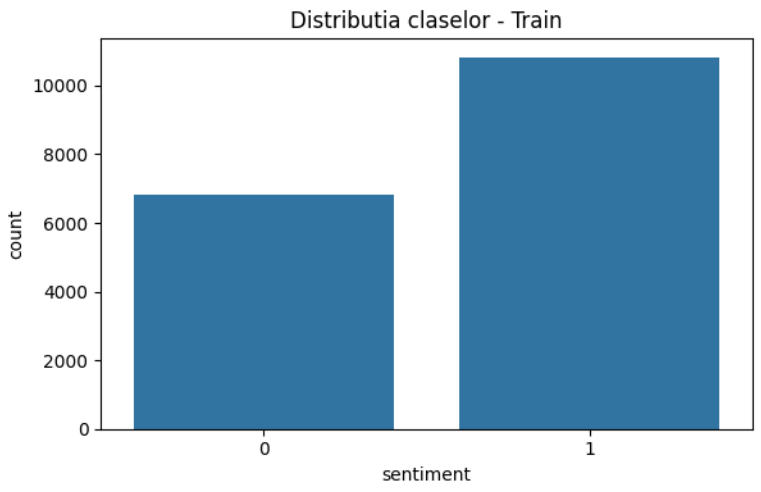
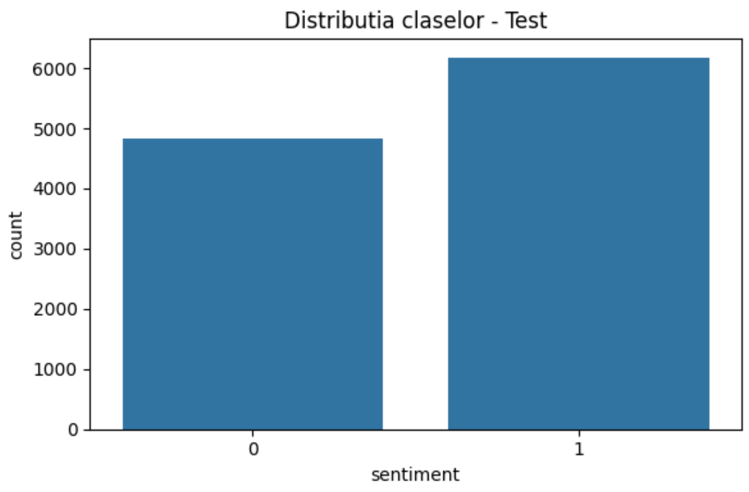
Text Sentiment Analysis

Vlad Schiopu

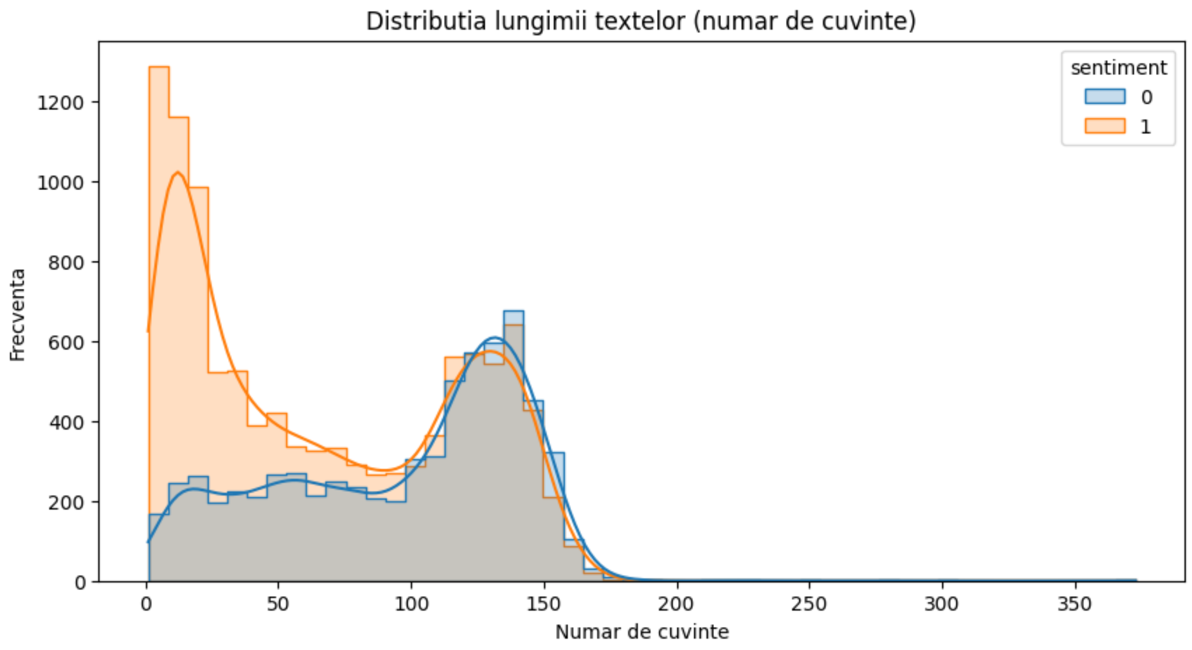
Sentiment Analysis on Text in Romanian

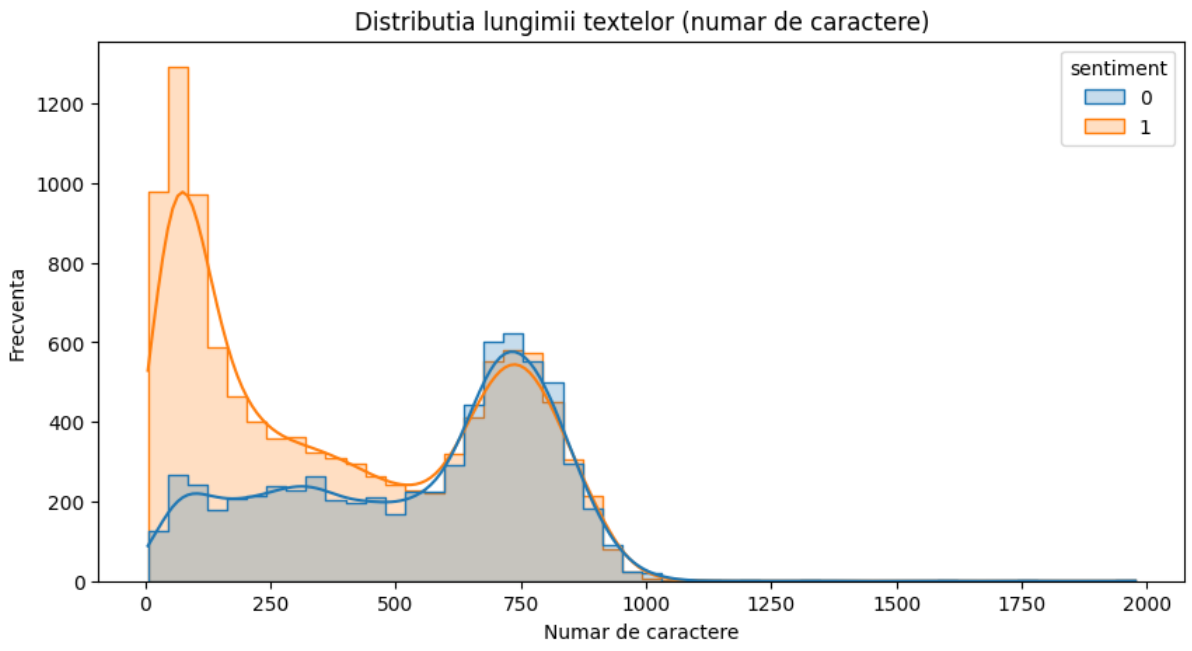
1 Data Exploration



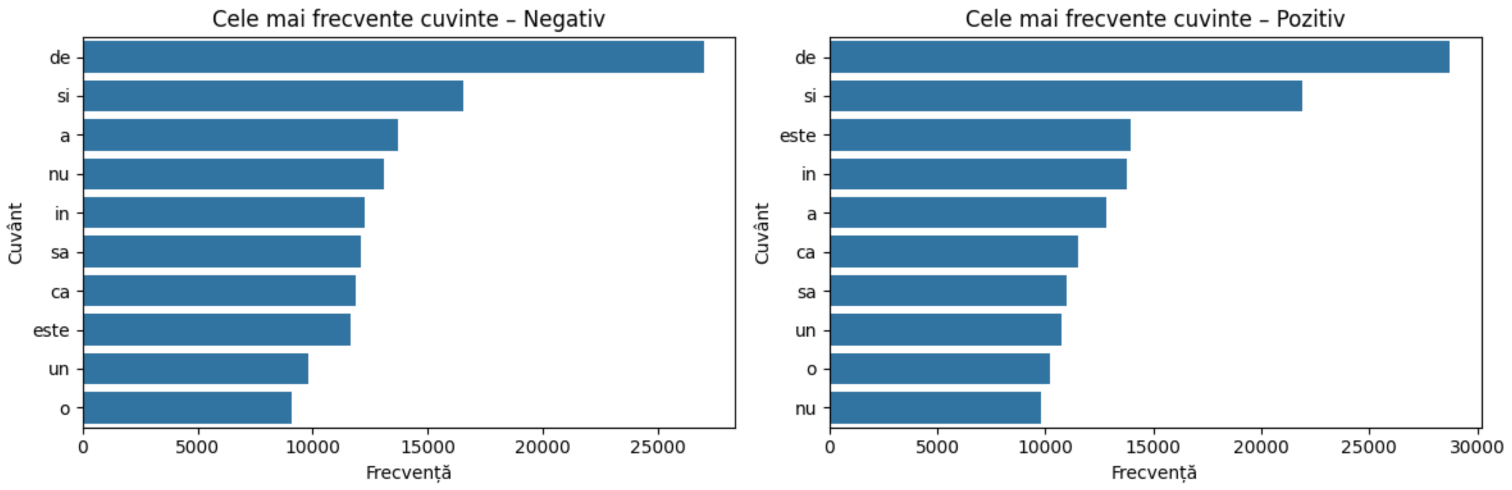


The distribution of classes is relatively balanced with a slight inclination towards positive reviews.

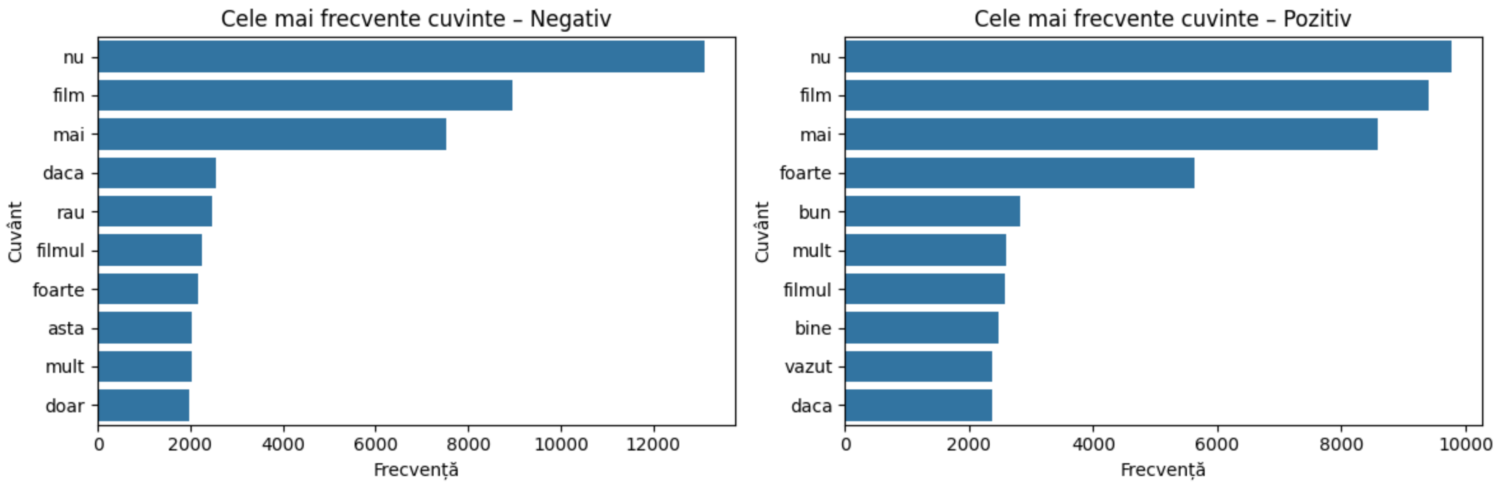




Statistics about the length of texts show that regardless of whether we consider the number of words or characters, very short reviews are generally positive, but longer reviews have an equal distribution between the 2 classes. The longest reviews reach about 170 words.



The most common words for both classes are prepositions, conjunctions, articles, and common verbs. For a more specific analysis I will have to remove these stopwords.



Following the elimination of common words we notice that the first 3 most common words are common, but we also notice some dominant words specific to each class: negative (bad, only), positive (very, good, much, good).

2 Tokenization and Embedding Layer

To clean the text I replaced the hyphens "-" with spaces " " to separate the words and I kept only the sequences formed by letters. We have also removed from the dataframe common words that do not have predictive value for sentiment.

For tokenization we used the ro_core_news_sm model and removed components that were not useful for tokenization. We built the vocabulary with a limit of 20000 most common words. And we added the <PAD> token to fill the shorter text sequences thus bringing each input to the same size, and the <UNK> token to mark words outside the vocabulary. Then I turned the words into indexes.

For embedding we used the FastText model cc.ro.300.vec which transforms each token into a vector of 300 features. And we created the embeddings matrix by turning each token into a model-based vector or a random vector if the token isn't in the vocabulary. We then created the embedding layer that will be used as the first step in the architecture of the training networks.

3 Using Recurrent Neural Network Models

-Simple RNN

When initializing the vectors X_train and X_test we used the MAX_LEN parameter to truncate the inputs to a certain number of words. Initially we had MAX_LEN = 100 which made the model predict only positive or negative on almost any example. I reduced it to 50 which gave much better results making the model even learn.

I ran several hyperparameter configurations and architectures to find the optimal configuration

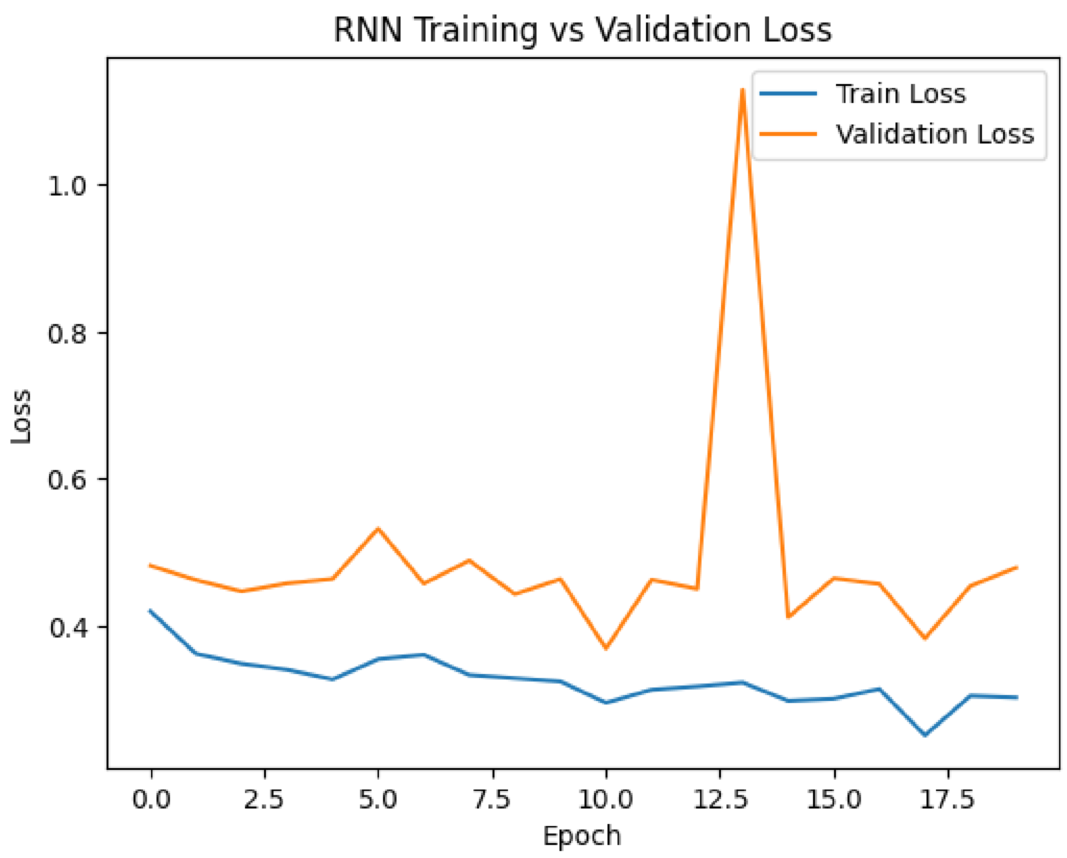
-SimpleRNN(128,False,tanh) with BatchNorm and Dropout(0.3)

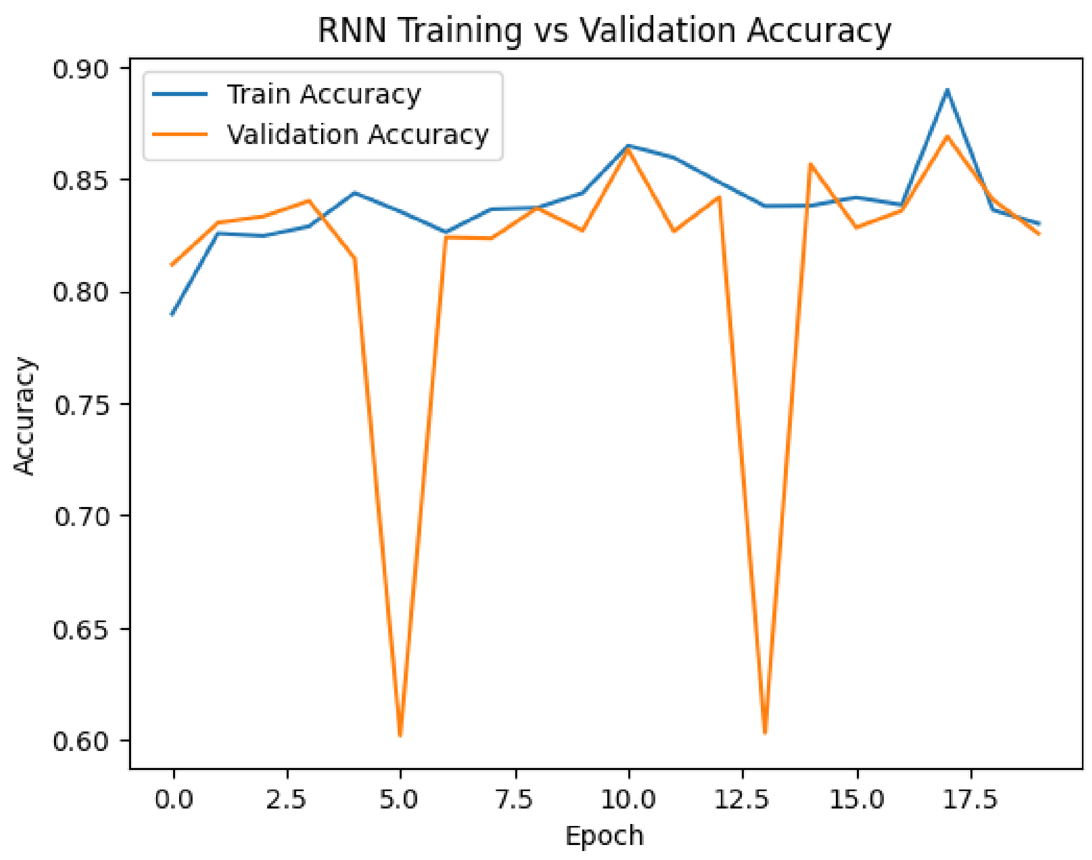
batchsize = 64

lr = 1e-3

-Epoch 20: accuracy: 0.8559 - loss: 0.2898 - val_accuracy: 0.8257 - val_loss: 0.4791 - learning_rate: 6.2500e-05

-Test: accuracy: 0.6856 - loss: 1.1668 Test accuracy: 0.7007





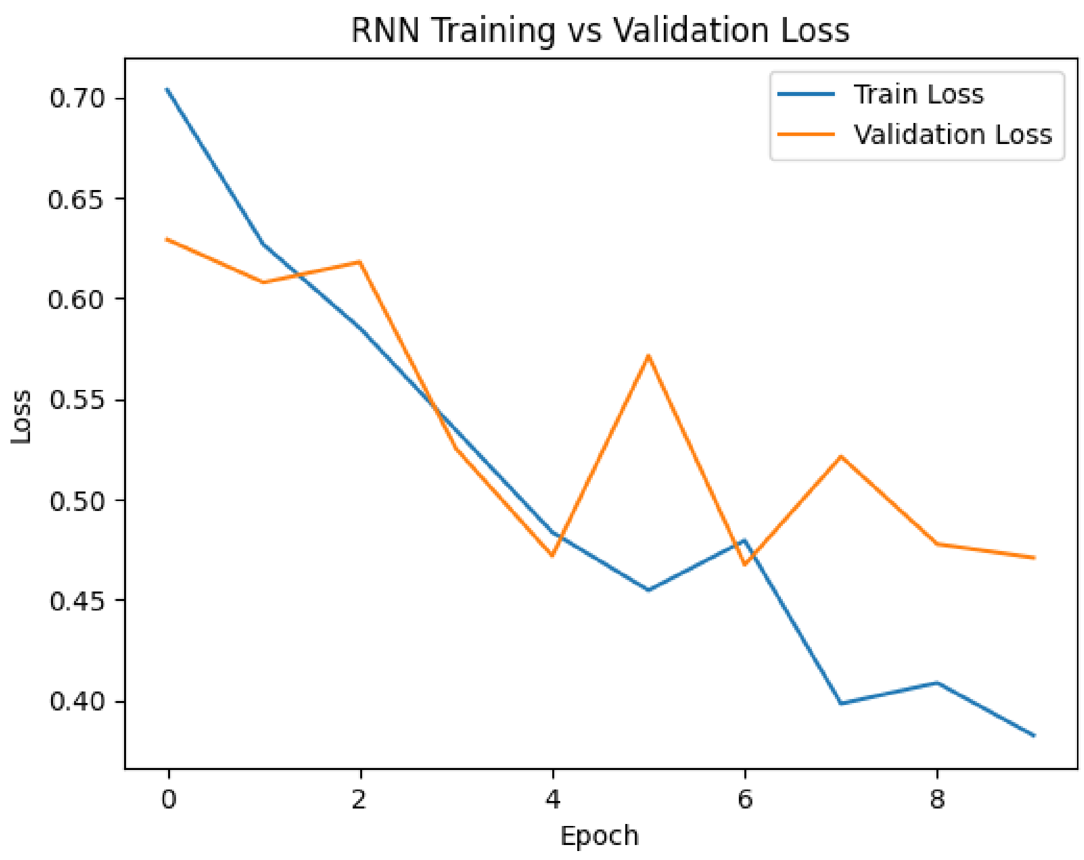
-SimpleRNN(128,False,tanh) with BatchNorm and Dropout(0.3)

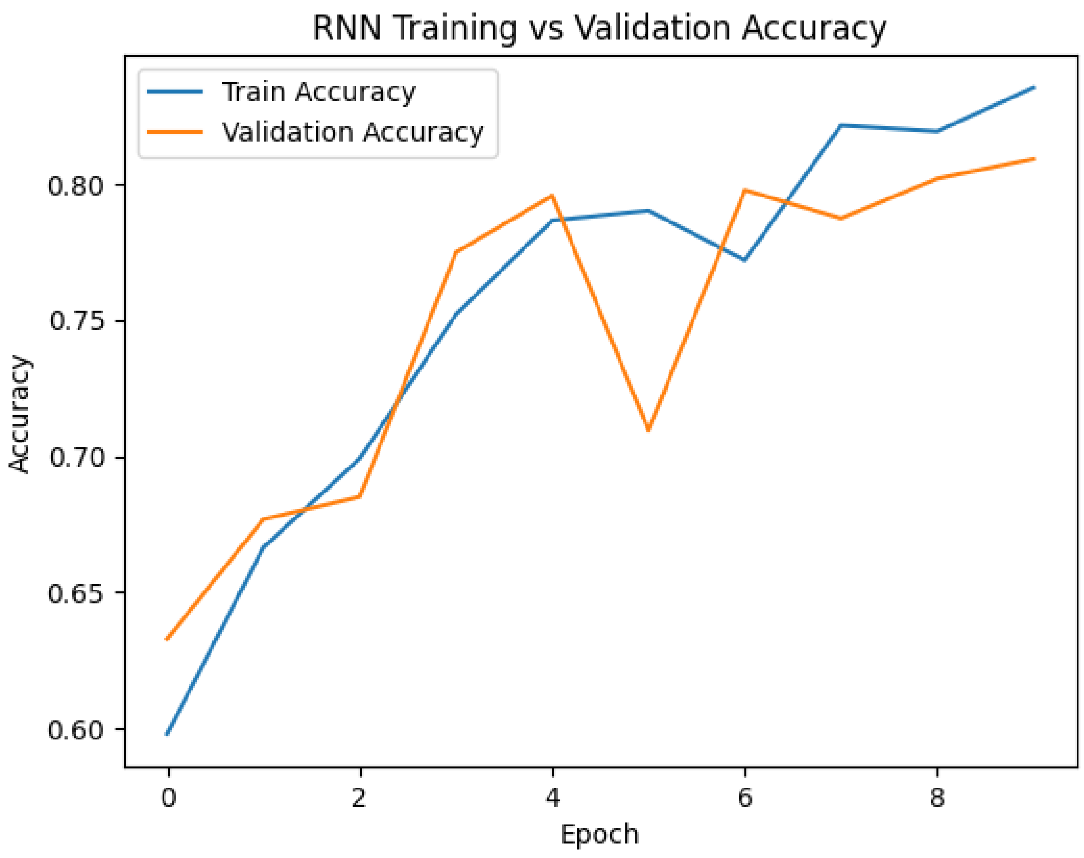
batchsize = 8

lr = 1e-3

-Epoch 20: accuracy: 0.8299 - loss: 0.3617 - val_accuracy: 0.6403 - val_loss: 1.0868 - learning_rate: 5000e-04

