Identification of Toxicity in Speech

by
Gargi Roy

Examination Roll No.:- MCA226029

Registration No.:- 136830 of 2016-17

Class Roll No.:- 001910503030

Session: 2019 - 2022

This dissertation is submitted in partial fulfillment
of the requirements for the degree of
Master of Computer Application

Under the Guidance and Supervision of
Dr. Sudip Kumar Naskar
Department of Computer Science & Engineering
Jadavpur University
Kolkata-700032

2022

Declaration Of Authorship

I, hereby declare that this dissertation contains literature survey and original research work by the undersigned candidate, as part of Master of Computer Application studies.
All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.
I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work.

Sign: ____________________________ **Date:** ____________________________
Name: Gargi Roy
Examination Roll No.: MCA226029
Registration No.: 136830 of 2016-17
Thesis Title: Identification of Toxicity in Speech.

Certificate of Recommendation

This is to certify that the dissertation entitled "Identification of Toxicity in Speech" has been carried out by Gargi Roy (University Registration No.: 136830 of 2016-17, Examination Roll No.: MCA226029) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master of Computer Application. The research results presented in the thesis have not been included in any other thesis submitted for the award of any degree in any other University or Institute.

Dr. Sudip Kumar Naskar
Department of Computer Science & Engineering
Jadavpur University, Kolkata-700032

Signature: ____________________________

Prof. Anupam Sinha
Head of Department
Department of Computer Science & Engineering
Jadavpur University, Kolkata-700032

Signature: ____________________________

Prof. Chandan Majumder
Dean
Faculty of Engineering & Technology
Jadavpur University, Kolkata-700032

Signature: ____________________________

Certificate Of Approval

This is to certify that the thesis entitled "Identification of Toxicity in Speech" is a bonafide record of work carried out by Gargi Roy **(University Registration No: 136830 of 2016-17, Examination Roll No: MCA226029)** in partial fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University, during the period of January 2022 to June 2022. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, the opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Examiner:

Signature: ____________________ **Date:** ____________________

Dr. Sudip Kr. Naskar
Department of Computer Science & Engineering
Jadavpur University, Kolkata-700032

Signature: ____________________ **Date:** ____________________

Acknowledgement

I would like to express my sincere gratitude to my advisor, **Dr. Sudip Kumar Naskar**, for his continuous motivation, guidance and patience throughout my thesis work. I have been very lucky to have an advisor who cared so much about my work. More importantly, the scientific and personal support along with his valuable suggestions softened the journey.
I am also thankful to all other NLP lab members for their continuous support.

Gargi Roy
Signature: ____________________________

Abstract

Communication of information is apparently instantaneous nowadays than ever before. With the advent of social media platforms, such as, online forums, and networking websites, interactions between people today is at an all-time high. In these interactions, even though the primary form of passing information is text, rapid technological developments have allowed people to communicate instantly over the internet using audio and visual modes as well. With the increase in interactions, there has also been an increase in the form of violence that are incited due to the spread of hatred through speech, and has many times proved dangerous if not prevented at the source. Over the years, toxicity detection has been a popular field of interest for research work, however toxicity detection in speech remains relatively unexplored.

Since speech is the one mode of communication that is, by far, unique to mostly humans, the recognition of vicious and spiteful elements that are spread through speech is an important step in preventing violent, hateful crimes. An apparent solution is the application of Natural Language Processing. Natural Language Processing is a technique that is popularly used to work with language in the way it is written or spoken by humans. Combining linguistics and artificial intelligence, it paves the way for a sufficiently reliable solution. In this thesis, we have attempted to investigate the identification of offensive elements in speech using some of the methods from Natural Language Processing, and presented our findings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Speech

Human species is capable of an advanced mode of communication, known as speech, which is, by far, an extremely rare occurrence in other species of animals. Speech allows us to our convey thoughts and ideas of different complexities to another human being. Even though this feature allows us to distinguish ourselves from other animals, speech can also to be used against another human being as an offensive.

1.1.1 Hate Speech

Speech, when used against another person or a group of persons to denigrate on the basis of race, ethnicity, gender, sexual orientation, religion, age, physical or mental disability, and others - is known as hate speech. Hate speech can show up in any form of transmission of speech, and is used to incite hatred or violence against other individuals or groups based on the above stated factors.

The spread of hate speech to incite violence amongst marginalised communities has seen an explosive rise, thanks to the internet and social media, which gives anyone the freedom to share their thoughts and opinions on any matter, even if it can potentially cause harm to other people. The internet gives individuals the advantage of anonymity. This enables people, and also bots these days, to create fake personas, and spread hatred and violence. Hate speech can take on many forms - spams, cyber-bullying, and misinformation - to point a few. Even though big tech and social media giants have taken steps to combat hate speech online, it has had little effect on the problems that plague the society still.

1.1.2 Detecting Hate Speech

A lot many machine learning models and frameworks using hate speech detection algorithms are currently in use by the different online platforms, and there are rules that apply to the different platforms, which, when violated, can cause suspension of accounts or permanent bans. However, they can be bypassed easily most of the time. Hence, better ways to detect and remove hate speech from online content are needed all the more day by day.

Current hate speech detection methods mainly use text to detect inciting content. This uses content from posts, comments, replies, etc., to detect and remove hate speech. However, hate speech can also be spread as audio, in audio or video format. Since, most of the research combating hate speech is based on textual format, hate speech detection from audio or video files remain relatively unexplored. This study aims at detecting hate speech from voices in audio formats.

1.1.3 Hate Speech Detection Approaches

Over the years, many approaches have been used for detecting hateful content online (MacAvaney et al. 2019). This consists of detecting keywords in the content which are defined as offensive, identifying hateful content in the metadata of contents, or, the one that is currently more popular - using machine learning to learn and recognise vicious or inciting content from data.

Detection using Keywords

This approach uses a dictionary of keywords that are marked as offensive, and all the words in the content being checked is compared to these keywords. If such keywords are detected, the content is marked as offensive. This technique is fast and very straightforward, but presents with a lot of challenges, as the keywords that are present in the dictionary against which the content is being checked can vary from person to person. Moreover, offense is highly subjective. It can be subtle and harm can be done without using explicit swear words. The keyword-based approach fails to consider those cases.

Detection using Source Metadata

Similar to the keyword-based approach, this technique uses additional information to understand the content, context, and the demographics to which it is targeted to. Information about the people or group that post such content, such as location, timings, and user groups, can also be used to identify and check the spread of offensive content.

Detection using Machine Learning

This approach involves the use of various machine learning algorithms to classify hateful content, based on the information it learns from content that is previously labelled by content reviewers as offensive or harmful. The machine learning models learn the features which correlate to the content being toxic or not, and then identify new content as such, based on the features present in those.

1.2 Objectives

This thesis furthers the work of detecting toxic features in speech using the machine learning approach. It offers a glimpse into what can be done to detect the presence of hateful utterances in speech. Compared to most of the previous work done on the identification of toxicity in text or videos, this work is a study applying the same procedures on utterances, and classifying them as hateful or not.

First, we present a brief summary of the previous research done in this area in chapter 2. Chapter 3 discusses feature extraction methodologies used, and the model architectures that were implemented by us. Chapter 4 explains the description of the set-up that was used by us to carry out the experiments. And finally, chapter 5 presents the results, with concluding remarks in chapter 6.

Chapter 2

Literature Survey

Since the past few years, a lot of research has been done to deal with the problem of hate speech, and solutions have been devised to combat it. Almost all of these solutions implement some machine learning or deep learning algorithm. However, the work done on toxicity detection from speech by extraction of audio features has not been that common.

2.1 Hate-Speech Detection in Text

Njagi et al. (2015) create a classification model that implements sentiment analysis techniques to detect hate speech. In the paper, hate speech is categorised into three main classes of *race*, *nationality*, and *religion*. The classifier in the research performs subjectivity detection hate speech, as well as rates the polarity of the sentiment expressions. Using semantic and subjective features of the hate speech, the authors successfully generate lexicons that relate to hate speech, and use it to classify blog posts into *strongly hateful*, *weakly hateful*, and *not hateful* classes. It begins by the isolation of sentences that have subjective expressions from those that express objective sentiments. This is intended to make detection of opinions easier. Then, the authors go on to create a lexicon of words related to hate speech using semantic features from the corpus and subjective features identified from the texts. In the third and final stage, the authors create a classifier that uses the features generated previously, and utilise it to identify hate speech in documents.

The authors used standard evaluation metrics, such as, precision, recall, and F-score for evaluation of the classifier. The evaluation method used is incremental, and focused on the effects of the different feature-sets. The authors then compare the results by using all sentences from their annotated corpus. One added feature of their research is the extension of the domain of topics that are considered in hate speech. The authors have also included abuses towards ethnicity and religion in the domain of hate speech.

Another researcher (Biere, Bhulai, and Analytics 2018) discuss the use of Natural Language Processing techniques to detect hate speech in tweets. The authors applied deep learning, more precisely, a Convolutional Neural Network (CNN) model, and classified each tweet as belonging to one of the classes: *hate*, *offensive language*, and *neither*. To normalise the content of each tweet, pre-processing was used. It includes removing characters, lowercase and stemming, and, splitting text into tokens.

The CNN architecture used in this paper has a non-linear convolution layer (Albawi, Mohammed, and Al-Zawi 2017), a max-pooling layer (Gholamalinezhad and Khosravi 2020), and a softmax layer (Iwana, Kuroki, and Uchida 2019). It also uses Adam algorithm to train the CNN model. In conclusion, the model was able to predict each class with an accuracy of 91%. However, due to insufficient data, 80% of the data was mis-classified, showing a tendency to be biased towards categorising tweets as hate speech. Overall, the paper shows the potential of using Convolutional Neural Networks as good classifiers of hate speech if a reasonably large data-set is present.

Warner and Hirschberg (2012) discuss an approach of using linear Support Vector Machine (SVM) classifier for detecting hate speech in online text content. The authors used a template-based strategy to create features from the corpus, with each template centred around a single word. Parts-of-Speech tagging of each sentence on *Yahoo! News Group* posts was used, which provided the parts-of-speech windows as features.

These features were fed in to an Support Vector Machine classifier, with 1 as a sign of anti-semitic, and -1 for otherwise. The authors were able to establish a baseline accuracy by finding the accuracy of assuming the majority of classification always. In conclusion, they were able to successfully model hate speech as a classification problem, and their unigram feature sets outperformed the entire set, and the feature set that was the smallest, and comprised of positive unigrams only, performed the best.

Georgakopoulos et al. (2020), propose a model for tracking of toxic comments on twitter over time. All tweets related certain hashtags were parsed, and used as input for this experiment. A Convolutional Neural Network classifier model was designed, and trained for hateful words in tweets. The authors also implemented a change detection approach to observe any change in the toxic trend within the tweets using those hashtags. They were able to demonstrate that the identification of toxicity has the potential to provide significant information, most of which are otherwise unable to be found by using other sentiment-based methods. Similarly, the authors (Hashida, Tamura, and Sakai 2018) demonstrated the classification performance of a deep-learning-based classification method that implemented multi-channel distributed representation, a new distributed representation for words. It was found to be much better than using a naive Bayes classifier. Along with that, the authors also showed that a Convolutional Neural Network with multi-channel distributed representation can identify toxicity in tweets much better than a Convolutional Neural Network without multi-channel distributed representation.The authors Silva, Coletta, and Hruschka (2016) performed a survey on sentiment analysis on tweets, using semi-supervised learning. It includes topic-based, wrapper-based, and graph-based classification techniques. The survey also includes a comparative study of algorithms based on different approaches: self-training, co-training, distant supervision, and topic modeling. The authors are able to throw light on the biases that need to be considered while designing real-world applications.

2.2 Hate-Speech Detection in Videos

Wu and Bhandary (2020) explores a methodology to implement hate speech detection in videos, based on the spoken content. The audio is extracted from the videos, then a transcript is obtained on which the hate speech detection models are applied. Three different feature sets are extracted from the data-set, which are used to train four models:

1. Naive Bayes Classifier
2. Random Forest Classifier
3. Linear Support Vector Machines
4. Recurrent Neural Network

This paper goes on to describe two different experiments to compare the results of the different models. One experiment dealt with the classification of videos into normal and hateful categories. The other one dealt with the classification of the videos into normal, racist, and sexist categories. The data-set was split in the ratio 70:30 for training and testing purposes, before feature extraction was done. In the feature extraction process, the word counts, their frequencies, and n-grams were extracted. Evaluation of the models was done using metrics, such as, accuracy, precision score, recall score, and F1 score. The paper concludes with the result that Random Forrest Classifier model provides much better classification than the remaining models.

Razavi et al. (2010) discusses the implementation of an automatic flame detection model that does feature extraction at various levels of concepts, and applies multi-level classification to detect rants, taunts, and squalid phrases in textual content. A message is considered a “flame” if the main intent of it is *to be hostile*, or *abusive*, or *to attack*. Training was done on 90% of the data-set, and the remaining 10% was used for testing.

