International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I12P120

Volume 72 Issue 12, 164-170, December 2024
© 2024 Seventh Sense Research Group®

Original Article

Hiding Expletive Comments in Mobile Applications using CNN and LSTM based NLP Classification Models

Shivam Tomar

Independent Scholar, California, USA.

Corresponding Author : tomar.shivam@gmail.com

Received: 06 November 2024 Revised: 30 November 2024 Accepted: 19 December 2024 Published: 31 December 2024

Abstract *- Often, e-commerce mobile applications show the public comments of consumers about the products they sell. Sometimes, these comments contain foul language, which is inappropriate to be shown on the public platform. App developers would want to hide them and show them only after the consent from the app user. The goal of this study is to find the optimal way to classify comments as expletive or not using the NLP classification model. This study utilized CNN and LSTM algorithms to train the expletive language classification model. These models are used by the mobile application to find whether comments from users are expletive in nature or not. If a comment is found to be expletive, it will be hidden. The mobile app will also provide an option to unhide the expletive comments if the user wants to see them. LSTM models are found to be more accurate than CNN models with large datasets. Hiding expletive comments is very important for organizations to meet the guidelines of various countries. Deep learning provides an accurate and novel approach to achieve this feature.*

Keywords *- CNNs, Deep Learning, LSTM, NLP, Mobile Applications, Android.*

1. Introduction

E-commerce mobile applications usually have a feature to let their customers add feedback for the products they purchase. These feedback comments are shown on the mobile application to all the users. Sometimes, these comments contain foul language and are inappropriate to be shown on public platforms. These comments should be hidden by default and made visible only if the app users consent to it.

Major mobile platforms like Android and iOS SDKs lack any API to classify comments text as expletive or non-expletive. There is also a lack of reliable Web APIs that mobile apps can use to detect expletive comments. The goal of this research is to train DL base NLP classification models for expletive language detection using various approaches and compare their performance to find the best model training algorithm for this use case.

Deep learning-based NLP is an excellent way to classify comments as expletive or non-expletive, and mobile apps can leverage these models to hide the expletive comments, only to be shown upon consent from users.

This paper used the CADD dataset to train two models with CNN and LSTM algorithms, respectively and then found the accuracy of these algorithms by using AOC matrices to find the best-performing algorithm. The best-performing algorithm is deployed on an AWS EC2 instance to be used by the mobile apps.

2. Literature Review

Detecting expletive language is crucial for fostering a safe digital environment in mobile apps. After the detection of such language, mobile apps can hide it to protect users from being exposed to it, and machine learning-based techniques have proven to play a crucial role in detecting expletive language. Existing research on expletive language detection is shared here, along with the techniques involved and existing shortcomings.

2.1. Keyword based Techniques

Traditional ways for expletive language detection involve known keywords and/or regular expressions. Keyword/Regular expression-based technique has the following shortcomings:
- Missing the context of a sentence
- Variation in spelling of keywords
- Known expletive keywords missing in the sentence

2.2. Machine Learning based Approach

This approach includes using ML algorithms like Naïve Bayes, SVM, etc., for expletive language classification. This approach works quite well on small datasets but struggles with complex context. This approach is also not suitable for multilingual data.

2.3. Deep Learning based Approach

It includes DL algorithms like CNN, RNN, LSTM and transformers. DL-based text classification techniques are proven to be very accurate, including in expletive language detection. These techniques are quite good at remembering the context of words used in a sentence. This study aims to find and compare the performance of LSTM and CNN DL algorithms in expletive language detection.

3. DL Model Training Algorithms for Expletive Language Detection

3.1. Dataset Used for Training

The dataset used for training the DL model is CADD (Comprehensive Abuse Detection Dataset), which contains the abusive comments collected from English Reddit posts with multifaceted labels and context.