-Test: accuracy: 0.7453 - loss: 0.7460 Test accuracy: 0.6945


RNN Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
2
4
6
8


RNN Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8

Bottom Line: Lower Batchsize Delivers Similar Accuracy Performance But Longer Training Time

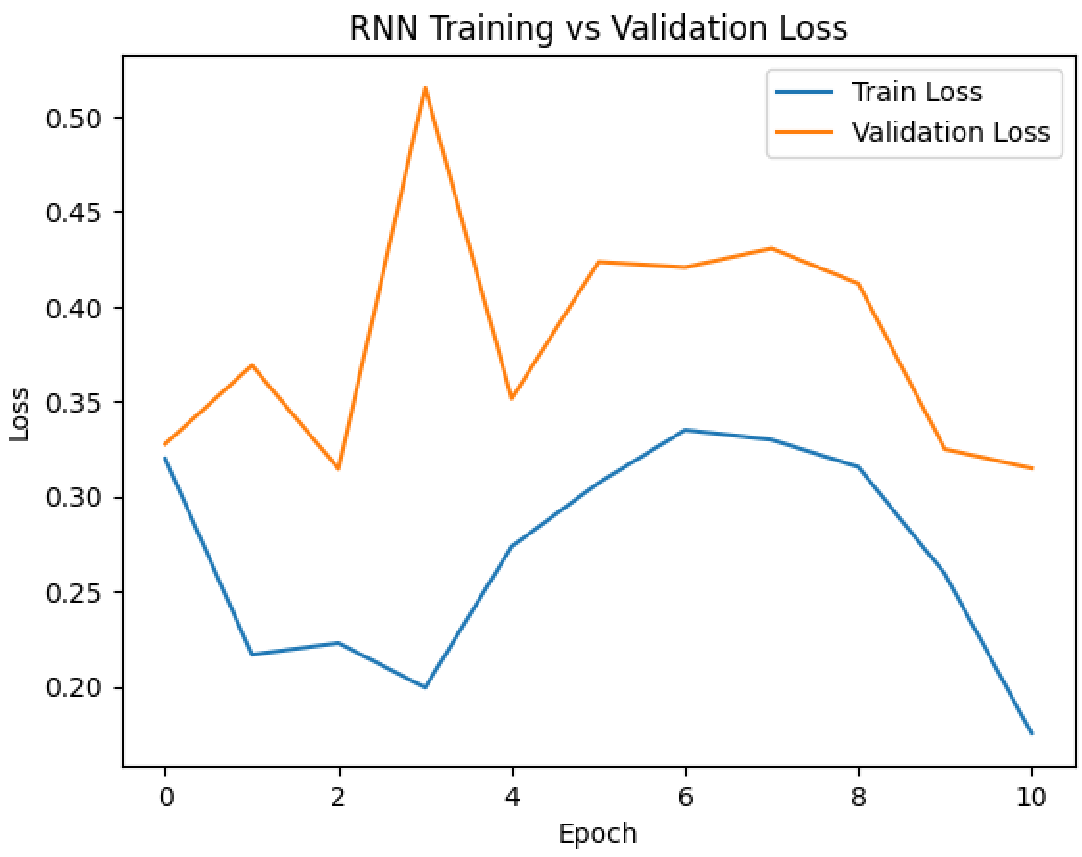
-SimpleRNN(128,False,tanh) with BatchNorm and Dropout

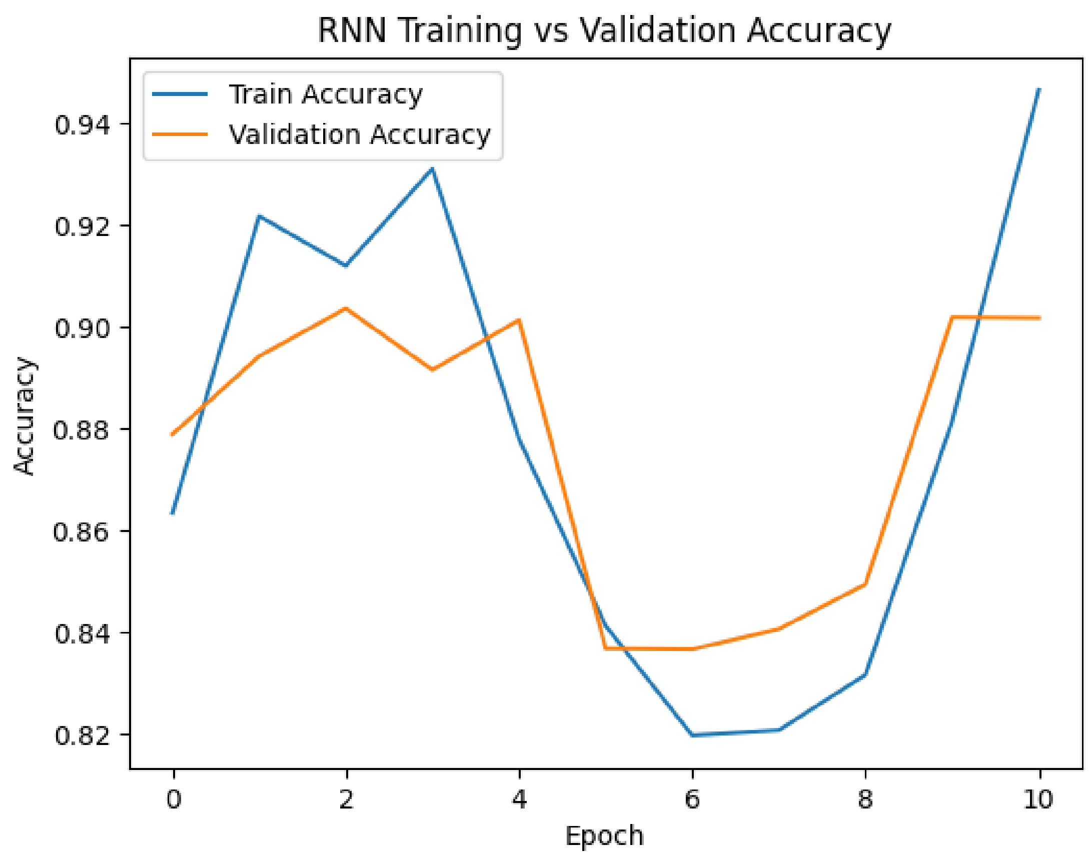
batchsize = 64

lr = 1e-4

-Epoch 11: accuracy: 0.9466 - loss: 0.1770 - val_accuracy: 0.9017 - val_loss: 0.3149 - learning_rate: 5000e-05

-Test: accuracy: 0.7586 - loss: 0.9557 Test accuracy: 0.7564





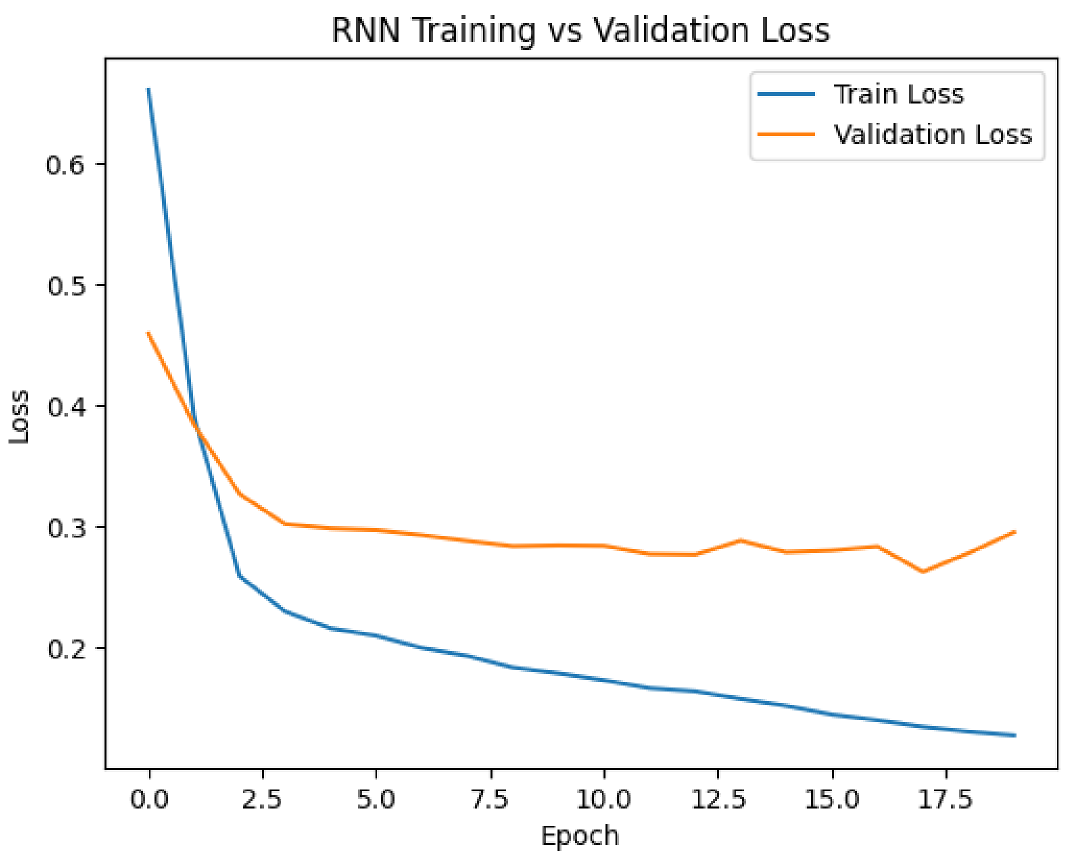
-SimpleRNN(128,False,tanh) with BatchNorm and Dropout

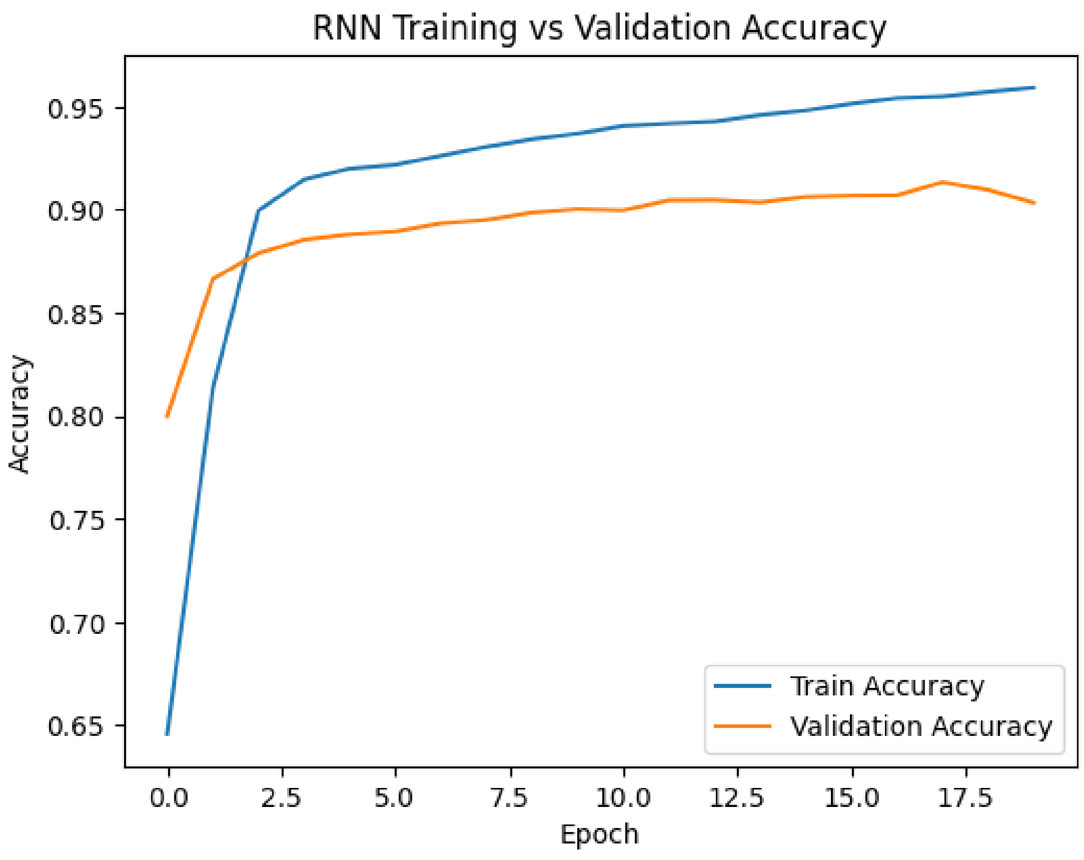
batchsize = 64

lr = 1e-5

-Epoch 20: accuracy: 0.9604 - loss: 0.1254 - val_accuracy: 0.9034 - val_loss: 0.2952 - learning_rate: 1.0000e-05

-Test: accuracy: 0.7286 - loss: 1.0602 Test accuracy: 0.7492


RNN Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


RNN Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Conclusion: Lower initial learning rate provides better results and leads to early stopping (I keep 5e-5 for stability and performance)

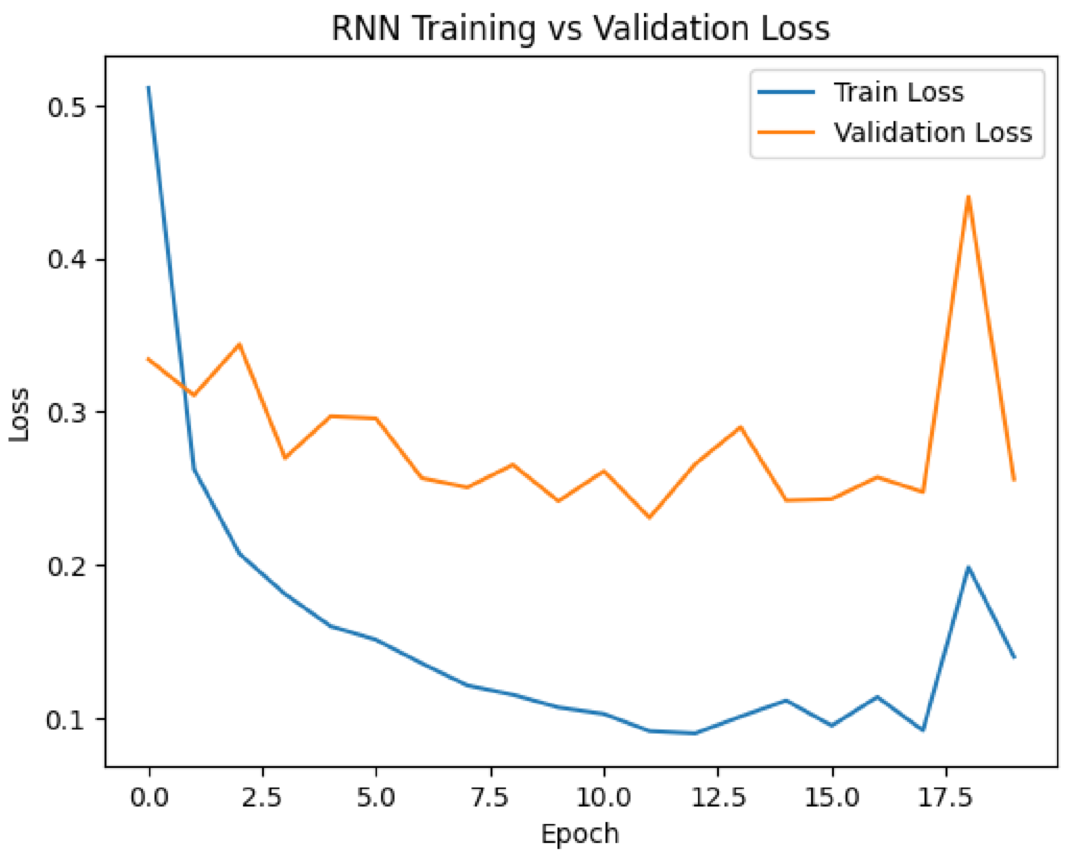
-SimpleRNN(**32**,False,tanh) with BatchNorm and Dropout

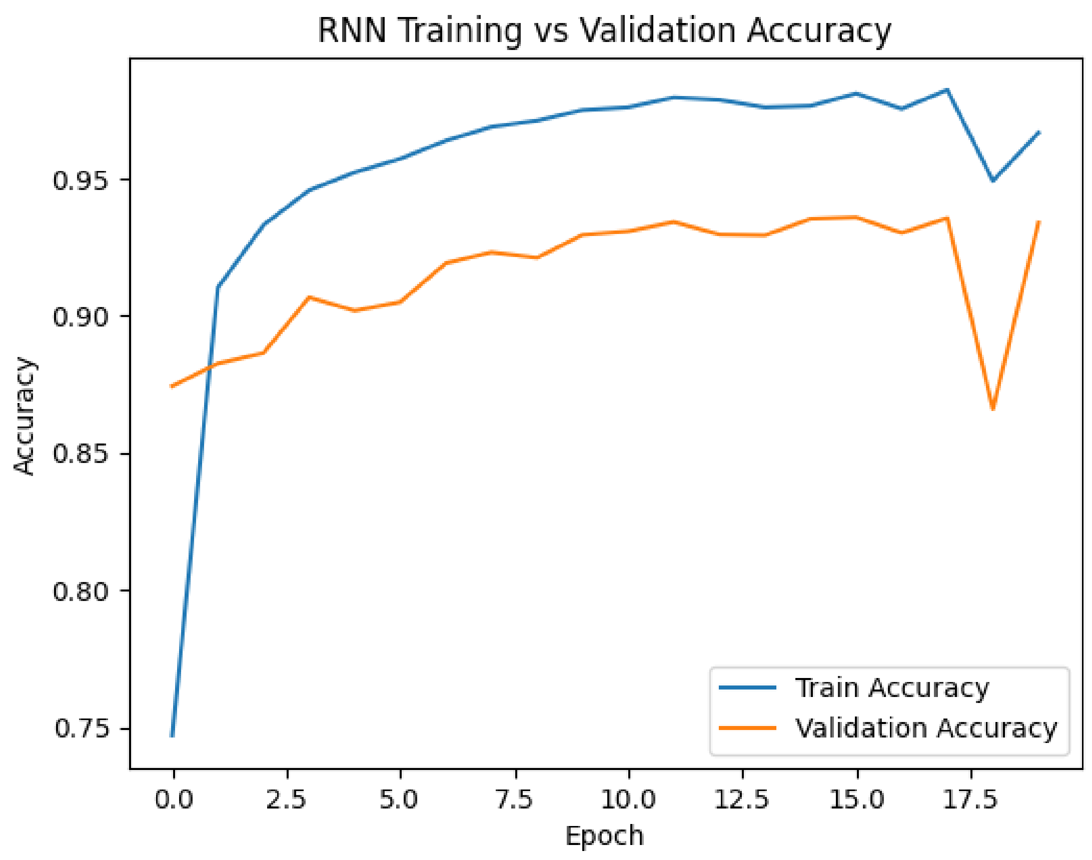
batchsize = 64

lr = 5e-5

-Epoch 20: accuracy: 0.9440 - loss: 0.2026 - val_accuracy: 0.9337 - val_loss: 0.2559 - learning_rate: 5000e-05

-Test: accuracy: 0.7576 - loss: 0.9482 Test accuracy: 0.7507





Conclusion: Fewer neurons make the model a little more unstable

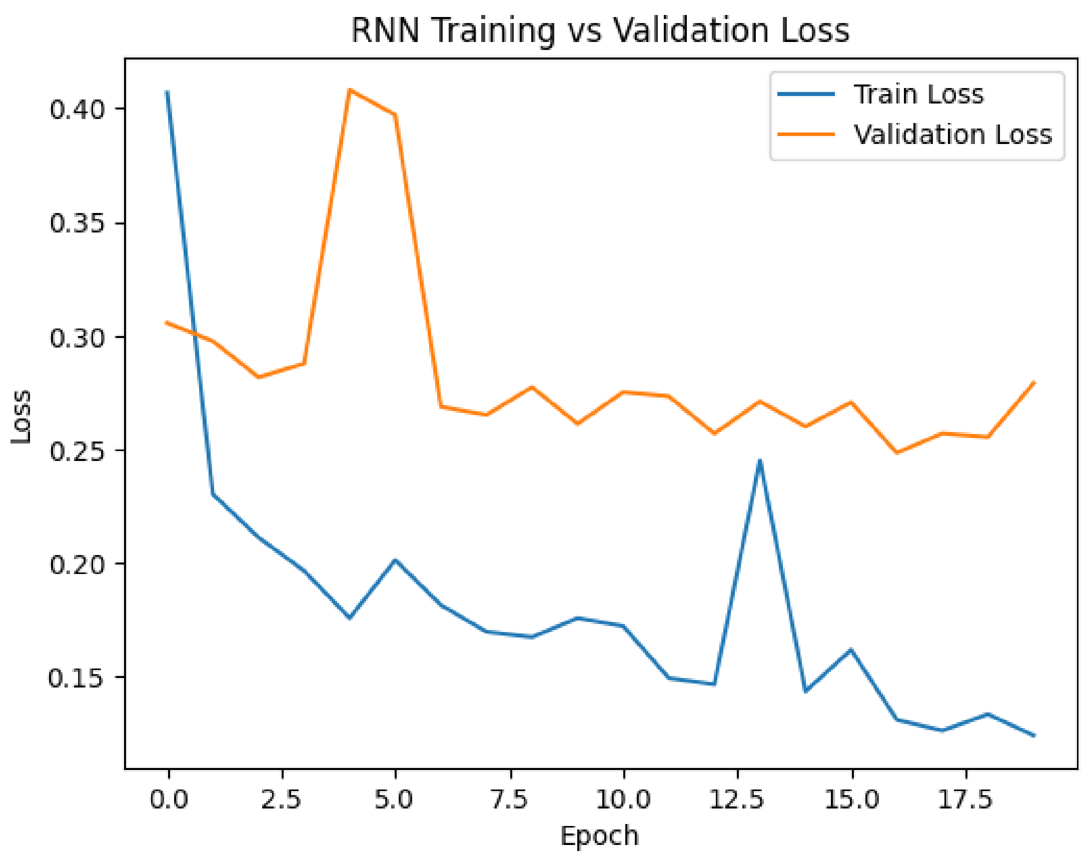
-SimpleRNN(128,False,tanh) **without** BatchNorm and Dropout

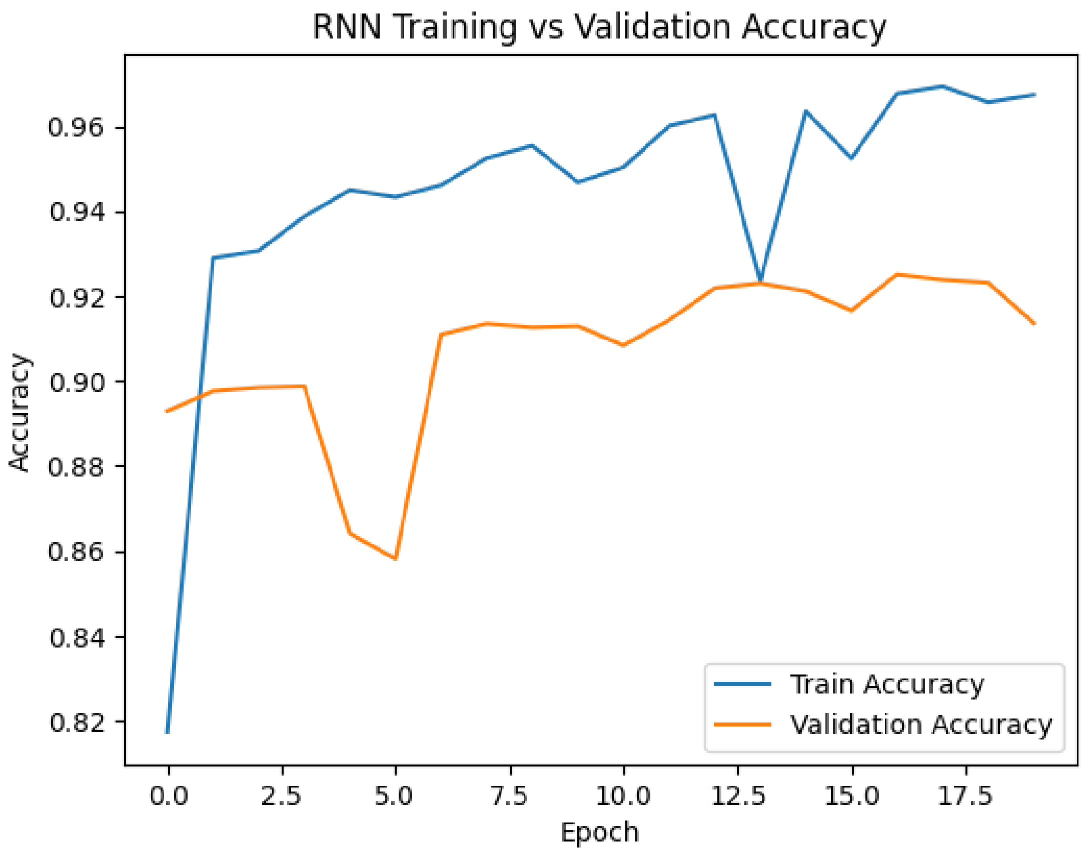
batchsize = 64

lr = 5e-5

-Epoch 20: accuracy: 0.9682 - loss: 0.1263 - val_accuracy: 0.9136 - val_loss: 0.2791 - learning_rate: 5000e-05

-Test: accuracy: 0.7588 - loss: 0.8663 Test accuracy: 0.7601


RNN Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.40
0.35
0.30
0.25
0.20
0.15
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


RNN Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Conclusion: without Batchnorm we notice a more unstable progress but slightly better performance on the test

