# Multiple Events Detection

**Parampreet Singh**
PhD (SPCOM)
Roll No: 21104268
params21@iitk.ac.in

## 1 Introduction

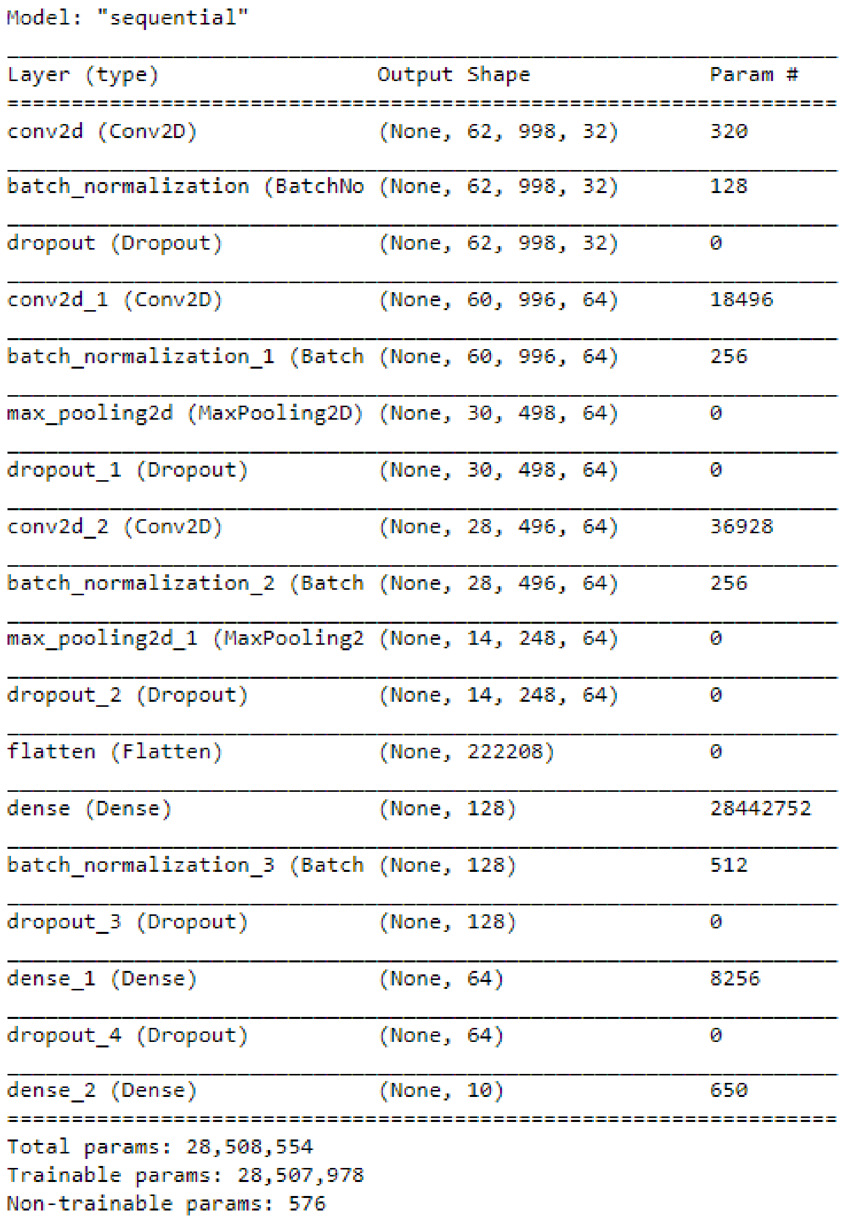
Sound event detection (SED) is a research field that aims to detect and identify events in audio signals. In addition to playing a significant role in understanding the audio of real-life sensing, it also has a wide range of applications such as automatic driving, surveillance systems, health care, and humanoid robots. The problem in hand is a case of multiple events detection. In this case, there may be multiple events in a single audio sample, making it a multi-label classification problem. In multi label classification tasks, we have samples which may belong to more than one class at the same time. It is different from multi class classification in a way that in multi class classification, one sample belongs to only 1 class at a time, but in case of multi label, we may have 1 or more labels for a single sample at the same time. That is why, in spite of sounding similar, these 2 are completely different tasks with different non linearity (softmax for multi class and sigmoid for multi label case) at the output later and also different loss functions (binary cross entropy for multi label and categorical cross entropy for multi class cases) for both of these classification methods. Multi label classification essentially maps the input vector x to binary vector y. The vector y has each element to be a binary number 0 or 1 with 0 denoting belonging to a particular class out of n classes of y and 1 denoting belonging to a particular class. So, we get a multi-hot vector in the form of y.

