B. TECH. PROJECT REPORT

On

ADVERSARIAL ATTACKS ON AUDIO

BY

Harsh Vardhan (160001025)
Vishal Maurya (160001061)
Pranay Munda (160001045)

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

November 2019

ADVERSARIAL ATTACKS ON AUDIO

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Harsh Vardhan, Vishal Maurya, Pranay Munda

Guided by:
Dr. Puneet Gupta, Assistant Professor

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November, 2019

CANDIDATE’S DECLARATION

We hereby declare that the project entitled **“Adversarial Attacks On Audio”** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in ‘Computer Science And Engineering’ completed under the supervision of **Dr. Puneet Gupta,** IIT Indore is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Signature of BTP Guide(s) with dates and their designation

<u>Preface</u>

This report on “**Adversarial Attacks On Audio**" is prepared under the guidance of **Dr. Puneet Gupta**, Assistant Professor.

Through this project we have aimed to demonstrate targeted adversarial attacks on audio. We have covered every aspect of the project from prerequisites and preparations needed to set up the environment for the project to how to actually improve the quality of the attack.

We have tried to the best of our abilities and knowledge to explain the content in a clear manner while tried not to include any unnecessary details and have added tables and figures to make it more illustrative to help understand the concepts very easily.

Harsh Vardhan, Vishal Maurya, Pranay Munda
B.Tech. 4th Year
Discipline of Computer Science And Engineering
IIT Indore

Acknowledgements

We wish to thank **Dr. Puneet Gupta** for his kind support and valuable guidance. It is with his help and support, due to which we were able to complete the project. On every step, he reviewed and provided us with valuable feedback and resources because of which we were able to pass through many obstacles. Under his guidance, we acquired many skills as the project proceeded. He was always there to support with any kind of help he could provide. Without his support this project would not have been possible.

Harsh Vardhan, Vishal Maurya, Pranay Munda
B.Tech. 4th Year
Discipline of Computer Science And Engineering
IIT Indore

Abstract

We know that neural networks are vulnerable where we provide slightly different input and classified by the neural network as an incorrect target. As of now image domain has been studied most extensively in contrast to audio domain where it is much more difficult to deal with because it is very easy for humans to detect perturbations, and they do not work when we play it over the air while in the case of images it is relatively easy to create illusions to the eye. We then construct an audio attack that is targeted on automatic speech recognition (which is open-source not proprietary like Siri Or Cortana) in which the output is what we want. We can easily produce another waveform from any audio waveform that sounds similar to the original one and produces any output sentence we choose. White box iterative optimization based attack is done on the open-source Mozilla's implementation of DeepSpeech automatic speech recognition, and show that it works perfectly. And finally, we make the noise introduced in the generated audio imperceptible through the application of psychoacoustics.

Contents

Candidate's Declaration **5**

Certificate By BTP Guide **5**

Preface **6**

Acknowledgments **7**

Abstract **8**

Table Of Contents **9**

List Of Figures **11**

1. Introduction **13**

2. Background **14**

2.1 Psychoacoustics 14

3. Problem Description And The Proposed Solution **16**

4. Frequency Masking And Psychoacoustic Model **17**

5. Challenges Faced **19**

5.1 Google Colab Cloud Platform 19

6. Experiments Performed **21**

6.1 For 1000 iterations (with Psychoacoustic) 22

7. Results Obtained **24**

7.1 What the graph signifies 24

Conclusion **27**

References **28**

List of Figures

- **Figure 1.** Illustration of our attack.
- **Figure 2.** Illustration of the concept of psychoacoustics.
- **Figure 3.** Code Snippet for Normalized PSD.
- **Figure 4.** Code Snippet for Two Slops Method.
- **Figure 5.** Code Snippet for Generating Threshold.
- **Figure 6.** Interface of Google Colab (Landing page).
- **Figure 7.** Interface of Google Colab (Execution of the code.)
- **Figure 8.** Command 1.
- **Figure 9.** Output 1.
- **Figure 10.** Command 2.
- **Figure 11.** Output 2.
- **Figure 12.** Command 3.
- **Figure 13.** Output 3.
- **Figure 14.** Plotting the graph using Matplotlib.
- **Figure 15.** Graph Comparison 1.
- **Figure 16.** Graph Comparison 2.
- **Figure 17.** Graph Comparison 3.

Chapter 1

Introduction

As the use of neural networks continues to grow, it is critical to examine their behaviour in adversarial settings. Existing work on adversarial examples has focused largely on the space of images, be it image classification, image segmentation, face detection, etc. There has been comparatively little study on the space of audio, where the most common use is performing automatic speech recognition. In automatic speech recognition, a neural network is given an audio waveform ***x*** and perform the speech-to-text transform that gives the transcription ***y*** of the phrase being spoken. Constructing targeted adversarial examples on speech recognition has proven difficult. Hidden and inaudible voice commands are targeted attacks, but require synthesizing new audio and cannot modify existing audio (analogous to the observation that neural networks can make high confidence predictions for unrecognizable images). In Fig. 1. a noise is added to an audio sample which fools the neural network to produce different output which is the required target phrase.

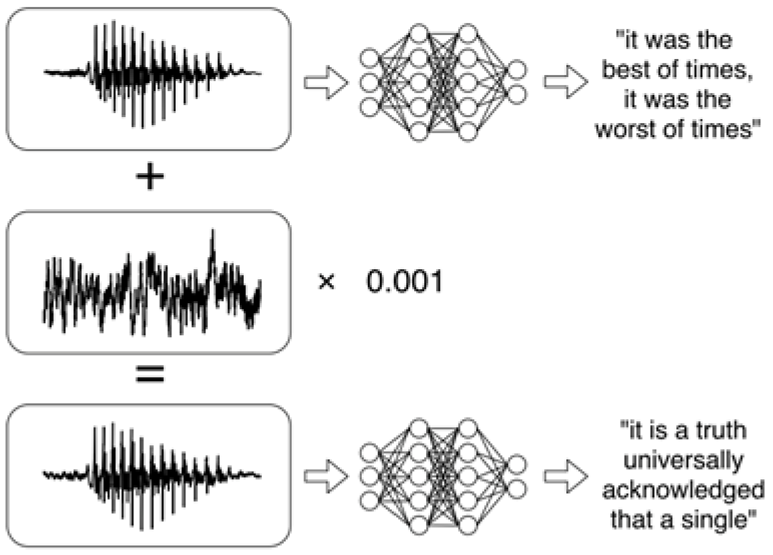

"it was the best of times, it was the worst of times"
+
× 0.001
=
"it is a truth universally acknowledged that a single"

Figure. 1. Illustration of our attack: given any waveform, adding a small perturbation makes the result transcribe as any desired target phrase. [1].