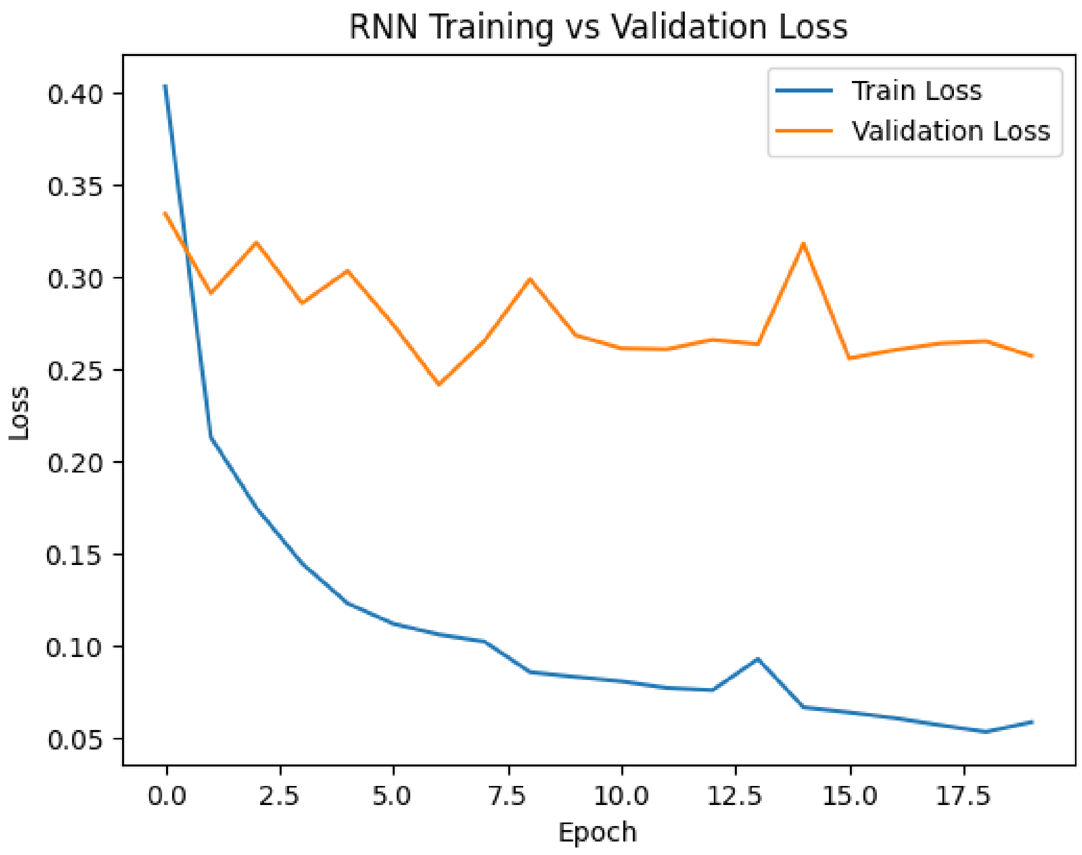
-SimpleRNN(128) + SimpleRNN(64) with BatchNorm and Dropout(0.3)

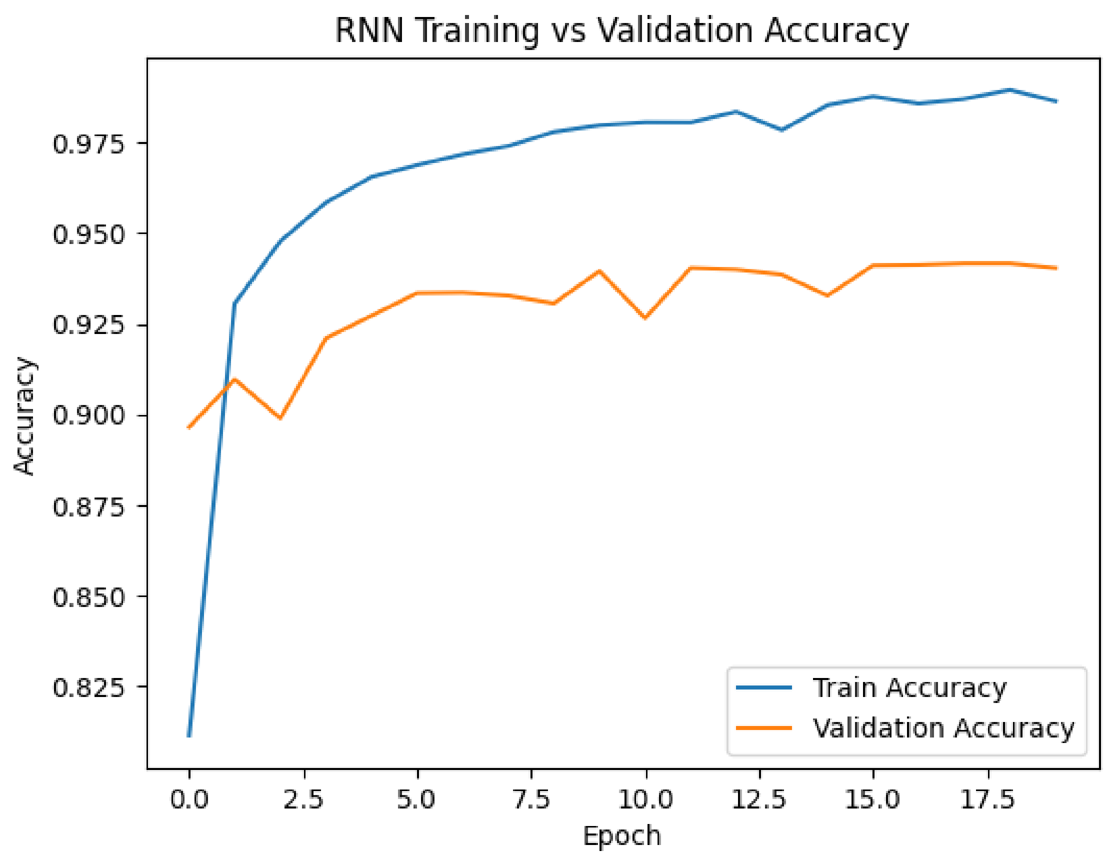
batchsize = 64

lr = 5e-5

-Epoch 20: accuracy: 0.9876 - loss: 0.0551 - val_accuracy: 0.9404 - val_loss: 0.2573 - learning_rate: 5000e-05

-Test: accuracy: 0.7571 - loss: 1.4009 Test accuracy: 0.7549





Conclusion: I don't notice any significant improvement with 2 layers

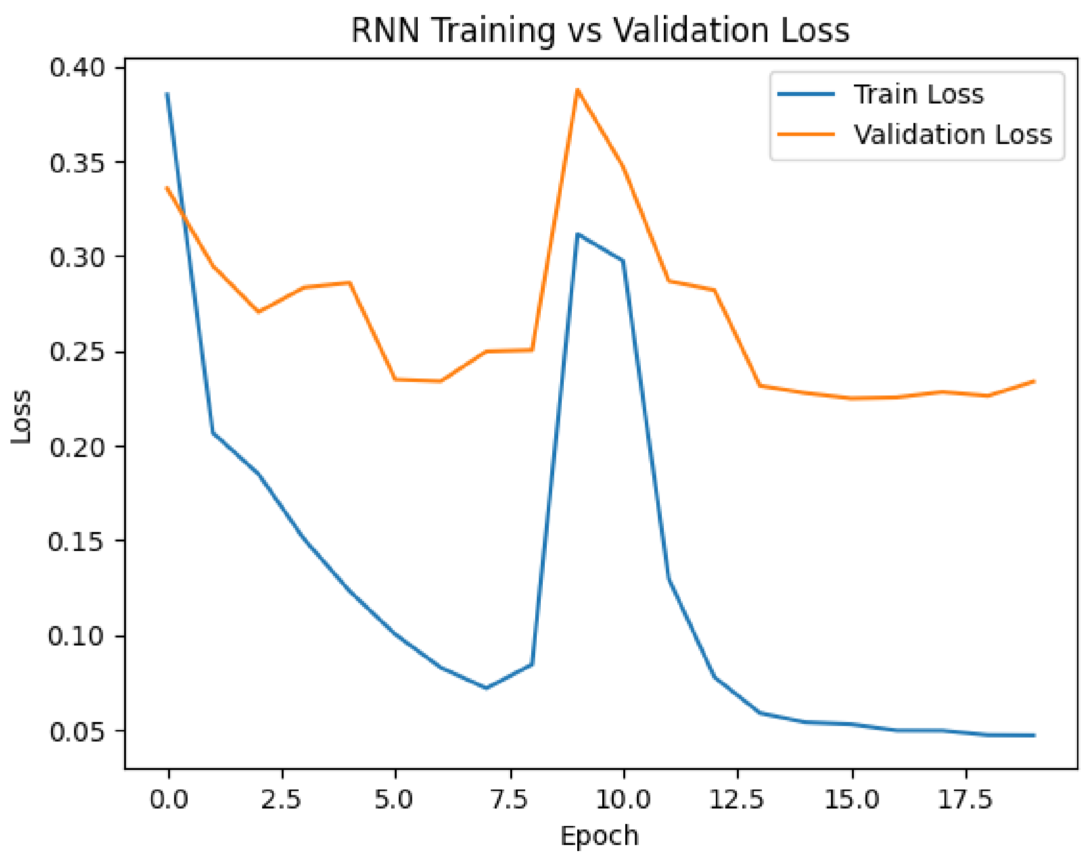
-SimpleRNN(128) + SimpleRNN(64) **without** BatchNorm and Dropout(0.3)

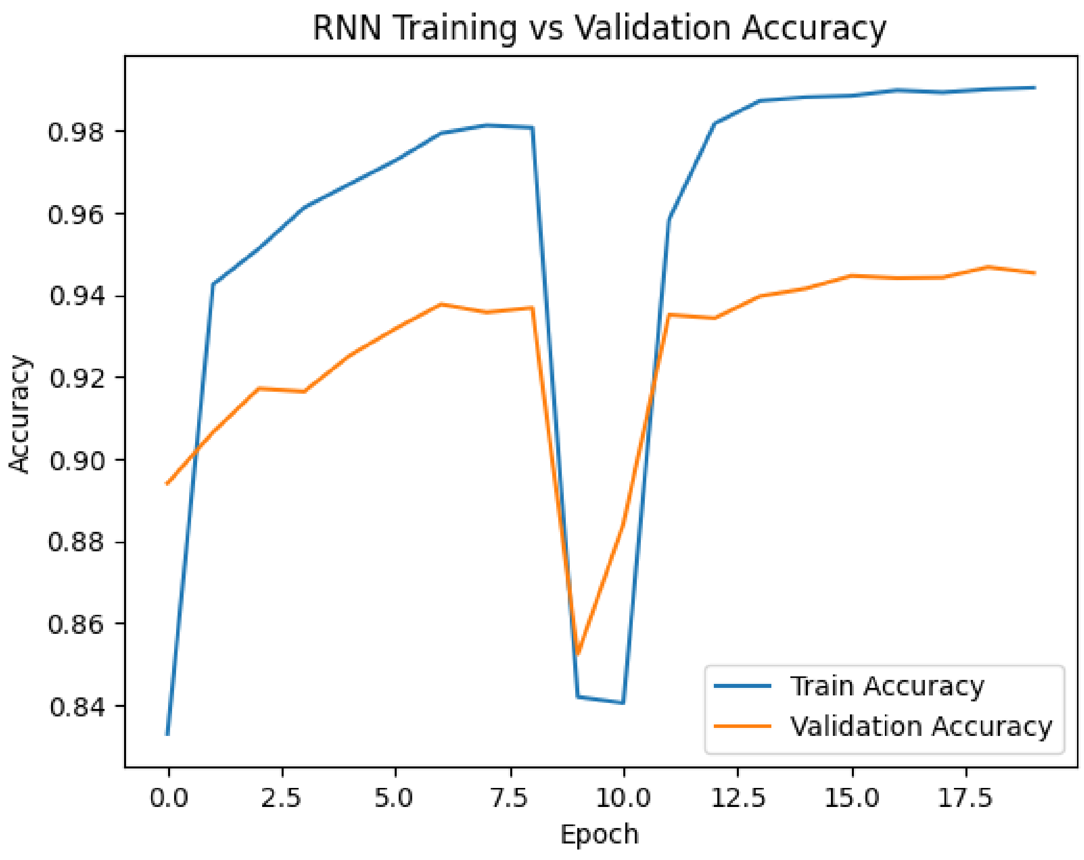
batchsize = 64

lr = 5e-5

-Epoch 20: accuracy: 0.9905 - loss: 0.0477 - val_accuracy: 0.9453 - val_loss: 0.2336 - learning_rate: 1.2500e-05

-Test: accuracy: 0.7560 - loss: 1.1372 Test accuracy: 0.7542


RNN Training vs Validation Loss
Train Loss
Validation Loss
Loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


RNN Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

Conclusion: A little more overfitting without batchnorm

-SimpleRNN(128) + SimpleRNN(64) with BatchNorm and Dropout(0.3)

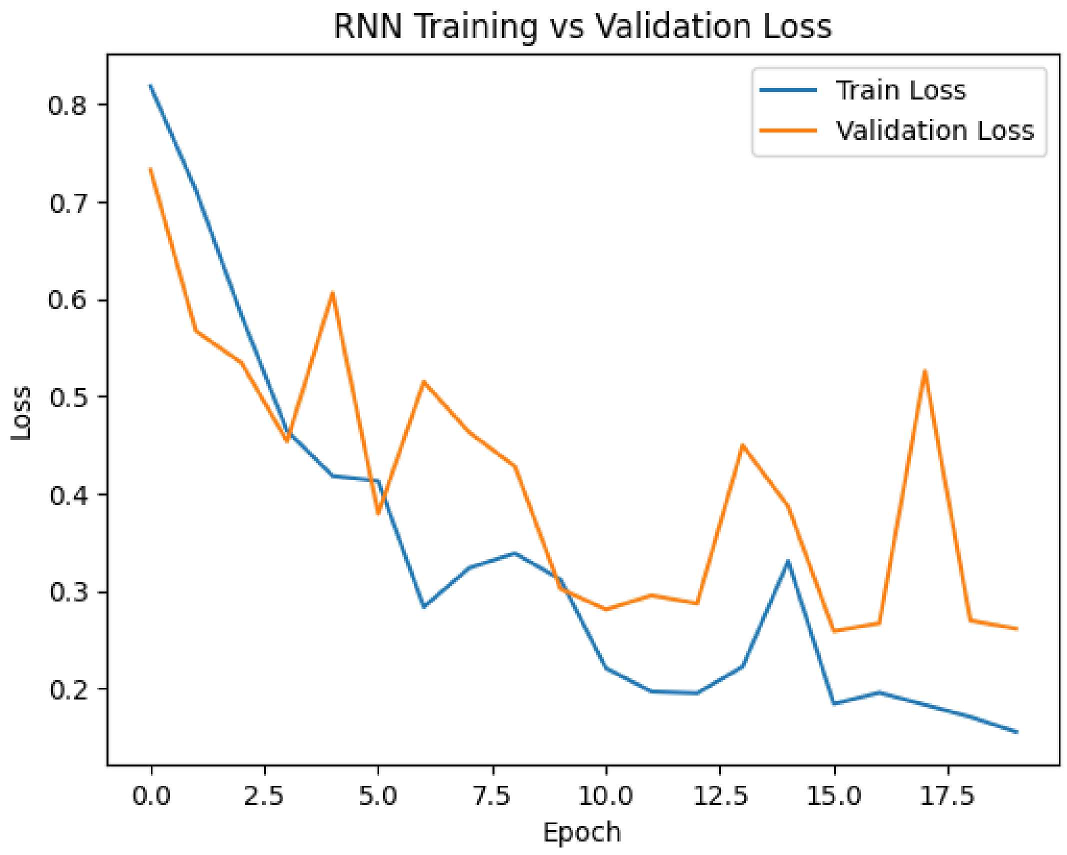
+delete and swap augumentations

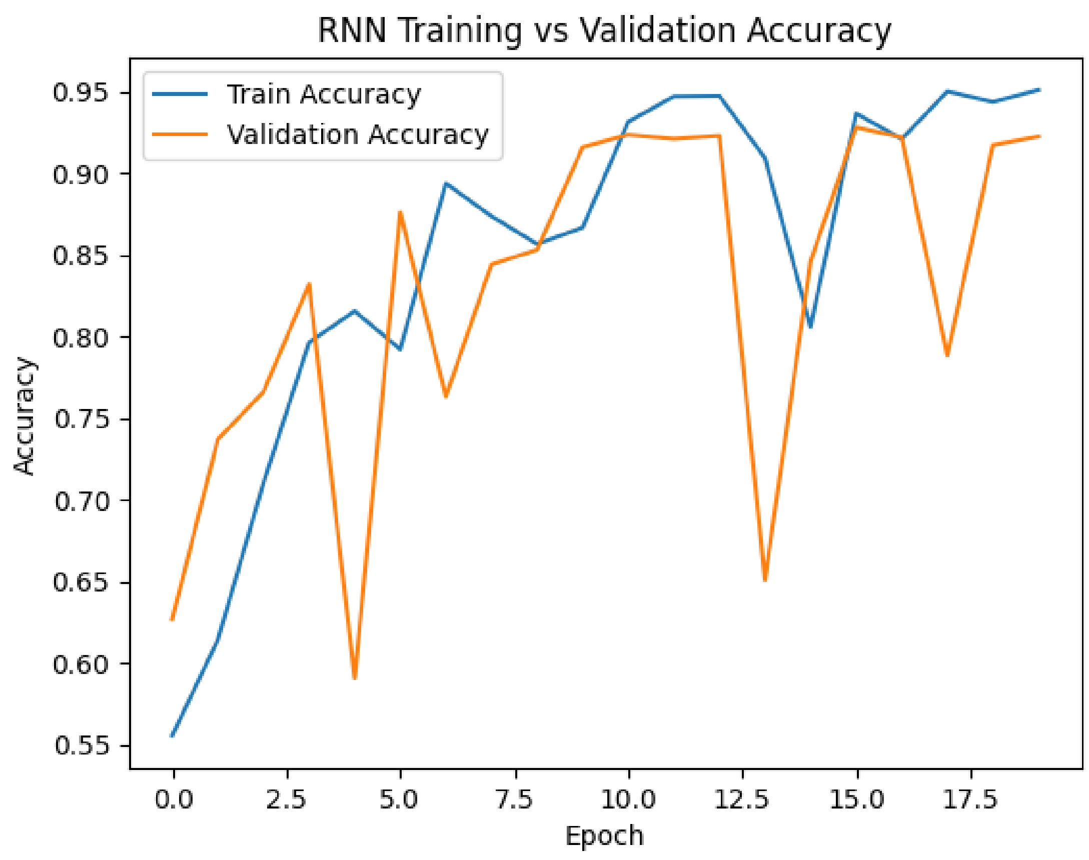
batchsize = 64

lr = 5e-5

-Epoch 10: accuracy: 0.9493 - loss: 0.1575 - val_accuracy: 0.9222 - val_loss: 0.2613 - learning_rate: 6.2500e-06

-Test: accuracy: 0.7561 - loss: 1.1298 Test accuracy: 0.7726





Conclusion: Although augmentations add large fluctuations in accuracy, they ultimately provide better accuracy on the test.

-SimpleRNN(128) + SimpleRNN(64) with BatchNorm and Dropout(0.3)

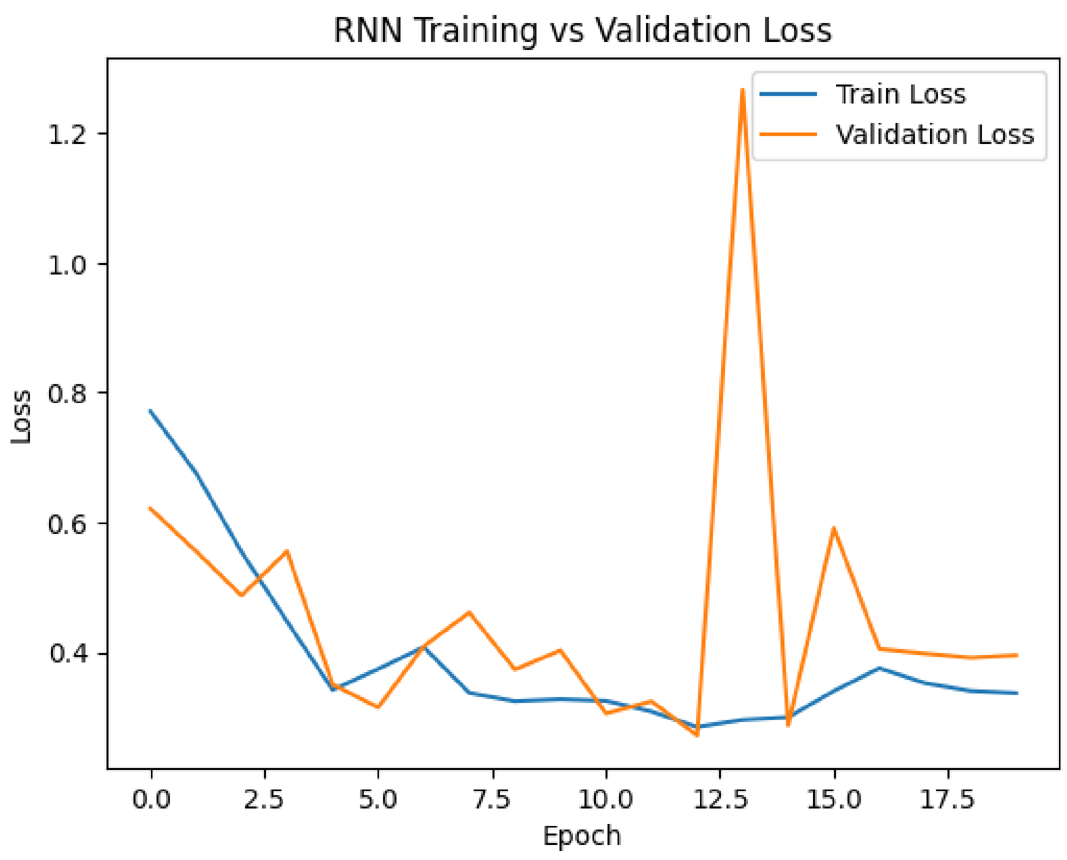
+ BERT context replace

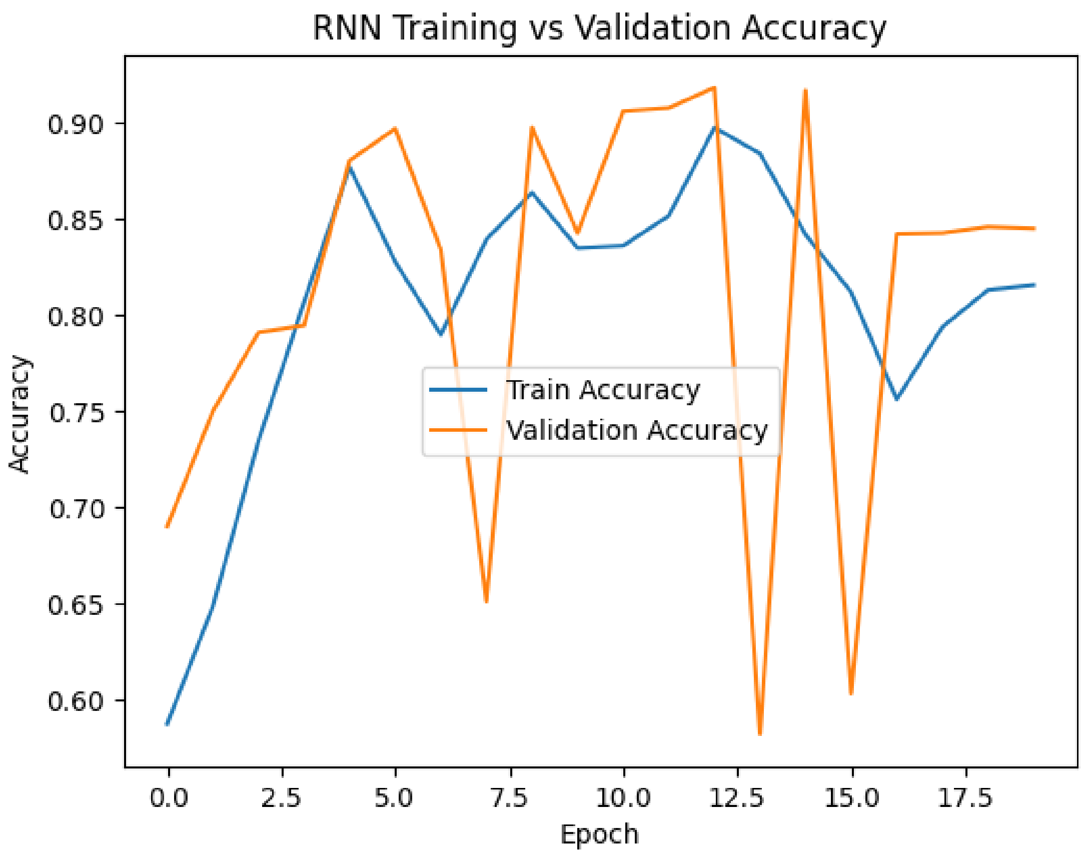
batchsize = 64

lr = 5e-5

-Epoch 10: accuracy: 0.9493 - loss: 0.1575 - val_accuracy: 0.9222 - val_loss: 0.2613 - learning_rate: 6.2500e-06

