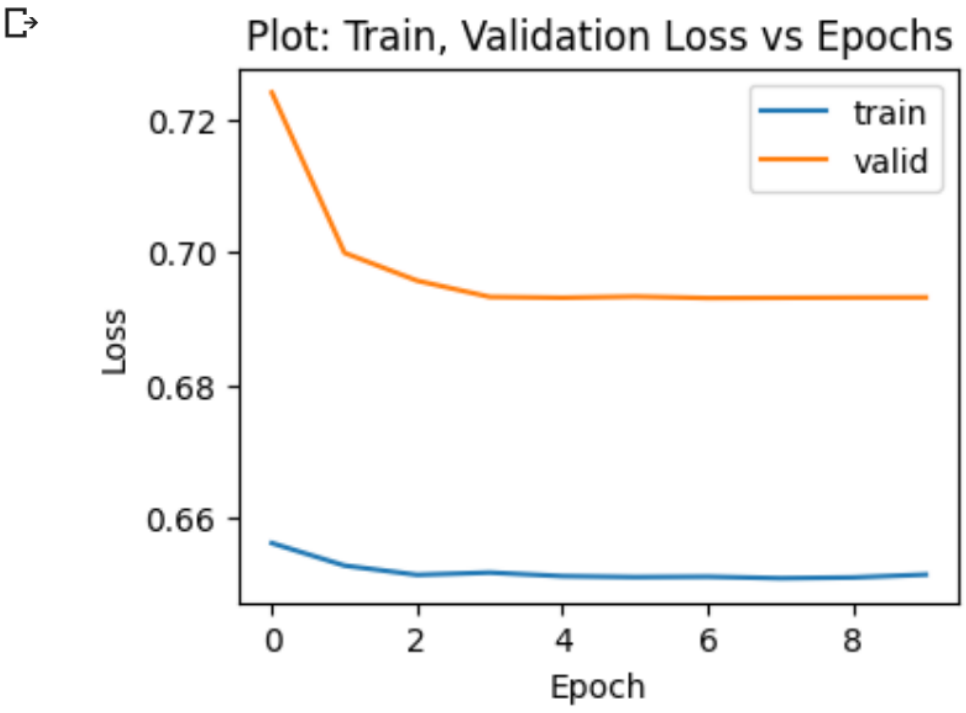
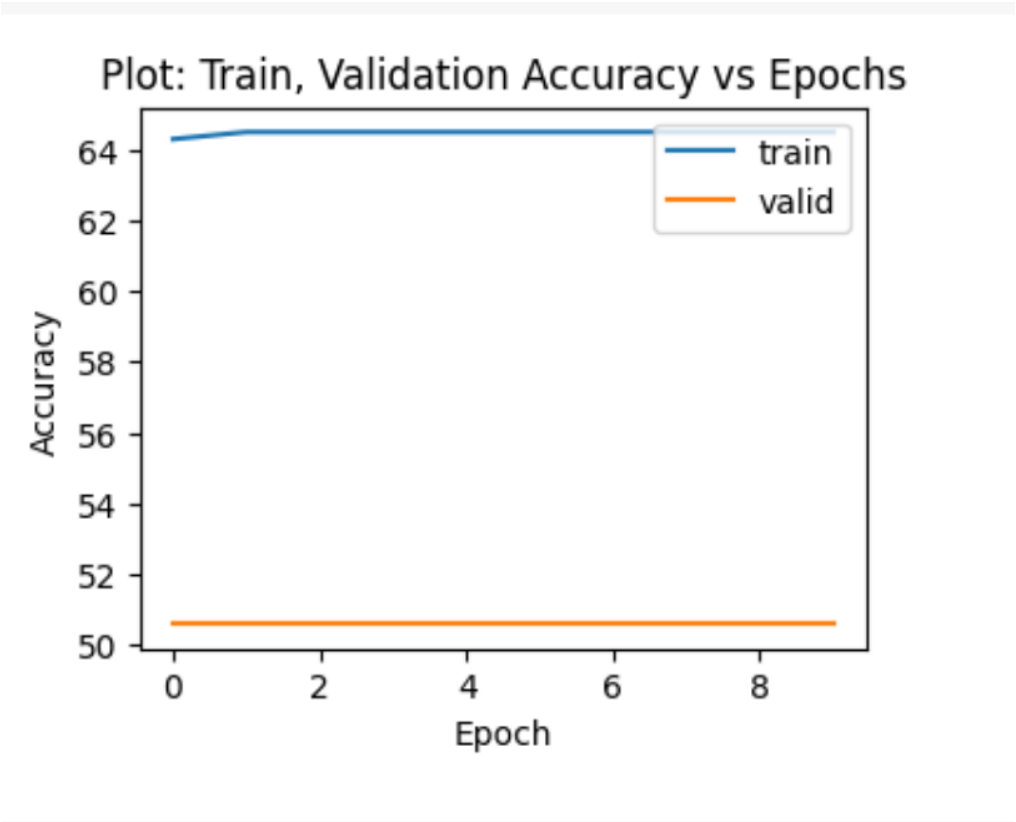
REPORT