The research describes a three-level classifier that consists of a Complement Naive Bayes classifier as the first level. It selects the most discriminatory features as the new training feature space, and passes it to the next level - a Multinomial Updatable Naive Bayes Classifier (Mccallum and Nigam 1998). In this level, new labelled sentences were added for adaptive learning. The result of this level were new aggregated features which were extracted from the features space of the previous level. Finally, in the third level, a Decision Table and Naive Bayes hybrid Classifier was used, which helped improve the accuracy of the classification by a good 6%. The choice of classifiers was based on various factors, such as, adaptability for online applications, and time-efficiency. The classifiers providing the most discriminatory power were considered for this paper. It is complemented by an *Insulting or Abusive Language Dictionary*, a dictionary containing words, phrases, and expression patterns for abusive text detection.

In Barakat, Ritz, and Stirling (2012), a Dynamic Time Wrapping (DTW) based approach is used in the detection of hate speech in video blogs using audios as features. It discusses a speaker independent keyword spotting (KWS) approach that is applied on the audio content of video blogs of users to help in the automatic analysis, indexing, search, and traversal. The keyword templates are then compared to a segment of speech. For this, initially, extraction of features is done from both keywords and utterance. Mel-Frequency Cepstral Coefficients (MFCCs) are used as the feature extraction techniques. Comparison of the extracted feature vectors from the template and the utterance is done using automatic determination of the threshold for the distance between the vectors.

For this, the authors selected a word template randomly from outside the test database, for each keyword. This template was then used to retrieve the utterances that had the keyword in it. Then, evaluation metrics precision and recall were used to evaluate the performance of this technique. The authors were able to obtain a maximum recall of 93.75%, whereas a maximum precision of 42%, indicating that the technique gives a higher recall and a lower precision. When applied on user video blogs, this technique gave a maximum recall of 80% with a precision of 45.4%. This shows that keywords can be detected from video blogs without the use of language information or extensive training of the models.

In Kandakatla (2016), the authors discuss a framework for determining hateful content in YouTube videos. The data-set used for training the classifier consisted of videos from YouTube containing offensive audio. From the data-set, 90% videos were used as training data, and the rest 10% were used as test data. Feature extraction was done using two methods: Comment-based and Metadata-based. The authors then used sentiment analysis on the extracted textual features to understand the sentiment of the features. The sentiment was categorised into three classes: *positive*, *negative*, and *neutral.*

The authors then applied Naive Bayes and Support Vector Machine classifiers to detect offensive content from the extracted features. The results were compared by evaluating the F-scores of the various extracted feature vectors using the two classification models. The F-scores achieved from both the classifiers were comparable, with the maximum F-score was 0.86 from the SVM classifier.

2.3 Hate-Speech Detection in Speech

Spoken Language Processing task has remained an important problem that has not yet been solved for use in interactive intelligent systems. It also has various utilisation in the field of content moderation – in audio and video content online, customer service, gaming forums, and so on.

In Xie et al. (2019), the authors proposed a method applying frame-level speech features along with attention based Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNN) for recognition of emotion in human speech. They identified a range of emotions in separate frames by improving upon the existing Long Short-Term Memory model (Hochreiter and Schmidhuber 1997) using the mechanism of attention. The authors introduced the attention mechanism into the forget gate of the LSTM as well as the output of the LSTM. The new and improved gate is related to the historical cell state only. It is not dependent on the current input. This helps reduce the computational complexity.

The results showed that the rate of recognition increased. The unweighted average recall also improved quite a lot on some datasets. This indicates the advantage of using the modified attention based LSTM for better categorising of emotions using frame-level features.

In Lakomkin et al. (2019), the authors argue that the usage of ground-truth transcriptions and evaluation phases results in a major inconsistency in the performance of the system, as compared to real-world scenarios. This is because recognition of speech has to be done on-the-go, and can have mistakes due to that. To combat that, the authors propose a technique that applies Automatic Speech Recognition on a character-level recurrent neural network for sentiment analysis. The authors also perform various experiments based on the recognition of sentiments for interaction between humans and robots in noisy, realistic situations.

The authors were able to find improvements when weighted up against the acoustic modality in the recognition of sentiment only. They were able to achieve an accuracy of 73.6% in a classification task consisting of only binary sentiments, by adding acoustic features, and using a mixture of their own automatic speech recognition model and Google's automatic speech recognition model, demonstrating a performance gap of 1% better accuracy and 1.3% F-score against the best-reported model that uses linguistic and acoustic model.

In Ghosh et al. (2021), the authors propose a novel task of classifying toxic elements in spoken language. The authors also present the first annotated dataset for detection of hateful content in speech in English language. The authors describe toxicity as *rude*, *disrespectful*, or *otherwise likely to make someone leave a discussion.*

For annotating the dataset, the authors have used a two-step process, where, initially, they model a toxicity classifier for text content and train it using the BERT base model (Devlin et al. 2018). This was utilised to filter out the datasets that were found to have greater than or equal to 10% of its entire utterances marked as toxic by the model. Then, the authors use two different approaches: 2-Step approach and End-to-End approach.

1. **2-step approach**: The 2-step approach had two main parts - an Automatic Speech Recognition component and a Sequence Classification Model. The automatic speech recognition model was used to produce the transcriptions from spoken speech. It had a transformer architecture, along with a Convolutional Neural Network layer for feature encoding, similar to (Baevski et al. 2020). The two-step training involved first pre-training the model using unlabelled audio. This was self-supervised. Then, Connectionist Temporal Classification was applied on labelled speech data for fine-tuning of the model. The second step of the 2-step approach involved using the BERT base transformer model, and fine-tuning it on a large classification dataset for hate speech.

2. **End-to-End approach**: The End-to-End approach had two implementations, one using Mel Filter Banks, and the other using Transformers. The implementation using filter banks gave the probability of toxicity for each utterance, given an audio input sequence. The other implementation had extra fine-tuning of the parameters of the transformer architecture. Both of them were trained using Cross-Entropy as the loss function, and Softmax as the final activation function.

Chapter 3

Methodology

3.1 Feature Extraction Techniques

Automatic Speech recognition(ASR), also known as Speech Recognition, computer speech recognition, or speech-to-text, is the process of identification of human speech, and converting it into written format, all done by a computer program. It involves topics from various fields, such as, mathematics, statistics, and linguistics. Speech recognizers consist of different parts, namely – the speech input, feature extraction that forms feature vectors, then a decoder, and a word output (*What is Speech Recognition?-IBM* 2020). This section is mainly focused on feature extraction and its many methods. Feature extraction methods involve analysis of the speech signal. Broadly, the methods can be categorised into two parts (Iqbal 2018) - temporal analysis, which analyzes the waveform of the speech signal, and spectral analysis, which analyzes the speech signal's spectral representation. Power Estimation and Fundamental Frequency Estimation are some Temporal Analysis techniques, whereas Linear Predictive Coding analysis, Cepstral Analysis, Mel-Frequency Cepstral Analysis, Filter Banks, Perceptually Based Linear Predictive Analysis, and so on, are included in Spectral Analysis techniques. This section discusses the various Spectral Analysis techniques. Then, it discusses the advantages of Filter Banks over the other techniques, and how it can be used.

3.1.1 Linear Prediction Coefficients (LPC)

This method is based on the source-filter model of speech production, and is one of the most widely used speech signal analysis methods which give good speech features. LPC tries to recreate the source signal, and estimates the formant frequencies. These formants are then used to create a filter.

The filter is then applied on the source signal, and an approximation of the original speech signal is obtained. LPC takes the preceding speech samples, and predicts the speech signal at any point in time using the linear weighted aggregate of the samples. The prediction error is then calculated between the original signal and the estimated signal, the square of which is minimised to get an unique set of coefficients which are used in the filter.

In LPC, the speech signal is divided into short frames of a number of samples. Then, a certain number of samples are selected at a time for analysis. This is called windowing. Hereafter, each frame of the signal is auto-correlated, following which the LPCs are generated.

LPC is based upon the idea that the n^{th} speech sample can be obtained by a combining the previous p samples linearly (Vishnubhatla 1992). Considering a signal $s(t)$ sampled in time every T s, such that $s(n) = s(nT)$ for an integer n. Similar to the Laplace Transform, the spectral representation of this signal is called the z-transform $S(z)$.

The function of LPC analysis is to "deconvolve" the excitation and filter estimates, such that the product does not deviate much from the original $S(z)$.

The excitation source is considered as a flat spectral envelope, and the filter contains the relevant spectral details. A flat source spectrum is a proper assumption in case of speech signals, since, for the noisy "unvoiced" sounds, the excitation bears similarities with white noise. Again, for periodic "voiced" sounds, the source is considered as a uniform, periodic, N-sample train, that has a line spectrum and uniform-area harmonics.

The signal $s(n)$ is considered to be stationary during a frame of N samples, which is aimed to simplify the calculation for finding an estimate of the filter, $\hat{H}(z)$, when $s(n)$ is provided. The filter $H(z)$ is then modeled with constant coefficients, called predictor coefficients, which are estimated for every frame. The predictor coefficients are represented by a_k, and the gain factor for the excitation is given by G.

$$H(z) = \frac{G}{1 - \sum a_k z - k} \tag{3.1}$$

Hence,

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + a_3 s(n-3) + \cdots + a_p s(n-p) \tag{3.2}$$

where $a_1, a_2, a_3, \ldots$ are the predictor coefficients.

The error refers to the difference between the actual and the predicted speech samples.

$$e(n) = s(n) - \hat{s}(n) - \sum_{k=1}^{p} a_k s(n-k) \tag{3.3}$$

where, $e(n)$ is the error in prediction, $s(n)$ is the original speech signal, $\hat{s}(n)$ is a predicted speech signal, $a_k s$ are the predictor coefficients.

LPC also determines the accurate frequency/pitch period by including a pitch-detecting algorithm. The predictor coefficients, the gain, and the pitch do not remain constant with time, from one frame to another. However, the predictor coefficients exhibit high variance during experiments, and hence, are not used in recognition very often. In practice, the cepstral coefficients are used, which are a better suited feature.

Performance analysis of LPCs generally involve the bit rates, the overall delay of the arrangement, the computational complexity, and lastly, an objective metric.

Depending on various factors, the LPCs can be of different types – Voice-excitation LPC, Residual Excitation LPC, Pitch Excitation LPC, Multiple Excitation LPC (MPLPC), Regular Pulse Excited LPC (RPLP), and Coded Excited LPC (CELP).

3.1.2 Linear Prediction Cepstral Coefficients (LPCC)

The cepstral coefficients derived from either linear prediction (LP) analysis or a filter bank approach are almost treated as standard front end features. Speech systems developed based on these features have achieved a very high level of accuracy, for speech recorded in a clean environment. The methodology of LPCC is almost the same as that of LPC. As stated previously, the fundamental idea behind LPC is that the n^{th} speech sample can be derived by combining the previous p samples linearly.

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + a_3 s(n-3) + \cdots + a_p s(n-p) \tag{3.4}$$

where, $a_1, a_2, a_3, \ldots,$ are assumed to be constants over a speech analysis frame.

The speech signal is initially divided into frames and windowed. Autocorrelation is then applied on the windowed frames, and the result is acted upon by LPC analysis. Then, the extra step of converting from LPC to LPCC is done, following which the LPCCs are generated. LPCCs are used to capture emotion-specific information, manifested through vocal tract features.

LPCC are cepstral coefficients that are derived from the spectral envelope of the LPC analysis. The cepstral coefficients are obtained using linear prediction analysis of a speech signal. They are derived from spectral coefficients by applying Fourier Transform to the logarithms of the spectral envelope (Alim and Rashid 2018).

$$C_0 = \log_e p \tag{3.5}$$

$$C_m = a_m + \sum_{k=1}^{m-1} \frac{k}{m} C_k a_{m-k}, for 1 < m < p \tag{3.6}$$

$$C_m = \sum_{k=m-p}^{m-1} \frac{k}{m} C_k a_{m-k}, for m > p \tag{3.7}$$

3.1.3 Line Spectral Frequencies (LSF)

LSF is another method of analysing the speech signals. The LSF representation is a different way of representing the LPCs. The individual lines of the Line Spectral Pairs are called line spectral frequencies (LSF). Initial steps of this technique are the same as that of the LPC method. It takes into account the two resonant conditions in the inter-connected tube model of the human vocal tract, that is, the shape of the mouth and the nasal cavity. The two resonance situations define the vocal tract as either being completely open or completely closed at the glottis. The two situations begets two groups of resonant frequencies, with the number of resonances in each group being deduced from the quantity of linked tubes. The resonances of each situation are the odd and even line spectra correspondingly, and are interwoven into a singularly rising group of LSF.

