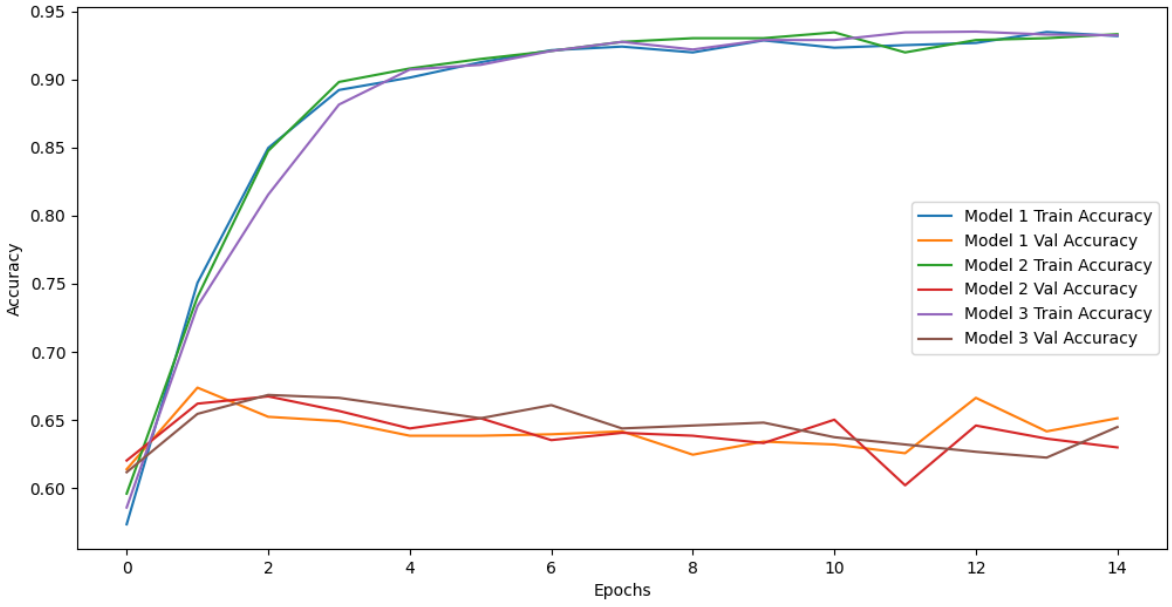
Thomas Lamont
CS4275
Small Project 3

The goal for this project was to implement an LSTM model for financial sentiment analysis. This sentiment was gathered from 5843 financial statements labeled either positive, negative or neutral. To start I tokenized the sentences to convert the raw text into a numerical form that the LSTM model can process.

After tokenization I started with some normal parameters I grabbed from examples on the internet and was immediately greeted with overfitting. The analysis model was doing significantly better on the training data than the test data. My main goal after this baseline was to reduce overfitting by tuning the parameters and hyperparameters of the model. Some causes are likely that the model is too complex, it lacks regularization, the model is too imbalanced or too limited.



