*Dissertation on*

# "Noise2Voice: Speech Denoising without Clean Training Audio"

*Submitted in partial fulfilment of the requirements for the award of degree of*

**Bachelor of Technology**
**in**
**Computer Science & Engineering**

**UE17CS490B – Capstone Project Phase - 2**

***Submitted by:***

| | |
|---|---|
| **Anuj Sanjay Tambwekar** | **PES1201700056** |
| **Madhav Mahesh Kashyap** | **PES1201700227** |
| **Krishnamoorthy Manohara** | **PES1201700636** |

*Under the guidance of*

**Dr. S Natarajan**
Professor, Department of CS&E
PES University

**January - May 2021**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
FACULTY OF ENGINEERING
**PES UNIVERSITY**
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

**FACULTY OF ENGINEERING**

# CERTIFICATE

*This is to certify that the dissertation entitled*

**‘Noise2Voice: Speech Denoising without Clean Training Audio’**

*is a bonafide work carried out by*

| | |
|---|---|
| **Anuj Sanjay Tambwekar** | **PES1201700056** |
| **Madhav Mahesh Kashyap** | **PES1201700227** |
| **Krishnamoorthy Manohara** | **PES1201700636** |

in partial fulfilment for the completion of eighth semester Capstone Project Phase - 2 (UE17CS490B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2021 – May. 2021. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

| | | |
|---|---|---|
| Dr. S Natarajan | Dr. Shylaja S S | Dr. B K Keshavan |
| Professor, Dept. of CS&E | Chairperson | Dean of Faculty |

**External Viva**

| **Name of the Examiners** | **Signature with Date** |
|---|---|
| **1.** ____________________________ | ____________________________ |
| **2.** ____________________________ | ____________________________ |

# DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **“Noise2Voice: Speech Denoising without Clean Training Audio”** has been carried out by us under the guidance of Dr. S Natarajan, Professor, Department of Computer Science and Engineering, and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

| | | |
|---|---|---|
| **PES1201700056** | **Anuj Sanjay Tambwekar** | [illegible] |
| **PES1201700227** | **Madhav Mahesh Kashyap** | MadhavMK |
| **PES1201700636** | **Krishnamoorthy Manohara** | Krishna |

# ACKNOWLEDGEMENT

We would like to express our gratitude to Dr. S Natarajan, Professor, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE17CS490B - Capstone Project Phase – 2.

We are grateful to the project coordinators, Prof. Silviya Nancy J and Prof. Sunitha R, for organizing, managing, and helping with the entire process.

We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from the department. We would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing us various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

# ABSTRACT

Speech denoising is a challenging task aimed at removing noise from audio samples in order to make speech more intelligible and easier to understand. With the surge in remote working and increased virtual communication, it is imperative to ensure high quality speech can be delivered to listeners with minimum distortion. However, traditional signal processing approaches are not very efficient owing to their ability to remove only pre-defined noise patterns. This problem has been solved using deep learning approaches, but they come at the cost of being very difficult to train, as it requires an extremely large amount of noisy and clean target audio pairs for training. Obtaining these clean audio samples involves a large economic investment for recording equipment such as microphones and soundproof studios, along with a large time investment in order to ensure the clean samples are completely free of noise.

In this project, we show the efficacy of a new state of the art audio denoising method called Deep Complex U-Net that denoises audio by viewing the waveform as an image, and propose a new training regime based on the well-researched Noise2Noise technique that illustrates that clean audio samples need not be required in the training process at all. We show this by creating a pipeline, trained only on noisy data that can denoise both artificially generated white noise and real-world noise such as cars and dogs. We also provide extensive metrics and comparisons to show that there is no performance degradation by using this approach, but rather that networks trained in this manner are more generalizable and hence perform better on complex real-world noises. Finally, we host this pipeline on the cloud as a web application, to allow for users to clean speech audio with minimal effort.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

Speech denoising is the process of removing unwanted signal (termed noise), from a sample of human speech with the intent of making the speech easier to understand and listen to. With the current surge in virtual conferencing and communication, it has become imperative for telecommunications software to provide some form of speech denoising in their services so as to increase user satisfaction.

When viewed programmatically, audio is simply one-dimensional time series data, representing amplitude at various intervals of time. As shown in Fig. 1, the goal of speech denoising is to obtain a waveform as close to the original clean audio sample. Traditional speech denoising approaches work under the assumption that the noise distribution is well defined. They attempt to remove the noise by adding another signal to the input that is the estimated negation of the noise distribution present in the speech. This method is the simplest, and as such, is the least feasible to use in real-world scenarios, as very few noises in the real world have this kind of predictable nature.

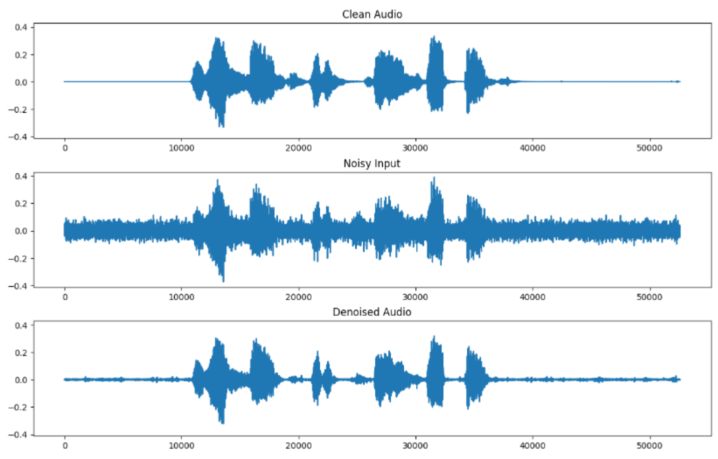


*Fig 1.1: A visualization of the speech denoising process*

Broadly speaking, noises come from two main sources. The first is due to the issues with equipment. In this case, the noise is introduced into the signal due to cheap components or issues with recording or processing equipment, and mainly occurs due to electrical disturbances. This type of noise is random in nature, in terms of its waveform, while being predictable in terms of occurrence, as it is likely that all the samples recorded by this setup will contain the disturbance, just that the values of the disturbance waveform will be different across both. The second major type of noise is noise that is introduced externally. This can vary from a dog barking, to a plane taking off and is extremely unpredictable in nature, as they may occur randomly and without the recorder's knowledge. However, the noise itself is less random in nature, as each different cause of noise will have its own set of distinct patterns and properties.

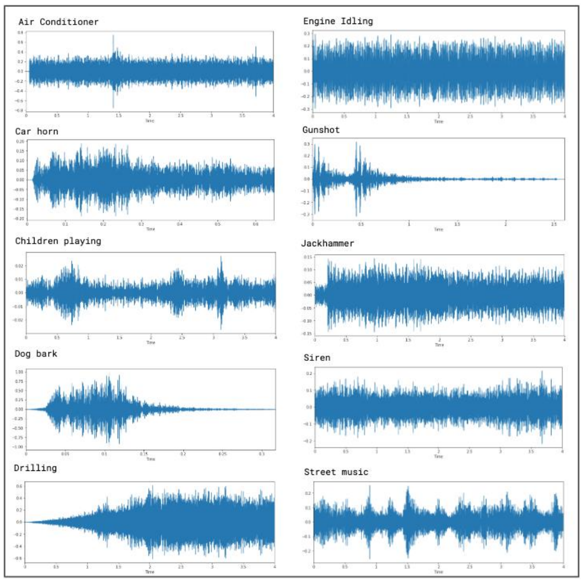


*Fig 1.2: Waveforms of various noise samples from the Urbansound Dataset*

Traditional methods for noise removal have now been outclassed by deep neural networks, as these models are able to identify the category of noise and estimate which components are actually parts of the speech signal. These methods are of the form of encoder-decoder networks, wherein the encoder part acts as a compressor, that finds important feature from the audio, and the decoder acts as a decompressor and attempts to recreate the original audio from the features given it by the encoder.

The main drawback to this approach however is the effort needed to train these models. Traditionally, a lot of work has gone into obtaining the data to train these models, as there must be perfectly clean audio samples for each corresponding noisy sample. This in turn means that the samples must be recorded in a pristine noise-proof environment, with excellent equipment, so as to ensure no contamination from either noise type. However, this is an extremely expensive process, and for low-resource languages, this creates a huge obstacle.

In an attempt to tackle this program, we approach the training from an image restoration angle and show that it is feasible to train a model to achieve state-of-the-art performance without being exposed to any clean data in its training process. This is inspired by the new Noise2Noise image restoration technique, which is elaborated upon further in section 3.

To show this, we create a pipeline that is trained only on noisy data, and host it as a webservice in order for users to easily clean their audio files. We also provide an extensive overview of the performance evaluation metrics, and show that we have no significant performance degradation despite never using clean data in our training, and instead see improved results in cases with complex noises. We believe that showing this approach is successful will encourage data collection in low-resource language speaking regions and will reduce the barrier of entry to create models for these languages.

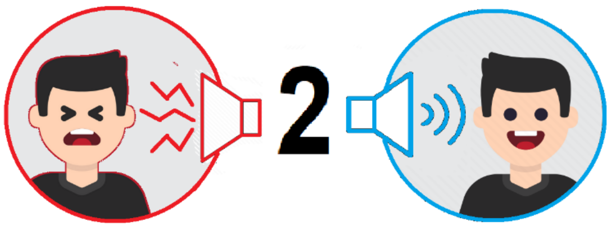


*Fig 1.3: The Noise2Voice application's logo*

# CHAPTER 2

# PROBLEM DEFINITION

## 2.1 Overview

With the need for deep networks to denoise audio established, along with their drawbacks in terms of requirements of extensive amounts of clean data, we wish to show that a training regime, comprised of solely noisy data is sufficient to create a network that performs extremely well.

## 2.2 Objective

The objective of this project is twofold. First, we prove that we can build a robust speech denoising pipeline whose core is a deep neural network trained only on noisy data. By showing that the data need not be perfectly clean, and needs to meet only one core requirement, we incentivise the collection of noisy data for low-resource languages, enabling us to create new models for these underrepresented languages.
The second objective is to allow anyone to enhance their audio files with minimum effort. We do this by creating a web app that feeds into our pipeline.

## 2.3 Scope

Our model is still reliant on being exposed to various noise types, and as such, we will be focusing our efforts on denoising a specific set of noise types. These noise types are those which can be easily generated, or come from the Urbansound 8K dataset [12]. A comprehensive list of every noise type we target is given in Table 2.1.

| No. | Noise Type | Source |
|---|---|---|
| 1 | White Noise | Synthesized |
| 2 | Mixed Noise | All Urbansound Classes |
| 3. | A/C in the background | Urbansound Class 0 |
| 4. | Car horns | Urbansound Class 1 |
| 5. | Children playing | Urbansound Class 2 |
| 6. | Dog’s barking | Urbansound Class 3 |
| 7. | Drilling | Urbansound Class 4 |
| 8. | Car Engine Idling | Urbansound Class 5 |
| 9. | Gunfire | Urbansound Class 6 |
| 10. | Jackhammer and construction | Urbansound Class 7 |
| 11. | Police and Ambulance sirens | Urbansound Class 8 |
| 12. | Background music and street noise | Urbansound Class 9 |

*Table 2.1: List of noise Types targeted*

The degree of noise in the signal is measured using the Signal-to-noise ratio (SNR). A negative SNR corresponds to a case where there is more noise volume than speech signal volume. Meanwhile a positive SNR indicates more speech than noise. Recent studies [3, 4] indicate that the average noise a listener experiences averages to about 0-10 SNR. To create challenging samples, we overlayed noises on the clean data, such that the average SNR for the noise file, prior to repeating the noise is 5db. We then repeat the scaled noise throughout the length of the audio file.

Each processed file should be its own standalone segment of speech. This is because the network can process roughly 5 seconds of speech at a time. In order to process larger files, we must sample the file in to multiple slices and stitch the cleaned slices together to create a clean file. If there is discontinuity in the original file, the stitching may not occur optimally.

