Victor Karani

victor.karani@gmail.com

Spoiler Shield Mini Project

Table of Contents

Summary and overall Approach

I chose the spoiler shield for my mini project. I went through several approaches to end up with the final result for this project.

EDA

Data Overview

The data for this project comes in 2 files:

- IMDB_reviews.json
- IMDB_movie_details.json

IMDB_reviews

The **is_spoiler** and **review_text** are the important columns here, with is_spoiler containing the **labels** and **review_text** containing the actual reviews that are to be classified. We also need **movie_id** as it will be used to join with the next data file.

IMDB_movie_details

The **plot_summary** , **plot_synopsis** and **movie_id** are the important columns here. The **plot_summary** contains a movie description provided by the studio themselves, ideally written to be enticing but NOT to have spoilers. The **plot_synopsis** contains movie details and can be assumed to have spoilers in it.

Overall data management plan

I read the 2 files into pandas dataframes and then merged them on the **movie_id** column.

Dataset size

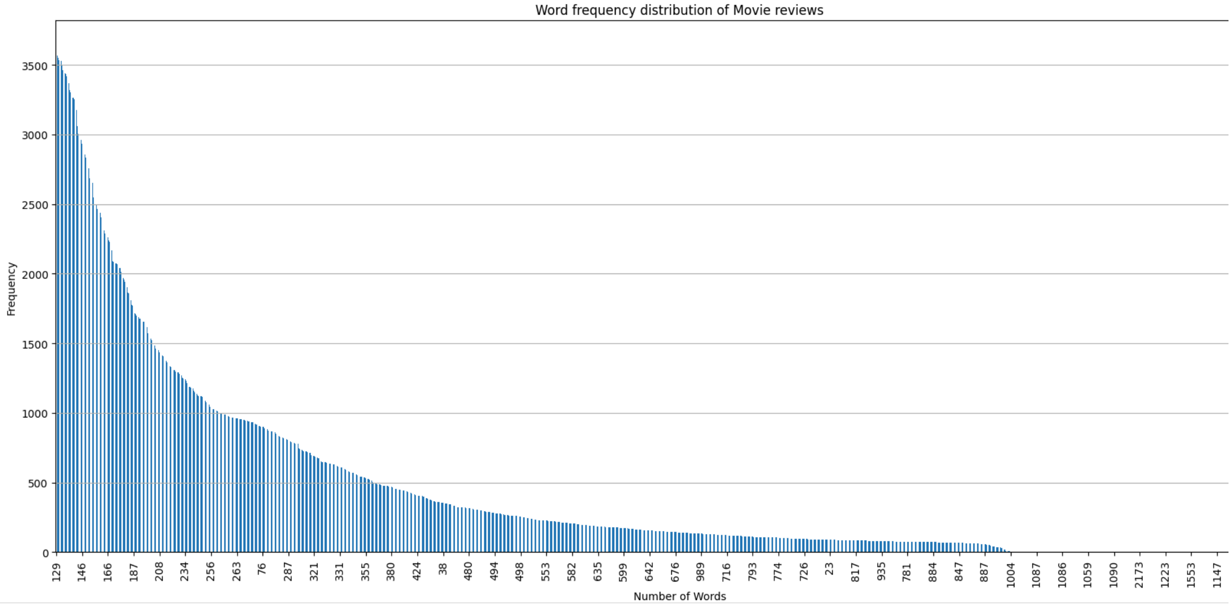
This is a very large dataset with 573906 rows and 13 columns after the merge.

Dataset size and Imbalance

There are 422601 cases where is_spoiler is **FALSE** and 150862 cases where is_spoiler is **TRUE**. I will solve the giant size problem by only training on a portion of the dataset. I will solve the imbalance problem by creating a balanced dataset before training. This dataset is large enough that I don't believe I need synthetic data of any kind.

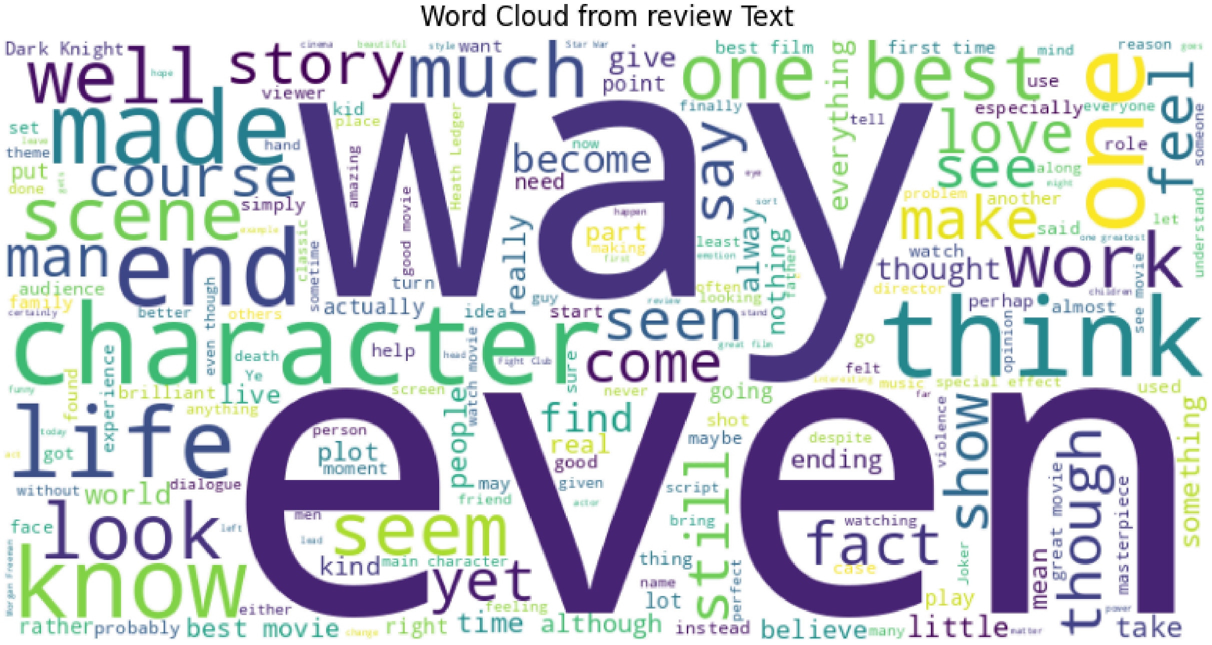
Word Count

I performed a word count analysis. Looks like there is a majority (~ 3500 examples) of movie reviews at 129 words.



Word Cloud

The word cloud gives a feeling of the most commonly used words. Not super useful, but interesting to look at:



+ Code + Markdown

TF-IDF -Cosine similarity

I got a cosine similarity between the review_text and plot_summary of 0.39, which is pretty close.

Model training and hyperparameter tuning

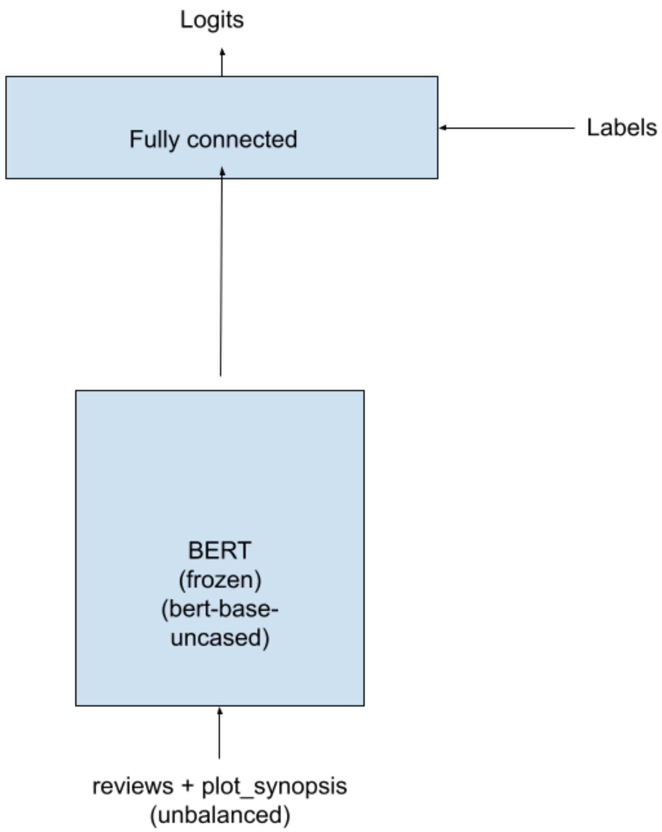
I had 3 distinct iterations for the approach to solve this problem. I also tried some experiments with unbalanced datasets

Model 1

Imbalanced data test (not presented in the notebook)

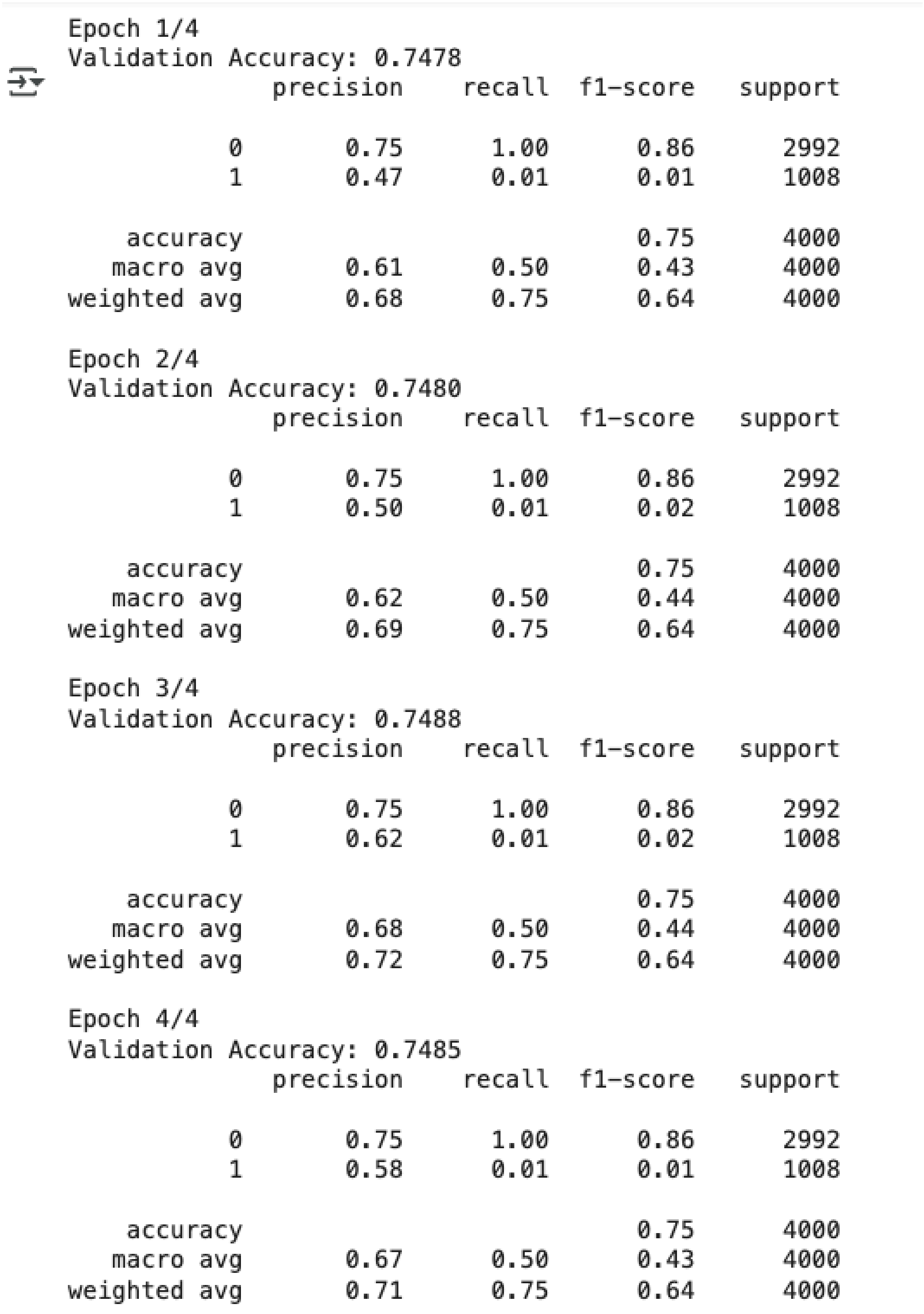
As an initial experiment, I tried at the very beginning to use unbalanced data.

Model 1 Block diagram



Model 1 results

The results were *spectacularly* bad. Below is a screenshot of what typically happened:



```
Epoch 1/4
Validation Accuracy: 0.7478
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      2992
           1       0.47      0.01      0.01      1008

    accuracy                           0.75      4000
   macro avg       0.61      0.50      0.43      4000
weighted avg       0.68      0.75      0.64      4000

Epoch 2/4
Validation Accuracy: 0.7480
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      2992
           1       0.50      0.01      0.02      1008

    accuracy                           0.75      4000
   macro avg       0.62      0.50      0.44      4000
weighted avg       0.69      0.75      0.64      4000

Epoch 3/4
Validation Accuracy: 0.7488
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      2992
           1       0.62      0.01      0.02      1008

    accuracy                           0.75      4000
   macro avg       0.68      0.50      0.44      4000
weighted avg       0.72      0.75      0.64      4000

Epoch 4/4
Validation Accuracy: 0.7485
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      2992
           1       0.58      0.01      0.01      1008

    accuracy                           0.75      4000
   macro avg       0.67      0.50      0.43      4000
weighted avg       0.71      0.75      0.64      4000
```

The accuracy seems high at 75% but, just look how uneven the precision, recall, and f1-score are between is_spoiler 0 and 1 . This is so bad I didn’t even bother with the confusion matrix here.