3.2. Preprocessing of Data

The training data is pre-processed by following techniques prior to training the model:

- Lowercase training data: For this study, training data was lowercase to reduce data complexity and improve computational efficiency. Lowercasing reduces data complexity by normalizing the input data and improving generalization. For example, The words “cat” and “Cat” will be considered the same, and the model will focus on the meaning of words rather than cases. This also reduces the vocabulary size, hence less embedding generation, which improves computation efficiency and reduces training time.
- Tokenization: This step includes breaking down the training data texts into smaller units called tokens. Tokens can be words or sub-words. The text is split based on white space. This study has used “words_tokenize” from the “nltk.tokenize” python library. Tokenization transforms unstructured text into manageable tokens that are essential for training NLP models.
- Stop words removal: It is an essential step to improve the vocabulary size, which in turn improves computation efficiency. Stop words are the words that add little to no value to the semantics of text. Examples of stop words are: “is”, “an”, “in”, etc.
- Collating bi grams: This study collates the bigrams in the training data corpus before performing model training. For example, Tokens “New” and “York” are collated to “New_York”.
- Stemming: This study has used stemming to clean the data further and improve computational efficiency. Stemming is a technique to algorithmically reduce words to their base form by removing prefixes and suffixes while maintaining the core meaning of the word. Stemming might end up changing the word to another word, which is invalid in that language. For ex, running is stemmed to run.

3.3. CNNs Model Details

Convolutional Neural Networks, or CNNs, are based on the concept of sliding(convolving) a small window over the data sample. This study uses triplets of words in the training data for the expletive language classification model.

3.3.1. CNN Model Hyperparameters

The following hyperparameters are used for the training of the CNN model:

Table 1. CNN models hyperparameters

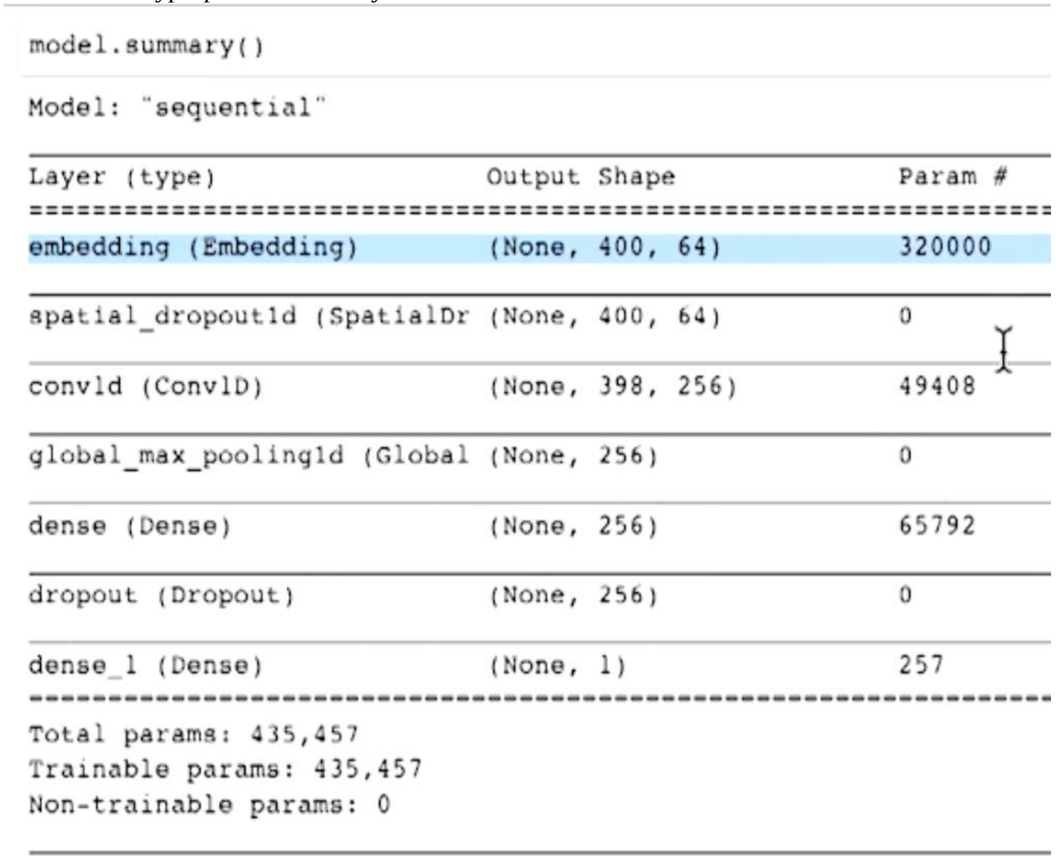
Hyperparameter	Value	Description
epochs	4	Number of passes of training dataset through model during training
batch size	128	The size of the training sample that models the process in one forward and backward pass during training
Vector space embedding	64	Size of the word vector space embedding
Unique Words	5000	Count of most significant words used from the training data corpus
Max review length	400	Max length of the comments
Embedding dropout	0.2	Dropping 20% of embedding at any given round of training to generalize on new training data
CNN filters	256	Count of filters on Convolutional layers.
Kernel length	3	Size of each filter is 3 tokens to find the triplets of words relevant for expletive comment detection.
Dense layer size	256	Every dense hidden layer will have 256 neurons
Dropout	0.2	Dropping 20% of neurons from dense layer to generalize on unseen data

