Project Report

On

"Transcribing phone call conversation at call centre and flagging the users for using foul language"

Submitted as a part of in-house training

In

Bachelors of Technology (B.Tech)

(Computer Science and Engineering)

Guided By:

Ms. Anjali Kapoor

Ms. Aakankshi Gupta

Submitted By:

Karan Gupta (01410402715)

Divyansh Gaba (01110407216)

Chander Negi (00810407216)

Varun Sharma (03010402715)

Department of Computer Science and Engineering

Amity School of Engineering and Technology

580, Delhi-Palam Vihar Road, Bijwasan, New Delhi-110061

[Affiliated to Guru Gobind Singh Indraprastha University, Delhi]

CERTIFICATE

This is to certify that below listed students of B.Tech. (Computer Science and Engineering) has successfully completed their In-House Summer Training Project entitled "Transcribing phone call conversation at call centre and flagging the users for using foul language"

Name	*Enrolment Number*
1. Karan Gupta	(01410402715)
2. Divyansh Gaba	(01110407216)
3. Chander Negi	(00810407216)
4. Varun Sharma	(03010402715)

Is an authentic work carried out by them under my guidance at AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY, NEW DELHI. The matter embodied in this project report has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

_____ _____

Ms. Anjali Kapoor Ms. Aakankshi Gupta

ACKNOWLEDGEMENT

Before getting into thick of things, we would like to add a few words of appreciation for people who have a part of this project right from its inception. It is to them we owe our deepest gratitude.

We are thankful to **Late Prof. BP. Singh,** Senior Director and **Prof. (Dr.) Rekha Aggarwal** for their moral support and providing us good infrastructure facilities of the institute which helped us in implementing the project. It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our respected **Prof. M.N. Gupta**, HOD, Department of CSE and IT and esteemed guidance, encouragement and help for completing this work.

We are also thankful to **Mr. Amrit Nath Thulal, Ms. Anjali Kapoor, and Ms. Aakankshi Gupta** for their timely suggestions, guidance and encouragement through this project. At the end, we would like to express our sincere thanks to all others who helped us directly or indirectly during this project work.

Karan Gupta	Divyansh Gaba	Chander Negi	Varun Sharma
(01410402715)	(01110402715)	(00710402715)	(03010402715)

Objective

To transcribe phone call conversations at call centres by implementing DeepMind's Wavenet neural network to perform speech recognition and then passing that transcription through a profanity filter to check for foul language.

List of Figures

Table of Contents

Chapter 1

Introduction

Speech is the primary means of communication between people. For reasons ranging from technological curiosity about the mechanisms for mechanical realization of human speech capabilities, to the desire to automate simple tasks inherently requiring human-machine interactions, research in automatic speech recognition (and speech synthesis) by machine has attracted a great deal of attention over the past five decades.

The purpose of this project is to implement Google's DeepMind WaveNet Neural Network model for end-to-end sentence level English Speech Recognition.

WaveNet is a deep neural network for generating raw audio created by researchers at London-based artificial intelligence firm DeepMind. The technique, outlined in a paper in September 2016, is able to generate realistic-sounding human-like voices by sampling real human speech and directly modelling waveforms. It is a type of feed-forward artificial neural network known as a deep convolutional neural network (CNN). These consist of layers of interconnected nodes similar to brain's neurons. The CNN takes a raw signal as an input and synthesises an output one sample at a time [1].

WaveNet is able to accurately model different voices, with the accent and tone of the input correlating with the output. For example, if it is trained on with Hindi, it produces Hindi speech.

This model can be straightforwardly used for Speech-to-text conversion [2].

In this project we used VCTK and TEDLIUM open-source speech corpus for training, validation and testing of the neural network.

These dataset includes speech data uttered by over 100 native speakers of English with various accents. Each speaker reads out about 400 sentences, most of which were selected from a newspaper, the Rainbow Passage and an elicitation paragraph intended to identify the speaker's accent. These dataset also includes transcription for each utterance which was manually transcribed.

First, all the audio files and their transcriptions were pre-processed. Mel-frequency cepstral coefficients (MFCCs) of each audio file was calculated and stored in a separate file. Their

transcriptions were converted to a sequence of integers by mapping each character of the alphabet to an integer value.

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The WaveNet model was implemented using TensorFlow, using convolutional and dialated convolutional layers. Connectionist temporal classification (CTC) was used as loss function and AdamOptimizer with a learning rate of 10^{-4} was used for optimizing weights of the network [3].

The flexible architecture of TensorFlow allows us to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

The extracted MFCC for audio files were used as features for speech recognition and labels were then used as input and expected output for supervised training of the neural network.

This model was trained for 50 epochs on the pre-processed data.

The audio files which are to be transcribed are then fed into the model and their transcription is produced as the output.

This transcription is then fed to Profanity filter for flagging if in case the user has used foul language.

Chapter 2

Project Analysis

2.1 Activity List

Activity	Predecessor	Duration (days)
A. Planning	None	2
B. Model layout	A	1
C. Dataset Preparation	B	2
D. Pre-processing of Dataset	C	2
E. Implementing WaveNet Model	B	5
F. Training of Model	D, E	4
G. Recognizing Module	E	3
H. Profanity Filter Implementation	B	1
I. Combining Modules	E, H	1
J. Testing	I	2

