UNIVERSITI MALAYA

Faculty of Computer Science and Information Technology

WID3006: MACHINE LEARNING COURSE ASSIGNMENT

Semester 2 (2020/2021)

PROJECT : caLLar

Members of Group 26:

No.	Name	Matric Number
1.	Nurin Batrisyia Binti Mohd Azmi	17207087/2
2.	Siti Norhidayah Binti Abdul Bari Arbee	17206657/2
3.	Nur Izzati Binti Shalahudin	17205321/2
4.	Nursara Ain Binti Harzany	U2005391/1
5.	Sarah Binti Fauzi	17204412/2
6.	Yasmin Hannah Binti Zainol Abidin	17203246/2

Table of Contents

1. Introduction to Problem

As the theme for our Machine Learning course assignment is Love, we decided to tackle a different kind of love which is the love between a pet and its owner. For our project purpose we choose cats as our targeted pet. As pet owners, we sometimes find it difficult to satisfy our pets' needs as we could not understand their behaviour clearly. Thus, we decided to create a device which can help the owner to understand their pets better. Since cats are known to be quite vocal in expressing themselves, we will use the pet's different vocalization and portray it in a form of colours on our product after performing deep categorization through CaLLa mobile application. This will help to increase the understanding of the owners towards their pets behavior as well as improve their relationship.

1.1 Problem Statement

Difficulties to understand pets' emotions and behaviours lead to wrong interpretation and unsuitable actions taken.

1.2 Hypothesis For The Problem

The interpretation of the cat's behaviour changes according to vocalization of the cat retrieved.

<u>Implementation to our product</u>

Colour of the device changes as the result from the application changes after retrieving the sound of the cat and going through a deep learning process on recognizing the type of vocalization.

1.3 Project Objectives

1. Increase the quality of understanding cat's behaviour.
2. Using machine learning algorithms to solve the problem of detecting different pets' vocalization and determine what emotion it is.
3. Propose a device which can be easily used to detect pets' vocalizations, display it in a way that pets' owners can understand and easy to use for both sides.
4. Propose usage of application of phones to make the process easier

1.4 Similar Works or Solutions to our project

During our process of data searching and data collection we have found several works and solutions that bring great help for enhancing our understanding about voice vocalization with deep neural networks. The most similar work to our project was Music genre classification with python by Pandey, P.

References:

Pandey, P. (2018). Music genre classification with python. Retrieved from https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8

2. Methodology

We have four items that we want to highlight in our research methodology.

A. Identifying type of data to collect

For our CaLLar Project we have identified that we need to conduct a mixed-method methodology that attempts to combine the best of both qualitative and quantitative methodologies to integrate perspectives and create a rich picture. We have decided to collect audio of cats and categorize the labels for each of them.

B. Sampling design

Our main sampling design approach is non-probability sampling. We retrieve information and reliable results from similar work to our project. Due to time constraints we have discussed that it is better to use a convenience sample of categorized cats audio rather than a generalized audio collection. Therefore our sampling design may be considered as reliable and will bring great help towards completing this assignment.

C. Data collection methods

The data we got is from **theZenodo,** an open-access repository, which has a collection of cat vocalizations divided into three classes, which are brushing, isolation and waiting for food. The data consist of 440 observations and all files are named based on which class it belongs to. The sound excerpts are digital audio files in .wav format. The audio files contain meows emitted by cats in different contexts:

- **Brushing**
    - Cats were brushed by their owners in their home environment for a maximum of 5 minutes.
- **Isolation in an unfamiliar environment**
    - Cats were transferred to an unusual place by their owners (e.g., other than their home). Distance between them was minimized and the usual transportation routine was adopted to avoid discomfort to the cats. The journey only lasted less than 30 minutes and cats were allowed to be with their owners for 30 minutes before being isolated in a different environment for a maximum of 5 minutes.
- **Waiting for food**
    - Owner started the routine that preceded the usual meal time in the usual environment the cat was familiar with. Food was given at most 5 minutes after the beginning of the experiment.

D. Data cleaning

After several discussions on how to implement our ideas in our code assignment, we decided to make our own metadata to ensure that we will be able to find data, use data, and preserve and re-use data in the future.