1. Image-only Classification:
    a) The customly implemented architecture is the VGG16 model, with 5 different blocks (each containing roughly Conv2D layers and a pooling layer) with 2 final dense layers and a flattened layer and finally, training on the 8500 training samples.
    b) The preprocessing steps include normalization, grayscaling and resizing inputs according to the model architecture.
    c) The training roughly provided the accuracy of around 50% with a decrease in the loss, while the validation loss tended to increase owing to the need of more training samples and better optimization of the model and maybe unequal distributions of samples of class0 and class1.

Plots:





Analysis:
Both training and validation loss curves are decreasing as the no. of epochs increases. Initially losses decrease and model learns increases then tend to saturate at a particular value. After that losses do not decrease for training and testing dataset.

Incase of accuracy vs epoch curve for training dataset model initial start increases its accuracy and then saturates.
Incase of accuracy vs epoch curve for validation dataset model accuracy saturates as increase in number of epochs.

d)

```
[ ]  print("Accuracy = {}".format(round(accuracy_score(y_test, y_pred), 2)))
     print("Precision = {}".format(round(precision_score(y_test, y_pred), 2)))
     print("Recall = {}".format(round(recall_score(y_test, y_pred), 2)))
     print("F1-score = {}".format(round(f1_score(y_test, y_pred), 2)))
```

```
Accuracy = 0.51
Precision = 0.0
Recall = 0.0
F1-score = 0.0
```

```
print(metrics.classification_report(y_test, y_pred, digits=2))
```

```
              precision    recall  f1-score   support

         0.0       0.51      1.00      0.68       510
         1.0       0.00      0.00      0.00       490

    accuracy                           0.51      1000
   macro avg       0.26      0.50      0.34      1000
weighted avg       0.26      0.51      0.34      1000
```

2. Text-only Classification:

a) Pre-processing:

We are using pip install transformers which is a helpful library for transformers. After that, we call all required libraries. Then after Loading the data by the given link. After that, I print the train test and val data. Import BERT tokenizer to tokenize the data and make sentids for each data set which helps evaluate our model. Define the data labels for each data set. We also define the variable for each set, like input ids, attention masks and data labels. Create a data loader for each data set. After that, freeze all the BERT parameters.

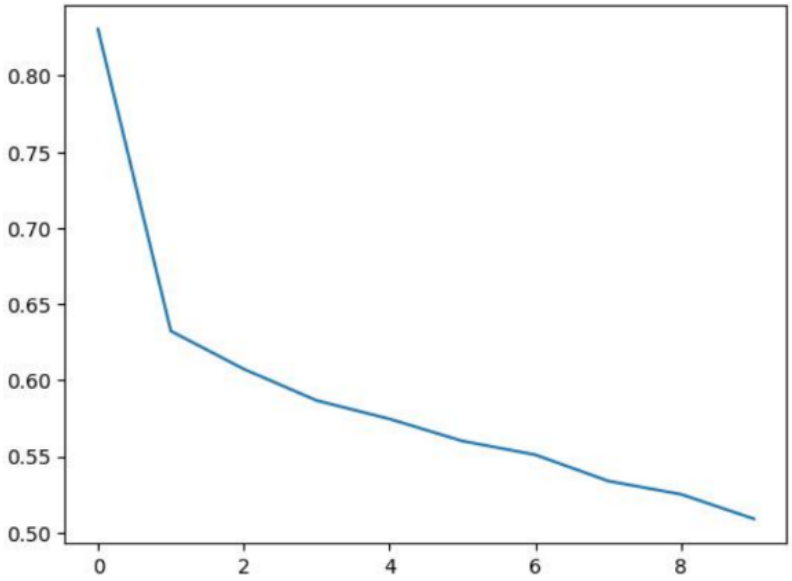
b) Approach:

Define the model architecture of bert_arch and connect with cuda. We will use Adam optimizer and class weights.array([0.77540595, 1.40775091]). convert class weights in tensor. After that, fine-tune the model with all total accuracy. We call sklearn metrics for the confusion matrix. After that, we train the model on ten epochs and print training loss, validation loss, training and val accuracy.