3.3.2. CNN Model Architecture

In this case, the CNN model will have the following layers: (Embedding, SpatialDropout1D), (Conv1D, GlobalMaxPooling1D), (Dense layer, Dropout), (Dense, Sigmoid for output) [1].

The first layer in the CNN is an embedding layer with a 20% dropout in each training. This layer will create a word embedding vector of length 64. The next layer is one dimensional CNN layer. Here, 256 filters with a kernel length of 3 are used [2][3]. The output of this layer will be a vector of size 256. This vector will further be passed to a dense layer where each neuron has 256 weights and a "ReLU" activation function. The final layer is a dense layer with the activation function 'sigmoid'. This layer determines whether a given comment is expletive or not.

3.3.3. Trained CNN Model Hyperparameter Analysis



```
model.summary()

Model: "sequential"

Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 400, 64)           320000

spatial_dropout1d (SpatialDr (None, 400, 64)           0

conv1d (Conv1D)              (None, 398, 256)          49408

global_max_pooling1d (Global (None, 256)               0

dense (Dense)                (None, 256)               65792

dropout (Dropout)            (None, 256)               0

dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 435,457
Trainable params: 435,457
Non-trainable params: 0
```

Fig. 1 Parameters of CNN network

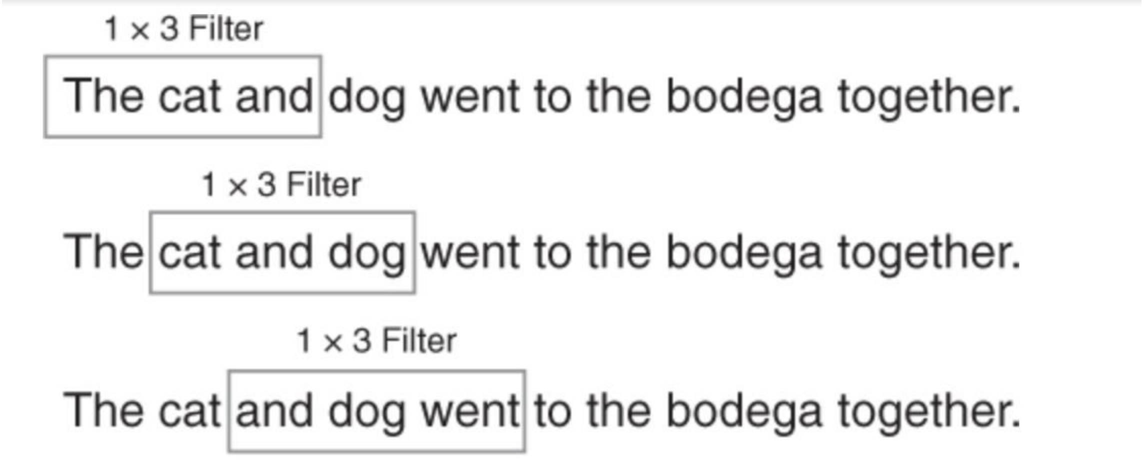

1 × 3 Filter
The cat and dog went to the bodega together.
1 × 3 Filter
The cat and dog went to the bodega together.
1 × 3 Filter
The cat and dog went to the bodega together.