LP analysis gives the equation (Alim and Rashid 2018)

$$\hat{s}(n) = \sum_{k=1}^{p} a_k s(n-k) \tag{3.8}$$

where, k is the time index, p is the order of the linear prediction, $\hat{s}(n)$ is the predictor signal, and a_k are the LPC coefficients. The coefficients are determined by the method of auto-correlation or covariance, in order to reduce the prediction error. Further in LSF, the inverse filter polynomial is decomposed into a symmetric polynomial, $P(z)$, and an asymmetric polynomial, $Q(z)$. The roots of the two polynomials give the LSF coefficients (Alim and Rashid 2018)

$$P(z) = A(z) + z^{-(p+1)} A(z^{-1}) \tag{3.9}$$

$$Q(z) = A(z) - z^{-(p+1)}A(z^{-1}) \tag{3.10}$$

where, $P(z)$ is the vocal tract with the glottis closed, and $Q(z)$ is the LPC analysis filter of order p.

LSF is used mostly in speech compression, although, speech recognition and speaker recognition also utilise this technique. LSF is also used in musical instrument recognition and coding, animal noise identification, and financial market analysis.

LSF has the capability to characterize bandwidths and resonance locations, and localize spectral sensitivities. In most instances, the LSF representation provides a near-minimal data set for subsequent classification.

A careful use of processing and analysis methods in the domain of Line Spectral Pairs can result in decrease in complexity than other methods that are applied on the raw input data. This is because Line spectral frequencies represent the spectral shape information at a lower rate of data, compared to raw input samples. Hence, they can have an important part in transmitting information pertaining to the vocal tract, from a speech coder to a decoder. This is also, partly, due to their great property of quantization. Computing line spectral pair parameters can be achieved using various techniques. These can vary in complexities. One main issue is that of calculating the roots of the polynomials P and Q defined earlier. However, these can be calculated by applying standard root solving methods, and it is usually applied while in the cosine domain.

3.1.4 Mel-Frequency Cepstral Coefficients (MFCCs)

The Mel-frequency scale is based on the fact that the frequency response inside the cochlear canal is not linear. It decreases as the frequency increases above 1kHz. Analysis using MFCCs is based on the workings of our inner ear. In this process, first, the speech signal is framed and windowed, and this signal is Fourier transformed to change to the frequency domain. Then, the log-spectrum is derived by applying logarithm to the FT signal. Mel-scaling is then applied to the log-spectrum to get the Mel spectrum of the speech signal. Finally, discrete cosine transform is applied to the Mel spectrum. This results in the MFCCs, which give information about the formant frequencies.

Pre-emphasis

This is a set of filters that highlight the frequencies at the higher end of the spectrum, and can minimise deviations in the spectrum of those kinds of voiced sound signals that have a roll-off with a high gradient in the region of higher frequencies. Usually, the glottal source has a slope of about -12 dB/octave, which can increase by +6 dB/octave when the acoustic energy is released from the lips. This is why speech signal recorded using microphones have a downward slope of nearly -6 dB/octave than the true spectrum of the vocal tract. The filter that is most commonly used for the purpose of pre-emphasis is expressed by the following equation, where b has a value between 0.4 and 1.0, and gives the slope of the filter (Picone 1993).

$$H(z) = 1 - bz^{-1} \tag{3.11}$$

Frame blocking and windowing

For extracting the Mel-Frequency Cepstral Coefficients, stable acoustic characteristics are required. Because of this, speech is generally observed over a very small period of time within which the speech signal is assumed to be stationary, since, speech signals are quasi-stationary, or slowly varies with time. These measurements are carried out over time frames of 20 ms, which proceed every 10 ms (Deller, Proakis, and Hansen 1993; Benesty, Sondhi, and Huang 2008). This allows the tracking of the time-varying characteristics of individual speech signals. The 20ms time window is enough for the proper spectral resolution of the signals, even though, being small enough for the resolution of temporal characteristics. Due to this overlapping analysis, every single speech sound of the input speech signal is nearly at the centre of some frame or the other. Next, a window is applied on every frame. This tapers the short signals towards the frame boundaries. Usually, in such cases, Hamming windows are used. They reduce the edge effect when the Discrete Fourier Transformation is applied on the signal, and also, smoothens the edges, and enhances the harmonics of the signal (Picone 1993).

DFT spectrum

Next, every single windowed frame is passed through a Discrete Fourier Transformation, and transformed into the magnitude spectrum, $X(k)$, where, N is the numbers of points considered while computing the Discrete Fourier Transforms.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi nk}{N}}; 0 \leq k \leq N-1 \tag{3.12}$$

Mel-spectrum

Next, from the Fourier Transformed signal, the Mel-Spectrum is calculated. This is done by applying a set of band-pass filters on the Fourier Transformed signal. This set of filters is called a Mel-Filter Bank. Here, the Mel-spectrum is used since the Mel scale corresponds to the gradient of perception of sound frequencies by the human ear. Similar to the non-linear perception of sound frequencies of the human ear, the Mel-scale does not particularly line-up linearly according to the physical frequency of the sound. Below 1 kHz, this frequency spacing of this scale is nearly linear, with this frequency spacing becoming logarithmic above 1 kHz. The relation between Mel frequency and physical frequency can be given by the following equation (Stevens, Volkmann, and Newman 1937), where, f denotes the physical frequency in Hz, and f_{mel} denotes the perceived frequency in the Mel-scale:

$$f_{mel} = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3.13}$$

Discrete Cosine Transform (DCT)

Before the Discrete Cosine Transformation is applied on the Mel-spectrum, it is represented on the logarithmic scale. The Discrete Cosine Transformation is applied on the Mel-frequency coefficients, which results in a set of cepstral coefficients. These coefficients correspond to the correlated energy levels of the vocal tract in the adjoining vocal bands. This gives a signal in the cepstral domain, which has a “que-frequency” peak. This que-frequency peak corresponds to the signal pitch and a number of low que-frequency formants. The system is made increasingly effective by selecting only those few Mel-Frequency Cepstral Coefficients that represent most of the signal. Generally, the first few coefficients are selected. The higher order components of the Discrete Cosine Transformation. Then, the Mel-Frequency Cepstral Coefficients are calculated as per the following equation (Deller, Proakis, and Hansen 1993):

$$c(n) = \sum_{m=0}^{M-1} \log_{10}(s(m)) cos\left(\frac{\pi n m}{M}\right) \tag{3.14}$$

where, $c(n)$ denotes the cepstral coefficients, and C denotes the number of coefficients. Usually, 8-13 coefficients are used, with the zeroth one not included, because it does not carry much speaker-specific information, and denotes the average low-energy of the input speech signal.

3.1.5 Filter Banks

This is another feature extraction technique which is similar to Mel Frequency Cepstral Coefficients. Gaining popularity day-by-day, filter banks and Mel Frequency Cepstral Coefficients are mostly the same in terms of steps. In case of filter banks, the filter bank coefficients are computed. Adding a few more extra steps to this, gives the Mel frequency cepstral coefficients. For computing filter bank coefficients, a pre-emphasis filter is applied on the signal in the first step. The signal is then split by applying framing and windowing. To the result of this, a Short-Time Fourier Transformation is applied. This results in the power spectrum. And lastly, the filter bank coefficients are calculated by applying a number of triangular-shaped filters on the result of the power spectrum (Fayek 2016). This results in information extracted about the formant frequencies.

Pre-Emphasis

In this initial step, the high frequencies of the signal are intensified by applying a pre-emphasis filter. The pre-emphasis filter can be applied to a signal $x(t)$ using the first order filter in the following equation to get the amplified signal $y(t)$ (Fayek 2016):

$$y(t) = x(t) - \alpha x(t-1) \tag{3.15}$$

Applications of this filter include:

1. Smoothing the frequency spectrum. This is because, compared to lower frequencies, higher frequencies generally have smaller magnitude.
2. Avoiding problems related to computation when applying the Fourier transformation, and
3. Enhancing the Signal-to-Noise ratio by decreasing the noise.

Framing

Following the pre-emphasis step, the signal is sliced into frames of short time intervals. This is done since the frequencies in a signal vary with time. So, applying the Fourier transformation over the full signal would not have much of an effect. This means, in that case, over time, we would not be able to retain the frequency contours of the signal. We can overcome this effect by assuming that the frequencies in a signal remain stationary over a very short time interval. Frame sizes are usually found to range between 20 ms to 40 ms, and have an overlap in between the adjacent frames of about 50% (+/-10%). A frame size of 25 ms and a stride of 10 ms are most commonly used. This indicates an overlap of 15 ms. Now, since the Fourier transformation was applied over a short-time interval, a satisfactory approximation of the frequency contours of the signal can be obtained by placing the consecutive frames one after the other and concatenating them.

Windowing

After the audio signal has been sliced into short-time frames, a windowing function is applied to every single frame. The windowing function is applied to reduce the spectral leakage, and, to cancel out the effect of the assumption which was made earlier by the Fast Fourier Transform, namely, that the data is infinite. In our experiment, the windowing function called the *Hamming window* was used. The expression for the Hamming window function is as follows:

$$hammingwindow[frame\_length] = w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \tag{3.16}$$

where, $0 \leq n \leq N-1$, N being the length of the window.

Figure 3.1 represents the graph of the Hamming Window function.

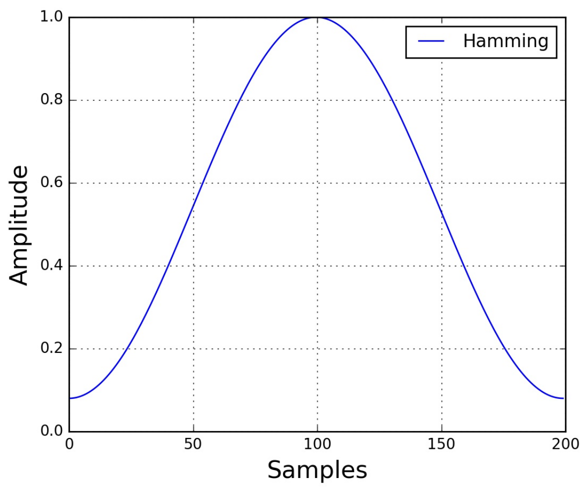

Figure 3.1: Graph of the Hamming Window Function (Fayek 2016)

Fourier-Transform and Power Spectrum

Next, the frequency spectrum of the frames of the audio signal are calculated. To do this, an N-point Fast Fourier transform, also known as a Short-Time Fourier Transform, is done on each windowed frame. The power spectrum is then calculated by using the equation (Fayek 2016):

$$P = \frac{|FFT(x_i)|^2}{N} \tag{3.17}$$

where x_i was the i^{th} frame of the signal x.

The Fast Fourier Transform FFT can be calculated using the following expression:

$$FFT = \sum_{n=0}^{N-1} x(n)e^{\frac{-i2\pi kn}{N}} \tag{3.18}$$

In the final step to calculate filter banks, triangular filters are applied to the frequency spectrum. These filters are applied on a Mel-scale to the power spectrum. This is done to extract the frequency bands. It has a higher discriminatory action for lower frequencies and lower discriminatory action at higher frequencies. The conversion between the more commonly used Hertz (f) scale, and the Mel (m) scale is done as per the equations below

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3.19}$$

$$f = 700(10^{\frac{m}{2595}} - 1) \tag{3.20}$$

Every single filter in the filter bank has a triangular shape, with a response of 1 at the frequency which is at the center, which decreases linearly towards 0, until it reaches the center frequencies of the two filters which are adjacent to it, and has a response of 0 at those points. The Mel-scale is applied since it is based on the non-linear perception of audio signals by the human ear. After applying the filter bank to the power spectrum of the signal, we obtain the filter bank coefficients.

When this filter bank is applied to the power spectrum of the audio signal previously obtained, a frequency spectrogram of the audio signal can be obtained. However, due to the high correlation of the coefficients of the filter bank, problems could be encountered in some machine learning algorithms. This problem is avoided using Discrete Cosine Transformation. Discrete Cosine Transformation reduces the correlation between the filter bank coefficients, and results in a representation of the filter banks which are compressed. Generally, for Automatic Speech Recognition purposes, the first 2-13 cepstral or filter bank coefficients are retained, and the remaining coefficients are discarded, since, the remaining coefficients do not contribute much to the classification. Then, a sinusoidal liftering is applied to the Mel-Frequency Cepstral coefficients for de-emphasizing higher MFCCs. It is claimed to improve speech recognition in noisy signals. Then Mean Normalization is performed, that is, the mean of each coefficient is subtracted from all the frames. This improves the Signal-to-Noise ration and balances the spectrum.