Model 2

(not presented in the notebook)

I clearly need to balance the model.

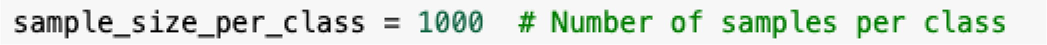
It turns out that I don’t have that much processing power even in Kaggle. I could only approach 10000 samples total. As a reminder, there are 422601 cases where is_spoiler is **FALSE** and 150862 where is_spoiler is **TRUE.**

So all I had to do was decide the maximum number of samples that I wanted to test, and then, pick an equal number of FALSE and TRUE is_spoiler samples.

Below is a typical False/True (0/1) sample distribution prior to training

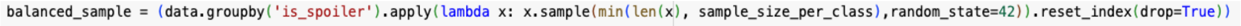
Balanced sample distribution before training

This is how I balance the classes in code…



```
sample_size_per_class = 1000  # Number of samples per class
```

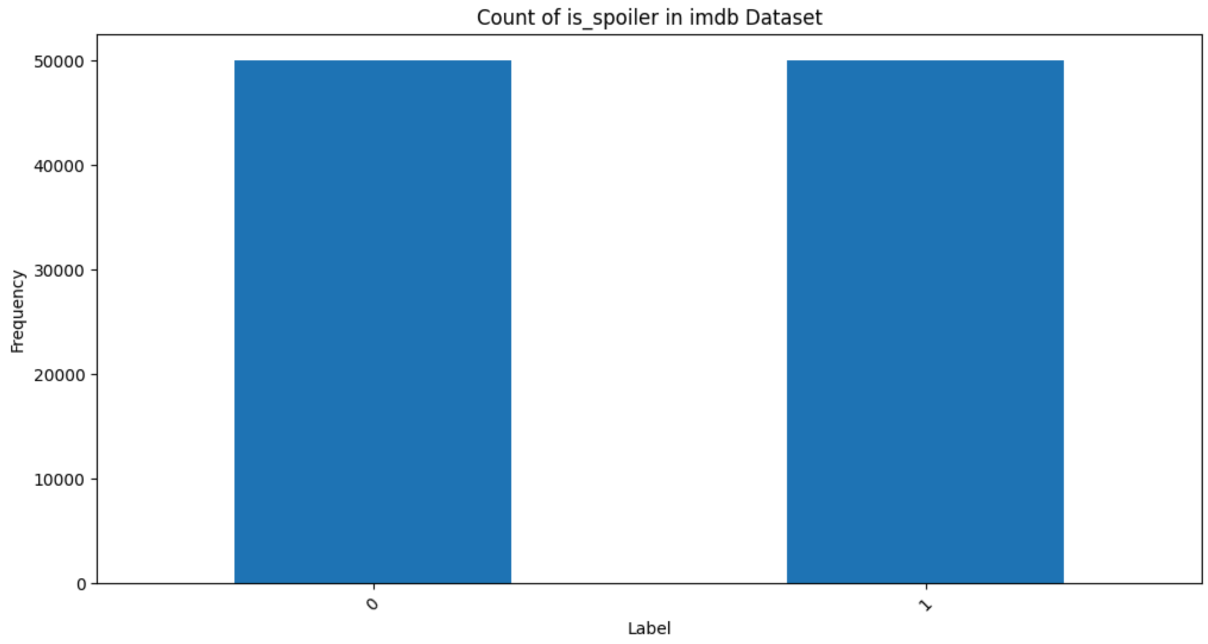
And



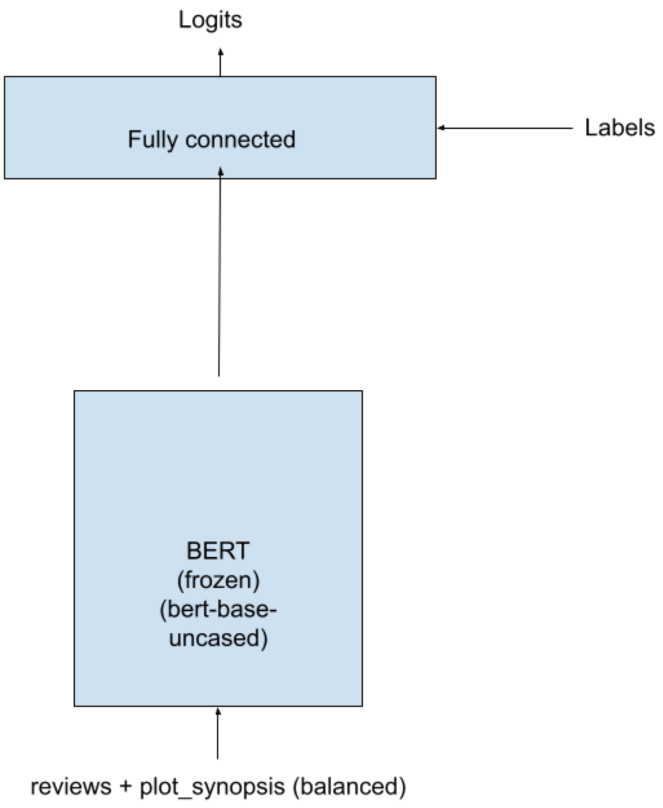
```
balanced_sample = (data.groupby('is_spoiler').apply(lambda x: x.sample(min(len(x), sample_size_per_class),random_state=42)).reset_index(drop=True))
```

From here on in the document, I will refer to sample_size_per_class as ‘sample size’.

Below is one example where the sample size is 50000. As you can see is_spoiler T and F are well balanced.


Count of is_spoiler in imdb Dataset
Frequency
50000
40000
30000
20000
10000
0
0
1
Label

Model 2 Block diagram



Hyperparameter tuning and interpretation with Model 2

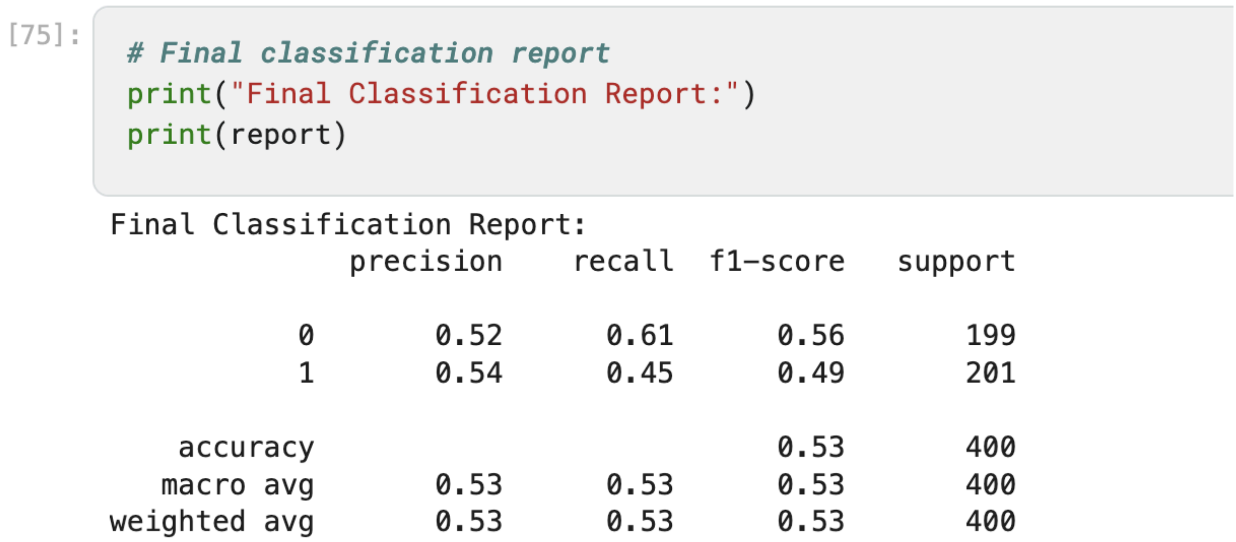
I did most of my hyperparameter tuning with Model 2, so I will go through all the hyperparameter tuning process here

Model 2: Sample size 1000, 6 epochs, lr 2e-5

Huggingface Model = bert-base-uncased
Sample size 1000
Balanced (True vs false are equal in number)
num_epochs=6
Learning_rate 2e-5

Results for Model 2: Sample size 1000, 6 epochs, lr 2e-5

Classification report



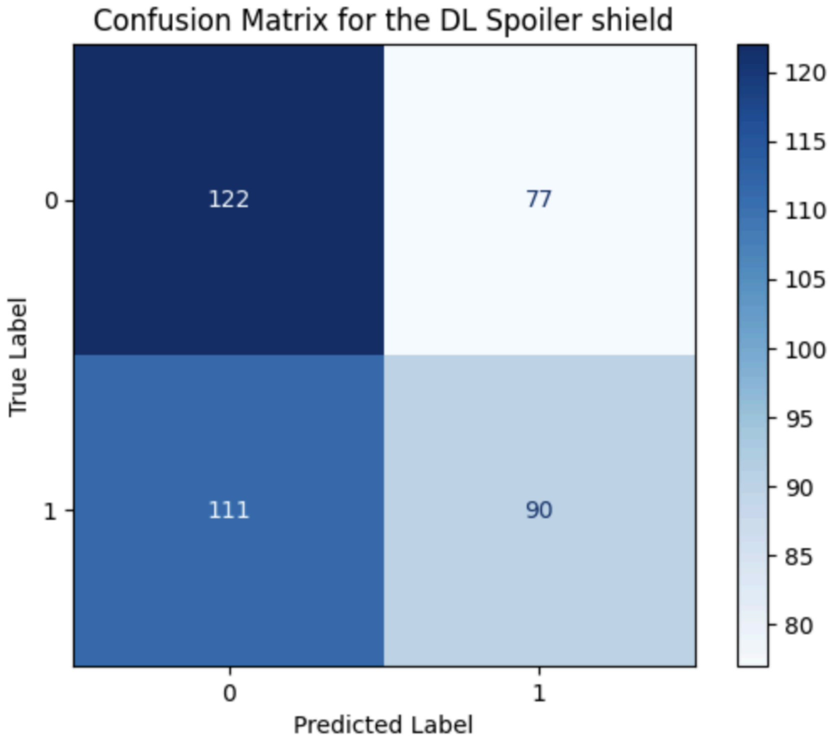
```
[75]: # Final classification report
      print("Final Classification Report:")
      print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.61      0.56       199
           1       0.54      0.45      0.49       201

    accuracy                           0.53       400
   macro avg       0.53      0.53      0.53       400
weighted avg       0.53      0.53      0.53       400
```

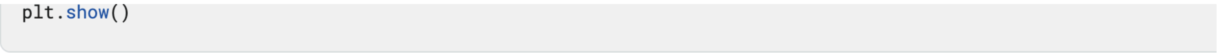
Classification report result:
Results (precision, recall, f1-summary are much better balanced than the previous unbalanced version. Accuracy is still pretty low, so we have a ways to go. Note that here, I am using a TINY sample size of 1000

Confusion MatrixTraining vs validation

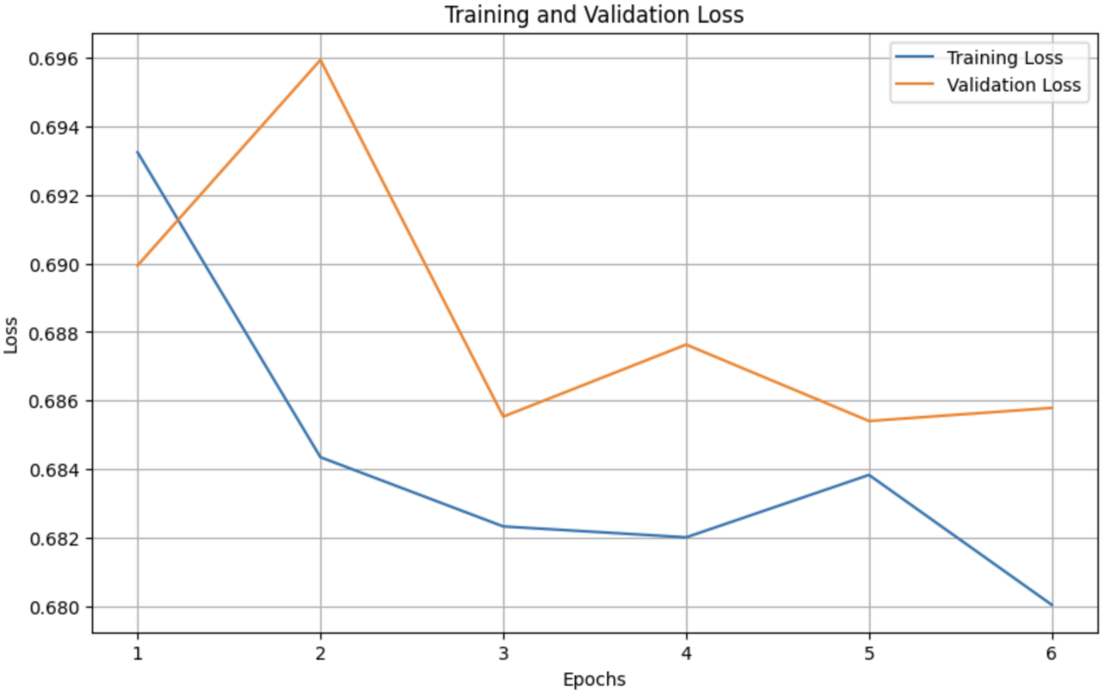
```
[76]: # Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Training Vs Validation Loss



