Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

Thor Aleksander Buan

In depth analysis of Long-Short-Term-Memory Neural Networks with the purpose of detecting cyberbullying

Master's thesis in Information Security
Supervisor: Raghavendra Ramachandra
June 2019

NTNU
Norwegian University of
Science and Technology

Thor Aleksander Buan

In depth analysis of Long-Short-Term-Memory Neural Networks with the purpose of detecting cyberbullying

Master’s thesis in Information Security
Supervisor: Raghavendra Ramachandra
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

Norwegian University of
Science and Technology

Preface

This thesis concludes my master grade at NTNU. This thesis was carried out during the spring semester of 2019. The idea of studying cyberbullying detection was brought up by my supervisor Prof. Raghavendra Ramachandra.

This thesis is intended to be read by Cyber Security specialists, Artificial Intelligence specialists or enthusiasts that want to learn about Neural Network designs, in regard to detecting cyberbullying.

01-06-2019

Acknowledgment

I would like to thank my supervisor Prof. Raghavendra Ramachandra for his valuable guidance throughout this project. I will also thank my brother Kim Eide and Frank Nunes for reading my thesis and supplying me with very usable input. Lastly, I will thank my parents for reminding me of the importance of taking brakes.

T.A.B

Abstract

Social media usage has skyrocketed the last decades. Along with all the upsides of social media, there are also a few downsides. One of the most problematic downsides has been the rise of cyberbullying. New statistics shows that this problem has kept on escalating in the recent years. The problem with detecting and preventing cyberbullying is that social media generates so much data that it is virtually impossible to detect cyberbullying by performing manual inspections. This has motivated researchers to come up with new automated methods of detecting cyberbullying. In the most recent years, it has become popular among researchers to utilize a type of artificial Neural Network called Long-Short-Term-Memory Neural Network.

In this thesis we present the empirical evaluation of different types of artificial Neural Network to see how the network can be designed to maximize its performance with regards to detecting cyberbullying. To find the best design of a Long-Short-Term-Memory Neural Network we conduct two experiments, testing 56 different designs. These experiments have been designed as factorial experiments, meaning that a set of variables has been modified in combination with each other. This lets us track any cause and effect relationships between the different variables in a much more accurate way, than what is possible by studying the existing research within the field.

In order to make the results of this study as relevant as possible to the state-of-the-art cyberbullying research, we base the whole Natural Language Processing pipeline that leads up to the artificial Neural Network on the existing state-of-the-art research. We also explain this Natural Language Processing pipeline in depth, such that it can be recreated, and used for continuous study of other variables that are not tested by us, in future research projects.

Four main variables are tested in this thesis; composition of different layer types, stacking of equal layer types, training epochs and activation mechanisms for the classes. Our findings show that the best composition of layer types are found by combining a traditional Convolutional Neural Network with a Long-Short-Term-Memory Neural Network. We also find that all of the Neural Networks, no matter what layer composition, benefit from going from no stacking to stacking its layers twice. As expected, we also find that Neural Networks perform better when properly trained, but there is no correlation between how much training a model needs and how well it performs. Lastly, we prove that there is no reason to blindly choose Softmax as the activation for the classes in the Neural Network models, as most of the researchers tend to do. Our findings show that using an activation mechanism that mimics the mechanisms of a Support-Vector-Machines classifier outperforms the Softmax activation, with all four different layer compositions of the Neural Network that we tested.

Contents

Preface i
Acknowledgment ii
Abstract iii
Contents iv
List of Figures vii
List of Tables xiii
Listings xiv
1 Introduction 1
1.1 Keywords 1
1.2 Problem description 1
1.3 The purpose 1
1.4 Topics covered by the project 2
1.5 Justification, motivation and benefits 2
1.6 Research questions 2
2 Theory 3
2.1 Machine learning 3
2.1.1 Why do we need Machine learning? 3
2.1.2 What is Machine learning? 3
2.2 Natural Language Processing 5
2.2.1 Why we need Natural Language Processing 5
2.2.2 State-of-the-art NLP components 6
2.2.3 Tokenization 6
2.2.4 Stop word removal 6
2.2.5 Removal of repeating characters 7
2.2.6 Stemming 7
2.2.7 Part-of-speech tagging 8
2.2.8 Bag-of-word 8
2.2.9 Global Vectors for Word Representation 8
2.3 Natural Language Processing Classifiers 9
2.3.1 Classifiers used within the field 9
2.3.2 Random Forest classifier 10
2.3.3 Support vector machines 11
2.3.4 Neural Network 12
2.3.5 State-of-the-art Neural Network types 15

2.3.6 Measuring the performance of a classification model . . . 16
2.3.7 Challenges with comparing methods . . . 17
3 Methodology . . . 19
3.1 Experiment . . . 19
3.1.1 The two experiments . . . 19
3.1.2 About the experiments . . . 19
3.1.3 Challenges and counter measures . . . 20
3.1.4 The Natural Language Processing pipeline . . . 21
3.1.5 Experiment setup . . . 21
3.1.6 Experimental environment . . . 22
3.2 Data set . . . 23
3.3 Quantitative assessment . . . 23
3.4 Details about the models to be tested . . . 24
3.4.1 Experiment 1 . . . 24
3.4.2 Experiment 2 . . . 25
3.5 Model selection during training . . . 27
4 Implementation . . . 29
4.1 Third party libraries used . . . 29
4.2 Implementation dependencies and components needed to run the program . . . 29
4.3 Test setup implementation . . . 29
4.3.1 Data set acquisition . . . 29
4.3.2 Preprocessing . . . 30
4.3.3 Classification testing . . . 35
4.3.4 Performance report . . . 40
4.3.5 The alternative activations . . . 40
5 Results . . . 44
5.1 Experiment 1 . . . 44
5.2 Experiment 2 . . . 44
6 Analysis . . . 48
6.1 The effects of increased training . . . 48
6.1.1 The effect of training, analyzed as a function of average epochs used . . . 48
6.1.2 The effect of training, analyzed by training a model for 200 and 800 epochs . 49
6.2 The effects of stacking similar layers . . . 51
6.3 Comparison of the different network types . . . 53
6.4 Comparison of the different activation mechanisms . . . 53
6.4.1 The best model compared with the state of the art models . . . 56
7 Discussion . . . 58
7.1 Challenges . . . 58
7.1.1 Limited python experience . . . 58
7.1.2 Creating a state-of-the-art NLP pipeline . . . 58

7.2 Evaluation . . . 59
8 Conclusion . . . 60
8.1 Future Work . . . 61
Bibliography . . . 62
A Appendix . . . 66
A.1 Data set lables . . . 66
A.2 Proof of faulty oversampling technique . . . 67
A.3 The code for the models in experiment 1 . . . 68
A.3.1 LSTM . . . 68
A.3.2 LSTM x2 . . . 68
A.3.3 LSTM x3 . . . 68
A.3.4 LSTM x4 . . . 69
A.3.5 LSTM x5 . . . 69
A.3.6 BLSTM LSTM . . . 70
A.3.7 BLSTM LSTM x2 . . . 70
A.3.8 BLSTM LSTM x3 . . . 71
A.3.9 BLSTM LSTM x4 . . . 71
A.3.10 BLSTM LSTM x5 . . . 72
A.3.11 BLSTM . . . 72
A.3.12 BLSTM x2 . . . 72
A.3.13 BLSTM x3 . . . 73
A.3.14 BLSTM x4 . . . 73
A.3.15 BLSTM x5 . . . 74
A.3.16 ConvLSTM . . . 74
A.3.17 ConvLSTM x2 . . . 75
A.3.18 ConvLSTM x3 . . . 75
A.3.19 ConvLSTM x4 . . . 76
A.3.20 ConvLSTM x5 . . . 77
A.4 The code for the models in experiment 2 . . . 77
A.4.1 SVM like activation . . . 77
A.4.2 The SVM and Random Forest Classifier used . . . 79
A.5 The code for the testing environment . . . 80
A.5.1 Original script for retrieving the Tweets . . . 80
A.5.2 Modified script for retrieving the Tweets . . . 81
A.5.3 Text preprocessing script . . . 83
A.5.4 Full example of the testing script of one model . . . 84
B Performance Reports Experiment 1, first test run with 200 Epochs . . . 93
C Performance Reports from Experiment 1, second test run with 800 Epochs . . . 124
D Performance reports from experiment 2 . . . 158

List of Figures

1 The training phase of machine learning (illustration based on [1]) 4
2 The testing phase of machine learning (illustration based on [1]) 5
3 A decision tree with two features (F_1 and F_2) and two classes (C_1 and C_2) 10
4 How a SVM separates the classes with a hyperplane 11
5 A four-layered feed-forward Neural Network 12
6 Test setup 22
7 Sketch of the models to be tested in experiment 1 26
8 Sketch of the models to be tested in experiment 2 28
9 First part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1 41
10 Second part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1 42
11 Third and last part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1 42
12 F1 score for the bully class of all models tested in experiment 1, as a function of average epochs used in the training (Y-axis: F1 score for the bully class, X-axis: average epochs used) 49
13 The highest amount of epochs needed in the 10 cross fold run of all models in order to get the best trained model, with maximum training set to 200 epochs and 800 epochs 50
14 The highest amount of epochs needed in the 10 cross fold run of all models in order to get the best trained model, with maximum training set to 800 epochs 51
15 Performance comparison of the models; BLSTM LSTM x2, BLSTM x2, LSTM x3, LSTM x4, LSTM x5, ConvLSTM and ConvLSTM x2 with the training constrained to 200 and 800 epochs 52
16 F1 score for the bully class of all models in the second test run where the models were trained with 800 epochs 53
17 F1 score for the bully class yielded by the standard Neural Network model with Softmax activation, a Neural Network where the Softmax activation layer were replaced with a SVM classifier and a Neural Network where the Softmax activation layer were replaced with a Random Forrest classifier 54
18 F1 score for the bully class yielded by the standard Softmax activation and the SVM alike activation 55

19 F1 score for the bully class yielded by the standard Softmax activation and the SVM alike activation . . . 55
20 Performance comparison between the new ConvLSTM model with its layers stacked three times with SVM alike activation and three models featured in or based on the state-of-the-art research; BLSTM, LSTM and ConvLSTM without any stacking and with Softmax activation . . . 57
21 Proof that the testing data set is leaking information to the training data set, with the oversampling technique used by [2] . . . 67
22 ConvLSTM (200 epochs) performance report (part 1) . . . 94
23 ConvLSTM (200 epochs) performance report (part 2) . . . 94
24 ConvLSTM (200 epochs) performance report (part 3) . . . 95
25 ConvLSTM x2 (200 epochs) performance report (part 1) . . . 95
26 ConvLSTM x2 (200 epochs) performance report (part 2) . . . 96
27 ConvLSTM x2 (200 epochs) performance report (part 3) . . . 96
28 ConvLSTM x3 (200 epochs) performance report (part 1) . . . 97
29 ConvLSTM x3 (200 epochs) performance report (part 2) . . . 98
30 ConvLSTM x3 (200 epochs) performance report (part 3) . . . 98
31 ConvLSTM x4 (200 epochs) performance report (part 1) . . . 99
32 ConvLSTM x4 (200 epochs) performance report (part 2) . . . 100
33 ConvLSTM x4 (200 epochs) performance report (part 3) . . . 100
34 ConvLSTM x5 (200 epochs) performance report (part 1) . . . 101
35 ConvLSTM x5 (200 epochs) performance report (part 2) . . . 102
36 ConvLSTM x5 (200 epochs) performance report (part 3) . . . 102
37 BLSTM LSTM (200 epochs) performance report (part 1) . . . 103
38 BLSTM LSTM (200 epochs) performance report (part 2) . . . 103
39 BLSTM LSTM (200 epochs) performance report (part 3) . . . 104
40 BLSTM LSTM X2 (200 epochs) performance report (part 1) . . . 104
41 BLSTM LSTM X2 (200 epochs) performance report (part 2) . . . 105
42 BLSTM LSTM X2 (200 epochs) performance report (part 3) . . . 106
43 BLSTM LSTM X3 (200 epochs) performance report (part 1) . . . 106
44 BLSTM LSTM X3 (200 epochs) performance report (part 2) . . . 107
45 BLSTM LSTM X3 (200 epochs) performance report (part 3) . . . 107
46 BLSTM LSTM X4 (200 epochs) performance report (part 1) . . . 108
47 BLSTM LSTM X4 (200 epochs) performance report (part 2) . . . 109
48 BLSTM LSTM X4 (200 epochs) performance report (part 3) . . . 109
49 BLSTM LSTM X5 (200 epochs) performance report (part 1) . . . 110
50 BLSTM LSTM X5 (200 epochs) performance report (part 2) . . . 111
51 BLSTM LSTM X5 (200 epochs) performance report (part 3) . . . 111
52 BLSTM (200 epochs) performance report (part 1) . . . 112
53 BLSTM (200 epochs) performance report (part 2) . . . 112

54 BLSTM (200 epochs) performance report (part 3) 113
55 BLSTM X2 (200 epochs) performance report (part 1) 113
56 BLSTM X2 (200 epochs) performance report (part 2) 114
57 BLSTM X2 (200 epochs) performance report (part 3) 114
58 BLSTM X3 (200 epochs) performance report (part 1) 115
59 BLSTM X3 (200 epochs) performance report (part 2) 115
60 BLSTM X3 (200 epochs) performance report (part 3) 116
61 LSTM (200 epochs) performance report (part 1) 116
62 LSTM (200 epochs) performance report (part 2) 117
63 LSTM (200 epochs) performance report (part 3) 117
64 LSTM X2 (200 epochs) performance report (part 1) 118
65 LSTM X2 (200 epochs) performance report (part 2) 118
66 LSTM X2 (200 epochs) performance report (part 3) 119
67 LSTM X3 (200 epochs) performance report (part 1) 119
68 LSTM X3 (200 epochs) performance report (part 2) 120
69 LSTM X3 (200 epochs) performance report (part 3) 120
70 LSTM X4 (200 epochs) performance report (part 1) 121
71 LSTM X4 (200 epochs) performance report (part 2) 121
72 LSTM X4 (200 epochs) performance report (part 3) 122
73 LSTM X5 (200 epochs) performance report (part 1) 122
74 LSTM X5 (200 epochs) performance report (part 2) 123
75 LSTM X5 (200 epochs) performance report (part 3) 123
76 ConvLSTM (800 epochs) performance report (part 1) 125
77 ConvLSTM (800 epochs) performance report (part 2) 125
78 ConvLSTM (800 epochs) performance report (part 3) 126
79 ConvLSTM x2 (800 epochs) performance report (part 1) 126
80 ConvLSTM x2 (800 epochs) performance report (part 2) 127
81 ConvLSTM x2 (800 epochs) performance report (part 3) 127
82 ConvLSTM x3 (800 epochs) performance report (part 1) 128
83 ConvLSTM x3 (800 epochs) performance report (part 2) 129
84 ConvLSTM x3 (800 epochs) performance report (part 3) 129
85 ConvLSTM x4 (800 epochs) performance report (part 1) 130
86 ConvLSTM x4 (800 epochs) performance report (part 2) 131
87 ConvLSTM x4 (800 epochs) performance report (part 3) 131
88 ConvLSTM x5 (800 epochs) performance report (part 1) 132
89 ConvLSTM x5 (800 epochs) performance report (part 2) 133
90 ConvLSTM x5 (800 epochs) performance report (part 3) 133
91 BLSTM LSTM (800 epochs) performance report (part 1) 134
92 BLSTM LSTM (800 epochs) performance report (part 2) 134
93 BLSTM LSTM (800 epochs) performance report (part 3) 135

94 BLSTM LSTM x2 (800 epochs) performance report (part 1) . . . 135
95 BLSTM LSTM x2 (800 epochs) performance report (part 2) . . . 136
96 BLSTM LSTM x2 (800 epochs) performance report (part 3) . . . 136
97 BLSTM LSTM x3 (800 epochs) performance report (part 1) . . . 137
98 BLSTM LSTM x3 (800 epochs) performance report (part 2) . . . 137
99 BLSTM LSTM x3 (800 epochs) performance report (part 3) . . . 138
100 BLSTM LSTM x4 (800 epochs) performance report (part 1) . . . 138
101 BLSTM LSTM x4 (800 epochs) performance report (part 2) . . . 139
102 BLSTM LSTM x4 (800 epochs) performance report (part 3) . . . 139
103 BLSTM LSTM x5 (800 epochs) performance report (part 1) . . . 140
104 BLSTM LSTM x5 (800 epochs) performance report (part 2) . . . 141
105 BLSTM LSTM x5 (800 epochs) performance report (part 3) . . . 141
106 BLSTM (800 epochs) performance report (part 1) . . . 142
107 BLSTM (800 epochs) performance report (part 2) . . . 142
108 BLSTM (800 epochs) performance report (part 3) . . . 143
109 BLSTM x2 (800 epochs) performance report (part 1) . . . 143
110 BLSTM x2 (800 epochs) performance report (part 2) . . . 144
111 BLSTM x2 (800 epochs) performance report (part 3) . . . 144
112 BLSTM x3 (800 epochs) performance report (part 1) . . . 145
113 BLSTM x3 (800 epochs) performance report (part 2) . . . 146
114 BLSTM x3 (800 epochs) performance report (part 3) . . . 146
115 BLSTM x4 (800 epochs) performance report (part 1) . . . 147
116 BLSTM x4 (800 epochs) performance report (part 2) . . . 147
117 BLSTM x4 (800 epochs) performance report (part 3) . . . 148
118 BLSTM x5 (800 epochs) performance report (part 1) . . . 148
119 BLSTM x5 (800 epochs) performance report (part 2) . . . 149
120 BLSTM x5 (800 epochs) performance report (part 3) . . . 149
121 LSTM (800 epochs) performance report (part 1) . . . 150
122 LSTM (800 epochs) performance report (part 2) . . . 150
123 LSTM (800 epochs) performance report (part 3) . . . 151
124 LSTM x2 (800 epochs) performance report (part 1) . . . 151
125 LSTM x2 (800 epochs) performance report (part 2) . . . 152
126 LSTM x2 (800 epochs) performance report (part 3) . . . 152
127 LSTM x3 (800 epochs) performance report (part 1) . . . 153
128 LSTM x3 (800 epochs) performance report (part 2) . . . 153
129 LSTM x3 (800 epochs) performance report (part 3) . . . 154
130 LSTM x4 (800 epochs) performance report (part 1) . . . 154
131 LSTM x4 (800 epochs) performance report (part 2) . . . 155
132 LSTM x4 (800 epochs) performance report (part 3) . . . 155
133 LSTM x5 (800 epochs) performance report (part 1) . . . 156

134 LSTM x5 (800 epochs) performance report (part 2) 157
135 LSTM x5 (800 epochs) performance report (part 3) 157
136 ConvLSTM x2 with SVM classifier (part 1) 159
137 ConvLSTM x2 with SVM classifier (part 2) 160
138 ConvLSTM x2 with RFC classifier (part 1) 161
139 ConvLSTM x2 with RFC classifier (part 2) 161
140 ConvLSTM x2 with Softmax activation (part 1) 162
141 ConvLSTM x2 with Softmax activation (part 2) 163
142 ConvLSTM x2 with Softmax activation (part 3) 163
143 ConvLSTM x2 with SVM like activation (part 1) 164
144 ConvLSTM x2 with SVM like activation (part 2) 165
145 ConvLSTM x2 with SVM like activation (part 3) 165
146 BLSTM LSTM x2 with SVM classifier (part 1) 166
147 BLSTM LSTM x2 with SVM classifier (part 2) 166
148 BLSTM LSTM x2 with RFC classifier (part 1) 167
149 BLSTM LSTM x2 with RFC classifier (part 2) 167
150 BLSTM LSTM x2 with Softmax activation (part 1) 168
151 BLSTM LSTM x2 with Softmax activation (part 2) 168
152 BLSTM LSTM x2 with Softmax activation (part 3) 169
153 BLSTM LSTM x2 with SVM like activation (part 1) 169
154 BLSTM LSTM x2 with SVM like activation (part 2) 170
155 BLSTM LSTM x2 with SVM like activation (part 3) 170
156 BLSTM x2 with SVM classifier (part 1) 171
157 BLSTM x2 with SVM classifier (part 2) 171
158 BLSTM x2 with RFC classifier (part 1) 172
159 BLSTM x2 with RFC classifier (part 2) 172
160 BLSTM x2 with Softmax activation (part 1) 173
161 BLSTM x2 with Softmax activation (part 2) 173
162 BLSTM x2 with Softmax activation (part 3) 174
163 BLSTM x2 with SVM like activation (part 1) 174
164 BLSTM x2 with SVM like activation (part 2) 175
165 BLSTM x2 with SVM like activation (part 3) 175
166 LSTM x4 with SVM classifier (part 1) 176
167 LSTM x4 with SVM classifier (part 2) 176
168 LSTM x4 with RFC classifier (part 1) 177
169 LSTM x4 with RFC classifier (part 2) 177
170 LSTM x4 with Softmax activation (part 1) 178
171 LSTM x4 with Softmax activation (part 2) 178
172 LSTM x4 with Softmax activation (part 3) 179
173 LSTM x4 with SVM like activation (part 1) 179

174 LSTM x4 with SVM like activation (part 2) 180
175 LSTM x4 with SVM like activation (part 3) 180

List of Tables

1 Example that illustrates the bag-of-word model 8
2 Third party libraries used in the implementation 29
3 Vocabulary dict with indexes 31
4 The vectors corresponding to data sample 1, 2 and 3 31
5 Performance results from experiment 1, with 200 epochs 45
6 Performance results from experiment 1, with 800 epochs 46
7 Performance results from the test with BLSTM with added LSTM layers design in experiment 2 47
8 Performance results from the test with BLSTM design in experiment 2 47
9 Performance results from the test with Conv LSTM design in experiment 2 47
10 Performance results from the test with LSTM design in experiment 2 47

Listings

4.1 Code example of how to create the vectors from the text in the preprocessed data set, code was inspired by [3] 31
4.2 The implementation of Glove embeddings, the code was inspired by [3] 31
4.3 Splitting the data into 10 folds 33
4.4 Creating a set of oversampled folds 34
4.5 Saving the data to NPY files 35
4.6 Saving the data to NPY files 35
4.7 Lists used to save and keep track of all the trained models in one training run 36
4.8 The 10 cross fold loop and how the Neural Network is initiated on the beginning of each loop run 37
4.9 The splitting of the data set into training and testing sets 38
4.10 Initiating the training of the model 39
4.11 Manual testing of all the model versions created during training 39
4.12 How the data from the second last layer in the Neural Network is collected 41
4.13 How the best parameters of the Random Forrest classifier was found 43
A.1 The implementation of the LSTM model 68
A.2 The implementation of the LSTM x2 model 68
A.3 The implementation of the LSTM x3 model 68
A.4 The implementation of the LSTM x4 model 69
A.5 The implementation of the LSTM x5 model 69
A.6 The implementation of the BLSTM LSTM model 70
A.7 The implementation of the BLSTM LSTM x2 model 70
A.8 The implementation of the BLSTM LSTM x3 model 71
A.9 The implementation of the BLSTM LSTM x4 model 71
A.10 The implementation of the BLSTM LSTM x5 model 72
A.11 The implementation of the BLSTM model 72
A.12 The implementation of the BLSTM x2 model 73
A.13 The implementation of the BLSTM x3 model 73
A.14 The implementation of the BLSTM x4 model 73
A.15 The implementation of the BLSTM x5 model 74
A.16 The implementation of the ConvLSTM model 74
A.17 The implementation of the ConvLSTM x2 model 75
A.18 The implementation of the ConvLSTM x3 model 75
A.19 The implementation of the ConvLSTM x4 model 76

A.20 The implementation of the ConvLSTM x5 model 77
A.21 The implementation of the LSTM model with SVM like activation 77
A.22 The implementation of the BLSTM LSTM model with SVM like activation 78
A.23 The implementation of the BLSTM model with SVM like activation 78
A.24 The implementation of the ConVLSTM model with SVM like activation 79
A.25 The SVM classifier and Random Forest classifier 79
A.26 Original script made by [4] for retrieving the Tweets 80
A.27 The script used for retrieving the Tweets 81
A.28 The script used for preprocessing the raw text in the tweets 83
A.29 The script used for preprocessing the raw text in the tweets 84

1 Introduction

1.1 Keywords

Neural Network, Natural Language Processing, Cyber bullying, Deep Learning, Text Classification

1.2 Problem description

The usage of social media has skyrocketed in the last few decades. According to Statistics Norway (SSB) the proportions of Norwegians that use social media on a regular basis have now reached 80%[5]. The elephant in the room, however, is that cyberbullying also has become popular along with the rise of social media. According to the Norwegian Media Authority[6] 28% of Norwegian children in the age group nine to eighteen years old have experienced that people have been mean to them or bullied them on social media.

Because of the large amounts of data generated by the high usage of social media, it is virtually impossible to manually assess all the text logs, to capture and stop attempts of bullying online. Even if it was possible, it is not certain it would be desirable, since it would raise questions regarding privacy concerns. Therefore, there has been an increase in research towards finding a method to automatically detect bullying content online. In the most recent years researchers have found Neural Networks to be quite a good tool for this task. The usage of Neural Networks for text analysis purposes is not a new idea as its usage within this field can be traced back to 2003[7]. However, it is only in recent years the method has become truly popular for text classification tasks, and been applied to detecting incidents of cyberbullying. Neural Networks is not just one distinct model, there are several different types of Neural Networks with different designs and architectures. To increase the complexity even more, Neural Networks are often referred to as "black boxes", since we cannot fully understand why they makes the predictions that they do. This makes it a hard task to decide which Neural Network design or architecture to choose. Therefore, to find the best designated Neural Network model for a certain task, it is crucial to test several different designs and architectures, in a laboratory manner, to identify if there is a large performance difference between them.

1.3 The purpose

The objective of this thesis is to look into different architectures for Long-Short-Term-Memory (LSTM) Neural Networks, with the purpose of detecting cyberbullying. The reason for why we focus on the LSTM architecture is because these types of Neural Networks have become very popular within the field of cyberbullying detection in the most recent years. But while most of the new research feature this type of Neural Network, they often feature slightly different designs of the Neural Network type. We therefore want to empirically evaluate these different designs, and see if we can detect any cause and effect relationships that can be used to improve the designs.

The variables that will be tested are the amount of training, type of design (composition of different type of layers), the complexity of the models and the activation leading to the classification. By studying these variables, in a laboratory environment, the goal is to detect relevant cause and effect relationships, in respect to detecting cyberbullying, which can be used to define the Long-Short-Term-Memory Neural Network models of the future.

1.4 Topics covered by the project

This thesis will cover the process of pre-processing text for classification purposes, also known as Natural Language Processing. It will also cover some of the commonly used machine learning algorithms for classifying text, such as Random Forrest classifier and Support Vector Machines, but the main focus will be on Artificial Neural Network classifiers, especially Long-Short-Term-Memory Neural Networks. All of the topics will be covered from the perspective of detecting cyberbullying incidents.

1.5 Justification, motivation and benefits

The motivation for this research project is partly based on the issues surrounding the increased reporting of cyberbullying incidents, and the problems of efficiently detecting such incidents. Furthermore, parts of the motivation also stems from the confusing landscape of researchers utilizing different Neural Network designs for the same purpose. While most researchers compare their Neural Network design with other designs, they seldomly test other designs in their own environment. This makes it difficult to assess if there really is a big difference in the performance among the different Neural Network designs or whether the performance difference is due to other variables.

By creating a standardized state-of-the-art method for pre-processing the textual data for the Neural Network, the hope is to limit other variables that can affect the performance, so the true performance differences between the Neural Networks can be assessed. This will hopefully limit the workload on future research within the cyberbullying field in the search for improved future methods.

1.6 Research questions

The following questions will be researched within this thesis:

1. How does the amount of training impact the performance of the Neural Network?
2. How does the performance of the different types of Long-Short-Term-Memory Neural Networks compare, when tested in a controlled environment?
3. What is the performance effect of blindly increasing a Neural Network's complexity by stacking equal layers on top of each other?
4. Is there a viable alternative to the commonly used Softmax activation layer?

All of these questions will be researched with the goal of detecting cyberbullying.

2 Theory

The purpose of this chapter is to give the reader insight into the theoretical background needed to understand this thesis. This chapter includes brief reviews of important concepts like machine learning and Natural Language Processing, and the algorithms relevant for this thesis that are used for these concepts. The chapter also includes an overview of different state-of-the-art methods or models used within the field of text classification and cyberbullying detection which are found relevant for this thesis.

2.1 Machine learning

2.1.1 Why do we need Machine learning?

Every action made on a device generates data in some form or another. For instance, when visiting a website like Facebook, data is generated in the form of IP logs, browser history and so on. The posts that are published are stored on servers, and the same goes for "likes" or other reactions that are registered. In total all actions and use of the internet amounts to great quantities of data. According to DOMO's sixth report on data[8] 2.5 quintillion bytes of data are generated every day.

Data is generated so that it is possible to keep track of everything going on, but it can also be utilized for several other purposes. Data is for instance essential when doing research. Whether you are researching what strategy a business should use for the next year or researching the correlation between online activity and depression, data is a key factor. Because of the importance of data, it is often quoted to be the new oil.

But since the amounts of data generated in today's society are so gigantic, it is virtually impossible to analyze this data manually. This is where machine learning comes in. Machine learning let's us automate the process of analyzing large quantities of data in a relatively short amount of time, with the aim of detecting patterns.

2.1.2 What is Machine learning?

Machine learning refers to different types of models that are able to train themselves by analyzing the data, and extracting knowledge from the data. Machine learning is a sub-field of artificial intelligence (AI), that has grown exponentially in the last 20 years. The idea is to create models that are able to form their own rules, functions, probability distributions, or other knowledge representations based on the data it is fed, in order to detect patterns[9]. These knowledge representations can then be used by the model to classify new data. For example, by saying that the sentiment of a text is either positive or negative.

Simply put the process of machine learning is to train the model on one set of known data samples, and use the trained model on new unknown data in order to make classifications of this

new data[10]. This process is shown in more detail in figure 1 which gives an example of the training scheme of such a model, and figure 2 which shows the classification scheme of the same model.

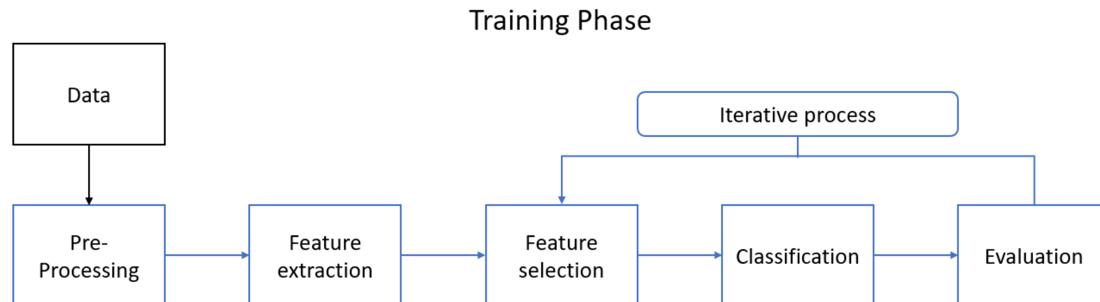


Figure 1: The training phase of machine learning (illustration based on [1])

A typical training scheme in machine learning starts by pre-processing the data set to be utilized. This can be as easy as normalizing the values or it can be more complex tasks of processing text, by for example removing irrelevant parts of the text like punctuations. After the processing the features need to be extracted from the data. This often means to create new features from a set of existing features. For example, if the data exists of height and weight measurements of people, one such feature may be the body-mass-index, which is a product of both height and weight. This step can be done before the data is fed to the actual machine learning model, or the machine learning model can do the feature extraction itself. This depends on the model design and what kind of data that is being analyzed. Then the model attempts to classify the data based on the features that have been extracted and some default knowledge representations. After the data has been classified it evaluates the results. How this is done depends on the model type, whether it belongs to the category of models using unsupervised learning or the category using supervised learning. In this thesis only supervised learning is relevant, since all of the different model types that will be studied belong to this category. In such a case the model evaluates its performance by comparing its own classifications of the data samples, with the true classes that the data samples belong to. It then slightly changes its knowledge representations, or what features it focuses on, before it tries to classify the data again. The effect of the adjustments are then evaluated. If the adjustments made, increases the performance, it may try to adjust again in the same direction to further improve the performance, or if the performance did not improve, it may try some other adjustments. This process is repeated again and again for a set amount of times, or until the performance of the model reaches a preset threshold.