3.1.6 Perceptual Linear Prediction (PLP)

Perceptual linear prediction extracts information that is relevant from speech samples by combining the critical bands, intensity-to-loudness compression, and equal loudness pre-emphasis. It was initially intended to be used in speech recognition systems by keeping only the speaker independent features. This technique applies a nonlinear scale - the *Bark* scale. Perceptual linear prediction gives an expression conforming to a smoothed short-term spectrum. This expression is equalized and compressed, which makes it similar to the human hearing. This feature of Perceptual linear prediction is similar to the Mel scale of Mel Frequency Cepstral Coefficients. In case of Perceptual linear prediction, however, the replication of several prominent features of hearing takes place. An auto-regressive all-pole model approximates this consequent auditory like spectrum of speech. Perceptual linear prediction results in minimized resolution at high frequencies. This is similar to auditory filter bank based approach. However, it still gives the orthogonal outputs as results, that are similar to the cepstral analysis. To perform spectral smoothing, perceptual linear prediction uses linear prediction. This gives the name "Perceptual Linear Prediction". This technique combines two other techniques - linear prediction analysis, and spectral analysis.

For computing the perceptual linear prediction features, the speech is windowed using a windowing function. Then, Fast Fourier Transformation is applied to it, and the square of the magnitude is computed, resulting in the power spectral estimates. Next, integration of the overlapping critical band filter responses is done by applying a trapezoidal filter. This is applied at 1-bark interval in the power spectrum. This results in the effective compression of the higher frequencies into a narrow band. Then, the spectrum is smoothed by symmetric frequency domain convolution, applied on the Bark warped frequency, which, also allows the higher frequencies to be masked by the lower frequencies. Following this, the spectrum is pre-emphasized to approximate the non-linear sensitivity of human ears to different frequencies. The output is compressed to lower the variation in amplitude in the spectral resonances. Then, to obtain the auto-correlation coefficients, an *Inverse Discrete Fourier Transformation* is applied, following which, a spectral smoothing is applied again. The resultant coefficients are converted to cepstral variables. The Bark scale frequency is calculated using the following equation

$$bark(f) = \frac{26.81f}{1960 + f} - 0.53 \tag{3.21}$$

where, $bark(f)$ is the frequency (bark) and f is the frequency (Hz).

Comparing to the LPC, the identification achieved by perceptual linear prediction is more accurate. This is because, perceptual linear prediction is an improvement over the initial linear prediction coefficients, since it is successful in suppressing the information that is speaker dependent. Along with that, perceptual linear prediction is resistant to variations in the channel and noise, and has more advanced speaker independent recognition performance. Perceptual linear prediction is able to precisely reconstruct the auto-regressive noise components. Any Perceptual linear prediction based front-end is able to detect minute changes in the formant frequency.

3.2 Convolutional Neural Network

Inspired by the way the neurons in the human brain process information, Artificial Neural Networks (ANN) work collectively to learn from the input provided, in order to output an optimised result. Artificial Neural Networks consist of a large number of interconnected processing centers or nodes, called neurons, which are connected via the tails of the neurons. The input is loaded to the first layer, known as the input layer. The input is generally in the form of a multidimensional vector. Learning is done mainly in the hidden layers. Learning is the process of making decisions based on output from the previous layers, and deciding whether a change derogates or improves the output. A simplified model showing the structure of an Artificial Neural Network is shown in Figure 3.2.

Similar to Artificial Neural Networks, *Convolutional Neural Networks (CNN)* are comprised of many such individual computing nodes or neurons that can optimise themselves by learning repeatedly. Also similar to artificial neural networks, every neuron of a Convolutional Neural Network can improve and optimise itself through the learning process. The whole network, beginning from the multidimensional vectors as input till the final results of the class probabilities, expresses one weight or one perceptive score function.

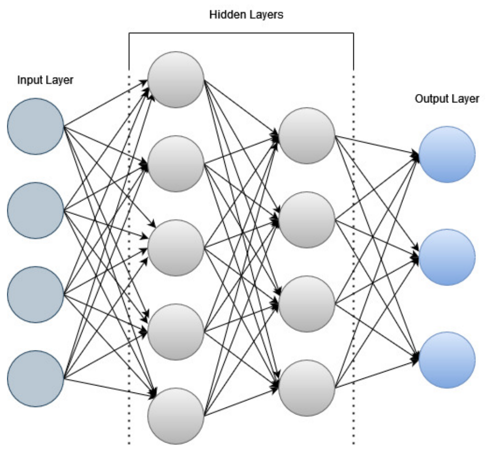

Figure 3.2: Simplified Structure of an Artificial Neural Network

The final layer has the loss functions that are associated with the classes, and prior to the final layer, it has fully-connected layers, just like artificial neural networks. The architecture of the Convolutional Neural Network can be set up in the way that best suits the particular type of data to be dealt with. However, contrary to artificial neural networks, in Convolutional Neural Network , the layers of neurons are organised into three dimensions - height, width, and depth. The depth refers to the activation volume, a third dimension. In convolutional neural networks, the neurons within any particular layer connects just with a minuscule region of the layer before that.

Convolutional Neural Network is one of the many types of deep learning methods that are most commonly used to analyse visual imagery or audio files, since, they are good at identifying characteristics within audio, speech, or image data. Convolutional Neural Networks consist of three types of layers – *Convolutional layer*, *Pooling layer*, and *Fully-Connected layer*. The convolutional layer is the initial layer present in a convolutional neural network. A convolutional layer can be followed by additional convolutional layers, or can be followed by pooling layers, or a combination of both. Finally, the last layer is the fully connected layer. The CNN increases in its complexity with each layer. Each layer provides the CNN the capability of recognising greater parts of the image. Initial layers focus on simple features. As the image or audio data progresses through the various layers of the convolutional neural network, the CNN starts to recognize larger, and more complex elements or shapes of the object until it can finally identify the required object. A simplified structure of a Convolutional Neural Network can be seen in Figure 3.3.

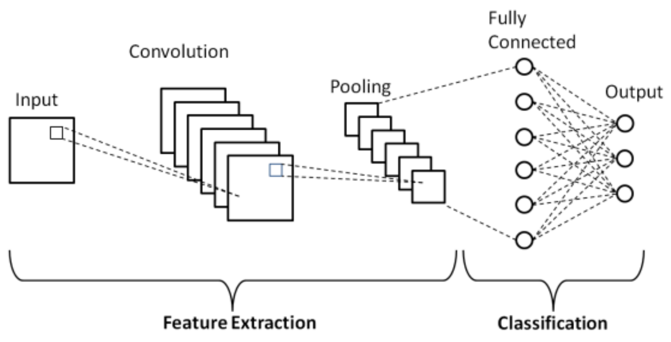

Figure 3.3: Simplified Structure of a Convolutional Neural Network (Phung and Rhee 2018)

3.2.1 Convolutional Layers

Each unit of a convolutional layer has the following parts - the input data, one filter, and one feature map. Each filter has a small height and width, and extends down the depth of the input volume. In case of an image, the input consists of three dimensions, represented by the RGB values in the colour image. The set of filters can learn correlations in an input. They are responsible for the task of convolution, which involves identifying the features in the image, and check for the presence of certain features by gliding across the image. Each filter stands for a feature. They are a matrix of parameters, two-dimensional, and also set the proportions of the receptive field. On passing a portion of the image through the filter, the filter computes a dot product between the feature detector parameters and the input, and inputs that into a vector. Following this, the filter glides to the next set of pixels. These steps continue until the filter glides through the whole input, resulting in the output, which is also known as an activation map. It is also known as a convolved feature, or, an activation map.

The filter weights do not change as the filter glides across the image. Even though a few parameters, like the weight values, are modified during model training by the process of back-propagation and gradient descent, there are some values that require to be fixed prior to model training. They are:

1. The number of pixels in the input matrix over which the kernel crosses. Higher number leads to a lower result. This is also called distance or stride.
2. The number of filters. Each separate feature detector results in one separate activation map, adding to the depth.
3. Padding is used when filters are not able to fit the input. This results in an equally sized, or a bigger output.

When the convolution is done, a Rectified Linear Unit Transformation is applied on the activation map. This introduces some non-linearity to the model. A model can have more than one convolutional layer. And finally, this layer transforms the image into numbers. This helps the rest of the neural network to extract required patterns (Education 2020).

3.2.2 Pooling Layer

Each normalization layer is followed by a pooling layer. Pooling layers are also called as downsampling. They reduce the dimensionality of the input. In other words, they decrease the number of parameters in the input. Just like the convolutional layer, this layer glides a filter across the whole inputs. However, this filter is not associated with any weights, but, an aggregation function is applied to the receptive field. This populates the output array. Pooling is mainly of two categories

1. **Max Pooling:** The filter selects the pixel with the highest value while moving across the input, to send to the output. This is mostly used.
2. **Average Pooling:** The filter calculates the average value of the receptive field as it glides across the input.

The pooling layer is generally applied independently to each depth slice of the input volume after a convolution layer, since it reduces the spatial size of the input.

3.2.3 Fully Connected Layers

After the convolutional layers have been set, a flattening layer is applied. A flattening layer in a convolutional neural network is used to convert all the resultant 2-Dimensional arrays from the pooled feature maps of the previous layer into a single, long, continuous linear vector. The resultant flattened matrix is then fed as input to the next fully connected layer for further processing. The fully connected layer in a convolutional neural network serves the purpose of detecting certain features in the audio. More specifically, each neuron in the fully connected layer corresponds to one particular feature that might be present in the audio. The neuron passes on a certain value to the next layer. This value represents the probability that the feature is present in our audio sample.

The end of the fully connected layer coincides with the end of the convolutional neural network. In other words, the artificial neural network at the end of a convolutional neural network predicts what is present in the image that the convolutional neural network tries to recognize. The first two fully-connected layers are each followed by a dropout layer. Dropout layers are important in the training of convolutional neural networks, since they prevent the model to become overfitted on the training data. Without the presence of dropout layers, the learning would be influenced by the first batch of training samples to a disproportionately large extent. This would, in turn, prevent the learning of features that appear only in the later batches or samples.

The final fully-connected layer marks the end of the model definition. This layer has units equal to the number of required output classes, and has the *Softmax* function as the activation function. *Softmax* is a function that converts an array of numbers into an array of probabilities, which are proportional to the relative scale of each value in the array. In other words, each value in the output of the *Softmax* function is interpreted as the probability of membership for each class. For example, for one audio sample belonging to one of the three output classes – *positive*, *negative*, and *neutral* - the output of this layer can be in this form [0.090030 0.665240 0.244728], indicating that the audio sample has a higher probability (66.42%) of being negative.

3.3 Long Short-Term Memory

For the next classifier, we used a Long Short-Term Memory network. Long Short-Term Memory (LSTM) networks are a variation of recurrent neural networks, which are a type of artificial neural network that are used to model time-dependent and sequential data problems. A recurrent neural network is commonly used for speech recognition, image captioning, language translation, and natural language processing. These neural networks are implemented in to popular voice activated digital assistants, such as *Siri*, voice search, and *Google Translate*. Like *Feed-Forward* and CNN, recurrent neural networks learn from the training data. These networks have a *memory*, which helps to take in information from previous outputs to utilise it with the current input to predict the current output. While traditional deep neural networks consider the assumption that inputs are independent of the outputs, in recurrent neural networks, the outputs depend on the prior elements within the sequence. While future outputs would also assist in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these outputs in their predictions.

As an example to help us in the explanation of recurrent neural networks, an idiom can be considered, such as "feeling under the weather", which is an expression that denotes that someone is not feeling well. Now, in order for this idiom to make the sense that we intend it to do, it needs to be expressed in that particular order only. Hence, the position of each word in the idiom needs to be accounted for by the recurrent neural networks. Recurrent neural networks use this information to predict the next word in the sequence of words.

In the Figure 3.4, the *Rolled RNN* visual of the recurrent neural network represents the whole neural network, or rather, the entire predicted phrase, like “feeling under the weather”. The *Unrolled RNN* image represents the individual layers, or time steps, of the neural network. Here, each layer of the *unrolled RNN* maps to a single word in the idiom, such as “weather”. In the third time-step, the previous inputs, such as “feeling”, and “under”, would be represented as hidden states to predict the next output in the sequence, “the”.

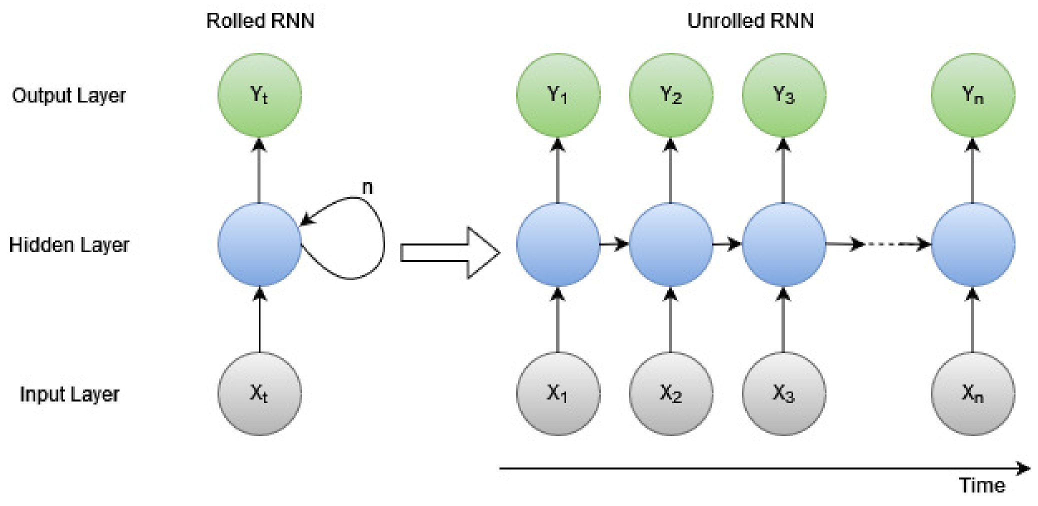

Figure 3.4: Simplified Structure of a Rolled RNN and Unrolled RNN.

