Spotify Playstore Sentiment Analysis

Table of Contents

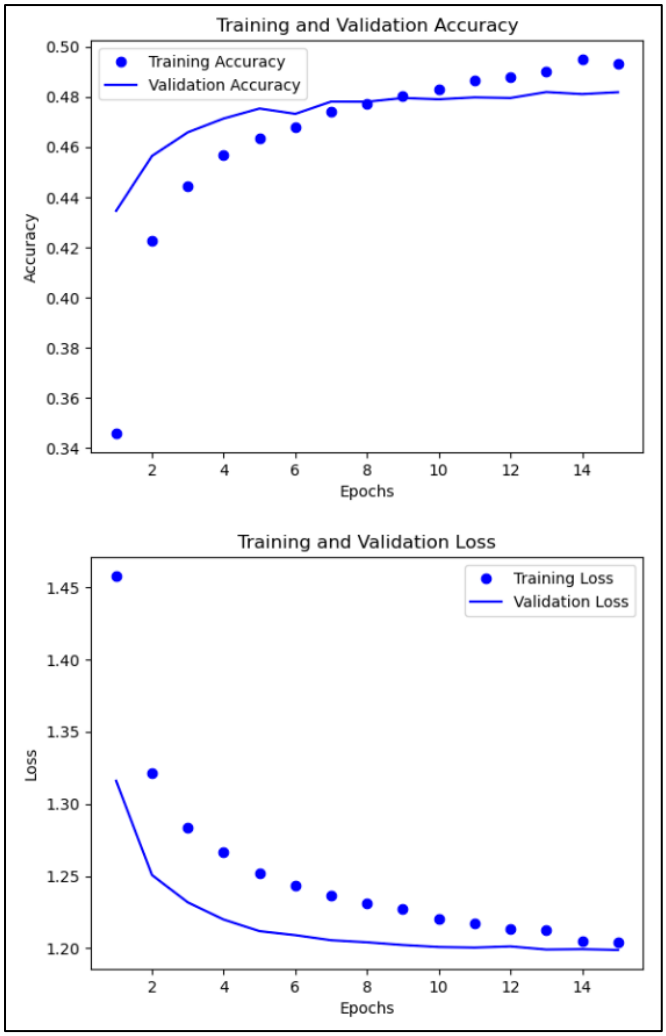
Model 1

```
# Build the LSTM model
test4_model = Sequential()
test4_model.add(keras.Input(shape=(maxlen,)))
test4_model.add(Embedding(10000, 32))
test4_model.add(LSTM(16, dropout=0.3, recurrent_dropout=0.3))
test4_model.add(Dropout(0.5))  # Helps prevent overfitting
test4_model.add(Dense(5, activation='softmax'))
test4_model.summary()

# Train the Model
test4_model.compile(optimizer = RMSprop(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history = test4_model.fit(X_train, y_train,
                    epochs=15,
                    batch_size=32,
                    validation_split=0.2)
```

After completing the data cleaning and processing, I started with a simple model architecture by adding an Embedding layer to process the text data, followed by a single LSTM layer with 16 units as I thought that seemed a reasonable starting point. To address potential overfitting, I incorporated dropout both within the LSTM layer and after the subsequent Dense layer. For optimization, I chose RMSprop with a learning rate of 0.0005, and since the labels are encoded as integers, I used sparse categorical cross-entropy as the loss function. The training was conducted over 15 epochs with a batch size of 32.



Looking at the accuracy plot, both training and validation accuracy improve steadily, plateauing around 0.48 to 0.49. The validation accuracy closely follows the training accuracy, which is a good sign that the model isn't overfitting too badly.

Both training and validation loss decrease consistently, which means the model is learning. By the end of the 15th epoch, the validation loss has completely flattened, suggesting that additional training may or may not lead to significant improvements.

Given this, my next step was to extend the training to 20 epochs using the same model architecture. This is because, since the loss was still decreasing and accuracy hadn't hit a hard ceiling, more epochs might squeeze out some extra performance. So, the goal was to see if five more epochs offered meaningful improvements or if the model had reached its limit with this configuration.

