Fake News Detector

Rashmi R - 19BCS067
Kiruthika P - 19BCS077
Bhavika Arage - 19BCS079
Josphine Cynthia - 19BCS092

Problem Statement

With recent booming of social media, users can get infected by fake news easily, which has brought about tremendous effects on the offline society already. Consumers are creating and sharing more information than ever before, some of which are misleading with no relevance to reality.

We have tried to implement a solution to classify any given news as fake or real using the latest machine learning technique for natural language processing pre-training - BERT (Bidirectional Encoder Representations from Transformers)

Components

We have used

- a virtual gpu,
- google colab and
- bert models.

How we implemented

Cleaning the Dataset

Reading Dataset

In [3]:
```
df=pd.read_csv('data.csv')
df.columns = ['URLs','Headline','Body','Label']
del df['URLs']
del df['Headline']
df = df.fillna('No data')
df.head(10)
```

Out[3]:

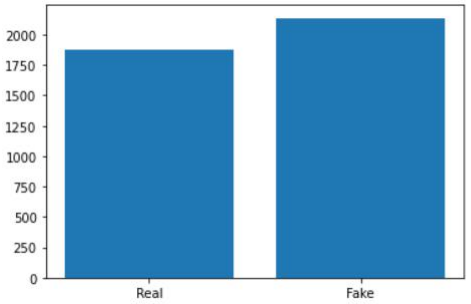
	Body	Label
0	Image copyright Getty Images\nOn Sunday mornin...	1
1	LONDON (Reuters) - "Last Flag Flying", a comed...	1
2	The feud broke into public view last week when...	1
3	MEXICO CITY (Reuters) - Egypt's Cheiron Holdin...	1
4	Country singer Jason Aldean, who was performin...	1
5	JetNation FanDuel League; Week 4\n% of readers...	0
6	In 2012, Kansas lawmakers, led by Gov. Sam Bro...	1
7	The Reserve Bank of India (RBI) Governor Urjit...	1
8	Scott Pruitt, Administrator of the U.S. Enviro...	1
9	FILE PHOTO - An Air Berlin sign is seen at an ...	1

In [4]:
```
df.Label.unique()
```

Out[4]:
```
array([1, 0])
```

In [5]:
```
classes = df.Label.unique()
counts=[]
for i in classes:
  count=len(df[df.Label==i])
  counts.append(count)

plt.bar(["Real","Fake"],counts)
plt.show()
```

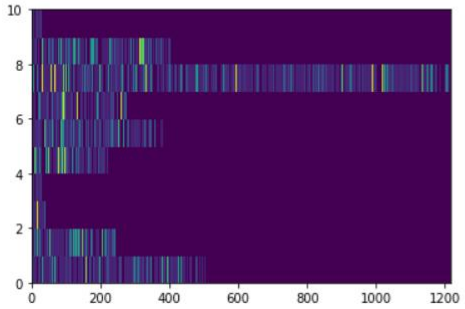


We declare the bert tokenizer and tokenize the text. The 3 input required for a bert model are tokens, masks and input type.

Tokens (into ids)

In [18]:

```
cls = [tokenizer.convert_tokens_to_ids(['[CLS]'])]*articles.shape[0]
input_word_ids = tf.concat([cls, articles], axis=-1)
_ = plt.pcolormesh(input_word_ids[0:10].to_tensor())
```



Function to get the input into the right format for bert model

Function to return the 3 arguments easily

In [23]:
```
max_seq_length=512
```

In [24]:
```
def encode_names(n, tokenizer):
   tokens = list(tokenizer.tokenize(n))
   tokens.append('[SEP]')
   return tokenizer.convert_tokens_to_ids(tokens)

def bert_encode(string_list, tokenizer, max_seq_length):
  num_examples = len(string_list)

  string_tokens = tf.ragged.constant([
      encode_names(n, tokenizer) for n in np.array(string_list)])

  cls = [tokenizer.convert_tokens_to_ids(['[CLS]'])]*string_tokens.shape[0]
  input_word_ids = tf.concat([cls, string_tokens], axis=-1)

  input_mask = tf.ones_like(input_word_ids).to_tensor(shape=(None, max_seq_length))

  type_cls = tf.zeros_like(cls)
  type_tokens = tf.ones_like(string_tokens)
  input_type_ids = tf.concat(
      [type_cls, type_tokens], axis=-1).to_tensor(shape=(None, max_seq_length))

  inputs = {
      'input_word_ids': input_word_ids.to_tensor(shape=(None, max_seq_length)),
      'input_mask': input_mask,
      'input_type_ids': input_type_ids}

  return inputs
```

In [25]:
```
X_train = bert_encode(x_train, tokenizer, max_seq_length)
X_test = bert_encode(x_test, tokenizer, max_seq_length)
```

Training the model and getting the accuracy

In [32]:

```
history = model.fit(X_train,
                    dummy_y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=(X_test, dummy_y_test))
```

```
Epoch 1/3
27/27 [==============================] - 66s 2s/step - loss: 0.5639 - accuracy: 0.6938 - val_loss: 0.3021 - val_accuracy: 0.9500
Epoch 2/3
27/27 [==============================] - 45s 2s/step - loss: 0.1539 - accuracy: 0.9438 - val_loss: 0.2046 - val_accuracy: 0.9500
Epoch 3/3
27/27 [==============================] - 44s 2s/step - loss: 0.0368 - accuracy: 0.9937 - val_loss: 0.1846 - val_accuracy: 0.9250
```

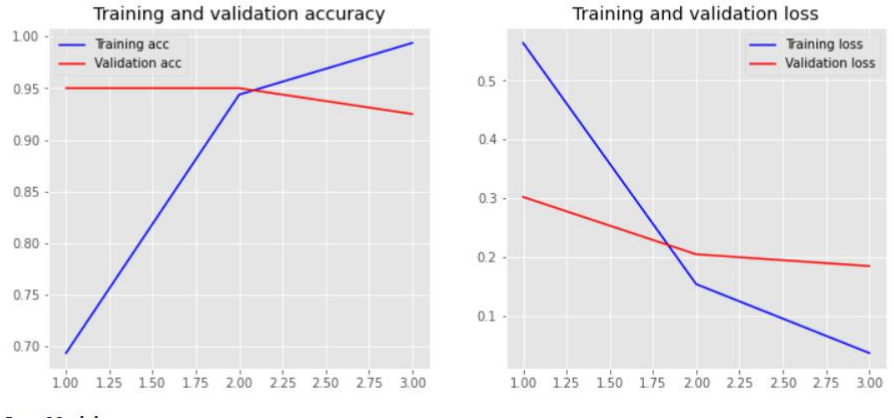
In [33]:

```
loss, accuracy = model.evaluate(X_train, dummy_y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, dummy_y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 1.0000
Testing Accuracy:  0.9250
```

Graph for accuracy

In [35]:
```
plot_history(history)
```



Future Plans

Train it on a different and bigger datasets to get more accurate results.

Try a different bert model.

Thank You!