

Speech data Augmentation in the frequency domain using Deep Learning Methods

*A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of*

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

Submitted by

Shashank Shirol

170905178

Under the guidance of

Dr. Chng Eng Siong

Assoc. Professor

School of Computer Sci. and Engg.

Nanyang Technological University

Dr. Muralikrishna SN

Asst. Prof. – Senior Scale

Dept. of Computer Sci. and Engg.

Manipal Institute of Technology



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

July 2021



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
12/07/2021

CERTIFICATE

This is to certify that the project titled **Speech data Augmentation in the frequency domain using Deep Learning Methods** is a record of the bonafide work done by **Shashank Shirol** (*Reg. No. 170905178*) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in **COMPUTER SCIENCE & ENGINEERING** of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2021.

Dr. Muralikrishna SN
Asst. Prof. – Senior Scale, CSE Dept.
M.I.T, MANIPAL

Prof. Dr. Ashalatha Nayak
HOD, CSE Dept.
M.I.T, MANIPAL

Project / Internship Offer Letter



Reg. No. 200604393R

2 November 2020

Matric No: N2000494L

SHASHANK SHANTAVEERAPPA SHIROL
shashank.shirol1@gmail.com

LETTER OF ADMISSION AS NON-GRADUATING STUDENT

Congratulations and welcome to the Nanyang Technological University, Singapore (NTU) fraternity!

We are pleased to inform you that your application for admission as an international non-graduating student at NTU under the NTU-India Connect Research Internship Programme has been successful.

Due to the current COVID-19 pandemic, you will be required to work full-time remotely in your home university, Manipal University during the research internship period from **4 January 2021 to 4 June 2021** and not allowed to travel to NTU during the approved internship period.

NTU will not apply for a valid visa or Training Employment Pass (TEP) for you to enter Singapore. Under the supervision of Assoc Prof Chng Eng Siong from NTU India Connect, you agree to abide by the above mentioned terms, stays contactable at all times, reports to the faculty supervisor via online communication, and delivers your research outcomes as required by your supervisor by the stipulated timeline. NTU will provide the e-resources and network access for you to conduct your research project remotely.

You will be required to pay for the miscellaneous student fee of an amount of SGD207.00. The fees will be reimbursable and processed by the India Connect Office upon the completion of the internship and the receipt of the proof of payment from you. However, in the event that the internship has not been successfully completed or you choose to drop out, the India Connect Office reserves the right not to refund the amount paid by you.

Please submit a weekly/monthly progress report to Assoc Prof Chng Eng Siong after the start of the internship.

Kindly provide your contact details in Annex A in order for your supervisor in NTU, to be able to contact you during this remote internship period.

Yours sincerely

A handwritten signature in black ink, appearing to read "Vun Chan Hua".

Associate Professor Vun Chan Hua, Nicholas
Associate Chair (Academic)

cc NTU-India Connect Office
Chair,
Supervisor

School of Computer Science and Engineering

N4-02a-32, 50 Nanyang Avenue, Singapore 639798, T: +65 6790 5786, F: +65 6792 6559, www.ntu.edu.sg

Project Completion Letter



Reg. No. 200604393R

24 June 2021

Re: Shashank Shirol internship with NTU, Jan-June 2021


This to certify that **Mr. Shashank S Shirol (Matric No. N2000495H; Reg. No. 170905178)** has successfully completed his internship under my supervision at Nanyang Technological University, Singapore. Due to COVID-19 pandemic, the work was conducted remotely with him working from home.

Shashank worked on a project titled, ***“Speech Data Augmentation in the Frequency Domain using Deep Learning”*** from **4th January 2021 to 4th June 2021** (5 months). The aim of this project was to explore deep learning solutions for the task of speech data augmentation while dealing with the spectral representation of audio. Shashank was successfully able to design and implement a pipeline that allowed for a cost-effective model to generate noisy speech samples from previously unseen clean samples, as was the goal of this project.

During the internship, he demonstrated good character and self-motivation to see the project through in a systematic manner. He was able to obtain the desired results, document them meticulously, and present them cogently in our weekly meetings. He possesses a strong work ethic and exudes positivity. His performance exceeded expectations and was able to deliver on the project on time. I have been very happy to have hosted him, he has been an excellent student to mentor.

Please feel free to contact me at aseschn@ntu.edu.sg if you have any questions.

Yours sincerely,

Chng Eng Siong, 

A/Prof School of Computer Science and Engineering,

Nanyang Technological University, Singapore

Email: ASCHNG@ntu.edu.sg

Website: <https://personal.ntu.edu.sg/aseschn/>

Page 1 of 1

School of Computer Science and Engineering

N4-02a-32, 50 Nanyang Avenue, Singapore 639798, T: +65 6790 5786, F: +65 6792 6559, www.ntu.edu.sg

ACKNOWLEDGMENTS

This project titled, “Speech data Augmentation in the frequency domain using Deep Learning,” would not have been possible without the support and guidance of my mentors, Dr. Chng Eng Siong, Assoc. Professor, NTU Singapore and Dr. Muralikrishna SN, Asst. Professor – senior scale, MIT Manipal. I would like to extend my sincerest thanks to them for helping me throughout the duration of the internship. It was a great privilege and an honour working, learning, and growing under their guidance.

I would also like to extend my gratitude towards our Dept. HOD and the teachers at Department of Computer Science at MIT, Manipal for their constant support and motivation. Their unwavering support through the internship and the past 4 years at MIT has developed me into the person I am today, both on professional and on personal front. I would also like to thank the Director, MIT Manipal for providing a platform for us, students, that enables us to pursue these foreign internships without any hassle and for also maintaining transparency through the internal working of the college administration.

I would also like to take this opportunity to thank my parents for their love and support during these trying times. Their unwavering support was a key reason for me to be able to successfully complete the project under a variety of changes, both external and internal.

Last, but surely not the least, I would like to thank the Almighty God who has always guided me along the right paths of life.

Shashank Shirol

ABSTRACT

Data augmentation for ASR (Automatic Speech Recognition) has been explored in various ways, from applying time shifts to speed perturbation. The key aspect of these techniques is that they augment the raw audio to generate new samples. Another approach to the problem is the application of augmentation techniques on spectrograms of the input audio. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

This project is trying to solve the problem of generating noisy speech data from clean speech data by treating the spectrograms of the clean audio signals. We intended to find and modify a suitable GAN (Generative Adversarial Network) that can learn the noise characteristics mapping from a small dataset to successfully be able to generate new noisy samples given clean samples.

We studied two such GANs – SinGAN [1] and CUT [2]. SinGAN is a generative model that is trained to learn from a single natural image. The model learns the internal distribution of patches in an image and then, can generate high quality diverse samples that have the same visual content as the input image. CUT (Contrastive Unpaired Translation) is an Image-to-Image translation model that works by maintaining correspondence in content but not appearance – by maximizing the mutual information between corresponding input and output patches. We were successful in designing a pipeline that allowed for cost-effective speech data augmentation, our method allows us to train a neural network with just 5 minutes' worth of data. Once trained, the model allows us to generate speech data that is infused with noise characteristics of the training speech data.

Throughout the rest of the report/thesis, we will touch on the importance of data-augmentation for ASR, we will see what other methods have been explored, and look at deep-learning approaches; we will also explore the extent of effectiveness of these models via several experimental setups. We will rely on a distance metric called, LSD (Log Spectral Distance) – a distance measure (expressed in dB) between two spectra, to determine the efficacy of our system.

Table of Contents			Page No
Acknowledgement			i
Abstract			ii
List Of Tables			iv
List Of Figures			v
Chapter 1 INTRODUCTION			1
1.1	General Introduction to the topic		1
1.2	Problem Definition		1-2
1.3	Motivation and Goal		2
Chapter 2 BACKGROUND THEORY / LITERATURE REVIEW			3
2.1	Background Theory		4-8
2.2	Objectives		9
Chapter 3 METHODOLOGY			10
3.1	Methodology		10-12
3.2	Implementational Details		12-17
Chapter 4 RESULT ANALYSIS			18
4.1	Experimental Setups		18-20
4.2	Results		20-28
Chapter 5 CONCLUSION AND FUTURE SCOPE			29
5.1	Conclusion		29-30
5.2	Future Scope		30
REFERENCES			31
ANNEXURES			32-38
PROJECT DETAILS			39
PLAGIARISM REPORT			

LIST OF TABLES

Table No	Table Title	Page No
4.1	Summary of Experiments	20
4.2	Results for Experiment – I	21
4.3	Results for Experiment – II	22

LIST OF FIGURES

Figure No	Figure Title	Page No
1.1	Data Augmentation techniques used in SpecAugment	2
2.1	Flow of data in a trained model	3
2.2	A spectrogram example	4
2.3	A generic GAN architecture	5
2.4	Multi-Scale SinGAN architecture	6
2.5	Internal structure of a Generator in SinGAN	7
2.6	Demonstration of the Paint2Image task of SinGAN	7
2.7	The CUT architecture	8
3.1	Flow of data in a SinGAN model	11
3.2	Flow of data in a CUT model	12
3.3	A spectrogram generated by a visualization package	13
3.4	Mapping and Un-mapping Process	13
3.5	One-to-many sample generation	14
3.6	Modifications to the window dimensions in SinGAN	15
3.7	Patch-wise Contrastive learning for spectrograms	17
4.1	SinGAN Experiment – I: output	21
4.2	SinGAN Experiment – I: LSD vs. Scale graph	21
4.3	SinGAN Experiment – II: output	22
4.4	SinGAN Experiment – II: LSD vs. Scale graph	22
4.5	CUT Experiment – I: output	24
4.6	CUT Experiment – II: output	24
4.7	CUT Experiment – III: output	25
4.8	CUT Experiment – IV: output	26
4.9	CUT Experiment – V: output	26
4.10	Salt and Pepper Digital Noise Example	27
4.11	CUT Digital Noise Experiment – I: output	27
4.12	Speckle Digital Noise Example	28
4.13	CUT Digital Noise Experiment – II: output	28

CHAPTER 1

INTRODUCTION

1.1 GENERAL INTRODUCTION TO THE TOPIC

Automatic speech recognition (ASR) is a process of transcribing an utterance, given the speech waveform, and further processing it to derive meaning. In our day-to-day lives, we use ASR almost ubiquitously – from talking to personal assistants like the Apple Siri and/or Google Assistant to using speech-to-text dictation software to create documents.

Training or developing a robust ASR using deep learning models requires a lot of speech data, which is almost never readily available. One way we can tackle this problem is through data augmentation. Data augmentation, traditionally, is a process of deriving new samples from a given set of samples by applying a set of transforms. For example, in case of image data, techniques like scaling, translation, rotation, flipping, adding noise, etc. have been employed previously.

When it comes to speech data, we have two options – applying augmentation techniques on the waveform or on the spectrograms (*a visual representation of the spectrum of frequencies of a signal as it varies with time*). When working with speech data, different techniques employed are noise injection, shifting time, changing pitch, changing speed, etc.

In this project, we explore a deep learning-based solution (as opposed to previously explored solutions that use handwritten policies) to data augmentation of speech data that works on the spectrograms of the given samples. In this project, we extensively studied two Generative Adversarial Networks (GANs) – SinGAN and CUT. We found success in our rendition of CUT that uses a special preprocessing pipeline to achieve the desired results.

1.2 PROBLEM DEFINITION

To train Automatic Speech Recognition (ASR) systems, having a large dataset is crucial, which may not be readily available at times. So, we need a method to generate new data from existing data – data augmentation. Data augmentation is a process of synthesizing new data from previously seen data.

One aspect of augmenting data is generation of noisy samples given clean samples i.e., adding of new synthetic samples. Here, we feed a system with examples of clean and noisy samples (or, sometimes just the noisy samples, in case of SinGAN) for it to learn the characteristics of the noise and then be able to generate corresponding noisy samples of the unseen clean samples.