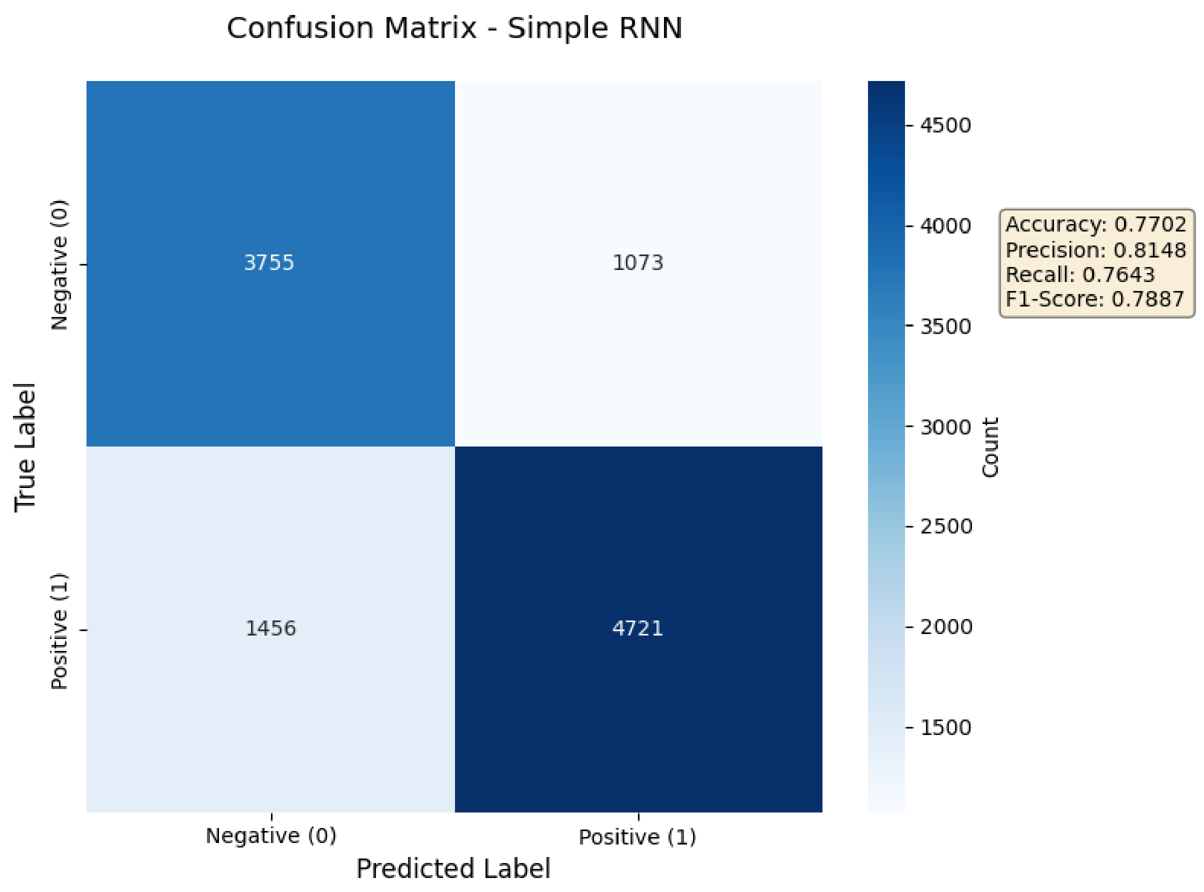
-Test: accuracy: 0.7561 - loss: 0.9117 Test accuracy: 0.7791





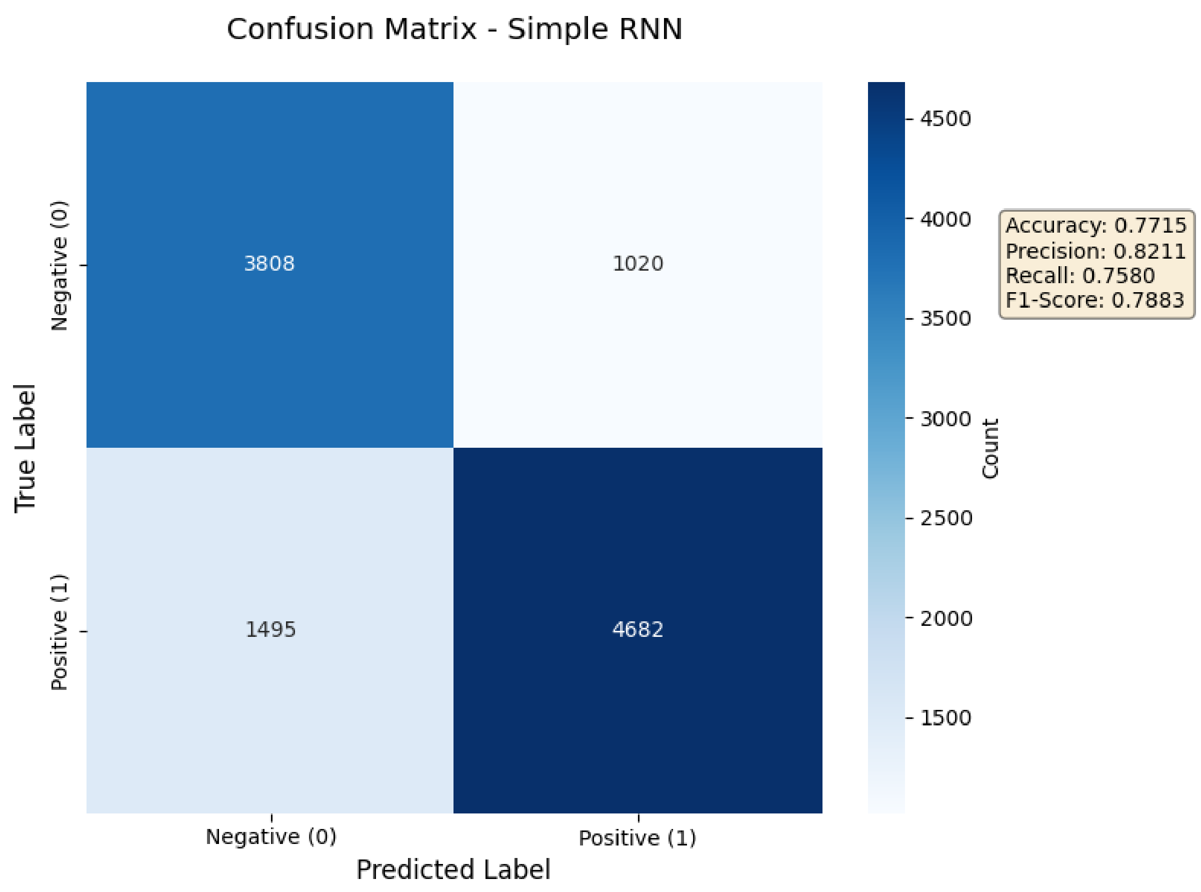
Conclusion: Changing the context makes the model generalize better

SimpleRNN:



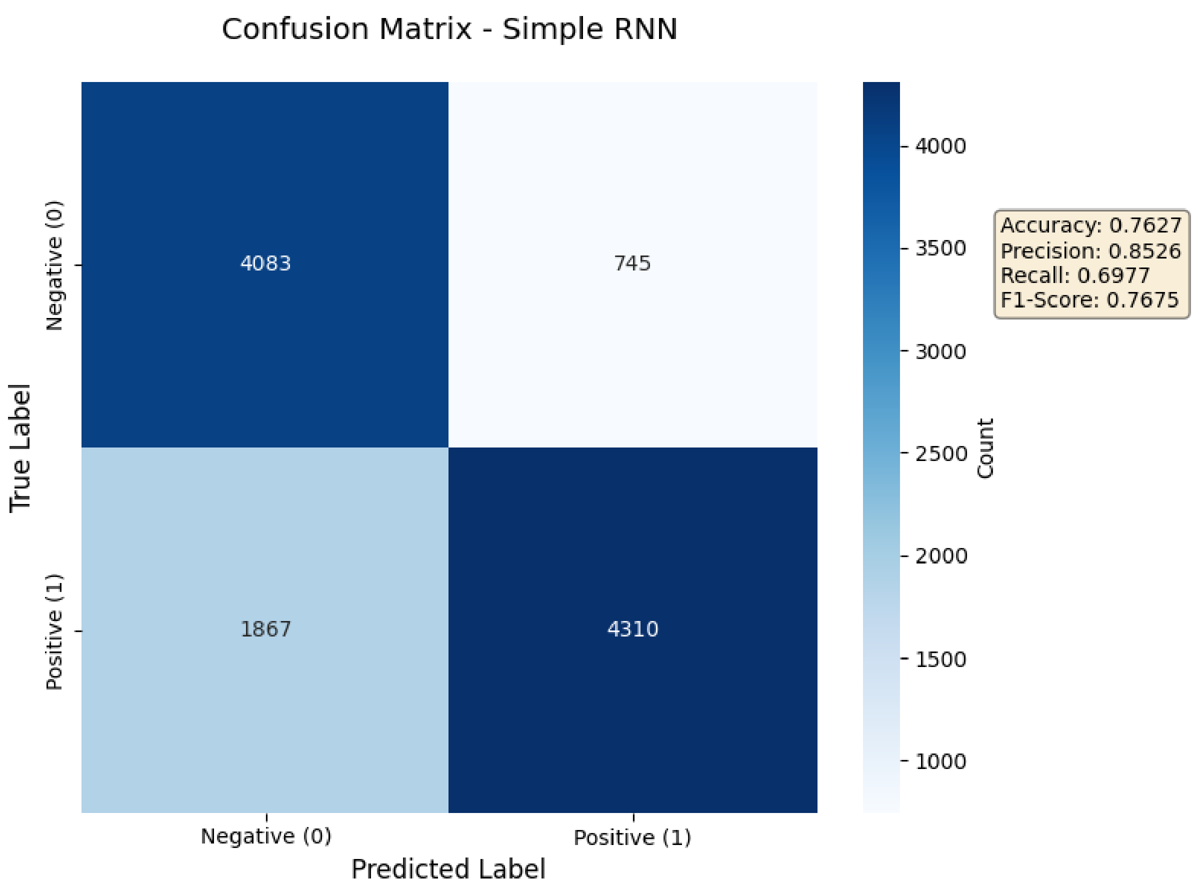
SimpleRNN:

+Swap and Delete Augs:

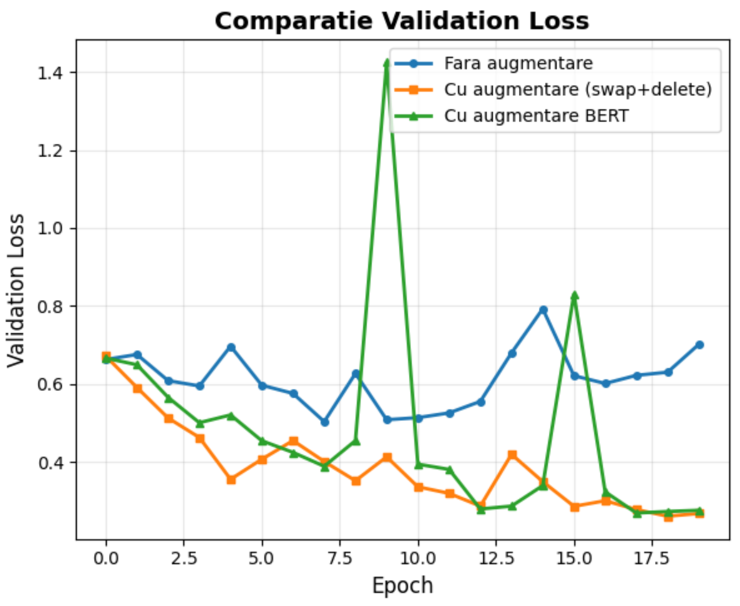


SimpleRNN:

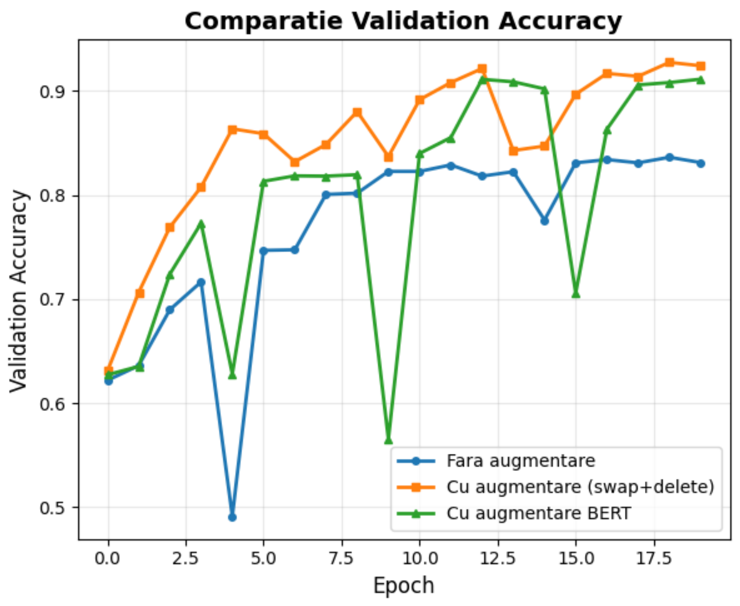
+BERT context swap:



Confusion Matrix - Simple RNN
True Label
Negative (0)
Positive (1)
4083
745
1867
4310
Negative (0)
Positive (1)
Predicted Label
Count
4000
3500
3000
2500
2000
1500
1000
Accuracy: 0.7627
Precision: 0.8526
Recall: 0.6977
F1-Score: 0.7675



Comparatie Validation Loss
Validation Loss
Epoch
Fara augmentare
Cu augmentare (swap+delete)
Cu augmentare BERT



Comparatie Validation Accuracy
Validation Accuracy
Epoch
Fara augmentare
Cu augmentare (swap+delete)
Cu augmentare BERT

-LSTM

The LSTM architecture being more complex led to a significantly longer training duration. To explore configurations, we trained the model on only part of the dataset (usually 0.3) and adapted MAX_LEN to the size of the data.

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 50

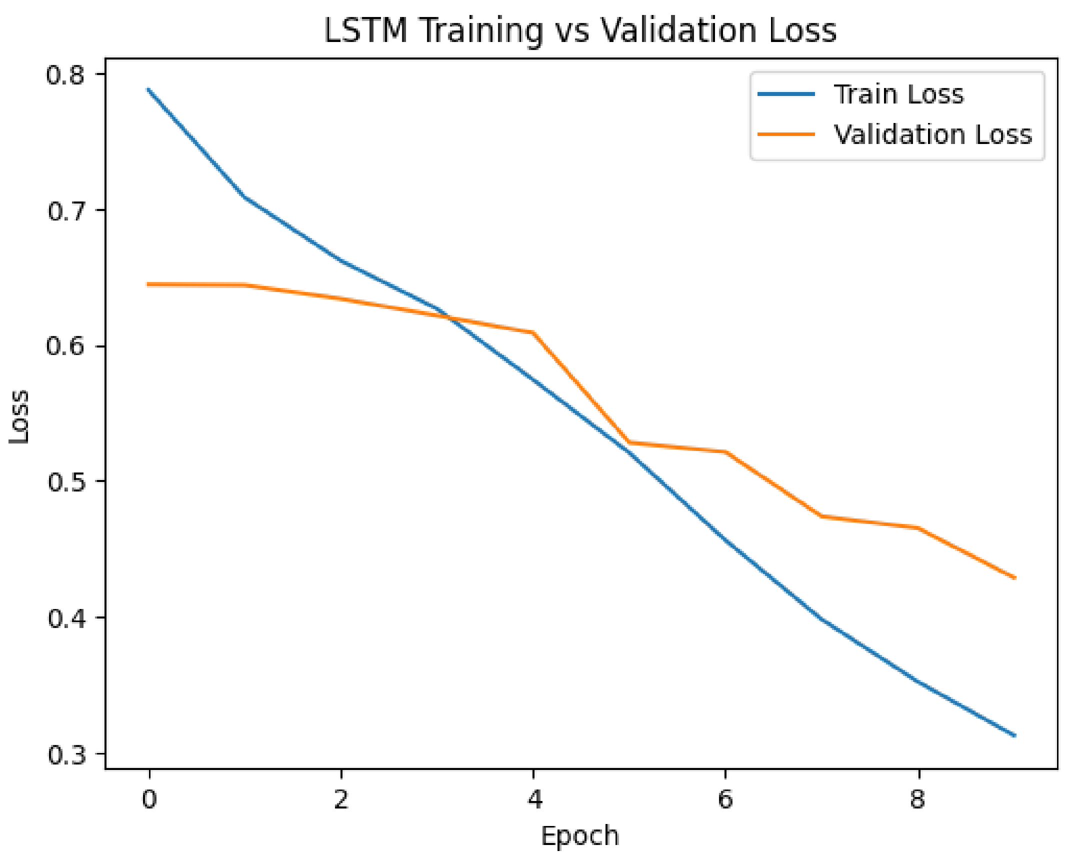
batchsize = 64

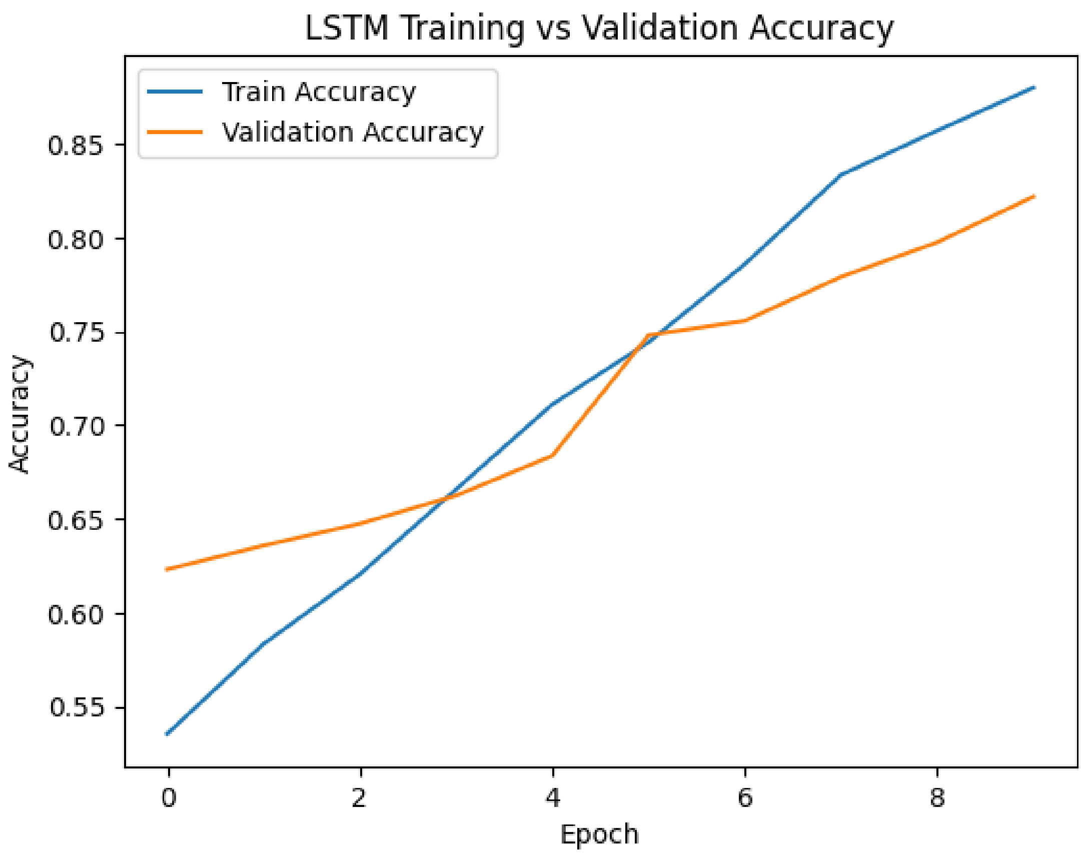
lr = 1e-4

sample_size = 0.5

-Epoch 10: accuracy: 0.8787 - loss: 0.3131 - val_accuracy: 0.8219 - val_loss: 0.4288 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7551 - loss: 0.5931 Test accuracy: 0.7811


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
2
4
6
8

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

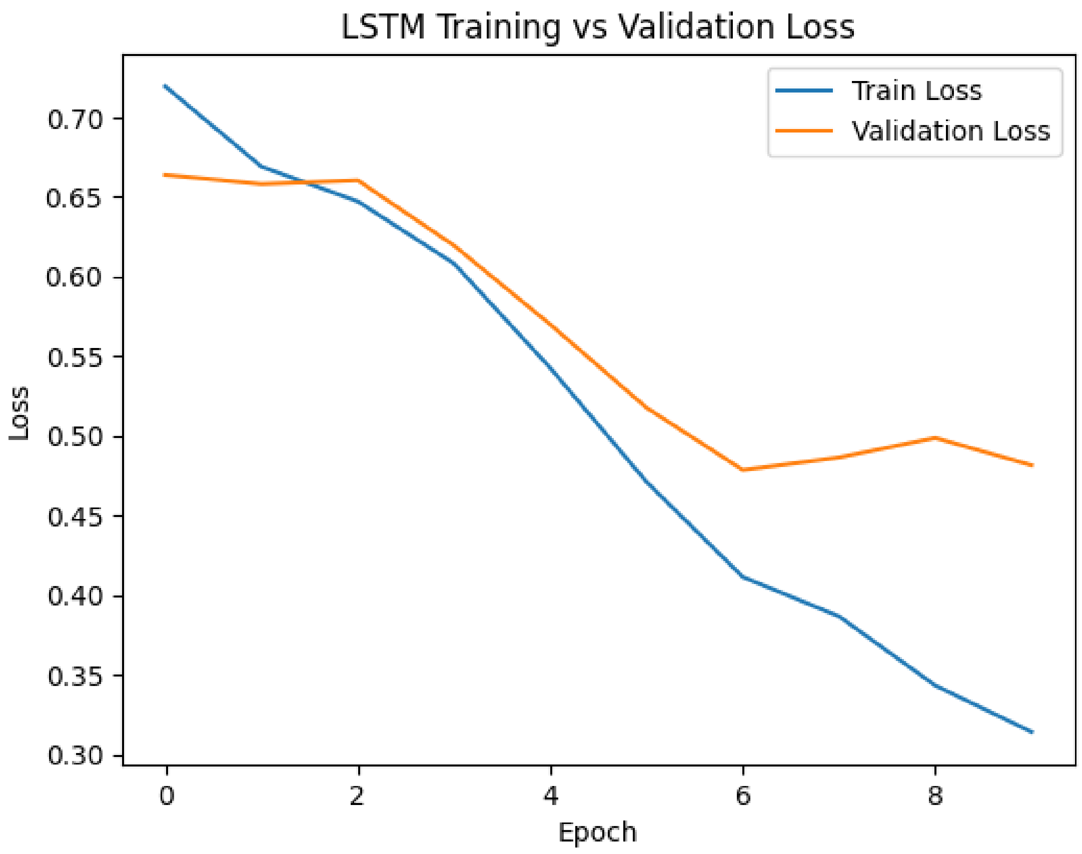
batchsize = 64

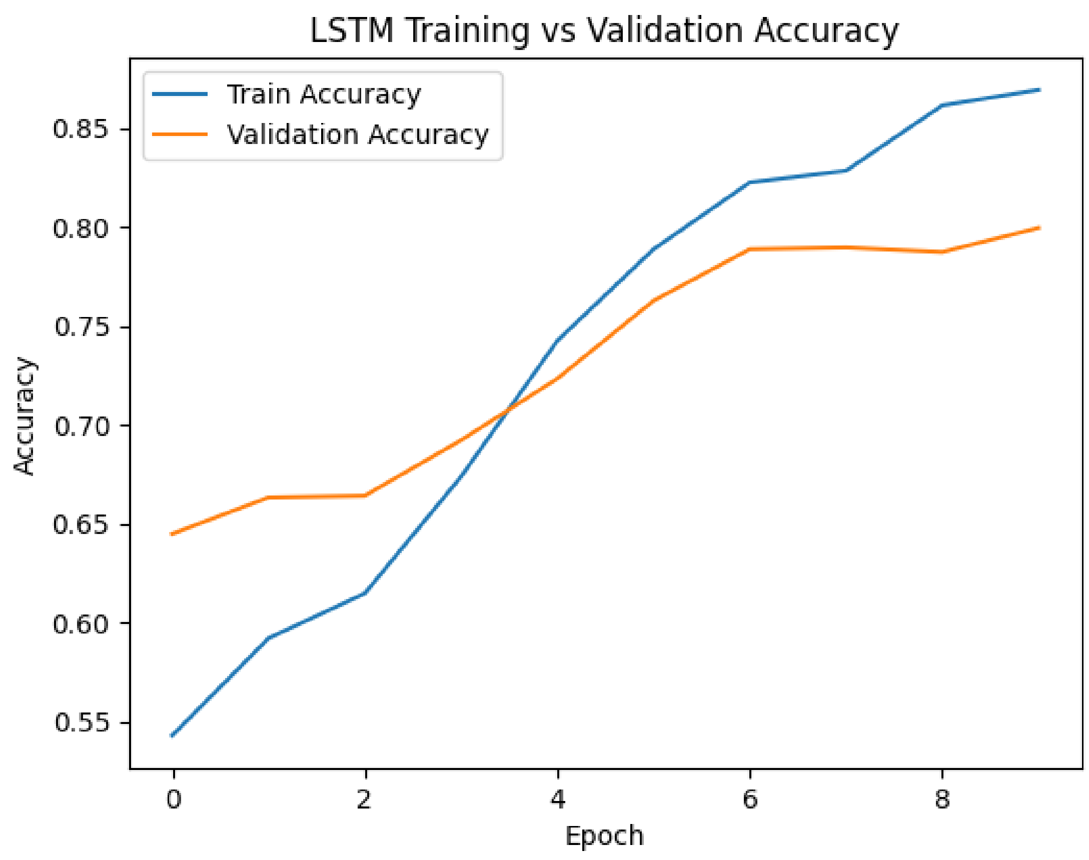
lr = 1e-4

sample_size = 0.3

-Epoch 10: accuracy: 0.8675 - loss: 0.3093 - val_accuracy: 0.7995 - val_loss: 0.4814 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7427 - loss: 0.5979 Test accuracy: 0.7680





Conclusion: Less data requires a lower MAX_LEN and thus a slightly lower accuracy but it is ok for testing parameters