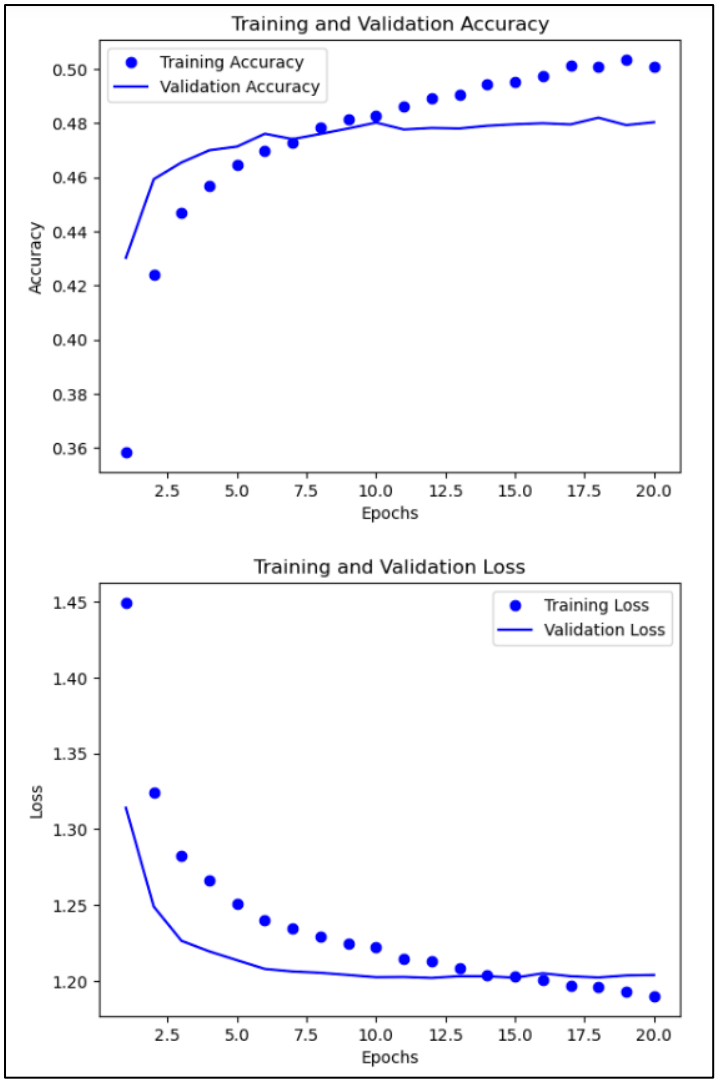
Model 2

```
test5_model = Sequential()
test5_model.add(keras.Input(shape=(maxlen,)))
test5_model.add(Embedding(10000, 32))
test5_model.add(LSTM(16, dropout=0.3, recurrent_dropout=0.3))
test5_model.add(Dropout(0.5))  # Helps prevent overfitting
test5_model.add(Dense(5, activation='softmax'))
test5_model.summary()

# Train the Model
test5_model.compile(optimizer = RMSprop(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history = test5_model.fit(X_train, y_train,
                    epochs=20,
                    batch_size=32,
                    validation_split=0.2)
```

The same model architecture as the first model was used, but the number of epochs increased from 15 to 20 to see if further training would yield better performance, or if the model had already reached its limit with this configuration.



This is the second model, where I increased the number of epochs from 15 to 20 to observe its impact on performance. Looking at the accuracy plot, while training accuracy continues to improve steadily, validation accuracy shows only slight improvement and starts to plateau after the 10th epoch. This suggests that while the model is learning patterns in the training data more effectively, its ability to generalize to unseen data isn't improving as much, potentially indicating diminishing returns or slight overfitting.

On the other hand, the loss plot shows a steady decline in both training and validation loss, which suggests the model is still learning. However, since validation accuracy is not increasing proportionally, it indicates that the model may not be learning more meaningful generalizable patterns. This suggests that simply increasing the number of epochs is not significantly enhancing validation performance.