```
plt.show()
```



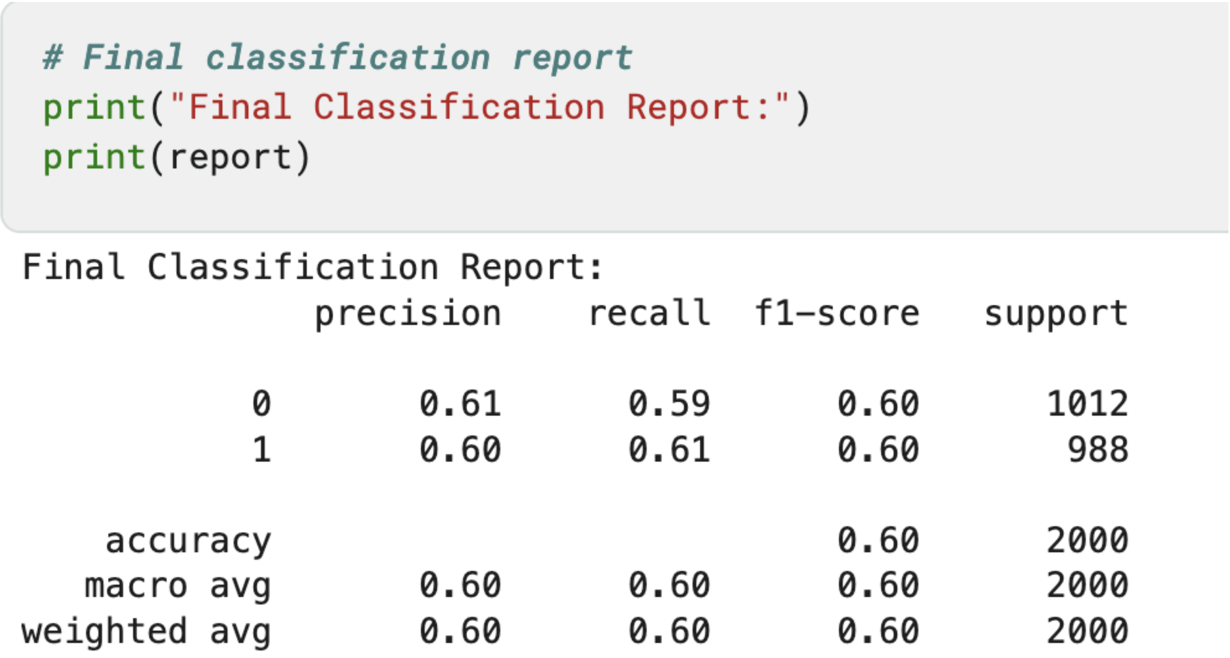
Validation seems to be bottoming out , so perhaps running this for more epochs might help. The tiny sample size of 1000 probably contributes to the very uneven validation loss seen here.

Model 2: increased learning rate (2e-4), 12 epochs,sample size of 5000

Note: I first tried increasing the number of epochs (12) separately and found the performance was better. I also increased the sample size (5000) separately and found the performance also improved . Then, here, I also increased the learning rate.

Huggingface Model = bert-base-uncased
Sample size 5000
Balanced (True vs false are equal in number)
num_epochs=12
Learning_rate 2e-4

Classification report

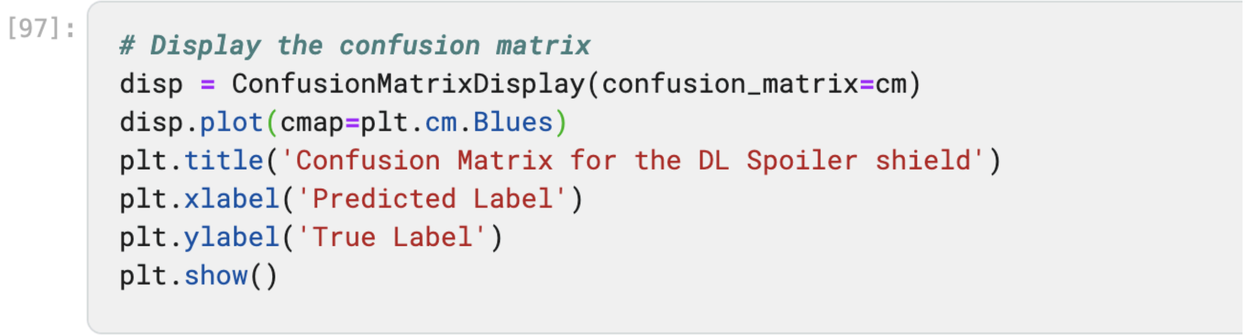


```python
# Final classification report
print("Final Classification Report:")
print(report)
```

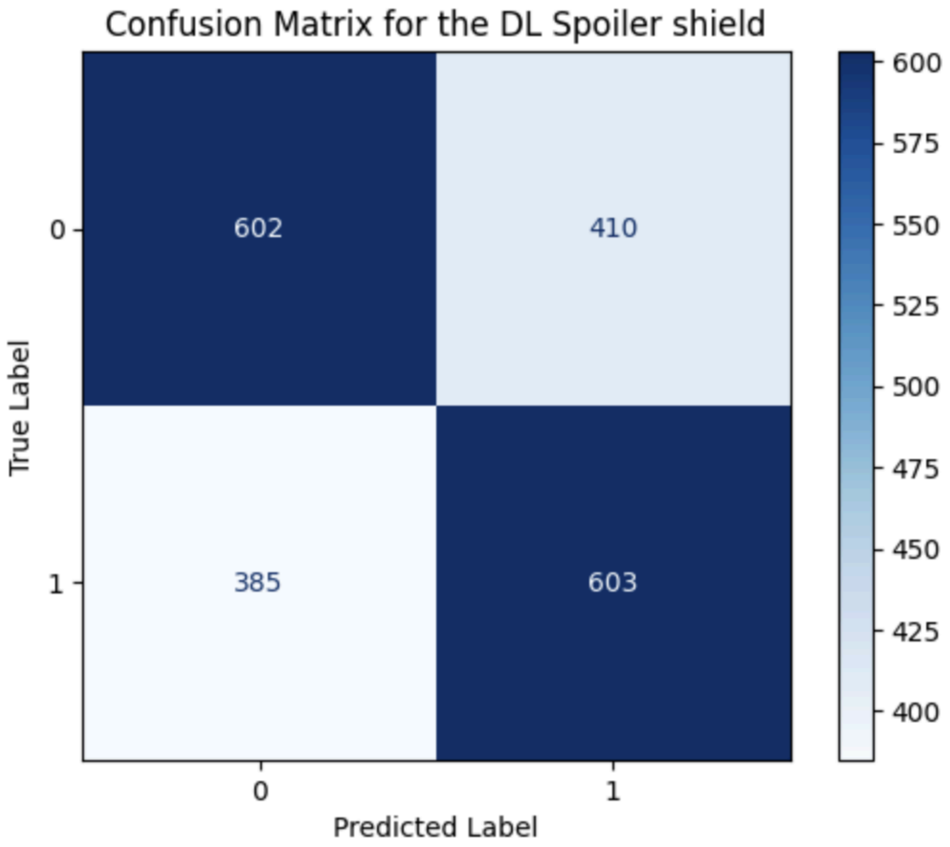
```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.59      0.60      1012
           1       0.60      0.61      0.60       988

    accuracy                           0.60      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.60      0.60      0.60      2000
```

Classification report result:
Results (precision, recall, f1-summary and accuracy) are pretty good. Note that I am using a TINY sample size of 5000

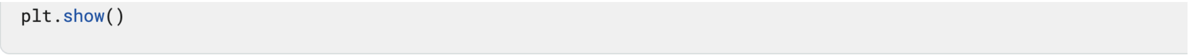
Confusion Matrix



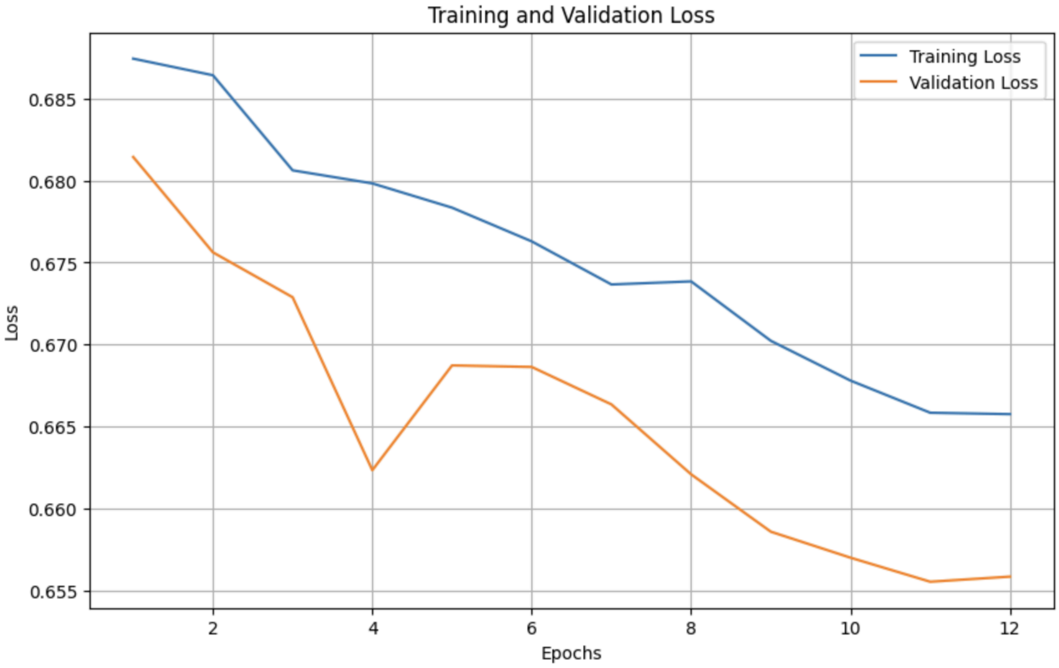
```
[97]:
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Confusion Matrix summary
The false positives and false negatives are higher than I would like. True positives and True negatives are at least pretty solid at ~ 60 %.



```
plt.show()
```



Training and validation loss

The changes here (increased learning rate (2e-4), 12 epochs,sample size of 5000) provide a decent loss reduction (bottoming out at 11 epochs)

Model 2: increased learning rate (2e-3)

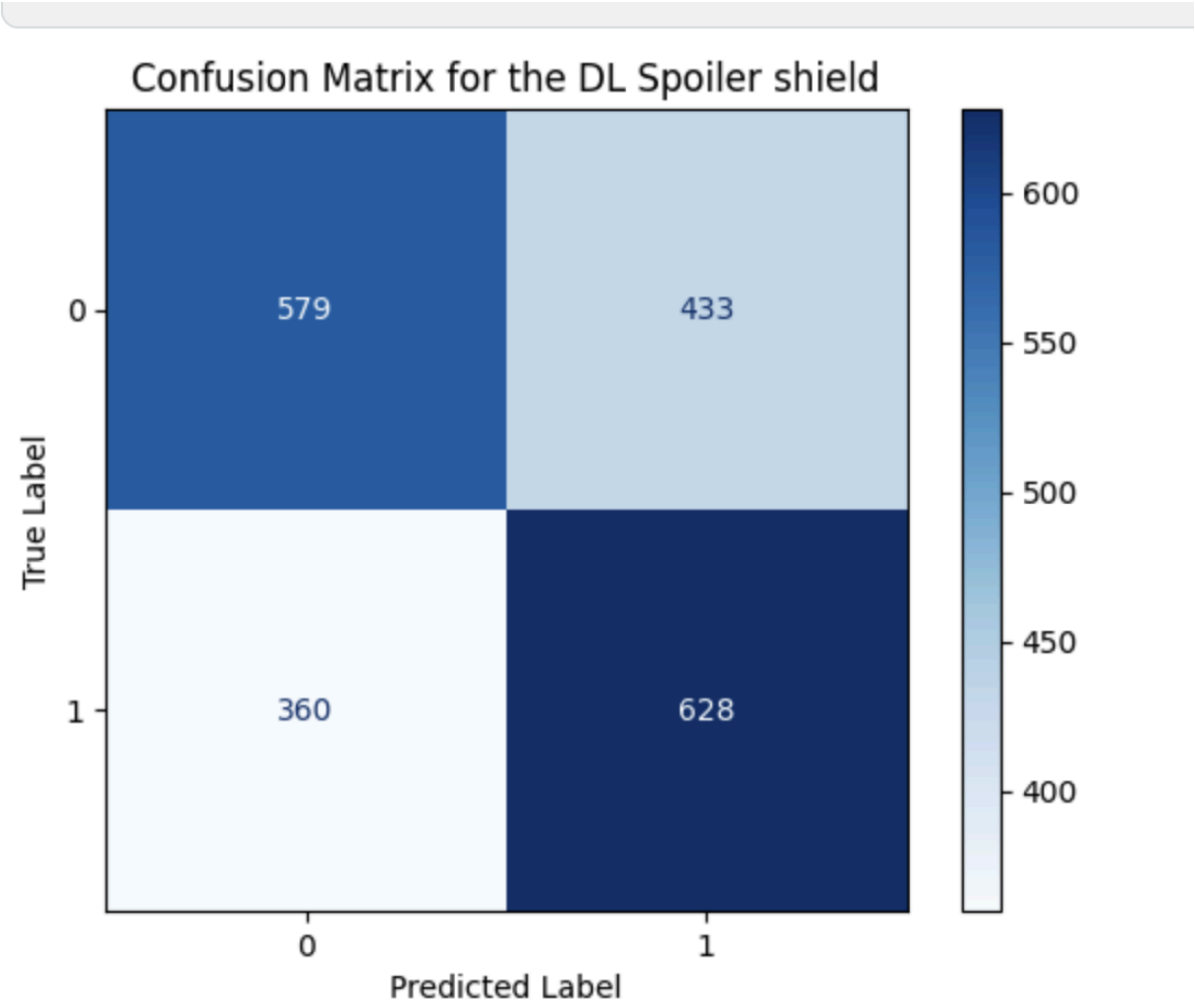
I reran the exact same test as before but tried to increase the learning rate even more.