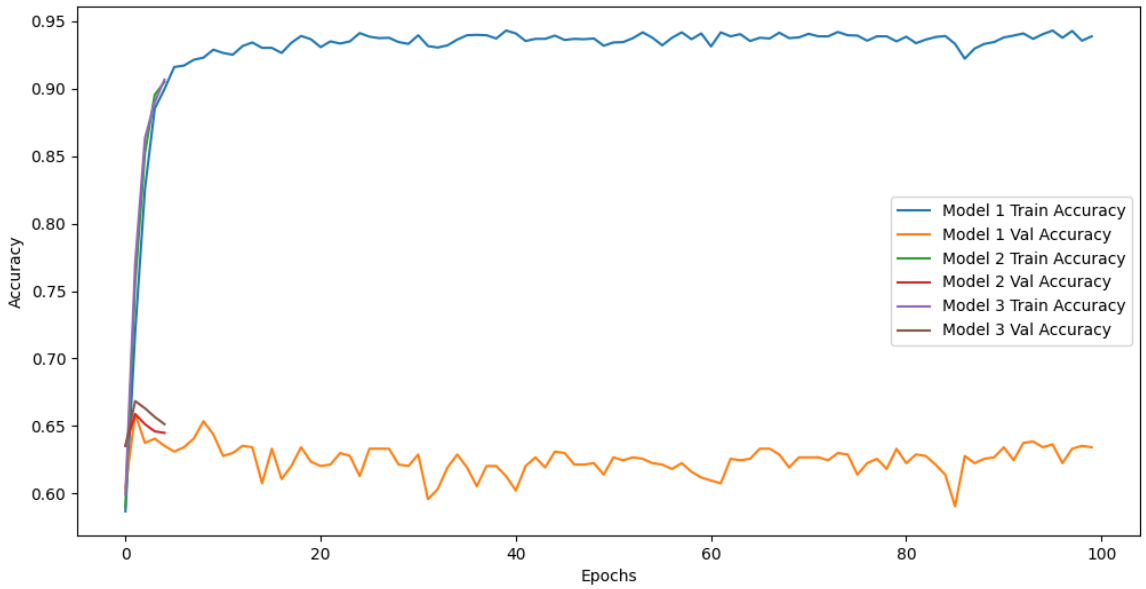
MODEL 1 - ACCURACY: 0.6629598140716553 LOSS: 1.768964409828186

MODEL 2 - ACCURACY: 0.6629598140716553 LOSS: 1.8302464485168457

MODEL 3 - ACCURACY: 0.6732249855995178 LOSS: 1.589659571647644

Thomas Lamont
CS4275
Small Project 3

To drive home the overfitting, I ran my first model for 100 epochs to see where the accuracy plateaus. With these initial models as a baseline I set out to improve upon the issues.



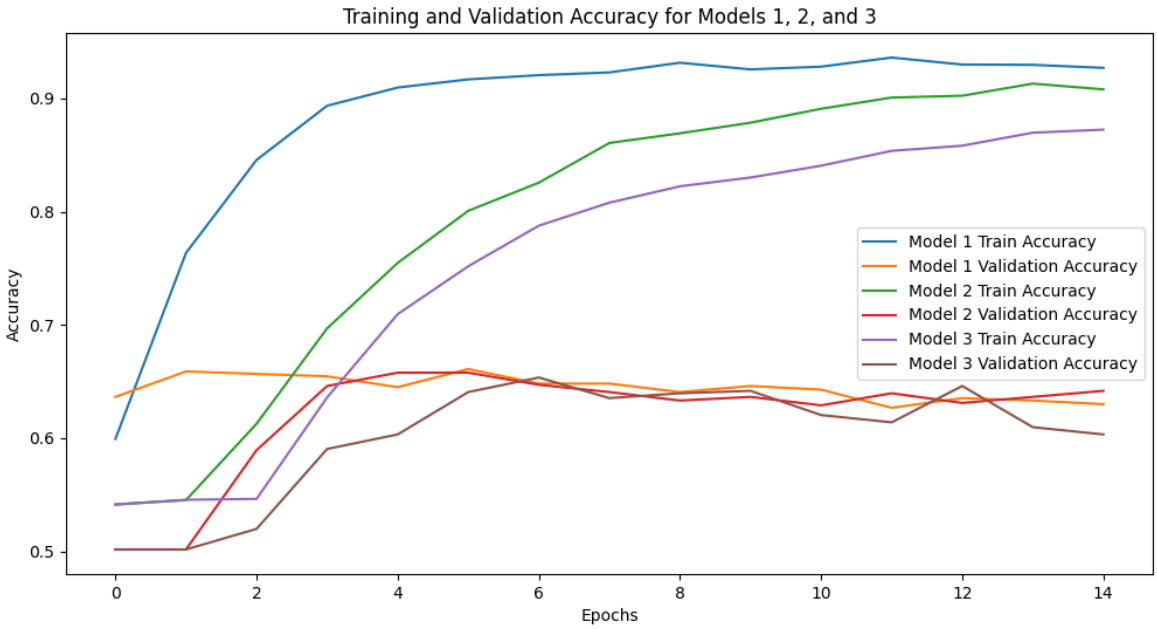
MODEL 1 - ACCURACY: 0.6629598140716553 LOSS: 2.9439449310302734