## 2 Literature Survey

In most of the research papers, models used deep neural networks (including recurrent neural networks and convolutional neural networks (ConvNet)), Gaussian mixture model, or random forests. Regarding feature extraction method, mel-spectrogram or mel-frequency cepstral coefficients are most frequently used methods. Although it is not easy to determine the optimal approach for the task but, the 'deep learning' approaches seem to have better performance than traditional approaches. Another way can be converting this problem to multi-class problem. I will discuss this in detail in method 5. Also, data augmentation could be a good approach as this dataset is highly uneven. We may can try generative modelling to generate new samples or simple augmentation to increase the dataset and get better results.

## 3 Methods Used:

The given dataset was 10000 samples which were already in the form of spectrograms, each having nmels = 64 and 1000 time frames. We also received additional dataset of 2000 files of same shape as validation data. For this study, I mixed both of these datasets into single file and then split training and validation dataset out of it with split factor of 1.67, leaving 9995 files for training and 2005 files for validation. Training dataset had shape (9995, 1,64,1000) and validation dataset was of shape(2005,1,64,1000), where 9995 or 2005 is the number of samples, 64 is the number of frequency values corresponding to each frame of spectrogram, 1000 is the number of frames in each sample and 1 represents greyscale image. As the shape of all the samples was equal, so no need was there for zero padding.



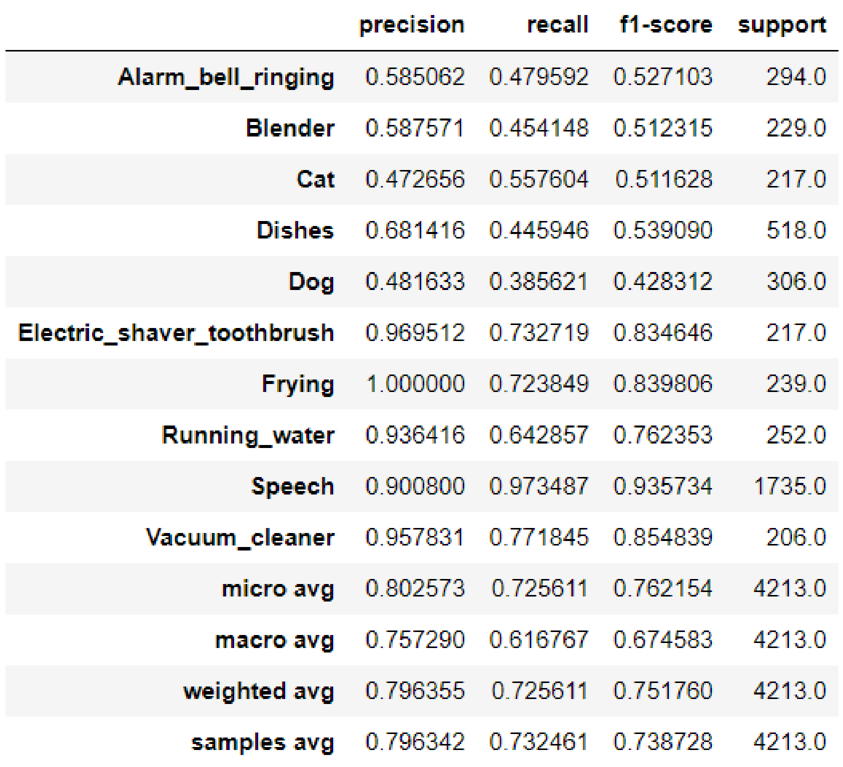
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 998, 32)       320
_________________________________________________________________
batch_normalization (BatchNo (None, 62, 998, 32)       128
_________________________________________________________________
dropout (Dropout)            (None, 62, 998, 32)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 996, 64)       18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 60, 996, 64)       256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 30, 498, 64)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 498, 64)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 496, 64)       36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 496, 64)       256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 248, 64)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 248, 64)       0
_________________________________________________________________
flatten (Flatten)            (None, 222208)            0
_________________________________________________________________
dense (Dense)                (None, 128)               28442752
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 28,508,554
Trainable params: 28,507,978
Non-trainable params: 576
```

Figure 1: Model Architecture for model 1

### 3.1 Method 1:

As my first model, I built a sequential CNN model with specifications as shown in the figure 1.

