f'Train accuracy: {train_nn_results[1]*100:0.2f}')
print(f'Test accuracy: {test_nn_results[1]*100:0.2f}')
```

```
Train accuracy: 87.13
Test accuracy: 87.77
```

```
y_pred=nn_model.predict_classes(xtest_pad)

    /usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequent:
      warnings.warn('`model.predict_classes()` is deprecated and '
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(np.asarray(ytest),y_pred)

    array([[4606,  116],
           [ 982, 3276]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(np.asarray(ytest),y_pred)

    0.877728285077951
```