The classification scheme, as seen in figure 2, is very similar to the training scheme. The data goes through the same pre-processing step(s) as used in the training, and the features are extracted in the same manner. But instead of having an iterative process of feature selection, classification and evaluation, as seen in figure 1, the model measures the values of the features chosen to give

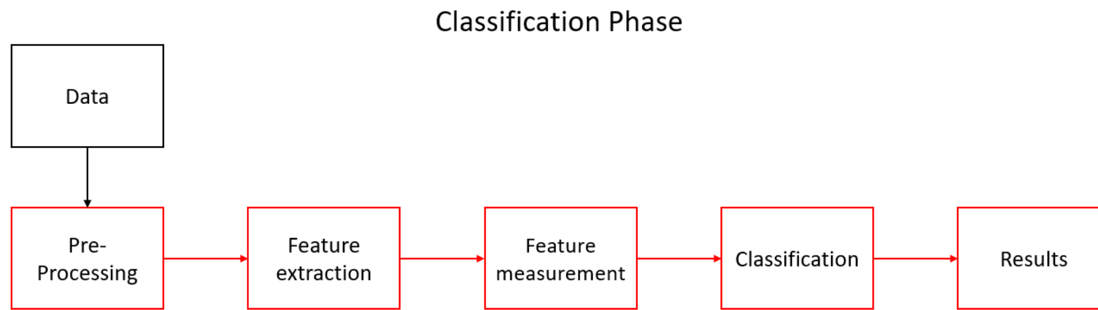


Figure 2: The testing phase of machine learning (illustration based on [1])

the best performance in the training scheme, and the classification is made from the knowledge representations (rules, probability representations etc.) that the model learned during the training. The output of the model contains the classes of all the data samples fed in to the model.

2.2 Natural Language Processing

2.2.1 Why we need Natural Language Processing

Natural Language Processing (NLP) is a sub-field of linguistics and artificial intelligence. Natural Language Processing is the processing of human language data, in order to make the data understandable to computers and their algorithms[11]. In theory one could just feed raw human language data to a machine learning algorithm, but this would lead to poor performance. The reason for this is that a computer cannot correctly process human language data. Computers use the binary system of ones and zeroes, and not the Oxford dictionary. While the two languages obviously consist of a very different vocabulary, ones and zeroes versus words, there are several other challenges as well. The human language is a quite complex language with many different rules that are needed in order to understand the structure. A computer would be able to translate the sentence "The dog bit a man" into ones and zeroes with the help of different tools like for example the ASCII table [12], but it would not be able to understand the meaning of it.

In order to make computers able to understand we therefore use NLP. NLP is not one technique but a set of components. An implemented NLP setup is often referred to as a NLP pipeline. The reason for this is because an implemented NLP system often contains several different NLP components which are run in a certain sequence, just as a pipeline. The NLP components have a big range in complexity. The more complex the NLP components are, the more information can be extracted from the textual data. The only problem is that these NLP components are not bullet proof, and by increased complexity follow an increased rate of errors[13]. In addition, there are also other factors than just the complexity of the NLP components that may increase the error rate. One of the main reasons for increased error rates in NLP tasks are noisy unseen data[13]. These are typically data from sources like social media platforms or chat logs. This is text that is written in a very informal manner, where slang and emoticons are frequently used, and the text is usually not checked for

spelling or grammar mistakes.

2.2.2 State-of-the-art NLP components

There are several different types of NLP components that are featured in state-of-the-art research within the field of cyberbullying detection. The reason for this is because the source of the data sets containing cyberbullying usually are different social media platforms. As already discussed, text from these kinds of sources contains a lot of noise, thus making many of the NLP components more prone to errors. As a result, there has been a few attempts of creating original NLP components, specialized for the task of cyberbullying detection. One of these attempts was done by Nandhini and Sheeba [14]. They proposed a method where the Levenshtein distance between text samples and a dictionary of cyberbullying terms were calculated as part of the NLP pipeline (see [15], to learn more about the Levenshtein distance metric). While the issues of processing noisy text in cyberbullying detection is driving some researchers in the direction of new original NLP solutions, it is still common to use well known NLP components. Some of the most popular components within the field as identified by [16] are;

- Tokenization
- Stop word removal
- Removal of repeating characters
- Grammar and spelling correction
- Stemming

2.2.3 Tokenization

Tokenization is a process where data is split into smaller distinctive pieces. In the case of text processing this usually means splitting a text into separate words. The tokenization process is often thought of as a sub-process within NLP. This is because tokenization is usually used as a mean to apply other NLP tasks. In most cases a text is first tokenized into separate words, before other NLP tasks are applied, then the data pieces are detokenized into a full text again. Detokenization is tokenization in reverse, taking many distinctive pieces and stitching them back together into one piece. Tokenization is claimed to be one of the more reliable NLP components. There are however situations where most tokenization algorithms will struggle as well. This can occur if the text being processed is, for instance, sourced from Twitter where it is popular to use hashtags, or if the text contains mathematical or chemical formulas[17].

2.2.4 Stop word removal

Stop word removal is an example of a NLP process that demands that the text first has been tokenized. Stop word removal is, as the name implies, the process of removing the stop words from the text. A stop word, within the field of NLP, means a useless word[18]. This is usually done by checking each word against a dictionary of such stop words. If the word is present in the dictionary the word is dropped from the data sample. Which words are considered stop words is a bit ambiguous, but it usually are words like "the", "a", "an" , "it" and so on. There are several readymade stop

word dictionaries. NLTK's [19] stop word dictionary is probably one of the most popularly used. The main object of doing stop word removal is to wash away any unnecessary data that does not add any information. By doing so the computational cost of doing the classification will be reduced. The potential problem with this is that this automated process may wash away too much, potentially removing words that do in fact add information.

2.2.5 Removal of repeating characters

Another frequently used text processing task identified by Salawu et al. [16] is the removal of repeating characters. This process, as the name implies, is to removes any characters that are excessively repeated. The need for this process comes from the fact that computers do not have any understanding of human language. Because of this a word that has a character repeated, like "niceee", would be interpreted as a completely different word than the original word, in this case the word "nice". Since the data processed in cyberbullying detection usually is very noisy, it makes a lot of sense to apply a process that reduces this noise, as removing repeated characters. But as mentioned by [16], and shown by [20], this process may also corrupt or remove wanted information as well. The corruption may happen when repeating characters are blindly removed. This may lead to legitimate words being transformed into non-existing words, like the word "call" being transformed to the non-existing word "cal". Maybe worse it can change one legitimate word like "good" into another legitimate word like "god", which potentially can completely change the meaning of a text. Another unwanted effect of this process can be that empathized information in the data is removed. This can happen when a word is emphasized by excessively repeating its characters, like "you are so duuuuuumb", where the "u" in "dumb" is excessively repeated to accentuate how dumb the person in interest is. This could be information that would be relevant for the classification task.

2.2.6 Stemming

As discussed in the section 2.2.5 a computer interprets every word completely differently by default. This is not only a problem when words have misspellings like repeated characters, but it is also a problem when it comes to different versions of a word. This can for example be verbs that are conjugated. For example, the words "run" and "running" have a slightly different meaning, the first is in present tense and the second is in present continuous tense. However, both of the words "run" and "running" are tied to the same activity. For a computer, these words will just be interpreted as different words with no connection. The stemming process is a process that is aimed at solving this problem. The stemming process tries to solve this by defining a core (called stem) of each word, which all the versions of a word in the data set is transformed to. This means that both the word "run" and "running" will be represented by the stem "run".

There exists a range of pre-made stemming algorithms freely available for use. Most of these work by utilizing a dictionary and a set of rules. This means that for a word to be correctly stemmed it has to be in the dictionary used by the stemming algorithm[7]. This is problematic in the case of cyberbullying since the data, as mentioned, usually is quite noisy, with spelling errors and slang, which means that the dictionary approach will be useless in many cases.

2.2.7 Part-of-speech tagging

Part-of-speech (POS) tagging is aimed at extracting some of the structural features from the textual data. The POS tagging aims to tag all the words in a corpus with a structural tag. These tags are usually based on what type of word it is, for example verb, adverb, noun etc. They can also be more specific as marking whether it is plural or singular[21]. A POS tagger algorithm typically consists of a dictionary for word recognition and some kind of a classifier for deciding the tag within the context of the rest of the textual data where the word exists. The classifiers used vary a bit for the different implementations, some of the most popular POS tagging algorithms use Hidden Markow models (see [22], for an explanation of this model type) and rule-based models (see section 2.3.2, for an example of such an model type). POS taggers are reported to have a good accuracy score of around 97%, but this is on the same type of data that they have been pre trained on, which typically are news articles. On more noisy data, the accuracy will be lower[17].

2.2.8 Bag-of-word

The bag-of-word model, also referred to as the feature space model, is a model where the data set is represented by vectors. There is one main vector that holds all the unique words in the data set. The main vector therefore has the length N, where N equals the size of the vocabulary for the whole data set. Each data sample in the data set is also represented by a vector of the same length. The vector that represents the data samples consists of counts of, how many times each word in the main vector, occurs in the data sample. This is illustrated by the example in table 1. Where the data set consist of three samples. Data sample 1 is "Bob shot Fred", data sample 2 is "The dog shot the dog" and lastly data sample 3 is "Fred shot Bob".

Main vector:	Bob	shot	Fred	the	dog
Data sample 1:	1	1	1	0	0
Data sample 2:	0	1	0	2	2
Data sample 3:	1	1	1	0	0

Table 1: Example that illustrates the bag-of-word model

This example illustrates one of the biggest flaws of the bag-of-word model, which is that the structure of the data samples is not represented[23]. This is easily shown by the representation of data sample 1 and 3 in table 1. Even though the sentences have completely different meanings concerning who shot who, they are represented with the same vector. The bag-of-word model is popular despite its flaws. This is mainly due to its simplicity and the fact that many of the flaws can be minimized by combining the bag-of-words model with other NLP methods. For example, by applying POS-tags before using the bag-of-word model one would be able to keep some of the structural elements in the data.

2.2.9 Global Vectors for Word Representation

Global Vectors for Word Representation (Glove) is an algorithm that was developed by [24] in 2014. The algorithm is referred to as a global log-bilinear regression model that combines two

previously popular methods of capturing statistics from a corpus: global matrix factorization and local context window methods [24]. In other words, the Glove model is an unsupervised model which creates word vectors that are connected to a word and that explain their statistical properties within a corpus. This is performed not just by looking at the co-occurrence of words, but also on the ratio between the different co-occurrences. This allows the method to indicate that, for instance, the words "man" and "woman" are related, by their co-occurrences, but that they are also quite different, by the ratio between other co-occurrences, such as that "man" often occurs in combination with "masculine", and "woman" occurs in combination with "feminine". The output of this model is a vector of a predefined size, called the embedding size, that captures these ratios between the co-occurrences. The model can be trained on a specific corpus, or more popularly one of the pre-trained versions available at [25] can be used to create an embedding matrix for the data set to be used.

This model also aims to capture knowledge as with the stemming or POS-tagging process, but one main advantage of Glove is that it does not rely on a dictionary. By training the model on the data set to be classified, or by using one of the pre-trained models that have been trained on data from the same source as the data to be classified. One should not incur the problem that the words in the data set cannot be processed with this method, as a consequence of the word not being in the dictionary. Which as mentioned earlier is one of the main drawbacks with the stemming and POS-tagging processes.

2.3 Natural Language Processing Classifiers

This section will briefly introduce the classifiers most used within the field of cyberbullying detection. Furthermore, the three classifiers most relevant for this thesis will be discussed in more depth. The classifier that will be most thoroughly described is the Neural Network classifier, as this is the main classifier used in this thesis.

2.3.1 Classifiers used within the field

There is a great range of different classifier algorithms that have been applied to natural language processing. Some of the most popular classifiers applied for cyberbullying detection identified by Salawu et al.[16] and for hate detection by Agarwal and Sureka[26] are:

- Support-vector-machines (SVM)
- Naïve Bayes
- K-nearest-neighbours (KNN)
- Rule Based Classifiers
- Clustering
- Exploratory Data Analysis (EDA)
- Link Analysis/Topical Crawler

There have also been several experiments made by combining different classifiers. According to [16], a survey conducted in 2017, Naïve Bayes and SVM were detected as the best performing clas-

sifier algorithms. While this survey is a great resource for identifying the state-of-the-art algorithms applied for detecting cyberbullying incidents, the development within the field is continuous, and in the latest two-three years another method named Neural Network has become popular within the field of cyberbullying detection.

The use of Neural Networks for text classification purposes is actually a quite old idea. One of the first implementations of this method was presented by Bengio et al. [7] back in 2003, but it took some years before Neural Networks became popular. Today Neural Networks are considered to be one of the state-of-art methods within text classification overall, but it is just recently that it has found its way into the field of cyberbullying detection.

2.3.2 Random Forest classifier

A Random Forest classifier (RFC) is one of many rule-based classifiers. A RFC is in essence an ensemble of a decision tree classifier, hence the name forest. A decision tree is a classifier that consists of internal nodes, which are the features, edges that correspond to a subset of the feature value and lastly leaves (also called terminal nodes) that correspond to the classes[9]. A simple example of such a decision tree can be seen in figure 3. For a data sample to be classified as C_2 in this decision tree, it must match the decision rule $(F_1 < V_1) \wedge (F_2 = V_2)$.

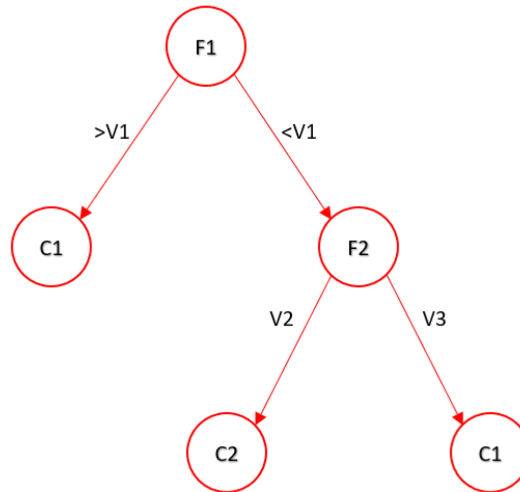


Figure 3: A decision tree with two features (F_1 and F_2) and two classes (C_1 and C_2)

When training a decision tree, one of the key tasks is to decide which feature that is the best to use. To do this metrics like information gain, Gini-index or ReliefF is often calculated for all the features. Then the best features are selected and the edges are generated[9]. This process is a bit different with an RFC. With an RFC a set of different decision trees are created, and the best features are determined from a random selection of some of the features. In the end all of the results are averaged[27].

The Random Forest classifier is quite popular to use since it is so easy to implement. Because it combines several decision trees and makes random selections, it is also quite robust against overfitting. The model has its disadvantages, however. One of them is that it can become quite slow and resource heavy. This is because it may need to build many decision trees to improve the accuracy[27].

2.3.3 Support vector machines

SVMs are one of the most successful machine learning algorithms throughout history, so it is therefore not a surprise that it is also one of the most popular methods used for detecting cyberbullying. The basic principle is to place a hyperplane in the space of attributes that separates the classes optimally. The data points closest to the hyperplane are called the support vectors, and the distance between the support vectors and the hyperplane is called the margins. The optimal hyperplane position is found by maximizing the margins[9]. In figure 4 the hyperplane (the solid red line) and how it is used to separate the two classes of data points (red stars and red circles) by maximizing its margin to the support vectors (dotted red lines) are illustrated.

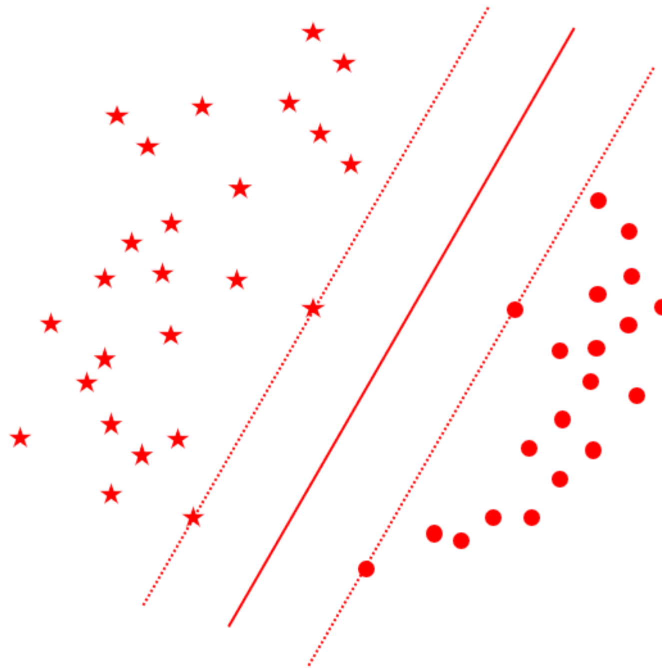


Figure 4: How a SVM separates the classes with a hyperplane

2.3.4 Neural Network

Neural Network models are models which aim to imitate how the brain works. It consists of several layers and each layer has a set of neurons in it. The amount of layers and the count of neurons they hold varies from each Neural Network design. At the very least a Neural Network needs two layers, an input and an output layer, but there is no upper limit. Each neuron can be viewed as a processing unit. A neuron takes one or more inputs, applies weights to them and uses a activation function in order to sum the input values and an output function to produce its output. Each layer of neurons is connected sequentially to each other from the input layer to the output layer, as pictured in figure 5. The number of neurons in the first layer is equal to the number of features that will be used. The number of neurons in the output layer decides how many classes the Neural Network is able to detect. The layers between the input and output layer are called the hidden layers. These can have any number of neurons in them, and there can be none or several hidden layers. The hidden layers are what adds the complexity to the Neural Network. A Neural Network with just an input and output layer with linear activation functions will only be able solve cases with linearly separable data[9]. In order to solve more complex classification cases hidden layers or nonlinear activation functions must be added. The more hidden layers that are added, the more complexity is added. While more complexity may increase the accuracy of the model, it does also add a cost in form of resources needed to train the Neural Network and/or the time needed for training.

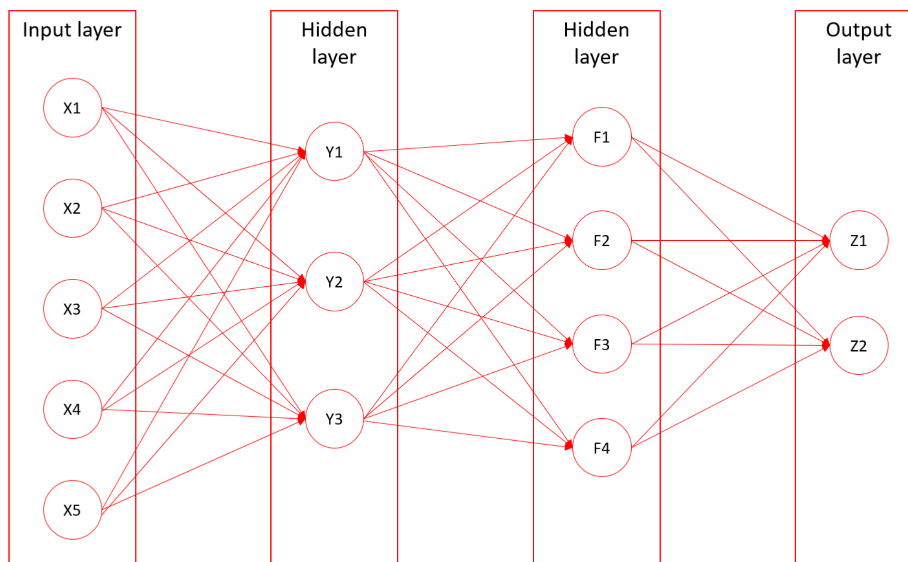


Figure 5: A four-layered feed-forward Neural Network

Training

The training of a Neural Network is an iterative process, consisting of running the Neural Network for a set number of runs, called epochs. The training process consists of changing the weights associ-

ated with each connection between the neurons. There are several different learning rules that can be implemented to a Neural Network. The four most popular ones are the Hebbian learning rule, Delta learning rule, competitive learning rule and forgetting. For this thesis only the generalized Delta learning rule is relevant. The generalized delta learning rule is more commonly known under the name backpropagation[9]. This is the learning rule that is most frequently used in the ready-made deep learning libraries like Keras [28] which is to be used in this thesis. With this learning rule the weights are set randomly when the Neural Network is initialized. For each sample in the training data the output of the model is computed and compared with the expected output. As the name backpropagation implies, the model is corrected in the opposite way of how the calculations are done. First the weights for the output layer are corrected with the difference (also referred to as the error) between the expected and actual output of the model. Then the difference between the expected and the actual values of the second last layer is computed and used to correct the weights for that layer. This process is repeated all the way back to the input layer. The rate at which the weights are updated is called the learning rate. The learning rate decides how much each weight is to be adjusted each time[9].

Activation functions and output functions

As mentioned, each neuron is a processing unit that uses an activation function to calculate the sum of its inputs, and a output function to calculate its output.

The activation function calculates the sum of the neurons input values. There are different kinds of activation functions but the most frequently used is a standard linear activation. With linear activation the input values of a neuron X_j is, the state of the neuron it is connected to (X_i), multiplied with the weight ($W_i j$) that the neuron has associated with that input neuron, and a constant (C_j) that the neuron has (usually referred to as bias or threshold). This is shown in formula 2.1[9].

$$A(X_j) = \sum_i W_i j X_i + C_j \tag{2.1}$$

The output function is a function that takes the value of the activation function and produces a normalized value based on a threshold. There are several different output functions that are used in the different designs of Neural Networks. In this thesis we will use Keras implementation of the functions; linear, Sigmoid, Rectified Linear Unit and Softmax. Too add some confusion, these functions are referred to as activation functions in the Keras API. This is however not accurate as Keras uses the linear activation function as shown in formula 2.1 for the activation of the neurons.

The Sigmoid function is a function that is able to map arbitrary real values into a real value interval[9]. There are a few different versions of this function, the function to be used in this thesis is the Sigmoid function as defined in formula 2.2, which maps the values to the real value in the interval (1,0).

$$Output(x_i) = \frac{1}{1 + e^{-x_i}} \tag{2.2}$$

The Rectified Linear Unit (ReLu) is a function that will output zero if x is not greater than zero. This means that if x is zero or negative the output will be zero, for any other case the output will be equal to the value of x. The Rectified Linear Unit formula is displayed in formula 2.3[29].

$$Output(x_i) = max(0, x_i) \tag{2.3}$$

The Softmax functions formula can be seen in formula 2.4 and 2.5 [30]. The Softmax function will always output a number between zero and one, and the sum off all of the neurons output values will be equal to one. The Softmax function has over time become the go to function for deep learning and is almost always recommended by default to be used in the output layer of a Neural Network. This can be seen by looking at the papers, [31], [32], [33] and [2], mentioned in section 2.3.5 about the state-of-the-art Neural Networks used for cyberbullying detection, where all of the four papers uses Softmax activation in the output layer.

$$P(y = j|\Theta^{(i)}) = \frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}} \tag{2.4}$$

$$\Theta = W_0X_0 + W_1X_1 + ... + W_kX_k = \sum_{i=0}^{k} W_iX_i = W^TX \tag{2.5}$$

Different types of Neural Networks

There are several different types of Neural Networks. Two examples are convolutional Neural Network and Long-short-term-memory networks. These are networks that are designed in a special way. A convolutional network usually consists of a set of hidden convolutional layers and pooling layers. While the name may sound totally different from ordinary hidden Neural Network layer, what differentiate them is simply the use of special activation functions.

In this thesis there are six main types of layers that will be used; Dense layer, Embedding layer, one dimensional convolutional layer, Max pooling layer, Long-short-term-memory layer and Bidirectional Long-short-term-memory layer.

The dense layer is an ordinary Neural Network layer that uses the linear activation function mentioned earlier.

The embedding layer is a special type of input layer that is heavily used with textual data. The embedding layer can have several different purposes, but in this thesis, it will be used to apply a pre-trained Glove embedding to the input data of the Neural Network.

A one dimensional convolutional layer is a layer that takes two different signals and combine them into one. One of the signals is the input signal that comes from the previous layer in the nerual network, while the other signal comes from the kernel. Mathematically this is a combination of two vectors as shown in formula 2.6 [34]. Where the input vector is defined as f with length n and the kernel vector is defined as g with length m. The output of this operation will create a modified signal (or vector). The idea is that through training of the Neural Network the modified signal will enhance the relevant features of the input signal [34].

$$(f * g)(i) = \sum_{j=1}^{m} g(j) * f(i - j + m/2) \tag{2.6}$$

A convolutional layer is usually followed up with a max pooling layer. The max pooling layer is a layer that down-samples the input data. There are two different types of pooling layers, max pooling and min pooling. With max pooling only the maximum values from each selected region of the input data are chosen, while the rest of the values are removed. Min pooling is the complete opposite, where the minimum values are selected, and the maximum values are removed[35].

A Long-Short-Term-Memory layer is a type of Recurrent layer. A recurrent layer is special in the sense that it does not only have an input and an output at each neuron, but also a loop. The loop helps the neuron to not immediately "forget" its previous outputs since they are looped back to the neuron. This makes it possible for the neurons to analyze its input based on both the current input, but also the previous input. Mathematically this is described in formula 2.7 [36]. The state of the hidden recurrent layer at time t is h_t. h_t equals a function of, the weights W, current states input X_t added to the previous state of the hidden recurrent layer at time t-1 h_{t-1}, which is multiplied with it own hidden-state-to-hidden-state matrix U[36]. While this mechanism will let a layer take its past values into consideration when deciding on the value for the current input, it will still struggle to detect long dependencies going several states back. This is where Long-short-term-memory (LSTM) layers come in. These layers have internal mechanisms referred to as gates and cell state. The cell states keep information from previous states the neuron has encountered. The gates let the neurons decide if the information seen should be added to the state or removed from the state. This means that the LSTM layer is able to both remember previous states for a longer time, in addition to forgetting information that does not seem relevant.

$$h_t = \mu(WX_t + Uh_{t-1} \tag{2.7}$$

A bidirectional LSTM is a LSTM layer that works in two directions instead of just one. As previously described, a traditional LSTM layer takes previous states into consideration when calculating its current state. A bidirectional LSTM takes this mechanism a step further by not only looking at the data in one direction, but also the opposite direction. In practice these are just two LSTM layers combined in one, one that processes the input data in the original sequence, and one that processes the input data in a reverse sequence. This will allow the layer not only to detect dependencies with previously processed input data, but also future input data.

2.3.5 State-of-the-art Neural Network types

The most significant new trend for Neural Networks when it comes to cyberbullying detection is the Long-Short-Term-Memory (LSTM) Neural Networks. The LSTM design has been applied in several of the new research studies published most recently. [31] showed that by using several distinctive LSTM classifiers in combination with features based on the users' behavior, one could achieve performance beating the previous state-of-the-art methods. While not using an original LSTM design, [32] showed that by combining traditional Convolutional Neural Network (CNN) layers with Gated

Recurrent Unit (GRU) layers one could also beat the state-of-the-art methods performance. The GRU layers are quite similar to the LSTM layers in the sense that they have the same goal of tracking long term dependencies, but the GRU layers have fewer trainable parameters. LSTM and GRU have been proven to yield very similar performance[37]. LSTM based Neural Networks were also proposed by [33], which displayed that LSTM based networks where able to outperform a state-of-the-art CNN and SVM methods. They later did further research to the LSTM architecture adding a Bidirectional LSTM (BLSTM) based Neural Network, the results, published in [2], showed that the BLSTM model was able to achieve even better performance than the traditional LSTM model.

2.3.6 Measuring the performance of a classification model

The performance of a classification model is based on how many classes the model successfully classifies as correct. In order to assess this there are a few different measurements that are used. Some of the most popular ones are; classification accuracy, precision, recall and F-score.

In order to describe these measurements, we first need to define some classification measurements that are used to calculate them. In a classification scheme with two possible classes, one of the classes are often referred to as the positive class, and the other is referred to as the negative class. If the goal is to detect cyberbullying for example the positive class would be the data classified as bullying while the negative class would be the data that are not classified as bullying. The results of a classification test is then divided into four categories, True positives (TP), False positives (FP), True negatives (TN) and False negatives (FN). The samples that are counted as True positives are the samples that are from the positive class, and that the model classifies as positive. The samples that are regarded as false positives are samples that really belong to the negative class, but the model falsely classifies as positive. The true negatives are the samples that really are negative, and that the model classified as negative. The false negatives are the samples that actually are positive, but the model falsely classifies as negative.

Classification accuracy (hereby referred to as just accuracy) is defined as the frequency of correct classifications[9]. The formula for accuracy can be seen in formula 2.8. Accuracy is the product of correctly classified samples (n_{corr}) divided by the total number of samples (n), and multiplied with 100 in order to get the answer in percent. A accuracy score of 100%, means that all of the samples classified by the model were correctly classified, and a accuracy score of 0% means that no samples were classified as correct.

$$Accuracy = \frac{n_{corr}}{n} * 100 = \frac{TP + TN}{TP + FP + TN + FN} * 100 \tag{2.8}$$

Recall (also known as true positive rate or sensitivity) is defined as the relative frequency of correctly classified positive samples[9]. The mathematical formula for calculating the recall can be seen in formula 2.9. Recall is the product of dividing the number of correctly classified positive samples (TP), with all of samples that belong to the positive class ($TP + FN$)).

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

Precision is defined as the portion of correctly classified samples that were classified as positive[9]. The mathematical formula for calculating the precision score is presented in formula 2.10. The precision score is calculated by dividing the number of samples that were correctly classified as positive by the model (TP), with the total number of samples that were classified as positive by the model ($TP + FP$).

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

F1-score (also known as F-measure) is the harmonic mean of precision and recall. The mathematical formula for calculating the F1-score is presented in formula 2.11. In order to get the harmonic mean of recall and precision one have to take 2 multiplied with the ratio between the product of precision and recall, and the sum of the precision and recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \tag{2.11}$$

The problem with the accuracy is that for most of the classification tasks the class representation in the data sets are skewed. Meaning that maybe 90% of the data samples belong to the negative class, which means that only 10% of the samples are from the positive class. For a model to achieve a respectable accuracy score of 90% it can then just classify all the data samples as negative. This is a problem as it usually is the positive class, we are interested in identifying. Therefore, in order to better understand the model's ability to classify the positive class, the F1-score is often used when comparing the performance of models. The problem with the F1-score, however, is that it does not take into account the possible huge number of negative samples that are correctly classified[9]. In order to account for this, it is usual to calculate the F1-score of both the classes separately, meaning that both classes will be set as positive and negative ones. It is also natural in such a case to calculate the average F1-score. In such a case there are two different averages that are popular to use; macro and weighted. Macro average is the normal average between the two F1-scores scores. Weighted average, however, takes the sample count into account. This means that each F1-score is weighted by their support in the data set. If a class only is represented by 10% of the classes, the F1-score of this class only will count 10% towards the average score.

2.3.7 Challenges with comparing methods

There are several challenges with comparing Neural Network architectures presented in different research articles. Since a NLP pipeline consist of so many different variables, as the components used, how they are implemented and what data set that is used, it is nearly impossible to define any clear cause and effect relationships. Because of this it might end up invalid by directly comparing the performance achieved by the researched methods mentioned in section 2.3.1, in regard to determining what Neural Network architecture that has the best performance. These challenges are also mentioned by the survey of state-of-the-art methods for cyberbullying detection [16]. Since the goal of this thesis is to detect such cause and effect relationships in order to determine the best performing Neural Network classifier architecture, it will be required to test the different architectures

in a lab environment with as few changing variables as possible.

3 Methodology

In this chapter the methodology of this project will be described. This includes description of the experiment setup, testing scheme and type of measurements used.

3.1 Experiment

In order to answer the research questions, two experiments have been conducted. The first experiment was conducted in order to answer the research question concerning what type of Long-Short-Term-Memory Neural Network yields the best results (research question 2), what effect increased training has (research question 1), and if it is beneficial to stack equal layers upon each other (research question 3). The second experiment was conducted in order to investigate whether or not the traditional Softmax activation layer in Long-Short-Term-Memory Neural Networks could be replaced(research question 4). This chapter will cover the details about both experiments.

3.1.1 The two experiments

Experiment:

1. Test the difference between different types of LSTM Neural Networks, and the effect of stacking similar layers. All models were tested in two test runs, first test run was with 200 epochs of training, while in the second test run the models were trained for 800 epochs.
2. Test if Softmax activation can be replaced by either a SVM classifier, a Random forest classifier or hybrid solution consisting of a Neural Network with a SVM like activation.

3.1.2 About the experiments

Both of the experiments were conducted with the OFAT approach. The OFAT (one-factor-at-a-time) is, as the name implies, an experiment strategy which consists of only testing one factor at a time[38]. This is one of the most popular experiment designs[38]. The main idea with this strategy is to create a common baseline for the experiment. Then the variables which are to be experimented with are changed one at a time. One of the big advantages with the OFAT strategy is that "one can readily assess the factor effects as the experiment progresses, because only a single factor is being studied at any stage"[39].

In order to be able to apply the OFAT strategy to the experiments being conducted in this project a standardized Natural Language Processing pipeline has been developed. This pipeline pre-processes the data set and produces feature vectors for every data sample. The details of this pipeline can be assessed in the section 3.1.4. A standard layout for the Neural Networks has also been established. In experiment 1 all of the Neural Networks have the same initial and final layer structures, only the intermediate layers where changed. While in experiment 2 a candidate from all four of the Neural Network types tested in experiment 1 were used, and only the activation

mechanism of these Neural Networks was changed. The details of the models to be tested in both experiments is described in section 3.4.