This model is supposed to work very well as CNN is good for images and the spectrograms here can be considered as images only. We feed the whole datasets with a batch size of 32 and trained for 40 epochs. The training loss decreased very quickly, validation loss also decreased gradually but was somewhat fluctuating. and a good overall precision of 0.80 and recall of 0.72 was achieved with precision and recall close to 1 for some lables as shown in the figure 2.



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.585062 | 0.479592 | 0.527103 | 294.0 |
| **Blender** | 0.587571 | 0.454148 | 0.512315 | 229.0 |
| **Cat** | 0.472656 | 0.557604 | 0.511628 | 217.0 |
| **Dishes** | 0.681416 | 0.445946 | 0.539090 | 518.0 |
| **Dog** | 0.481633 | 0.385621 | 0.428312 | 306.0 |
| **Electric_shaver_toothbrush** | 0.969512 | 0.732719 | 0.834646 | 217.0 |
| **Frying** | 1.000000 | 0.723849 | 0.839806 | 239.0 |
| **Running_water** | 0.936416 | 0.642857 | 0.762353 | 252.0 |
| **Speech** | 0.900800 | 0.973487 | 0.935734 | 1735.0 |
| **Vacuum_cleaner** | 0.957831 | 0.771845 | 0.854839 | 206.0 |
| **micro avg** | 0.802573 | 0.725611 | 0.762154 | 4213.0 |
| **macro avg** | 0.757290 | 0.616767 | 0.674583 | 4213.0 |
| **weighted avg** | 0.796355 | 0.725611 | 0.751760 | 4213.0 |
| **samples avg** | 0.796342 | 0.732461 | 0.738728 | 4213.0 |

Figure 2: Various Scores on Validation Set for model 1

As we can see there is a high support value for speech as compared to other classes. So data seems to be highly imbalanced which will effect our overall precision and recall for our model. Now, the model was tested on an unseen test data. As we can see, a precision of around 0.79 and a recall of around 0.66 was observed as in figure 3. These values are quite close to validation results which is a good thing. Precision and recall are really good for some clases but very bad for others. The f1score is great only for speech which has the highest support value. So it might be the reason of higher precision and recall despite of low values for some of the classes.