Thomas Lamont
CS4275
Small Project 3

For simplification, in my first model I reduced the number of LSTM layers from 2 to 1, this simplifies the architecture. I also added a flattening layer after the LSTM layer before passing the data to the dense layer, further simplifying the data. The goal of this model was simplification. If the old model was too complex, it would be more prone to overfitting.

For the second model I added recurrent dropout to the LSTM layer as well as regularization. Both changes also aimed to reduce complexity. Instead of the flatten layer from model 1, I used GlobalAveragePooling1D again reducing output size and complexity.

In model 3 the filter size and kernel size were reduced and L2 regularization was implemented.



MODEL 1 - ACCURACY: 0.6749358177185059 LOSS: 1.6790037155151367

MODEL 2 - ACCURACY: 0.6663815379142761 LOSS: 1.1321494579315186

MODEL 3 - ACCURACY: 0.6304534077644348 LOSS: 1.3573684692382812

Thomas Lamont
CS4275
Small Project 3

The changes overall had some effect, but the results were quite similar to the initial runs.

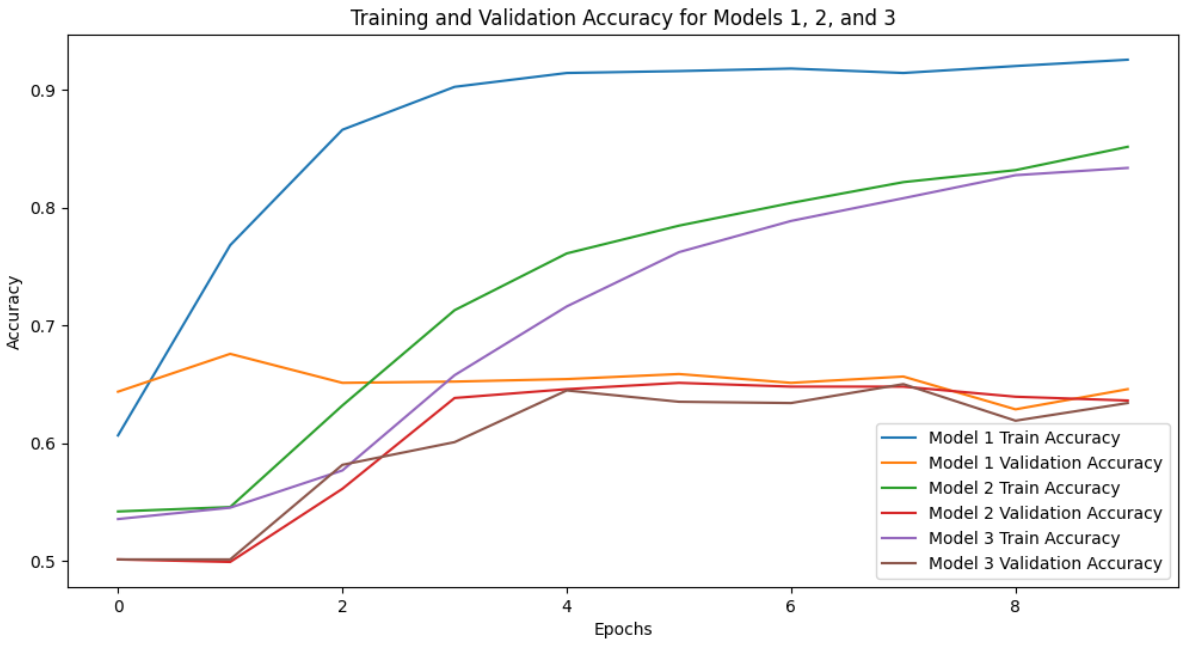
I tried reducing the size of the test split to .1 so that I had more training data, this decreased accuracy, possibly because the larger set meant more overfitting and there was less data to validate the model on.