Fig. 2 CNN kernel sliding over text

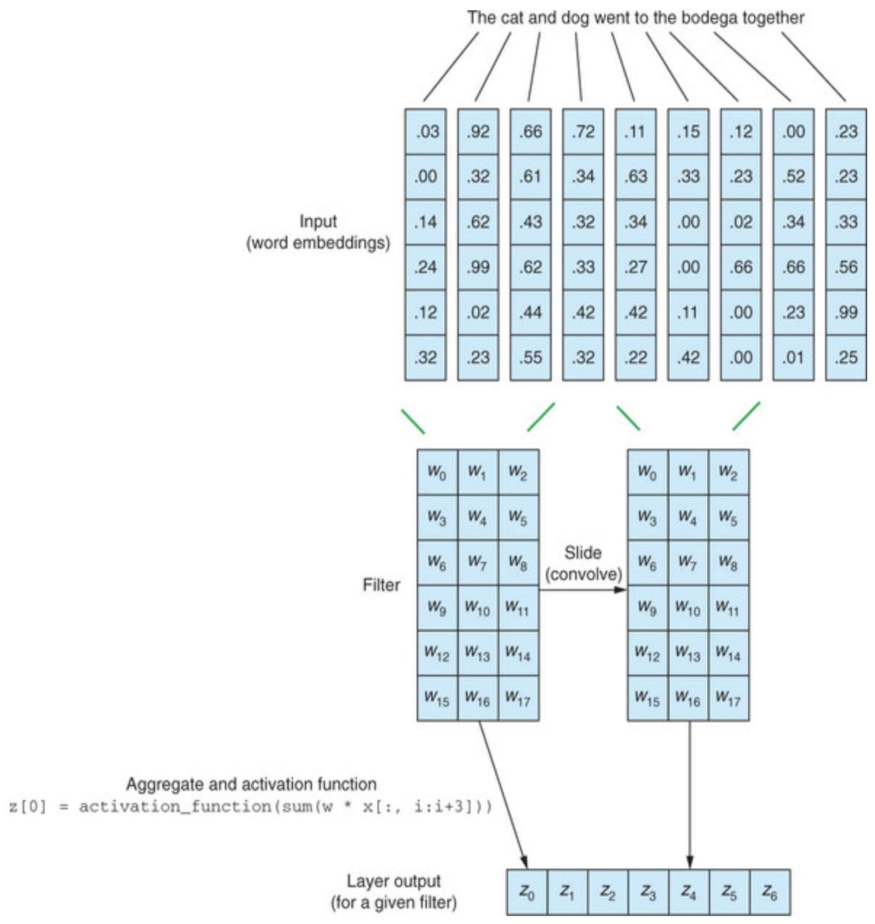

The cat and dog went to the bodega together
Input (word embeddings)
Filter
Slide (convolve)
Aggregate and activation function
z[0] = activation_function(sum(w * x[:, i:i+3]))
Layer output (for a given filter)

Fig. 3 Working of CNN in text classification

3.4. LSTM

LSTMs are a kind of RNN that maintains a state for each layer in the recurrent network. They overcome a challenging problem in the RNNs i.e. Vanishing/Exploding gradient. The modern version of LSTM typically uses a special neural network unit called a "Gated recurrent unit (GRU)". LSTMs contain memory for each layer, which is governed by a trained neural network. These neural networks can be trained to learn what to remember.