-LSTM(**256**) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

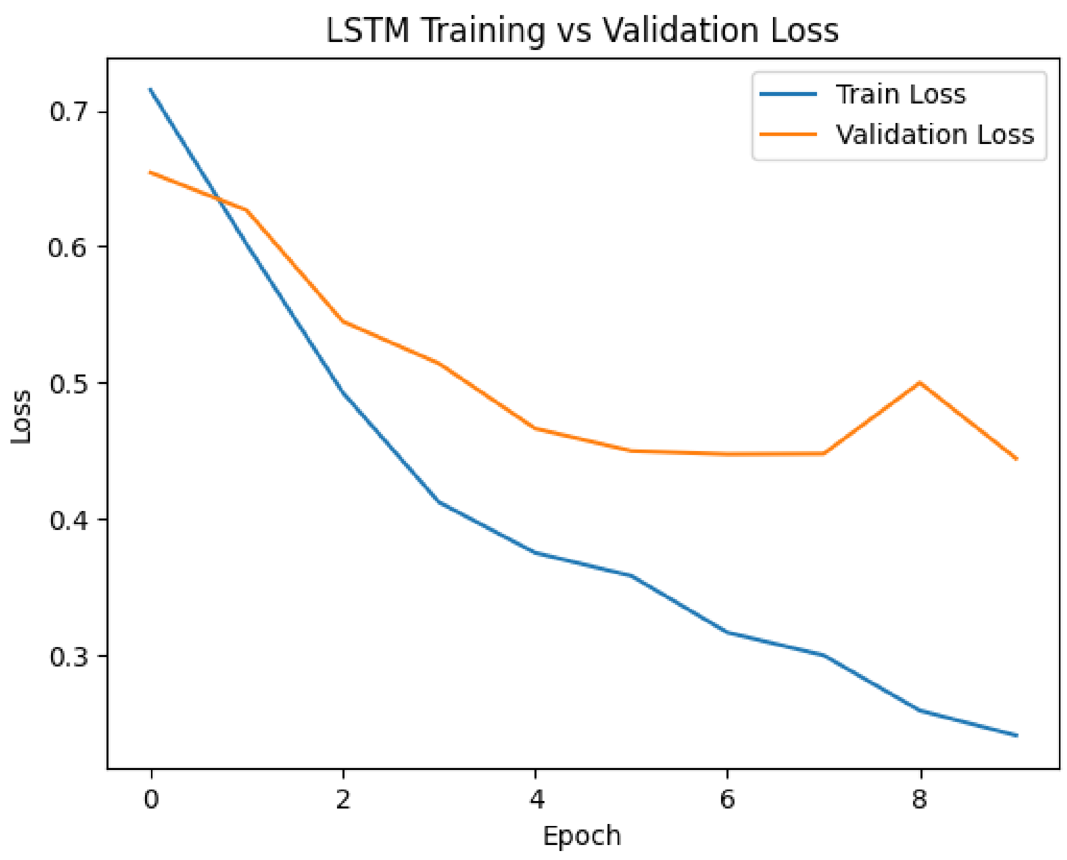
batchsize = 64

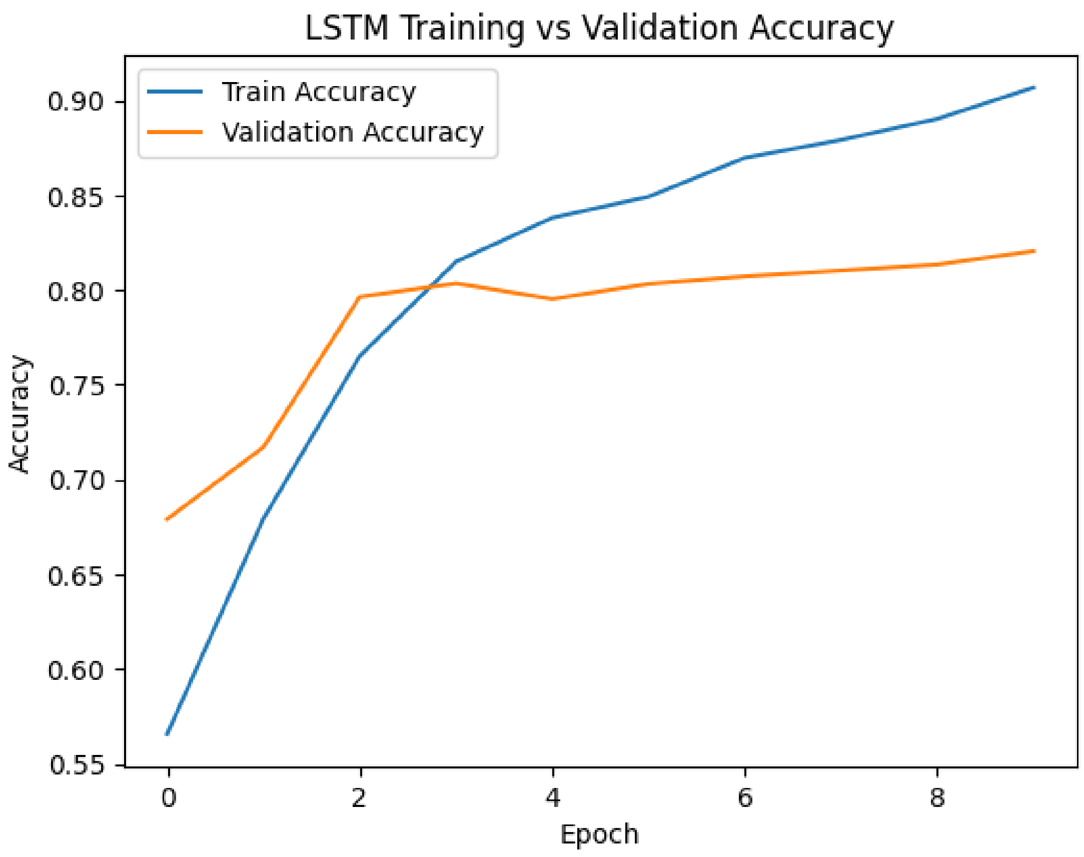
lr = 1e-4

sample_size = 0.3

-Epoch 10: accuracy: 0.9130 - loss: 0.2340 - val_accuracy: 0.8204 - val_loss: 0.4446 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7123 - loss: 0.7022 Test accuracy: 0.7622





Conclusion: several units did not help the model to generalize

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

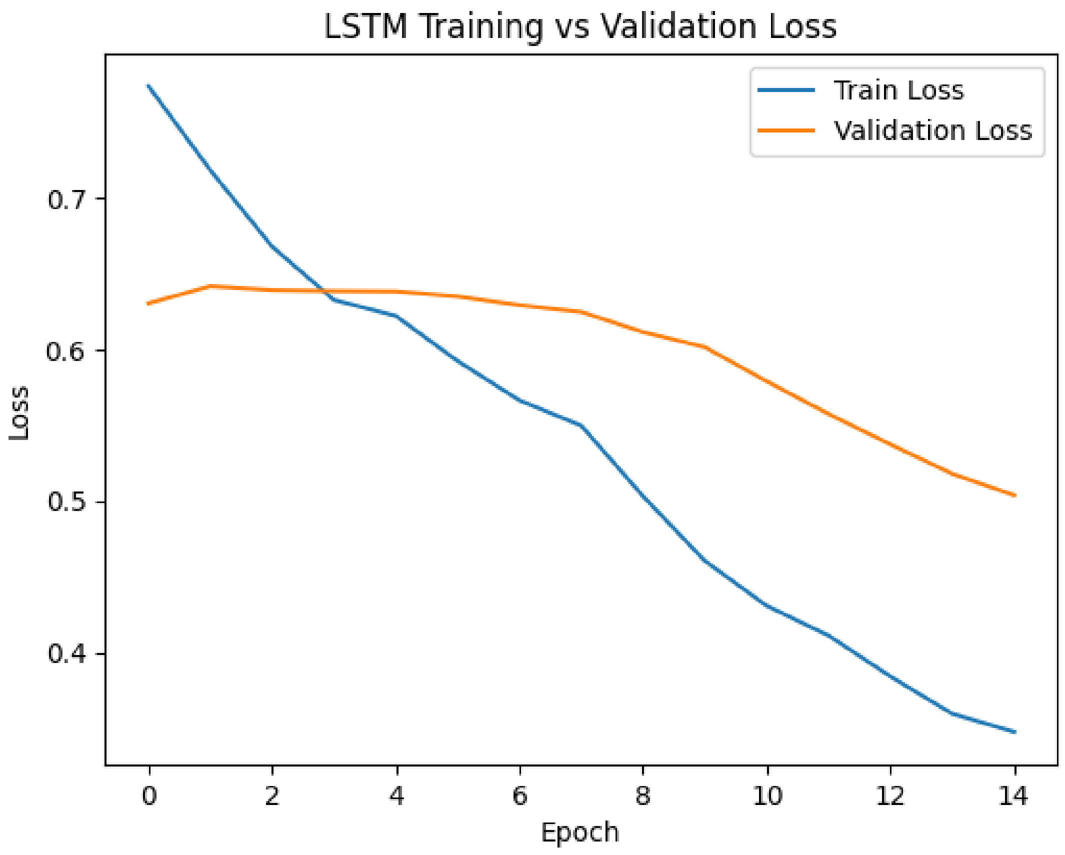
batchsize = 256

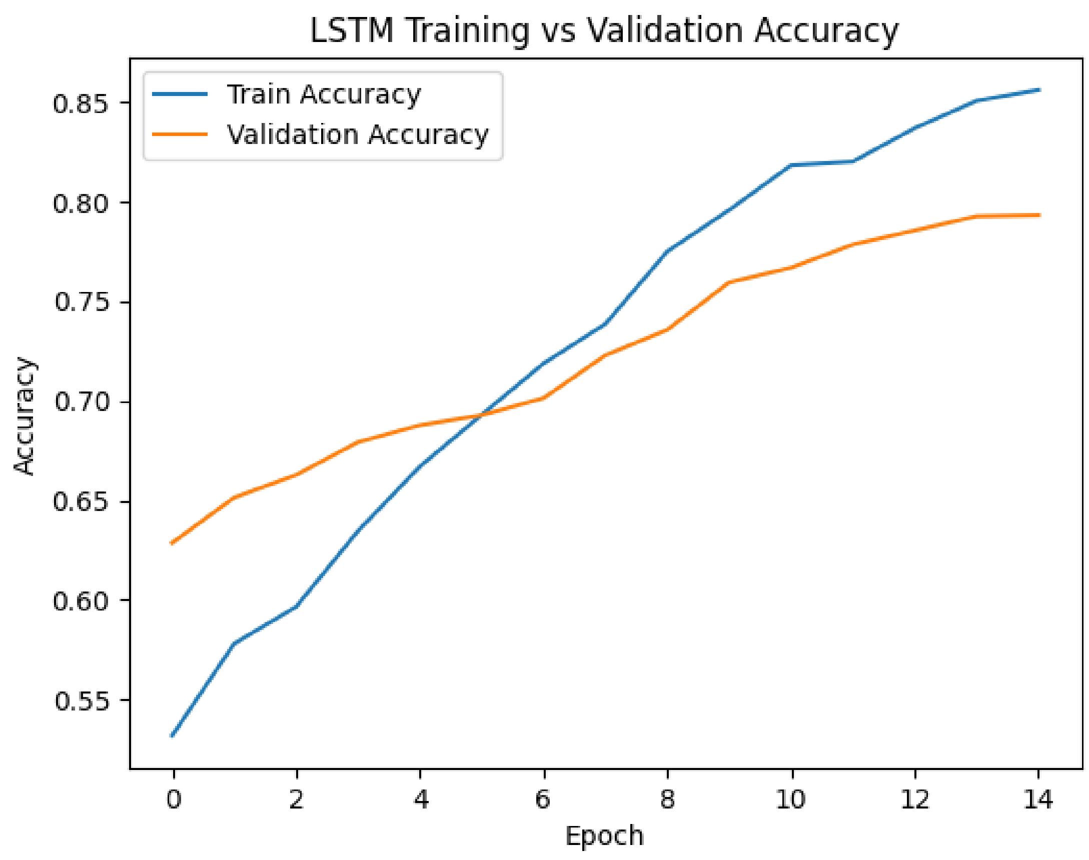
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.8547 - loss: 0.3414 - val_accuracy: 0.7930 - val_loss: 0.5037 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7305 - loss: 0.5624 Test accuracy: 0.7591





Conclusion: Although a larger batch size runs faster and offers slightly worse performance, 64 is balanced

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

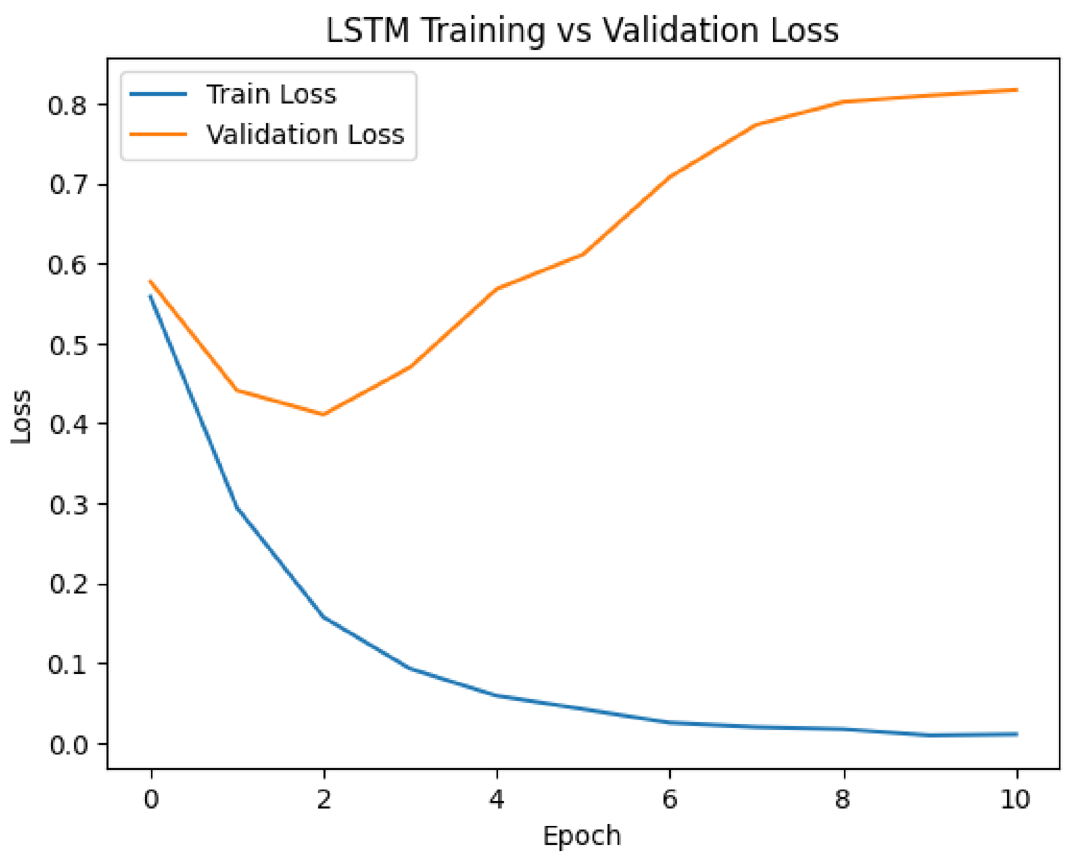
batchsize = 64

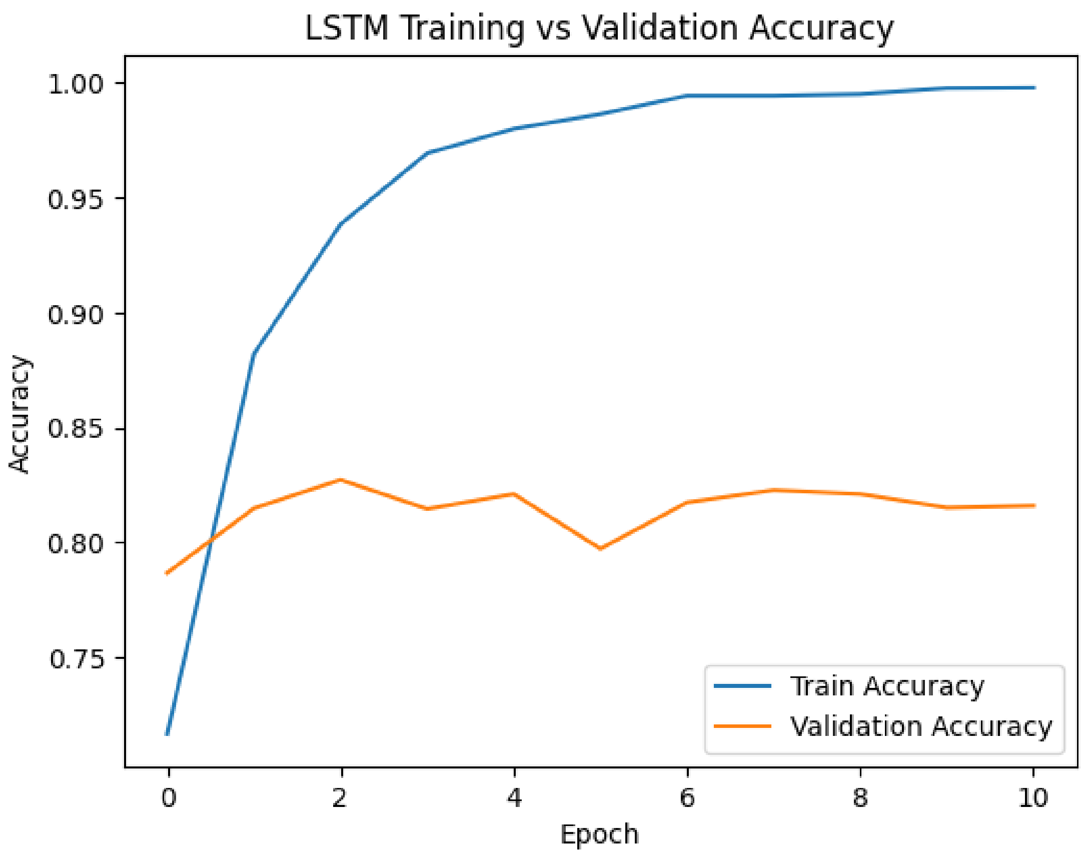
lr = 1e-3

sample_size = 0.3

-Epoch 15: accuracy: 0.9987 - loss: 0.0092 - val_accuracy: 0.8159 - val_loss: 0.8172 - learning_rate: 5000e-04

-Test: accuracy: 0.7941 - loss: 0.4573 Test accuracy: 0.7630


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8
10

Conclusion: Initial learning rate too high causes strong overfitting, we keep 2e-4

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and **without BatchNorm**

+ Dense(1,sigmoid)

MAX_LEN = 30

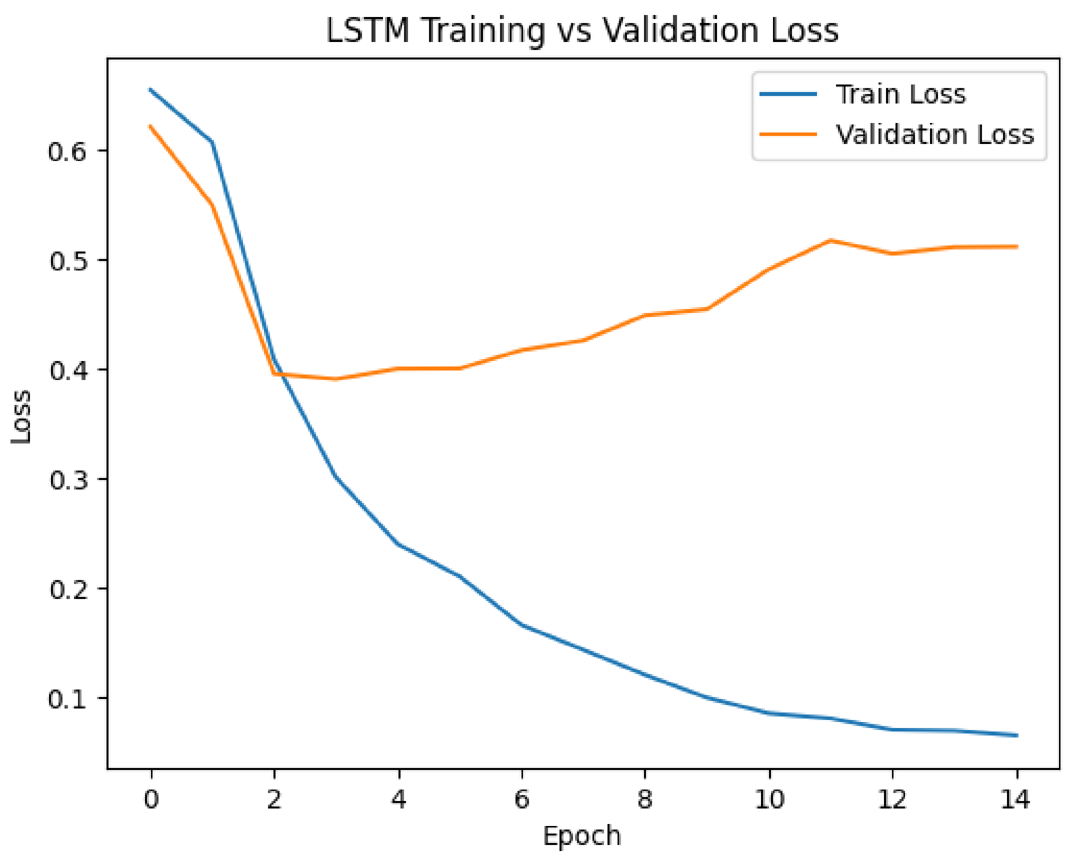
batchsize = 64

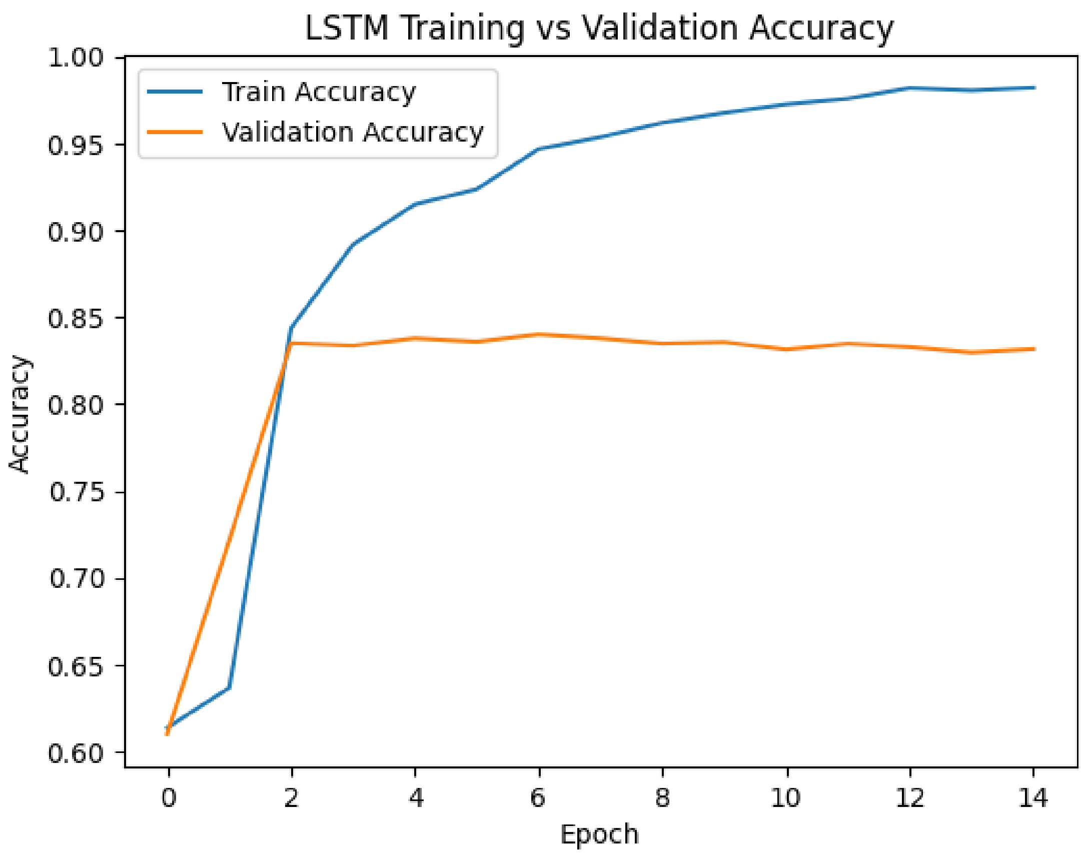
lr = 2e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.9811 - loss: 0.0697 - val_accuracy: 0.8315 - val_loss: 0.5111 - learning_rate: 5.0000e-05

-Test: accuracy: 0.7201 - loss: 0.6913 Test accuracy: 0.7614


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8
10
12
14

Conclusion: Without BatchNorm the learning rate dropped a lot and the model overfitted.