Model 3

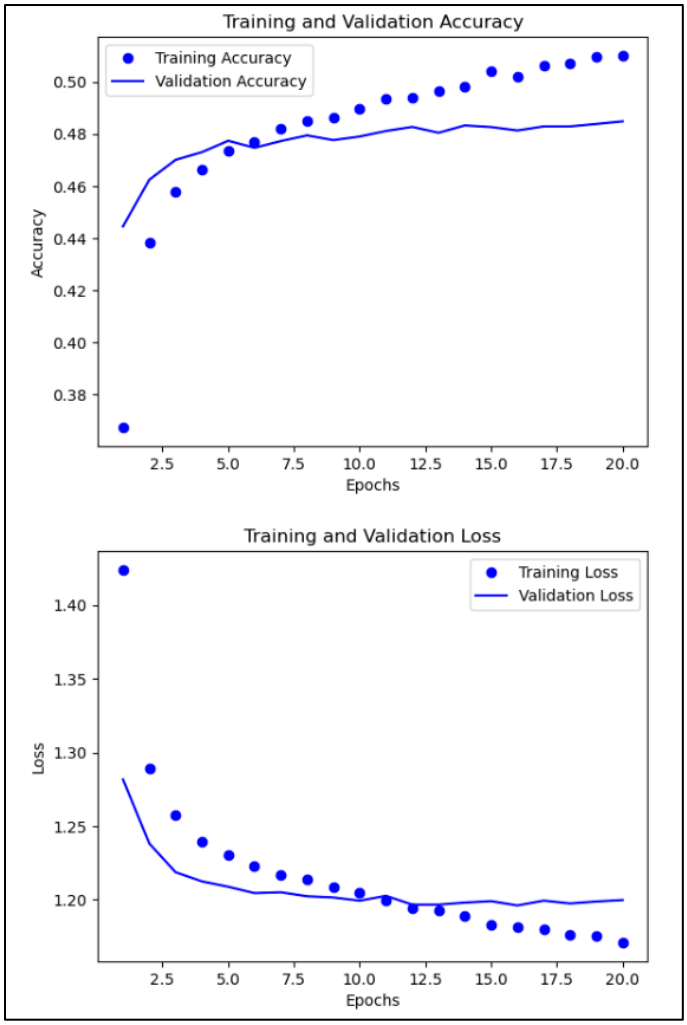
```
# Build the LSTM model
test8_model = Sequential()
test8_model.add(keras.Input(shape=(maxlen,)))
test8_model.add(Embedding(10000, 32))
test8_model.add(LSTM(32, dropout=0.3, recurrent_dropout=0.3))
test8_model.add(Dropout(0.5))  # Helps prevent overfitting
test8_model.add(Dense(5, activation='softmax'))
test8_model.summary()

# Train the Model
test8_model.compile(optimizer = RMSprop(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history_test8 = test8_model.fit(X_train, y_train,
                    epochs=20,
                    batch_size=32,
                    validation_split=0.2)
```

For my third model iteration, I aimed to boost performance by increasing the capacity of the LSTM layer. In my initial model, I used only 16 LSTM units, and although it demonstrated some learning, the accuracy fell short of expectations. The second model retained the same architecture but was trained for additional epochs.

In this third version, dubbed test8_model, I doubled the LSTM units to 32. I maintained the same embedding layer settings and dropout layers and trained the model over 20 epochs at a learning rate of 0.0005. I decided against modifying too many parameters simultaneously to clearly assess the impact of the increased LSTM capacity. I wanted to determine whether the enhanced LSTM capacity would lead to improved performance or simply result in overfitting and excessive memorisation of the training data.



Looking at the curves for the third model, I saw that the training and validation accuracy, as well as the loss curves, were quite similar in shape. This suggests the model is learning consistently across both sets, with no major overfitting or underfitting. I expected that doubling the LSTM units might help the model capture more complex patterns. But since this and the previous model's training and validation curves mirror each other so closely, it shows that simply adding more LSTM units isn't enough to improve overall performance. Thus, I would need to explore other modifications, like trying a new architecture for the next model.