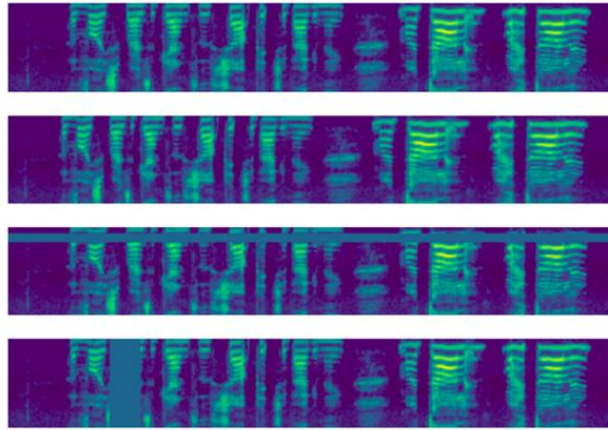


Fig. 1.1: From top to bottom, the figures depict the Log Mel spectrogram of the base input with no augmentation, time warp, frequency masking and time masking applied.

1.3 MOTIVATION AND GOAL

An approach of augmenting audio data via their spectrograms is explored by Google AI in their 2019 paper SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition [3]. They explore augmentation by modifying spectrograms by warping them in the time direction, masking blocks of consecutive frequency channels, and masking blocks of utterances in time. Fig. 1.1 shows these modifications for clarity.

We, in this project, take the hand-engineered policies out of the equation and make this an unsupervised learning problem by training neural networks to learn the characteristics of noise. Our goal for this project, in other words, is to develop a GAN that, when trained on noisy spectrogram(s), can produce noisy spectrograms from input clean spectrograms.

In the sections that follow, we shall discuss the background and existing solutions, along with our proposed method and, ultimately, we will test the efficacy of the above two models in the generation of noisy samples given clean samples and document the findings. We shall conclude with some analysis of the results, discuss use-cases of this project, and explore the future scope for improvement.

CHAPTER 2

BACKGROUND THEORY / LITERATURE REVIEW

The inspiration for this project stems from the lack of availability of large, paired speech audio data in the real world, which is extremely important in training a robust Automatic Speech Recognition (ASR) system. To solve this problem, we take a new approach – deep learning. We explore two different GANs (Generative Adversarial Networks) to learn noise characteristics from noisy samples such that these can be infused to unseen clean samples at a later stage.

Both these architectures, on their own, are powerful at several image-manipulation tasks. We tune and modify these architectures to work on spectrograms of speech audio data to, ultimately, achieve data augmentation. Both SinGAN and CUT are relatively new works published in 2019 and 2020, respectively.

To put the whole project into perspective, what we are trying to achieve can be modelled as in Fig. 2.1:

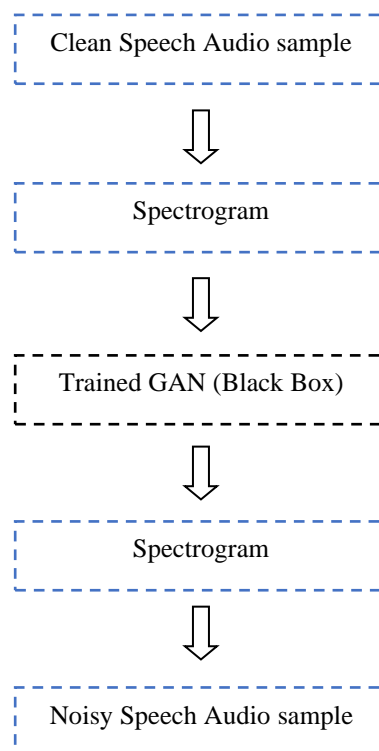


Fig. 2.1 Flow of data in a trained model

2.1 BACKGROUND THEORY

2.1.1 Spectrogram

Waveforms of audio signals show us the loudness of the sound wave changing with time. These amplitudes, however, are not very informative, as they only talk about the loudness of the recording. To better understand the audio signal, it is necessary to transform it into frequency-domain which tells us what frequencies are present in the signal. This is done by applying Fourier Transform on the signal. This, however, is still not enough as we lose time information, i.e., when the frequencies recorded were heard, or captured.

This is where Spectrograms come into picture. A visual representation of frequencies of a given signal with time is called Spectrogram. In a spectrogram representation plot – one axis represents the time; the second axis represents frequencies, and the colours represent magnitude of the observed frequencies at a particular time. Fig. 2.2 shows a sample spectrogram.

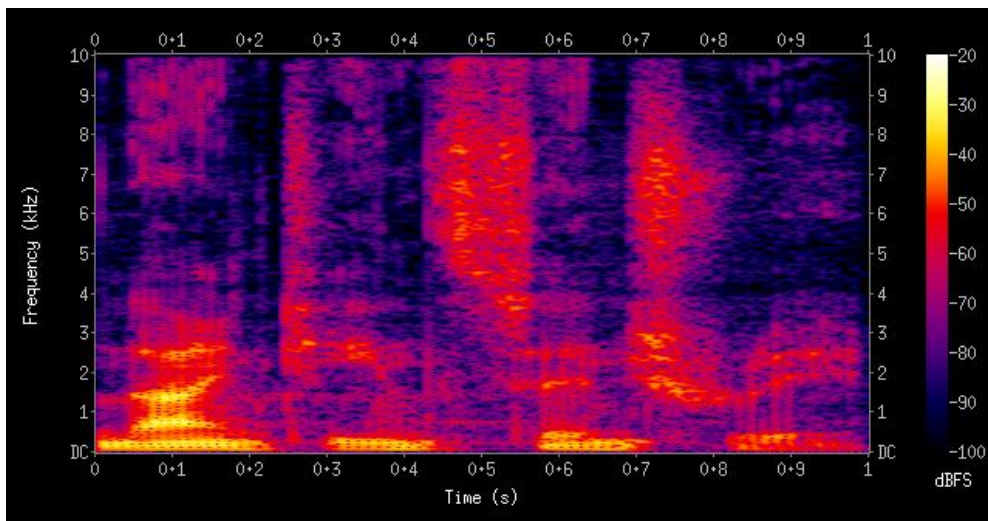


Fig. 2.2: A Spectrogram of the spoken words "nineteenth century".

2.1.2 Generative adversarial network (GAN)

Generative Adversarial Network, or GAN for short, is a technique for generative modeling using deep learning; generative modeling is an unsupervised task wherein the machine automatically discovers and learns patterns in input data in such a way that the model can then be used to generate or output new samples that appear to be from the original dataset but are not. GANs are a clever way of training a generative model by framing the problem as a supervised problem with two-components: the generator and the discriminator. Simply put, the generator model is used to generate new samples while the discriminator model is trained to classify samples as either real (from the dataset) or fake (generated).

The two components work in tandem in a zero-sum game, adversarial, until the discriminator model is fooled about 50 percent of the time, meaning the generator is generating believable samples. GANs have been researched extensively since they were first introduced in 2014 by Ian Goodfellow in his paper Generative Adversarial Nets [4], most notably in image-to-image translation tasks. Fig. 2.3 captures an example of the GAN architecture.

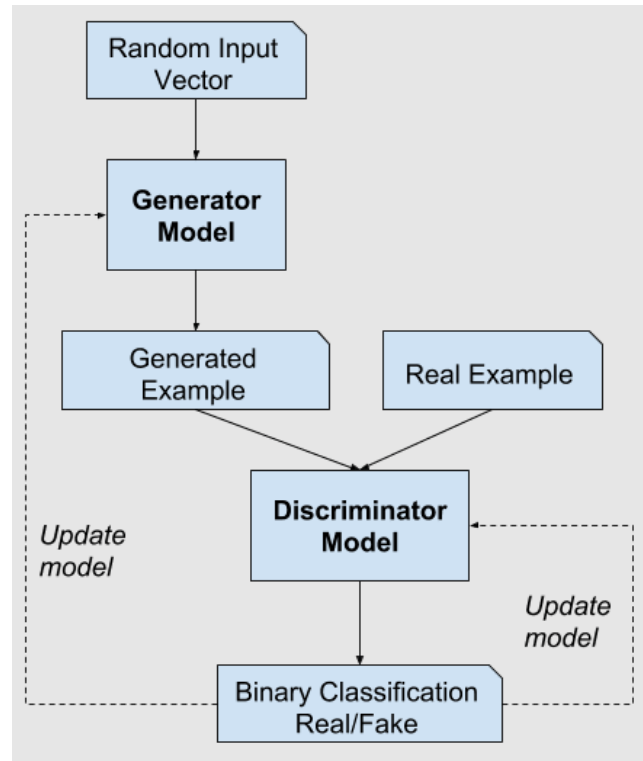


Fig. 2.3: The Generative Adversarial Network architecture.

The two components, the generator and discriminator, are trained together. The generator generates a batch of samples which, along with a batch “true” samples, are passed to the discriminator that classifies them as “real” or “fake”. The discriminator is then updated to get better at its task and so is the generator, depending on how much the generated samples were able to fool the discriminator. These models are bettered on each iteration and when the generator can generate samples that can fool the discriminator about half the time, we stop the training process.

In the sections that follow, we discuss two GAN architectures that we studied and tweaked during this project to obtain the desired results. The reason for choosing these architectures becomes apparent as we explain how they work.

2.1.3 SinGAN – Learning a Generative Model from a Single Natural Image

SinGAN is a generative model that is trained to learn from a single natural image. The model learns the internal distribution of patches in an image and then, can generate high quality diverse samples that have the same visual content as the input image.

SinGAN uses a layered architecture that enables it to learn patch distribution of different scale images at every layer. A multi-scale GAN has a series of pairs of generators and discriminators that are trained one-by-one. The discriminators are modelled such that they work on patches of the image and not the entire image (to prevent the generator from learning to reproduce the entire image).

Once the generators learn to produce good images that “fool” the discriminator at that level, we keep it fixed and move up the level. The architecture is trained in a coarse-to-fine fashion i.e., lower-level generators learning coarser features and upper-level generators learning finer features.

Since we only have one training image, we train using its patches. SinGAN can learn coarse features at lower levels and finer features at higher levels because effective patch size changes across levels, Fig. 2.4 establishes this. Fig. 2.5 depicts what makes up a generator.

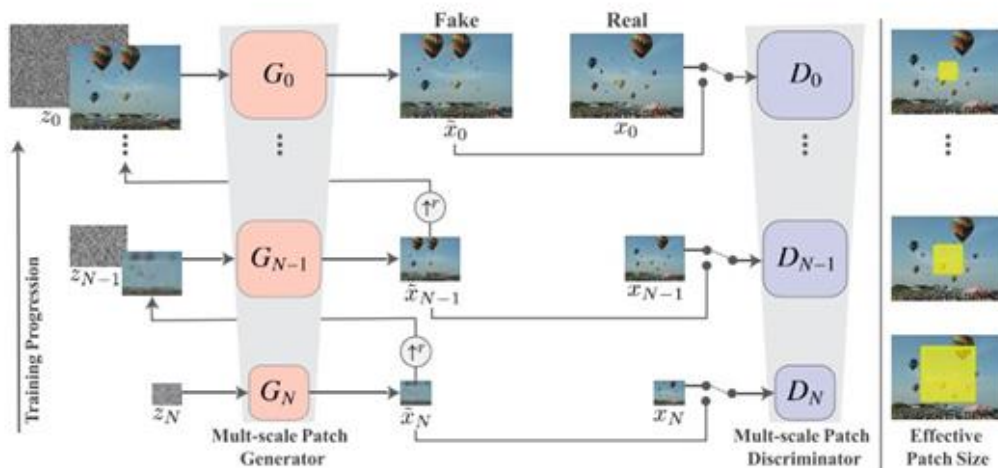


Fig. 2.4: Multi-scale SinGAN architecture.

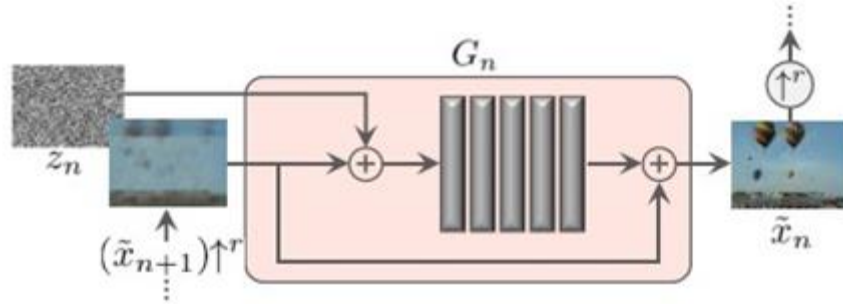


Fig. 2.5: Internal structure of a Generator in SinGAN with 5 convolutional layers.