Chapter 2

Background

A differentiable parameterized function ***f(•)*** used in a neural network that has parameters that can be easily updated by standard techniques like gradient descent which can be used to learn any function. We represent audio as an N-dimensional vector $\boldsymbol{x}$. Each element $\boldsymbol{x_i}$ is a signed 16-bit value, sampled at 16KHz. To reduce the input dimensionality, the Mel-Frequency Cepstrum (MFC) transform is often used as a preprocessing step [5]. The Mel-Frequency Cepstrum (MFC) is used to split the waveform in 50 frames per second. Further it maps each frame into its corresponding frequency range. Standard classification of neural networks take a single input and produces a probability distribution as output over all labels. But, for speech-to-text (ASR) systems, there are numerous possible labels, which makes it computationally very difficult for enumeration on all the possible phrases. Connectionist Temporal Classification (CTC) [7] is a process of training a neural network which is sequence-to-sequence that is when we do not know the alignment between the input and output sequences. We have used DeepSpeech [6] which internally uses CTC as the inputs are the voice of a person and the sentences which are arranged in a non-aligned fashion, in which we don't know the exact location of each word in the audio.

2.1 Psychoacoustics

Psychoacoustic is a phenomenon where the louder tone played masks (hides) the milder tone nearby. We will not hear a milder tone for sometime when a louder tone is being played at the same time. Here we are talking in terms of frequency. Since by using CTC as our loss function, we could not minimize the perturbation we used an additional frequency masking threshold we minimize the distortion so that the audio generated is imperceptible to noise. In Fig. 2. the tone with higher decibel hides the softer tone nearby for a few seconds. Here the louder tone is called as the masking tone and the tone under the curve mentioned in Fig. 2. is inaudible due to the masking effect.

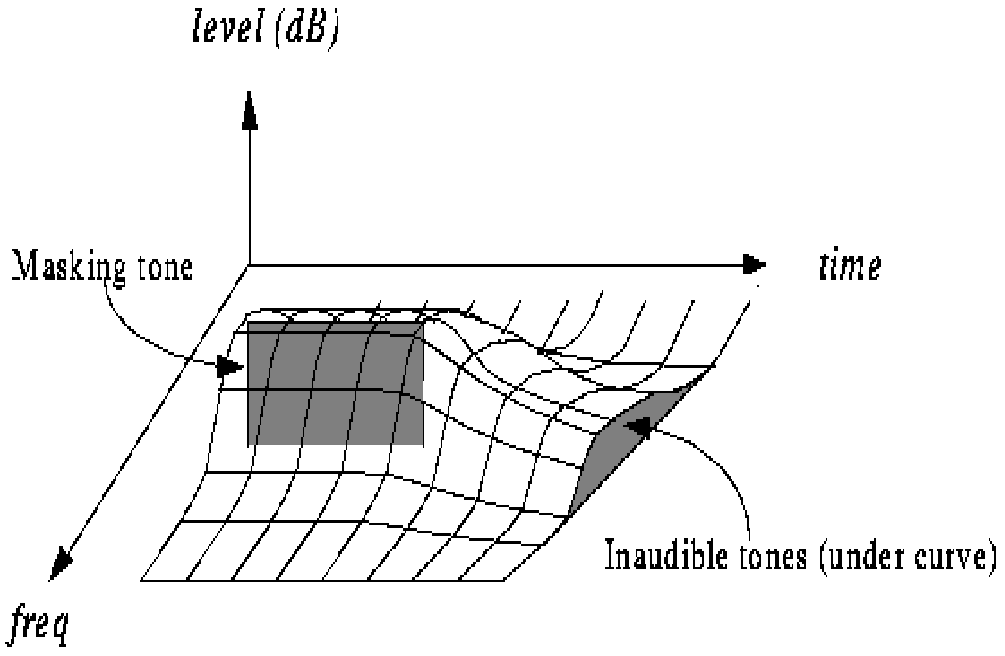


*Figure. 2. Illustration of the concept of psychoacoustics. (**Reference: https://users.ece.utexas.edu/~ryerraballi/MSB/pdfs/M4L4.pdf).***

Chapter 3

Problem Description And The Proposed Solution

Given any audio signal $\boldsymbol{x}$, and any target phrase $\boldsymbol{y}$, Another audio signal can be produced such that the input is $\boldsymbol{x_0 = x + \delta}$ in such a manner that $\boldsymbol{x}$ and $\boldsymbol{x_0}$ sound familiar and also $\boldsymbol{C(x_0) = y}$ where $\boldsymbol{C(.)}$ is an output function for any given audio. Success is only reported when the output exactly matches the target phrase. To quantify the introduced distortion, we measure it in Decibels (dB).

$$dB(x) = \max 20 \cdot \log10(x_i). \qquad \ldots\text{(i)}$$

where $\boldsymbol{x}$ is the audio input vector and dB level is compared with original waveform $\boldsymbol{x}$ with the distortion $\boldsymbol{\delta}$. So we can formulate this as follows:

$$dB_x(\delta) = dB(\delta) - dB(x) \qquad \ldots\text{(ii)}$$

This problem is finally constructed as an optimization problem of which the formulas are mentioned below:

$$\textbf{minimize } dB_x(\delta)$$

$$\textbf{such that } C(x + \delta) = t$$

$$x + \delta \in [-M, M] \qquad \ldots\text{(iii)}$$

Here, $\boldsymbol{M}$ signifies the maximum permissible value (2^{15} in our case). Since the constraint $\boldsymbol{C(x + \delta) = t}$ is non-linear, we cannot use standard techniques like gradient descent. It is shown in [3] have shown that we can reformulate the problem in equation (iii) as:

$$\textbf{minimize } dB_x(\delta) + c \cdot l\,(x + \delta, t) \qquad \ldots\text{(iv)}$$

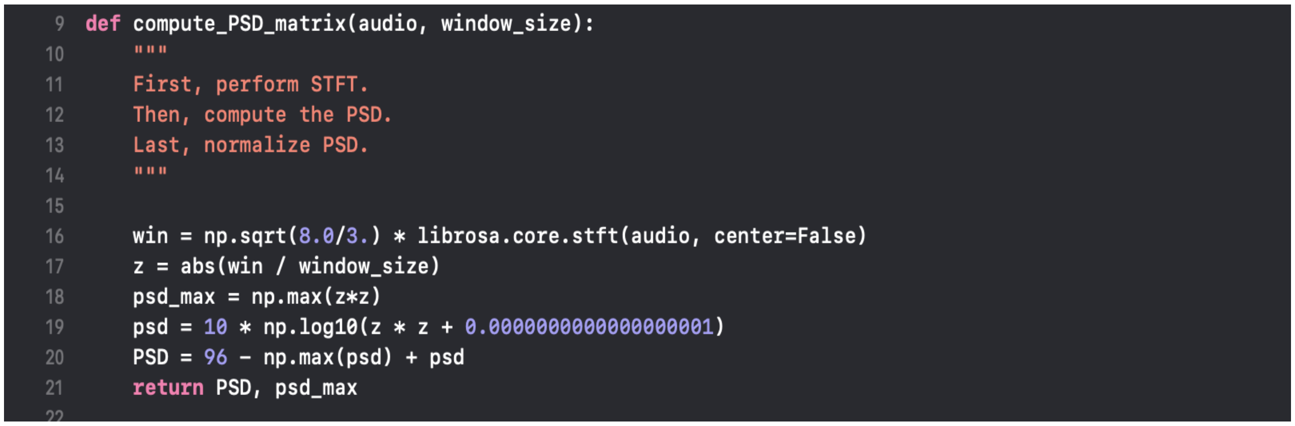
where $\boldsymbol{l}$ is the CTC-loss function. We use Adam optimizer [4] to solve this minimization problem where we use a learning rate as 10, and the maximum number iterations done are 5,000. We can easily find it's implementation in an open-source library known as Tensorflow, it is also available in multiple programming languages according to user's need.