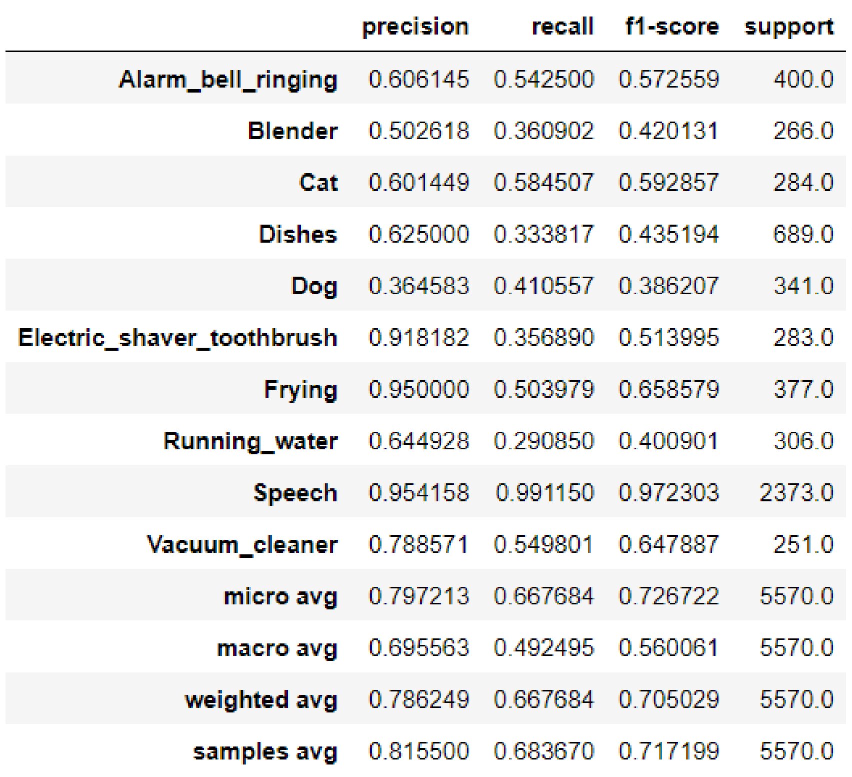
## 3.2 Method 2:

In this model, I used a simple dense neural network, with specifications as shown in the figure 4. This model did well for training data, but could not seem to converge and give good results for validation dataset. It showed a validation precision of just around 0.6 and recall of 0.46.

Its performance for validation data can be seen in the figure 5. But surprisingly, this model performed really well for the test data. If we see the test precision and recall as in figure 6, precision was about 0.79 and recall was about 0.67 with f1 score being 0.72. This kind of performance was totally unexpected from this model. I could not think of a reason for this behaviour of the results other than some examples in the dataset itself which seemed to work well for the model.

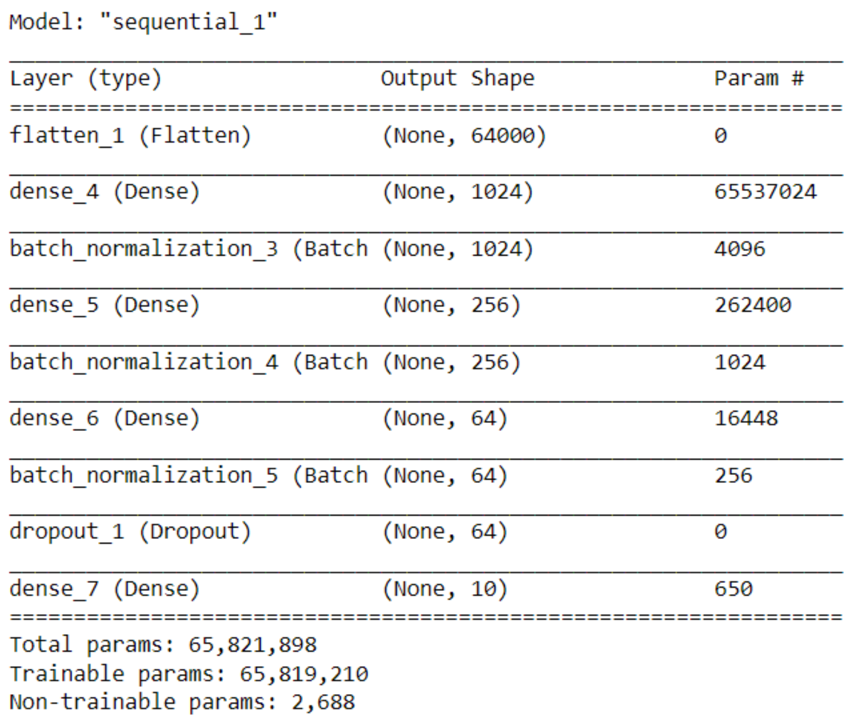
## 3.3 Method 3:

This time, I extracted mfcc features with n_mfcc = 32 from the mel spectrogram given, using a pre-defined function in librosa. Although mfcc features are supposed to work better for music data as these audio could be well beyond the hearing range also and mfcc is known to capture very high frequency sounds badly. We will see how it works if we apply a CNN network after extracting mfcc features from the input dataset. A model as shown in the figure 7 was made and trained using mfcc features. Here, the shape of input data needed to be reformed for it to extract mfcc features. So



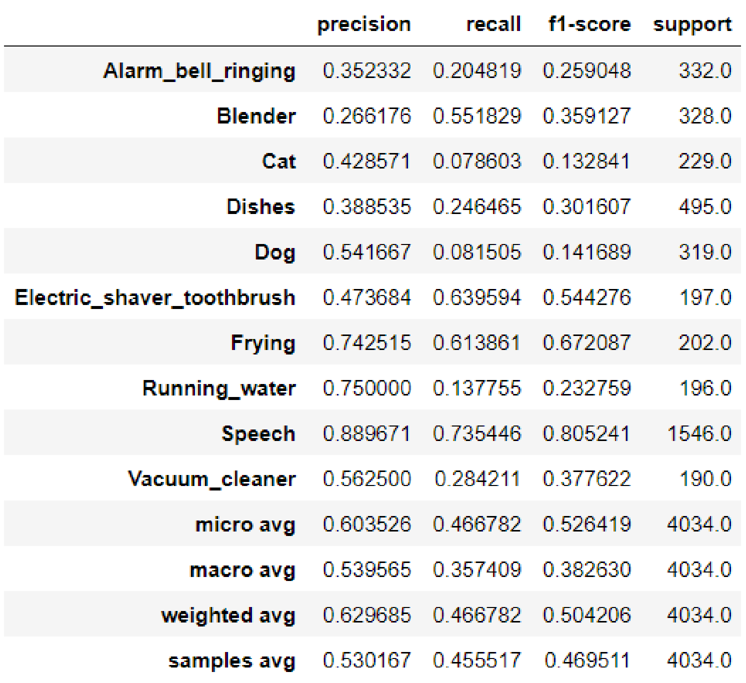
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.606145 | 0.542500 | 0.572559 | 400.0 |
| **Blender** | 0.502618 | 0.360902 | 0.420131 | 266.0 |
| **Cat** | 0.601449 | 0.584507 | 0.592857 | 284.0 |
| **Dishes** | 0.625000 | 0.333817 | 0.435194 | 689.0 |