Model 4

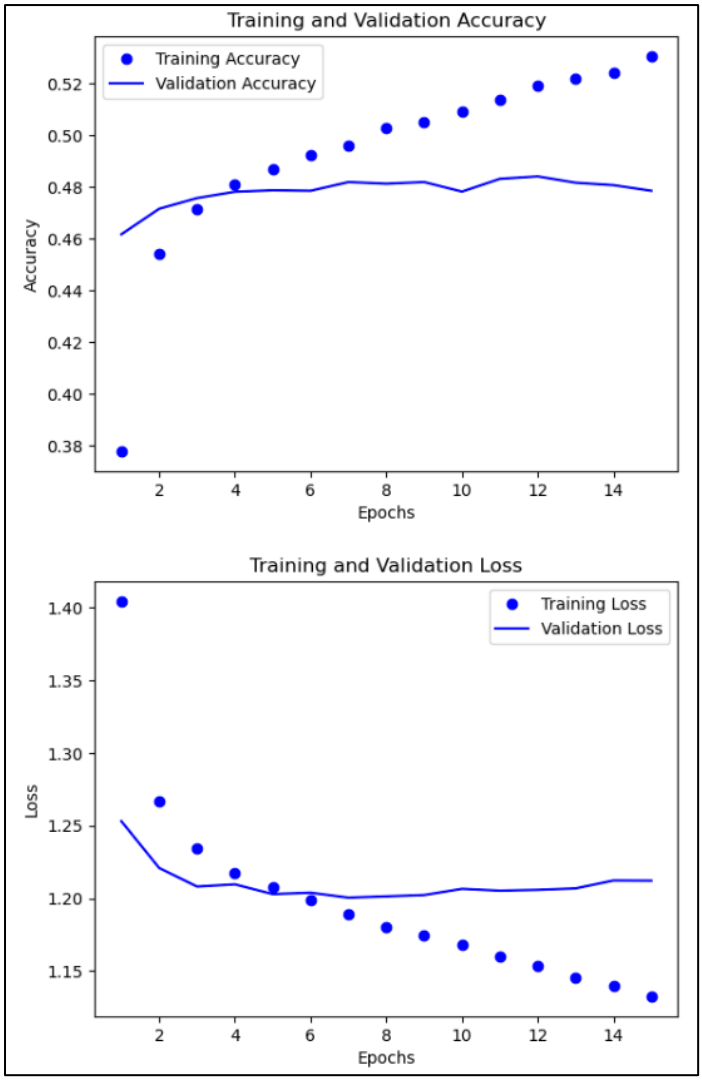
```
# Build the hybrid model: Conv1D + LSTM
hybrid_model = Sequential()
hybrid_model.add(keras.Input(shape=(maxlen,)))
hybrid_model.add(Embedding(10000, 32))
hybrid_model.add(Conv1D(64, kernel_size=5, activation='relu'))
hybrid_model.add(MaxPooling1D(pool_size=2))
hybrid_model.add(LSTM(32, dropout=0.3, recurrent_dropout=0.3))
hybrid_model.add(Dropout(0.5))  # Helps prevent overfitting
hybrid_model.add(Dense(5, activation='softmax'))
hybrid_model.summary()

# Train the Model
hybrid_model.compile(optimizer = RMSprop(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history = hybrid_model.fit(X_train, y_train,
                    epochs=15,
                    batch_size=32,
                    validation_split=0.2)
```

I decided to try something different with my fourth model. I used a hybrid architecture, combining a Conv1D layer (CNN) with an LSTM layer (RNN). The idea is that Conv1D captures local patterns in the text, while the LSTM handles long-range dependencies.

I kept the embedding layer the same as before I then added a Conv1D layer with 64 filters and a kernel size of 5, using ReLU activation. I added a Conv1D layer with 64 filters and a kernel size of 5, using ReLU activation. After that, I included a MaxPooling1D layer for downsampling, followed by an LSTM layer with 32 units (same as the third model). I also kept the dropout layers to prevent overfitting. The final Dense layer with softmax activation stayed the same.



Looking at the curves, I think the model starts to show signs of overfitting around the 5th or 6th epoch. The validation accuracy reaches its highest point around epoch 6 and then seems to flatten out. While the training accuracy keeps climbing, the validation accuracy doesn't follow suit, which is a key indicator of overfitting. The same pattern can be seen in the loss graph below where the validation loss reaches a minimum around epoch 6, while the training loss continues to decrease. And after epoch 6, both the validation accuracy and loss curves appear to stagnate, showing no significant further improvement. Where the model has hit a performance ceiling on the validation set, even though it's still getting better on the training set.