Chapter 4

Frequency Masking and Psychoacoustic Model

In order to make the attack imperceptible, we will use the psychoacoustic model with frequency masking techniques. This will help us significantly in reducing the noise in output audio after the attack. Frequency masking refers to a process where the louder tone called “masker” mask the softer tone around it. The following steps are used to compute the frequency masking threshold:

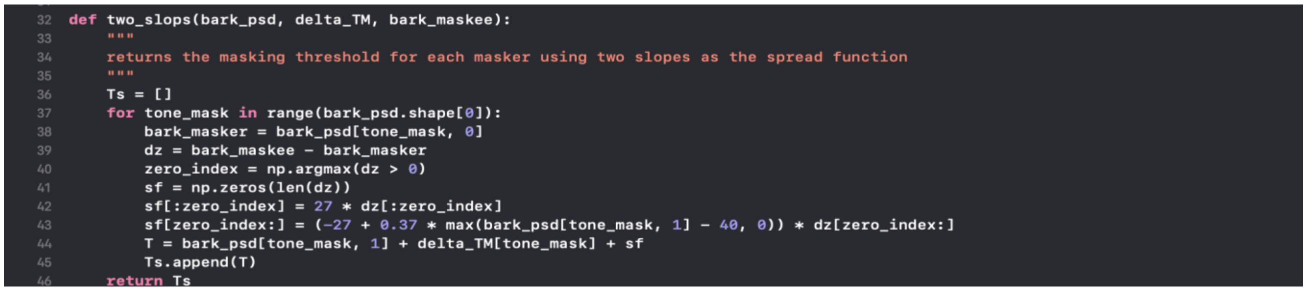
- Identify the masker (i.e. the louder parts of audio).
- In order to compute the frequency masking threshold we need to compute normalized PSD [2] as the audio signals change rapidly over time [8]. In Fig. 3. the same has been implemented by passing the audio and the corresponding window size which finally returns *PSD* and *psd_max*.



```python
def compute_PSD_matrix(audio, window_size):
    """
    First, perform STFT.
    Then, compute the PSD.
    Last, normalize PSD.
    """

    win = np.sqrt(8.0/3.) * librosa.core.stft(audio, center=False)
    z = abs(win / window_size)
    psd_max = np.max(z*z)
    psd = 10 * np.log10(z * z + 0.0000000000000000001)
    PSD = 96 - np.max(psd) + psd
    return PSD, psd_max
```

Figure. 3. Code Snippet for Normalized PSD

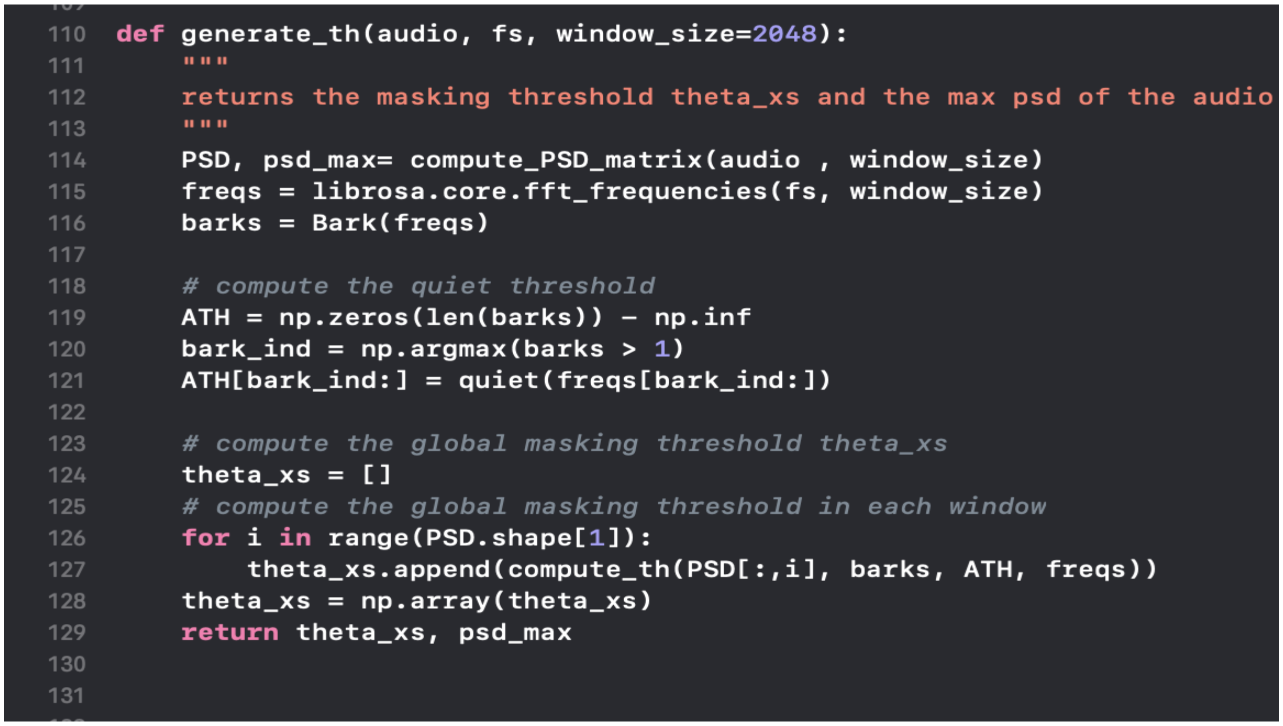
- Ensure that there must be a local maximum in the spectrum of PSD.
- PSD should also have its peak amplitude lower than 0.5 Bark (a psychoacoustically-motivated frequency scale) [2].
- After that, two-slope spread function is used to approximate the masking threshold [2]. In Fig. 4. the same has been implemented in python where the function is returning the masking threshold the individual maskers.



```python
def two_slops(bark_psd, delta_TM, bark_maskee):
    """
    returns the masking threshold for each masker using two slopes as the spread function
    """
    Ts = []
    for tone_mask in range(bark_psd.shape[0]):
        bark_masker = bark_psd[tone_mask, 0]
        dz = bark_maskee - bark_masker
        zero_index = np.argmax(dz > 0)
        sf = np.zeros(len(dz))
        sf[:zero_index] = 27 * dz[:zero_index]
        sf[zero_index:] = (-27 + 0.37 * max(bark_psd[tone_mask, 1] - 40, 0)) * dz[zero_index:]
        T = bark_psd[tone_mask, 1] + delta_TM[tone_mask] + sf
        Ts.append(T)
    return Ts
```

Figure. 4. Code Snippet for Two Slope Method

- Finally, we compute the global masking threshold by combining each of the individual masking threshold and threshold in quiet via addition. In the following Fig. 5. the function is returning individual masking threshold for each window size where *theta_xs* is the masking threshold.



```python
def generate_th(audio, fs, window_size=2048):
    """
    returns the masking threshold theta_xs and the max psd of the audio
    """
    PSD, psd_max= compute_PSD_matrix(audio , window_size)
    freqs = librosa.core.fft_frequencies(fs, window_size)
    barks = Bark(freqs)

    # compute the quiet threshold
    ATH = np.zeros(len(barks)) - np.inf
    bark_ind = np.argmax(barks > 1)
    ATH[bark_ind:] = quiet(freqs[bark_ind:])

    # compute the global masking threshold theta_xs
    theta_xs = []
    # compute the global masking threshold in each window
    for i in range(PSD.shape[1]):
        theta_xs.append(compute_th(PSD[:,i], barks, ATH, freqs))
    theta_xs = np.array(theta_xs)
    return theta_xs, psd_max
```

Figure. 5. Code Snippet for Generating Threshold

After adding ***δ*** as a perturbation to the input audio ***x***, ***δ*** will be masked out by the raw audio if the normalized PSD ***p'δ(k)*** is below the frequency masking threshold of the original waveform and hence it will be inaudible to humans.