As metadata was not given, we created one ourselves using the audio labels. The metadata has 3 columns, the first one being the file name, the second one, the class ID and the last column being the class name. This is the attached link to see our metadata: metadata

Here is a view of the first 5 elements in the metadata table:

	slice_file_name	classID	class
0	B_ANI01_MC_FN_SIM01_101.wav	1	brushing
1	B_ANI01_MC_FN_SIM01_102.wav	1	brushing
2	B_ANI01_MC_FN_SIM01_103.wav	1	brushing
3	B_ANI01_MC_FN_SIM01_301.wav	1	brushing
4	B_ANI01_MC_FN_SIM01_302.wav	1	brushing

Figure 1: The first five elements in the metadata table.

Here is a general view of our collected data. We decided to use 3 features (labels) in our dataset.

class	Audio categorization
brushing	127
isolation	221
waiting_for_food	92

Table 1: General view of the class and occurrences of the data collected.

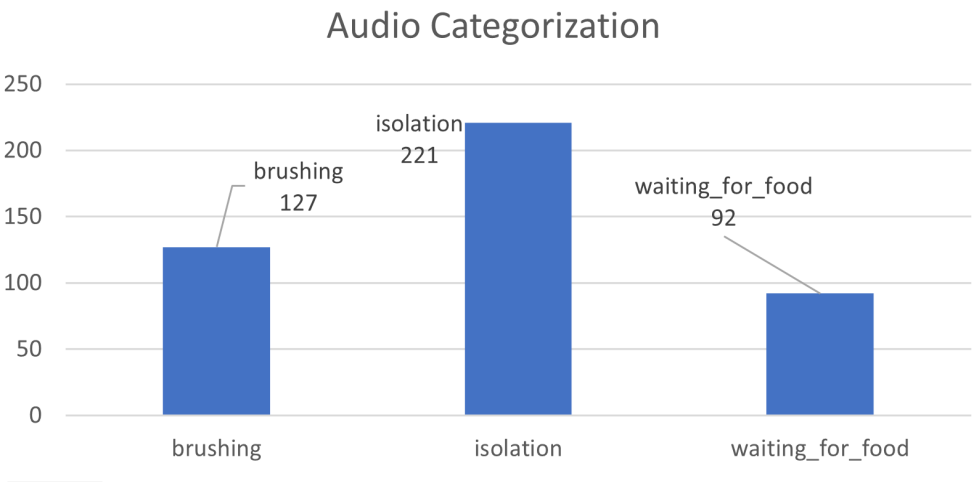


Figure 2: Occurrences of audio categorization collected bar chart.

We can see that our data is not evenly distributed as the isolation has the most number of audio files out of the three classes. This is the problem that we encountered and reached a hypothesis where our end prediction might be skewed.

E. Code implementations

Before we start making our code, we make sure we do some research and try to understand some packages and its usage to make sure we know how to

create a good understandable code. We organized our key ideas in bullet points below:

- To know about the librosa package and MFCC.
- Change the anaconda environment to a new one with tensorflow and keras downloaded as packages.
- To know the function of tensorflow.keras.callbacks import ModelCheckpoint.
    - Used in conjunction with training using model.fit() to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

E.1 Setting the random seed

Since our group is analyzing audio files, deep neural networks is the best choice to perform a supervised learning method. We import librosa for audio analysis, installing tensorflow and keras to perform deep learning and also importing other useful packages to strengthen our code. In our code we set the random seed to 42 and use tf.random.set_seed() and np.random.seed() to minimize the amount of correlations.

```
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

E.2 Handling imbalance data

Although we have checked that our metadata may be imbalanced we decided to let our metadata be that way in order to achieve the maximum type of cat vocalization we have. We choose this solution because if we reduce the isolation and brushing to 92, we think that it might lead our code to have low accuracy.