2.2 PERT Chart

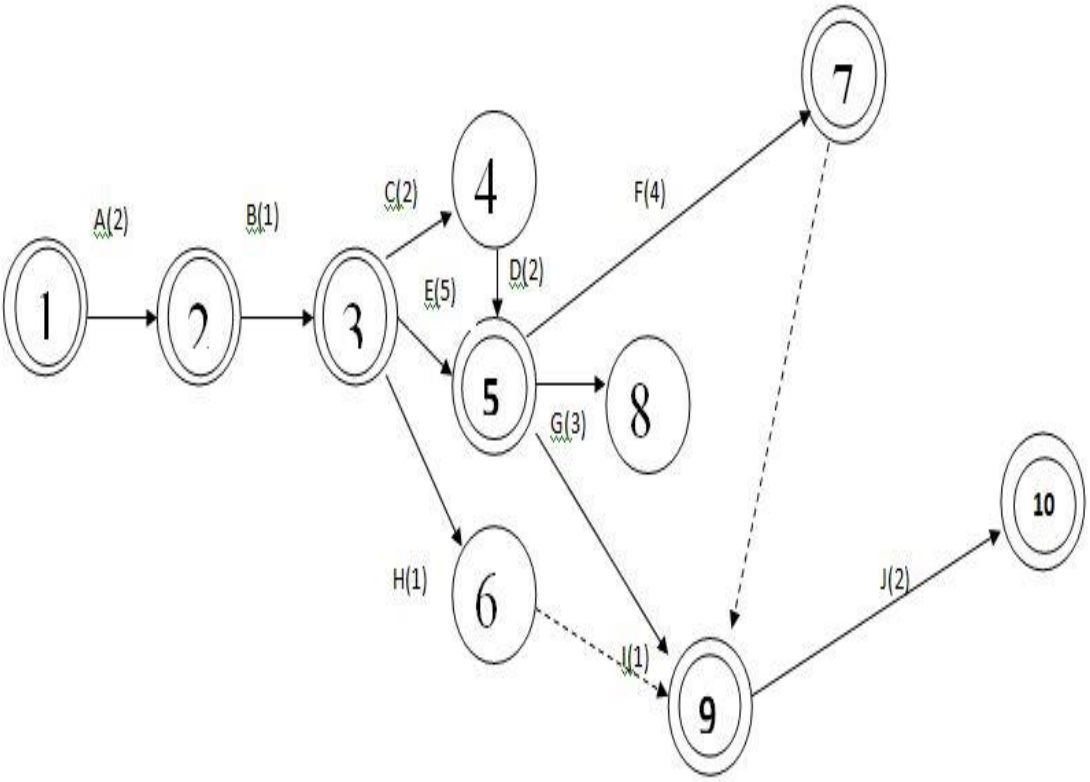


Figure 1 PERT chart based on Activity List

2.3 GANTT Chart

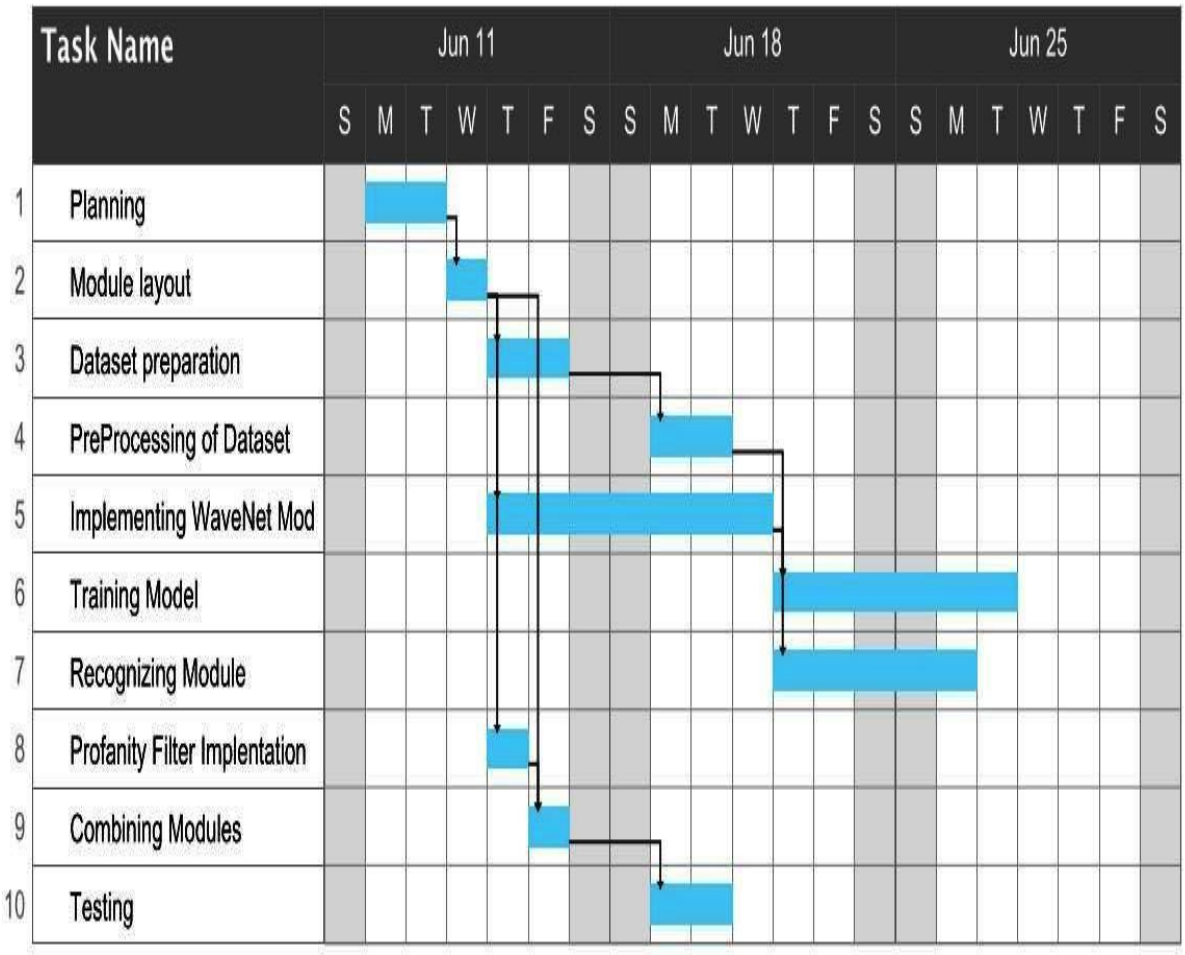
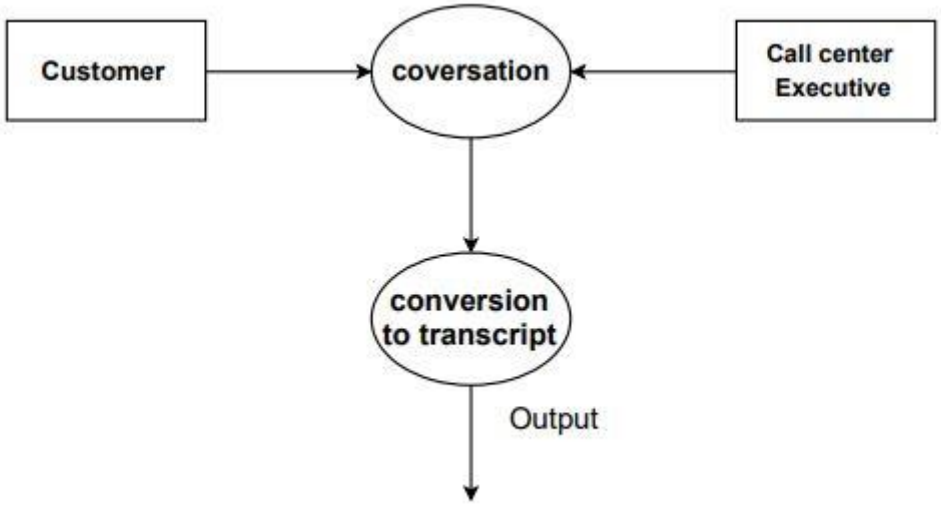


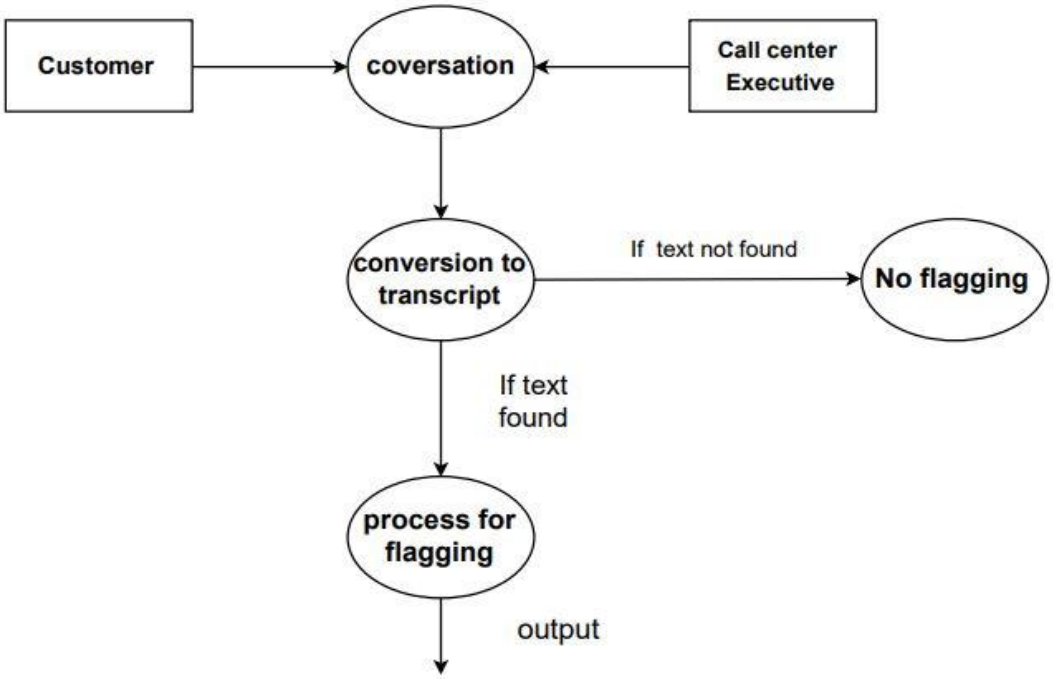
Figure 2 GANTT Chart based on Activity List

