Discrimination of reflected sound signals

Masters in Information Technology
Machine Learning
Abhishek Makam
1116427
abhishek@stud.fra-uas.de

Masters in Information Technology
Machine Learning
Komala Balakrishna
1325111
komala.balakrishna@stud.fra-uas.de

Masters in Information Technology
Machine Learning
Sahana Prasad
1324329
sahana.prasad@stud.fra-uas.de

***Abstract*— Deep learning is the part of Machine learning which focuses on representational learning. Convolutional Neural Network (CNN) is a form of a deep neural network. CNNs are known to have many advantages over other techniques and hence are used in problems related to patterns and image recognition. Handcrafted features and shallow trainable architectures make up the components of traditional image processing methods. With the exponential growth of deep learning, more efficient techniques are implemented to solve the problems that occur in traditional structures, allowing them to adapt to semantic, high level, deeper features. The evolution of Deep Neural Networks (DNN) also has seen a significant advancement in classification of sound. This paper covers the fundamentals of CNN's, including an overview of the various architectures, layers employed in image recognition or processing applications and classification of sound with Convolutional Neural Network (CNN) which is dependent on spectrograms obtained from different sound samples.**

Keywords—Deep Learning, Convolutional Neural Networks, Image Recognition, Sound Classification, Spectogram.

I. INTRODUCTION

Deep learning is a branch of machine learning that focuses on learning representation levels, which is a hierarchy of properties, variables, or concepts in which lower-level concepts reflect higher-level concepts and the same lower-level concepts can help describe other higher-level concepts. Deep learning is the process of learning multiple layers of representation and abstraction in order to better understand data such as text, images, and audio. Deep learning models can achieve cutting-edge precision, often outperforming humans in terms of efficiency and consistency. Each level learns to process its input data into a slightly more abstract and composite representation through deep learning. The word deep in deep learning represents the number of layers that transform the data[1]. Convolutional Neural Networks are one of the most common types of Deep Neural Networks. Convolutional Neural Networks are neural networks that are primarily used to recognize images, cluster images based on similarity (photo look), and recognize in-scene objects. As shown in Fig. 1., a neural network is a series of artificial "neurons" that are interconnected to exchange messages. The weights of the connections can be changed during the training process, allowing a properly trained network to respond precisely when presented with a pattern or picture to recognize. For feature detection, the network is made up of many layers of neurons. Each layer is made up of a large number of neurons that respond to various combinations of inputs from the previous layers[2].

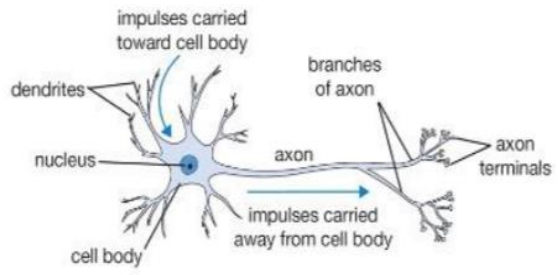

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Fig. 1. Biological Neuron

Traditionally audio recognition algorithms were used for the processing of speech and music signals. There were more problems in recognizing, classifying, and analysing environmental sounds than compared to the processing of speech and music signals because of the unstructured nature of environmental sounds. These environmental sounds will usually not have phonemes which are elementary blocks of meaningful sequences or strong stationary patterns like rhythm or melody, but it might include Spectro-temporal signatures. Hence, it is very essential to consider the nonstationary aspects of signal and capture its variations in both frequency and time domain.

The Deep Neural Networks (DNN) has obtained immense success in various learning tasks, including classification of sound. Convolutional neural network (CNN) is a type of hierarchical feature learning neural network which has a biologically inspired structure. It captures the Spectro-temporal patterns from spectrogram. Spectrogram is visual representation of spectrum of frequencies of sound and other signals which differ with time. Also, for any given task CNN will automatically start learning the unique set of features of the task.

II. CONVOLUTIONAL NEURAL NETWORK MODEL

The characteristics of CNN are similar to those of multilayer perceptron. For classification problems, complex architectures are designed by stacking multiple and different layers in a CNN, as shown in Fig. 2.

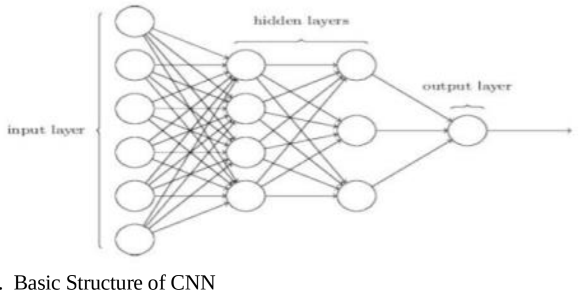


Fig. 2. Basic Structure of CNN

They have an input layer, an output layer, and at least one hidden layer(s) in between, such as a Convolution Layer, a

Pooling Layer, a Rectifier Linear Unit Layer, and a Fully Connected Layer[3].

A. *Input Layer*

The input layer of CNN takes images and resizes them before transferring them to additional layers for feature extraction. Image data is contained in the input layer. A three-dimensional matrix is used to represent image data.

B. *Convolution Layer*

Owing to the extraction of image features within this layer, the Convolution layer is also known as the extractor layer. The first convolution layer removes low-level features such as borders, curves, and edges. In the upper layers, higher-level features are extracted. The existence of a convolution function at that position determines the performance of the Convolution layer. We're going to do some element-by-element multiplication here. H kernels or filters of size fh * fw * d are convolved with input of size h * w * d. By convolving an input with one kernel and H kernels, one output feature is generated, and H features are generated independently, yielding an output of size (h-fh+1)*(w-fw+1)*1. Starting at the top left corner of the image, each kernel moves from left to right, one element at a time.

C. *Pooling Layer*

Sub sampling layer is another name for the pooling layer. In this layer, the feature's resolution is reduced. The pooling layer is in charge of reducing the input image's spatial volume after convolution. Max pooling or Average pooling may be used to minimize the spatial volume of the input image. The maximum value from the kernel-covered portion of the image is returned by Max Pooling. Similarly, average Pooling returns the average value from the image's kernel-covered area. Fig. 3. illustrates this[4].

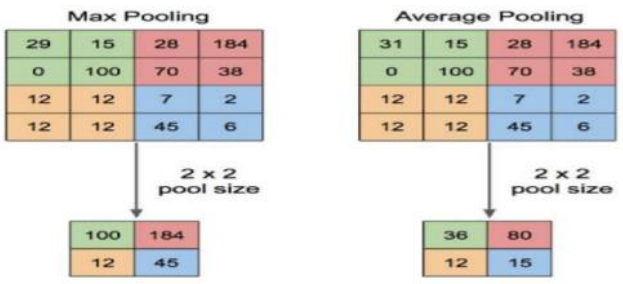


Fig. 3. Max and Average Pooling with size (2*2)

D. *ReLU Layer*

Nonlinear triggering is implemented using Rectified Linear Units in CNNs. The function y=max(x,0) is used to ensure that the input and output sizes are the same. It improves the decision function's nonlinear properties. It replaces the negative number obtained from the pooling layer with a 0 to keep the CNN mathematically stable, as seen in Fig. 4. The key benefit of ReLU is that it allows you to practice quicker.

E. *Fully Connected Layer*

The Fully Connected Layer is the final layer of CNN. This layer categorizes the high-level filtered images and assigns labels to them. Each element of the output layer is determined using the elements obtained from the features of the previous layers.

III. ARCHITECTURE OF CONVOLUTION NEURAL NETWORK

In CNN, there are a variety of architectures. Then there's the issue of deciding on the right architecture. The model is best described as simple or efficient in terms of producing accuracy. Over the years, various attempts have been made to improve CNN's results. The evolution of CNN architectures can be divided into five distinct periods, as described below. Regularization, parameter optimization, and other improvements have been classified[5].

A. *LeNet*

LeNet was mostly used for optical and character recognition in the 1990s. CNN was successfully developed by Yann LeCun. Their LeNet experiments presented the first credible proof that convolutional neural networks could be trained using back propagation.

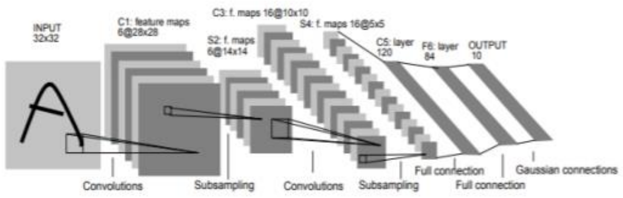


Fig. 4. LeNet Architecture

B. *AlexNet*

Alex Krizhevsky was the author of the architecture. Alexnet was an ILSVRC 2012 winning architecture. The input to this architecture is an RGB image with a size of 2456*256 pixels, which means that the images in the training and test data must be 256*256 pixels. Photos that aren't this size must be translated to 256*256 pixels. This architecture has 5 convolutional layers and 3 fully connected layers.

C. *ZFNet*

Zeiler & Fergus Net is the abbreviation for ZFNet. This design outperforms AlexNet by a broad margin. It was an improvement over AlexNet because the architecture's hyperparameters were tweaked, especially the size of the intermediate convolution layers and the filter and stride sizes on the first layer. By reducing the filter size to 7*7, it was able to achieve a significant reduction in filter size over AlexNet.

D. *Google Net*

GoogLeNet is a 22-layer deep neural network that has been pre-trained. The most important contribution was the creation of an Inception Module, which greatly reduced the number of parameters in the network. Transfer learning can be used to retrain the GoogleNet network to perform new tasks. When it comes to transfer learning, the most common approach is to use networks that have been pre-trained on the ImageNet data collection.

E. *VGG Net*

Visual Geometry Group is the abbreviation for VGG. Simonyan and Zisserman came up with the idea. VGGNet is made up of 16 layers of convolutional neural networks. It appeals to me because of the standardized architecture. It is currently the best choice for extracting features from photos in the group. VGG uses 224*224 RGB images. VGG uses a lot of memory and has a lot of settings.

F. ResNet

ResNet is an acronym that stands for Residual Neural Network. It's the most impressive work in computer vision or deep learning that I've ever seen. ResNet allows for the training of hundreds or even thousands of layers. It employs batch normalization and special skip connections. Layers that are completely connected are also absent from this architecture.

IV. CNN Usage

Despite the fact that neural networks and other pattern recognition methods have been around for over 50 years, the field of Convolutional Neural Networks has seen tremendous growth in recent years. The advantages of using CNN in the field of image recognition are discussed in this section[6].

- Image recognition using CNN is resistant to changes or distortions such as changes in lighting conditions, positions, vertical or horizontal adjustments, and so on.
- As opposed to shallow versions, a deeper architecture has considerably more expressive potential.
- The architecture of CNN allows for the optimization of several tasks at the same time.
- As the number of parameters in a CNN is greatly decreased, the training time decreases proportionally.
- Since the same coefficients are used in various locations, memory requirements are significantly reduced.

V. Hyper-Parameters

A hyperparameter is a parameter that is set before the learning process begins. Such parameters are tunable and can have a significant effect on a model's ability to train. Hyperparameters may have a direct effect on the training of machine learning algorithms. It is thus critical to comprehend how to maximize them in order to achieve optimum performance[7].

A. Number of Filters

As the depth of the feature map decreases, the number of filters in the layers closest to the input layer appear to decrease, while higher layers will have more.

B. Learning Rate

Learning Rate is a Gradient Descent algorithm parameter that lets us monitor how much our network's weight changes in response to gradient loss. The model's training would be incredibly slow if the learning rate was kept low. This will result in over-fitting since the weights would be adjusted in small increments.

C. Batch Rate

The batch size refers to the total number of training samples spread across the network. Our aim is to maximize productivity when practicing deep learning by and the amount of time it takes to compute. Though the learning rate's value has no effect on training time, batch size does.

D. Stride

From left to right, top to bottom, the filter moves across the image. The stride of the filter determines how it convolves or travels around the input image. The default stride is (1,1).

E. Padding

We use the padding hyper parameter to keep details at the edges. To preserve the spatial sizes, padding is used to add rows and columns of zeros.

F. Activation Function

Depending on the function it serves, an activation filter may be linear or nonlinear. It is used to monitor the network's performance. It's a node that's either in the middle or at the end of a Neural Network. It computes the weighted sum of input and biases to determine if the neuron will fire or not. Sigmoid Function, Hyperbolic Tangent Function (Tanh), Softmax Function, Rectified Linear Unit (ReLU) Function, and other Activation Functions are used[8].

G. Epochs

Epoch is a hyper parameter that can also be defined as the number of iterations a neural network has gone through. It consists of two passes, one forward and one backward. We need more than one epoch because using only one causes underfitting. Choosing the right number of epochs becomes increasingly necessary as the curve progresses from underfitting to optimal to overfitting. The number of epochs is not fixed or normal, and it varies depending on the dataset.

H. Batch Size

The number or quantity of samples to work through is defined by batch size. We must split the dataset into batches since we cannot pass the entire dataset to the neural network at once.

VI. Applications of CNN

Image recognition or classification, such as the identification of images obtained from satellites containing roads or the classification of letters and digits written by hand, is the most common application for CNNs. CNN 4 excels at signal processing and image segmentation as well.

A. Facial Recognition

The facial recognition task is broken down into the following components using CNN:

- Identifying all the faces in the image.
- Highlighting every face despite of various external factors like angle, light, pose etc.
- In order to match a face with a name , the collected data is compared with the existing data.

B. Document Analysis

In document analysis, CNNs are commonly used. This is not only useful for handwriting research, but it also plays a significant role in recognizers. It must perform a large number of commands per minute in order to scan an individual's handwriting and compare it to a large database. The error rates have been reduced thanks to advances in models and algorithms.

C. *Climate Change*

Without a doubt, one of the most important issues in recent years has been the changing environment. Most climate forecasting research is currently focused on various physics-based models, such as earth system models. The neural network has a high learning capacity and is a useful method for dealing with difficult-to-describe dynamic nonlinear processes and interactions.

D. *Text Classification*

In today's world, Text Classification is a requisite technology. Initially, text classifiers were used to categorize postal mail. The first Optical Character Recognition program, which allowed for the recognition of handwritten types, was created in the early 1970s. However, modern techniques such as Convolution Neural Networks allow us to search and understand words with a degree of precision never seen before in history.

VII. Methodology

This section goes through a thorough step-by-step process for discriminating Time Signal. Conv1D Sequential model was built with Python and libraries from Tensor Flow, Keras, and Sklearn. Model_selection were also used in model creation.

A. *Reading and Merging of data into excel*

We need data to train any neural network. Deep learning requires a large amount of data to learn and predict the outcome. To begin, gather as much information as possible. After data collection, analyse it and gains a thorough understanding of it. To choose the best CNN algorithm, you must first gain a better understanding of the data.

We are given two datasets, Object 1 and Object 2, to work with. The data in these files is contained in.csv files, and there is also a file for each calculated time signal, which contains 16,384 scan points. We manually merged all data from Object 1 and Object 2 into an excel file called 'object1.csv' and 'object2.csv' to make the training easier. The given data for training and validation is shown in Fig. 5.

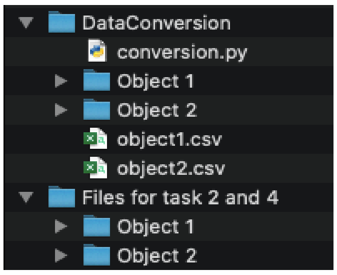
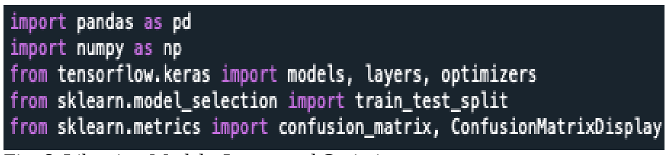


Fig. 5. Training Data

B. *Libraries, Models and Layers*

In The first step is to import all of the libraries needed to build a Conv1D Time Signal classification model. The required Python packages for building the model are shown in the diagram below.



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

Fig. 6. Libraries, Models, Layers and Optimizers

The TensorFlow-Keras library is the most important library imported for this project. When building a deep learning model, the TensorFlow framework comes in handy. Tensor flow accepts data in the form of tensors, which are multi-dimensional arrays. Keras, on the other hand, is a very useful library that contains all of the deep learning resources needed to quickly create and train models. Keras is a great API that runs on top of Tensor Flow and other frameworks.

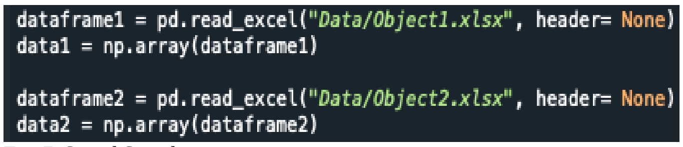
We will use the Sequential Model from the layers input, Conv1D, Dense, MaxPooling, Dropout, and Flatten, as well as the RMSprop optimizer from the optimizer library. For the test train split feature, another library used is sklearn. The values of a confusion matrix are returned by confusion matrix. ConfusionMatrixDisplay is used to display it, with rows representing real values and columns representing expected values.

C. *Training and Testing of Data*

Following the import of the necessary libraries, the data files 'Object1' and 'Object2' must be imported. The data is first translated to data frames (dataframe1, dataframe2), and the data is stored as an Array. The following is a screenshot of the python code for preparing the training and testing results.

Pandas can be used to import an Excel file into Python. Pandas is a robust Python program for performing statistical analysis. The read excel() command was used to achieve this aim.

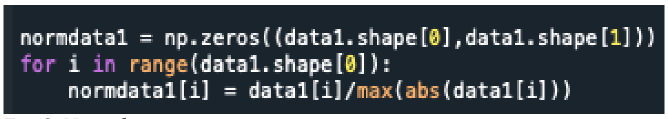
The excel file object1.xlsx is imported into the Python environment from the "data" folder. The first row in an excel data file is referred to as the header. None is used because we don't have a header. After that, the imported file is assigned to a Data frame named dataframe1. To use the Numpy package's high-level mathematical functions, we need to convert a Pandas dataframe to a Numpy Array.



```
dataframe1 = pd.read_excel("Data/Object1.xlsx", header= None)
data1 = np.array(dataframe1)

dataframe2 = pd.read_excel("Data/Object2.xlsx", header= None)
data2 = np.array(dataframe2)
```

Fig. 7. Sound Signals

We convert data to float32 and normalize data values to [0,1] by normalizing the input data, as shown in Fig. 8. The method for normalizing data from object2.xlsx is the same.

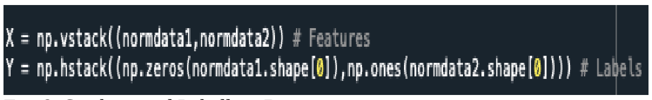


```
normdata1 = np.zeros((data1.shape[0],data1.shape[1]))
for i in range(data1.shape[0]):
    normdata1[i] = data1[i]/max(abs(data1[i]))
```

Fig. 8. Normalizing

D. Data Stacking and Labelling

We will use the vstack command to stack the arrays vertically to construct features for the Convolution Neural Network Model since we'll be processing both sound signals Object1 and Object2 at the same time.



```
X = np.vstack((normdata1,normdata2)) # Features
Y = np.hstack((np.zeros(normdata1.shape[0]),np.ones(normdata2.shape[0]))) # Labels
```

Fig. 9. Stacking and Labelling Data

Second, we must mark our information. As a result, sound signals 'Object1' and 'Object2' are identified as zeros and ones, respectively. The hstack command is then used to stack the labels horizontally.

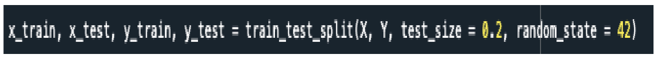
As a result, the model discovers that sound signal 'Object1' is label 0 and sound signal 'Object2' is label 1 during the training process. As a result, it is simple to distinguish between sound signals during the prediction and evaluation stages by using features and labels.

E. Categorizing Data into Test and Train

Divide the stacked data into two datasets: one for training and one for testing. Test data will be used to evaluate the algorithm's prediction and to double-check the learning process' performance.

Sklearn model selection has a feature called **`train_test_split()`** that splits data arrays into two subsets for training and testing. This function is used to eliminate the need to manually divide the dataset.

`train_test_split` creates random partitions for the two subsets by default. However, for the operation, we can define a random state[10].



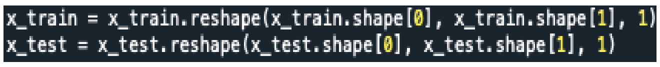
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

Fig. 10. Splitting Data

`X, Y:` is the dataset to be use.
`train_size:` The size of the training dataset is determined by this parameter. None, which is the norm, Int, which includes the exact number of samples, and float, which ranges from 0.1 to 1.0, are the three choices.
`test_size:` The size of the research dataset is defined by this parameter. The default state is appropriate for the scale of the training. If the training size is set to normal, it will be set to 0.2.
`random_state:` The default mode uses **np.random** to perform a random split. You can also add an integer by using an exact number. A random state of 42 is used in this example[10].

F. Reshaping the Data

Changing the shape of an array is referred to as reshaping. The number of elements in each dimension or the addition of a new dimension determines the form of an array. However, the total number of elements will remain the same. In our case, the third dimension is only necessary for the model to function properly. For the computation to take place in the following layers, the dimension of the arrays must be compatible.



```
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)
```

Fig. 11. Reshaping Data

G. Conv1D Model

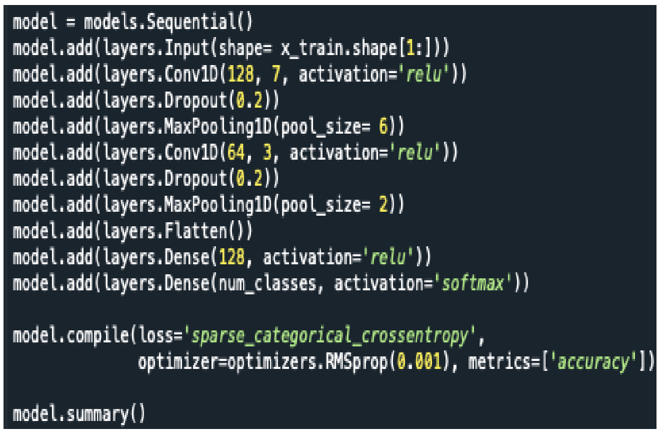
Convolutional Neural Network (CNN) models were primarily designed for image classification, with the model accepting a two-dimensional input representing an image's pixels and color channels in a process known as feature learning.

One-dimensional data sequences can be processed in the same way. The model extracts features from sequence data and maps the sequence's internal features. A 1D CNN is particularly useful for extracting features from a fixed-length segment of a larger dataset, where the location of the feature in the segment is less relevant.

One of the key reasons for using this method is that we have a Time Signal with a fixed duration of data[9].

1) Conv1D Model:
   - 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[9].

The python code for the model is given the Fig. 12. Each of the layers and its argument will be discussed here in detail.



```
model = models.Sequential()
model.add(layers.Input(shape= x_train.shape[1:]))
model.add(layers.Conv1D(128, 7, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 6))
model.add(layers.Conv1D(64, 3, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D(pool_size= 2))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

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

model.summary()
```

Fig. 12. Conv1D Model

`model = models.Sequential(name = "model_conv1D")`
By using this command, we start with a sequential model giving it the name of *"model_conv1D"*.

`model.add(layers.Input(shape= x_train.shape[1:]))`
By using this command input layer is added to the model making use of **`layers`** as per Fig. 6. and data is given to the model through the input layer.

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

By using these commands, the 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[11]. Two times Conv1D layer is used in the Model with a different number of filters and kernel-size.

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

This command allows to add dropout layer to the model

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

This command allows to add MaxPooling layer to the model

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

This command allows to add Flatten() layer to the model

```
output = activation(dot(input, kernel) + bias)
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(n_classes,activation='soft
max'))
```

This command allows to add dense layer to the model

model.compile(loss=’sparse_categorial_crossentropy’ , optimizer=optimizers.RMSprop(0.001), metrics=[‘accuracy’])

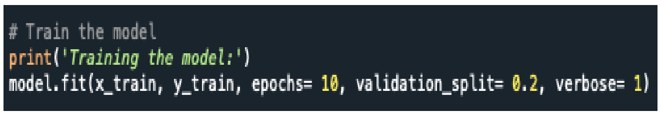
Finally, we must compile the model using a loss function and an optimizer. A loss function is used by machines to understand. It's a way of determining how well a given algorithm models the data. The loss function would provide a very large number if forecasts deviated too much from actual results.

The loss function gradually learns to reduce prediction error with the aid of some optimization function. Sparse categorical cross entropy was chosen for this Model because our classification is binary because we have two signals that are numbered as 0 and 1. RMSprop is the optimizer. Section II has already gone over RMSprop in depth (H).

model.summary() is used to summarize the model. A description is a textual document that describes the layers and their order in the model, as well as the output shape of each layer, the number of parameters in each layer, and the total number of parameters. In the right column, you'll find a rundown of the Conv1D model.

2) Fitting the model

Using the training data collection, the actual learning process will be carried out at this point. Along with training data, the number of **epochs** is often specified. An epoch is a hyperparameter that specifies how many times the learning algorithm can iterate over the entire training dataset. Each parameter can only be updated once per epoch[12]. **Split validation** is a fictitious dataset that has been separated for the purpose of testing. In this scenario, 0.2 indicates that 20% of the training dataset will be used for validation and the remaining 80% for training. **Verbose** mode is a device setting that offers extra information about what program will load during initialization or detailed output for diagnostic purposes.

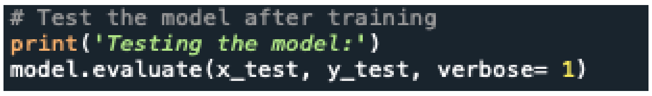


```
# Train the model
print('Training the model:')
model.fit(x_train, y_train, epochs= 10, validation_split= 0.2, verbose= 1)
```

Fig. 13. Model Training

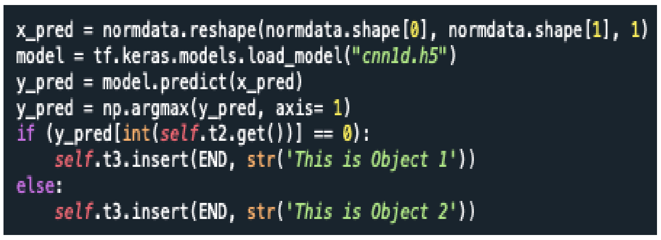
3) Evaluation and Prediction

Examine the model by forecasting the performance for test data and comparing the forecast to the test data's actual outcome. Predict the output for unknown input data (data that isn't already in the training and test set)[13]. The model has been evaluated and predicts the overall model in this section. We estimate the data of the evaluation section, which is isolated from the training data, in the evaluating portion. For the new incoming test data, **model.predict()** is used. Finally, the output signals will be labelled as ‘Object1' and ‘Object2', respectively.



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

Fig. 14. Evaluate the Model



```
x_pred = normdata.reshape(normdata.shape[0], normdata.shape[1], 1)
model = tf.keras.models.load_model("cnn1d.h5")
y_pred = model.predict(x_pred)
y_pred = np.argmax(y_pred, axis= 1)
if (y_pred[int(self.t2.get())] == 0):
    self.t3.insert(END, str('This is Object 1'))
else:
    self.t3.insert(END, str('This is Object 2'))
```

Fig. 15. Predict the Model

4) Confusion Matrix

A confusion matrix is a complex table structure that displays the outcome of an algorithm in the field of machine learning. The True labels are represented by each row of the matrix, while the expected labels are represented by each column. The number of predictions made by the model where it categorized the classes correctly or incorrectly is represented by and entry in a confusion matrix.

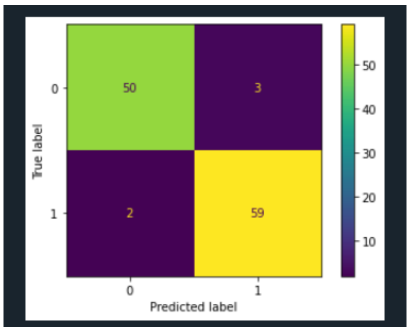
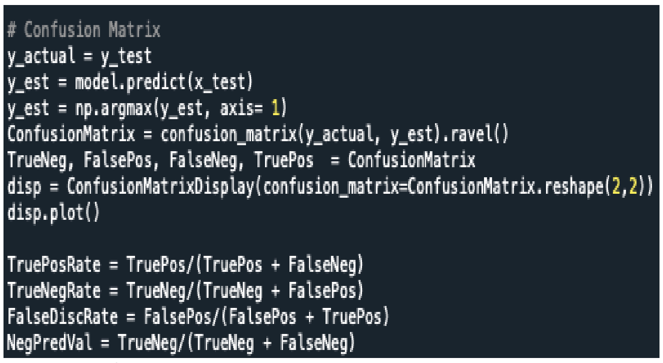


Fig. 16. Confusion Matrix

It It's a unique kind of table with two dimensions ("True" and "predicted"). We mark our data for training with true labels, and then after model training and predicting validation or test data with expected labels, we get labels from the model. So, after the **train_test_split()**, we took samples mixed of sound signals ‘Object1' and ‘Object2' for our project. These samples were already labeled as 0 and 1. So, after the model has been trained and a prediction for those samples has been made, samples.

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 50 True Positive values means that the sample was sound signal **'object2'** which was labeled as 1 and is also predicted as 1. Similarly, 59 True Negative values mean 59 samples of sound signal **'object1'** which were labeled as 0 is predicted as 0. We also have 2 False Positive which were labelled as 1 and predicted as 0 and 3 False Nagative values, labelled as 0 but predicted as 1. This makes it a total of 114 samples. Therefore our model is 96.49% accurate.



```
# Confusion Matrix
y_actual = y_test
y_est = model.predict(x_test)
y_est = np.argmax(y_est, axis= 1)
ConfusionMatrix = confusion_matrix(y_actual, y_est).ravel()
TrueNeg, FalsePos, FalseNeg, TruePos  = ConfusionMatrix
disp = ConfusionMatrixDisplay(confusion_matrix=ConfusionMatrix.reshape(2,2))
disp.plot()

TruePosRate = TruePos/(TruePos + FalseNeg)
TrueNegRate = TrueNeg/(TrueNeg + FalsePos)
FalseDiscRate = FalsePos/(FalsePos + TruePos)
NegPredVal = TrueNeg/(TrueNeg + FalseNeg)
```

Fig. 17. Confusion Matrix Code

"Prediction Class" and "Actual Class" are the two sections. There are subsections of each class. We can now calculate the following metrics to test our model accuracy using this visualized uncertainty matrix:

True Positive (TruePos): The number of true positive predictions is the number of times the object type is accurately predicted to be the actual class.

True Negative (TrueNeg): The number of true negative predictions is the number of times the object is genuinely predicted to be outside of the given class.

False Positive (FalsePos): The number of false positive predictions is the number of times the classifier predicts the target object even though it is not the actual object.

False Negative (FalseNeg): The number of false negative predictions is the number of times the classifier makes a false prediction about the object that is actually being targeted.

True Positive Rate (TruePosRate): For a given object class, the True Positive Rate is defined as the total number of true positives divided by the total sum of true positives and false negatives.

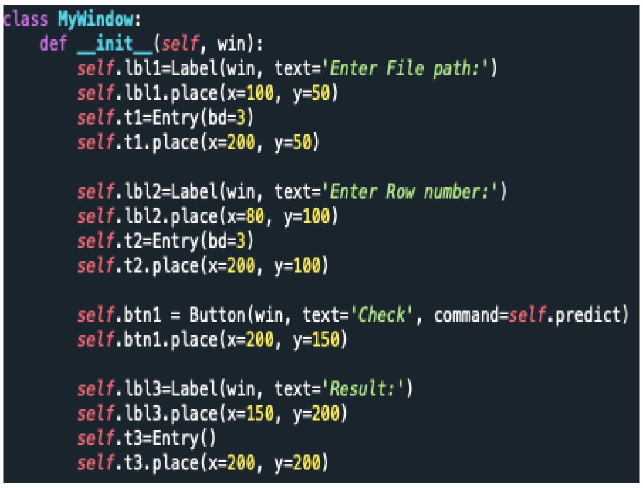
True Negative Rate (TrueNegRate): For a given object type, the True Negative Rate is the ratio of true negatives to the number of true negatives and false positives.

False Discovery Rate (FalseDiscRate): For a given object class, the False Discovery Rate is the ratio of false positives to the number of false positives and true positives.

Negative Predictive Value (NegPredVal): For a given object class, Negative Predictive Value is the ratio of true negative to the sum of true negative and false negative.

5) GUI

The below code represents box measurements that we have used to design our GUI result window. Once the code in Fig. 18 is run we get output window as shown in Fig. 20.

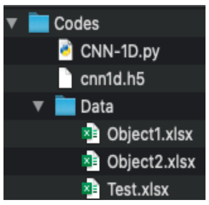


```
class MyWindow:
    def __init__(self, win):
        self.lbl1=Label(win, text='Enter File path:')
        self.lbl1.place(x=100, y=50)
        self.t1=Entry(bd=3)
        self.t1.place(x=200, y=50)

        self.lbl2=Label(win, text='Enter Row number:')
        self.lbl2.place(x=80, y=100)
        self.t2=Entry(bd=3)
        self.t2.place(x=200, y=100)

        self.btn1 = Button(win, text='Check', command=self.predict)
        self.btn1.place(x=200, y=150)

        self.lbl3=Label(win, text='Result:')
        self.lbl3.place(x=150, y=200)
        self.t3=Entry()
        self.t3.place(x=200, y=200)
```

Fig. 18. GUI Box Measurements

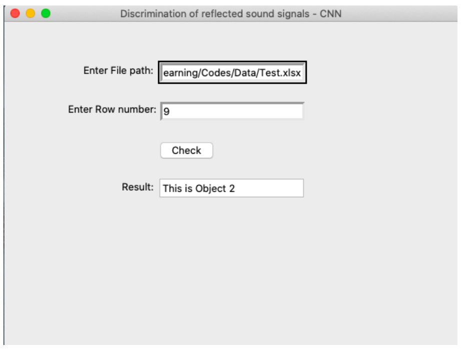
VIII. Results

The results show that when the model first started training in the first 1 to 2 epochs, it was still learning, so the accuracy was low, but in the tenth epoch, it achieved an accuracy of 93.48 percent. It also shows us an accuracy of 96.49 percent during the testing process, which is a pretty good result. Figure 21 shows the number of layers and parameters used in the model, as well as the number of epochs, loss, and accuracy at each epoch. Two files were given for testing the algorithm. Each file contains 50 or more data, and each test data has 16384 scan points. Data is randomly stored from 'Object1' and 'Object2' into the Test.xlsx file. The result shows whether data is from 'Object1' or 'Object2' based on the row number and local file path of Test.xlsx data sheet.



```
▼ Codes
    CNN-1D.py
    cnn1d.h5
  ▼ Data
      Object1.xlsx
      Object2.xlsx
      Test.xlsx
```

Fig. 19. Test data



Discrimination of reflected sound signals - CNN

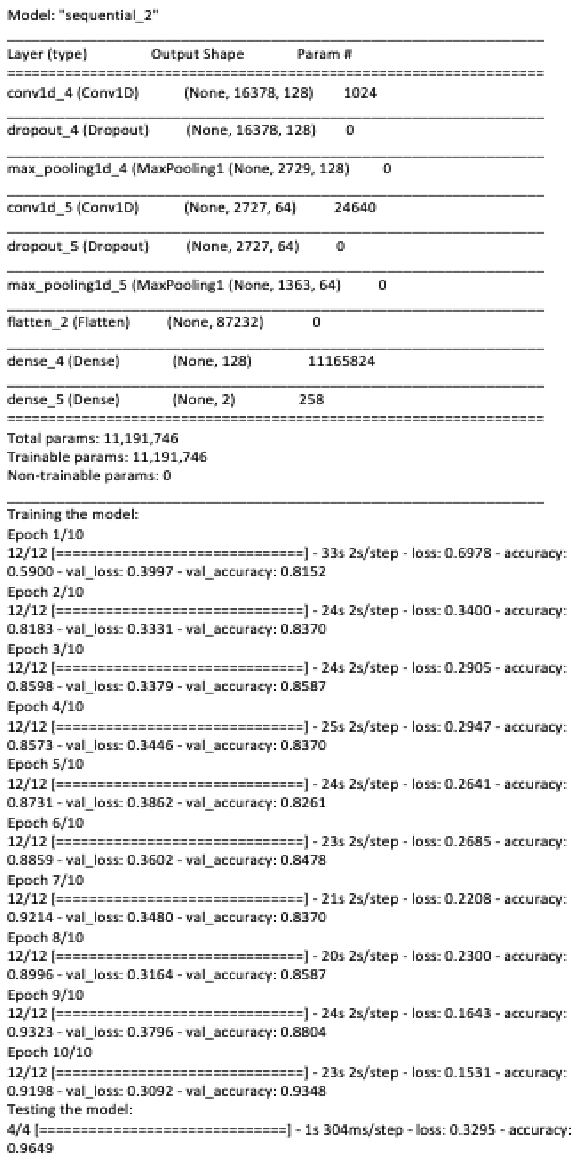
Enter File path: earning/Codes/Data/Test.xlsx

Enter Row number: 9

Check

Result: This is Object 2

Fig. 20. GUI



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)            Output Shape           Param #
=================================================================
conv1d_4 (Conv1D)        (None, 16378, 128)      1024
_________________________________________________________________
dropout_4 (Dropout)       (None, 16378, 128)      0
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2729, 128)       0
_________________________________________________________________
conv1d_5 (Conv1D)        (None, 2727, 64)        24640
_________________________________________________________________
dropout_5 (Dropout)       (None, 2727, 64)        0
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 1363, 64)        0
_________________________________________________________________
flatten_2 (Flatten)       (None, 87232)          0
_________________________________________________________________
dense_4 (Dense)          (None, 128)           11165824
_________________________________________________________________
dense_5 (Dense)          (None, 2)            258
=================================================================
Total params: 11,191,746
Trainable params: 11,191,746
Non-trainable params: 0
_________________________________________________________________
Training the model:
Epoch 1/10
12/12 [==============================] - 33s 2s/step - loss: 0.6978 - accuracy:
0.5900 - val_loss: 0.3997 - val_accuracy: 0.8152
Epoch 2/10
12/12 [==============================] - 24s 2s/step - loss: 0.3400 - accuracy:
0.8183 - val_loss: 0.3331 - val_accuracy: 0.8370
Epoch 3/10
12/12 [==============================] - 24s 2s/step - loss: 0.2905 - accuracy:
0.8598 - val_loss: 0.3379 - val_accuracy: 0.8587
Epoch 4/10
12/12 [==============================] - 25s 2s/step - loss: 0.2947 - accuracy:
0.8573 - val_loss: 0.3446 - val_accuracy: 0.8370
Epoch 5/10
12/12 [==============================] - 24s 2s/step - loss: 0.2641 - accuracy:
0.8731 - val_loss: 0.3862 - val_accuracy: 0.8261
Epoch 6/10
12/12 [==============================] - 23s 2s/step - loss: 0.2685 - accuracy:
0.8859 - val_loss: 0.3602 - val_accuracy: 0.8478
Epoch 7/10
12/12 [==============================] - 21s 2s/step - loss: 0.2208 - accuracy:
0.9214 - val_loss: 0.3480 - val_accuracy: 0.8370
Epoch 8/10
12/12 [==============================] - 20s 2s/step - loss: 0.2300 - accuracy:
0.8996 - val_loss: 0.3164 - val_accuracy: 0.8587
Epoch 9/10
12/12 [==============================] - 24s 2s/step - loss: 0.1643 - accuracy:
0.9323 - val_loss: 0.3796 - val_accuracy: 0.8804
Epoch 10/10
12/12 [==============================] - 23s 2s/step - loss: 0.1531 - accuracy:
0.9198 - val_loss: 0.3092 - val_accuracy: 0.9348
Testing the model:
4/4 [==============================] - 1s 304ms/step - loss: 0.3295 - accuracy:
0.9649
```

Fig. 21. Results

IX. Conclusion

We have tried to clarify how 1D convolutional neural networks can be used to classify time series in this article. It's worth noting that the approach suggested is not the only one available. There are ways to view time series in the form of images using their spectrograms, which can be done with a standard 2D convolution..

CNNs have made significant strides in image processing and related fields in recent years. Due to its powerful and formidable learning capacity, image detection has become a major research focus. Major advancements have been identified by numerous researchers around the world in the fields of image marking, object recognition, image categorization, and other areas.
This makes it possible to develop approaches to object detection and image classification. However, CNNs can only outperform humans at image recognition tasks if they are given a well-prepared data set. However, they are also insufficient devices for detecting stimuli such as noise and glare, which are readily detectable by humans.

Contributions

TABLE I. Team Contribution

Team Members	Contributions
Abhishek Makam	• Designed the Python code to distinguish between two Object classes and combine them into excel data. • Developed and performed conversion and implementation of datasets into normalized form thereby reshaping and training the model by assigning number of epochs. • Researched, documented and proofread about the CNN model, algorithm and methodologies.
Komala Balakrishna	• Designed the Python code to distinguish between two Object classes and combine them into excel data. • Designed and performed testing of the Conv1D model by adding different filters, kernel size and optimizing the data. • Studied, documented and proofread about CNN usage, hyperparameters and methodologies.
Sahana Prasad	• Wrote a python code to predict the normalized model for an unknown output and implement confusion matrix followed by creation of GUI window. • Designed and performed testing of the Conv1D model by adding different filters, kernel size and optimizing the data. • Investigated, proofread and documented the CNN applications, methodologies and final result.

Acknowledgment

We would like to thank Prof. Dr. Andreas Pech for giving us an opportunity to closely work with him on this topic. We would also like to mention him for all the guidance and support throughout the work.

References

[1] G Liu, T., Fang, S., Zhao, Y., Wang, P., & Zhang, J. (2015). Implementation of training convolutional neural networks. arXiv preprint arXiv:1506.01195.

[2] Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA, USA:: Determination press.

[3] Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. Procedia computer science, 132, 377-384.K.

[4] Muhamad Yani et al 2019 J. Phys.: Conf. Ser. 1201 012052

[5] Khan, A., Sohail, A., Zahoora, , A. S. (2019). A survey of the recent architectures of deep convolutional neural networks. arXiv preprint arXiv:1901.06032.

[6] Hijazi, S., Kumar, R., & Rowen, C. (2015). Using convolutional neural networks for image recognition. Cadence Design Systems Inc.: San Jose, CA, USA.

[7] Aszemi, N. M., & Dominic, P. D. D. Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms.

[8] Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning.arXiv preprint arXiv:1811.03378

[9] Liao, C., Wang, J., Xie, Q., Baz, A. A., Huang, X., Shang, J., & He, Y. (2020). Synergistic Use of Multi-Temporal RADARSAT-2 and VENµS Data for Crop Classification Based on 1D Convolutional Neural Network. Remote Sensing, 12(5), 832.

[10] Agarwal, A., & Saxena, A. (2018). Malignant tumor detection using machine learning through scikit-learn. International Journal of Pure and Applied Mathematics, 119(15), 2863-2874.

[11] Kumar, B. A. Training data sets for TensorFlow models from TeleEcho data.

[12] Brownlee, J. (2018). What is the Difference Between a Batch and an Epoch in a Neural Network?. Deep Learning; Machine Learning Mastery: Vermont, VIC, Australia.

[13] Chollet, F. (2016). Building autoencoders in keras. The Keras Blog, 14.