| **Dog** | 0.364583 | 0.410557 | 0.386207 | 341.0 |
| **Electric_shaver_toothbrush** | 0.918182 | 0.356890 | 0.513995 | 283.0 |
| **Frying** | 0.950000 | 0.503979 | 0.658579 | 377.0 |
| **Running_water** | 0.644928 | 0.290850 | 0.400901 | 306.0 |
| **Speech** | 0.954158 | 0.991150 | 0.972303 | 2373.0 |
| **Vacuum_cleaner** | 0.788571 | 0.549801 | 0.647887 | 251.0 |
| **micro avg** | 0.797213 | 0.667684 | 0.726722 | 5570.0 |
| **macro avg** | 0.695563 | 0.492495 | 0.560061 | 5570.0 |
| **weighted avg** | 0.786249 | 0.667684 | 0.705029 | 5570.0 |
| **samples avg** | 0.815500 | 0.683670 | 0.717199 | 5570.0 |

Figure 3: Various Scores on Test Set for model 1



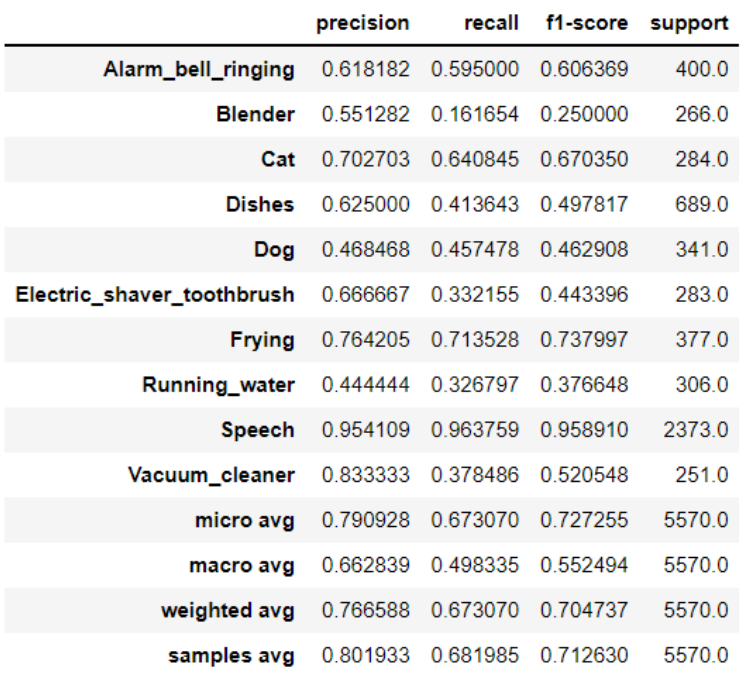
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 64000)             0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              65537024
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dense_5 (Dense)              (None, 256)               262400
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dense_6 (Dense)              (None, 64)                16448
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)                256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650
=================================================================
Total params: 65,821,898
Trainable params: 65,819,210
Non-trainable params: 2,688
```

Figure 4: Model Architecture for model 2



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.352332 | 0.204819 | 0.259048 | 332.0 |
| **Blender** | 0.266176 | 0.551829 | 0.359127 | 328.0 |
| **Cat** | 0.428571 | 0.078603 | 0.132841 | 229.0 |
| **Dishes** | 0.388535 | 0.246465 | 0.301607 | 495.0 |
| **Dog** | 0.541667 | 0.081505 | 0.141689 | 319.0 |
| **Electric_shaver_toothbrush** | 0.473684 | 0.639594 | 0.544276 | 197.0 |
