**SPEECH TO IMAGE**

Table of Contents

[1. Project use-case and overview 2](#_Toc51958997)

[2. Architecture of the application 2](#_Toc51958998)

[3. Speech – Audio signals 2](#_Toc51958999)

[a. Properties of audio signals 2](#_Toc51959000)

[b. Information in audio files(.wav/.mp3) 3](#_Toc51959001)

[c. Time domain and Frequency domain 4](#_Toc51959002)

[d. Feature Extraction – Audio signals 6](#_Toc51959003)

[4.Speech Recognition Pre-trained models 10](#_Toc51959004)

[a. Nvidia Quartznet model 10](#_Toc51959005)

[b. Deep Speech 11](#_Toc51959006)

[c. Speech recognition API 11](#_Toc51959007)

[d. Seq2Seq model –Build model using MELD dataset 11](#_Toc51959008)

[5.Text-to-image generation 12](#_Toc51959009)

[a.GANs Overview : 12](#_Toc51959010)

[b.Attention GAN 14](#_Toc51959011)

[c.Object-Pathway GAN-SOA metrics 16](#_Toc51959012)

[d.Object GAN 18](#_Toc51959013)

[6.User interface API and deployment 19](#_Toc51959014)

[a. Streamlit API 19](#_Toc51959015)

[i. Deploy streamlit app in Heroku 19](#_Toc51959016)

[ii. Deploy streamlit app in Docker 20](#_Toc51959017)

[b. Flask API 20](#_Toc51959018)

[i. Deploy flask app in Heroku 21](#_Toc51959019)

[ii. Deploy flask app in Docker 21](#_Toc51959020)

[7. ISSUES FACED while deployment and SOLUTION 22](#_Toc51959021)

[8.User Interface architecture for SpeechToImage using Flask API - HTML , CSS , JS 26](#_Toc51959022)

[9.Widely used commands 27](#_Toc51959023)

[b. Git commands 27](#_Toc51959024)

[c. Docker commands 28](#_Toc51959025)

[d. Heroku commands 29](#_Toc51959026)

[e. LINUX Commands 30](#_Toc51959027)

[10. Ongoing project 31](#_Toc51959028)

[11.References 31](#_Toc51959029)

[12.Projects in Github 31](#_Toc51959030)

# 1. Project use-case and overview

The main purpose of the project is to Visualize anything we say. This can be used to communicate with people having hearing impairment , or, in case of planning house constructions, where requirements like "Design a room with x-y-z dimensions , yellow walls, TV placed at middle of the wall.." can be told by the customer and the app shows an image with the given requirements.

# 2. Architecture of the application

Capture voice from Microphone

Extract audio features

Pre-trained Speech recognition models

AUDIO TRANSCRIPT

Generative Adversial Network (GAN)

Generated Image

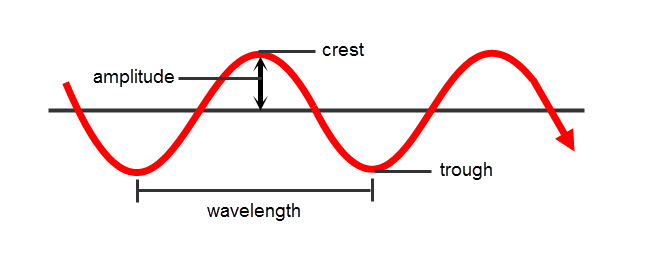
# 3. Speech – Audio signals

Speech Recognition aims at automatically transcribing spoken language (speech-to-text).

Let us understand about sound waves. Air molecules are present everywhere. When we speak, these molecule vibrate / change position/transmit energy to neighboring molecules and cause other air molecules around to change the position. This vibration (disturbance of the medium) travels in the atmosphere and is called a sound wave or audio signal

1. Properties of audio signals
   * **Amplitude(A)** – Maximum displacement of air molecules from equilibrium(rest) position
     + High amplitude means high energy
     + **Amplitude is directly related Loudness**

* **Wavelength (**λ**) –** Distance between two successive crests or troughs



* + **Time period(T)** – Time taken(s) by air molecule to complete one cycle
    - Number of seconds per oscillation
    - **Time period is inversely related to the Pitch** i.e. As the time taken by the molecule to cover one cycle reduces, we do not hear the words clearly or say we hear the noise (buzz sound). This is referred as high pitch
  + **Frequency(f)** – Number of oscillations per second
    - **Frequency is directly related to Pitch**
    - If someone speaks very much faster, it causes faster movement of molecules which results in more waves per second, thus, higher frequency. Due to faster movements of molecules, the voice will not be clear and appears as noise(buzz sound). This is called high pitch.
    - **Frequency is inversely related to Time Period** (T), f= 1/T
    - Eg : In music, note A has frequency = 440Hz i.e in 1 second, air molecules make 440 oscillations and hence it is referred as High Frequency / High pitch . Similarly note C is referred as Low frequency or Low pitch sound

**NOTE**:

Speed of sound = distance travelled / Time period

**Speed = wavelength \* Frequency**

As per the formula, increasing f, looks like increasing the speed.

**Actually , as F increase, number of oscillations per second increases, wavelength decrease, thus speed of sound remains same**

To change the speed of sound , we need to change the medium from air to water or change the temperature of the air

## Information in audio files(.wav/.mp3)

Audio files will have the following information –

* + duration of the audio - Eg : 1.2 minutes
  + sampling rate – Eg : 22KHz
  + sample values i.e amplitude values

**Sampling rate**

* Sampling rate is the frequency at which the audio signals
* Sampling rate = 44100 means, 44100 samples are captured in a second.
* Sampling rate depends on the file format

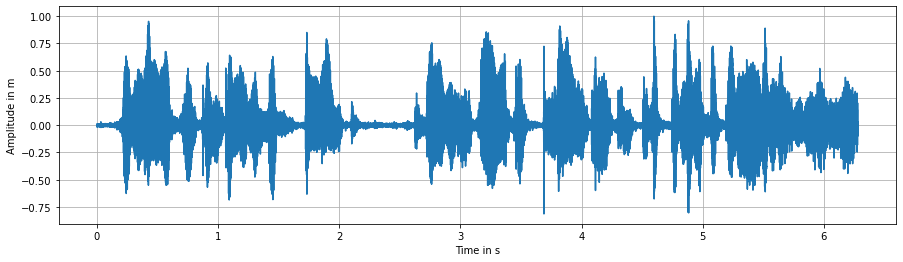
Reference : <https://medium.com/game-of-bits/audio-analysis-part-1-sound-waves-things-you-must-know-1e10851cc109>

1. Time domain and Frequency domainanalysis

Audio signals are time series(periodic) signals. These periodic signals can be examined from 2 domain – Time domain and Frequency domain