Chapter 5

Challenges Faced

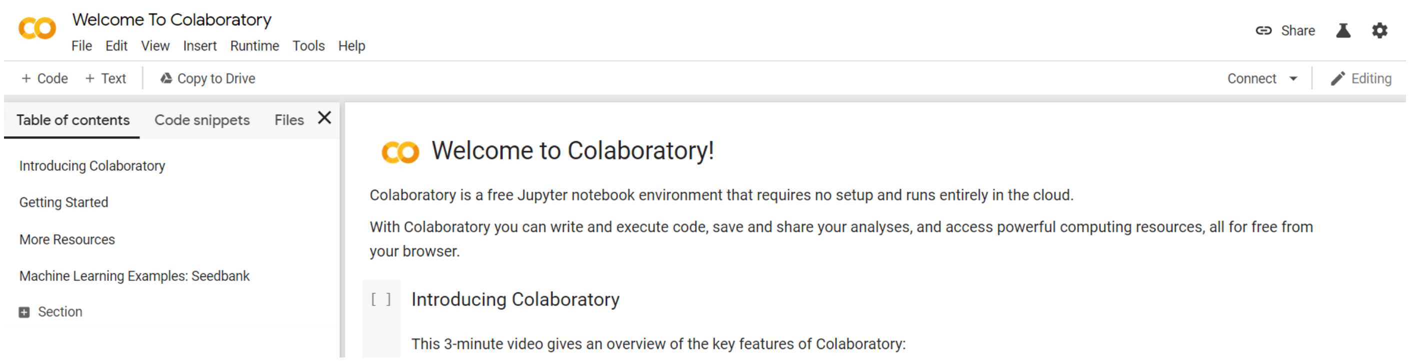
While working on this topic, we faced the following challenges:

- Limited resources (e.g. less ram and lower GPU) made it difficult for us to run the code for larger iterations like 1000.
- Understanding the relationship between psychoacoustics model and adversarial attacks.
- Very less documentation on psychoacoustics model made it difficult for us to implement.

5.1 Google Colab Cloud Platform (https://colab.research.google.com/)

As mentioned above, we were facing problems due to the limited resources of our local computer. We used **Google Colab** to overcome this problem.

- It provides us with 12 GBs of RAM and over 300 GBs of disk space with Tesla K80 GPU which significantly improve the performance and speed of execution of the codebase.
- We were able to run our code for a larger number of iterations without any problem.
- As can be seen in Fig. 6. the interface of Google colab jupyter notebook is quite user friendly. The welcome page explains the working and settings needed to set up the environment. While in Fig. 7. we can see the code execution in action after setting up the environment (i.e. selecting the GPU, connecting to the ram, etc.). It executes the code in chunks and any piece of a code block can be inserted anywhere between any two blocks.


Welcome To Colaboratory
File Edit View Insert Runtime Tools Help
Share
+ Code + Text Copy to Drive
Connect Editing
Table of contents Code snippets Files
Introducing Colaboratory
Getting Started
More Resources
Machine Learning Examples: Seedbank
Section

Welcome to Colaboratory!

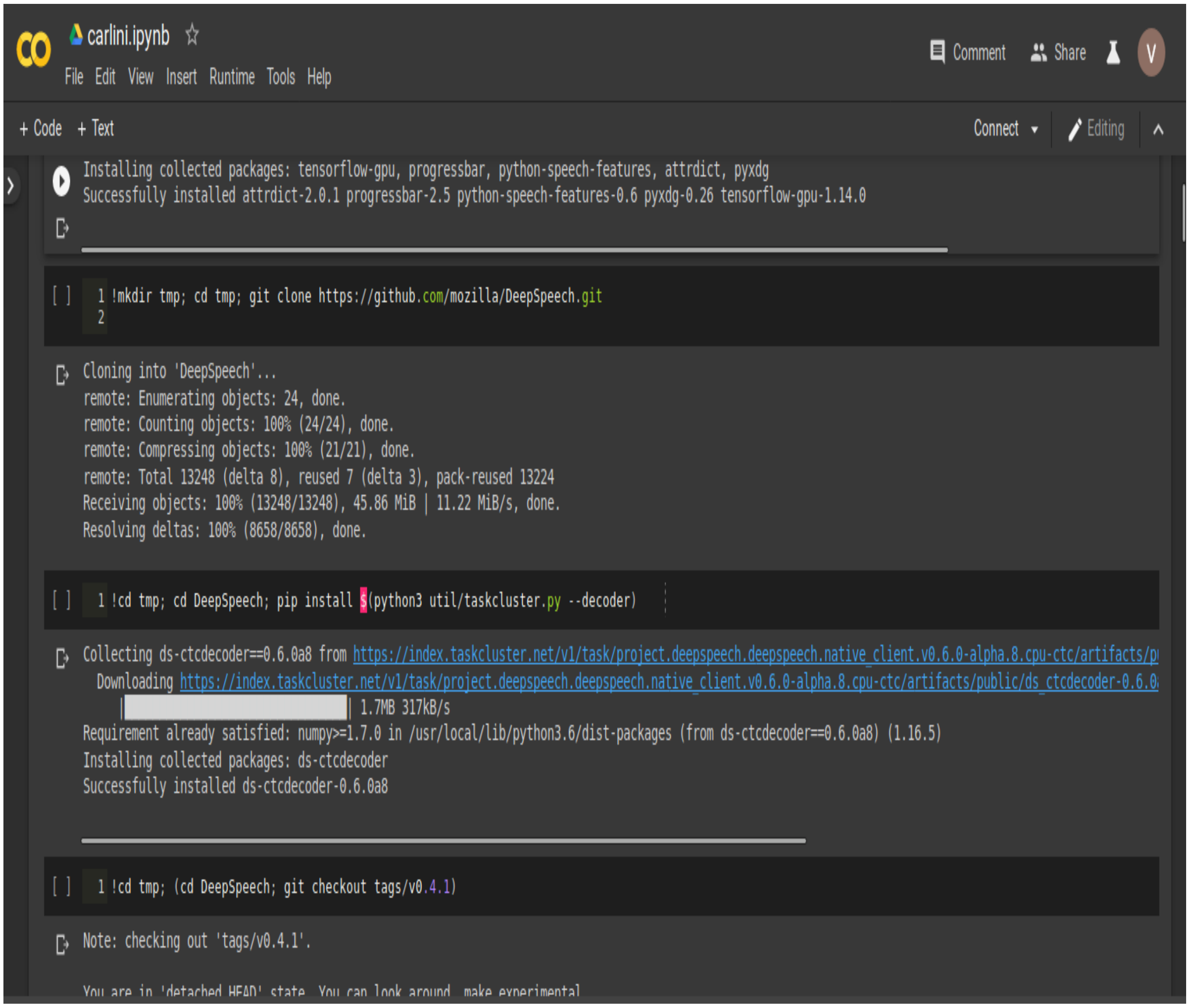
Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