```
In [187]: ##check wheter the dataset is imbalanced
          metadata['class'].value_counts()

Out[187]: isolation          221
          brushing           127
          waiting_for_food    92
          Name: class, dtype: int64
```

E.3 Identify waveforms for each labels

To see how each audio looks, we use waveforms and spectrograms to help us in this endeavour. We pick one .wav file from each label as an example to view its wavelengths and spectrograms. Here is a view of our findings:

a. Brushing class Waveform and Spectrogram

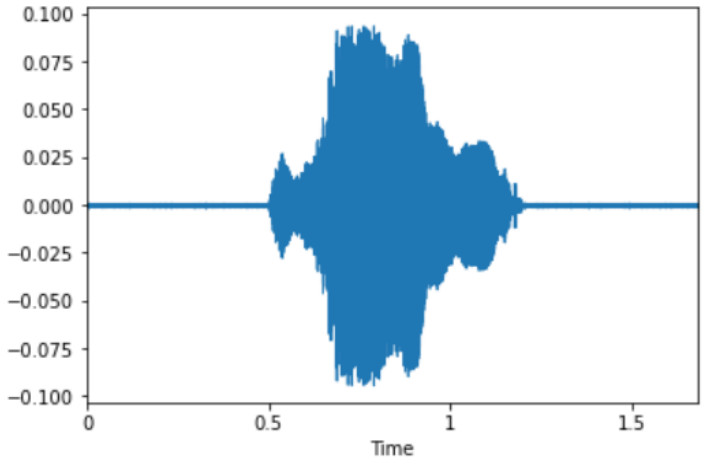


Figure 3: Waveform of brushing class.

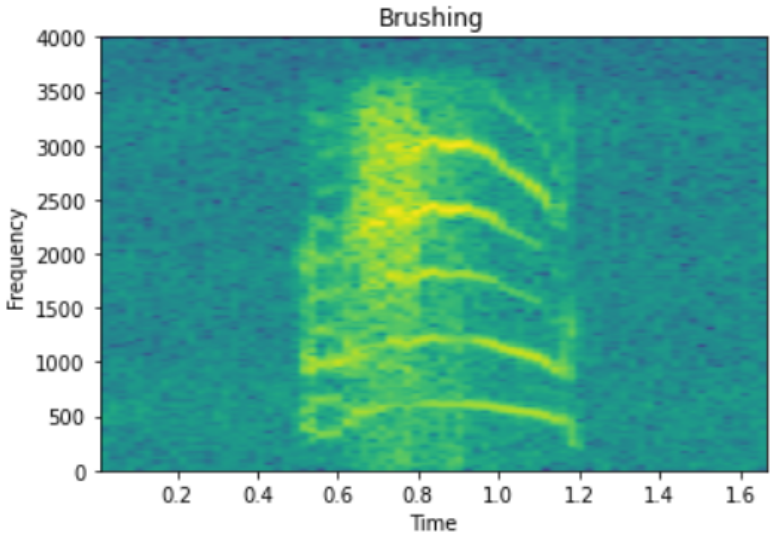


Figure 4: Spectrogram of brushing class.

b. Isolation class Waveform and Spectrogram