Huggingface Model = bert-base-uncased
Sample size 5000
Balanced (True vs false are equal in number)
num_epochs=12
Learning_rate 2e-3

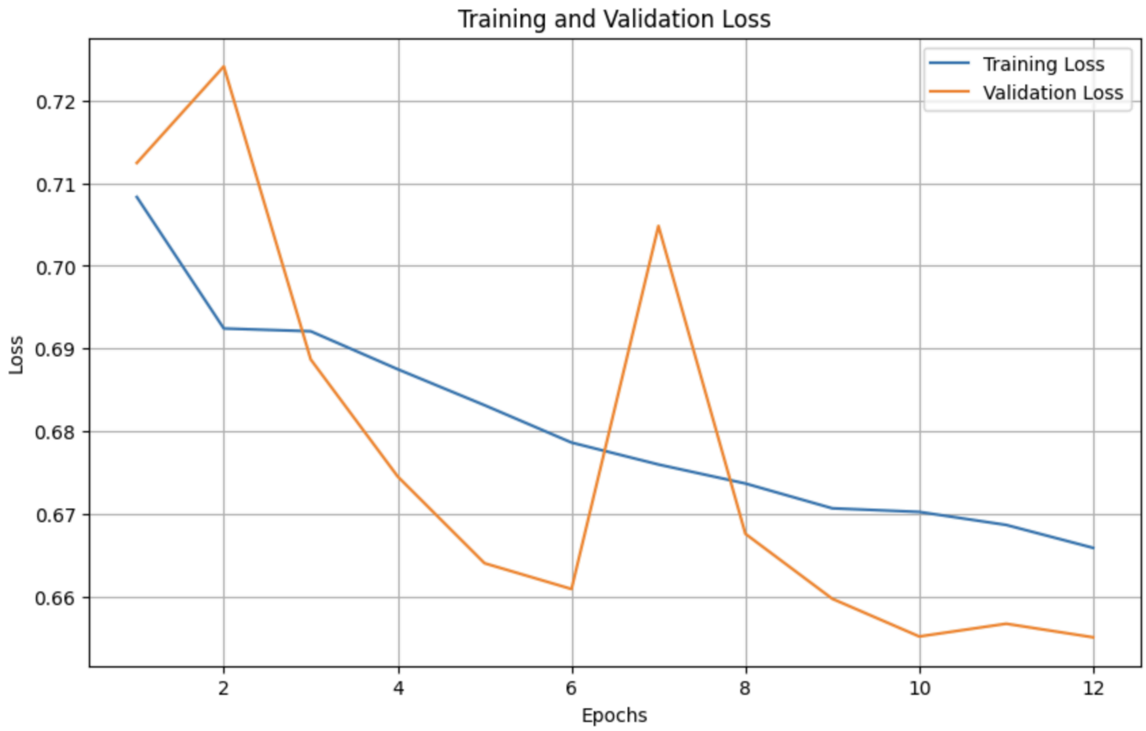
[129]:

```python
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.57      0.59      1012
           1       0.59      0.64      0.61       988

    accuracy                           0.60      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.60      0.60      0.60      2000
```



True Label \ Predicted Label	0	1
0	579	433
1	360	628



Conclusion: increased learning rate (2e-3)

The classification report and Confusion matrix are similar. Unfortunately, I was too greedy. The validation loss is very unstable. I will try again with a lr slightly smaller than 2e-3.

Model 2: increased learning rate (5e-4), 12 epochs,sample size of 5000

Huggingface Model = bert-base-uncased
Sample size 5000
Balanced (True vs false are equal in number)
num_epochs=12
Learning_rate 5e-4

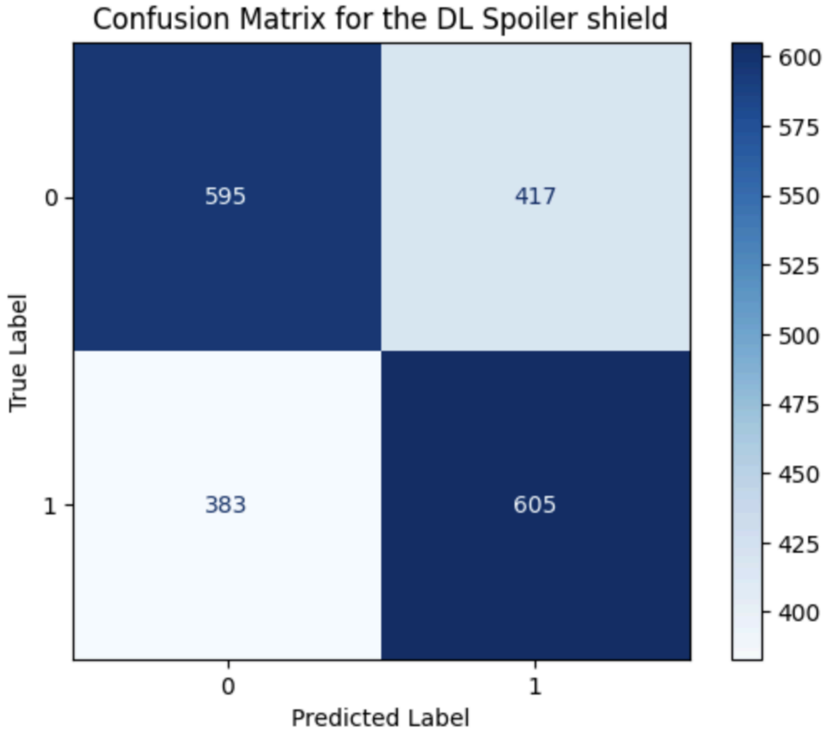
[195]:

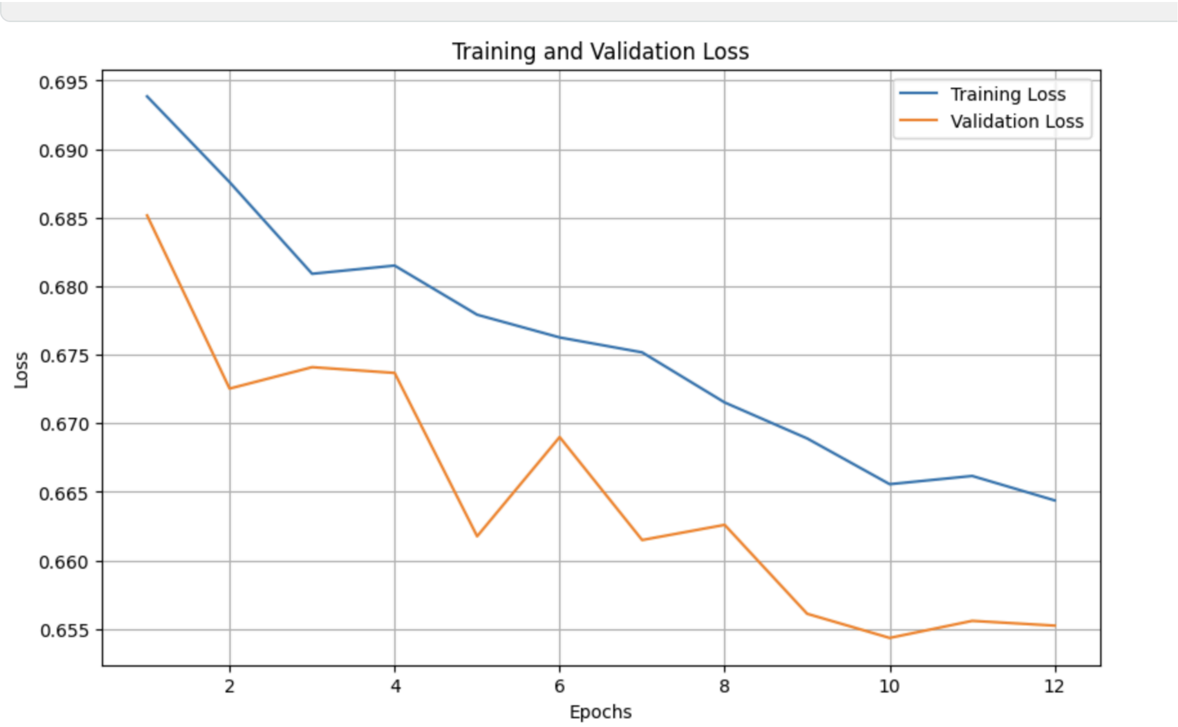
```python
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.59      0.60      1012
           1       0.59      0.61      0.60       988

    accuracy                           0.60      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.60      0.60      0.60      2000
```