Introducing Colaboratory

This 3-minute video gives an overview of the key features of Colaboratory:

Figure. 6. Interface of Google Colab Jupyter Notebook (Landing Page).



```
Installing collected packages: tensorflow-gpu, progressbar, python-speech-features, attrdict, pyxdg
Successfully installed attrdict-2.0.1 progressbar-2.5 python-speech-features-0.6 pyxdg-0.26 tensorflow-gpu-1.14.0
```

```
!mkdir tmp; cd tmp; git clone https://github.com/mozilla/DeepSpeech.git
```

```
Cloning into 'DeepSpeech'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 13248 (delta 8), reused 7 (delta 3), pack-reused 13224
Receiving objects: 100% (13248/13248), 45.86 MiB | 11.22 MiB/s, done.
Resolving deltas: 100% (8658/8658), done.
```

```
!cd tmp; cd DeepSpeech; pip install $(python3 util/taskcluster.py --decoder)
```

```
Collecting ds-ctcdecoder==0.6.0a8 from https://index.taskcluster.net/v1/task/project.deepspeech.deepspeech.native_client.v0.6.0-alpha.8.cpu-ctc/artifacts/p
  Downloading https://index.taskcluster.net/v1/task/project.deepspeech.deepspeech.native_client.v0.6.0-alpha.8.cpu-ctc/artifacts/public/ds_ctcdecoder-0.6.0
     |████████████████████████████████| 1.7MB 317kB/s
Requirement already satisfied: numpy>=1.7.0 in /usr/local/lib/python3.6/dist-packages (from ds-ctcdecoder==0.6.0a8) (1.16.5)
Installing collected packages: ds-ctcdecoder
Successfully installed ds-ctcdecoder-0.6.0a8
```

```
!cd tmp; (cd DeepSpeech; git checkout tags/v0.4.1)
```

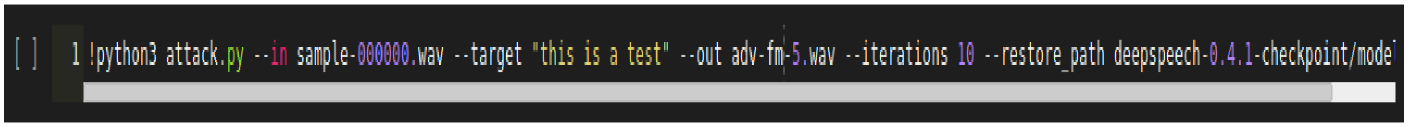
```
Note: checking out 'tags/v0.4.1'.

You are in 'detached HEAD' state. You can look around, make experimental
```

Figure. 7. Interface of Google Colab Jupyter Notebook (while running the code having selected the GPU)

Chapter 6

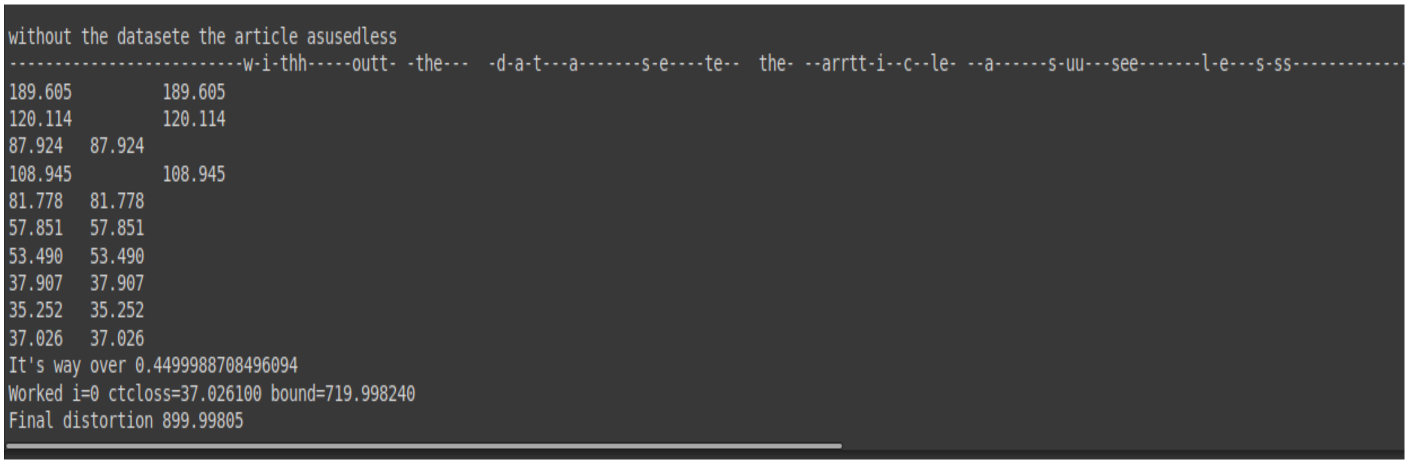
Experiments Performed

As we can see here, we have generated audio files corresponding to the target phrase, “this is a test”, we had gradually increased the number of iterations and noted down the results and plot the graph for the same. The distortion introduced in the original audio waveform is minimized in each step so that it resembles the same to the human ear. In Fig. 8. the command for executing the code for 10 iterations is shown, it takes as the argument the target phrase, number of iterations, the original audio sample and the pre-trained model for Deepspeech taken from Mozilla foundation. As the execution of the code proceeds, it first decodes what is being spoken in the audio sample as can be seen in the Fig. 9. the first line reads "without the dataset the article is useless" and after that it reports the progress of the attack in terms of ctc-loss at each iteration. Similar to Fig. 8. in Fig. 10. only the number of iterations are changed for the experimental process. Upon increasing the number of iterations the distortion is minimized to a greater extent. Similar to Fig. 9. the Fig. 11. reports the ctc-loss after each iteration and the final distortion after a total of 1000 iterations as performed above.