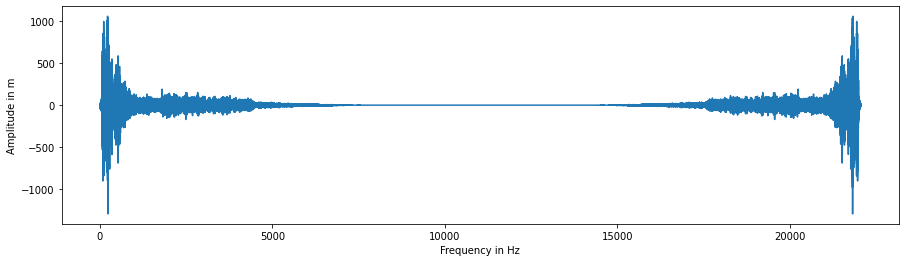
* **Time Domain** - Amplitude(y-axis) as a function of Time(x-axis)
  + Feature measured – How loud(amplitude) is the sound in one second time period
  + Considers sequential order of the signal

Eg: Audio file of duration 6.28 seconds



* **Frequency domain (a.k.a Spectrum)** – Amplitude(y-axis) as a function of frequency( x-axis)
  + Feature measured – How strong(amplitude) the signal is at all the frequencies
  + Ignores sequential order of the signal

Spectrum of the audio signal , obtained by applying Fourier Transform



Inference :

Peaks in Spectrum convey Dominant Frequency components(a.k.a FORMANTS) in the Speech signal (Time domain).

Every sound has its own distinctive Formants in the Freq domain

Eg : ‘ih’ in the word “bit” has formants(Peaks) at 400Hz, 1900Hz, 2500Hz

In a Frequency domain , if we have peaks at these given frequencies, we can say that the sound is ‘ih’

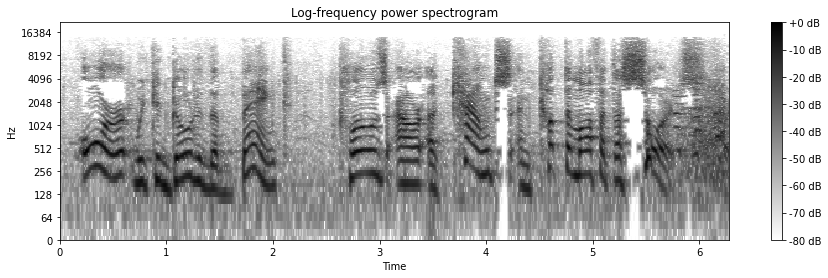
* Peak at 5Hz, indicates there are 5 cycles per second in the Time domain
* Here, we see peaks at lower and higher frequencies.
  + Peak in lower frequency region indicates slow varying signal in Time domain
  + Peak in higher frequency region indicates fluctuation signal in Time domain

To convert signals from Time domain to Frequency domain – Apply Fourier Transform

To convert signals from Frequency domain to Time domain – Apply Inverse Fourier Transform

1. **Spectrogram**

* Time – Frequency representation of Speech signal
* Spectrogram – Sequence of spectral vectors represented using grey level values in Time domain.
* Spectrogram is the representation of the energy levels (i.e. amplitude, or "loudness") of each frequency of the signal over the duration of the audio
* Spectrogram is obtained by breaking up the signal into smaller, usually overlapping chunks and performing Fourier transform on each chunk.



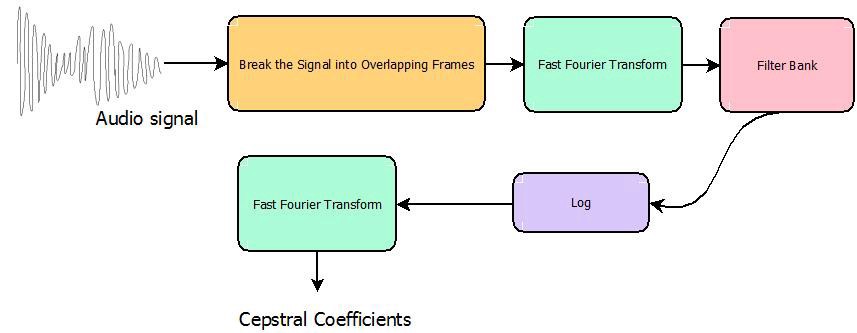
**Inference**:

Dark grey(black) indicates Peak(Formant) i.e voice signal,

White indicates NonPeak region i.e non-voiced or silence , which means amplitude(Energy) of the signal is almost zero

* We can see white region between 3.1s and 3.2s, this indicates that amplitude at that region is almost zero, which can be verified against Time-domain graph
* In the time-frame 0.5s, we can see dark region/peak at 256Hz i.e 256 cycles per second which indicates that we have voiced data , which can be verified against Time-domain graph

## Feature Extraction – Audio signals



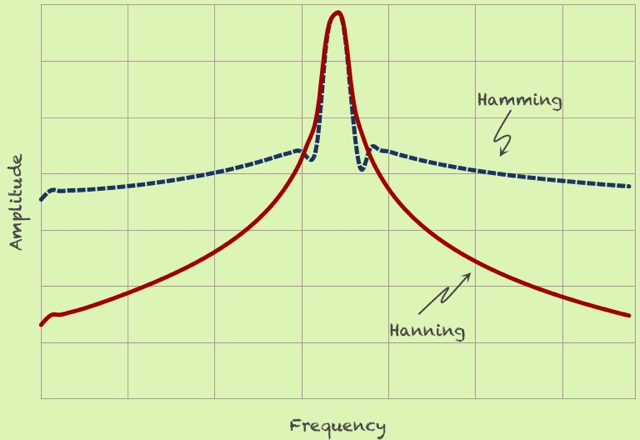
1. Audio signal is broken into smaller overlapping chunks , with **window\_duration = 20ms and stride\_duration(overlap) = 10ms**.
   * Optimum **window duration is 20-30ms**, as humans cannot utter more than one phoneme in this span. For Speech-recognition, we consider window overlap(stride) as 50%
   * Number of samples in each window is calculated using the formula (similar to applying filters in CNN) , **window\_size = int(0.001 \* sample\_rate \* window\_ms)**
   * Number of samples in the overlap region is calculated using the formula, (similar to applying stride on image CNN) **stride\_size = int(0.001 \* sample\_rate \* stride\_ms)**
   * Total number of windows or time-frames in the entire audio is calculated using

**total\_windows\_in\_audio = int((len(samples) - window\_size) / stride\_size) + 1**

* + The amount by which the filter shifts is the **stride**. In case of audio signals, we shift by certain number of bytes. This is calculated using in-build method **stride\_tricks.as\_strided**