Now, after training, we can use SinGAN for a variety of image-manipulation tasks. We take advantage of the fact that at inference, SinGAN can only produce images with same patch distribution as the training image. Thus, manipulation can be achieved by injecting a down-sampled image into the generation pyramid at some scale.

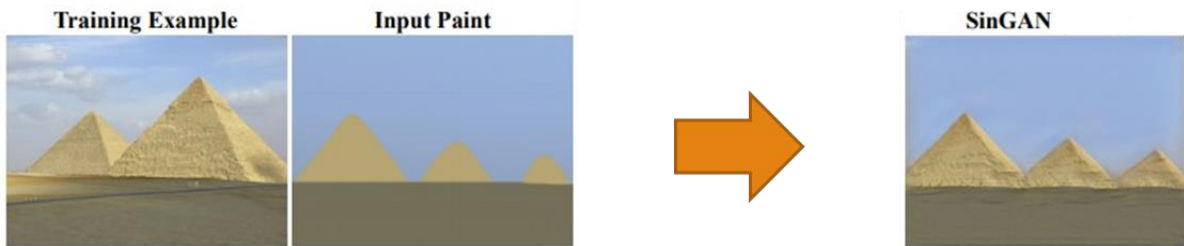


Fig. 2.6: Paint2Image task.

One such application or image-manipulation task is Paint2Image – a style transfer module that uses a style image (training image) and a content image (paint image) to generate outputs using the trained model. The generated images preserve the layout and general structure of the paint image while infusing fine details that match the training image. Fig. 2.6 shows what Paint2Image module does.

2.1.4 CUT – Contrastive Learning for Unpaired Image-to-Image Translation

Image-to-Image translation is a class of computer vision problem where the goal is to learn a mapping between an input image and an output image. There are several different architectures that help achieve this, one such GAN is called CycleGAN [5]. CycleGAN is an unpaired Image-to-Image translation model that is trained in an unsupervised manner using a collection of images from source and target domain that do not need to be related in any way.

We stress on the use of an unpaired translation model to mimic the real-world scenario wherein paired speech audio data is seldom available. Upon further study of the CycleGAN architecture, we realize that it could be made faster and more efficient; CUT, or Contrastive Unpaired Translation, which is a slightly modified variant of CycleGAN builds upon the shortcomings of CycleGAN – need to learn mapping in both directions, and an assumption that the mapping is always 1:1. Meaning, if we are only interested in learning the mapping from A to B, CycleGAN had to be unnecessarily trained on a reverse mapping as well, and CycleGAN assumes that there is always a 1:1 mapping for a given pair of images, which is not necessarily true.

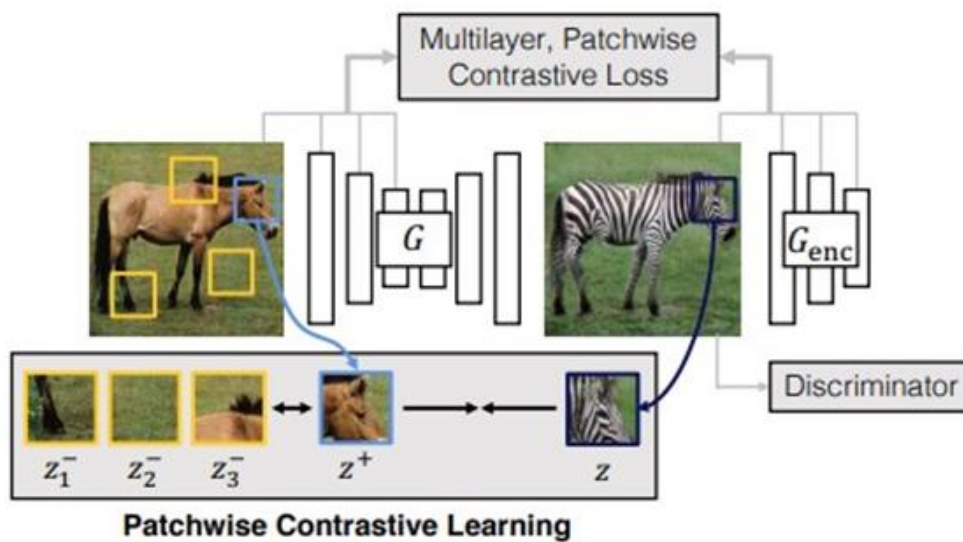


Fig. 2.7: CUT architecture.

CUT, therefore, learns twice as fast and is also able to produce comparable results to CycleGAN. Fig 2.7 shows the CUT architecture.

Patch-wise Contrastive Learning uses a simple model – the generated output-patch (Z) must appear closer (similar) to its corresponding input patch (Z^+) in comparison to other random patches ($Z_{\{1,2,3\}}$). This illustrates a minimax learning objective. This makes up the final loss function in addition to the traditional adversarial loss.

We use CUT in our project to generate a mapping from clean spectrograms to noisy spectrograms, such that when a trained model is presented with a clean spectrogram of previously unseen sample, it can generate its corresponding noisy spectrogram.

2.2 OBJECTIVES

The main objective of this project is to find a suitable Generative Adversarial Network architecture and tune it so that it can successfully learn the noise characteristics of noisy spectrogram(s) fed to it, and generate noisy spectrograms from previously unseen clean spectrograms, thus achieving a data augmentation pipeline. More aptly, the objective is to solve the main shortcoming of the data augmentation technique used in SpecAugment i.e., use of hand-engineered policies. We approach the problem as an unsupervised learning problem wherein we provide the system with examples of both clean and noisy spectrograms and expect the system to learn a mapping from one to another.

As we progress through the task, we also want to study the various factors affecting training and generation. These factors include duration of the audio sample (as this directly contributes to the size of its spectrogram), presence of a certain frequency band, gender of the speaker, and presence of non-stationary noise.

All in all, the objectives can be summarized as follows:

- Find Suitable Generative Adversarial Network Architectures for our use-case.
- Tune them and subject to various experimental setups.
- Collect data from the experiments and rework the code to improve performance.
- Document the findings and compare the performance of different GANs.

CHAPTER 3

METHODOLOGY

We began by exploring the use of SinGAN for our task, we selected SinGAN because of its unique architecture that can learn a representation using just one training sample, hence the name “Sin” GAN. SinGAN uses a robust architecture that can be utilised for many tasks, from Super-resolution to Style-transfer (or, Paint-to-Image) without altering the main architecture. We make use of its Style-transfer module since we want to learn characteristics from one image (a noisy spectrogram) and apply those to an input image (a clean spectrogram). This is exactly like the Style image – Content image dynamic that Neural Style transfer follows.

After sufficient tinkering with SinGAN and extracting the best performance out of it, we started looking at other architectures that might be helpful for our use-case. We concluded that our task is extremely similar to Image-to-Image translation task which has been studied extensively in the recent years. Based on performance and applicability, we decided to experiment with CUT (Contrastive unpaired translation), it is an Image-to-Image translation model that can learn a mapping from one domain of images to another without the need for parallel training examples.

In the sections that follow, we shall discuss how we use both the architectures and further delve into the implementational details of the same.

3.1 METHODOLOGY

3.1.1 SinGAN

We operate in the frequency domain of the signal. We take the Spectrograms of audio signals (noisy) and feed the multi-scale architecture of SinGAN; we tune it to learn the noise characteristics from the input spectrogram. We then feed the model with a Paint Image (Spectrogram of a clean speech audio signal), we want SinGAN to map the noise characteristics of the training image onto the Paint Image, while preserving the global structure of the Paint Image.

By Global Structure we mean that the content of the audio signal, after reconstruction from spectrogram, should not change. This is very similar to Neural Style Transfer [6] methods, where we have a content image and a style image and the GAN outputs an image with the content of the Content image in the style of the Style image.

The following block diagram (Fig. 3.1) shows an overview of the flow of data through the system.

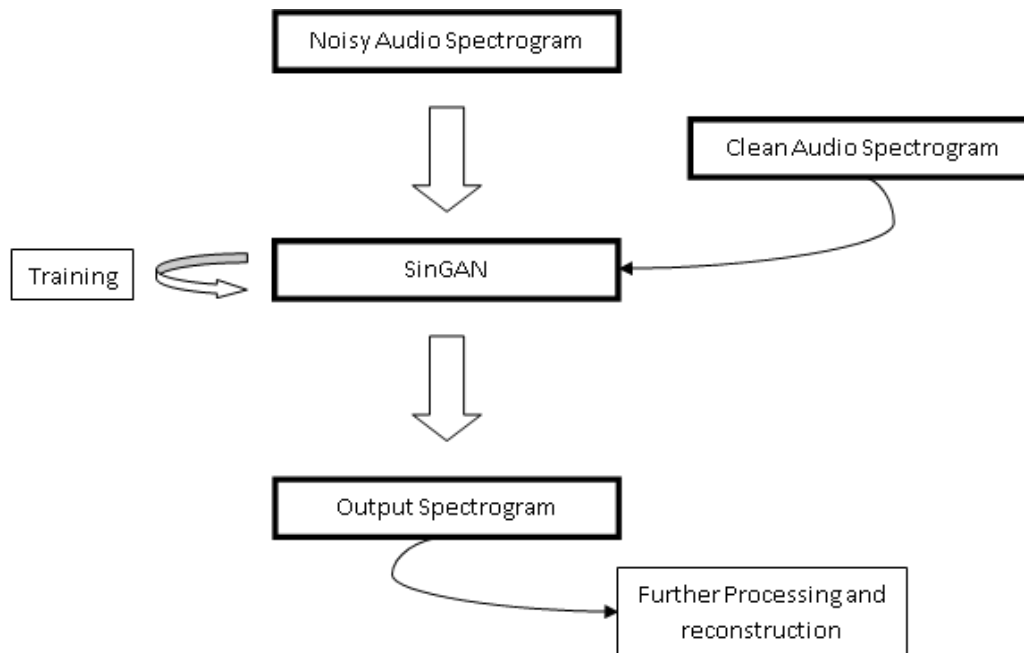


Fig. 3.1 Flow of data in a SinGAN model – during training and generation

3.1.2 CUT

We train CUT with spectrograms of clean as well as noisy samples. Since CUT is an image-to-image translation model, it attempts to learn a mapping from images of one domain to another, for the sake of clarity we will refer to these as: source domain (domain A) and target domain (domain B).

In our case, since we wish to learn a mapping from clean to noisy spectrograms, our domain A training files will be clean samples and domain B training files will be noisy samples. We try various combinations of these training files – from paired samples (same context clean and noisy samples) to unpaired samples (different context clean and noisy samples).

After training the model, we begin the generation process. The output generates the learned characteristics while retaining the context of the clean sample (speech content of the clean sample). The process can be similarly modelled as above and is shown in Fig. 3.2.