Memory can be used to learn dependencies between tokens that stretch across the entirety of data samples, such as sentences or documents. An LSTM cell contains 3 gates and a memory unit. The gates in LSTM cells contain neurons inside them. The gates in the LSTM network are:

3.4.1. Forget Gate

The forget gate determines how much cellular memory you want to erase. The forget gate contains a feed-forward neural network that outputs a vector of values between 0 and 1 using a sigmoid activation function. Forget gate output is further used to modify the values of the memory vector.

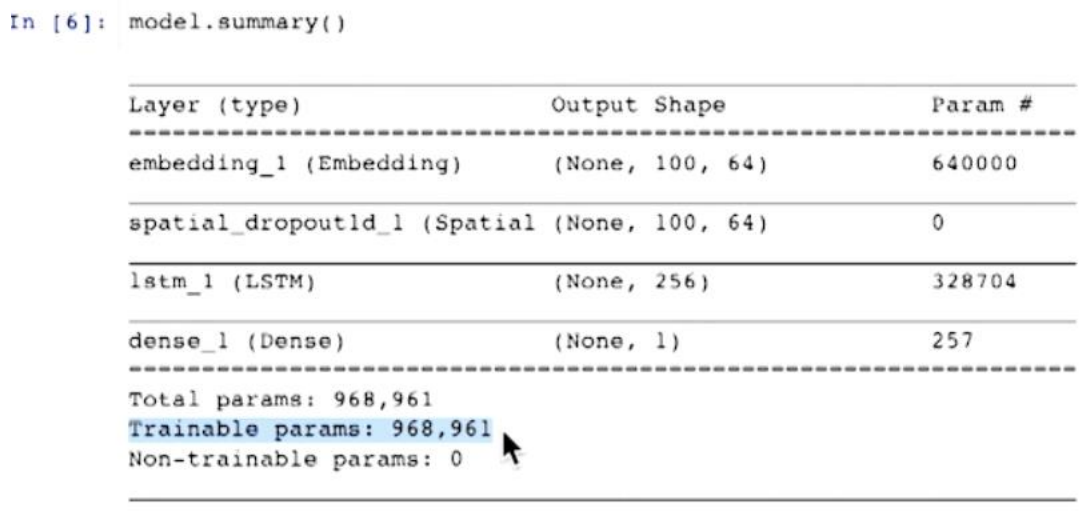
3.4.2. Candidate Gate

This gate has 2 neurons and performs the following: 1). Decided which input vector element needs to be remembered 2). Route the remember input element to the right memory.
3). Output gate: The output of the LSTM cell is formed with the help of memory.

3.4.3. LSTM Model Hyperparameters

- Epochs: 4
- Batch size: 128
- Word embedding vector dimension: 64
- Number of unique words: 10000
- Max review length: 100
- pad type = trunc type = 'pre'
- drop embedding = 0.2
- number of LSTM cells = 256
- Dropout LSTM layer = 0.2
- Size of dense layer = 256
- Dropout dense layer = 0.2

3.4.4. Trained LSTM model Hyperparameter Analysis



```
In [6]: model.summary()

Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 64)           640000

spatial_dropout1d_1 (Spatial (None, 100, 64)           0

lstm_1 (LSTM)                (None, 256)               328704

dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 968,961
Trainable params: 968,961
Non-trainable params: 0
```