[196]:

```
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



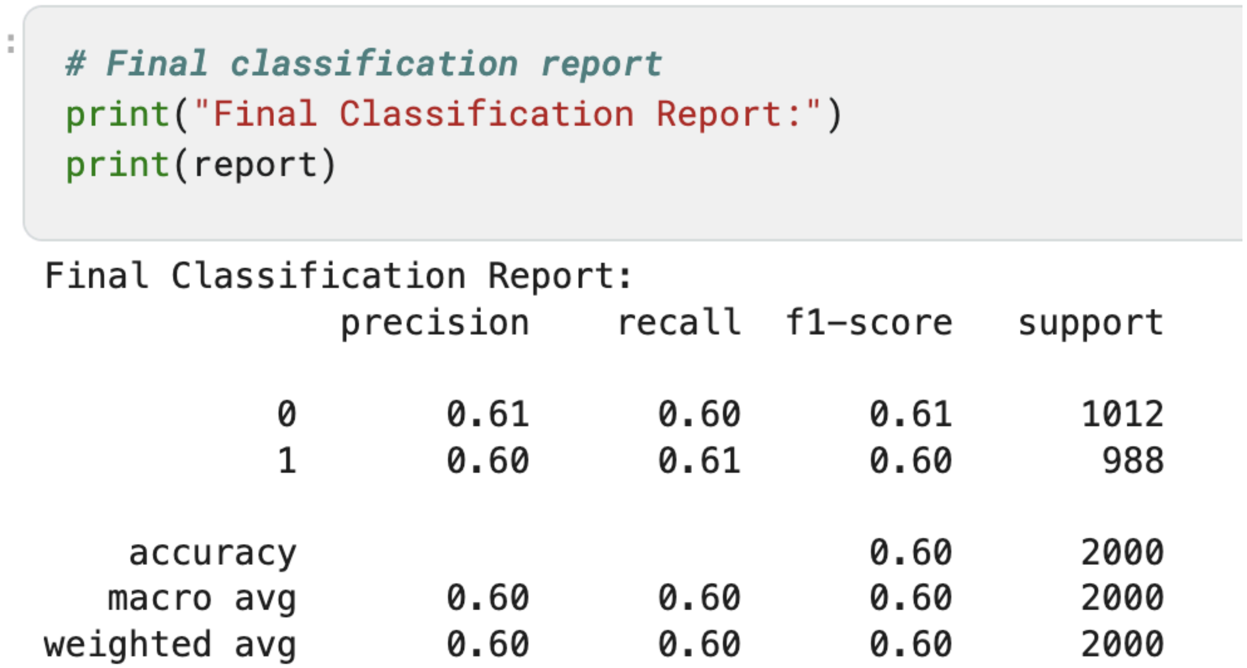


Conclusion: increased learning rate (5e-4)

The classification report and Confusion matrix are similar. Validation loss is a little less smooth than 2e-3 but I think offers more chance to converge faster. I'll keep 5e-4 for future runs.

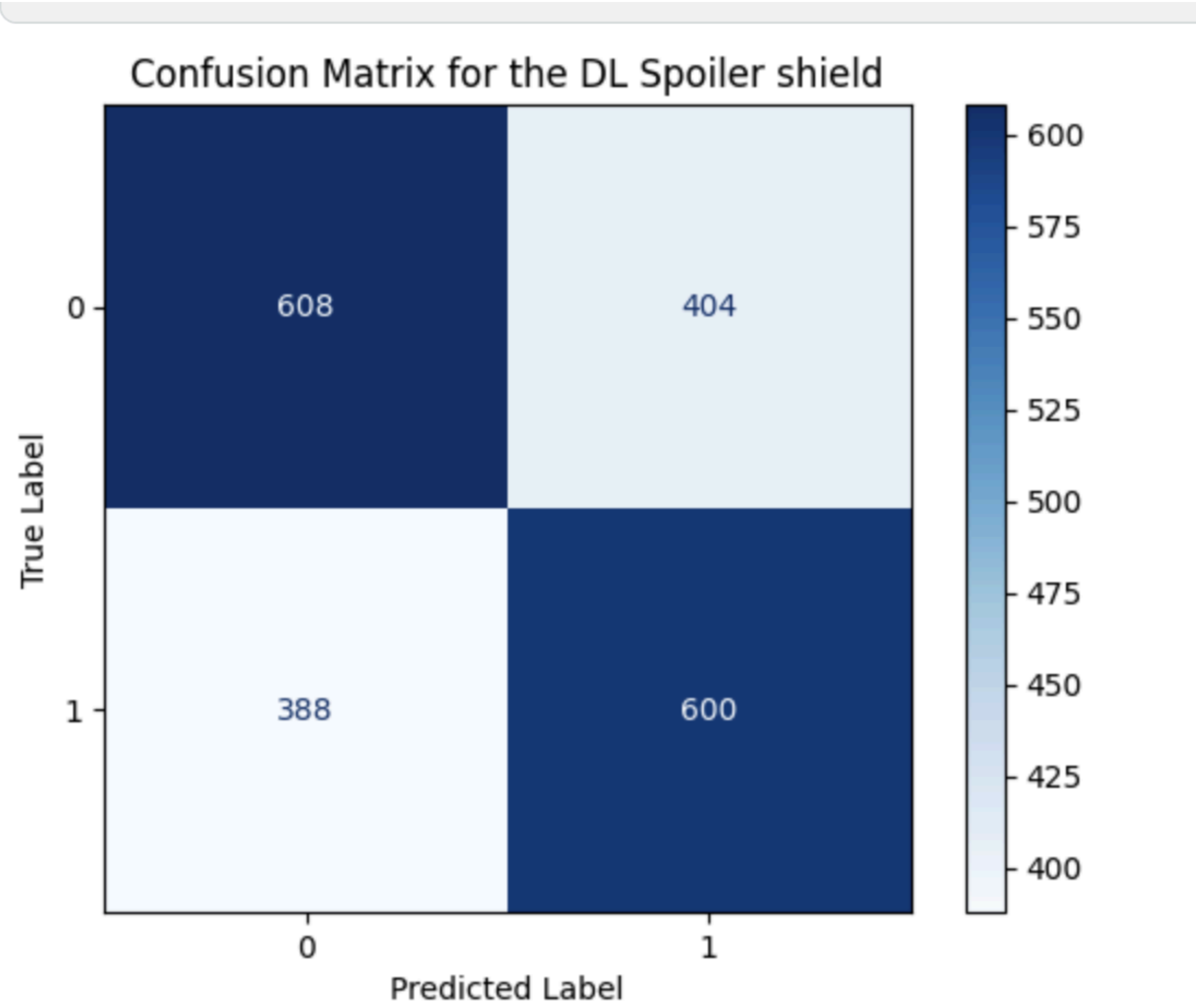
Model 2: bert-large-uncased

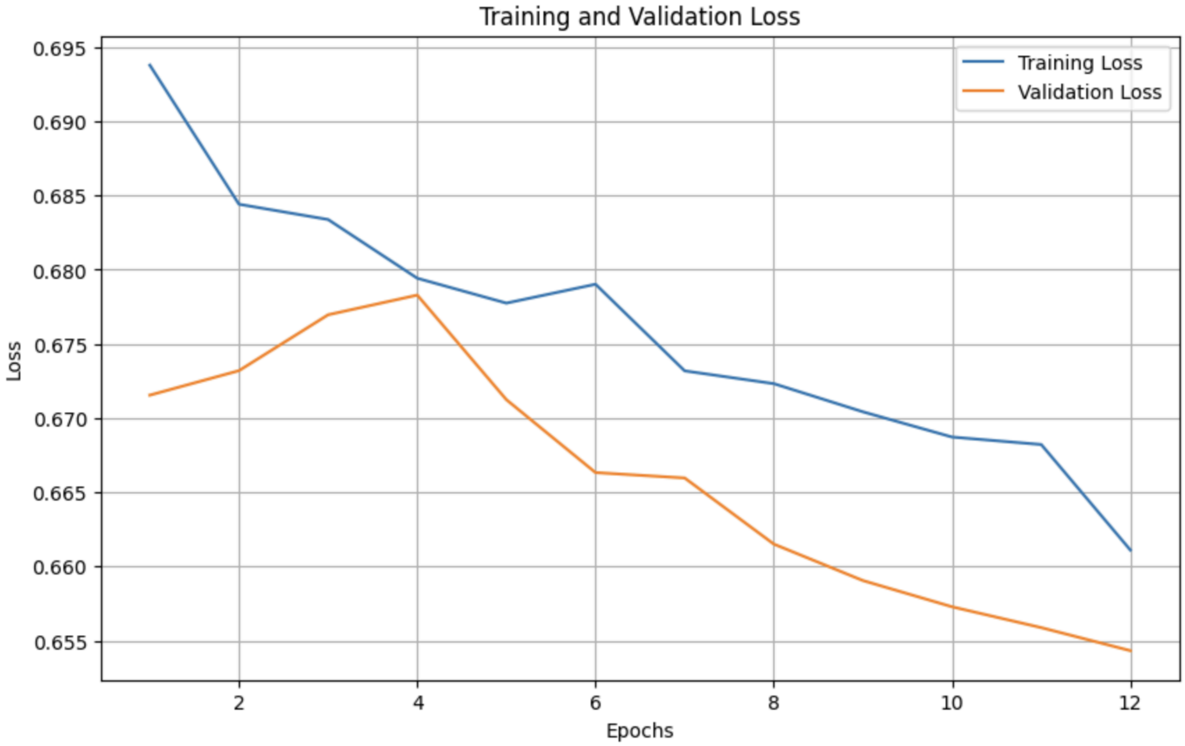
I tried to use the bigger **bert-large-uncased** model to see if it would be dramatically better than the previously used bert-base-uncased.
Huggingface Model = bert-large-uncased
Sample size 5000
Balanced (True vs false are equal in number)
num_epochs=12
Learning_rate 5e-4



```python
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.60      0.61      1012
           1       0.60      0.61      0.60       988

    accuracy                           0.60      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.60      0.60      0.60      2000
```


Confusion Matrix for the DL Spoiler shield
True Label
0
1
608
404
388
600
0
1
Predicted Label
600
575
550
525
500
475
450
425
400



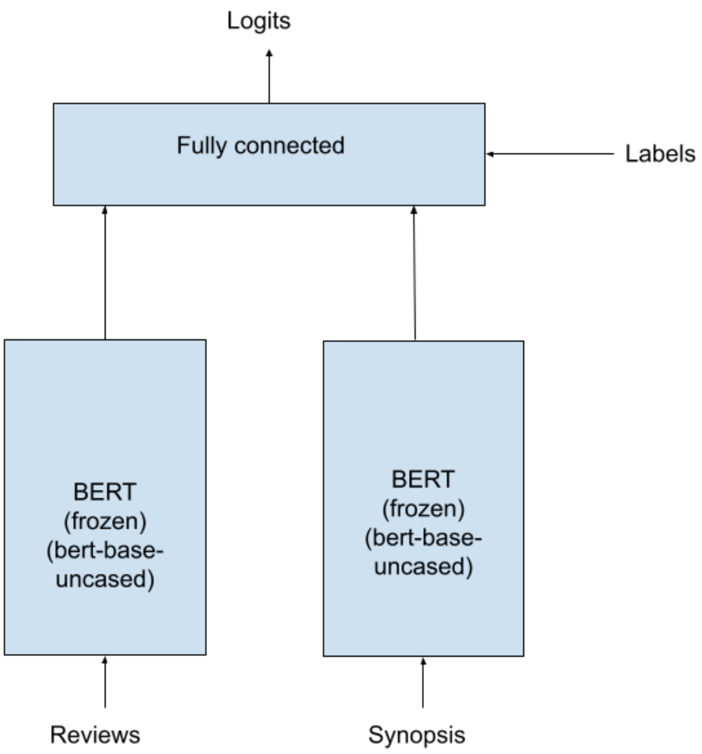
Conclusion: bert-large-uncased

The classification report and Confusion matrix are similar. Validation loss is a little less smooth than 2e-3. Overall, I don't think that the bert-large-uncased gives me dramatic improvements over bert-base-uncased. So I will keep bert-based-uncased.

Model 3

(not presented in the notebook)
I think there are still more ways to improve the model. Next I tried the 2 tower model

Model 3 Block diagram


Logits
Fully connected
Labels
BERT (frozen) (bert-base-uncased)
BERT (frozen) (bert-base-uncased)
Reviews
Synopsis

Model 3: two tower with 10000 sample size

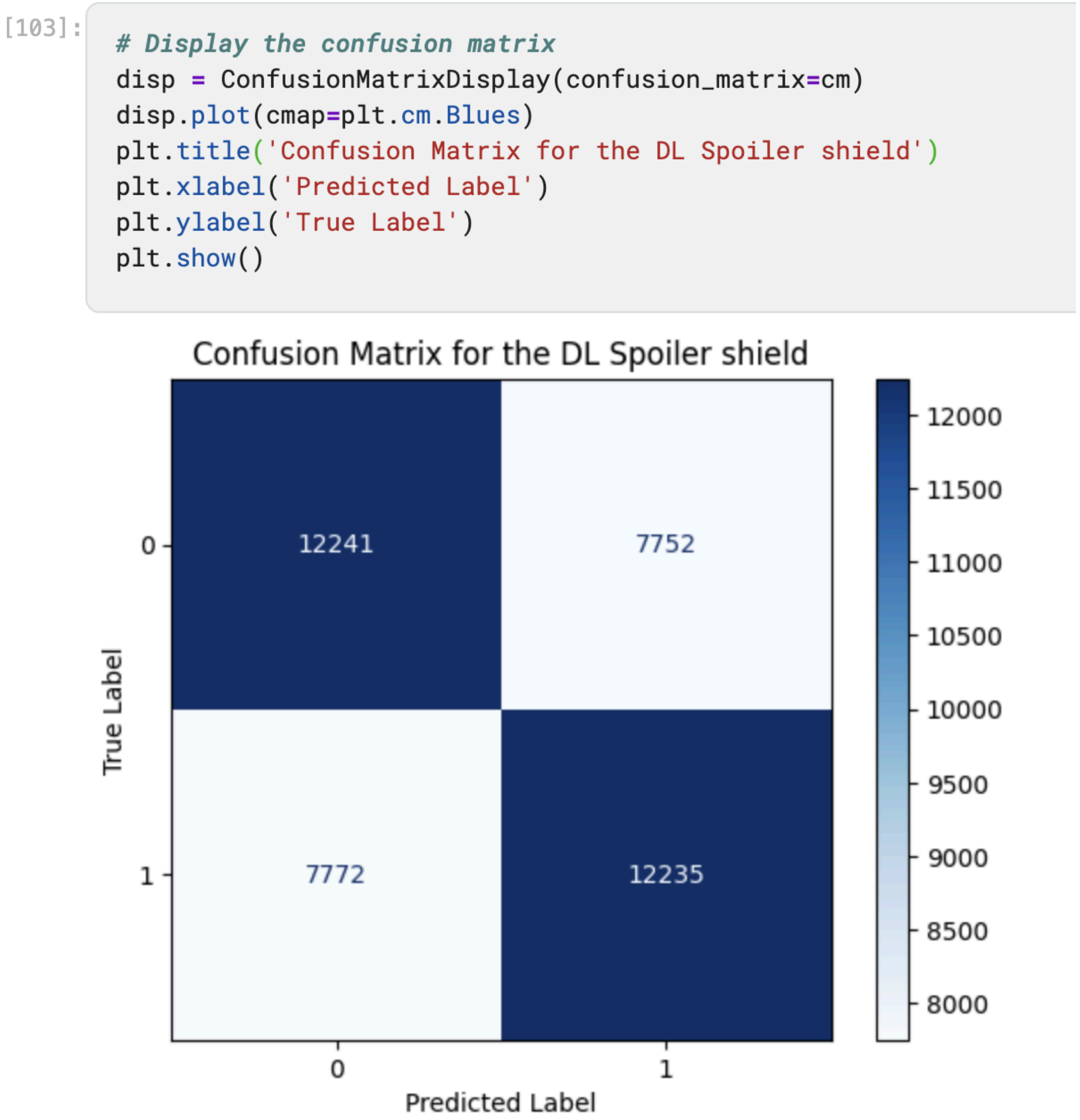
Huggingface Model = bert-base-uncased
Sample size **100000**
Balanced (True vs false are equal in number)
num_epochs=**4**
Learning_rate 5e-4

I used only 4 epochs for GPU time reasons.

[102]:

```
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.61      0.61     19993
           1       0.61      0.61      0.61     20007

    accuracy                           0.61     40000
   macro avg       0.61      0.61      0.61     40000
weighted avg       0.61      0.61      0.61     40000
```

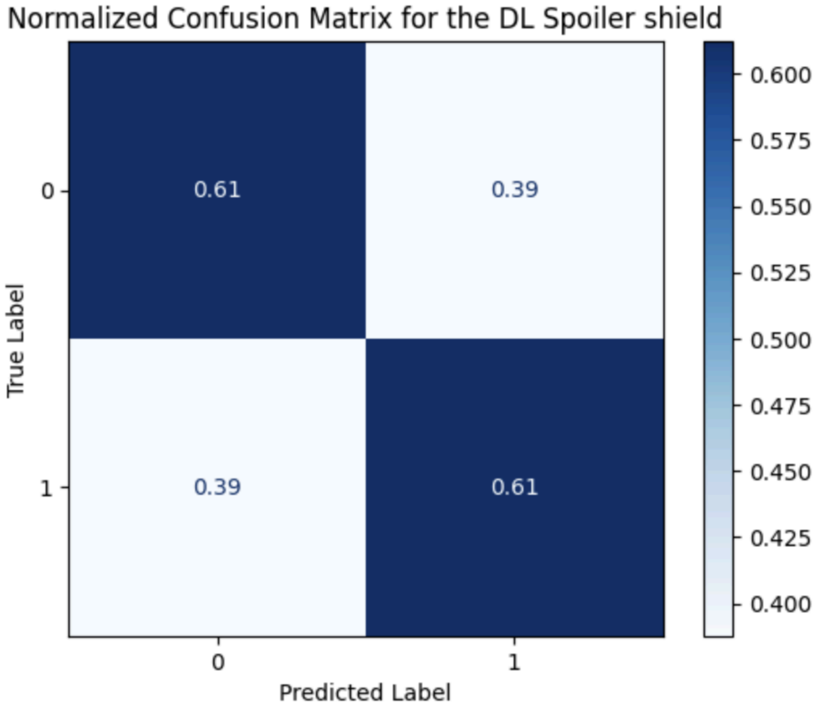


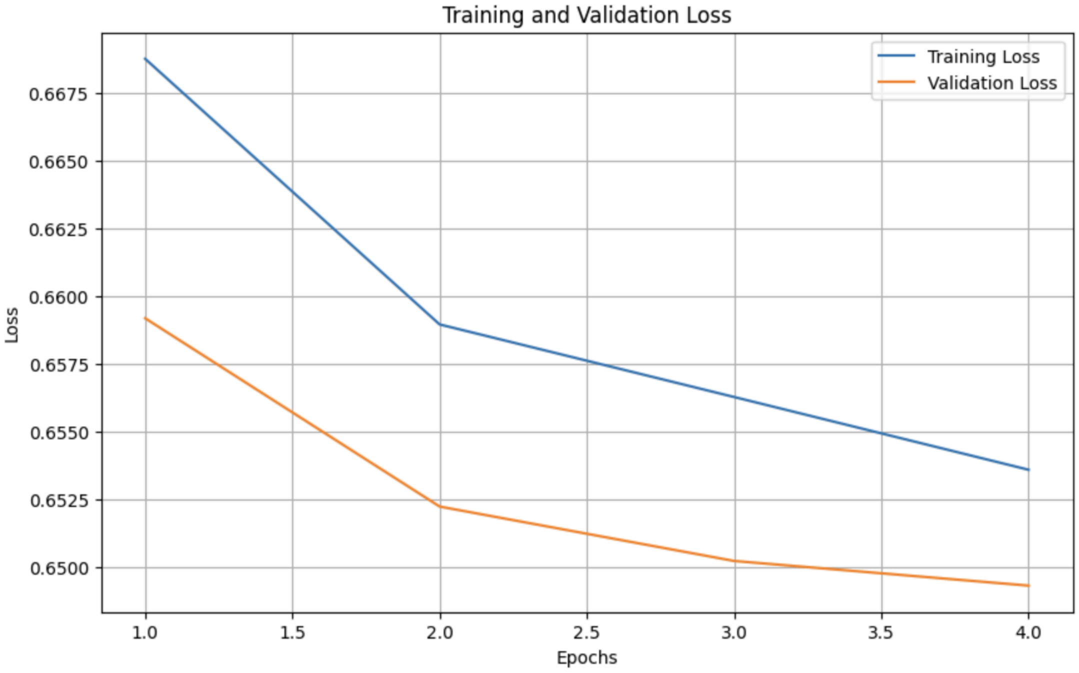
[103]:

```
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

[104]:

```python
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
disp_normalized = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
disp_normalized.plot(cmap=plt.cm.Blues)
plt.title('Normalized Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```





Conclusion:

The two tower accuracy is at 61% which is just about the top level that I was able to get with this project.

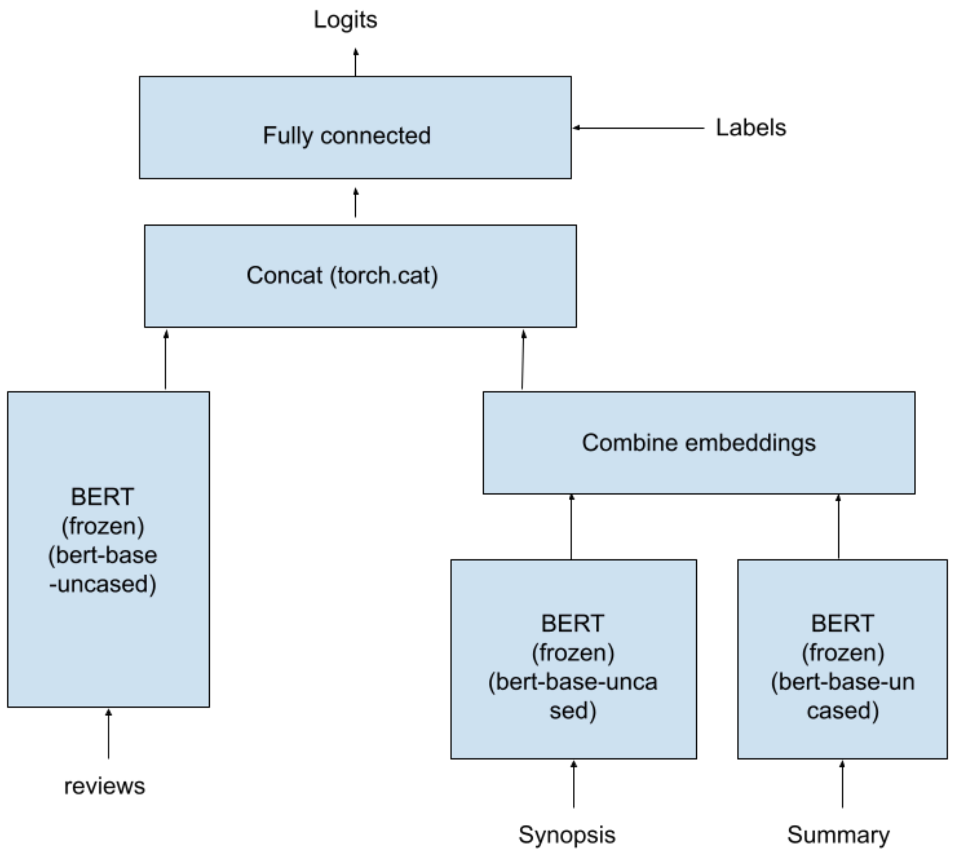
The large sample size and 2 tower method enables the model to converge to validation loss below .65 but still slightly underfit. Unfortunately, I didn’t have enough GPU time to go over 4 epochs.

Model 4

(presented in this notebook)

I am aware that the 3 BERT heads described below involve more overhead than the classic 2 tower model, but I wanted to investigate it and see if I could get better performance.

Model 4 Block diagram


Logits
Fully connected
Labels
Concat (torch.cat)
BERT (frozen) (bert-base -uncased)
Combine embeddings
BERT (frozen) (bert-base-unca sed)
BERT (frozen) (bert-base-un cased)
reviews
Synopsis
Summary

I tried several methods for combining the synopsis and summary

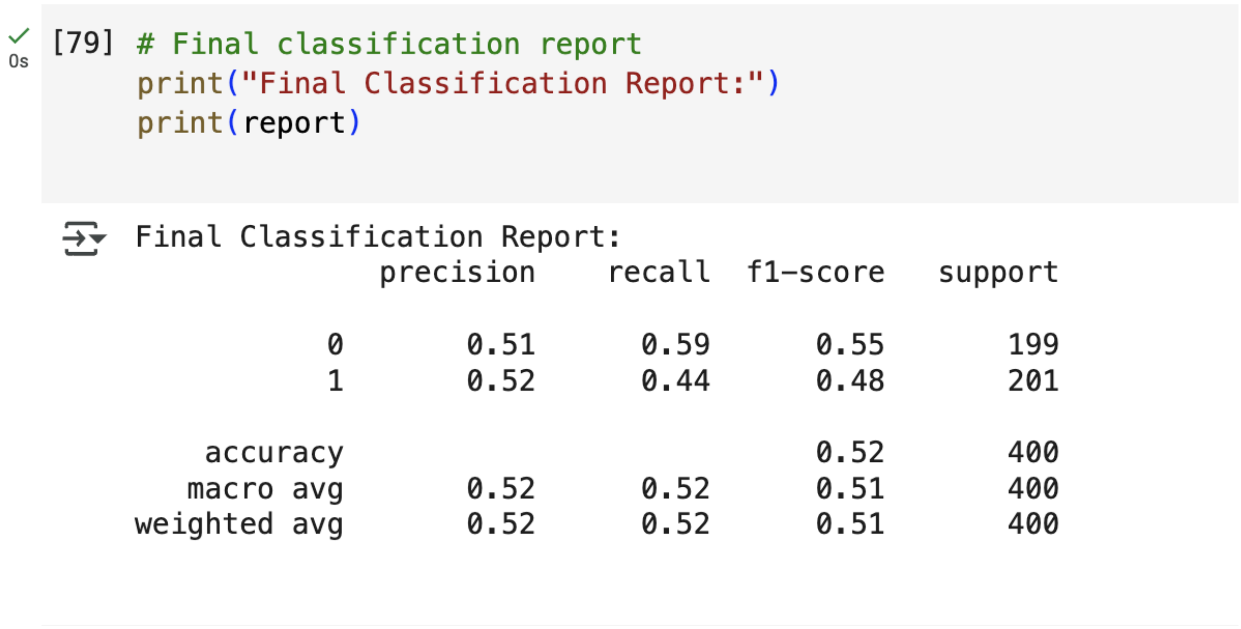
- Subtracting the synopsis and summary embeddings
- Adding the synopsis and summary embeddings
- Concatenating the summary and summary embeddings

None of these seemed to be dramatically better than the 2 tower method but the concatenation method seemed to provide the best results, so I went with it.

Model 4 . Concatenate synopsis and summary

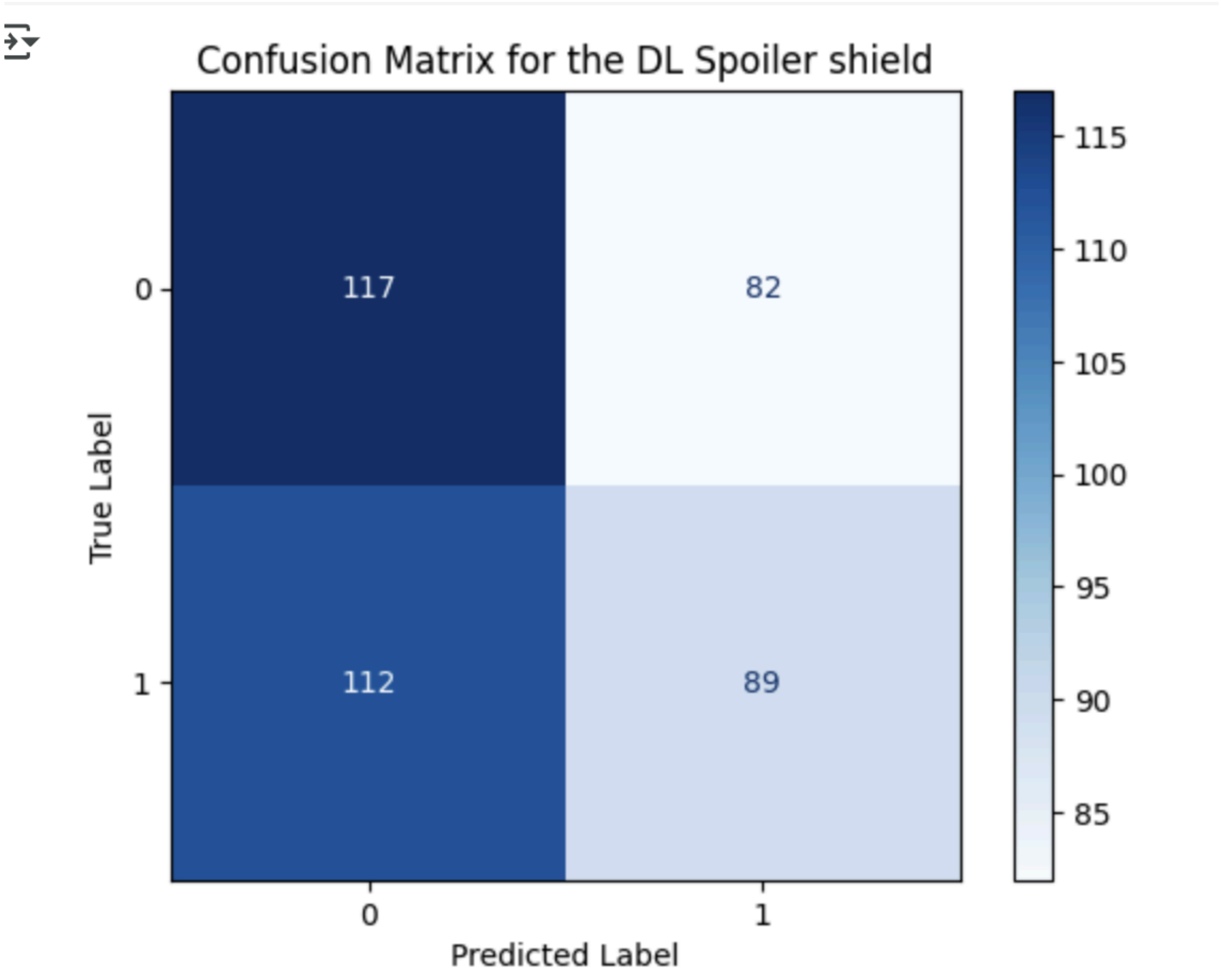
Huggingface Model = bert-base-uncased
Sample size **1000**

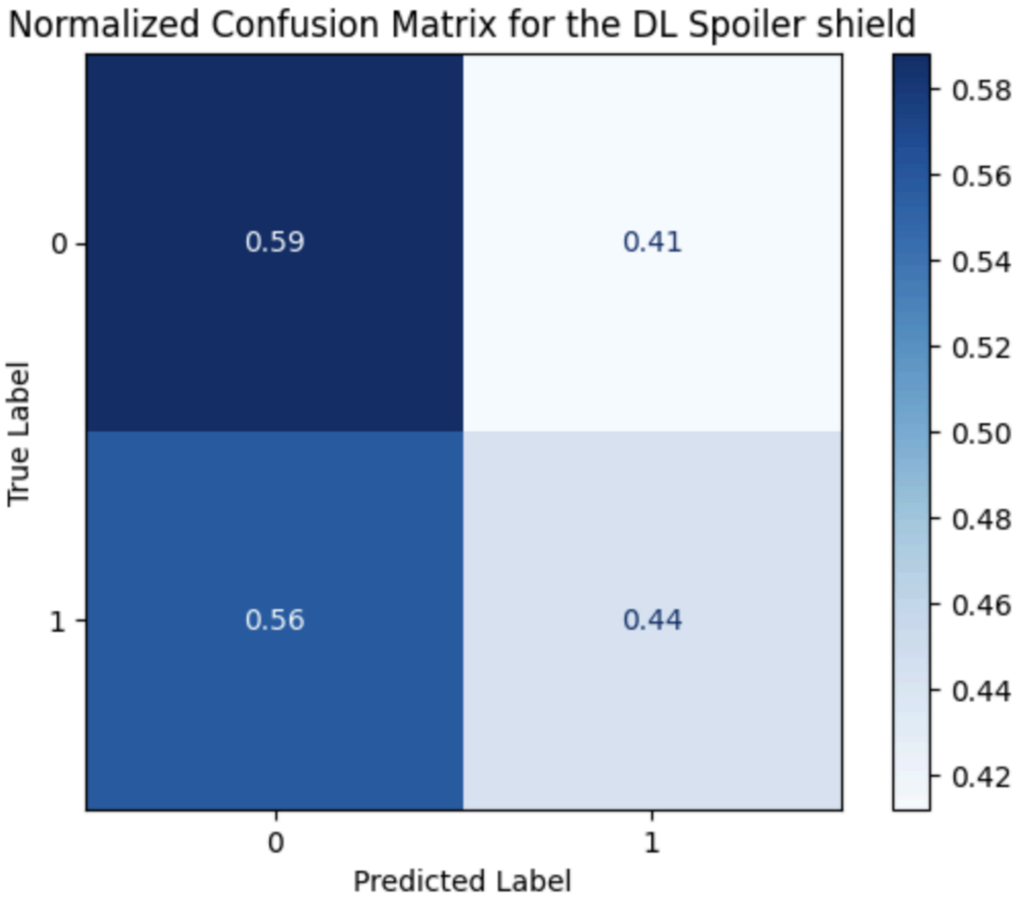
Balanced (True vs false are equal in number)
num_epochs=**4**
Learning_rate 5e-4



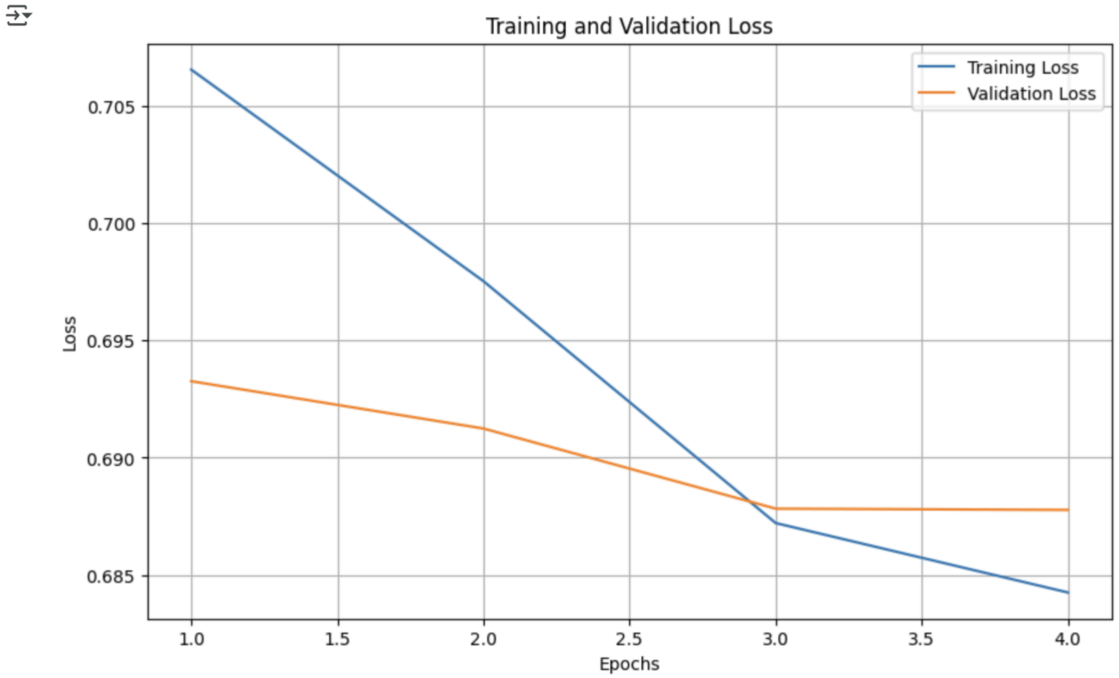
```
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.59      0.55       199
           1       0.52      0.44      0.48       201

