FINAL REPORT

Deepika Padmanabhan: A20456289
Dimple Mehra : A20457225

FEW SHOT LEARNING

Introduction:
Few-shot learning refers to understanding new concepts from only a few examples. We define a training objective that aims to extract as much information as possible from each training batch by effectively optimizing over all relative orderings of the batch points simultaneously.

This is a challenging setting that necessitates different approaches from the ones commonly employed when the labelled data of each new concept is abundant. Indeed, many recent success stories of machine learning methods rely on large datasets and suffer from overfitting in the face of insufficient data. It is however not realistic nor preferred to always expect many examples for learning a new class or concept, rendering few-shot learning an important problem to address

1 INTRODUCTION:

Fake news and hoaxes have been there since before the advent of the Internet. The widely accepted definition of Internet fake news is: fictitious articles deliberately fabricated to deceive readers”. Social media and news outlets publish fake news to increase readership or as part of psychological warfare.

The purpose of the work is to come up with a solution that can be utilized by users to detect and filter out sites containing false and misleading information. We use carefully selected features to accurately identify fake posts.

Our project aims at classifying the given news articles as fake or true based on the content using few shot learning models with less data. To begin with, we have used pre-trained BERT models as our baseline.

2 PREPROCESSING AND FEATURE ENGINEERING:

Datasets Used:

Smaller Dataset:
https://drive.google.com/file/d/1d10zW73C_Vd5JF5PKBXw9vmjV5MSS6C2/view?usp=sharing
This is the first dataset containing id & content of fake news. We have trained our model for 322 samples.

Larger Dataset :
https://drive.google.com/file/d/1WnoSO6U0RSxxgFIG5Sijnq6C1j2GsKQu/view?usp=sharing

Dataset is loaded and preprocessed as follows:

- The dataset is loaded using pandas library as a dataframe.
- train_test _split package is imported from sklearn library and the data is split into training and test dataframes.
- Data encoding is so as to feed to the BERT (base) layer.
- To encode the data, firstly every text/row in the dataframe is split into tokens, masks and segments.

3 MODEL DESCRIPTION:

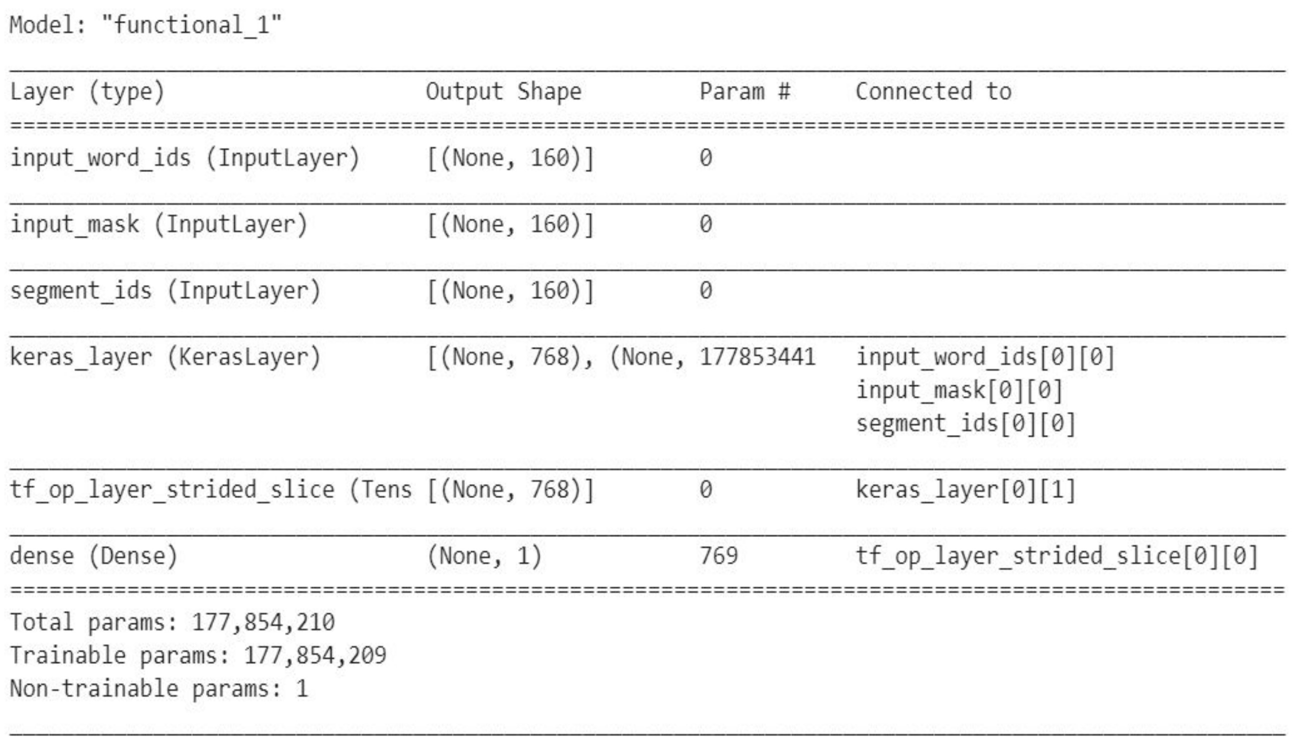
BERT model:

BERT is one of the best easily available and popular language models. Here we have used the base architecture of BERT and there are a total of 12 layers and each of these layer outputs can be used as word embedding.

We have implemented the BERT model by using custom helper functions like bert_encode() and build_model(). The custom encode function splits each sentence into words(tokens) and a [CLS] token is inserted at the beginning of the sentence and [SEP] token is inserted at the end of each sentence. These split tokens are converted to ids using convert_tokens_to_ids() method imported from the tokenization library. The output of this function returns tokens, masks and segments. The length of the segment depends on the max_len parameter that is set to 160 in our case.

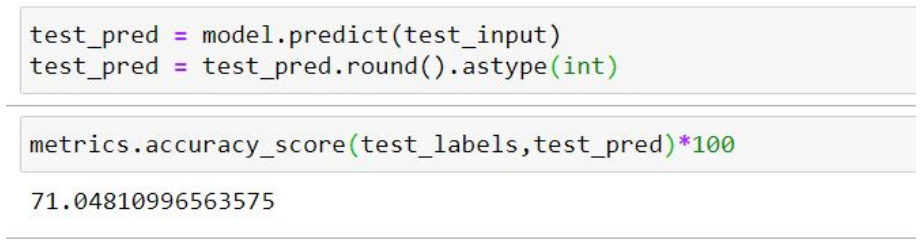
Once the data is encoded , build_model() function is called and trained for ~500 samples. We decreased the labelled data for training and evaluated the performance of the BERT model.

Summary of the BERT model is as follows:



```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)     [(None, 160)]        0
__________________________________________________________________________________________________
input_mask (InputLayer)         [(None, 160)]        0
__________________________________________________________________________________________________
segment_ids (InputLayer)        [(None, 160)]        0
__________________________________________________________________________________________________
keras_layer (KerasLayer)        [(None, 768), (None, 177853441   input_word_ids[0][0]
                                                                 input_mask[0][0]
                                                                 segment_ids[0][0]
__________________________________________________________________________________________________
tf_op_layer_strided_slice (Tens [(None, 768)]        0           keras_layer[0][1]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            769         tf_op_layer_strided_slice[0][0]
==================================================================================================
Total params: 177,854,210
Trainable params: 177,854,209
Non-trainable params: 1
__________________________________________________________________________________________________
```

After training the model , the obtained accuracy is ~71%



```
test_pred = model.predict(test_input)
test_pred = test_pred.round().astype(int)
```

```
metrics.accuracy_score(test_labels,test_pred)*100
```