2.3 Data Flow Diagram

DFD level 0:



DFD level 1:



DFD level 2:

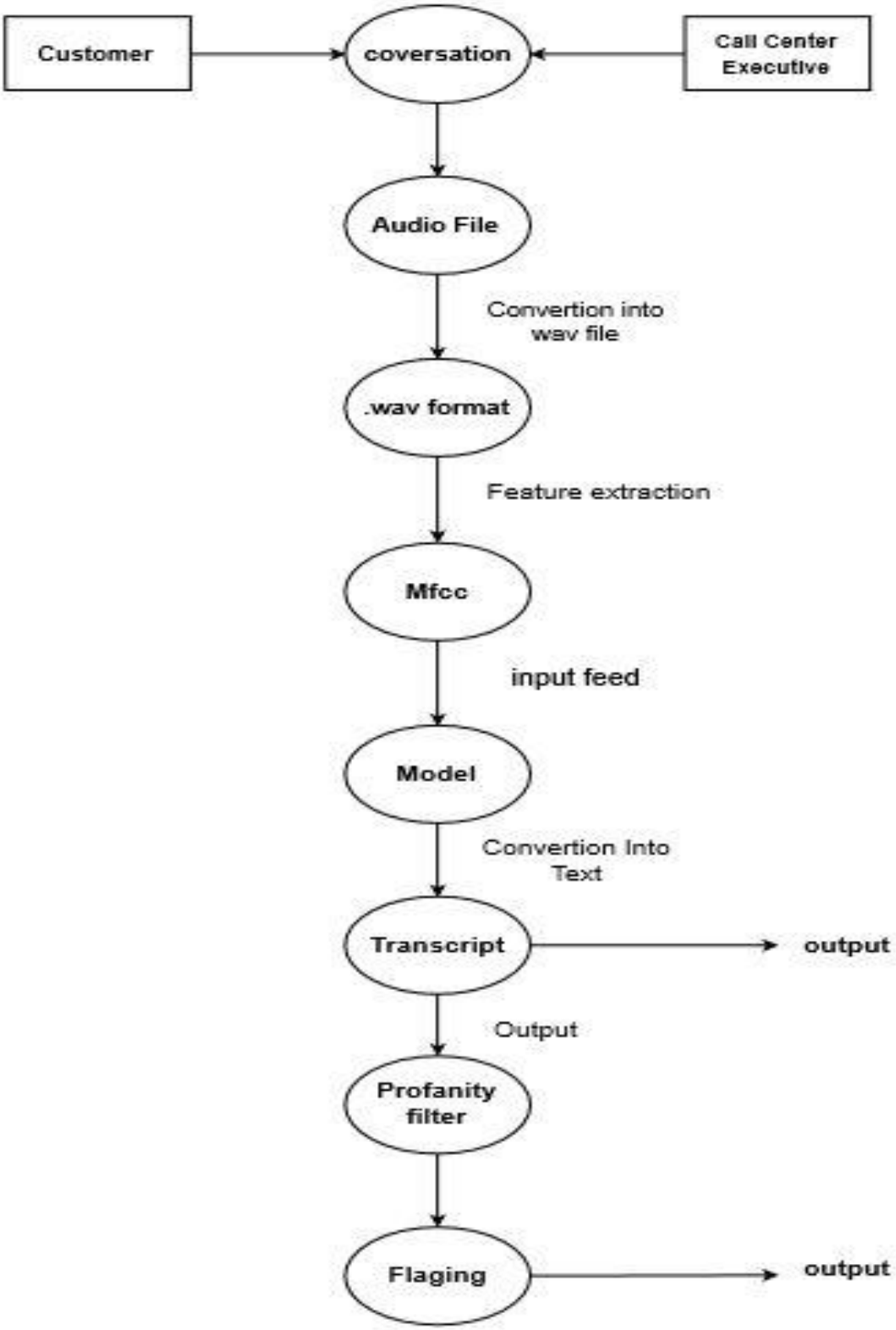


Figure 3 Data Flow Diagram

Chapter 3

System Requirements

3.1 Software Requirement:

This implementation of WaveNet can be run on any platform that supports Python 3.5.x and Tensorflow. As TensorFlow is still under development, this implementation might require some changes in future versions.

- **Framework**: Tensorflow
- **Languages**: Python
- **Operating System**: Linux/Windows 10 (64 Bit)
- **Other Software**: Python Idle

3.2 Hardware Requirements (Recommended):

- 16 GB RAM
- 250 GB HDD
- Intel I7 6th Gen 6820HK 4.10 GHz
- NVIDIA GTX 980M GPU

3.3 Packages Requirements:

- Tensorflow
- Pandas
- Numpy
- Csv
- Librosa
- Matplotlib
- Scikits.audiolab
- Glob

3.4 Dataset:

- TEDLIUM: The TED-LIUM corpus was made from audio talks and their transcriptions available on the TED website.
- VTCK Corpus: Speech corpus developed by The university of Edinburgh