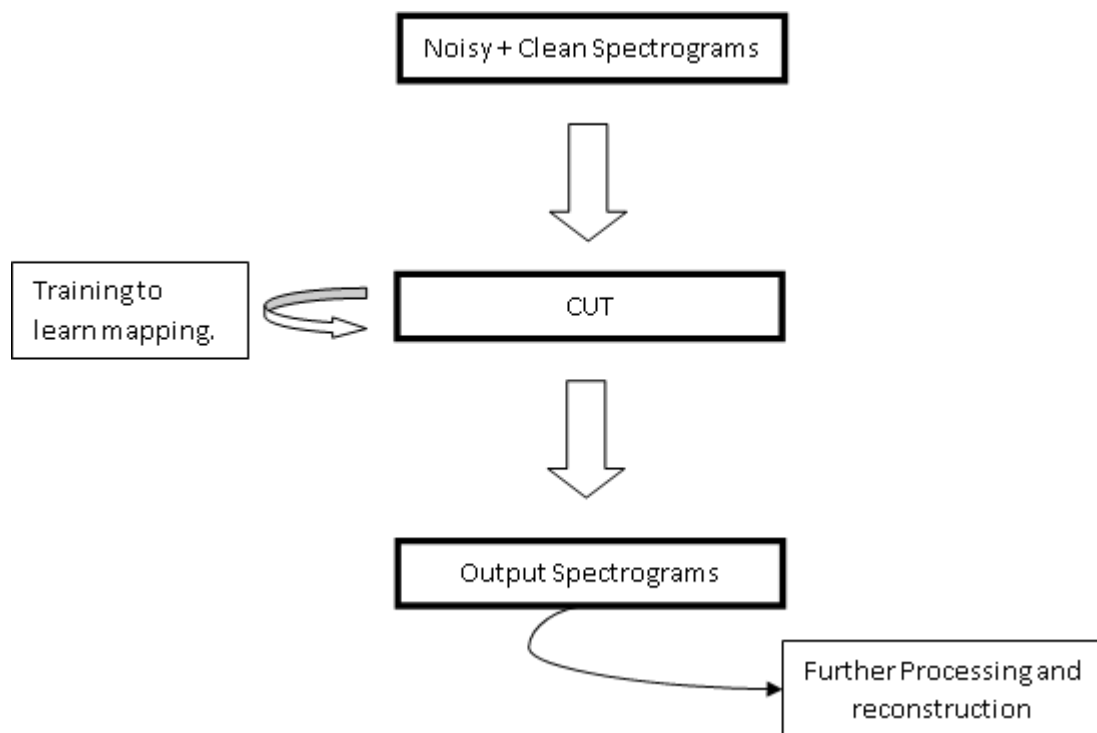


Fig. 3.2 Flow of data in a CUT I2I translation model – during training and generation

3.2 IMPLEMENTATIONAL DETAILS

While we touch upon both architectures, we will extensively cover the use of CUT since that is where we found our success in this project. Our rendition of CUT with a custom pre-processing pipeline showed drastic improvement in performance while at the same time cutting down on the amount of training data required.

In the following sub-sections, we cover implementational details of the project along with modifications to the two models that allow us to use them for our use-case.

3.2.1 Modifications of spectrogram representation

We know that a visual representation of the frequencies present in a given signal is called a spectrogram, and that in a spectrogram representation plot one axis represents time, the second axis represents frequencies, and the colors represent magnitude of the signal at a particular time. Internally, a spectrogram is represented as a 2D-matrix of real numbers, this form is usually mapped to a color scheme by various visualization packages before being presented to the user, e.g., Fig. 3.3.

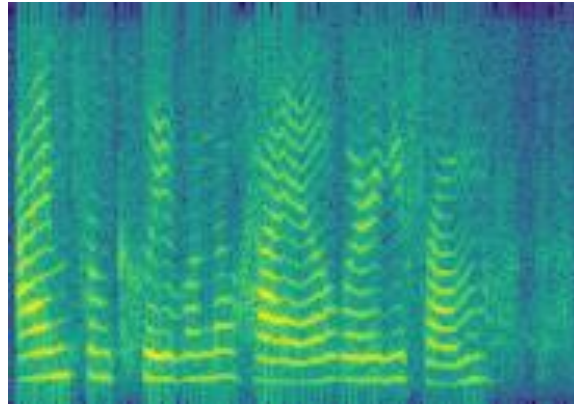


Fig. 3.3 A spectrogram generated by a popular visualization package.

The problem with such a representation is that we cannot recover the original spectrogram matrix for reconstructing the audio. Therefore, one of the first challenges was to modify the spectrogram representation such that we can reconstruct the audio signal from the output of the generative model. We then came up with a custom “mapping” function that maps a real-numbered matrix (a spectrogram) to a 0-255 scaled grey-scale image that can be fed to the generative models for training, and we then use an “un-mapping” function to obtain the original spectrogram used for reconstruction. This process is illustrated in Fig. 3.4.

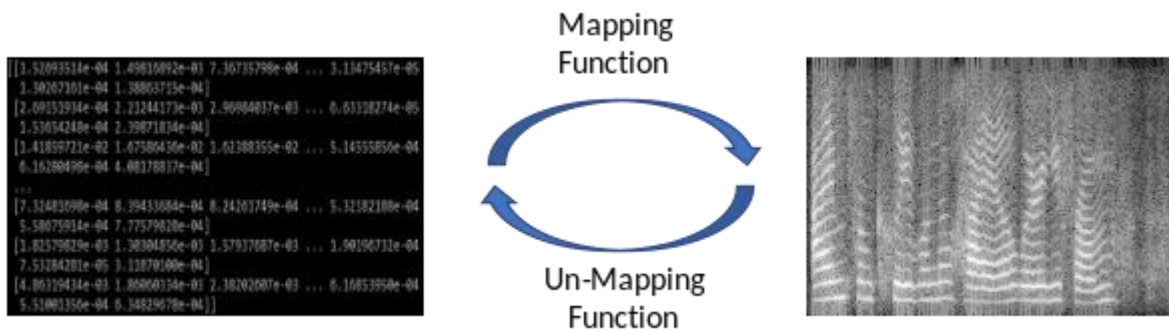


Fig. 3.4 Mapping and Un-mapping Process.

3.2.2 One-to-many sample generation

The shape and size of a spectrogram depends on the frequency bands present and the duration of the audio file it is extracted from; also, we only want to learn a mapping from clean to noisy, given the noisy samples share the noisy characteristics throughout the sample i.e., if two sections are extracted from a noisy audio sample, they are bound to share the noisy characteristics. We capitalize on this and split a large spectrogram into smaller components and treat them as individual samples. This allows us to train a model with less than 5 minutes’ worth of data.

The process of splitting and re-joining of the components is taken care of by a custom module internally. A high-level graphic illustrating how components (treated as individual samples) are fed to CUT (we do not use this technique with SinGAN since it only uses *one* training sample) is shown in Fig. 3.5.

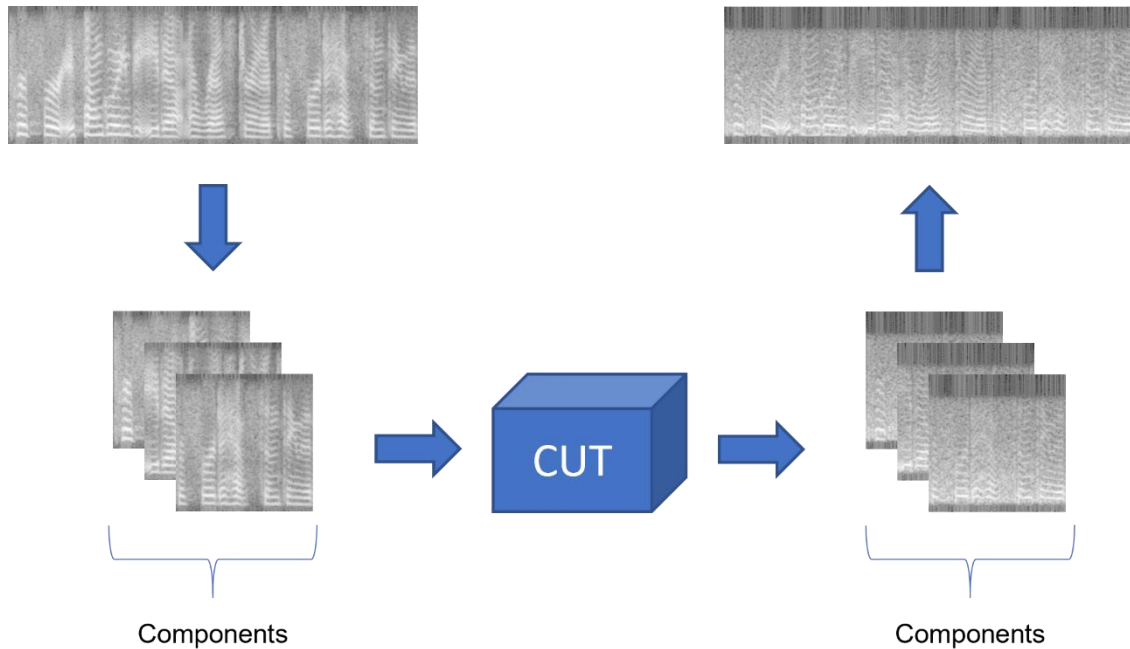


Fig. 3.5 One-to-many sample generation

3.2.3 Working changes to SinGAN and the effects:

We already know that SinGAN uses a multi-scale architecture to facilitate learning using only one training example i.e., the same image is used at various scales to train generators and discriminators at that level. This structure allows SinGAN to learn the characteristics from the image with varying details, from learning coarser features at lower levels to learning finer details at the higher levels.

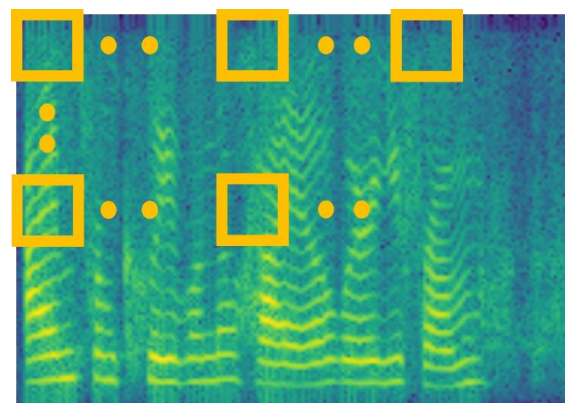
This allows us to control the level of detail infused in the generated images by SinGAN. Here, level 1 (starting level) will be “most” like the training sample but will lack the global structure of the input image, while level N (last level) will be “most” like the input image but will lack the characteristics of the training image. So, the optimal output is obtained by a level x such that $1 \leq x \leq N$.

This furthered our assumptions that SinGAN could be used to generate audio samples with varying degree of similarity to the ground-truth, thus helping with data augmentation. We

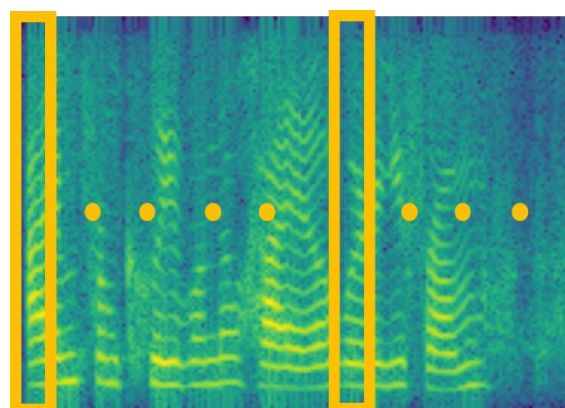
extensively experimented with SinGAN, the details of which are covered in the following chapter.

We worked on the official implementation of SinGAN provided by the original authors, we added helper functions to facilitate the use of spectrograms (instead of RGB images, as the authors intended), and we also incorporated a reconstruction module that took care of reconstructing the audio sample from the generated spectrogram internally, without any interference from the user.

We also worked on a modified version of SinGAN that used a different windowing policy than the original one. As we know, SinGAN uses a square window of fixed size that traverses the input reference image while training the discriminator(s) so that it has, technically, more samples to train on and with the setup as in, we were getting unsatisfactory results on our experiments, so in-order to better capture the characteristics of the training image, we altered the window dimensions to resemble a vertical cross-section, this forced the discriminator to learn that the vertical cross-sections are what are to be remembered (since a vertical cross-section covers the full range of frequencies). This idea is represented in the Fig. 3.6.



(A)



(B)

Fig. 3.6. (A) Depicts the original window shape.
(B) Depicts the *new* window shape.

Although this idea was logically sound, it worsened the performance. This deterioration in performance is attributed to the decrease in the number of samples such a window generated. Earlier when we had a square window, the number of samples the discriminator had for training was: $(W - s)^2$, where W is the width of the whole spectrogram (assume a square spectrogram for the sake of simplicity) and s is the width of the square window, this number was reduced to $(W - s)$ when we chose the vertical cross-section window shape.

So, we had to scrape this idea moving forward. And, once we realized we had squeezed the maximum performance from SinGAN, we started looking at other options, leading us to CUT.