## 2.4 Outcomes

The outcome of this project will be the creation of a pipeline capable of denoising speech that was never exposed to any clean audio samples. We additionally provide a set of exhaustive evaluations and scores to show the efficacy of this training regime and network architecture.
The end outcome as a product, would be an easy-to-use web application that allows users to upload their files, which is automatically cleans and then returns.

# CHAPTER 3

# LITERATURE SURVEY

## 3.1 Evaluation Techniques

Without first establishing methodology to evaluate denoising performance, we are unable to actively report how well our network is able to do its job, and what possible performance sacrifice occurs when we use only noisy training samples as targets. While MSE is often used as an evaluation technique [1] shows how MSE is a misleading metric for image denoising. Since our approach to denoising audio involved viewing the problem from an image denoising viewpoint we shall not be using MSE in our evaluation. The very fact that we can introduce 0 mean white noise and still have no change in MSE indicates it is a flawed technique.

Audio quality can also be viewed from a subjective background with SIG, BAK and OVL [5] being the most common ones. They measure signal distortion, background intrusiveness and overall quality respectively.

While SNR is also a valid method of measurement, it does not take into account distortion at specific locations in the signal, which the metric Segmented SNR (SSNR) is able to do [2]. One of the most powerful and widely used new evaluation metrics is PESQ [3, 4]. In addition to PESQ, a series of 3 composite metrics – CSIG, CBAK and COVL provide an objective version of the previously mentioned subjective evaluation methods [4, 6].

In addition to these techniques, a brand-new measurement known as STOI (Short Term Objective measurement of Intelligibility) [7] acts as an efficient way of recording denoising performance while keeping in mind the ability to retain speech quality over short durations of time. This metric was designed to evaluate techniques which involve speech separation and time-frequency weighing.

For this work, we report the SNR, SSNR, PESQ, CSIG, CBAK, COVL and STOI values for our results and we consider the PESQ score to be the main score of importance.

## 3.2 Datasets

Any model can only be as good as the data it trains on. As such the datasets selected are of utmost importance. While there exist, multiple datasets aimed at natural language generation and speech to text [8, 19, 20], few are explicitly designed for audio denoising. We selected University of Edinburgh's Noisy speech database for training speech enhancement algorithms [9]. A new upcoming dataset in this domain is the Microsoft Scalable Noisy Speech Dataset (MS-SNSD) [10], however most literature uses [9] and reports metrics on it and hence we use the same. Additionally, it possesses over 11,000 speech samples for training. Our technique is heavily reliant on the size of the training data, as described in the theoretical justification in Chapter 4.
While [9] does possess noisy samples in the data as well, the noise is generated artificially, or using the DEMAND dataset [11]. The DEMAND dataset has ambient noise based on the location of the audio recordings and not of the individual types of noise. Instead, we use the Urbansound 8K noise dataset [12], which consists of recordings from the environment which are also classified. This allows us to pinpoint noise categories that are difficult to tackle as well.

## 3.3 Image and Audio Denoising Techniques

The seminal paper we base this work on is Noise2Noise [13]. In it, the authors establish that it is possible to restore images by only looking at corrupted images. They go on to show that this is trivial statistically, as the nature of minimization of L1 and L2 losses means that the network tends to learn the average/median target distribution of the samples. As such, there are only two requirements that the target must have – the mean of the noise across the targets should be 0, and there should be little to no correlation between the noise distributions of the input and target, as if the latter were true, the network would be able to simply transform the noise from one type to the other. Results showed that training using Noisy targets had equivalent performance to training with Clean targets. This technique worked for various types of noise types including Poisson, Gaussian and Bernoulli.

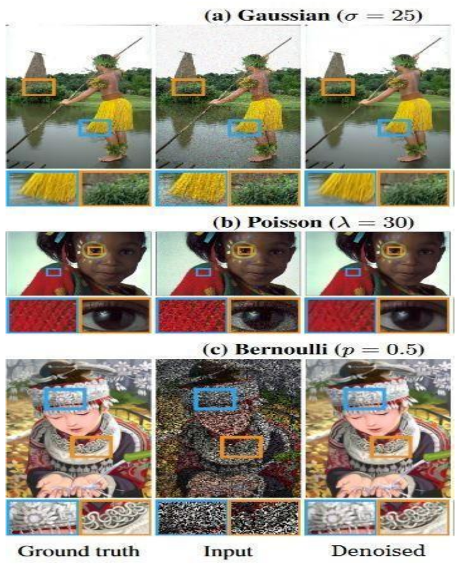

(a) Gaussian ($\sigma = 25$)
(b) Poisson ($\lambda = 30$)
(c) Bernoulli ($p = 0.5$)
Ground truth
Input
Denoised

*Fig 3.1: Noise2Noise [13] image denoising for Gaussian, Poisson and Bernoulli noise types.*

Denoising audio has been solved for many decades with moderate success by using traditional statistical methods. These however require access to clean targets to train with, and produce low quality denoised audio with many audible artifacts.

Discrete Wavelet Transform is one such method discussed in [14]. It is focused on audio signals corrupted with white noise. The authors used Discrete Wavelet Transform (DWT) to transform noisy audio signals in wavelet domain. It was assumed that signal is represented by high amplitude DWT coefficients and noise is represented by low amplitude coefficients. To get audio signal with less noise, thresholding of coefficients is used and they are transformed back to time domain. The authors proposed modified universal thresholding of coefficients which results with better audio signal. Error estimation is achieved using a ratio between square root energy of difference between original and denoised signal and square root energy of added noise (MSE). Another statistical method called Greedy Time Frequency Shrinkage is detailed in [15]. It uses Matching Pursuit, a greedy algorithm. Matching Pursuit (MP) is a greedy algorithm that iteratively builds a sparse signal representation. A residual signal is initialized R0 = y. At each iteration, the time-frequency atom with the greatest correlation with residual signal is assigned a weight, which is then shrunk or attenuated based on a hard threshold. If the threshold is passed, it is subtracted from the residual signal and moves on to the next iteration. Process continues until a stopping condition is met. [16] describes another statistical method, namely Adaptive Block Attenuation. The basic motivation of the block attenuation is that, if neighbouring coefficients contain some signal, then it is likely that the coefficients of current interest also do, and consequently a lower threshold should be used. A modification of Discrete Wave Transform (DWT) wherein signal is divided into blocks of size b, and each block has a different threshold calculated over a slice of the noisy signal of size 2b, using the same methods as DWT. If noise is spread across multiple frequencies, different amplitude thresholds can be set for different frequencies based on frequencies in neighbouring signal blocks.

## 3.4 Deep Learning for Audio Denoising

Deep learning-based methods have been shown to outperform the performance of statistical methods described above, if given a large and representative training dataset. We can represent our Audio denoising problem in the form of an Image denoising problem by converting the Audio into a Mel frequency-amplitude-time Spectrogram. A common trend has been to use a combination of convolutional layers and skip connections – techniques predominantly used in the image denoising space, to denoise the spectrograms of the speech signals, which can then be recombined to generate the original speech signal [21, 22, 23, 24].

This brings us to the deep learning architecture model which we will be using for our implementation. The Deep-Complex-U-Net (DCU-Net) architecture [17] was presented at ICLR 2018. It is the current state of the art for denoising tasks on the VoiceBank + DEMAND dataset combination. The model is based upon the already very successful Wave-U-Net architecture [18]. The Wave-U-Net is a Symmetric Encoder Decoder Convolution Neural architecture built originally for medical image segmentation tasks. The DCU-Net adds Skip connections as well as Complex valued Neurons to the Wave-U-Net. The input weights and threshold activation functions are complex valued. Even the inputs and outputs produced by the network are complex valued. The initial conversion of the Waveform representation to the Spectrogram representation uses Short Time Fourier Transforms which internally use transformations in the Complex and Imaginary number space. They go on to prove that Complex valued neural networks work better than Real valued neural networks in denoising this Spectrogram. Specifically, the phase and amplitude prediction in audio signal tasks are improved. Additionally, there are symmetric Skip connections between each equally sized Encoder and Decoder layers. The Skip Connections add/concatenate the outputs from previous layers to the outputs of stacked layers. This results in the ability to train much deeper networks than what was previously possible, without increasing the number of layers.

In addition to these techniques, recent work [25, 26] sheds light on the benefits of including noisy data in the training regime, leading to more generalized networks. These techniques have shown that noisy data can improve the performance of networks trained on clean data, or networks that are self-supervised. However, our work shows that no clean data is required outright, provided the quantity of data is sufficient

## 3.5 Summary and Discussion of Literature Survey

Traditional statistical techniques achieve decent results for generalized denoising. Deep learning approaches triumph in denoising audio signals in specific scenarios, namely if our training data contains numerous representative Clean targets of that specific Speech and Noise distribution.

We aim to provide a novel technique that can work when clean targets do not exist, and when we do not know details about the Noise distributions. We use the current state of the art DCUNet architecture for this purpose. The standard dataset for clean voices will be the VoiceBank dataset, and the standard dataset for noise distributions will be the Urbansound dataset.

# CHAPTER 4

# THEORETICAL JUSTIFICATION

In this section, we lay down the theoretical justification as to why the proposed Noise2Noise (N2N) training methodology is a sound paradigm. Consider a Deep Neural Network (DNN) with parameters $\theta$, loss function $L$, input $x$, output $f_\theta(x)$, and target output $y$. The DNN learns to denoise the input audio by solving the optimization problem shown in Equation 1 below:

$$\{L(f_\theta(x), y)\}$$

Consider $x_1$ and $x_2$ to be two noisy audio samples, made by overlaying a clean audio sample $y$ with noises sampled from distributions $N$ and $M$ respectively. That is,

$$x_1 = y + n \sim N \; and \; x_2 = y + m \sim M$$

The two requirements of these noise distributions are:

1. The mean of the noise is zero – a requirement that is true for most audio noise. All the noise categories in Urbansound possess this requirement

   $$\mu(N) = 0 \text{ for all noise categories N}$$

2. The correlation between $N$ and $M$ is close to zero – this ensures the network does not learn a mapping from one noise type to the other

   $$R(N, M) \approx 0$$

Traditional Noise2Clean (N2C) deep denoising approaches use the clean audio $y$ in the target. N2C networks that use $L_2$ losses, will hence aim to target the optimization problem below:

$$argmin_\theta \; L_{2,n2c} = argmin_\theta \; E_{(x_1,y)} \, \{(f_\theta(x_1) - y)^2\}$$

Our N2N approach, does not use clean audio in the target, and uses noisy audio instead. As such, the $L_2$ loss for the N2N case becomes

$$L_{2,n2n} = E_{(x_1,x_2)}\,\{(f_\theta(x_1) - x_2)^2\}$$

$$= E_{(x_1,x_2,m\sim M)}\,\{(f_\theta(x_1) - (y + m))^2\}$$

$$= E_{(x_1,x_2,m\sim M)}\,\{(f_\theta(x_1) - y)^2\} - E_{(x_1,x_2,m\sim M)}\,\{2m(f_\theta(x_1) - y)\} + E_{m\sim M}\,\{m^2\}$$

Mathematically, the expectation of $m^2$ is equal to the variance of $m$ plus the square of the expectation of $m$, and hence

$$L_{2,n2n} = L_{2,n2c} + Var(m) + E_{m\sim M}\,\{m\}^2$$

Condition 1 however, ensures that $E_{m\sim M}\,\{m\} = 0$, and the variance of the sample distribution $Var(m)$ is equal to the variance of the population divided by the sampling size. Hence, as the size of the dataset grows, this term starts to diminish, and hence,

$$\lim_{|TrainingDataSet|\to\infty} L_{2,n2n} = L_{2,n2c}$$

In the same fashion, it can be proven that $L_1$ losses possess the same property

$$argmin_\theta\ L_{1,n2c} = argmin_\theta\ E_{(x_1,x_2)}\,\{|f_\theta(x_1) - x_2|\}$$

$$\lim_{|TrainingDataSet|\to\infty} L_{1,n2n} = L_{1,n2c}$$

# CHAPTER 5

# PROJECT REQUIREMENTS SPECIFICATION

## 5.1 Introduction

This section is the project requirement specification of the project Speech Denoising without Clean Speech. The goal of the project is to create an end-to-end speech denoising pipeline using a deep neural network that has not been trained on any clean speech samples, and to host this pipeline on a server to create a web-based application to allow users to clean the audio they upload.