Chapter 4

Implementation

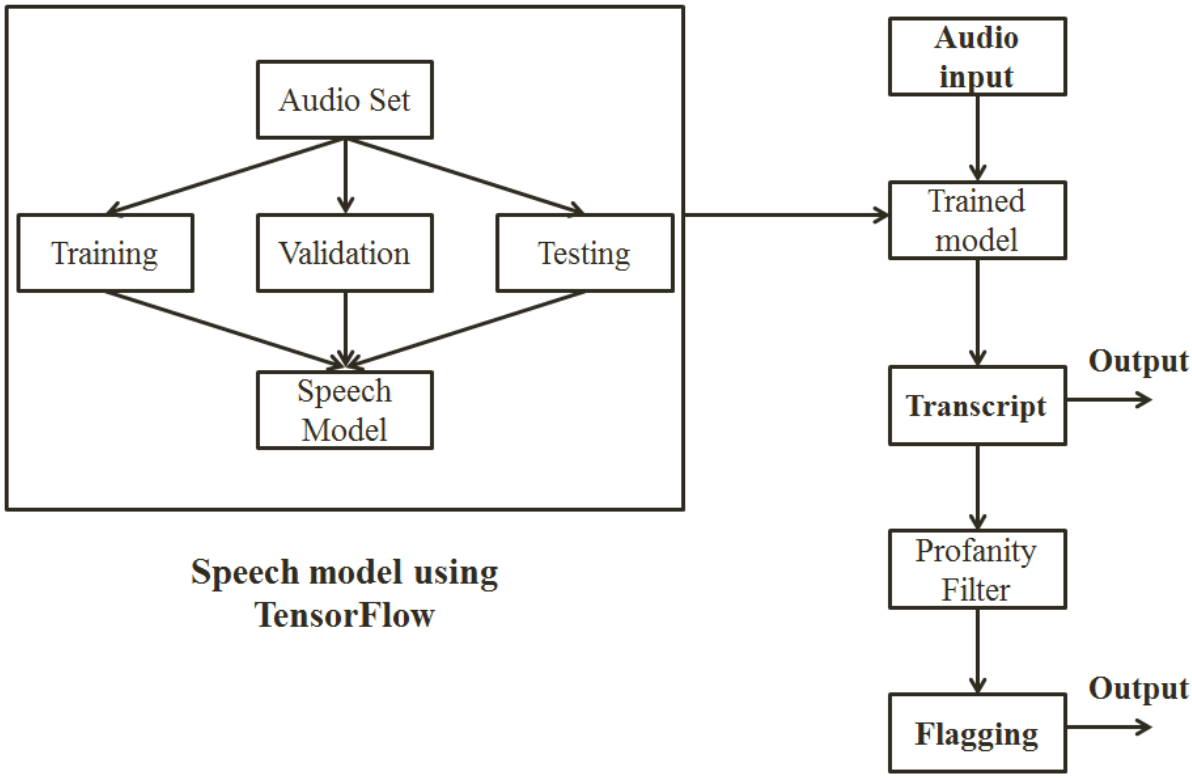


Figure 4 Implementation overview

4.1 Pre-processing of Audio Set

The audio files were first converted to WAVE (.wav) Format using ‘sox’ package.

Sampling rate for all audio files was set at 16000 samples per second.

A speech audio plot using ‘matplotlib’ is shown in Figure 5.

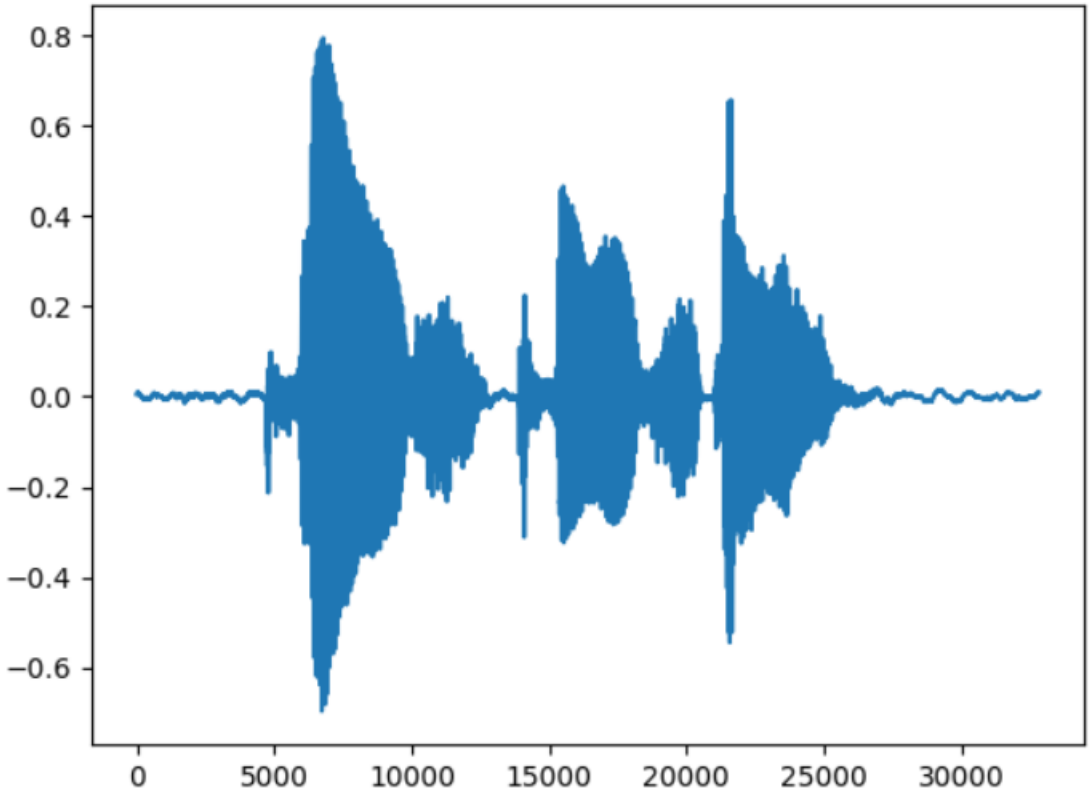


Figure 5 A speech signal

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound. Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear “spectrum-of-a-spectrum”). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system’s response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound [4].

MFCCs are commonly derived as follows:

- Take the Fourier transform of (a windowed excerpt of) a signal.
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.

- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

To calculate MFCCs we used python package called ‘librosa’. Plot for MFCCs of audio file in Figure 5 is shown in Figure 6.

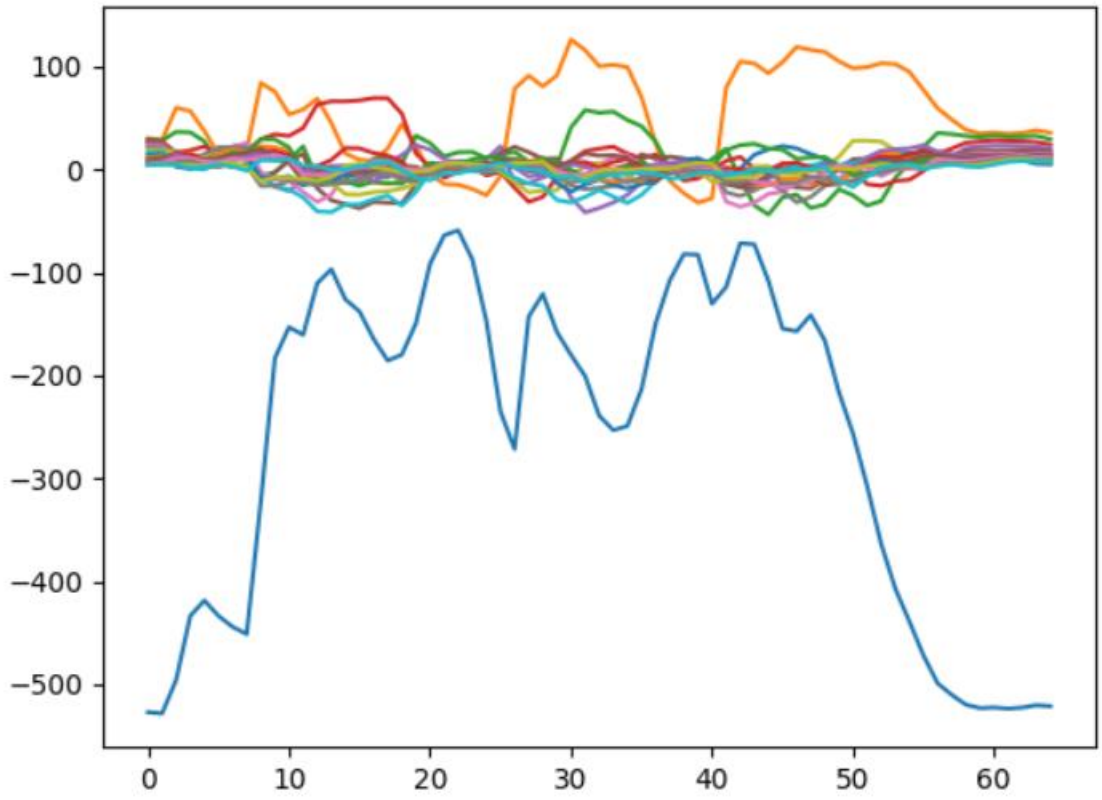


Figure 6 MFCC features of a speech signal

These features were stored in a separate file as arrays (.npy format) for later use.