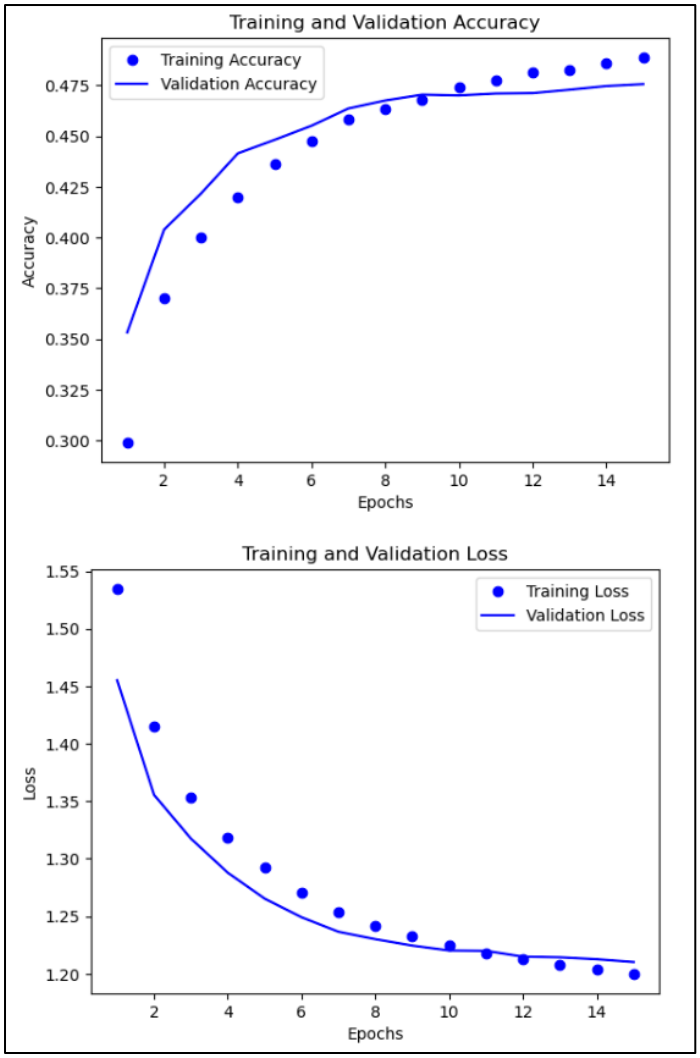
Model 5

```
hybrid_model_2 = Sequential()
hybrid_model_2.add(keras.Input(shape=(maxlen,)))
hybrid_model_2.add(Embedding(10000, 16))
hybrid_model_2.add(Conv1D(64, kernel_size=5, activation='relu'))
hybrid_model_2.add(MaxPooling1D(pool_size=2))  # Downsample the feature maps
hybrid_model_2.add(Conv1D(64, kernel_size=5, activation='relu'))  # Additional Conv1D layer
hybrid_model_2.add(MaxPooling1D(pool_size=2))  # Downsample again after the second Conv1D layer
hybrid_model_2.add(LSTM(32, dropout=0.3, recurrent_dropout=0.3))
hybrid_model_2.add(Dropout(0.3))
hybrid_model_2.add(Dense(5, activation='softmax'))
hybrid_model_2.summary()


# Train the Model
hybrid_model_2.compile(optimizer = RMSprop(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

history = hybrid_model_2.fit(X_train, y_train,
                    epochs=15,
                    batch_size=32,
                    validation_split=0.2)
```

For my fifth model, I decided to maintain the hybrid approach, but I refined it further. I added an additional Conv1D layer followed by another MaxPooling layer, with the intention of allowing the convolutional part of the model to learn more intricate local features before passing them to the LSTM. I also modified the embedding layer by reducing its dimensionality to 16, which I thought might help improve generalization. I kept the LSTM layer with 32 units, as I found that setup worked well in previous models. Additionally, I adjusted the dropout rate, lowering it after the LSTM layer to 0.3. Finally, I reduced the learning rate for the RMSprop optimizer to 0.0001, as I suspected the model was fluctuating too much during training, and a smaller learning rate could help it converge to a better solution. Other elements, such as the epoch size, validation split dimensions, dense output layer, and loss function, remained unchanged.