-**Bidirectional**(LSTM(64) with Dropout(0.3)) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

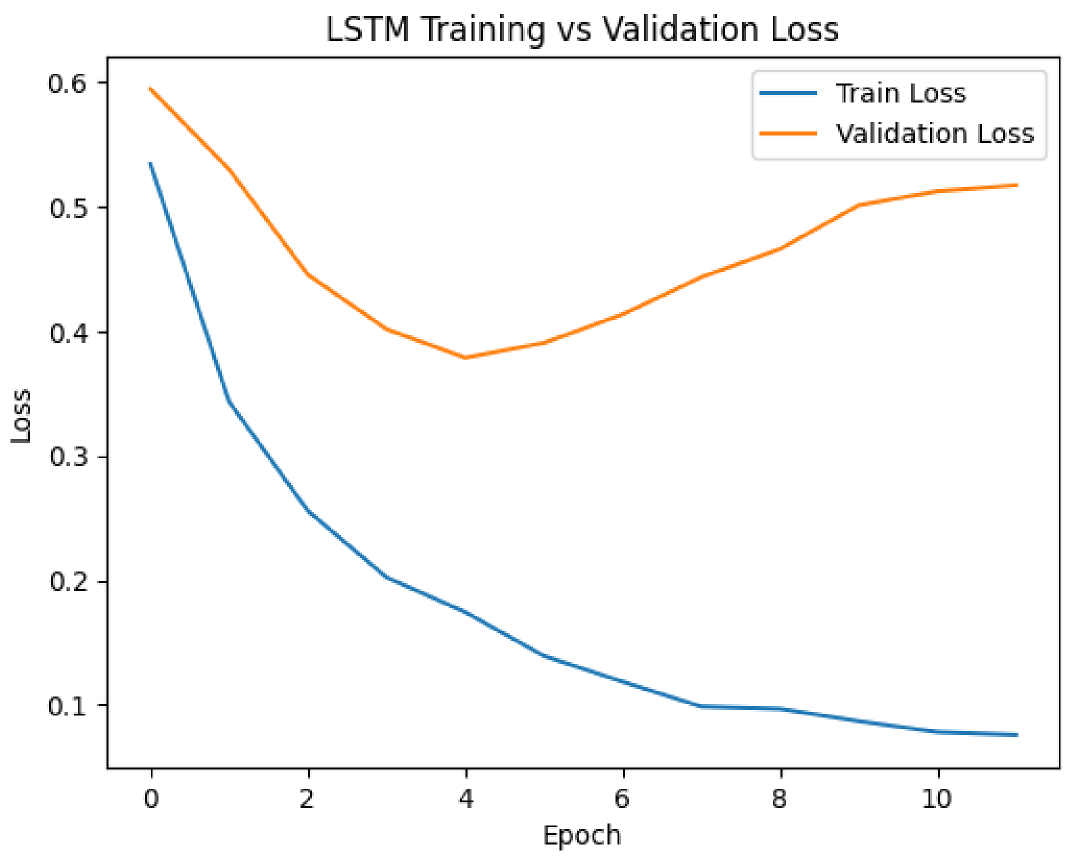
batchsize = 64

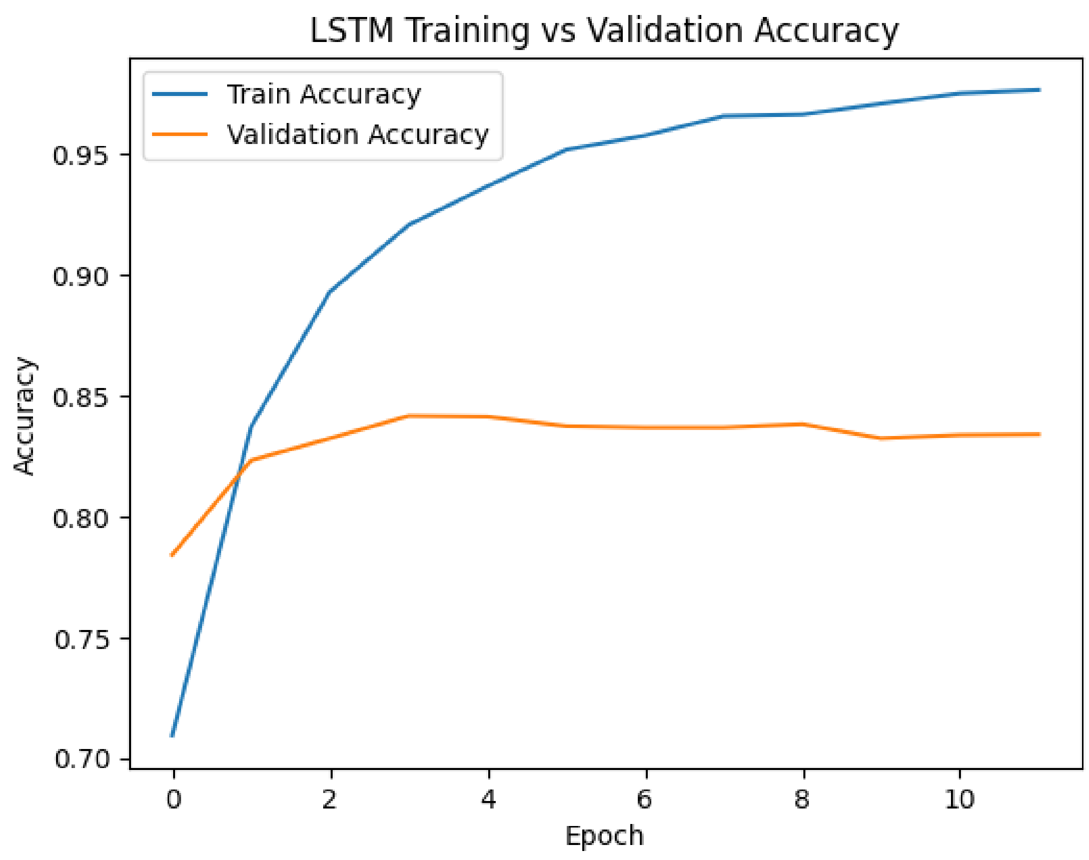
lr = 2e-4

sample_size = 0.3

-Epoch 12: accuracy: 0.9773 - loss: 0.0735 - val_accuracy: 0.8340 - val_loss: 0.5172 - learning_rate: 5.0000e-05

-Test: accuracy: 0.7512 - loss: 0.5236 Test accuracy: 0.7672





Conclusion: once again overfitting, learning rate drops to 1e-4 for bidirectional

-**Bidirectional**(LSTM(64) with Dropout(0.3)) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

MAX_LEN = 30

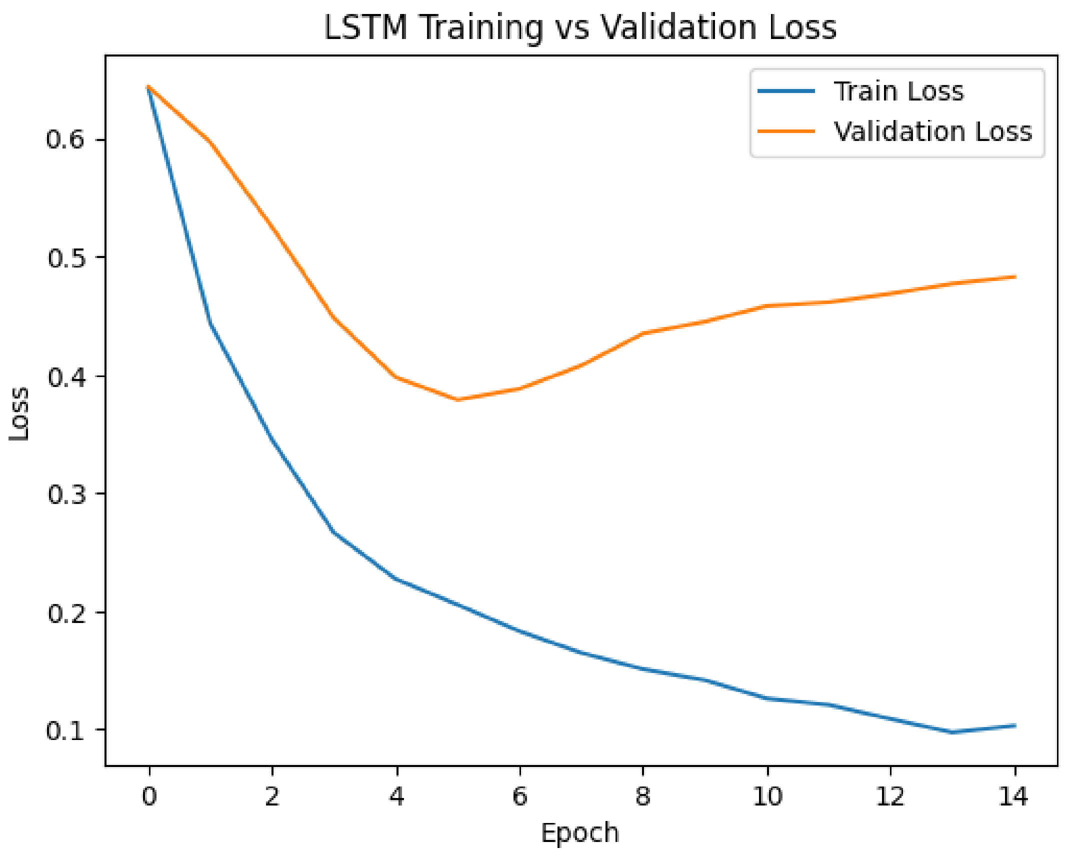
batchsize = 64

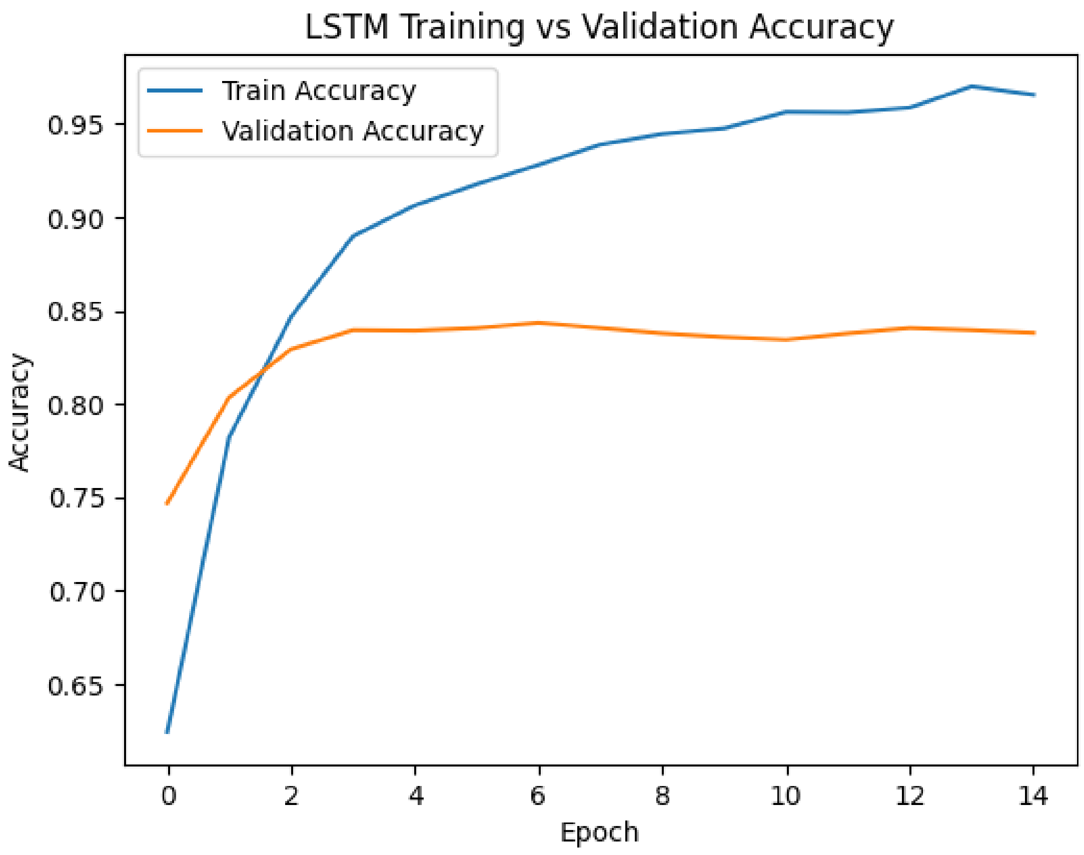
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.9655 - loss: 0.1056 - val_accuracy: 0.8380 - val_loss: 0.4829 - learning_rate: 5000e-05

-Test: accuracy: 0.7443 - loss: 0.5795 Test accuracy: 0.7683


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0
2
4
6
8
10
12
14

Bottom Line: Bidirectional Delivers Similar Results But Does A Little Overfitting

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(64,sigmoid)

+ Dense(32,sigmoid)

+ Dense(16,sigmoid)

+ Dense(1,sigmoid)

MAX_LEN = 30

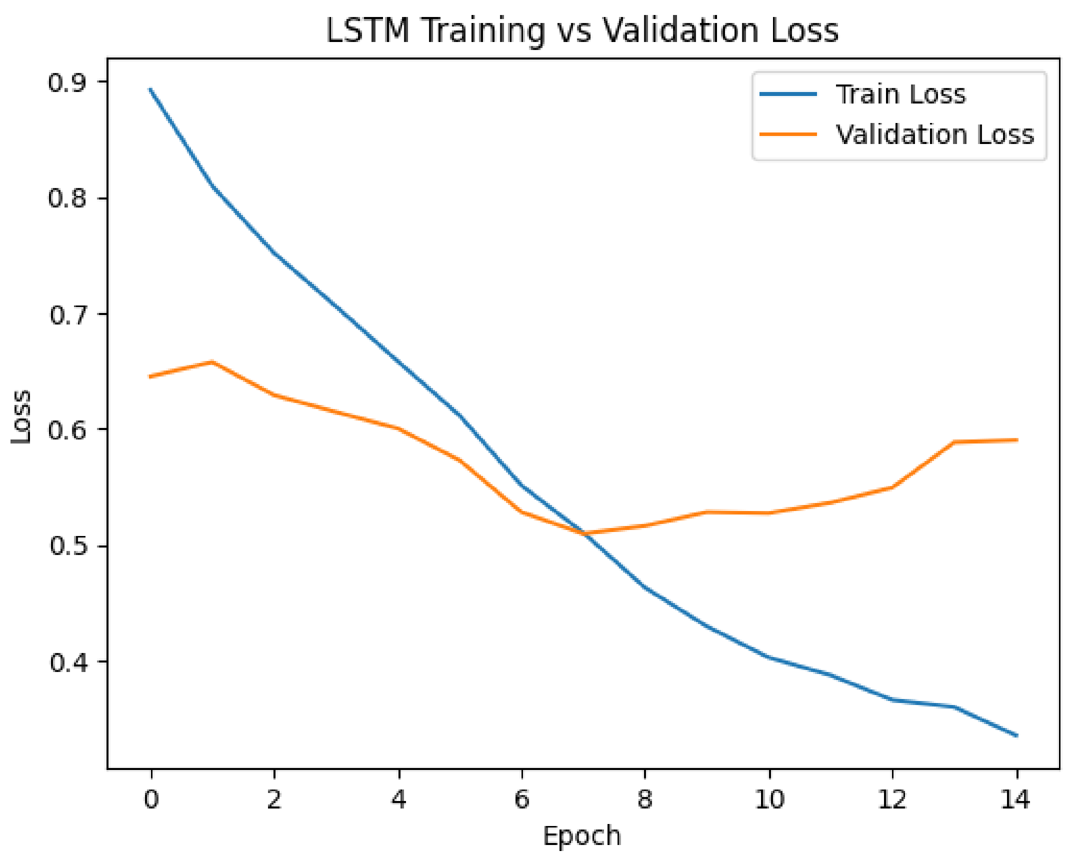
batchsize = 64

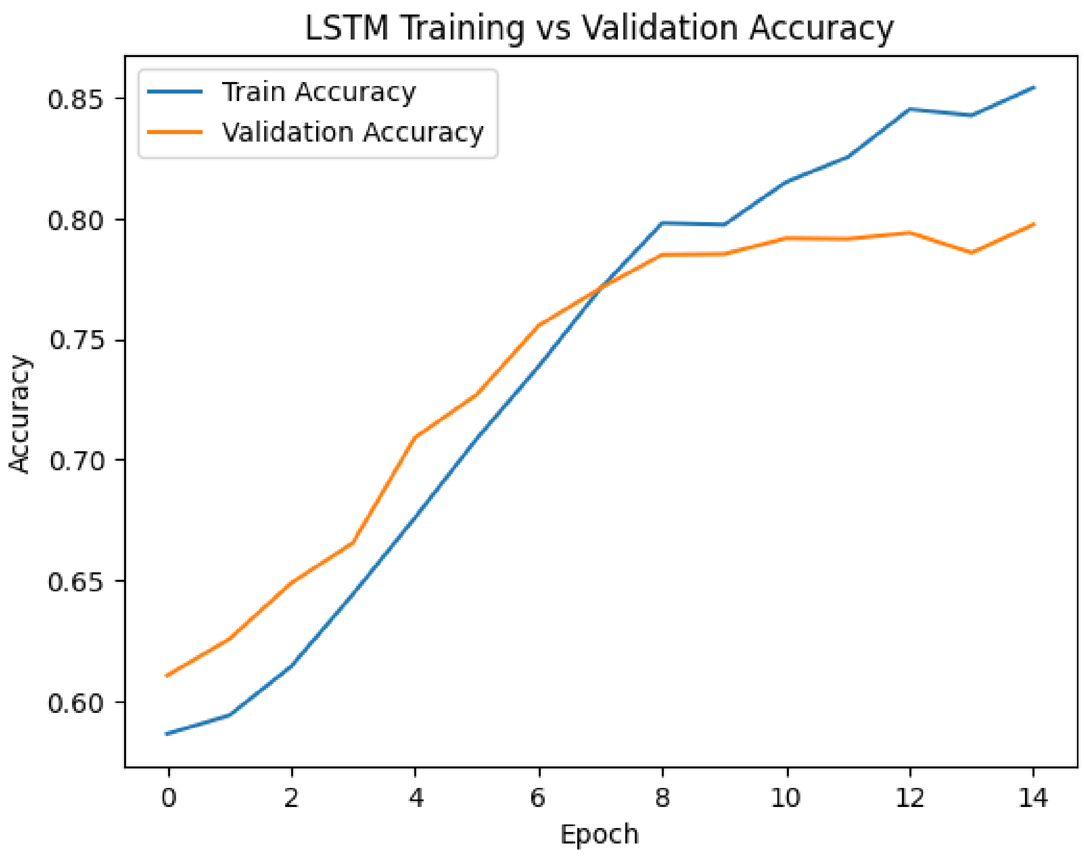
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.8560 - loss: 0.3388 - val_accuracy: 0.7975 - val_loss: 0.5904 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7364 - loss: 0.7920 Test accuracy: 0.7622


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10
12
14


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
10
12
14

Bottom line: too many linear layers make running time harder and don't improve performance

-LSTM(64) + LSTM(32) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(16,sigmoid)

+ Dense(1,sigmoid)

MAX_LEN = 30

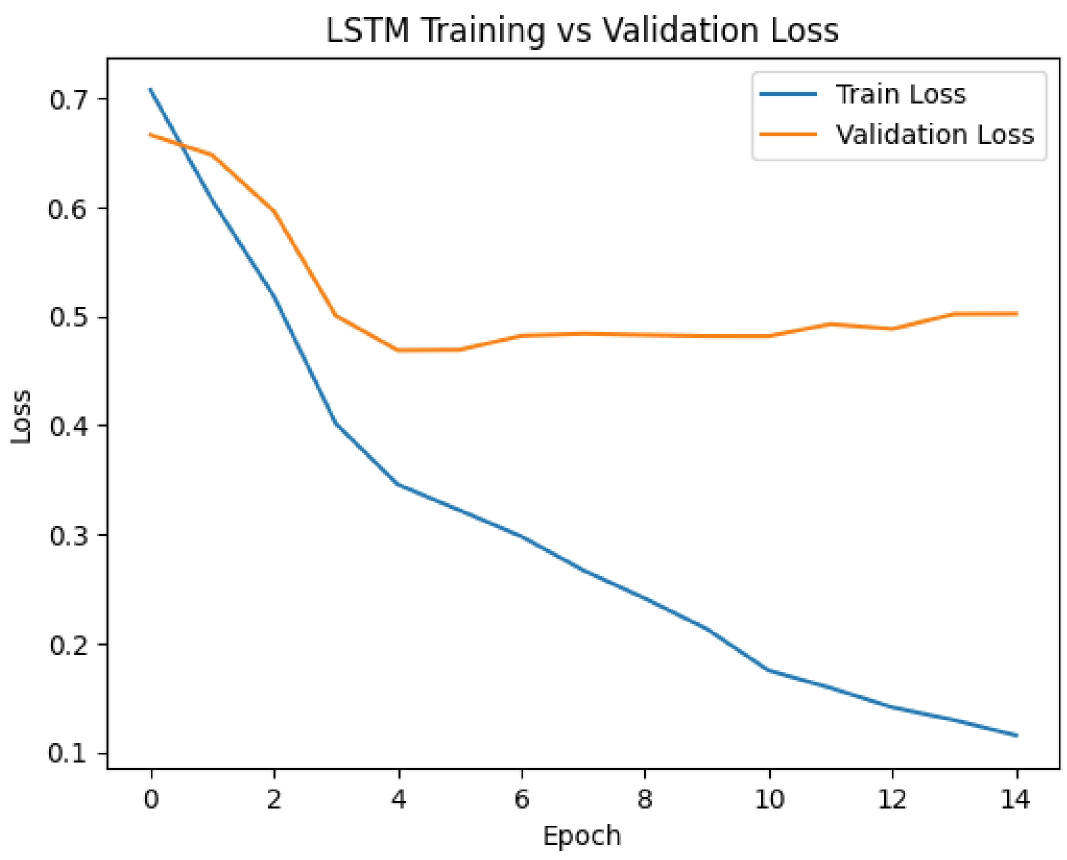
batchsize = 64

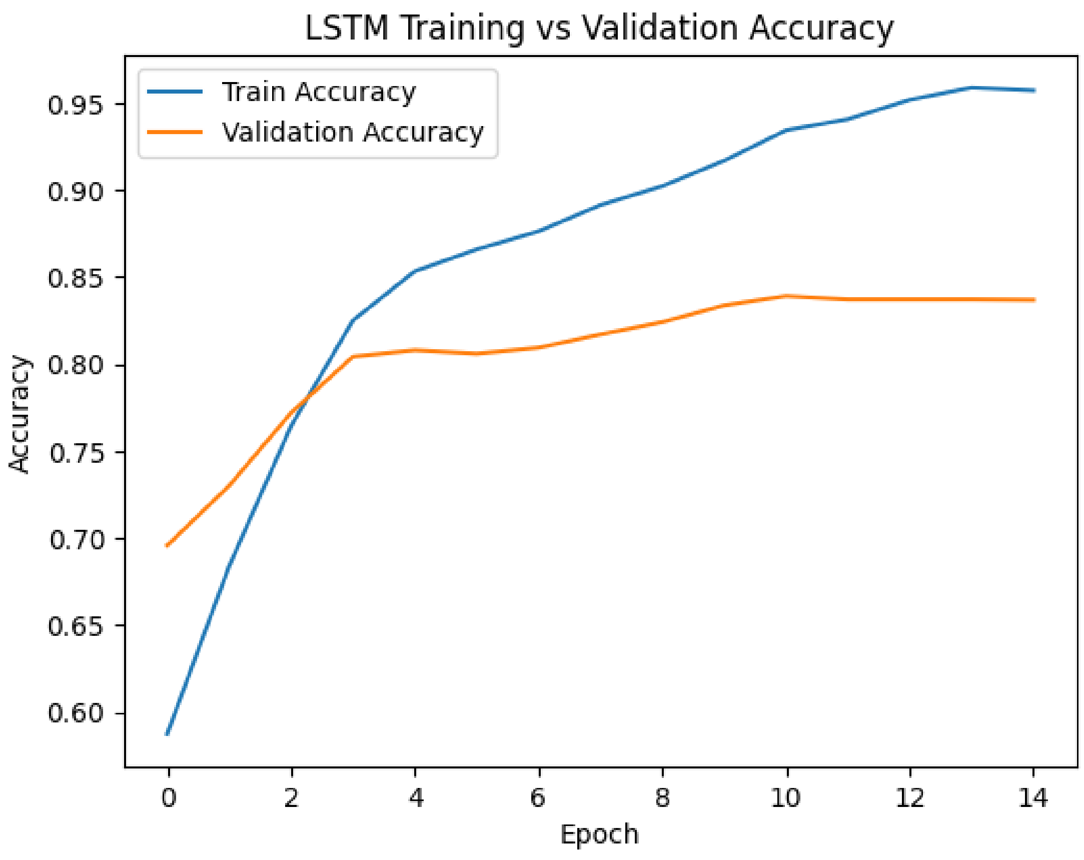
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.9589 - loss: 0.1587 - val_accuracy: 0.8394 - val_loss: 0.4763 - learning_rate: 5.0000e-05

-Test: accuracy: 0.7762 - loss: 0.6180 Test accuracy: 0.7698


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
Epoch
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
6
8
10
12
14


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0
2
4
6
8
10
12
14

Conclusion: It has a longer runtime but similar performance as the LSTM single-layer network.

