1. Introduction

Scope of the Project

Sound classification is a growing area of research that everyone is trying to learn and implement on some kinds of projects. The problem statement is applying deep learning techniques to classify environmental sounds, specifically identifying urban sounds.

Given an audio sample of some category with a specific duration in .wav extension, determine whether it contains target urban sounds. It lies under the supervised machine learning category, so we have a dataset and a target category.

Problem Statement

The primary challenge is to build an accurate model that, when given an audio file as input, should determine whether the audio features correctly contain one of the target labels.

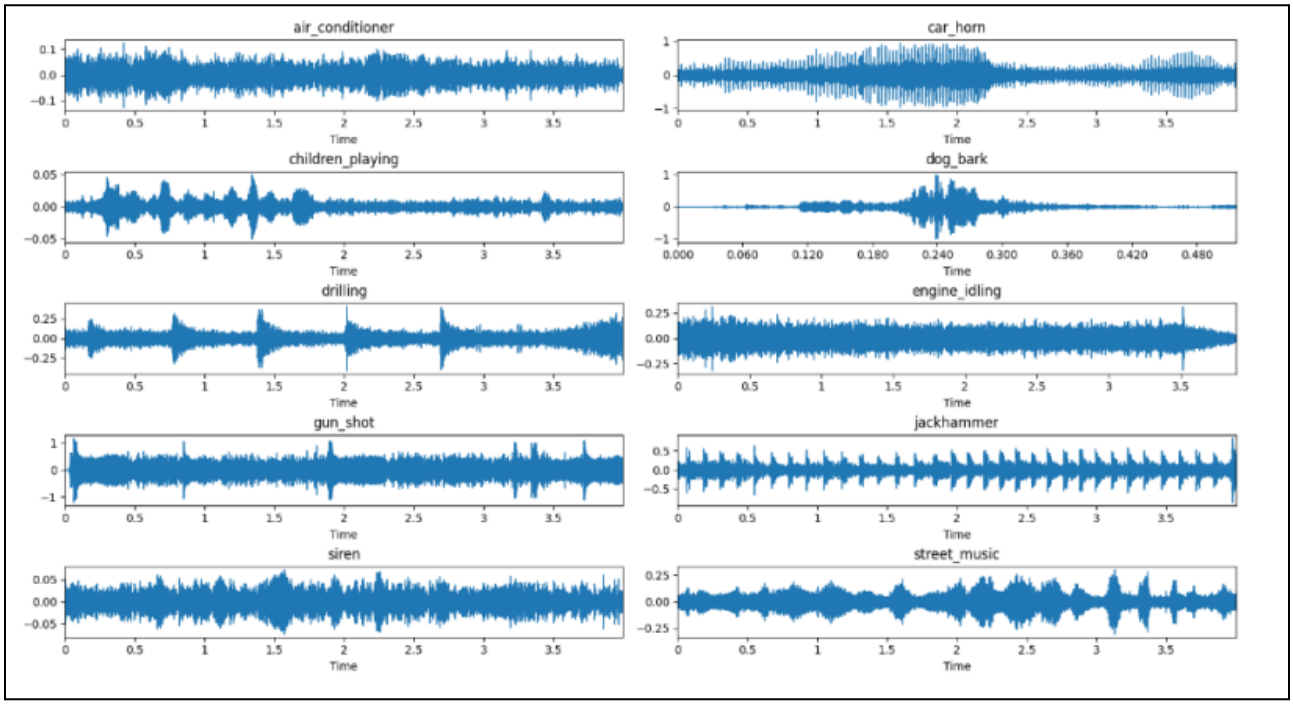
Dataset Overview

The dataset used in this project is called the Urban Sound 8k dataset. The dataset contains 8732 audio files(in .wav format) of 10 different classes and is listed below. Each audio file is anywhere between 3.75 seconds to 4 seconds long. When downloading the data, the data is pre-sorted into ten folds (labelled fold1, fold2, …, fold10) to support a 10-fold cross-validation. Our task is to extract different features from these files and classify the corresponding audio files into categories.

1. Air Conditioner
2. Car Horn
3. Children Playing
4. Dog Bark
5. Drilling Machine
6. Engine Idling
7. Gun Shot
8. Jackhammer
9. Siren
10. Street Music

2. Exploratory Data Analysis

Data scientists use exploratory data analysis (EDA) to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers we need, making it easier to discover patterns, spot anomalies, test a hypothesis, or check assumptions.



We must discover whether our dataset is balanced or imbalanced in the EDA part.
Balanced Dataset: A balanced dataset in the context of machine learning refers to a dataset where the number of instances or examples of each class is roughly equal.
Imbalanced Dataset: A dataset where the distribution of instances across different classes is unequal, meaning that some classes have significantly more or fewer examples than others.

Balanced datasets are essential in audio classification for several reasons:

1. Preventing Bias: When dealing with imbalanced datasets, models are more likely to be biased towards the majority class. In audio classification, if one class significantly outnumbers the others, the model may become overly focused on that class, potentially leading to poor performance in the minority classes.

2. Improving Model Generalization: Models trained on balanced datasets are more likely to generalize well to unseen data. Suppose a model is trained on an imbalanced dataset. In

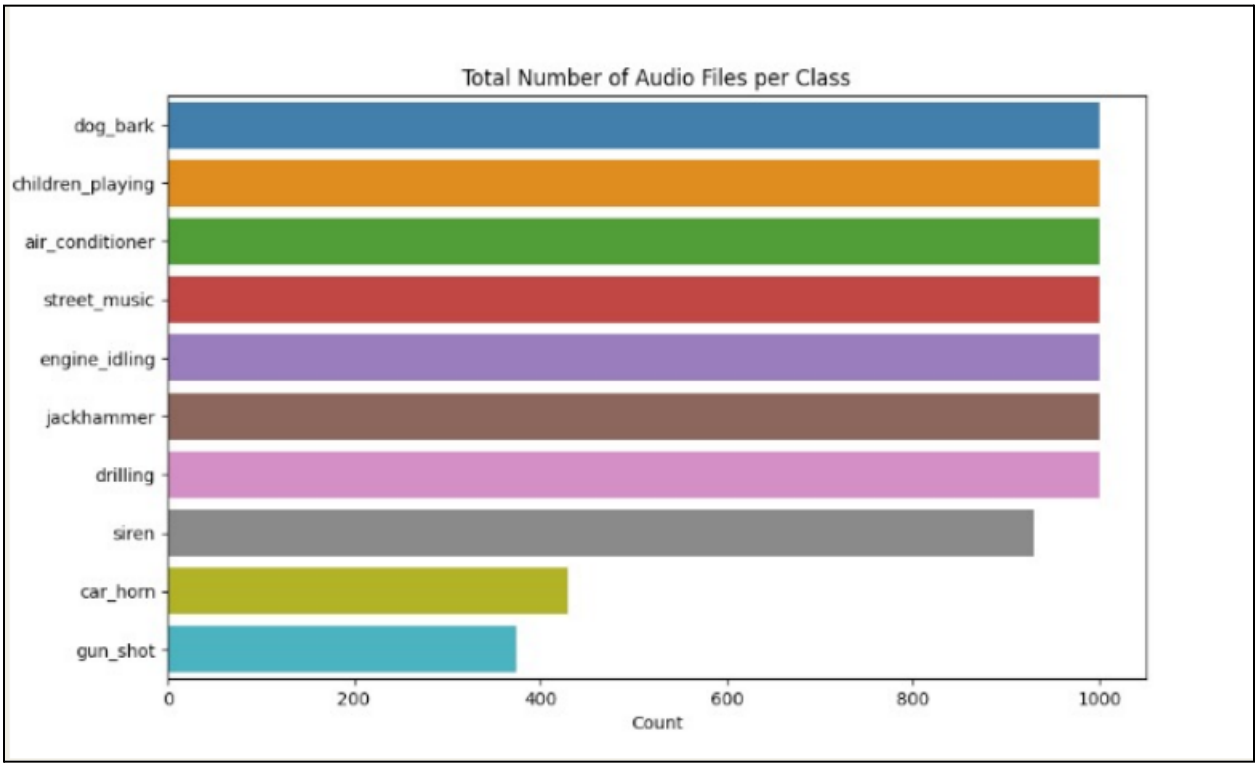
that case, it might not learn the underlying patterns of the minority classes effectively, leading to suboptimal performance when faced with new, real-world examples.

3. Avoiding Misleading Accuracy Metrics: Accuracy, a commonly used metric, can be misleading in imbalanced datasets. A model that predicts the majority class for all instances could achieve a high accuracy simply by exploiting the class imbalance. However, such a model would be useless for the minority class, which might be the more critical class in some applications.

4. Addressing Real-World Distribution: In many real-world scenarios, class distribution is unequal. However, creating a balanced dataset can help the model account for the proper distribution of classes, making it more robust and applicable in practical situations.

```
dog_bark            1000
children_playing    1000
air_conditioner     1000
street_music        1000
engine_idling       1000
jackhammer          1000
drilling            1000
siren                929
car_horn             429
gun_shot             374
Name: class, dtype: int64
```

	slice_file_name	fsID	start	end	salience	fold	classID	class	num_channels	sampling_rate
0	177742-0-0-99.wav	177742	49.500000	53.500000	2	3	0	air_conditioner	2	48000
1	24074-1-0-10.wav	24074	18.060993	22.060993	1	1	1	car_horn	2	44100
2	60591-2-0-4.wav	60591	2.000000	6.000000	1	2	2	children_playing	2	44100
3	174026-3-1-5.wav	174026	11.719766	15.719766	2	4	3	dog_bark	2	48000
4	59594-4-0-3.wav	59594	1.500000	5.500000	2	2	4	drilling	2	44100
5	154758-5-0-7.wav	154758	3.500000	7.500000	1	4	5	engine_idling	2	48000
6	148841-6-2-0.wav	148841	9.132153	10.787741	1	5	6	gun_shot	2	44100
7	162134-7-7-0.wav	162134	129.628486	133.628486	1	10	7	jackhammer	2	96000
8	118279-8-0-13.wav	118279	6.500000	10.500000	2	1	8	siren	2	48000
9	89443-9-0-16.wav	89443	8.000000	12.000000	1	7	9	street_music	2	44100

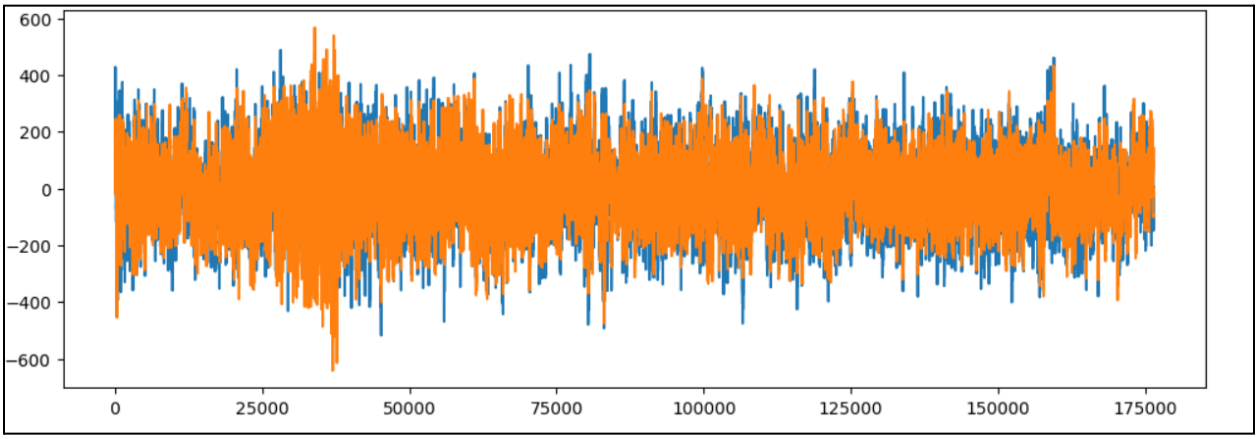


3. Data Preprocessing

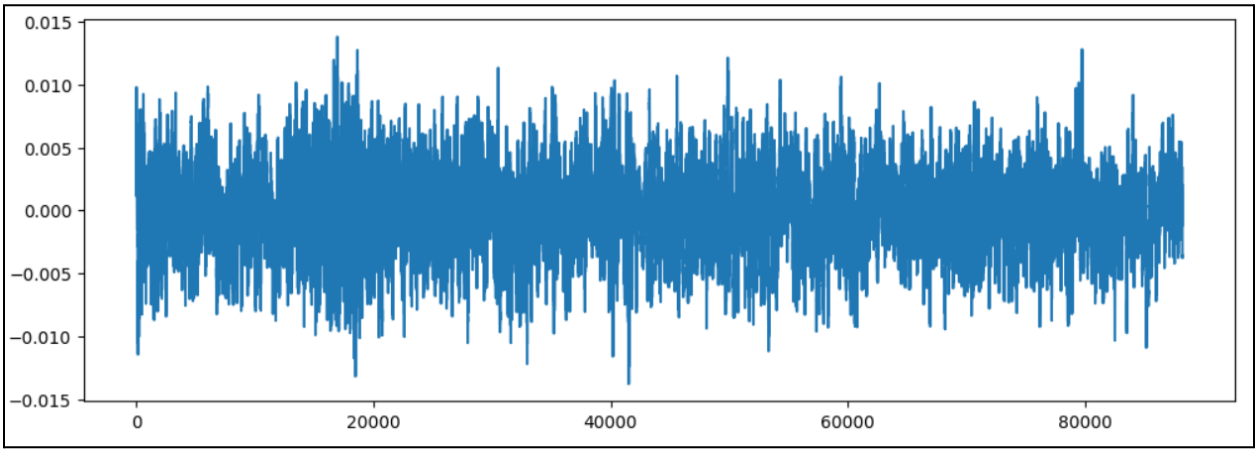
Data preprocessing is crucial in the data analysis and machine learning pipeline. It involves a series of operations and transformations applied to raw data before it is fed into a machine-learning model. The goal of data preprocessing is to clean, format, and organize the data to enhance the performance and accuracy of the model. The specific tasks involved in data preprocessing can vary depending on the nature of the data and the requirements of the machine learning algorithm.

Most audio files have two channels, meaning they are stereophonic sounds. A stereo audio sends two signals using two different channels, generating one signal for each speaker. Stereo audios are used to create directionality and perspective to sound. Furthermore, there are some audio files with one channel, which is a monophonic sound. A mono audio means that only a single audio signal is sent to all speakers. Unlike stereo audio, mono will produce the same signal through all its speakers, and there won't be any difference between them. Therefore, we must convert all the 2-channel files to a

1-channel file to make each audio input have the exact dimensions. This was done simply by using the Librosa library.



2-Channel audio file



1-Channel audio file

Normalization of data refers to scaling and transforming numerical features of the dataset to a standard range. The goal is to ensure all features have a similar scale and distribution, preventing some features from dominating others.

In the case of audio classification ML projects, normalization can be crucial for several reasons:

1. Magnitude Consistency: Different audio features may have different units or magnitudes. For instance, the amplitude of audio signals may be on a different scale than the

frequency content. Normalization ensures that all features contribute equally to the model by bringing them to a consistent scale.

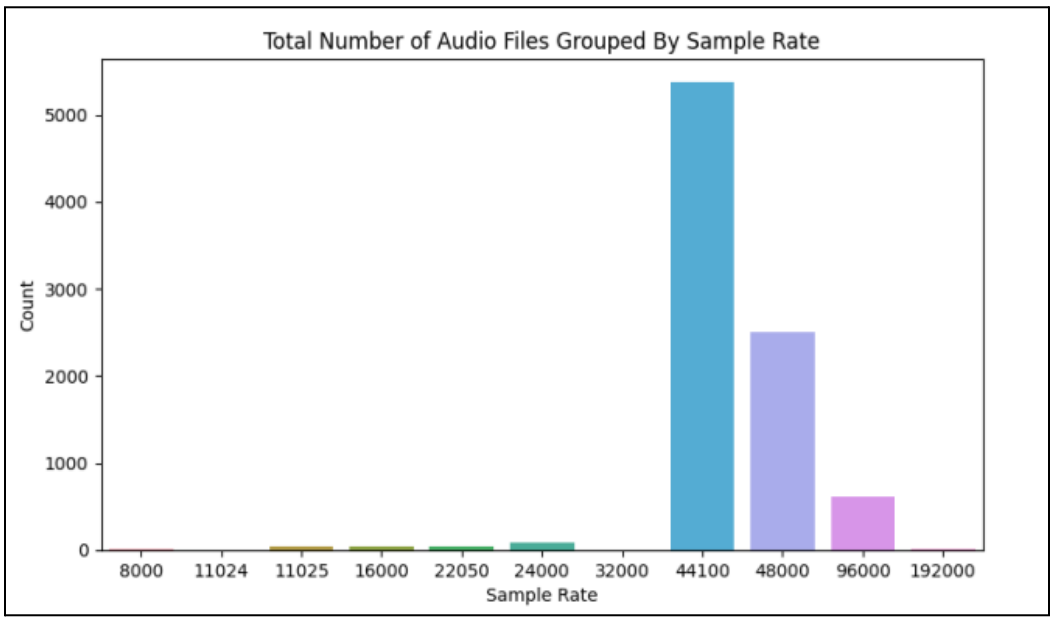
2. Improved Convergence: Many machine learning algorithms, especially those based on gradient descent, converge faster when the input features are on a similar scale. Normalizing the data helps in achieving faster convergence during model training.

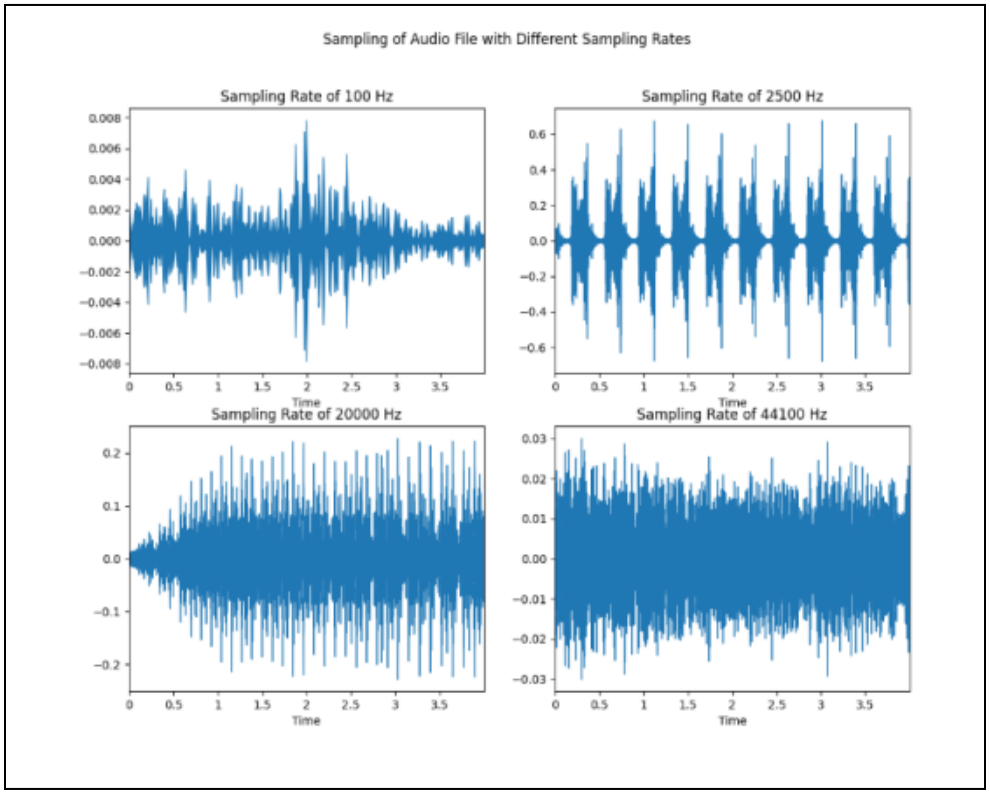
3. Preventing Numerical Instabilities: In some cases, large numerical values in the input features can lead to numerical instability issues, especially in algorithms that involve matrix operations or exponentials. Normalization can mitigate these issues.

4. Enhanced Model Performance: Normalization can improve the performance of specific algorithms, such as k-nearest neighbours or support vector machines, which are sensitive to the scale of input features.

5. Regularization Effect: Normalization can act as a form of regularization, helping to prevent overfitting by keeping the model's weights within a reasonable range.

Standardizing the sampling rate of each audio file. Most audio files are sampled at 44.1 kHz (44,100 Hz). This means that for every 1 second of audio, the array will have a size of 44,100 values (4 seconds of audio will have a size of 176,400). We will convert all the audio files to a sampling rate of 22.05 kHz using the Librosa library.





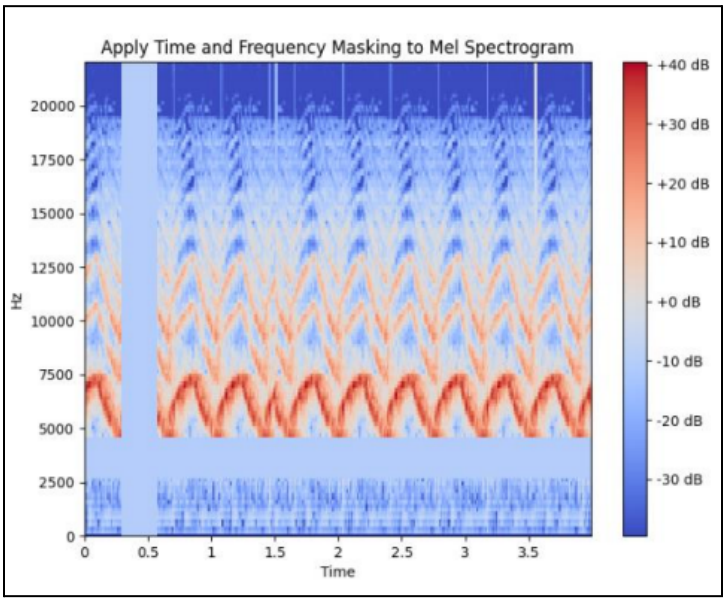
Feature Extraction:

Feature extraction transforms raw data into features more suitable for analysis, modelling, and machine learning tasks. In audio classification, feature extraction involves extracting relevant information or characteristics from the raw audio data to represent it more manageable and informatively. The goal is to capture essential patterns, structures, or properties of the data relevant to the task.

Audio Feature Extraction Methods:

1. Mel-frequency Cepstral Coefficients (MFCCs) (used in this project): MFCCs are coefficients that represent the short-term power spectrum of a sound signal. They are derived from the mel-frequency scale, which approximates the human ear's response to different frequencies. The MFCCs capture the spectral characteristics of the audio signal and are often used as features in audio analysis tasks.

2. Spectrogram: Represents the spectrum of signal frequencies as they vary with time. It helps capture temporal and frequency information.
   This spectrogram uses a Mel Scale instead of Frequency to help generalise the frequency into smaller batches. Furthermore, a Mel Spectrogram uses a Decibel scale instead of Amplitude, which provides more helpful information to a deep learning model. Converting the audio files to a Mel 10 Spectrogram now gives us an input size of several channels by several Mel's by decibels (2 x 64 x 344), which is much better input into a CNN.
   The masked sections are replaced with the mean value of each Mel Spectrogram. This aims to prevent overfitting and help the model generalise more.



3. Chromogram: This represents the energy distribution of pitch classes (notes) in an audio signal.

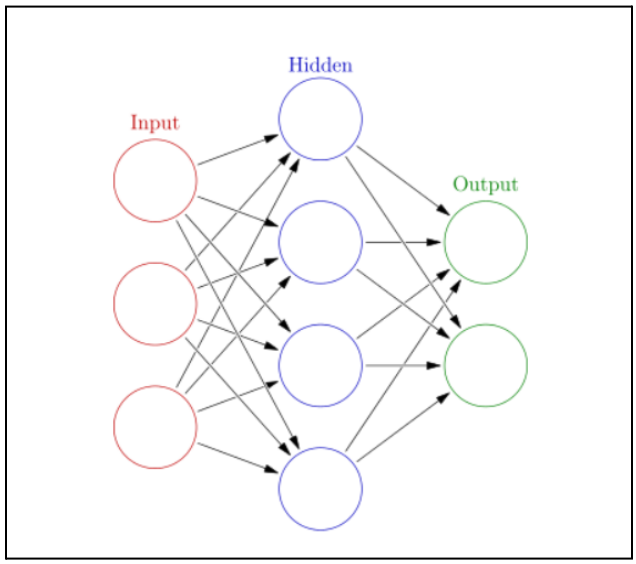
4. Deep Learning Model Creation

Implement an ANN model using Keras sequential API. The number of classes is 10, which is the output shape(number of classes) and creates ANN with 3 hidden dense layers and one output layer.

ANN Working:

The working of ANN takes its roots from the neural network residing in the human brain. ANN operates on something referred to as Hidden State. These hidden states are similar to

neurons. Each of these hidden states is a transient form with probabilistic behaviour. A grid of such a hidden state acts as a bridge between the input and the output.
We have an input layer, the data we provide to the ANN. We have the hidden layers, which is where the magic happens. Lastly, we have the output layer, where the finished network computations are placed for us.



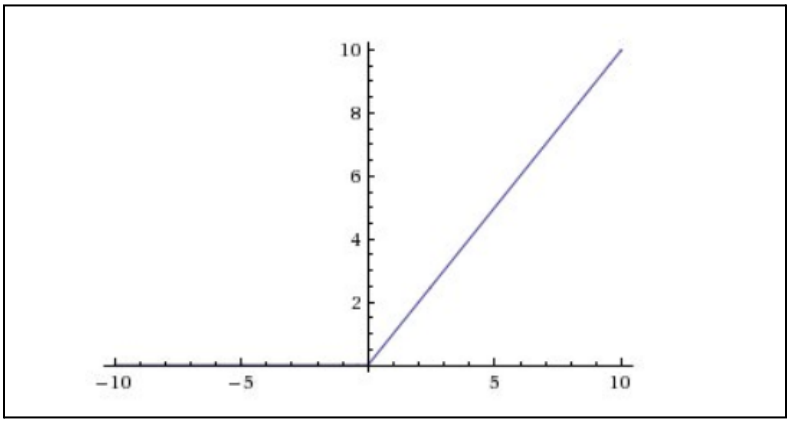
1. Assign Random weights to all the linkages to start the algorithm
2. Using the inputs and the (Input ->Hidden node) linkages find the activation rate of Hidden Nodes
3. Using the activation rate of Hidden nodes and linkages to Output, find the activation rate of Output Nodes
4. Find the error rate at the output node and recalibrate all the linkages between Hidden Nodes and Output Nodes
5. Using the Weights and error found at Output node, cascade down the error to Hidden Nodes
6. Recalibrate the weights between hidden node and the input nodes
7. Repeat the process till the convergence criterion is met
8. Using the final linkage weights score the activation rate of the output nodes

Activation Functions:

Activation functions are essential for an artificial neural network to learn and make sense of complicated and complex non-linear functional mappings between the inputs and response variables. They introduce non-linear properties to our Network. Their primary purpose is to convert an input signal of a node in an ANN to an output signal. That output signal is now used as input in the next layer in the stack.

Activation Function used in this project:

1. ReLU Activation Function: (Rectified Linear Units) The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x, it returns that value. So it can be written as $f(x)=max(0,x)$

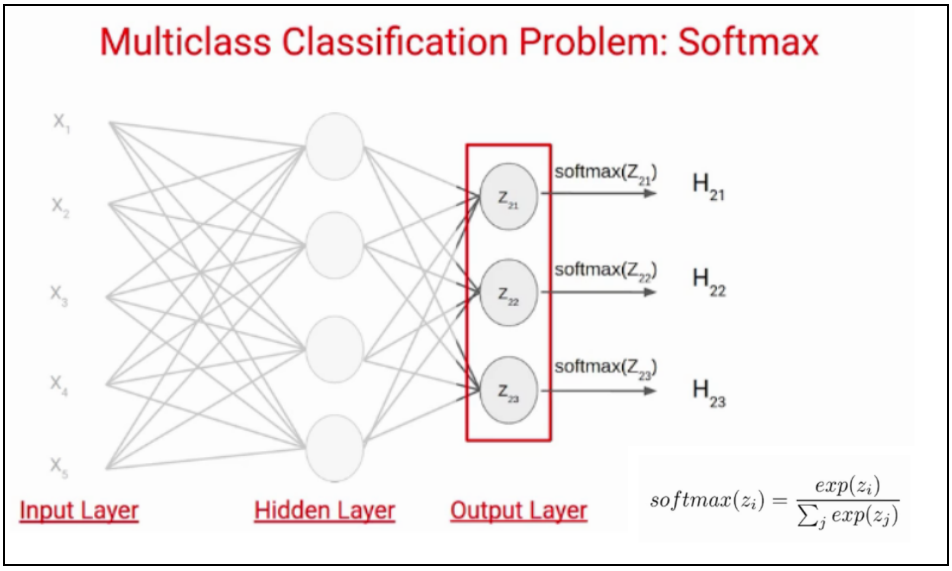


ReLU Activation Function Graph

However, its limitation is that it should only be used within Hidden layers of a Neural Network Model.

Hence, for output layers, we should use a Softmax function for a Classification problem to compute the probabilities for the classes. A regression problem should simply use a linear function.

2. Softmax Activation Function: Softmax is used for multilabel classification. Softmax is used at the output layer of the neural network. It takes an N-dimensional vector of real numbers, converts it into an N-dimensional vector of real numbers between 0 and 1, and sums it up to 1. Softmax outputs a probability distribution, making it suitable for probabilistic interpretation in classification tasks.



Model Parameters:

An Epoch is one complete pass through the entire training dataset; for this project, the number of Epochs is 100.

Batch size is the number of training examples in one forward/backward pass. The higher the batch size, the more memory space is needed. Batch Size for this project is 32.

Optimizer: Adam
Loss Function: CrossEntropyLoss
Loss: 0.7819
Accuracy:0.808242678642273
Validation Loss: 0.6143
Validation Accuracy: 0.8117
Total Training Time: 0:01:34.022634

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               4100

 activation (Activation)     (None, 100)               0

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 200)               20200

 activation_1 (Activation)   (None, 200)               0

 dropout_1 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 100)               20100

 activation_2 (Activation)   (None, 100)               0

 dropout_2 (Dropout)         (None, 100)               0

 dense_3 (Dense)             (None, 10)                1010

 activation_3 (Activation)   (None, 10)                0

=================================================================
Total params: 45410 (177.38 KB)
Trainable params: 45410 (177.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

5. Model Testing

Testing some Audio Data

Steps:

1. Preprocess the new audio data
2. Predict the classes using the model
3. Inverse transform your Predicted Label to get the class name

```
filename="/content/drive/MyDrive/UrbanSound8K/UrbanSound8K/dog_bark_1.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)

class_probabilities = model.predict(mfccs_scaled_features)
predicted_class_index = np.argmax(class_probabilities, axis=1)[0]
predicted_class = labelencoder.inverse_transform([predicted_class_index])

# Print the predicted class
print(f"Predicted class: {predicted_class[0]}")
```

```
[-3.0586047e+02  1.2752591e+02 -1.5775964e+01 -2.5376625e+01
 -3.0284182e+01  5.6979527e+00 -1.9110247e+01 -1.1571763e+01
 -1.0826708e+01 -2.6324925e+00 -6.1536469e+00 -5.5842834e+00
 -3.1616607e+00 -4.7196853e-01 -2.7999718e+00  7.7700839e+00
  1.1307980e+01  5.8785081e+00  1.2114114e+00 -5.4405947e+00
 -6.3454385e+00 -4.2748408e+00 -5.6409841e+00 -7.5247226e+00
 -6.9808292e+00 -5.2572632e+00 -6.2929072e+00 -5.1851692e+00
 -1.9119037e+00 -8.5910934e-01  1.4149481e+00  3.2930329e+00
  5.4473906e+00  2.7657357e-01 -5.8353931e-01 -4.7659393e-02
 -8.2374918e-01 -2.3783512e+00 -1.2069672e+00 -2.4756317e+00]
[[-3.0586047e+02  1.2752591e+02 -1.5775964e+01 -2.5376625e+01
  -3.0284182e+01  5.6979527e+00 -1.9110247e+01 -1.1571763e+01
  -1.0826708e+01 -2.6324925e+00 -6.1536469e+00 -5.5842834e+00
  -3.1616607e+00 -4.7196853e-01 -2.7999718e+00  7.7700839e+00
   1.1307980e+01  5.8785081e+00  1.2114114e+00 -5.4405947e+00
  -6.3454385e+00 -4.2748408e+00 -5.6409841e+00 -7.5247226e+00
  -6.9808292e+00 -5.2572632e+00 -6.2929072e+00 -5.1851692e+00
  -1.9119037e+00 -8.5910934e-01  1.4149481e+00  3.2930329e+00
   5.4473906e+00  2.7657357e-01 -5.8353931e-01 -4.7659393e-02
  -8.2374918e-01 -2.3783512e+00 -1.2069672e+00 -2.4756317e+00]]
(1, 40)
1/1 [==============================] - 0s 33ms/step
Predicted class: dog_bark
```

```
filename="/content/drive/MyDrive/UrbanSound8K/UrbanSound8K/jack_hammer.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)

class_probabilities = model.predict(mfccs_scaled_features)
predicted_class_index = np.argmax(class_probabilities, axis=1)[0]
predicted_class = labelencoder.inverse_transform([predicted_class_index])

# Print the predicted class
print(f"Predicted class: {predicted_class[0]}")
```

```
[-1.8683372e+02  1.0108185e+02 -4.1835213e+00  2.5747143e+01
 -1.2561851e+01  3.0396769e+01 -1.4910374e+01  3.0646818e+01
 -1.9199541e+01  2.9225269e+01 -1.4794398e+01  1.6194155e+01
 -4.5088754e+00  1.0370352e+01 -1.0605338e+00  1.5773680e+00
  2.5857542e+00  3.2918239e+00  7.9766738e-01  8.8248386e+00
 -6.0337253e+00  1.2098957e+01 -9.2808962e+00  9.5526724e+00
 -8.6434202e+00  5.1417751e+00 -5.5730472e+00  2.1250973e+00
  6.8320388e-01  1.5112017e+00  2.6297493e+00 -1.4028767e-01
  3.4744461e+00 -4.3258962e-01  2.3588732e-01  1.0565585e+00
 -7.0410448e-01  2.6812928e+00 -2.4252865e+00  4.7444406e+00]
[[-1.8683372e+02  1.0108185e+02 -4.1835213e+00  2.5747143e+01
  -1.2561851e+01  3.0396769e+01 -1.4910374e+01  3.0646818e+01
  -1.9199541e+01  2.9225269e+01 -1.4794398e+01  1.6194155e+01
  -4.5088754e+00  1.0370352e+01 -1.0605338e+00  1.5773680e+00
   2.5857542e+00  3.2918239e+00  7.9766738e-01  8.8248386e+00
  -6.0337253e+00  1.2098957e+01 -9.2808962e+00  9.5526724e+00
  -8.6434202e+00  5.1417751e+00 -5.5730472e+00  2.1250973e+00
   6.8320388e-01  1.5112017e+00  2.6297493e+00 -1.4028767e-01
   3.4744461e+00 -4.3258962e-01  2.3588732e-01  1.0565585e+00
  -7.0410448e-01  2.6812928e+00 -2.4252865e+00  4.7444406e+00]]
(1, 40)
1/1 [==============================] - 0s 31ms/step
Predicted class: jackhammer
```

References:

DatasetLink: https://urbansounddataset.weebly.com/download-urbansound8k.html
https://paperswithcode.com/dataset/urbansound8k-1
https://urbansounddataset.weebly.com/urbansound8k.html
https://jovian.ai/charmzshab/urban-sound-dataset
https://www.kaggle.com/code/shrutimechlearn/deep-tutorial-1-ann-and-classification
https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning

Signatures