The corresponding transcription for each of the audio file was pre-processed as well. Firstly, all the punctuation was removed and each character was then mapped to a single integer value. Each character of the transcription was replaced by its corresponding index shown in Figure 7.

```
vocabulary = ['<EMP>', ' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
              'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
              'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

Figure 7 Vocabulary used by the model

‘<EMP>’ is used in vocabulary to denote ‘empty’ meaning that no output is given for a corresponding frame. This is because the lengths of input and output sequence are not similar.

4.2 WaveNet Model

This model was created by researchers at London-based artificial intelligence firm DeepMind, a subsidiary of Google [2].

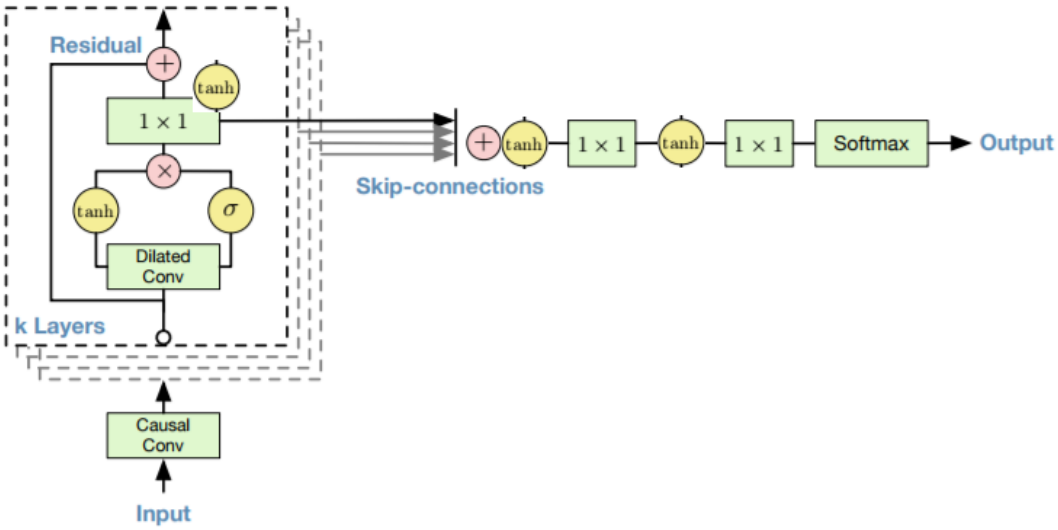


Figure 8 Overview of the residual block and the entire architecture

The main ingredient of WaveNet is use of Casual Convolutional layers. By using causal convolutions, we make sure the model cannot violate the ordering in which we model the data: the prediction emitted by the model at timestep t cannot depend on any of the future timesteps as shown in Fig. 9 [2].

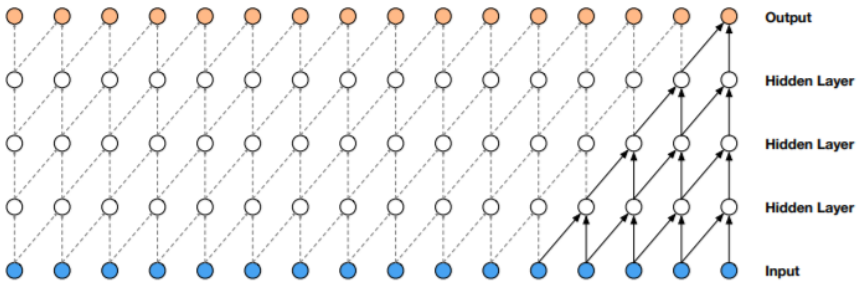


Figure 9 Visualization of a stack of causal convolutional layers

Models with causal convolutions do not have recurrent connections; they are typically faster to train than RNNs, especially when applied to very long sequences. One of the problems of causal convolutions is that they require many layers, or large filters to increase the receptive field. For example, in Figure 6 the receptive field is only 5 (= #layers + filter length – 1). In this Model, dilated convolutions are used to increase the receptive field by orders of magnitude, without greatly increasing computational cost.

A dilated convolution (also called atrous', or convolution with holes) is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient. This is similar to pooling or strided convolutions, but here the output has the same size as the input. As a special case, dilated convolution with dilation 1 yields the standard convolution. Figure 10 depicts dilated causal convolutions for dilations 1, 2, 4, and 8 [2].

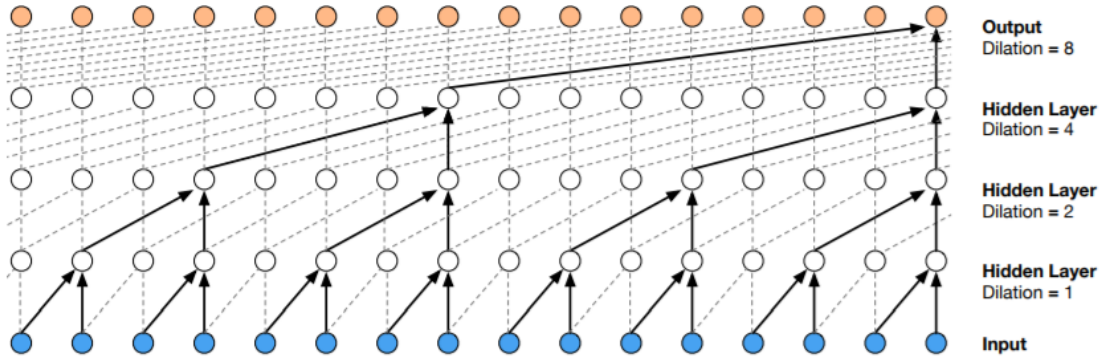


Figure 10 Visualization of a stack of dilated causal convolutional layers

Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency as shown in Figure 7.

Element-wise Multiplication of 'tanh' and 'sigmoid' was used as the gated activation unit.

Adamoptimizer from the tensorflow library was used with learning rate set to 10^{-4}.

Connectionist Temporal Classification (CTC) loss function was used as cost to minimize.

4.3 Training of network

The model was then trained on the pre-processed data. MFCC features of each file were used as input feed to the model and labels of transcriptions were used as true output value for optimizing weights of the network.

This training was done for 50 epochs of the dataset with a batch size of 4 files.

4.4 Profanity filter

Profanity is socially offensive language, which may also be called bad language, strong language, offensive language, coarse language, foul language, bad words, vulgar language, lewd language, choice words or expletives. The use of such language is called swearing, cursing or cussing.

We used a dictionary of bad words, with possible misspellings of curse words as our model may sometimes not be able to produce correct spelling.

We first loaded all the words and their corresponding rating in the memory using hash maps.

The transcript of the audio file was then read word by word and if that word is in the hash map of bad words, its rating is added to the total rating. If this total rating is above a threshold value, the user is then flagged.

Chapter 5

System Testing

System testing is the testing of a complete and fully integrated software product.

Usually software is only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware systems. System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

System test falls under the black box testing category of software testing. White box testing is the testing of the internal workings or code of a software application. In contrast, black box or system testing is the opposite. System test involves the external workings of the software from the user's perspective.
System testing involves testing the software code for following

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

5.1 Black Box Testing

Black Box Testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

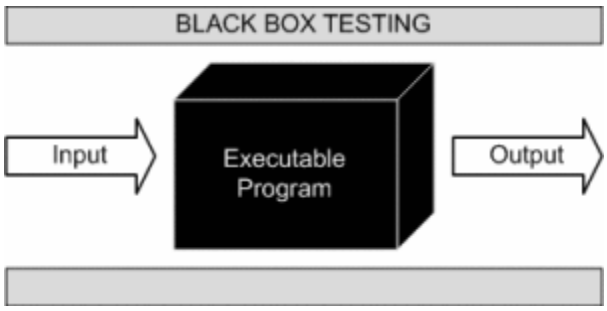


Figure 11 Black Box Testing

5.1.1 Black Box Testing Techniques

Following are some techniques that can be used for designing black box tests.

- Equivalence partitioning: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- Boundary Value Analysis: It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- Cause Effect Graphing: It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

5.1.2 Black Box Testing Advantages

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

5.1.3 Black Box Testing Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which are the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/ developer has already run a test case.

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behaviour or performance errors
- Initialization and termination errors

5.2 White Box Testing

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/ transparent box; inside which one clearly sees.

5.2.1 White Box Testing Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

5.2.2 White Box Testing Disadvantages

- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.

White Box Testing is contrasted with Black Box Testing.

5.3 Unit Testing

Unit testing is a level of software testing where individual units/ components of software are tested. The purpose is to validate that each unit of the software performs as designed.

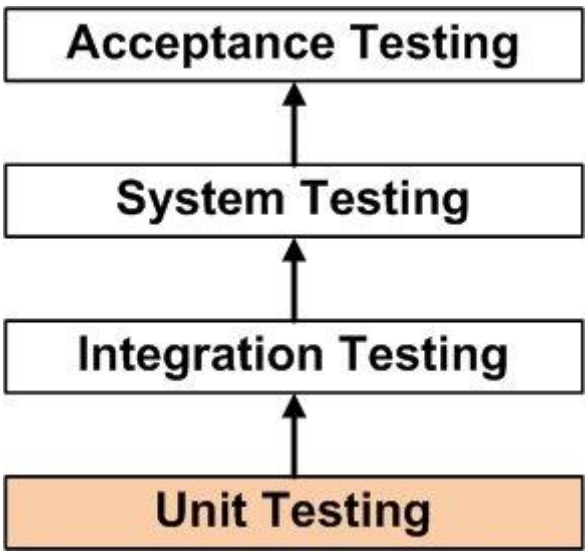


Figure 12 Levels of Testing

A unit is the smallest testable part of software. It usually has one or a few inputs and usually a single output. In procedural programming a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.)

Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

Unit Test Plan

- Prepare
- Review
- Rework
- Baseline

- Unit Test Cases/Scripts

    - Prepare
    - Review
    - Rework
    - Baseline
- Unit Test
    - Perform

Benefits

- Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
- Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.

5.4 Integration Testing

Integration Testing is a level of software testing where individual units are combined and tested as a group.

The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

- Integration testing: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
- Component integration testing: Testing performed to expose defects in the interfaces and
  interaction between integrated components.
- System integration testing: Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet).

5.4.1 Approaches for integration testing

- Big bang is an approach to Integration Testing where all or most of the units are combined together and tested at one go. This approach is taken when the testing team

receives the entire software in a bundle. So what is the difference between Big Bang Integration Testing and System Testing? Well, the former tests only the interactions between the units while the latter tests the entire system.

- Top Down is an approach to Integration Testing where top level units are tested first and lower level units are tested step by step after that. This approach is taken when top down development approach is followed. Test Stubs are needed to simulate lower level units which may not be available during the initial phases.
- Bottom Up is an approach to Integration Testing where bottom level units are tested first and upper level units step by step after that. This approach is taken when bottom up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases.
- Sandwich/Hybrid is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.

Chapter 6

Results

Audio files with the following quotes were used:

1. rainbow.wav