## 5.2 Project Scope

The objective of this project is twofold. First, it is to prove that machine learning based speech denoising techniques can be trained without any clean audio samples. Second, it is to develop a POC application that can use a model trained in such a fashion to create an end-to-end cloud-based speech denoising tool, where users can upload recorded speech to get it cleaned.
The benefit of this system is that, one, it shows that expensive soundproof recording environments and equipment are not required to train deep speech cleaning networks. This makes it substantially easier to generate datasets for scarce languages, incentivizing creating speech datasets for a lot of Indian languages. Second, it provides an easy-to-use speech enhancement tool that anyone can use without prior audio mixing or cleaning expertise.

## 5.3 Project Perspective

A user uploads a file and then receives a cleaned file after processing. The user can choose between using a predefined network to remove a specific category of noise, or let the application handle it, in which case it uses a generalized denoiser to remove background noises.

### 5.3.1 Project Features

1. Targeted Removal: Users can specify what kind of noise they wish to remove – this is ideal when there is a single source of noise that the user has clearly identified
2. General Removal: This uses the generalized DCUNet20 background noise removal network and aims to remove all forms of disturbance that the network identifies

### 5.3.2 Operating Environment

The system will operate on a server hosted on a cloud server (either Azure or AWS, depending on rates), running Ubuntu 18.04. The server will have a GPU (Nvidia Tesla K80). The system will use CUDA 10.1 for hardware acceleration.

The system will be built using Python 3.7. The modules used will be Pytorch 1.6, Torchaudio 0.6 and NumPy. The email notification system will use Azure pipeline's Email notification feature if hosted on Azure, or will send requests via Python's requests library

### 5.3.3 Constraints and Dependencies

The platform will run on a GPU keeping CUDA acceleration in mind. As such, only NVIDIA GPUs may be used, and CUDA accelerated versions of Pytorch and associated libraries must be used.

The model will be trained using the Edinburgh Speech Dataset, with real world noise distributions coming from the UrbanSound8K noise corpus. The assumption is that both datasets are an accurate reflection of speech data and noise data. While this is backed by literature, there is no data to show it is sufficient for an Indian scenario.

An additional assumption is that the speech to be processed is more powerful than the noise, IE the SNR is positive.

Users will only be allowed to upload audio till a certain maximum duration, depending on the available server resources. Currently, the suggested limit is 5 minutes in duration.

The pipeline we create will have the capability of removing 11 distinct types of noise, 10 that we can obtain from the dataset - A/C noise, car horns, children playing, dogs barking, drilling, engine idle sound, loud noises like gun shots, construction, sirens, background murmurs and white noise, which is normally due to poor quality microphones.

### 5.3.4 Risks

GPU Memory Bottleneck: As this is a deep learning model, we would be using a GPU based server. Multiple incoming requests can cause a large usage in GPU memory and so handling multiple concurrent users will be challenging.

File stitching: The deep learning model will process files in chunks depending on the available memory. The file must be sampled, cleaned and stitched back properly to prevent distortion.

## 5.4 Functional Requirements

1. The system must be able to convert standard audio file formats to wav
2. The system must be able to split a file into chunks for processing
3. The system must clean audio files by removing background noise belonging to one of the aforementioned classes
4. The system must be able to stitch chunks together to create a new file
5. The system must automatically initiate a download when the cleaning is complete
6. The system must be able to gracefully handle errors associated with the file, its processing and cleaning users and inform users of the same.

## 5.5 External Interface Requirements

### 5.5.1 Hardware Requirements

The application will be hosted on a server, and thus the user hardware requirements are very lightweight. The application will be able to run on any modern browser (Any version of Chrome, Edge or Firefox released from 2017 onwards).

The user must have a device capable of running any of the aforementioned browsers, along with an internet connection, either wired via LAN or wireless via 802.11 AC WiFi, that allows them to freely upload and download files. There is no minimum speed required, but users will have a better experience, the faster their network speed is.

### 5.5.2 Software Requirements

The project will be built using Python ver. 3.7. with Pytorch, NumPy, Torchaudio and Scipy being the main libraries. The server running the web application will be running Ubuntu 18.04
The datasets used for training the model are the Edinburgh Speech Dataset, and the Urbansound8K Noise dataset. The web application will be hosted on Flask.

### 5.5.3 Communication Interfaces

The connection between the client and the platform will be established using HTTPS. The audio file will be transferred using FTP. The client will connect to the system via a standard LAN network or 802.11AC WiFi

## 5.6 Non-Functional Requirements

### 5.6.1 Performance Requirements

The application server must be able to handle large files by splitting them up.

For speech quality, we aim to target a narrow band PESQ score of 2.0 for the cleaned output speech we generate. This score can only be validated for a standardized test set where the true speech is available for reference. Our tests indicate we reach these scores on the testing data

### 5.6.2 Safety Requirement

The processed audio must remain available for a certain duration on the server until the user downloads it. The original and processed file must be stored redundantly so as to ensure that in the event of a server outage, the user need not upload his data to the server once again.

### 5.6.3 Security Requirement

As we deal with speech data, there must be no way to associate any of the audio files with the original uploader or speaker so as to ensure user privacy. The original audio should not be accessible to any user.

The audio files must be stored on the server only temporarily, and must be deleted after a certain time limit. These files should only be accessible by the original uploader until it is time for the file to be deleted.

The original audio should be deleted from the server once the file has been cleaned.

The application server should prevent users from overloading the system with requests by using a soft time out.

## 5.7 Other Requirements

- SNR Requirement: The audio to be cleaned must have an SNR greater than or equal to zero. Negative SNRs correspond to audio where there is more noise than actual speech.
- Individual files will be restricted to a certain max file size limit. Total files submitted by a user will also be subjected to a limit.

# CHAPTER 6

# SYSTEM REQUIREMENT SPECIFICATION

## 6.1 Hardware Requirements

- 1 or more Nvidia Tesla K80 or any equivalent CUDA acceleration supported GPU with more than 8GB of VRAM
- Intel i5 4670K CPU or greater
- 8GB or more RAM
- Additional blob storage space of at least 500 MB to house cleaned files

## 6.2 Software Requirements

- Ubuntu: Version 18.04 | Ubuntu is a Linux based open-source operating system. It is the supported operating system for all the components of this project.
- Python: Version 3.7 + | Python is a high-level programming language. This entire project has been built using Python and various libraries.
- CUDA: Version 10.01 | CUDA is a proprietary hardware acceleration library that allows for fast and parallel computations on GPUs. CUDA is used to accelerate our machine learning workloads.
- Pytorch: Version 1.6 | Pytorch is an open-source machine learning library for Python, and is used to develop, train and use our models.
- Torchaudio: Version 0.7 | Torchaudio is a wrapper for Pytorch that allows easy and fast computations on audio files.
- Scipy: Version 1.5 | Scipy is a library in Python for mathematical and scientific computation.
- Flask: Version 1.1 | Flask is a framework for Python used to serve web applications.

- PyPESQ: Version 1.0 | PyPESQ is an opensource implementation of the PESQ evaluation metric for speech quality measurement.
- Sox: Version 14.4.2 | Sox is an audio processing utility used to change sample rates, file types and more. It serves as the backend for Torchaudio on Linux.
- Librosa: Version 0.8.0 | Librosa is a python package used for audio processing.
- Matplotlib: Version 3.3.3 | Matplotlib is a library in Python used for plotting graphs and visualizations.
- Numpy: Version 1.19.1 | Numpy is the Python scientific library providing a high-performance multidimensional array object, and tools for working with these arrays.
- Gc: Version Python inbuilt library | Gc exposes the underlying automatic garbage collector, memory management mechanism of Python. Includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

## 6.3 Data Requirements

- Edinburgh Speech Dataset for Speech Denoising and Text to Speech tasks (28 speaker version)
- Urbansound 8K Noise Dataset

# CHAPTER 7

# SYSTEM DESIGN

## 7.1 High Level Application Design Overview

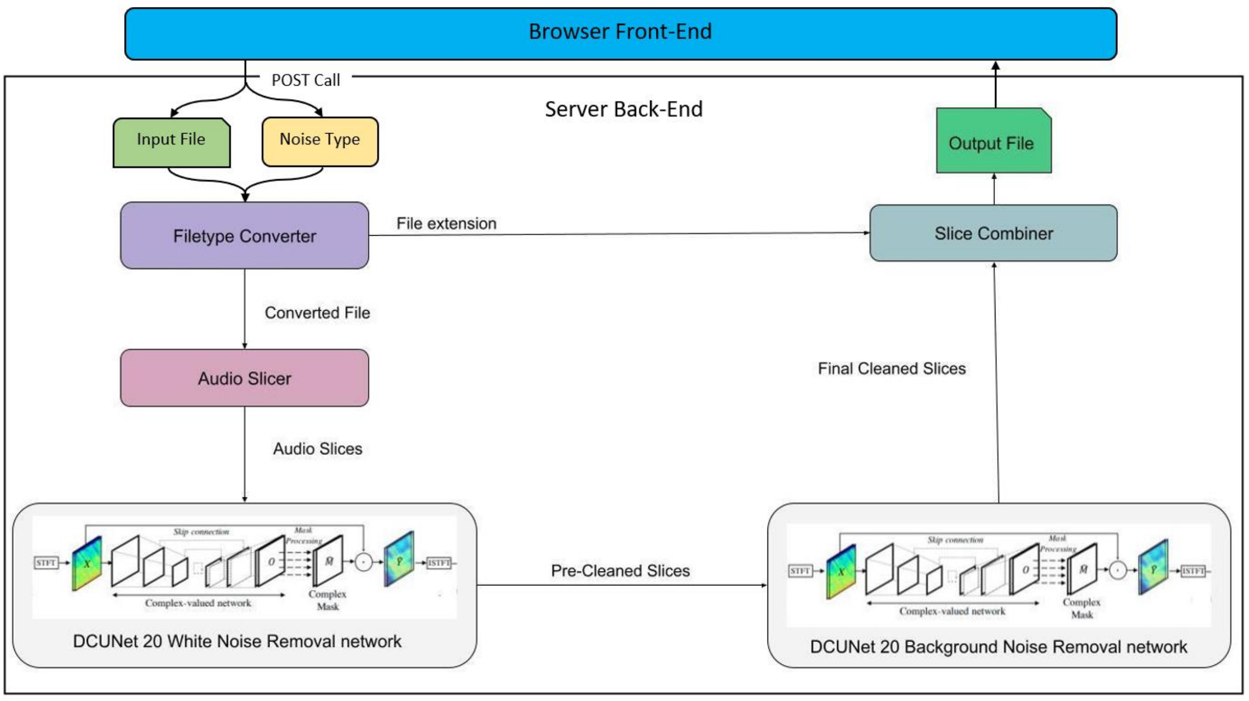


*Fig 7.1: The Audio Denoising Pipeline and its Components.*

We now describe each of the modules above in greater detail.

## 7.2 Detailed Low Level Design Overview

The audio denoiser pipeline consists of 5 main components. A converter to standardize file formats, an audio slicer to reduce the size of each chunk that is processed, a network to remove white noise from electrical interference, a background noise removal network and the slice combiner

Through our experimentation, we compared the efficacy of a combined model trained on multiple noise distributions and found that our approach allowed for a unified noise removal network to be trained much more efficiently, reaching PESQ NB scores of 2.11 on average. These numbers are similar to the standard PESQ NB numbers we obtain for various distributions, indicating that a single network is likely to be sufficient in order to denoise mixed distributions. Additionally, we trained classifiers to attempt to categorise various noise distributions, and found that even for the standalone noises we reached accuracies of only 80%. An incorrect classifier would therefore cause more harm than good, and hence we opt for the 2-network approach using a general denoiser network trained using the Mixed Urbansound noise category.