| **Frying** | 0.742515 | 0.613861 | 0.672087 | 202.0 |
| **Running_water** | 0.750000 | 0.137755 | 0.232759 | 196.0 |
| **Speech** | 0.889671 | 0.735446 | 0.805241 | 1546.0 |
| **Vacuum_cleaner** | 0.562500 | 0.284211 | 0.377622 | 190.0 |
| **micro avg** | 0.603526 | 0.466782 | 0.526419 | 4034.0 |
| **macro avg** | 0.539565 | 0.357409 | 0.382630 | 4034.0 |
| **weighted avg** | 0.629685 | 0.466782 | 0.504206 | 4034.0 |
| **samples avg** | 0.530167 | 0.455517 | 0.469511 | 4034.0 |

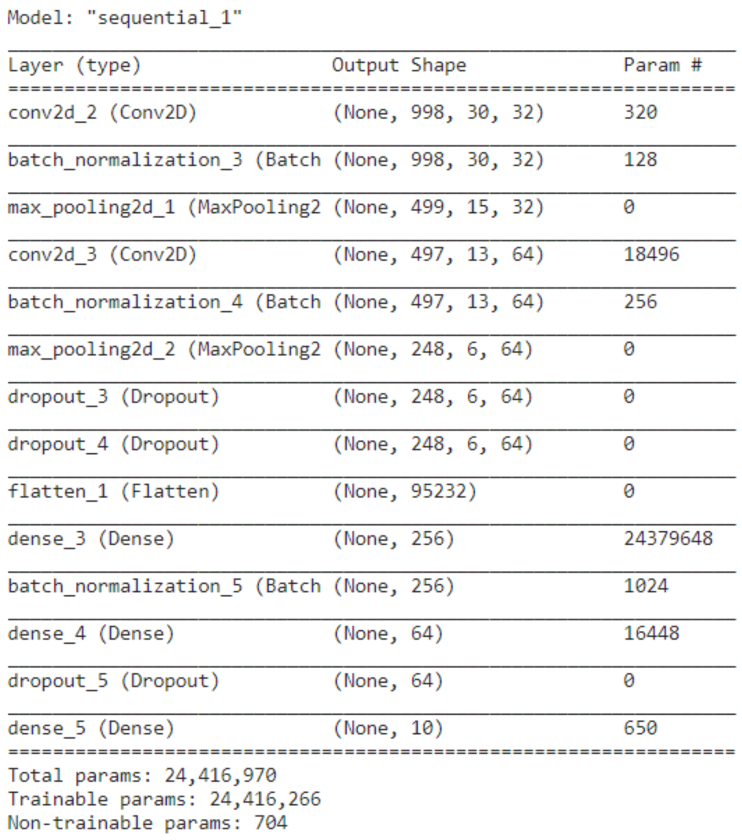
Figure 5: Various Scores on Validation Set for model 2



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.618182 | 0.595000 | 0.606369 | 400.0 |
| **Blender** | 0.551282 | 0.161654 | 0.250000 | 266.0 |
| **Cat** | 0.702703 | 0.640845 | 0.670350 | 284.0 |
| **Dishes** | 0.625000 | 0.413643 | 0.497817 | 689.0 |
| **Dog** | 0.468468 | 0.457478 | 0.462908 | 341.0 |
| **Electric_shaver_toothbrush** | 0.666667 | 0.332155 | 0.443396 | 283.0 |
| **Frying** | 0.764205 | 0.713528 | 0.737997 | 377.0 |
| **Running_water** | 0.444444 | 0.326797 | 0.376648 | 306.0 |
| **Speech** | 0.954109 | 0.963759 | 0.958910 | 2373.0 |
| **Vacuum_cleaner** | 0.833333 | 0.378486 | 0.520548 | 251.0 |
| **micro avg** | 0.790928 | 0.673070 | 0.727255 | 5570.0 |
| **macro avg** | 0.662839 | 0.498335 | 0.552494 | 5570.0 |
| **weighted avg** | 0.766588 | 0.673070 | 0.704737 | 5570.0 |
| **samples avg** | 0.801933 | 0.681985 | 0.712630 | 5570.0 |

Figure 6: Various Scores on Test Set for model 2

after extraction of features, the shape of each sample was made to be (1000,32,1) i.e. the features were extracted along the frequency dimension for each time frame. This method got a good training performance, but the validation performance was not up to the mark as the model seemed to have overfit. Validation precision is reported to be 0.78 and recall being 0.61 which is not that good or bad value as shown in figure 8.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 998, 30, 32)       320       