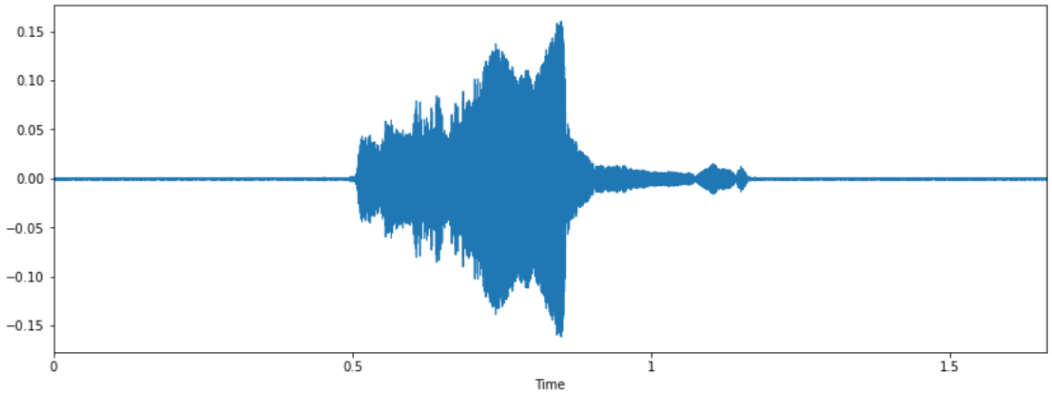


Figure 5: Waveform of isolation class.

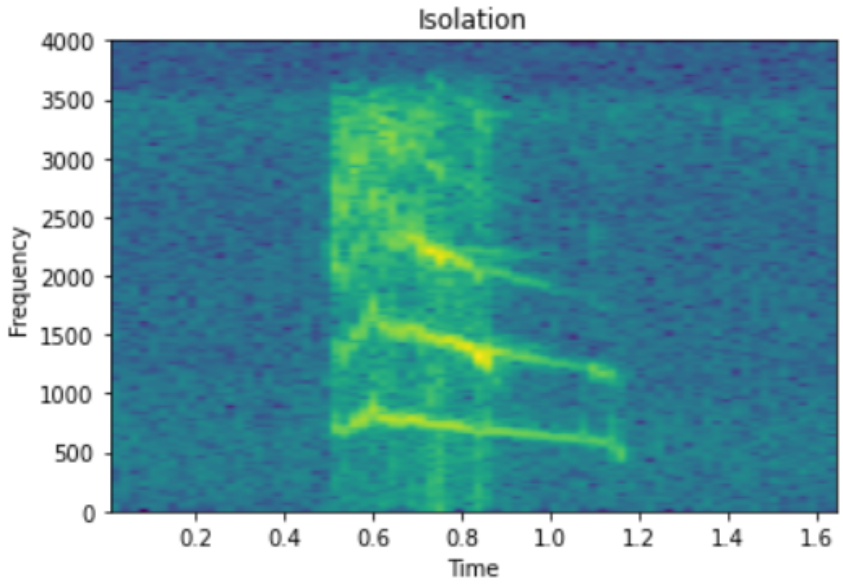


Figure 6: Spectrogram of isolation class.

c. Waiting for food class Waveforms and Spectrograms

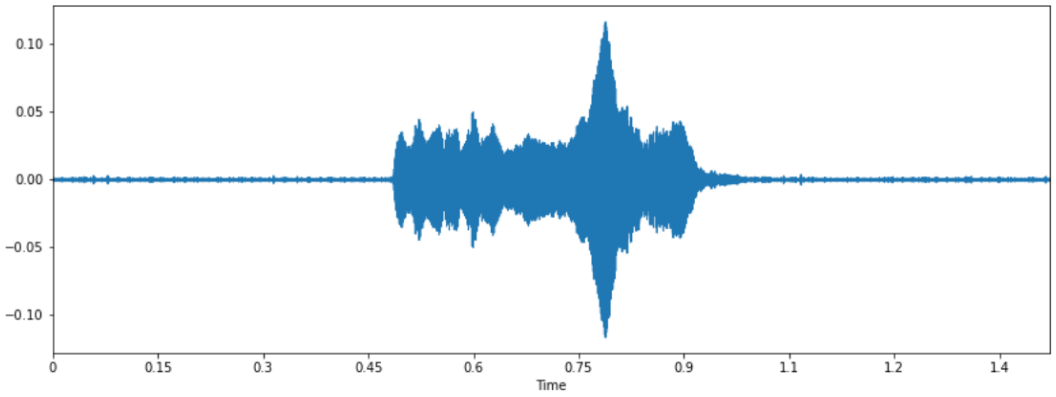


Figure 7: Waveform of waiting for food class.

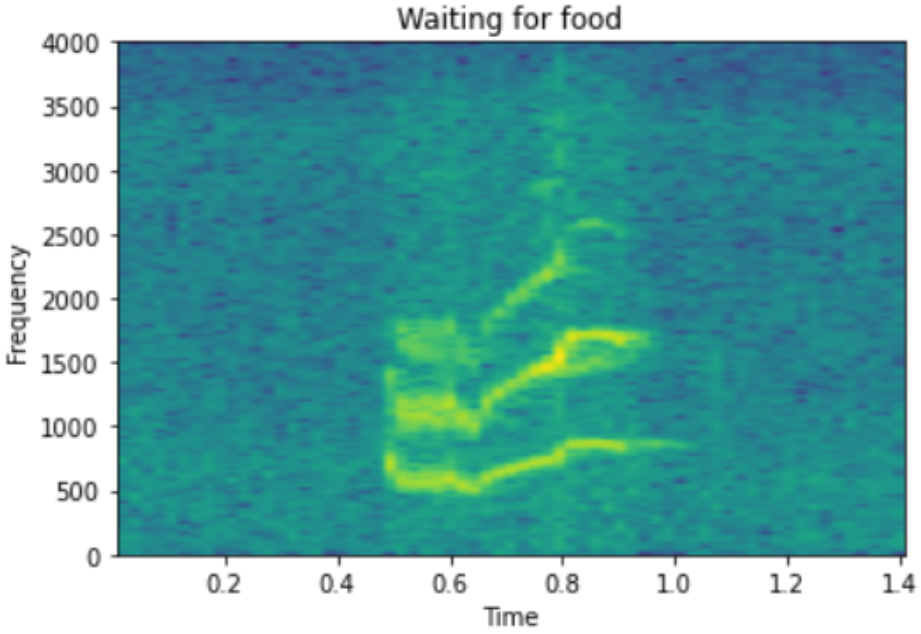


Figure 8: Spectrogram of waiting for food class.

E.4 Mel-Frequency Cepstral Coefficients (MFCC)

Next we use MFCC from the librosa package to extract features from the audio dataset. We define *features_extracted(file)* method and initialize the number of features extracted to 40. Then we iterate through every audio file and extract features using MFCC.

