Discrimination of Time Signals with CNN

Professor Andreas Pech
Individual Project
Aamir Muhammad: 1272918
amuhamma@stud.fra-uas.de

***Abstract*— This documentation enlightens a brief overview on how we can differentiate Time Signals using Convolution Neural Networks (CNN) strategies. The concept of supervised deep learning was the key behind implementing this project. Extracting the key Features from the data, assigning Labels to the data, and implementing the chosen machine learning algorithm on that data. The implementation was done using the Conv1D algorithm. The documentation is aided with the theoretical background and flow diagrams to make it easily understandable. Intuitively the given data is pre-processed, labeled, and Stacked. After that, the stacked data has been split into training and testing data for validation. The Sequential model with Dense Layer and Max Pooling Layer has been incorporated. For model compilation loss function of sparse_categorical_crossentropy and RMSprop optimizer is used. The model is evaluated, prediction has been made for the test data, and Confusion Matrix has been displayed.**

Python programming language with Keras neural network library and open-source library TensorFlow has been used. The final output gives the accuracy and loss for the algorithm. Moreover, with the help of this algorithm, we can only predict and differentiate between the 2 sound sources namely 'sound-source-A' and 'sound-source-G'. if we want to predict signals other than these it is necessary to train the model beforehand.

Keywords*— *Time Signals, supervised deep Learning, Feature Extraction, labeling, Sparse_categorical_crossentropy, RMSprop optimizer, TensorFlow, Keras

I. INTRODUCTION

The project in Machine Learning is to use Convolutional Neural Networks for the differentiation or classification of Time Signals for example sound sources. The Flow Diagram of how the project work is carried out is given in *Figure 1*. We will briefly discuss it.

In Section II Theoretical topics such as *Time Signals*, Machine *Learning, Activation Functions, convolution neural networks and Keras library* related to the project have been discussed briefly and references have been added with every topic. How the project is done is described in detail in Section III *Methodology*. *Accuracy*, *Loss*, and *Prediction* are discussed in Section IV *Results*. *References* are added and presented in Section V. And finally, the *Python Code* is listed in Section VI *Index*.

We have two sound sources namely ***Sound Source A*** and ***Sound Source G***. The data is in the *XLSX* file. For training, a total of 150 samples of *A* and 200 samples of *G* is used. First, the data has been analyzed and preprocessed. Sound source *A* is labeled as *0's* and sound source *G* is labeled as *1's*. After this, the data has been split up into training and testing sets. This testing set will be used for the validation of the algorithm.

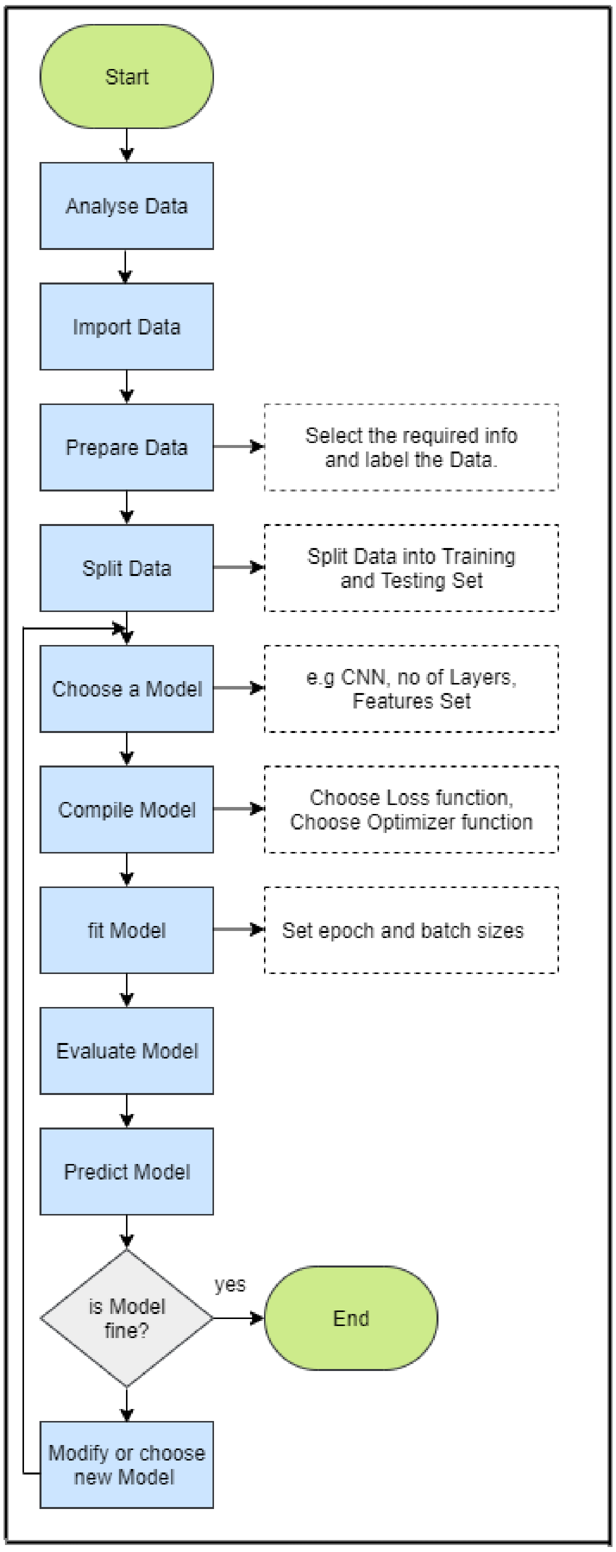

Start
Analyse Data
Import Data
Prepare Data
Select the required info and label the Data.
Split Data
Split Data into Training and Testing Set
Choose a Model
e.g CNN, no of Layers, Features Set
Compile Model
Choose Loss function, Choose Optimizer function
fit Model
Set epoch and batch sizes
Evaluate Model
Predict Model
is Model fine?
yes
End
Modify or choose new Model

Figure 1: Conv1D Model Flow Diagram

After splitting of data, we start with *Sequential Model*. *Conv1D* has been used in this model the reasons for the selection of this algorithm will be discussed in detail in Section III. Two *Conv1D* layers two *Dense* layers two *max-pooling* layers and two *dropout* layers have been utilized. The model has been compiled by using *RMSprop Optimizer* and *sparse_categorical_crossentropy* loss function. The actual learning of the model has been done in 10 Epochs using *Model.fit( )*. Finally, the algorithm has been tested with 5 *test files* each containing 50 samples and a prediction has been made. After the evaluation of the Model confusion matrix is being displayed to check and make comparison between actual and predict values. The predicted output will be either *'A'* or *'G'* if the corresponding test sample is either ***sound source A*** or ***sound source G*** respectively.

First, we tried to address the problem by using *Conv2D*. In *Conv2D* case, we convert each sample to a *Spectogram*[1] and then train the *Conv2D* Model with those images. But the accuracy we got was very low somewhere between 0.5 to 0.6. As the data is Time signals and *Conv1D* work best for this scenario that is why this model is used.

The algorithm has been tested with different *Dense Layers* and *Optimizers*. Overall, it is observed that Conv1D is the best approach for dealing with *Time Signals* as it takes very little time in training and testing and gives an accuracy of almost 100%.

Our project work can be split into distinct sections which we will discuss extensively in the following parts of this documents.

II. THEORETICAL BACKGROUND

In this section, a brief overview has been presented about theoretical topics that are necessary to discuss before jumping into the *Section II Methodology*. For instance, what is machine learning, Keras, what is CNN, which machine learning algorithm is incorporated, and which programming language is applied in the project.

A. *Machine Learning*

Machine learning *(ML)* is a branch of Artificial Intelligence *(AI)* that enables a computer to learn and improve automatically after it has been trained on an initial set of data. So basically, through experience with past data computers are deciding the data received in the future that means no need for Humans to program explicitly each time [1].

The main intention of machine learning is to permit computers to learn automatically without Human support and modify actions accordingly. In the past step by step, algorithms were designed for computers to solve the specific problems at hand telling the computer exactly how to solve this problem. So, on the computer's part, no learning was needed. Humans are the only ones to analyze and learn. This approach is only possible for small and simple tasks for more advanced and complicated tasks for us Humans it will be very difficult to manually create the needed algorithms. So rather than Human programmers, it will be more practical to let computers create and modify the algorithms itself each time it gets new information [1].

B. *Types of Machine Learning*

We have many different types of machine learning approaches. Broadly we can classify it into three *Supervised*, *Unsupervised*, and *Reinforcement* machine learning. The area of our interest is *Supervised Machine Learning* as *Conv1D* is a *Supervised Machine Learning* algorithm.

Our main task is to distinguish between 2 sound sources using *CNN*, so which means we have a classification task that falls under Supervised Learning. Further in *CNN*, the algorithm which is incorporated in this project is *Conv1D*.

A brief overview of the machine learning approaches is given in this section. Every category has a bunch of different algorithms which is not shown here but with the help of *Figure 2*, it is demonstrated where *Conv1D* falls in the Machine Learning algorithms hierarchy. As *Conv1D* is a *Supervised Machine Learning* algorithm so just a block diagram of that particular chain is demonstrated as under.

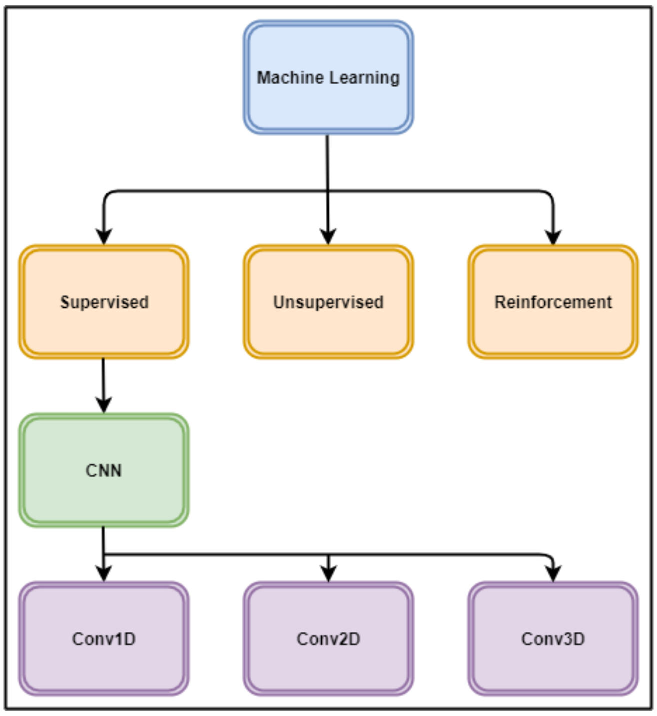


Figure 2: Conv1D position in ML hierarchy

a) *Supervised learning*

In a supervised machine learning algorithm, a Human expert teaches the computer with the training data if we get this kind of input data the answer must be this output. And

[1] Spectogram: A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

after a set of trained data, the computer must be able to predict future patterns. So supervised learning is a kind of spoon-feeding and the objective is to learn a general rule that maps inputs into outputs [2]. In our project, we will make use of the *Conv1D* algorithm. In this kind of algorithms, labeled data is used. The ***sound source A*** data is labeled *0's* and ***sound source G*** is labeled as *1's.*

b) Unsupervised learning

In this type of learning the computer is trained with unlabeled data. No labels are given to the learning algorithm, leaving it on its own to find structure in its input. So here basically we don't need an expert to help learn the computer instead, in this kind of learning working with data most of the time unsupervised learning algorithm let the experts know about the different kind of patterns present in data. These algorithms use different kinds of methods and procedures on the input data to discover patterns, mine for rules, and group and summarize the data points which help in deriving meaningful insights and describe the data better to the users [3].

c) Reinforcement Learning

Reinforcement learning directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged or 'reinforced', and non-favorable outputs are discouraged or 'punished'.

In typical reinforcement learning use-cases, such as finding the shortest route between two points on a map, the solution is not an absolute value. Instead, it takes on a score of effectiveness, expressed in a percentage value. The higher this percentage value is, the more reward is given to the algorithm. Thus, the program is trained to give the best possible solution for the best possible reward [4].

C. Time Signals Classification

From stock market prices to the spread of an epidemic, and from the recording of an audio signal to sleep monitoring, it is common for real-world data to be registered taking into account some notion of *Time*. When collected together, the measurements compose what is known as a *Time-Series* [5].

As the input is *Time Signal* so before introducing the neural network architectures, we go through some formal definitions for *Time Signals classification* [6].

Definition 1: A univariate time series $X = [x_1, x_2, . . . , x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T.

Definition 2: An M-dimensional MTS, $X = [X^1 , X^2 , . . . , X^M]$ consists of M different univariate time series with $X^i \in R^T$.

Definition 3: A dataset $D = \{(X_1, Y_1),(X_2, Y_2), . . . ,(X_N, Y_N)\}$ is a collection of pairs (X_i, Y_i) where Xi could either be a univariate or multivariate time series with Y_i as its corresponding one hot label vector. For a dataset containing K classes, the one-hot label vector Y_i is a vector of length K where each element $j \in [1, K]$ is equal to 1 if the class of X_i is j and 0 otherwise.

The neural network architecture used for this classification is CNN. Convolution can be seen as applying and sliding a filter over the time series. Unlike images, the filters exhibit only one dimension (time) instead of two dimensions (width and height). The filter can also be seen as a generic non-linear transformation of a time series. Concretely, if we are convoluting (multiplying) a filter of length 3 with a univariate time series, by setting the filter values to be equal to [1/3; 1/3; 1/3], the convolution will result in applying a moving average with a sliding window of length 3 [6].

D. Convolutional Neural Networks

A convolutional neural network is one of the most popular ANN. It is widely used in the fields of image and video recognition. It is based on the concept of convolution, a mathematical concept. It is almost similar to a multi-layer perceptron except it contains series of convolution layers and pooling layers before the fully connected hidden neuron layer. It has three important layers [7] A simple CNN can be represented as below

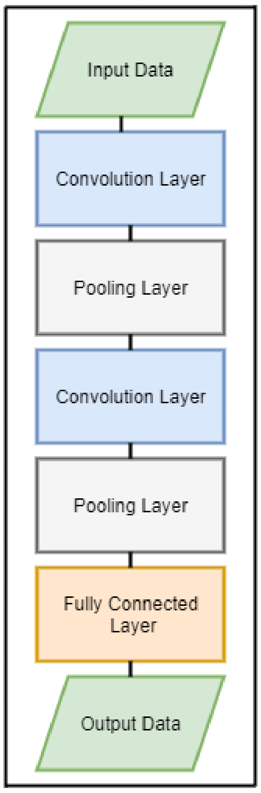


Figure 3: Simple Convolution Neural Network

- **Convolution layer** – It is the primary building block and performs computational tasks based on convolution function.
- **Pooling layer** – It is arranged next to the convolution layer and is used to reduce the size of inputs by

removing unnecessary information so computation can be performed faster.

- **Fully connected layer** – It is arranged next to series of convolution and pooling layers and classifies input into various categories.

2 series of Convolution and pooling layer is used, and it receives and processes the input (e.g., sound source). A single fully connected layer is used, and it is used to output the data (e.g., classification of the sound source) [7].

E. *Keras*

Keras is an open-source deep learning framework for python. It has been developed by an artificial intelligence researcher at Google named Francois Chollet. Leading organizations like Google, Square, Netflix, Huawei, and Uber are currently using Keras. Deep learning is supported by various libraries such as Theano, TensorFlow, Caffe, Mxnet, etc., Keras is one of the most powerful and easy-to-use Python libraries, which is built on top of popular deep learning libraries like TensorFlow, Theano, etc., for creating deep learning models. [7].

a) *Overview of Keras*

Keras runs on top of open-source machine libraries like TensorFlow, Theano, or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is a deep learning framework developed by Microsoft. It uses libraries such as Python, C#, C++, or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries for creating neural networks [7].

Keras is based on a minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications [7].

b) *Features*

Keras leverages various optimization techniques to make high-level neural network API easier and more performant. It supports the following features [7]

- Consistent, simple, and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is a user-friendly framework that runs on both CPU and GPU.
- Highly scalable of computation.

Keras is a neural network library that provides convenient methods for creating Convolutional Neural Networks (CNNs) of 1, 2, or 3 dimensions i.e. Conv1D, Conv2D, and Conv3D. One of the approaches which are used in this project is using Keras Conv1D. In the following sections, we will focus on the reasons why we used Keras Conv1D, its parameters and functions will be discussed in detail.

c) *Architecture of Keras*

Keras API can be divided into three main categories.

- Model
- Layer
- Core Modules

In *Keras*, every *Artificial Neural Network (ANN)* is represented by *Keras Models*. In turn, every *Keras Model* is a composition of *Keras Layers* and represents *ANN* layers like input, hidden layer, output layers, convolution layer, pooling layer, etc., Keras model and layer access Keras modules for activation function, loss function, regularization function, etc., Using Keras model, Keras Layer, and Keras modules, any *ANN* algorithm (*CNN*, *RNN*, etc.,) can be represented simply and efficiently [7].

The following diagram depicts the relationship between *Model*, *Layer*, and *Core Module API*.

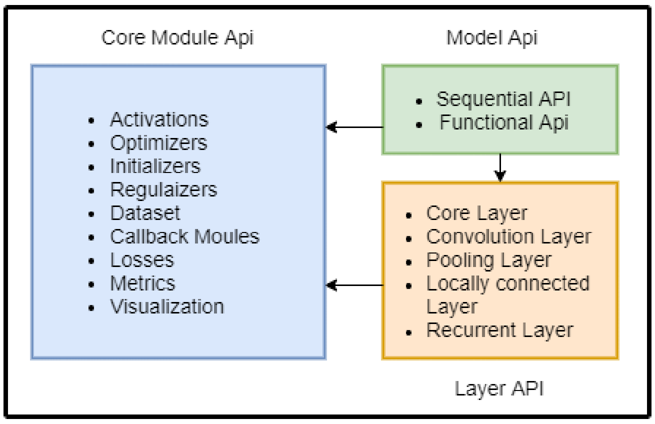


Figure 4: Keras Architecture

Let us see the overview of Keras models, Keras layers, and Keras modules.

d) *Model*

Keras Models are of two types as mentioned below.

Sequential Model: Sequential model is a linear composition of Keras Layers. The sequential model is easy, minimal as well as can represent nearly all available neural networks [7].

A simple sequential model is as follows.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential()
```

```
model.add(Dense(512, activation = 'relu',
input_shape = (784,)))
```

Line 1 imports **Sequential** model from Keras models
Line 2 imports the **Dense** layer and **Activation** module
Line 3 create a new **sequential** model using Sequential API
Line 4 adds a dense layer (Dense API) with the **relu** activation (using Activation module) function.

Sequential model exposes **Model** class to create customized models as well. We can use the sub-classing concept to create our complex model [7].

Functional API Model: Functional API is used to create complex models.

e) Layers

Each Keras layer in the Keras model represents the corresponding layer (input layer, hidden layer, and output layer) in the actual proposed neural network model. Keras provides a lot of pre-build layers so that any complex neural network can be easily created. Some of the important Keras layers are specified below [7].

- Core Layers
- Convolution Layers
- Pooling Layers
- Recurrent Layers

A simple python code to represent a neural network model using a sequential model is as follows

```
model.add(Dropout(0.2))
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes,activation=
'softmax'))
```

Line 1 adds a dropout layer to handle over-fitting.
Line 2 and **Line 4** adds dense layers with **relu** and **Softmax** activation function (using Activation module) function.
Line 3 adds another dropout layer.

Keras also provides options to create our customized layers. A customized layer can be created by sub-classing the **Keras.Layer** class and is similar to sub-classing Keras models [7].

f) Core Modules

Keras also provides a lot of built-in neural network-related functions to properly create the Keras model and Keras layers. Some of the functions are as follows [7].

- **Activations module** − Activation function is an important concept in *ANN* and activation modules provide many activation functions like *softmax*, *relu*, *selu*, *linear*, etc.
- **Loss module** − The loss module provides loss functions like *mean_squared_error*, *mean_absolute_error*, *Poisson*, etc.
- **Optimizer module** − The optimizer module provides optimizer functions like *adam*, *sgd*, *RMSprop*, etc.
- **Regularizers** − Regularizer module provides functions like *L1 regularizer*, *L2 regularizer*, etc.

g) Summary

To summarize, the Keras layer requires the below modules to complete a layer [7].

- The shape of the input data
- Number of neurons/units in the layer
- Initializers
- Regularizers/ Dropout
- Constraints
- Activations

In our case instead of *Regularizers*, we used the *Dropout* technique. Constraints are not used and Initialization is done at the beginning that is why initializers are also not used inside the *Conv1D* Model.

F. Features Extraction

Deep learning is an evolving subfield of machine learning. Deep learning involves analyzing the input in a layer-by-layer manner, where each layer progressively extracts higher-level information about the input. By using this approach, we can process a huge number of features, which makes deep learning a very powerful tool. Deep learning algorithms are also useful for the analysis of unstructured data [7].

G. Activation Function

In machine learning, the activation function is a special function used to find whether a specific neuron is activated or not. The activation function does a nonlinear transformation of the input data and thus enables the neurons to learn better. The output of a neuron depends on the activation function [7].

The output of a perceptron (neuron) is simply the result of the activation function, which accepts the summation of all input multiplied with its corresponding weight plus overall bias if any available [7].

```
result = Activation(SUMOF(input * weight) + bias)
```

So, the activation function plays an important role in the successful learning of the model. Keras provides a lot of activation functions in the activation's module. Let us learn all the activations available in the module. We have a lot of activation functions available from which we can choose namely linear, elu, selu, relu, softmax, softplus, softsign, tanh, sigmoid, hard_sigmoid, and exponential [7]. Relu is the activation function used in our project.

a) Rectified Linear Unit (ReLU)

In deep learning one of the most common Activation Function which is mostly used nowadays is ReLU. ReLU activation function offers the best performance compared to the other activation function. The main advantage of using is that it does not activate all the neurons all at once. In ReLU if the input is negative it will be converted to 0, and the neuron does not get activated. This means that at a time, only a few neurons are activated, making the network sparse and very efficient. Also, the ReLU function was one of the main advancements in the field of deep learning that led to overcoming the vanishing gradient problem. ReLU has been applied to the hidden layers as it overcomes the vanishing gradient problem [8].

ReLU function can be written as:

$$a = max\ (0, z) = \begin{cases} 0\ if\ x < 0 \\ xi\ if\ x \geq 0 \end{cases} \qquad (1)$$

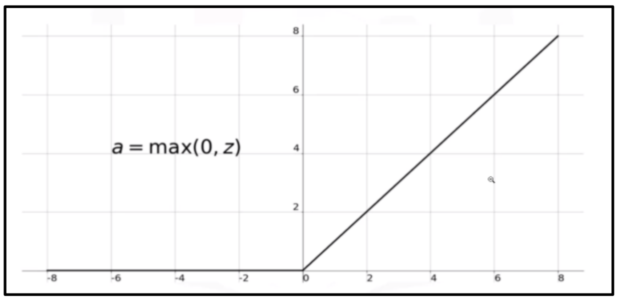


Figure 5: ReLU Function **[8]**

b) SoftMax Function

The SoftMax function is another type of AF used in neural networks to compute probability distribution from a vector of real numbers. This function generates an output that ranges between values 0 and 1 and with the sum of the probabilities being equal to 1. This activation function was applied at the output layer node as like sigmoid function this also does not support much in the vanishing gradient problem. The SoftMax function is represented as follows [9].

$$f(x) = \frac{e^x}{\sum_j e^{xj}} \qquad (2)$$

H. Optimization

Optimizers are algorithms or methods used to change the attributes of your neural network such as ***weights*** and ***learning rate*** to reduce the losses. How you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use [10].

We have a lot of optimizers that we can incorporate into our project, but we test our algorithm with ***RMSprop.*** The results of the optimizer will be discussed in upcoming sections.

a) RMSprop

In machine learning a lot of optimizers are available namely Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent (sgd), adam, RMSprop, Adagrad, AdaDelta, and many more. But in this project, RMSprop is used [11].

RMSprop is a gradient-based optimization technique used in training neural networks. Gradients of very complex functions like neural networks tend to either vanish or explode as the data propagates through the function.

RMSprop was developed as a stochastic technique for mini-batch learning. RMSprop deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing [11].

Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. This means that the learning rate changes over time [11].

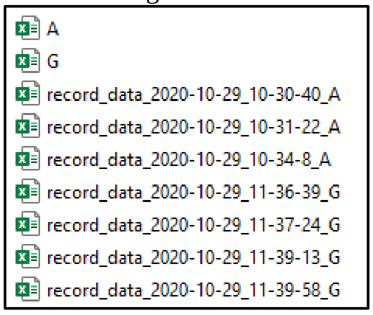
III. METHODOLOGY

A detailed step-by-step procedure for the discrimination of Time Signal is discussed in this section. Conv1D Sequential model was created using python and some defined libraries from tensor flow, Keras, and sklearn.model_selection were also used in model creation.

A. Arranging training data

To train any neural network first we need data. Deep learning requires a lot of input data to successfully learn and predict the result. first, collect as much data as possible. After collection analyzes the data and acquires a good understanding of the data. A better understanding of the data is required to select the correct ANN algorithm.

So basically we have already available excel data files from two sound sources A and G. Each file contains 50 samples of the respective sound source. As we have 3 files of A that means 150 samples of A and 4 files of G that means 200 samples of G. The data of sound source A is combined in excel file A and sound source G is combined in G manually to make training easy. The amount of given data for training and validation is shown in figure 6.

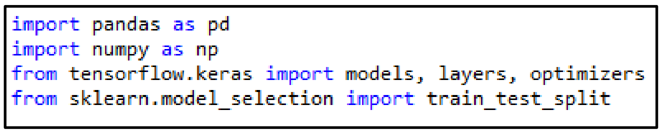


A
G
record_data_2020-10-29_10-30-40_A
record_data_2020-10-29_10-31-22_A
record_data_2020-10-29_10-34-8_A
record_data_2020-10-29_11-36-39_G
record_data_2020-10-29_11-37-24_G
record_data_2020-10-29_11-39-13_G
record_data_2020-10-29_11-39-58_G

Figure 6: Training Data

B. *Importing Required Libraries Models and Layers*

The first step is to import all the necessary libraries required to create a Conv1D model for Time Signal classification. The following figure shows the essential python packages for creating the model.



```
import pandas as pd
import numpy as np
from tensorflow.keras import models, layers, optimizers
from sklearn.model_selection import train_test_split
```

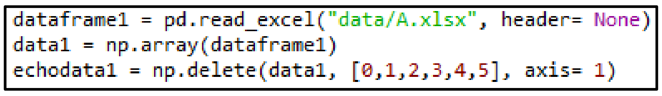
Figure 7: Libraries Models Layers and Optimizer

The most important library imported for this project is the Tensorflow-Keras library. A TensorFlow framework is very useful for creating a deep learning model. Tensor flow accepts the data in the form of multi-dimensional arrays called tensors. Keras, on the other hand, is also a very useful library which contains all the deep learning tools for creating and training model very easily. Keras is a very nice API that runs on top of frameworks like Tensor Flow.

We will use *Sequential Model* from `models` different layers namely *input*, *Conv1D*, *Dense*, *MaxPooling*, *Dropout*, and *Flatten* from ***layers*** and *RMSprop optimizer* from `optimizers` library. Similarly, another library used is sklearn for the **`test_train_split`** functionality.

C. *Preparing Data for Training and Testing*

After importing the required libraries, the next step is to import the data files A and G. first the data is converted to data frames **`(dataframe1, dataframe2, pred_dataframe)`** after that the first 6 columns are deleted as it is not necessary, and the data is saved in the form of Array. The screenshot of the python code for preparing the training and testing data is given as under.



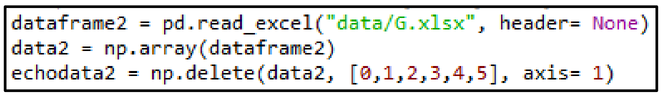
```
dataframe1 = pd.read_excel("data/A.xlsx", header= None)
data1 = np.array(dataframe1)
echodata1 = np.delete(data1, [0,1,2,3,4,5], axis= 1)
```

Figure 8: Sound Source A

To import an Excel file into Python use ***`Pandas`*****.** **`Pandas`** is a powerful Python package that can be used to perform statistical analysis. To accomplish this goal, **`read_excel ()`** command has been used.

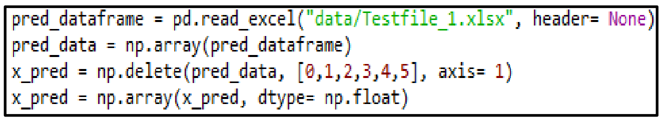
The excel file **`A.xlsx`** is imported from the *"data"* folder to the python environment. **`header`** means the first row in the excel data file. As we don't have a header, so **`None`** is used. The imported file is then assigned to a Data frame **`dataframe1`**. We need to convert a **`Pandas dataframe`** to **`Numpy`** Array to perform some high-level mathematical functions supported by the **`Numpy`** package.

The same procedure is repeated for sound source G and Test data.



```
dataframe2 = pd.read_excel("data/G.xlsx", header= None)
data2 = np.array(dataframe2)
echodata2 = np.delete(data2, [0,1,2,3,4,5], axis= 1)
```

Figure 9: Sound Source G



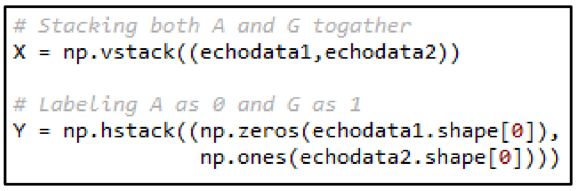
```
pred_dataframe = pd.read_excel("data/Testfile_1.xlsx", header= None)
pred_data = np.array(pred_dataframe)
x_pred = np.delete(pred_data, [0,1,2,3,4,5], axis= 1)
x_pred = np.array(x_pred, dtype= np.float)
```

Figure 10: Test Data

The python code for the full model can also be referenced from the Index Section in the end.

D. *Stacking and Labeling Data*

As we will process both the sound sources A and G simultaneously, so we need to stack the arrays **`echodata1`** and **`echodata2`** vertically to create ***features*** for the Convolution Neural Network Model using the **`vstack`** command.



```
# Stacking both A and G togather
X = np.vstack((echodata1,echodata2))

# Labeling A as 0 and G as 1
Y = np.hstack((np.zeros(echodata1.shape[0]),
               np.ones(echodata2.shape[0])))
```

Figure 11: Stacking and Labeling Data

Secondly, we need to ***label*** our data. So, the sound source A is labeled as zeros and sound source G is labeled as ones. Then the set of labels are stack horizontally using the **`hstack`** command.

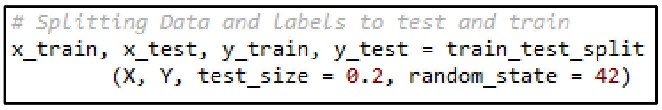
So, during the training phase, the model learns that sound source A is label 0 and sound source G is label 1. So, in the prediction and evaluation stage making use of features and labels it is easy to discriminate between the sound sources.

E. *Splitting Data into Test and Train*

Split the stacked data into training and test datasets. Test data will be used to evaluate the prediction of the algorithm and to cross-check the efficiency of the learning process.

`train_test_split()` is a function in *Sklearn model selection* for splitting data arrays into two subsets for training data and testing data. This function is used so no need to divide the dataset manually.

By default, **`train_test_split`** will make random partitions for the two subsets. However, we can also specify a random state for the operation [12].



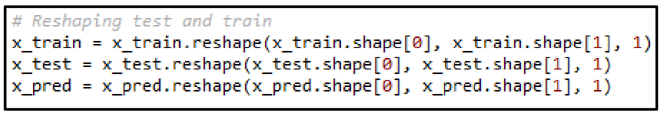
```
# Splitting Data and Labels to test and train
x_train, x_test, y_train, y_test = train_test_split
        (X, Y, test_size = 0.2, random_state = 42)
```

Figure 12: Splitting Data

`X, Y:` is the dataset being selected to use.
`train_size:` This parameter sets the size of the training dataset. There are three options: None, which is the default, Int, which requires the exact number of samples, and float, which ranges from 0.1 to 1.0.
`test_size:` This parameter specifies the size of the testing dataset. The default state suits the training size. It will be set to 0.2 if the training size is set to default.
`random_state:` The default mode performs a random split using **`np.random`**. Alternatively, you can add an integer using an exact number. Here a random state of 42 is used [12].

F. Reshaping train and test Data

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension or introducing a new dimension. But overall, the number of elements will be the same. In our case, we are only introducing the third dimension for the model to work correctly. The dimension of the arrays must be compatible for the computation to take place in the following layers.



```
# Reshaping test and train
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)
x_pred = x_pred.reshape(x_pred.shape[0], x_pred.shape[1], 1)
```

Figure 13: Reshaping Data

G. Conv1D Model

Convolutional Neural Network (CNN) models were mainly developed for image classification, in which the model accepts a two-dimensional input representing an image’s pixels and color channels, in a process called feature learning.

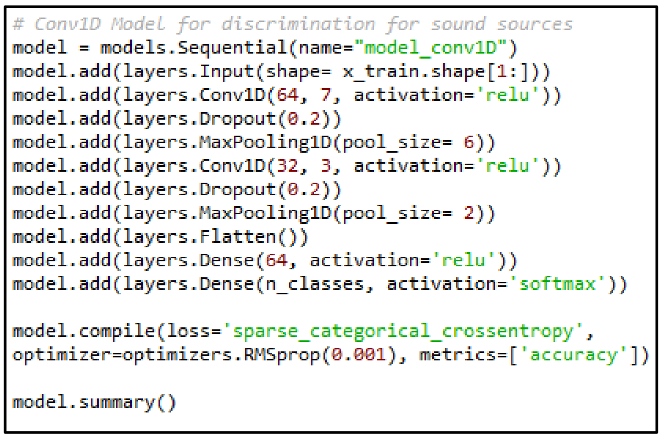
This same process can be applied to one-dimensional sequences of data. The model extract features from sequence data and maps the internal features of the sequence. A 1D CNN is very effective for deriving features from a fixed-length segment of the overall dataset, where it is not so important where the feature is located in the segment.

As we have Time Signal with a fixed length of data, that is one of the main reasons this approach is used [13].

a) Conv1D works well for

- Analysis of a time series of sensor data.
- Analysis of signal data over a fixed-length period, for example, an audio recording.
- Natural Language Processing (NLP), although Recurrent Neural Networks which leverage Long Short Term Memory (LSTM) cells are more promising than CNN as they take into account the proximity of words to create trainable patterns [13].

The python code for the model can also be referenced from the Index Section and a screenshot is also given the figure 14. Each of the layers and its argument will be discussed here in detail.



```
# Conv1D Model for discrimination for sound sources
model = models.Sequential(name="model_conv1D")
model.add(layers.Input(shape= x_train.shape[1:]))
model.add(layers.Conv1D(64, 7, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 6))
model.add(layers.Conv1D(32, 3, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 2))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(n_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer=optimizers.RMSprop(0.001), metrics=['accuracy'])

model.summary()
```

Figure 14: Conv1D Model

b) Conv1D Sequential Model

In the Conv1D case, we will use Sequential Model. For that, we have to import models from tensorflow.keras which is already mentioned in figure 7.

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. The core data structure of Keras is a model, which lets us organize and design layers. Sequential and Functional are two ways to build Keras models. A sequential model is the simplest type of model, a linear stack of layers. If we need to build arbitrary graphs of layers, Keras functional API can do that for us [14].

```
model = models.Sequential(name = "model_conv1D”)
```

By using this command, we start with a sequential model giving it the name of *“model_conv1D”*.

c) Input data to the Model

By using the below command input layer is added to the model making use of **`layers`** as per figure 7 and data is given to the model through the input layer.

```
model.add(layers.Input(shape= x_train.shape[1:]))
```

d) Adding Conv1D layer to the Model

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs [15]. Two times Conv1D layer is used in the Model with a different number of filters and kernel-size. The code is mentioned as under.

```
model.add(layers.Conv1D(64, 7, activation='relu'))
model.add(layers.Conv1D(32, 3, activation='relu'))
```

`filter:` Integer, the dimensionality of the output space (i.e., the number of output filters in the convolution) or the number of resulting feature maps. filters of 64 and 32 are used respectively.

`Kernel_size:` An integer or tuple/list of a single integer, specifying the length of the 1D convolution window. kernel_size of 7 and 3 is used.

`Activation function:` **activation** refers to the activation function of the layer. It can be specified simply by the name of the function and the layer will use corresponding activators. relu activation function is used.

e) Adding Dropout layer to the Model

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. This significantly reduces overfitting [16].

```
model.add(keras.layers.Dropout(0.2))
```

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Here we set the rate to 0.2. Inputs not set to 0 are scaled up by *1/ (1 - rate)* such that the sum over all inputs is unchanged. Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference [17].

f) Adding MaxPooling layer to the Model

Max pooling operation for 1D temporal data down samples the input representation by taking the maximum value over the window defined by `pool_size.` The window is shifted by strides. A common CNN model architecture is to have several convolution and pooling layers stacked one after the other [18].

```
model.add(layers.MaxPooling1D(pool_size= 6))
model.add(layers.MaxPooling1D(pool_size= 2))
```

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust [7].

We have different types of pooling. **Max pooling** is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after the max-pooling layer would be a feature map containing the most prominent features of the previous feature map. **Average pooling** computes the average of the elements present in the region of the feature map covered by the filter. Thus, while max-pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch [7].

g) Adding Flatten( ) Layer

Flatten is used to flatten the input. For example, if flatten is applied to a layer having an input shape as (2,2), then the output shape of the layer will be a single row of 4 elements which is then used as input to the following layers.

```
model.add(layers.Flatten())
```

h) Adding Dense Layer

Dense layer is the regular deeply connected neural network layer. It is the most common and frequently used layer. A dense layer does the below operation on the input and returns the output [7].

```
output = activation(dot(input, kernel) + bias)
```

input: represent the input data
kernel: represent the weight data
dot: represent NumPy dot product of all input and its corresponding weights
bias: represent a biased value used in machine learning to optimize the model
activation: represent the activation function.

The dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models. In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are parameters that can be trained and updated with the help of backpropagation [19].

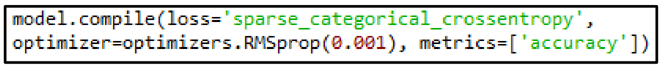
```
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(n_classes,activation='soft
max'))
```

The output generated by the dense layer is an 'm' dimensional vector. Thus, a dense layer is used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, translation on the vector [19]. In our case, we use the first Dense layer with an output of 64 and the second one according to the number of classes. Here we have 2 classes because of two sound sources so `n_classes` is 2. And activation function of relu is used. The activation parameter helps apply the element-wise activation function in a dense layer.

i) Compiling the Model

In the end, we have to compile the model with loss function and optimizer. Machines learn using a loss function. It's a method of evaluating how well specific algorithm models the given data. If predictions deviate too much from actual results, the loss function would give a very large number.

Gradually, with the help of some optimization function, the loss function learns to reduce the error in prediction [20]. As our classification is binary because we have 2 signals and is labeled as 0 and 1 so **sparse_categorical_crossentropy** is chosen for this Model. and the optimizer is RMSprop. RMSprop is already discussed in detail in section II (H).



```
model.compile(loss='sparse_categorical_crossentropy',
optimizer=optimizers.RMSprop(0.001), metrics=['accuracy'])
```

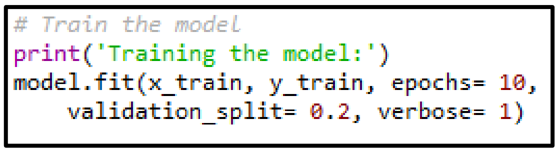
Figure 15: Compile the Model

j) Summarize the Model

The model is summarized with **`model.summary().`** A summary is textual what it does is give information about the layers and their order in the model, output shape of layer, number of the parameter in each layer, and the total number of parameters. The summary of the Conv1D model can be seen in the right column.

k) fit the Model

The actual learning process will be done in this stage using training data set. Along with training data, no of **epochs** is also mentioned here, epoch is a hyperparameter that defines how many times the learning algorithm will work through the entire training dataset. In a single epoch, each parameter can update once [21]. **Split validation** is a hypothetical dataset which is separated explicitly for testing purpose. In this case, 0.2 means 20% of the training dataset will be used for validation purposes, and the remaining 80% for training. **Verbose** mode is a setup in the computer which provides additional details as to what software will load during the startup or it will provide the detailed output for diagnostic purposes.

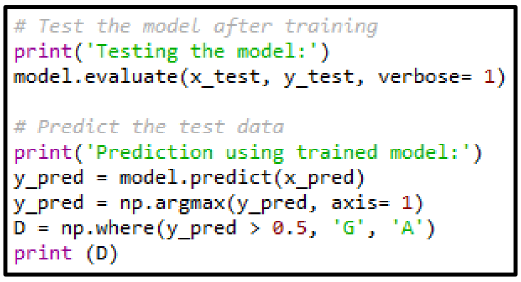


```
# Train the model
print('Training the model:')
model.fit(x_train, y_train, epochs= 10,
    validation_split= 0.2, verbose= 1)
```

Figure 16: Model Training

l) Evaluate and Predict Model

Evaluate the model by predicting the output for test data and cross-comparing the prediction with the actual result of the test data. Predict the output for the unknown input data (other than existing training and test data) [7]. In this section, the model has been evaluated and predicts the overall model. In evaluating portion we predict the data of the test portion which is separated from the training data. Where **`model.predict()`** is used for the new incoming test data. In the end, the output signal will be given names correspondingly as ‘A’ and ‘G’.



```
# Test the model after training
print('Testing the model:')
model.evaluate(x_test, y_test, verbose= 1)

# Predict the test data
print('Prediction using trained model:')
y_pred = model.predict(x_pred)
y_pred = np.argmax(y_pred, axis= 1)
D = np.where(y_pred > 0.5, 'G', 'A')
print (D)
```

Figure 17: Evaluate and Predict

m) Confusion Matrix

In the field of machine learning a confusion matrix is a specific table layout that shows the result of an algorithm. Each row of the matrix represents the True labels, while each column represents the predicted labels. Each entry in a confusion matrix denotes the number of predictions made by the model where it classified the classes correctly or incorrectly.

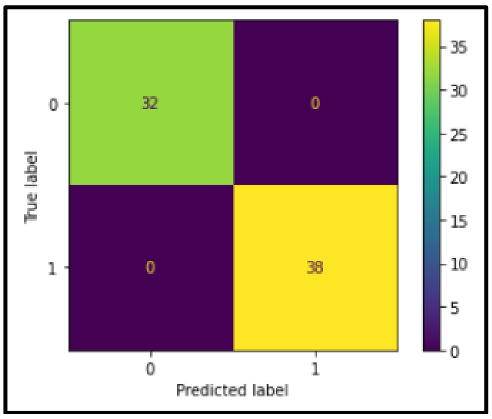
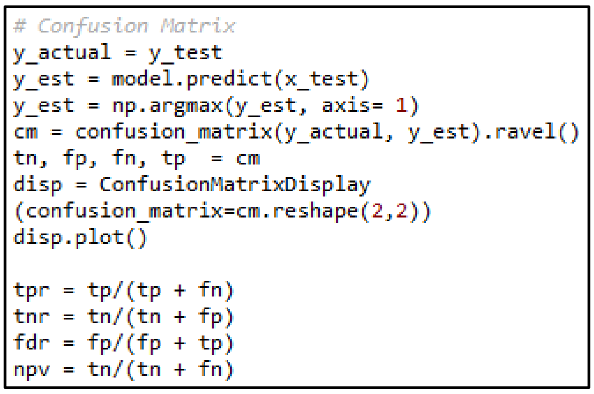


Figure 18: Confusion Matrix

It is a special kind of a table, with two dimensions ("True" and "predicted"). In the beginning we label our data for training those are true labels and then after model training and predicting the validation or test data then labels, we got form the model are predicted labels. So, in our project we took 70 samples mixed of **sound source A** and **sound source G** after the **`train_test_split( ).`** These samples were already labeled as 0 and 1. So after the model is trained and prediction has been made for those samples. The model predicts it if either it is **A** or **G**. for example, we have

True Labels = [1,1,1,1,1,1,1,1,0,0, 0,…,0,0,0,1,0,1,0,0,0]
Predicted Labels= [1,1,1,1,1,1,1,1,0,0,0,…,0,0,0,1,0,1,0,0,0]

As our model is 100% accurate so the labels will match if the accuracy is not 100% surely some labels will mismatch. To show the overall performance of the model confusion matrix is used which make it a lot easier to check the overall behavior of the model. From the confusion matrix it is evident that we have a total of 38 True Positive values means that the sample was sound source **G** which was labeled as 1 and is also predicted as 1. Similarly, 32 True Negative values mean 32 samples of sound source **A** which were labeled as 0 is predicted as 0. Which makes it a total of 70 samples. The values other than the main diagonal is zero which means the model haven't predicted wrong and is 100% accurate.



```
# Confusion Matrix
y_actual = y_test
y_est = model.predict(x_test)
y_est = np.argmax(y_est, axis= 1)
cm = confusion_matrix(y_actual, y_est).ravel()
tn, fp, fn, tp  = cm
disp = ConfusionMatrixDisplay
(confusion_matrix=cm.reshape(2,2))
disp.plot()

tpr = tp/(tp + fn)
tnr = tn/(tn + fp)
fdr = fp/(fp + tp)
npv = tn/(tn + fn)
```

Figure 19: Confusion Matrix Code

There are two sections "Prediction Class" and "Actual Class". Each class have subsections. Now using this visualized confusion matrix, we can be able to calculate the following metrics to evaluate our model accuracy:

True Positive (TP): True positive is the number of predictions where the object class is truly predicted what actual class is.

True Negative (TN): True negative is the number of predictions where the object is truly predicted to be not part of the defined class.

False Positive (FP): False positive is the number of predictions where the classifier predicts the target object as its prediction though the target object is not the actual object.

False Negative (FN): False negative is the number of predictions where the classifier makes false prediction about the object which is targeted actual object.

True Positive Rate (TPR): TPR is the accuracy which is defined as the total sum of true positives ratio the total sum of true positive and false negative for a particular object class.

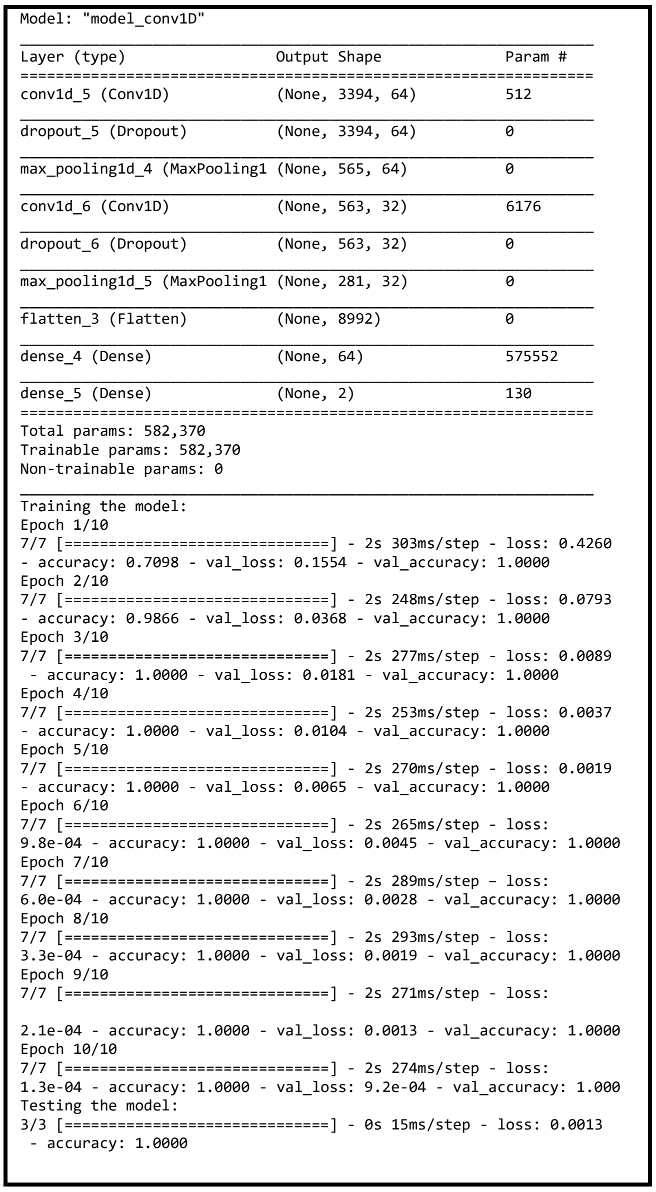
True Negative Rate (TNR): TNR is the ratio of true negative to the sum of true negatives and false positives for a particular object class.

False Discovery Rate (FDR): FDR is the ratio of false positive to the sum of false positive and true positive for a particular object class.

Negative Predictive Value (NPV): NPV is the ratio of true negative to the sum of true negative and false negative for a particular object class

IV. Results

From the results, it is evident that when the model starts training in the first 1 to 2 epochs it was learning so the accuracy was less about 70% but ultimately it achieves an accuracy of 100% from the third epoch. Also, during the testing phase, it shows us an accuracy of 100% which is a pretty good result. From the summary below we can see all the results i.e. The number of layers and parameters used in the model and number of epochs and the loss and accuracy at each epoch.



```
Model: "model_conv1D"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_5 (Conv1D)            (None, 3394, 64)          512
_________________________________________________________________
dropout_5 (Dropout)          (None, 3394, 64)          0
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 565, 64)           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 563, 32)           6176
_________________________________________________________________
dropout_6 (Dropout)          (None, 563, 32)           0
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 281, 32)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 8992)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                575552
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 130
=================================================================
Total params: 582,370
Trainable params: 582,370
Non-trainable params: 0
_________________________________________________________________
Training the model:
Epoch 1/10
7/7 [==============================] - 2s 303ms/step - loss: 0.4260
- accuracy: 0.7098 - val_loss: 0.1554 - val_accuracy: 1.0000
Epoch 2/10
7/7 [==============================] - 2s 248ms/step - loss: 0.0793
- accuracy: 0.9866 - val_loss: 0.0368 - val_accuracy: 1.0000
Epoch 3/10
7/7 [==============================] - 2s 277ms/step - loss: 0.0089
 - accuracy: 1.0000 - val_loss: 0.0181 - val_accuracy: 1.0000
Epoch 4/10
7/7 [==============================] - 2s 253ms/step - loss: 0.0037
- accuracy: 1.0000 - val_loss: 0.0104 - val_accuracy: 1.0000
Epoch 5/10
7/7 [==============================] - 2s 270ms/step - loss: 0.0019
- accuracy: 1.0000 - val_loss: 0.0065 - val_accuracy: 1.0000
Epoch 6/10
7/7 [==============================] - 2s 265ms/step - loss:
9.8e-04 - accuracy: 1.0000 - val_loss: 0.0045 - val_accuracy: 1.0000
Epoch 7/10
7/7 [==============================] - 2s 289ms/step - loss:
6.0e-04 - accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 8/10
7/7 [==============================] - 2s 293ms/step - loss:
3.3e-04 - accuracy: 1.0000 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 9/10
7/7 [==============================] - 2s 271ms/step - loss:

2.1e-04 - accuracy: 1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 10/10
7/7 [==============================] - 2s 274ms/step - loss:
1.3e-04 - accuracy: 1.0000 - val_loss: 9.2e-04 - val_accuracy: 1.000
Testing the model:
3/3 [==============================] - 0s 15ms/step - loss: 0.0013
 - accuracy: 1.0000
```

Five files were given for testing the algorithm. Each file contains 50 or more 50 samples. The prediction of the test files is given in figure 16. As we have two sound sources that are 'A' and 'G' so the result shows if either the test sample is 'A' or 'G'.

Prediction for **Testfile_1.xlsx:**

```
['A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'G' 'A'
'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'
'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A']
```

Prediction for **Testfile_2.xlsx:**

```
['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G']
```

Prediction for **Testfile_3.xlsx:**

```
['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G']
```

Prediction for **TEST 15 02 2021 1:**

```
['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'A'
'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'
'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A']
```

Prediction for **TEST 15 02 2021 2:**

```
['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G'
'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'A'
'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'G' 'A' 'A'
'A' 'A']
```

Figure 20: Test files Predicted Values

V. References

[1] Wikipedia, "Machine Learning," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning.

[2] Wikipedia, "Supervised Machine Learning," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Supervised_learning.

[3] Wikipedia, "Unsupervised Learning," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Unsupervised_learning.

[4] Potentiaco Analytic, "Potentiaco," [Online]. Available: https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/.

[5] J. Gamboa, "Deep Learning for Time-Series Analysis," University of Kaiserslautern, Kaiserslautern, Germany, 2017.

[6] H. I. Fawaz, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery,* p. 44, 14 May 2019.

[7] Tutorials Point, "Tutorials Point," [Online]. Available: https://www.tutorialspoint.com/keras/index.htm.

[8] M. Toprak, "Medium," Activation Functions for Deep Learning, 14 06 2020. [Online]. Available: https://medium.com/@toprak.mhmt/activation-functions-for-deep-learning-13d8b9b20e. [Accessed 10 01 2021].

[9] K. Goyal, "upGrad blog," 6 Types of Activation Function in Neural Networks You Need to Know, 13 02 2020. [Online]. Available: https://www.upgrad.com/blog/types-of-activation-function-in-neural-networks/. [Accessed 14 01 2021].

[10] S. Doshi, "towards data science/ Various Optimization Algorithms For Training Neural Network," 13 January 2019. [Online]. Available: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6#:~:text=Optimizers%20are%20algorithms%20or%20methods,help%20to%20get%20results%20faster.

[11] DeepAI, "DeepAI," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/rmsprop#:~:text=RMSprop%20is%20a%20gradient%20based%20optimization%20technique%20used%20in%20training%20neural%20networks.&text=This%20normalization%20balances%20the%20step,small%20gradients%20to%20av.

[12] "www.bitdegree.org/learn/train-test-split," BitDegree, [Online]. Available: https://www.bitdegree.org/learn/train-test-split#:~:text=train_test_split%20function%20first.-,What%20is%20train_test_split%3F,partitions%20for%20the%20two%20subsets..

[13] Missing Link, "missinglink.ai," 2016. [Online]. Available: https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/.

[14] M. Hanifi, 22 December 2019. [Online]. Available: https://hanifi.medium.com/sequential-api-vs-functional-api-model-in-keras-266823d7cd5e.

[15] TensorFlow, "tf.keras.layers.Conv1D," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D.

[16] G. H. A. K. I. S. a. R. S. Nitish Srivastava, "Dropout: A Simple Way to Prevent Neural Networks from overfitting," *Journal of Machine Learning Research 15 (2014) 1929-1958,* vol. 15, 2014.

[17] Tom O'Malley , "Keras," keras google group, [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/.

[18] "GeeksforGeeks," 26 August 2019. [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/#:~:text=Max%20pooling%20is%20a%20pooling,of%20the%20previous%20feature%20map..

[19] P. Sharma, "Keras Dense Layer Explained for Beginners," 20 October 2020. [Online]. Available: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/.

[20] R. Parmar, "Common Loss functions in machine learning," 02 September 2018. [Online]. Available: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[21] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," Machine Learning Mastery, 26 October 2019. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=at%20an%20epoch.-,What%20Is%20an%20Epoch%3F,update%20the%20internal%20model%20parameters..

[22] S. Ruder, "An overview of gradient descent optimization," Insight Centre for Data Analytics, Dublin, 2017.

VI. INDEX

a) Import Libraries

```
import pandas as pd
import numpy as np
from tensorflow.keras import models, layers, optimizers
from sklearn.model_selection import train_test_split
```

b) Import data files and test files

```
# import sound source A, store in form of Array and remove the first 6 column
dataframe1 = pd.read_excel("data/A.xlsx", header= None)
data1 = np.array(dataframe1)
echodata1 = np.delete(data1, [0,1,2,3,4,5], axis= 1)

# import sound source G, store in form of Array and remove the first 6 column
dataframe2 = pd.read_excel("data/G.xlsx", header= None)
data2 = np.array(dataframe2)
echodata2 = np.delete(data2, [0,1,2,3,4,5], axis= 1)

# import the TESTFILE, store in form of Array and remove the first 6 column
pred_dataframe = pd.read_excel("data/TEST 15 02 2021 2.xlsx", header= None)
pred_data = np.array(pred_dataframe)
x_pred = np.delete(pred_data, [0,1,2,3,4,5], axis= 1)
x_pred = np.array(x_pred, dtype= np.float)
```

Figure 21: Import and Prepare train and test data

c) Processing and Reshaping Data into Test and Train

```
# Stacking both A and G togather
X = np.vstack((echodata1,echodata2))

# Labeling A as 0 and G as 1
Y = np.hstack((np.zeros(echodata1.shape[0]),np.ones(echodata2.shape[0])))

# Splitting Data and labels to test and train
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 42)

# Reshaping test and train
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)
x_pred = x_pred.reshape(x_pred.shape[0], x_pred.shape[1], 1)
```

d) Conv1D Model Code

```
model = models.Sequential()
model.add(layers.Input(shape= x_train.shape[1:]))
model.add(layers.Conv1D(64, 7, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 6))
model.add(layers.Conv1D(32, 3, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 2))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(n_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer=optimizers.RMSprop(0.001), metrics=['accuracy'])

model.summary()
```

e) Training and Testing Model

```
    # Train the model
    print('Training the model:')
    model.fit(x_train, y_train, epochs= 10, validation_split= 0.2, verbose= 1)

    # Test the model after training
    print('Testing the model:')
    model.evaluate(x_test, y_test, verbose= 1)
```

f) Confusion Matrix

```
    # Confusion Matrix
    y_actual = y_test
    y_est = model.predict(x_test)
    y_est = np.argmax(y_est, axis= 1)
    cm = confusion_matrix(y_actual, y_est).ravel()
    tn, fp, fn, tp  = cm
    disp = ConfusionMatrixDisplay(confusion_matrix=cm.reshape(2,2))
    disp.plot()

    tpr = tp/(tp + fn)
    tnr = tn/(tn + fp)
    fdr = fp/(fp + tp)
    npv = tn/(tn + fn)
```

Figure 22: Diplaying Confusion Matrix Plot

g) Prediction using trained Model

```
    # Predict the test data
    print('Prediction using trained model:')
    y_pred = model.predict(x_pred)
    y_pred = np.argmax(y_pred, axis= 1)
    D = np.where(y_pred > 0.5, 'G', 'A')
    print (D)
```