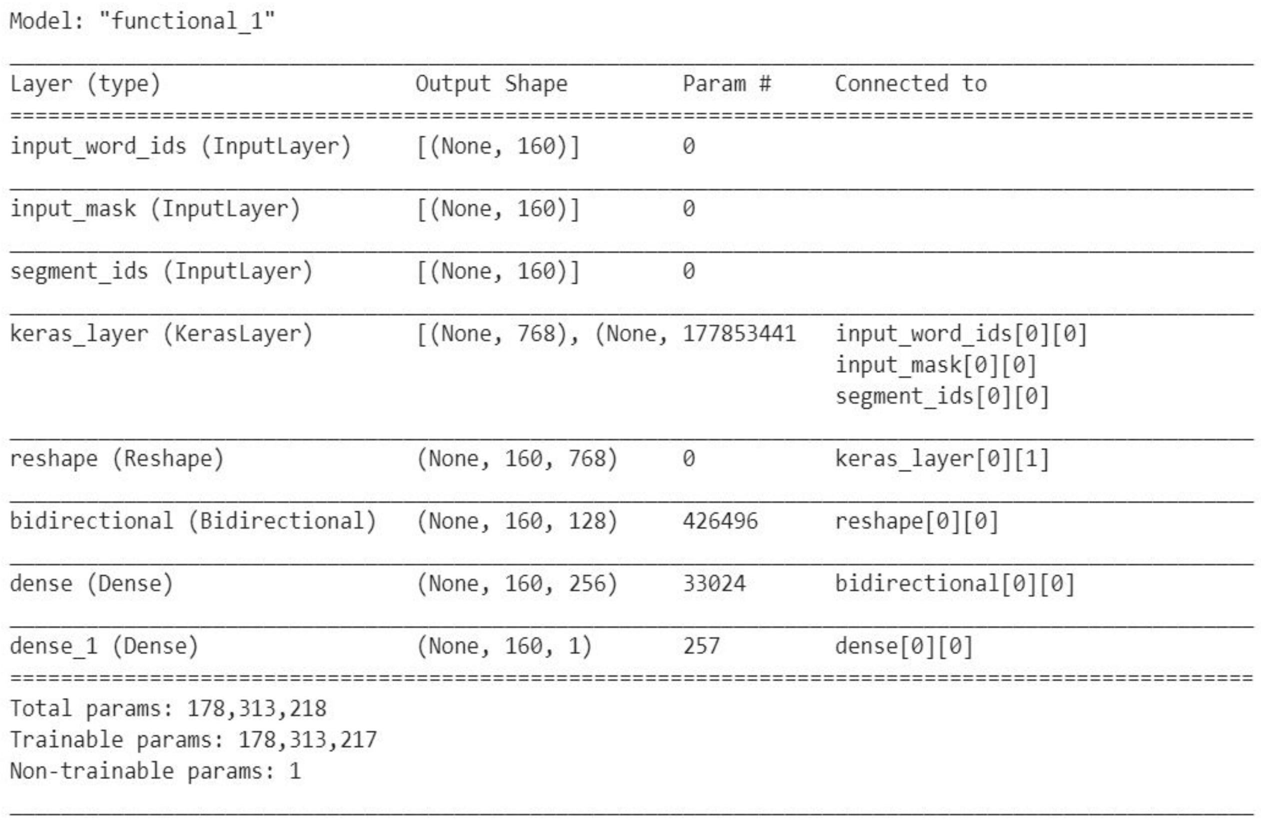
```
71.04810996563575
```

Fine Tuned BERT models:

In this, instead of relying upon pre-trained BERT, we have fine-tuned the BERT model according to the few-shot setting and dataset.

We have trained the following models on 500 samples of data.

The following is the model summary of fine-tuned BERT.



```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)     [(None, 160)]        0

__________________________________________________________________________________________________
input_mask (InputLayer)         [(None, 160)]        0

__________________________________________________________________________________________________
segment_ids (InputLayer)        [(None, 160)]        0

__________________________________________________________________________________________________
keras_layer (KerasLayer)        [(None, 768), (None, 177853441   input_word_ids[0][0]
                                                                 input_mask[0][0]
                                                                 segment_ids[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 160, 768)     0           keras_layer[0][1]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 160, 128)     426496      reshape[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 160, 256)     33024       bidirectional[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 160, 1)       257         dense[0][0]
==================================================================================================
Total params: 178,313,218
Trainable params: 178,313,217
Non-trainable params: 1
__________________________________________________________________________________________________
```

Pre-trained BERT + BiLSTM:

Here, we applied Bi-directional LSTM on top of the pre-trained BERT and modified the word embedding in such a way that the output of the LSTM layer captures relevant information from the BERT embedding layer for classifying the text.