3.1.3 Challenges and counter measures

There are several challenges present when doing these kinds of experiments. One big challenge is that the training process of a Neural Network is not one hundred percent consistent. Even though the Neural Network and the data fed into it is kept one hundred percent the same, the Neural Network will yield slightly different performance measurements each time. In order to cope with this issue there are different types of strategies that can be applied, like k-fold-cross validation or leave-one-out-cross validation. As discussed by [40] the leave-one-out-cross validation will be preferable in order to minimize the prediction error. The leave-one-out-cross validation method is however extremely computationally heavy. In this method the training is done on one minus all N observations in the data set, and repeated for N times, in such way that all the data samples are left out of the training one time individually and used for testing. Running the training of the Neural Network N times, where in our case N = 4368 (see section 3.2 for detailed information on the data set used), would take too much time. Therefore, the k-fold-cross validation has been chosen instead. According to [40] the K-fold-cross validation is not perfect and will produce some bias, the bias is however smaller the larger the K is. Therefore, a K value of 10 has been selected for this experiment. This will reduce the bias to a minimum while still making it possible to do the testing with our given resources, which is described in section 3.1.6.

While the OFAT strategy is good for assessing single factor effects, the design also has its disadvantages. One of the biggest disadvantages is that it will not be able to discover relational effects between different factors. In order to deal with this problem, one should conduct a factorial experiment. Factorial experiment design is based on varying different factors at the same time, and testing every combination[38]. For example, if there is a case with two factors that can have two states, the experiments will require a 2^2 factorial experiment, which will equal 4 runs in total. The problem of course with this design is that the complexity of the experiment increases exponentially with the amount of factors and states. Because of this experiment 1 will not utilize a factorial design that includes the factors of the NLP pipeline. Instead the experiment will only focus on the following three factors; type of LSTM network, amount of stacked layers and the amount of epochs used in training. There are four LSTM types to be tested, the layers will be stacked up to five times and the training will be done with 200 and 800 epochs. This experiment will therefore require a $2 * 4^5$ (The set of epochs, times, the types of LSTM designs, to the power of, the number of stacked layers) factorial design, which will involve the experiment to be run for a total of 40 times

In experiment 2 the factorial design has also been utilized to some degree. Both the type of LSTM design and activation will be varied. In total 4 LSTM designs will be tested, and 4 activation techniques will be tested. This means that the experiment is 4^4 factorial experiment, which will require 16 runs.

3.1.4 The Natural Language Processing pipeline

Background

There is no de facto standard for the Natural Language Processing pipelines used within the research community. The NLP pipeline design can have a great impact on the performance of the classification. The NLP pipeline designs are therefore often experimented with by researchers, and as a consequence of this almost all the published research within this field will have smaller or bigger changes to NLP pipeline design. The development of the implemented standard NLP pipeline in this project has therefore been based on several tests with different NLP pipeline implementations inspired by published research within the field of cyberbullying detection. The implemented NLP pipeline consists only of components that are commonly used within the field. This is done to make the experiment as relevant as possible.

Components of the NLP pipeline

The implemented NLP pipeline consists of a quite basic text pre-processor. The implemented text pre-processor removes different kinds of punctuation, numbers, names mentioned by the @-symbol and repeating characters. In addition, all the URLs in the texts are replaced by the generic text "URL". Next step of the NLP is to transform the texts into vectors. To do this, the Keras Tokenizer function is used. This implementation creates a dictionary based on the unique words found in the data set. All the words are then given an index in the range of zero to the maximum number of unique words identified. Each text data sample is then represented by a vector of word indices. The vectors are also padded at the end so that they all end up having the same length.

Last step of the NLP is to create an embedding matrix. In order to create the embedding matrix for the data set the Glove[41] algorithm developed by Stanford University is utilized. The implementation in this project uses the Glove algorithm that is pre-trained on a corpus consisting of 27 billion tokens collected from Twitter, with the embedding size equal to 200.

The all in depth details of the implemented NLP pipeline and the code used can be found in chapter 4.

3.1.5 Experiment setup

Both experiment 1 and experiment 2 follow the same experiment scheme. The scheme is illustrated in figure 6. The testing scheme is divided into three main components; Data set acquisition, Preprocessing and Classification testing. The data set acquisition and preprocessing components are only run once when starting the experiment. The classification testing component is an iterative process, that is repeated for every model to be tested.

The data set acquisition component has the task of acquiring the complete data set to be used, this is explained more thoroughly in section 3.2. The preprocessing component preprocesses the textual data, transforms the textual data into vectors and creates the embedding matrix. This component is explained in more detail in section 3.1.4. The classification component is as already stated an iterative process that is repeated once for every model to be tested. The model to be tested is trained and tested in a 10 cross fold manner, meaning that each model is trained and tested ten

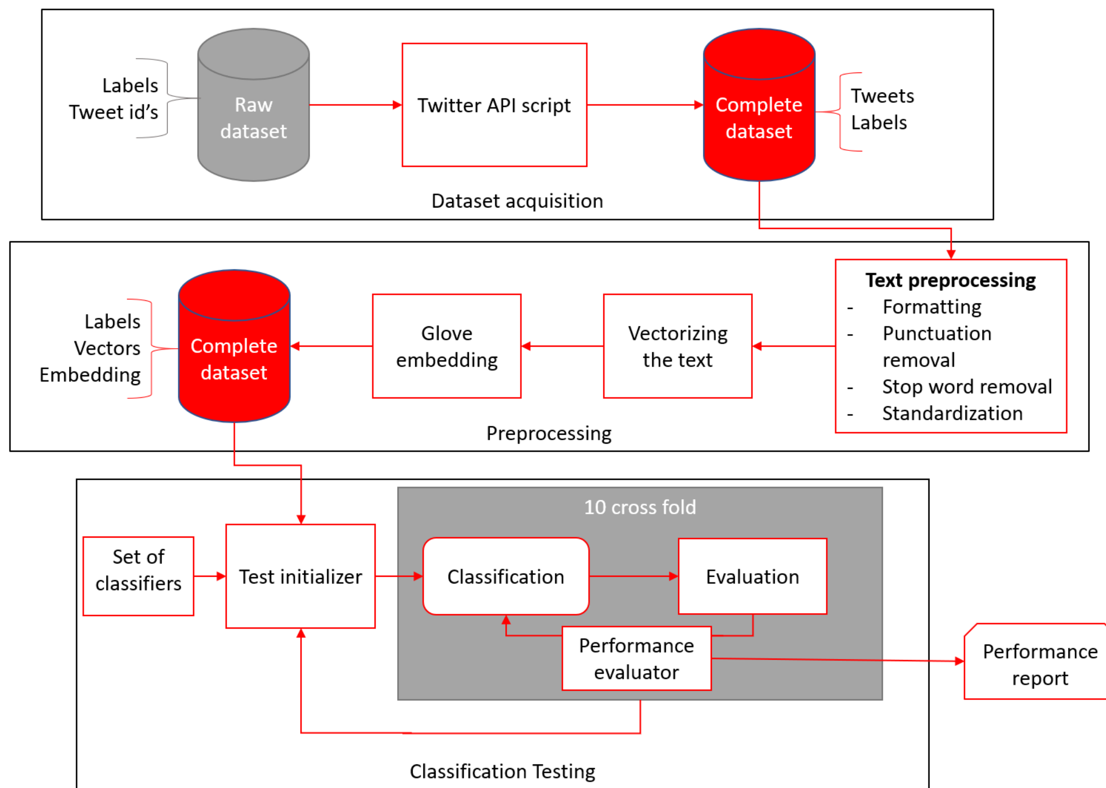


Figure 6: Test setup

times. In every epoch the weights from the model is stored, and at the end of the training run the script called performance evaluator loads all of the model weights to see which model performs the best. The performance measure from the best model from the training run is then stored. This process is repeated for all of the ten folds, and at the end, the performance evaluator script calculates the average performance measures for the 10 runs and create a performance report for the model tested. The performance measures that are used along with more information about the performance reports can be seen in section 3.3.

3.1.6 Experimental environment

Two different machines have been used to run the experiments. Machine 1 was used to run experiment 1 with 200 epochs and experiment 2. While machine 2 was used to run experiment 1 with 800 epochs.

Machines:

1. Machine:
    - Processor: Intel Core i7-8700K

    - Graphics Processing Unit: Nvidia Geforce 1070TI 8 GB
    - Random Access Memory: 32 GB
    - Owner: Thor Buan

2. Machine:

    - Processor:Intel Xeon
    - Graphics processing unit: Nvidia Quadro P5000 16 GB
    - Random Access Memory: 32 GB
    - Owner: Prof. Raghavendra Ramachandra

The script used to run the experiment also require approximately 18 GB of free disk space available to be run, as the weights for the Neural Network model has to be saved during training.

3.2 Data set

The data set used in this project is the bullying traces data set [4] created by Professor Xiaojin Zhu and the University of Wisconsin-Madison. The version 3.0 of this data set which is used in this thesis consist of 7321 tweet ID's and labels for these tweets. There are in total six different labels associated with every tweet ID. All of the labels and their possible values are listed in section A.1. For this project the only relevant label is the bullying label, which is yes or no, that indicates if the data sample is bullying or not.

The data set originally consists of 7321 tweets represented by their ID and the labels associated with them. The tweets in the data set were collected back in 2011. The tweets in their original form were retrieved by creating a script that sends request to the Twitter API ([42]) to retrieve the tweets belonging to the tweet ID's in the data set. Due to the nature of Twitter several of the tweets from the data set could not be retrieved. This can be because the tweets had been deleted by the users themselves or that Twitter had removed the tweets. Because of this the data set that was retrieved in February 2019 through the Twitter API was only counting 4367, whereas 1140 samples are labeled as bullying. Since Neural Network models favor balanced data sets, the data set has been over sampled with samples from the bullying class to even out the label representation to a fifty-fifty distribution. The oversampling technique used is described in section 4.3.2.

3.3 Quantitative assessment

In order to assess the performance of each candidate in the two experiments a set of predefined measurements are collected from the training and testing runs. These performance measurements among other relevant data from the model is saved in a report, these reports can be assessed in chapter B, C and D. The following performance measures are collected from each model test:

- Accuracy
- F1 score
    - Macro average
    - Weighted average

  - Bully class
  - None class

- Precision score
  - Macro average
  - Weighted average
  - Bully class
  - None class

- Recall score
  - Macro average
  - Weighted average
  - Bully class
  - None class

All of the performance measures is averaged over the 10 cross fold run. In addition to these measures the epoch number for the best model found from each training run in each fold is stored. On the basis of these numbers, the best performing model can easily be detected, but also the epochs needed to achieve such a score with the respective models will also be easily found. This should give enough information to decide which models performs the best, and detect any differences in required training time for the models.

3.4 Details about the models to be tested

In this section details about each model to be tested in both of the experiments are described.

3.4.1 Experiment 1

As mentioned earlier to minimize the variables all the models have a standard layout which they are based on. In experiment 1 this standard layout consists of:

- Initial layer
  - Type: Embedding layer
  - Input dim: 11478 (vocabulary size)
  - Input length: 34 (length of vectors)
  - Embedding used: Glove embedding matrix
  - Embedding size: 200

- Second last layer
  - Type: Regular densely-connected Neural Network layer
  - Size: 11478 (vocabulary size)
  - Activation: Rectified Linear Unit

- Last layer (activation layer)

  - Type: Regular densely-connected Neural Network layer
  - Size: 2 (number of classes)
  - Activation: Softmax

- Loss function: categorical cross-entropy
- Optimizer function: Adam
- Metric to evaluate performance during training: Accuracy
- Number of epochs: 200 and 800 (both tested individually)

The only variable(s) that are changed in this experiment are the intermediate layer(s) between the initial layer and the second last layer, as illustrated in figure 7.

There are four types of LSTM based Neural Networks to be tested; ordinary LSTM (hereby referred to as "LSTM network"), Bidirectional LSTM (hereby referred to as "BLSTM network"), Convolutional LSTM (hereby referred to as "ConvLSTM network") and BLSTM mixed with ordinary LSTM (hereby referred to as "BLSTM with added LSTM network" or just "BLSTM LSTM"). The ordinary LSTM network and the BLSTM network will be designed by simply stacking ordinary LSTM or BLSTM layers of the size 200. The BLSTM with added LSTM network is a bit different, as this network design starts with a BLSTM layer, but are stacked consequently with ordinary LSTM layers. The Conv LSTM design is also a bit different. One Conv LSTM component consists of a Conv1D layer, a MaxPooling1D layer and a LSTM layer. In the Conv LSTM design the Conv LSTM components are stacked. The raw code of all the models in experiment 1 can be found in the Appendix section A.3.

3.4.2 Experiment 2

The standard layout of the model in experiment 2:

- Initial layer
  - Type: Embedding layer
  - Input dim: 11478 (vocabulary size)
  - Input length: 34 (length of vectors)
  - Embedding used: Glove embedding matrix
  - Embedding size: 200
- Intermediate layers, one of the following designs
  - LSTM stacked four times
  - BLSTM stacked twice
  - BLSTM with two added LSTM layers
  - ConvLSTM stacked Twice
- Second last layer
  - Type: Regular densely-connected Neural Network layer
  - Size: 11478 (vocabulary size)

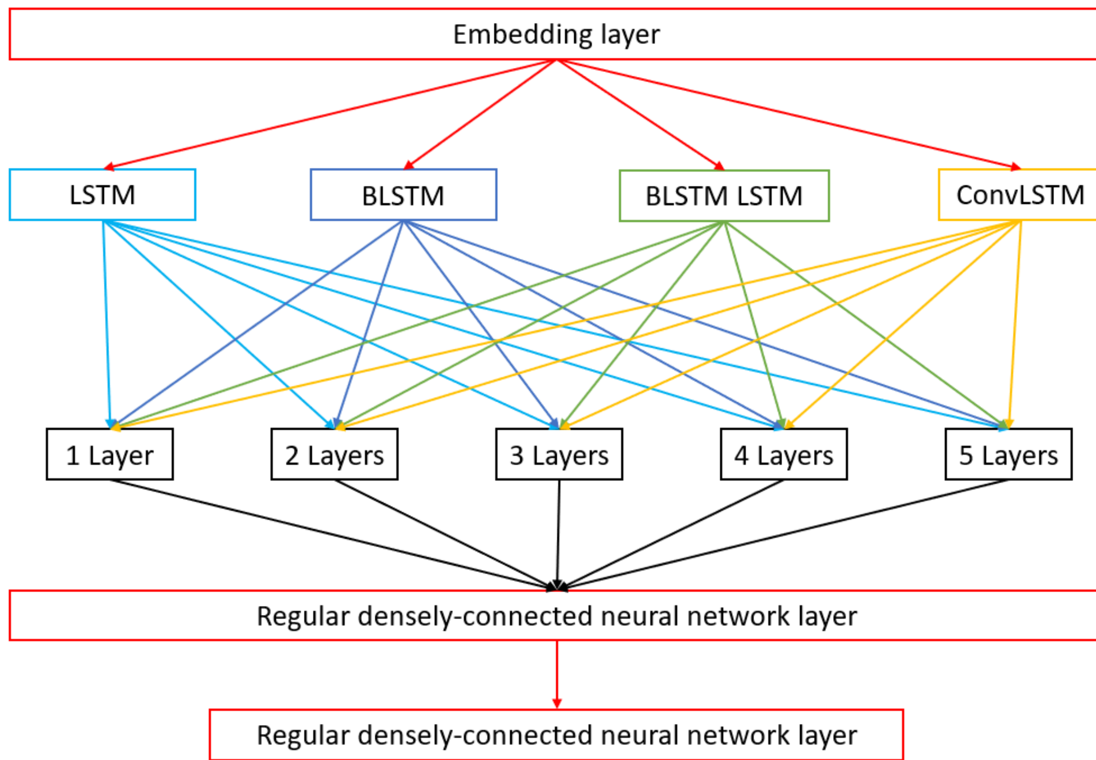


Figure 7: Sketch of the models to be tested in experiment 1

  - Activation: Rectified Linear Unit
- Optimizer function: Adam
- Metric to evaluate performance during training: Accuracy
- Number of epochs: 800

The main variable to be tested in this experiment is the activation mechanism. In total four different activation algorithms will be tested:

- Softmax activation
- An SVM like activation
- Classical SVM classifier
- Classical RFC classifier

Softmax has become almost a de facto standard as the activation function to use in Neural Network within text classification. In this experiment some quite unorthodox activations will be tested to see if they can compete with the Softmax activation function. One of these is a hybrid approach between Neural Networks and Support Vector Machines, hereby referred to as SVM like activation. The SVM like activation has three major differences from the typical Softmax activated Neural Network. In the SVM like activation the loss function used with the Neural Network has been changed to hinge loss, from cross entropy loss, the activation function of the last layer is changed from Softmax to Linear, and lastly a L2 kernel regularizer is applied to the last layer. A similar approach, just with GRU layers instead of LSTM layers, was proposed by [43] in order to combine the powers of a SVM classifier with a Neural Network classifier for the purpose of intrusion detection.

In addition to the Softmax and SVM activation functions, a traditional SVM classifier and a traditional Random Forest classifier was tested as activation for the classes.

3.5 Model selection during training

All of the models to be tested are based on the Neural Network implementation from Keras[28]. While this is a really comprehensive and popular deep learning implementation it does have its limitations. One of these is the function for selecting the best performing model from the training of the Neural Network. It is only possible to use accuracy and loss as measurements for deciding the best model. Since f1 measure is the preferred way of choosing the best performing model, the model selection is done manually. This is done by saving the weights of the model from each epoch in the training stage. After the training, all of the model weights are loaded, and the models are tested individually. The model with the highest f1 score for the bully class is selected, as bully detection is the main goal for these models. This procedure has a cost in sense of increased time and resources spent on the training, but since F1 score is the only truly valid technique of measuring performance, the procedure will ensure the best trained version of each model is selected.

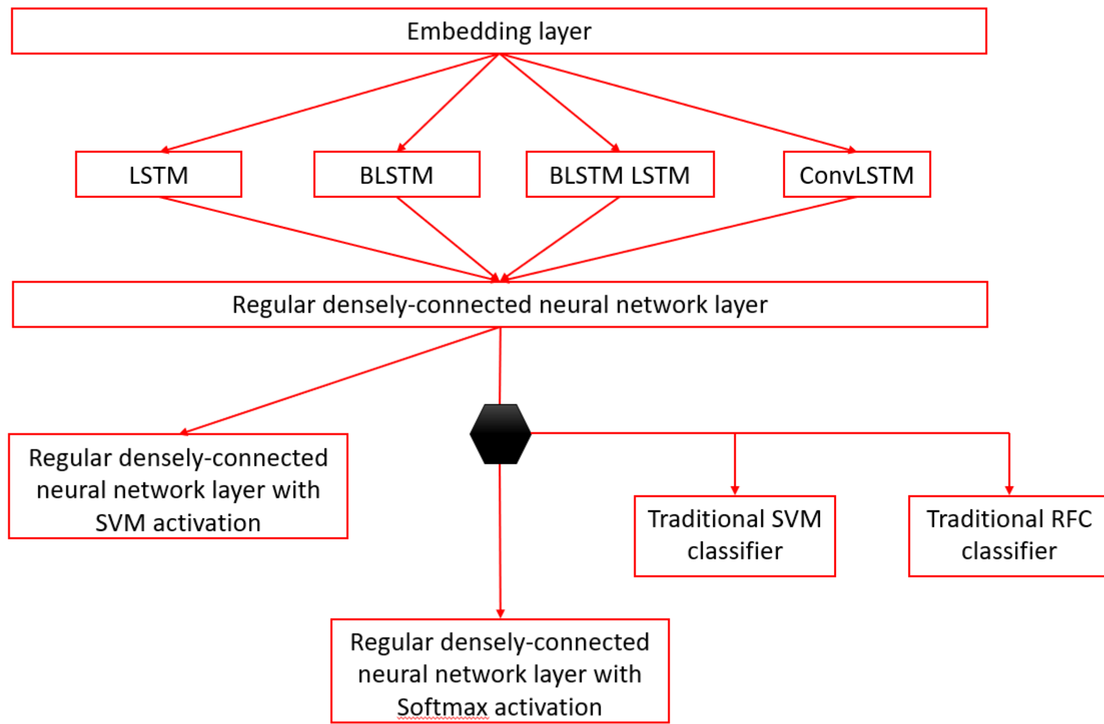


Figure 8: Sketch of the models to be tested in experiment 2

4 Implementation

This chapter contains an overview of the third party libraries used in the implementation and other dependencies, and describes how the test program has been implemented.

4.1 Third party libraries used

Library	Source	Used for
Tweepy	[44]	Data acquisition
Numpy	[45]	Data set handling
Pandas	[46]	Data set handling
NLTK	[19]	Preprocessing
Preprocessor	[47]	Tweet preprocessing
Keras	[28]	Preprocessing and Neural Network classification
TensorFlow	[48]	Back-end for the Neural Network classification
SKlearn	[49]	Performance measurements and, RFC and SVM classifier

Table 2: Third party libraries used in the implementation

4.2 Implementation dependencies and components needed to run the program

Python version 3 [50] is required to run the program. For the development the Jupyter notebook [51] and Jupyterlab [52] were utilized. All of these programs can easily be obtained by installing the Anaconda environment package [53], which was used to develop this program. In addition to this the pre-trained embedding model from Glove [25] and the data set called Bullyingtraces[4] are required.

4.3 Test setup implementation

In this section the implementation of each core of the testing program is described. This section is divided into the same core parts as found in figure 6; Data set acquisition, Preprocessing and Classification testing.

4.3.1 Data set acquisition

The raw data set which already has been described in section 3.2, only consist of Tweet IDs, User IDs and five different labels; bullying trace, type, form, teasing, author role and emotion. For this thesis however only the bullying traces label was used. This label has one of two values, "y" meaning it is indeed a bullying trace or "n" meaning it is not a bullying trace. Full overview of all of the labels and their possible values are listed in the Appendix section A.1.

In order to collect the raw Tweets associated with the Tweet IDs the Tweepy [44] library was utilized. The data set downloaded from [4] already comes with an script for retrieving the Tweets with the use of Tweepy through the Twitter API. This script however had to be slightly modified in order to get the data into CSV format instead of JSON objects and to remove all the data samples where it was not possible to retrieve the Tweets. The original script made by [4] can be seen in the appendix section A.5.1, and the modified script that has been used in this thesis can be seen in the appendix section A.5.2.

4.3.2 Preprocessing

After the data set has been fully acquitted the pre processing can begin. This process is a three step process. First step is called the text preprocessing. In this step the raw text was processed in order to maximize the classification performance by removing some of the noise in the textual data. The following text processing tasks where implemented:

- Remove numbers
- Replace any URLs with the text "URL"
- Remove any characters that are repeated more than two times
- Remove the names in the @ mentions, but keeps the @ sign
- Remove all of the following signs:

```
'!"%&\()$*+,-./:;<=>?[\\]^_`{|}~'
```

- Transform all the characters to lowercase

The code of all of these operations can be assessed in the Appendix section A.5.3. As discussed in section 2.2.5, removing repeating characters can corrupt the data. To avoid this, the implemented algorithm that removes repeating characters in this thesis, only removes any characters that are repeated more than two times. Thus avoiding any corruption of words that has double letters. The drawback with such an implementation is that words that have mistakenly been spelled with double letters, will not be corrected.

Next step in the preprocessing is to transform the text data into vectors. This was done by utilizing the Keras library [28], as shown in listing 4.1. First the function `fit_on_text` is called. This function creates an internal vocabulary dict based on the textual data passed to it. Next all of the data samples are transformed to vectors with the use of the function `text_to_sequences`. This function creates vectors based on the internal vocabulary dict previously generated. For each data sample, all of the word is switched with their corresponding index in the internal vocabulary dict. Finally the function `pad_sequences` is called on the data set holding the vectors. This function pads all of the vectors so that they become the same length as the longest data sample. The finished product is actually quite similar to that of the bag-of-word model previously discussed in section 2.2.8. The difference is that the length of all of the vectors is not equal to the vocabulary size, but instead the length of the longest data sample. In addition, the sequence of the words in the data samples are not removed as it is with the bag-of-word model, instead of identifying the words in

Words:	Bob	shot	Fred	the	dog
Indexes:	1	2	3	4	5

Table 3: Vocabulary dict with indexes

Data sample 1	1	2	3	0	0
Data sample 2	4	5	2	4	5
Data sample 3	3	2	1	0	0

Table 4: The vectors corresponding to data sample 1, 2 and 3

the sample with their placement in the vector, the words are identified with their indexes in the internal vocabulary dict. To better understand this it can be useful to continue the exampled used with the bag-of-word model in section 2.2.8. Given the data set with three data samples. Where data sample 1 is "bob shot fred", data sample 2 is "The dog shot the dog" and data sample 3 "fred shot bob". In this case we will have an internal vocabulary dict as shown in table 3, and the data samples will be represented with the vectors shown in table 4.

Listing 4.1: Code example of how to create the vectors from the text in the preprocessed data set, code was inspired by [3]

```
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
vocabulary_size = None
tokenizer = Tokenizer(num_words= vocabulary_size)
#num_words=None means that all the indexes will be mapped
tokenizer.fit_on_texts(dataset[:,0])
#dataset[:,0] is the text column in the data set
sequences = tokenizer.texts_to_sequences(dataset[:,0])
#Pads the vectors such that they all get the same length:
dataTokenized = pad_sequences(sequences,
maxlen=inputsize_)
```

Last step of the preprocessing is to generate the embedding matrix. As already mentioned, the pre-trained Glove embedding has been used in this project. Therefore, in order to generate the embedding matrix for the data set, all that is needed to do is to load the pre-trained embeddings that are downloaded from Glove [25]. Create an empty matrix with the size equal to the length of the internal vocabulary dict times the embedding size (in this thesis the embedding size used was 200). Then compare the words from the Glove embedding with the words in the internal vocabulary dict, and on a match of words, fill inn the embeddings from the Glove embedding into our embedding matrix. Such that we get the embeddings made by the Glove model for all of the words in our data set. The code for this operation is shown in listing 4.2

Listing 4.2: The implementation of Glove embeddings, the code was inspired by [3]

```
embeddings_index = dict()
```

```
f = open(GloveFile , encoding="utf8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocabulary_size, embsize_))
for word, index in tokenizer.word_index.items():
    if index > vocabulary_size - 1:#out of scope
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            #embedding for word is found
            embedding_matrix[index] = embedding_vector
```

Preparing the data for the classification scheme

After all of the data has been preprocessed it has to be prepared for the classification scheme. This involve three different tasks; split the data into 10 folds (or parts) for the 10 cross fold validation, balance the data set by oversampling, and saving the new data set.

Firstly the data set containing the vectors and labels are divided into 10 folds of equal size, this procedure can be seen in listing 4.3. These folds are going to be used in a 10 cross fold scheme, as discussed in section 3.1.3. This means that each fold, at some time, is going to be used as the testing data for the classification, thus it is crucial to not leak any information from one fold to another. Because of this it is very important that the data set is split into the folds before any oversampling is done. Information leakage between the folds happens when a data sample is copied from one fold to another.

After this each fold is oversampled individually to reach close to an equal representation of both classes, this procedure can be seen in listing 4.4. This procedure creates a new set of 10 folds. This is done in order to keep both a oversampled version of the data set, and a version of the data set that is not oversampled. The oversampled data set will be used for training the model, while the non-oversampled data set will be used for the performance testing of the model. One important notice to make about this is that the two sets has the exact same indexing. The data in fold one of the not oversampled data is copied over to fold one of the oversampled data set before the fold then is oversampled. This means that fold one in the not oversampled data is equal to fold one of the oversampled data. Then the oversampling is done individually on each fold in the oversampled

data set. The oversampling is done by collecting all the class one (bullying) data samples from the fold. Then the population size of class one data samples is divided on the population size of the class zero (not bullying) data samples, in order to get the oversample rate needed to get close to a fifty-fifty representation of the classes in the fold. The class one data samples is then duplicated in accordance with the oversample rate.

Last task is to save the all the data needed to run the classification. This is done in order to ensure that all the models in both experiment 1 and 2 are trained and tested on the exact same data. This procedure is shown in listing 4.5. This produces 22 files of the type `.npy` in a folder called `test_data`. Each fold from both the oversampled data set and the non-oversampled data set are stored in individual files. The ten oversampled folds have the file names `TraindataFolds_1.npy` to `TraindataFolds_10.npy`. The ten non oversampled folds have the file names `DataFold_1.npy` to `DataFold_10.npy`. In addition to this the embedding matrix is saved in its own file called `embedding_matrix.npy`, and other variables that the classification is dependent on are saved in a file called `Settings.npy`.

Listing 4.3: Splitting the data into 10 folds

```
#10-Cross folding
#datasetTokenized = data set with the vectors and the labels

#First fold:
foldsize = int(np.round(len(datasetTokenized)*(0.1)))
datasetTokenizedF1=datasetTokenized[:foldsize,:]

#Seccond Fold
Foldstart = foldsize
Foldeend = Foldstart+foldsize
datasetTokenizedF2=datasetTokenized[Foldstart:Foldeend,:]

#Third to tenth fold
Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF3=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF4=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF5=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF6=datasetTokenized[Foldstart:Foldeend,:]
```

```
Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF7=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF8=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
Foldeend = Foldstart+foldsize
datasetTokenizedF9=datasetTokenized[Foldstart:Foldeend,:]

Foldstart = Foldeend
datasetTokenizedF10=datasetTokenized[Foldstart:,:]


#List with all the data set folds
DataFolds=[datasetTokenizedF1,datasetTokenizedF2,
datasetTokenizedF3,datasetTokenizedF4,datasetTokenizedF5,
datasetTokenizedF6,datasetTokenizedF7,datasetTokenizedF8,
datasetTokenizedF9,datasetTokenizedF10]
```

Listing 4.4: Creating a set of oversampled folds

```
#New array for holding the oversampled folds:
TraindataFolds = DataFolds.copy()

#Loop through each fold individually:
for index, fold in enumerate(DataFolds):
    tempdataoversampled=fold
    class1datapoints= np.array([])
    neutralclasscount=0
    #Find all data samples of class 1:
    for datapoints in fold:
        if datapoints[ClassCollumn] == 1:#Class 1:
            if class1datapoints.size == 0:
                class1datapoints=datapoints
            else:
                class1datapoints = np.vstack(
                [class1datapoints,datapoints])
    #Calculate class 0 population:
    neutralclasscount = int(np.round(len(fold)))
    -len(class1datapoints)
    #Calculate the difference in the population size
    #of class 0 and 1:
    oversamplerateclass1 = int(np.round(neutralclasscount/
```

```
    len(class1datapoints)))
    #oversample class 1:
    oversampledclass1 = class1datapoints
    for i in range(oversamplerateclass1-2):
        oversampledclass1= np.vstack(
        [oversampledclass1,class1datapoints])

    tempdataoversampled = np.vstack([tempdataoversampled,
    oversampledclass1])
    #Add the fold to the list of oversampled folds:
    TraindataFolds[index] = tempdataoversampled
```

Listing 4.5: Saving the data to NPY files

```
#Save all individual folds from the original data set:
for ind, fold in enumerate(DataFolds):
    filename= "test_data\DataFold_"+str(ind+1)+".npy"
    np.save(filename, fold)

#Save all individual folds from the oversampled data set:
for ind, fold in enumerate(TraindataFolds):
    filename= "test_data\TraindataFolds_"+str(ind+1)+".npy"
    np.save(filename, fold)

#Save the embedding matrix
filename= "test_data\embedding_matrix.npy"
np.save(filename, embedding_matrix)

#Save other important variables needed
Settings = [EpochCount,embsize_,inputsize_,
batch_size_,ClassCollumn]
filename= "test_data\Settings.npy"
np.save(filename, Settings)
```

4.3.3 Classification testing

Load the data

Before the classification testing can start, all of the data is loaded from the files created in listing 4.5. This is a very simple procedure that can be seen in listing 4.6. As seen in this listing one variable called `vocabulary_size.npy` is not loaded but instead set manually, this is due to a mistake made when the data was stored. This variable was forgotten when the data was saved, and therefore it had to be set manually.

Listing 4.6: Saving the data to NPY files

```
#Load previously generated classification data
from numpy import genfromtxt
```

```
TraindataFolds=[[],[],[],[],[],[],[],[],[],[]]
DataFolds=[[],[],[],[],[],[],[],[],[],[]]
for i in range(10):
    TraindataFolds[i] =
    np.load("test_data\TraindataFolds_"+str(i+1)+".npy")
    DataFolds[i] =
    np.load("test_data\DataFold_"+str(i+1)+".npy")
Settings=np.load("test_data\Settings.npy")
embedding_matrix = np.load("test_data\embedding_matrix.npy")
EpochCount=int(Settings[0])
embsize_=int(Settings[1])
inputsize_=int(Settings[2])
batch_size_=int(Settings[3])
ClassCollumn=int(Settings[4])
vocabulary_size=11478
```

About the classification testing script

The implementation of the classification script can be divided into three main parts; the preparation part, the 10 cross fold loop, and the performance report generation. A full example of the whole script can be seen in the Appendix section A.5.4.

Classification testing script: The preparation part

There are two main tasks performed in this part of the script, importing all the resources needed from third party libraries and preparing a model list. As described in section 3.5, selecting the best performing trained version of the model is done manually instead of the automatic selection algorithm implemented in Keras.

Because of this it is needed to save each version of the model from all of the epochs during the training. The classification script therefore starts with creating a list of model names, where the names are on the form `saved-model-*Epochnumber*.hdf5`. Where `*Epochnumber*` is a three digit number from `001` to the amount of epochs that the model is to be trained (200 or 800). In addition, an empty list called `bestmodels` is initiated. This list will be used to save the name of the best performing model from each training run in the 10-cross-fold scheme. This will allow us to keep track of the epochs needed to get the best model from each run. This whole procedure is shown in listing 4.7.