```
def features_extractor(file):
    audio,sample_rate= librosa.load(file_name,res_type = 'kaiser_fast')
    mfcss_features=librosa.feature.mfcc(y=audio,sr=sample_rate,n_mfcc=40)
    mfccs_scaled_features= np.mean(mfcss_features.T,axis=0)

    return mfccs_scaled_features
```

```
#Now we iterate through every audio file and extract features using MFCC
extracted_features=[]
for index_num, row in tqdm (metadata.iterrows()):
    file_name= os.path.join(os.path.abspath(audio_data_path),str(row["slice_file_name"]))
    final_class_label=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_label])
```

E.5 Refining the extracted features

We convert the extracted_features to pandas dataframe because it makes it easier to load the data from the dataset. The extracted_features are separated into features and class columns.

```
extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','class'])
extracted_features_df.head()
```

We set the features to be extracted as 40 features. Using the function we created, we iterated through all audio files that we downloaded from the dataset and extracted each audio's features. Once done, we turn the collected list of features into a pandas dataframe. Here is an overview of the data frame created:

	feature	class
0	[-396.81778, 136.41545, -74.92035, -11.478852,...	brushing
1	[-542.6039, 158.09341, -73.43401, -11.935203, ...	brushing
2	[-517.51764, 142.89006, -69.55315, -7.46689, 3...	brushing
3	[-476.4709, 112.17069, -65.29729, -10.651881, ...	brushing
4	[-511.12573, 135.90286, -62.427002, -11.154292...	brushing
...	...	...
435	[-429.12808, 116.40315, -54.10716, -5.52177, 2...	isolation
436	[-419.37286, 113.27804, -57.091076, -8.141786,...	isolation
437	[-429.6887, 127.5269, -63.110947, -8.256847, 1...	isolation
438	[-409.2688, 126.90218, -66.3937, -6.637888, 21...	isolation
439	[-441.6429, 108.86509, -46.91663, -14.050992, ...	isolation

After that we split the dataset into independent and dependent dataset. We can think of **independent and dependent variables** in terms of cause and effect: An **independent variable** is the **variable** you think is the cause, while a **dependent variable** is the effect. In an experiment, we manipulate the **independent variable** and measure the outcome in the **dependent variable**.

```
X=np.array(extracted_features_df['feature'].tolist())
y=np.array(extracted_features_df['class'].tolist())
```

We also implement label encoding in our code. Since y is categorical data, we used pandas get_dummies function to turn it into indicator variables.

```
y=np.array(pd.get_dummies(y))
#labelencoder = LabelEncoder()
#y=to_categorical(labelencoder.fit_transform(y))
```

E.6 Train test split

We use the sklearn package, to split the dataset into training and test sets to prevent the model from overfitting and to accurately evaluate the model.

```
In [200]: X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)

In [201]: print("X_train: ", X_train.shape)
          print("X_test: ", X_test.shape)
          print("y_train: ", y_train.shape)
          print("y_test: ", y_test.shape)

          X_train:  (352, 40)
          X_test:  (88, 40)
          y_train:  (352, 3)
          y_test:  (88, 3)
```

Based on the result we get from these two chunks of code, the dataset is split into two parts. One is the test set containing 20% of the dataset and the other is the training set containing 80% of the dataset.

E.7 Adding layers for models and compiling it

Here we add layers for the models in our code which consist of the first layer, second layer, third layer and output layer.

```
def create_model():
  model =Sequential([
        ###first layer
        Dense(100,activation='relu',input_shape=(40,)),
        Dropout(0.5),
        ### second layer
        Dense(200,activation='relu'),
        Dropout(0.5),
        ### third layer
        Dense(100,activation='relu'),
        Dropout(0.5),
        ### last layer
        Dense(num_labels,activation='softmax')

  ])
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer='adam')
  return model
```

- First layer
  - Has 100 nodes, dropout-0.5, activation func - relu
- Second layer
  - Has 200 nodes, dropout-0.5, activation func - relu
  - Implementing a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
  - Reasoning of usage: easier to train and often achieves better performance.
- Third layer
  - Has 100 nodes, dropout - 0.5 , activation func - relu
- Output layer
  - Has 3 nodes(of number of classes)
  - The activation function we implement the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

We manage to implement these functions using tensorflow and keras packages. We compile the model with optimizer ‘adam’, loss = ‘categorical_crossentropy’ and metrics ‘accuracy’.

The summary of the model tells us how the model looks like.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 100)               4100
_________________________________________________________________
dropout_15 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_21 (Dense)             (None, 200)               20200
_________________________________________________________________
dropout_16 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_22 (Dense)             (None, 100)               20100
_________________________________________________________________
dropout_17 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_23 (Dense)             (None, 3)                 303
=================================================================
Total params: 44,703
Trainable params: 44,703
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: The summary of model.

E.8 Model training

The model training is done to ensure the best accuracy percentage. It is important to have the model to get used to classifying different types of cats' sounds to its respective type specified by the data provided.

The training is done by using ModelCheckpoint. It will save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

In this model training, the epoch is set to 100. The number of **epochs** is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One **epoch** means that each sample in the training dataset has had an opportunity to update the internal model parameters.

In [212]:
```
## Training the model
## features = 40
num_epochs= 100
num_batch_size=32
checkpoint_path = "training_1/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


checkpointer=ModelCheckpoint(filepath=checkpoint_path,save_weights_only=True,verbose=1,save_best_only=True)
start=datetime.now()

model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test,y_test), callbacks= [checkpointer])


duration = datetime.now()- start
print("Training completed in time: ", duration);
```

E.9 Predicting the accuracy

We intend to do accuracy testing to see how well our model has worked and to evaluate our model. We use tensorflow and keras packages and model.evaluate() function to predict the accuracy of the model.

After training the model a few times, we print out the model's accuracy percentage which is 70.45%.

```
In [218]: test_accurancy=model.evaluate(X_test,y_test,verbose=1)
          print(test_accurancy[1])

          3/3 [==============================] - 0s 997us/step - loss: 0.9814 - accuracy: 0.7045
          0.7045454382896423
```

Then we test the model by creating a new untrained model and then loading the saved weights from ModelCheckpoint to see if the prediction percentage stays the same.

- ➢ Untrained model, accuracy: 48.86%
- ➢ Restored model, accuracy: 69.32%

```
In [219]: # Create a basic model instance
          model2 = create_model()

          # Evaluate the model
          loss, acc = model2.evaluate(X_test, y_test, verbose=1)
          print("Untrained model, accuracy: {:5.2f}%".format(100 * acc))

          3/3 [==============================] - 0s 665us/step - loss: 31.5889 - accuracy: 0.4886
          Untrained model, accuracy: 48.86%
```

```
In [220]: # Loads the weights
          model2.load_weights(checkpoint_path)

          # Re-evaluate the model
          loss, acc = model2.evaluate(X_test, y_test, verbose=0)
          print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

          Restored model, accuracy: 69.32%
```

3. Data analysis methods: Result and discussions

3.1 Confusion matrix

n = 88		Predicted class		
		Brushing	Isolation	Waiting for food
Actual class	Brushing	18	3	4
	Isolation	1	38	4
	Waiting for food	6	8	6

True Positives (60)

- Brushing = 18
- Isolation = 38
- Waiting for food = 6

False Positives (26)

- Brushing = 7
- Isolation = 5
- Waiting for food = 14

False Negatives (26)

- Brushing = 7
- Isolation = 11
- Waiting for food = 8

True Negatives (150)

- Brushing = 56
- Isolation = 34
- Waiting for food = 60

The model correctly predicts 60 audios into its classes. It also correctly predicted 150 audios that did not belong to the correct classes. This means it is able to differentiate the audio to the respective classes. However, the model also predicted 26 audios to a wrong class and another 26 audios incorrectly predicted that it did not belong to a certain class despite that it is. This confusion matrix will be useful to calculate performance metrics.

3.2 Performance metrics

Class	Precision	Recall	F1-score
Brushing	0.72	0.72	0.72
Isolation	0.78	0.88	0.83
Waiting for food	0.43	0.30	0.35
Macro average:	0.64	0.63	0.63
Weighted average:	0.68	0.70	0.69
Accuracy:	0.70		

For the 'Brushing' class, the model's precision is 0.72 while for the 'Isolation' class it is 0.78 but the precision for 'Waiting for food' is 0.43, lower than 50%. Same goes to Recall and F1-Score. This could be caused by lack of data in the class. In the data provided, the 'Waiting for food' has only 97 audios whereas the other class has more than 100 audios. In total, it affected the accuracy to 0.70 which is in the lower part of the good range.

3.3 Area under ROC curve (AUC)

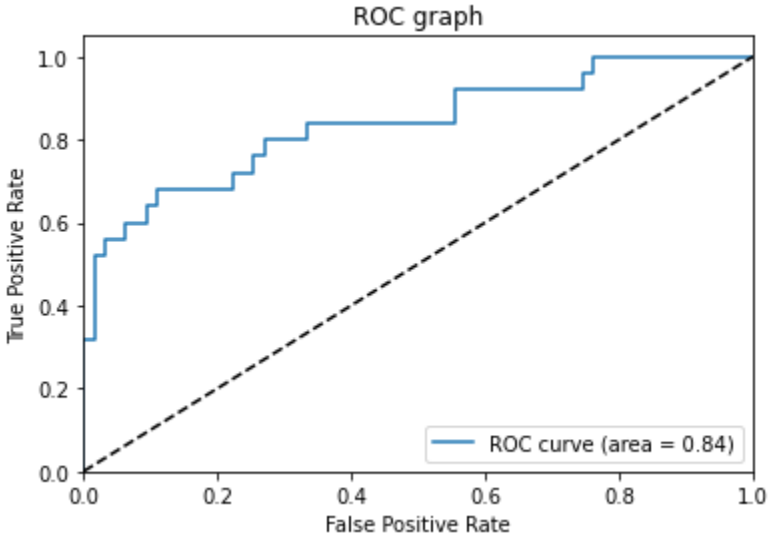
ROC tells us how good the model is for distinguishing the given class, in terms of the predicted probability. It shows us the relationship between False Positive Rate (FPR) and True Positive Rate (TPR) across different thresholds.

FPR = False positives/Negatives
TPR = True positives/Positives

AUC of 0.0 = predictions 100% wrong
AUC of 1.0 = predictions 100% correct

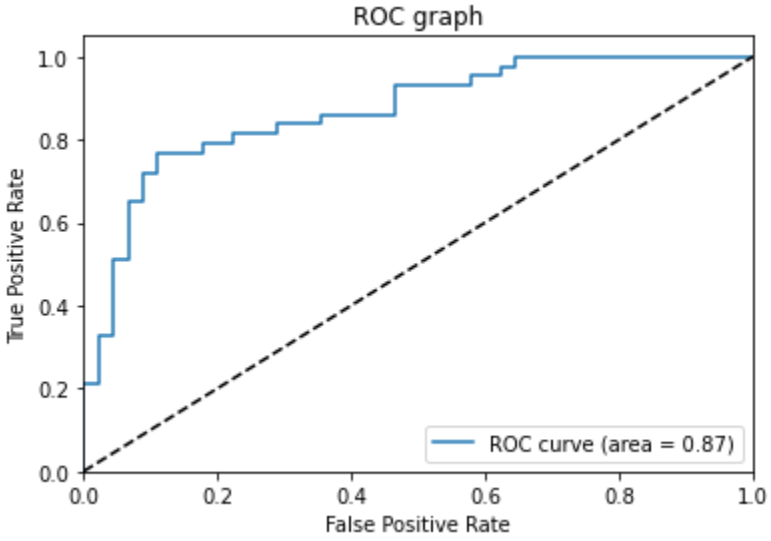
The area under the ROC curve (AUC) is between 0.7 and 0.87. This means that the model will have type 1 and type 2 errors. It will also mean the model will have 70% to 87% chance to distinguish between positive class and negative class correctly.

a. ‘Brushing’ class



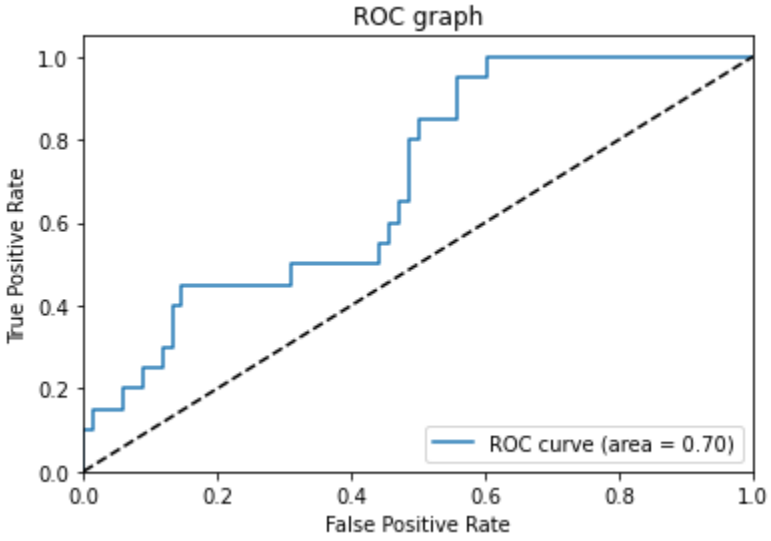
The above ROC curve shows that for the ‘Brushing’ class it will give 84% correct predictions and 16% wrong predictions. The curve is shown to be far from the diagonal line which means that the model is classifying the audio accurately by 84%.

b. ‘Isolation’ class



This shows that for ‘Isolation’ class it will give 87% correct predictions and 13% wrong predictions. The curve is also shown to be far from the diagonal line and a little bit close to the y-axis which means that the model is predicting more accurately towards this class than others.

‘Waiting for food’ class



This shows that for the ‘Waiting for food’ class it will give 70% correct predictions and 30% wrong predictions.The curve is also shown to be close to the diagonal line which means the model is prone to incorrectly classify the class to a different class.

4. Suggestion For Future Works

For conducting future works or experiments related to our project, we suggest collecting each type of the data equally. For example the number of data collected for each label must be equal. On the other hand it is better to use more types of audio dataset instead of just three types to increase the accuracy of the product and we can widen our product testing.

5. Appendix

Presentation Slide: CaLLar Pitch Deck.pptx

Video: https://youtu.be/cCeHmQ-r0xA

Source code: https://github.com/Nurlzzati11/CaLLaR/blob/main/project-CaLLaR%20-%20Final.ipynb