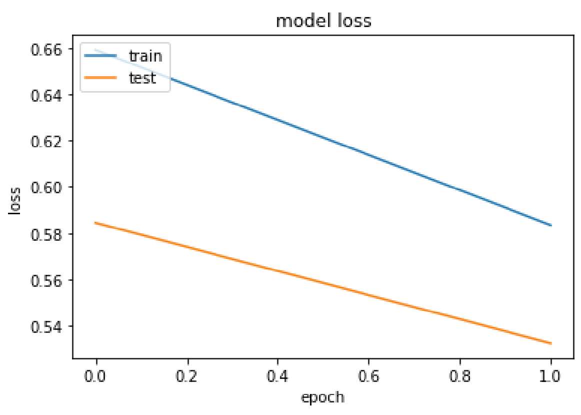
Pre-trained BERT + BiGRU:

Here the implementation is similar to the above method. Howere, instead of BiLSTM we have used BiGRU.

4 RESULTS AND DISCUSSION

Small Dataset (BERT):

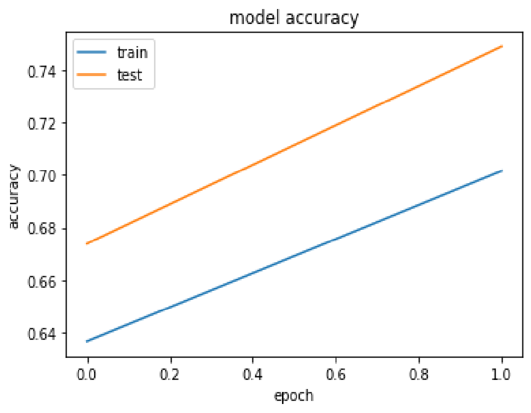
Training data size=332
Epochs=2
Bert model=https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/1
With the above parameters, the BERT model gave an accuracy of ~80% on the test dataset.

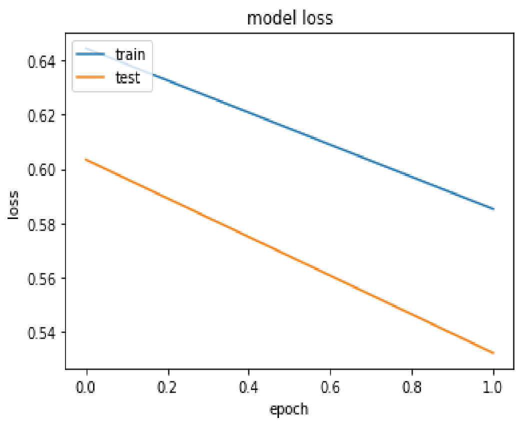


Large Dataset(BERT):
Training data size=4654
Epochs=2
Bert model=https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/1

With the above parameters, the BERT model gave an accuracy of ~71% on the validation dataset.

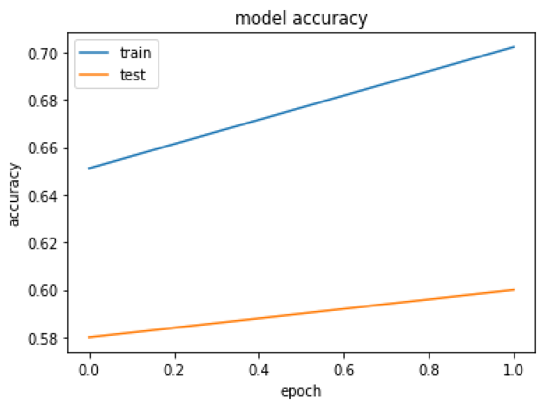
Training and validation accuracy **Training and validation loss**



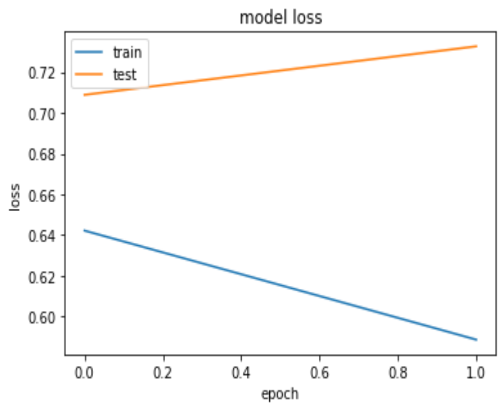


Training data size=500
Epochs=2
Bert model=https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/1
With the above parameters, the BERT model gave an accuracy of ~68% on the test dataset.