MODEL 1 - ACCURACY: 0.6484641432762146 LOSS: 1.8590898513793945

MODEL 2 - ACCURACY: 0.6484641432762146 LOSS: 1.2274656295776367

MODEL 3 - ACCURACY: 0.5426621437072754 LOSS: 0.9716310501098633

I noticed that for the models the training accuracy began to plateau at around 10 epochs. I reduced the number of epochs from 15 to 10 and this is where I ultimately got my best results.



MODEL 1 - ACCURACY: 0.6749358177185059 LOSS: 1.6006885766983032

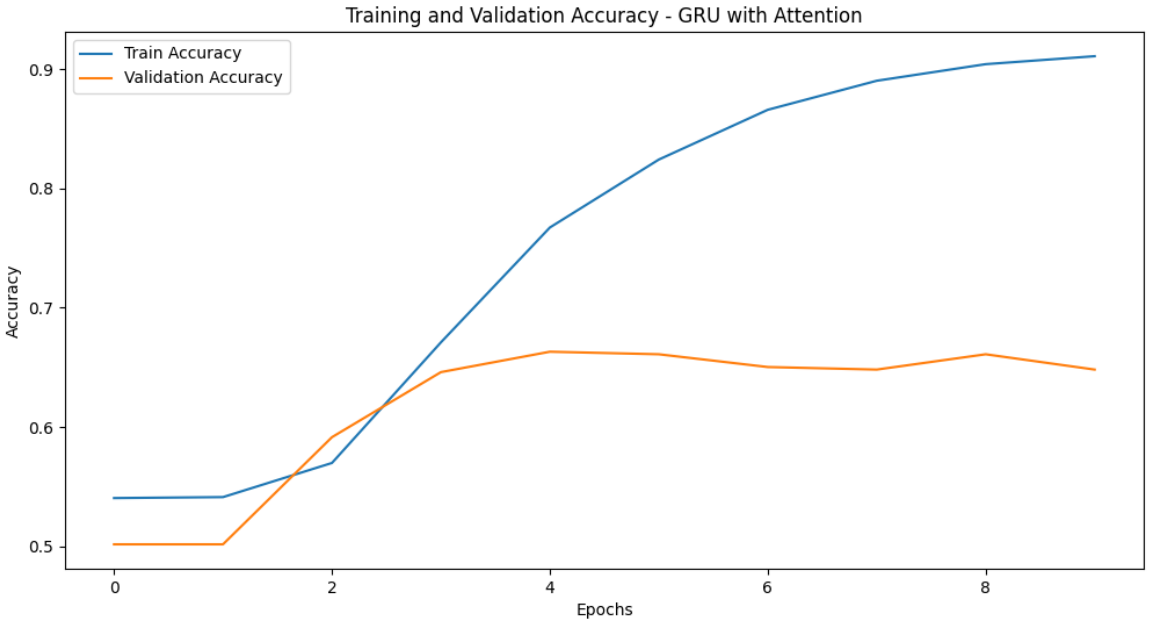
MODEL 2 - ACCURACY: 0.6809238791465759 LOSS: 0.9155956506729126

MODEL 3 - ACCURACY: 0.6663815379142761 LOSS: 1.0555953979492188

Thomas Lamont
CS4275
Small Project 3

Model 2 ended up being the most balanced with its use of dropout layers, GlobalAveragePooling and smaller kernel size.

Out of curiosity I tried implementing a GRU with Attention model on the same data set. The idea being that the attention mechanism could help highlight the most informatize parts of the sentence. With less work in tuning the parameters, the GRU model performed slightly better than the LSTM models, but the disparity between the accuracy in the train and test splits was still equally present.



GRU WITH ATTENTION - ACCURACY: 0.6928999423980713 LOSS: 1.2269393205642