    accuracy                           0.52       400
   macro avg       0.52      0.52      0.51       400
weighted avg       0.52      0.52      0.51       400
```


Confusion Matrix for the DL Spoiler shield
117
82
112
89
0
1
True Label
0
1
Predicted Label
115
110
105
100
95
90
85



Normalized Confusion Matrix for the DL Spoiler shield
True Label
Predicted Label
0
1
0
1
0.59
0.41
0.56
0.44
0.58
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.42



Conclusion: Model 4. Concatenate synopsis and Summary

I limited this model to 1000 sample size because I just wanted a quick test without using too much CPU time. The accuracy is actually not that great. It is not really better than the 2 tower model.

Model 4 . Concatenate synopsis and summary, 50000 samples

This is the model presented in the document

Huggingface Model = bert-base-uncased
Sample size **50000**
Balanced (True vs false are equal in number)
num_epochs=**4**
Learning_rate 5e-4

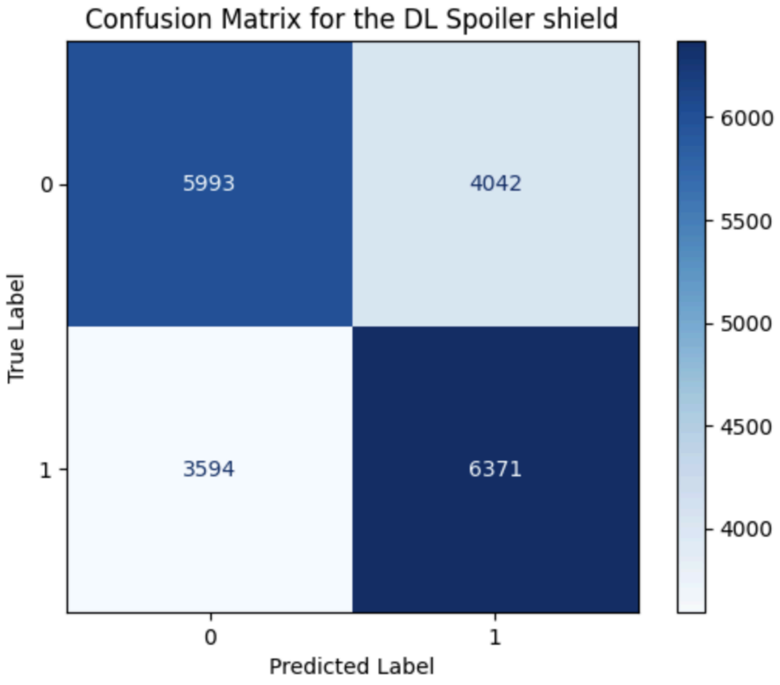
[39]:

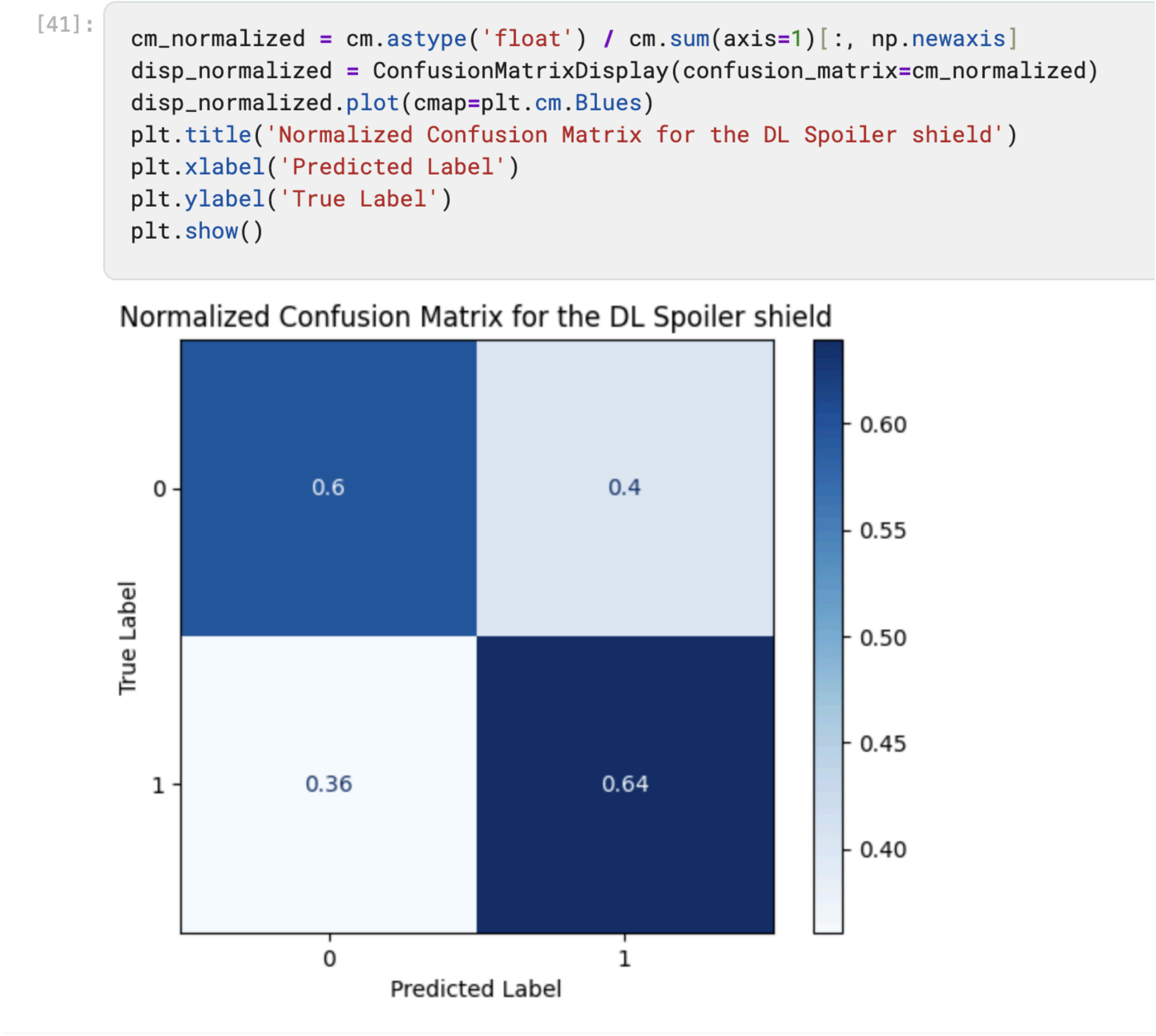
```
# Final classification report
print("Final Classification Report:")
print(report)
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.60      0.61     10035
           1       0.61      0.64      0.63      9965