Training and validation accuracy



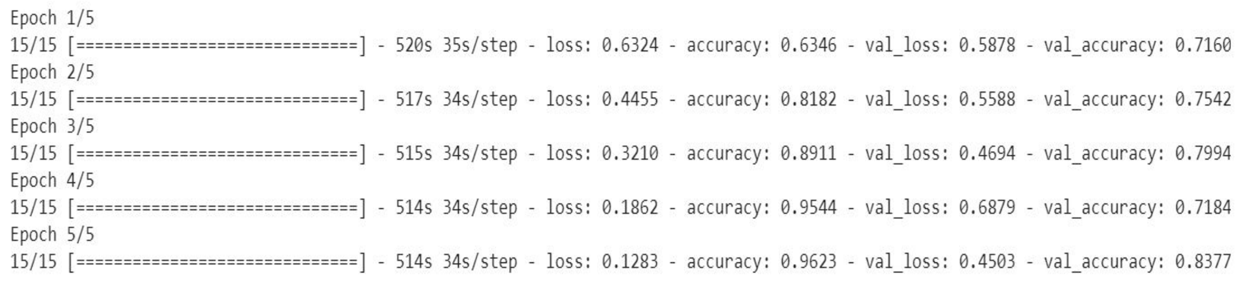
Training and validation loss



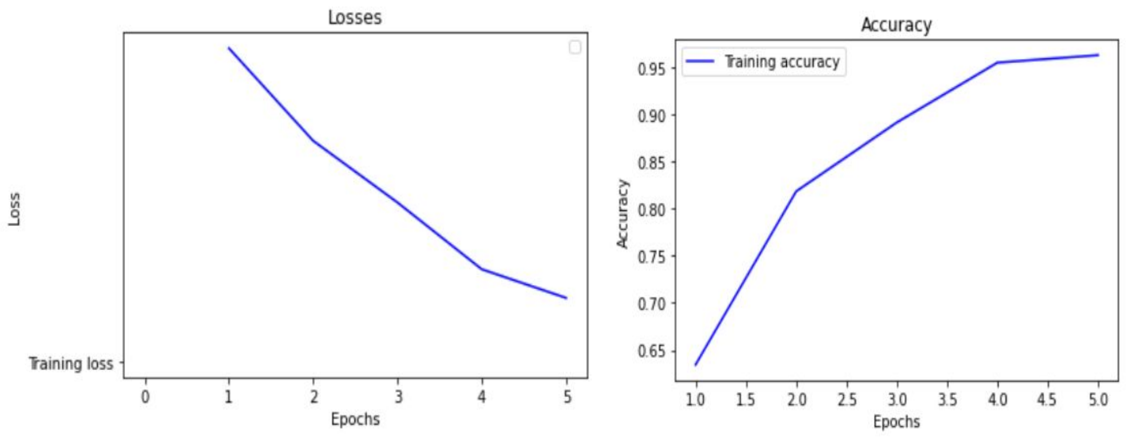
From the above results, we notice that when training data size is reduced from the large dataset, the accuracy also reduced. However, the goal here is to build an efficient model that gives good accuracy with less training data using few shot learning methods.

Small dataset(Fine Tuned BERT models)

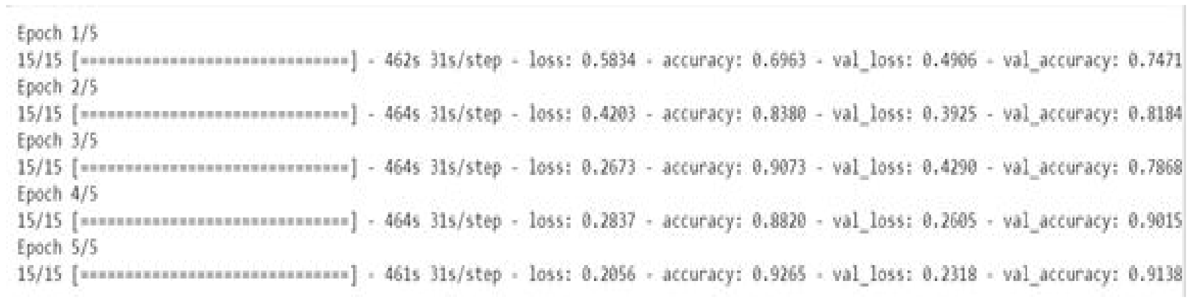
Training data size: 272
Epochs: 5
Validation Accuracy: 83.77%
Validation Loss: 0.4503



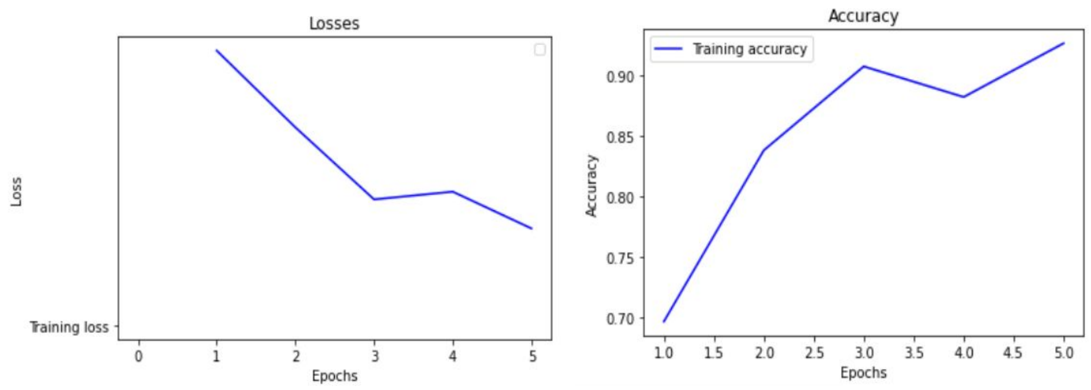
```
Epoch 1/5
15/15 [==============================] - 520s 35s/step - loss: 0.6324 - accuracy: 0.6346 - val_loss: 0.5878 - val_accuracy: 0.7160
Epoch 2/5
15/15 [==============================] - 517s 34s/step - loss: 0.4455 - accuracy: 0.8182 - val_loss: 0.5588 - val_accuracy: 0.7542
Epoch 3/5
15/15 [==============================] - 515s 34s/step - loss: 0.3210 - accuracy: 0.8911 - val_loss: 0.4694 - val_accuracy: 0.7994
Epoch 4/5
15/15 [==============================] - 514s 34s/step - loss: 0.1862 - accuracy: 0.9544 - val_loss: 0.6879 - val_accuracy: 0.7184
Epoch 5/5
15/15 [==============================] - 514s 34s/step - loss: 0.1283 - accuracy: 0.9623 - val_loss: 0.4503 - val_accuracy: 0.8377
```



Training data size: 272
Epochs: 5
Validation Accuracy: 91.38%
Validation Loss: 0.2318



```
Epoch 1/5
15/15 [==============================] - 462s 31s/step - loss: 0.5834 - accuracy: 0.6963 - val_loss: 0.4906 - val_accuracy: 0.7471
Epoch 2/5
15/15 [==============================] - 464s 31s/step - loss: 0.4203 - accuracy: 0.8380 - val_loss: 0.3925 - val_accuracy: 0.8184
Epoch 3/5
15/15 [==============================] - 464s 31s/step - loss: 0.2673 - accuracy: 0.9073 - val_loss: 0.4290 - val_accuracy: 0.7868
Epoch 4/5
15/15 [==============================] - 464s 31s/step - loss: 0.2837 - accuracy: 0.8820 - val_loss: 0.2605 - val_accuracy: 0.9015
Epoch 5/5
15/15 [==============================] - 461s 31s/step - loss: 0.2056 - accuracy: 0.9265 - val_loss: 0.2318 - val_accuracy: 0.9138
```



Conclusion:

From the above graphs we can see that Fine-tuned BERT by adding BiLSTM or BiGRU improves the accuracy a lot and outperforms the BERT model by a large margin.

6 REFERENCES

[1]https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/1
[2]https://blog.floydhub.com/n-shot-learning/
[3]https://github.com/sudharsan13296/Hands-On-Meta-Learning-With-Python
[4]https://medium.com/analytics-vidhya/few-shot-learning-a-case-study-3-84f6ea3cb322

7 GROUP JUSTIFICATION

Data preprocessing	Combined
BERT encode function	Deepika Padmanabhan
BERT build model function	Dimple Mehra
Fine tuned BERT (BERT+ BiLSTM)	Deepika Padmanabhan
Fine tuned BERT (BERT + BiGRU)	Dimple Mehra
Model evaluation and testing	combined
Presentation slides	combined
Report	combined