Unfortunately, in practice, the range of contextual information that standard recurrent neural networks can access is somewhat limited. The problem is that the influence of a given input on the hidden layer, and therefore, on the network output, either blows up, or decays exponentially, as it cycles around the network’s recurrent connections. This problem is also known as the vanishing gradient problem. Long Short-Term Memory networks are recurrent neural network architectures specifically designed to address this vanishing gradient problem. Long Short-Term Memory (LSTM) networks are able to learn order dependence in problems related to sequence prediction. This behavior is needed in complex problem domains, such as speech recognition, machine translation, and so on. As shown in Figure 3.5, an LSTM network has two main types of layers - the LSTM layer and the Fully-Connected layer.

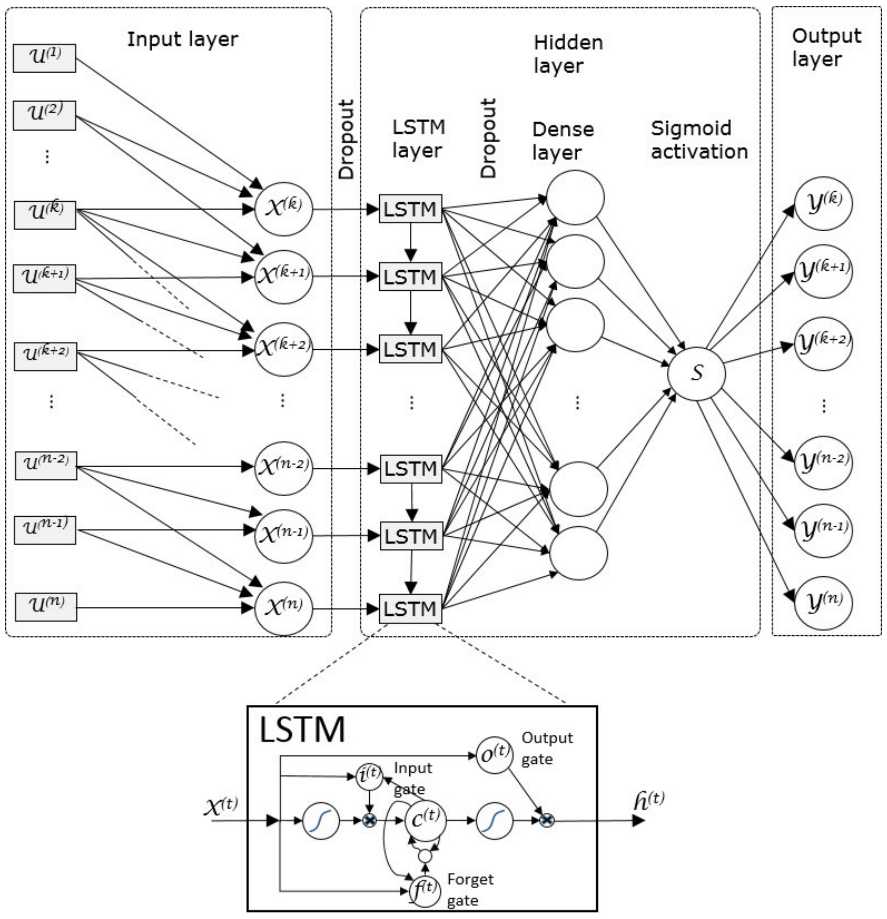

Figure 3.5: RNN with LSTM Deep Learning Architecture (Loukas et al. 2018)

3.3.1 LSTM Layer

An LSTM model has a cell state and three gates. These gates provide the model the power to selectively learn, unlearn, or retain information from each of the units. The cell state in Long Short-Term Memory networks helps the information flow through the units without being altered. It does so by allowing only a few linear interactions. Every single unit has an input, an output, and a forget gate. These gates can add or remove the information to and from the cell state. The forget gate decides which information from the previous cell state should be forgotten. It does so by utilising the *sigmoid* function. The input gate controls the information flow to the current cell state. It uses a point-wise multiplication operation of *sigmoid* and *tanh*, sequentially. In the end, the output gate decides which information is to be passed on to the following hidden state. Hence, in a step-by-step format, the following processing is done in an LSTM unit.

1. **New Inputs:** The inputs at the current step, which combine with the hidden state before passing on to the other gates.
2. **Hidden State:** The state of the cell from the previous iteration, which with the new inputs prior to being passed on to the three gates.
3. **Input Gate:** Identifies the significant elements that should be added to the current cell state. The information present in the output of this gate that is considered important by the input gate is only added to the cell state.
4. **Forget Gate:** Identifies what part of the information needs to be forgotten and not added to the next cell state. It can decide which values from the current cell state are to be discarded, which values are to be remembered, and which values are to be partially remembered by the next cell state.
5. **Updating Cell State:** After the results of the Forget gate have been multiplied by the previous cell state, new information is added from the multiplication of the input gate and the cell state candidate and the latest cell state is obtained.
6. **Updating Hidden State:** Lastly, the hidden state is updated by passing the latest cell state through a tanh activation function before multiplying it with the results of the output gate to get the updated hidden state.

At the end of all of this, the hidden state and the cell states return to the recurring unit. The process is repeated until the end of the input sequence is reached.

3.3.2 Fully Connected Layers

The LSTM layer is followed by the *Fully-Connected layers*. The *Dense* layers correspond to the fully connected layers in our model. As stated previously, the units in a dense layer correspond to the number of neurons in that particular layer. The last and the final layer of the fully-connected layer is also the final layer of the LSTM model. It is also a *Dense* layer, having number of neurons equal to the number of output classes, and has an *activation function*, which decides the types of output.

3.4 Transformer Architecture

3.4.1 Encoder-Decoder Architecture

To deal with Natural Language Processing problems whose input and output are both variable-length sequences, *encoder-decoder* architectures were designed. They have two major components – an *encoder* and a *decoder.*

- **Encoders** can accept input sequences of variable lengths and transform them into a fixed-shaped state.
- **Decoders** map the encoded state having a fixed shape to a sequence of varying length.

The encoder-decoder architecture handles varying length input-output sequences. This makes them suitable for sequence transduction problems, such as, speech recognition.

3.4.2 Attention Mechanism

The concept of attention in artificial neural networks is the same as that of the general concept of attention paid by the brain. It mimics the technique of attention paid by the brain, in which, the brain only focuses on certain words or intonations in an utterance, and ignores the rest, which the brain deems unimportant. Attention-like mechanisms started to show up in the 1990s from problems that deal with data sequences that vary with time. Currently, they have become more and more popular in a broader area of sequencing problems in Natural Language Processing. They can better deal with the "vanishing gradient problem", where the contribution from the previous steps becomes not that important for the plain recurrent neural network unit. They are also effective in dealing with the "bottleneck problem", where, the current state of a long sequence cannot encode all the information present in the previous states.

Attention mechanisms are a type of neural architecture which helps a model to dynamically highlight relevant features from the input sequence. The attention mechanism can be applied directly to the raw input input, or to an higher level representation. They are becoming increasingly popular, and are used in various kinds of neural architectures. In the simplest architecture, an attention unit has 3 fully-connected or dense layers that require training. These layers – called *Query*, *Key*, and *Value.* The Query and Key combine in the encoder side to have the desired effect on each target output. Attention can be of different kinds based on how many inputs are considered, or how it is calculated. Based on how many units of the input is considered, there are two kinds of attention.

1. Global Attention - focuses on each and every word in the input sequence for every target word in the output.
2. Local Attention - focuses on only a small subset of words from the input sequences per target word in the output.

Based on how the attention is calculated, there are again two kinds of attention:

1. Hard Attention - is when attention scores are used instead of weighted average of all the hidden states to select one hidden state of the encoder.
2. Soft Attention - is when the context vectors are calculated as a weighted sum of the encoder hidden states.

Generalised Attention

Generalised attention can be calculated by taking a weighted sum of the values, the weights being a factor of the query and the keys. The query provides the attention mechanism with the information on which values to attend or give importance to. The expression is as follows (Lihala 2019).

$$A(q, K, V) = \sum_i \frac{exp(e_{qk_i})}{\sum_j exp(e_{qk_j})} \tag{3.22}$$

Self-Attention

In self-attention mechanism, every hidden state gives attention to hidden states of the previous iteration of the same recurrent neural network.

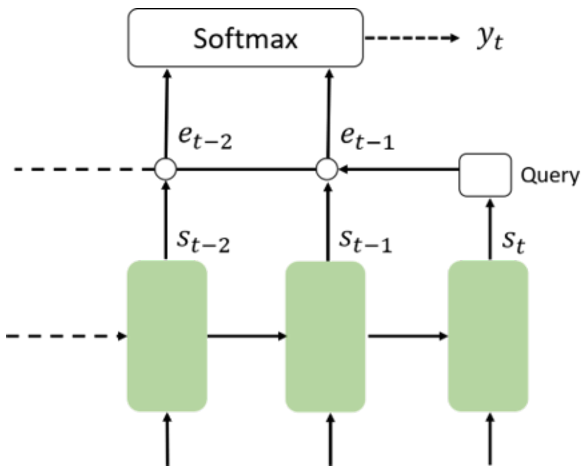

Figure 3.6: Self-Attention Mechanism (Lihala 2019)

In Figure 3.6, s_t is the query, and the hidden states of the decoder, s_0 to s_{t-1}, are a combination of both keys and values of previous states.

3.4.3 Transformer Model

The main idea of building a transformer architecture is its ability to attend to different areas of the input sequence to find out another representation of that sequence. This is known as *self-attention*. It has self-attention layers stacked one after the other, and processes variable-length input sequences using these self-attention layers, instead of employing recurrent or convolutional neural networks.

Positional Encoding

Attention layers consider their input as a set of non-sequential vectors. Hence, to feed to the model information about the relative position of the tokens in the sequence, since, the model does not have any convolutional or recurrent neural networks. The formula for calculating positional encoding (TensorFlow n.d.) is as follows.

$$PE_{pos,2i} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.23}$$

$$PE_{pos,2i+1} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.24}$$

Multi-Head Attention

In the attention model, instead of having one attention head, the queries, keys and values are split into multiple heads. This is done to permit the model to pay attention to information contained in the input from various representation sub-spaces at different positions. Splitting of heads reduces the dimensionality of the resulting heads, and hence, the cost of computing the attention remains the same as that of a single headed attention, with all the dimensions preserved.

Multi-head attention consists of four parts:

- **Linear Layers:** Every single block in multi-head attention gets three inputs - query (Q), key (K), and value (V). They are passed through a linear or a dense layer first. The output of this layer is rearranged to the dimension *(batch, num_heads, ...)*, and then, it is passed through the attention function.
- **Scaled Dot-Product Attention:** This function is applied on the inputs, in one function call. In case if efficient performance is required, a broadcast is used.

- **Final Linear Layer:** The output of the scaled dot-product attention layer from all the heads is concatenated, and passed through the last and final dense layer.

Scaled Dot-Product Attention

This layer also takes in three inputs from the multi-head layer – the query Q, the keys K, and the values V. Then the weights of the values are scaled by a factor of square root of the depth. The calculation uses the following equation.

$$Weight(Q, K) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{3.25}$$

The attention is then calculated by finding the dot-product of the scaled weights and the value vector (TensorFlow n.d.)

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right).V \tag{3.26}$$

Since the normalization is done for the keys along the dimension, these values of the attention actually decides the importance to be given to the keys for every query.

3.5 Evaluation Metrics

3.5.1 Confusion Matrix

After the model is trained, we create the confusion matrix to know how the model has performed in categorising hate speech from non-hate. Confusion matrix is a measurement of performance for machine learning models. It is a table with four different combinations of actual and predicted output values. Confusion matrices are extremely useful for calculating other important performance measurement parameters, namely, *Recall*, *Precision*, *F1-Score*, etc.

The four cells of the confusion matrix holds four values – the number of predictions that are *True Positives (TP)*, or which were correctly classified as belonging to a particular class of outputs, the number of predictions that are *False Positives (FP)*, or which were incorrectly classified as belonging to a particular class but actually didn't belong to that class, the number of predictions that are *True Negatives (TN)*, or which were correctly classified as not belonging to a particular class and did not belong to the same, and finally, the number of predictions which are *False Negatives (FN)*, or the ones that were incorrectly categorised as Negatives, but were actually Positives.