Listing 4.7: Lists used to save and keep track of all the trained models in one training run

```
part1="saved-model-"
part3="-.hdf5"
ModelsTemp=[]
#EpochCount=200 or 800
for i,x in enumerate(range(EpochCount)):
    i=i+1
    if i<10:
        part2="00"+str(i)
```

```
    if i<100 and i>9:
        part2="0"+str(i)
    if i>99:
        part2=str(i)
    modelname=part1+part2+part3
    ModelsTemp.append(modelname)
bestmodels=[]
```

Classification testing script: 10 cross fold loop

The backbone of the testing script is a "for loop" that goes through each fold in the data set. During each loop the following tasks are done:

- Defining and resetting the Neural Network model
- Splitting the data into a training and testing set
- Training the model
- Manual testing of all the trained versions of the model
- Extensive performance evaluation of the best version identified of the model

Defining and resetting the Neural Network model

For each loop of the 10 cross fold run the model is reset and redefined. This is done in order to ensure that each training run is done entirely from scratch. An example of the code used to create the model is shown in listing 4.8. This example is from the testing of the Convolutional LSTM with no layer stacking from experiment 1. One important part of the model initiation to take note of is that the model is created with a checkpoint function (last line in the listing). This is done in order to save the weights of the model from each epoch of the training, so that the weights later can be loaded to test the model performance from each epoch manually. The `filepath` variable is used to give the file names to the weights saved. The value of this variable is set to `saved-model-{epoch:03d}-.hdf5` which will result in the weights files being given the same names as stored in the list of model names that was described earlier.

Listing 4.8: The 10 cross fold loop and how the Neural Network is initiated on the beginning of each loop run

```
#Loop through all the folds
for index, fold in enumerate(DataFolds):
    try:
        modelNTNU.reset_states()#Try to reset the model,
        #will throw an error on first run, therefore the try
    except:
        n=0#Do nothing, it's just because its the first run
    finally:
        #Create the model
        K.clear_session()
        modelNTNU = Sequential()
        modelNTNU.add(Embedding(vocabulary_size, embsize_,
        input_length=inputsize_, embeddings_initializer=
```

```
Constant(embedding_matrix), trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size,
activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])

filepath = "saved-model-{epoch:03d}-.hdf5"
mcp = ModelCheckpoint(filepath, monitor='val_acc',
verbose=0,save_weights_only=True,
save_best_only=False, mode='max',period=1)
```

Splitting the data into a training and testing set

As already mentioned, the testing script uses a "for loop" to loop through the folds in the data set. The fold selected by the "for loop" is therefore used as the testing fold for each run. The training set is thereby defined as the remaining folds in the data set. The code for defining the training and testing data of each loop can be seen in listing 4.9. One thing to note is that the oversampled data fold is selected as testing set. This is because this is the testing set to be used as validation by Keras. The testing set is later reset to the non-oversampled set before the manual performance testing, in order to get as accurate performance measures as possible.

Listing 4.9: The splitting of the data set into training and testing sets

```
TestSet=TraindataFolds[index]#Testing fold
TrainSet= np.array([])#Empty array
for index2, fold in enumerate(DataFolds):
    if index2 != index:
        #Not testing set so include in training:
        if TrainSet.size == 0:#First fold
            TrainSet=TraindataFolds[index2]
        else:#the rest of the folds
            TrainSet = np.vstack([TrainSet,
            TraindataFolds[index2]])
#Prepare the classes to be fed into keras:
Trainclasses = to_categorical(np.array(
TrainSet[:,ClassCollumn]))
Testclasses = to_categorical(np.array(
TestSet[:,ClassCollumn]))
```

Training the model

The initiation of the training is done by a function call as showed in listing 4.10. While this is just one call, this call will also take a long time to process, as this is where the training of the model over 200 or 800 epochs take place. By using the callback function `mcp`, that was previously defined in listing 4.8, all of the model's weights from each of the epochs are also saved.

Listing 4.10: Initiating the training of the model

```
modelNTNU.fit(TrainSet[:,:inputsize_], Trainclasses,
epochs=EpochCount, batch_size=batch_size_,
validation_data=(TestSet[:,:inputsize_],
Testclasses), shuffle=False, callbacks=[mcp],
verbose=0)
```

Manual testing of all the trained versions of the model

After the model has been trained all of the trained versions of the model is manually tested. This is done by utilizing another "for loop" that loops through the list containing the file name of all the saved weights. For each loop the selected file is loaded, and the weights are applied to the model. The model is then set to predict the classes of the testing data samples. For each loop the F1 score of the bully class (class 1) predictions are calculated. Based on the F1 score the best performing model is selected. This manual testing loop can be assessed in listing 4.11.

Listing 4.11: Manual testing of all the model versions created during training

```
#Reset measurements
bestf1=-1
bestmodel=""
#Go through all the models from all the epochs
for modelname in Models[1]:
    modelNTNU.load_weights(modelname)#Load the weights
    #Use it to classify the test samples:
    predict = modelNTNU.predict_classes(
    TestSet[:,:inputsize_])
    #calculate the F1 score:
    f1result=f1_score(TestClassesList,predict,
    average=None)#calculate F1-measure
    #F1 is higher than highest F1 registered?:
    if f1result[1]>bestf1:
        bestf1=f1result[1]#Save results
        bestmodel=modelname#Save model
#Add file name of the best model to the list
bestmodels.append(bestmodel)
```

Extensive performance evaluation

The best version of the model that was identified is then loaded again in order to calculate more detailed performance measurements. The performance measurements calculated for each model is

as earlier mentioned in section 3.3:

- Accuracy
- F1 score
  - Macro average
  - Weighted average
  - Bully class
  - None class
- Precision score
  - Macro average
  - Weighted average
  - Bully class
  - None class
- Recall score
  - Macro average
  - Weighted average
  - Bully class
  - None class

These scores are stored in temporal lists, that is used to generate the performance report.

4.3.4 Performance report

After the 10 cross fold loop is done, all of the performance measurements of the best model from each loop are averaged, creating the overall performance score of the model averaged over the ten cross fold run. Then the script creates a performance report of the model. The performance report contains information on the model overview, performance scores and the file name of the best model weights from each epoch. The reason for storing the file names of the best model weights are because these names contain the epoch number they are from. Thus, making it possible to also collect information on the amount of training needed for each model design. The performance reports are stored as text files. An example of the performance report can be seen in figure 9, 10 and 11. The performance reports generated for all of the models tested in this thesis can be assessed in chapter B,C and D. While most of the content in the performance reports are generated automatically by the script, the average epochs used by the model to reach peak performance is calculated and added to the reports manually.

4.3.5 The alternative activations

For experiment 2 the aim was to test a set of four different activations. Two of these where a bit special and needed some custom coding in order to work, these where the implementation of the separate SVM classifier and Random Forest classifier as activation mechanisms for the Neural Network. This was done in practice by training the Neural Network with Softmax activation as seen

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               365600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,554,692
Trainable params: 3,259,092
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 9: First part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1

implemented in section 4.3.3, but with a minor change. First the model was trained as ordinary and the best performing trained version of the model was selected. Then the best performing model was used to classify the test data one extra time, but this time the data from the second last layer of the Neural Network model was collected and put into a new data set. The code used to achieve this can be seen in listing 4.12.

Listing 4.12: How the data from the second last layer in the Neural Network is collected

```
layer_name="dense_1"#The name of the seccond last layer

#Define a new model that only exist of the layers up to the
#second last layer:
intermediate_layer_model = Model(inputs=modelNTNU.input,
outputs=modelNTNU.get_layer(layer_name).output)

#Use this new model to predict the test data and
#collect the output
SVM_folds[index] = intermediate_layer_model.predict(
TestSet[:,:inputsize_])

#combine the collected data and the labels in order to get a
#complete data set:
```

```
Model Performance (10-Cross-Fold):
Accuracy average:                                     0.85734
F1 score average for None-class:                      0.89882
F1 score average for Bully-class:                     0.75636
F1 score weighted average:                            0.86171
F1 score macro average:                               0.82759
Precision average for None-class:                     0.94074
Precision average for Bully-class:                    0.68833
Precision weighted average:                           0.87516
Precision score macro average:                        0.81453
Recall average for None-class:                        0.86181
Recall average for Bully-class:                       0.84543
Recall weighted average:                              0.85734
Recall score macro average:                           0.85362
=================================================================
```

Figure 10: Second part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1

```
=================================================================
The best model in fold 1: saved-model-013-.hdf5
The best model in fold 2: saved-model-022-.hdf5
The best model in fold 3: saved-model-018-.hdf5
The best model in fold 4: saved-model-076-.hdf5
The best model in fold 5: saved-model-015-.hdf5
The best model in fold 6: saved-model-017-.hdf5
The best model in fold 7: saved-model-021-.hdf5
The best model in fold 8: saved-model-158-.hdf5
The best model in fold 9: saved-model-029-.hdf5
The best model in fold 10: saved-model-033-.hdf5
Avg epochs: 40,2
```

Figure 11: Third and last part of the performance report for the Convolutional LSTM model with one layer stacking from experiment 1

```
SVM_folds_complete[index] = np.column_stack((
SVM_folds[index].astype(np.object),TestSet[:,ClassCollumn]))
```

After the full 10 cross fold run the data set is complete, but instead of a data set containing vectors and labels, the data set now contain the output from the second last layer in the Neural Network and the labels. This data set was then used in another 10 cross fold set up where the SVM classifier and the Random forest classifier were fitted to the data and their performance were measured. The script used for this is identical to that of the Neural Network classifier described in section 4.3.3, however, the model used was switched with SVM and Random Forest classifiers. The SVM and Random forest classifiers were implemented by using the SKLearn library. The full code of the SVM classifier and Random forest classifier can be assessed in the Appendix section A.4.2. The same SVM classifier and Random Forrest classifier were used in combination with all of the Neural Networks. In order to find the best parameters for the SVM classifier and Random Forrest classifier the `GridSearchCV` model from SKLearn was used. This model is able to do an exhaustive search over a preset set of parameters for a model, in order to detect what combination of parameters that yield the highest score. An example of how this model was used to find the best possible parameters for the Random Forrest classifier can be seen in listing 4.13. This model was run several times with different sets of parameters.

Listing 4.13: How the best parameters of the Random Forrest classifier was found

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [100,125],
    'max_features': [100,125],
    'min_samples_leaf': [400,500,600],
    'min_samples_split': [200,250,300],
    'n_estimators': [50,70,90]
}
# Make grid search classifier
clf_grid = GridSearchCV(RandomForestClassifier(), param_grid,
verbose=2, scoring= "f1_weighted", n_jobs =7)