```
(tensorflow) D:\Divyansh\Documents\Summer-Training-Project>python predict.py data/extras/rainbow.wav
the rainbow is ta division of whide light into many beautful colors
```

True: The rainbow is a division of white light into many beautiful colors.

Predicted: **the rainbow is ta division of whide light into many beautful colors**

2. people.wav

```
(tensorflow) D:\Divyansh\Documents\Summer-Training-Project>python predict.py data/extras/people.wav
people look the noeal ever finds it
```

True: People look, but no one ever finds it.

Predicted: **people look the noeal ever finds it**

3. centuries.wav

```
(tensorflow) D:\Divyansh\Documents\Summer-Training-Project>python predict.py data/extras/centuries.wav
througout the centuries people have explained the rainbow in farious ways
```

True: Throughout the centuries people have explained the rainbow in various ways.

Predicted: **througout the centuries people have explained the rainbow in farious ways**

Chapter 7

Conclusion

This project helps Call Centre employees avoid dealing with annoying and disrespectful users. The proposed system transcribes the conversation and flags the users cursing too often during the conversation. By using Google's Deepmind wavenet, this project first converts the audio file of the pre-recorded conversations into a text file, and using the Profanity filter that we have developed in Python; it detects curse words, if found it flags the user as a spam.

Future Scope

We can add a language model to get correct spellings of misspelled words and a grammar model for punctuations. There is also scope to perform natural language processing for sentiment analysis.

References

[1]: Creators of WaveNet, DeepMind https://deepmind.com

[2]: WaveNet: A Generative Model for Raw Audio https://arxiv.org/abs/1609.03499

[3]: Open source TensorFlow Documentation https://github.com/tensorflow/tensorflow

[4]: Speech Recognition using MFCC https://www.researchgate.net/profile/Sloveby_Suksri/publication/281446199_Speech_Recognition_using_MFCC/links/55e7e61d08aeb6516262ec4c.pdf

Appendix

Audio_preprocessing.py

```
import numpy as np
import pandas as pd
import glob
import csv
import librosa
import os
import subprocess
import helper
import matplotlib.pyplot as plt

data_path = "data/"

def process_vctk(csv_file):

    writer = csv.writer(csv_f, delimiter =',')
    df = pd.read_table(data_path + "VCTK-Corpus/speaker-info.txt", usecols =['ID'],
                index_col=False, delim_whitespace=True)

    fid = []

    for d in [data_path + 'VCTK-Corpus/txt/p%d/' % uid for uid in df.ID.values]:
        fid.extend([f[-12:-4] for f in sorted(glob.glob(d+ '*.txt'))])

    for i,f in enumerate(fid):

        wave_f = data_path + 'VCTK-Corpus/wav48/%s/' % f[:4] + f + '.wav'
        fn = wave_f.split('/')[-1]
        targ_f = 'data/preprocess/mfcc/' + fn + '.npy'
        print("[VCTK corpus preprocess (%d/%d) - '%s']" % (i,len(fid),wave_f))
        wave, sr = librosa.load(wave_f, mono=True, sr=None)
        wave = wave[::3]
        mfcc = librosa.feature.mfcc(wave, sr=16000)
        mfcc = np.transpose(mfcc)
        plt.plot(mfcc)
        plt.show()
        contents = open(data_path + 'VCTK-Corpus/txt/%s/' % f[:4] + f + '.txt').read()
        label = helper.string_to_index(contents)
        if len(label) < mfcc.shape[1]:
            writer.writerow([fn] + label)
            np.save(targ_f,mfcc,allow_pickle=False)


def sph_to_wav(sph, wav):
    command = ['sox', '-t', 'sph', sph, '-b', '16', '-t', 'wav' ,wav]
    subprocess.check_call(command)
```