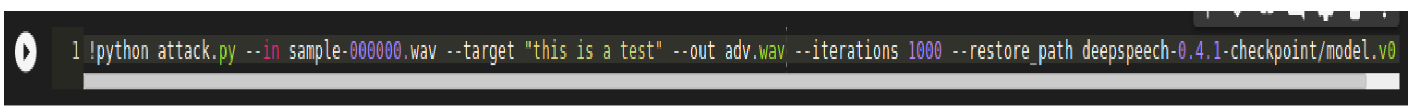
```
[ ]  !python3 attack.py --in sample-000000.wav --target "this is a test" --out adv-fm-5.wav --iterations 10 --restore_path deepspeech-0.4.1-checkpoint/mode
```

Figure. 8. Input command for 10 iterations



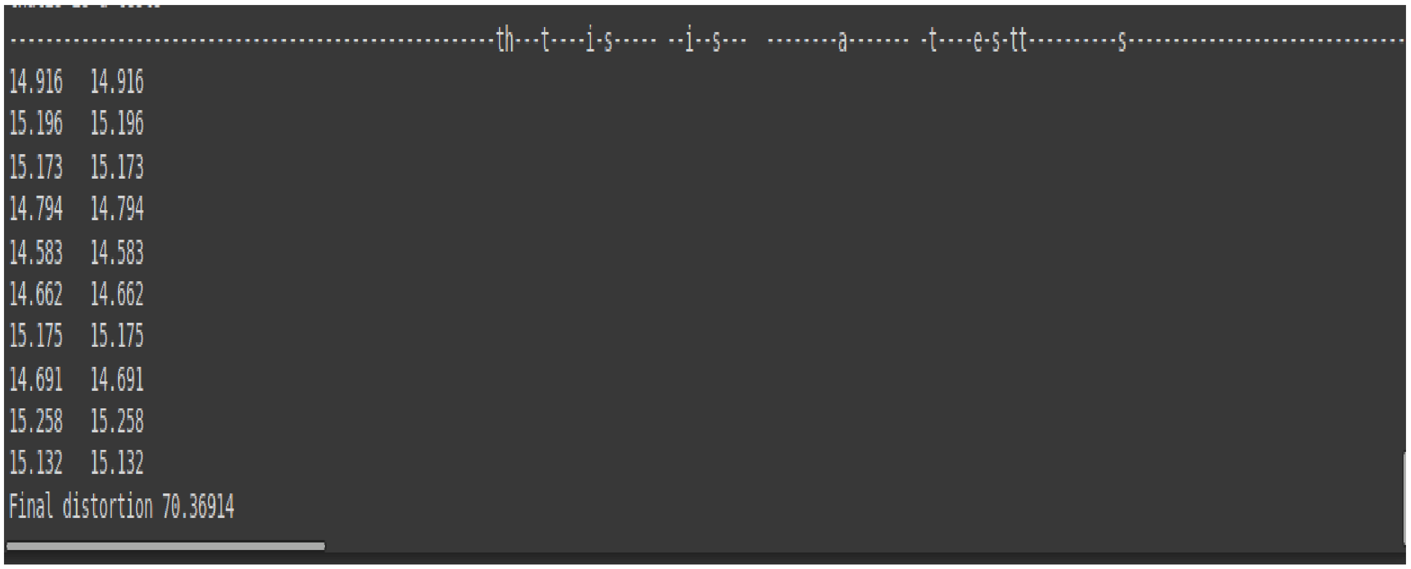
```
without the datasete the article asusedless
-------------------------w-i-thh-----outt- -the---  -d-a-t---a-------s-e----te--  the- --arrtt-i--c--le- --a------s-uu---see-------l-e---s-ss------------
189.605         189.605
120.114         120.114
87.924   87.924
108.945         108.945
81.778   81.778
57.851   57.851
53.490   53.490
37.907   37.907
35.252   35.252
37.026   37.026
It's way over 0.4499988708496094
Worked i=0 ctcloss=37.026100 bound=719.998240
Final distortion 899.99805
```

Figure. 9. Output after 10 iterations



```
!python attack.py --in sample-000000.wav --target "this is a test" --out adv.wav --iterations 1000 --restore_path deepspeech-0.4.1-checkpoint/model.v0
```

Figure. 10. Input command for 1000

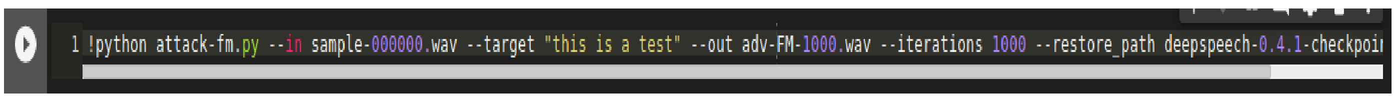


```
--------------------------------------------------th---t----i-s----- --i-s---  -------a------ -t----e-s-tt---------s------------------------------
14.916   14.916
15.196   15.196
15.173   15.173
14.794   14.794
14.583   14.583
14.662   14.662
15.175   15.175
14.691   14.691
15.258   15.258
15.132   15.132
Final distortion 70.36914
```

Figure. 11. Output after 1000 iterations

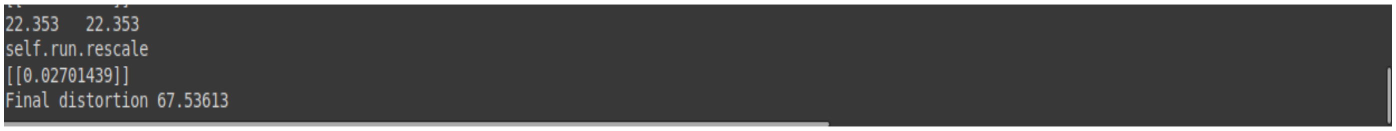
6.1 For 1000 iteration (with Psychoacoustic)

Through the discussion with our guide, we reached a conclusion that with the help of frequency masking concept of psychoacoustics, we could make our noise imperceptible. Hence, we calculated the threshold values accordingly and rescaled the distortion with the same. (Corresponding code for masking, the audio files obtained and the attack can be found on our Github repository, *https://github.com/vishKurama/audio-adversarial-attack*). In Fig. 12. everything is same as in Fig. 10. except that the mechanism of the attack has the principle of psychoacoustics applied and the corresponding output in Fig. 13. can be seen as the final distortion is relatively less than that was originally present however small.



```
!python attack-fm.py --in sample-000000.wav --target "this is a test" --out adv-FM-1000.wav --iterations 1000 --restore_path deepspeech-0.4.1-checkpoi
```