# Train the classifier
clf_grid.fit(Traindata, Trainclasses)


print("Best␣Parameters:\n", clf_grid.best_params_)
print("Best␣Estimators:\n", clf_grid.best_estimator_)
```

5 Results

In this chapter the raw results from experiment 1 and 2 will be presented. These results will later be analyzed in chapter 6. (For more information about the different performance measures, please see section 2.3.6)

5.1 Experiment 1

The results from experiment 1 are presented in table 5 and table 6. Table 5 contain the results of all the models trained over 200 epochs, while table 6 contains the results from the models trained on 800 epochs. The raw performance reports from all the individual models tested in experiment 1, that contains the results that these tables are based on, can be found in the chapter B and C.

It should be noted that the two models BLSTM x4 and BLSTM x5, did not get tested in the first test run with 200 epochs. The reason for this was that this test run was done on machine 1. Which unfortunately did not have powerful enough hardware to train and test these models, as they are the most complex models, and therefore demand a lot of resources. They were however tested successfully in the second test run which was run on machine 2.

5.2 Experiment 2

The raw results from experiment 2 are presented in table 7, 8, 9 and 10. Table 7 presents the performance results of the four different activations used with a Neural Network of the type Bidirectional LSTM with added LSTM layers (in this case two LSTM layers where added). Table 8 contains the performance results of the four types of activation with a Neural Network of the type Bidirectional LSTM, where the Bidirectional LSTM layers where stacked twice. Table 9 contains the performance results of the four different activations in combination with a Convolutional LSTM Neural Network, where the Convolutional LSTM layers were stacked twice. Lastly table 10 contains the results of the four different activations in combination with a LSTM Neural Network, where the LSTM layers was stacked four times. The raw performance reports that these tables are based on can be assessed in chapter D.

Model:	F1 Macro	F1 Weighted	F1 bully	F1 None	Precision Macro	Recall macro	Accuracy	Avg. epochs	Max epoch
BLSTM LSTM	0,82956	0,86276	0,76047	0,89865	0,81393	0,86049	0,8578	46,4	61
BLSTM LSTM x 2	0,83066	0,86452	0,75999	0,90134	0,81626	0,85566	0,86031	76,4	170
BLSTM LSTM x 3	0,82497	0,85895	0,75425	0,8957	0,80968	0,85487	0,8539	86,2	142
BLSTM LSTM x 4	0,82527	0,85804	0,75697	0,89357	0,8094	0,8602	0,85231	130,4	197
BLSTM LSTM x 5	0,20691	0,10828	0,41382	0	0,13052	0,5	0,26105	1,5	3
BSLTM	0,82604	0,85949	0,75632	0,89576	0,80947	0,85858	0,85414	23,8	36
BLSTM x 2	0,82753	0,86098	0,75786	0,8972	0,81142	0,85784	0,85598	59,9	137
BLSTM x 3	0,83046	0,86367	0,76144	0,89949	0,81581	0,85934	0,85895	62,1	99
LSTM	0,82675	0,86015	0,75712	0,89638	0,8121	0,85727	0,85504	27	50
LSTM x 2	0,83258	0,86455	0,76597	0,8992	0,81673	**0,86528**	0,8594	81,1	171
LSTM x 3	0,82977	0,86316	0,76025	0,8928	0,81497	0,85887	0,85848	76,9	129
LSTM x 4	0,82802	0,86156	0,75833	0,89771	0,81273	0,85782	0,85666	93,5	127
LSTM x 5	0,82558	0,85915	0,75542	0,89574	0,80971	0,85767	0,8539	158	196
CONVLSTM	0,82759	0,86171	0,75363	0,89882	0,81453	0,85362	0,85736	40,2	158
CONVLSTM x 2	0,83229	0,86547	0,76311	0,90147	0,81692	0,86011	0,861	36,5	68
CONVLSTM x 3	**0,83399**	**0,86746**	**0,76431**	**0,90367**	**0,81942**	0,85934	**0,86353**	41,8	66
CONVLSTM x 4	0,83008	0,8647	0,75786	0,90229	0,81697	0,85279	0,86101	39,7	57
CONVLSTM x 5	0,83144	0,86499	0,76148	0,90139	0,81646	0,85748	0,86077	53,7	112

Table 5: Performance results from experiment 1, with 200 epochs

Modell	F1 macro	F1 Weighted	F1 bully	F1 none	Precision macro	Recall Macro	Accuracy	Avg. Epochs	Max epoch
BLSTM LSTM	0,82724	0,861	0,75691	0,89757	0,81219	0,85588	0,85619	68,2	154
BLSTM LSTM x 2	0,83154	0,86435	0,76304	0,90003	0,81648	0,86113	0,85964	85,9	175
BLSTM LSTM x 3	0,83233	0,86584	0,76243	0,90222	0,8179	0,85768	0,86169	212,7	764
BLSTM LSTM x 4	0,82488	0,85761	0,75653	0,89322	0,81025	0,86056	0,85184	186,1	513
BLSTM LSTM 5	0,83007	0,86287	0,76146	0,89867	0,81509	0,85979	0,85802	438,5	561
BLSTM	0,82629	0,86031	0,75532	0,89726	0,81118	0,85558	0,85551	25,7	53
BLSTM x 2	0,83218	0,86492	0,76396	0,90039	0,81655	0,86235	0,8601	131,7	770
BLSTM x 3	0,82794	0,86108	0,75895	0,89694	0,81275	0,85904	0,85597	115,4	377
BLSTM x 4	0,82947	0,86252	0,76066	0,89828	0,81415	0,85951	0,85757	107,5	392
BLSTM x 5	0,8331	0,86623	0,76397	0,90222	0,82009	0,85814	0,86215	166,1	512
LSTM	0,82609	0,85917	0,7571	0,89507	0,81027	0,85928	0,85367	30	40
LSTM x 2	0,82944	0,86278	0,75997	0,89892	0,81465	0,85809	0,85803	65,6	114
LSTM x 3	0,82973	0,86273	0,76111	0,89835	0,81469	0,86066	0,85781	90,5	230
LSTM x 4	0,83182	0,86501	0,76275	0,9009	0,81662	0,85964	0,86055	146,7	372
LSTM x 5	0,83018	0,86347	0,76088	0,89949	0,8147	0,85876	0,85872	248,6	684
ConvLSTM	0,83267	0,86534	0,76445	0,90088	0,81852	0,86139	0,86077	43,3	224
ConvLSTM x 2	**0,83808**	**0,87087**	**0,76969**	**0,90647**	**0,82419**	0,86169	**0,86719**	312,8	694
ConvLSTM x 3	0,83193	0,86561	0,7618	0,90206	0,8196	0,85684	0,86171	96,5	301
ConvLSTM x 4	0,83588	0,86866	0,76767	0,90409	0,822	0,86145	0,864666	62,6	185
ConvLSTM x 5	0,83606	0,86823	0,76901	0,9031	0,82011	**0,86518**	0,86375	80,4	328

Table 6: Performance results from experiment 1, with 800 epochs

Activation	F1 macro	F1 Weighted	F1 bully	F1 none	Precision macro	Recall macro	Accuracy
NN Softmax	0,82708	**0,85998**	0,75854	**0,89563**	0,81131	0,86042	**0,85459**
NN SVM	**0,82776**	0,85997	**0,76047**	0,89504	**0,81333**	**0,86343**	0,85435
SVM	0,18952	0,18344	0,20572	0,17333	0,18544	0,30431	0,22818
RFC	0,4247	0,6281	0	0,84981	0,36948	0,5	0,73895

Table 7: Performance results from the test with BLSTM with added LSTM layers design in experiment 2

Activation	F1 macro	F1 Weighted	F1 bully	F1 none	Precision macro	Recall macro	Accuracy
NN Softmax	0,82832	0,86321	0,7556	**0,90105**	0,81456	0,85033	**0,8594**
NN SVM	**0,83244**	**0,86457**	**0,76543**	0,89944	**0,81627**	**0,86615**	0,85939
SVM	0,71252	0,75343	0,62818	0,79686	0,72742	0,77327	0,74322
RFC	0,4249	0,6281	0	0,84981	0,36948	0,5	0,73895

Table 8: Performance results from the test with BLSTM design in experiment 2

Activation	F1 macro	F1 Wheigted	F1 bully	F1 none	Presision macro	Recall macro	Accuracy
NN Softmax	0,83385	0,86667	0,76541	0,90228	0,82002	0,86078	0,86238
NN SVM	**0,84021**	**0,87262**	**0,77269**	**0,90774**	**0,82566**	**0,86424**	**0,86902**
SVM	0,42095	0,59027	0,06496	0,77693	0,4007	0,45339	0,64257
RFC	0,70667	0,7881	0,538	0,87534	0,70231	0,74073	0,81341

Table 9: Performance results from the test with Conv LSTM design in experiment 2

Activation	F1 macro	F1 Wheigted	F1 bully	F1 none	Presision macro	Recall macro	Accuracy
NN Softmax	0,82862	0,8615	0,7601	0,89713	0,81419	0,85992	0,85643
NN SVM	**0,83625**	**0,86853**	**0,76879**	**0,90372**	**0,82094**	**0,86412**	**0,8642**
SVM	0,20889	0,13028	0,37583	0,04196	0,16947	0,46316	0,25807
RFC	0,4249	0,6281	0	0,84981	0,36948	0,5	0,73895

Table 10: Performance results from the test with LSTM design in experiment 2

6 Analysis

The purpose of this chapter is to analyze the results from the two experiments that where presented in the previous chapter (chapter: 5). The goal of the analysis is to extract knowledge from the raw results, so that arguments can be made towards a conclusion of the research questions (described in section 1.6).

6.1 The effects of increased training

The first concept to be analyzed is what effect increased training has on the performance of a model. The performance measurement we will focus on in this analysis is the F1 score for the bully class as this was used as the decision criteria for the model selection during training (as explained in section 3.5).

There are two ways one could argue that a model is utilizing the training more than another model. One is by looking at the average epochs used to find the prime trained version of the model from each training run. This measurement is calculated by averaging the number of epochs used to find the best performing trained version of a model from each training run. As explained in section 3.1.5, each model is trained 10 times on different parts of the data set. As we will discover in the analysis in the next subsection(6.1.1), this is not necessarily a good way to evaluate how much training a model receives. The second way one could decide that a model is receiving more training, is by comparing a model's performance with two different constraints in number of epochs it is trained for. This effect will be analyzed in subsection 6.1.2.

6.1.1 The effect of training, analyzed as a function of average epochs used

Figure 12 shows a plot of the performance and average epochs used for all of the models that was trained and tested in experiment 1. As seen in this figure it does not seem to be a clear connection between the two measurements. The correlation coefficient of the F1 score for the bully class and the average epochs used equals 0,249. This indicates that there may be a positive correlation between the two measurements. A positive correlation between two variables means that if one of them increase, the other one is also likely to increase. Which in this case may indicate that increased training might lead to an increased performance. But since the correlation coefficient is only 0,249 it is too low for there to be a strong correlation, meaning that in this case a clear cause and effect relationship between the variables cannot be proven.

In order to understand more about what is going on here we have to look at another measurement, the maximum epochs needed in the training to find the best trained version of a model. In figure 13 the maximum epoch needed in the training for every model is displayed. From this figure we can see that the maximum epochs needed for every model type increases when the model goes from not being stacked repeatably to the model being stacked twice. An example of this can be seen

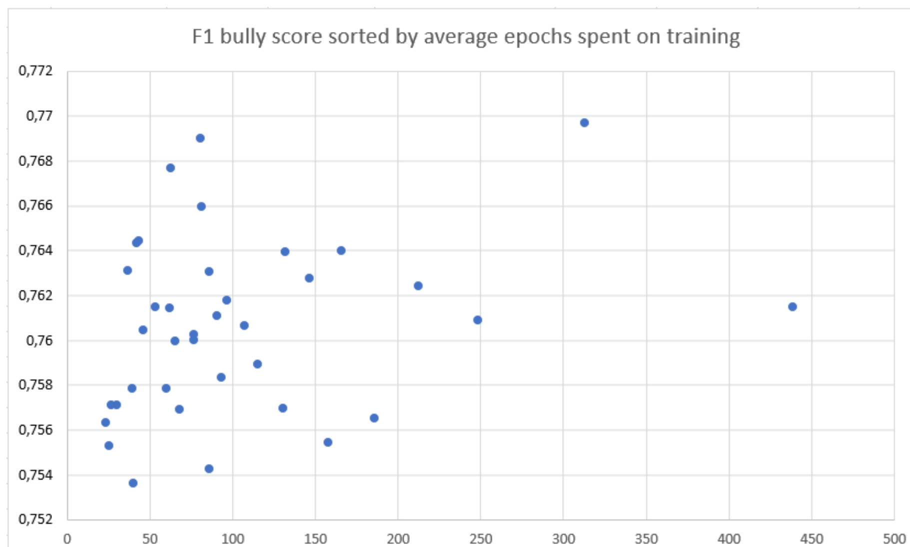


Figure 12: F1 score for the bully class of all models tested in experiment 1, as a function of average epochs used in the training (Y-axis: F1 score for the bully class, X-axis: average epochs used)

by looking at the BLSTM models graph from the 800 epochs run, with no stacking (BLSTM) and its layers stacked twice (BLSTM x2). The reason for this is that the stacking of the layers increases the complexity of the model, and increased complexity requires that the model has to be trained for an increased amount of epochs in order to be fully trained. Therefore, we can assume that the BLSTM with its layers stacked three times (BLSTM x3) is not fully trained, since we from the figure can see that the BLSTM model with its layers stacked only two times is close to the constraint of 800 epochs. We can also assume from these numbers that the BLSTM model with its layers stacked twice do not get enough training in the first test run, where the number of epochs were constrained to 200. Since in the second test run, with epochs constrained to 800, the models maximum epochs used was close to this constraint.

This suggests that by comparing the performance of the models in figure 12, we are actually comparing fully trained models with models that have not been trained enough. Thus we are not actually looking into the effects of increased training. But this goes to show that there is not a clear correlation between average amount of epochs a model needs to find its prime trained version and the performance it yields.

6.1.2 The effect of training, analyzed by training a model for 200 and 800 epochs

In order to analyze the effect of increased training we have to identify the models that were likely to not get enough training in the first test run with 200 epochs of training, but who may have been fully trained in the second test run with 800 epochs. In order to identify these models, we can study the graph in figure 14, which shows the stacked versions of all the models and the maximum

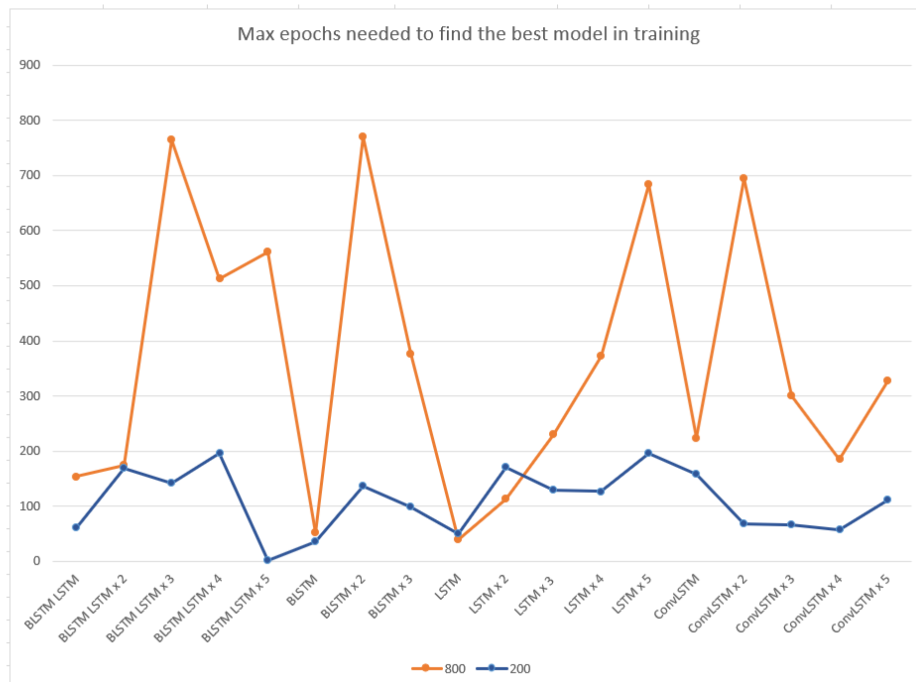


Figure 13: The highest amount of epochs needed in the 10 cross fold run of all models in order to get the best trained model, with maximum training set to 200 epochs and 800 epochs

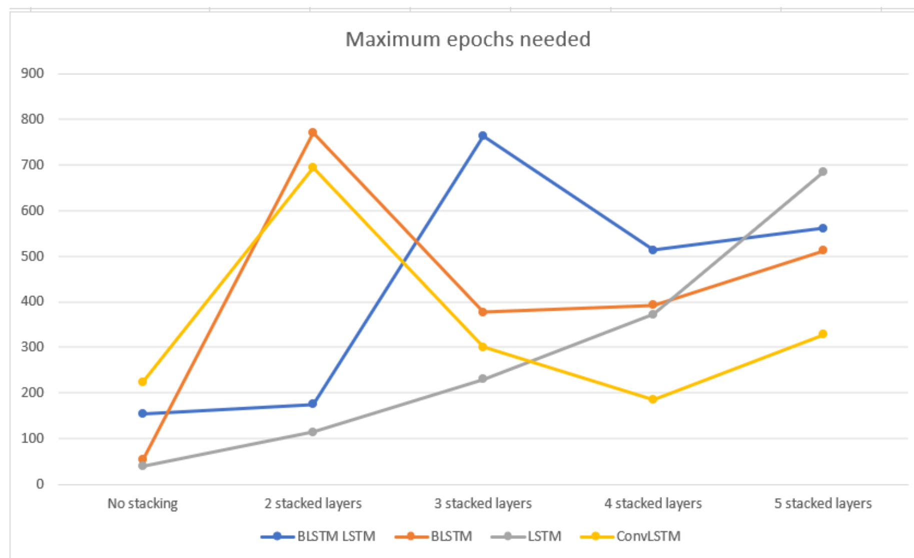


Figure 14: The highest amount of epochs needed in the 10 cross fold run of all models in order to get the best trained model, with maximum training set to 800 epochs

number of epochs they needed in the second test run, where the training was constrained to 800 epochs. From the figure we identify these models by taking the first stacked version of a model that have a maximum number of epochs needed higher than 200 epochs, and we include all the stacked versions of the models until the maximum number of epochs get close to 800. The models identified to apply to these criteria is the model BLSTM LSTM stacked three times, BLSTM stacked twice, LSTM stacked three, four and five times, and the ConvLSTM without stacking and stacked twice. In figure 15, the performance of these models for the first and second test run is presented. And by inspecting this graph we can indeed see that all of the models performed best in the second test run where the models were trained for an increased amount of epochs. By calculating the correlation coefficient for the performance of these selected models and the constraint of training epochs of either 200 or 800, we get a correlation coefficient equal to 0,714. This tells us that it is indeed a noticeable positive correlation between the number of epochs the training is run and the performance the model yields. It is however not a perfect correlation, which means that the performance of the model is not only determined by the increased training.

6.2 The effects of stacking similar layers

The effect the stacking of layer's has on the performance the model's yield can be assessed by studying figure 16. The graph in this figure shows the F1 score for the bully class yielded by every model type when no stacking is applied, and all the way up to stacking the layers five times. As visible from the graph, all of the models increase in performance by going from no stacking to stacking the layers twice. Further stacking beyond two times does not yield any clear trend, as the

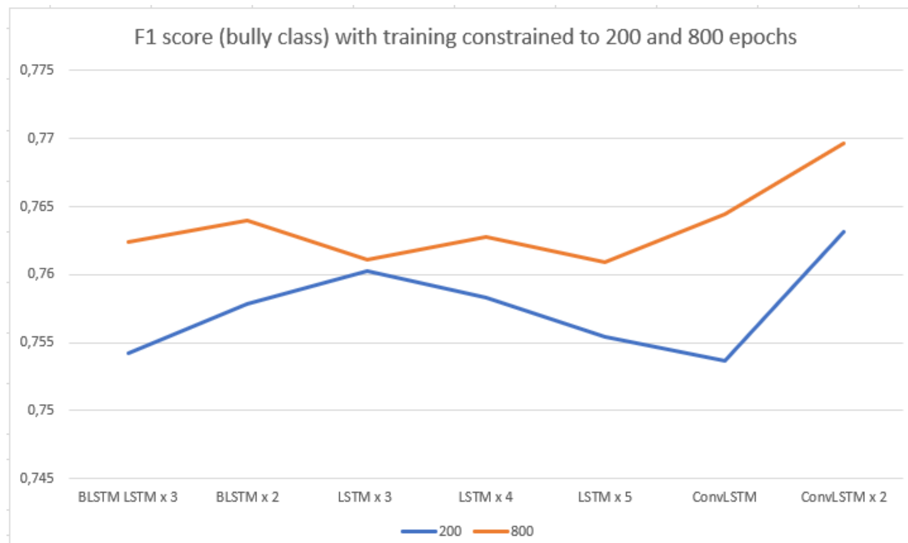


Figure 15: Performance comparison of the models; BLSTM LSTM x2, BLSTM x2, LSTM x3, LSTM x4, LSTM x5, ConvLSTM and ConvLSTM x2 with the training constrained to 200 and 800 epochs

performance for the ConvLSTM model and the BLSTM model decreases, the performance of the BLSTM LSTM model stays approximately the same, and the LSTM model's performance increases. This could be a sign that the increased complexity of the model is not needed in order to classify the data. But we see this more probable to be a result of some models not being able to train enough with the added complexity. From figure 14, we can see that the BLSTM model and ConvLSTM model has a maximum epochs needed for training quite close to the constraint of 800 when the layers are stacked twice. As already discussed, this most likely indicates that these models stacked three times is not going to be fully trained with just 800 epochs of training. The BLSTM LSTM model however does not obtain the maximum epochs needed in training close to 800 before it is stacked three times. Lastly the LSTM model only get close to this constraint when it is stacked five times. While the BLSTM LSTM model does not increase its performance by being stacked three times, it does not decrease its performance either. This could be assessed as the model not benefiting from the stacking beyond two times or that it merely is not trained enough with its layers stacked three times. The LSTM model however seem to increase its performance all the way up to stacking its layers four times. This is interesting since this is the model that is the furthest away from the constraint of 800 epochs when looking at the maximum epochs needed for training. The only time it gets close to this constraint is when it is stacked five times, which also is the first time we see that the performance of the LSTM model decreases as a result of increased stacking. Therefore, we cannot prove that stacking beyond five times with a standard LSTM network will yield any gain in performance, but it should perhaps be tested with a even higher epoch constraint before being evaluated as fully disproved.

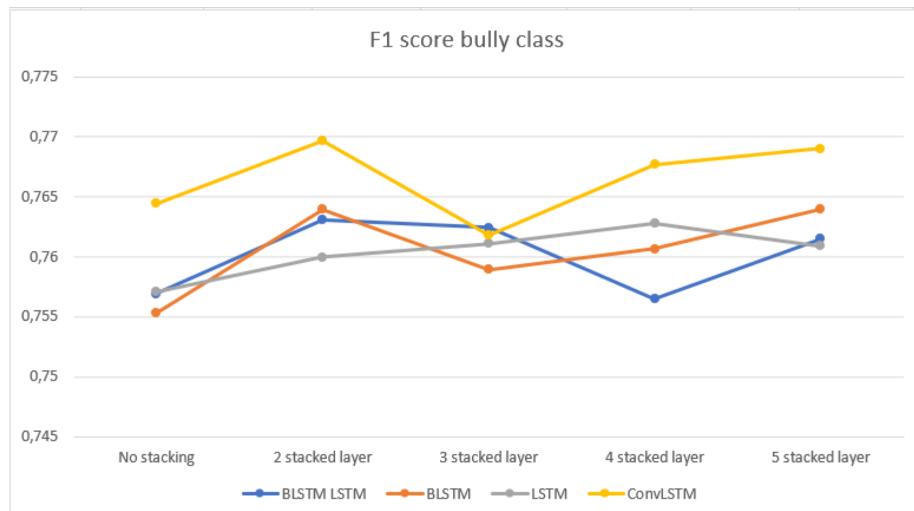


Figure 16: F1 score for the bully class of all models in the second test run where the models were trained with 800 epochs

6.3 Comparison of the different network types

The most important remark to make about the different LSTM designs is that there does not seem to be a large performance difference between them. In the second test run where the models were trained over 800 epochs the best model performed 1,867% better than the worst performing model. While this difference may not sound like much there still seem to be a significant difference. The ConvLSTM design looks to be performing repeatedly better than the other designs, as can be seen in figure 16. The only time that the ConvLSTM performed worse than any of the other models was when its layers were stacked three times. This is a bit weird as the ConvLSTM model with its layers stacked two or four times performs very good. This is most likely due to the fact that the model is not trained enough, as discussed earlier. The difference between the designs should therefore be measured by how they perform when they are not stacked, and when the layers are stacked twice, since these versions of the models are most likely to have been trained enough. From these two versions of the models we see that the BLSTM, LSTM and BLSTM LSTM yields very similar performance when not stacked, but when the layers are stacked two times it may seem like the BLSTM model and the BLSTM LSTM model are performing slightly better than the LSTM model. The performance winner is however the ConvLSTM model that performs better than the other three models both when not stacked, and stacked twice.

6.4 Comparison of the different activation mechanisms

In experiment 2 we tested three different activation mechanism as an alternative to the traditional Softmax activation. Two of these where to replace the activation layers in the Neural Network

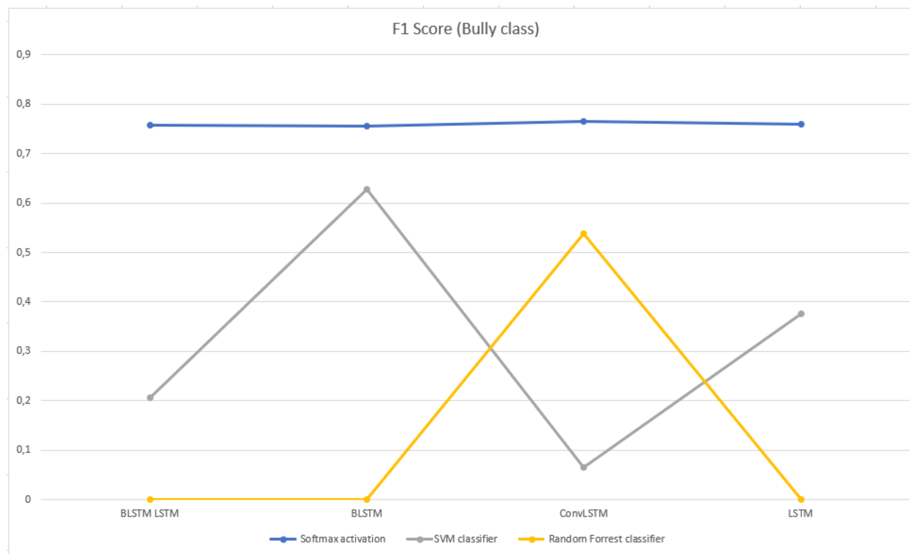


Figure 17: F1 score for the bully class yielded by the standard Neural Network model with Softmax activation, a Neural Network where the Softmax activation layer were replaced with a SVM classifier and a Neural Network where the Softmax activation layer were replaced with a Random Forrest classifier

with two completely different classifiers; a Random Forest classifier and a Support-Vector-Machines classifier. Figure 17, shows the performance of these methods compared to the Softmax activated model for the four different network designs tested. As seen from this figure these models were not able to yield any performance that could compete with the Softmax activated models. It is however interesting to see that the Random Forest classifier performs best when combined with a ConvLSTM network, but the SVM classifier performs best when combined with the BLSTM network.

The last of the three alternative activation mechanism tested was to replace the Softmax activation layer with a SVM inspired activation layer. In figure 18 the performance of these two activation methods are presented, in combination with four different network types. The comparison shows that the SVM alike activation outperforms the standard Softmax activation in combination with all of the network types that was tested. The SVM like activation is also the only method tested in these experiments that was able to yield a F1 score for the bully class over 0,77.

As mentioned in section 3.4.2, the models in this experiment was only tested with maximum epochs set to 800. So while we cannot compare the results with the amount of training, it is still interesting to take a look at the average epochs needed to find the prime trained version from each fold in the 10 cross fold testing of the models. These numbers are illustrated in the graph in figure 19. As this graph shows the SVM alike activation drastically increases the average epochs needed for training the model. The biggest increase is seen for BLSTM LSTM model where the average epochs needed for training increased with approximately 325%, from 82,2 epochs with the Softmax activation to 349,7 epochs with the SVM alike activation. The lowest increase, measured in percentage,

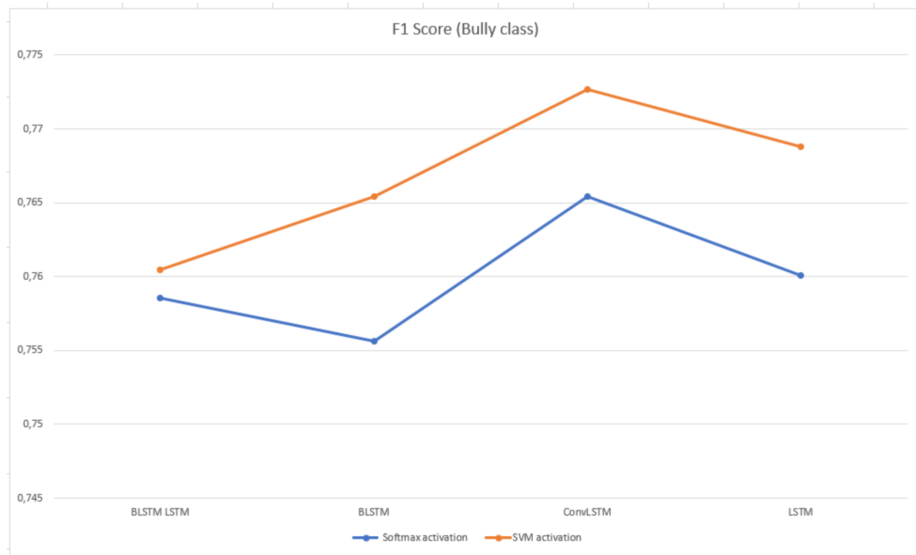


Figure 18: F1 score for the bully class yielded by the standard Softmax activation and the SVM alike activation

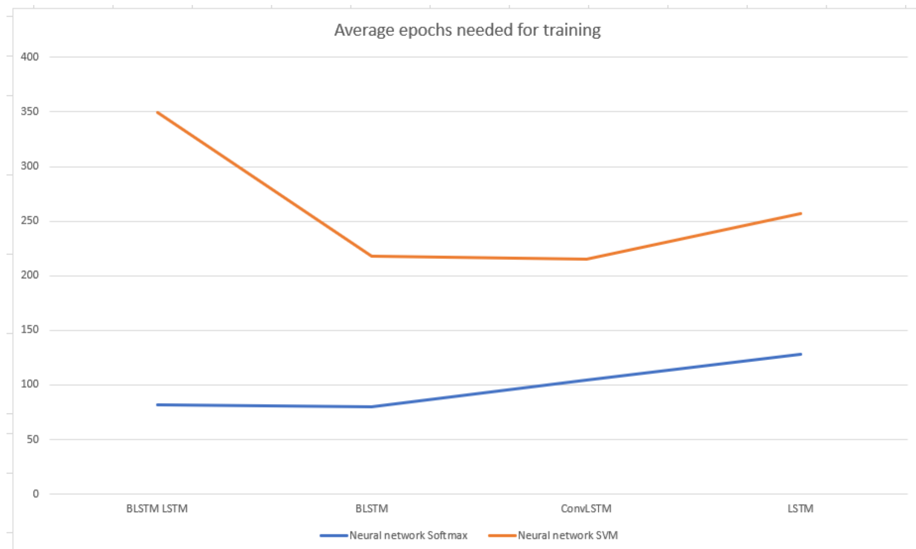


Figure 19: F1 score for the bully class yielded by the standard Softmax activation and the SVM alike activation

was seen for the LSTM model where the increase was 101%, from 128,1 with the Softmax activation to 257 with the SVM alike activation. This indicates that the performance improvement seen in figure 18 comes with a cost of increased training. This could suggest that the model with the SVM activation would perform even better with increased training. It is however just an indication of increased training being required for the SVM activation though, as the models have not been tested with different constraints for the amount of epochs in the training.

6.4.1 The best model compared with the state of the art models

As we have seen, the ConVLSTM with its layers stacked twice with the new SVM alike activation, performs better than the other models tested. But how good is it actually compared to the models used in the state-of-the-art research within the field? To answer this question, we compare the weighted averaged F1 score, the F1 score of the bully class, the F1 score of the not bully class and the overall accuracy of the new proposed model with models featured in the state-of-the-art research. The comparison can be seen in figure 20. The state-of-the-art models chosen for the comparison is the LSTM, BLSTM and ConvLSTM models, all without any stacking from experiment 1. The LSTM model were featured in the research [33], the BLSTM model were presented in the research article [2], and the ConvLSTM method is based on the model presented by [32]. From the comparison we can see that the new ConvLSTM model with layer stacking and SVM alike activation are outperforming the state-of-the-art models on all of the performance measures. The difference is not huge though as the new model, averaged over all five of the performance measures, performed 0,9% better than the ConvLSTM model based on [32], 1,6% better than the BLSTM model presented by [2], and 1,7% better than the LSTM model designed by [33].

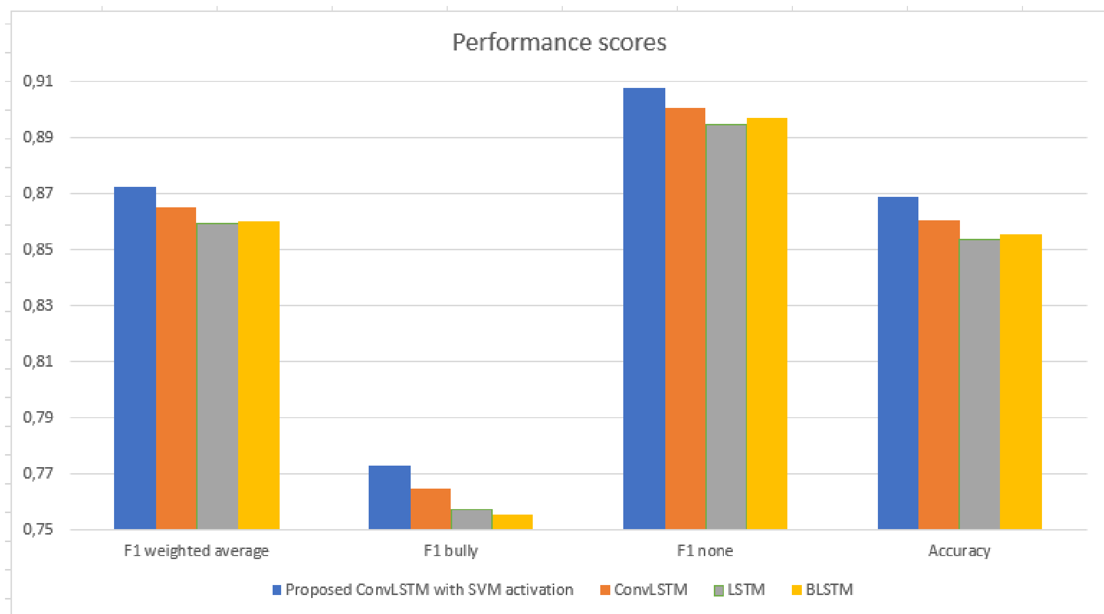


Figure 20: Performance comparison between the new ConvLSTM model with its layers stacked three times with SVM alike activation and three models featured in or based on the state-of-the-art research; BLSTM, LSTM and ConvLSTM without any stacking and with Softmax activation

7 Discussion

In this chapter we will discuss some of the main challenges that was encountered in the work with this thesis, and give a short evaluation of the project as a whole.

7.1 Challenges

7.1.1 Limited python experience

We had close to no experience with coding in the program language Python before this project started. Since the Python environment has so many third-party libraries for NLP tasks it was decided that this would be the preferred programming language. This did however slow down the development process somewhat, thus leaving less time to run tests and write the thesis.

7.1.2 Creating a state-of-the-art NLP pipeline

One of the biggest challenges with the implementation of this thesis was to build a state-of-the-art NLP pipeline that both represented all of the groundbreaking existing research and yielded performance that could also be compared with the state-of-the-art methods. There are several reasons why this was a challenge. One of the reasons is the several different data sets which are used by the researchers in the field of cyberbullying, and as described by [16] these data sets tend to yield very different performance. This made it very difficult to compare the different methods in order to decide which techniques to base the NLP pipeline in this thesis on. Another challenge is that the code used in similar research papers rarely is released. It also varies a great deal how detailed the implementations are explained in these papers, often only the core parts of the NLP pipeline that the researchers want to draw attention to is described in detail.

Discovering invalid performance results in the state-of-the-art research

The work with the NLP-pipeline started out as a quest to recreate the NLP-pipeline made by [2], as they are one of very few that has published all of their code. Because of some errors it was not possible to simply run their code, so the work of recreating their work had to begin, with the goal of yielding the same performance as they had reported to achieve. Since we wanted to really understand all of the code the decision was made to not only copy their code, but to rewrite it in our own way. The problems become prevalent when the program was finished and the performance testing began, as the performance we were able to get was not close to what [2] had reported. After a lot of investigation. it was discovered that the oversampling technique used by [2] was faulty. Since they were oversampling their data sets before splitting it into training and testing sets. This resulted in information (data samples) being leaked from the testing set to the training set, thus making the reported performance results invalid. Proof of this claim can be seen in the Appendix section A.2.

This was very frustrating as we had spent a lot of time on recreating their results. Therefore in order to not let all the work be wasted we decided to base our NLP pipeline on this work and make some changes in order to get a NLP pipeline that yielded respectable results as well as represented the other NLP pipelines used in other research within the field of cyberbullying. The search for the perfect NLP pipeline consisted of testing a bunch of different NLP pipelines. From very complex pipelines that:

- Removed any stop words
- Identified the type of language used in the data sample and applied specialized stemming and POS-tagging algorithms customized for that language.
- Identified possibly masked profanity words by calculating the Levenshtein distance between the words in the data samples, and a list of profanity words

To the simplest NLP-pipelines that only vectorized the data. In the end we landed on a good middle ground between the most complex and the simplest pipelines. Which we believe is a pipeline that reflects the pipelines used in most of the state-of-the-art research in a worthy manner, and that yields decent performance on the data set that we used.

7.2 Evaluation

This thesis actually started out as a project to make a tool for identifying high risk persons operating on social media platforms. This plan was however changed when our plan to acquire our data set from Kripos fell through. Then we started to look in to the cyberbullying field instead. We looked into several different options for what to research within this field before we decided to analyze the performance effects of different types of Long-short-term-memory Neural Network designs. So, to state that the project has not followed a streamline plan is not exaggerating. In the end though we feel that we were able to create a project and a thesis that challenge the state-of-the-art research within its field in a beneficial way. By testing so many different models in controlled a lab environment we believe that we have been able to discover several cause and effect relationships within the Neural Network designed to detect cyberbullying. These are cause and effect relationships that can be utilized by future research within the field in order to achieve even better results, than what today's state-of-the-art research is able to achieve.

8 Conclusion

The conclusion of this thesis is split into four parts, one part for each research question.

How does the amount of training impact the performance of the Neural Network?

Through the experiments conducted in this thesis we have found that there is no correlation between the amount of epochs a Long-Short-Term-Memory based Neural Network model uses in its training to find its prime trained version, and the performance the model yields. But it is however a correlation between increasing the amount of epochs used in the training, and the performance, if the models are not trained enough in the first place.

How does the performance of the different types of Long-Short-Term-Memory Neural Networks compare, when tested in a controlled environment?

Our findings show that a Long-Short-Term-Memory Neural Network who is a combination of a Convolutional Neural Network and a Long-Short-Term-Memory Neural Network (ConvLSTM) yields the highest performance. The difference between the three other types of Long-Short-Term-Memory networks tested; LSTM, Bidirectional LSTM (BLSTM) and BLSTM with added LSTM layer(s), was marginal, and they should be considered as equal in term of performance yielded.

What is the performance effect of blindly increasing a Neural Network's complexity by stacking equal layers on top of each other?

Our experiment with stacking the same type of layers on top of each other showed that there is indeed a performance increase going from no stacking to stacking the layers twice. Stacking the layers more than two times however had a varying result. This was mainly because the stacking comes with a cost of an increased need for training the model over more epochs. The LSTM model did however benefit stacking the layers all the way up to four times. Which may indicate that stacking beyond two times could increase the performance if only the models are trained enough.

Is there a viable alternative to the commonly used Softmax activation layer?

Our findings showed that a Neural Network which uses an activation layer that is inspired by the mechanisms of the Support-Vector-Machines (SVM) classifier is indeed a good alternative to the standard Softmax activation. In our experiment this alternative activation outperformed the Softmax activation in all of the tests with different types of Long-Short-Term-Memory Neural Networks. Compared with models featured in state-of-the-art research that was tested with our setup, the ConvLSTM model with its layers stacked twice and with a SVM like activation yielded a F1 score of the bully class that was 0,9-1,7% higher.

8.1 Future Work

One of the main findings from this project has been that the SVM inspired activation layer works very well with cyberbullying detection. This method has never been tested for such a purpose before. It would therefore be interesting to test this method further within this field. This method was as mentioned inspired by the work of [43], that used this method for intrusion detection. [43] Identified this new method to be faster than the Softmax activated Neural Network in both the testing and training phase. As classification time would be of the essence within the field of cyberbullying detection as well, because of the huge amounts of data, it would be very interesting to see if the same would be true when classifying text data as well. Especially since our findings found that the SVM alike activation seemed to require a higher amount of epochs needed for training, but our experiment did not consider the time used.

Furthermore, it would also be interesting to see if this study could be reproduced with a different NLP pipeline. Since only one standard NLP pipeline was used for all the tests in this study, we could not investigate any cause and effect relationships between the Neural Network designs and the rest of the NLP pipeline. There may therefore be a minor chance that the findings from this study is only relevant for the specific NLP pipeline used in this study.

Similarly, it would also be interesting to see if the results could be reproduced with any of the other cyberbullying data sets available. As mentioned earlier, the work by [16] showed that there is a big difference on what kind of performance it is possible to get from the different data sets. So it would be interesting to see if the performance relationships between the different Neural Network designs identified in this thesis would stay the same. Or if perhaps the effects of the variables tested would be greater.

Finally, we hope this thesis and the discoveries made here will help in fighting cyberbullying in the years to come. Our hope is that one day technology will eliminate this kind of interactions on social media, and hopefully this work could help spark the future solutions.

Bibliography

[1] Shalaginov, A. January 2018. Lecture notes: "lecture 1: Introduction to data science in forensics and security".

[2] Agrawal, S. & Awekar, A. 2018. Deep learning for detecting cyberbullying across multiple social media platforms. *Advances in Information Retrieval*, 141–153. URL: http://dx.doi.org/10.1007/978-3-319-76941-7_11, doi:10.1007/978-3-319-76941-7_11.

[3] Chollet, F. 2016. Using pre-trained word embeddings in a keras model. https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html. Accessed: 2019-04-01.

[4] Zhu, X. 2015. Data and code for the study of bullying. http://research.cs.wisc.edu/bullying/data.html. Accessed: 2019-01-31.

[5] Fire av fem nordmenn bruker sosiale medier (online). August 2018. URL: https://www.ssb.no/teknologi-og-innovasjon/artikler-og-publikasjoner/fire-av-fem-nordmenn-bruker-sosiale-medier.

[6] Medietilsynet. 2018. Barn og medierundersøkelsen 2018. In *Barn og medier 2018*, 6. Medietilsynet. URL: https://www.medietilsynet.no/globalassets/publikasjoner/barn-og-medier-undersokelser/2018-barn-og-medier.

[7] Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. March 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3, 1137–1155. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

[8] DOMO. 2018. Data never sleeps. https://www.domo.com/solution/data-never-sleeps-6. Accessed: 2019-04-20.

[9] Kononenko, I. 2007. Machine learning and data mining : introduction to principles and algorithms.

[10] Louridas, P. & Ebert, C. Sep. 2016. Machine learning. *IEEE Software*, 33(5), 110–115. doi:10.1109/MS.2016.114.

[11] Linckels, S. & Meinel, C. *Natural Language Processing*, 61–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. URL: https://doi.org/10.1007/978-3-642-17743-9_4, doi:10.1007/978-3-642-17743-9_4.

[12] AB, I. 2019. Ascii code - the extended ascii table. https://www.ascii-code.com. Accessed: 2019-05-02.

[13] Lehmann, J. & Volker, J. 2014. *Perspectives on Ontology Learning: Studies on the Semantic Web 18*, volume 18 of *Studies on the Semantic Web 18*. Ios Press.

[14] Nandhini, B. S. & Sheeba, J. I. 2015. Cyberbullying detection and classification using information retrieval algorithm. In *Proceedings of the 2015 International Conference on Advanced Research in Computer Science Engineering & Technology (ICARCSET 2015)*, ICARCSET '15, 20:1–20:5, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2743065.2743085, doi:10.1145/2743065.2743085.

[15] Babar, N. 2018. The levenshtein distance algorithm. https://dzone.com/articles/the-levenshtein-algorithm-1. Accessed: 2019-05-05.

[16] Salawu, S., He, Y., & Lumsden, J. 2018. Approaches to automated detection of cyberbullying: A survey. *IEEE Transactions on Affective Computing*, 1–1. doi:10.1109/TAFFC.2017.2761757.

[17] Maynard, D. & Bontcheva, K. 2014. Natural language processing. In *Perspectives on Ontology Learning*, volume 18, 51–67. IOS Press.

[18] geeksforgeeks. Removing stop words with nltk in python. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. Accessed: 2019-03-02.

[19] Project, N. 2019. Natural language toolkit. https://www.nltk.org. Accessed: 2019-03-02.

[20] Kontostathis, A., Reynolds, K., Garron, A., & Edwards, L. 2013. Detecting cyberbullying: Query terms and techniques. In *Proceedings of the 5th Annual ACM Web Science Conference*, WebSci '13, 195–204, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2464464.2464499, doi:10.1145/2464464.2464499.

[21] Pennington, J., Socher, R., & Manning, C. D. 2018. Part-of-speech tagging. https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf. Pages: 151-154.

[22] Pennington, J., Socher, R., & Manning, C. D. 2018. Hidden markov models. URL: https://web.stanford.edu/~jurafsky/slp3/A.pdf.

[23] Mctear, M., Callejas, Z., & Griol, D. 2016. *The Conversational Interface: Talking to Smart Devices*. Springer International Publishing, Cham.

[24] Pennington, J., Socher, R., & Manning, C. D. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[25] Jeffrey Pennington, R. S. & Manning, C. D. 2014. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/. Accessed: 2019-02-25.

[26] Agarwal, S. & Sureka, A. 2015. Applying social media intelligence for predicting and identifying on-line radicalization and civil unrest oriented threats. *CoRR*, abs/1511.06858. URL: http://arxiv.org/abs/1511.06858, arXiv:1511.06858.

[27] Donges, N. 2018. The random forest algorithm. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd. Accessed: 2019-05-01.

[28] Keras. 2019. Keras: The python deep learning library. https://keras.io. Accessed: 2019-02-20.

[29] Keras. 2019. Usage of activations. https://keras.io/activations/. Accessed: 2019-02-23.

[30] Mahmood, H. 2018. The softmax function, simplified. https://towardsdatascience.com/softmax-function-simplified-714068bf8156. Accessed: 2019-04-23.

[31] Pitsilis, G., Ramampiaro, H., & Langseth, H. 2018. Effective hate-speech detection in twitter data using recurrent neural networks. *Applied Intelligence*, 48(12), 4730–4742.

[32] Zhang, Z., Luo, L., & Espinosa Anke, L. 2018. Hate speech detection: A solved problem? the challenging case of long tail on twitter. *Semantic Web*, 1–21.

[33] Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. 2017. Deep learning for hate speech detection in tweets. *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*. URL: http://dx.doi.org/10.1145/3041021.3054223, doi:10.1145/3041021.3054223.

[34] February 2013. Lecture notes from cornell university: "cs1114 section 6: Convolution". https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf.

[35] Nigam, V. 2018. Understanding neural networks. from neuron to rnn, cnn, and deep learning. https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90. Accessed: 2019-04-02.

[36] Skymind. A beginner's guide to lstms and recurrent neural networks. https://skymind.ai/wiki/lstm#long. Accessed: 2019-04-08.

[37] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555.

[38] Montgomery, D. C. 2013. Design and analysis of experiments.

[39] Mason, R. L. 2003. Statistical design and analysis of experiments : with applications to engineering and science.

[40] Fushiki, T. Apr 2011. Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2), 137–146. URL: https://doi.org/10.1007/s11222-009-9153-8, doi:10.1007/s11222-009-9153-8.

[41] Pennington, J., Socher, R., & Manning, C. D. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[42] Twitter. 2019. Docs. https://developer.twitter.com/en/docs. Accessed: 2019-02-05.

[43] Agarap, A. F. M. 2018. A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, ICMLC 2018, 26–30, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3195106.3195117, doi:10.1145/3195106.3195117.

[44] Rivera, P. Tweepy: An easy-to-use python library for accessing the twitter api. https://www.tweepy.org. Accessed: 2019-02-20.

[45] NumPy. Numpy. https://www.numpy.org. Accessed: 2019-02-15.

[46] pandas. pandas: Python data analysis library. https://pandas.pydata.org. Accessed: 2019-02-17.

[47] rrmina. Elegant and easy tweet preprocessing in python. https://github.com/rrmina/preprocessor. Accessed: 2019-02-20.

[48] Google. An end-to-end open source machine learning platform. https://www.tensorflow.org/about. Accessed: 2019-02-10.

[49] den Bossche, J. V. scikit-learn machine learning in python. https://scikit-learn.org/stable/. Accessed: 2019-02-10.

[50] Foundation, P. S. Python is a programming language that lets you work quickly and integrate systems more effectively. https://www.python.org. Accessed: 2019-01-20.

[51] Jupyter, P. The jupyter notebook. https://jupyter.org. Accessed: 2019-01-21.

[52] Jupyter, P. Jupyterlab. https://github.com/jupyterlab/jupyterlab. Accessed: 2019-01-21.

[53] Anaconda, I. The enterprise data science platform for.... https://www.anaconda.com. Accessed: 2019-01-20.

A Appendix

A.1 Data set lables

All of the labels and their possible values in the data set:

- bullying:
  - yes
  - no
- author role:
  - accuser
  - assistant
  - bully
  - defender
  - reinforcer
  - reporter
  - victim
  - other
  - NA - Not a bullying trace
- teasing
  - yes
  - no
  - NA - Not a bullying trace
- type
  - accusation
  - cyberbullying
  - denial
  - report
  - self-disclosure
  - NA - Not a bullying trace
- form
  - cyberbullying
  - other
  - physical

  - property damage
  - relational
  - verbal
  - NA - Not a bullying trace

- emotion
  - anger
  - embarrassment
  - empathy
  - fear
  - none
  - other
  - pride
  - relief
  - sadness
  - NA - Not a bullying trace

A.2 Proof of faulty oversampling technique

```
#Proof that Awekar oversampling techniqe is faulty
#Reads the indata:
labels=[]
x_text=[]
for index, row in data.iterrows():
    if row[ClassFieldName] == "none ":
        row[ClassFieldName]="none"
    x_text.append(row[CommentFieldName])
    labels.append(row[ClassFieldName])


#Awekar oversampling techniqe, as seen in DNNs.ipynb method: "def get_data(data, oversampling_rate)
NUM_CLASSES = 3
dict1 = {'racism':2,'sexism':1,'none':0}
labels = [dict1[b] for b in labels]
oversampling_rate = 4
racism = [i for i in range(len(labels)) if labels[i]==2]
sexism = [i for i in range(len(labels)) if labels[i]==1]
x_text = x_text + [x_text[x] for x in racism]*(oversampling_rate-1)+ [x_text[x] for x in sexism]*(oversampling_rate-1)
labels = labels + [2 for i in range(len(racism))]*(oversampling_rate-1) + [1 for i in range(len(sexism))]*(oversampling_rate-1)
labels

from sklearn.model_selection import train_test_split, KFold
X_train, X_test, Y_train, Y_test = train_test_split( x_text, labels, random_state=42, test_size=0.10)


#Test wheter samples in train set is in the test set
count_overlapping_samples=0
for sample_train in X_train:
    for sample_test in X_test:
        if sample_train == sample_test:
            count_overlapping_samples=count_overlapping_samples+1
print("Number of overlapping samples found: ", count_overlapping_samples)
```

```
Number of overlapping samples found:  5445
```

Figure 21: Proof that the testing data set is leaking information to the training data set, with the oversampling technique used by [2]

A.3 The code for the models in experiment 1

A.3.1 LSTM

The implementation of the LSTM model is seen in listing A.1

Listing A.1: The implementation of the LSTM model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.2 LSTM x2

The implementation of the LSTM x2 model is seen in listing A.2

Listing A.2: The implementation of the LSTM x2 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_, return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.3 LSTM x3

The implementation of the LSTM x3 model is seen in listing A.3

Listing A.3: The implementation of the LSTM x3 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_, return_sequences=True))
```

```
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.4 LSTM x4

The implementation of the LSTM x4 model is seen in listing A.4

Listing A.4: The implementation of the LSTM x4 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_, return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.5 LSTM x5

The implementation of the LSTM x5 model is seen in listing A.1

Listing A.5: The implementation of the LSTM x5 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_, return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
```

```
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.6 BLSTM LSTM

The implementation of the BLSTM LSTM model is seen in listing A.6

Listing A.6: The implementation of the BLSTM LSTM model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.7 BLSTM LSTM x2

The implementation of the BLSTM LSTM x2 model is seen in listing A.7

Listing A.7: The implementation of the BLSTM LSTM x2 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.8 BLSTM LSTM x3

The implementation of the BLSTM LSTM x3 model is seen in listing A.8

Listing A.8: The implementation of the BLSTM LSTM x3 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.9 BLSTM LSTM x4

The implementation of the BLSTM LSTM x4 model is seen in listing A.9

Listing A.9: The implementation of the BLSTM LSTM x4 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.10 BLSTM LSTM x5

The implementation of the BLSTM LSTM x5 model is seen in listing A.10

Listing A.10: The implementation of the BLSTM LSTM x5 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.11 BLSTM

The implementation of the BLSTM model is seen in listing A.11

Listing A.11: The implementation of the BLSTM model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_)))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.12 BLSTM x2

The implementation of the BLSTM x2 model is seen in listing A.12

Listing A.12: The implementation of the BLSTM x2 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid')))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.13 BLSTM x3

The implementation of the BLSTM x3 model is seen in listing A.13

Listing A.13: The implementation of the BLSTM x3 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid')))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.14 BLSTM x4

The implementation of the BLSTM x4 model is seen in listing A.14

Listing A.14: The implementation of the BLSTM x4 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
```

```
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid')))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.15 BLSTM x5

The implementation of the BLSTM x5 model is seen in listing A.15

Listing A.15: The implementation of the BLSTM x5 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid',return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid')))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.16 ConvLSTM

The implementation of the ConvLSTM model is seen in listing A.16

Listing A.16: The implementation of the ConvLSTM model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.17 ConvLSTM x2

The implementation of the ConvLSTM x2 model is seen in listing A.17

Listing A.17: The implementation of the ConvLSTM x2 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(128,9, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.18 ConvLSTM x3

The implementation of the ConvLSTM x3 model is seen in listing A.18

Listing A.18: The implementation of the ConvLSTM x3 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
```

```
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(128,9, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(64,7, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.19 ConvLSTM x4

The implementation of the ConvLSTM x4 model is seen in listing A.19

Listing A.19: The implementation of the ConvLSTM x4 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(128,9, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(64,7, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(32,5, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
```

```
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.3.20 ConvLSTM x5

The implementation of the ConvLSTM x5 model is seen in listing A.20

Listing A.20: The implementation of the ConvLSTM x5 model

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(128,9, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(64,7, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(32,5, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(16,4, padding='same', activation='relu',
strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

A.4 The code for the models in experiment 2

A.4.1 SVM like activation

LSTM

Listing A.21: The implementation of the LSTM model with SVM like activation

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(LSTM(embsize_, return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='linear',
kernel_regularizer=l2(0.01)))
modelNTNU.compile(loss='categorical_hinge',
          optimizer='adam',
          metrics=['accuracy'])
```

BLSTM LSTM

Listing A.22: The implementation of the BLSTM LSTM model with SVM like activation

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(LSTM(embsize_, activation='sigmoid',
return_sequences=True))
modelNTNU.add(LSTM(embsize_, activation='sigmoid'))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='linear',
kernel_regularizer=l2(0.01)))
modelNTNU.compile(loss='categorical_hinge',
          optimizer='adam',
          metrics=['accuracy'])
```

BLSTM

Listing A.23: The implementation of the BLSTM model with SVM like activation

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
```

```
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Bidirectional(LSTM(embsize_,
return_sequences=True)))
modelNTNU.add(Bidirectional(LSTM(embsize_,
activation='sigmoid')))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='linear',
kernel_regularizer=l2(0.01)))
modelNTNU.compile(loss='categorical_hinge',
          optimizer='adam',
          metrics=['accuracy'])
```

ConVLSTM

Listing A.24: The implementation of the ConVLSTM model with SVM like activation

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_,return_sequences=True))
modelNTNU.add(Conv1D(128,9, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size, activation='relu'))
modelNTNU.add(Dense(2, activation='linear',
kernel_regularizer=l2(0.01)))
modelNTNU.compile(loss='categorical_hinge',
          optimizer='adam',
          metrics=['accuracy'])
```

A.4.2 The SVM and Random Forest Classifier used

Listing A.25: The SVM classifier and Random Forest classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

#The random forest classifier:
RFC = RandomForestClassifier(bootstrap=True,
    class_weight=None, criterion='gini',
```

```
    max_depth=1000, max_features=2000, max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=500, min_samples_split=500,
    min_weight_fraction_leaf=0.0, n_estimators=300,
    n_jobs=None, oob_score=False, random_state=None,
    verbose=0, warm_start=False)

#The SVM classifier:
clf = svm.SVC(C=0.5, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3,
    gamma=0.01, kernel='rbf', max_iter=-1, probability=False,
    random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

A.5 The code for the testing environment

A.5.1 Original script for retrieving the Tweets

Listing A.26: Original script made by [4] for retrieving the Tweets

```
#!/usr/bin/env python

"""
Sample code of getting tweet JSON objects by tweet ID lists.

You have to install tweepy (This script was tested with
Python 2.6 and Tweepy 3.3.0)
https://github.com/tweepy/tweepy
and set its directory to your PYTHONPATH.

You have to obtain an access tokens from dev.twitter.com
with your Twitter account.
For more information, please follow:
https://dev.twitter.com/oauth/overview/application-owner
-access-tokens

Once you get the tokens, please fill the tokens in the
squotation marks in the
following "Access Information" part. For example, if your
consumer key is
LOVNhsAfB1zfPYnABCDE, you need to put it to Line 33
consumer_key = 'LOVNhsAfB1zfPYnABCDE'



"""
```

```
# call user.lookup api to query a list of user ids.
import tweepy
import sys
import json
import codecs
from tweepy.parsers import JSONParser

####### Access Information #################

# Parameter you need to specify
consumer_key = ''
consumer_secret = ''
access_key = ''
access_secret = ''

inputFile = 'tweet_id'
outputFile = 'tweet.json'

#############################################
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth_handler=auth, parser=JSONParser())

l=[];
with open(inputFile, 'r') as inFile:
        with codecs.open(outputFile, 'w', encoding='utf8')
        as outFile:
                for line in inFile.readlines():
                        l.append(line.rstrip());
                        if (len(l)>=99):
                                rst = api.statuses_lookup
                                (id_=l);
                                for tweet in rst:
                                        outFile.write(
                                        json.dumps(tweet)
                                        + "\n");
                                l=[];
                if (len(l) > 0):
                        rst = api.statuses_lookup(id_=l);
                        for tweet in rst:
                                outFile.write(json.dumps(
                                tweet) + "\n");
```

A.5.2 Modified script for retrieving the Tweets

Listing A.27: The script used for retrieving the Tweets

```
import tweepy
import sys
import json
import codecs
from tweepy.parsers import JSONParser
import pandas as pd
import numpy as np

fields = ["Tweet␣ID", "User␣ID", "Bullying_Traces", "Type",
"Form", "Teasing", "Author_Role", "Emotion"];
idata = pd.read_csv("data.csv", header=None, names=fields)
inputFile = 'data.csv'
outputFile = 'tweet.json'


idata["Tweet␣ID"]
idataWTweets=idata
idataWTweets["Tweets"]=np.nan
####### Access Information #################
#NB: Information has been censored,
#must be inserted in order to work
consumer_key = 'Censored'
consumer_secret = 'Censored'
access_key = 'Censored'
access_secret = 'Censored'


#############################################
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth_handler=auth, parser=JSONParser())
l=[];
with codecs.open(outputFile, 'w', encoding='utf8') as outFile:
        for row in idata["Tweet␣ID"]:
                l.append(row);
                if (len(l)>=99):
                        rst = api.statuses_lookup(id_=l);
                        for tweet in rst:
                                idataWTweets["Tweets"].loc[
                                idataWTweets['Tweet␣ID']
                                ==tweet["id"]]
                                =tweet["text"]
                        l=[];
        if (len(l) > 0):
                rst = api.statuses_lookup(id_=l);
```

```
                for tweet in rst:
                        idataWTweets["Tweets"].loc[
                        idataWTweets['Tweet␣ID']
                        ==tweet["id"]]=tweet["text"]
#Remove entries with no tweets
#(Dataset is some years old and some tweets therefore
#have been removed from Twitter)
CleanedidataWTweets=idataWTweets.dropna(subset=['Tweets'])
#Write to file:
CleanedidataWTweets.to_csv('TwitterDataCleanedWTweets.csv',
index=False)
```

A.5.3 Text preprocessing script

Listing A.28: The script used for preprocessing the raw text in the tweets

```
import re
import pandas as pd
import string
from nltk.tokenize.treebank import TreebankWordTokenizer,
TreebankWordDetokenizer
from nltk.corpus import stopwords
import preprocessor as p #Source:
#https://github.com/rrmina/preprocessor
nltk.download('stopwords')
detokenizer = TreebankWordDetokenizer()#Detokenizer module
tokenizer = TreebankWordTokenizer()#Tokenizer module
CommentFieldName = "Tweets"
Inputfile = "Twitter-W-userdata.csv"
idata = pd.read_csv(Inputfile)
idataTemp=idata.copy()
stopWords = set(stopwords.words())#Get list of stopwords
maxlen=0
#Loop the dataset:
for index, row in idata.iterrows():
    #Remove numbers:
    p.set_options(p.OPT.NUMBER)
    temp=p.clean(row[CommentFieldName])
    #Replace URLS with $URL$:
    p.set_options(p.OPT.URL)
    tempword=p.tokenize(temp)
    #Tokenize:
    words = tokenizer.tokenize(tempword)
    cleanedWords = []
    #Go through all the tokenz/words in each data sample
    for word in words:
```

```
        #Remove repeating characters:
        word=re.sub(r'(.)\1+', r'\1\1', word)
        cleanedWords.append(word)

    detected=0
    cleanedWords2 = []
    #Loop that removes the name after a @ sign
    for word in cleanedWords:
        if detected==1:
            detected=0
        else:
            cleanedWords2.append(word)
        if word=="@":
            detected=1
    #stitch the words back to a sentence/data sample
    words = detokenizer.detokenize(cleanedWords2)
    #Remove special signs:
    pattern=set('!"%&\()$*+,-./:;<=>?[\\]^_`{|}~')
    words = ''.join(ch for ch in words if ch not in pattern)
    #Tranform to lowercase:
    words=words.lower()
    #Insert back into the data set
    idataTemp.at[index, CommentFieldName] = words


idataTemp.to_csv("Twittter_cleaned.csv", index=False)
```

A.5.4 Full example of the testing script of one model

Listing A.29: The script used for preprocessing the raw text in the tweets

```
#Training and testing classifier (Bidirectional-LSTM-DNN)
#Preparation Part
#Modules needed:
from keras.callbacks import ModelCheckpoint
from keras import backend as K
from keras.models import Sequential
from keras.layers import Bidirectional, ConvLSTM2D,
Dense, Flatten, LSTM, Conv1D, MaxPooling1D,
Dropout, Activation, TimeDistributed
from keras.layers.embeddings import Embedding
from keras.initializers import Constant
from keras.utils import to_categorical
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config)

print("\x1B[3mWARNING:␣Training␣and␣testing␣will␣take␣quite␣a
bit␣of␣time,␣ETA:␣40␣min␣using␣GPU(1070TI)␣and␣CPU(8700k␣@
4,3Ghz)\x1B[23m")
print("Number␣of␣epochs␣set␣to:␣", EpochCount, "␣Input␣size
set␣to:␣",inputsize_, "␣Embedding␣size␣set␣to:␣", embsize_,
"␣Batch␣size␣set␣to:␣", batch_size_)
print('\x1B[3mNB:␣There␣may␣be␣some␣warning␣stating␣"F-score
is␣ill-defined",␣this␣is␣expected,␣this␣warning␣is␣caused
when␣the␣performance␣of␣poorly␣trained␣models␣are␣measured
and␣the␣model␣does␣not␣classify␣any␣samples␣as
positive\x1B[23m')

Models= [[] for _ in range(10)]
bestmodels=[]
part1="saved-model-"
part3="-.hdf5"
for ind,t in enumerate(Models):
    ModelsTemp=[]
    for i,x in enumerate(range(EpochCount)):
        i=i+1
        if i<10:
            part2="00"+str(i)
        if i<100 and i>9:
            part2="0"+str(i)
        if i>99:
            part2=str(i)
        modelname=part1+part2+part3
        ModelsTemp.append(modelname)
    Models[ind]=ModelsTemp
#10 cross fold testing:
#For loop to go through all the folds
for index, fold in enumerate(DataFolds):
    try:
        #Try to reset the model, will throw an error on first
        #run, therefore the try
        modelNTNU.reset_states()
    except:
        n=0#Do nothing, it's just because its the first run
    finally:
        #Create the model
        K.clear_session()
```

```
modelNTNU = Sequential()
modelNTNU.add(Embedding(vocabulary_size, embsize_,
input_length=inputsize_,
embeddings_initializer=Constant(embedding_matrix),
trainable=False))
modelNTNU.add(Conv1D(256,11, padding='same',
activation='relu', strides=1))
modelNTNU.add(MaxPooling1D(pool_size=2))
modelNTNU.add(LSTM(embsize_))
modelNTNU.add(Dense(vocabulary_size,
activation='relu'))
modelNTNU.add(Dense(2, activation='softmax'))
modelNTNU.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])

filepath = "saved-model-{epoch:03d}-.hdf5"
mcp = ModelCheckpoint(filepath, monitor='val_acc',
verbose=0,save_weights_only=True,
save_best_only=False, mode='max',period=1)
#fetch data:
TestSet=TraindataFolds[index]#Testing fold
#Empty array to hold the training set:
TrainSet= np.array([])
#Fill the trainset:
for index2, fold in enumerate(DataFolds):
    if index2 != index:
        #Not testing set so include in training:
        if TrainSet.size == 0:#First fold
            #set the training set equal to this fold
            TrainSet=TraindataFolds[index2]
        else:#the rest of the folds
            #Append the trainingfold to the
            #other trainingfolds:
            TrainSet =
            np.vstack([TrainSet,
            TraindataFolds[index2]])
#Prepare the classes to be fed into keras:
Trainclasses =
to_categorical(np.array(TrainSet[:,ClassCollumn]))
Testclasses =
to_categorical(np.array(TestSet[:,ClassCollumn]))

#Fit/train the model:
print("Running,␣Fold:␣", index+1, "␣of␣",
len(DataFolds))
```

```
        print("Training␣models␣in␣fold␣", index+1, "...")
        modelNTNU.fit(TrainSet[:,:inputsize_],
        Trainclasses, epochs=EpochCount,
        batch_size=batch_size_,
        validation_data=(TestSet[:,:inputsize_], Testclasses),
        shuffle=False, callbacks=[mcp], verbose=0)

        #Set the test fold to not oversampled fold:
        TestSet=DataFolds[index]

        #ID the best performing model found through
        #the training:
        print("Testing␣performance␣of␣all␣trained␣models␣in
␣␣␣␣␣␣␣␣fold␣", index+1, "...")
        from keras.models import load_model
        TestClassesList= TestSet[:,ClassCollumn].astype(int)
        #Reset measurements
        bestf1=-1
        bestmodel=""
        #Go through all the models from all the epochs
        for modelname in Models[1]:
            modelNTNU.load_weights(modelname)#Load the model
            #Use it to classify the test samples:
            predict =
            modelNTNU.predict_classes(TestSet[:,:inputsize_])
            #calculate F-measure:
            f1result=f1_score(TestClassesList,predict,
            average=None)
            #Test if F1 is higher than highest F1 registered:
            if f1result[1]>bestf1:
                bestf1=f1result[1]#Save results
                bestmodel=modelname#Save model
        bestmodels.append(bestmodel)
        #Measure the performance of the best model yielded
        #through the training:
        modelNTNU.load_weights(bestmodel)#Load the best model
        #Use it to classify the test samples:
        predict =
        modelNTNU.predict_classes(TestSet[:,:inputsize_])
        #Measure accuracy, precission, recall and F1:
        if index==0:#First fold
            accuracyresults =
            accuracy_score(TestClassesList,predict)
            Precisionresults=precision_score(TestClassesList,
            predict, average=None)
            Precisionresults_Macro=precision_score(
```

```
        TestClassesList,predict, average="macro")
        Precisionresults_Weighted=precision_score(
        TestClassesList,predict, average="weighted")
        recallresults=recall_score(
        TestClassesList,predict, average=None)
        recallresults_Macro=recall_score(
        TestClassesList,predict, average="macro")
        recallresults_Weighted=recall_score(
        TestClassesList,predict, average="weighted")
        f1results_Macro=f1_score(
        TestClassesList,predict, average="macro")
        f1results_Weighted=f1_score(
        TestClassesList,predict, average="weighted")
        f1results=f1_score(
        TestClassesList,predict, average=None)
    else:#Rest of the folds
        accuracyresults = np.vstack([accuracyresults,
        accuracy_score(TestClassesList,predict)])
        Precisionresults = np.vstack([Precisionresults,
        precision_score(TestClassesList,predict,
        average=None)])
        recallresults = np.vstack([recallresults,
        recall_score(TestClassesList,predict,
        average=None)])
        Precisionresults_Macro =
        np.vstack([Precisionresults_Macro,
        precision_score(TestClassesList,predict,
        average="macro")])
        recallresults_Macro =
        np.vstack([recallresults_Macro,recall_score(
        TestClassesList,predict, average="macro")])
        Precisionresults_Weighted =
        np.vstack([Precisionresults_Weighted,
        precision_score(
        TestClassesList,predict, average="weighted")])
        recallresults_Weighted =
        np.vstack([recallresults_Weighted,recall_score(
        TestClassesList,predict, average="weighted")])
        f1results_Macro =
        np.vstack([f1results_Macro,f1_score(
        TestClassesList,predict, average="macro")])
        f1results_Weighted =
        np.vstack([f1results_Weighted,f1_score(
        TestClassesList,predict, average="weighted")])
        f1results =
        np.vstack([f1results,f1_score(
```

```
                TestClassesList,predict, average=None)])

        print("Fold:␣", index+1, "␣is␣done,␣best␣model␣was:",
        bestmodel,"␣with␣an␣f1-score␣of:␣",
        f1_score(TestClassesList,predict, average="macro"))

#10 cross fold testing finished
print("All␣training␣and␣performance␣testing␣is␣done,
writing␣report...")

#script to create reprt:
#Average all the performance measures from each fold:
f1_Weighted_sum=0
f1_Weighted_avg=0
for f1_Weighted in f1results_Weighted:
    f1_Weighted_sum = f1_Weighted_sum+f1_Weighted
f1_Weighted_avg=f1_Weighted_sum/len(f1results_Weighted)

f1_Macro_sum=0
f1_Macro_avg=0
for f1_Macro in f1results_Macro:
    f1_Macro_sum = f1_Macro_sum+f1_Macro
f1_Macro_avg=f1_Macro_sum/len(f1results_Macro)

f1results
f1results1_sum=0
f1results1_avg=0
f1results2_sum=0
f1results2_avg=0
for f1 in f1results:
    f1results1_sum = f1results1_sum+f1[0]
    f1results2_sum = f1results2_sum+f1[1]
f1results1_avg=f1results1_sum/len(f1results)
f1results2_avg=f1results2_sum/len(f1results)


Precision1_sum=0
Precision1_avg=0
Precision2_sum=0
Precision2_avg=0
for Precision in Precisionresults:
    Precision1_sum = Precision1_sum+Precision[0]
    Precision2_sum = Precision2_sum+Precision[1]
Precision1_avg=Precision1_sum/len(Precisionresults)
Precision2_avg=Precision2_sum/len(Precisionresults)
```

```
Precision_Macro_sum=0
Precision_Macro_avg=0
for Precision_Macro in Precisionresults_Macro:
    Precision_Macro_sum = Precision_Macro_sum+Precision_Macro
Precision_Macro_avg=Precision_Macro_sum/len(
Precisionresults_Macro)

Precision_Weighted_sum=0
Precision_Weighted_avg=0
for Precision_Weighted in Precisionresults_Weighted:
    Precision_Weighted_sum =
    Precision_Weighted_sum+Precision_Weighted
Precision_Weighted_avg=Precision_Weighted_sum/len(
Precisionresults_Weighted)

accuracy_sum=0
accuracy_avg=0
for accuracy in accuracyresults:
    accuracy_sum = accuracy_sum+accuracy
accuracy_avg=accuracy_sum/len(accuracyresults)

recall1_sum=0
recall1_avg=0
recall2_sum=0
recall2_avg=0
for recall in recallresults:
    recall1_sum = recall1_sum+recall[0]
    recall2_sum = recall2_sum+recall[1]
recall1_avg=recall1_sum/len(recallresults)
recall2_avg=recall2_sum/len(recallresults)

recall_Weighted_sum=0
recall_Weighted_avg=0
for recall_Weighted in recallresults_Weighted:
    recall_Weighted_sum = recall_Weighted_sum+recall_Weighted
recall_Weighted_avg=recall_Weighted_sum/len(
recallresults_Weighted)

recall_Macro_sum=0
recall_Macro_avg=0
for recall_Macro in recallresults_Macro:
    recall_Macro_sum = recall_Macro_sum+recall_Macro
recall_Macro_avg=recall_Macro_sum/len(recallresults_Macro)
```

```
#Write the report to file:
line="-------------------------------------------------------
---------"
doubleline="==================================================
==============="
with open('Basemodell-v2-convlstm-report.txt','w') as fh:
    fh.write('Model␣overview:\n')
    fh.write('Batch␣size:'+str(batch_size_)+
    '\tNumber␣of␣epochs:␣'+str(EpochCount)+
    '\tEmbedding␣size:␣'+str(embsize_)+'␣\n')
    #Next line is from the source:
    #https://stackoverflow.com/questions/41665799/keras-model-
    #summary-object-to-string
    modelNTNU.summary(
    print_fn=lambda x: fh.write(x + '\n'))
    fh.write(doubleline+'\n')
    fh.write('\nModel␣Performance␣(10-Cross-Fold):\n')
    fh.write(line+'\n')
    fh.write('Accuracy␣average:␣\t\t\t\t\t\t\t\t\t\t'+
    str(round(accuracy_avg[0], 5))+'\n')
    fh.write(line+'\n')
    fh.write('F1␣score␣average␣for␣None-class:␣\t\t\t\t\t\t'+
    str(round(f1results1_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('F1␣score␣average␣for␣Bully-class:␣\t\t\t\t\t\t'+
    str(round(f1results2_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('F1␣score␣weighted␣average:␣\t\t\t\t\t\t\t\t'+
    str(round(f1_Weighted_avg[0], 5))+'\n')
    fh.write(line+'\n')
    fh.write('F1␣score␣macro␣average:␣\t\t\t\t\t\t\t\t'
    +str(round(f1_Macro_avg[0], 5))+'\n')
    fh.write(line+'\n')
    fh.write('Precision␣average␣for␣None-class:␣\t\t\t\t\t\t'
    +str(round(Precision1_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('Precision␣average␣for␣Bully-class:␣\t\t\t\t\t\t'
    +str(round(Precision2_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('Precision␣weighted␣average:␣\t\t\t\t\t\t\t'
    +str(round(Precision_Weighted_avg[0], 5))+'\n')
    fh.write(line+'\n')
    fh.write('Precision␣score␣macro␣average:␣\t\t\t\t\t\t\t'
    +str(round(Precision_Macro_avg[0], 5))+'\n')
    fh.write(line+'\n')
```

```
    fh.write('Recall␣average␣for␣None-class:␣\t\t\t\t\t\t\t'
    +str(round(recall1_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('Recall␣average␣for␣Bully-class:␣\t\t\t\t\t\t'
    +str(round(recall2_avg, 5))+'\n')
    fh.write(line+'\n')
    fh.write('Recall␣weighted␣average:␣\t\t\t\t\t\t\t\t'
    +str(round(recall_Weighted_avg[0], 5))+'\n')
    fh.write(line+'\n')
    fh.write('Recall␣score␣macro␣average:␣\t\t\t\t\t\t\t'
    +str(round(recall_Macro_avg[0], 5))+'\n')
    fh.write(doubleline+'\n')
    for indexes, r in enumerate(bestmodels):
        fold=indexes+1
        fh.write('The␣best␣model␣in␣fold␣'
        +str(fold)+':␣'+r+'␣\n')

print("Report␣is␣written,␣and␣script␣is␣finished.")
```

B Performance Reports Experiment 1, first test run with 200 Epochs

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               365600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,554,692
Trainable params: 3,259,092
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 22: Part one of the performance report for ConvLSTM (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                        0.85734
_________________________________________________________________
F1 score average for None-class:                         0.89882
_________________________________________________________________
F1 score average for Bully-class:                        0.75636
_________________________________________________________________
F1 score weighted average:                               0.86171
_________________________________________________________________
F1 score macro average:                                  0.82759
_________________________________________________________________
Precision average for None-class:                        0.94074
_________________________________________________________________
Precision average for Bully-class:                       0.68833
_________________________________________________________________
Precision weighted average:                              0.87516
_________________________________________________________________
Precision score macro average:                           0.81453
_________________________________________________________________
Recall average for None-class:                           0.86181
_________________________________________________________________
Recall average for Bully-class:                          0.84543
_________________________________________________________________
Recall weighted average:                                 0.85734
_________________________________________________________________
Recall score macro average:                              0.85362
=================================================================
```

Figure 23: Part two of the performance report for ConvLSTM (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-013-.hdf5
The best model in fold 2: saved-model-022-.hdf5
The best model in fold 3: saved-model-018-.hdf5
The best model in fold 4: saved-model-076-.hdf5
The best model in fold 5: saved-model-015-.hdf5
The best model in fold 6: saved-model-017-.hdf5
The best model in fold 7: saved-model-021-.hdf5
The best model in fold 8: saved-model-158-.hdf5
The best model in fold 9: saved-model-029-.hdf5
The best model in fold 10: saved-model-033-.hdf5
Avg epochs: 40,2
```

Figure 24: Part three of the performance report for ConvLSTM (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
________________________________________________________________
================================================================
```

Figure 25: Part one of the performance report for ConvLSTM x2 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.861
F1 score average for None-class:	0.90147
F1 score average for Bully-class:	0.76311
F1 score weighted average:	0.86547
F1 score macro average:	0.83229
Precision average for None-class:	0.94518
Precision average for Bully-class:	0.68867
Precision weighted average:	0.87848
Precision score macro average:	0.81692
Recall average for None-class:	0.86209
Recall average for Bully-class:	0.85812
Recall weighted average:	0.861
Recall score macro average:	0.86011

```
=================================================================
```

Figure 26: Part two of the performance report for ConvLSTM x2 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-020-.hdf5
The best model in fold 2: saved-model-028-.hdf5
The best model in fold 3: saved-model-011-.hdf5
The best model in fold 4: saved-model-048-.hdf5
The best model in fold 5: saved-model-017-.hdf5
The best model in fold 6: saved-model-038-.hdf5
The best model in fold 7: saved-model-041-.hdf5
The best model in fold 8: saved-model-050-.hdf5
The best model in fold 9: saved-model-068-.hdf5
The best model in fold 10: saved-model-044-.hdf5
Avg epochs: 36,5
```

Figure 27: Part three of the performance report for ConvLSTM x2 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 8, 200)            263200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 8, 64)             89664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               212000
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,350,084
Trainable params: 4,054,484
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 28: Part one of the performance report for ConvLSTM x3 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.86353
F1 score average for None-class:	0.90367
F1 score average for Bully-class:	0.76431
F1 score weighted average:	0.86746
F1 score macro average:	0.83399
Precision average for None-class:	0.94359
Precision average for Bully-class:	0.69525
Precision weighted average:	0.87881
Precision score macro average:	0.81942
Recall average for None-class:	0.86751
Recall average for Bully-class:	0.85117
Recall weighted average:	0.86353
Recall score macro average:	0.85934

Figure 29: Part two of the performance report for ConvLSTM x3 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-034-.hdf5
The best model in fold 2: saved-model-026-.hdf5
The best model in fold 3: saved-model-028-.hdf5
The best model in fold 4: saved-model-047-.hdf5
The best model in fold 5: saved-model-056-.hdf5
The best model in fold 6: saved-model-059-.hdf5
The best model in fold 7: saved-model-038-.hdf5
The best model in fold 8: saved-model-066-.hdf5
The best model in fold 9: saved-model-034-.hdf5
The best model in fold 10: saved-model-030-.hdf5
Avg epochs: 41,8
```

Figure 30: Part three of the performance report for ConvLSTM x3 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 8, 200)            263200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 8, 64)             89664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 4, 200)            212000
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 4, 32)             32032
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2, 32)             0
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               186400
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,568,516
Trainable params: 4,272,916
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 31: Part one of the performance report for ConvLSTM x4 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
________________________________________________________________
Accuracy average:                                        0.86101
________________________________________________________________
F1 score average for None-class:                         0.90229
________________________________________________________________
F1 score average for Bully-class:                        0.75786
________________________________________________________________
F1 score weighted average:                               0.8647
________________________________________________________________
F1 score macro average:                                  0.83008
________________________________________________________________
Precision average for None-class:                        0.93779
________________________________________________________________
Precision average for Bully-class:                       0.69615
________________________________________________________________
Precision weighted average:                              0.87492
________________________________________________________________
Precision score macro average:                           0.81697
________________________________________________________________
Recall average for None-class:                           0.8701
________________________________________________________________
Recall average for Bully-class:                          0.83548
________________________________________________________________
Recall weighted average:                                 0.86101
________________________________________________________________
Recall score macro average:                              0.85279
================================================================
```

Figure 32: Part two of the performance report for ConvLSTM x4 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-022-.hdf5
The best model in fold 2: saved-model-032-.hdf5
The best model in fold 3: saved-model-028-.hdf5
The best model in fold 4: saved-model-045-.hdf5
The best model in fold 5: saved-model-031-.hdf5
The best model in fold 6: saved-model-033-.hdf5
The best model in fold 7: saved-model-055-.hdf5
The best model in fold 8: saved-model-057-.hdf5
The best model in fold 9: saved-model-042-.hdf5
The best model in fold 10: saved-model-051-.hdf5
Avg epochs: 39,7
```

Figure 33: Part three of the performance report for ConvLSTM x4 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 8, 200)            263200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 8, 64)             89664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 4, 200)            212000
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 4, 32)             32032
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2, 32)             0
_________________________________________________________________
lstm_4 (LSTM)                (None, 2, 200)            186400
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 2, 16)             12816
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 1, 16)             0
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               173600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,754,932
Trainable params: 4,459,332
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 34: Part one of the performance report for ConvLSTM x5 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.86077
F1 score average for None-class:	0.90139
F1 score average for Bully-class:	0.76148
F1 score weighted average:	0.86499
F1 score macro average:	0.83144
Precision average for None-class:	0.94304
Precision average for Bully-class:	0.68988
Precision weighted average:	0.87688
Precision score macro average:	0.81646
Recall average for None-class:	0.86364
Recall average for Bully-class:	0.85131
Recall weighted average:	0.86077
Recall score macro average:	0.85748

Figure 35: Part two of the performance report for ConvLSTM x5 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-062-.hdf5
The best model in fold 2: saved-model-025-.hdf5
The best model in fold 3: saved-model-048-.hdf5
The best model in fold 4: saved-model-044-.hdf5
The best model in fold 5: saved-model-112-.hdf5
The best model in fold 6: saved-model-034-.hdf5
The best model in fold 7: saved-model-050-.hdf5
The best model in fold 8: saved-model-045-.hdf5
The best model in fold 9: saved-model-046-.hdf5
The best model in fold 10: saved-model-071-.hdf5
Avg epochs: 53,7
```

Figure 36: Part three of the performance report for ConvLSTM x5 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               480800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,748,036
Trainable params: 3,452,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 37: Part one of the performance report for the model: BLSTM LSTM (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.8578
_________________________________________________________________
F1 score average for None-class:                          0.89865
_________________________________________________________________
F1 score average for Bully-class:                         0.76047
_________________________________________________________________
F1 score weighted average:                                0.86276
_________________________________________________________________
F1 score macro average:                                   0.82956
_________________________________________________________________
Precision average for None-class:                         0.94816
_________________________________________________________________
Precision average for Bully-class:                        0.6797
_________________________________________________________________
Precision weighted average:                               0.87829
_________________________________________________________________
Precision score macro average:                            0.81393
_________________________________________________________________
Recall average for None-class:                            0.85465
_________________________________________________________________
Recall average for Bully-class:                           0.86632
_________________________________________________________________
Recall weighted average:                                  0.8578
_________________________________________________________________
Recall score macro average:                               0.86049
=================================================================
```

Figure 38: Part two of the performance report for the model: BLSTM LSTM (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-052-.hdf5
The best model in fold 2: saved-model-050-.hdf5
The best model in fold 3: saved-model-038-.hdf5
The best model in fold 4: saved-model-038-.hdf5
The best model in fold 5: saved-model-039-.hdf5
The best model in fold 6: saved-model-039-.hdf5
The best model in fold 7: saved-model-061-.hdf5
The best model in fold 8: saved-model-060-.hdf5
The best model in fold 9: saved-model-042-.hdf5
The best model in fold 10: saved-model-045-.hdf5
Avg epochs: 46,4
```

Figure 39: Part three of the performance report for the model: BLSTM LSTM (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,068,836
Trainable params: 3,773,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 40: Part one of the performance report for the model: BLSTM LSTM x2 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.86031
F1 score average for None-class:	0.90134
F1 score average for Bully-class:	0.75999
F1 score weighted average:	0.86452
F1 score macro average:	0.83066
Precision average for None-class:	0.94047
Precision average for Bully-class:	0.69204
Precision weighted average:	0.87595
Precision score macro average:	0.81626
Recall average for None-class:	0.86584
Recall average for Bully-class:	0.84547
Recall weighted average:	0.86031
Recall score macro average:	0.85566

Figure 41: Part two of the performance report for the model: BLSTM LSTM x2 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-170-.hdf5
The best model in fold 2: saved-model-042-.hdf5
The best model in fold 3: saved-model-058-.hdf5
The best model in fold 4: saved-model-060-.hdf5
The best model in fold 5: saved-model-053-.hdf5
The best model in fold 6: saved-model-108-.hdf5
The best model in fold 7: saved-model-093-.hdf5
The best model in fold 8: saved-model-064-.hdf5
The best model in fold 9: saved-model-074-.hdf5
The best model in fold 10: saved-model-042-.hdf5
Avg epochs: 76,4
```

Figure 42: Part three of the performance report for the model: BLSTM LSTM x2 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,389,636
Trainable params: 4,094,036
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 43: Part one of the performance report for the model: BLSTM LSTM x3 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
```

Accuracy average:	0.8539
F1 score average for None-class:	0.8957
F1 score average for Bully-class:	0.75425
F1 score weighted average:	0.85895
F1 score macro average:	0.82497
Precision average for None-class:	0.94495
Precision average for Bully-class:	0.67441
Precision weighted average:	0.87426
Precision score macro average:	0.80968
Recall average for None-class:	0.8519
Recall average for Bully-class:	0.85783
Recall weighted average:	0.8539
Recall score macro average:	0.85487

```
================================================================
```

Figure 44: Part two of the performance report for the model: BLSTM LSTM x3 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-093-.hdf5
The best model in fold 2: saved-model-040-.hdf5
The best model in fold 3: saved-model-076-.hdf5
The best model in fold 4: saved-model-075-.hdf5
The best model in fold 5: saved-model-099-.hdf5
The best model in fold 6: saved-model-115-.hdf5
The best model in fold 7: saved-model-061-.hdf5
The best model in fold 8: saved-model-142-.hdf5
The best model in fold 9: saved-model-092-.hdf5
The best model in fold 10: saved-model-069-.hdf5
Avg epochs: 86,2
```

Figure 45: Part three of the performance report for the model: BLSTM LSTM x3 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,710,436
Trainable params: 4,414,836
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 46: Part one of the performance report for the model: BLSTM LSTM x4 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.85231
F1 score average for None-class:	0.89357
F1 score average for Bully-class:	0.75697
F1 score weighted average:	0.85804
F1 score macro average:	0.82527
Precision average for None-class:	0.95119
Precision average for Bully-class:	0.66762
Precision weighted average:	0.8772
Precision score macro average:	0.8094
Recall average for None-class:	0.84328
Recall average for Bully-class:	0.87711
Recall weighted average:	0.85231
Recall score macro average:	0.8602

Figure 47: Part two of the performance report for the model: BLSTM LSTM x4 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-003-.hdf5
The best model in fold 2: saved-model-001-.hdf5
The best model in fold 3: saved-model-002-.hdf5
The best model in fold 4: saved-model-001-.hdf5
The best model in fold 5: saved-model-001-.hdf5
The best model in fold 6: saved-model-001-.hdf5
The best model in fold 7: saved-model-001-.hdf5
The best model in fold 8: saved-model-001-.hdf5
The best model in fold 9: saved-model-002-.hdf5
The best model in fold 10: saved-model-002-.hdf5
Avg epochs: 1,5
```

Figure 48: Part three of the performance report for the model: BLSTM LSTM x5 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_6 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 7,031,236
Trainable params: 4,735,636
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 49: Part one of the performance report for the model: BLSTM LSTM x5 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                    0.26105
F1 score average for None-class:                     0.0
F1 score average for Bully-class:                    0.41382
F1 score weighted average:                           0.10828
F1 score macro average:                              0.20691
Precision average for None-class:                    0.0
Precision average for Bully-class:                   0.26105
Precision weighted average:                          0.06835
Precision score macro average:                       0.13052
Recall average for None-class:                       0.0
Recall average for Bully-class:                      1.0
Recall weighted average:                             0.26105
Recall score macro average:                          0.5
================================================================
```

Figure 50: Part two of the performance report for the model: BLSTM LSTM x5 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-003-.hdf5
The best model in fold 2: saved-model-001-.hdf5
The best model in fold 3: saved-model-002-.hdf5
The best model in fold 4: saved-model-001-.hdf5
The best model in fold 5: saved-model-001-.hdf5
The best model in fold 6: saved-model-001-.hdf5
The best model in fold 7: saved-model-001-.hdf5
The best model in fold 8: saved-model-001-.hdf5
The best model in fold 9: saved-model-002-.hdf5
The best model in fold 10: saved-model-002-.hdf5
Avg epochs: 1,5
```

Figure 51: Part three of the performance report for the model: BLSTM LSTM x5 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               641600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 7,562,836
Trainable params: 5,267,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 52: Part one of the performance report for the model: BLSTM (experiment 1, first test run with 200 epochs)

```
=================================================================

Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85414
_________________________________________________________________
F1 score average for None-class:                          0.89576
_________________________________________________________________
F1 score average for Bully-class:                         0.75632
_________________________________________________________________
F1 score weighted average:                                0.85949
_________________________________________________________________
F1 score macro average:                                   0.82604
_________________________________________________________________
Precision average for None-class:                         0.94788
_________________________________________________________________
Precision average for Bully-class:                        0.67107
_________________________________________________________________
Precision weighted average:                               0.87585
_________________________________________________________________
Precision score macro average:                            0.80947
_________________________________________________________________
Recall average for None-class:                            0.84936
_________________________________________________________________
Recall average for Bully-class:                           0.8678
_________________________________________________________________
Recall weighted average:                                  0.85414
_________________________________________________________________
Recall score macro average:                               0.85858
=================================================================
```

Figure 53: Part two of the performance report for the model: BLSTM (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-028-.hdf5
The best model in fold 2: saved-model-024-.hdf5
The best model in fold 3: saved-model-036-.hdf5
The best model in fold 4: saved-model-015-.hdf5
The best model in fold 5: saved-model-012-.hdf5
The best model in fold 6: saved-model-029-.hdf5
The best model in fold 7: saved-model-019-.hdf5
The best model in fold 8: saved-model-026-.hdf5
The best model in fold 9: saved-model-022-.hdf5
The best model in fold 10: saved-model-027-.hdf5
Avg epochs: 23,8
```

Figure 54: Part three of the performance report for the model: BLSTM (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 55: Part one of the performance report for the model: BLSTM x2 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.85598
F1 score average for None-class:	0.8972
F1 score average for Bully-class:	0.75786
F1 score weighted average:	0.86098
F1 score macro average:	0.82753
Precision average for None-class:	0.94644
Precision average for Bully-class:	0.6764
Precision weighted average:	0.87589
Precision score macro average:	0.81142
Recall average for None-class:	0.85312
Recall average for Bully-class:	0.86256
Recall weighted average:	0.85598
Recall score macro average:	0.85784

Figure 56: Part two of the performance report for the model: BLSTM x2 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-064-.hdf5
The best model in fold 2: saved-model-137-.hdf5
The best model in fold 3: saved-model-047-.hdf5
The best model in fold 4: saved-model-049-.hdf5
The best model in fold 5: saved-model-044-.hdf5
The best model in fold 6: saved-model-060-.hdf5
The best model in fold 7: saved-model-035-.hdf5
The best model in fold 8: saved-model-062-.hdf5
The best model in fold 9: saved-model-039-.hdf5
The best model in fold 10: saved-model-062-.hdf5
Avg epochs: 59,9
```

Figure 57: Part three of the performance report for the model: BLSTM x2 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_3 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 9,486,036
Trainable params: 7,190,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 58: Part one of the performance report for the model: BLSTM x3 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                          0.85895
_________________________________________________________________
F1 score average for None-class:                           0.89949
_________________________________________________________________
F1 score average for Bully-class:                          0.76144
_________________________________________________________________
F1 score weighted average:                                 0.86367
_________________________________________________________________
F1 score macro average:                                    0.83046
_________________________________________________________________
Precision average for None-class:                          0.94674
_________________________________________________________________
Precision average for Bully-class:                         0.68488
_________________________________________________________________
Precision weighted average:                                0.87839
_________________________________________________________________
Precision score macro average:                             0.81581
_________________________________________________________________
Recall average for None-class:                             0.85757
_________________________________________________________________
Recall average for Bully-class:                            0.86111
_________________________________________________________________
Recall weighted average:                                   0.85895
_________________________________________________________________
Recall score macro average:                                0.85934
=================================================================
```

Figure 59: Part two of the performance report for the model: BLSTM x3 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-054-.hdf5
The best model in fold 2: saved-model-058-.hdf5
The best model in fold 3: saved-model-031-.hdf5
The best model in fold 4: saved-model-084-.hdf5
The best model in fold 5: saved-model-061-.hdf5
The best model in fold 6: saved-model-056-.hdf5
The best model in fold 7: saved-model-099-.hdf5
The best model in fold 8: saved-model-065-.hdf5
The best model in fold 9: saved-model-068-.hdf5
The best model in fold 10: saved-model-045-.hdf5
Avg epochs: 62,1
```

Figure 60: Part three of the performance report for the model: BLSTM x3 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 4,946,436
Trainable params: 2,650,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 61: Part one of the performance report for the model: LSTM (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                         0.85504
F1 score average for None-class:                          0.89638
F1 score average for Bully-class:                         0.75712
F1 score weighted average:                                0.86015
F1 score macro average:                                   0.82675
Precision average for None-class:                         0.94601
Precision average for Bully-class:                        0.6782
Precision weighted average:                               0.87638
Precision score macro average:                            0.8121
Recall average for None-class:                            0.85279
Recall average for Bully-class:                           0.86175
Recall weighted average:                                  0.85504
Recall score macro average:                               0.85727
=================================================================
```

Figure 62: Part two of the performance report for the model: LSTM (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-027-.hdf5
The best model in fold 2: saved-model-029-.hdf5
The best model in fold 3: saved-model-014-.hdf5
The best model in fold 4: saved-model-017-.hdf5
The best model in fold 5: saved-model-024-.hdf5
The best model in fold 6: saved-model-043-.hdf5
The best model in fold 7: saved-model-028-.hdf5
The best model in fold 8: saved-model-050-.hdf5
The best model in fold 9: saved-model-019-.hdf5
The best model in fold 10: saved-model-019-.hdf5

Avg epochs: 27
```

Figure 63: Part three of the performance report for the model: LSTM (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,267,236
Trainable params: 2,971,636
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 64: Part one of the performance report for the model: LSTM x2 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                          0.8594
_________________________________________________________________
F1 score average for None-class:                           0.8992
_________________________________________________________________
F1 score average for Bully-class:                          0.76597
_________________________________________________________________
F1 score weighted average:                                 0.86455
_________________________________________________________________
F1 score macro average:                                    0.83258
_________________________________________________________________
Precision average for None-class:                          0.9521
_________________________________________________________________
Precision average for Bully-class:                         0.68137
_________________________________________________________________
Precision weighted average:                                0.88149
_________________________________________________________________
Precision score macro average:                             0.81673
_________________________________________________________________
Recall average for None-class:                             0.85263
_________________________________________________________________
Recall average for Bully-class:                            0.87792
_________________________________________________________________
Recall weighted average:                                   0.8594
_________________________________________________________________
Recall score macro average:                                0.86528
=================================================================
```

Figure 65: Part two of the performance report for the model: LSTM x2 (experiment 1, first test run with 200 epochs)

```
================================================================
The best model in fold 1: saved-model-053-.hdf5
The best model in fold 2: saved-model-093-.hdf5
The best model in fold 3: saved-model-055-.hdf5
The best model in fold 4: saved-model-115-.hdf5
The best model in fold 5: saved-model-037-.hdf5
The best model in fold 6: saved-model-040-.hdf5
The best model in fold 7: saved-model-086-.hdf5
The best model in fold 8: saved-model-171-.hdf5
The best model in fold 9: saved-model-051-.hdf5
The best model in fold 10: saved-model-110-.hdf5
Avg epochs: 81,1
```

Figure 66: Part three of the performance report for the model: LSTM x2 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
================================================================
Total params: 5,588,036
Trainable params: 3,292,436
Non-trainable params: 2,295,600
________________________________________________________________
================================================================
```

Figure 67: Part one of the performance report for the model: LSTM x3 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.85848
F1 score average for None-class:	0.89928
F1 score average for Bully-class:	0.76025
F1 score weighted average:	0.86316
F1 score macro average:	0.82977
Precision average for None-class:	0.94625
Precision average for Bully-class:	0.68369
Precision weighted average:	0.87784
Precision score macro average:	0.81497
Recall average for None-class:	0.8576
Recall average for Bully-class:	0.86015
Recall weighted average:	0.85848
Recall score macro average:	0.85887

Figure 68: Part two of the performance report for the model: LSTM x3 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-074-.hdf5
The best model in fold 2: saved-model-058-.hdf5
The best model in fold 3: saved-model-066-.hdf5
The best model in fold 4: saved-model-129-.hdf5
The best model in fold 5: saved-model-067-.hdf5
The best model in fold 6: saved-model-098-.hdf5
The best model in fold 7: saved-model-065-.hdf5
The best model in fold 8: saved-model-062-.hdf5
The best model in fold 9: saved-model-064-.hdf5
The best model in fold 10: saved-model-086-.hdf5
Avg epochs: 76,9
```

Figure 69: Part three of the performance report for the model: LSTM x3 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 70: Part one of the performance report for the model: LSTM x4 (experiment 1, first test run with 200 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                          0.85666
_________________________________________________________________
F1 score average for None-class:                           0.89771
_________________________________________________________________
F1 score average for Bully-class:                          0.75833
_________________________________________________________________
F1 score weighted average:                                 0.86156
_________________________________________________________________
F1 score macro average:                                    0.82802
_________________________________________________________________
Precision average for None-class:                          0.94686
_________________________________________________________________
Precision average for Bully-class:                         0.6786
_________________________________________________________________
Precision weighted average:                                0.8767
_________________________________________________________________
Precision score macro average:                             0.81273
_________________________________________________________________
Recall average for None-class:                             0.85403
_________________________________________________________________
Recall average for Bully-class:                            0.8616
_________________________________________________________________
Recall weighted average:                                   0.85666
_________________________________________________________________
Recall score macro average:                                0.85782
=================================================================
```

Figure 71: Part two of the performance report for the model: LSTM x4 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-097-.hdf5
The best model in fold 2: saved-model-093-.hdf5
The best model in fold 3: saved-model-079-.hdf5
The best model in fold 4: saved-model-080-.hdf5
The best model in fold 5: saved-model-078-.hdf5
The best model in fold 6: saved-model-117-.hdf5
The best model in fold 7: saved-model-084-.hdf5
The best model in fold 8: saved-model-127-.hdf5
The best model in fold 9: saved-model-087-.hdf5
The best model in fold 10: saved-model-093-.hdf5
Avg epochs: 93,5
```

Figure 72: Part three of the performance report for the model: LSTM x4 (experiment 1, first test run with 200 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 200   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,229,636
Trainable params: 3,934,036
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 73: Part one of the performance report for the model: LSTM x5 (experiment 1, first test run with 200 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.8539
F1 score average for None-class:	0.89574
F1 score average for Bully-class:	0.75542
F1 score weighted average:	0.85915
F1 score macro average:	0.82558
Precision average for None-class:	0.94652
Precision average for Bully-class:	0.6729
Precision weighted average:	0.87554
Precision score macro average:	0.80971
Recall average for None-class:	0.85075
Recall average for Bully-class:	0.86459
Recall weighted average:	0.8539
Recall score macro average:	0.85767

Figure 74: Part two of the performance report for the model: LSTM x5 (experiment 1, first test run with 200 epochs)

```
=================================================================
The best model in fold 1: saved-model-130-.hdf5
The best model in fold 2: saved-model-136-.hdf5
The best model in fold 3: saved-model-152-.hdf5
The best model in fold 4: saved-model-178-.hdf5
The best model in fold 5: saved-model-196-.hdf5
The best model in fold 6: saved-model-180-.hdf5
The best model in fold 7: saved-model-171-.hdf5
The best model in fold 8: saved-model-157-.hdf5
The best model in fold 9: saved-model-154-.hdf5
The best model in fold 10: saved-model-126-.hdf5
Avg epochs: 158
```

Figure 75: Part three of the performance report, from the first test run with 200 epochs, for the model: LSTM x5 (experiment 1, first test run with 200 epochs)

C Performance Reports from Experiment 1, second test run with 800 Epochs

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               365600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,554,692
Trainable params: 3,259,092
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 76: Part one of the performance report for the model: ConvLSTM (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.86077
_________________________________________________________________
F1 score average for None-class:                          0.90088
_________________________________________________________________
F1 score average for Bully-class:                         0.76445
_________________________________________________________________
F1 score weighted average:                                0.86534
_________________________________________________________________
F1 score macro average:                                   0.83267
_________________________________________________________________
Precision average for None-class:                         0.94664
_________________________________________________________________
Precision average for Bully-class:                        0.69039
_________________________________________________________________
Precision weighted average:                               0.88004
_________________________________________________________________
Precision score macro average:                            0.81852
_________________________________________________________________
Recall average for None-class:                            0.86053
_________________________________________________________________
Recall average for Bully-class:                           0.86225
_________________________________________________________________
Recall weighted average:                                  0.86077
_________________________________________________________________
Recall score macro average:                               0.86139
=================================================================
```

Figure 77: Part two of the performance report for the model: ConvLSTM (experiment 2, second test run with 800 epochs)

```
================================================================
The best model in fold 1: saved-model-013-.hdf5
The best model in fold 2: saved-model-029-.hdf5
The best model in fold 3: saved-model-025-.hdf5
The best model in fold 4: saved-model-017-.hdf5
The best model in fold 5: saved-model-017-.hdf5
The best model in fold 6: saved-model-016-.hdf5
The best model in fold 7: saved-model-224-.hdf5
The best model in fold 8: saved-model-031-.hdf5
The best model in fold 9: saved-model-025-.hdf5
The best model in fold 10: saved-model-036-.hdf5
```

Figure 78: Part three of the performance report for the model: ConvLSTM (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 79: Part one of the performance report for the model: ConvLSTM x2 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                     0.86719
F1 score average for None-class:                      0.90647
F1 score average for Bully-class:                     0.76969
F1 score weighted average:                            0.87087
F1 score macro average:                               0.83808
Precision average for None-class:                     0.94312
Precision average for Bully-class:                    0.70526
Precision weighted average:                           0.8812
Precision score macro average:                        0.82419
Recall average for None-class:                        0.87319
Recall average for Bully-class:                       0.8502
Recall weighted average:                              0.86719
Recall score macro average:                           0.86169
=================================================================
```

Figure 80: Part two of the performance report for the model: ConvLSTM x2 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-024-.hdf5
The best model in fold 2: saved-model-694-.hdf5
The best model in fold 3: saved-model-023-.hdf5
The best model in fold 4: saved-model-641-.hdf5
The best model in fold 5: saved-model-672-.hdf5
The best model in fold 6: saved-model-674-.hdf5
The best model in fold 7: saved-model-294-.hdf5
The best model in fold 8: saved-model-034-.hdf5
The best model in fold 9: saved-model-046-.hdf5
The best model in fold 10: saved-model-026-.hdf5
```

Figure 81: Part three of the performance report for the model: ConvLSTM x2 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200

Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600

conv1d_1 (Conv1D)            (None, 34, 256)           563456

max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0

lstm_1 (LSTM)                (None, 17, 200)           365600

conv1d_2 (Conv1D)            (None, 17, 128)           230528

max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0

lstm_2 (LSTM)                (None, 8, 200)            263200

conv1d_3 (Conv1D)            (None, 8, 64)             89664

max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0

lstm_3 (LSTM)                (None, 200)               212000

dense_1 (Dense)              (None, 11478)             2307078

dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,350,084
Trainable params: 4,054,484
Non-trainable params: 2,295,600

=================================================================
```

Figure 82: Part one of the performance report for the model: ConvLSTM x3 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                  0.86171
F1 score average for None-class:                   0.90206
F1 score average for Bully-class:                  0.7618
F1 score weighted average:                         0.86561
F1 score macro average:                            0.83193
Precision average for None-class:                  0.94274
Precision average for Bully-class:                 0.69645
Precision weighted average:                        0.8784
Precision score macro average:                     0.8196
Recall average for None-class:                     0.86617
Recall average for Bully-class:                    0.84751
Recall weighted average:                           0.86171
Recall score macro average:                        0.85684
=================================================================
```

Figure 83: Part two of the performance report for the model: ConvLSTM x3 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-023-.hdf5
The best model in fold 2: saved-model-301-.hdf5
The best model in fold 3: saved-model-034-.hdf5
The best model in fold 4: saved-model-075-.hdf5
The best model in fold 5: saved-model-070-.hdf5
The best model in fold 6: saved-model-027-.hdf5
The best model in fold 7: saved-model-066-.hdf5
The best model in fold 8: saved-model-274-.hdf5
The best model in fold 9: saved-model-029-.hdf5
The best model in fold 10: saved-model-066-.hdf5
```

Figure 84: Part three of the performance report for the model: ConvLSTM x3 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 8, 200)            263200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 8, 64)             89664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 4, 200)            212000
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 4, 32)             32032
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2, 32)             0
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               186400
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,568,516
Trainable params: 4,272,916
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 85: Part one of the performance report for the model: ConvLSTM x4 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                  0.86466
F1 score average for None-class:                   0.90409
F1 score average for Bully-class:                  0.76767
F1 score weighted average:                         0.86866
F1 score macro average:                            0.83588
Precision average for None-class:                  0.94553
Precision average for Bully-class:                 0.69846
Precision weighted average:                        0.88088
Precision score macro average:                     0.822
Recall average for None-class:                     0.867
Recall average for Bully-class:                    0.85591
Recall weighted average:                           0.86466
Recall score macro average:                        0.86145
=================================================================
```

Figure 86: Part two of the performance report for the model: ConvLSTM x4 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-027-.hdf5
The best model in fold 2: saved-model-039-.hdf5
The best model in fold 3: saved-model-061-.hdf5
The best model in fold 4: saved-model-040-.hdf5
The best model in fold 5: saved-model-031-.hdf5
The best model in fold 6: saved-model-073-.hdf5
The best model in fold 7: saved-model-085-.hdf5
The best model in fold 8: saved-model-044-.hdf5
The best model in fold 9: saved-model-041-.hdf5
The best model in fold 10: saved-model-185-.hdf5
```

Figure 87: Part three of the performance report for the model: ConvLSTM x4 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 8, 200)            263200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 8, 64)             89664
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 4, 200)            212000
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 4, 32)             32032
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2, 32)             0
_________________________________________________________________
lstm_4 (LSTM)                (None, 2, 200)            186400
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 2, 16)             12816
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 1, 16)             0
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               173600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,754,932
Trainable params: 4,459,332
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 88: Part one of the performance report for the model: ConvLSTM x5 (experiment 2, second test run with 800 epochs)

Model Performance (10-Cross-Fold):

Accuracy average:	0.86375
F1 score average for None-class:	0.9031
F1 score average for Bully-class:	0.76901
F1 score weighted average:	0.86823
F1 score macro average:	0.83606
Precision average for None-class:	0.94935
Precision average for Bully-class:	0.69087
Precision weighted average:	0.88185
Precision score macro average:	0.82011
Recall average for None-class:	0.86153
Recall average for Bully-class:	0.86883
Recall weighted average:	0.86375
Recall score macro average:	0.86518

Figure 89: Part two of the performance report for the model: ConvLSTM x5 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-042-.hdf5
The best model in fold 2: saved-model-069-.hdf5
The best model in fold 3: saved-model-049-.hdf5
The best model in fold 4: saved-model-094-.hdf5
The best model in fold 5: saved-model-056-.hdf5
The best model in fold 6: saved-model-033-.hdf5
The best model in fold 7: saved-model-328-.hdf5
The best model in fold 8: saved-model-048-.hdf5
The best model in fold 9: saved-model-042-.hdf5
The best model in fold 10: saved-model-043-.hdf5
```

Figure 90: Part three of the performance report for the model: ConvLSTM x5 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               480800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,748,036
Trainable params: 3,452,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 91: Part one of the performance report for the model: BLSTM LSTM (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85619
_________________________________________________________________
F1 score average for None-class:                          0.89757
_________________________________________________________________
F1 score average for Bully-class:                         0.75691
_________________________________________________________________
F1 score weighted average:                                0.861
_________________________________________________________________
F1 score macro average:                                   0.82724
_________________________________________________________________
Precision average for None-class:                         0.94417
_________________________________________________________________
Precision average for Bully-class:                        0.68021
_________________________________________________________________
Precision weighted average:                               0.8753
_________________________________________________________________
Precision score macro average:                            0.81219
_________________________________________________________________
Recall average for None-class:                            0.85599
_________________________________________________________________
Recall average for Bully-class:                           0.85577
_________________________________________________________________
Recall weighted average:                                  0.85619
_________________________________________________________________
Recall score macro average:                               0.85588
=================================================================
```

Figure 92: Part two of the performance report for the model: BLSTM LSTM (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-147-.hdf5
The best model in fold 2: saved-model-053-.hdf5
The best model in fold 3: saved-model-036-.hdf5
The best model in fold 4: saved-model-154-.hdf5
The best model in fold 5: saved-model-052-.hdf5
The best model in fold 6: saved-model-076-.hdf5
The best model in fold 7: saved-model-029-.hdf5
The best model in fold 8: saved-model-042-.hdf5
The best model in fold 9: saved-model-050-.hdf5
The best model in fold 10: saved-model-043-.hdf5
```

Figure 93: Part three of the performance report for the model: BLSTM LSTM (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,068,836
Trainable params: 3,773,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 94: Part one of the performance report for the model: BLSTM LSTM x2 (experiment 2, second test run with 800 epochs)

Model Performance (10-Cross-Fold):

Accuracy average:	0.85964
F1 score average for None-class:	0.90003
F1 score average for Bully-class:	0.76304
F1 score weighted average:	0.86435
F1 score macro average:	0.83154
Precision average for None-class:	0.94706
Precision average for Bully-class:	0.6859
Precision weighted average:	0.87917
Precision score macro average:	0.81648
Recall average for None-class:	0.8584
Recall average for Bully-class:	0.86387
Recall weighted average:	0.85964
Recall score macro average:	0.86113

===

Figure 95: Part two of the performance report for the model: BLSTM LSTM x2 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-069-.hdf5
The best model in fold 2: saved-model-104-.hdf5
The best model in fold 3: saved-model-102-.hdf5
The best model in fold 4: saved-model-175-.hdf5
The best model in fold 5: saved-model-086-.hdf5
The best model in fold 6: saved-model-056-.hdf5
The best model in fold 7: saved-model-063-.hdf5
The best model in fold 8: saved-model-083-.hdf5
The best model in fold 9: saved-model-069-.hdf5
The best model in fold 10: saved-model-052-.hdf5
```

Figure 96: Part three of the performance report for the model: BLSTM LSTM x2 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,389,636
Trainable params: 4,094,036
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 97: Part one of the performance report for the model: BLSTM LSTM x3 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.86169
_________________________________________________________________
F1 score average for None-class:                          0.90222
_________________________________________________________________
F1 score average for Bully-class:                         0.76243
_________________________________________________________________
F1 score weighted average:                                0.86584
_________________________________________________________________
F1 score macro average:                                   0.83233
_________________________________________________________________
Precision average for None-class:                         0.9422
_________________________________________________________________
Precision average for Bully-class:                        0.6936
_________________________________________________________________
Precision weighted average:                               0.87747
_________________________________________________________________
Precision score macro average:                            0.8179
_________________________________________________________________
Recall average for None-class:                            0.86605
_________________________________________________________________
Recall average for Bully-class:                           0.84931
_________________________________________________________________
Recall weighted average:                                  0.86169
_________________________________________________________________
Recall score macro average:                               0.85768
=================================================================
```

Figure 98: Part two of the performance report for the model: BLSTM LSTM x3 (experiment 2, second test run with 800 epochs)

```
================================================================
The best model in fold 1: saved-model-087-.hdf5
The best model in fold 2: saved-model-099-.hdf5
The best model in fold 3: saved-model-099-.hdf5
The best model in fold 4: saved-model-225-.hdf5
The best model in fold 5: saved-model-070-.hdf5
The best model in fold 6: saved-model-253-.hdf5
The best model in fold 7: saved-model-764-.hdf5
The best model in fold 8: saved-model-348-.hdf5
The best model in fold 9: saved-model-086-.hdf5
The best model in fold 10: saved-model-096-.hdf5
```

Figure 99: Part three of the performance report for the model: BLSTM LSTM x3 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,710,436
Trainable params: 4,414,836
Non-trainable params: 2,295,600

_________________________________________________________________
=================================================================
```

Figure 100: Part one of the performance report for the model: BLSTM LSTM x4 (experiment 2, second test run with 800 epochs)

Model Performance (10-Cross-Fold):

Accuracy average:	0.85184
F1 score average for None-class:	0.89322
F1 score average for Bully-class:	0.75653
F1 score weighted average:	0.85761
F1 score macro average:	0.82488
Precision average for None-class:	0.95162
Precision average for Bully-class:	0.66889
Precision weighted average:	0.87817
Precision score macro average:	0.81025
Recall average for None-class:	0.84293
Recall average for Bully-class:	0.87819
Recall weighted average:	0.85184
Recall score macro average:	0.86056

Figure 101: Part two of the performance report for the model: BLSTM LSTM x4 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-145-.hdf5
The best model in fold 2: saved-model-127-.hdf5
The best model in fold 3: saved-model-116-.hdf5
The best model in fold 4: saved-model-146-.hdf5
The best model in fold 5: saved-model-513-.hdf5
The best model in fold 6: saved-model-160-.hdf5
The best model in fold 7: saved-model-205-.hdf5
The best model in fold 8: saved-model-151-.hdf5
The best model in fold 9: saved-model-162-.hdf5
The best model in fold 10: saved-model-136-.hdf5
```

Figure 102: Part three of the performance report for the model: BLSTM LSTM x4 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_6 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 7,031,236
Trainable params: 4,735,636
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 103: Part one of the performance report for the model: BLSTM LSTM x5 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                    0.85802
F1 score average for None-class:                     0.89867
F1 score average for Bully-class:                    0.76146
F1 score weighted average:                           0.86287
F1 score macro average:                              0.83007
Precision average for None-class:                    0.94642
Precision average for Bully-class:                   0.68376
Precision weighted average:                          0.87798
Precision score macro average:                       0.81509
Recall average for None-class:                       0.85637
Recall average for Bully-class:                      0.86322
Recall weighted average:                             0.85802
Recall score macro average:                          0.85979
=================================================================
```

Figure 104: Part two of the performance report for the model: BLSTM LSTM x5 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-370-.hdf5
The best model in fold 2: saved-model-490-.hdf5
The best model in fold 3: saved-model-499-.hdf5
The best model in fold 4: saved-model-416-.hdf5
The best model in fold 5: saved-model-471-.hdf5
The best model in fold 6: saved-model-320-.hdf5
The best model in fold 7: saved-model-398-.hdf5
The best model in fold 8: saved-model-561-.hdf5
The best model in fold 9: saved-model-486-.hdf5
The best model in fold 10: saved-model-374-.hdf5
```

Figure 105: Part three of the performance report for the model: BLSTM LSTM x5 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               641600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 7,562,836
Trainable params: 5,267,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 106: Part one of the performance report for the model: BLSTM (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85551
_________________________________________________________________
F1 score average for None-class:                          0.89726
_________________________________________________________________
F1 score average for Bully-class:                         0.75532
_________________________________________________________________
F1 score weighted average:                                0.86031
_________________________________________________________________
F1 score macro average:                                   0.82629
_________________________________________________________________
Precision average for None-class:                         0.94404
_________________________________________________________________
Precision average for Bully-class:                        0.67831
_________________________________________________________________
Precision weighted average:                               0.87488
_________________________________________________________________
Precision score macro average:                            0.81118
_________________________________________________________________
Recall average for None-class:                            0.85555
_________________________________________________________________
Recall average for Bully-class:                           0.8556
_________________________________________________________________
Recall weighted average:                                  0.85551
_________________________________________________________________
Recall score macro average:                               0.85558
=================================================================
```

Figure 107: Part two of the performance report for the model: BLSTM (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-021-.hdf5
The best model in fold 2: saved-model-026-.hdf5
The best model in fold 3: saved-model-053-.hdf5
The best model in fold 4: saved-model-011-.hdf5
The best model in fold 5: saved-model-019-.hdf5
The best model in fold 6: saved-model-022-.hdf5
The best model in fold 7: saved-model-033-.hdf5
The best model in fold 8: saved-model-025-.hdf5
The best model in fold 9: saved-model-029-.hdf5
The best model in fold 10: saved-model-018-.hdf5
```

Figure 108: Part three of the performance report for the model: BLSTM (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 109: Part one of the performance report for the model: BLSTM x2 (experiment 2, second test run with 800 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.8601
F1 score average for None-class:	0.90039
F1 score average for Bully-class:	0.76396
F1 score weighted average:	0.86492
F1 score macro average:	0.83218
Precision average for None-class:	0.94818
Precision average for Bully-class:	0.68492
Precision weighted average:	0.87978
Precision score macro average:	0.81655
Recall average for None-class:	0.85782
Recall average for Bully-class:	0.86688
Recall weighted average:	0.8601
Recall score macro average:	0.86235

===

Figure 110: Part two of the performance report for the model: BLSTM x2 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-059-.hdf5
The best model in fold 2: saved-model-070-.hdf5
The best model in fold 3: saved-model-039-.hdf5
The best model in fold 4: saved-model-110-.hdf5
The best model in fold 5: saved-model-065-.hdf5
The best model in fold 6: saved-model-770-.hdf5
The best model in fold 7: saved-model-077-.hdf5
The best model in fold 8: saved-model-031-.hdf5
The best model in fold 9: saved-model-044-.hdf5
The best model in fold 10: saved-model-052-.hdf5
```

Figure 111: Part three of the performance report for the model: BLSTM x2 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_3 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 9,486,036
Trainable params: 7,190,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 112: Part one of the performance report for the model: BLSTM x3 (experiment 2, second test run with 800 epochs)

```
=================================================================
Total params: 9,486,036
Trainable params: 7,190,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================

Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85597
_________________________________________________________________
F1 score average for None-class:                          0.89694
_________________________________________________________________
F1 score average for Bully-class:                         0.75895
_________________________________________________________________
F1 score weighted average:                                0.86108
_________________________________________________________________
F1 score macro average:                                   0.82794
_________________________________________________________________
Precision average for None-class:                         0.94791
_________________________________________________________________
Precision average for Bully-class:                        0.6776
_________________________________________________________________
Precision weighted average:                               0.87738
_________________________________________________________________
Precision score macro average:                            0.81275
_________________________________________________________________
Recall average for None-class:                            0.85199
_________________________________________________________________
Recall average for Bully-class:                           0.8661
_________________________________________________________________
Recall weighted average:                                  0.85597
_________________________________________________________________
Recall score macro average:                               0.85904
=================================================================
```

Figure 113: Part two of the performance report for the model: BLSTM x3 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-059-.hdf5
The best model in fold 2: saved-model-040-.hdf5
The best model in fold 3: saved-model-046-.hdf5
The best model in fold 4: saved-model-063-.hdf5
The best model in fold 5: saved-model-076-.hdf5
The best model in fold 6: saved-model-377-.hdf5
The best model in fold 7: saved-model-322-.hdf5
The best model in fold 8: saved-model-069-.hdf5
The best model in fold 9: saved-model-049-.hdf5
The best model in fold 10: saved-model-053-.hdf5
```

Figure 114: Part three of the performance report for the model: BLSTM x3 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_3 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_4 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 10,447,636
Trainable params: 8,152,036
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 115: Part one of the performance report for the model: BLSTM x4 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85757
_________________________________________________________________
F1 score average for None-class:                          0.89828
_________________________________________________________________
F1 score average for Bully-class:                         0.76066
_________________________________________________________________
F1 score weighted average:                                0.86252
_________________________________________________________________
F1 score macro average:                                   0.82947
_________________________________________________________________
Precision average for None-class:                         0.94726
_________________________________________________________________
Precision average for Bully-class:                        0.68103
_________________________________________________________________
Precision weighted average:                               0.87775
_________________________________________________________________
Precision score macro average:                            0.81415
_________________________________________________________________
Recall average for None-class:                            0.85475
_________________________________________________________________
Recall average for Bully-class:                           0.86427
_________________________________________________________________
Recall weighted average:                                  0.85757
_________________________________________________________________
Recall score macro average:                               0.85951
=================================================================
```

Figure 116: Part two of the performance report for the model: BLSTM x4 (experiment 2, second test run with 800 epochs)

```
================================================================
The best model in fold 1: saved-model-058-.hdf5
The best model in fold 2: saved-model-073-.hdf5
The best model in fold 3: saved-model-043-.hdf5
The best model in fold 4: saved-model-059-.hdf5
The best model in fold 5: saved-model-058-.hdf5
The best model in fold 6: saved-model-392-.hdf5
The best model in fold 7: saved-model-063-.hdf5
The best model in fold 8: saved-model-057-.hdf5
The best model in fold 9: saved-model-061-.hdf5
The best model in fold 10: saved-model-211-.hdf5
```

Figure 117: Part three of the performance report for the model: BLSTM x4 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_3 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_4 (Bidirection (None, 34, 400)           961600
_________________________________________________________________
bidirectional_5 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
```

Figure 118: Part one of the performance report for the model: BLSTM x5 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
-----------------------------------------------------------------
Accuracy average:                                        0.86215
-----------------------------------------------------------------
F1 score average for None-class:                         0.90222
-----------------------------------------------------------------
F1 score average for Bully-class:                        0.76397
-----------------------------------------------------------------
F1 score weighted average:                               0.86623
-----------------------------------------------------------------
F1 score macro average:                                  0.8331
-----------------------------------------------------------------
Precision average for None-class:                        0.94313
-----------------------------------------------------------------
Precision average for Bully-class:                       0.69705
-----------------------------------------------------------------
Precision weighted average:                              0.87878
-----------------------------------------------------------------
Precision score macro average:                           0.82009
-----------------------------------------------------------------
Recall average for None-class:                           0.86583
-----------------------------------------------------------------
Recall average for Bully-class:                          0.85045
-----------------------------------------------------------------
Recall weighted average:                                 0.86215
-----------------------------------------------------------------
Recall score macro average:                              0.85814
=================================================================
```

Figure 119: Part two of the performance report for the model: BLSTM x5 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-105-.hdf5
The best model in fold 2: saved-model-061-.hdf5
The best model in fold 3: saved-model-071-.hdf5
The best model in fold 4: saved-model-512-.hdf5
The best model in fold 5: saved-model-131-.hdf5
The best model in fold 6: saved-model-201-.hdf5
The best model in fold 7: saved-model-132-.hdf5
The best model in fold 8: saved-model-240-.hdf5
The best model in fold 9: saved-model-116-.hdf5
The best model in fold 10: saved-model-092-.hdf5
```

Figure 120: Part three of the performance report for the model: BLSTM x5 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200

Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600

lstm_1 (LSTM)                (None, 200)               320800

dense_1 (Dense)              (None, 11478)             2307078

dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 4,946,436
Trainable params: 2,650,836
Non-trainable params: 2,295,600

=================================================================
```

Figure 121: Part one of the performance report for the model: LSTM (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):

Accuracy average:                                          0.85367

F1 score average for None-class:                           0.89507

F1 score average for Bully-class:                          0.7571

F1 score weighted average:                                 0.85917

F1 score macro average:                                    0.82609

Precision average for None-class:                          0.94875

Precision average for Bully-class:                         0.67178

Precision weighted average:                                0.87676

Precision score macro average:                             0.81027

Recall average for None-class:                             0.84787

Recall average for Bully-class:                            0.87069

Recall weighted average:                                   0.85367

Recall score macro average:                                0.85928
=================================================================
```

Figure 122: Part two of the performance report for the model: LSTM (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-036-.hdf5
The best model in fold 2: saved-model-024-.hdf5
The best model in fold 3: saved-model-027-.hdf5
The best model in fold 4: saved-model-027-.hdf5
The best model in fold 5: saved-model-023-.hdf5
The best model in fold 6: saved-model-033-.hdf5
The best model in fold 7: saved-model-030-.hdf5
The best model in fold 8: saved-model-038-.hdf5
The best model in fold 9: saved-model-040-.hdf5
The best model in fold 10: saved-model-022-.hdf5
```

Figure 123: Part three of the performance report for the model: LSTM (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,267,236
Trainable params: 2,971,636
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 124: Part one of the performance report for the model: LSTM x2 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                   0.85803
F1 score average for None-class:                    0.89892
F1 score average for Bully-class:                   0.75997
F1 score weighted average:                          0.86278
F1 score macro average:                             0.82944
Precision average for None-class:                   0.94526
Precision average for Bully-class:                  0.68403
Precision weighted average:                         0.87716
Precision score macro average:                      0.81465
Recall average for None-class:                      0.85762
Recall average for Bully-class:                     0.85856
Recall weighted average:                            0.85803
Recall score macro average:                         0.85809
=================================================================
```

Figure 125: Part two of the performance report for the model: LSTM x2 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-087-.hdf5
The best model in fold 2: saved-model-059-.hdf5
The best model in fold 3: saved-model-114-.hdf5
The best model in fold 4: saved-model-082-.hdf5
The best model in fold 5: saved-model-061-.hdf5
The best model in fold 6: saved-model-065-.hdf5
The best model in fold 7: saved-model-041-.hdf5
The best model in fold 8: saved-model-046-.hdf5
The best model in fold 9: saved-model-047-.hdf5
The best model in fold 10: saved-model-054-.hdf5
```

Figure 126: Part three of the performance report for the model: LSTM x2 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,588,036
Trainable params: 3,292,436
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 127: Part one of the performance report for the model: LSTM x3 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85781
_________________________________________________________________
F1 score average for None-class:                          0.89835
_________________________________________________________________
F1 score average for Bully-class:                         0.76111
_________________________________________________________________
F1 score weighted average:                                0.86273
_________________________________________________________________
F1 score macro average:                                   0.82973
_________________________________________________________________
Precision average for None-class:                         0.94901
_________________________________________________________________
Precision average for Bully-class:                        0.68037
_________________________________________________________________
Precision weighted average:                               0.87889
_________________________________________________________________
Precision score macro average:                            0.81469
_________________________________________________________________
Recall average for None-class:                            0.85379
_________________________________________________________________
Recall average for Bully-class:                           0.86754
_________________________________________________________________
Recall weighted average:                                  0.85781
_________________________________________________________________
Recall score macro average:                               0.86066
=================================================================
```

Figure 128: Part two of the performance report for the model: LSTM x3 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-067-.hdf5
The best model in fold 2: saved-model-064-.hdf5
The best model in fold 3: saved-model-065-.hdf5
The best model in fold 4: saved-model-098-.hdf5
The best model in fold 5: saved-model-079-.hdf5
The best model in fold 6: saved-model-048-.hdf5
The best model in fold 7: saved-model-128-.hdf5
The best model in fold 8: saved-model-073-.hdf5
The best model in fold 9: saved-model-053-.hdf5
The best model in fold 10: saved-model-230-.hdf5
```

Figure 129: Part three of the performance report for the model: LSTM x3 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 130: Part one of the performance report for the model: LSTM x4 (experiment 2, second test run with 800 epochs)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.86055
F1 score average for None-class:	0.9009
F1 score average for Bully-class:	0.76275
F1 score weighted average:	0.86501
F1 score macro average:	0.83182
Precision average for None-class:	0.94589
Precision average for Bully-class:	0.68736
Precision weighted average:	0.8783
Precision score macro average:	0.81662
Recall average for None-class:	0.86054
Recall average for Bully-class:	0.85875
Recall weighted average:	0.86055
Recall score macro average:	0.85964

Figure 131: Part two of the performance report for the model: LSTM x4 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-124-.hdf5
The best model in fold 2: saved-model-372-.hdf5
The best model in fold 3: saved-model-079-.hdf5
The best model in fold 4: saved-model-271-.hdf5
The best model in fold 5: saved-model-080-.hdf5
The best model in fold 6: saved-model-181-.hdf5
The best model in fold 7: saved-model-064-.hdf5
The best model in fold 8: saved-model-082-.hdf5
The best model in fold 9: saved-model-107-.hdf5
The best model in fold 10: saved-model-107-.hdf5
```

Figure 132: Part three of the performance report for the model: LSTM x4 (experiment 2, second test run with 800 epochs)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_5 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,229,636
Trainable params: 3,934,036
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 133: Part one of the performance report for the model: LSTM x5 (experiment 2, second test run with 800 epochs)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                        0.85872
F1 score average for None-class:                         0.89949
F1 score average for Bully-class:                        0.76088
F1 score weighted average:                               0.86347
F1 score macro average:                                  0.83018
Precision average for None-class:                        0.94542
Precision average for Bully-class:                       0.68399
Precision weighted average:                              0.8773
Precision score macro average:                           0.8147
Recall average for None-class:                           0.85828
Recall average for Bully-class:                          0.85924
Recall weighted average:                                 0.85872
Recall score macro average:                              0.85876
=================================================================
```

Figure 134: Part two of the performance report for the model: LSTM x5 (experiment 2, second test run with 800 epochs)

```
=================================================================
The best model in fold 1: saved-model-160-.hdf5
The best model in fold 2: saved-model-128-.hdf5
The best model in fold 3: saved-model-151-.hdf5
The best model in fold 4: saved-model-175-.hdf5
The best model in fold 5: saved-model-158-.hdf5
The best model in fold 6: saved-model-164-.hdf5
The best model in fold 7: saved-model-176-.hdf5
The best model in fold 8: saved-model-684-.hdf5
The best model in fold 9: saved-model-175-.hdf5
The best model in fold 10: saved-model-524-.hdf5
```

Figure 135: Part three of the performance report for the model: LSTM x5 (experiment 2, second test run with 800 epochs)

D Performance reports from experiment 2

```
Neural network model with SVM classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 136: Part one of the performance report for the model: ConvLSTM x2 with a SVM classifier as activation (experiment 2)

Model Performance SVM(10-Cross-Fold):

Accuracy average:	0.64257
F1 score average for None-class:	0.77693
F1 score average for Bully-class:	0.06496
F1 score weighted average:	0.59027
F1 score macro average:	0.42095
Precision average for None-class:	0.71733
Precision average for Bully-class:	0.08407
Precision weighted average:	0.55143
Precision score macro average:	0.4007
Recall average for None-class:	0.85177
Recall average for Bully-class:	0.05501
Recall weighted average:	0.64257
Recall score macro average:	0.45339

Figure 137: Part two of the performance report for the model: ConvLSTM x2 with a SVM classifier as activation (experiment 2)

```
Neural network model with RFC classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 138: Part one of the performance report for the model: ConvLSTM x2 with a RFC classifier as activation (experiment 2)

```
Model Performance RFC(10-Cross-Fold):
_______________________________________________________________
Accuracy average:                                       0.81341
_______________________________________________________________
F1 score average for None-class:                        0.87534
_______________________________________________________________
F1 score average for Bully-class:                       0.538
_______________________________________________________________
F1 score weighted average:                              0.7881
_______________________________________________________________
F1 score macro average:                                 0.70667
_______________________________________________________________
Precision average for None-class:                       0.88075
_______________________________________________________________
Precision average for Bully-class:                      0.52386
_______________________________________________________________
Precision weighted average:                             0.78695
_______________________________________________________________
Precision score macro average:                          0.70231
_______________________________________________________________
Recall average for None-class:                          0.88772
_______________________________________________________________
Recall average for Bully-class:                         0.59374
_______________________________________________________________
Recall weighted average:                                0.81341
_______________________________________________________________
Recall score macro average:                             0.74073
===============================================================
```

Figure 139: Part two of the performance report for the model: ConvLSTM x2 with a RFC classifier as activation (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 140: Part one of the performance report for the model; ConvLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

Model Performance (10-Cross-Fold):

Accuracy average:	0.86238
F1 score average for None-class:	0.90228
F1 score average for Bully-class:	0.76541
F1 score weighted average:	0.86667
F1 score macro average:	0.83385
Precision average for None-class:	0.94559
Precision average for Bully-class:	0.69444
Precision weighted average:	0.88011
Precision score macro average:	0.82002
Recall average for None-class:	0.86383
Recall average for Bully-class:	0.85774
Recall weighted average:	0.86238
Recall score macro average:	0.86078

Figure 141: Part two of the performance report for the model; ConvLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-774-.hdf5
The best model in fold 2: saved-model-020-.hdf5
The best model in fold 3: saved-model-031-.hdf5
The best model in fold 4: saved-model-038-.hdf5
The best model in fold 5: saved-model-045-.hdf5
The best model in fold 6: saved-model-022-.hdf5
The best model in fold 7: saved-model-037-.hdf5
The best model in fold 8: saved-model-030-.hdf5
The best model in fold 9: saved-model-028-.hdf5
The best model in fold 10: saved-model-023-.hdf5
```

Figure 142: Part three of the performance report for the model; ConvLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 34, 256)           563456
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 17, 256)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 17, 200)           365600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 17, 128)           230528
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 8, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 200)               263200
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,048,420
Trainable params: 3,752,820
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 143: Part one of the performance report for the model: ConvLSTM x2 with SVM like activation (experiment 2)

```
Model Performance (10-Cross-Fold):
Accuracy average:                                    0.86902
F1 score average for None-class:                     0.90774
F1 score average for Bully-class:                    0.77269
F1 score weighted average:                           0.87262
F1 score macro average:                              0.84021
Precision average for None-class:                    0.94505
Precision average for Bully-class:                   0.70627
Precision weighted average:                          0.88271
Precision score macro average:                       0.82566
Recall average for None-class:                       0.87365
Recall average for Bully-class:                      0.85484
Recall weighted average:                             0.86902
Recall score macro average:                          0.86424
=================================================================
```

Figure 144: Part two of the performance report for the model: ConvLSTM x2 with SVM like activation (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-352-.hdf5
The best model in fold 2: saved-model-053-.hdf5
The best model in fold 3: saved-model-740-.hdf5
The best model in fold 4: saved-model-161-.hdf5
The best model in fold 5: saved-model-024-.hdf5
The best model in fold 6: saved-model-029-.hdf5
The best model in fold 7: saved-model-034-.hdf5
The best model in fold 8: saved-model-369-.hdf5
The best model in fold 9: saved-model-352-.hdf5
The best model in fold 10: saved-model-038-.hdf5
```

Figure 145: Part three of the performance report for the model: ConvLSTM x2 with SVM like activation (experiment 2)

```
Neural network model with SVM classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 34, 200)	2295600
bidirectional_1 (Bidirection	(None, 34, 400)	641600
lstm_2 (LSTM)	(None, 34, 200)	480800
lstm_3 (LSTM)	(None, 200)	320800
dense_1 (Dense)	(None, 11478)	2307078
dense_2 (Dense)	(None, 2)	22958

Figure 146: Part one of the performance report for the model: BLSTM LSTM x2 with a SVM classifier as activation (experiment 2)

Model Performance SVM(10-Cross-Fold):

Accuracy average:	0.22818
F1 score average for None-class:	0.17333
F1 score average for Bully-class:	0.20572
F1 score weighted average:	0.18344
F1 score macro average:	0.18952
Precision average for None-class:	0.23814
Precision average for Bully-class:	0.13275
Precision weighted average:	0.21214
Precision score macro average:	0.18544
Recall average for None-class:	0.14032
Recall average for Bully-class:	0.4683
Recall weighted average:	0.22818
Recall score macro average:	0.30431

Figure 147: Part two of the performance report for the model: BLSTM LSTM x2 with a SVM classifier as activation (experiment 2)

```
Neural network model with RFC classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,068,836
Trainable params: 3,773,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 148: Part one of the performance report for the model: BLSTM LSTM x2 with a RFC classifier as activation (experiment 2)

```
Model Performance RFC(10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                        0.73895
_________________________________________________________________
F1 score average for None-class:                         0.84981
_________________________________________________________________
F1 score average for Bully-class:                        0.0
_________________________________________________________________
F1 score weighted average:                               0.6281
_________________________________________________________________
F1 score macro average:                                  0.4249
_________________________________________________________________
Precision average for None-class:                        0.73895
_________________________________________________________________
Precision average for Bully-class:                       0.0
_________________________________________________________________
Precision weighted average:                              0.54626
_________________________________________________________________
Precision score macro average:                           0.36948
_________________________________________________________________
Recall average for None-class:                           1.0
_________________________________________________________________
Recall average for Bully-class:                          0.0
_________________________________________________________________
Recall weighted average:                                 0.73895
_________________________________________________________________
Recall score macro average:                              0.5
=================================================================
```

Figure 149: Part two of the performance report for the model: BLSTM LSTM x2 with a RFC classifier as activation (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,068,836
Trainable params: 3,773,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 150: Part one of the performance report for the model; BLSTM LSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                          0.85459
_________________________________________________________________
F1 score average for None-class:                           0.89563
_________________________________________________________________
F1 score average for Bully-class:                          0.75854
_________________________________________________________________
F1 score weighted average:                                 0.85998
_________________________________________________________________
F1 score macro average:                                    0.82708
_________________________________________________________________
Precision average for None-class:                          0.95032
_________________________________________________________________
Precision average for Bully-class:                         0.67229
_________________________________________________________________
Precision weighted average:                                0.87774
_________________________________________________________________
Precision score macro average:                             0.81131
_________________________________________________________________
Recall average for None-class:                             0.8476
_________________________________________________________________
Recall average for Bully-class:                            0.87324
_________________________________________________________________
Recall weighted average:                                   0.85459
_________________________________________________________________
Recall score macro average:                                0.86042
=================================================================
```

Figure 151: Part two of the performance report for the model; BLSTM LSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-067-.hdf5
The best model in fold 2: saved-model-037-.hdf5
The best model in fold 3: saved-model-116-.hdf5
The best model in fold 4: saved-model-054-.hdf5
The best model in fold 5: saved-model-047-.hdf5
The best model in fold 6: saved-model-080-.hdf5
The best model in fold 7: saved-model-207-.hdf5
The best model in fold 8: saved-model-107-.hdf5
The best model in fold 9: saved-model-067-.hdf5
The best model in fold 10: saved-model-040-.hdf5
```

Figure 152: Part three of the performance report for the model; BLSTM LSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           480800
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 6,068,836
Trainable params: 3,773,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 153: Part one of the performance report for the model: BLSTM LSTM x2 with SVM like activation (experiment 2)

```
Model Performance (10-Cross-Fold):
________________________________________________________________
Accuracy average:                                        0.85435
________________________________________________________________
F1 score average for None-class:                         0.89504
________________________________________________________________
F1 score average for Bully-class:                        0.76047
________________________________________________________________
F1 score weighted average:                               0.85997
________________________________________________________________
F1 score macro average:                                  0.82776
________________________________________________________________
Precision average for None-class:                        0.95329
________________________________________________________________
Precision average for Bully-class:                       0.67337
________________________________________________________________
Precision weighted average:                              0.88055
________________________________________________________________
Precision score macro average:                           0.81333
________________________________________________________________
Recall average for None-class:                           0.84509
________________________________________________________________
Recall average for Bully-class:                          0.88176
________________________________________________________________
Recall weighted average:                                 0.85435
________________________________________________________________
Recall score macro average:                              0.86343
================================================================
```

Figure 154: Part two of the performance report for the model: BLSTM LSTM x2 with SVM like activation (experiment 2)

```
================================================================
The best model in fold 1: saved-model-545-.hdf5
The best model in fold 2: saved-model-296-.hdf5
The best model in fold 3: saved-model-046-.hdf5
The best model in fold 4: saved-model-423-.hdf5
The best model in fold 5: saved-model-235-.hdf5
The best model in fold 6: saved-model-670-.hdf5
The best model in fold 7: saved-model-036-.hdf5
The best model in fold 8: saved-model-792-.hdf5
The best model in fold 9: saved-model-403-.hdf5
The best model in fold 10: saved-model-051-.hdf5
```

Figure 155: Part three of the performance report for the model: BLSTM LSTM x2 with SVM like activation (experiment 2)

```
Neural network model with SVM classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 156: Part one of the performance report for the model: BLSTM x2 with a SVM classifier as activation (experiment 2)

```
Model Performance SVM(10-Cross-Fold):
__________________________________________________________________
Accuracy average:                                          0.74322
__________________________________________________________________
F1 score average for None-class:                           0.79686
__________________________________________________________________
F1 score average for Bully-class:                          0.62818
__________________________________________________________________
F1 score weighted average:                                 0.75343
__________________________________________________________________
F1 score macro average:                                    0.71252
__________________________________________________________________
Precision average for None-class:                          0.93718
__________________________________________________________________
Precision average for Bully-class:                         0.51765
__________________________________________________________________
Precision weighted average:                                0.82726
__________________________________________________________________
Precision score macro average:                             0.72742
__________________________________________________________________
Recall average for None-class:                             0.70719
__________________________________________________________________
Recall average for Bully-class:                            0.83934
__________________________________________________________________
Recall weighted average:                                   0.74322
__________________________________________________________________
Recall score macro average:                                0.77327
==================================================================
```

Figure 157: Part two of the performance report for the model: BLSTM x2 with a SVM classifier as activation (experiment 2)

```
Neural network model with RFC classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 158: Part one of the performance report for the model: BLSTM x2 with a RFC classifier as activation (experiment 2)

```
Model Performance RFC(10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                      0.73895
_________________________________________________________________
F1 score average for None-class:                       0.84981
_________________________________________________________________
F1 score average for Bully-class:                      0.0
_________________________________________________________________
F1 score weighted average:                             0.6281
_________________________________________________________________
F1 score macro average:                                0.4249
_________________________________________________________________
Precision average for None-class:                      0.73895
_________________________________________________________________
Precision average for Bully-class:                     0.0
_________________________________________________________________
Precision weighted average:                            0.54626
_________________________________________________________________
Precision score macro average:                         0.36948
_________________________________________________________________
Recall average for None-class:                         1.0
_________________________________________________________________
Recall average for Bully-class:                        0.0
_________________________________________________________________
Recall weighted average:                               0.73895
_________________________________________________________________
Recall score macro average:                            0.5
=================================================================
```

Figure 159: Part two of the performance report for the model: BLSTM x2 with a RFC classifier as activation (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 160: Part one of the performance report for the model; BLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                        0.8594
_________________________________________________________________
F1 score average for None-class:                         0.90105
_________________________________________________________________
F1 score average for Bully-class:                        0.7556
_________________________________________________________________
F1 score weighted average:                               0.86321
_________________________________________________________________
F1 score macro average:                                  0.82832
_________________________________________________________________
Precision average for None-class:                        0.93627
_________________________________________________________________
Precision average for Bully-class:                       0.69285
_________________________________________________________________
Precision weighted average:                              0.87269
_________________________________________________________________
Precision score macro average:                           0.81456
_________________________________________________________________
Recall average for None-class:                           0.86865
_________________________________________________________________
Recall average for Bully-class:                          0.83201
_________________________________________________________________
Recall weighted average:                                 0.8594
_________________________________________________________________
Recall score macro average:                              0.85033
=================================================================
```

Figure 161: Part two of the performance report for the model; BLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-050-.hdf5
The best model in fold 2: saved-model-087-.hdf5
The best model in fold 3: saved-model-040-.hdf5
The best model in fold 4: saved-model-178-.hdf5
The best model in fold 5: saved-model-056-.hdf5
The best model in fold 6: saved-model-072-.hdf5
The best model in fold 7: saved-model-057-.hdf5
The best model in fold 8: saved-model-166-.hdf5
The best model in fold 9: saved-model-044-.hdf5
The best model in fold 10: saved-model-048-.hdf5
```

Figure 162: Part three of the performance report for the model; BLSTM x2 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 34, 400)           641600
_________________________________________________________________
bidirectional_2 (Bidirection (None, 400)               961600
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             4602678
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 8,524,436
Trainable params: 6,228,836
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 163: Part one of the performance report for the model: BLSTM x2 with SVM like activation (experiment 2)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.85939
F1 score average for None-class:	0.89944
F1 score average for Bully-class:	0.76543
F1 score weighted average:	0.86457
F1 score macro average:	0.83244
Precision average for None-class:	0.95268
Precision average for Bully-class:	0.67986
Precision weighted average:	0.88187
Precision score macro average:	0.81627
Recall average for None-class:	0.85258
Recall average for Bully-class:	0.87973
Recall weighted average:	0.85939
Recall score macro average:	0.86615

Figure 164: Part two of the performance report for the model: BLSTM x2 with SVM like activation (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-449-.hdf5
The best model in fold 2: saved-model-057-.hdf5
The best model in fold 3: saved-model-057-.hdf5
The best model in fold 4: saved-model-295-.hdf5
The best model in fold 5: saved-model-365-.hdf5
The best model in fold 6: saved-model-068-.hdf5
The best model in fold 7: saved-model-550-.hdf5
The best model in fold 8: saved-model-071-.hdf5
The best model in fold 9: saved-model-159-.hdf5
The best model in fold 10: saved-model-105-.hdf5
```

Figure 165: Part three of the performance report for the model: BLSTM x2 with SVM like activation (experiment 2)

```
Neural network model with SVM classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 166: Part one of the performance report for the model: LSTM x4 with a SVM classifier as activation (experiment 2)

```
Model Performance SVM(10-Cross-Fold):
__________________________________________________________________
Accuracy average:                                          0.25807
__________________________________________________________________
F1 score average for None-class:                           0.04196
__________________________________________________________________
F1 score average for Bully-class:                          0.37583
__________________________________________________________________
F1 score weighted average:                                 0.13028
__________________________________________________________________
F1 score macro average:                                    0.20889
__________________________________________________________________
Precision average for None-class:                          0.10082
__________________________________________________________________
Precision average for Bully-class:                         0.23811
__________________________________________________________________
Precision weighted average:                                0.13706
__________________________________________________________________
Precision score macro average:                             0.16947
__________________________________________________________________
Recall average for None-class:                             0.03052
__________________________________________________________________
Recall average for Bully-class:                            0.8958
__________________________________________________________________
Recall weighted average:                                   0.25807
__________________________________________________________________
Recall score macro average:                                0.46316
==================================================================
```

Figure 167: Part two of the performance report for the model: LSTM x4 with a SVM classifier as activation (experiment 2)

```
Neural network model with RFC classifier instead of last dense layer
Neural network model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 168: Part one of the performance report for the model: LSTM x4 with a RFC classifier as activation (experiment 2)

```
Model Performance RFC(10-Cross-Fold):
__________________________________________________________________
Accuracy average:                                         0.73895
__________________________________________________________________
F1 score average for None-class:                          0.84981
__________________________________________________________________
F1 score average for Bully-class:                         0.0
__________________________________________________________________
F1 score weighted average:                                0.6281
__________________________________________________________________
F1 score macro average:                                   0.4249
__________________________________________________________________
Precision average for None-class:                         0.73895
__________________________________________________________________
Precision average for Bully-class:                        0.0
__________________________________________________________________
Precision weighted average:                               0.54626
__________________________________________________________________
Precision score macro average:                            0.36948
__________________________________________________________________
Recall average for None-class:                            1.0
__________________________________________________________________
Recall average for Bully-class:                           0.0
__________________________________________________________________
Recall weighted average:                                  0.73895
__________________________________________________________________
Recall score macro average:                               0.5
==================================================================
```

Figure 169: Part two of the performance report for the model: LSTM x4 with a RFC classifier as activation (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

Figure 170: Part one of the performance report for the model; LSTM x4 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model Performance (10-Cross-Fold):
_________________________________________________________________
Accuracy average:                                         0.85643
_________________________________________________________________
F1 score average for None-class:                          0.89713
_________________________________________________________________
F1 score average for Bully-class:                         0.7601
_________________________________________________________________
F1 score weighted average:                                0.8615
_________________________________________________________________
F1 score macro average:                                   0.82862
_________________________________________________________________
Precision average for None-class:                         0.94882
_________________________________________________________________
Precision average for Bully-class:                        0.67957
_________________________________________________________________
Precision weighted average:                               0.87851
_________________________________________________________________
Precision score macro average:                            0.81419
_________________________________________________________________
Recall average for None-class:                            0.85195
_________________________________________________________________
Recall average for Bully-class:                           0.86788
_________________________________________________________________
Recall weighted average:                                  0.85643
_________________________________________________________________
Recall score macro average:                               0.85992
=================================================================
```

Figure 171: Part two of the performance report for the model; LSTM x4 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-083-.hdf5
The best model in fold 2: saved-model-060-.hdf5
The best model in fold 3: saved-model-070-.hdf5
The best model in fold 4: saved-model-271-.hdf5
The best model in fold 5: saved-model-081-.hdf5
The best model in fold 6: saved-model-127-.hdf5
The best model in fold 7: saved-model-096-.hdf5
The best model in fold 8: saved-model-083-.hdf5
The best model in fold 9: saved-model-343-.hdf5
The best model in fold 10: saved-model-067-.hdf5
```

Figure 172: Part three of the performance report for the model; LSTM x4 with Softmax activation, used as the Neural Network for the SVM and RFC classifier tests (experiment 2)

```
Model overview:
Batch size:3000 Number of epochs: 800   Embedding size: 200
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 34, 200)           2295600
_________________________________________________________________
lstm_1 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_2 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_3 (LSTM)                (None, 34, 200)           320800
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               320800
_________________________________________________________________
dense_1 (Dense)              (None, 11478)             2307078
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22958
=================================================================
Total params: 5,908,836
Trainable params: 3,613,236
Non-trainable params: 2,295,600
_________________________________________________________________
=================================================================
```

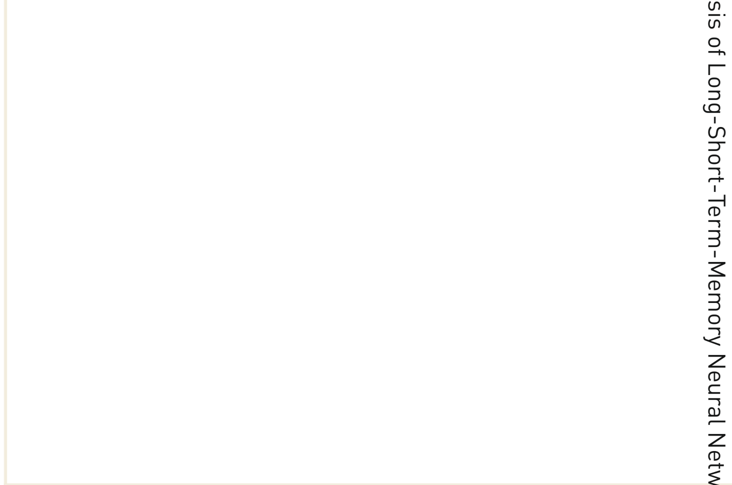
Figure 173: Part one of the performance report for the model: LSTM x4 with SVM like activation (experiment 2)

Model Performance (10-Cross-Fold):	
Accuracy average:	0.8642
F1 score average for None-class:	0.90372
F1 score average for Bully-class:	0.76879
F1 score weighted average:	0.86853
F1 score macro average:	0.83625
Precision average for None-class:	0.9469
Precision average for Bully-class:	0.69497
Precision weighted average:	0.88141
Precision score macro average:	0.82094
Recall average for None-class:	0.86484
Recall average for Bully-class:	0.8634
Recall weighted average:	0.8642
Recall score macro average:	0.86412

Figure 174: Part two of the performance report for the model: LSTM x4 with SVM like activation (experiment 2)

```
=================================================================
The best model in fold 1: saved-model-079-.hdf5
The best model in fold 2: saved-model-171-.hdf5
The best model in fold 3: saved-model-628-.hdf5
The best model in fold 4: saved-model-116-.hdf5
The best model in fold 5: saved-model-063-.hdf5
The best model in fold 6: saved-model-583-.hdf5
The best model in fold 7: saved-model-149-.hdf5
The best model in fold 8: saved-model-126-.hdf5
The best model in fold 9: saved-model-564-.hdf5
The best model in fold 10: saved-model-091-.hdf5
```

Figure 175: Part three of the performance report for the model: LSTM x4 with SVM like activation (experiment 2)



Thor Aleksander Buan

In depth analysis of Long-Short-Term-Memory Neural Networks with the purpose of detecting cyberbullying

NTNU
Norwegian University of
Science and Technology