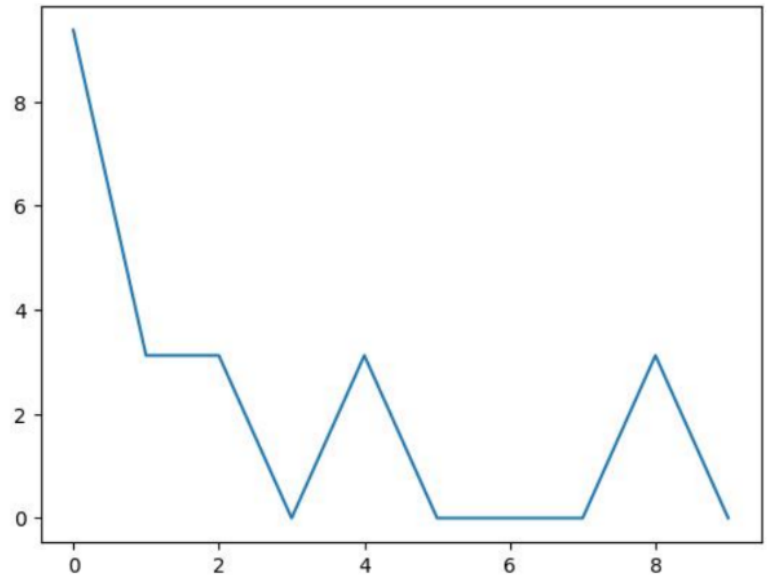
c) Results:

Training loss decreases with epochs, so our model performs best for the training set.val loss is not stable with epochs, for two epochs, its got the highest loss, but after that, it decreased. Train accuracy is also increasing for some epochs and decreasing for some epochs, but val accuracy is constant. So our model is good, at most that, according to our evaluation.

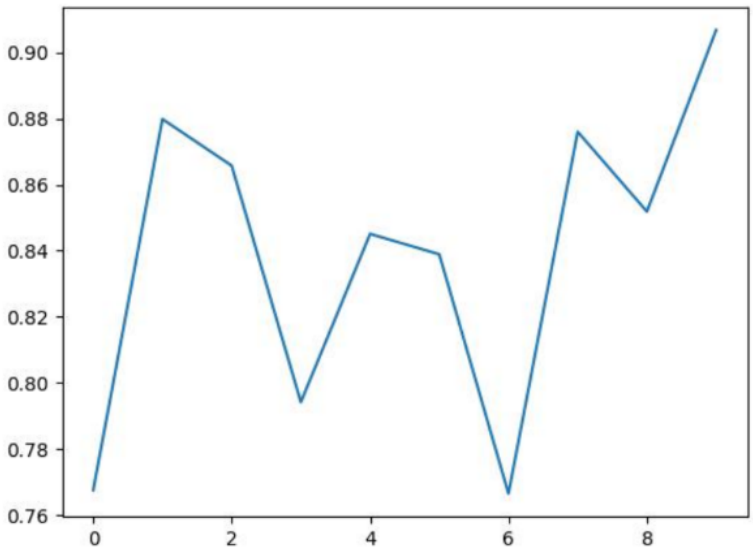
Training Loss vs Epochs:



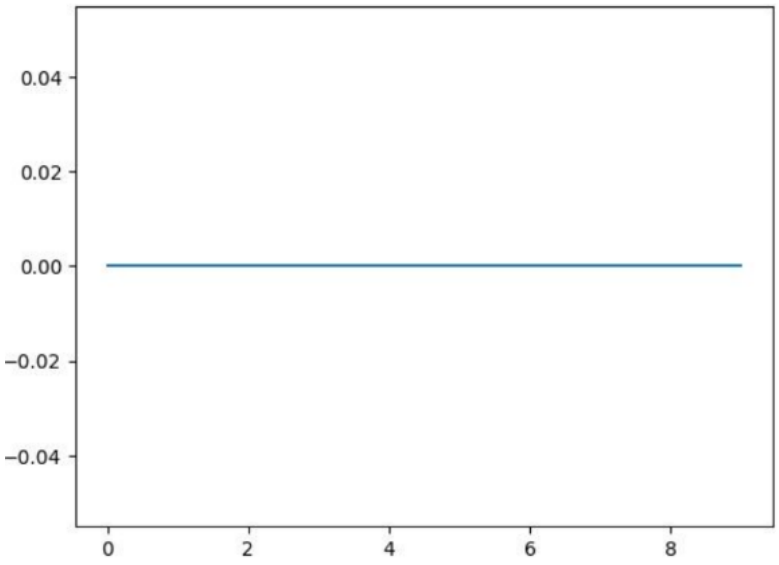
Training Accuracy vs Epochs:



Val Loss vs Epochs:



Val Accuracy vs Epochs:



3. Multimodal Classification:

Pre-processing Steps:

a) Image: Grayscaling, Resizing, Normalization.
b) Text: Tokenization, Padding, Text Embedding (generating matrices using One-Hot encoding), with max_word=5000 and max_len=400.

Text classification model:

The model incorporated is the classic LSTM based model with one embedding layer and 512 LSTM cells, with relu as the activation.

Image Classification Model:

The model implemented is a generic CNN based model with 3 convolution layers, 3 pooling layers, one flattened dense layer, all with relu as the activation.

Fusion:

The fusion of the above-mentioned models is the early-fusion where the extracted features from both the models get concatenated into one, and fed onto the final flattened dense layer with softmax as the activation for final predictions.

Overall Scores

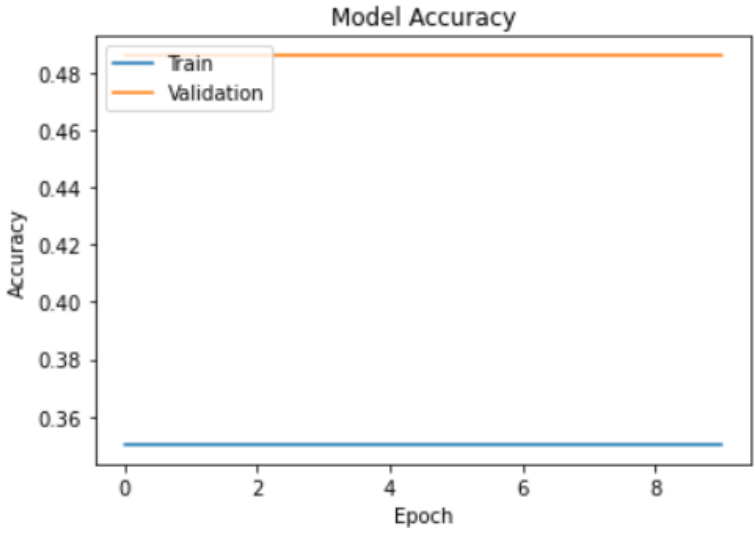
a) Fused Model:

Accuracy: 0.5142857142857142
Precision: 0.5142857142857142
Recall: 1.0
F1 score: 0.6792452830188679

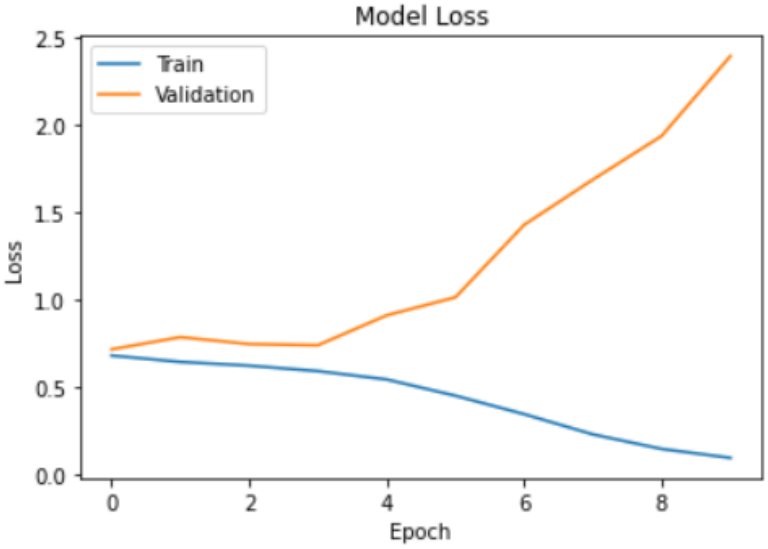
Conclusion:

The overall performance of the fused model is surely better than the individual model performances (which had accuracies around 35% for train set and around 48% for the val set), but the stable training and val accuracies and an increased val loss depicts that the training done is not fully optimized.