3.2.4 Working changes to CUT and the effects:

Contrastive unpaired translation, or CUT, is an image-to-image translation model that can learn a mapping from images of one domain to images of another domain, without the need for paired training examples. This attribute of CUT is what is most attractive, the ability to learn a mapping with unpaired samples is a game-changer. This scenario is most prominent in the real world, where good datasets are already scarce, and availability of large, paired datasets are virtually non-existent. In this subsection we explore CUT and learn how it successfully obtained the desired results of this project.

CUT uses a patch-wise contrastive learning technique for one-sided translation. To learn more about the algorithm, please refer to the original work in [2] where they have explained in detail how they achieve this and why this approach “works” better. We will restrict ourselves with the working of CUT with respect to spectrograms and will be using actual data generated by CUT for this demonstration. A detailed graphic showing the working of patch-wise contrastive learning for spectrograms can be found in Fig. 3.7.

Here, a generated output patch (highlighted with dark blue border) must appear closer to its corresponding input patch (highlighted with light blue border), when compared to other random patches (highlighted with yellow borders). CUT makes use of multilayer, patch-wise contrastive loss which maximizes the mutual information between corresponding input and output patches and minimizes the mutual information between a selected patch and randomly chosen patches. In other words, we try to maximize the similarity between corresponding input-output patches (positives) while maximizing dissimilarity between the selected patch and other random patches (negatives).

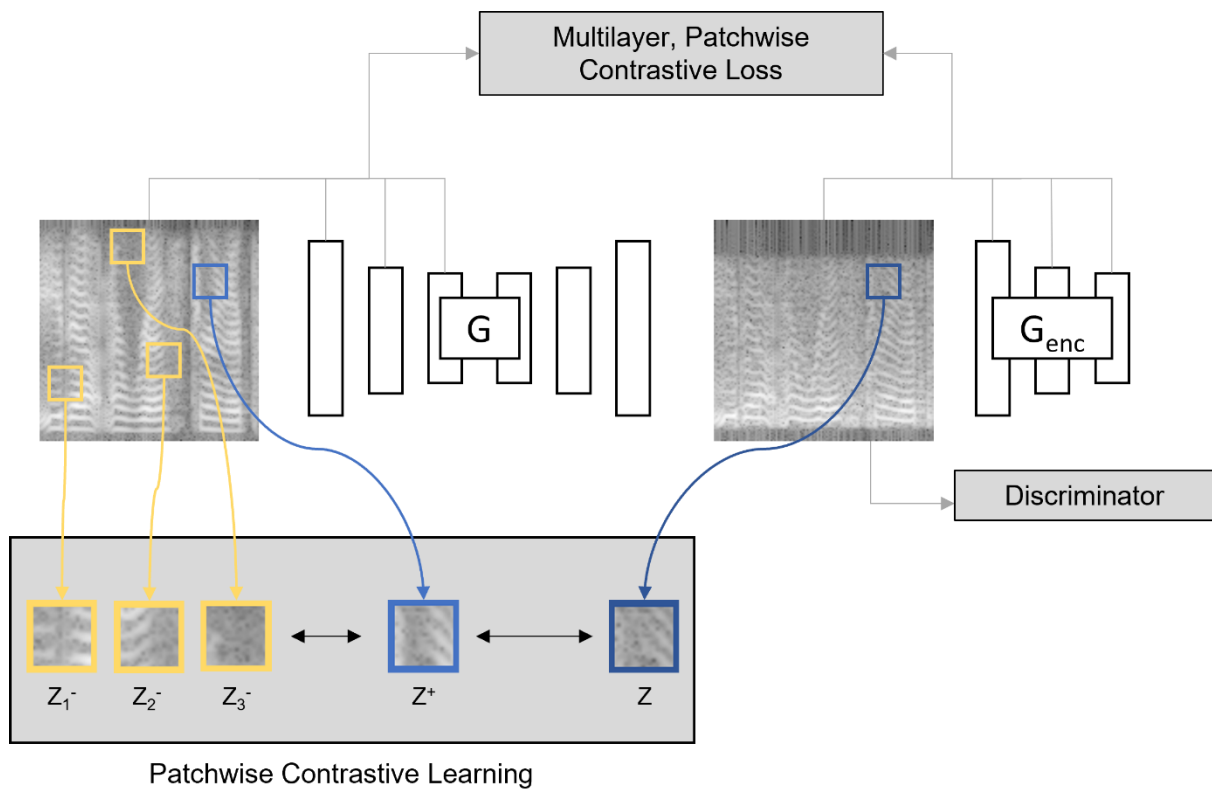


Fig. 3.7 Patch-wise Contrastive learning for spectrograms.

Here, while training CUT, we make use of the one-to-many sample generation module that allows us to train a satisfactory model with about than 5 minutes' worth of data. To facilitate fast splitting, we make use of threading and parallel computation. This reduces the time required to initialize the data-loader (for 10 audio files with about 60 components, in total) from $\sim 2.0s$ to $\sim 0.7s$ on a 4-core CPU.

Apart from this, we make use of the custom modules we wrote for SinGAN that allow for directly passing audio input and receiving audio output. The user is not burdened with extracting the spectrograms and the complexity of reconstruction process. We also adapt the custom transforms from SinGAN that have proven to work on spectrograms. To facilitate on-the-go re-joining of the generated spectrograms, we wrote a new data-structure that keeps track of the original spectrograms along with the number of components of each input spectrogram, and the order of the spectrograms.

CHAPTER 4

RESULT ANALYSIS

The novelty of the proposed approach is in its simplicity and low implementational cost: we can train a satisfactory model with less than 5 minutes' worth of data; this model can then be used to generate noisy samples. We test the efficacy of the proposed system on the RATS speech corpus [7] (for English only) which is a collection of sampled clean telephonic conversations and their parallel noisy versions. We make use of this dataset to verify the performance of the model since we have access to the ground-truth files (true noisy samples of clean samples that are fed to the models). We completely assess the strengths and weaknesses of the system, we also test it with speech data containing non-stationary noise i.e., when the statistics of the noise are not constant throughout the signal. For this we use the NOIZEUS corpus [8]. And, for assessment we make use of LSD (Log-Spectral Distance) metric to validate the goodness of results.

In the sections that follow, we will discuss the evaluation metric, along with its implementational details, and expand upon the several experimental setups that demonstrate the robustness of the proposed system.

4.1 EXPERIMENTAL SETUPS

4.1.1 Log-Spectral Distance

We measure the goodness of the results with a distance metric called, Log-Spectral Distance (LSD) [9]. LSD is a distance measure between two spectra and is defined as:

$$LSD = \frac{1}{L} \sum_{l=0}^{L-1} \sqrt{\frac{1}{N} \sum_{k=0}^{N-1} \left[10 \log_{10} \frac{|\hat{X}(l, k)|^2}{|X(l, k)|^2} \right]^2} \quad (Eq. 1)$$

Where l is the frame index, k is the index of the frequency bins, L is the total frames, and N is the frame length. $X(l, k)$ denotes the DFT coefficient of the ground-truth noisy speech signal and $\hat{X}(l, k)$ denotes the DFT coefficient of the output noisy speech signal from the model.

Before we can compute LSD of two audio files, we must take care of certain abnormalities that may arise due to processing with GAN. By doing so, we make the comparison free from any ambiguity.

The process consists of four steps as described below:

➤ **Time Alignment:**

This correction is required since the spectrogram generated by the GAN model may have shifted a few frames from their original location i.e., the spectrogram may have moved forward or backward in time. This is done in four sub-steps as follows:

- Pad the smaller signal with 0s to match the length of the bigger signal.
- Add a small random noise to the 0s in the signal to avoid $\log_{10}(0)$ (-inf) problem in spectrograms.
- Compute the lag (no. of samples to move) in the signals by performing cross-correlation on the spectrograms of the signals.
- Use NumPy to correct the lag by rolling required number of samples.

➤ **Normalization:**

This step is required to match the loudness of the output signal with the reference signal since GAN outputs might be either louder or quieter than the reference signal, and a distance metric would levy a penalty for this. This is done in three sub-steps as follows:

- Create a BS.1770 meter to detect the loudness of the reference file.
- Use the meter to detect the loudness of the reference file.
- Normalize the output file to the same loudness as the reference file.

➤ **Energy Equalization:**

This step ensures we match the energies of the two signals. We perform this step by calculating a gain of the output signal over the reference signal. To save computation time, we consider the top 10% data with the highest energy to calculate gain, and we equalize the weaker signal by multiplying it with the obtained gain value.

➤ **LSD Calculation:**

The final step of this process is the calculation of LSD as per equation 1, and after all the previous corrections are applied to the two files.

4.1.2 Experiments

We first try to simulate the most likely scenario wherein we have a mix of clean samples and a mix of noisy samples. We perform this experiment twice, once with paired samples and once

with unpaired samples, to see the effect on performance. We use the models to train/learn a mapping with this data and document the results. Second, we assess the impact on performance when we try to learn a mapping from clean to noisy with clean samples comprising of *only* female speakers and noisy samples comprising of *only* male speakers. And, for the final experiment we assess the system's performance on learning a mapping with the presence of non-stationary noise.

The following table summarizes the class of experiments we conducted with both SinGAN and CUT:

Table 4.1 Summary of Experiments

Name	Domain A (Clean)	Domain B (Noisy)	Paired
RATS (channel A)			
• Mixed Speakers – I	Mixed	Mixed	YES
• Mixed Speakers – II	Mixed	Mixed	NO
• Speaker Segregated	Female	Male	NO
NOIZEUS			
• Babble	Mixed	Mixed	NO
• Train	Mixed	Mixed	NO

Apart from the above stated experiments, we also subject CUT to another class of experiments. These are described and discussed in detail in the following sections.

4.2 RESULTS

4.2.1 SinGAN

Since SinGAN differs from CUT in both, implementation, and architecture, we conduct a separate set of experiments on SinGAN that loosely fall in the same classes of experiments summarized in Table 4.1.

We know that SinGAN trains with ONLY one training sample, we provide it with a single noisy RATS sample (as opposed to a set of noisy RATS samples to CUT) for learning a mapping. Once the model is trained, we use a clean RATS sample to generate outputs that have the characteristics of the training input file i.e., noise. And since we can control the detail infused in the generated output (by generating output at different scales), we generate output with different levels of detail and asses which one is the closest to the ground truth.

Note: We cannot test SinGAN with paired samples i.e., the train file and the generation file having the same content. This would break the purpose of testing the mapping. Hence, we only test with files that are contextually different.

- **Training File:** RATS Female Noisy (8k Hz)
- Generation File:** RATS Female Clean (8k Hz)
- Context:** Different

For the experiment above, the LSD scores between the ground truth file and for outputs with various level of detail are as follows:

Table 4.2 Results for this experiment

Scale	1	2	3	4	5	6	7
LSD	12.48	11.99	12.53	13.02	13.67	13.71	13.96

Spectrogram (for the best SinGAN output):

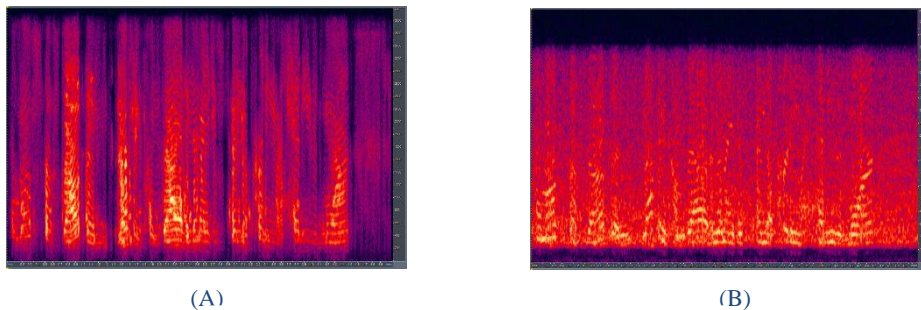


Fig. 4.1 (A) SinGAN output – scale 2; (B) GT

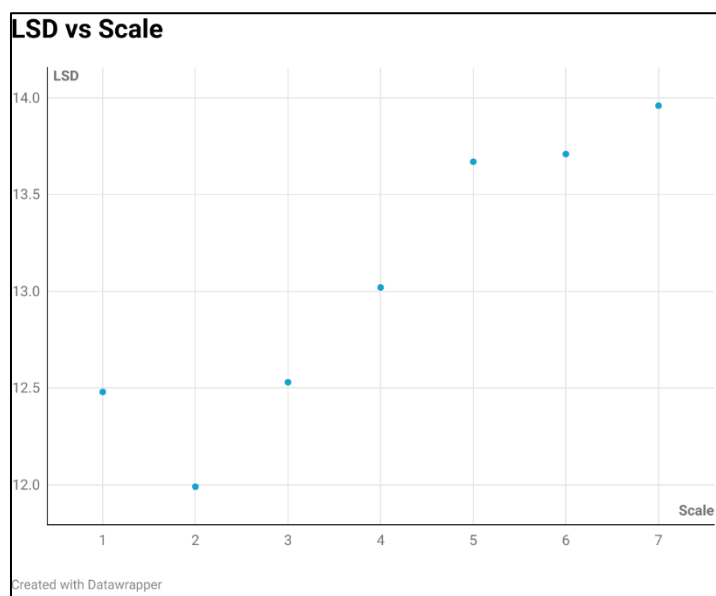


Fig. 4.2 LSD v Scale for this experiment

- **Training File:** RATS Female Noisy (8k Hz)
- Generation File:** RATS Male Clean (8k Hz)
- Context:** Different

For the experiment above, the LSD scores between the ground truth file and for outputs with various level of detail are as follows:

Table 4.3 Results for this experiment

Scale	1	2	3	4	5	6	7
LSD	15.65	14.49	14.44	14.60	14.81	14.83	15.09

Spectrogram (for the best SinGAN output):

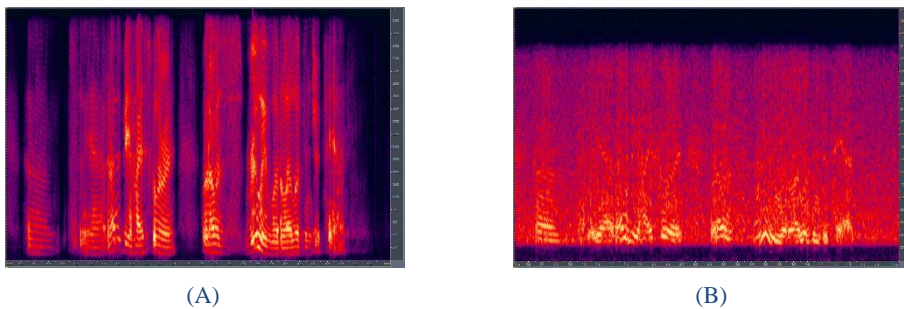


Fig. 4.3 (A) SinGAN output – scale 3; (B) GT

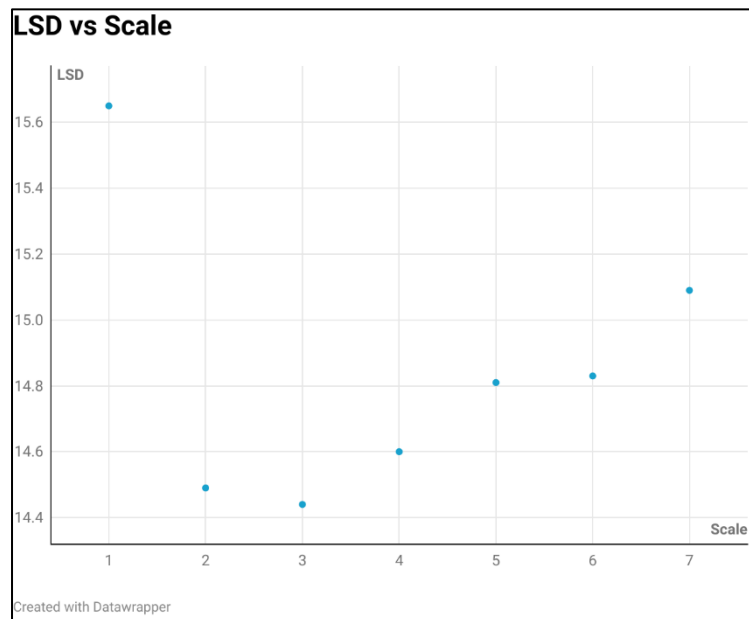


Fig. 4.4 LSD v Scale for this experiment

The results of the experiments follow the trend we assumed they would, the LSD scores first decrease and then increase. This is partly due to the fact that output generated at each scale is has varying level of detail from the training file. The output at scale 1 is more like the training file (noisy) and the output at scale 7 is more like the generation file (clean). So, the best output is obtained at a scale between 1 and 7 where the model infuses just the right amount of detail to make it, somewhat, resemble the ground truth file (a true noisy version of the generation file).

From Experiment 2 above, we see that the model takes a hit in performance when we use a female speaker for training and a male speaker for generation, so we may conclude that the model is not speaker independent, a characteristic that is true with CUT as we shall see ahead.

With sub-par performance as seen above, we felt it was futile to invest time and efforts into testing the model out in other scenarios. The results we obtained were after considerable tweaking and tuning of the model to extract every ounce of performance out it. After realizing we cannot extend SinGAN's performance any further, we started looking for other GAN architectures that might help: CUT.

4.2.2 CUT

CUT is an image-to-image translation model that can learn a mapping from one domain to another without the need for paired training samples. On paper, this architecture looked like it would work best for our use-case, and it did. CUT worked better than SinGAN from the get-go, producing results that were far superior to SinGAN's results. Tweaking and tuning the CUT as mentioned above only improved the results. Unlike SinGAN, there is no way to control the level of detail in the generated output and hence, for each generation file fed to the model we get one output file with the learned mapping applied.

As described in sub-section 4.1.2, we subject CUT to the aforementioned experiments and assess the performance of the model. Surprisingly enough, CUT sometimes performed worse when fed with parallel training data. So, we stick with the use of non-parallel data for our experiments, except in experiment 1. The details and the results of the experiment are discussed below:

- **Domain A:** Mixed (Male + Female) clean audio files (8k)
- Domain B:** Mixed (Male + Female) noisy audio files (8k)
- Context:** Same (i.e., parallelly trained)
- Generation:** Mixed (Male + Female) clean audio files (8k) from test set.

For the experiment, the LSD score (avg.), and an example ground truth spectrogram and the output spectrogram are given as follows:

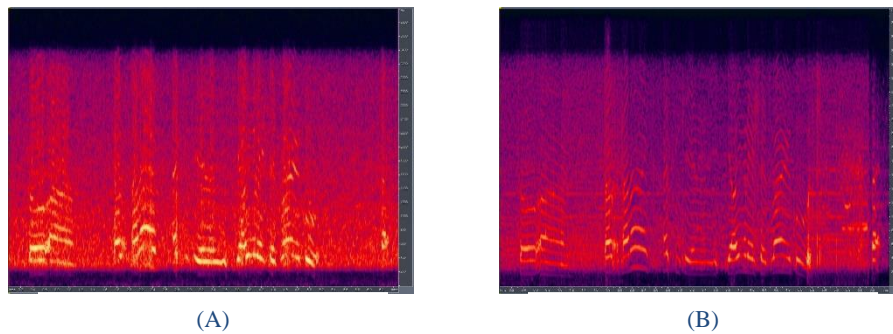


Fig. 4.5 (A) GT; (B) CUT output

LSD: 7.75

- **Domain A:** Fixed (Male + Female) clean audio files (8k)
- Domain B:** Fixed (Male + Female) noisy audio files (8k)
- Context:** Different (i.e., non-parallel files)
- Generation:** Fixed (Male + Female) clean audio files (8k) from train set.

In this experiment, we use the same set of files used to train the model, for generation because the model has never seen the ground truth files for them. This would, in no way, contaminate the generation process. This experiment demonstrates the most likely scenario in that we have access to a set of clean audio files and a set of noisy audio files (not parallel), and we wish to learn a mapping from clean \rightarrow noisy.

For the experiment, the LSD score (avg.), and an example ground truth spectrogram and the output spectrogram are given as follows:

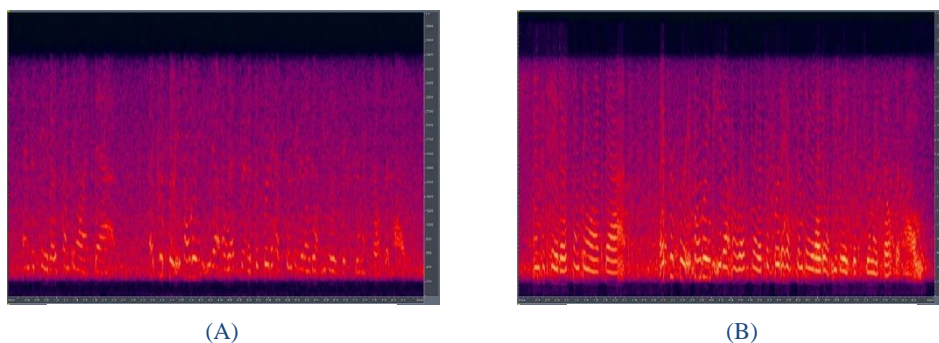


Fig. 4.6 (A) GT; (B) CUT output

LSD: 6.22

- **Domain A:** Fixed (Female) clean audio files (8k)
- Domain B:** Fixed (Male) noisy audio files (8k)
- Context:** Different (i.e., non-parallel files)
- Generation:** Fixed (Female) clean audio files (8k) from train set.

We can use the files from the train set for the same reasons as stated above. The LSD scores do not vary by much and the model is able to retain the content of the generation files while producing the outputs i.e., the outputs do not have a male voice or any other characteristics like deep voice, low pitch, etc.

For the experiment, the LSD score (avg.), and an example ground truth spectrogram and the output spectrogram are given as follows:

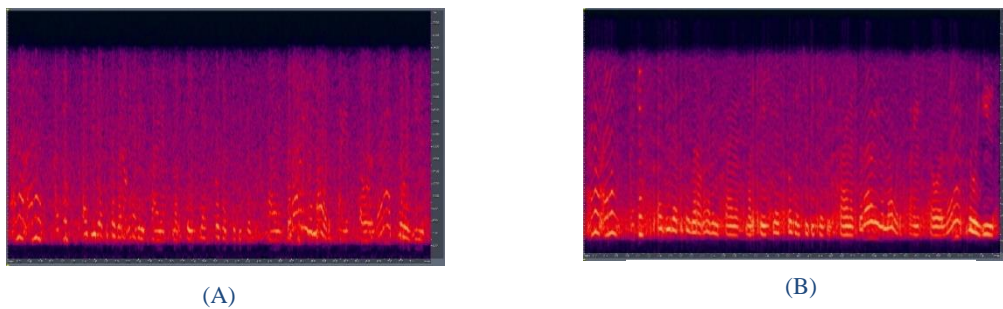


Fig. 4.7 (A) GT; (B) CUT output

LSD: 6.78

From the results obtained for the experiment, we could conclude that CUT learns a robust mapping that is speaker independent.

- **Domain A:** Mixed (Male + Female) clean audio files (8k)
- Domain B:** Mixed (Male + Female) noisy (non-stationary - *Babble*) audio files (8k)
- Context:** Different (i.e., non-parallel files)
- Generation:** Mixed (Male + Female) clean audio files (8k) from train set.

Here, we encounter one of the shortcomings of CUT. While CUT is great at learning a mapping when the noise characteristics are constant throughout, it fails when the noise is non-stationary i.e., the noise characteristics vary with different cross-sections of the sample. Under this class of experiments, we test with two kinds of noises – babble and a moving train.

For the experiment, the LSD score (avg.), and an example ground truth spectrogram and the output spectrogram are given as follows:

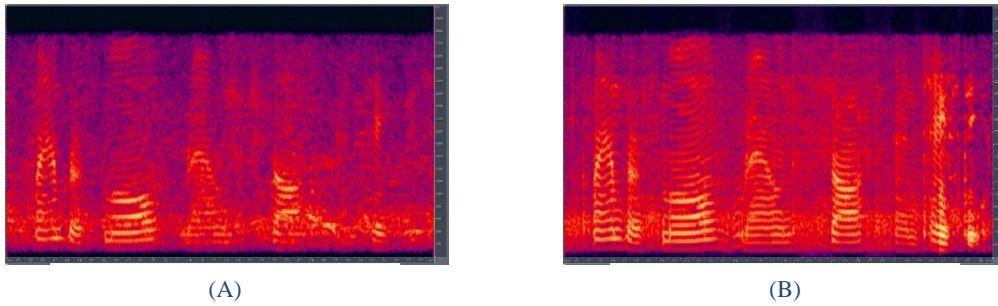


Fig. 4.8 (A) GT; (B) CUT output

LSD: 7.39

Although the LSD is deceptively small, on inspecting the generated output via a hearing device, we can conclude that CUT fails in capturing the noise. One of the reasons why the LSD is low because, non-stationary noise, unlike stationary noise seen until now, is not captured properly in the spectrogram. Which is also why CUT fails in this task.

- **Domain A:** Mixed (Male + Female) clean audio files (8k)
- Domain B:** Mixed (Male + Female) noisy (non-stat. – *Moving Train*) audio files (8k)
- Context:** Different (i.e., non-parallel files)
- Generation:** Mixed (Male + Female) clean audio files (8k) from train set.

The premise of this experiment is identical the one above, but instead of having babble noise, we will test with “moving train” noise.

For the experiment, the LSD score (avg.), and an example ground truth spectrogram and the output spectrogram are given as follows:

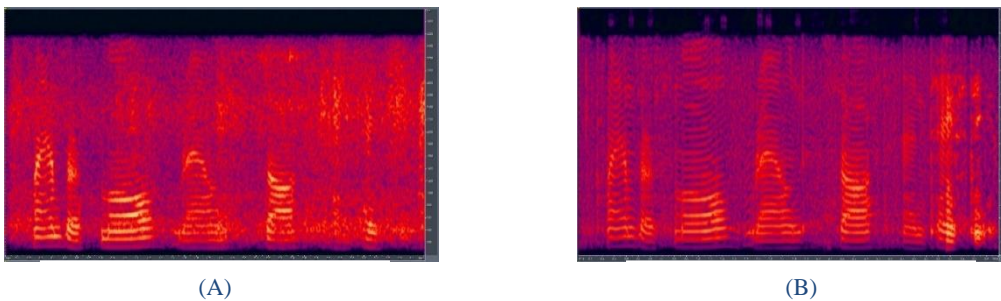


Fig. 4.9 (A) GT; (B) CUT output

LSD: 8.07

In addition to the experiments conducted above, we also test CUT against presence of two kinds of digital noises. These noises are added digitally to spectrograms of clean audio files. The purpose of these experiments was to determine if CUT could replicate any visual distortion, the details and the results of the experiments are discussed below:

➤ **Salt and Pepper (SnP):**

Salt-and-pepper noise, or impulse noise, is caused by sharp and sudden disturbances in the image signal. It can be thought of as sparsely occurring white and black pixels. Fig. 4.10 illustrates a spectrogram that is infused with salt-and-pepper noise randomly.

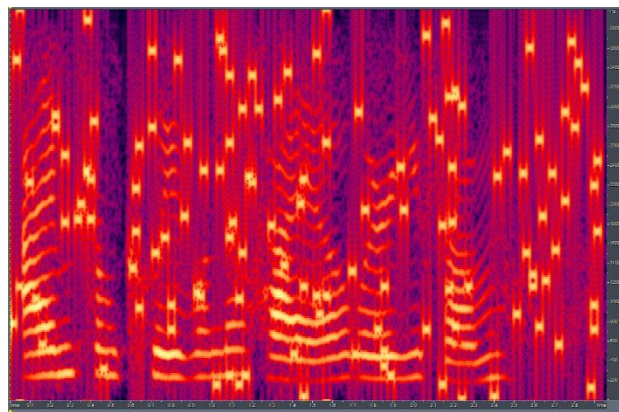


Fig. 4.10 A spectrogram infused with SnP noise.

Here, the setup follows the same trend as in previous experiments. It can be summarized as:

Domain A – Clean Audio Samples

Domain B – Clean Audio Samples + SnP noise added to spectrograms.

Context – Different

The results of this experiment are as follows in Fig. 4.11:

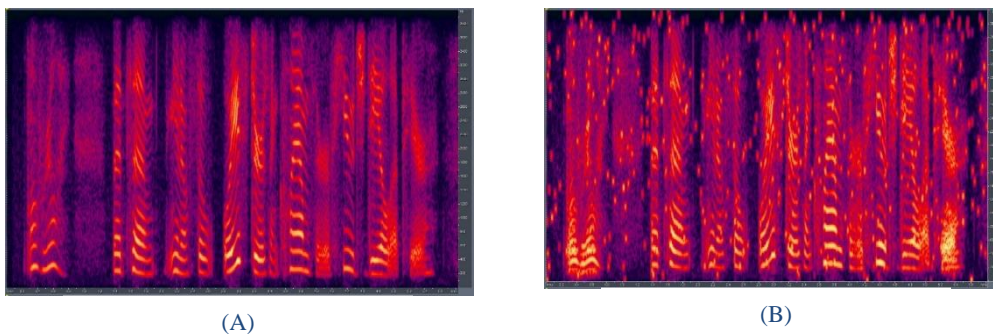


Fig. 4.11 (A) Input Spec; (B) CUT output

➤ **Speckle:**

Speckle noise arises due to environmental conditions affecting the imaging sensor while acquiring an image. It can also be described as granular noise that exists in an image and degrades its quality. It is generated by multiplying random pixel values with different pixels of an image. Fig. 4.12 illustrates a spectrogram that is infused with speckle noise.

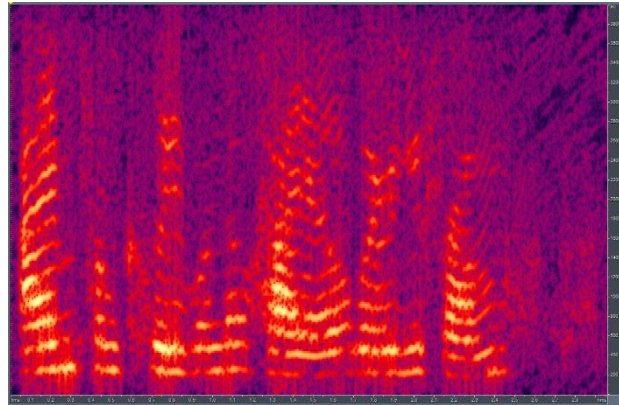


Fig. 4.12 A spectrogram infused with Speckle noise.

Here, the setup follows the same trend as in previous experiments. It can be summarized as:

Domain A – Clean Audio Samples

Domain B – Clean Audio Samples + Speckle noise added to spectrograms.

Context – Different

The results of this experiment are as follows in Fig. 4.13:

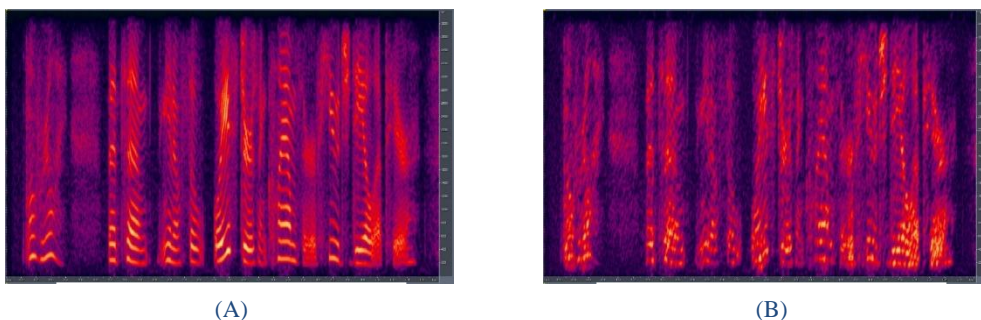


Fig. 4.13 (A) Input Spec; (B) CUT output

As is evident from the results above, CUT was successful in learning the digital noise characteristics.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

We began this project with one goal in mind: To come up with a system that can be used for data augmentation of speech data, while treating it as an unsupervised problem. This problem of data augmentation was studied earlier by Google AI in [3], but they made use of hand-engineered policies, we wanted to explore the use of GANs as they have seen an upsurge of work put into them with the newer architectures outputting extraordinary results. We wanted to utilize a suitable GAN for our use-case, which led us to experiment with SinGAN which put-forth an attractive claim of being able to train on a single training example. We worked on this architecture and extracted as much performance as possible before realizing the results obtained were not satisfactory. We then moved on to CUT, an unpaired image-to-image translation model that claimed to learn a mapping from one domain to another with nothing but unpaired training samples. This piqued our interest, and we began experimenting with CUT. The initial results with CUT were surprisingly good, and with a little tuning and tweaking of the model, we were sure we could come up with a viable solution to our problem.

Our efforts with CUT proved worthwhile when we were able to train a CUT model with less than 5 minutes' worth of data and produce satisfactory results, thanks to the one-to-many generation module that we fitted CUT with. Along with changes to the data-loader, we tweaked the internal representation of input images (in our case, spectrograms) so that once we generated the outputs, they can be reconstructed to audio files.

5.1 CONCLUSION

We now have a system that can be trained to learn a mapping from clean to noisy (stationary) spectrograms with less than 5 minutes' worth of data. Making this approach cost effective in terms of training data required, this is likely to be the most realistic scenario wherein we do not always have access to abundant data from one domain.

We started with SinGAN, took inspiration from its implementation where it uses a sliding window to scan over the single input image, allowing it to train the discriminator on the "patches" of the image. We implement this as a one-to-many generation module where we split an input spectrogram into components and treat them as individual samples. This has two benefits – we can make do with less training data and the aspect ratio of the spectrograms is preserved. This rendition of CUT has proven to be very effective in getting us the desired results, doing so with very little data.

In the end, we are left a robust image-to-image translation model that has more merits than demerits. The model is extremely light-weight, can generate samples at a very fast rate, and can be trained in less than 4 hours on a mid-tier GPU. When we put all these together, we can confidently say that the scope of applications of this model are abundant. Even with all these merits, the shortcomings of the model cannot be ignored. We discuss the range of problems with the model in the next section where we shed light on future work that can be conducted to further improve the performance.

5.2 FUTURE SCOPE

While we were more than happy with the results obtained from CUT, we believe it has some untapped potential that can be extracted. The ideas and approaches that we feel will further improve CUT's performance are detailed below:

We have been working on the magnitude spectrogram of the audio files; these magnitude spectrograms are obtained by separating the phase information from the complex spectrogram that is obtained by taking a Fast-Fourier transform. We are now looking at solutions that do not separate the phase information when generating the output, or in other words, when dealing with the complex spectrograms.

Another avenue for improving the performance is the inclusion of phase information in the loss function that is trained by the model. In its current implementation, we use the phase of the clean audio + the generated output (noisy) spectrogram to reconstruct the audio file. If we include phase information in the loss function of the model, we might be able to predict phase information for the output spectrogram, and the reconstruction would be better.

Lastly, one of the most crucial drawbacks of CUT is its inability to learn a mapping in the presence of non-stationary noise. The current implementation fails to accurately capture noises like babble (people talking in the background) and moving trains, this might be attributed to the fact that these kinds of noise are not captured properly in the spectrogram whilst noises like a constant gaussian noise, white noise, or any other stationary noise is seen in a spectrogram and is constant throughout the sample. One way to tackle this might be to delete segments in the training samples where the background noise falls below a certain threshold, this way CUT can learn a consistent mapping that is reproducible on, later, previously unseen clean samples.

REFERENCES

- [1] Tamar Rott Shaham, Tali Dekel, Tomer Michaeli; SinGAN: Learning a Generative Model from a Single Natural Image, Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 4570-4580
- [2] T. Park, A. A Efros, R. Zhang, J.Y. Zhu. Contrastive Learning for Unpaired Image-to-Image Translation, ECCV, 2020
- [3] Park, Daniel S. and Chan, William and Zhang, Yu and Chiu, Chung-Cheng and Zoph, Barret and Cubuk, Ekin D. and Le, Quoc V., SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition, Interspeech 2019, 2019
- [4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.
- [5] Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017 (* indicates equal contributions)
- [6] Leon A. Gatys and Alexander S. Ecker and Matthias Bethge, A Neural Algorithm of Artistic Style, 2015
- [7] Walker, Kevin, et al. RATS Speech Activity Detection LDC2015S02. Hard Drive. Philadelphia: Linguistic Data Consortium, 2015.
- [8] Hu, Y. and Loizou, P. (2007). "Subjective evaluation and comparison of speech enhancement algorithms," *Speech Communication*, 49, 588-601.
- [9] Abramson and I. Cohen, "Simultaneous detection and estimation approach for speech enhancement", *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 8, pp. 2348–2359, 2007.
- [10] Sue Grace, Phil Gravestock, *Inclusion and Diversity: Meeting the Needs of All Students*, Taylor & Francis, 2008
- [11] Ibo van de Poel, Lamber Royakkers, *Ethics, Engineering and Technology*, Wiley-Blackwell Publishers, May 2011.
- [12] *Engineering ethics in practice: a guide for engineers*, The Royal Academy of Engineering, August 2011 URL: <https://www.raeng.org.uk/publications/other/engineering-ethics-inpractice-full>

ANNEXURES

Full code available at: <https://github.com/shashankshirol/GeneratingNoisySpeechData>

I. ONE-TO-MANY GENERATION MODULE:

```
def split_and_save(spec, pow=1.0, state = "Train", channels = 1):
    """
        Info: Takes a spectrogram, splits it into equal parts;
        uses median padding to achieve this.
        Created: 13/04/2021
        By: Shashank S Shirol
        Parameters:
            spec - Magnitude Spectrogram
            pow - value to raise the spectrogram by
            state - Decides how the components are returned.
    """

    fix_w = 128 # because we have 129 n_fft bins; this will
    result in 129x128 spec components
    orig_shape = spec.shape

    ##### adding the padding to get equal splits
    w = orig_shape[1]
    mod_fix_w = w % fix_w
    extra_cols = 0
    if(mod_fix_w != 0):
        extra_cols = fix_w - mod_fix_w

    #making padding by repeating same audio (takes care of edge
    case where actual data < padding columns to be added)
    num_wraps = math.ceil(extra_cols/w)
    temp_roll = np.tile(spec, num_wraps)
    padd=temp_roll[:, :extra_cols]
    spec = np.concatenate((spec, padd), axis=1)
    #####
    spec_components = []
    spec = functions.power_to_db(spec**pow)
    X, X_min, X_max = functions.scale_minmax(spec, 0, 255)
    X = np.flip(X, axis=0)
    np_img = X.astype(np.uint8)

    curr = [0]
    while(curr[-1] < w):
        temp_spec = np_img[:, curr[-1]:curr[-1] + fix_w]
        rgb_im = functions.to_rgb(temp_spec, chann = channels)
        img = Image.fromarray(rgb_im)
        spec_components.append(img)
        curr.append(curr[-1] + fix_w)
```

```

    if (state == "Train"):
        return spec_components if extra_cols == 0 else
spec_components[:-1] # No need to return the component with
padding.
    else:
        return spec_components # If in "Test" state, we need all
the components.

```

II. SPECTROGRAM EXTRACTION

```

STANDARD_LUFS = -23.0
def extract(filename, sr=None, energy = 1.0, hop_length = 64,
state = None):
    """
        Extracts spectrogram from an input audio file
        Important Arguments:
            filename: path of the audio file
            n_fft: length of the windowed signal after padding
with zeros.
    """
    data, sr = librosa.load(filename, sr=sr)
    data *= energy

    ##Normalizing to standard -23.0 LuFS
    meter = pyln.Meter(sr)
    loudness = meter.integrated_loudness(data)
    data = pyln.normalize.loudness(data, loudness,
target_loudness = STANDARD_LUFS)
    #####

    comp_spec = librosa.stft(data, n_fft=256, hop_length =
hop_length, window='hamming')

    mag_spec, phase = librosa.magphase(comp_spec)

    phase_in_angle = np.angle(phase)
        return mag_spec, phase_in_angle, sr

```

III. MODIFYING SPECTROGRAM REPRESENTATION

```
# To convert the spectrogram (an 2d-array of real numbers) to a
storable form (0-255)
```

```
def scale_minmax(X, min=0.0, max=1.0):
    X_std = (X - X.min()) / (X.max() - X.min())
    X_scaled = X_std * (max - min) + min
    return X_scaled, X.min(), X.max()
```

```
# To get the original spectrogram (an 2d-array of real numbers)
from an image form (0-255)
```

```
def unscale_minmax(X, X_min, X_max, min=0.0, max=1.0):
    X = X.astype(np.float)
    X -= min
    X /= (max - min)
    X *= (X_max - X_min)
    X += X_min
    return X
```

IV. ADD DIGITAL NOISE (SALT AND PEPPER)

```
def salt_and_pepper(img, prob = 0.005):
    """
        Infuses an input image with Salt and Pepper noise with a
        probability of P = prob;
        Arguments:
            img : input image, can be a PIL image.
            prob : 0.005 (default) probability with which the
        noise is infused
    """
    img = np.asarray(img)
    output = np.zeros(img.shape, np.uint8)
    thres = 1 - prob
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            ran = random.random()
            if ran < prob:
                output[i][j] = 0
            elif ran > thres:
                output[i][j] = 255
            else:
                output[i][j] = img[i][j]

    return Image.fromarray(output)
```

V. ADD DIGITAL NOISE (SPECKLE)

```
def speckle(img, prob = 0.1):
    """
        Adds speckle noise to an input image with probability P =
    prob;
        Arguments:
            img : input image, can be a PIL image.
            prob: 0.1 (default) probability with which the noise
    is added.
    """
    img = np.asarray(img)
    output = np.zeros(img.shape, np.uint8)
    thres = 1 - prob
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            ran = random.random()
            if ran < prob:
                output[i][j] = 128
                for k in range(5):
                    output[i - k][j - k] = 128 + 10 * ran
            else:
                output[i][j] = img[i][j]

    return Image.fromarray(output)
```

VI. LSD CALCULATION

A. TIME ALIGNMENT AND ENERGY EQUALIZATION

```
def AddNoiseFloor(data):
    """
        To prevent the log(0) computation, we add a small value
    to the frames.
    """
    frameSz = 128
    noiseFloor = (np.random.rand(frameSz) - 0.5) * 1e-5
    numFrame = math.floor(len(data)/frameSz)
    st = 0
    et = frameSz-1

    for i in range(numFrame):
        if(np.sum(np.abs(data[st:et+1])) < 1e-5):
            data[st:et+1] = data[st:et+1] + noiseFloor
            st = et + 1
            et += frameSz
    return data
```

```

def time_and_energy_align(data1, data2, sr):
    nfft = 256
    # hop_length = win_length or frameSz - overlaps
    hop_length = 1
    win_length = 256

    ##Adding small random noise to prevent -Inf problem with Spec
    data1 = AddNoiseFloor(data1)
    data2 = AddNoiseFloor(data2)

    ##Pad with silence to make them equal
    zeros = np.zeros(np.abs((len(data2)-len(data1))), dtype=float)
    padded = -1

    if(len(data1) < len(data2)):
        data1 = np.append(data1, zeros)
        padded = 1
    elif(len(data2) < len(data1)):
        data2 = np.append(data2, zeros)
        padded = 2

    # Time Alignment
    # Cross-Correlation and correction of lag using the
spectrograms
    spec1 = abs(librosa.stft(data1, n_fft=nfft,
hop_length=hop_length,
win_length=win_length, window='hamming'))

    spec2 = abs(librosa.stft(data2, n_fft=nfft,
hop_length=hop_length,
win_length=win_length, window='hamming'))

    energy1 = np.mean(spec1, axis=0)
    energy2 = np.mean(spec2, axis=0)
    n = len(energy1)

    corr = signal.correlate(energy2, energy1, mode='same') /
np.sqrt(signal.correlate(energy1,
energy1, mode='same')[int(n/2)] * signal.correlate(energy2,
energy2, mode='same')[int(n/2)])

    delay_arr = np.linspace(-0.5*n/sr, 0.5*n/sr,
n).round(decimals=6)

    #print(np.argmax(corr) - corr.size//2) no. of samples to move

    delay = delay_arr[np.argmax(corr)]

    if(delay*sr < 0):
        to_roll = math.ceil(delay*sr)
    else:
        to_roll = math.floor(delay*sr)

```

```

# correcting lag
# if both signals were the same length, doesn't matter which
one was rolled
if(padded == 1 or padded == -1):
    data1 = np.roll(data1, to_roll)
elif(padded == 2):
    data2 = np.roll(data2, -to_roll)

# Energy Alignment

data1 = data1 - np.mean(data1)
data2 = data2 - np.mean(data2)

sorted_data1 = -np.sort(-data1)
sorted_data2 = -np.sort(-data2)

L1 = math.floor(0.01*len(data1))
L2 = math.floor(0.1*len(data1))

gain_d1d2 = np.mean(np.divide(sorted_data1[L1:L2+1],
sorted_data2[L1:L2+1]))

#Apply gain
data2 = data2 * gain_d1d2

return data1, data2

```

B. LOUDNESS NORMALIZATION

```

def normalize(sig1, sig2):
    """sig1 is the ground_truth file
       sig2 is the file to be normalized"""

    # Here we can either normalize loudness to match the
reference file or -23.0 LUFS standard
    # In this case, we are matching it to the LUFS standard

    data1, sr1 = sf.read(sig1)
    data2, sr2 = sf.read(sig2)

    assert sr1 == sr2

    meter1 = pyln.Meter(sr1)
    meter2 = pyln.Meter(sr2)

    loudness1 = meter1.integrated_loudness(data1)
    loudness2 = meter2.integrated_loudness(data2)

```

```

    data2_normalized = pyln.normalize.loudness(data2, loudness2,
target_loudness=-23.0)
    data1_normalized = pyln.normalize.loudness(data1, loudness1,
target_loudness=-23.0)

    return data1_normalized, data2_normalized, sr1

```

C. LSD CALCULATION

```

def calc_LSD_spectrogram(a, b):
    """
    Computes LSD (Log - spectral distance)
    Arguments:
        a: vector (torch.Tensor), modified signal
        b: vector (torch.Tensor), reference signal (ground
truth)
    """
    if(len(a) == len(b)):
        diff = torch.pow(a-b, 2)
    else:
        stop = min(len(a), len(b))
        diff = torch.pow(a[:stop] - b[:stop], 2)

    sum_freq = torch.sqrt(torch.sum(diff, dim=1)/diff.size(1))

    value = torch.sum(sum_freq, dim=0) / sum_freq.size(0)

    return value.numpy()

```

PROJECT DETAILS

<i>Student Details</i>			
Student Name	Shashank S Shirol		
Registration Number	170905178	Section / Roll No	C / 27
Email Address	shashank.shirol1@gmail.com	Phone No (M)	+91 8788704268
<i>Project Details</i>			
Project Title	Generating noisy speech data from clean data in the frequency domain using Deep Learning Methods		
Project Duration	6 Months	Date of reporting	04/01/2021
<i>Organization Details</i>			
Organization Name	Nanyang Technological University, Singapore		
Full postal address with pin code	50 Nanyang Ave, Singapore 639798		
Website address	https://www.ntu.edu.sg/Pages/home.aspx		
<i>External Guide Details</i>			
Name of the Guide	Dr. Chng Eng Siong		
Designation	Assoc. Professor		
Full contact address with pin code	School of Computer Science and Engineering Nanyang Technological University (NTU), 50 Nanyang Ave, Singapore 639798		
Email address	ASESChng@ntu.edu.sg	Phone No (M)	
<i>Internal Guide Details</i>			
Faculty Name	Dr. Muralikrishna SN		
Full contact address with pin code	Dept of Computer Science & Engg, Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	murali.sn@manipal.edu		