Fig. 4 Parameters of the LSTM network

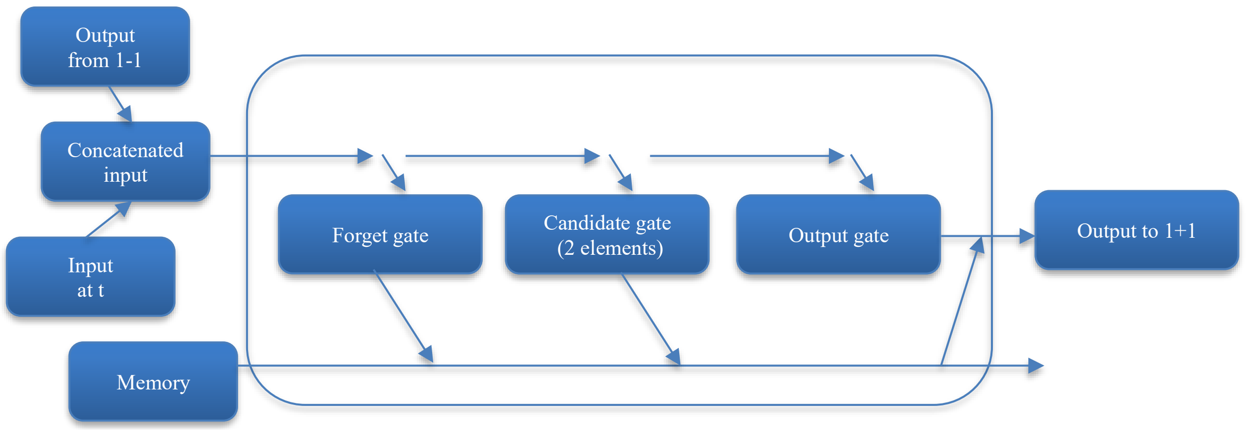


Fig. 5 Working of LSTM network

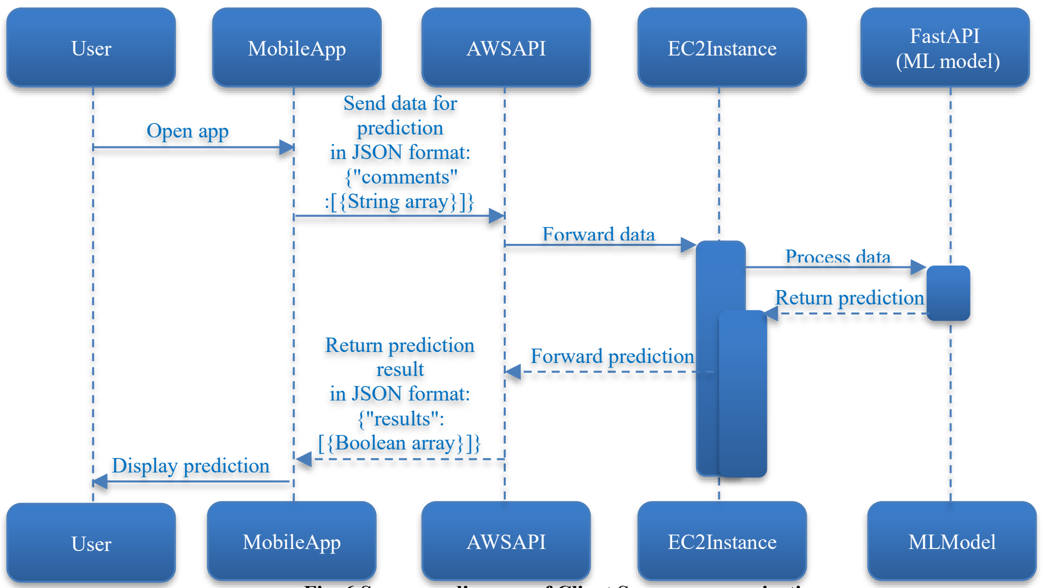


Fig. 6 Sequence diagram of Client Server communication

4. Deployment

4.1. Deploy model to AWS EC2 instance

For integrating the model's functionality into mobile apps, the model is deployed on an AWS EC2 instance [6]. Python's Fast API framework is used to expose a POST endpoint "/predict" that accepts a JSON payload with key "comments". Value for the key comments is a String array containing comments that the mobile app wants to classify as either expletive or non-expletive.

The response from the POST /predict API is a Boolean array where any Boolean values depict whether the String in the input at the corresponding index as a Boolean value is an expletive comment or not.

4.2. Mobile App Setup

Mobile apps will use the endpoint POST /predict to classify user comments as expletive or not. The mobile app will send the array of user comments to the POST API for classification. As a response, the mobile app will receive an array of Boolean values where true indicates that the corresponding comment is expletive. These Boolean values can be used to hide the expletive comments.