1. **To get the Spectrum (Frequency domain analysis) , we apply Fourier transform (FFT) onto Overlapping Frames**
   * Before applying FFT, we apply windowing function like “Hamming/Hanning/Kaiser” to enhance the ability of FFT to extract spectral data from signals(amplitude accuracy) , reduce the noise(side lobes) and eliminate discontinuity in the wave. Here, we apply hanning window function on all the frames

**weighting\_on\_our\_window = windows \* hanWeight**

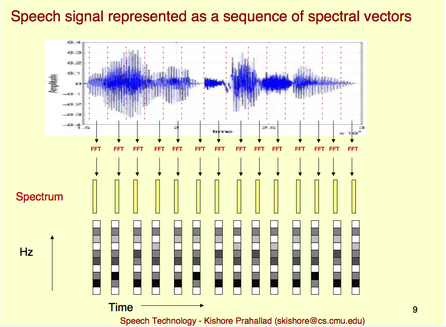
Hamming and Hanning window FFT of the wave formed by windowing

* + We apply FFT on the window frames (obtained from previous step), which gives the spectrum

Spectrum = Sequence of Spectral vectors across the windows

* + Spectral vectors can be given to the Neural network for Speech Recognition task at this point
  + Spectral amplitude values can be represented as grey values, which gives us Spectrogram

TASKS PERFORMED CAN BE EXPLAINED HERE



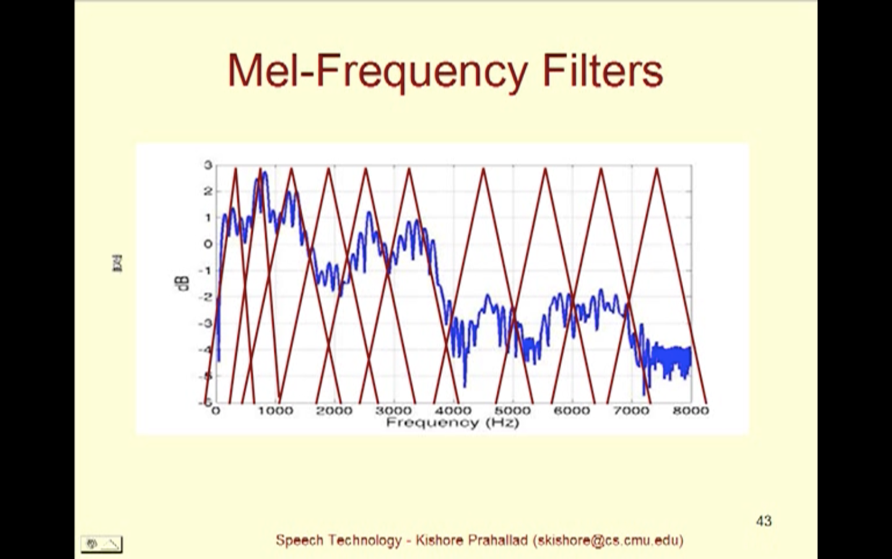
1. **Apply Filter bank and take log on mel-spectrum**

Human ears concentrate on certain frequencies, rather than the entire spectrum. Hence, we apply filters(non-uniformly distributed) on the spectrum. There will be more filters in the low-frequency regions, and less number of filters in the high-frequency regions. n\_mels specify the number of filters we use on the spectrum. Applying the mel-filters on the spectrum gives the Mel-Spectrum. Take log on the mel-spectrum

**Mel Spectrum**

1. Mel Spectrum is a combination of Spectral Envelope and Spectral details log X[k] = log H[k] + log E[k] , where log
2. Spectral Envelope is the smooth curve connecting the dominant(significant) frequency component(FORMANT)
3. Spectral details is the remaining noise signal

Spectrum -> Mel-Filters -> Mel-Spectrum



1. **Cepstral Analysis**
2. Aim of cepstral analysis is to separate spectral envelope and spectral details
3. We have log mel-spectrum. Applying IFFT (Cepstral analysis) on the log mel-spectrum gives Cepstrum. In our case, it is **Mel frequency Cepstrum**

* log X[k] = log H[k] + log E[k]
* IFFT on log X[k] gives x[k] = h[k] + e[k], where , x[k] is the Mel-frequency Cepstrum

1. Mel-frequency Cepstrum provides signals in the high and low frequency regions. The low-frequency region of Cepstrum provides Cepstral coefficients (of spectral envelope)
2. Cepstral coefficients h[k] obtained for Mel- spectrum are referred to as Mel-Frequency Cepstral Coefficients often denoted by **MFCC**

**There are around 20 MFCCs, each of which represents certain audio feature**

* **1st coefficient will provide info on average power in the spectrum**
* **2nd coefficient will provide info on spectral centroid**
* **8-13th coefficients represent spectral envelope/ shape of the signal**
* **19th coefficients represents pitch or spectral details**

**These Cepstral coefficients are provided as input to Speech recognition model**

Once we get the 10 MFCCs coefficients(vectors), these vectors are grouped and matched to one or more [phonemes](https://en.wikipedia.org/wiki/Phoneme)—a fundamental unit of speech

Example of phonemes -

Letter B is a combination of 2 phonemes – ‘bha’ , ‘ee’

Finally, a special algorithm is then applied to determine the most likely word (or words) that produce the given sequence of phonemes.

Grouping the vectors to match the phonemes involves lots of training, as it varies from one speaker to other, and even varies from one utterance to another by the same speaker.

**Reference :**

<https://archive.org/details/SpectrogramCepstrumAndMel-frequency_636522>

**Github repo link**

<https://github.com/savitha91/AudioSignal_Analysis>

# 4.Speech Recognition Pre-trained models

Speech recognition system to work properly for anyone, we need lots of samples.

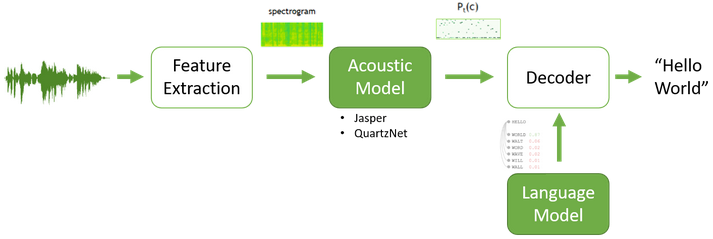
There are several models training on different datasets like -

1. NVIDIA model
2. Deep Speech
3. Speech recognition API
4. Seq2Seq model

## Nvidia Quartznet model

This model was trained on 6 datasets - LibriSpeech, Mozilla Common Voice , WSJ, Fisher, Switchboard, and NSC Singapore English