3.5.2 Recall

From the confusion matrix, using the values of TP and FN, we can calculate the Recall of the model by using the following formula.

$$Recall = \frac{TP}{TP + FN} \tag{3.27}$$

This value gives us the idea that from all the inputs that belonged to a particular class, how many inputs were accurately classified as belonging to that class. The Recall value has to be high for a well-designed model.

3.5.3 Precision

The Precision value gives us the estimate that from all the inputs that were classified to be belonging to particular classes, how many actually belonged to those classes. This is calculated from using the values of TP and FP by using the following formula.

$$Precision = \frac{TP}{TP + FP} \tag{3.28}$$

This value also needs to be high for a model.

3.5.4 F1-Score

The F1-Score helps us compare the performance of two or more models. Comparing two models using only the Precision and Recall creates difficulties. It is not easy to compare two models having low recall and high precision, or vice-versa. This is where F1-Score comes in. It uses both the Recall value and the Precision value for calculation, and then finds the Harmonic Mean of the two to calculate the F-Score. This measures both Recall and Precision at the same time. The equation for finding the F1-Score shows this.

$$F1 - Score = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{3.29}$$

Chapter 4

System Description

In this work, we perform the spoken language processing task of hate speech identification in spoken language. We aim to study different ways of detection of toxic utterances in speech using three types of classification models:

1. Convolutional Neural Networks
2. Long Short-Term Memory
3. Attention-based Convolutional Neural Network

One of the main difficulties that are faced in the detection of hate speech in spoken language is the learning of representations that precisely capture signals of hateful content from speech. These signals are inherent in linguistic content, such as, semantic meaning and emotion, together with acoustic contents, such as, tone, sentiments, as well as speaker characteristics, in different speech variations.

Our approach involves the use of filter banks for the extraction of speech features from the raw waveform of spoken language. We have used parts of a dataset that has been curated carefully for the purpose of the spoken language processing task of identification of toxicity in speech. The dataset, called DeToxy (Ghosh et al. 2021), is the first publicly available dataset for toxic speech classification tasks. Then, we used the filter bank features extracted from this dataset to train the three classifier models. These types of models have long been widely used in the areas of classification of text, classifying emotions, sentiment analysis, and so on.

4.1 Dataset Description

For this experiment, we have used parts of the DeToxy (Ghosh et al. 2021) dataset, the first of its kind annotated dataset for the detection of toxic elements in speech in the English Language. This dataset is a subset of various open-source datasets, details of which are in Table 4.1. These datasets were combined and used as a single dataset for our experiments.

Table 4.1: Details of the DeToxy dataset (Ghosh et al. 2021)

Dataset	**No. of Samples**
CMU-MOSEI	640
CMU-MOSI	200
Common Voice	8595
LJ Speech	111
MELD	427
Social-IQ	485
VCTK	145

4.1.1 Carnegie Mellon University - Multimodal Opinion Sentiment and Emotion Intensity (CMU-MOSEI)

This dataset (Zadeh, Liang, et al. 2018) is the considered to be the largest and most extensive dataset for emotion recognition tasks and multimodal sentiment analysis at the time this experiment was done. Consisting of greater than 23,500 videos from 1000+ speakers on YouTube. Gender balanced, and having a variety of topics of video content, transcription of the videos is done and they are properly punctuated.

4.1.2 Carnegie Mellon University - Multimodal Corpus of Sentiment Intensity (CMU-MOSI)

This dataset (Zadeh, Liang, et al. 2018) is another dataset by Carnegie Mellon University, which consists of 2199 video clips of different opinions, annotated with sentiment. It is annotated in the range [-3,3], using various parameters for sentiment intensity, subjectivity, per-milliseconds annotations of audio features. It contains 97% non-toxic and nearly 3% toxic utterances.

4.1.3 Common Voice

This dataset (Ardila et al. 2019) by Mozilla Developer Network is an open-source, dataset of voices of multiple languages for the use of training speech enabled systems, with 20,217 hours of recorder audio and 14,973 hours of validated speech audios.

4.1.4 LJ Speech

This is another open-source dataset (Ito and Johnson 2017) which has 13,100 clips of short audio segments, with one speaker reading texts from a collection of seven books of non-fiction. Every clip is transcribed, and have a varying length of 1 to 10 seconds.

4.1.5 Multimodal Emotion Lines Dataset (MELD)

MELD (Poria et al. 2019) has over 1400 dialogues and 13000 dialogues from the television show "Friends". Each utterance in a dialogue has been labeled by one of the emotions – Anger, Disgust, Sadness, Joy, Neutral, Surprise and Fear. MELD also has annotations for sentiments – positive, negative and neutral.

4.1.6 Social-IQ

Another dataset (Zadeh, Chan, et al. 2019) by the Carnegie Mellon University, this dataset has videos that are thoroughly validated and annotated, along with questions, answers, and annotations for the level of complexity of the said questions and answers.

4.1.7 VCTK

The VCTK corpus (Yamagishi, Veaux, and MacDonald 2019) contains 110 speakers' speech data spoken in English, having various accents. Every single speaker reads a passage, selected from newspapers, archives, and so on.

4.2 Feature Extraction

4.2.1 Filter Banks

Filter Banks can provide a better resolution at low frequencies, and lower resolution at higher frequencies. It takes input the energy at each critical frequency band, and results in an estimation of the spectrum shape. Filter banks have various advantages, including:

- High resolution with low spectral leakage

- More stable peaks
- The more accurate and consistent noise floor
- By using filter banks, we can set channelizers to get different resolutions at different frequency bands. This is used in audio spectral analysis.

Filter banks are used in applications where signal compression is required to emphasize particular frequencies more than other frequencies. This includes speech processing, image compression, communications systems, antenna systems, analog voice privacy systems, and in the digital audio industry.

Moreover, filter banks are also used as graphic equalizers, which can attenuate the components of the signal differently and reconstructs them into an improved version of the original signal. Filter banks using Discrete Cosine Transformation are used in various places, such as, speech processing, audio signal processing, and so on.

For feature extraction of spoken utterances, we used Filter Banks. For calculating Filter Banks, we had *pre_emphasis*, frame size represented as f_size, frame stride represented as f_stride, N-point Fast Fourier transform represented as $NFFT$, low frequency mel represented as lf, number of filter represented as $nfilt$, number of cepstral coefficients represented as $ncoef$, and cepstral lifter represented $lifter$ of values 0.97, 0.025 (25ms), 0.015 (15ms overlapping), 512, 0, 40, 13, and 22, respectively. We used frame size of 25 ms, as, typically, frame sizes in speech processing domain uses 20ms to 40ms with 50% (in our case 15ms) overlapping between consecutive frames.

$$hf = 2595 \times \log_{10}(1 + \frac{0.5 \times sr}{700}) \tag{4.1}$$

We used low frequency mel (*lf*) as 0, and calculated the high frequency mel (*hf*) using the equation 4.1. *lf* and *hf* were used to generate the non-linear human ear perception of sound, by more discriminative of lower frequencies and less discriminative at higher frequencies.

$$emphasized\_signal = [sig[0], sig[1:] - pre\_emphasis * sig[:-1]] \tag{4.2}$$

As shown in equation 4.2, emphasized signal is calculated by using pre-emphasis filter applied on signal using first order filter. Number of frames is calculated by taking the ceiling value of the dividend of absolute value of difference between signal length (*sig_len*), and the product of filter size (f_size) and sample rate (*sr*) with the product of frame stride (f_stride) and sample rate (*sr*) as shown in equation 4.3. Signal length is the length of *emphasized_signal* calculated in equation 4.2.

$$n\_frames = \lceil \frac{|sig\_len - (f\_size \times sr)|}{(f\_stride \times sr)} \rceil \tag{4.3}$$

Using equation 4.4, we generated pad_signal from concatenation of $emphasized\_signal$ and zero value array of dimension $(pad\_signal\_length - signal\_length) \times 1$, where, pad_signal_length was calculated by $n\_frames \times (f\_stride \times sr) + (f\_size \times sr)$.

$$pad\_signal = [emphasized\_signal, [0]_{((n\_frames \times (f\_stride \times sr) + (f\_size \times sr)) - sig\_len) \times 1}] \tag{4.4}$$

Frames are calculated as shown in equation 4.5, from the pad_signal elements, where, elements are the addition of array of positive natural number from 0 to $f\_size \times sr$, repeated n_frames, and transpose of array of size of num_frames, where, each element is the difference of $(f\_stride \times sr)$.

$$\begin{gathered} frames = pad\_signal[(\{x \in Z^+ : 0 < x < (f\_size \times sr)\}_n)_{n=0}^{(n\_frames,1)} + \\ ((\{r : r = (f\_stride \times sr) \times (i-1), \\ i \in \{0, \ldots, n\_frames \times (f\_stride \times sr)\}\}_n)_{n=0}^{((f\_size \times sr),1)})^{\intercal}] \end{gathered} \tag{4.5}$$

Power frames, shown in equation 4.6, are calculated as the square of the absolute value of Discrete Fourier Transform (DFT) of the product of hamming window, and the frames of each element with NFFT.

$$pf = \frac{|DFT((frames \times (0.54 - (\sum_{N=0}^{(f\_size \times sr)-1} 0.46 \times \cos \frac{2\pi N}{(f\_size \times sr)-1}))), NFFT)|^2}{NFFT} \tag{4.6}$$

$$mel\_points = \{r : r = lf + \frac{hf - lf}{(nfilt + 2) - 1} \times i, i \in \{lf, \ldots, hf\}\} \tag{4.7}$$

Mel_points is the array where elements are calculated as shown in the equation 4.7, where, i are the values belonging from lf to hf.

$$bins = \lfloor \frac{(NFFT + 1) \times (700 \times (10^{\frac{mel\_points}{2595}} - 1))}{sample\_rate} \rfloor \tag{4.8}$$

From equation 4.8, bins are calculated, where the floor value of the elements are taken, which is the product of hertz points and $NFFT + 1$, divided by sample rate. Hertz points is calculated by multiplying 700 to the subtraction of 1 from 10 raised to the power of $\frac{mel\_points}{2595}$.

$$fbank_m(k) = \begin{cases} 0 & k < bins(m-1) \\ \frac{k-bins(m-1)}{bins(m)-bins(m-1)} & bins(m-1) \leq k \leq bins(m) \\ \frac{bins(m+1)-k}{bins(m+1)-bins(m)} & bins(m) \leq k \leq bins(m+1) \\ 0 & k > bins(m+1) \end{cases} \tag{4.9}$$

Bins, calculated from equation 4.8, are used to calculate filter banks, as shown in equation 4.9. Each filter in the filter bank is triangular, with a response of 1 at the central frequency, and a linear drop to 0 till it meets the central frequencies of the two adjacent filters, where the response is 0.

Filter bank features, generated from equation 4.9, is provided as input to the neural network, which expects the same dimension followed by convolution layers. Raw speech signal cannot be provided as input to the architecture, as it contains lots of noise data. Therefore, extracting features from the speech signal and using it as input to the model will result in better performance than directly considering raw speech signal as input. Our motivation to use filter banks features as the feature count is small enough to force us to learn the information of the sample. Parameters are related to the amplitude of frequencies, and provide us with frequency channels to analyze the speech specimen.

4.3 Model Architecture

4.3.1 The CNN Classifier

For our Convolutional Neural Network classifier, shown in Figure 4.1, a Sequential model was considered with four Convolutional layers. The layers had increasing number for filters, with the first layer having 32 filters, second layer having 64, third one having 128, and the last one having 512 filters. All the layers had a kernel size of 3 by 3, and utilised the Rectified Linear Unit as its activation function. The rectified linear activation function (ReLU) is a piece-wise linear function that returns the input unchanged if the value of the input is positive, or else, returns zero as the output (Agarap 2018). A model that uses it is easier to train, and results in an overall higher performance, hence, it has become the default activation function for various types of neural networks.

Each convolutional layer was followed by a normalization layer. The normalization layer normalizes each feature so that the contribution of every feature in the model is maintained, as some feature has higher weightage than others (Ba, Kiros, and Hinton 2016).