    accuracy                           0.62     20000
   macro avg       0.62      0.62      0.62     20000
weighted avg       0.62      0.62      0.62     20000
```

[40]:

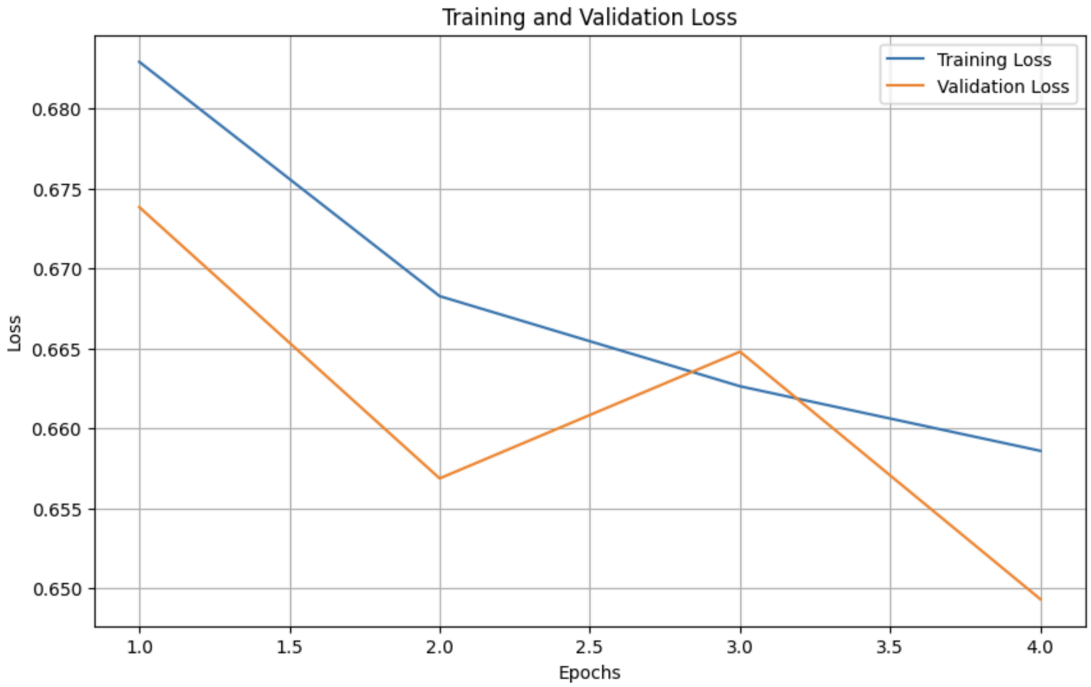
```python
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

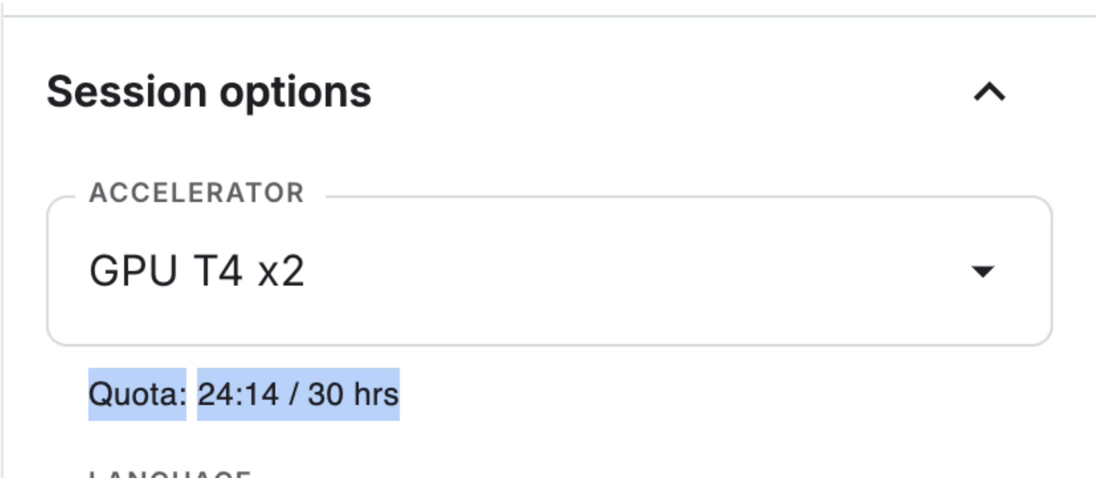




[41]:

```
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
disp_normalized = ConfusionMatrixDisplay(confusion_matrix=cm_normalized)
disp_normalized.plot(cmap=plt.cm.Blues)
plt.title('Normalized Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```





Session options

ACCELERATOR

GPU T4 x2

Quota: 24:14 / 30 hrs

LANGUAGE

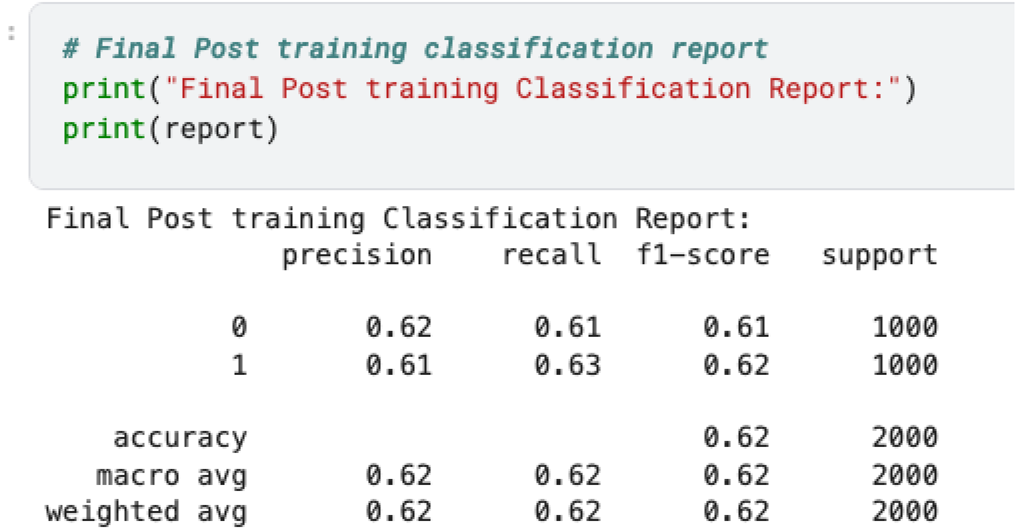
One training run for 4 epochs used up all this kaggle compute time!!

Conclusion: Model 4. Concatenate synopsis and Summary

The 3 tower model is very expensive in CPU time but not demonstrably much better than the 2 tower model which I did earlier. However, I think If I had more time, I could extract much more performance from this model.

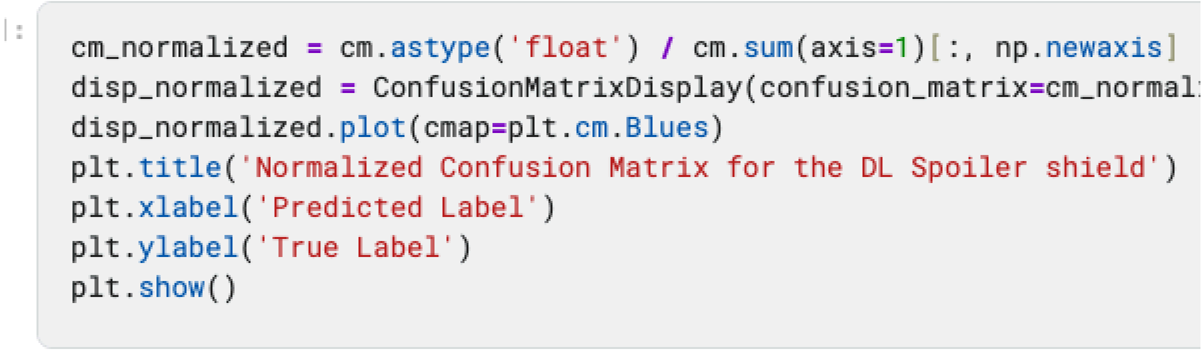
Model 4 Post training

I also saved this model and used it on brand new data and found that the Post training performance was consistent with the training

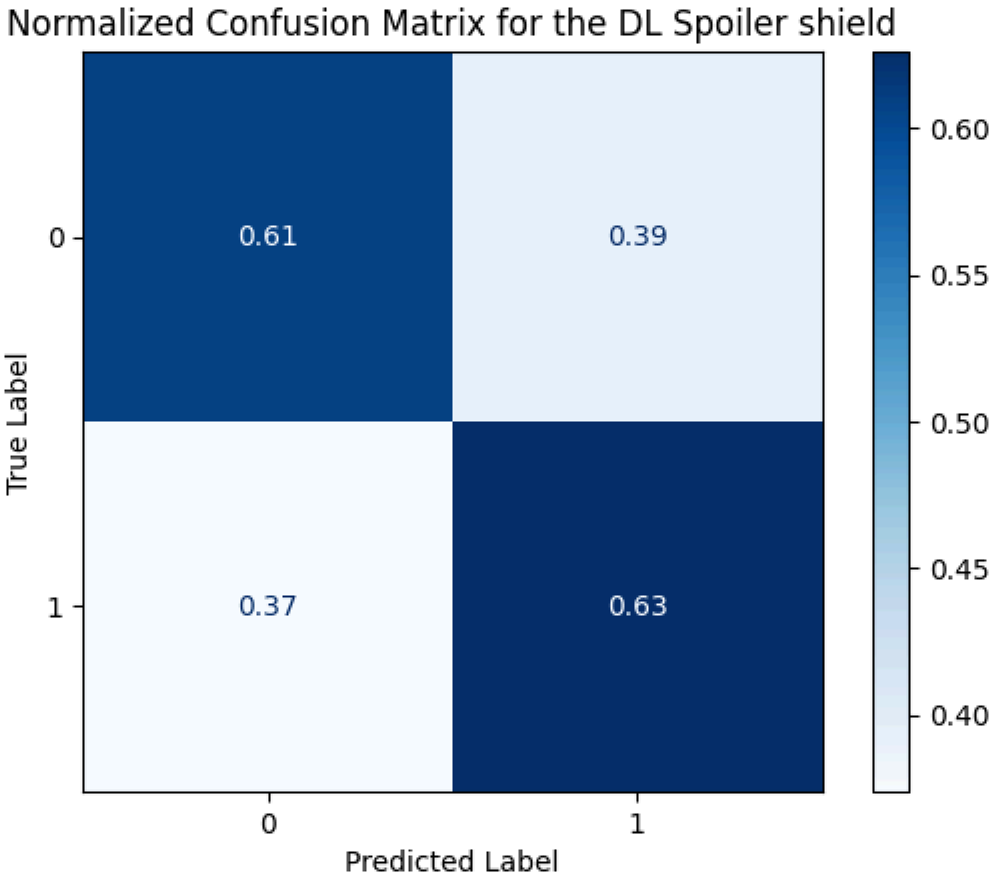


```
# Final Post training classification report
print("Final Post training Classification Report:")
print(report)
```

```
Final Post training Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.61      0.61      1000
           1       0.61      0.63      0.62      1000

    accuracy                           0.62      2000
   macro avg       0.62      0.62      0.62      2000
weighted avg       0.62      0.62      0.62      2000
```



```
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
disp_normalized = ConfusionMatrixDisplay(confusion_matrix=cm_normali
disp_normalized.plot(cmap=plt.cm.Blues)
plt.title('Normalized Confusion Matrix for the DL Spoiler shield')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Future Work

I would love to have had more time to pursue the 3- tower model. I think I could have extracted more performance there.

In retrospect, I should probably have used a lighter model like ALBERT to do initial training then switched to bert proper. This could have probably saved me some time and CPU.

I also tried BERT for NSP but many of the texts kept getting truncated, invalidating the runs. I am sure there is a way to fix this but didn’t have time to pursue this.

The is_spoiler data is probably noisy. I think that overall the system performance is good.

Overall, this was a great project. It got me hands on with BERT. I can’t wait to use more sophisticated models in upcoming projects.