```
def process_ted(csv_file,category):

    path_to_parent = data_path + 'TEDLIUM-Corpus/' + category + '/'
    labels, wave_files, offsets, durations = [], [], [], []

    writer = csv.writer(csv_f, delimiter=',')

    stm_list = sorted(glob.glob(path_to_parent + 'stm/*'))
    stm_list = [stm.replace("\\","/") for stm in stm_list]
    for stm in stm_list:
        with open(stm,'rt') as f:
            records = f.readlines()
            for record in records:
                # format of each line in stm file is : <file_name_stm> 1 <file_name_sph>
<start_time> <end_time> <speaker-info> <words..>
                field = record.split()
                wave_f = path_to_parent + 'sph/%s.sph.wav' % field[0]
                wave_files.append(wave_f)
                labels.append(helper.string_to_index(' '.join(field[6:])))
                start, end  = float(field[3]), float(field[4])
                offsets.append(start)
                durations.append(end-start)

    for i, (wave_f, label, offset, duration) in enumerate(zip(wave_files, labels, offsets,
durations)):
            fn = "%s-%.2f" % (wave_f.split('/')[-1], offset)
            targ_f = 'data/preprocess/mfcc/' + fn + '.npy'
            if os.path.exists(targ_f):
                continue
            print("[TELDIUM corpus preprocessing (%d/%d) - '%s-%.2f']" % (i,
len(wave_files), wave_f, offset))
            if not os.path.exists(wave_f):
                sph_f = wave_f.rsplit('.',1)[0]
                if os.path.exists(sph_f):
                    sph_to_wav(sph_f, wave_f)
                else:
                    raise RuntimeError("Missing sph file : %s" % (sph_f))
            wave, sr = librosa.load(wave_f, mono=True, sr=None, offset=offset,
duration=duration)
            mfcc = librosa.feature.mfcc(wave, sr=16000)
            if len(label) < mfcc.shape[1]:
                writer.writerow([fn] + label)
                np.save(targ_f, mfcc, allow_pickle=False)


if not os.path.exists('data/preprocess'):
    os.makedirs('data/preprocess')
```

```
if not os.path.exists('data/preprocess/meta'):
   os.makedirs('data/preprocess/meta')
if not os.path.exists('data/preprocess/mfcc'):
   os.makedirs('data/preprocess/mfcc')

#VCTK
csv_f = open('data/preprocess/meta/train.csv', 'a+')
process_vctk(csv_f)
csv_f.close()


#TELDIUM
csv_f = open('data/preprocess/meta/train.csv','a+')
process_ted(csv_f,'train')
csv_f.close()
csv_f = open('data/preprocess/meta/test.csv', 'a+')
process_ted(csv_f, 'test')
csv_f.close()
csv_f = open('data/preprocess/meta/test.csv', 'a+')
process_ted(csv_f, 'dev')
csv_f.close()
```

model.py

```
import tensorflow as tf
import numpy as np

class Model():
   def __init__ (self, number_of_outputs, batch_size=1, n_mfcc=20, is_training=True):

      n_dim = 128
      self.is_training = is_training

      self.input_data = tf.placeholder(dtype=tf.float32, shape =[batch_size, None,
n_mfcc])
      self.seq_len = tf.reduce_sum(tf.cast(tf.not_equal(tf.reduce_sum(self.input_data,
reduction_indices=2), 0.), tf.int32), reduction_indices=1)
      self.targets = tf.placeholder(dtype=tf.int32, shape = [batch_size, None])
      self.conv1d_index = 0
      out = self.conv1d_layer(self.input_data, dim=n_dim)
      number_of_blocks = 3
      skip = 0
      self.aconv1d_index = 0
      for _ in range(number_of_blocks):
         for r in [1, 2, 4, 8, 16]:
            out, s = self.residual_block(out, size=7, rate=r, dim=n_dim)
            skip += s
      logit = self.conv1d_layer(skip, dim=skip.get_shape().as_list()[-1])
      self.logit = self.conv1d_layer(logit, dim=number_of_outputs, bias=True,
activation=None)
      indices = tf.where(tf.not_equal(tf.cast(self.targets, tf.float32),0.))
      values = tf.gather_nd(self.targets, indices)-1
      dense_shape = tf.cast(tf.shape(self.targets), tf.int64)
      target = tf.SparseTensor(indices=indices, values=values,
dense_shape=dense_shape)
      loss = tf.nn.ctc_loss(target, self.logit, self.seq_len, time_major=False)
      self.cost = tf.reduce_mean(loss)

      optimizer = tf.train.AdamOptimizer()

      var_list = [var for var in tf.trainable_variables()]
      gradients = optimizer.compute_gradients(self.cost, var_list=var_list)
      self.optimizer_op = optimizer.apply_gradients(gradients)

   def residual_block(self, input_tensor, size, rate, dim):
      conv_filter = self.aconv1d_layer(input_tensor, size=size, rate=rate,
activation='tanh')
      conv_gate = self.aconv1d_layer(input_tensor, size=size, rate=rate,
activation='sigmoid')
      out = conv_filter*conv_gate
      out = self.conv1d_layer(out, size=1, dim=dim)
```

```
        return out + input_tensor, out

    def conv1d_layer(self, input_tensor, size=1, dim=128, bias=False, activation='tanh'):
        with tf.variable_scope('conv1d_' + str(self.conv1d_index)):
            shape = input_tensor.get_shape().as_list()
            kernel_shape = (size, shape[-1], dim)
            kernel = tf.get_variable(name='kernel', shape=kernel_shape, dtype=tf.float32,
initializer=tf.contrib.layers.xavier_initializer())
            if bias:
                b_shape = (dim)
                b = tf.get_variable('b',shape=b_shape,dtype=tf.float32,
initializer=tf.constant_initializer(0))
            out = tf.nn.conv1d(input_tensor, kernel, stride=1, padding='SAME') + (b if bias
else 0)
            if not bias:
                out = self.batch_norm_wrapper(out)
            out = self.activation_wrapper(out, activation)
            self.conv1d_index += 1
            return out


    def aconv1d_layer(self, input_tensor, size=7, rate=2, bias=False, activation='tanh'):
        with tf.variable_scope('aconv1d_'+str(self.aconv1d_index)):
            shape = input_tensor.get_shape().as_list()
            kernel_shape = (1,size,shape[-1],shape[-1])
            kernel = tf.get_variable('kernel', kernel_shape, dtype=tf.float32,
initializer=tf.contrib.layers.xavier_initializer())
            if bias:
                b_shape = [shape[-1]]
                b = tf.get_variable('b',b_shape, dtype=tf.float32,
initializer=tf.constant_initializer(0))
            out = tf.nn.atrous_conv2d(tf.expand_dims(input_tensor, dim=2), kernel,
rate=rate, padding='SAME')
            out = tf.squeeze(out, [1])
            if not bias:
                out = self.batch_norm_wrapper(out)
            out = self.activation_wrapper(out, activation)
            self.aconv1d_index +=1
            return out

    def batch_norm_wrapper(self, inputs, decay=0.999):
        epsilon = 1e-3
        shape = inputs.get_shape().as_list()

        beta = tf.get_variable('beta',shape[-1],dtype=tf.float32,
initializer=tf.constant_initializer(0))
        gamma = tf.get_variable('gamma',shape[-1],dtype=tf.float32,
initializer=tf.constant_initializer(0))
```

```
      pop_mean = tf.get_variable('mean',shape[-1],dtype=tf.float32,
initializer=tf.constant_initializer(0))
      pop_var = tf.get_variable('variance', shape[-1], dtype=tf.float32,
initializer=tf.constant_initializer(1))
      if self.is_training:
         batch_mean, batch_var = tf.nn.moments(inputs, axes=list(range(len(shape)-1)))
         train_mean = tf.assign(pop_mean, pop_mean*decay + batch_mean*(1-decay))
         train_var = tf.assign(pop_var, pop_var*decay + batch_var*(1-decay))
         with tf.control_dependencies([train_mean, train_var]):
            return tf.nn.batch_normalization(inputs, batch_mean, batch_var, beta, gamma,
epsilon)
      else:
         return tf.nn.batch_normalization(inputs, pop_mean, pop_var, beta, gamma,
epsilon)

   def activation_wrapper(self, inputs, activation):
      out = inputs

      if activation == 'sigmoid':
         out = tf.nn.sigmoid(out)
      elif activation == 'tanh':
         out = tf.nn.tanh(out)
      elif activation == 'relu':
         out = tf.nn.relu(out)

      return out
```

train.py

```
import sugartensor as tf
from helper import SpeechLoader, vocab_size
from model import *

batch_size = 1

data = SpeechLoader(batch_size=batch_size)

inputs = tf.split(data.mfcc, 1, axis=0)

labels = tf.split(data.label, 1, axis=0)


seq_len = []
for input_ in inputs:
   seq_len.append(tf.not_equal(input_.sg_sum(axis=2), 0.).sg_int().sg_sum(axis=1))


@tf.sg_parallel
def get_loss(opt):
   logit = model(opt.input[0], voca_size=vocab_size)
   return logit.sg_ctc(target=opt.target[0], seq_len=opt.seq_len[0])

tf.sg_train(lr=0.0001, loss=get_loss(input=inputs, target=labels, seq_len=seq_len),
          ep_size=data.num_batch, max_ep=50)
```

predict.py

```
# -*- coding: utf-8 -*-
import sugartensor as tf
import numpy as np
import librosa
from model import *
import helper
import sys

batch_size = 1

voca_size = helper.vocab_size

x = tf.placeholder(dtype=tf.sg_floatx, shape=(batch_size, None, 20))
seq_len = tf.not_equal(x.sg_sum(axis=2), 0.).sg_int().sg_sum(axis=1)

logit = model(x, voca_size=voca_size)

decoded, _ = tf.nn.ctc_beam_search_decoder(logit.sg_transpose(perm=[1, 0, 2]), seq_len,
merge_repeated=False)
y = tf.sparse_to_dense(decoded[0].indices, decoded[0].dense_shape, decoded[0].values) + 1

tf.sg_arg_def(file=(str(sys.argv[1]), ''))
wav, _ = librosa.load(tf.sg_arg().file, mono=True, sr=16000)

mfcc = np.transpose(np.expand_dims(librosa.feature.mfcc(wav, 16000), axis=0), [0, 2, 1])

with tf.Session() as sess:

    tf.sg_init(sess)
    saver = tf.train.Saver()
    saver.restore(sess, tf.train.latest_checkpoint('data/model'))
    label = sess.run(y, feed_dict={x: mfcc})
    helper.print_index(label)
```

helper.py

```
import tensorflow
import numpy as np
import csv
import string
import matplotlib.pyplot as plt
import sugartensor as tf
import random
from datetime import timedelta
import time
data_path = 'data/'

vocabulary = ['<EMP>', ' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
          'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
          'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

vocab_index = {}
for i, ch in enumerate(vocabulary):
     vocab_index[ch] = i

vocab_size = len(vocabulary)

def string_to_index(contents):

     contents = ' '.join(contents.split())

     remove_punctuation = str.maketrans("","",string.punctuation)
     contents = contents.translate(remove_punctuation).lower()
     res = []
     for ch in contents:
          try:
               res.append(vocab_index[ch])
          except KeyError:
               pass
     return res
def index_to_string(contents):

     res = ''

     for c in contents:
          if c > 0:
               res += vocabulary[c]
          elif c == 0:
               break
     return res

def print_index(indices):
     for index in indices:
          print(index_to_string(index))
```

```
@tf.sg_producer_func
def _load_mfcc(src_list):

   label, mfcc_file = src_list
   label = np.fromstring(label, np.int)
   mfcc = []
   mfcc = np.load(mfcc_file, allow_pickle=False)
   mfcc = _augment_speech(mfcc)
   return label, mfcc


def _augment_speech(mfcc):

   r = np.random.randint(-2, 2)
   mfcc = np.roll(mfcc, r, axis=0)

   if r > 0:
      mfcc[:r, :] = 0
   elif r < 0:
      mfcc[r:, :] = 0

   return mfcc


class SpeechLoader(object):

   def __init__(self, batch_size=16, set_name='train'):

      label, mfcc_file = [], []
      with open(data_path + 'preprocess/meta/%s.csv' % set_name) as csv_file:
         reader = csv.reader(csv_file, delimiter=',')
         for row in reader:
            if len(row) == 0:
                  continue
            mfcc_file.append(data_path + 'preprocess/mfcc/' + row[0] + '.npy')
            label.append(np.asarray(row[1:], dtype=np.int).tostring())

      label_t = tf.convert_to_tensor(label)
      mfcc_file_t = tf.convert_to_tensor(mfcc_file)
      label_q, mfcc_file_q=tf.train.slice_input_producer([label_t, mfcc_file_t],
shuffle=True)

      label_q, mfcc_q = _load_mfcc(source=[label_q, mfcc_file_q],dtypes=[tf.sg_intx,
tf.sg_floatx],capacity=256, num_threads=64)

      batch_queue = tf.train.batch([label_q, mfcc_q], batch_size,shapes=[(None,), (20,
None)],num_threads=64, capacity=batch_size*32,dynamic_pad=True)
```

```
self.label, self.mfcc = batch_queue
self.mfcc = self.mfcc.sg_transpose(perm=[0, 2, 1])
self.num_batch = len(label) // batch_size
tf.sg_info('%s set loaded.(total data=%d, total batch=%d)'
        % (set_name.upper(), len(label), self.num_batch))
```

Profanity.py

```
import os
import sys
bad_word_rate_sum = 0
total_words = 0
bad_word_dict = {}


def init():
   f = open("data/profanity/bad_words.txt","r")
   for line in f:
      word, rating = line.split(",")
      rating = int(rating)
      bad_word_dict[word] = rating

def rating(input_file):
   f = open(input_file)
   for line in f.readlines():
      words = line.split(' ')
      for word in words:
         global total_words
         total_words += 1
         if word in bad_word_dict:
            global bad_word_rate_sum
            bad_word_rate_sum += bad_word_dict[word]
   return bad_word_rate_sum / total_words


if __name__ = '__main__':
   if sys.argv != 3:
      print("Incorrect command !\n Usage: pythong profanity.py input.txt")
      exit()
   if sys.argv[2].split('.')[-1] != 'txt':
      print("Incorrect File format! Please give a text file")
      exit()
   init()
   print(rating(sys.argv[2]))
```