WER of 3.79% on LibriSpeech dev-clean, and a WER of 10.05% on dev-other



Language Model : CTC model

Decoder output is dependent on Softmax + Language model output

Language model used : prefix beam search KenLM  and Transformer-XL

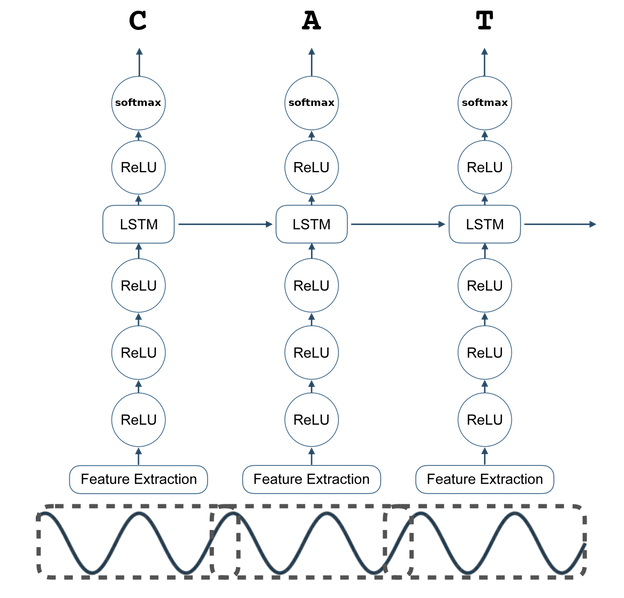
**Evaluation metric**

Word Error rate (WER) = (Substitutions + Insertions + Deletions) / Number of Words Spoken

Nvidia models take .wav file as input. Video files(.mp4) need to be converted to .wav format

## Deep Speech

Pre-trained models and checkpoints are available



Evaluation metrics –

Connectionist Temporal Classification (CTC)

## Speech recognition API

This API provides function to

* record audio from microphone
* Save the recorded audio to .wav file and
* Recognise the audio i.e speech to text recognition

We use Speech recognition API for our application

## Seq2Seq model –Build model using MELD dataset

**Dataset : MELD training dataset**

Seq2Seq models are used for Machine translations. Our model is similar to this.

Input to Encoder : Mel Frequency components obtained from the audio signals

Input to Decoder : Utterances encoded using BERT tokenizer

ENCODER

DECODER

MFCCS obtained from audio analysis

Utterances encoded using BERT

Predicted Transcript

# 5.Text-to-image generation

## a.GANs Overview :

**Generative adversarial networks are used to generate images**

**Working of GAN :**

1. Real image passed to discriminator

REAL IMAGE

(Eg : image names fetched from filenames.pickle)

**DISCRIMINATOR**

Loss back-propogated to Discriminator model

y =1 (indicating real image)

yhat calculated

1. Random noise is passed to the Generator model, which generates fake images

RANDOM NOISE

**GENERATOR**

Generate Fake images

1. Real image and Generated images are both passed to the Discriminator model, with expected output set to 1 , i.e no difference between the images. The loss is then back-progogated to the Generator model, using which , Generator model is supposed to generate Fake image almost close to the real image

Fake image

Noise

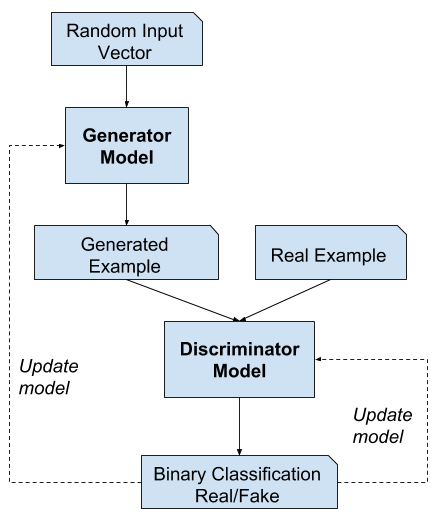
Generatorr

Discriminator

Expected y = 1

Real image

E**ND TO END FLOW**

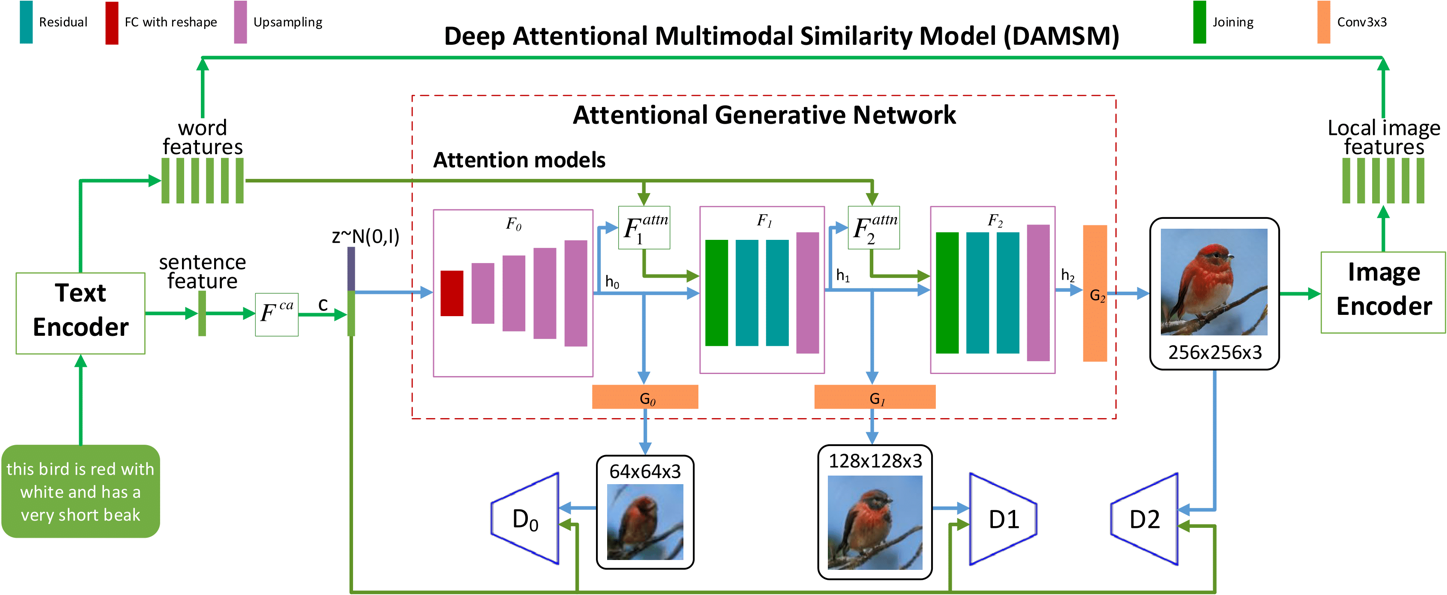


GANs that can be used for our project

* Attention GAN
* Object GAN

## b.Attention GAN

**PAPER**: <https://openaccess.thecvf.com/content_cvpr_2018/papers/Xu_AttnGAN_Fine-Grained_Text_CVPR_2018_paper.pdf>



“AttnGAN1” architecture has one attention model and two generators,

“AttnGAN2” architecture has two attention models stacked with three generators.