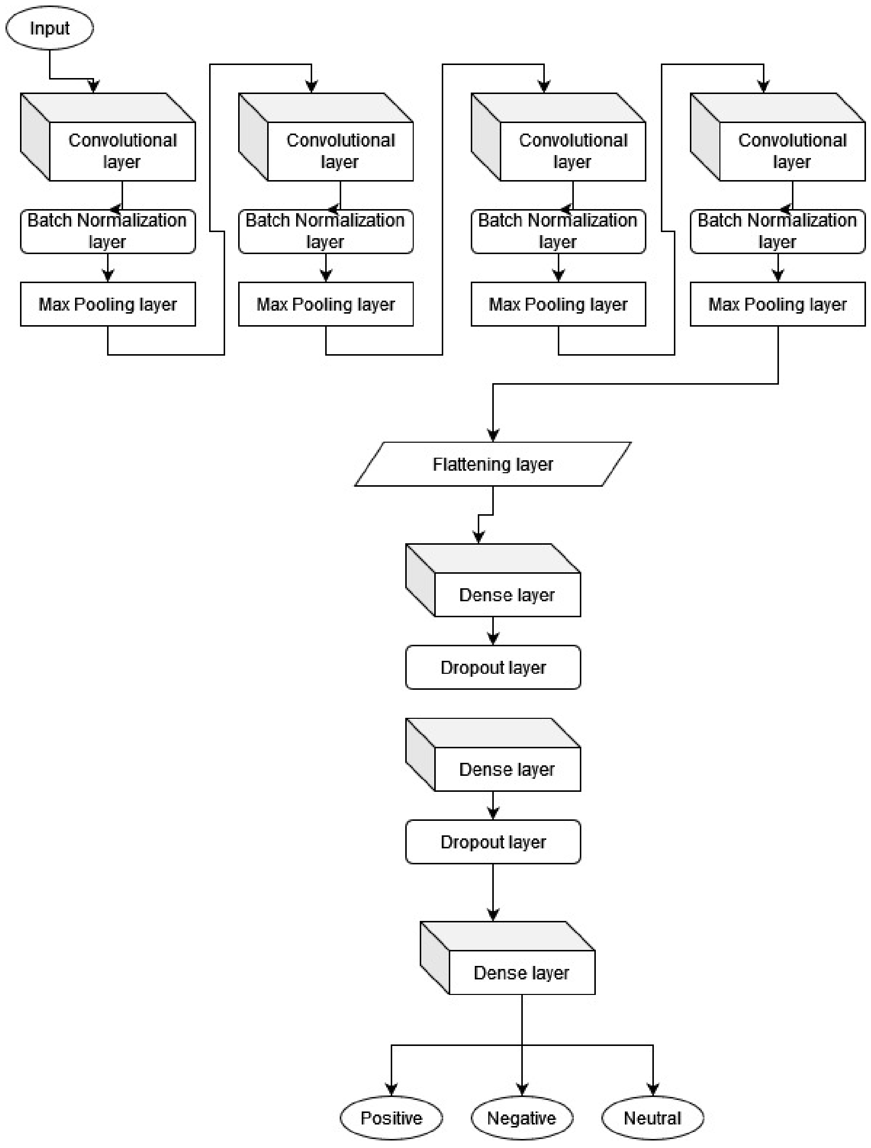

Figure 4.1: Simplified Structure of the CNN Classifier Model Used

This ensures that our model is unbiased. There are different types of normalization. Here, we have used batch normalization in our model (Thakkar, Tewary, and Chakraborty 2018). Batch normalization is a normalization technique that standardizes the inputs to a neural network, applied to either the activations of a prior layer, or, to the inputs directly. Batch normalization speeds up the time taken for training (Rajagopal 2018). It does so by halving the epochs or better in some cases. Other than that, it also provides some regularization, reducing the error caused due to generalization. And lastly, we used MaxPooling1D as the pooling layer, with a pool size of 2.

The three Dense layers correspond to the fully connected layers in our model. The first dense layer has 550 units, and the second dense layer has 400 units. These units in a dense layer correspond to the number of neurons in that particular layer. Hence, the first dense layer has 550 neurons and the next dense layer has 400 neurons. Both of them have rectified linear unit has their activation functions.

Finally, the third and final dense layer completes our model. It has three neurons. The three neurons correspond to the three classes of our expected output - *positive*, *negative*, and *neutral*. This layer has a *Softmax* function as its activation function.

4.3.2 The LSTM Classifier

In the LSTM layer, we kept the output dimensions as 128, represented by the number of units. The activation function used was the *tanh* function, and the recurrent activation function used was the *sigmoid* function, respectively. These are set by default by the LSTM constructor, so no changes were made. This LSTM layer was followed by a *Dropout layer*. The function of the *Dropout layer* is to randomly set input units to 0 with a frequency of *rate* at each step during training time. This helps in the prevention of over-fitting of the training data.

Our LSTM classifier model, shown in Figure 4.2, consisted of an LSTM layer, three Dense layers each followed by one dropout layer, and a final Dense layer for the output with a *Softmax* activation function. The first dense layer had 128 units, the second dense layer had 64 units, and the third one had 48 units. So, the first dense layer had 128 neurons, the next dense layer had 64 neurons, and the third layer had 48 neurons. The first three *Dense layers* had rectified linear unit as their activation functions. The first three dense layers were each followed by a dropout layer, which again prevents over-fitting of the data, with varying dropout rates of 0.1 and 0.3. The final *Dense* layer had 3 units, which corresponded to the three output classes that the audio samples were to be categorised to - *positive*, *negative*, and *neutral*. Similar to the previous convolutional neural network, the activation function used here was the *Softmax* function, which gives the probability of an input belonging to each of the output classes. Again, similar to the CNN classifier, the LSTM classifier model was trained on for 300 epochs with batch size equal to 64.

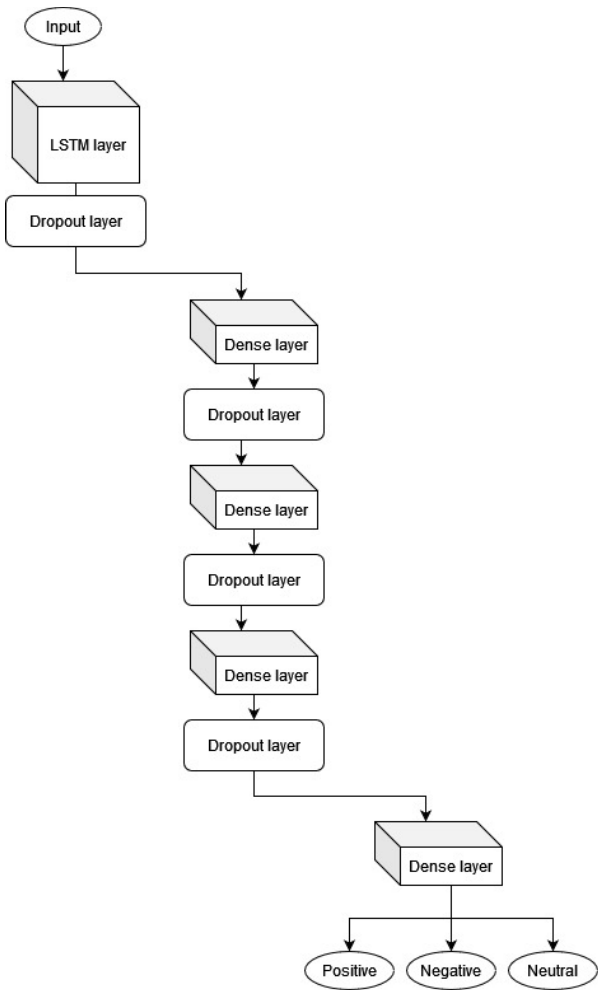

Figure 4.2: Simplified Structure of the LSTM Classifier Model Used

4.3.3 The Attention-Based CNN Classifier

For the Attention-based CNN model, a *MultiHeadAttention* class was defined with initialised properties like number of heads *num_heads*, and depth of the model *depth*. Three initial and one final *Dense* layers were also defined in the constructor. Next, a *split_heads* method was defined that was used to split the last dimension into *(num_heads, depth)*, and then, was used to transpose the result into the dimension *(batch_size, num_heads, seq_len, depth)*. The *call* method was used for calling the initial three dense layers in the beginning, followed by calling the scaled dot product attention method, concatenating the output of the scaled dot product attention method, and finally, calling the final dense layer. The scaled dot product attention method was used to calculate the attention weights of the individual heads. It does the matrix multiplications and scales the result by the factor of $\sqrt{d_k}$. It then applies the *Softmax* function to that, and then does a matrix multiplication of the result to the value vector, and returns the individual scaled attention weights and the attention values.

$$Attention(Q, K, V) = softmax_k \left(\frac{QK^\intercal}{\sqrt{d_k}} \right) V \tag{4.10}$$

Just like the previous two classifier models, the Attention model was initialised with two convolutional layers, each followed by a *BatchNormalization* layer. After this layer, the attention layer was added. This was the *multi-head attention layer*. Then, a flattening layer was added, with lastly, a Dense layer after it. Figure 4.3 shows a simplified diagram of the entire structure.

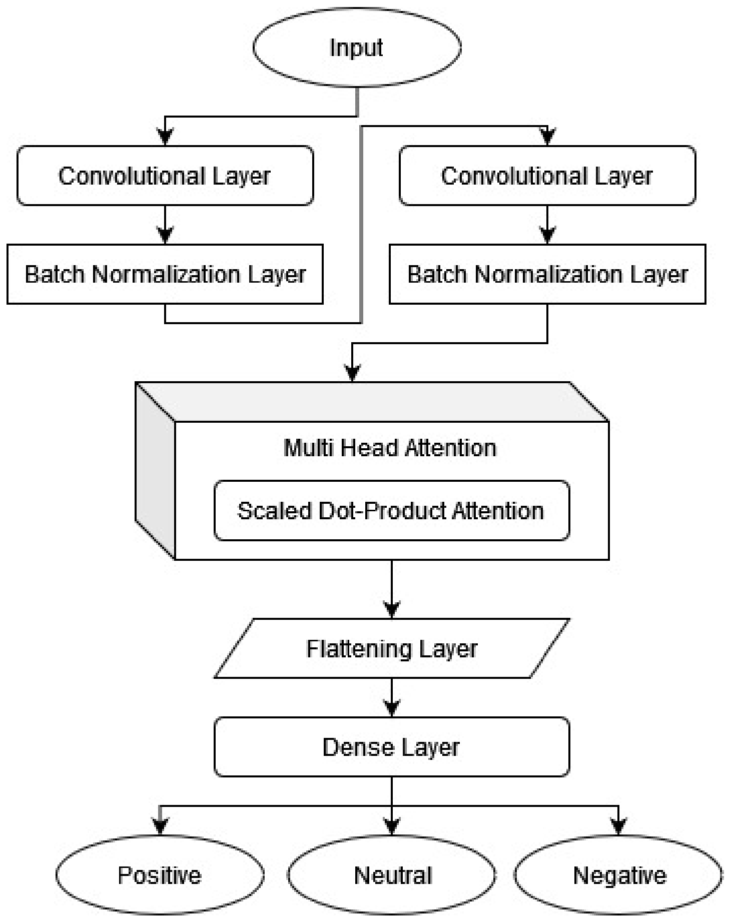

Figure 4.3: Simplified Structure of the Attention-Based CNN Classifier Model Used

Chapter 5

Experiments and Results

As mentioned in chapter 4, our experiments were conducted on a dataset consisting of all the seven datasets taken together. Entries in each dataset belonged to one of the three classes - *positive*, *negative*, and *neutral*. The sample size of the classes belonging to each dataset is given in Table 5.1, and the sample size for the three classes of the entire dataset is given in Table 5.2.

Table 5.1: Class Sample Sizes of Each of the Individual Datasets

Sl. No.	**Dataset**	**Positive**	**Negative**	**Neutral**
1	CMU-MOSEI	156	384	100
2	CMU-MOSI	57	109	34
3	Common Voice	2,360	4,746	1,488
4	LJ Speech	18	74	19
5	MELD	103	233	91
6	Social-IQ	149	242	87
7	VCTK	46	73	26
	Total	2,889	5,861	1,845

Table 5.2: Class Samples of the whole Dataset

Classes	**Sample size**
positive	2889
negative	5861
neutral	1845
	10595

Our datasets were not split into training, validation, and testing sets. Hence, our experiments were performed on 5-fold training sets, where samples were randomly distributed into 5 sets, and each set had nearly equal number of samples from each class, as shown in Table 5.3.

Table 5.3: 5-fold training data

	Set 1	**Set 2**	**Set 3**	**Set 4**	**Set 5**	
positive	578	578	577	578	578	2889
negative	1173	1172	1172	1172	1172	5861
neutral	374	367	367	368	369	1845

Five independent training were done for the three architectures, where the 1st training consisted of set 1, set 2, set 3, and set 4 as training data, and set 5 as test data. Similarly, for the 2nd training, 3rd training, 4th training, and 5th training, set 1, set 2, set 3, and set 5 were taken as training data, and set 4 as test data, set 1, set 2, set 4, and set 5 were taken as training data, and set 3 as test data, set 1, set 3, set 4, and set 5 were taken as training data and set 2 as test data, set 2, set 3, set 4, and set 5 were taken as training data, and set 1 as test data, respectively.

