VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI - 590018

Project Report

on

"SOUND EVENT LOCALIZATION USING AN END TO END DEEP CONVOLUTIONAL NEURAL NETWORK"

Submitted in fulfilment of the requirements for the VIII Semester

Bachelor of Engineering

in

ELECTRONICS AND COMMUNICATION ENGINEERING

For the Academic Year

2020-2021

BY

NOEL ALBEN	**1PE17EC084**
NIRAJ ANIL BABAR	**1PE17EC083**
J SUMANTH	**1PE17EC054**
NEELURU SAI DEEPIKA	**1PE17EC080**

UNDER THE GUIDANCE OF

Dr. Ajey S.N.R.

Professor and Chairperson, Dept. of ECE, PESITBSC

Department of Electronics and Communication Engineering

PESIT - BANGALORE SOUTH CAMPUS

Hosur Road, Bengaluru - 560100

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI - 590018

Project Report

on

"'SOUND EVENT LOCALIZATION USING AN END TO END DEEP CONVOLUTIONAL NEURAL NETWORK"

Submitted in fulfilment of the requirements for the VIII Semester

Bachelor of Engineering

in

ELECTRONICS AND COMMUNICATION ENGINEERING

For the Academic Year

2020-2021

BY

NOEL ALBEN	**1PE17EC084**
NIRAJ ANIL BABAR	**1PE17EC083**
J SUMANTH	**1PE17EC054**
NEELURU SAI DEEPIKA	**1PE17EC080**

UNDER THE GUIDANCE OF

Dr. Ajey S.N.R.

Professor and Chairperson, Dept. of ECE, PESIT-BSC

Department of Electronics and Communication Engineering

PESIT - BANGALORE SOUTH CAMPUS

Hosur Road, Bengaluru - 560100

PESIT - BANGALORE SOUTH CAMPUS

HOSUR ROAD, BENGALURU - 560100

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "Sound Event Localization Using an End to End Deep Convolutional Neural Network" carried out by "Noel Alben, Niraj Anil Babar, J Sumanth, Neeluru Sai Deepika, bearing USN's 1PE17EC084, 1PE17EC083, 1PE17EC054, 1PE17EC080" respectively in fulfillment for the award of Degree of Bachelors (Bachelors of Engineering) in Electronics and Communication Engineering of Visvesvaraya Technological University, Belagavi during the year 2020-2021. It is certified that all corrections/ suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Dr. Ajey S.N.R.	**Dr. Subhash Kulkarni**	**Dr. Subhash Kulkarni**
Professor, Chairperson	**HOD**	**Principal, PESIT-BSC**

External Viva

Name of the Examiners	**Signature with Date**
1.	
2.	

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our effort with success.

We are indebted to our Guide, Dr. Ajey S.N.R, Professor, Chairperson Department of Electronics and Communication Engineering, PESIT - Bangalore South Campus, who has not only coordinated our work but also given suggestions from time to time.

We would like to express our gratitude to Dr. Subhash Kulkarni, Principal and ECE Head of Dept. PESIT Bangalore South Campus, for providing us with the relevant infrastructure.

We gratefully acknowledge the help lent out to us by all faculty members of the Department of Electronics and Communication Engineering, PESIT Bangalore South Campus, at all difficult times. We would also take this opportunity to thank our college management for the facilities provided. Furthermore, we acknowledge the support and feedback of our parents and friends.

Noel Alben
Niraj Anil Babar
J Sumanth
Neeluru Sai Deepika

ABSTRACT

In this project we will implement an end-to-end deep convolutional neural network operating on multi-channel raw audio data to localize multiple simultaneously active acoustic sources in space as proposed by Harshavardhan Sundar, Weiran Wang, Ming Sun and Chao Wang in [4]. It makes use of a novel Sample DCNN architecture and encoding scheme to represent the spatial coordinates of multiple sources, which facilitates 2D localization of multiple sources in an end-to-end fashion. We aim to experiment and test our implementation of this novel method.

Keywords: Acoustic Source Localization; Deep Learning; Sample CNN; Room Impulse; Sample Based Multiple Encoded Source Location Predictor (SMESLP).

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

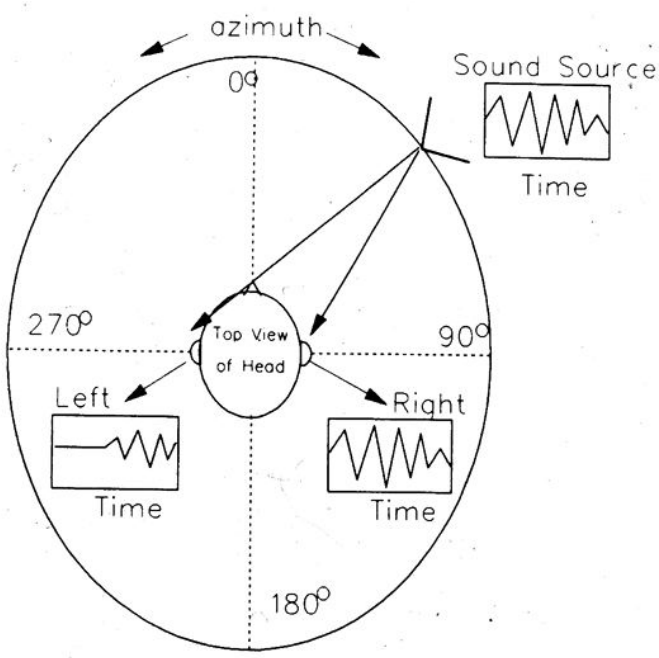
1.1 Sound Event Localization

Recognition and localization of acoustic events are relatively recent practical applications of audio signal processing, Acoustic source localization (ASL) pertains to the problem of localizing an acoustic source in space using only the audio data captured by an array of microphones. Most traditional methods incorporate simplifying assumptions such as single source localization, time-invariant sources, etc. However, these assumptions may be seriously violated in a range of emerging real-life applications. In these applications, the environment is extremely challenging, with spatially distributed sources, reverberation and reflections, simultaneously active speakers, interference signals, and time-varying positions of sources.

This special issue presents recent advances in the development of signal processing methods for localization and tracking of acoustic sources and the associated theory and applications. Addressing the challenges raised by real-life environments, novel methods are introduced representing advances in array processing, speech processing, and the use of data inference tools.

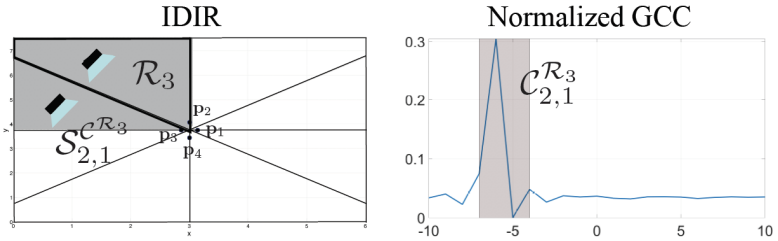
Acoustic source localization is a challenging problem wherein the goal is to estimate the coordinates of a sound source using the signals received by an array of microphones. Historically, this problem of localizing a single acoustic source has attracted several signal processing based solutions like Time Difference Of Arrival between pairs of microphones for acoustic source localization. In realistic scenarios, multiple sources may be simultaneously active causing Single Source Localisation (SSL) approaches to fail in such scenarios. However, it is not straightforward to extend the TDOA based SSL approaches to address Multiple Source Localisation, because it is difficult to obtain the individual source Time Differences of Arrivals, and also to determine a unique mapping between TDOA estimates and the corresponding sources. This problem is referred to as the association ambiguity i.e, the permutation problem.

Figure 1.1: Acoustic Source Localization Basic



To overcome this problem, HV. Sundar proposes a novel encoding scheme to represent the spatial coordinates of multiple sources and introduces inverse delay interval region (IDIR), where IDIR is an inter-hyperboloidal spatial region corresponding to an interval of delays for a given pair of microphones and proceeds to formulate a 2 stage scheme used for localizing multiple acoustic sources. In the first stage, the given enclosure is partitioned into non-overlapping elemental regions and the ones that contain a source are detected using a measure based on the generalized cross-correlation with phase transform (GCC-PHAT), and the IDIR's. In the second stage, the sources are finely localized within each of the detected elemental regions by identifying IDIRs containing a single source and a novel region-constrained localization approach.

Figure 1.2: Two stage TDOA



(a) Mic. pair: (2,1) (b) $\overline{\Psi}_{21}[\tau]$

More recently, HV Sundar extends his research into this problem statement and proposes a raw waveform based end to end Deep Convolutional Neural Network ar-

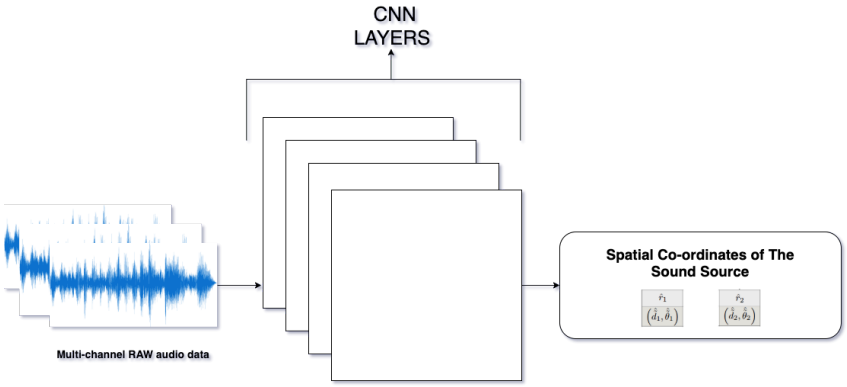
chitecture that facilitates 2D localization of multiple sources in an end-to-end fashion, avoiding the permutation problem and achieving arbitrary spatial resolution. In his work the experiments on a simulated data set and real recordings demonstrate that the proposed method generalizes well to unseen test conditions, and outperforms his own 2 stage TDOA based multiple source localization approach for multiple acoustic sources.

1.1.1 Proposed Approach

The Sound Event Localization Deep Convolutional Neural Network, reported in the literature serves as the base paper for this project. Here, the author presents a deep convolutional neural network (DCNN) architecture to map the raw audio data from a microphone array to the 2D coordinates of the sources, without any pre-/post-processing of inputs/outputs. For avoiding the permutation problem, HV. Sundar proposes a joint Coarse-Fine localization strategy, in which a source is associated with a coarse region followed by a fine location of distance and azimuth angle. Instead of outputting the source coordinates directly, the approach is proposed to output encoded coordinates based on the coarse location of the source.

The "SampleCNN" architecture is proposed with the output encoding layer to detect the coarse regions containing an active source, while simultaneously estimating the source location finely within each such active coarse region. The proposed approach is referred to as Sample-based Multiple Encoded Source Location Predictor (SMESLP).

Figure 1.3: Proposed Approach



1.1.2 Applications

Aside from classic source localization scenarios, present-day application areas include audio recording with mobile devices (e.g. cell phones, action cameras, and robots), video conferencing on the go, and recording for 3D reproduction and virtual reality.

- Imagine being inside a Madison Square Garden concert game or concert. Everyone around you is yelling at the top of their lungs, except one person. Now, we want to find that one person. Sound source localization will help us isolate this person and determine where he or she is in the crowd.
- While this is a trivial example, multiple applications require sound source localization such as in hearing aids, robotics, navigation for ships as well as self-driving cars, and in surveillance too.
- Sound event localization in robotic platforms allows a robot to locate a sound source by sound alone. It enriches human-robot interaction by complementing the robot's perceptual capabilities.

1.2 Definitions and Abbreviations:

- SSL (Single source localization): Localising a single acoustic source in a space.
- MSL (Multiple source localization):Localisation of multiple simultaneously active acoustic sources in a space.
- Deep Learning: Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning.
- CNN: Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- TDOA (Time Difference Of Arrival): TDOA algorithms locate a signal source based on the difference in the time of arrival of the source at the receivers.
- SMESLP (Sample-based Multiple Encoded Source Location Predictor): The "SampleCNN" architecture is proposed with the output encoding layer to detect the coarse regions containing an active source, while simultaneously estimating the source location finely within each such active coarse region. The proposed approach is referred to as Sample-based Multiple Encoded Source Location Predictor (SMESLP).

- IDIR (Inverse Delay Interval Region) : is defined as an inter-hyperboloidal spatial region corresponding to an interval of delays for a given pair of microphones.
- GCCPHAT: The cross-correlation is computed using the generalized cross-correlation phase transform (GCC-PHAT) algorithm. gccphat finds the location of the peak of the cross-correlation between the signal and the reference signal.
- ASL (Acoustic source localization): Acoustic source localization is the task of locating a sound source given measurements of the sound field. The sound field can be described using physical quantities like sound pressure and particle velocity. By measuring these properties one can obtain a source direction.

Chapter 2

LITERATURE SURVEY

In our preliminary research and literature survey, we came across different methodologies to tackle this problem, first, we were introduced to SOLO:2D Localisation with Single Sound Source and Single Microphone [1] proposed by Y. Zhang, Z. Wang, and J. Wang, where they considered the solution to localizing 2D sources as a classifying task with different frequencies at each group within the 2D area have different strengths and 1D distance measured using the traditional phase-based approach. The 1D distance and the sound strengths are taken as features to a neural network to classify the sample into different classes, each representing a region in 2D space.
Soon we came across a solution that incorporated Time Difference Of Arrival (TDOA) between pairs of microphones for multiple acoustic source localization and its improved approach which resolves the association ambiguity of microphone pairs to the sources, [2] proposed by H Sundar, TV Sreenivas and CS Seelamantula, this introduce the concept of Inverse Delay Interval Region (IDIR), IDIR is defined as an inter-hyperboloidal spatial region corresponding to an interval of delays for a given pair of microphones. The two-stage scheme proposed in [2] to localize multiple acoustic sources serves as our baseline to compare and understand the results of Raw waveform based end to end deep CNN for spatial localization of multiple acoustic sources [4] proposed by H Sundar, W Wang, M Sun and C Wang, the base paper we are referring to implement this project.

Furthermore, we came across, SELDnet for static scenes with spatially stationary sources [3] proposed by Sharath Advanne, giving us an insight into how Deep Learning can be used to decode and overcome the various ambiguities of the problem statement and its various approaches. As a result of our literature survey, the choice to implement [4] was a desire to incorporate an end to end approach, where the raw waveforms were used without needing to pre-process the datasets. We believe that once implemented, this methodology will make it possible to integrate the network into a real-world application.

Raw waveform DCNN [4] implements an end to end deep Convolutional Neural Network architecture to localize multiple acoustic sources in a space using multi-channel raw audio data. It makes use of a novel encoding scheme to represent the spatial coordinates of multiple acoustic sources which facilitates 2D localization of the sources in an end to end fashion. The architecture is derived from and built off of "Sample-Level Deep Convolutional Neural Networks for Music Auto-Tagging using raw waveforms" [5]and [6] T. Kim, J. Lee, J. Nam, "Comparison and analysis of samplecnn architectures for audio classification" novel CNN architectures that use Raw waveforms as inputs.

For training and testing the model , as suggested in the base paper[4], we sort to generate the dataset. For the reverberant dataset generation the clean speech signals from the TIMIT DR8 are convolved with the Room Impulse Response generator. For the RIR generation we chose the image method [7].A mex-function, which can be used in MATLAB, has been created to generate multi-channel Room Impulse Responses (RIR) using the image method. This function enables the user to control the reflection order, room dimension and microphone directivity. Another advantage of this mex-function, compared to a standard MATLAB function, concerns the computation time. The mex-function is around 100 times faster than the standard MATLAB.

Chapter 3

HARDWARE AND SOFTWARE REQUIREMENT SPECS.

3.1 Software Requirements

The software requirements are vital for the implementation for our project. High level programming languages and Deep learning environments make up most of our requirements and we will go over them in this section.

a. Python $>$ 3.5 [Jupyter Notebook] :

- Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability and easy understanding for quick debugging and implementation.

- Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.

- The programming language used to implement this project is python. The Deep learning architectures were built using python and the subsequent I/O requirements were processed using the same.

- Our collaborative work was compiled and verified using the jupyter notebook environment.

b. TensorFlow, Keras library :

- TensorFlow is a free and open-source software library for machine learning. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

- Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

- These libraries were used as a python interface for artificial neural networks. They enabled us to train and test our CNN models with the appropriate datasets.

c. MATLAB, MATLAB MEX Functions :

- MATLAB is a proprietary multi-paradigm programming language and numeric computing environment. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

- Mex file names compile and link one or more C++ source files written with the MATLAB Data API into a binary MEX file in the current folder.

- The datasets required to train and test the CNN model were simulated using MATLAB. The MEX function was used to link the C++ code source with the MATLAB data.

d. C++ :

- C++ is a general-purpose programming language created as an extension of the C programming language.
- The rir generator function was written using C++ which was then used in the MATLAB code for dataset simulation.

e. GitHub :

- GitHub is a provider of Internet hosting for software development and offers source code management functionality.
- All of our codes and modules are hosted on GitHub.
- nol-alb/HVsundarSourceLocalization.git

3.2 Harware Requirements

This section gives a brief overview of the operating environment in which the system will run, and the hardware prerequisites of the project.

a. Any suitable computer with an active internet connection.

b. RAM - 4GB or higher.

c. Processor - Intel i3 or higher.

d. Hard Disk - 500GB or higher.

Chapter 4

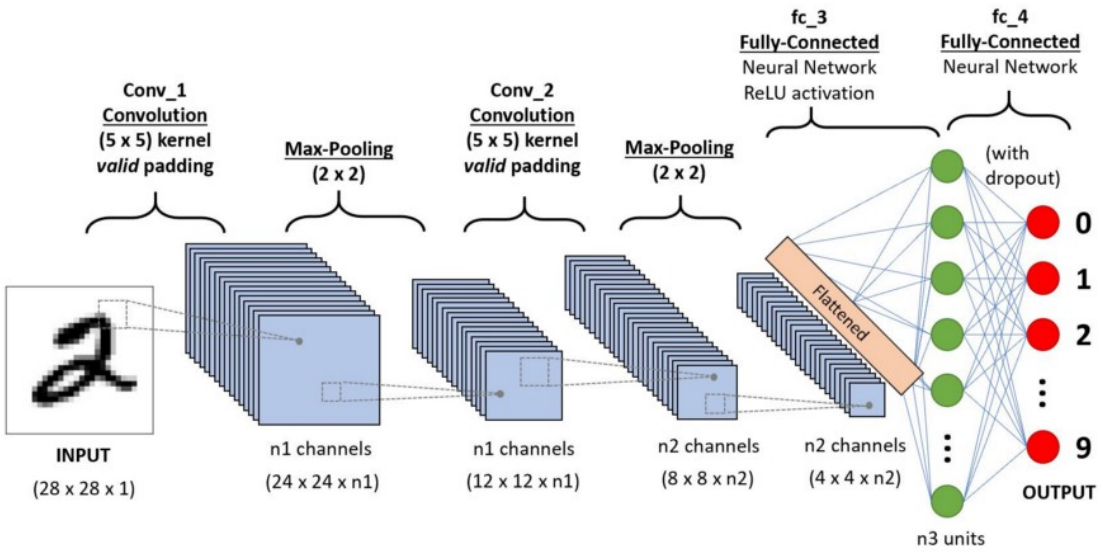
PRELIMINARY STUDIES

These are the various components we have used in the implementation of our project.

4.1 Convolutional Neural Network

A Convolutional Neural Network is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other [8]. The role of CNN is to reduce the dimensionality of the data without losing features which are critical for getting a good prediction.The filter parameter of the convolution layers dictate the hyper parameters which help train the model.The most important layers of a CNN are Conv layers, Max pooling layers and Fully connected layers. There are a number of CNN architectures present and we will be using the Sample level CNN for our implementation.

Figure 4.1: Convolutional Neural Network



4.1.1 Sample level CNN

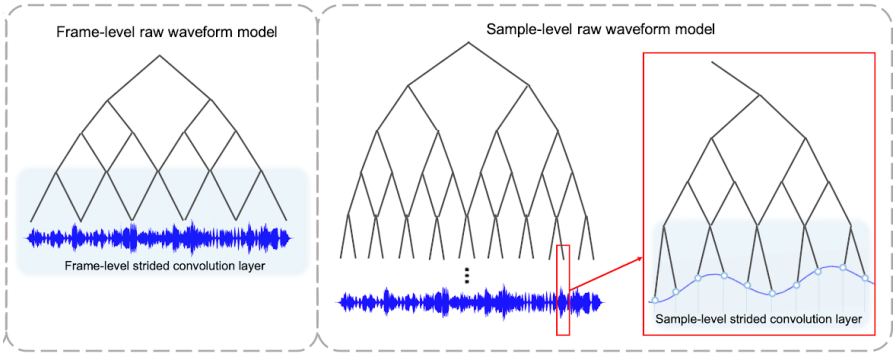


Figure 4.2: Sample Level CNN

This is an architecture which is specially used for Music Information Retrieval. This learns features from very small grains like 2-3 samples. MIR typically uses frame level strides to learn the features but all the possible phase variations cannot be learned from these frames. Harshavardhan Sundar uses this Sample level CNN architecture along with 2 more blocks - Residual Squeeze and Excitation block ReSeNet[6].
Residual Networks allow training a very deep CNN using skip connections. The shortcuts make backpropagation of gradients more fluent in the neural networks and even enables training a 1001-layer ResNet.
The Res network was incorporated with a Squeeze and Excitation block to form ReSe-n block.

We implemented the Sample CNN architecture as proposed by Jongpil Lee for Music Auto-Tagging [5]. Our architecture was implemented on Jupyter notebooks, using Keras and TensorFlow backend to train, test, and validate the architecture.

As the length of the audio clip varies in general, the following issues were considered during the implementation of CNN architecture:

- Convolution filter length and sub-sampling length.
- The temporal length of the hidden layer activations on the last sub-sampling layer.
- The segment length of audio corresponds to the input size of the network.

layer	stride	output	No. of Parameters
conv3-128	3	19683 X 128	512
conv3-128	1	19683 X 128	
maxpool 3	3	6561 X 128	49280
conv3-128	1	6561 X 128	
maxpool 3	3	2187 X 128	49280
conv3-256	1	2187 X 256	
maxpool 3	3	729 X 256	98560
conv3-256	1	729 X 256	
maxpool 3	3	243 X 256	196864
conv3-256	1	243 X 256	
maxpool 3	3	81 X 256	196864
conv3-256	1	81 X 256	
maxpool 3	3	27 X 256	196864
conv3-256	1	27 X 256	
maxpool 3	3	9 X 256	196864
conv3-256	1	9 X 256	
maxpool 3	3	3 X 256	196864
conv3-512	1	3 X 512	
maxpool 3	3	1 X 512	393728
conv1-512	1	1 X 512	
dropout 0.5	-	1 X 512	262656
Sigmoid	-	50	25650
Total Parameters		=	$1.9X10^6$

Table 4.1: Sample CNN parameters

4.1.2 Parameters of The Sample CNN Architecture

By building the models this way, the number of parameters is reduced between the last sub-sampling layer and the output layer.

The proposed CNN architecture is a 3^9 model with 19683 frames and 59049 samples as input. 9 layers of Convolution Layers followed by Max pool layers with batch normalization.

The performance is compared by the depth of layers and the stride of the first convolution layer. The CNN model built is of greater depth so it yielded a high accuracy and achieves highly comparable results.

4.2 Sample CNN

SampleCNN is an end-to-end CNN model that takes raw waveforms as input . The architecture is composed of stacks of convolutional layer and subsampling layer. The main difference from other CNN models that take raw waveforms is that it has a very small filter size such as 2 or 3 samples in all layers. An example of SampleCNN is shown above where sub-sampling is done by max-pooling except the

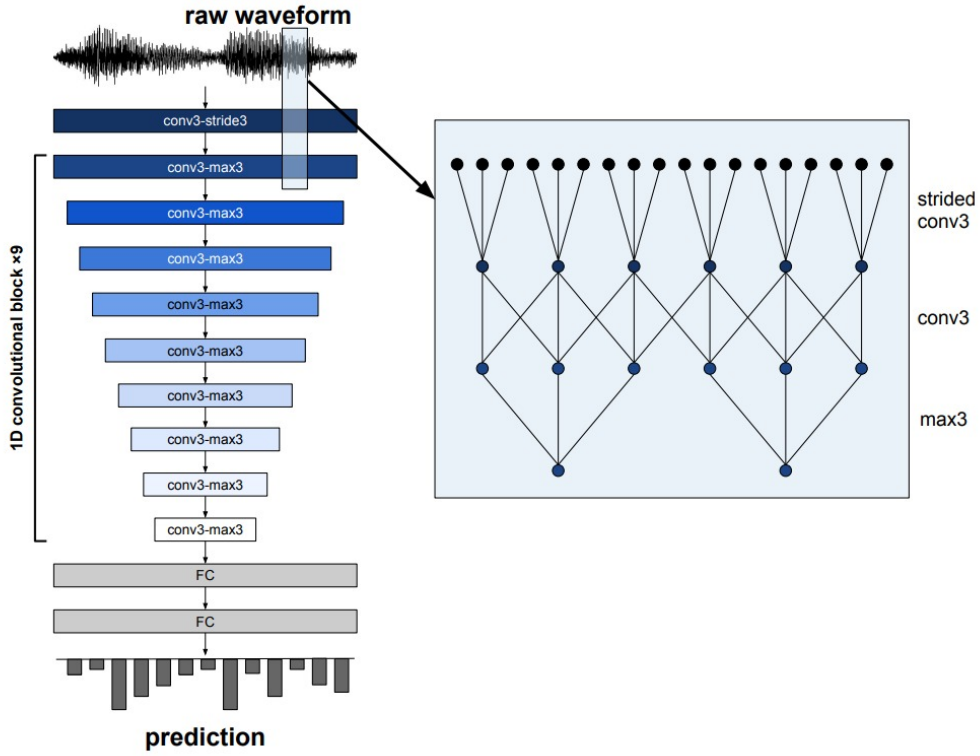


Figure 4.3: Sample CNN

first convolutional layer that operates with striding, and the filter size, stride size, and max-pooling size are all 3 samples.

4.2.1 Extended Sample CNN

The SampleCNN model was enhanced by adding residual connections, squeeze-and-excitation modules as proposed in [6]. The residual connection makes gradient propagation more fluent, allowing training deeper networks . The Squeeze-and-Excitation (SE) module recalibrates filter-wise feature responses.

Res-n block

A residual network consists of residual units or blocks which have skip connections, also called identity connections.Residual block has a convolution layer followed by a batch normalization layer and a ReLU activation function. This is again continued by a convolution layer and a batch normalization layer.
The skip connection basically skips both these layers and adds directly before the ReLU activation function. Such residual blocks are repeated to form a residual network.

Here the basic block with a single skip connection is defined as Res-n Block. n counts the number of convolutional layers and we handle only two cases when n is

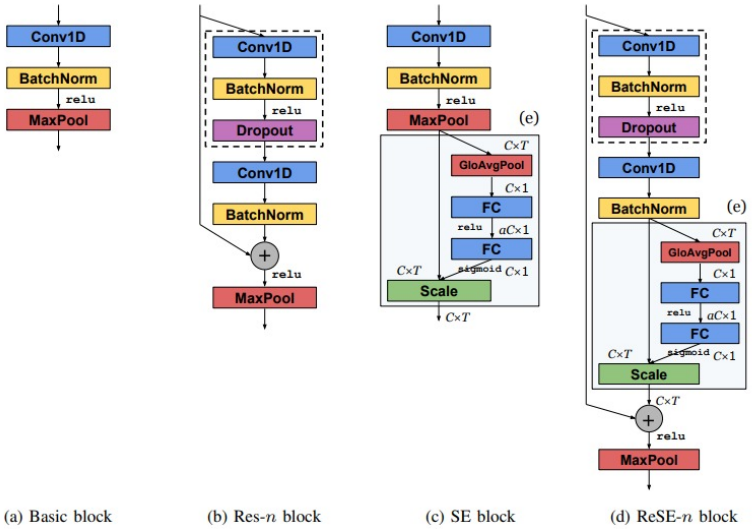


Figure 4.4: Res & SE blocks

1 or 2. Res-2 block is shown in Fig. 4.4b. It has a dropout layer with a drop ratio of 0.2 before the second convolutional layer to prevent overfitting.

Squeeze-and-Excitation (SE) Block

SE block is a new architectural unit that can be added to the basic unit as shown in Fig. 4.4c. It consist of two operations: squeeze and excitation operations.

The squeeze operation is a global average pooling of the features obtained from the convolution of the last layer in the CNN the feature maps after the squeeze are reduced by a fully connected layer and are nonlinearized by the ReLU activation function, hereafter upgraded through the fully connected layer, and then weight is activated by sigmoid, that is, the excitation operation.

ReSE-n block

ReSE-n block is built by combining the Res-n block and the SE block to take advantages of both blocks. There are multiple possible positions where the SE block can be integrated with the Res-n block . We chose the combination as shown in Fig. 3d .

For our implementation, we decided to approach this project by understanding the fundamentals of Convolutional Neural Networks and more importantly the Sample CNN architecture that forms the basic blocks and extended Sample CNN architecture with ReSE block for the raw waveform based sound event localization deep CNN architecture of the base paper.

Chapter 5

MAIN ARCHITECTURE IMPLEMENTATION

5.1 Simulating the datasets for training, testing and validation

There are few publicly available datasets,but due to the requirement of large quantity and the constraints mentioned in base paper[4] like choice of microphone array,room dimensions, we chose to simulate the datasets for training and testing our model.We require both anechoic and reverberant datasets for training and testing of the proposed network architecture.

5.1.1 Room dimension and Sources

As mentioned in the paper we are using a room of dimension 6 x 7.5 x 4.5 . In the center of the room we have a Uniform Circular Array(UCA) with 8 microphones such that the room is divided into 8 regions - R1 to R8. The architecture aims at performing Coarse as well as Fine localization of sources. Coarse localization tells us in which region the source is present and Fine localization gives us the distance and the azimuthal angle of the source in the region.

We are using a 3D room to simulate the required Room Impulse Response but the

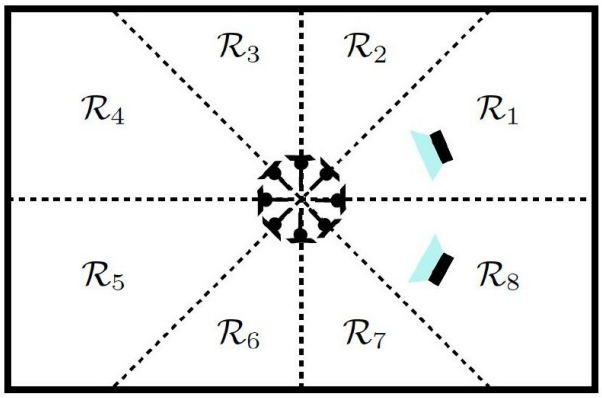


Figure 5.1: Partitioning the room

sound event localization is happening in the 2D plane - height at which the micro-

phone is placed.

5.1.2 Reverberant Dataset

The required reverberant datasets are simulated using the RIR generation function. The enclosure and sources are generated from E.A.P.Habets, 'Room Impulse Response Generator' [7].
The reverberant data are created by convolving clean speech signals as shown in equation 5.1 with room impulse response(RIR) generated.

$$x_i[n] = \sum_{i=1}^{M} s_i[n] * h_{ij}[n] \tag{5.1}$$

Where $x_i[n]$ is the microphone signal , $s_i[n]$ is the clean speech signal from TIMIT DR8 and $h_{ij}[n]$ is the RIR generated. The RIR generator function uses the image

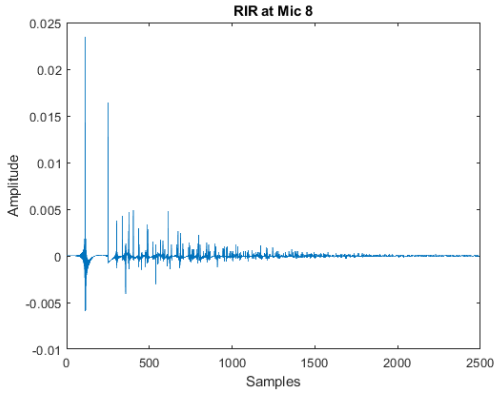


Figure 5.2: Room Impulse Generated

method proposed by Allen and Berkley. The function includes the parameters like room dimension, velocity of sound, mike type, receiver positions, sampling frequency, source position, mic type, etc.
For RIR generation an enclosure of size 6*4.5*7.5 is considered and it is divided into 8 sector-like regions with an azimuthal angle of 45°. A uniform circular array of 8 microphones with a radius of 10cm is placed at the center as shown below.

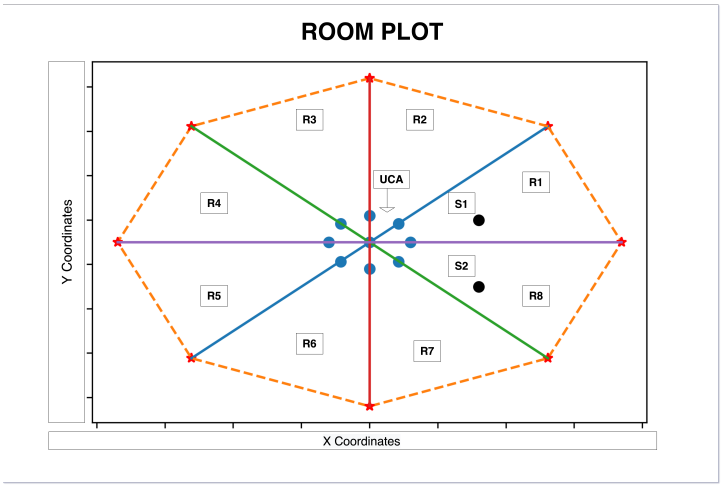


Figure 5.3: Room Enclosure Simulated on Matlab

The clean speech signals from the TIMIT DR8 database have been convolved with the room impulse responses and the pickup of the microphones is simulated.

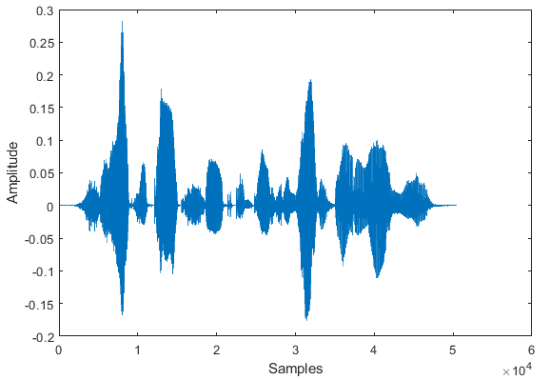


Figure 5.4: Convolved TIMIT DR8 Sample

```
%_____________room impulse generation_____________%

c = a.speed_sound;                      % Sound velocity (m/s)
fs = 16000;                  % Sample frequency (samples/s)
r = [a.Mic_pos(1,:) 0;a.Mic_pos(2,:) 0;a.Mic_pos(3,:) 0;a.Mic_pos(4,:) 0
;a.Mic_pos(5,:) 0;a.Mic_pos(6,:) 0;a.Mic_pos(7,:)  0;a.Mic_pos(8,:) 0];
  % Receiver positions [x_1 y_1 z_1 ; x_2 y_2 z_2] (m)
L = [6 7.5 4.5];           % Room dimensions [x y z] (m)
beta = 0.3;                 % Reverberation
```

5.1.3 Anechoic Dataset

The anechoic data set is created by superposition of appropriately shifted and attenuated clean speech signals for each active source based on the source-microphone distance.An anechoic chamber is a closed room whose design blocks out outside sound or electromagnetic energy. Used to absorb acoustic (sound) echoes caused by internal reflections of a room, to simulate an anechoic chamber we need to ensure the audio has no reflections within the proposed room.

To train our model we needed to simulate anechoic test data from the existing TIMIT Dataset. To simulate our required audio samples, the clean speech signal is superpositioned, appropriately shifted and attenuated for each active source based on the source-microphone distance and source microphone angles. We use the time shift property of fourier transforms .

$$x(t-t_0) \overset{\text{F.T}}{\longleftrightarrow} e^{-jwt_0}X(\omega) \tag{5.2}$$

The shifted waveforms are then attenuated by multiplying the resultant waveform with $(1+cos(x))/2$ where x represents the angle between the microphone and the source. The angle is calculated in two steps:

1. The normal passing through the centre of the room is considered to be the common axis for angle calculation. We then find the angle the source makes with the axis, α_1. Subsequently, we calculate the angle made between the normal which passes through the microphone in question and the axis, α_2.
2. Next, we subtract α_2 from α_1 to get the angle α subtended by the source from the normal of the microphone.

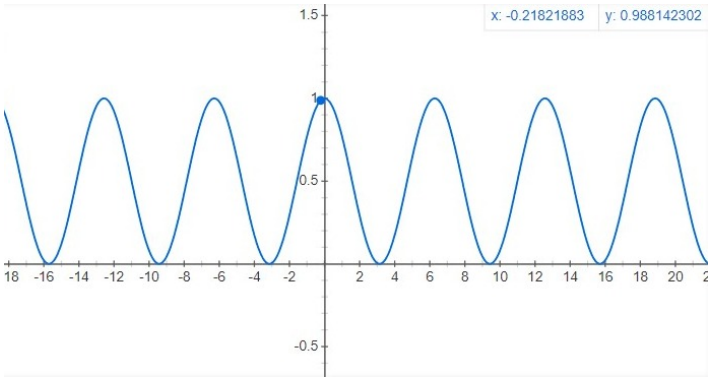


Figure 5.5: Attenuation function

By varying the source position and placing two sources simultaneously, the various datasets were simulated. These simulated datasets will be used in training and testing the CNN model.

Anechoic_Simulation_Code

July 16, 2021

0.0.1 Caculating Angle from Normal

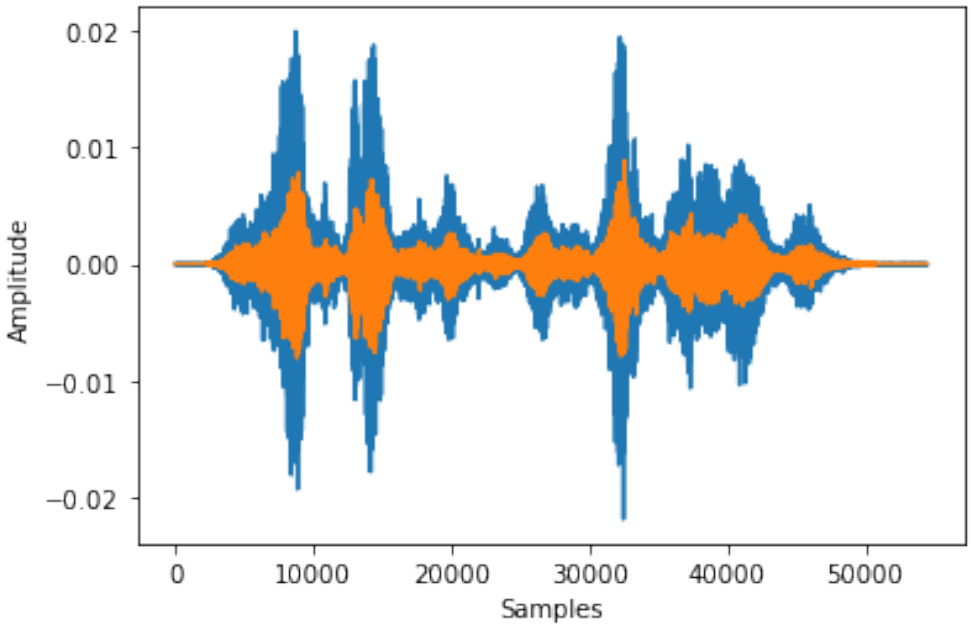
[1]:
```
import math
def angle(custom):
    straight=[]
    center = [3,3.75,2];
    r=math.sqrt((center[0]-custom[0])**2+(center[1]-custom[1])**2)
    straight.append(center[0]);
    straight.append(center[1]+r);
    straight.append(2);
    chord=math.sqrt((straight[0]-custom[0])**2+(straight[1]-custom[1])**2);
    alpha=round((math.acos((2*r**2-chord**2)/(2*r**2))),4)
    if custom[0]>center[0]:
        alpha=alpha+math.pi
    return alpha
```

0.0.2 Anechoic time shift + Attenuation

[4]:
```
def anechoic_attenuation(X,Y,Mx,My,alpha,audio):
    fs = 16000
    D = math.sqrt(np.power(X-Mx,2)+np.power(Y-My,2))
    t = D/340
    tDelayInSamples = math.floor(fs*t)
    fftData = np.fft.fft(audio)
    N = fftData.shape[0]
    k = np.linspace(0, N-1, N)
    timeDelayPhaseShift = np.exp(((-2*np.pi*1j*k*tDelayInSamples)/
 ↪(N))+(tDelayInSamples*np.pi*1j))
    # print(timeDelayPhaseShift)
    #timeDelayPhaseShift = np.fft.fftshift(timeDelayPhaseShift)
    #print(timeDelayPhaseShift)
    #timeDelayPhaseShift = np.fft.fftshift(timeDelayPhaseShift)
    fftWithDelay = np.multiply(fftData, timeDelayPhaseShift)
    shiftedWaveform = np.fft.ifft(fftWithDelay)
    attenuation=(1+math.cos(alpha))/2
    shiftedWaveform=shiftedWaveform*attenuation
    return shiftedWaveform
```

[25]:
```
shifted = anechoic_attenuation(2.23,1.68,2.23,3.5,1.76,y)
```

[28]:
```
plt.plot(y)
plt.plot(shifted)
plt.xlabel("Samples")
plt.ylabel("Amplitude")
```



5.2 Encoding the coordinates of the source

The enclosure is partitioned into L sector like coarse regions denoted by $R_l{}_{l=1}^{l=L}$ centered around the uniform circular array.A region is said to be active if it contains at leat one source and inactive otherwise.The assumption here, is there can be at most one active source in any active region.The 2D schematic of a enclosure partitioned into $L = 8$ sector like regions, along with a UCA of M = 8 microphones and two sources - one in R_1 and the other in R_8 is shown in Figure 5.1.

In figure[5.5] let the 2D coordinates of source in R_l, located at p_{S_l},be (d_l) and

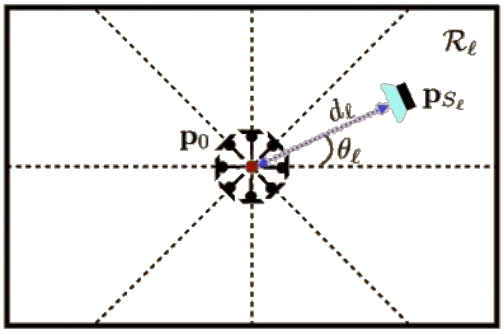


Figure 5.6: Output Encoding

(θ_l).Where d_l is the radial distance and θ_l is the azimuthal angle computed with respect to the center of microphone array (p_o). The d_l and θ_l are given by,

$$d_l = ||p_{S_l} - p_o|| \tag{5.3}$$

$$\theta_l = \angle(p_{S_l} - p_o) \tag{5.4}$$

where $||.||$ is the euclidean distance between the vectors and $\angle$ is the azimuthal angle operator Instead of representing the source coordinates directly, we encode the source coordinates in each sector as:

$$\tilde{d}_l = \frac{d_l - d_l^{min}}{d_l^{max} - d_l^{min}} \tag{5.5}$$

$$\tilde{\theta}_l = \frac{\theta_l - \theta_l^{min}}{\theta_l^{max} - \theta_l^{min}} \tag{5.6}$$

where the limits (d_l^{min}, d_l^{max}) represent the minimum and maximum possible radial distance, and $(\theta_l^{min}, \theta_l^{max})$ represent the minimum and maximum possible azimuthal angle for points in R_l.

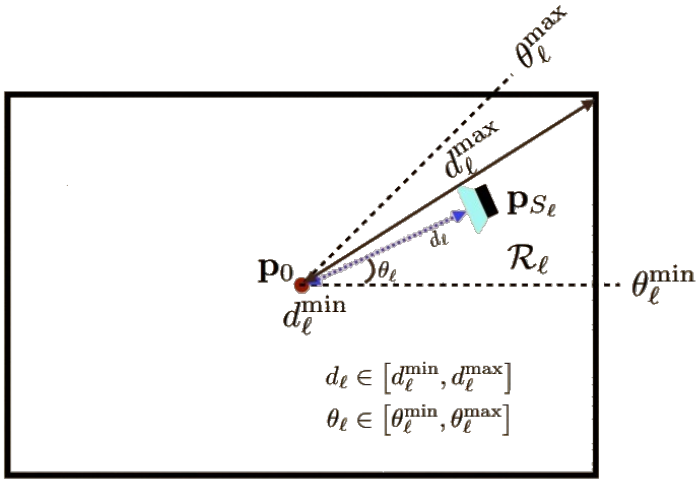


Figure 5.7: Normalizing the source coordinates

5.2.1 Coarse and fine localization

This network architecture presents a Coarse-Fine localization startegy, wherein a source is associated with a coarse and fine location. As presented in the section above instead of the source coordinates directly the encoded coordinates become the output of this architecture.The encoded coordinates along with the active region the architecture detects allows us to localize any source within the 2D schematics.The discretized grid of angular locations at a radial distance avoids the permutation problem for source localization.

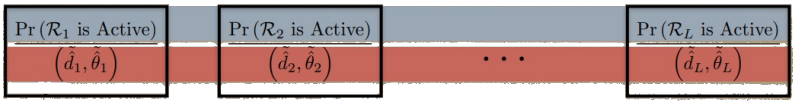


Figure 5.8: Coarse and Fine localization

5.3 Overall Network Architecture

5.3.1 Overview of Problem Statement

In this section, we present the complete architecture used to solve the problem of acoustic source localization in a 2D plane in an end-end fashion. As described in the image above the architecture takes in multichannel raw audio data as input and the custom Sample Level CNN architecture generates 3 outputs, respectively giving us the Coarse and Fine localization for the source present in the multichannel audio input received by the UCA of microphones.The proposed approach is referred to as Sample based Multiple Encoded Source Location Predictor (SMESLP).

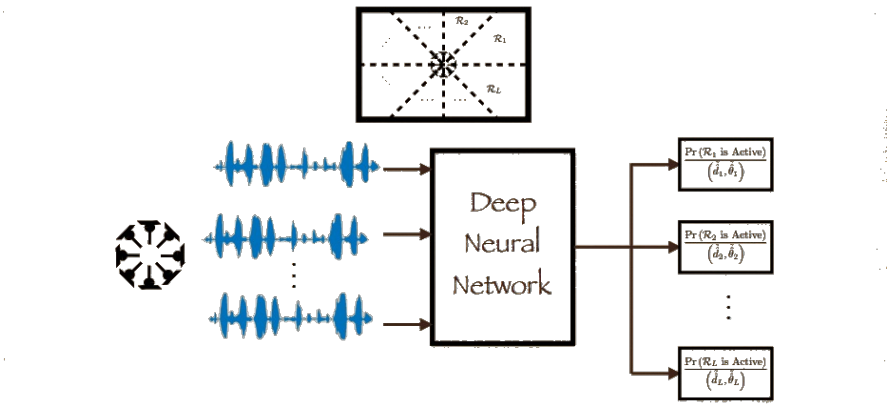


Figure 5.9: Overview

5.3.2 Sample based Multiple Encoded Source Location Predictor (SMESLP)

- The multi-channel audio, forms the input to the network , which is of duration T sec.This input is first processed by a wave-norm layer where each input waveform is normalized to have a maximun of unit amplitude.
- These normalized audio samples are fed to the basic block as in fig 5.5a.containing a single convolutional layer with 64 1D convolutional filters, followed by the batch normalization layer and rectified linear unit (ReLU).
- The output from the basic block is down-sampled by a factor of 3, and fed to a series of residual squeeze and excitation blocks(mentioned in section 4.2.1) containing 2 convolutional layers each and abbreviated as “ReSE-2” blocks. The operations performed by each ReSE-2 block is summarized in Figure 5.5(b) and 5.5(c).
- In each ReSE-2 block the input passes through 2 convolutional layers with 128 1D convolutional filters and is down-sampled by a factor of 3.
- The 2D output of the final ReSE-2 block is then transformed into a 256 dimensional vector, through global max pooling layer and 2 fully connected layers.
- The 2 fully connected layers are used for generating the final classification outputs (referred to as coarse location predictions), and regression outputs (referred to as fine location predictions), to detect active regions and localize the sources finely within each active region, respectively.

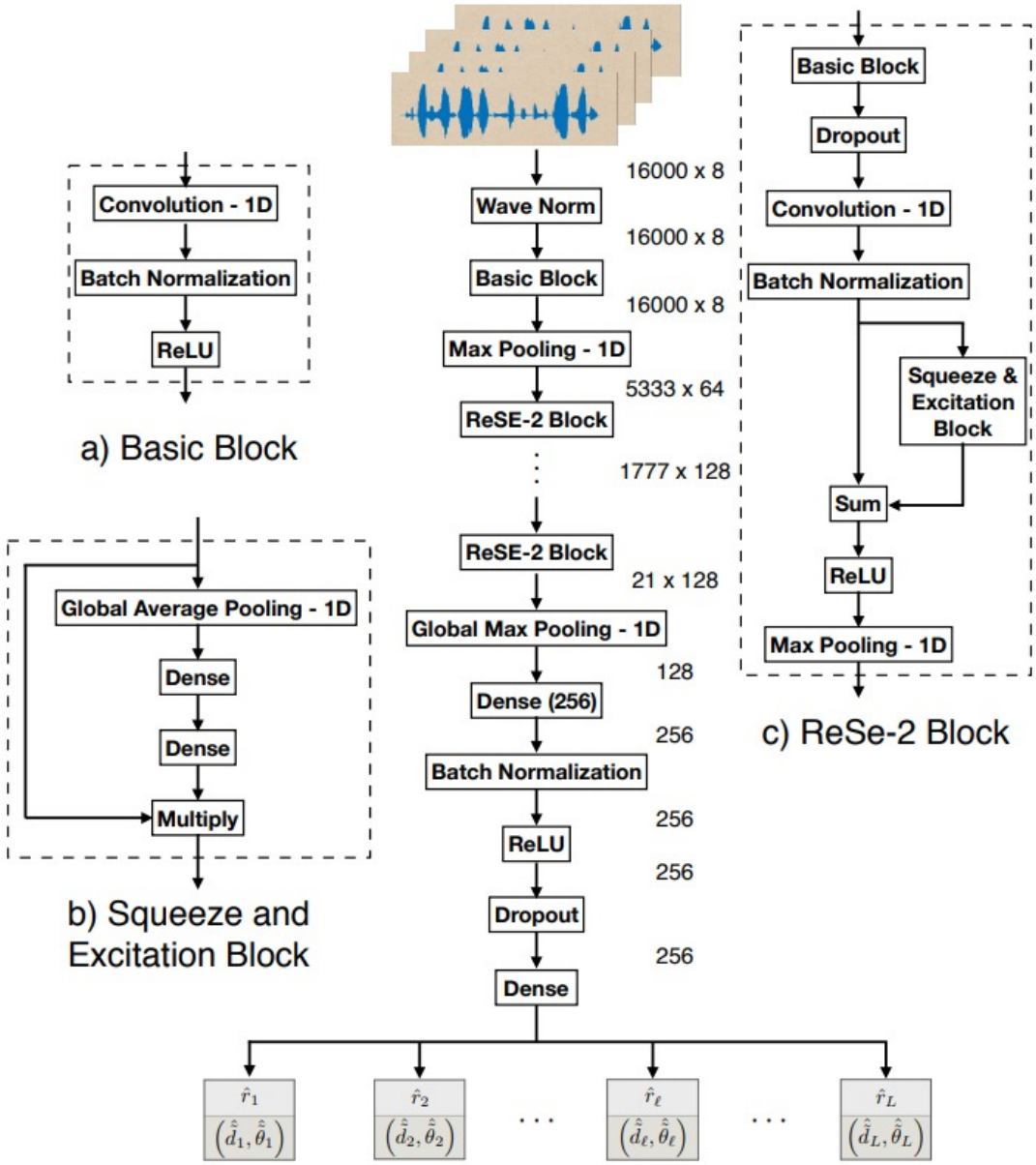


Figure 5.10: Overall architecture

5.3.3 Loss Function

Training Data Split

The training data for proposed network architecture is (W, R, D). W denotes the raw audio sample input into the network, where j in the figure above represents the number of training samples. R is coarse localization and represents the ground truth for the region labels, if $R_l = 1$, it represents an active region and 0 otherwise. D is the encoded 2-D location of the source with radial distance and azimuthal angle being the two coordinates.

For a given raw audio sample, the network architecture gives three outputs :

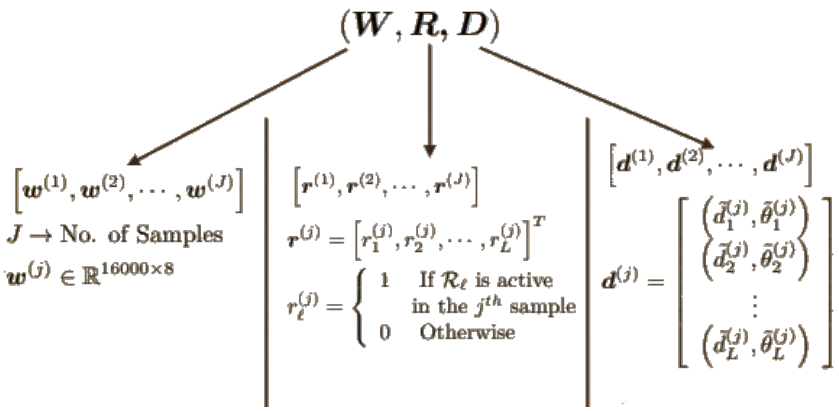


Figure 5.11: Training Data Triplet

- $R_l = 1$, indicating which region is active.
- (d_l, θ_l) is the encoded radial distance and the azimuthal angle.

Defining Loss Function

For the j^{th} training sample, the coarse localization prediction is a classification problem whose predictions are improved with respect to the loss function described below :

$$\mathcal{L}_{Coarse}^{(j)} = -\frac{1}{L}\sum_{l=1}^{L}[r_l^{(j)}\log(\hat{r}_l^{(j)}) + (1-r_l^{(j)})\log(1-\hat{r}_l^{(j)})] \tag{5.7}$$

The average Euclidean distance between the actual and predicted encoded coordinates of sources in active regions is used as a loss function to improve the regression predictions by the model :

$$\mathcal{L}_{Fine}^{(j)} = \frac{1}{L}\sum_{l=1}^{L} 1_{r_l^{(j)}=1}\sqrt{(\tilde{d}_l^{(j)} - \hat{\tilde{d}}_l^{(j)})^2 + (\tilde{\theta}_l^{(j)} - \hat{\tilde{\theta}}_l^{(j)})^2} \tag{5.8}$$

For our implementation we used the in-built *keras* categorical cross entropy for classification and mean square error for regression.

```
loss=[tf.keras.losses.CategoricalCrossentropy(),'mse'],
optimizer='adam',metrics=[tf.keras.metrics.CategoricalAccuracy(),
tf.keras.metrics.RootMeanSquaredError()]
```

5.4 Model Performance Progress

5.4.1 Types of architectures

We constructed the network architecture using keras and tensorflow , they provide three methods to implement a convolutional neural network architecture.During our experiments to improve our architecture's prediction accuracy we implemented all the three methodologies and they are described below.

Sequential model

A Sequential model allows to create model layer by layer in a step fashion.
A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
Using the sequential model we cannot create models that

Figure 5.12: Sequential Model

- Share layers
- Have branches
- Have multiple inputs
- Have multiple outputs

Functional model

The Keras functional model is a way to create models that are more flexible . The functional model can handle models with non-linear topology, shared layers, and even multiple inputs or outputs. Using the Functional model you can:

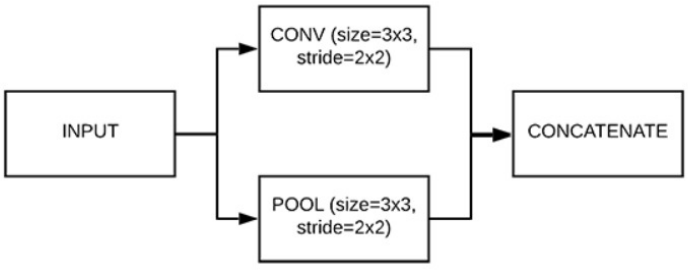


Figure 5.13: Functional Model

- Create more complex models.
- Have multiple inputs and multiple outputs.
- Easily define branches in your architectures .
- Easily share layers inside the architecture.

Model Subclassing

Model subclassing is fully-customizable and enables to implement own custom forward-pass of the model.

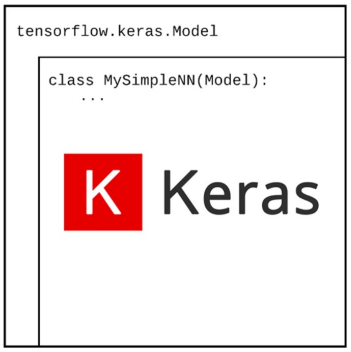


Figure 5.14: Model Subclassing

Trainable Parameters

Model	No.of Trainable Parameters
Sequential (Lamba)	36,134 [11%]
Functional API (Lambda) - With regression branching	37,674 [40%]
Sub Classing Architecture	1,39,082 [48%]
Sub Classing (regression)	1,36,002 [48%]
Functional API (without Lamba)	31,42,082 [85%]

Sequential_Model

July 16, 2021

[ ]:
```
classifier = keras.Sequential()
classifier.add(InputLayer(input_shape=(8,16000)))
classifier.add(BatchNormalization())
classifier.add(basic1)
classifier.add(MaxPool1D(pool_size=3))
classifier.add(rese)
classifier.add(rese)
classifier.add(rese)
classifier.add(rese)
classifier.add(GlobalMaxPool1D())
classifier.add(Dense(units=256, activation='relu'))
classifier.add(BatchNormalization())
classifier.add(Activation(activation))
classifier.add(Dropout(dropout_rate))
classifier.add(Dense(units=8, activation='softmax'))
classifier.compile(loss=custom_loss, optimizer='adam',metrics=[tf.keras.metrics.
↪CategoricalAccuracy()])
```

Functional_Model

July 16, 2021

[ ]:
```
def basic_block(x, num_features=128, weight_decay=0):
    x = Conv1D(num_features, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(weight_decay),␣
 ↪kernel_initializer='he_uniform')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
```

[ ]:
```
def se_fn(x,num_features=128,amplifying_ratio=16):
#     num_features = x.shape[-1].value
    shortcut=x
    num_features=128
    x = GlobalAvgPool1D()(x)
    x = Reshape((1, num_features))(x)
    x = Dense(num_features * amplifying_ratio, activation='relu',␣
 ↪kernel_initializer='glorot_uniform')(x)
    x = Dense(num_features, activation='sigmoid',␣
 ↪kernel_initializer='glorot_uniform')(x)
    x = Multiply()([x,shortcut])
    return x
```

[ ]:
```
def rese_block(x, num_features=128, weight_decay=0, amplifying_ratio=16):
    num_features=128
    x = basic_block(x,num_features,weight_decay)
    x = Dropout(0.2)(x)
    x = Conv1D(num_features, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(weight_decay),␣
 ↪kernel_initializer='he_uniform')(x)
    x = BatchNormalization()(x)
    shortcut=x
    if amplifying_ratio > 0:
        x = se_fn(x, amplifying_ratio)
    x = Add()([shortcut, x])
    x = Activation('relu')(x)
    x = MaxPool1D(pool_size=3)(x)
    return x
```

[ ]:
```
layer1=Input(shape=(16000,8))
layer2=basic_block1(layer1)
layer3=MaxPool1D(pool_size=3,strides=None)(layer2)
layer4=rese_block(layer3)
layer5=rese_block(layer4)
layer6=rese_block(layer5)
layer7=rese_block(layer6)
layer8=rese_block(layer7)
layer9=GlobalMaxPool1D()(layer8)
final=Dense(units=256, activation='relu')(layer9)
layer10=BatchNormalization()(final)
layer11=Activation(activation)(layer10)
layer12=Dropout(dropout_rate)(layer11)
out_class=Dense(units=8, activation='softmax')(layer12)
layer14=BatchNormalization()(final)
layer15=Activation(activation)(layer14)
layer16=Dropout(dropout_rate)(layer15)
out_reg=Dense(units=2, activation='linear')(layer16)
model=keras.Model(inputs=layer1,outputs=[out_class,out_reg])
```

Sub-classing_Model

July 16, 2021

[ ]:
```
class abc(layers.Layer):
    def __init__(self, out_channels, kernel_size=3):
        super(abc, self).__init__()
        self.conv = layers.Conv1D(out_channels, kernel_size, padding="same")
        self.bn = layers.BatchNormalization()

    def call(self, input_tensor, training=False):
        x = self.conv(input_tensor)
        x = self.bn(x, training=training)
        x = tf.nn.relu(x)
        return x
abc1=abc(64)
```

[ ]:
```
class se_fn(layers.Layer):
    def __init__(self,amp_ratio,out_channels):
        self.amp_ratio=amp_ratio
        self.gavgpool=layers.GlobalAveragePooling1D()
        self.dense1=Dense(out_channels * self.amp_ratio, activation='relu',␣
 ↪kernel_initializer='glorot_uniform')
        self.dense2=Dense(out_channels, activation='relu',␣
 ↪kernel_initializer='glorot_uniform')

        self.multiply=layers.Multiply()
        self.out_channels=out_channels
        super(se_fn, self).__init__()

    def call(self, input_tensor, training=False):
        shortcut=input_tensor
        x = self.gavgpool(input_tensor)
#         x = Reshape((1, self.out_channels))(x)
        x=self.dense1(x)
        x=self.dense2(x)
        x=self.multiply([x,shortcut])
        return x


se_fn_block =se_fn(16,128)
```

[ ]:
```
class rese_2(layers.Layer):
    def __init__(self,amplify_ratio,filters):
        self.amplify_ratio=amplify_ratio
        self.filters=filters
        self.maxpool=layers.MaxPooling1D(pool_size=3)
        self.conv1 = layers.Conv1D(128, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(0),
                                   kernel_initializer='he_uniform')

        self.dropout=layers.Dropout(0.1)
        self.bn1 = layers.BatchNormalization()
        self.add=layers.Add()
        super(rese_2, self).__init__()

    def call(self, input_tensor, training=False):
        x=xyz123(input_tensor)
        x=self.dropout(x)
        x=self.conv1(x)
        x=self.bn1(x)
        shortcut=x
        sefn=se_fn_block(x)
        x=self.add([x,sefn])
        x=tf.nn.relu(x)
        x=self.maxpool(x)
        return x


rese2_block =rese_2(16,128)
```

[ ]:
```
class rese_2_new(layers.Layer):
    def __init__(self,amplify_ratio,filters):
        self.amplify_ratio=amplify_ratio
        self.filters=filters
        self.maxpool=layers.MaxPooling1D(pool_size=3)
        self.conv1 = layers.Conv1D(128, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(0),
                                   kernel_initializer='he_uniform')

        self.dropout=layers.Dropout(0.1)
        self.bn1 = layers.BatchNormalization()
        self.add=layers.Add()
        super(rese_2_new, self).__init__()

    def call(self, input_tensor, training=False):
        x=xyz123_new(input_tensor)
        x=self.dropout(x)
        x=self.conv1(x)
```

```
        x=self.bn1(x)
        shortcut=x
        sefn=se_fn_block(x)
        x=self.add([x,sefn])
        x=tf.nn.relu(x)
        x=self.maxpool(x)
        return x


rese2_block_new =rese_2_new(16,128)
```

[ ]:
```
layer1=Input(shape=(16000,8))
layer2=abc1(layer1)
layer3=MaxPool1D(pool_size=3,strides=None)(layer2)
layer4=rese2_block(layer3)
layer5=rese2_block_new(layer4)
layer6=rese2_block_new(layer5)
layer7=rese2_block_new(layer6)
layer8=GlobalMaxPool1D()(layer7)
final=Dense(units=256, activation='relu')(layer8)
layer10=BatchNormalization()(final)
layer11=Activation(activation)(layer10)
layer12=Dropout(dropout_rate)(layer11)
out_class=Dense(units=8, activation='softmax')(layer12)
layer14=BatchNormalization()(final)
layer15=Activation(activation)(layer14)
layer16=Dropout(dropout_rate)(layer15)
out_reg=Dense(units=2, activation='linear')(layer16)
model=tf.keras.Model(inputs=layer1,outputs=[out_class,out_reg])
```

5.4.2 Training of the model

Datasets

To simulate reverberant and anechoic datasets as described in [5.1.2],[5.1.3] respectively, we need to sample random points from each region for the source locations within the room described in [5.1.1].

The reverberant simulations are generated using matlab and the anechoic simulations are generated using python. The random source locations are generated using a python script described below and we then export all the source coordinates in .csv and .mat format for ease of access.
Once the reverberant and anechoic simulations are completed the audio signals are stored in the directory scheme shown below.

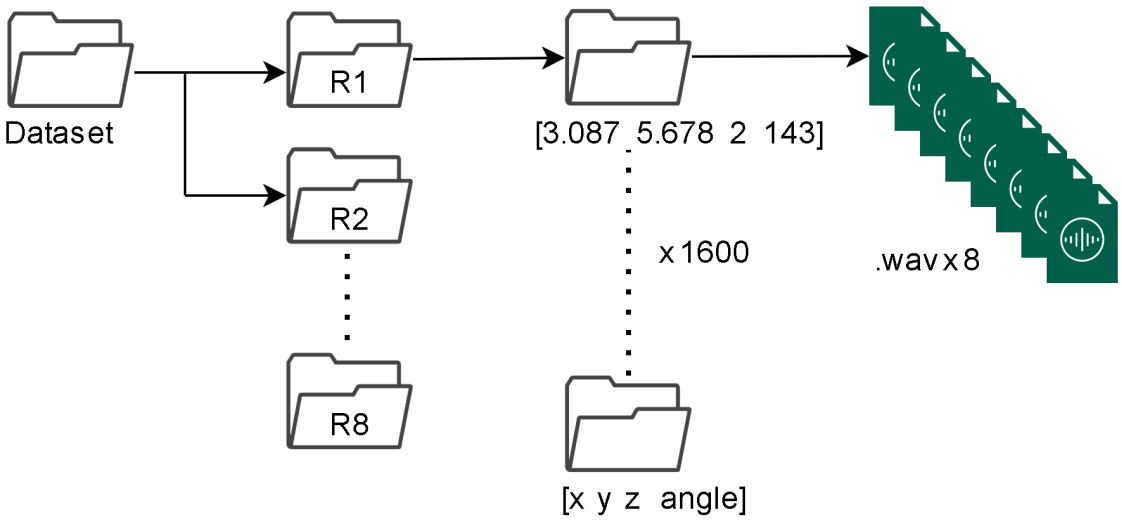


Figure 5.15: Dataset Directory scheme

Random_Source_Coordinate_Generator

July 16, 2021

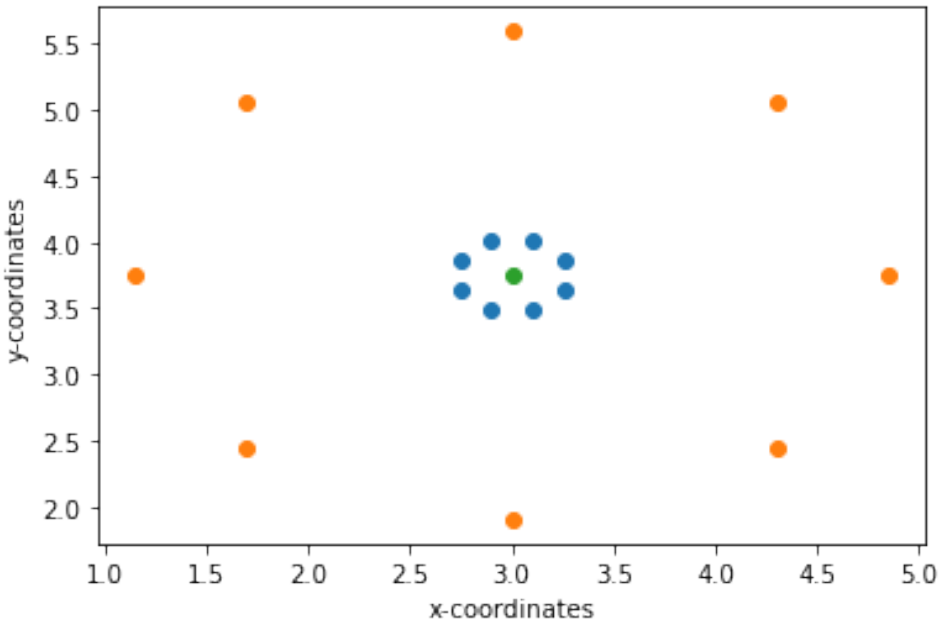
0.0.1 Function to Generate Random Points in a Triangle

[12]:
```
import random
import matplotlib.pyplot as plt
from scipy.io import savemat
import numpy as np
import scipy
from scipy.io import savemat,loadmat

def point_on_triangle(pt1, pt2, pt3):
    """
    Random point on the triangle with vertices pt1, pt2 and pt3.
    """
    s, t = sorted([random.random(), random.random()])
    return (s * pt1[0] + (t-s)*pt2[0] + (1-t)*pt3[0],
            s * pt1[1] + (t-s)*pt2[1] + (1-t)*pt3[1])
```

0.0.2 Uniform Circular Array and Edges of the Room

[14]:
```
X=[4.8478,3.7654,2.2346,1.1522,1.1522,2.2346,3.7654,4.8478]
Y=[4.5154,5.5978,5.5978,4.5154,2.9846,1.9022,1.9022,2.9846]

Mx=[3.2121,3.0,2.7878,2.7,2.7878,3.0,3.2121,3.3]
My=[3.96,4.05,3.96,3.75,3.53,3.45,3.53,3.75]

new_Mx=[]
new_My=[]
new_X=[]
new_Y=[]
for i in range(len(Mx)-1):
    new_Mx.append((Mx[i]+Mx[i+1])/2)
    new_My.append((My[i]+My[i+1])/2)
    new_X.append((X[i]+X[i+1])/2)
    new_Y.append((Y[i]+Y[i+1])/2)
new_Mx.append((Mx[7]+Mx[0])/2)
new_My.append((My[7]+My[0])/2)
new_X.append((X[7]+X[0])/2)
```

```
new_Y.append((Y[7]+Y[0])/2)

plt.scatter(new_Mx,new_My)
plt.scatter(new_X,new_Y)
plt.xlabel('x-coordinates')
plt.ylabel('y-coordinates')
plt.scatter(3,3.75)
X=new_X
Y=new_Y
Mx=new_Mx
My=new_My
```



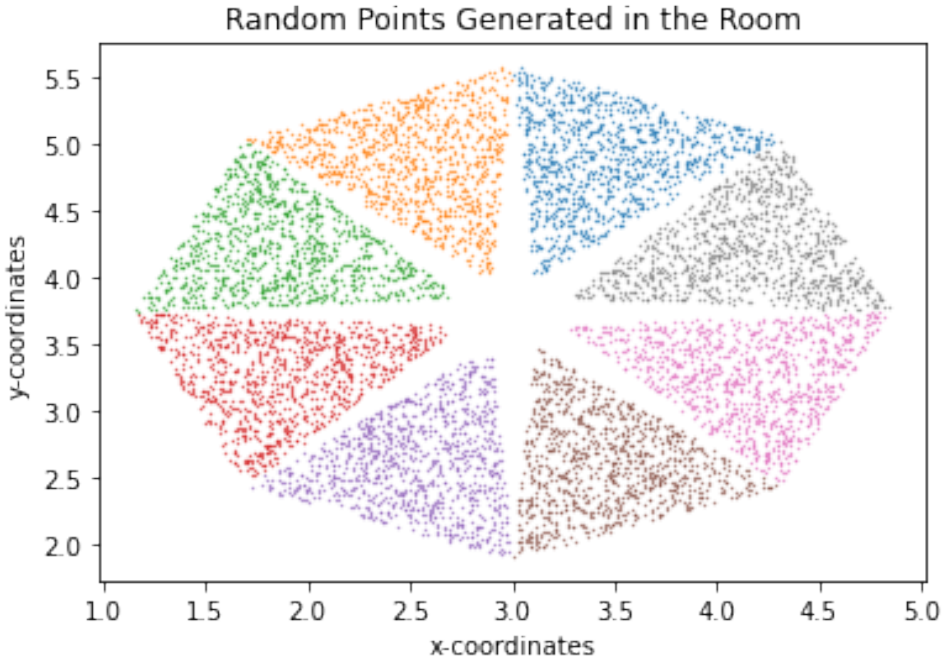
0.0.3 Generating Random Sources in each Region

[4]:
```
points1=[]
M = np.zeros(shape=(8,50))
k=0
for i in range(7):
    pt1 = (X[i], Y[i])
    pt2 = (X[k+1], Y[k+1])
    pt3 = (Mx[i], My[i])
    points = [point_on_triangle(pt1, pt2, pt3) for _ in range(1000)]
    points1.append(points)
    k=k+1
```

```
pt1 = (X[7], Y[7])
pt2 = (X[0], Y[0])
pt3 = (Mx[7], My[7])
points = [point_on_triangle(pt1, pt2, pt3) for _ in range(1000)]
points1.append(points)
```

0.0.4 Plotting a 1000 Random Sources in each Region

[11]:
```
for i in points1:
    x, y = zip(*i)
    plt.scatter(x, y, s=0.1)
plt.xlabel('x-coordinates')
plt.ylabel('y-coordinates')
plt.title('Random Points Generated in the Room')
plt.savefig('room.png')
plt.show()
```



0.0.5 Converting the random source coordinates to MATLAB readable data

[6]:
```
regions = ['R0','R1','R2','R3','R4','R5','R6','R7']
dictp = {i: [] for i in regions}
dictr = {i: [] for i in regions}

k=0
for i in regions:
    if (i=='R7'):
        break
    else:
        for o in range(200):
            pt1 = (X[k], Y[k])
            pt2 = (X[k+1], Y[k+1])
            pt3 = (Mx[k], My[k])
            points = point_on_triangle(pt1, pt2, pt3)
            dictp[i].append(points)
    k=k+1

for o in range(200):
    pt1 = (X[7], Y[7])
    pt2 = (X[0], Y[0])
    pt3 = (Mx[7], My[7])
    points = point_on_triangle(pt1, pt2, pt3)
    dictp['R7'].append(points)

scipy.io.savemat('outRegions.mat', dictr)
scipy.io.savemat('outCoordinates.mat', dictp)

d1 = scipy.io.loadmat('outRegions.mat')
```

Training

As we require both classification and regression tasks to predict the location of the source we need to construct a branched architecture that is capable of generating three outputs :

- Classification for predicting the active region where the source is present.The activation function used here is softmax.
- Multiple regression output to predict the encoded radial distance and azimuthal angle within the active region.Here the activation function used is linear.

Given the constraint of our computing power we decided to train the model seperately for the classification and regression tasks.
Based on trial and error methodology we improved the performance of our architecture in the following ways:

- Transfer Learning: We saved our trained model on a large dataset and then fine-tuned the same model for a smaller dataset. This improved our predictions.
- Learning rate : At each transfer learning stage we changed our learning rate from 0.001,0.0001 to 0.00001.
- Data sets from different instances : To extract maximum information from the audio signal, we took different 1 sec instances from it. This also increased our data size.
- Changing the loss function: Experimented with various loss functions and custom loss function to get the best output.

Final_Complete_Architecture

July 18, 2021

[220]:
```
import sys, os
import numpy as np
import pandas as pd
import numpy as np
import tensorflow as tf
from keras.models import load_model
import wave
import os
import glob
import librosa
import scipy.io.wavfile as wav
import scipy.signal as signal
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import (Conv1D, MaxPool1D,MaxPool2D, BatchNormalization,␣
↪GlobalAvgPool1D, Multiply, GlobalMaxPool1D,
                                    Dense, Dropout, Activation, Reshape,␣
↪Concatenate, Add)
from keras.layers import Input
from keras.utils import np_utils
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import load_model
import pandas as pd
from tensorflow.keras import Model
from keras.layers import InputLayer
from keras.layers import Lambda
from keras import backend as K
from keras.regularizers import l2
import math
from sklearn.datasets import make_regression
from sklearn.metrics import mean_absolute_error
```

0.0.1 Defining Each Layer

[228]:
```
#1. Basic Block 64 features (Input)
def basic_block1(x, num_features=64, weight_decay=0):
    x = Conv1D(num_features, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(weight_decay),␣
 ↪kernel_initializer='he_uniform')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
#2. Basic Block 128 features (Rese)
def basic_block(x, num_features=128, weight_decay=0):
    x = Conv1D(num_features, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(weight_decay),␣
 ↪kernel_initializer='he_uniform')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
#3, Squeeze and Excitation Block
def se_fn(x,num_features=128,amplifying_ratio=16):

    shortcut=x
    num_features=128
    x = GlobalAvgPool1D()(x)
    x = Reshape((1, num_features))(x)
    x = Dense(num_features * amplifying_ratio, activation='relu',␣
 ↪kernel_initializer='glorot_uniform')(x)
    x = Dense(num_features, activation='sigmoid',␣
 ↪kernel_initializer='glorot_uniform')(x)
    x = Multiply()([x,shortcut])
    return x
#Rese Block
def rese_block(x, num_features=128, weight_decay=0, amplifying_ratio=16):
    num_features=128
    x = basic_block(x,num_features,weight_decay)
    x = Dropout(0.2)(x)
    x = Conv1D(num_features, kernel_size=3, padding='same',␣
 ↪use_bias=True,kernel_regularizer=l2(weight_decay),␣
 ↪kernel_initializer='he_uniform')(x)
    x = BatchNormalization()(x)
    shortcut=x
    if amplifying_ratio > 0:
        x = se_fn(x, amplifying_ratio)
    x = Add()([shortcut, x])
    x = Activation('relu')(x)
    x = MaxPool1D(pool_size=3)(x)
    return x
```

```
kernel_initializer=tf.keras.initializers.GlorotUniform()
activation=keras.activations.relu
dropout_rate=0.1
```

0.0.2 Function To Build Model Based on Training Requirements

[229]:
```
def build_model(s):

    layer1=Input(shape=(16000,8))
    layer2=basic_block1(layer1)
    layer3=MaxPool1D(pool_size=3,strides=None)(layer2)
    layer4=rese_block(layer3)
    layer5=rese_block(layer4)
    layer6=rese_block(layer5)
    layer7=rese_block(layer6)
    layer8=rese_block(layer7)
    layer9=GlobalMaxPool1D()(layer8)
    final=Dense(units=256, activation='relu')(layer9)
    # 'c' is used only for classification task
    if (s == 'c'):
        layer10=BatchNormalization()(final)
        layer11=Activation(activation)(layer10)
        layer12=Dropout(dropout_rate)(layer11)
        out_class=Dense(units=8, activation='softmax')(layer12)
        model=keras.Model(inputs=layer1,outputs=out_class)
    elif (s == 'r'):
    # 'r' is used only for regression task
        layer14=BatchNormalization()(final)
        layer15=Activation(activation)(layer14)
        layer16=Dropout(dropout_rate)(layer15)
        out_reg=Dense(units=1, activation='linear')(layer16)
        model=keras.Model(inputs=layer1,outputs=out_reg)
    # Combination of classification and regression
    else:
        layer10=BatchNormalization()(final)
        layer11=Activation(activation)(layer10)
        layer12=Dropout(dropout_rate)(layer11)
        out_class=Dense(units=8, activation='softmax')(layer12)
        # Branching for regression
        layer14=BatchNormalization()(final)
        layer15=Activation(activation)(layer14)
        layer16=Dropout(dropout_rate)(layer15)
        out_reg=Dense(units=2, activation='linear')(layer16)
        model=keras.Model(inputs=layer1,outputs=[out_class,out_reg])

    return model
```

[5]:
```
model = build_model('')
```

[6]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
=================================================================================
==================
input_1 (InputLayer)            [(None, 16000, 8)]   0
_________________________________________________________________________________
__________________
conv1d (Conv1D)                 (None, 16000, 64)    1600        input_1[0][0]
_________________________________________________________________________________
__________________
batch_normalization (BatchNorma (None, 16000, 64)    256         conv1d[0][0]
_________________________________________________________________________________
__________________
activation (Activation)         (None, 16000, 64)    0
batch_normalization[0][0]
_________________________________________________________________________________
__________________
max_pooling1d (MaxPooling1D)    (None, 5333, 64)     0
activation[0][0]
_________________________________________________________________________________
__________________
conv1d_1 (Conv1D)               (None, 5333, 128)    24704
max_pooling1d[0][0]
_________________________________________________________________________________
__________________
batch_normalization_1 (BatchNor (None, 5333, 128)    512         conv1d_1[0][0]
_________________________________________________________________________________
__________________
activation_1 (Activation)       (None, 5333, 128)    0
batch_normalization_1[0][0]
_________________________________________________________________________________
__________________
dropout (Dropout)               (None, 5333, 128)    0
activation_1[0][0]
_________________________________________________________________________________
__________________
conv1d_2 (Conv1D)               (None, 5333, 128)    49280       dropout[0][0]
_________________________________________________________________________________
__________________
batch_normalization_2 (BatchNor (None, 5333, 128)    512         conv1d_2[0][0]
_________________________________________________________________________________
```

```
__________________
global_average_pooling1d (Globa (None, 128)          0
batch_normalization_2[0][0]
_____________________________________________________________________________
__________________
reshape (Reshape)               (None, 1, 128)       0
global_average_pooling1d[0][0]
_____________________________________________________________________________
__________________
dense (Dense)                   (None, 1, 2048)      264192      reshape[0][0]
_____________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 1, 128)       262272      dense[0][0]
_____________________________________________________________________________
__________________
multiply (Multiply)             (None, 5333, 128)    0           dense_1[0][0]
batch_normalization_2[0][0]
_____________________________________________________________________________
__________________
add (Add)                       (None, 5333, 128)    0
batch_normalization_2[0][0]
                                                                 multiply[0][0]
_____________________________________________________________________________
__________________
activation_2 (Activation)       (None, 5333, 128)    0           add[0][0]
_____________________________________________________________________________
__________________
max_pooling1d_1 (MaxPooling1D)  (None, 1777, 128)    0
activation_2[0][0]
_____________________________________________________________________________
__________________
conv1d_3 (Conv1D)               (None, 1777, 128)    49280
max_pooling1d_1[0][0]
_____________________________________________________________________________
__________________
batch_normalization_3 (BatchNor (None, 1777, 128)    512         conv1d_3[0][0]
_____________________________________________________________________________
__________________
activation_3 (Activation)       (None, 1777, 128)    0
batch_normalization_3[0][0]
_____________________________________________________________________________
__________________
dropout_1 (Dropout)             (None, 1777, 128)    0
activation_3[0][0]
_____________________________________________________________________________
__________________
conv1d_4 (Conv1D)               (None, 1777, 128)    49280       dropout_1[0][0]
_____________________________________________________________________________
```

```
__________________
batch_normalization_4 (BatchNor (None, 1777, 128)    512         conv1d_4[0][0]
____________________________________________________________________________
__________________
global_average_pooling1d_1 (Glo (None, 128)          0
batch_normalization_4[0][0]
____________________________________________________________________________
__________________
reshape_1 (Reshape)             (None, 1, 128)       0
global_average_pooling1d_1[0][0]
____________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 1, 2048)      264192      reshape_1[0][0]
____________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 1, 128)       262272      dense_2[0][0]
____________________________________________________________________________
__________________
multiply_1 (Multiply)           (None, 1777, 128)    0           dense_3[0][0]
batch_normalization_4[0][0]
____________________________________________________________________________
__________________
add_1 (Add)                     (None, 1777, 128)    0
batch_normalization_4[0][0]
multiply_1[0][0]
____________________________________________________________________________
__________________
activation_4 (Activation)       (None, 1777, 128)    0           add_1[0][0]
____________________________________________________________________________
__________________
max_pooling1d_2 (MaxPooling1D)  (None, 592, 128)     0
activation_4[0][0]
____________________________________________________________________________
__________________
conv1d_5 (Conv1D)               (None, 592, 128)     49280
max_pooling1d_2[0][0]
____________________________________________________________________________
__________________
batch_normalization_5 (BatchNor (None, 592, 128)     512         conv1d_5[0][0]
____________________________________________________________________________
__________________
activation_5 (Activation)       (None, 592, 128)     0
batch_normalization_5[0][0]
____________________________________________________________________________
__________________
dropout_2 (Dropout)             (None, 592, 128)     0
activation_5[0][0]
____________________________________________________________________________
```

```
__________________
conv1d_6 (Conv1D)               (None, 592, 128)     49280       dropout_2[0][0]
_____________________________________________________________________________
__________________
batch_normalization_6 (BatchNor (None, 592, 128)     512         conv1d_6[0][0]
_____________________________________________________________________________
__________________
global_average_pooling1d_2 (Glo (None, 128)          0
batch_normalization_6[0][0]
_____________________________________________________________________________
__________________
reshape_2 (Reshape)             (None, 1, 128)       0
global_average_pooling1d_2[0][0]
_____________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 1, 2048)      264192      reshape_2[0][0]
_____________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 1, 128)       262272      dense_4[0][0]
_____________________________________________________________________________
__________________
multiply_2 (Multiply)           (None, 592, 128)     0           dense_5[0][0]
batch_normalization_6[0][0]
_____________________________________________________________________________
__________________
add_2 (Add)                     (None, 592, 128)     0
batch_normalization_6[0][0]
multiply_2[0][0]
_____________________________________________________________________________
__________________
activation_6 (Activation)       (None, 592, 128)     0           add_2[0][0]
_____________________________________________________________________________
__________________
max_pooling1d_3 (MaxPooling1D)  (None, 197, 128)     0
activation_6[0][0]
_____________________________________________________________________________
__________________
conv1d_7 (Conv1D)               (None, 197, 128)     49280
max_pooling1d_3[0][0]
_____________________________________________________________________________
__________________
batch_normalization_7 (BatchNor (None, 197, 128)     512         conv1d_7[0][0]
_____________________________________________________________________________
__________________
activation_7 (Activation)       (None, 197, 128)     0
batch_normalization_7[0][0]
_____________________________________________________________________________
__________________
```

```
dropout_3 (Dropout)             (None, 197, 128)     0
activation_7[0][0]
__________________________________________________________________________________
__________________
conv1d_8 (Conv1D)               (None, 197, 128)     49280       dropout_3[0][0]
__________________________________________________________________________________
__________________
batch_normalization_8 (BatchNor (None, 197, 128)     512         conv1d_8[0][0]
__________________________________________________________________________________
__________________
global_average_pooling1d_3 (Glo (None, 128)          0
batch_normalization_8[0][0]
__________________________________________________________________________________
__________________
reshape_3 (Reshape)             (None, 1, 128)       0
global_average_pooling1d_3[0][0]
__________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 1, 2048)      264192      reshape_3[0][0]
__________________________________________________________________________________
__________________
dense_7 (Dense)                 (None, 1, 128)       262272      dense_6[0][0]
__________________________________________________________________________________
__________________
multiply_3 (Multiply)           (None, 197, 128)     0           dense_7[0][0]
batch_normalization_8[0][0]
__________________________________________________________________________________
__________________
add_3 (Add)                     (None, 197, 128)     0
batch_normalization_8[0][0]
multiply_3[0][0]
__________________________________________________________________________________
__________________
activation_8 (Activation)       (None, 197, 128)     0           add_3[0][0]
__________________________________________________________________________________
__________________
max_pooling1d_4 (MaxPooling1D)  (None, 65, 128)      0
activation_8[0][0]
__________________________________________________________________________________
__________________
conv1d_9 (Conv1D)               (None, 65, 128)      49280
max_pooling1d_4[0][0]
__________________________________________________________________________________
__________________
batch_normalization_9 (BatchNor (None, 65, 128)      512         conv1d_9[0][0]
__________________________________________________________________________________
__________________
activation_9 (Activation)       (None, 65, 128)      0
```

```
batch_normalization_9[0][0]
__________________________________________________________________________________
__________________
dropout_4 (Dropout)             (None, 65, 128)      0
activation_9[0][0]
__________________________________________________________________________________
__________________
conv1d_10 (Conv1D)              (None, 65, 128)      49280       dropout_4[0][0]
__________________________________________________________________________________
__________________
batch_normalization_10 (BatchNo (None, 65, 128)      512         conv1d_10[0][0]
__________________________________________________________________________________
__________________
global_average_pooling1d_4 (Glo (None, 128)          0
batch_normalization_10[0][0]
__________________________________________________________________________________
__________________
reshape_4 (Reshape)             (None, 1, 128)       0
global_average_pooling1d_4[0][0]
__________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 1, 2048)      264192      reshape_4[0][0]
__________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 1, 128)       262272      dense_8[0][0]
__________________________________________________________________________________
__________________
multiply_4 (Multiply)           (None, 65, 128)      0           dense_9[0][0]
batch_normalization_10[0][0]
__________________________________________________________________________________
__________________
add_4 (Add)                     (None, 65, 128)      0
batch_normalization_10[0][0]
multiply_4[0][0]
__________________________________________________________________________________
__________________
activation_10 (Activation)      (None, 65, 128)      0           add_4[0][0]
__________________________________________________________________________________
__________________
max_pooling1d_5 (MaxPooling1D)  (None, 21, 128)      0
activation_10[0][0]
__________________________________________________________________________________
__________________
global_max_pooling1d (GlobalMax (None, 128)          0
max_pooling1d_5[0][0]
__________________________________________________________________________________
__________________
dense_10 (Dense)                (None, 256)          33024
```

```
global_max_pooling1d[0][0]
__________________________________________________________________________________
__________________
batch_normalization_11 (BatchNo (None, 256)          1024        dense_10[0][0]
__________________________________________________________________________________
__________________
batch_normalization_12 (BatchNo (None, 256)          1024        dense_10[0][0]
__________________________________________________________________________________
__________________
activation_11 (Activation)      (None, 256)          0
batch_normalization_11[0][0]
__________________________________________________________________________________
__________________
activation_12 (Activation)      (None, 256)          0
batch_normalization_12[0][0]
__________________________________________________________________________________
__________________
dropout_5 (Dropout)             (None, 256)          0
activation_11[0][0]
__________________________________________________________________________________
__________________
dropout_6 (Dropout)             (None, 256)          0
activation_12[0][0]
__________________________________________________________________________________
__________________
dense_11 (Dense)                (None, 8)            2056        dropout_5[0][0]
__________________________________________________________________________________
__________________
dense_12 (Dense)                (None, 2)            514         dropout_6[0][0]
==================================================================================
==================
Total params: 3,145,162
Trainable params: 3,141,450
Non-trainable params: 3,712
__________________________________________________________________________________
__________________
```

0.0.3 Loading the Training Data

[7]:
```
with open('/Users/dralbens/Downloads/Reverberant.npy', 'rb') as f:
    g = np.load(f)
```

[20]:
```
r = [6,7,0,5,2,3,4,1]
y = []
regions = np.array([1,0,0,0,0,0,0,0])
for k in r:
    for i in range(200):
        o = np.roll(regions,k)
        y.append(o)
regs = []
for k in range(3):
    for i in y:
        regs.append(list(i))
regs = np.asarray(regs)
```

[238]:
```
df = pd.read_csv('/Users/dralbens/Desktop/Trained_Noel/angle_Anechoic.csv')
df1 =pd.read_csv('/Users/dralbens/Desktop/Trained_Noel/Distance_Anechoic.csv')
```

[242]:
```
angle = df['angle']
distance = df1['dist']
data = [df["angle"], df1["dist"]]
headers = ["ANGLE", "DISTANCE"]
df3 = pd.concat(data, axis=1, keys=headers)
print(len(df3))
df3
```

```
4800
```

[242]:

	ANGLE	DISTANCE
0	0.768391	0.408292
1	0.696835	0.406863
2	0.016670	0.010206
3	0.579824	0.278332
4	0.742671	0.497308
...	...	...
4795	0.354086	0.201728
4796	0.933664	0.033747
4797	0.420803	0.336969
4798	0.624904	0.169447
4799	0.170612	0.533778

[4800 rows x 2 columns]

0.0.4 Splitting and shuffling the loaded datasets into train and test sets

[128]:
```
#model.compile(,loss=['mse'], optimizer='adam',metrics=[tf.keras.metrics.
 ↪MeanAbsolutePercentageError(name="mean_absolute_percentage_error")])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train_c, y_test_c, y_train_r, y_test_r =␣
 ↪train_test_split(g,regs,df3, test_size=0.15, shuffle=True)
```

0.0.5 Compiling the model

[132]:
```
model.compile(loss=[tf.keras.losses.CategoricalCrossentropy(),'mse'],␣
 ↪optimizer='adam',metrics=[tf.keras.metrics.CategoricalAccuracy(),tf.keras.
 ↪metrics.RootMeanSquaredError()])
```

[133]:
```
y_train_r = np.asarray(y_train_r)
#All the training data in same datatype
y_train_r
```

[133]:
```
array([[0.26788726, 0.25566391],
       [0.85573728, 0.82931898],
       [0.24980024, 0.0885637 ],
       ...,
       [0.31920349, 0.75693779],
       [0.67711581, 0.11893599],
       [0.59193608, 0.26596699]])
```

0.0.6 Training The Model and Saving

[134]:
```
model.fit(X_train,[y_train_c,y_train_r],epochs=50,batch_size=8)
model.save('/Users/dralbens/Desktop/Trained_Noel/Complete_with_twist.h5')
```

```
Epoch 50/50
510/510 [==============================] - 268s 526ms/step - loss: 0.0818 -
dense_11_loss: 0.0288 - dense_12_loss: 0.0531 - dense_11_categorical_accuracy:
0.8572 - dense_11_root_mean_squared_error: 0.1072 -
dense_12_categorical_accuracy: 0.8572 - dense_12_root_mean_squared_error: 0.1072


/Users/dralbens/opt/anaconda3/lib/python3.8/site-
packages/keras/utils/generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
```

```
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '
```

0.0.7 Using the Model to predict Test data

[163]:
```
ans=model.predict(X_test)
```

[155]:
```
predictions = ans[0]
```

[156]:
```
predictions
```

[156]:
```
array([[1.78337050e-05, 1.10600865e-06, 1.11889054e-09, ...,
        1.46378397e-07, 9.99974489e-01, 4.21566983e-06],
       [1.18810337e-10, 5.44337263e-06, 9.99994397e-01, ...,
        3.84418364e-08, 1.98450180e-11, 1.16446095e-07],
       [1.68314722e-12, 9.99999642e-01, 1.82523845e-08, ...,
        3.11158061e-07, 2.34075819e-15, 3.02036973e-14],
       ...,
       [3.96985549e-17, 1.13715898e-10, 1.98726278e-11, ...,
        1.00000000e+00, 1.32028355e-09, 1.37895812e-13],
       [9.92097446e-11, 1.00000000e+00, 4.32773062e-09, ...,
        8.82849793e-10, 4.64171429e-15, 3.38762167e-13],
       [7.47450539e-12, 6.28426555e-09, 6.98782401e-07, ...,
        1.47493353e-07, 2.12522604e-08, 8.33690379e-08]], dtype=float32)
```

0.0.8 Threshold classification from the prediction results

[157]:
```
k=0

for i in predictions:
    m = np.argmax(i)
    predictions[k,m]=1
    k=k+1

for i in predictions:
    for k in range(len(i)):
        if i[k]!=1:
            i[k]=0
pre=predictions.astype(int)
print(pre)
fact=[]
count=0

for i,k in zip(predictions,y_test_c):
```

```
    fact.append(i==k)
    if(i.all()==k.all()):
        count=count+1
predictions=np.asarray(fact)
count=0
for i in predictions:
    if np.all(i)==True:
        count=count+1
```

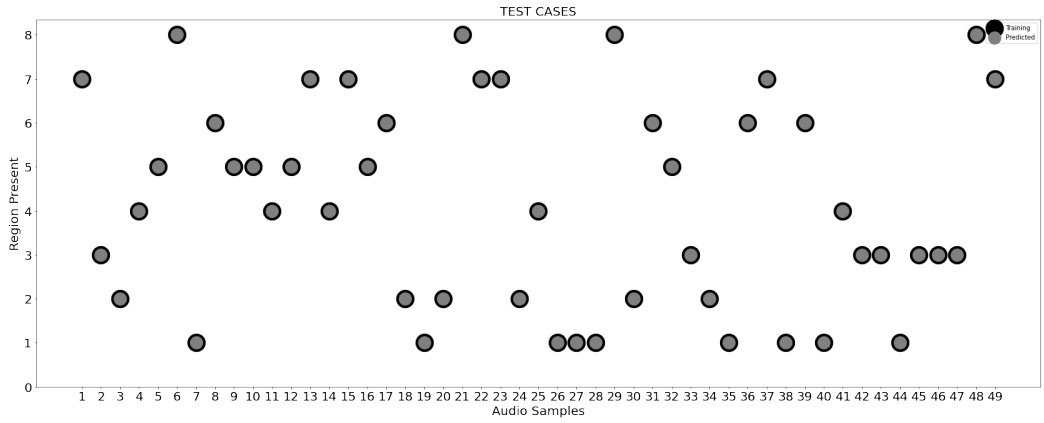
```
[[0 0 0 ... 0 1 0]
 [0 0 1 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

0.0.9 Visualizing the Classification Accuracy

[212]:
```
def plotter(x,y):
    post1 = []
    for i in y:
        m = np.where(i==1)
        post1.append(int(m[0])+1)
    post = []
    for i in x:
        m = np.where(i==1)
        post.append(int(m[0])+1)
    f = np.arange(1,len(x)+1)

    plt.scatter(f,post,label="Training",s = 800,color='black')
    plt.scatter(f,post1,label="Predicted",s=400,color='grey')

    plt.legend()
    plt.yticks([0,1,2,3,4,5,6,7,8],fontsize=20)
    plt.xlabel("Audio Samples").set_size(20)
    plt.ylabel("Region Present").set_size(20)
    f1 = np.arange(1,len(x)+1)
    plt.xticks(f1,fontsize=20)

    plt.rcParams["figure.figsize"] = (28.8,10.8)
```

[256]:
```
k=0
savers = ['testone','testtwo']
```

```
savers1 = ['/Users/dralbens/Desktop/Trained_Noel/setone.png','/Users/dralbens/
 ↪Desktop/Trained_Noel/settwo.png','/Users/dralbens/Desktop/Trained_Noel/
 ↪setthree.png','/Users/dralbens/Desktop/Trained_Noel/setfour.png','/Users/
 ↪dralbens/Desktop/Trained_Noel/setfive.png','/Users/dralbens/Desktop/
 ↪Trained_Noel/setsix.png','/Users/dralbens/Desktop/Trained_Noel/setseven.png','/
 ↪Users/dralbens/Desktop/Trained_Noel/set8.png']
i=0
for z in range(1):
    i=i+49

    plt.figure()
    plt.title('TEST CASES').set_size(20)
    plotter(pre[k:i],y_test_c[k:i])
    plt.savefig(savers1[z],dpi=100)
    k=i+1
```



0.0.10 Picking the Regions that were predicted accurately from testdata

[192]:
```
z=[]

for i in range(len(fact)):
    if (np.all(fact[i],axis=0)):
        z.append(i)
```

[193]:
```
result=[]
for i in z:
    res=np.where(pre[i]==1)
    res=res[0]+1
```

```
    result.append(res[0])
```

[196]:
```
anglea=[]
anglep=[]
#y_train_r=y_train_r.to_numpy(dtype='float32')

for i in z:
    anglea.append(y_train_r[i][0])
    anglep.append(ans[1][i][0])
anglep=np.array(anglep)
```

0.0.11 Decoding the angle regression predictions

[201]:
```
angmax=0
angmin=0
actualdeg=[]
preddeg=[]
for i in range(len(result)):
    if(result[i]==1):
        angmax=315
        angmin=270
    elif(result[i]==2):
        angmax=360
        angmin=315
    elif(result[i]==3):
        angmax=45
        angmin=0
    elif(result[i]==4):
        angmax=90
        angmin=45
    elif(result[i]==5):
        angmax=135
        angmin=90
    elif(result[i]==6):
        angmax=180
        angmin=135
    elif(result[i]==7):
        angmax=225
        angmin=180
    else:
        angmax=270
        angmin=225
    actualdeg.append(anglea[i]*45+angmin)
    preddeg.append(anglep[i]*45+angmin)
```

0.0.12 Saving the Predictions into csv format for comparison

[247]:
```
df = pd.DataFrame(list(zip(actualdeg,preddeg)),columns=['actual','predicted'])
df.to_csv('/Users/dralbens/Desktop/Trained_Noel/Compare_Angle.csv')
df[0:10]
```

[247]:

	actual	predicted
0	192.054927	202.376824
1	38.508178	22.466704
2	326.241011	338.056743
3	64.818610	67.291650
4	111.462894	113.441276
5	248.720242	247.026224

```
6  294.683022  292.628518
7  146.740221  156.180672
8   97.987347  112.989559
9   92.727174  113.724139
```

0.0.13 Calculating Mean Absolute Error in Degrees for Angle Regression Outputs

[227]:
```
mae = mean_absolute_error(actualdeg,preddeg)
mae
```

[227]:
```
10.27155641322805
```

0.0.14 Decoding the Distance regression outputs

[161]:
```
dmax = 2.557999
dmin=0.0055702760952
```

[180]:
```
anstest_dist=[]
for i in range(len(y_test_r)):
    temp = ans[1][i][1]*(dmax-dmin)+dmin
    anstest_dist.append(temp)
```

[182]:
```
y_test_r = np.asarray(y_test_r)
```

[184]:
```
ytest_dist=[]
for i in range(len(y_test_r)):
    temp = y_test_r[i][1]*(dmax-dmin)+dmin
    ytest_dist.append(temp)
```

0.0.15 Calculating Mean Absolute Error for Distance prediction

[222]:
```
mae =mean_absolute_error(anstest_dist,ytest_dist)
```

[223]:
```
mae
```

[223]:
```
0.38424164178578835
```

0.0.16 Saving the distance regression outputs in csv format for comparison

[246]:
```
df = pd.DataFrame(list(zip(ytest_dist,anstest_dist)), columns =␣
 ↪['Actual','Predicted'])
df.to_csv('/Users/dralbens/Desktop/Trained_Noel/Compare.csv')
df[0:10]
```

[246]:

	Actual	Predicted
0	1.641076	0.860226
1	0.390677	0.812933
2	1.101486	0.667248
3	1.981481	1.751100
4	1.623022	1.336593
5	0.693736	1.734495
6	1.401024	1.738194
7	0.127814	0.706831
8	0.632386	1.453747
9	1.338522	1.497801

0.0.17 Extracting the predicted regression outputs to plot the predicted source Locations

[252]:
```
def sc_plot(pred,act,color1,color2):

    for i in range(len(pred)):
        p=plt.scatter(pred[i][0],pred[i][1],c=color1)
        a=plt.scatter(act[i][0],act[i][1],c=color2)
    plt.legend((p,a),['predicted','actual'])
```

0.0.18 UCA Locations and ROOM edges

[253]:
```
M = [4.3066, 1.6934]
M1 = [5.0565,2.4434]
M2 = [1.6934, 4.3066]
M3 = [5.0565999995, 2.4434]
M4=[3.0,3.0]
M5 = [5.5978,1.9022]
M6 = [4.8478,1.1522]
M7=[3.75,3.75]
Mx=np.asarray([3.2121,3.0,2.7878,2.7,2.7878,3.0,3.2121,3.3,3])
My=np.asarray([3.96,4.05,3.96,3.75,3.53,3.45,3.53,3.75,3.75])
```

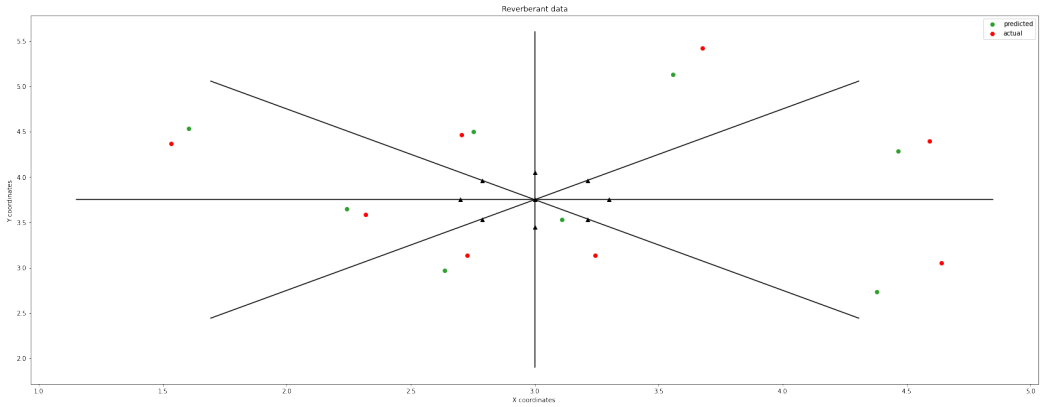
0.0.19 Projecting radial distance and angle on the X and Y axis

[254]:
```
def plotpoint(r,theta):
    center=[3,3.75]
    if(theta<90):
        theta=theta*(math.pi/180)
        y=3.75+(r*math.cos(theta))
        x=3-(r*math.sin(theta))
    elif(theta>90 and theta<180):
        theta =theta-90
        theta=theta*(math.pi/180)
        y=3.75-(r*math.sin(theta))
```

```
        x=3-(r*math.cos(theta))
    elif(theta>180 and theta<270):
        theta =theta-180
        theta=theta*(math.pi/180)
        y=3.75-(r*math.cos(theta))
        x=3+(r*math.sin(theta))
    else:
        theta =theta-270
        theta=theta*(math.pi/180)
        y=3.75+(r*math.sin(theta))
        x=3+(r*math.cos(theta))
    return(x,y)
```

[255]:
```
plt.plot(M,M1,'ko',color='white')
plt.plot(M,M1,color='black')
Mlinex = [M[0],M4[0],M2[0],M6[1],M[1],M4[1],M2[1],M6[0],M[0]]
Mliney = [M1[0],M5[0],M3[0],M7[1],M1[1],M5[1],M3[1],M7[0],M1[0]]
# plt.plot(Mlinex,Mliney,'--')
plt.plot(M2,M3,'ko',color='white')
plt.plot(M4,M5,'ko',color='white')
plt.plot(M6,M7,'ko',color='white')
plt.plot(M2,M3,color='black')
plt.plot(M4,M5,color='black')
plt.plot(M6,M7,color='black')
plt.scatter(Mx,My,marker='^',color='#000000')

pred=[]
act=[]
act.append(plotpoint(1.716,292))
pred.append(plotpoint(1.56,290))
act.append(plotpoint(0.67,156))
pred.append(plotpoint(0.86,155))
act.append(plotpoint(0.7757,22.32))
pred.append(plotpoint(0.789,18.24))
act.append(plotpoint(1.784,246.9))
pred.append(plotpoint(1.714,233.60))
act.append(plotpoint(0.6603,201.7))
pred.append(plotpoint(0.2466,206.6))
act.append(plotpoint(0.7021,103.49))
pred.append(plotpoint(0.7660981,97.688))
act.append(plotpoint(1.80,337.9))
pred.append(plotpoint(1.49,338))
act.append(plotpoint(1.59,67.2))
pred.append(plotpoint(1.60,60.68))
sc_plot(pred,act,'#2ca02c','#ff0000')

# pl1=sc_plot(pred,'#2ca02c')
```

```
# pl2=sc_plot(act,'#ff0000')
plt.xlabel('X coordinates ')
plt.ylabel('Y coordinates ')
plt.title('Reverberant data')
plt.savefig('Reverberant_predicted_noline.png',dpi=200)
# plt.legend([pred,act])
```



0.0.20 Generating the requirements.txt file

[ ]:
```
import pkg_resources
import types
def get_imports():
    for name, val in globals().items():
        if isinstance(val, types.ModuleType):
            # Split ensures you get root package,
            # not just imported function
            name = val.__name__.split(".")[0]

        elif isinstance(val, type):
            name = val.__module__.split(".")[0]

        # Some packages are weird and have different
        # imported names vs. system/pip names. Unfortunately,
        # there is no systematic way to get pip names from
        # a package's imported name. You'll have to add
        # exceptions to this list manually!
        poorly_named_packages = {
            "PIL": "Pillow",
            "sklearn": "scikit-learn"
        }
        if name in poorly_named_packages.keys():
```

```
            name = poorly_named_packages[name]

        yield name
imports = list(set(get_imports()))

# The only way I found to get the version of the root package
# from only the name of the package is to cross-check the names
# of installed packages vs. imported packages
requirements = []
for m in pkg_resources.working_set:
    if m.project_name in imports and m.project_name!="pip":
        requirements.append((m.project_name, m.version))

for r in requirements:
    print("{}=={}".format(*r))
```

Chapter 6

RESULTS ANALYSIS

6.1 Dataset Generation Results

- For the simulation of the datasets, eight room impulse responses have been generated (one for each microphone) and all that eight RIRs were convolved with the clean speech signals and corresponding audios are generated and corresponding plots were verified.
- The system performs well for two simultaneously active sources and the convolved speech signals were heard and plots for the convolved outputs were verified.

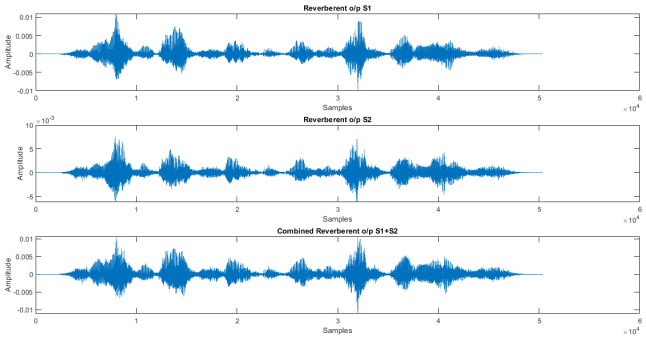


Figure 6.1: Convolved Outputs

- All the anechoic datasets required for training ,testing and validation of main architecture are generated as mentioned in [5.1.3].

6.2 Reverberant SMESLP

6.2.1 Classification results

The proposed model is trained and tested with the reverberant data and the classification results are as follows

Initial Classification outputs

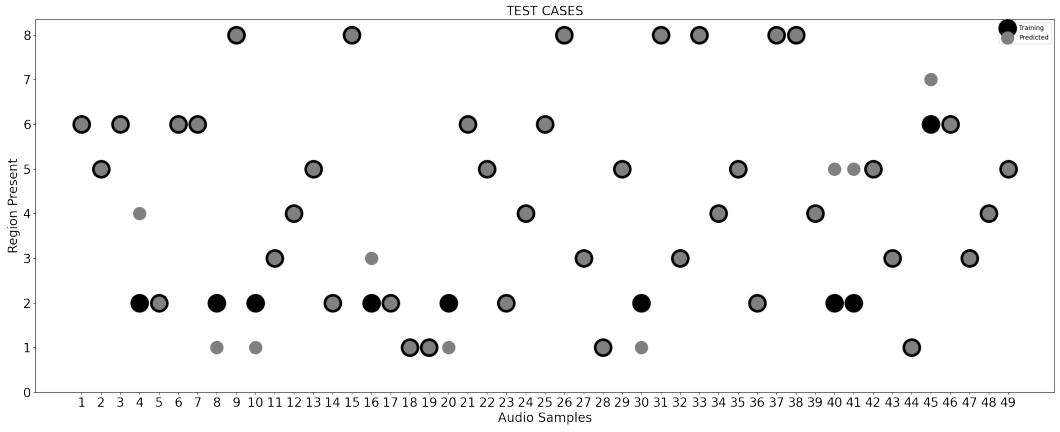


Figure 6.2: Initial Reverberant classification 1

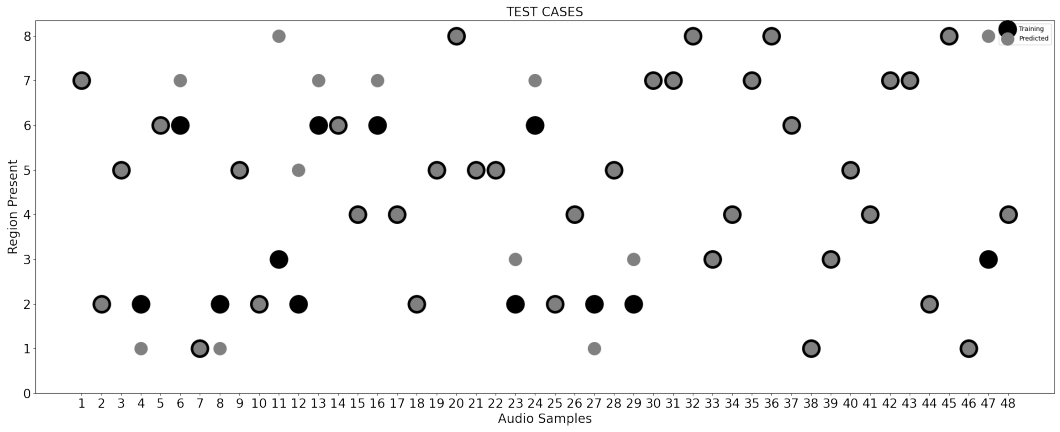


Figure 6.3: Initial Reverberant classification 2

Improved Classification outputs

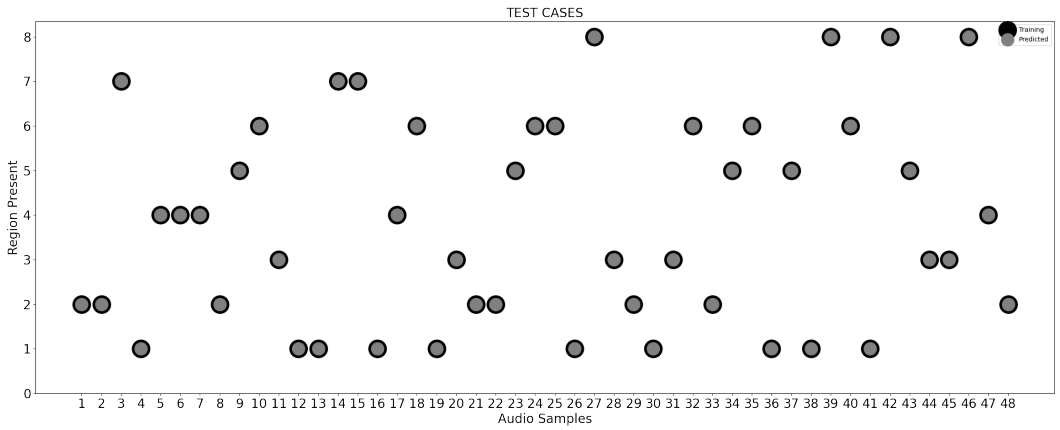


Figure 6.4: Improved Reverberant classification 1

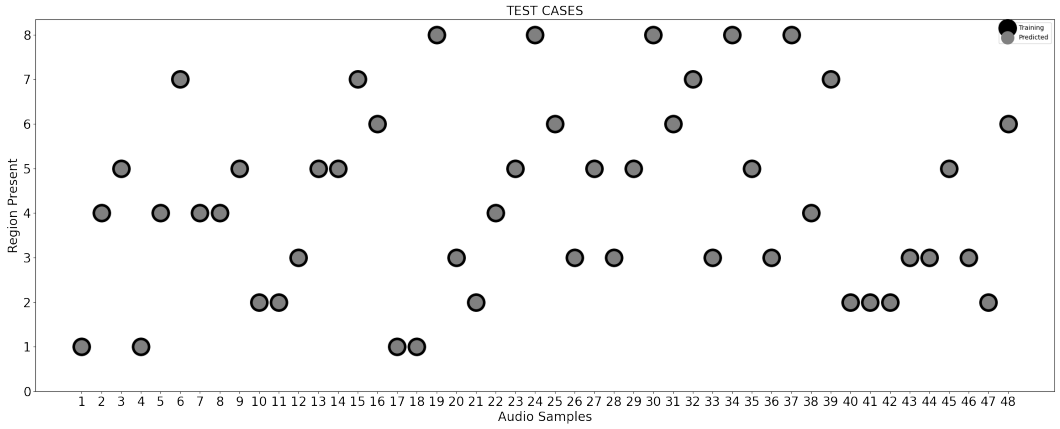


Figure 6.5: Improved Reverberant classification 2

From the above figures we infer the following

- The initial training on a small dataset which sampled 200 random points per region resulted in a classification accuracy of 75% for reverberant test cases.
- Using transfer learning and increasing the dataset by taking different instances from the audio simulation our classification accuracy jumped to 99.3%.

6.2.2 Regression results

DISTANCE PREDICTIONS			ANGLE PREDICTIONS		
S no.	ACTUAL	PREDICTED	S no.	ACTUAL	PREDICTED
10	1.6708984077366000	1.6597211810649000	10	66.11954001724670	67.58505821228030
11	2.0721427565488200	1.6493071260149600	11	127.36862308715700	115.07656753063200
12	0.9738454882652750	0.7211601748118490	12	206.01832916561500	201.9822183251380
13	2.204330184994790	1.7410626917155100	13	89.89082233020060	67.16855943202970
14	0.5347720289427710	0.7424554964330630	14	194.10643137527100	202.14920595288300
15	1.1090224924172800	1.578595247986970	15	112.48265342178400	113.88999849557900
16	0.4374476536643020	0.743777943892912	16	159.1211022567080	156.21715039014800
17	0.4346899947585820	0.8166820343652680	17	317.72717387524800	338.9520514011380
18	2.330552985996850	1.6088462431395600	18	297.1584099483510	293.39244067668900
19	0.19570177556828400	0.7199243691499160	19	322.8837940314250	339.0348780155180
20	2.3201225423982100	1.761890345405570	20	250.90363237701400	246.92268937826200

We calculated the Mean Absolute Error(MAE) for the regression outputs of our architecture:

- MAE_distance = 0.3842 m
- MAE_angle = 10.27°

The above MAE is calculated within each region with constraints

- Distance_max = 2.5799 m
- Distance_min = 0.0055 m
- Max_angle = 45° per region.

6.2.3 Visualizing the predictions

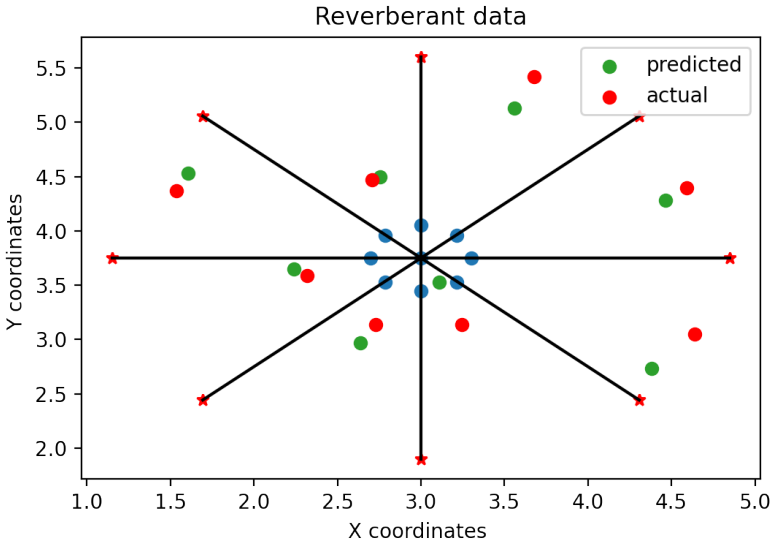


Figure 6.6: Reverberant sector wise prediction

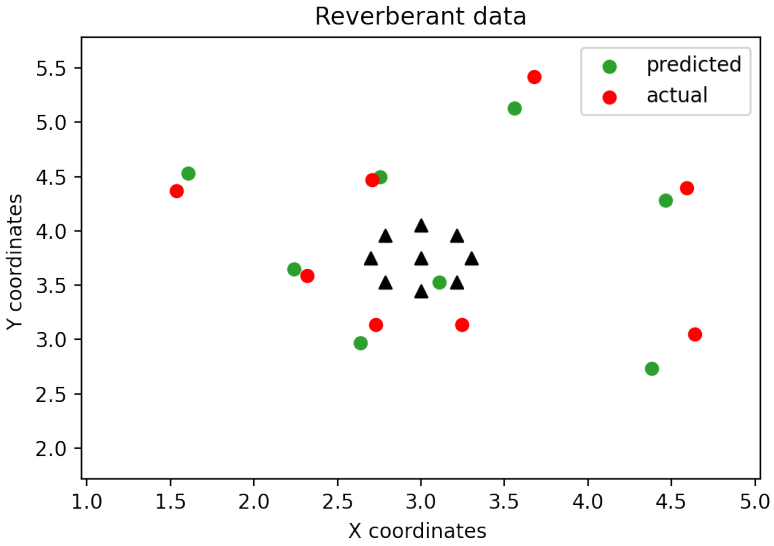


Figure 6.7: Reverberant prediction in a room

On converting the predicted radial distances and angles into x and y coordinates we can observe how accurately our architecture predicts the source within the 2D space in a reverberant environment.

6.3 Anechoic SMESLP

6.3.1 Classification results

The proposed model is trained and tested with the anechoic data and the classification results are as follows

Initial Classification outputs

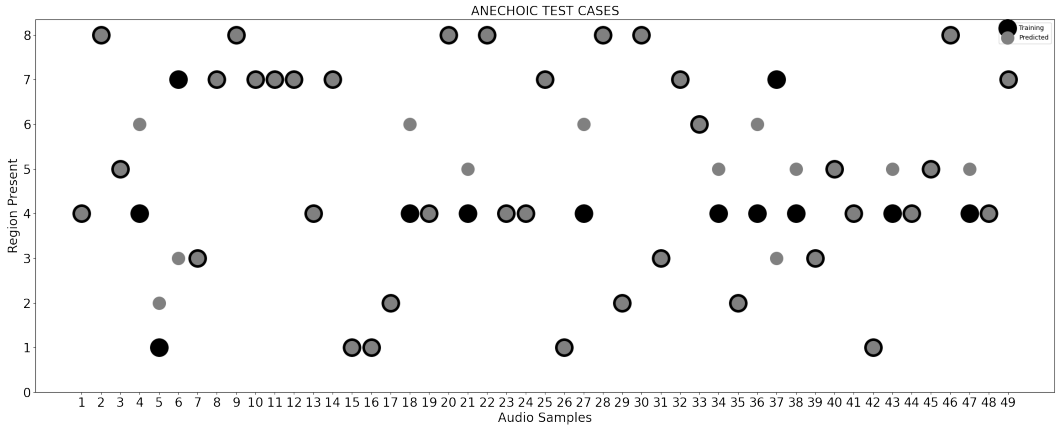


Figure 6.8: Initial Anechoic classification 1

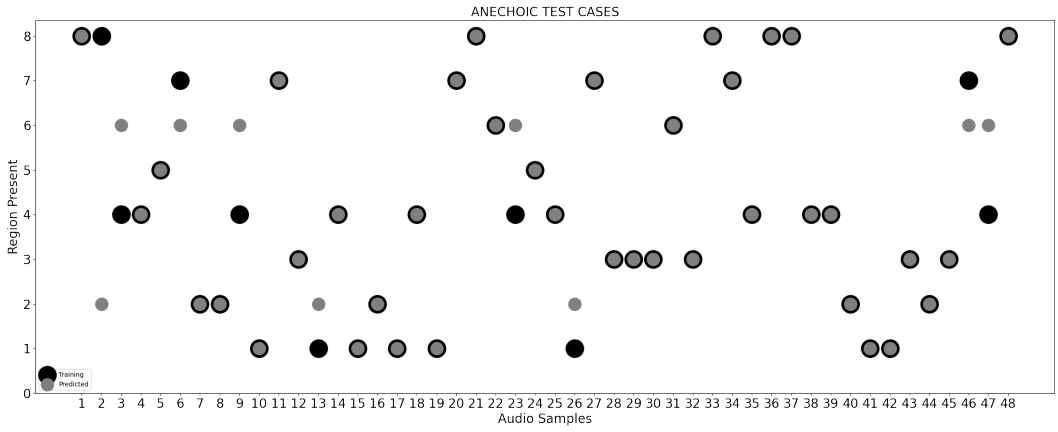


Figure 6.9: Initial Anechoic classification 2

Improved Classification outputs

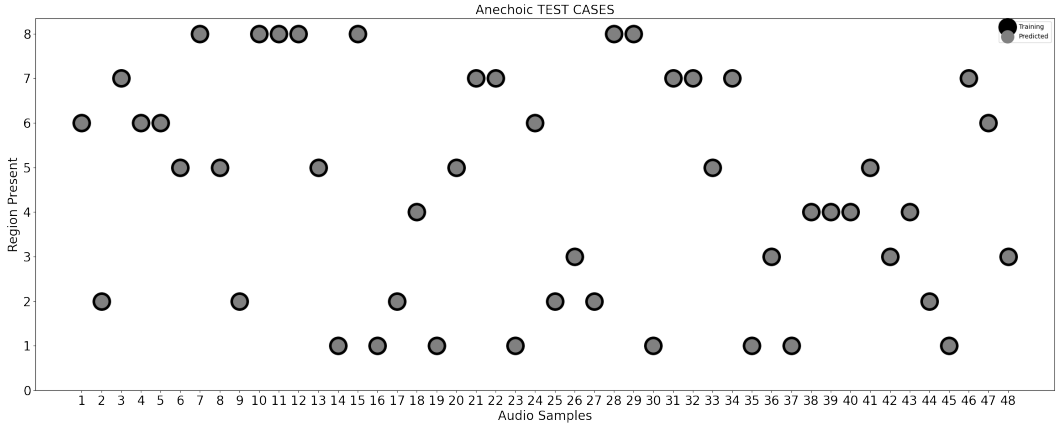


Figure 6.10: Improved Anechoic classification 1

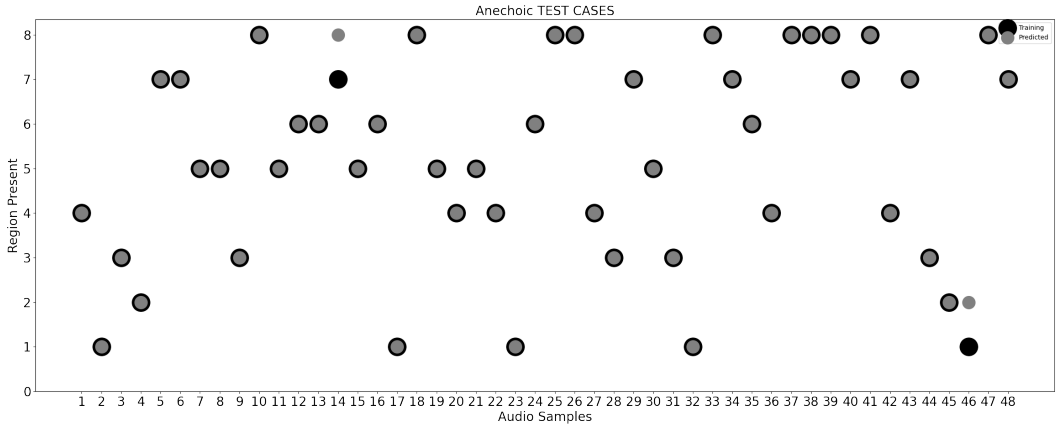


Figure 6.11: Improved Anechoic classification 2

From the above figures we infer the following

- The initial training on a small dataset which sampled 200 random points per region resulted in a classification accuracy of 85.25% for anechoic test cases.
- Using transfer learning and increasing the dataset by taking different instances from the audio simulation our classification accuracy jumped to 99.16%.

6.3.2 Regression results

DISTANCE PREDICTIONS			ANGLE PREDICTIONS		
S no.	ACTUAL	PREDICTED	S no.	ACTUAL	PREDICTED
67	1.6594144736643700	1.1284873109044600	67	103.1151091601190	115.22506535053300
68	1.6183626922058300	1.3237129613964700	68	86.7916468064171	70.83561927080150
69	1.6682974258974100	1.035613789406500	69	111.88115296647100	113.31994324922600
70	1.7109766126453500	1.0402018784227100	70	72.77709913751720	76.14429026842120
71	1.9411288723243800	2.271287856120530	71	285.14931979198400	287.42713138461100
72	1.838742456908340	0.9912278780161270	72	140.22569078031800	149.01736110448800
73	1.0428793507982500	0.6978182228348610	73	247.72340092531400	233.18606570363000
74	1.3975767227615500	1.5740553033058500	74	227.7039398747180	241.4134655892850
75	1.7240011797417100	1.6129574196939700	75	55.11832943243550	47.739647179842000
76	1.5878446973072400	1.5790522130345200	76	20.907019485174000	25.085738003253900
77	0.7434977843303800	0.4550369710706370	77	206.98589123236200	214.8466694355010

We calculated the Mean Absolute Error(MAE) for the regression outputs of our architecture:

- MAE_distance = 0.344 m
- MAE_angle = 11°

The above MAE is calculated within each region with constraints

- Distance_max = 2.5799 m
- Distance_min = 0.0055 m
- Max_angle = 45° per region.

6.3.3 Visualizing the predictions

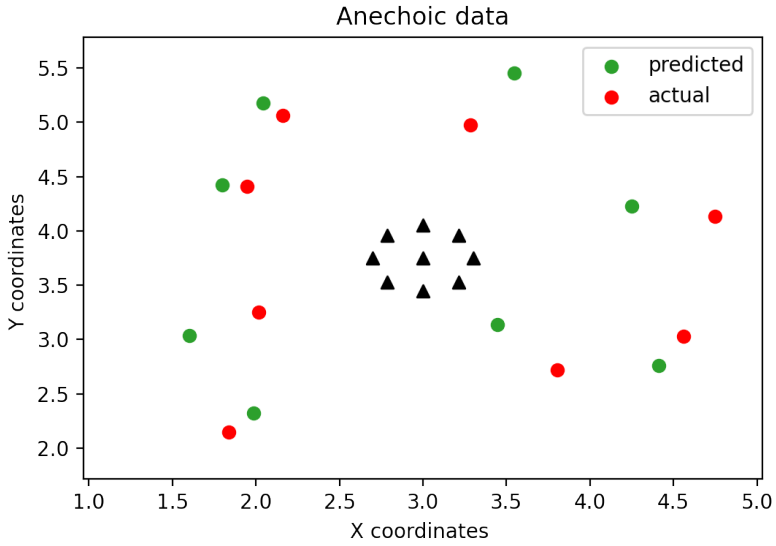


Figure 6.12: Anechoic sector wise prediction

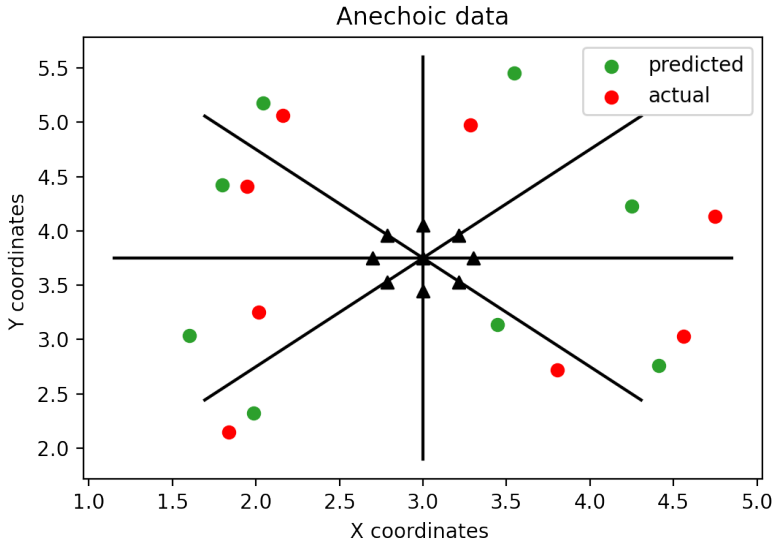


Figure 6.13: Anechoic prediction in a room

On converting the predicted radial distances and angles into x and y coordinates we can observe how accurately our architecture predicts the source within the 2D space in a anechoic environment.

Chapter 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

- We enhanced our knowledge in ML, Deep Learning and building CNN architectures.
- We succeeded in simulating anechoic as well as reverberant audio samples to train and test our architecture to completion.
- We improved our skills in dataset management for efficient use in the training process.
- Studied the proposed SMESLP approach and how it outperforms the existing signal processing approaches for sound event localization.
- Project was completed to 100% without code support from the author for this novel architecture.
- We were able to put to practice various DSP concepts while simulating relevant datasets.
- We successfully kept track with the Gantt chart from our project planning phase.

7.2 Future Scope

- The next stage is to implement multiple source localization in the same time instant.
- We also would like to deploy this technology into real world applications such as automatic steering of cameras, robotic arm etc.

Chapter 8

REFERENCES

1. Y. Zhang, Z. Wang, W. Wang, Z. Guo and J. Wang, "SOLO: 2D Localization with Single Sound Source and Single Microphone," 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, 2017, pp. 787-790, doi: 10.1109/ICPADS.2017.00108.
2. H.Sundar, T.V.Sreenivas,and C.S.Seelamantula,"TDOA-based multiple acoustic source localization without association ambiguity," IEEE/ACM Trans. on Audio, Speech, and Language Process.
3. Sharath Adavanne, Archontis Politis and Tuomas Virtanen, 'Multichannel sound event detection using 3D convolutional neural networks for learning inter-channel features' at International Joint Conference on Neural Networks (IJCNN 2018)
4. H. Sundar, W. Wang, M. Sun and C. Wang, "Raw Waveform Based End-to-end Deep Convolutional Network for Spatial Localization of Multiple Acoustic Sources," ICASSP 2020
5. Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms," in Sound and Music Computing Conference (SMC), 2017.
6. T. Kim, J. Lee, and J. Nam, "Comparison and analysis of samplecnn architectures for audio classification," IEEE Journal of Selected Topics in Signal Processing, vol. 13, no. 2, pp. 285–297, May 2019.
7. E.A.P.Habets, 'Room Impulse Response Generator,' Sep.2010
8. https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529
9. https://www.rebellionresearch.com/blog/what-is-sound-source-localization
10. https://www.safeopedia.com/definition/5521/anechoic-chamber
11. https://www.youtube.com/watch?v=AM0mTP7auCQ
12. https://www.pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/

Appendix A

Review Process

A.1 Phases of Review

Project was evaluated at phases with major checkpoints as follows :

Pre-Review : ***Defining Problem Statement and Feasibility analysis Review***

1. Understanding of the problem statement.
2. Technical understanding of domain.
3. Identification of differentiating features.
4. Feasibility of conversion to product or paper.

1st Review : ***Literature survey and Project plan Review***

1. Clarity in understanding of the problem/project.
2. Completion of Literature Survey.
3. Identification of sub-blocks and their interaction.
4. Timeline for completion of project using Gantt chart.

2nd Review : ***Module level design and Test Plan Review***

1. Detail design of each module.
2. Integration and module test plan.
3. Availability status of required Hardware and Software components.
4. 20% Completion of Block/Sub module implementation.

3rd Review : ***Project Progress Review***

1. Adherences to project plan.
2. Completion of module interaction interface design.

3. 50% Module level implementation.
4. Demonstration of completed modules using primitive interfaces

4th Review : ***Project Progress and Finishing plan Review***

1. 70% Module level implementation
2. Presentation of completed test data as per test plan.
3. Packaging plan.
4. Final demonstration plan.

5th Review : ***Module completion and Integration Review***

1. 100% Module implementation.
2. Completion of Integration.
3. Presentation of test data as per test plan.
4. Adherences to project plan.

6th Review : ***Demonstration Review***

1. Quality of packaged full demo presentation.
2. Integration test data as per test plan.
3. Quality of documentation made across project.
4. Group presentation / communication skill.

Appendix B

PROJECT PLANNING

B.1 Gantt Chart

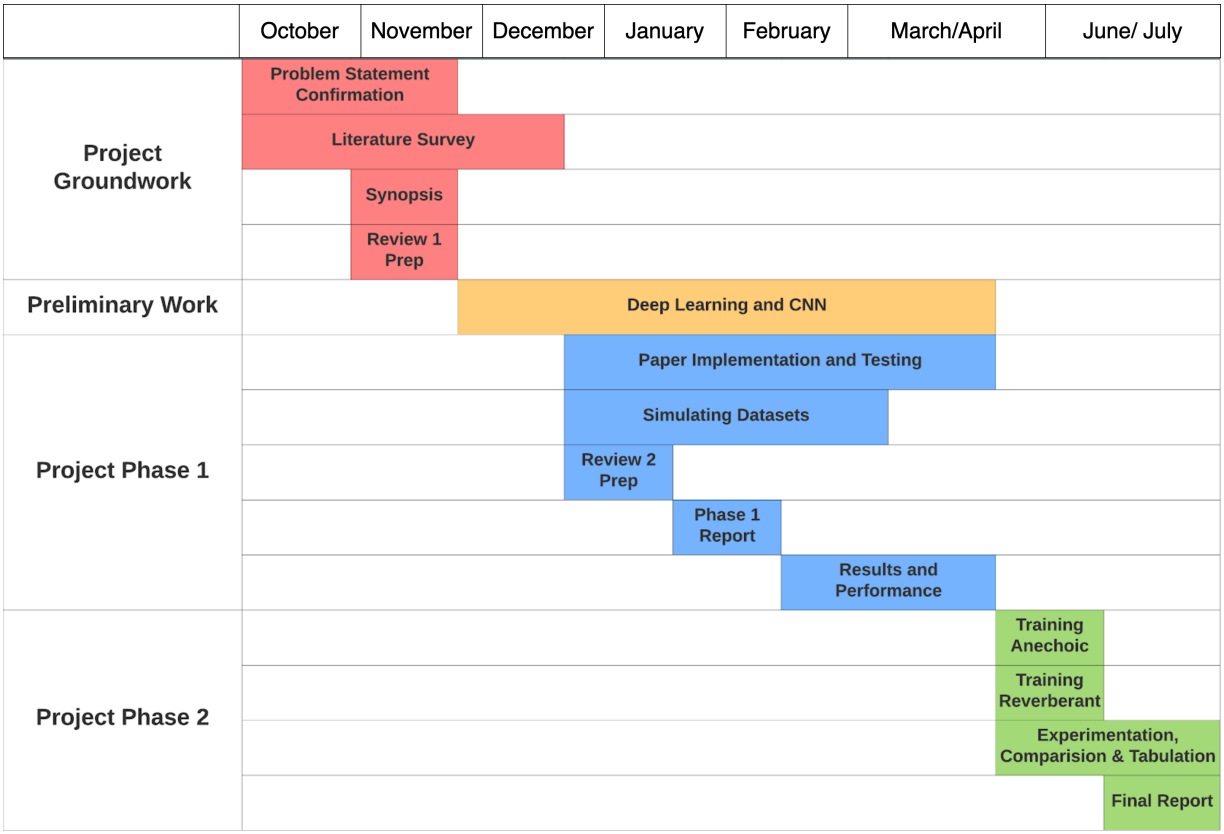


Figure B.1: Gantt Chart