**“IMAGE GENERATION CONDITIONED ON WORDS”**

This model has

1. 3 GENERATORS (G0, G1 and G2) , which take 3 Image features / hidden states (h0,h1,h2).
2. 3 Discriminators (D0,D1 and D2)
3. z is the “Noise vector”
4. NN models
   1. Fca – Conditioning Augmentation model, which converts the sentence vector to conditioning vector
   2. Fi -  - Takes noise and Condition augmented sentence vector as input and generates hidden layer (h0), which represents image vector(features)
   3. FAttn  - Attention Model – Word features and Image features(from previous hidden layer) are the inputs. Word features are first converted into semantic space of the image features . For each sub-region of image based on h (image feature), word-context vector is created.
   4. Gi – Generator model

Generator/ Discriminator Loss = Combination of Uncondition Loss and Conditional Loss

where, Unconditional loss determines whether the image is real or fake while the conditional loss determines whether the image and the sentence match or not.

**DAMSM** : Deep Attentional Multimodal Similarity Model - computes similarity between image and sentence (Image-text matching)

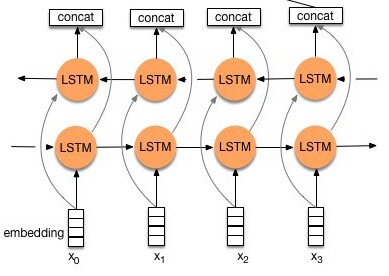
DAMSM has 2 Neural Networks

* Text Encoder : bi-directional LSTM , extracts semantic vectors from input sentence.

“Show red color bird”

**OUTPUT = WORD VECTOR**

**Sentence Vector**



Show Red color bird

* **Image Encoder** : CNN – High resolution generated image is the input. Model is Inception – v3 model.
  1. Rescale image to 299 \* 299
  2. Extract local features from sub-regions of image. Each image is split into 289 sub regions. Overall 768 local features are extracted.
  3. From last avg pooling layer, 2048 global features are extracted .
  4. Add new perceptron layer as last layer, which converts image features to text features
* **Attention-driven image-text matching score**: Measures the matching of an image-sentence pair based on Attention model

Represent image sub region related to the word

**DAMSM loss :** Given Entire image and full sentence , P(sentences(D) | Images(Q))

Aim is to reduce the Loss

L1word = - Summation(log P(D | Q) )

L2word = - Summation(log P(Q |D) )

{(Qi, , Di)} – Image sentence pair

* Image Encoder : CNN - Built upon Inception – v3 model. It maps images to semantic vectors .

Attn score measures the matching of image and text

DAMSM loss is calculate for last image. DAMSM loss is calculated as word and sentence loss

**Pre-Trained models -**

1. **DAMSM Text Encoder - Bi-directional LSTM**
2. **DAMSM Image Encoder**
3. **Attention model**

Note: DAMSM Image Encoder - Inception – v3 model is required only for training i.e to get better correlation between image and the text

**Evaluation** :

**Inception Score**: Evaluates how recognizable and diverse objects within images are. Higher the better. It doesn’t find link between image and text.

**Frechet Inception Distance (FID)** : compares the statistics of generated images with real images . Smaller the better

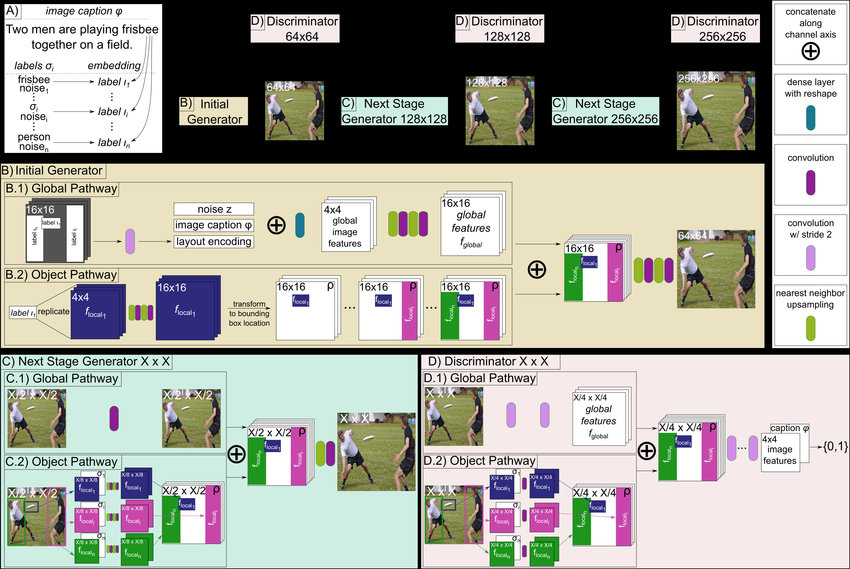
**Inception score and FID** evaluation scores are not designed to evaluate images that contain multiple objects and depict complex scenes.

**R-precision :** Image encoding and text encoding from DAMSM are used. For the generated image, captions are identified and verified against actual text. Often fail to evaluate more detailed aspects of an image, such as the quality of individual objects. Just compares the text encoding and image encoding

**Results**:

1. AttnGAN model gives better results on CUB(Birds dataset) compared to COCO dtaatset
2. Inception Score using AttnGAN is way better than StackGAN and other previous GAN models

## c.Object-Pathway GAN-SOA metrics



**Generator input :** Scene description (Image caption embedding , from pretrained CNN-RNN) + one hot encoded label for each bounding box (lonehot) + location and size of object bound box+ Random noise(z)

Label for individual bounding box (label) obtained from image caption + one-hot encoding

This label has label details + additional info like color, shape(bounding box)

**Global Pathway** takes Generator input(label for each bounding box) , represents spatially at the bounding box location and creates Scene layout encoding (High level features. Convolution is applied to this Scene layout to obtain high level layout encoding.

Layout encoding + Noise (Z) + Image caption embedding = General image layout fglobal

**Object Pathway** : The object pathway generates a feature of the object (flocal) at a location described by the respective bounding box and is applied iteratively over the scene at the locations specified by the individual bounding boxes. This pathway creates feature map, which is transformed with Spatial Transformer Network(STN) to fit objects into bounding boxes at given location on empty canvas. Convolution is applied to each label.

flocal + fglobal along channel axis to generate image by applying UpSampling(Conv2DTranspose) with leakyRelu (Standard GAN procedure)

**Discriminator** input : the image(real or fake), the bounding boxes location , size and their respective object labels (as one hot encoding), and the image caption embedding

Global pathway takes image as input, applies Conv and gets the global feature representation(fglobal) of the entire image

Object pathway uses Spatial Transformer Network (STN) to extract objects from given bounding boxes

The discriminator judges not only if the image is realistic and aligned to the natural language description, but also whether the specified objects are at the given locations and of the correct object category.