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

+delete and swap augumentations

MAX_LEN = 30

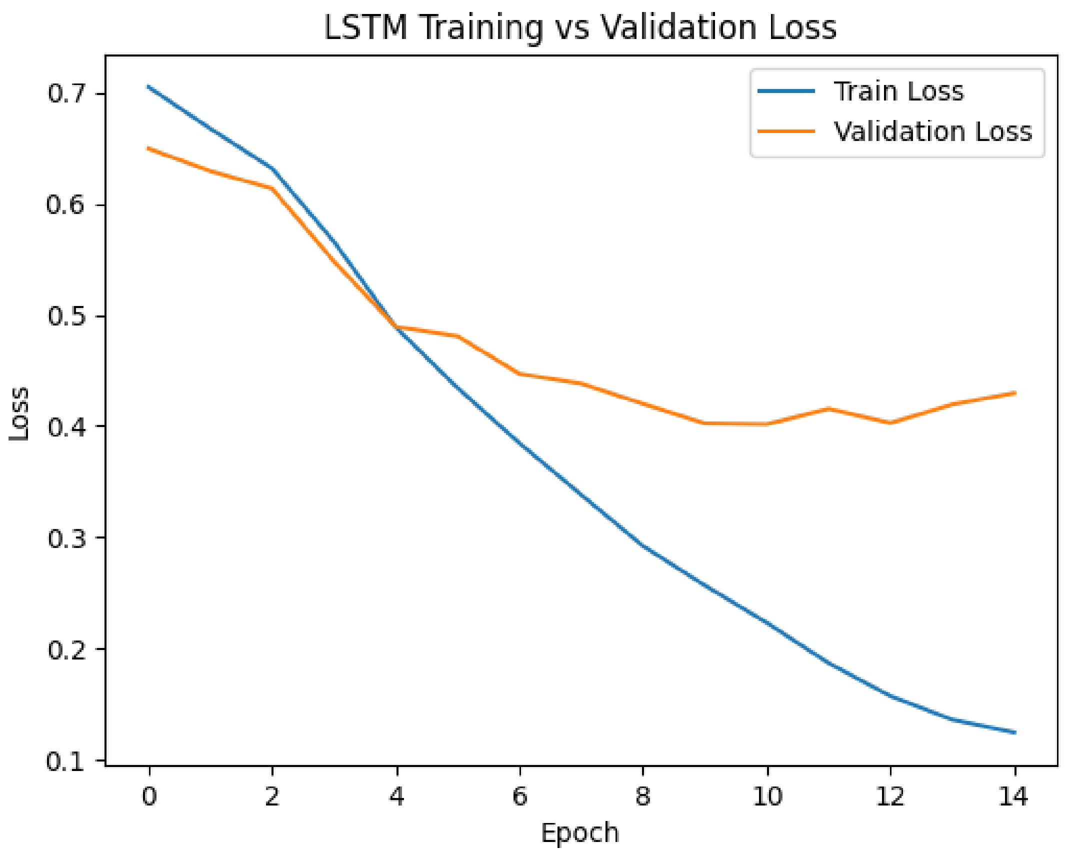
batchsize = 64

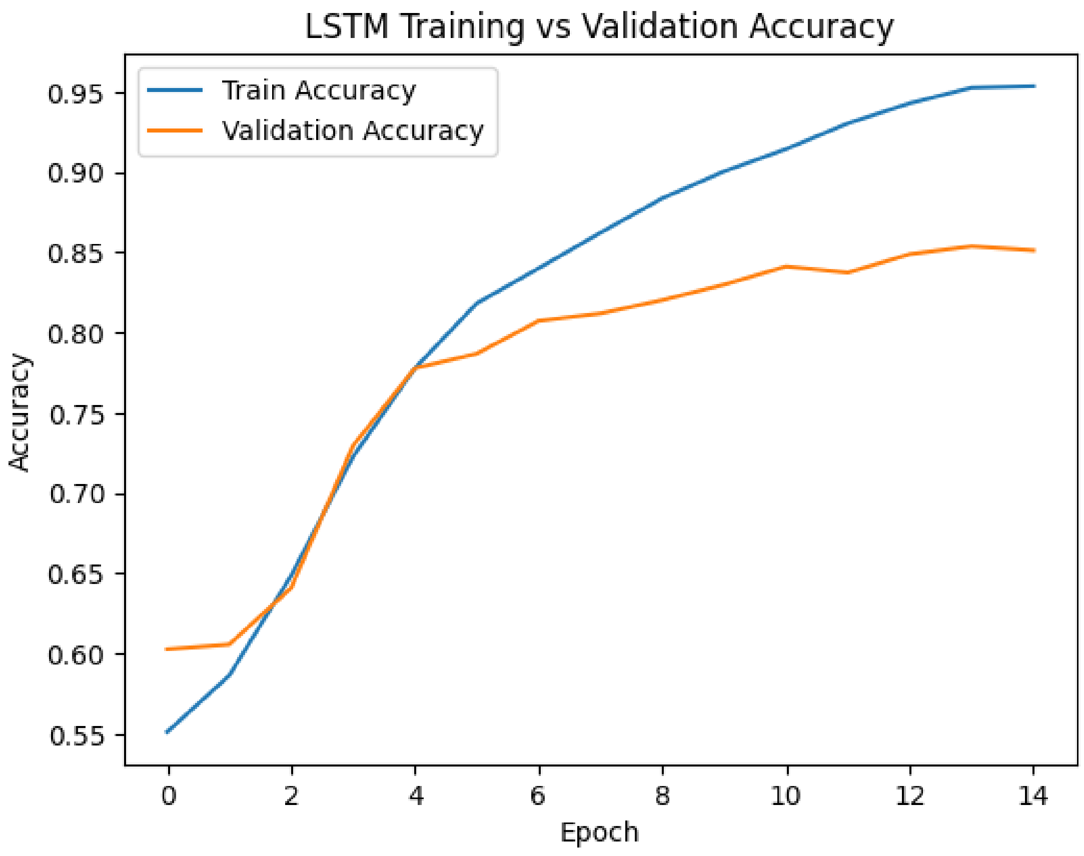
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: accuracy: 0.9516 - loss: 0.1289 - val_accuracy: 0.8513 - val_loss: 0.4297 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7427 - loss: 0.7757 Test accuracy: 0.7546


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
6
8
10
12
14
Epoch


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0
2
4
6
8
10
12
14
Epoch

Conclusion: The delete and swap increases stabilize the model but offer a slightly weaker performance. They may need more data.

-LSTM(64) + LSTM(32) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+Dense(16, activation='relu')

+ Dense(1,sigmoid)

+delete and swap augumentations

MAX_LEN = 30

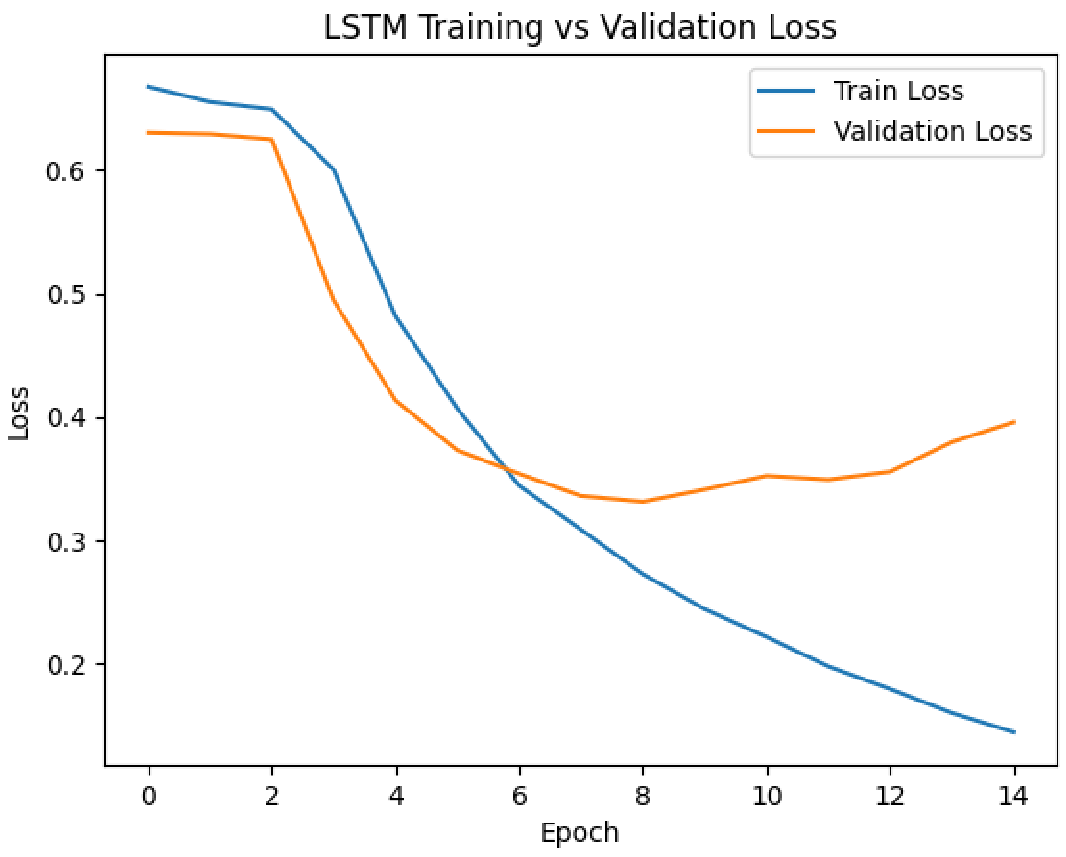
batchsize = 64

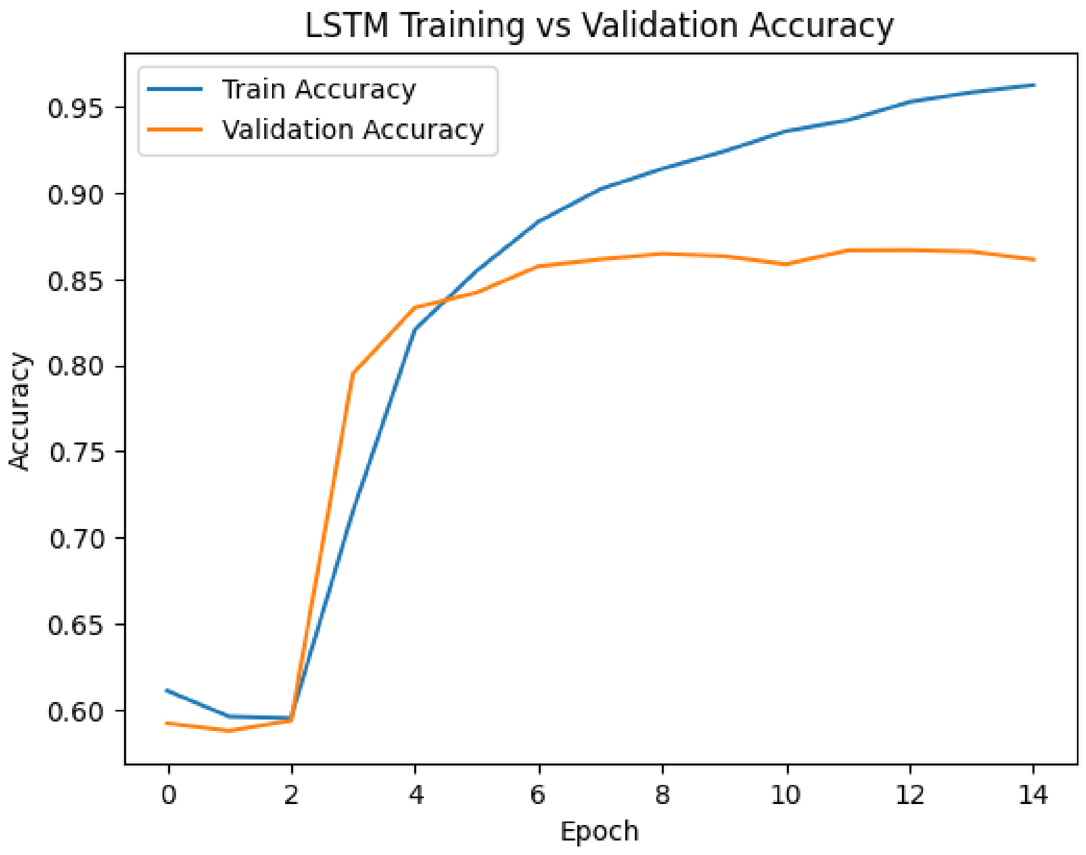
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: 0.9620 - loss: 0.1454 - val_accuracy: 0.8612 - val_loss: 0.3956 - learning_rate: 1.0000e-04

-Test: accuracy: 0.9620 - loss: 0.1454 - val_accuracy: 0.8612 - val_loss: 0.3956 - learning_rate: 1.0000e-04


LSTM Training vs Validation Loss
Train Loss
Validation Loss
Loss
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
10
12
14
Epoch


LSTM Training vs Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
10
12
14
Epoch

Conclusion: Although the augmentations achieved high accuracy on the train and validation on the test, they obtained worse results

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

+BERT context swap augumentations

MAX_LEN = 30

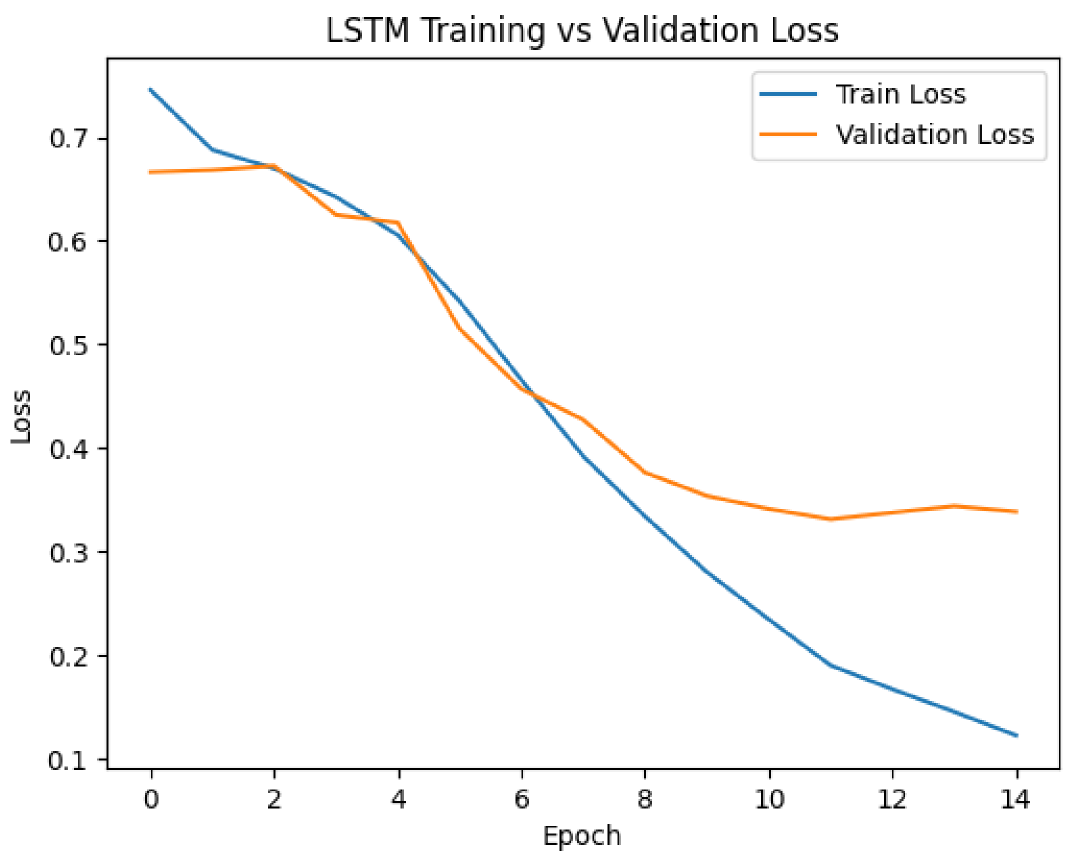
batchsize = 64

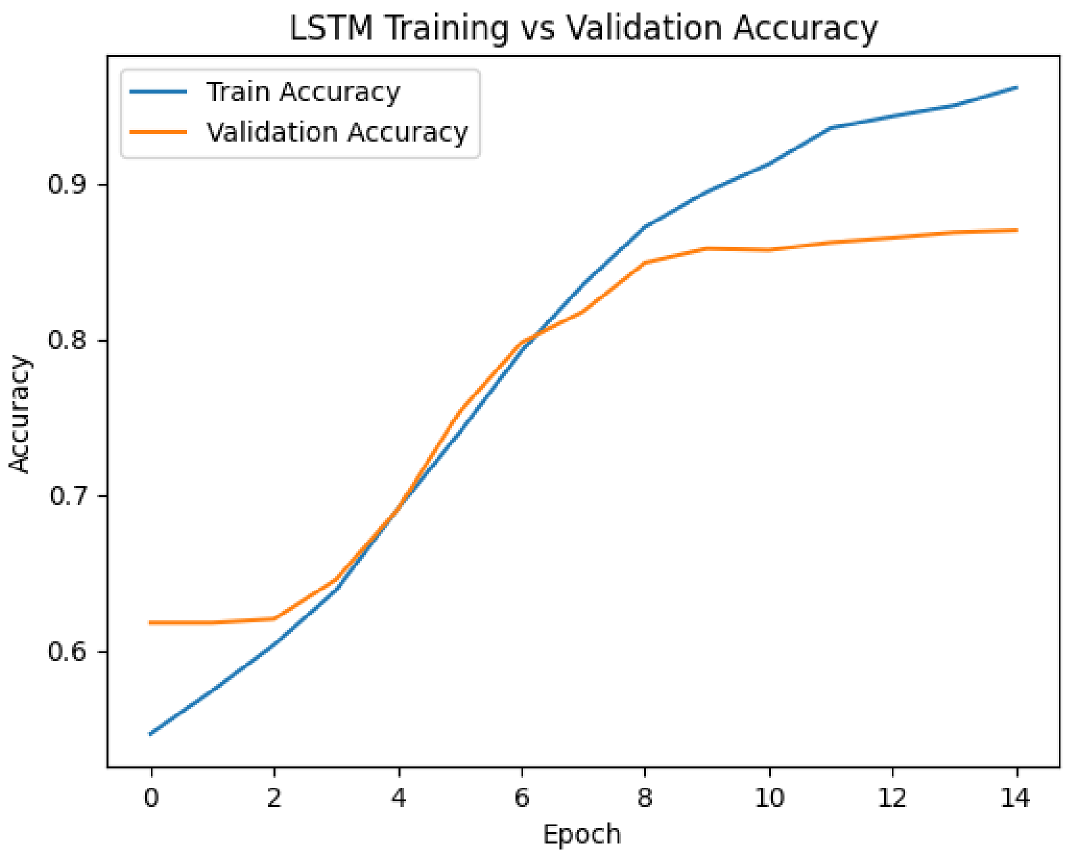
lr = 1e-4

sample_size = 0.3

-Epoch 15: accuracy: accuracy: 0.9596 - loss: 0.1283 - val_accuracy: 0.8697 - val_loss: 0.3387 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7509 - loss: 0.7142 Test accuracy: 0.7776





Conclusion: Context change helps the model to generalize better

-LSTM(64) + LSTM(32) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+Dense(16, activation='relu')

+ Dense(1,sigmoid)

+BERT Context swap augumentations

MAX_LEN = 30

batchsize = 64

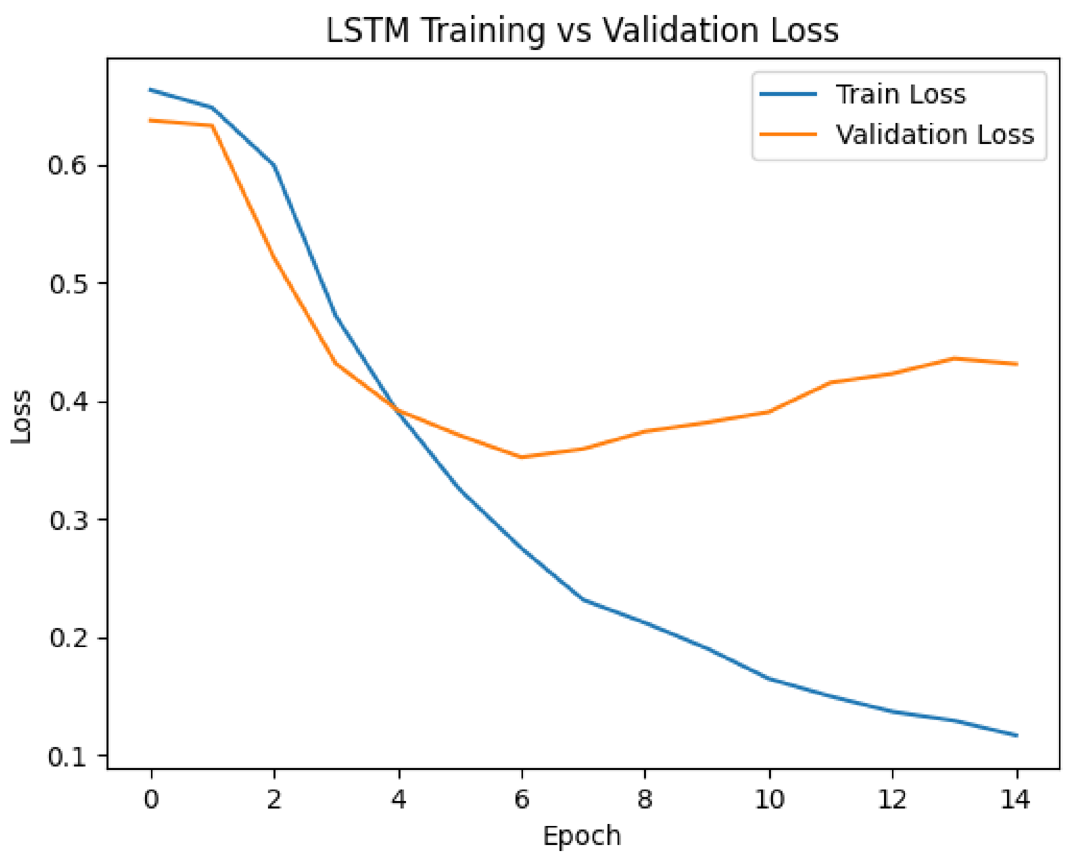
lr = 1e-4

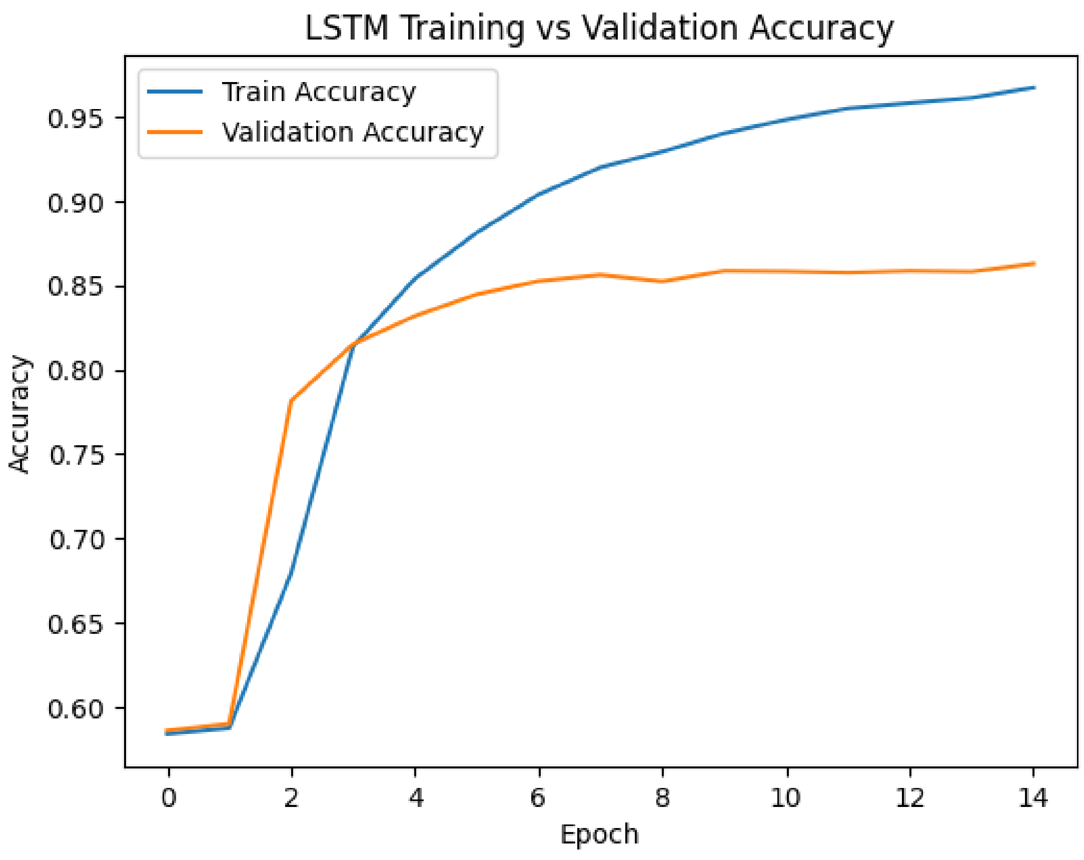
sample_size = 0.3

-Epoch 15: accuracy: 0.9620 - loss: 0.1454 - val_accuracy: 0.8612 - val_loss: 0.3956 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7418 - loss: 0.8235

Test accuracy: 0.7548





Conclusion: Simple networks are more suitable for augmentations

-LSTM(64) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

+BERT context swap augumentations

MAX_LEN = 50

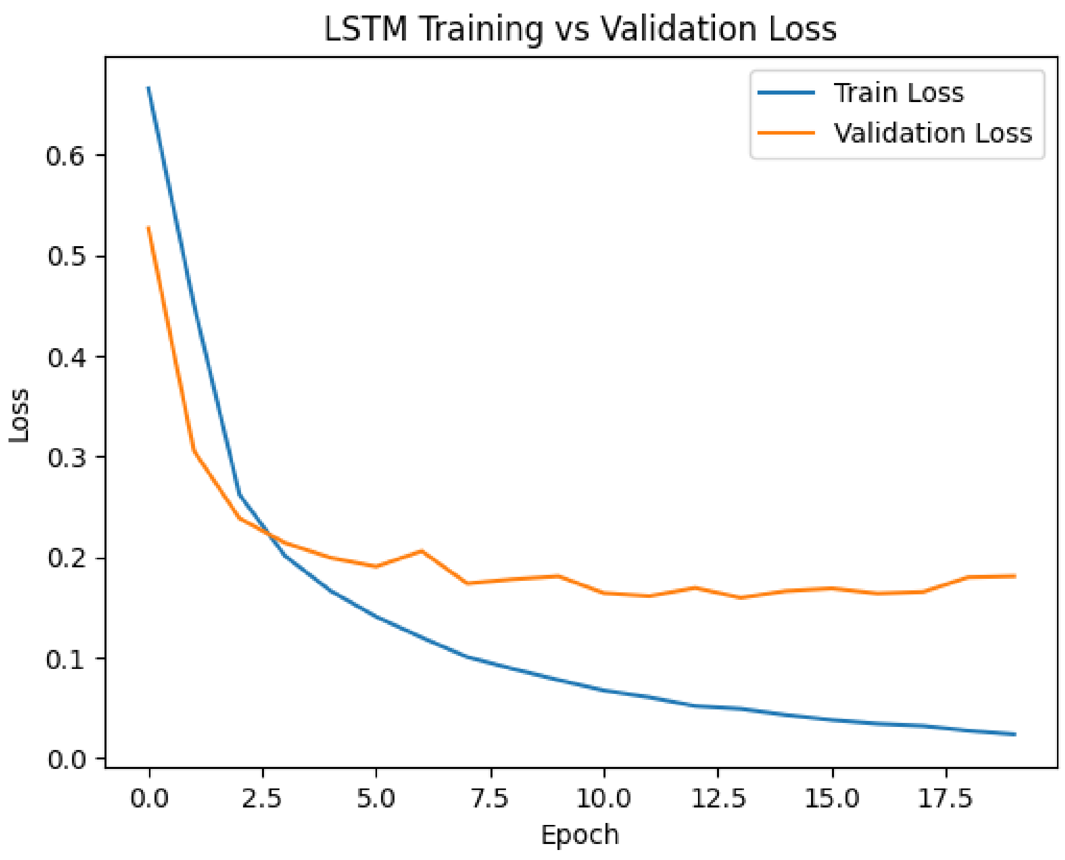
batchsize = 64

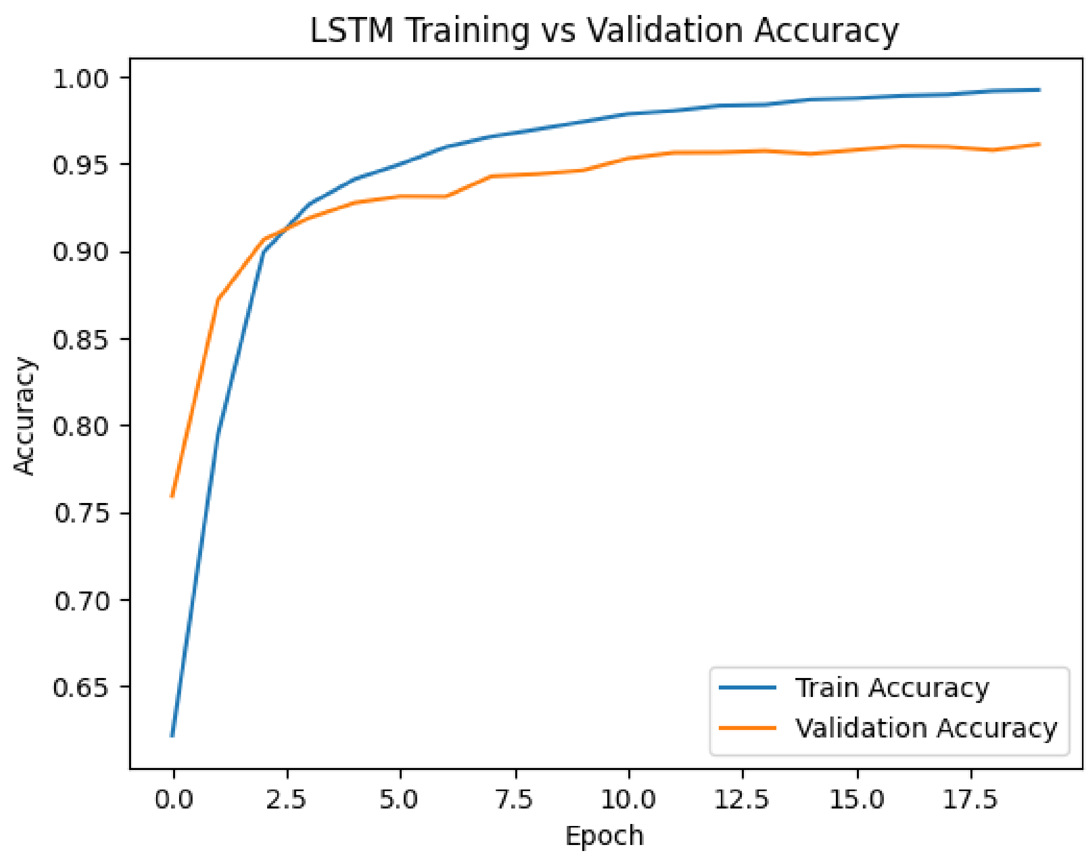
lr = 1e-4

sample_size = 1.0

-Epoch 20: accuracy: 0.9925 - loss: 0.0239 - val_accuracy: 0.9612 - val_loss: 0.1809 - learning_rate: 1.0000e-04