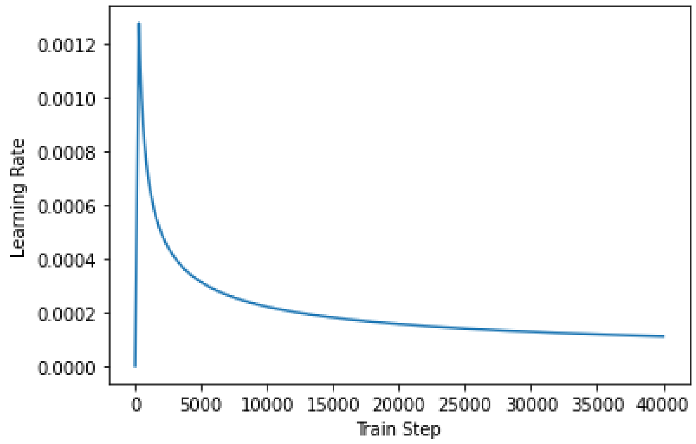

Figure 5.1: Learning rate according to global steps

For the CNN classifier, we trained for 300 epoch(s). Similarly, for the LSTM classifier and the Attention based CNN classifier, we trained for 300 and 1000 epoch(s), respectively. We used adaptive learning rate (Figure 5.1) for each classifier model, the formula for which is shown in equation 5.1.

$$lrate = \frac{(ilr \times hs^{-0.5}) \times \min(1.0, \frac{gs}{ws})}{\max(gs, ws)^{-0.5}} \tag{5.1}$$

where, *ilr* is initial learning rate,
hs is hidden size,
gs is global step, and
ws is warm-up step

For the CNN classifier, we achieved an accuracy of 55.9731 %. The confusion matrix and evaluation report are given in Table 5.4. As per the confusion matrix, we can deduce that due to imbalanced data, for each class, the model tends to predict the class in which larger values are present. Here, the negative class having more data compared to the neutral and the positive led to classifying maximum data of classes neutral and positive to the negative class.

Table 5.4: Confusion matrix for CNN Classifier

	negative	**neutral**	**positive**	**precision**	**recall**	**f1-score**	**accuracy**
negative	4687	536	701	0.656	0.7912	0.7173	
neutral	906	675	283	0.3454	0.3621	0.3536	**55.9731**
positive	1552	743	640	0.3941	0.2181	0.2808	

Table 5.5: Confusion matrix for LSTM Classifier

	negative	**neutral**	**positive**	**precision**	**recall**	**f1-score**	**accuracy**
negative	5921	0	0	0.5523	1	0.7116	
neutral	1864	0	0	0	0	0	**55.2332**
positive	2935	0	0	0	0	0	

Table 5.6: Confusion matrix for Attention-Based CNN Classifier

	negative	**neutral**	**positive**	**precision**	**recall**	**f1-score**	**accuracy**
negative	4056	717	1151	0.6939	0.6847	0.6893	
neutral	578	890	396	0.43	0.4775	0.4525	**57.8849**
positive	1211	463	1261	0.4491	0.4296	0.4391	

For the LSTM classifier, we obtained 55.2332 % but the classifier model was biased to a single class. The model classified data belonging to all the classes as negative class, since a large portion of the dataset belonged to negative class as compared to other classes. The confusion matrix and the evaluation results are shown in Table 5.5. Similarly, for Attention based CNN classifier, we achieved 57.8849% accuracy, but, in this case, the model was able to learn better with respect to the other two classifiers, as it was able to accurately predict more data with respect to other classifier architectures. The confusion matrix and evaluation matrix are shown in Table 5.6.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This work compared three popular design architectures of classification implemented on speech, with the aim of spotting spiteful utterances in speech. It was done using three different machine learning techniques – Convolutional Neural Network, Long Short-Term Memory, and an Attention-based CNN architecture.

The first chapter of this thesis presented an introduction to hate speech and hate speech detection approaches, the second one had many of the previous work related to this work, following which the system architecture and implementations have been discussed, and finally, we proceeded with our results.

Our experiments led us to the conclusion that the Long Short-Term Memory and Attention-based CNN architectures provided much better detection of toxic occurrences, compared to traditional CNN architectures. The Attention-based CNN performed marginally better than the two, with better accuracy values every time.

6.2 Future Works

This work has a lot of scope for development and diversification in the coming days. In the future, the work done here can be further expanded to cover larger and more well annotated datasets. Other various deep learning model designs can be applied to get better results, and to make hateful content detection in speech more robust. Moreover, this approach can be further localised to various geographies and communities by extending this to include more languages and their dialects.

These can be used to check the spread of hate crimes and violence against targeted groups via media, such as, audio posts, blogs, group calls, or, live or recorded meetings or such public gatherings. This can help prevent the disruption of harmony and ensure safety to some extent in the community.

Bibliography

Agarap, Abien Fred (2018). "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375. arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (2017). "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

Alim, Sabur Ajibola and Nahrul Khair Alang Rashid (2018). "Some Commonly Used Speech Feature Extraction Algorithms". In: *From Natural to Artificial Intelligence*. Ed. by Ricardo Lopez-Ruiz. Rijeka: IntechOpen. Chap. 1. DOI: 10.5772/intechopen.80419. URL: https://doi.org/10.5772/intechopen.80419.

Ardila, Rosana et al. (2019). *Common Voice: A Massively-Multilingual Speech Corpus*. DOI: 10.48550/ARXIV.1912.06670. URL: https://arxiv.org/abs/1912.06670.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. DOI: 10.48550/ARXIV.1607.06450. URL: https://arxiv.org/abs/1607.06450.

Baevski, Alexei et al. (2020). "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations". In: *CoRR* abs/2006.11477. arXiv: 2006.11477. URL: https://arxiv.org/abs/2006.11477.

Barakat, MS, CH Ritz, and David A Stirling (2012). "Detecting offensive user video blogs: An adaptive keyword spotting approach". In: *2012 International Conference on Audio, Language and Image Processing*. IEEE, pp. 419–425.

Benesty, Jacob, M Mohan Sondhi, and Yiteng Arden Huang (2008). "Introduction to speech processing". In: *Springer Handbook of Speech Processing*. Springer.

Biere, Shanita, Sandjai Bhulai, and Master Business Analytics (2018). "Hate speech detection using natural language processing techniques". In: *Master Business AnalyticsDepartment of Mathematics Faculty of Science*.

Deller, John R., John G. Proakis, and John H. Hansen (1993). *Discrete Time Processing of Speech Signals.* 1st. USA: Prentice Hall PTR. ISBN: 0023283017.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

Education, IBM Cloud (Oct. 2020). *Convolutional Neural Networks.* `https://www.ibm.com/cloud/learn/convolutional-neural-networks`. [Online; Accessed 19-June-2022].

Fayek, Haytham M. (2016). *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between.* URL: `https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html`.

Georgakopoulos, Spiros et al. (Jan. 2020). "Convolutional Neural Networks for Twitter Text Toxicity Analysis". In: pp. 370–379. ISBN: 978-3-030-16840-7. DOI: `10.1007/978-3-030-16841-4_38`.

Gholamalinezhad, Hossein and Hossein Khosravi (2020). "Pooling Methods in Deep Neural Networks, a Review". In: *CoRR* abs/2009.07485. arXiv: `2009.07485`. URL: `https://arxiv.org/abs/2009.07485`.

Ghosh, Sreyan et al. (Oct. 2021). "Speech Toxicity Analysis: A New Spoken Language Processing Task". In.

Hashida, Shuichi, Keiichi Tamura, and Tatsuhiro Sakai (2018). "Classifying Sightseeing Tweets Using Convolutional Neural Networks with Multichannel Distributed Representation". In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 178–183. DOI: `10.1109/SMC.2018.00041`.

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

Iqbal, Sania (Mar. 2018). "SPEECH RECOGNITION SYSTEMS -A REVIEW". In.

Ito, Keith and Linda Johnson (2017). *The LJ Speech Dataset.* `https://keithito.com/LJ-Speech-Dataset/`.

Iwana, Brian Kenji, Ryohei Kuroki, and Seiichi Uchida (2019). "Explaining Convolutional Neural Networks using Softmax Gradient Layer-wise Relevance Propagation". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 4176–4185. DOI: `10.1109/ICCVW.2019.00513`.

Kandakatla, Rajeshwari (2016). "Identifying offensive videos on YouTube". PhD thesis. Wright State University.

Lakomkin, Egor et al. (May 2019). “Incorporating End-to-End Speech Recognition Models for Sentiment Analysis”. In: pp. 7976–7982. DOI: `10.1109/ICRA.2019.8794468`.

Lihala, Anusha (Mar. 2019). *Attention and its Different Forms.* URL: `https://towardsdatascience.com/attention-and-its-different-forms-7fc3674d14dc`.

Loukas, George et al. (2018). “Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning”. In: *IEEE Access* 6, pp. 3491–3508. DOI: `10.1109/ACCESS.2017.2782159`.

MacAvaney, S. et al. (2019). “Hate speech detection: Challenges and solutions”. In: *JPloS one* 14(8) : e0221152. URL: `https://doi.org/10.1371/journal.pone.0221152`.

Mccallum, Andrew and Kamal Nigam (1998). “A Comparison of Event Models for Naive Bayes Text Classification”. In: *AAAI-98 Workshop on 'Learning for Text Categorization'.*

Njagi, Dennis et al. (Apr. 2015). “A Lexicon-based Approach for Hate Speech Detection”. In: *International Journal of Multimedia and Ubiquitous Engineering* 10, pp. 215–230. DOI: `10.14257/ijmue.2015.10.4.21`.

Phung, V.H. and E.J. Rhee (Jan. 2018). “A deep learning approach for classification of cloud image patches on small datasets”. In: *Journal of Information and Communication Convergence Engineering* 16, pp. 173–178. DOI: `10.6109/jicce.2018.16.3.173`.

Picone, J.W. (1993). “Signal modeling techniques in speech recognition”. In: *Proceedings of the IEEE* 81.9, pp. 1215–1247. DOI: `10.1109/5.237532`.

Poria, Soujanya et al. (Jan. 2019). “MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations”. In: pp. 527–536. DOI: `10.18653/v1/P19-1050`.

Rajagopal, Ilango (May 2018). *Batch Normalization — Speed up Neural Network Training.* `https://medium.com/@ilango100/batch-normalization-speed-up-neural-network-training-245e39a62f85`. [Online; Accessed 19-June-2022].

Razavi, Amir H. et al. (2010). “Offensive Language Detection Using Multilevel Classification”. In: *Advances in Artificial Intelligence.* Ed. by Atefeh Farzindar and Vlado Kešelj. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 16–27. ISBN: 978-3-642-13059-5. DOI: `10.1007/978-3-642-13059-5_5`.

Silva, Nadia Felix F. Da, Luiz F. S. Coletta, and Eduardo R. Hruschka (June 2016). “A Survey and Comparative Study of Tweet Sentiment Analysis via Semi-Supervised Learning”. In: *ACM Comput. Surv.* 49.1. ISSN: 0360-0300. DOI: `10.1145/2932708`. URL: `https://doi.org/10.1145/2932708`.

Stevens, S. S., John E. Volkmann, and Edwin B. Newman (1937). "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *Journal of the Acoustical Society of America* 8, pp. 185–190.

TensorFlow (n.d.). *Transformer model for language understanding.* URL: `https://www.tensorflow.org/text/tutorials/transformer`.

Thakkar, Vignesh, Suman Tewary, and Chandan Chakraborty (2018). "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data". In: *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1–5. DOI: `10.1109/EAIT.2018.8470438`.

Vishnubhatla, Suresh Kumar Venkata (1992). "A sample selective linear predictive analysis of speech signals". PhD thesis. DOI: `http://dx.doi.org/10.25669/6cfw-cbr8`.

Warner, William and Julia Hirschberg (2012). "Detecting hate speech on the world wide web". In: *Proceedings of the second workshop on language in social media*, pp. 19–26.

What is Speech Recognition?-IBM (Sept. 2020). Accessed: 2022-06-19. URL: `https://www.ibm.com/cloud/learn/speech-recognition`.

Wu, Ching Seh and Unnathi Bhandary (2020). "Detection of Hate Speech in Videos Using Machine Learning". In: *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 585–590. DOI: `10.1109/CSCI51800.2020.00104`.

Xie, Yue et al. (2019). "Speech Emotion Classification Using Attention-Based LSTM". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.11, pp. 1675–1685. DOI: `10.1109/TASLP.2019.2925934`.

Yamagishi, Junichi, Christophe Veaux, and Kirsten MacDonald (2019). *CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit.* DOI: `10.7488/ds/2645`.

Zadeh, Amir, Michael Chan, et al. (June 2019). "Social-IQ: A Question Answering Benchmark for Artificial Social Intelligence". In: pp. 8799–8809. DOI: `10.1109/CVPR.2019.00901`.

Zadeh, Amir, Paul Pu Liang, et al. (2018). "Multi-attention recurrent network for human communication comprehension". In: *Thirty-Second AAAI Conference on Artificial Intelligence.*