**Evaluation**

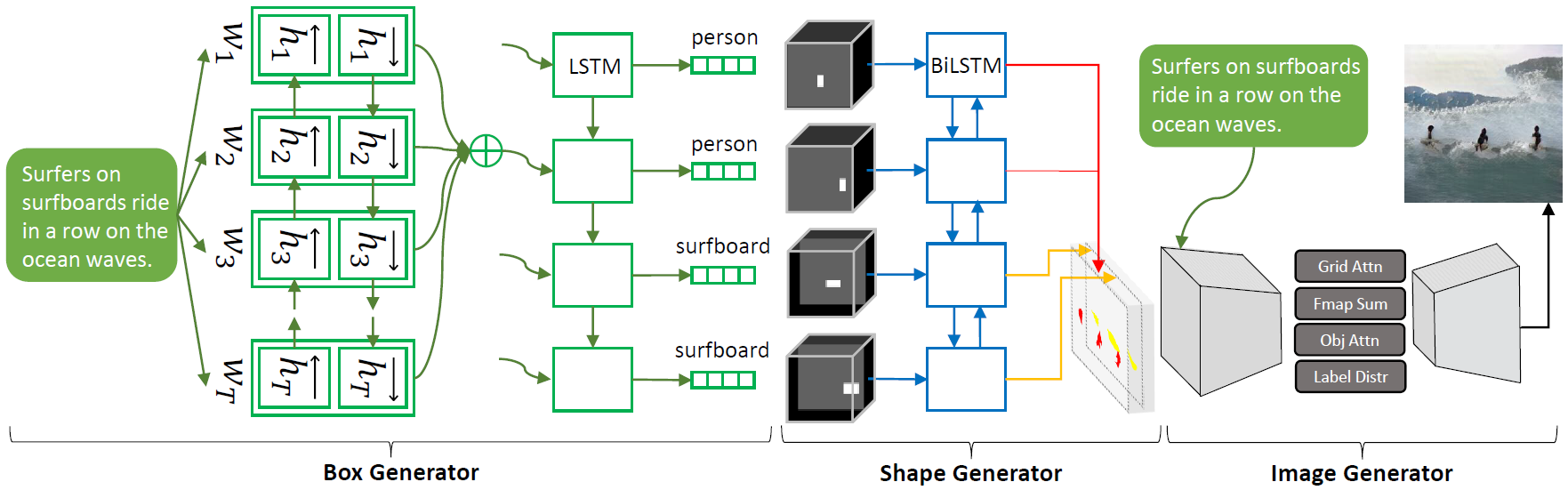
**SOA** focuses on individual objects and parts of an image and also takes the caption into consideration when evaluating an image.

All image captions from the COCO validation set that explicitly mention one of the 80 main object categories (e.e. “person”, “dog”, “car”, etc.) are samples and use them to generate images. We then use a pre-trained object detector and check whether it detects the explicitly mentioned objects within the generated images

Reference

<https://github.com/tohinz/semantic-object-accuracy-for-generative-text-to-image-synthesis>

## d.Object GAN



Object-driven attentive generative network and the Object-wise discriminator.

Input : text description and a pre-generated semantic layout

1. Bounding boxes are obtained for the text
2. Generator takes bounding box as input and generates image region within the bounding box, by focusing on the most relevant word
3. Object driven attention Generator (Obj-GAN) layer considers Class label of COCO to query the words in the sentence -> Word-context vector
4. Object wise discriminator : Checks if generated object matched the pre-generated semantic layout. All bounding box objects are evaluated simultaneously using Fast R-CNN with Binary-cross entropy loss for each bound box

Components

1. Attentive seq2seq bounding box generator,
2. attentive image generator
3. An object-wise discriminator.

Semantic layout : Class labels, bounding boxes, shapes of the objects

Reference : <https://github.com/jamesli1618/Obj-GAN>

# 6.User interface API and deployment

Front end for Machine learning apps can be developed using streamlit or Flask API . These apps can be deployed in Heroku/Docker/AWS,Azure

## Streamlit API

* Installation : pip install streamlit
* Default port : 8501
* Command to run streamlit app

streamlit run app\_streamlit.py

### Deploy streamlit app in Heroku

*Heroku is a container based cloud platform as a service(PaaS) ,used to* deploy, manage, and scale applications.

Mandatory files

* + - 1. **Procfile**

web: sh setup.sh && streamlit run app\_streamlit.py

* + - 1. **setup.sh**

mkdir -p ~/.streamlit/

echo "\

[general]\n\

email = \"your@domain.com\"\n\

" > ~/.streamlit/credentials.toml

echo "\

[server]\n\

headless = true\n\

enableCORS=false\n\

port = $PORT\n\

" > ~/.streamlit/config.toml

### Deploy streamlit app in Docker

**Dockerfile**

FROM baseimg

RUN mkdir -p /root/.streamlit  
RUN bash -c 'echo -e "\  
[general]*\n*\  
email = *\"\"\n*\  
" > /root/.streamlit/credentials.toml'  
RUN bash -c 'echo -e "\  
[server]*\n*\  
enableCORS = false*\n*\  
" > /root/.streamlit/config.toml'  
  
WORKDIR /app

EXPOSE 8501

COPY requirements.txt .  
RUN pip install –r requirements.txt  
COPY . .

CMD ["streamlit",”run”,"app\_streamlit.py"]

* + 1. Docker build and run command

docker build -f Dockerfile -t streamlitapp .

docker run –p 8001:8501 –ti streamlitapp

**Pros and Cons of using streamlit**

Pros: Streamlit is open source . Web-based frontend apps can be built with just few lines of code

Cons : Streamlit library is huge in size compared, hence deploying them in docker leads to large image size

## Flask API

Flask is a light weight python web framework . Application server framework for Flask is Gunicorn

* Required packages packages

Flask==1.1.2  
gunicorn==19.9.0  
itsdangerous==1.1.0  
Jinja2==2.11.2  
MarkupSafe==1.1.1  
Werkzeug==1.0.1

Flask-JSGlue==0.3.1

* Default port – 5000
* Command to run : python app\_flask.py
* Standard folder structure for Flask app

static

sample.css

sample.js

templates

sample.html

### Deploy flask app in Heroku

* + Mandatory files
    - * 1. **Procfile**

web:gunicorn && flask\_app:app

flask\_app is the python file. app is the Flask object initiated in the .py file , app =Flask(\_\_name\_\_)

### Deploy flask app in Docker

Dockerfile

FROM base\_image  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
COPY . .  
CMD ["python","flask\_app.py"]

* + - To make the server publicly available add host=0.0.0.0 in flask\_app.py file

if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=True,host='0.0.0.0')

* + - Docker build and run command

docker build -f Dockerfile -t flaskapp .

docker run –p 5002:5000 –ti flaskapp

# **7. ISSUES FACED while deployment and SOLUTION**

To record audio from microphone, speech\_recognition api requires PyAudio and portaudio packages

Code -

mic = speech\_recognition.Microphone()  
with mic as source:  
 audio = model.listen(source)

Recording using mic works fine in local , irrespective of OS. When deployed using docker, we have to map the audio to the docker image.