-Test: accuracy: 0.7734 - loss: 1.2602 Test accuracy: 0.7866





Conclusion: Even if this configuration is the best, it does not offer a significant improvement over the original model

Final:

-LSTM(64) + LSTM(32) with Dropout(0.3) and recurrent_dropout = 0.1 and BatchNorm

+ Dense(1,sigmoid)

+BERT context swap augumentations

MAX_LEN = 70

batchsize = 64

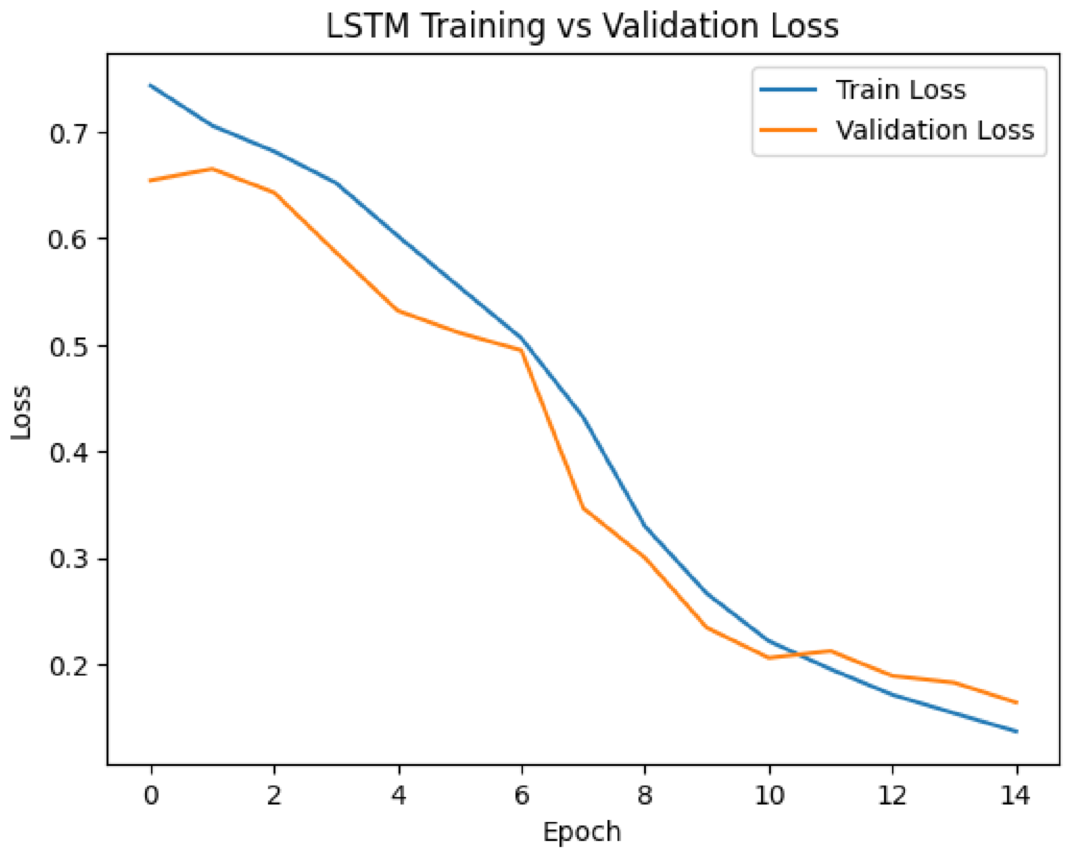
lr = 1e-4

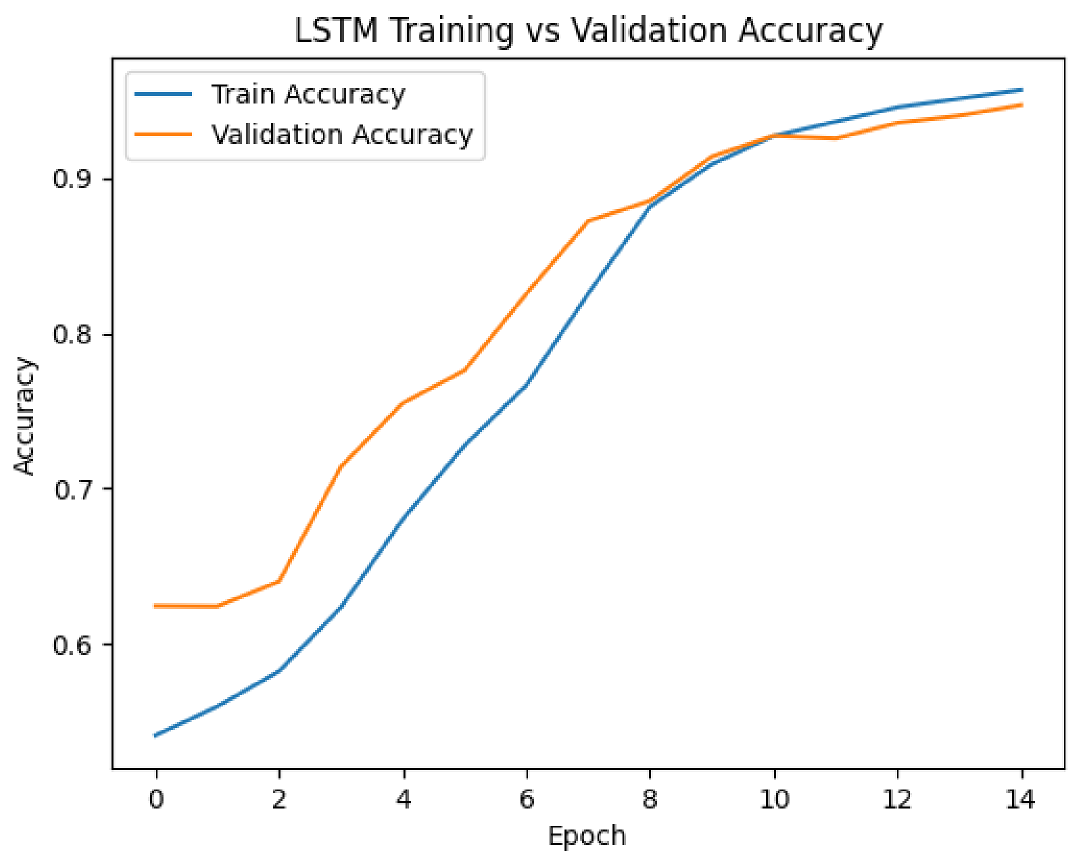
sample_size = 1.0

-Epoch 15: accuracy: 0.9570 - loss: 0.1347 - val_accuracy: 0.9467 - val_loss: 0.1643 - learning_rate: 1.0000e-04

-Test: accuracy: 0.8175 - loss: 0.6589

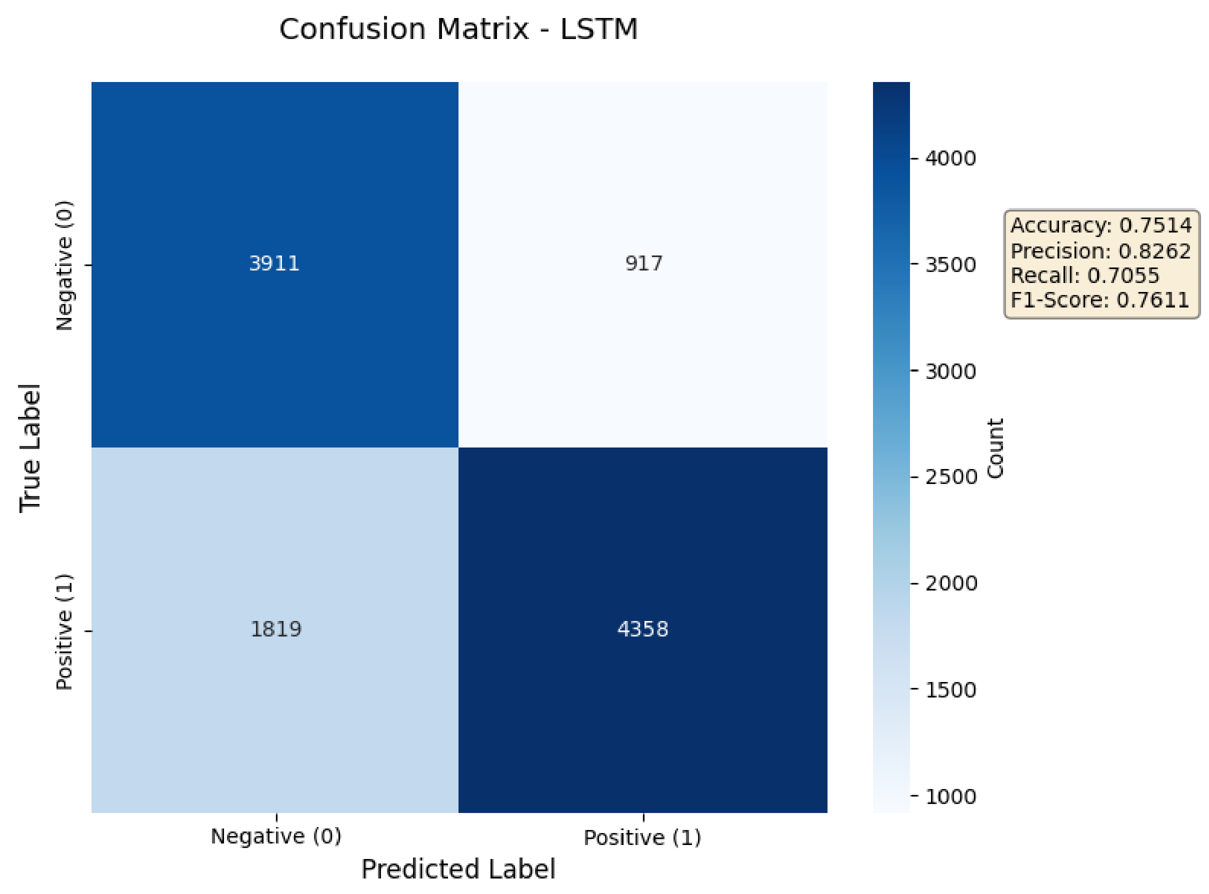
Test accuracy: 0.8194





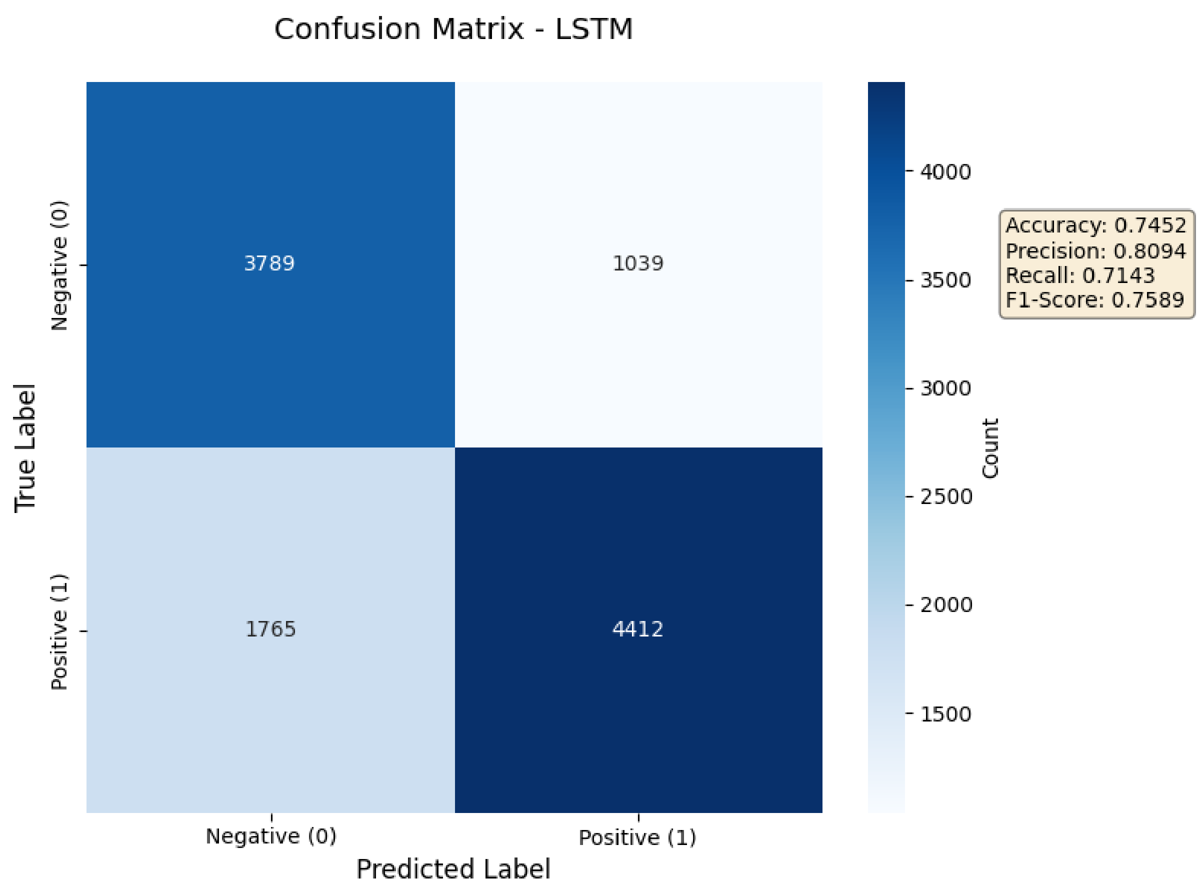
Conclusion: Although due to the high MAX_LEN, the more complex architecture and the entire size of the dataset, the training takes an hour (20 times longer than the others) we get good results, which proves that the LSTM architecture is superior if we give it enough time

LSTM:



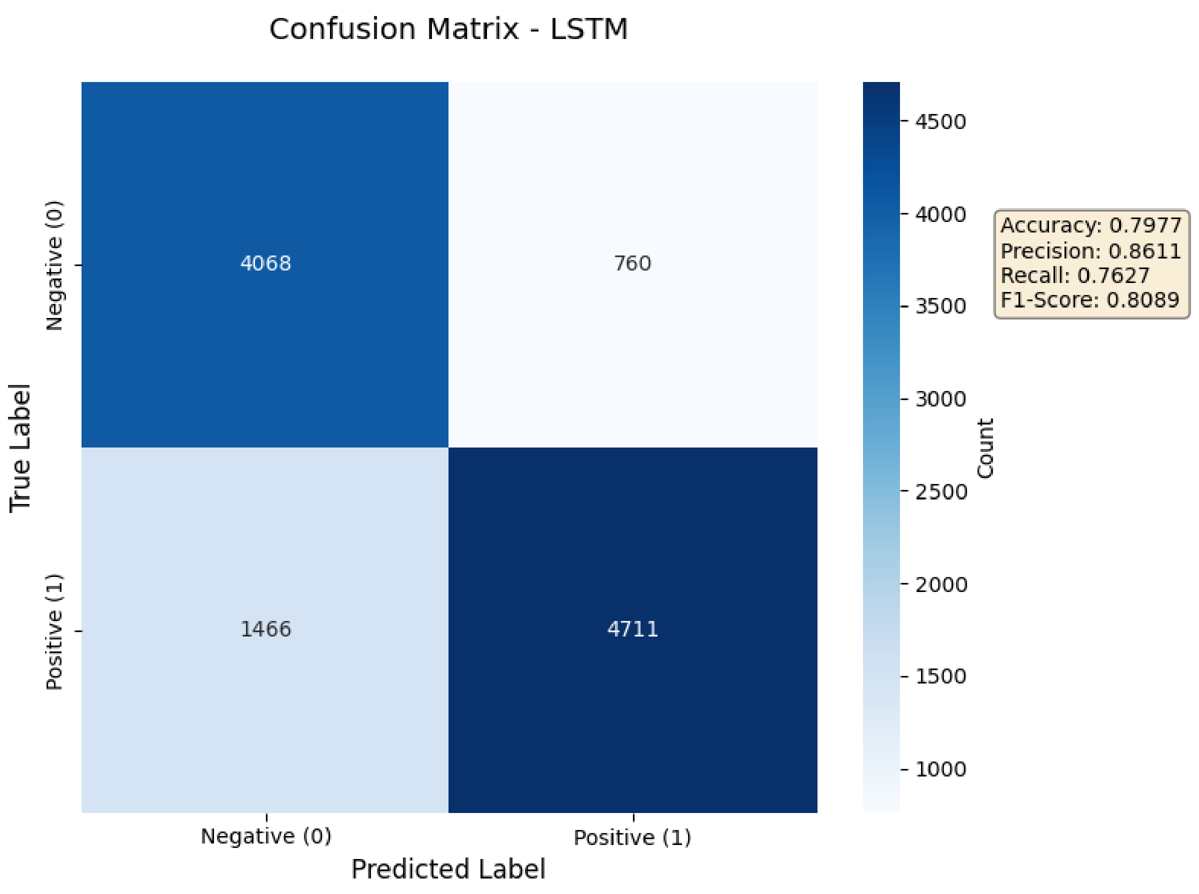
LSTM:

+Swap and Delete Augs:

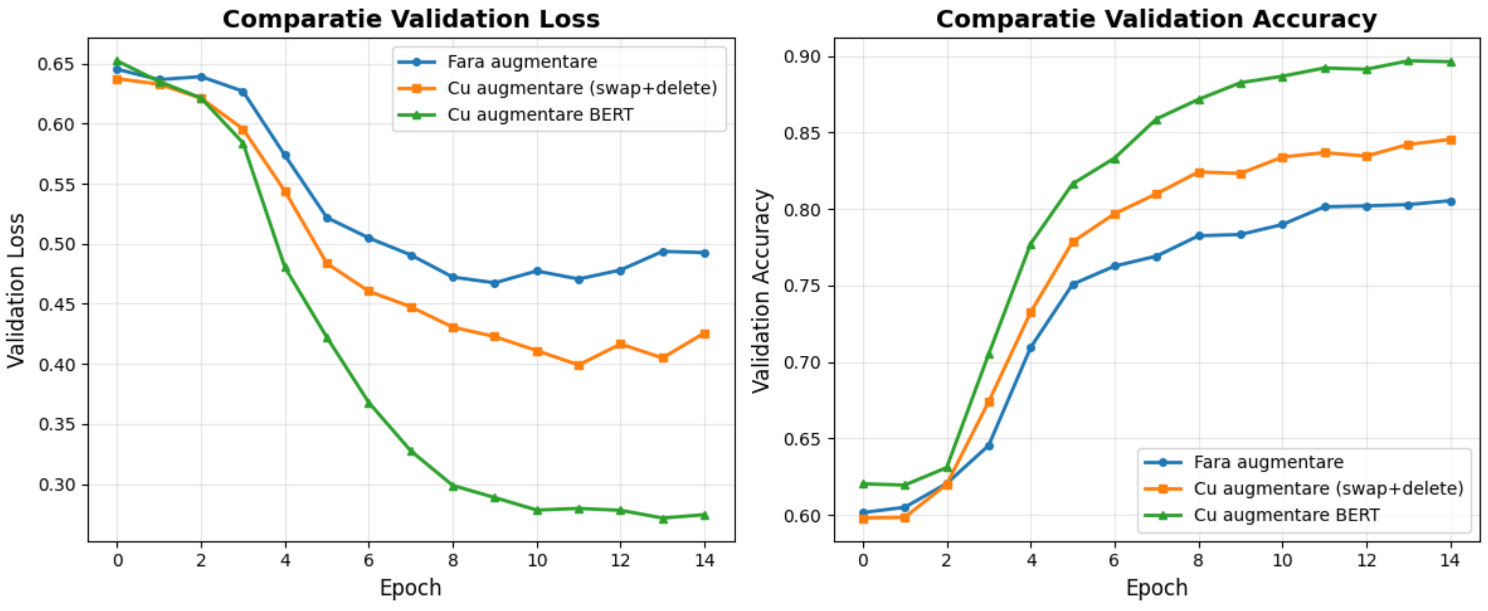


LSTM:

+BERT context SWAP:



Confusion Matrix - LSTM
True Label
Negative (0)
Positive (1)
4068
760
1466
4711
Predicted Label
Negative (0)
Positive (1)
Count
4500
4000
3500
3000
2500
2000
1500
1000
Accuracy: 0.7977
Precision: 0.8611
Recall: 0.7627
F1-Score: 0.8089



Comparatie Validation Loss
Validation Loss
Epoch
Fara augmentare
Cu augmentare (swap+delete)
Cu augmentare BERT
Comparatie Validation Accuracy
Validation Accuracy
Epoch
Fara augmentare
Cu augmentare (swap+delete)
Cu augmentare BERT

Configuration	Acuratete maxima train	Acuratete test

SimpleRNN(128,False,tanh)
+BatchNorm
batchsize = 64
lr = 1e-3
	0.8559	0.7007

SimpleRNN(128,False,tanh)
+BatchNorm
batchsize = 8
lr = 1e-3
	0.8299	0.6945

SimpleRNN(128,False,tanh)
+BatchNorm
batchsize = 64
lr = 1e-4
	0.9466	0.7564

SimpleRNN(128,False,tanh)
+BatchNorm
batchsize = 64
lr = 1e-5
	0.9604	0.7492

SimpleRNN(**32**,False,tanh)
+BatchNorm
batchsize = 64
lr = 5e-5
	0.9440	0.7507

SimpleRNN(128,False,tanh) **-**
NO BatchNorm
batchsize = 64
lr = 5e-5
	0.9682	0.7601

SimpleRNN(128,False,tanh)+
SimpleRNN(64,False,tanh)
+BatchNorm
batchsize = 64
lr = 5e-5
	0.9876	0.7549

SimpleRNN(128,False,tanh)+
SimpleRNN(64,False,tanh)
-NO BatchNorm
batchsize = 64
lr = 5e-5
	0.9905	0.7542

SimpleRNN(128,False,tanh)
+BatchNorm
+delete and swap augumentations
batchsize = 64
lr = 5e-5
	0.9493	0.7726

SimpleRNN(128,False,tanh)
+BatchNorm
+BERT context replace
batchsize = 64
lr = 5e-5
	0.9493	**0.7791**

-LSTM(64)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 50
batchsize = 64
lr = 1e-4
sample_size = 0.5
	0.8787	0.7811

-LSTM(64)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.8675	0.7680

-LSTM(256)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.9130	0.7622

-LSTM(64)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 256
lr = 1e-4
sample_size = 0.3
	0.8547	0.7591

-LSTM(64)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-3
sample_size = 0.3
	0.9987	0.7630

-LSTM(64)
+ Dense(1,sigmoid)
-NO BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 2e-4
sample_size = 0.3
	0.9811	0.7614

-Bidirectional(LSTM(64))
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
	0.9773	0.7672

lr = 2e-4
sample_size = 0.3		

-LSTM(64)
+ Dense(64,sigmoid)
+ Dense(32,sigmoid)
+ Dense(16,sigmoid)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.8560	0.7622

-LSTM(64)
+ LSTM(32)
+ Dense(1,sigmoid)
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.9589	0.7698

-LSTM(64)
+ Dense(1,sigmoid)
+delete and swap augumentations
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.9516	0.7546

-LSTM(64)
+ Dense(1,sigmoid)
+BERT context swap
+BatchNorm
MAX_LEN = 30
batchsize = 64
lr = 1e-4
sample_size = 0.3
	0.9596	0.7776

-LSTM(64)
+ Dense(1,sigmoid)
+BERT context swap
+BatchNorm
MAX_LEN = 70
batchsize = 64
lr = 1e-4
sample_size = 1.0
	0.9570	**0.8194**

The best performance was achieved by the models following the context replacement augmentations. The LSTM model, although it requires a larger volume of data and a longer training time, manages to achieve better results and has a more stable increase in training accuracy.