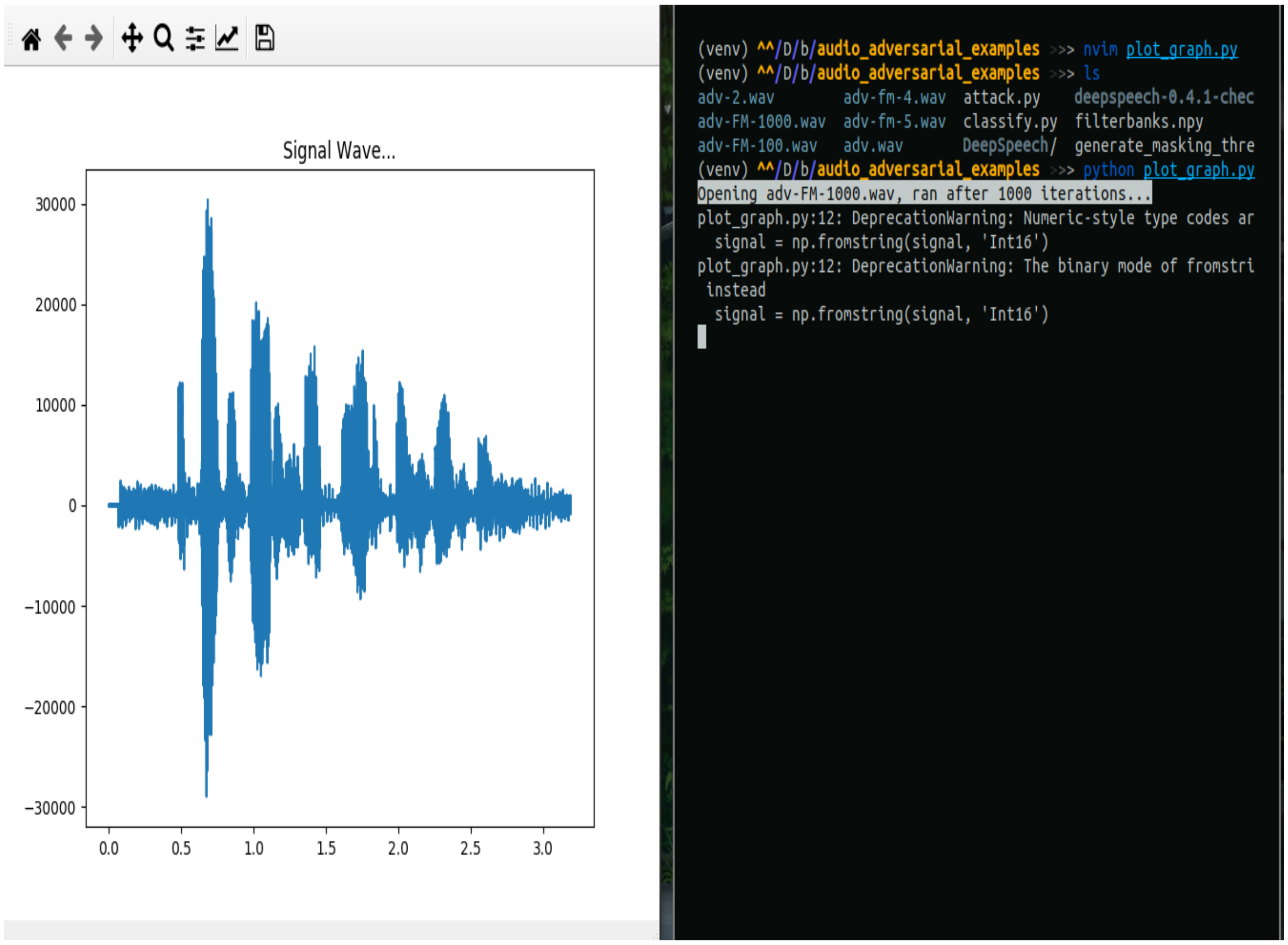
Figure. 12. Input command for 1000 iterations (Psychoacoustic).



```
22.353   22.353
self.run.rescale
[[0.02701439]]
Final distortion 67.53613
```

Figure. 13. Output after 1000 iterations (Psychoacoustic).

We are using the matplotlib library of python to plot the graph of obtained audio waveforms after the attack to get more visualization of the distortion added, as shown in the Fig. 14. the function takes input the audio waveform and produces a graph of it accordingly with amplitude on y-axis, and time on the x-axis.



```
(venv) ^^/D/b/audio_adversarial_examples >>> nvim plot_graph.py
(venv) ^^/D/b/audio_adversarial_examples >>> ls
adv-2.wav       adv-fm-4.wav  attack.py    deepspeech-0.4.1-chec
adv-FM-1000.wav adv-fm-5.wav  classify.py  filterbanks.npy
adv-FM-100.wav  adv.wav       DeepSpeech/  generate_masking_thre
(venv) ^^/D/b/audio_adversarial_examples >>> python plot_graph.py
Opening adv-FM-1000.wav, ran after 1000 iterations...
plot_graph.py:12: DeprecationWarning: Numeric-style type codes ar
  signal = np.fromstring(signal, 'Int16')
plot_graph.py:12: DeprecationWarning: The binary mode of fromstri
 instead
  signal = np.fromstring(signal, 'Int16')
```

Figure. 14. Input and output after plotting graphs

Chapter 7

Results Obtained

In *Table. 1.* ***C(x)*** is the output function of speech to text system which takes input as the audio sample and produces text as output before the application of psychoacoustics. As the number of iterations are increased from 1 to 1000 ***C(x)*** produces output which is similar to the target phrase. For example the output ***C(x)*** is "that a r tas sisc" after 10th iteration whereas the output ***C(x)*** is "this is a test" after 1000 iteration which is equal to the target phrase.

No. of Iterations	Original Input (.wav file)	Target Phrase	C(x) (Output)
1	Without the dataset the article is useless	This is a test	Without the dataset the article is useless
10	Without the dataset the article is useless	This is a test	that a r tas sisc
100	Without the dataset the article is useless	This is a test	this is a test
1000	Without the dataset the article is useless	This is a test	this is a test

Table. 1. Comparison between output when input and target phrase is same and the number of iterations is increased.

7.1 What the graph signifies

It can be observed that in Fig. 15. when the number of iterations is relatively less, the noise added has significantly altered the waveform of the audio. If the waveform is distorted more than a particular point, we would have obtained entirely new audio as opposed to the attack we wanted. From ***x = [2.5,3]*** denser graph is shown in attacked audio which does not resembles the graph of original audio. In Fig. 16. number of iterations are increased to 1000. It can also be seen that both original graph and attacked graph of the obtained waveform closely resembles to each other which helps in producing better results. From ***x = [0,3]*** graph of attacked audio and original audio are almost identical. In Fig. 17. the graphs of the original audio and attacked audio after 100 iterations are compared after applying psychoacoustics where it can be observed that noise has been added after all high amplitudes (According to psychoacoustic model). A denser graph can be observed at ***x = 1.5***.

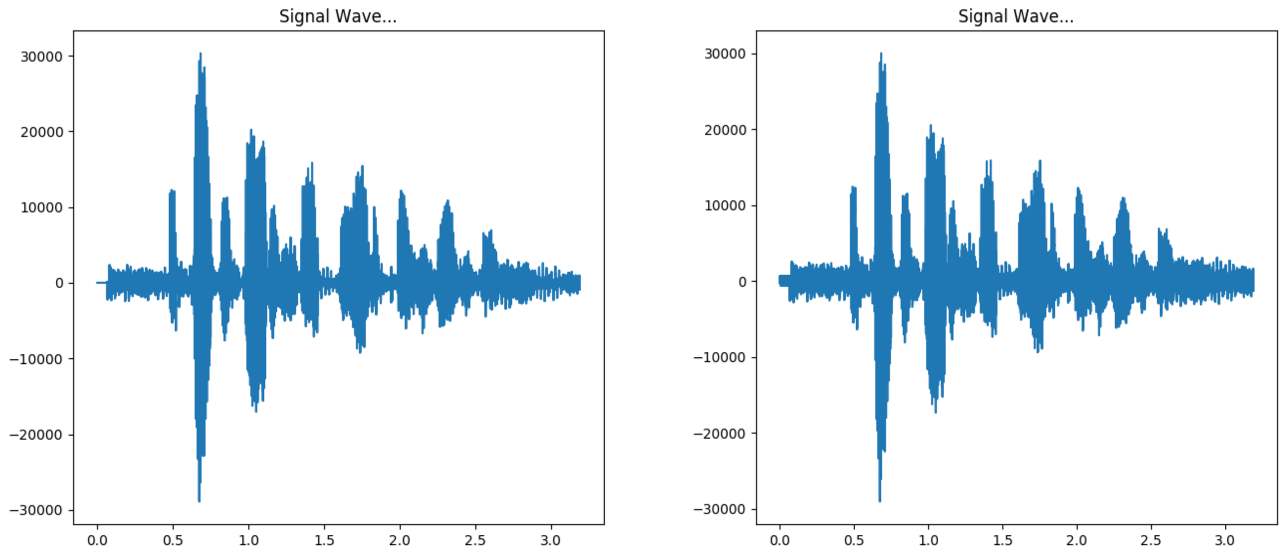


Figure. 15. Waveforms of original (left) and after 100 iterations of attack (right) on the audio sample. (Before the psychoacoustics).