Train and Val Accuracies vs Epochs:



Train and Val Losses vs Epochs:

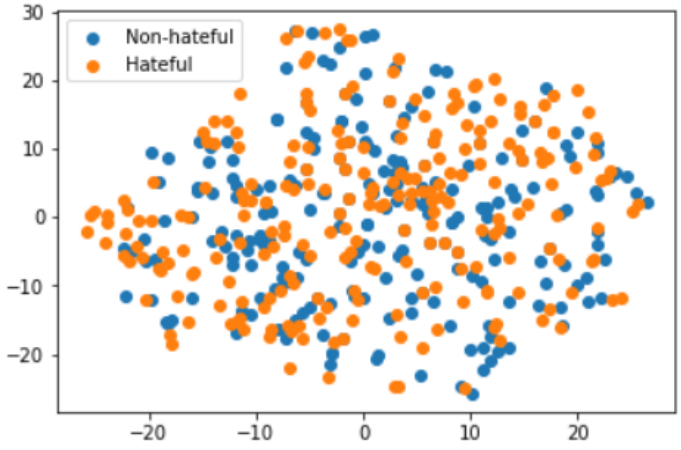


Class-Wise Scores:

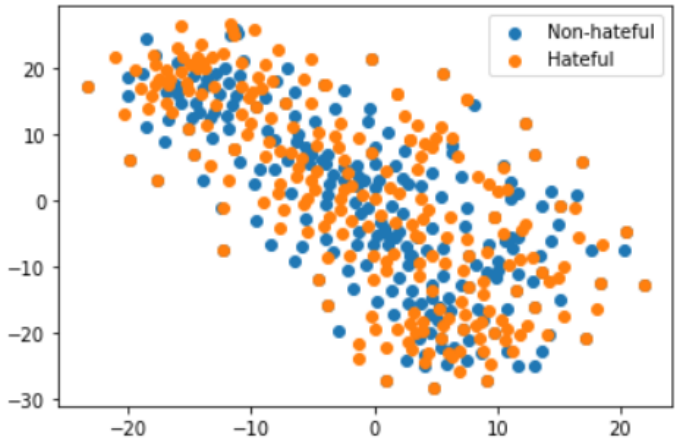
	precision	recall	f1-score
0	0.00	0.00	0.00
1	0.51	1.00	0.68
accuracy			0.51
macro avg	0.26	0.50	0.34
weighted avg	0.26	0.51	0.35

T-SNE Plots:

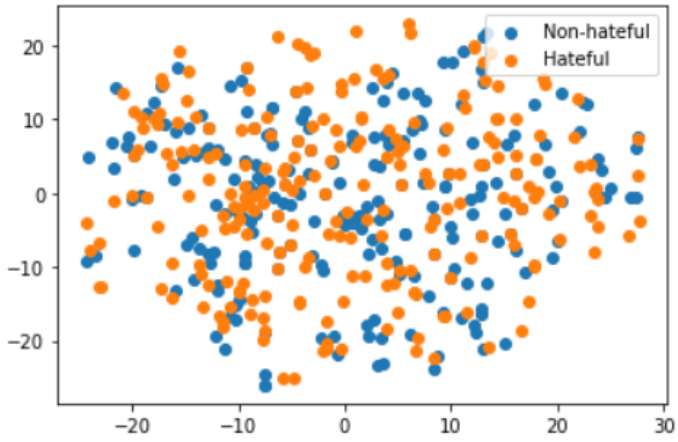
a) Text-Only Classifier:



b) Image-only Classifier:



c) Fused Model Classifier:



The t-sne plot for the fused model classifier has the variance in-between the text-only and image-only models and the bifurcation between the classes is more visible than the other two which shows that the classification is appropriate in this case, not overfitting, not underfitting.

Early-Fused Model Architecture: