INSTITUTE FOR ADVANCED

COMPUTING AND

SOFTWARE

DEVELOPMENT

AKURDI, PUNE

Documentation On

"Sound Detection and Classification"

PG-DBDA March 2022

Submitted By:

Group No:16

Sarthak Kshirsagar 223327

Shraddha Chougule 223352

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

Table of Contents:

Sr.No.	Contents	Page No.
1.	**Introduction** 1.1 Problem Statement 1.2 Abstract 1.3 Product Scope **1.4** Aims & Objectives	**4-5**
2	**Overall Description** 2.1 Workflow of Project 2.2 About Dataset 2.3 Data Preprocessing and Feature Extraction 2.3.1 Feature Extraction 2.3.2 Data Preprocessing	**6-10**
3	**Model Building**	**11-19**
4	**Performance Analysis**	**20**
5	**Requirements Specification** 5.1 Hardware Requirement 5.2 Software Requirement	**21**
6	**Conclusion & Future Work**	**22**
7	**Feasibility Study**	**23-24**
8	**References**	**25**

List of Figures

Sr.No.	Contents	Page No.
1	**Workflow Diagram**	**6**
2	**Classwise Data Distribution**	**7**
3	**MFCC visualization of Air Conditioner**	**9**
4	**Data Preprocessing of CNN1D**	**11**
5	**Building CNN1D model**	**12**
6	**Compiling and fitting CNN1D model**	**12**
7	**CNN1D prediction function**	**13**
8	**Prediction time**	**13**
9	**Model Building CNN2D**	**13**
10	**Output of model**	**14**
11	**Compiling model**	**14**
12	**Fitting model**	**15**
13	**Plotting loss per Epochs curve**	**16**
14	**Result wrt loss and Epochs**	**16**
15	**Plotting Accuracy per Epochs**	**17**
16	**Result wrt Accuracy per Epochs**	**17**
17	**Training time**	**18**
18	**CNN2D Prediction function**	**18**
19	**Testing model by custom input**	**18**

1. Introduction

1.1 Problem Statement :

Sound Detection and Classification

1.2 Abstract :

Audio classification or sound classification can be referred to as the process of analysing audio recordings. This amazing technique has multiple applications in the fields of AI and data science. In this project, we will explore audio classification using deep learning concepts involving algorithms like Artificial Neural Network (ANN), 1D Convolutional Neural Network (CNN1D), and CNN2D. The dataset contains 8732 labelled sound excerpts (=4s) of urban sounds from ten categories: air for audio prediction, car horns, children playing, dog barking, drilling, engine idling, gunshots, jackhammers, sirens, and street music are used for audio prediction. Before we develop models, we do some basic data pre-processing and feature extraction on audio signals. As a result, each model is compared in terms of accuracy, training time, and prediction time. This is explained by model deployment, where users are allowed to load a desired sound output for each model being deployed successfully.

1.3 Product Scope :

Sound classification is one of the most widely used applications in audio deep learning. It involves learning to classify sounds and predicting the category of that sound. This type of problem can be applied to many practical scenarios, e.g., classifying music clips to identify the genre of the music, or classifying short utterances by a set of speakers to identify the speaker based on the voice. Our project involves a comparison of some deep learning models. As our aim is to help deaf people know their surroundings, we have deployed our model in use. Here people can load the audio files (.wav) and, once they submit them, the audio sound is printed as the outcome for each model, thus achieving our goal for the project.

1.4 Aims & Objectives :

We have developed a useful application by applying deep learning concepts that help deaf people hear what is happening around them. We are all blessed with good hearing, but some are unlucky. So, people with such disabilities can upload the audio files and see a report which gives a detailed classification of the audio file. This not only helps them recognise what is going on around them, but it also helps them connect with the world. Our Goal is to develop a very useful application for deaf people that helps them know the voices around them and provides a summary of those voices.

2. Overall Description

2.1 Workflow of Project :

The diagram below shows the workflow of this project.

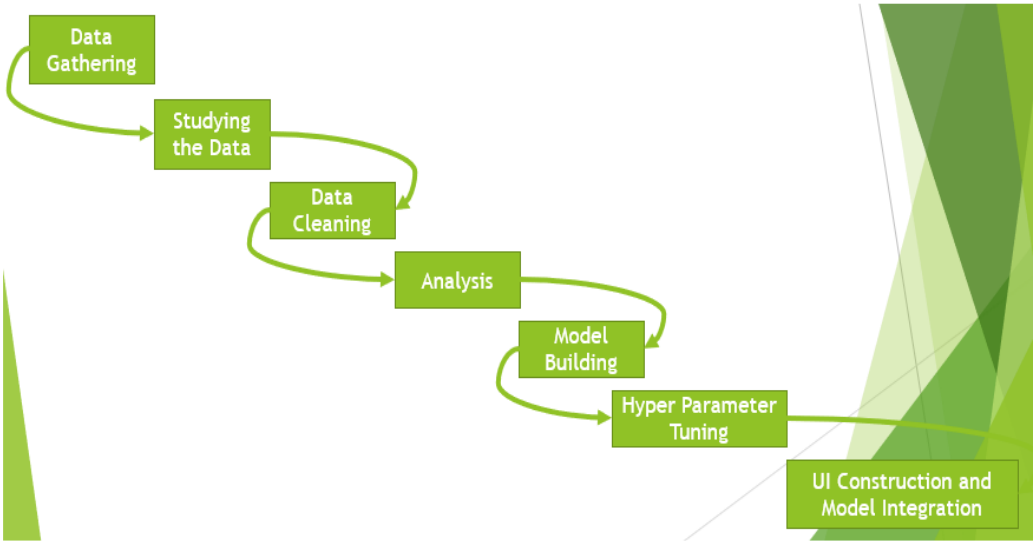

Figure 1 – Workflow Diagram

2.2 About Dataset :

Dataset taken from - [https://urbansounddataset.weebly.com/urbansound8k.html]

Description

This dataset contains 8732 labelled sound excerpts (=4s) of urban sounds from ten categories: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. The classes are based on the taxonomy of urban sounds. All excerpts are from field recordings. The files are pre-sorted into ten folds (folders named fold1-fold10) to aid in reproducing and comparing the results of the automatic classification.

Audio files included 8732 WAV audio files of urban sounds as described above.

This dataset also contains UrbanSound8k.csv that contains meta-data for each audio file in the dataset. This includes the following:

The name of the audio file: The name takes the following format: (fsID)-(classID)-(occurrenceID)-(sliceID).wav, where: (fsID) = the Freesound ID of the recording from which this excerpt (slice) is taken, (classID) = a numeric identifier of the sound

class (see description of classID below for further details), (occurrenceID) = a numeric identifier to distinguish different occurrences of the sound within the original recording, (sliceID) = a numeric identifier to distinguish different slices taken from the same occurrence

fsid : The Freesound ID of the recording from which this excerpt (slice) is taken

start : The start time of the slice in the original Freesound recording

end: The end time of slice in the original Freesound recording

salience: A (subjective) salience rating of the sound. 1 = foreground, 2 = background.

fold: The fold number (1-10) to which this file has been allocated.

classID : A numeric identifier of the sound class.

0 = air_conditioner, 1 = car_horn, 2 = children_playing, 3 = dog_bark, 4 = drilling, 5 = engine_idling, 6 = gun_shot, 7 = jackhammer, 8 = siren, 9 = street_music

class: The class name: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music.

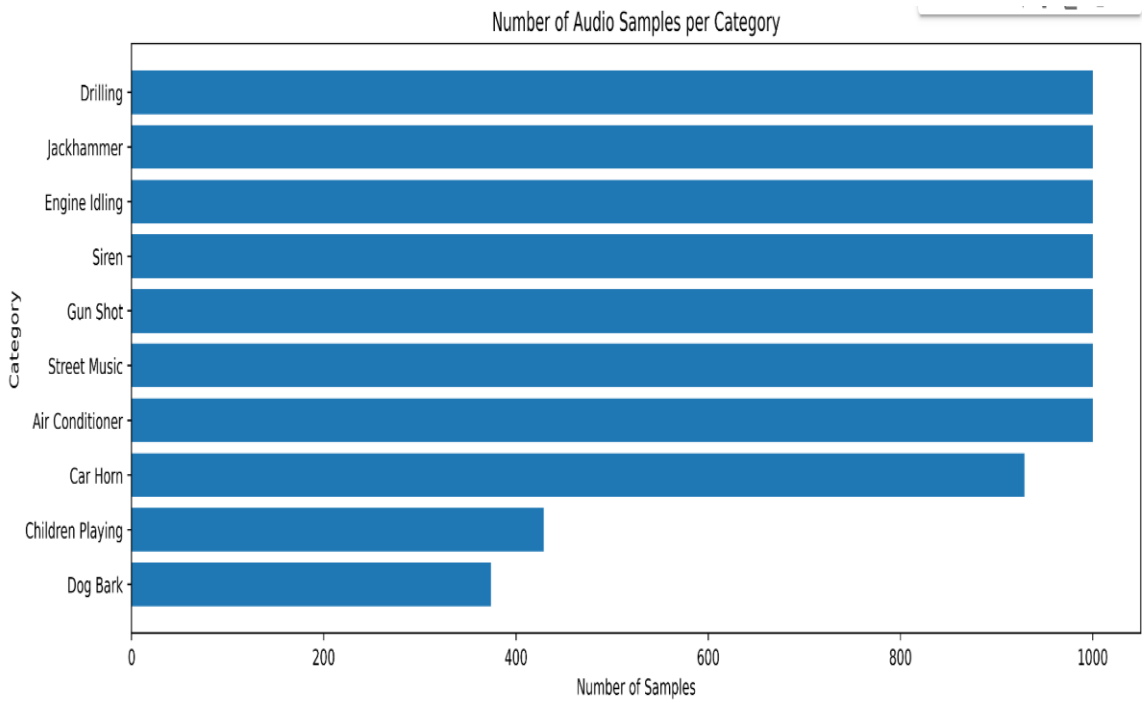

Figure 2 – Classwise Data Distribution.

From above figure we can say that above data is balanced.

2.3 Data Preprocessing and Feature Extraction:

We use the data we obtained by using librosa because we require data in numeric format. We use Mel-Frequency Cepstral Coefficients (MFCC) to extract independent data, which summarizes the frequency distribution across the window size, allowing us to analyze both the frequency and time characteristics of the sound. We can identify features for classification using these audio representations. We defined the features extractor function and passed in the path to the audio file as a parameter, after which we will extract the audio features using librosa. The feature_extractor function is applied to all rows, and the results are stored in a dataframe with features and class columns for further calculations.

MFCC :

MFCC stands for Mel-frequency cepstrum coefficients is a mathematical coefficients for sound modeling.

The MFCC uses the MEL scale to divide the frequency band to sub-bands and then extracts the Cepstral Coefficents using Discrete Cosine Transform (DCT). MEL scale is based on the way humans distinguish between frequencies which makes it very convenient to process sounds.

Below is code for MFCC visualizations. We have created visualizations for each class.

```
plt.rcParams["figure.figsize"] = (4, 5)
plt.rcParams["figure.dpi"] = 100
#audio_path=audio_dataset_path+"/content/drive/MyDrive/UrbanSound8K/UrbanSound8K/audio/fold1/127873-0-0-0.wav"
audio_path="/content/drive/MyDrive/UrbanSound8K/UrbanSound8K/audio/fold1/127873-0-0-0.wav"
(xf, sr) = librosa.load(audio_path)
mfccs = librosa.feature.mfcc(y=xf, sr=sr, n_mfcc=40)
librosa.display.specshow(mfccs, x_axis="time")
plt.colorbar()
```

plt.tight_layout()

plt.title("MFCC Of Air Conditioner")

plt.show

Output :

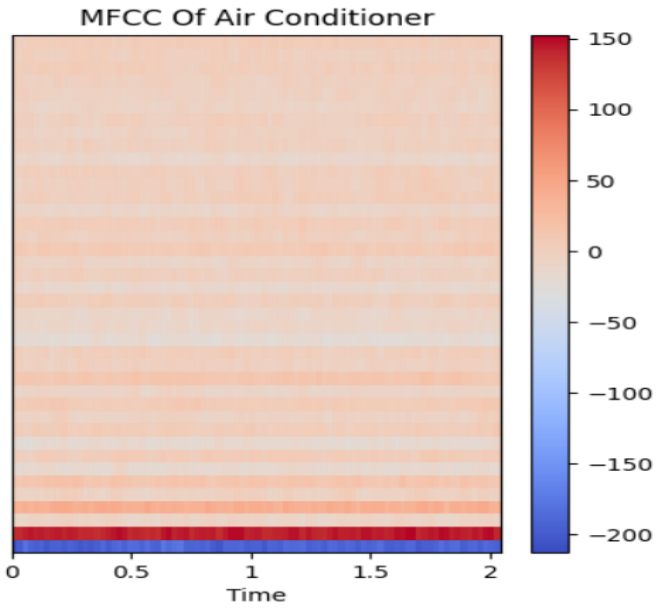

Figure 3 – MFCC visualization of Air Conditioner

2.3.1 Feature Extraction:

We obtained the dataset from UrbanSound8K, which contains over 8500 data files containing various audios such as a baby crying, birds sound, dog bark, and many others in the form of.wav files. It is divided into ten folders, indicating that the dataset has ten classes as described in the dataset section. Libraries that are required are added, as well as Librosa for feature extraction.

Librosa: Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.
The function bellow will extract mfcc feature

```
for index_num, row in tqdm(meta_data.iterrows()):
  print(row)
```

2.3.2 Data Preprocessing:

Distribute the data to X and Y

```
X = np.array(final["feature"].tolist())

y = np.array(final["class"].tolist())
```

Using LabelEncoder() to encode the string labels to an integer

```
# label encoding to get encoding
le = LabelEncoder()

# transform each category with it's respected label
Y = to_categorical(le.fit_transform(y))
```

Split the data into train and test sets

```
xtrain = X_train.reshape(X_train.shape[0], 16, 8, 1)
xtest = X_test.reshape(X_test.shape[0], 16, 8, 1)
```

3. Model Building

Artificial Neural Network (ANN)

Artificial neural networks (ANNs) are made up of node layers, each of which has an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is linked to another and has its own weight and threshold. If the output of any individual node exceeds the specified threshold value, that node is activated and begins sending data to the network's next layer. Otherwise, no data is passed to the next network layer.

Convolutional Neural Network (CNN)

A convolutional neural network (CNN, or ConvNet) is a type of artificial neural network used to interpret visual imagery in deep learning. Based on the shared-weight architecture of the convolution kernels or filters that slide along input features and give translation equivariant responses known as feature maps, they are also known as shift invariant or space invariant artificial neural networks (SIANN). Surprisingly, most convolutional neural networks are only equivariant under translation, rather than invariant. Image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series are just a few of the areas where they can be used.

CNN1D

A modified form of 2D CNNs known as 1D Convolutional Neural Networks (1D CNNs) has recently been developed as an alternative. In dealing with 1D signals, these research have proven that 1D CNNs are beneficial and consequently preferable to their 2D counterparts in some situations. 1D CNN, kernel moves in 1 direction. Input and output data of 1D CNN is 2 dimensional. Mostly used on Time-Series data. 1D CNN can perform activity recognition task from accelerometer data, such as if the person is standing, walking, jumping etc. This data has 2 dimensions. The first dimension is time-steps and other is the values of the acceleration in 3 axes. Similarly, 1D CNNs are also used on audio and text data since we can also represent the sound and texts as a time series data. Conv1D is widely applied on sensory data, and accelerometer data is one of it.

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.1, stratify=y, random_state=7)
```

Figure 4 – Data Preprocessing of CNN1D

Here we are preprocessing the data to build CNN1D model. Here we have used train_test_split() from sklearn library to split the data for training and testing.

```
CNN1D_Model = Sequential()
CNN1D_Model.add(Conv1D(256, 5, strides=1, padding="same", activation="relu", input_shape=(Xtrain.
CNN1D_Model.add(BatchNormalization())
CNN1D_Model.add(MaxPooling1D(3, strides=2, padding="same"))
CNN1D_Model.add(Conv1D(256, 5, strides=1, padding="same", activation="relu"))
CNN1D_Model.add(Dropout(0.3))
CNN1D_Model.add(MaxPooling1D(3, strides=2, padding="same"))
CNN1D_Model.add(Conv1D(128, 5, strides=1, padding="same", activation="relu"))
CNN1D_Model.add(Dropout(0.3))
CNN1D_Model.add(MaxPooling1D(3, strides=2, padding="same"))
CNN1D_Model.add(Conv1D(64, 5, strides=1, padding="same", activation="relu"))
CNN1D_Model.add(Dropout(0.3))
CNN1D_Model.add(MaxPooling1D(3, strides=2, padding="same"))
CNN1D_Model.add(Flatten())
CNN1D_Model.add(Dense(units=1024, activation="relu"))
CNN1D_Model.add(Dropout(0.3))
CNN1D_Model.add(Dense(units=10, activation="softmax"))
CNN1D_Model.summary()

```

Figure 5 - Building CNN1D model

```
t = time.time()

CNN1D_Results = CNN1D_Model.fit(Xtrain, Ytrain, batch_size=64, epochs=2, validation_dat

train_hist = pd.DataFrame(CNN1D_Results.history)
train_time = round(time.time() - t, 3)

```

Figure 6 – Compiling and fitting CNN1D model

```
[ ]  1 # function to predict the feature
     2 def CNN1D_Prediction(file_name):
     3     # load the audio file
     4     audio_data, sample_rate = librosa.load(file_name, res_type="kaiser_fast")
     5     # get the feature
     6     feature = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=128)
     7     # scale the features
     8     feature_scaled = np.mean(feature.T, axis=0)
     9     # array of features
    10     prediction_feature = np.array([feature_scaled])
    11     # expand dims
    12     final_prediction_feature = np.expand_dims(prediction_feature, axis=2)
    13     # get the id of label using argmax
    14     predicted_vector = np.argmax(CNN1D_Model.predict(final_prediction_feature), axis=-1)
    15     # get the class label from class id
    16     predicted_class = le.inverse_transform(predicted_vector)
    17     # display the result
    18     print("CNN1D has predicted the class as  --> ", predicted_class[0])
    19
```

Figure 7 – CNN1D prediction function

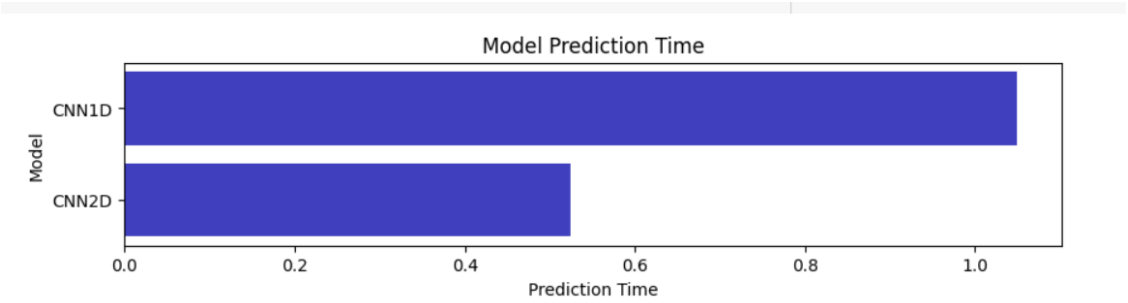

Figure 8 - Prediction time

CNN1D prediction takes more time of all modules. CNN2D prediction takes less time of all models.

CNN2D

The traditional deep CNNs are only designed to work with 2D data like photos and movies. This is why they're referred to as "2D CNNs" so frequently. 2D CNN, kernel moves in 2 directions. Input and output data of 2D CNN is 3 dimensional. Mostly used on Image data. This is the conventional Convolution Neural Network, which was introduced in the Lenet-5 architecture for the first time. On Image data, Conv2D is commonly used. Because the kernel slides along two dimensions on the

data, it is called 2 dimensional CNN. The main benefit of employing CNN is that it can extract spatial properties from data using its kernel, which is something that other networks can't achieve. CNN, for example, can detect edges, colour distribution, and other spatial aspects in an image, making these networks particularly robust in image classification and other data with spatial qualities.

Predicting the test audio on all the Models

We defined a function which will extract the features from the audio which is given in the parameter, and the features will be converted to proper input shapes for CNN2 models. The output will give the class label. The CNN2D_print_prediction function used.

```
CNN2D_Model = Sequential()
CNN2D_Model.add(Conv2D(64, (3, 3), padding="same", activation="tanh", input_shape=(16, 8, 1)))
CNN2D_Model.add(MaxPool2D(pool_size=(2, 2)))
CNN2D_Model.add(Conv2D(130, (3, 3), padding="same", activation="tanh"))
CNN2D_Model.add(MaxPool2D(pool_size=(2, 2)))
CNN2D_Model.add(Dropout(0.1))
CNN2D_Model.add(Flatten())
CNN2D_Model.add(Dense(1024, activation="tanh"))
CNN2D_Model.add(Dense(10, activation="softmax"))
CNN2D_Model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 16, 8, 64)         640

 max_pooling2d_6 (MaxPooling  (None, 8, 4, 64)         0
 2D)

 conv2d_7 (Conv2D)           (None, 8, 4, 130)         75010

 max_pooling2d_7 (MaxPooling  (None, 4, 2, 130)        0
 2D)
```

Figure 9 – Model Building CNN2D

After the data is ready to be fed to the model, we need to define the architecture of the model and compile it with necessary optimizer function, loss function and performance metrics.

The architecture followed here is 2 convolution layers followed by pooling layer, a fully connected layer and softmax layer respectively. Multiple filters are used at each convolution layer, for different types of feature extraction. One intuitive

explanation can be if first filter helps in detecting the straight lines in the image, second filter will help in detecting circles and so on.

After both maxpooling and fully connected layer, dropout is introduced as regularization in our model to reduce over-fitting problem.

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 16, 8, 64)         640

 max_pooling2d_6 (MaxPooling  (None, 8, 4, 64)         0
 2D)

 conv2d_7 (Conv2D)           (None, 8, 4, 130)         75010

 max_pooling2d_7 (MaxPooling  (None, 4, 2, 130)        0
 2D)

 dropout_11 (Dropout)        (None, 4, 2, 130)         0

 flatten_5 (Flatten)         (None, 1040)              0

 dense_10 (Dense)            (None, 1024)              1065984

 dense_11 (Dense)            (None, 10)                10250

=================================================================
Total params: 1,151,884
Trainable params: 1,151,884
Non-trainable params: 0
_________________________________________________________________
```

Figure 10 – Output of model

```
CNN2D_Model.compile(
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
)
```

Figure 11 – Compiling model

After the architecture of the model is defined, the model needs to be compiled. Here, we are using categorical_crossentropy loss function as it is a multi-class

classification problem. Since all the labels carry similar weight we prefer accuracy as performance metric.

```
t = time.time()

CNN2D_Results = CNN2D_Model.fit(xtrain, y_train, epochs=10, batch_size=40,
                                validation_data=(xtest, y_test))

train_hist = pd.DataFrame(CNN2D_Results.history)
train_time = round(time.time() - t, 3)
```

```
Epoch 1/10
```

Figure 12 – Fitting model

After the model architecture is defined and compiled, the model needs to be trained with training data to be able to recognize the audio that we give as an input. Hence we will fit the model with X_train and y_train.

Here, one epoch means one forward and one backward pass of all the training samples. Batch size implies number of training samples in one forward/backward pass.

```
plt.figure(figsize=(8, 4), dpi=100)
plt.plot(train_hist[["loss", "val_loss"]])
plt.legend(["Loss", "Validation Loss"])
plt.title("Loss Per Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

Figure 13 – Plotting loss per Epochs curve

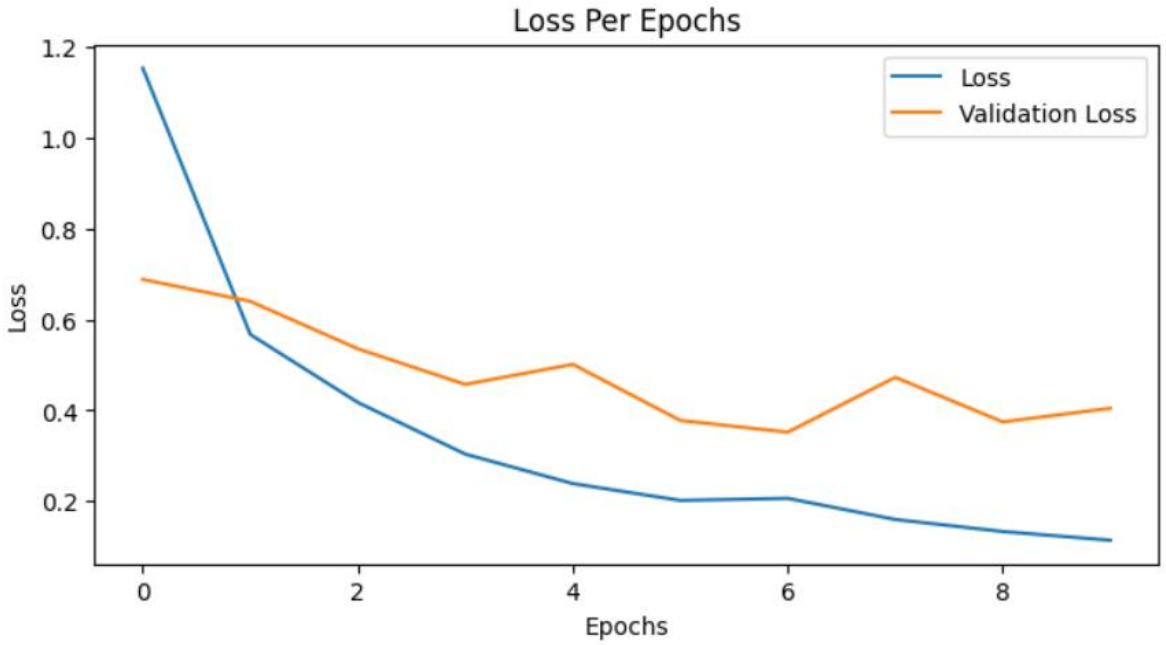

Figure 14 – Result wrt loss and Epochs*
**Wrt – With respect to*

```
1 plt.figure(figsize=(8, 4), dpi=100)
2 plt.plot(train_hist[["accuracy", "val_accuracy"]])
3 plt.legend(["Accuracy", "Validation Accuracy"])
4 plt.title("Accuracy Per Epochs")
5 plt.xlabel("Epochs")
6 plt.ylabel("Accuracy")
7 plt.show()
```

Figure 15 – Plotting Accuracy per Epochs

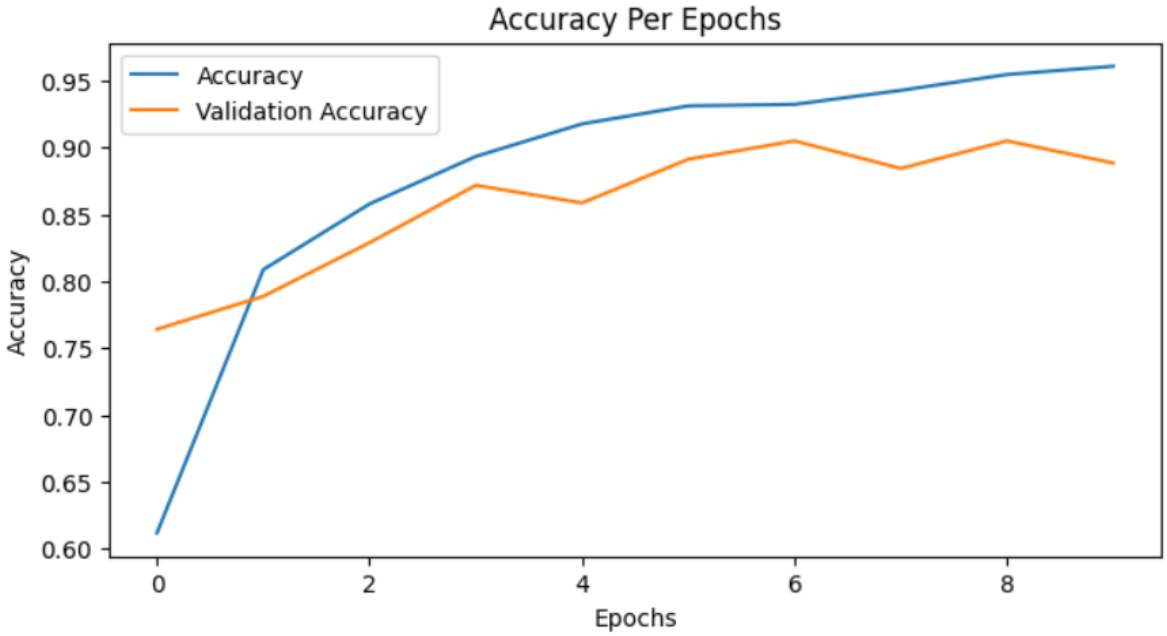

Figure 16 - Result wrt Accuracy per Epochs*

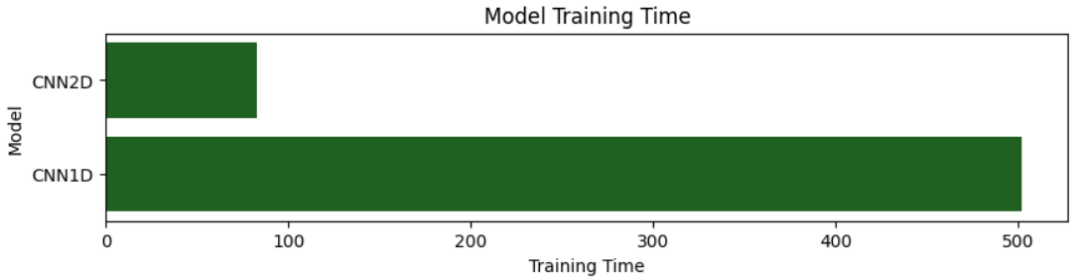

Figure 17 - Training time

```python
# function to predict the feature
def Predict(file_name):
    # load the audio file
    audio_data, sample_rate = librosa.load(file_name, res_type="kaiser_fast") # to reduce load time(resample_typ
    # get the feature
    feature = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=128)
    # scale the features
    feature_scaled = np.mean(feature.T, axis=0)
    # array of features
    prediction_feature = np.array([feature_scaled])
    # reshaping the features
    final_prediction_feature = prediction_feature.reshape(prediction_feature.shape[0], 16, 8, 1)
    # get the id of label using argmax
    predicted_vector = np.argmax(CNN2D_Model.predict(final_prediction_feature), axis=-1) # Returns the indices
    # get the class label from class id
    predicted_class = le.inverse_transform(predicted_vector)
    # display the result
    print("predicted sound is of class --> ", predicted_class[0])
```

Figure 18 – CNN2D Prediction function

```python
# File name
file_name = "/content/drive/MyDrive/UrbanSound8K/UrbanSound8K/audio/fold8/103076-3-0-0.wav"
# get the output
CNN2D_Prediction(file_name)
# play the file
ipd.Audio(file_name)
```

```
CNN2D has predicted the class as  -->  Dog Bark
```

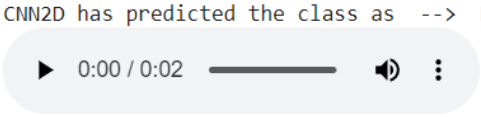

Figure 19 – Testing model by custom input

4. Performance Analysis

In our project, we got audio signals and sample rates for the selected audio files, which was a difficult challenge for us. We used numerous references for each algorithm and conducted a Using all the techniques, we tested the algorithm with various audio files and obtained the projected value as expected. We displayed all the comparisons we made, including accuracies, training time, and prediction time for model, to provide a visual depiction. In terms of accuracy, CNN2D has an 89.93% accuracy rate.

5. Requirements Specification

5.1 Hardware Requirement:

- 500 GB hard drive (Minimum requirement)
- 8 GB RAM (Minimum requirement)
- PC x64-bit CPU

5.2 Software Requirement:

- Windows/Mac/Linux
- Python-3.9.1
- VS Code/Anaconda/Spyder/Google Colab
- Python Extension for VS Code
- Libraries:
  - librosa
  - pandas
  - seaborn
  - numpy
  - tqdm
  - matplotlib
  - scikit-learn
  - tensorflow

6. Conclusion & Future Work

There are so many research papers that tells us how an algorithm works and how to predict any model or algorithm. Those references helped us a lot in achieving our project goal. As a result, we have successfully done a comparative analysis for accuracy rate, training time and prediction time.In future work; we can also do comparative analysis on many models to get a better understanding of any model. We can also develop User Interface for this project. Also, we can deploy an updated model that allows users to record their surroundings using a mic and get the desired output as well as we can deploy this project on cloud. This way, users can easily use the model deployment anywhere without any restrictions like file format not supported or large file size and so on.

7. Feasibility Study

A feasibility study is an analysis that takes all of a project's important factors into account—including economic, technical, legal, and scheduling problems—to ascertain the likelihood of completing the project successfully. Project directors use feasibility studies to distinguish the pros and cons of undertaking a project before they invest a lot of time and money into it.

Technical Feasibility

Gathering dataset online

The first technical problem we face is gathering the dataset. Because we are using .wav files, it took us longer to collect audio files comprising various voices. Though it took longer than expected, the subsequent process was relatively simple.

Memory used

Because the dataset we utilized was 6.60GB in size, training and testing each model required more memory and time. It nearly took us an hour to complete the period for each model. We didn't utilize much RAM because we used Google Colab.

Deploying the model

Initially, we considered utilizing any model and uploading the .wav file to acquire the appropriate outcome, as our major goal is to assist deaf individuals in recording and identifying any noises present. However, because we are also performing comparative analysis, we decided to display the outcome predicted by each of the models that we employed for better performance. This work was technically achievable as well.

Accessibility

The hardware required for this project is straightforward and widely available. All that is required is any type of optical instrument and a computer to interpret and assess the acquired sights.

Economic Feasibility

The question of whether the model can be proven to be economically feasible is one of the most significant barriers and misconceptions of any new technology. The following study explains the project's financial viability. This is a low-cost project that may be used by anyone who has a touch phone or a laptop. There aren't many hardware and software costs to worry about. The only thing to consider is the cost of maintenance. Following the successful deployment of our project, we must regularly evaluate the operation of the four projects; if there are more users, we must upgrade our project so that more users can benefit.

Social Feasibility

Deaf and blind people will be unaware of how their surroundings treat them. They will gain completely from our project. All they have to do is document their surroundings and be aware of what is going on around them. This will make them happy, and their worry of not knowing what is going on around them will be gone.

Environmental Feasibility

An Environmental Feasibility Study evaluates the environmental and social viability of a planned development, identifying potential challenges and dangers to the successful completion of the proposed development. Solutions and mitigating strategies are being researched. The goal of an Environmental Due Diligence Report is to assess potential risks and liabilities related to environmental and health and safety issues such as land contamination before entering into any contractual agreements. This is critical since these risks could result in financial liabilities for the parties concerned.

Environment and Health

This project is extremely beneficial to deaf individuals. The recording of their surroundings is critical to the success of our initiative. There will be no harm done to the environment's health because our project is entirely technical, and its safety will not be jeopardized by any factor.

8. References

[1] R. Kohavi and F. Provost, "Glossary of Terms", Machine Learning, vol. 30, no. 2-3, pp. 271- 274, 1998.

[2] J. Mueller and L. Massaron, Machine Learning For Dummies, 1st ed. 2016, pp. 40-43.

[3] M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of Machine Learning, 2nd ed. The MIT Press, 2012, pp. 101-105.

[4] S. Geman, E. Bienenstock and R. Doursat, "Neural Networks and the Bias/Variance Dilemma", Neural Computation, vol. 4, no. 1, pp. 1-58, 1992