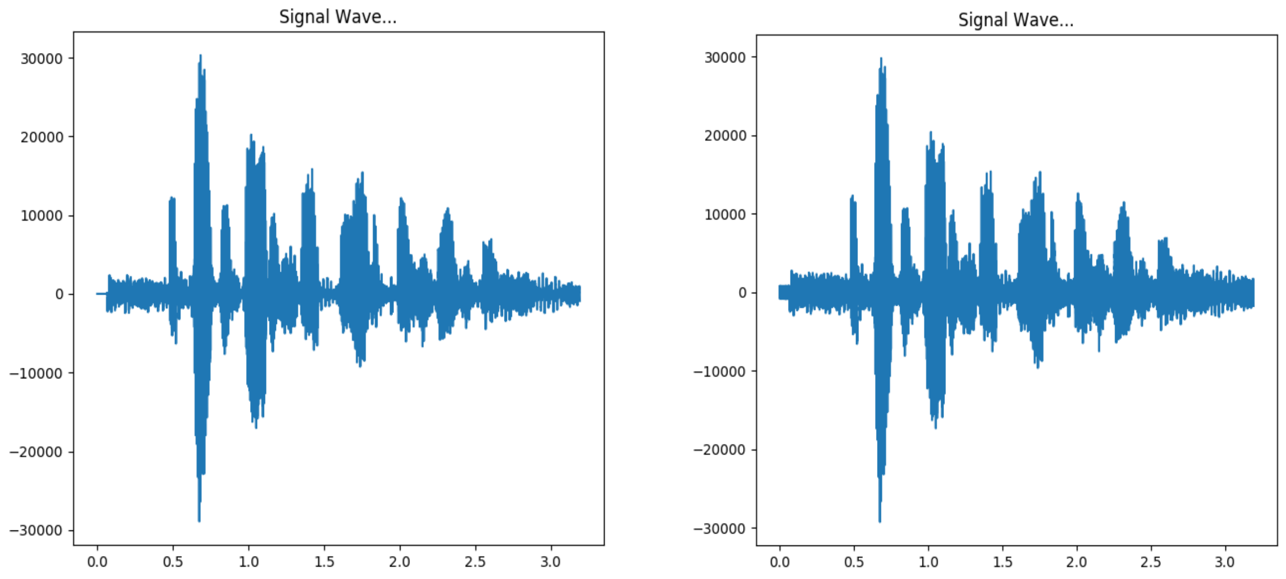
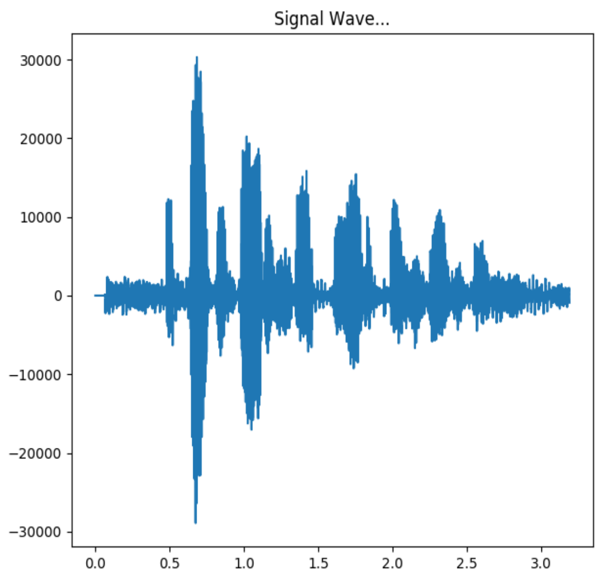


Figure. 16. Comparison between original (left) and 1000 iterations after the attack (right) on the audio (Before the psychoacoustics frequency masking).



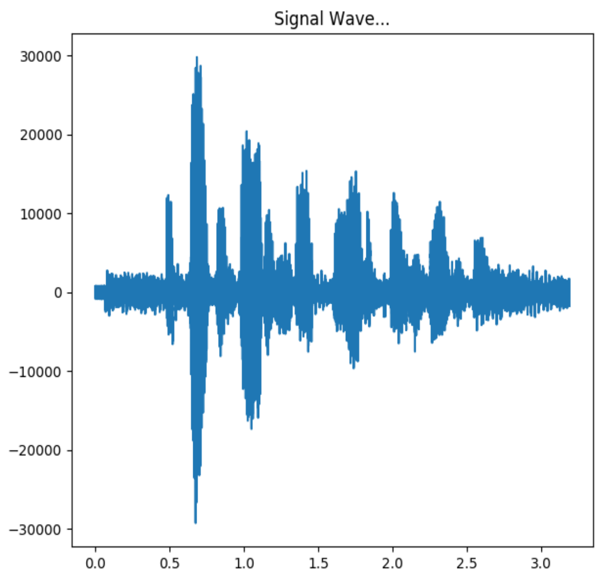


Figure. 17. Waveforms of original (left) and after 100 iterations (right) of attack (Psychoacoustics).

Conclusion

We have shown that the targeted audio adversarial examples works perfectly on the automatic speech recognition (ASR) systems. We can easily convert any audio waveform to any target phrase by adding a very minimal amount of noise which are in the end an optimization problem. In order to make the attack imperceptible to human ears, we had used the properties of the psychoacoustic model, which basically hides the noise near the louder tones in the input audio. We henceforth have successfully constructed imperceptible adversarial examples for few chosen samples by utilizing psychoacoustic principle of frequency masking.

References

1. Carlini, N. and Wagner, D. A. Audio adversarial examples: Targeted attacks on speech-to-text. IEEE Security and Privacy Workshops (SPW), 2018.
2. Yao Qin, Nicholas Carlini, Garrison Cottrell, Ian Goodfellow, Colin Raffel. Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. International Conference on Machine Learning (ICML), 2019.
3. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. International Conference on Learning Representations (ICLR), 2013.
4. D. Kingma and J. Ba. Adam. A method for stochastic optimization. International Conference for Learning Representations (ICLR), 2015.
5. A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deepspeech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567, 2014.
6. Mozilla. Project Deepspeech. https://github.com/mozilla/DeepSpeech, 2017
7. A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. International Conference on Machine Learning (ICML), 2006.