Math Concept Deep Learning-1

Sound Classification

Group Members

Frederick Ditliac Atiah

Belle Pensamiento

Helia Rahimi

Rishabh Sethi

Saba Tayebifar

Introduction

Noise cancellation is an important concept in the sound and audio industry (Baghdasaryan, 2018). Most hardware devices like earpieces, headphones, and mobile phones have incorporated technology to block outside noise from interfering with the good experience of listening to music and calls (Baghdasaryan, 2018) (Ranj & Ranj's, 2022). Moreover, the advancement of technologies like artificial intelligence (AI) and data availability has given rise to the software application of noise cancellation with machine learning []. Examples of such applications include zoom etc.

For a machine learning model to be able to cancel noise for a good user experience, the model must be able to identify different noises, which is the basic building block for a noise cancellation application.

Problem Statement

The scope of this project is not to develop a full-fleshed noise cancellation application or model but to build a sound classification model based on an urban sound dataset which will later contribute to a bigger noise cancellation project.

This project can be applied in areas where a user is in a noisy public space or room but need to receive an important call or have an important meeting online. The model can identify any sound and further mute it for a clearer sound and audio experience.

Data

The dataset for this project is the UrbanSound dataset developed by (Justin et al., 2014, 1041). The dataset contains 8732 labeled sound excerpts (<=4 seconds) of urban sounds from 10 classes:

1. air_conditioner,
2. car_horn,
3. children_playing,
4. dog_bark,
5. drilling,
6. enginge_idling,
7. gun_shot,
8. jackhammer,
9. siren, and
10. street_music.

The sounds are about 27 hours of audio with 18.5 hours of annotated sound event occurrences

across all the 10 sound classes. The dataset can be downloaded from their website.

As can be seen from figure 1, there is an imbalance of data the classes gun_shot and car_horn has less representation. This was tackled with stratification where the ratio of representation in the distribution remained the same when the data is split between training, validation, and testing.

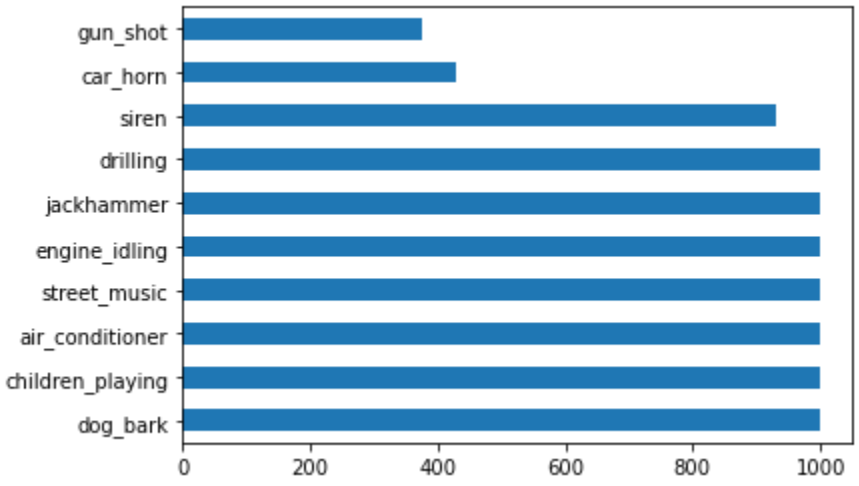


Fig 1 - distribution of classes in Urbansound 8k dataset

Tasks

This section outline all the tasks to complete the project:

1. Data gathering and pre-processing
   This task will describe in detail the processes involved in getting a good representation of soundwave for the model to learn.

2. CNN model building
   This task is to explore the application of 1D CNN on how to build a classifier for sounds. It also explores different architectures and parameters to help optimize the model.

3. Logistic model building
4. RNN model building
   The task is to enhance the LSTM model with a more complex deep learning architecture in order to achieve higher accuracy without overfitting. It is done by trying various architectures to optimize the model.

5. LSTM model building
6. Other model building

Table 1: list of tasks

Task	Who
Data pre-processing	Frederick
Logistic Regression	Belle
1D CNN	Frederick
ANN	Helia
2D CNN	Helia
LSTM	Saba
RNN-LSTM	Rishabh

Methodology

The methodology for the projects is an experimental research approach. The aim is to get to the best model for the underlying sound classification problem.

1. Data pre-processing

In order to get the audio data to train the model, there are two main ways to represent the audio. These are with raw audio or spectrogram.

Raw audio

This is in a waveform or soundwave that is fed into the network to build a classifier. As shown in Fig [], there is no pre-processing here. Examples of systems of models that use raw audio as input include WaveNet [] and Jukebox [].