In case our local system OS is Ubuntu/Linux , **Advanced Linux Sound Architecture** (**ALSA**) provides API for sound card device drivers and is a part of Linux kernel. Native sound-devices are present in /dev/snd folder. Hence, we can map the folder(/dev/snd) from local to the docker image(container) as shown

docker run -it -v /dev/snd:/dev/snd speechrecimage /bin/bash

Linux is a kernel , a component used to build OS. Different Linux based operating systems -

1. Debian – 1st Linux distribution , mainly for experienced users.
2. Ubuntu – Forked from Debian, miainly for Linux begineers
3. Centos – Forked from Red Hat Enterprise Linux(RHEL), mainly for business purpose. More stable than Ubuntu

Command to install dependencies

1. Debian and Ubuntu : apt-install
2. Centos/RHEL : yum install

**Centos7 based docker container for speech\_recognition api can be found here**

<https://hub.docker.com/r/binkybong/speech-recognition/>

In case of MAC OS, we donot have the folder /dev/snd, as the audio system is Core audio. Though the recording of audio using speech\_recognition api works find in local, we cannot map the audio to docker image.

When we use Centos7 docker container in MacOS, and run the image using command(note that we have removed the mapping /dev/snd) -

docker run -it speechrecimage /bin/bash

we get ALSA error -

ALSA lib confmisc.c:768:(parse\_card) cannot find card '0'

ALSA lib conf.c:4259:(\_snd\_config\_evaluate) function snd\_func\_card\_driver returned error: No such file or directory

ALSA lib confmisc.c:392:(snd\_func\_concat) error evaluating strings

This is because, the Centos container runs of MAC kernel, which doesn’t support ALSA.

Please find the below approaches -

**Approach 1:**

1. Need to install kernel, kernel-headers

$ yum install kernel-devel kernel-headers

1. Install alsa-driver, alsa-utils, alsa-lib , alsa-firmware from source

<https://alsa.opensrc.org/Quick_Install>

1. Create a dummy sound card (Now /dev/snd folder gets created )

<https://www.alsa-project.org/main/index.php/Matrix:Module-dummy>

1. Install libraries required for pyaudio

yum install portaudio-devel libtool autoconf alsa-lib-devel pulseaudio-libs-devel make -y

**Pros & Cons –**

Installing a kernel in docker container cannot be considered as a solution, since, docker works on Containerization concept, wherein the kernel is the host machine OS kernel

**Approach 2 :**

1. Start a Virtual machine(Eg: Ubuntu AWS instance), which uses Linux kernel.
2. We need the kernel to be changed to generic-kernel. Follow the steps mentioned in the link -

<https://meetrix.io/blog/aws/changing-default-ubuntu-kernel.html>

1. Reboot the instance and install alsa, pyaudio libraries

sudo apt-get install alsa-base alsa-firmware-loaders alsa-oss alsa-source alsa-tools portaudio19-dev python3-all-dev

python3-pyaudio pulseaudio python3-pip pulseaudio-utils dbus-x11 -y

1. MAY NOT BE NEEDED - Install loop back card module – Mainly to record and play audio from the system using the command-

sudo modprobe snd-aloop

This adds /dev/snd folder with loop back sound devices

1. Install dummy sound cards

<https://www.alsa-project.org/main/index.php/Matrix:Module-dummy>

1. Alsa mixer setting

<http://howto.blbosti.com/2010/03/ubuntu-server-install-alsa-sound-and-moc-music-on-console/>

1. Record using arecord

arecord -f S16\_LE -d 10 -r 16000 --device="hw:1,0" /tmp/test-mic.wa

1. Play audio using

aplay /tmp/test-mic.wav

**Approach 3:**

1. **Transfer audio through ssh using pulse audio**

<https://medium.com/@cristianduguet/play-remote-audio-over-an-ssh-connection-with-a-mac-client-9b7135dfe129>

1. Command

docker run env PULSE\_SERVER=tcp:192.168.0.103:34568 -ti speechrec

pactl load-module module-native-protocol-tcp port=34567 auth-ip-acl=172.17.0.5

**Approach 4:**

1. Record audio in the browser using Javascript’s MediaRecorder API , which creates a blob URL

<https://medium.com/jeremy-gottfrieds-tech-blog/javascript-tutorial-record-audio-and-encode-it-to-mp3-2eedcd466e78>

* 1. Record audio in google colab

https://gist.github.com/korakot/c21c3476c024ad6d56d5f48b0bca92be

1. Send blob data from Javascript to Flask using Fetch API. Convert the blob data to .wav file and save to the project folder

Javascript code

const res = fetch("/processSpeech", {  
method: "post", *//posting data to the server*body: blob1  
})

In Flask app

@app.route(**'/processSpeech'**, methods=[**'POST'**])  
def process\_speech():  
 f = open(“WAV\_FILE\_PATH” **'wb'**)  
 f.write(request.data)  
 f.close()

1. The recorded audio i.e .wav file can then be processed to recognize audio and generate image

**Pros and Cons:**

Using this approach, we can deploy the app in Heroku and Docker without any configuration . As we donot use speech\_recognition API to record audio, there is not necessity to install pyaudio dependent libraries

Reference

1. Alsa overview

<http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch13lev1sec2.html>

<https://tldp.org/HOWTO/Alsa-sound.html#toc2>

1. MAC core –audio

<https://developer.apple.com/library/archive/documentation/MusicAudio/Conceptual/CoreAudioOverview/WhatisCoreAudio/WhatisCoreAudio.html>

1. Dummy card

<https://www.alsa-project.org/main/index.php/Matrix:Module-dummy>

# 8.User Interface architecture for SpeechToImage using Flask API - HTML , CSS , JS

(speechToImg. startRecording())

Record audio in browser using MediaRecorder API

(speechToImg. stopRecording())

Send audio blob to a function which invokes Fetch rec.exportWAV(sendData)

(speechToImg.sendBlob())

Send data to Flask url as a post with audio blob

fetch("/processSpeech", {  
method: "post", *//posting data to the server*body: blob1  
})