### 7.2.1 Filetype Converter Module

**Inputs:**

1.An audio file of any type (ogg, wav, mp3, flac)

**Outputs:**

1.The contents of the file after converting to a wav file

2.The original Filetype of the input file

**Description:**

The DCUNet model and all the required functionality associated with it uses Torchaudio and wavfile, both of which expect wav files as inputs. However, imposing this restriction on the user is limiting as files are usually recorded in formats like mp3. Adding this converter allows for more flexibility and ease of use for the user.

### 7.2.2 Audio Slicer Module

**Inputs:**

1.Contents of the input in wav format

**Outputs:**

1.Slices of the input wav data

**Description:**

The DCUNet model is designed for short clips of audio owing to memory constraints. To bypass this, we create short overlapping slices of the audio data of roughly 5 seconds, padded with a few milliseconds of silence on each end. This allows us to pass the individual slices of the audio for cleaning and then stitch them back into one file.

### 7.2.3 DCUNet-20 White Noise Removal Module

**Inputs:**

1.A list of Audio Slices

**Outputs:**

1.Audio Slices without White noise

**Description:**

The white noise removal module consists of a DCUNet-20 model trained to remove white noise. The objective here is to remove the white noise from the signal, and is done first because in theory, the white noise is the noise added by electrical and random disturbances after the sound has been captured by the microphone, and hence is the last noise added into the file. Hence to reverse the process the white noise should be removed first.

### 7.2.4 DCUNet-20 Background Noise Removal Module

**Inputs:**

1.A list of audio slices

2.A list of sets of noise categories

**Outputs:**

1.Cleaned Audio Slices

**Description:**

This network is another DCUNet-20 based denoiser. Here, for each slice, we use the corresponding noise category set. The corresponding pre-trained weights are loaded from the disk and used to clean the slice of that category. This is repeated for each noise type in the set, and done for all the slices to create a collection of cleaned audio slices. The noise categories are can be provided by the user, in case there is a requirement to remove a specific type of noise. Alternatively, the catch-all mixed generalized denoiser is used. This denoiser is trained across multiple noise categories, and as such, allows for the removal of multiple noise distributions with one pass

### 7.2.5 Slice Combiner Module

**Inputs:**

1.A list of Cleaned Audio Slices

2.Original file metadata

**Outputs:**

1.The cleaned audio file

**Description:**

This is the last module in the pipeline. The slices are stitched back together into one long audio segment, by removing the terminal silence sections on all but the beginning of the first and the end of the last slice. The file is then saved to the original file type using the metadata obtained from the audio slicer module.

### 7.2.6 Browser Front-End Module

**Inputs:**

1. User uploaded Noisy audio file (MP3 or WAV format)
2. Noise type present in audio file

**Outputs:**

1. User downloadable denoised audio file

**Description:**

The front-end webpage is written using the HTML markup language, styles using CSS, and scripted using JavaScript. It has a form in which user uploads his noised speech file and selects the type of noise present in it. The denoised file returned by the backend server is automatically downloaded.

# CHAPTER 8

# IMPLEMENTATION AND PSEUDOCODE

## 8.1 DCUNet20 Model

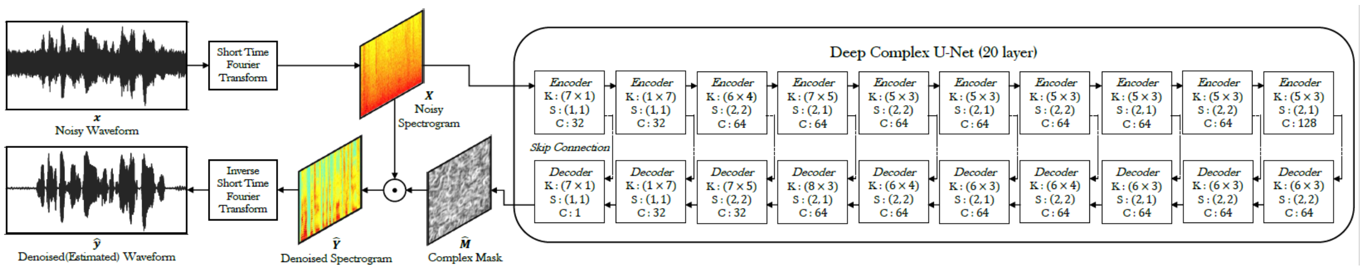

x
Noisy Waveform
Short Time Fourier Transform
X
Noisy Spectrogram
ŷ
Denoised(Estimated) Waveform
Inverse Short Time Fourier Transform
Ŷ
Denoised Spectrogram
M̂
Complex Mask
Deep Complex U-Net (20 layer)
Encoder K: (7 × 1) S: (1,1) C: 32
Encoder K: (1 × 7) S: (1,1) C: 32
Encoder K: (6 × 4) S: (2,2) C: 64
Encoder K: (7 × 5) S: (2,1) C: 64
Encoder K: (5 × 3) S: (2,2) C: 64
Encoder K: (5 × 3) S: (2,1) C: 64
Encoder K: (5 × 3) S: (2,2) C: 64
Encoder K: (5 × 3) S: (2,1) C: 64
Encoder K: (5 × 3) S: (2,2) C: 64
Encoder K: (5 × 3) S: (2,1) C: 128
Skip Connection
Decoder K: (7 × 1) S: (1,1) C: 1
Decoder K: (1 × 7) S: (1,1) C: 32
Decoder K: (7 × 5) S: (2,2) C: 32
Decoder K: (8 × 3) S: (2,1) C: 64
Decoder K: (6 × 4) S: (2,2) C: 64
Decoder K: (6 × 3) S: (2,1) C: 64
Decoder K: (6 × 4) S: (2,2) C: 64
Decoder K: (6 × 3) S: (2,1) C: 64
Decoder K: (6 × 3) S: (2,2) C: 64
Decoder K: (6 × 3) S: (2,2) C: 64

*Fig 8.1: Denoising transformations and DCUNet-20 architecture*

- The raw noisy Audio file is pre-processed before passing it onto the DCU-Net model. First the 'wav' encoded file is converted from an Amplitude v/s Time form into a Spectrogram. The Spectrogram converts the 2-D audio into a 3-D Spectrogram representation which depicts Time on the X-axis, Frequency on the Y axis, and Amplitude represented by a Colour scale (Z axis). Essentially, we have converted our Audio denoising problem into an Image denoising problem. The Pytorch function 'torch.stft()' performs a Short Time Fourier Transform converting the raw Numpy audio file into a Linear Scale Mel Spectrogram.
- This Spectrogram is passed as input to the trained 20 Layer Deep Complex U-Net architecture. This is a 20-layer symmetric Encoder Decoder architecture. A defining feature is that it uses Complex valued Convolution layers for the Encoder and Decoder sections. The basic building blocks of each Complex 2D Convolution layer is both a Real valued convolution layer and an Imaginary valued convolution layer. These real and imaginary portions are objects of the 'torch.nn.CConv2D' Class, and are stacked on top of each other using 'torch.stack()'.

- Each Encoder layer down samples the inputted spectrogram dimensions, using the trained weights. Each Decoder up samples the already down sampled spectrogram. There are Skip connections present symmetrically between each equally sized Encoder and Decoder layers. These concatenate the tensors from the Encoder part with the up sampled tensors received from the previous Decoder. The function ‘torch.cat()’ is used for this concatenation operation.
- The output of the Decoder layers is of the same dimensions as the original input to the Encoder since the layers up sample and down sample with equal magnitude. This output Spectrogram represents the cleaned Audio without noise. Conversion from the 3D Spectrogram to the raw 2D Audio file is done by using an Inverse Short Time Fourier Transform, ‘torch.istft()’.
- The technique for training the model was unique since we have no ‘Clean’ audio targets to train with. Instead, we use pass a pair of Noisy audio files containing the same underlying Clean speech. The model learns to convert a noisy instance of speech to another noisy instance of the same speech! However, when repeated over thousands of audio pairs, the function learnt is to convert from a Noisy speech to the Clean speech! We pass 4 audio training pairs per batch (batch size is 4). We also use an Adam optimizer ‘torch.optim.Adam()’, with a learning rate of $1*10^{-4}$. Empirically we have seen that 5 epochs, or fewer is sufficient to get maximum PESQ/SSNR audio quality metrics over the validation set, without overfitting on the trainset.  After training finishes, we save the model and optimizer weights to a ‘.pth’ file using the ‘torch.save()’ function.
- The noisy training data is generated by adding noise (from either Urban8K or random White noise) to clean speech (from VoiceBank).  The Decibel level of noise is randomly chosen between [0, 10], to simulate real world scenarios where we do not know the true amplitude of the noise. This technique is called ‘blind denoising’.
- The Audio quality metrics such are PESQ, SSNR STOI are calculated on the model Denoised audio relative to the Clean audio. The implementation of the various quality metrics is as referenced in the Literature survey section.

- The figures 8.2, 8.3 and 8.4 below demonstrate our model attempting to denoise a White-noise corrupted speech sample. As evident by the waveforms visually, we see that our model performs denoising exceedingly well.

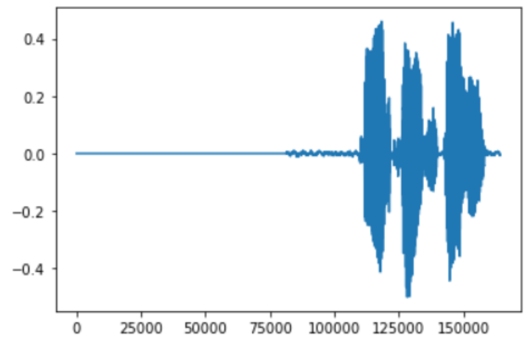


*Fig 8.2: Example Original Speech Waveform*

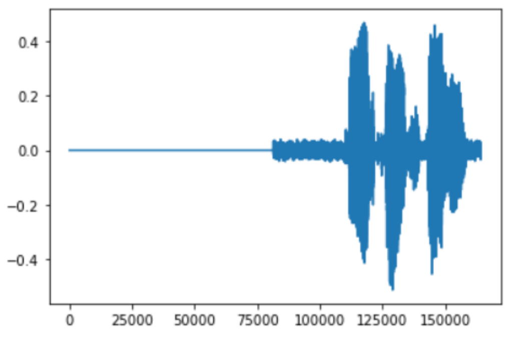


*Fig 8.3: Example Noisy Speech Waveform*

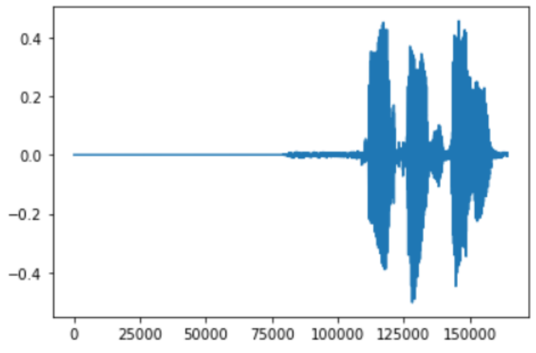


*Fig 8.4: N2N Denoised Waveform of the same example*

There are three key algorithms used in the denoising pipeline. All mentions of the DCUNet model use the network architecture explained earlier.

## 8.2 Denoising Pipeline Pseudocode

### 8.2.1 Speech Denoising Algorithm

**Algorithm 1** Speech Denoiser

```
    Input input_speech, category (Optional)
    Output denoised_speech
 1: wav_audio, sample_rate, original_filetype $\leftarrow$ convert(input_speech)
 2: audio_splits $\leftarrow$ split_audio(wav_audio)
 3: white_cleaned_splits $\leftarrow$ []
 4: full_cleaned_splits $\leftarrow$ []
 5: dcunet.load_weights(class = 'white')
 6: for each split $\in$ audio_splits do
 7:     white_cleaned_splits.append(dcunet.clean(split))
 8: if category $\neq$ NULL then
 9:     dcunet.load_weights(class = category)
10: else
11:     dcunet.load_weights(class = 'mixed'))
12: for each split $\in$ white_cleaned_splits do
13:     full_cleaned_splits.append(dcunet.clean(split))
14: recombined $\leftarrow$ combine_slices(full_cleaned_splits)
15: denoised_speech $\leftarrow$ savefile(recombined, sample_rate, original_filetype)
        return denoised_speech
```

### 8.2.2 Audio Split Algorithm

**Algorithm 2** Audio Split Algorithm

```
    Input wav_data, sample_rate
    Output audio_chunks
1: size $\leftarrow$ length of wav_data
2: step_size $\leftarrow 3 \times$ sample_rate
3: silence_size $\leftarrow 0.05 \times$ sample_rate
4: audio_chunks $\leftarrow$ []
5: for $i \leftarrow 0$ to size , step = step_size do
6:     silence $\leftarrow [0] *$ silence_size
7:     chunk $\leftarrow$ silence.extend(wav_data[$i : i +$ step_size])
8:     chunk $\leftarrow$ chunk.extend(silence)
9:     audio_chunks.append(chunk)
        return audio_chunks
```

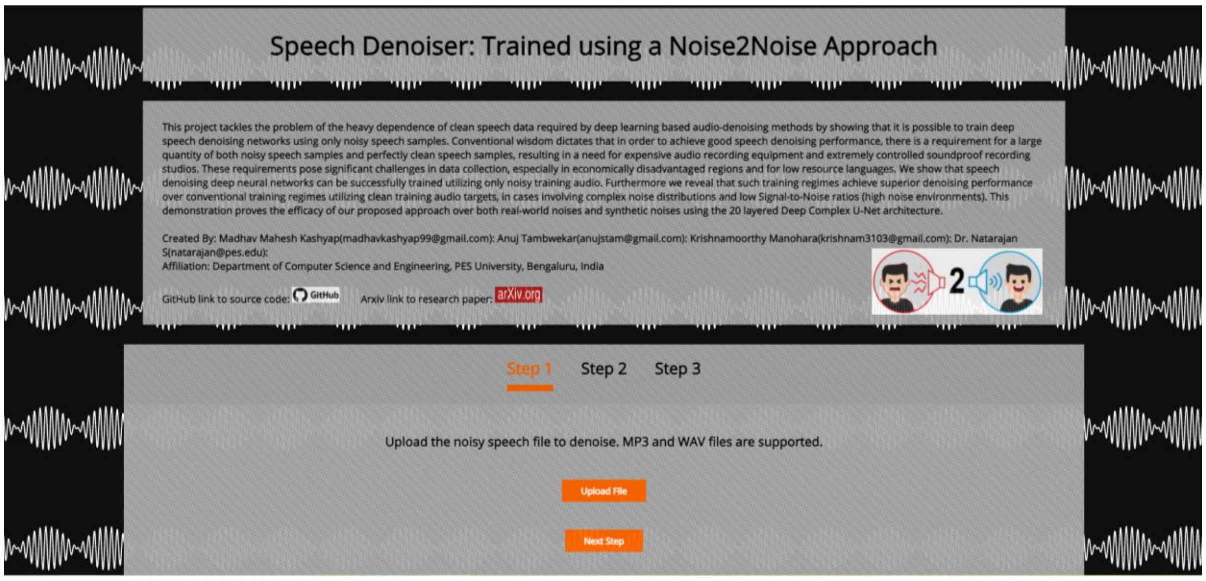
### 8.2.3 Audio Recombination Algorithm

**Algorithm 3** Audio Recombination Algorithm

```
    Input audio_chunks, sample_rate
    Output denoised_chunks
1: silence_size ← $0.05 \times$ sample_rate
2: denoised_chunks ← []
3: for each chunk ∈ audio_chunks do
4:     chunk_data ← chunk[silence_size :length(chunk) - silence_size]
5:     denoised_chunks ← denoised_chunks.extend(chunk_data)
     return denoised_chunks
```

As described earlier, the DCUNet model highlighted earlier is used as an independent model in these algorithms. We store the weights as .pth files, which is a standardized format used to store weights of models coded in Pytorch.

## 8.3 Front-End Browser View


Speech Denoiser: Trained using a Noise2Noise Approach

This project tackles the problem of the heavy dependence of clean speech data required by deep learning based audio-denoising methods by showing that it is possible to train deep speech denoising networks using only noisy speech samples. Conventional wisdom dictates that in order to achieve good speech denoising performance, there is a requirement for a large quantity of both noisy speech samples and perfectly clean speech samples, resulting in a need for expensive audio recording equipment and extremely controlled soundproof recording studios. These requirements pose significant challenges in data collection, especially in economically disadvantaged regions and for low resource languages. We show that speech denoising deep neural networks can be successfully trained utilizing only noisy training audio. Furthermore we reveal that such training regimes achieve superior denoising performance over conventional training regimes utilizing clean training audio targets, in cases involving complex noise distributions and low Signal-to-Noise ratios (high noise environments). This demonstration proves the efficacy of our proposed approach over both real-world noises and synthetic noises using the 20 layered Deep Complex U-Net architecture.

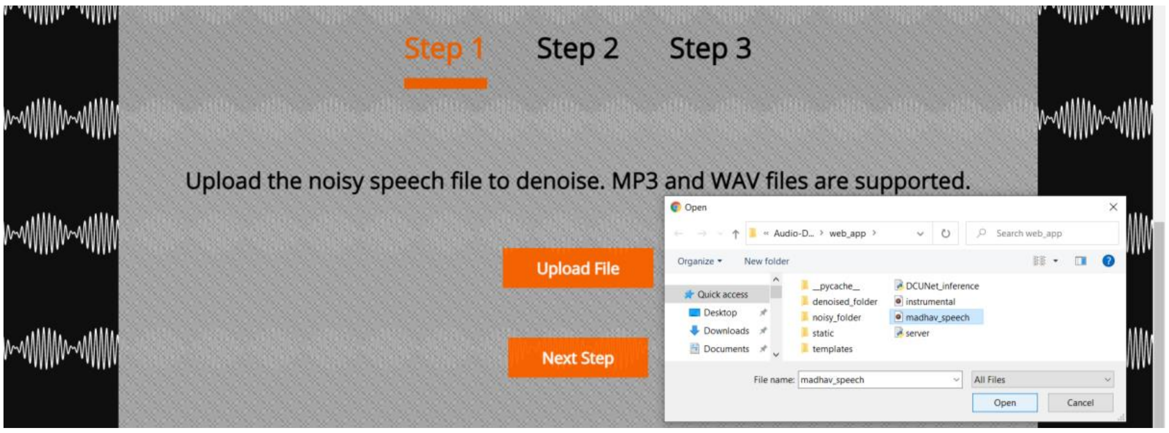
Created By: Madhav Mahesh Kashyap(madhavkashyap99@gmail.com): Anuj Tambwekar(anujstam@gmail.com): Krishnamoorthy Manohara(krishnam3103@gmail.com): Dr. Natarajan S(natarajan@pes.edu):
Affiliation: Department of Computer Science and Engineering, PES University, Bengaluru, India

GitHub link to source code: GitHub  Arxiv link to research paper: arXiv.org

Upload the noisy speech file to denoise. MP3 and WAV files are supported.

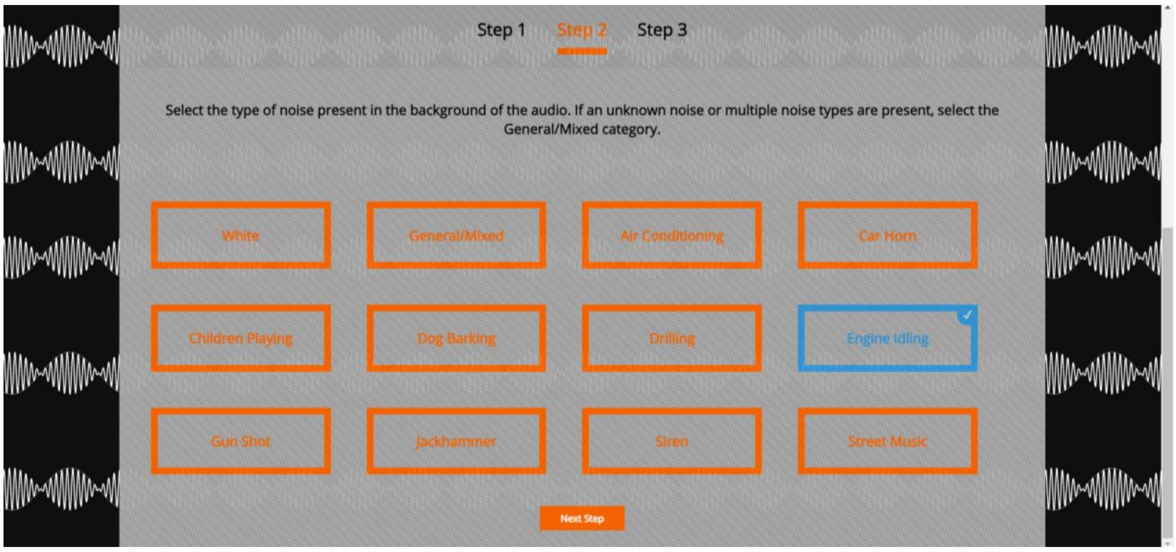
*Fig 8.5: Home Page Landing View*

The home page is the landing page users who use our denoiser product will first visit. We provide an introduction to our Noise2Noise approach, and provide links to our ArXiv hosted research paper and GitHub source code to encourage users to develop a deeper understanding. Our website uses a HTML form element to extract input from the user.



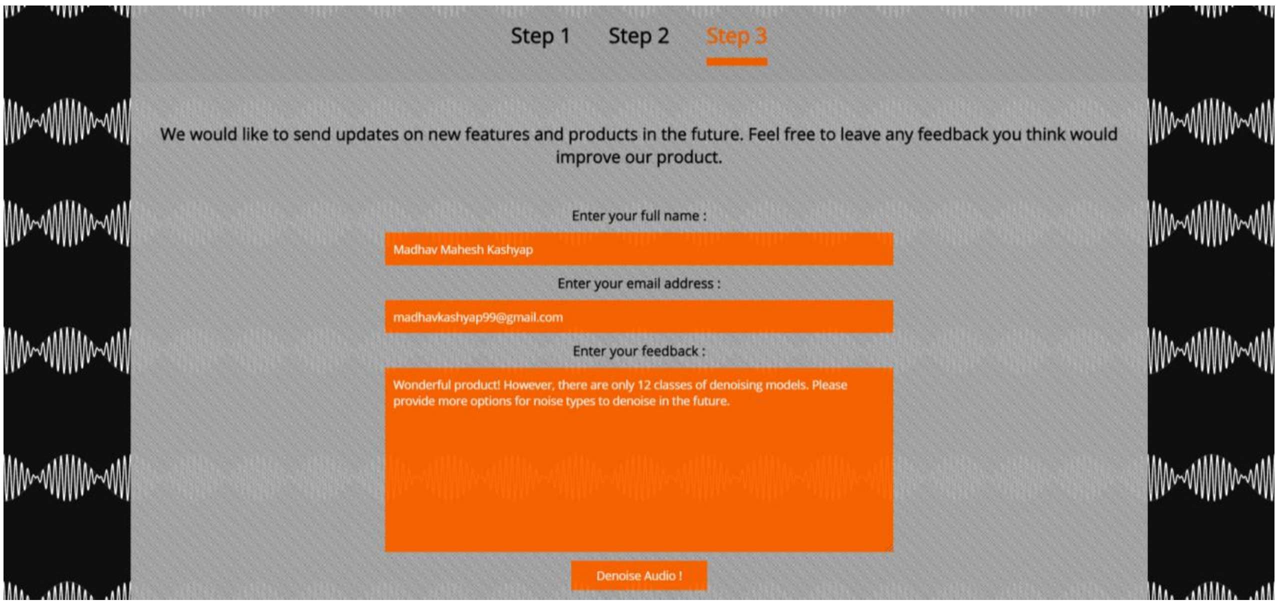
Upload the noisy speech file to denoise. MP3 and WAV files are supported.

*Fig 8.6: User Uploads their Noisy Speech File*



Select the type of noise present in the background of the audio. If an unknown noise or multiple noise types are present, select the General/Mixed category.

*Fig 8.7: User Selects the Noise Type Present in the Background*



We would like to send updates on new features and products in the future. Feel free to leave any feedback you think would improve our product.

Enter your full name :

Madhav Mahesh Kashyap

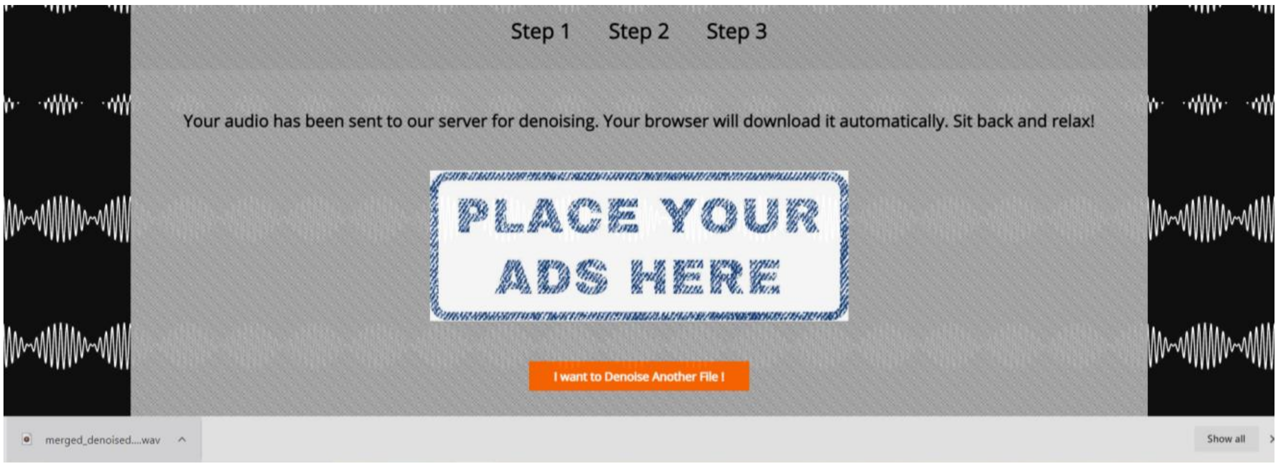
Enter your email address :

madhavkashyap99@gmail.com

Enter your feedback :

Wonderful product! However, there are only 12 classes of denoising models. Please provide more options for noise types to denoise in the future.

*Fig 8.8: User Information and Feedback is Retrieved*



Your audio has been sent to our server for denoising. Your browser will download it automatically. Sit back and relax!

*Fig 8.9: Denoised Audio will Automatically be Downloaded after Processing. Space for Potential Ad Revenue. User can Start Denoising other Speech Files.*

# CHAPTER 9

# EXPERIMENTAL SETUP AND RESULTS

## 9.1 Training Setup and Methodology

As highlighted earlier, we use the Urbansound dataset, along with synthesized white noise to generate the data files for our training. As there exists no reference dataset with noise in the inputs and the targets, we generate our own. To this we create a dataset for each noise distribution, where we first compute the SNR between the speech file and the noise, and then scale the volume of the noise such that the SNR between the two lies in the range of 0-10. Following this, the noise file is repeated until the length of the speech segment is complete. This process generates an input file. To create a target file to correspond to this input, a noise sample from a different distribution is picked (Please see Condition 2, Chapter 4 for further details), and a target file is generated the same way.

When dealing with white noise, we use white noise as the target and the input, due to its random nature. For the mixed category, the input noise was randomly selected for each speech sample, instead of selecting from a specific category, and the output noise is selected from a distribution other than the one used in the input.

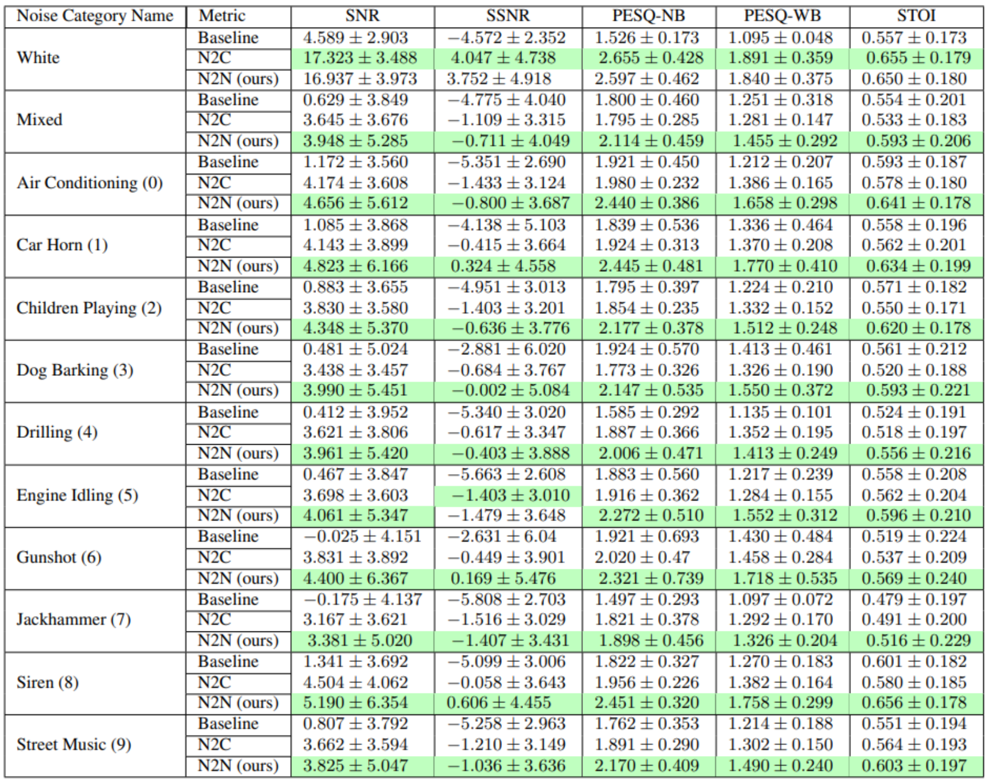
For each noise distribution, we train two identical DCUNet20 networks as described earlier, until they converge. The training data has audio from 26 speakers, totalling to 11572 speech samples, and the testing data contains audio from two speakers that have never been shown to the model before. For all distributions, training converged at roughly 4 epochs, and the SNR, Segmented SNR, narrow and wide band PESQ (PESQ NB, PESQ WB) and STOI were reported.

## 9.2 Training Environment

Due to the large size of the model, there was a requirement to train on a system with a large amount of compute capability. During inference, the model consumes roughly 4GB of VRAM, and while training, uses over 12GB. As a result, an Azure data science virtual machine was used to train the system, specifically an NC6 compute instance which contained an Nvidia K80

## 9.3 Results

Our results are encapsulated in Table 8.1. In the table, each row corresponds to a category of noise, with a column corresponding to an evaluation metric.



| Noise Category Name | Metric | SNR | SSNR | PESQ-NB | PESQ-WB | STOI |
|---|---|---|---|---|---|---|
| White | Baseline | $4.589 \pm 2.903$ | $-4.572 \pm 2.352$ | $1.526 \pm 0.173$ | $1.095 \pm 0.048$ | $0.557 \pm 0.173$ |
| | N2C | $17.323 \pm 3.488$ | $4.047 \pm 4.738$ | $2.655 \pm 0.428$ | $1.891 \pm 0.359$ | $0.655 \pm 0.179$ |
| | N2N (ours) | $16.937 \pm 3.973$ | $3.752 \pm 4.918$ | $2.597 \pm 0.462$ | $1.840 \pm 0.375$ | $0.650 \pm 0.180$ |
| Mixed | Baseline | $0.629 \pm 3.849$ | $-4.775 \pm 4.040$ | $1.800 \pm 0.460$ | $1.251 \pm 0.318$ | $0.554 \pm 0.201$ |
| | N2C | $3.645 \pm 3.676$ | $-1.109 \pm 3.315$ | $1.795 \pm 0.285$ | $1.281 \pm 0.147$ | $0.533 \pm 0.183$ |
| | N2N (ours) | $3.948 \pm 5.285$ | $-0.711 \pm 4.049$ | $2.114 \pm 0.459$ | $1.455 \pm 0.292$ | $0.593 \pm 0.206$ |
| Air Conditioning (0) | Baseline | $1.172 \pm 3.560$ | $-5.351 \pm 2.690$ | $1.921 \pm 0.450$ | $1.212 \pm 0.207$ | $0.593 \pm 0.187$ |
| | N2C | $4.174 \pm 3.608$ | $-1.433 \pm 3.124$ | $1.980 \pm 0.232$ | $1.386 \pm 0.165$ | $0.578 \pm 0.180$ |
| | N2N (ours) | $4.656 \pm 5.612$ | $-0.800 \pm 3.687$ | $2.440 \pm 0.386$ | $1.658 \pm 0.298$ | $0.641 \pm 0.178$ |
| Car Horn (1) | Baseline | $1.085 \pm 3.868$ | $-4.138 \pm 5.103$ | $1.839 \pm 0.536$ | $1.336 \pm 0.464$ | $0.558 \pm 0.196$ |
| | N2C | $4.143 \pm 3.899$ | $-0.415 \pm 3.664$ | $1.924 \pm 0.313$ | $1.370 \pm 0.208$ | $0.562 \pm 0.201$ |
| | N2N (ours) | $4.823 \pm 6.166$ | $0.324 \pm 4.558$ | $2.445 \pm 0.481$ | $1.770 \pm 0.410$ | $0.634 \pm 0.199$ |
| Children Playing (2) | Baseline | $0.883 \pm 3.655$ | $-4.951 \pm 3.013$ | $1.795 \pm 0.397$ | $1.224 \pm 0.210$ | $0.571 \pm 0.182$ |
| | N2C | $3.830 \pm 3.580$ | $-1.403 \pm 3.201$ | $1.854 \pm 0.235$ | $1.332 \pm 0.152$ | $0.550 \pm 0.171$ |
| | N2N (ours) | $4.348 \pm 5.370$ | $-0.636 \pm 3.776$ | $2.177 \pm 0.378$ | $1.512 \pm 0.248$ | $0.620 \pm 0.178$ |
| Dog Barking (3) | Baseline | $0.481 \pm 5.024$ | $-2.881 \pm 6.020$ | $1.924 \pm 0.570$ | $1.413 \pm 0.461$ | $0.561 \pm 0.212$ |
| | N2C | $3.438 \pm 3.457$ | $-0.684 \pm 3.767$ | $1.773 \pm 0.326$ | $1.326 \pm 0.190$ | $0.520 \pm 0.188$ |
| | N2N (ours) | $3.990 \pm 5.451$ | $-0.002 \pm 5.084$ | $2.147 \pm 0.535$ | $1.550 \pm 0.372$ | $0.593 \pm 0.221$ |
| Drilling (4) | Baseline | $0.412 \pm 3.952$ | $-5.340 \pm 3.020$ | $1.585 \pm 0.292$ | $1.135 \pm 0.101$ | $0.524 \pm 0.191$ |
| | N2C | $3.621 \pm 3.806$ | $-0.617 \pm 3.347$ | $1.887 \pm 0.366$ | $1.352 \pm 0.195$ | $0.518 \pm 0.197$ |
| | N2N (ours) | $3.961 \pm 5.420$ | $-0.403 \pm 3.888$ | $2.006 \pm 0.471$ | $1.413 \pm 0.249$ | $0.556 \pm 0.216$ |
| Engine Idling (5) | Baseline | $0.467 \pm 3.847$ | $-5.663 \pm 2.608$ | $1.883 \pm 0.560$ | $1.217 \pm 0.239$ | $0.558 \pm 0.208$ |
| | N2C | $3.698 \pm 3.603$ | $-1.403 \pm 3.010$ | $1.916 \pm 0.362$ | $1.284 \pm 0.155$ | $0.562 \pm 0.204$ |
| | N2N (ours) | $4.061 \pm 5.347$ | $-1.479 \pm 3.648$ | $2.272 \pm 0.510$ | $1.552 \pm 0.312$ | $0.596 \pm 0.210$ |
| Gunshot (6) | Baseline | $-0.025 \pm 4.151$ | $-2.631 \pm 6.04$ | $1.921 \pm 0.693$ | $1.430 \pm 0.484$ | $0.519 \pm 0.224$ |
| | N2C | $3.831 \pm 3.892$ | $-0.449 \pm 3.901$ | $2.020 \pm 0.47$ | $1.458 \pm 0.284$ | $0.537 \pm 0.209$ |
| | N2N (ours) | $4.400 \pm 6.367$ | $0.169 \pm 5.476$ | $2.321 \pm 0.739$ | $1.718 \pm 0.535$ | $0.569 \pm 0.240$ |
| Jackhammer (7) | Baseline | $-0.175 \pm 4.137$ | $-5.808 \pm 2.703$ | $1.497 \pm 0.293$ | $1.097 \pm 0.072$ | $0.479 \pm 0.197$ |
| | N2C | $3.167 \pm 3.621$ | $-1.516 \pm 3.029$ | $1.821 \pm 0.378$ | $1.292 \pm 0.170$ | $0.491 \pm 0.200$ |
| | N2N (ours) | $3.381 \pm 5.020$ | $-1.407 \pm 3.431$ | $1.898 \pm 0.456$ | $1.326 \pm 0.204$ | $0.516 \pm 0.229$ |
| Siren (8) | Baseline | $1.341 \pm 3.692$ | $-5.099 \pm 3.006$ | $1.822 \pm 0.327$ | $1.270 \pm 0.183$ | $0.601 \pm 0.182$ |
| | N2C | $4.504 \pm 4.062$ | $-0.058 \pm 3.643$ | $1.956 \pm 0.226$ | $1.382 \pm 0.164$ | $0.580 \pm 0.185$ |
| | N2N (ours) | $5.190 \pm 6.354$ | $0.606 \pm 4.455$ | $2.451 \pm 0.320$ | $1.758 \pm 0.299$ | $0.656 \pm 0.178$ |
| Street Music (9) | Baseline | $0.807 \pm 3.792$ | $-5.258 \pm 2.963$ | $1.762 \pm 0.353$ | $1.214 \pm 0.188$ | $0.551 \pm 0.194$ |
| | N2C | $3.662 \pm 3.594$ | $-1.210 \pm 3.149$ | $1.891 \pm 0.290$ | $1.302 \pm 0.150$ | $0.564 \pm 0.193$ |
| | N2N (ours) | $3.825 \pm 5.047$ | $-1.036 \pm 3.636$ | $2.170 \pm 0.409$ | $1.490 \pm 0.240$ | $0.603 \pm 0.197$ |