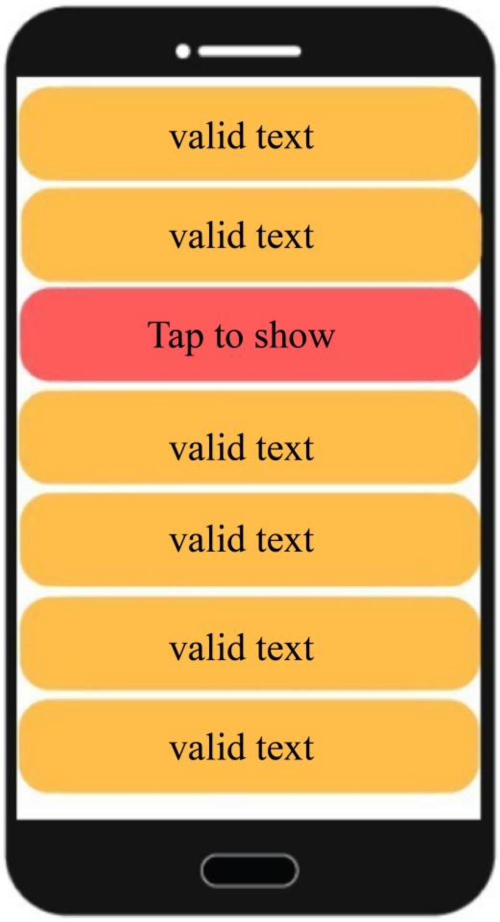


Fig. 7 Mobile app UI

5. Validation and Results

The performance of both models is measured on the validation data using the ROC AUC (Area under the receiver operating characteristic curve) metric. The following scores were found for the models:

Table 3. ROC-AUC scores for CNN and LSTM classification models

Model	ROC-AUC%
CNN	95.9
LSTM	93

The performance of the CNN model is better than that of the LSTM model on validation data in the experiment, even though LSTM is a more powerful algorithm for Natural Language Processing models. This is due to the superior ability to link context between tokens in large sentences and document text.

Further analysis suggests that the CNN model overperformed the LSTM model on validation data because of the small amount of data used for training the models. If the models are trained on a large amount of data, the LSTM model would outperform the CNN model.

6. Discussion

In this research, CNN and LSTM DL models were compared for expletive language classification. This research used a small data set to train these models. CNN model outperforms the LSTM model on validation data using the ROC AUC score. The LSTM model will perform better than the CNN model if trained on a larger dataset since LSTM has a better ability to remember context.

There are better DL algorithms that can be used to improve the performance even more, like Transformers, which can handle multilingual comments for expletive language classification, unlike CNN or LSTM models.

The dataset has biases that negatively impact the performance of models. Creating a bias-free dataset can improve the trained model's performance. Mobile apps of eCommerce companies are used all over the world and support multiple locales. Training models in multiple languages can increase the usefulness of the model.

7. Conclusion

It was found that the ROC-AUC score was better for the CNN-trained model compared to the LSTM model. It is also concluded that if we had increased the training data, then the LSTM model would have outperformed the CNN model. This is because the LSTM model is capable of remembering the context throughout large text samples and documents. CNN model is only capable of remembering context among nearby tokens.

References

[1] Jon Krohn, *Deep Learning for Natural Language Processing*, 2nd ed., Pearson, 2020. [Publisher Link]

[2] Metrics For Evaluating Machine Learning Classification Models, Medium, Towards Data Science, 2024. [Online]. Available: https://towardsdatascience.com/metrics-for-evaluating-machine-learning-classification-models-python-example-59b905e079a5

[3] Long Short-Term Memory, Wikipedia, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[4] Convolutional Neural Network, Wikipedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network

[5] Tomas Mikolov et al., "Efficient Estimation of Word Representations in Vector Space," *Arxiv Preprint*, pp. 1-12, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mourad Mars, "From Word Embeddings to Pre-Trained Language Models: A State-of-the-Art Walkthrough," *Applied Sciences*, vol. 12, no. 17, pp. 1-19, 2022. [CrossRef] [Google Scholar] [Publisher Link]