_________________________________________________________________
batch_normalization_3 (Batch (None, 998, 30, 32)       128       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 499, 15, 32)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 497, 13, 64)       18496     
_________________________________________________________________
batch_normalization_4 (Batch (None, 497, 13, 64)       256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 248, 6, 64)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 248, 6, 64)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 248, 6, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 95232)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 256)               24379648  
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024      
_________________________________________________________________
dense_4 (Dense)              (None, 64)                16448     
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650       
=================================================================
Total params: 24,416,970
Trainable params: 24,416,266
Non-trainable params: 704
```

Figure 7: Model Architecture for model 3

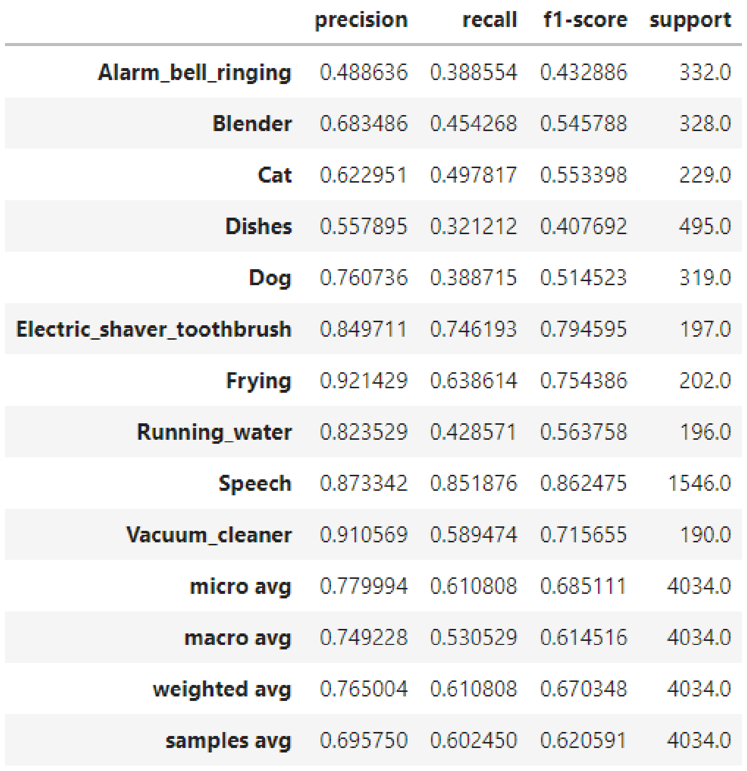
If we have a look at the test dataset performance as in figure 9, we got precision to be around 0.81 and recall being 0.67. So, again opposite to our belief, this model performed better for test dataset than for the validation dataset. This means that the model did not overfit and was trained well for validation dataset. That is why test data performance was similar.