*Table 9.1: Denoising performance for N2N and N2C across noise categories*

A cell in green highlights the best performance for a given metric-noise combination. The baseline row provides the average performance of the test set before denoising and acts as a control. We observe that N2C performs better than N2N on all metrics for White noise, and on the SSNR metric for UrbanSound8K category 5 (Engine Idling). However, these performance differences are marginal. This is due to the simplicity of these stationary noise categories allowing for the N2C network to perform marginally better.

In every other category and metric, N2N performs better than N2C, due to the ability of the network to generalize better and avoid getting stuck in a local optimum. The lack of this ability is likely why we observe a decrease from Baseline in STOI for N2C in Mixed and UrbanSound8K categories 0,2,3,4 and 8. In all these cases, N2C still results in improved SNR and SSNR scores, however as mentioned earlier SNR alone is not a valid indication of true audio quality. In these cases, the N2C networks are removing noise at the expense of speech intelligibility (STOI and PESQ); a drawback the N2N networks do not suffer from.

Overall, these results are in line with our expectations and match our theoretical conclusions. We find that our N2N based networks are able to generalize better than N2C for complex real-world noises, and perform very similarly for simpler noise distributions.

Below, we show the plots comparing the performance between the baseline scores, and the N2C and N2N methods, to visually convey the results from Table 9.1

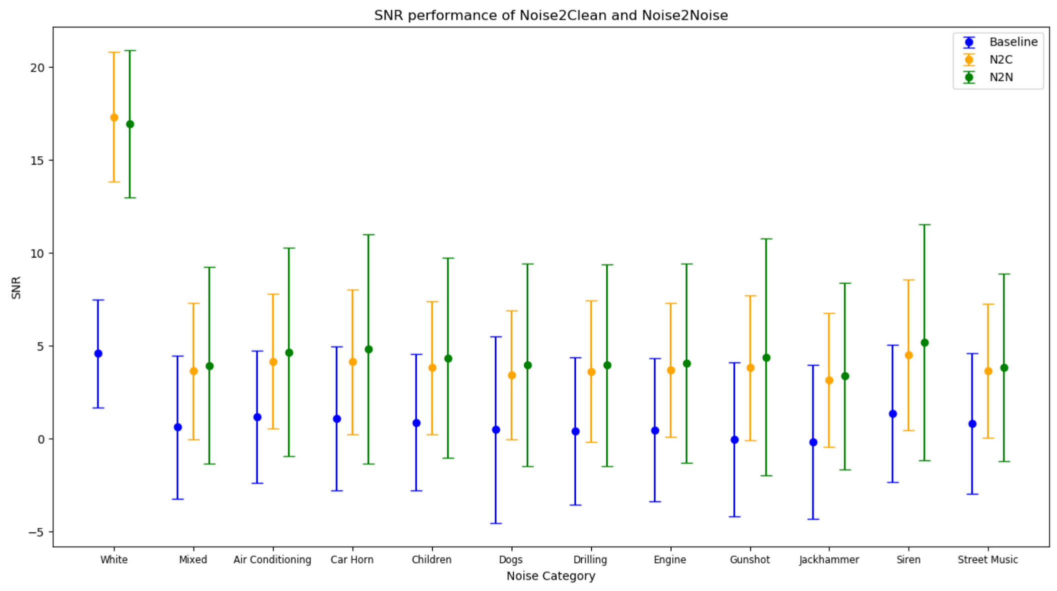


*Figure 9.1 SNR Performance Plot*

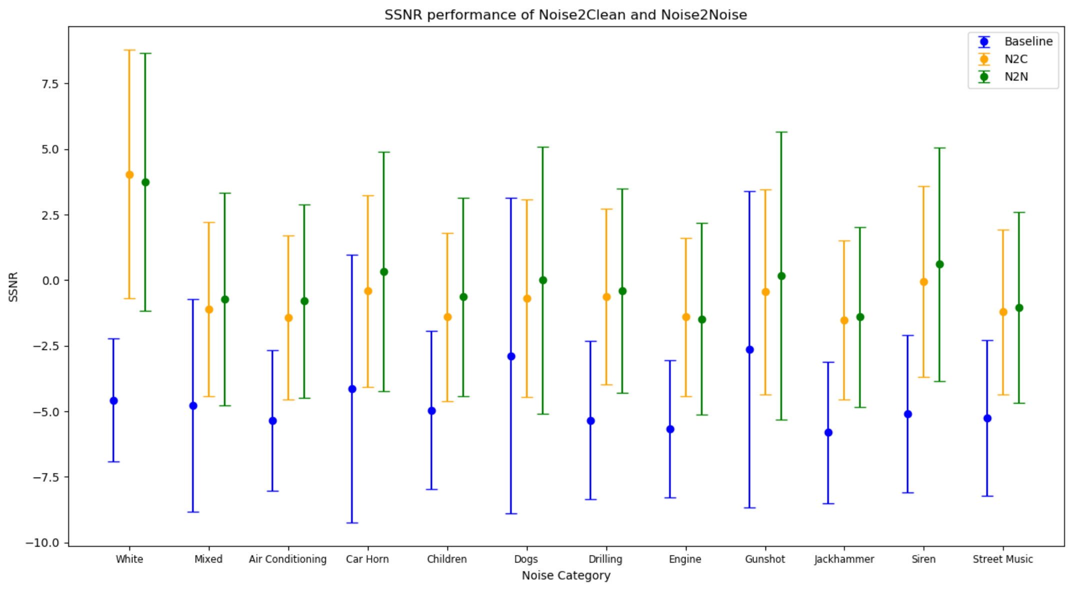


*Figure 9.2 SSNR Performance Plot*

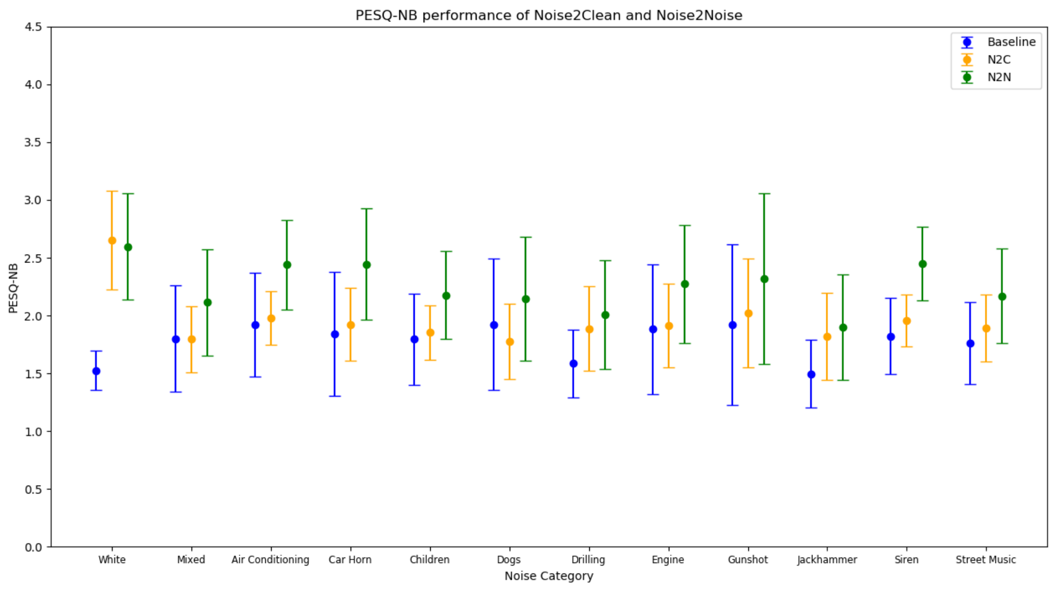


*Figure 9.3 PESQ-NB Performance Plot*

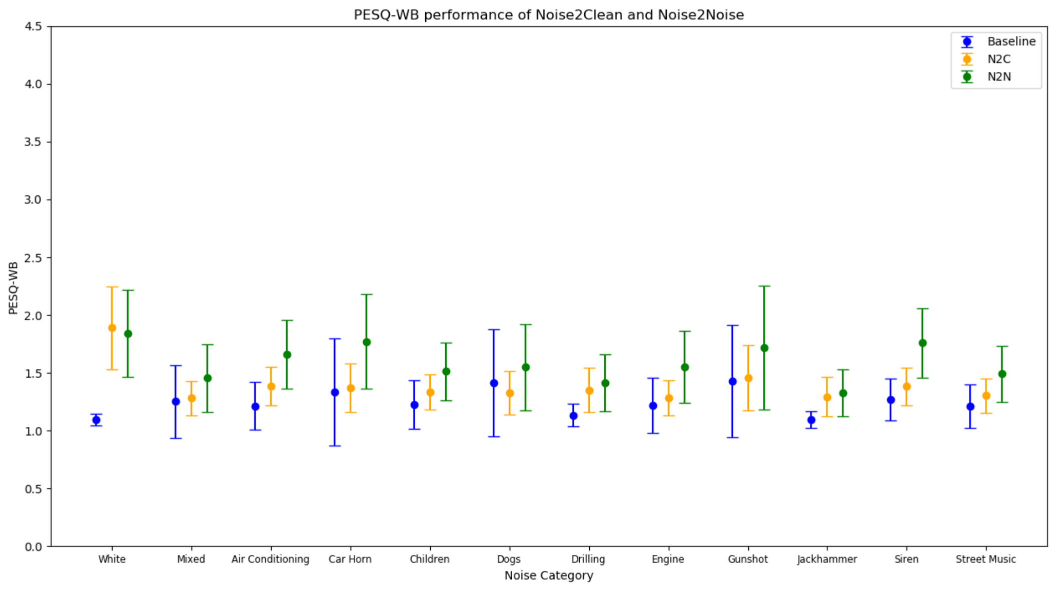


*Figure 9.4 PESQ-WB Performance Plot*

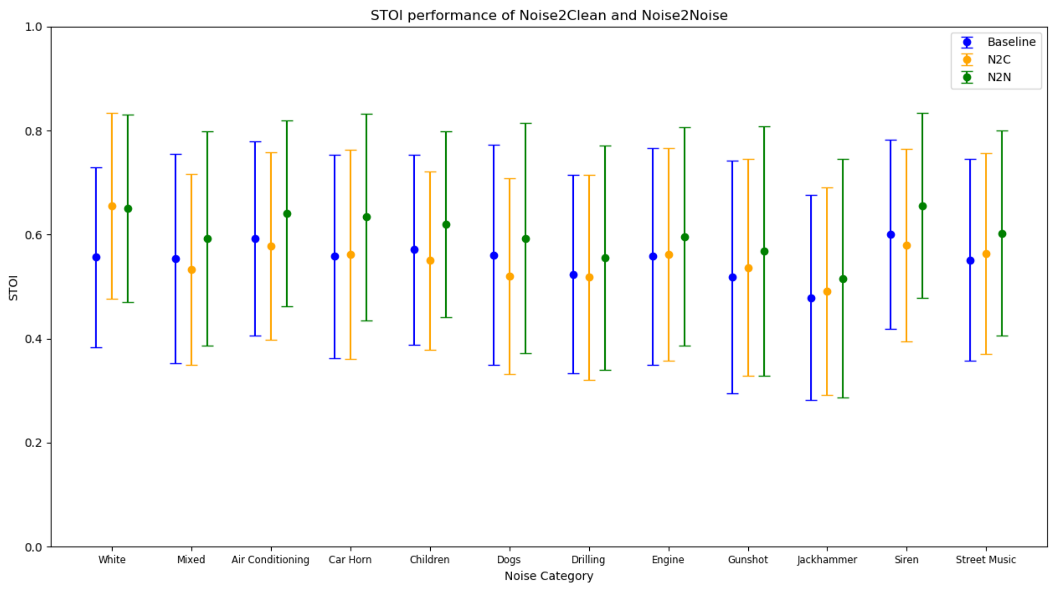


*Figure 9.5 STOI Performance Plot*

## 9.4 Visualizing Waveforms

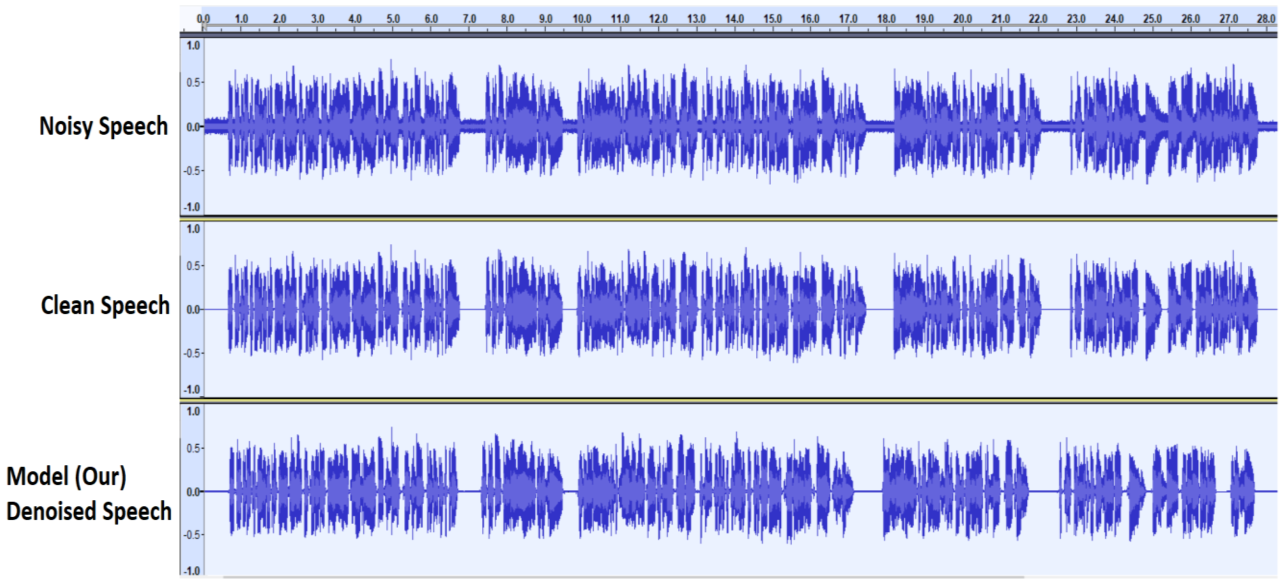


*Figure 9.6 Visualizing Noisy speech, Clean speech, and Model Denoised speech Waveforms on Audacity*

# CHAPTER 10

# CONCLUSION

In conclusion, we prove that it is sufficient to use only noisy data to train deep speech denoising networks, provided that a large amount of noisy data is available. This technique allows the creation of speech denoising models that are more generalizable and can preserve intelligibility. In addition, it removes an obstacle in the way of low-resource speech collection, in the form of a stringent requirement for perfectly clean speech. The elimination of this barrier can encourage the collection of recorded speech in regions that previously lacked the financial ability to purchase the equipment needed to record perfectly clean audio. Additionally, due to improved generalizability, the N2N technique allows for a single denoising network capable of removing multiple types of noise.

We see two key ways that this work can be taken forward. The first way is an analysis of the N2N technique on speech across various languages. The second is the simplification of the DCUNet model in order to make the training of these deep speech denoising networks more accessible, as N2N may solve the problem of poor data quality, but does not address the issue of difficulty of access to compute resources.

# REFERENCES/BIBLIOGRAPHY

[1] Ndajah, P., Kikuchi, H., Yukawa, M., Watanabe, H. and Muramatsu, S., 2011. An investigation on the quality of denoised images. International Journal of Circuit, Systems, and Signal Processing, 5(4), pp.423-434.

[2] J. Hansen and B. Pellom, “An effective quality evaluation protocol for speech enhancement algorithms,” in Proc. Int. Conf. Spoken Lang.Process., 1998, vol. 7, pp. 2819–2822

[3 “Perceptual evaluation of speech quality (PESQ), and objective method for end-to-end speech quality assessment of narrowband telephone net-works and speech codecs,” ITU, ITU-T Rec. P. 862, 2000.

[4] Loizou, P.C., 2013. Speech enhancement: theory and practice. CRC press.

[5] Recommendation, I.T.U.T., 2003. Subjective test methodology for evaluating speech communication systems that include noise suppression algorithm. ITU-T recommendation, p.835.

[6] Hu, Y. and Loizou, P.C., 2007. Evaluation of objective quality measures for speech enhancement. IEEE Transactions on audio, speech, and language processing, 16(1), pp.229-238.

[7] Taal, C.H., Hendriks, R.C., Heusdens, R. and Jensen, J., 2010, March. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In 2010 IEEE international conference on acoustics, speech and signal processing (pp. 4214-4217). IEEE.

[8] Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F.M. and Weber, G., 2019. Common voice: A massively-multilingual speech corpus. arXiv preprint arXiv:1912.06670.

[9] Valentini-Botinhao, C., 2017. Noisy speech database for training speech enhancement algorithms and TTS models.

[10] Reddy, C.K., Beyrami, E., Pool, J., Cutler, R., Srinivasan, S. and Gehrke, J., 2019. A scalable noisy speech dataset and online subjective test framework. arXiv preprint arXiv:1909.08050.

[11] Thiemann, J., Ito, N. and Vincent, E., 2013. The diverse environments multi-channel acoustic noise database: A database of multichannel environmental noise recordings. The Journal of the Acoustical Society of America, 133(5), pp.3591-3591.

[12] Salamon, J., Jacoby, C. and Bello, J.P., 2014, November. A dataset and taxonomy for urban sound research. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 1041-1044).

[13] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M. and Aila, T., 2018. Noise2noise: Learning image restoration without clean data. arXiv preprint arXiv:1803.04189.

[14] Matko Saric, Luki Bilicic and Hrvoje Dujmic, "White Noise Reduction of Audio Signal using Wavelets Transform with Modified Universal Threshold", University of Split, R. Boskovica b. b HR, volume 21000, 2005.

[15] K.P. Obulesu and P. Uday Kumar, "Implementation of Time Frequency Block Thresholding Algorithm in Audio Noise Reduction", International Journal of Science, Engineering and Technology Research (IJSETR), Volume 2, Issue 7, July 2013.

[16] Guoshen Yu, Emmanuel Bacry and Stephane Mallat, "Audio Signal Denoising with Complex Wavelets and Adaptive Block Attenuation", IEEE International Conference on Acoustics, Speech and Signal Processing, Volume 3, 2007.

[17] Choi, H.S., Kim, J.H., Huh, J., Kim, A., Ha, J.W. and Lee, K., 2018, September. Phase-aware speech enhancement with deep complex u-net. In International Conference on Learning Representations.

[18] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In International Conference on Medical image computing and computer-assisted intervention, pp. 234-241. Springer, Cham, 2015.

[19] Nagrani, A., Chung, J.S. and Zisserman, A., 2017. Voxceleb: a large-scale speaker identification dataset. arXiv preprint arXiv:1706.08612.

[20] Hain, T., Woodland, P.C., Evermann, G. and Povey, D., 2000, May. The cu-htk march 2000 hub5e transcription system. In Proc. Speech Transcription Workshop (Vol. 1).

[21] Y. Shi, W. Rong, and N. Zheng, “Speech enhancement using convolutional neural network with skip connections,” in 2018 11th International Symposium on Chinese Spoken Language Processing (ISCSLP), 2018, pp. 6–10.

[22] Z. Zhao, H. Liu, and T. Fingscheidt, “Convolutional neural networks to enhance coded speech,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 27, no. 4, pp. 663–678, 2019.

[23] F. G. Germain, Q. Chen, and V. Koltun, “Speech Denoising with Deep Feature Losses,” in Proc. Interspeech 2019, 2019, pp. 2723–2727.

[24] A. Azarang and N. Kehtarnavaz, “A review of multi-objective deep learning speech denoising methods,” Speech Communication, vol. 122, 05 2020.

[25] N. Alamdari, A. Azarang, and N. Kehtarnavaz, “Improving deep speech denoising by noisy2noisy signal mapping,” Applied Acoustics, vol. 172, p. 107631, 2021.

[26] R. E. Zezario, T. Hussain, X. Lu, H. M. Wang, and Y. Tsao, “Self-supervised denoising autoencoder with linear regression decoder for speech enhancement,” in ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020, pp. 6669–6673.

# APPENDIX A

Abbreviations Used:

- SNR - Signal to Noise Ratio
- SSNR - Segmented Signal to Noise Ratio
- PESQ - Perceptual Evaluation of Speech Quality
- PESQ-NB - Narrow Band PESQ
- PESQ-NB – Wide-Band PESQ
- DNN – Deep Neural Network
- CNN – Convolutional Neural Network
- DCUNet – Deep Complex U-net
- N2N – Noise2Noise (our approach)
- N2C – Noise2Clean (traditional approaches)