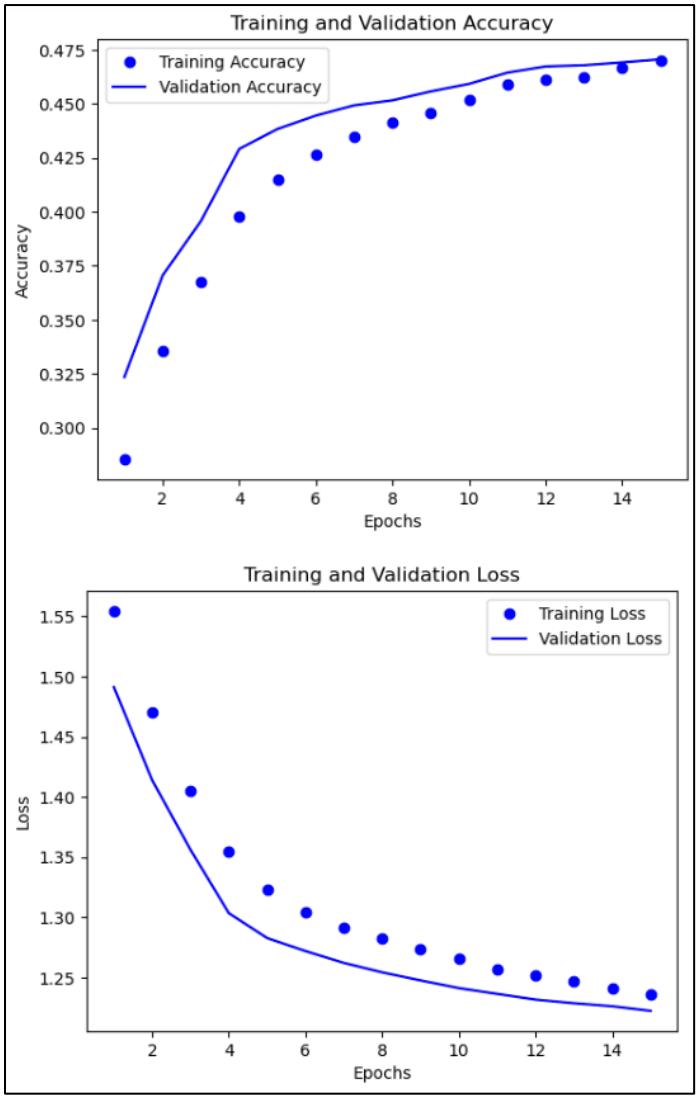
Looking at the performance curves, it seems the model is experiencing mild overfitting but significantly lesser than the previous model. The training accuracy is gradually increasing, but the validation accuracy lags and slowly becomes stagnant after 10 epochs. This suggests the model might be overfitting, but only slightly, as the gap between training and validation accuracy is relatively small. This pattern is also visible in the loss performance curves.

Model 6

```
model = Sequential()
model.add(keras.Input(shape=(maxlen,)))
model.add(Embedding(10000, 16))
model.add(Conv1D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(32, dropout=0.4, recurrent_dropout=0.3, return_sequences = True))
model.add(Dropout(0.3))
model.add(LSTM(32, dropout=0.4, recurrent_dropout=0.3))
model.add(Dropout(0.3))
model.add(Dense(5, activation='softmax'))
model.compile(optimizer=RMSprop(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(X_train, y_train,
                    epochs=15,
                    batch_size=16,
                    validation_split=0.2)
```

In this model, I made several modifications based on insights gained from the fifth model to further optimize feature extraction and sequence learning. While the fifth model incorporated two Conv1D layers with a kernel size of 5, I chose to reduce this to a single Conv1D layer with a smaller kernel size of 3. My reasoning behind this change was to focus on capturing more fine-grained local features while preventing excessive feature compression from repeated downsampling. Here, I also introduced two LSTM layers. Additionally, I slightly increased the dropout rate within the LSTM layers to 0.4 (from 0.3 in the fifth model) to further mitigate overfitting, particularly given the added depth.



Comparing the two models, the previous model shows a gradual increase in training accuracy with validation accuracy trailing slightly. While this model, which incorporates adjustments like expanded embedding dimensions, and larger LSTM units, converges more quickly and exhibits better generalization.

Overview of Models and Final Model

Model Number	Description	Test Accuracy (3sf)	Loss Score (3sf)
1	Baseline LSTM	48.1	1.19
2	Increased Epochs LSTM	48.0	1.20
3	Increased Capacity LSTM	48.6	1.20
4	Simple LSTM CNN Model	48.2	1.21
5	Stacked CNN, LSTM Model	47.6	1.21
6	Stacked LSTM, CNN Model	46.9	1.22

I chose Model 5 because, while it did not achieve the highest test accuracy on my original dataset, it performed consistently well across multiple external datasets that it wasn't trained on. This indicates strong generalization ability, meaning it can adapt well to new, unseen data rather than just memorizing patterns from the training set. Additionally, after analyzing overfitting levels, Model 5 demonstrated one of the lowest degrees of overfitting among all models. Hence why I think that Model 5 is the most reliable choice for real-world applications.

Final Model chosen for Tricia: Model 5