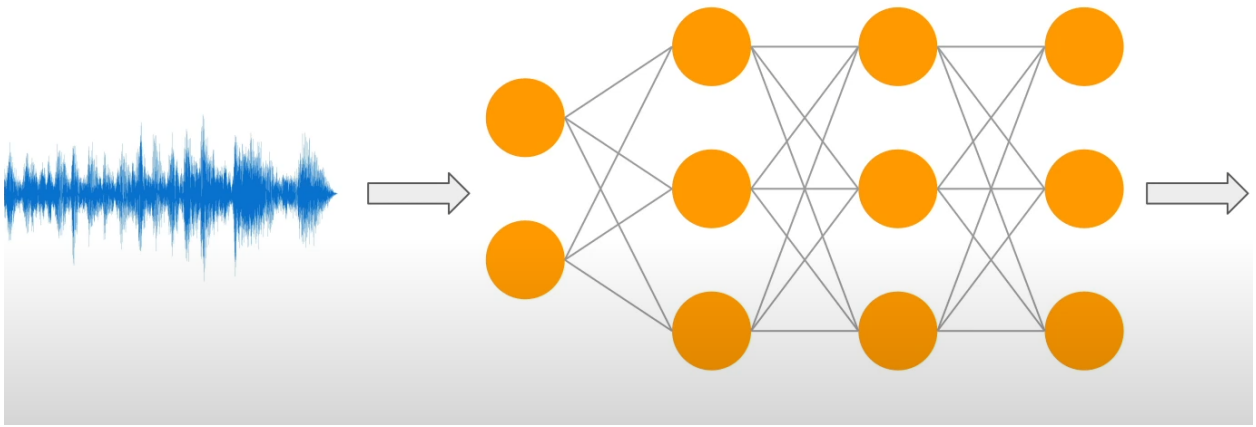


Fig 2 raw soundwave through a network

A sound wave is time-series data with a discrete point in time known as the sample points on the x-axis as depicted in Fig[]. One second worth of audio in a CD is equivalent to 44,100 samples, hence, this series of data points is usually a lot of data and heavy for one audio file.

Using raw audio as input has its challenges and some include:

- It is difficult to computer long-range dependencies
- It is computationally expensive, hence, it will take a very long time to train since the model will have a lot of parameters.

The above disadvantages lead us to the next data representation which is spectrogram.

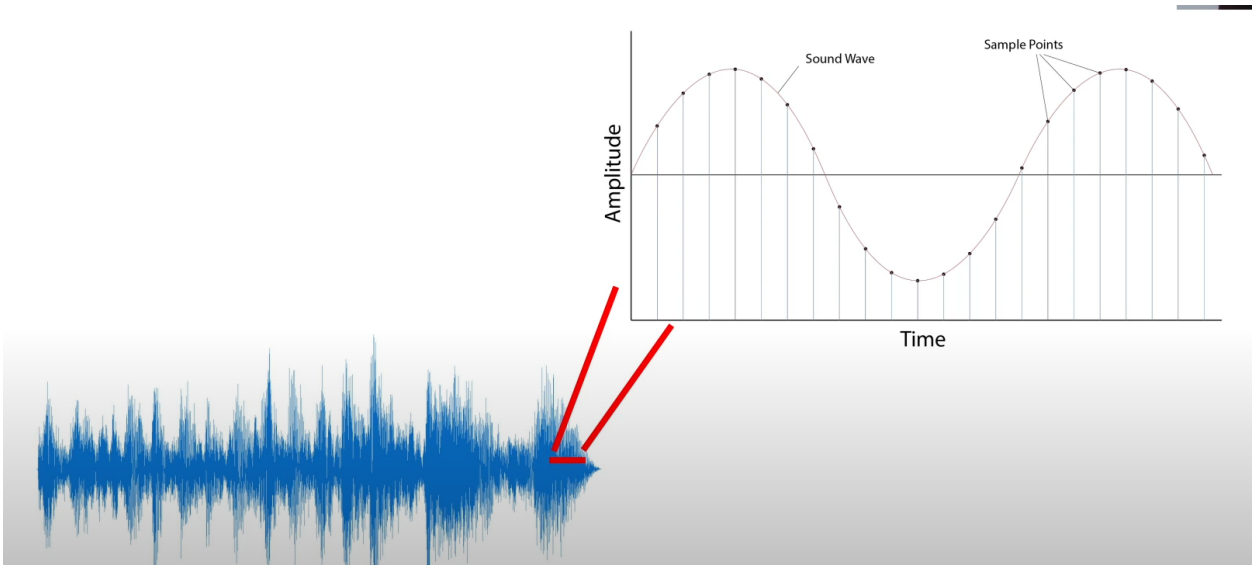


Fig. 3 A sound of a soundwave

Spectrogram

It is the transformation of a soundwave into a time-frequency domain. The y-axis shows the frequency bins while the x-axis shows the time with discrete values. The heat map depicted in Fig[] shows how much energy is at a particular frequency bin with respect to time. The brighter the color, the more energy is present in a frequency bin.

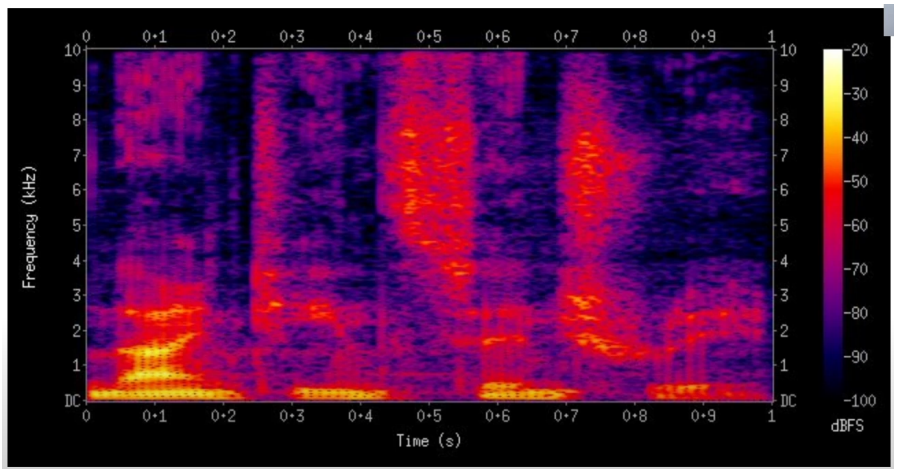


Fig. 4 a soundwave spectrogram

There are different types of spectrograms and some are; the vanilla log-amplitude, log frequency-amplitude, and Mel spectrograms. What is used for this project is the Mel spectrogram because that is widely used in literature like MelNet[] and DRUMGAN [].

The advantages of a spectrogram over a soundwave are;

- The temporal axis or x-axis is more compact
- It also captures long-time dependencies
- It uses less computational power.

Feature extraction

As indicated above, the Mel spectrogram is used for this project. As shown in fig [] the *librosa* library is used to extract features from the soundwave with mfcc coefficients into a one channel 1D vector which are the input variables. The input dimensions used differs per model as described with each model architecture. Librosa library automatically normalized the features in the range of [-1 and 1].

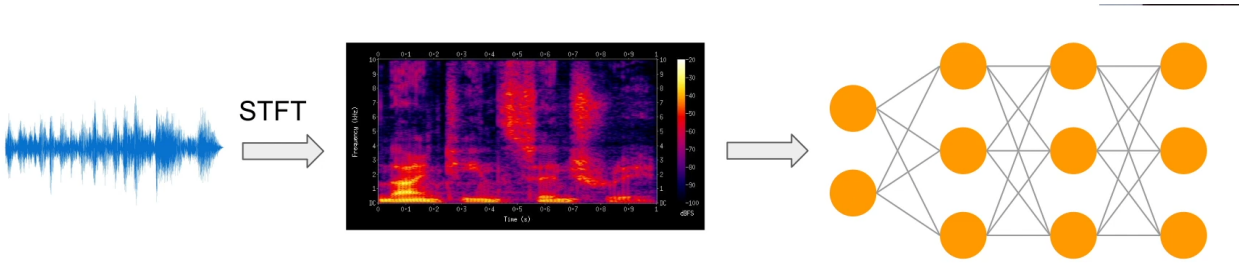


Fig. 5 - The flow of soundwave through to the network

2. ANN Model

As demonstrated in fig 6 the ANN architecture consists of 4 hidden layers (dense) with activation function of Relu, and the output layer with Softmax activation function. In order to prevent overfiting 3 Dropout layers with rate =0.4 were used. Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different “view” of the configured layer.

The model was optimized by ADAM optimizer using categorical_crossentropy loss function. The model was trained and evaluated in 50 epochs with batch_size= 64.

By observing loss and accuracy plot of training in each epoch, shown in fig 7, we can see the model is a good fit of data, the accuracy of %91.79.

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_72 (Dense)            (None, 1000)              129000

 dropout_27 (Dropout)        (None, 1000)              0

 dense_73 (Dense)            (None, 750)               750750

 dropout_28 (Dropout)        (None, 750)               0

 dense_74 (Dense)            (None, 500)               375500

 dropout_29 (Dropout)        (None, 500)               0

 dense_75 (Dense)            (None, 250)               125250

 dense_76 (Dense)            (None, 10)                2510

=================================================================
Total params: 1,383,010
Trainable params: 1,383,010
Non-trainable params: 0
_________________________________________________________________
```

```python
# Construct model
num_labels = y.shape[1]
model = Sequential()
model.add(Dense(1000,activation="relu",input_shape=(X_train.shape[1],)))
model.add(Dropout(0.4))
model.add(Dense(750,activation="relu"))
model.add(Dropout(0.4))
model.add(Dense(500,activation="relu"))
model.add(Dropout(0.4))
model.add(Dense(250,activation="relu"))
model.add(Dense(num_labels ,activation="softmax"))
model.summary()
```

Fig 6. ANN Model Architecture

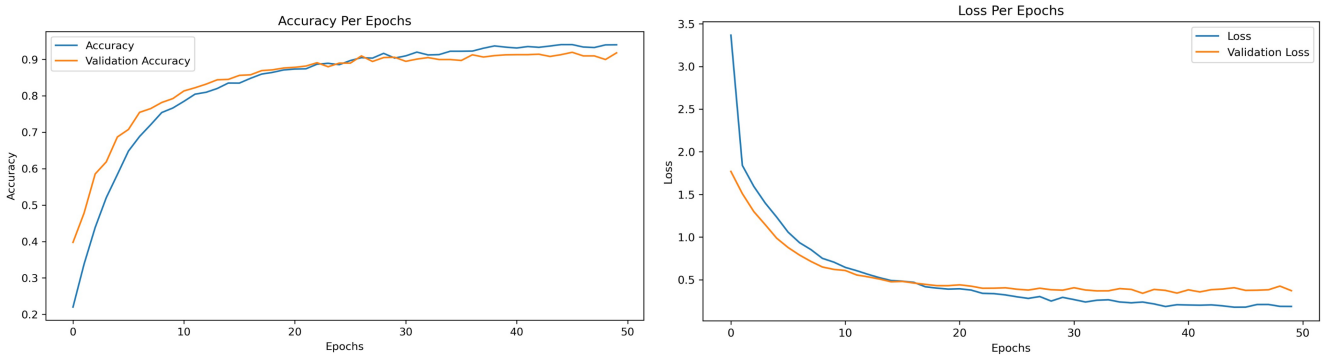


Fig 7. ANN Model Accuracy/epoch and Loss/epoch plot

3. CNN Model 1D

The CNN model architecture is shown in fig 8 with all parameters. The CNN model architecture is made up of 8 layers, 2 stack Conv1D with *relu* as activation function. A dropout of 50 percent is added to reduce complexity. MaxPooling1D of size 2 is added. The output is then flattened and another dropout of 50 percent to further reduce model complexity. The output is passed through a fully connected network with an output of 64 The finally fully connected network classifier with an output of the number of classes is applied with softmax.

Softmax activation helps get the probability with a total of one.

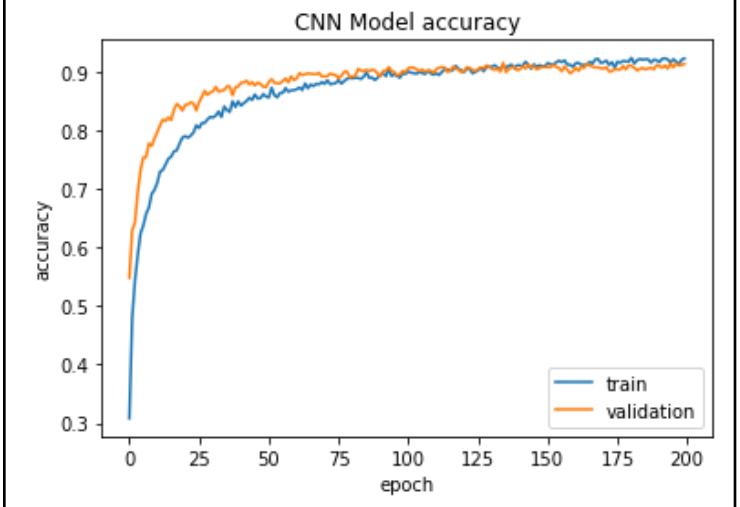
The last dropout layer was added to overcome overfitting. Fig 9 shows an overfitted model without the last dropout layer, where the validation diverges from the training accuracy, as well as the good fit for the model after adding the dropout layer.

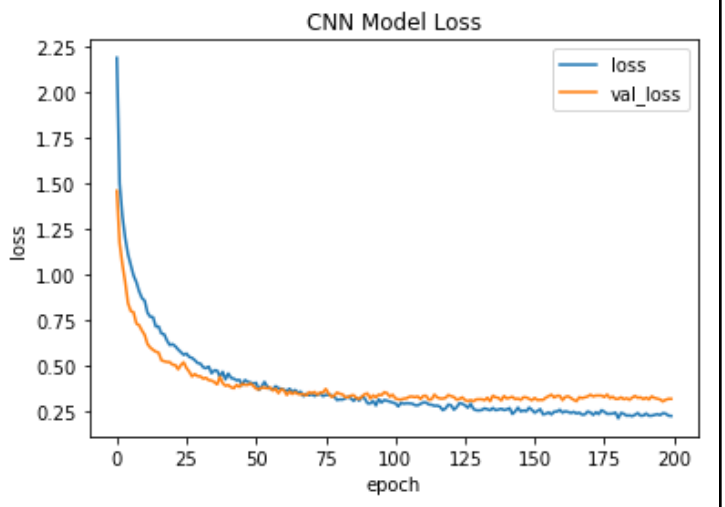
What was used to compute the loss is ***categorical_crossentropy*** and ***accuracy*** as the matric.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_5 (Conv1D)            (None, 38, 64)            256

 conv1d_6 (Conv1D)            (None, 36, 64)            12352

 dropout_5 (Dropout)          (None, 36, 64)            0

 max_pooling1d_4 (MaxPooling  (None, 18, 64)            0
 1D)

 flatten_2 (Flatten)          (None, 1152)              0

 dropout_6 (Dropout)          (None, 1152)              0

 dense_4 (Dense)              (None, 64)                73792

 dense_5 (Dense)              (None, 10)                650

=================================================================
Total params: 87,050
Trainable params: 87,050
Non-trainable params: 0
_________________________________________________________________
```

Fig 8. 1D CNN Model Architecture

Good CNN model





Overfitted Model

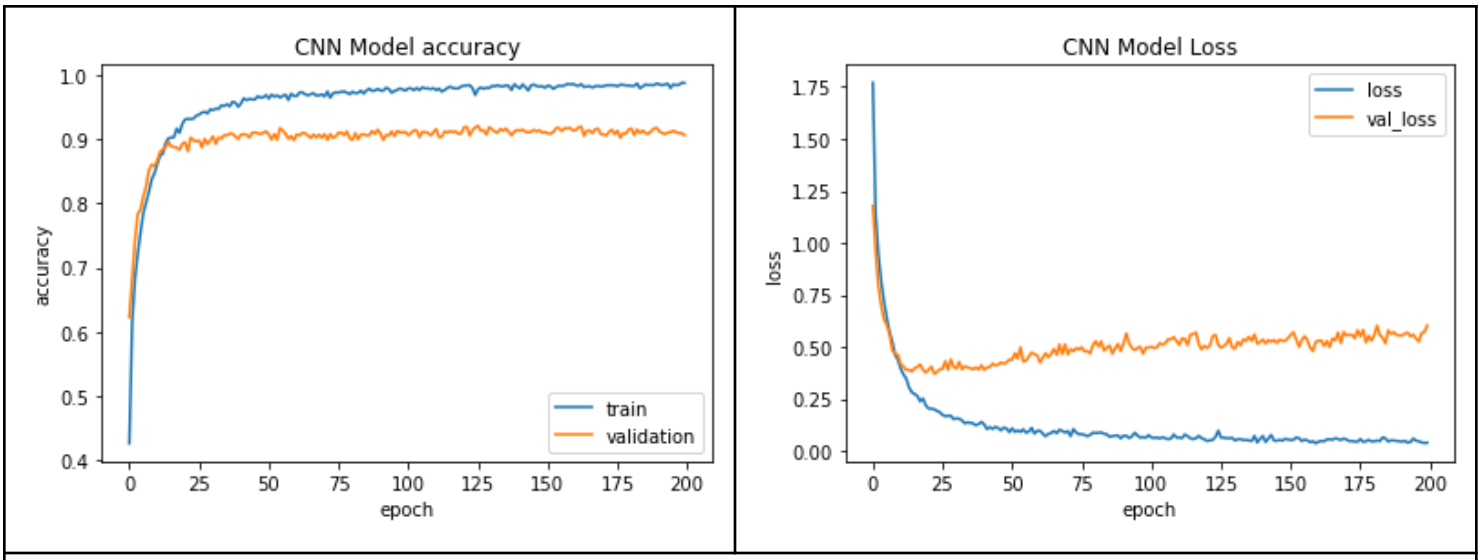


Fig 9. 1D CNN Overfitted and good fit models

4. CNN Model 2D

As demonstrated in fig 10 the 2 dimensional CNN architecture consists of 2 convolutional blocks and one dense layer in hidden layers with activation function of tanh, and the output layer with Softmax activation function. In order to prevent overfiting a Dropout layer with rate =0.4 were used.

As discussed before, data is a 1D spectogram. In order to feed the 1D spectogram to the 2D CNN the first step is to reshaping data. Consequently, we have used reshape() function to convert the 1 dimention spectogram with 128 feature to a multi-dimension spectogram with (16, 8, 1) features.

The model was optimized by ADAM optimizer using categorical_crossentropy loss function. The model was trained and evaluated in 50 epochs with batch_size= 64.

By observing loss and accuracy plot of training in each epoch, shown in fig 11, we can see than even though the accuracy of the model is %91.30, the model is overfitted. Overfitting occurs when a statistical model fits exactly against its training data. When this happens, the algorithm unfortunately cannot perform accurately against unseen data, defeating its purpose. Generalization of a model to new data is ultimately what allows us to use machine learning algorithms every day to make predictions and classify data. Here the divergence between training loss and validation loss shows the model is learning the training data very well but does not work well with validation data.

There are different strategies that can help us overcome the overfitng of the model. We have employed three of these strategies to enhance our 2D CNN model.

First, we tried increasing the Dropout rate and doing the regularization. However, this didnt help with overfitting after 10 epochs. Although the model started with a training accuracy much smaller than validation accuracy, after 10 epochs the model got overfitted.

Second, we tried to reduce the model complexity by decreasing the number of parameters as illustrated in fig 12 in which one convolutional block is eliminated. However, it can be seen that again the train and validation loss are diverging after about 10 epochs, shown in fig 13.

The last strategy we employed is early stopping. Fig 14 shows the first 20 epochs of our enhanced 2D CNN model.

To have a reliable result after 12 epochs, the model training is stopped. The accuracy for the enhanced model is %90.99.

```
Model: "sequential_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 16, 8, 64)         640

 max_pooling2d_32 (MaxPoolin  (None, 8, 4, 64)         0
 g2D)

 dropout_53 (Dropout)        (None, 8, 4, 64)          0

 conv2d_33 (Conv2D)          (None, 8, 4, 128)         73856

 max_pooling2d_33 (MaxPoolin  (None, 4, 2, 128)        0
 g2D)

 dropout_54 (Dropout)        (None, 4, 2, 128)         0

 flatten_17 (Flatten)        (None, 1024)              0

 dense_105 (Dense)           (None, 512)               524800

 dense_106 (Dense)           (None, 10)                5130

=================================================================
Total params: 604,426
Trainable params: 604,426
Non-trainable params: 0
_________________________________________________________________
```

```
CNN2D_Model = Sequential()
CNN2D_Model.add(Conv2D(64, (3, 3), padding="same", activation="tanh", input_shape=(16, 8, 1)))
CNN2D_Model.add(MaxPool2D(pool_size=(2, 2)))
CNN2D_Model.add(Dropout(0.2))
CNN2D_Model.add(Conv2D(128, (3, 3), padding="same", activation="tanh"))
CNN2D_Model.add(MaxPool2D(pool_size=(2, 2)))
CNN2D_Model.add(Dropout(0.4))
CNN2D_Model.add(Flatten())
CNN2D_Model.add(Dense(512, activation="tanh"))
CNN2D_Model.add(Dense(10, activation="softmax"))
CNN2D_Model.summary()
```

Fig 10. 2D CNN Model Architecture

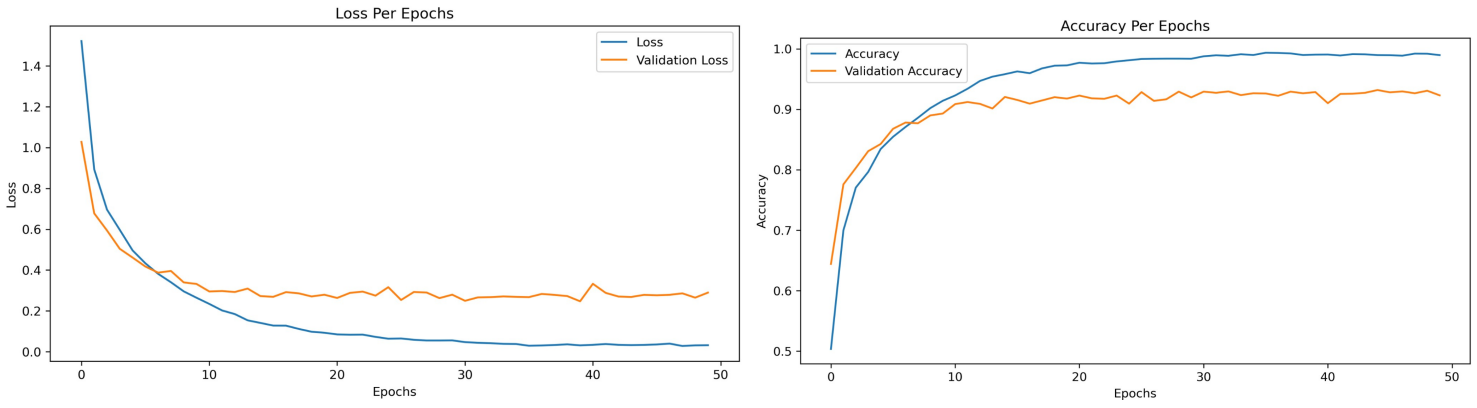


Fig 11. 2D CNN Model Accuracy/epoch and Loss/epoch plot

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_38 (Conv2D)          (None, 16, 8, 64)         640

 max_pooling2d_38 (MaxPoolin  (None, 8, 4, 64)         0
 g2D)

 dropout_59 (Dropout)        (None, 8, 4, 64)          0

 flatten_22 (Flatten)        (None, 2048)              0

 dense_115 (Dense)           (None, 512)               1049088

 dense_116 (Dense)           (None, 10)                5130

=================================================================
Total params: 1,054,858
Trainable params: 1,054,858
Non-trainable params: 0
_________________________________________________________________
```

```
CNN2D_Model = Sequential()
CNN2D_Model.add(Conv2D(64, (3, 3), padding="same", activation="tanh", input_shape=(16, 8, 1)))
CNN2D_Model.add(MaxPool2D(pool_size=(2, 2)))
CNN2D_Model.add(Dropout(0.4))
CNN2D_Model.add(Flatten())
CNN2D_Model.add(Dense(512, activation="tanh"))
CNN2D_Model.add(Dense(10, activation="softmax"))
CNN2D_Model.summary()
```

Fig 12. Enhanced 2D CNN Model Architecture

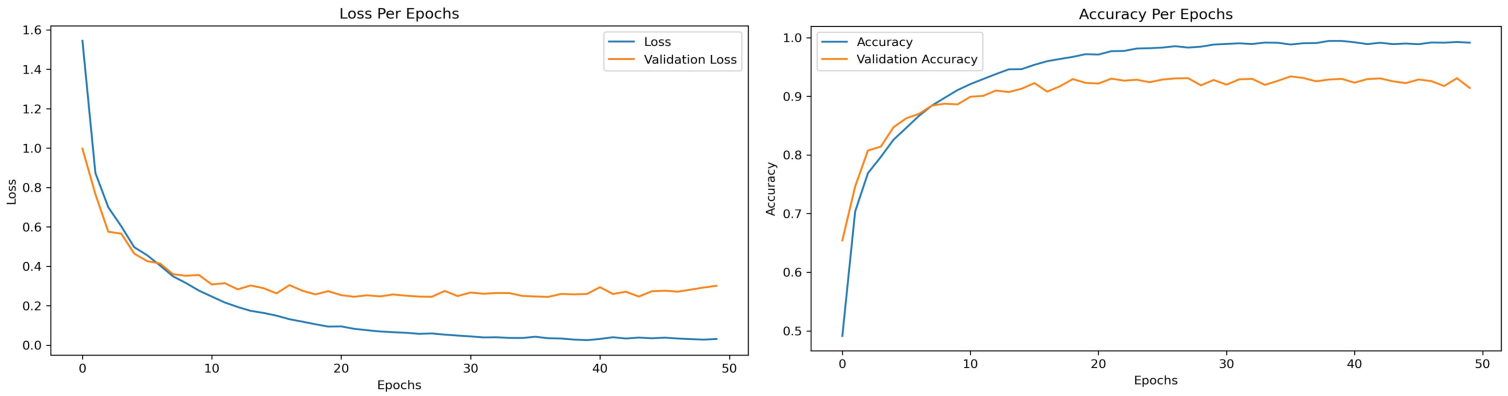


Fig 13. Enhanced 2D CNN Model Accuracy/epoch and Loss/epoch plot

	loss	accuracy	val_loss	val_accuracy
0	1.545085	0.491165	0.997398	0.654198
1	0.874107	0.703534	0.766674	0.746183
2	0.699536	0.768979	0.575589	0.807634
3	0.603442	0.796466	0.566079	0.814122
4	0.497383	0.826080	0.465091	0.847328
5	0.456479	0.846368	0.426570	0.862595
6	0.404274	0.866983	0.414405	0.870229
7	0.349415	0.884326	0.360333	0.884351
8	0.315396	0.897579	0.352367	0.887405
9	0.277015	0.910668	0.356423	0.886260
10	0.246615	0.920812	0.308107	0.899237
11	0.216714	0.929483	0.314786	0.900763
12	0.193839	0.937991	0.283368	0.909924
13	0.174262	0.946008	0.302971	0.907252
14	0.163628	0.946335	0.289254	0.912977
15	0.149827	0.953698	0.262974	0.922519
16	0.131737	0.959915	0.304804	0.908015
17	0.119393	0.963678	0.276186	0.917176
18	0.106320	0.967277	0.257882	0.929389
19	0.094331	0.971859	0.274050	0.922901
20	0.095298	0.971204	0.253870	0.921756

Fig 14. First 20 training epochs of enhanced 2D CNN Model

5. Enhanced Recurrent Neural Network

The RNN model was enhanced by increasing the depth and adding more layers hence, increasing the complexity of our RNN. This was done in order to achieve an enhanced performing model with high accuracy. The enhanced RNN model architecture is shown in fig with all parameters. The model architecture is made up of 8 layers, 2 stack LSTM with relu as activation function. Increasing the complexity of the model also raises the fear of overfitting therefore dropout layers are added to decrease complexity after each layer. The output is passed through a fully connected network with an output of 1000. The finally fully connected network classifier with an output of the number of classes is applied with softmax to get a total probability of one. During initial trial and error, the model reached high accuracy but it was overfitting as can be seen in the figure, where the validation diverges from the training accuracy. The final model architecture as shown in the figure below reached an accuracy of 89.42% without overfitting.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_12 (LSTM)               (None, 100, 1000)         4008000
_________________________________________________________________
dropout_24 (Dropout)         (None, 100, 1000)         0
_________________________________________________________________
lstm_13 (LSTM)               (None, 512)               3098624
_________________________________________________________________
dropout_25 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_27 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_28 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_26 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_29 (Dense)             (None, 10)                2570
=================================================================
Total params: 7,503,178
Trainable params: 7,503,178
Non-trainable params: 0
_________________________________________________________________
```

Fig 15. Enhanced RNN Model Architecture

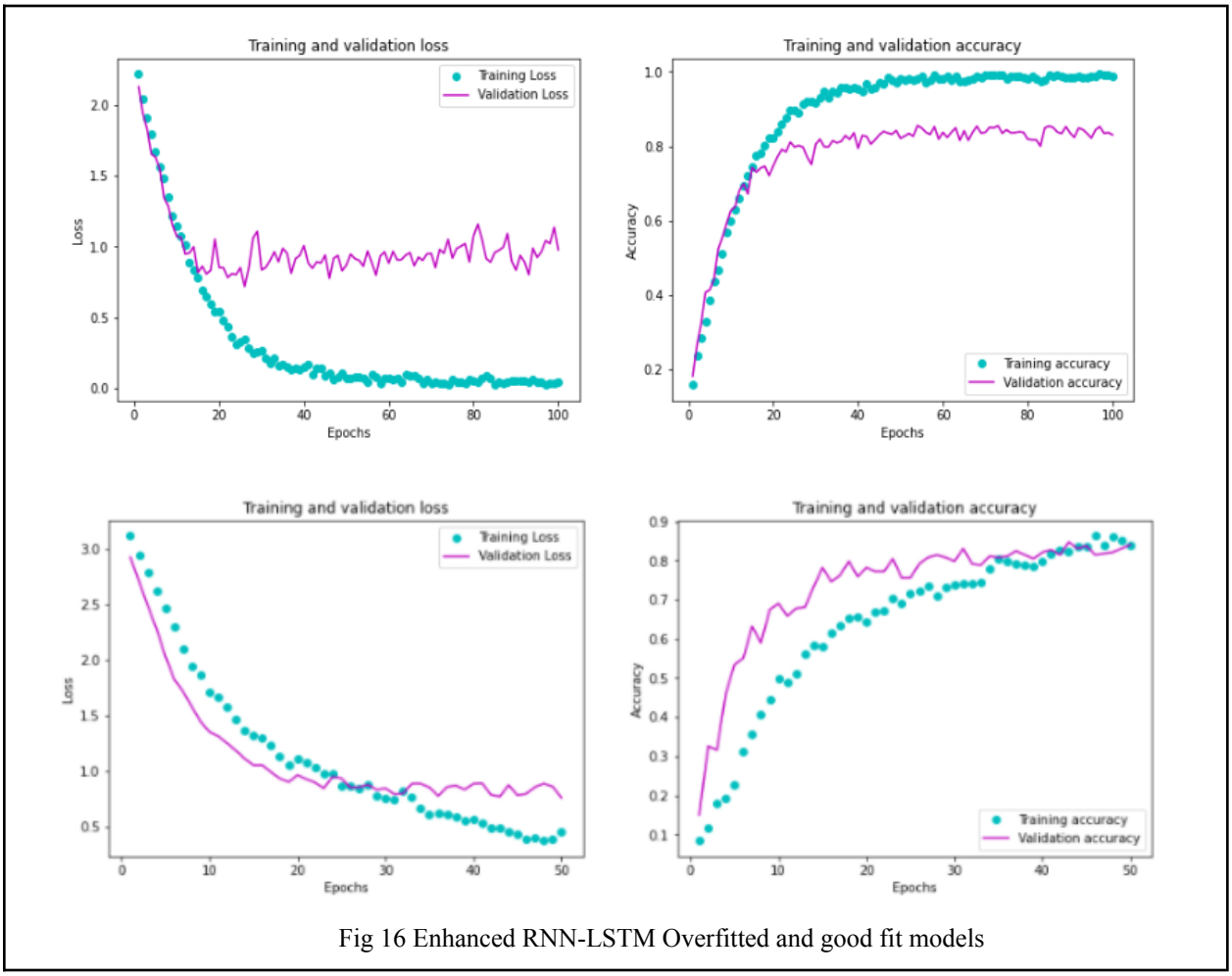


Fig 16 Enhanced RNN-LSTM Overfitted and good fit models

6. Logistic regression Model

Logistic Regression is a ML model we used to benchmark the other algorithms. It is not meant to be a serious attempt at predicting outcomes but giving us a baseline target to beat. It is a linear classification model that sorts our sounds into the ten categories.
Logistic Regression was applied with default parameters and the accuracy was found to be 55.57%.

GridSearchCV was then implemented to create a matrix of logistic regression models using different parameters. Penalties included L1 and L2 regularization as well as elasticnet which combines the two. Regularization is meant to prevent overfitting. L1 does this by assigning no weight to certain features. L2 assigns weights that are close to but not quite zero. Regularization strength C was set to numbers between -4 and 4. Solvers that can be used for multimodal classification include were newton-cg, sag, saga, and lbfgs. Newton-cg uses Newton's method using a quadratic loss function. Lbfgs is considered better for smaller datasets, but we included it just in case. SAG uses stochastic methods and is fast for large datasets. These are all only compatible with L2 regularization, while SAGA is compatible with L1 regularization, using stochastic methods as well.

Gridsearch determined that the best parameters were $C = 4.8 \times 10^{-3}$, L2 regularization, and a newton-cg solver where the loss function is quadratic which gave an accuracy of 59.16%.

The confusion matrix below shows how good this model was at accurately predicting each type of sound. This model predicted street music and air conditioners with a high degree of accuracy, but did not work well at all for identifying idling engines.

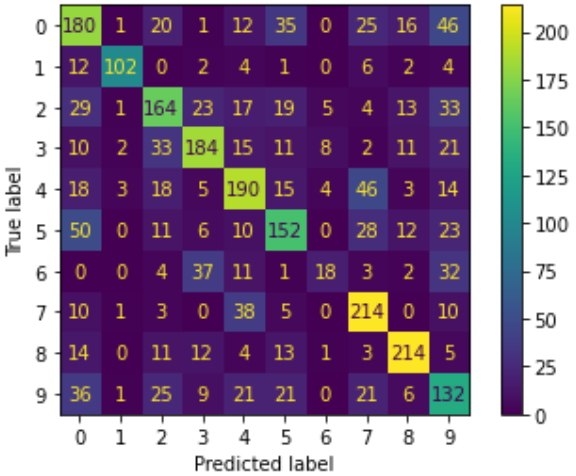


Fig 17 Confusion Matrix

7. LSTM

Long short-term memory (LSTM) uses feedback connections. Such a recurrent neural network can process not only single data points in this case audio. The problem with vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using backpropagation. The model architecture is made up of 8 layers 2 stack LSTM with relu as activation function. The result of the LSTM model though is not optimized in this case. The problem is the layers of LSTM being used in wrong positions for the audio to classify. The model reached the 78.85 accuracy which is not acceptable, though because of the better models found in the experiment the model was not pursued for optimization.

```
=================================================================
 lstm (LSTM)                 (None, 40, 64)            16896

 lstm_1 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 128)               4224

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Fig 18 LSTM Model Architecture

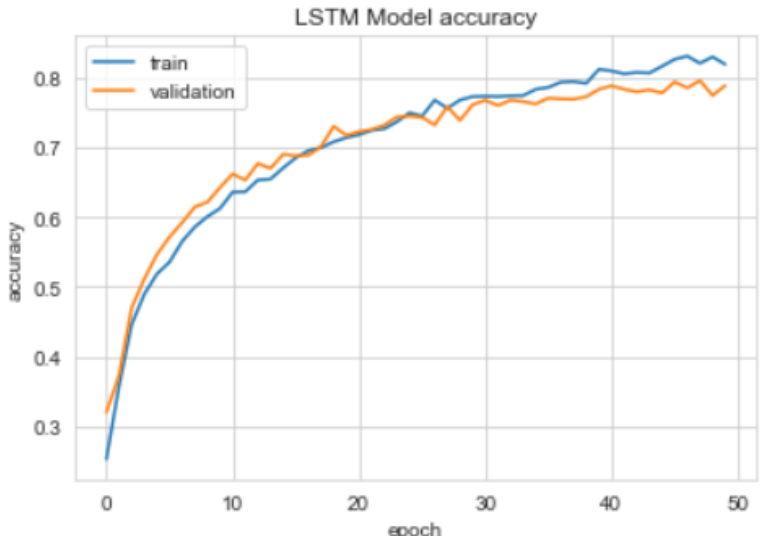


Fig 19 LSTM Model Accuracy

Fig 20 LSTM Model Loss

8. Results

Models	**Accuracy %**
Logistic regression	59.16
ANN Model	91.79
1D CNN Model	91.33
2D CNN Model	90.30
RNN Model	89.42
LSTM	78.8

9. Conclusion

By comparing the performance of the models, we can see almost all models, except our baseline, the logistic regression model, can classify 10 classes of UrbanSound 8k data with around %90 accuracy.

The ANN model demonstrates a slightly better result due to its more complex architecture and a more significant number of parameters. The 1D CNN also shows promising outcomes. We believe that by making some modifications to the 1D CNN architecture and adding more layers as well as controlling the overfitting, we can achieve a more promising result for the 1D CNN model. This also applies to Recurrent architectures, as we expected to see the best accuracy while using recurrent networks. Since recurrent networks, specifically the LSTM architecture, has a memory, they should work well for sequential data and time series. Consequently, we anticipate that we can have better results by modifying the LSTM network.

Another decent approach that can help us enhance the predictive performance of our classifier is the Ensemble Learning approach.
Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.
The three main classes of ensemble learning methods are bagging, stacking, and boosting, which are defined as follows:
Bagging involves fitting many decision trees on different samples of the same dataset and averaging the predictions.
Stacking involves fitting many different models types on the same data and using another model to learn how to best combine the predictions.
Boosting involves adding ensemble members sequentially that correct the predictions made by prior models and outputs a weighted average of the predictions.

10. Next Steps

For the next step we will the predictive model to classify sounds and ultimately denoise the Urbansounds by detecting them to have a noise cancellation mechanism.

11. Resources

Dataset:

- https://urbansounddataset.weebly.com/urbansound8k.html

Webpages:

- https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
- https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/