(AppFlask.process\_speech())

Write the audio blob to .wav file. Recognize audio using speech\_recognition api . Send the recognized text as response to JS

res = make\_response(jsonify(text))

(speechToImg.generateImg())

Check with user, whether the recognized text is same as what he recorded. If it matches, post the text to Flask for image generation

(speechToImg.generateImg())

Check with user, whether the recognized text is same as what he recorded. If it matches, post the text to Flask for image generation

(AppFlask.generate\_img())

Save recognized text to a file and generate image using Attention GAN pretrained model and send the success response to JS

(speechToImg.generateImg())

Show the generated image to the user

# 9.Widely used commands

## Git commands

1. Clone code from repo

*git clone* [*https://github.com/user/project.git*](https://github.com/user/project.git)

1. Push files to git master

*git add file*

*git commit -m "comment"*

*git remote add origin* [*https://github.com/user/project.git*](https://github.com/user/project.git)

*git push -u origin master*

1. Pull code from master

*git pull origin master*

1. Create a new branch

*git branch branchname*

*git fetch –-all*

*git checkout*

*This new branch will initially be replica of master branch*

1. Merge branch with master

*git checkout master*

*git merge branch*

1. Move files from master to new branch

*git checkout master*

*git checkout –b newbranch*

*git push origin newbranch*

1. Undo added file in local using

*git reset <filename>*

1. Remove files added (not yet pushed) by mistake

*git rm –cache /path/to/file*

*git commit –am “Remove file”*

*git push*

1. Undo added file in local using

*git reset <filename>*

1. Forcefully push the files to git

*git push –force*

1. Resolve merge issues
2. *Open the file which has merge issues*
3. search the file for the conflict marker <<<<<<<
4. *Modify the file contents and remove the conflict markers* <<<<<<< =======
5. *git commit –am “Resolved conflict”*
6. *git push origin master*

1. To avoid check-in of DLL, pycache files, .DS\_Store (mac file) into git, create .gitignore file with content -

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

.DS\_Store

1. Blacklist .idea folder

*git rm --cached -r .idea*

## Docker commands

1. To view exisiting docker images

*$ docker images*

1. Remove exisiting docker image

*$docker rmi –f img\_id*

1. Show all running containers

*$ docker ps*

1. Shows all containers that were stopped or run previously

*$ docker ps –a*

1. Inspect docker container

*$ docker inspect container\_id*

1. Removes docker container

*$ docker rm container\_id*

1. Copy file from local to a folder in docker container

*$ docker cp /host/path/File <containerId>: /path/within/container*

1. Copy file from local to a folder in docker container

*$ docker cp <containerId>:/file/path/within/container /host/path/target*

1. Remove all running containers

*$ docker container prune*

1. Check container logs

*$ docker logs container\_id*

1. Make docker image available for public

*$ docker push myimage*

1. To check the memory used by each command in DockerFile

*$ docker history image\_name*

1. Delete docker image, already running in a container

ps – Linux command to check the running process

* 1. Get running container id

$ docker ps -a

* 1. Stop the running container

$ docker stop container\_id

* 1. Delete docker image

$ docker rmi –f image\_id

1. Run python code inside docker container

* python3 -c "import pyaudio as p;print(p.\_\_version\_\_)"
* python -c "import pyaudio as p;p.pa.\_\_file\_\_"

## Heroku commands

1. Login to heroku

*$ heroku login*

1. Create new app in heroku

*$ heroku create*

1. Move code from git master to heroku git

*$ git push heroku master*

1. Move from git branch to heroku git

*$ git push heroku branch:master*

1. Delete apps from heroku

*$ heroku apps : destroy app\_name*

the additional :master here is saying push my local myfeature branch into the master branch on the remote - note: heroku can only deploy from the master branch.

1. Remove heroku git repo pointing to previously created app

*$ git remote remove heroku*

*$ git remote add heroku*

1. Check heroku logs

*$ heroku logs --tail*

## LINUX Commands

**In Linux everything is considered as a process**

1. Check the kernel version

*$ uname –r*

1. Download files from internet

*$ wget http://linktodownloadfile*

1. Move file from source to destination

*$ mv srcLocation destLocation*

1. Remove entire folder

*$ rm –rf /usr/myFolder/*

1. Copy all the files from source folder to destination folder (without being asked to overwrite)

*$ cd srcLocation*

*$ yes | cp -rl \* destLocation*

1. Edit and Save file using nano editor

*Edit required items*

*Press ctrl+o and (change the file name if required) press Enter*

*Press ctrl+x and then Y to save the file*

1. Edit and Save file using vim editor

*Press i to start editing*

*Once editing is done, press Esc*

*Type, :wq to save the file*

1. Run shell script file

*$ sh sample.sh*

1. Give permission to a folder

*$ sudo chmod a+rw /usr/src/folder*

1. Check size of a folder

$ df –h

# 10. Ongoing project

Currently working on building Seq2Seq model for Speech recognition mentioned here-

*Seq2Seq model –Build model using MELD dataset*

# 11.References

* Speech recognition API

<https://realpython.com/python-speech-recognition/>

<https://pypi.org/project/SpeechRecognition/>

* Audio analysis

<https://medium.com/game-of-bits/audio-analysis-part-1-sound-waves-things-you-must-know-1e10851cc109>

<https://archive.org/details/SpectrogramCepstrumAndMel-frequency_636522>

* GANs

<https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>

* Attention GAN

https://github.com/taoxugit/AttnGAN

* Obj GAN

<https://github.com/tohinz/semantic-object-accuracy-for-generative-text-to-image-synthesis>

* OP-GAN with SOA

<https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>

* Alsa overview

<http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch13lev1sec2.html>

<https://tldp.org/HOWTO/Alsa-sound.html#toc2>

* MAC core –audio

<https://developer.apple.com/library/archive/documentation/MusicAudio/Conceptual/CoreAudioOverview/WhatisCoreAudio/WhatisCoreAudio.html>

* Dummy card

<https://www.alsa-project.org/main/index.php/Matrix:Module-dummy>

# **12.Projects in Github**

* Deploy streamlit application in Heroku

[*https://github.com/savitha91/Streamlit\_HerokuDeploy*](https://github.com/savitha91/Streamlit_HerokuDeploy)

* Deploy Flask application in Heroku

[*https://github.com/savitha91/Flask\_HerokuDeploy*](https://github.com/savitha91/Flask_HerokuDeploy)

* Audio signal Analysis

[*https://github.com/savitha91/AudioSignal\_Analysis*](https://github.com/savitha91/AudioSignal_Analysis)

* Speech to image using Attention GAN

[*https://github.com/savitha91/SpeechToImg\_AttnGAN*](https://github.com/savitha91/SpeechToImg_AttnGAN)