### 3.4 Method 4:

This time we had a simple LSTM model having 2 layers of LSTM followed by dense layer. The model is shown in figure 10. This model was trained for 90 epochs and seemed to converge very well as training loss and validation loss were decreasing continuously without any unnecessary spikes in the validation loss. The classification report for Validation dataset can be seen in figure 11. With a precision of 0.83 and recall of 0.75 and hence the f1 score of 0.79, this model was supposed to work best out of all these models. But when we run it on test data, the classification report came out as shown in figure 12. The model did not work well for a few classes, but for others its performance was very good. The performance was bad mainly for first 3 classes with the overall precision of 0.77, recall of 0.67 and hence f1 score of 0.72. These results are not bad but very weak results of first 3 classes is still something that needs to be answered. I feel like model lacked a bit of non-linearity, adding more layers might have made the training slower, but could have worked better for the model.

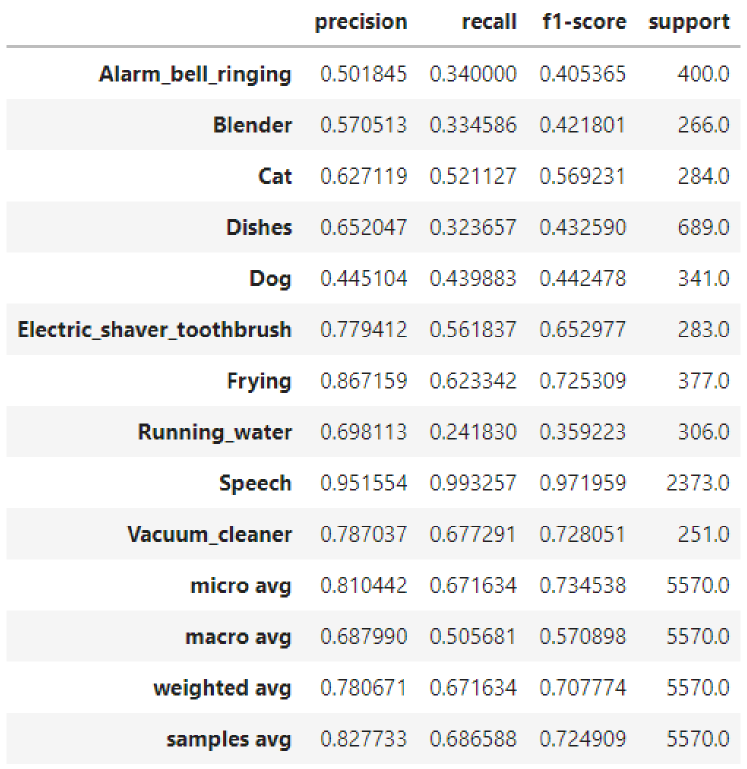
### 3.5 Method 5:

Now this time around, I tried to implement an out of the box method i.e. converting a multi label problem into multi class classification problem. I did the same by treating the multi hot vectors having same classes as a separate class. e.g., if we take the case of 2 classes, then 00, 01, 10 and 11 would make 4 classes and the output layer can have 4 neurons with softmax applied and input



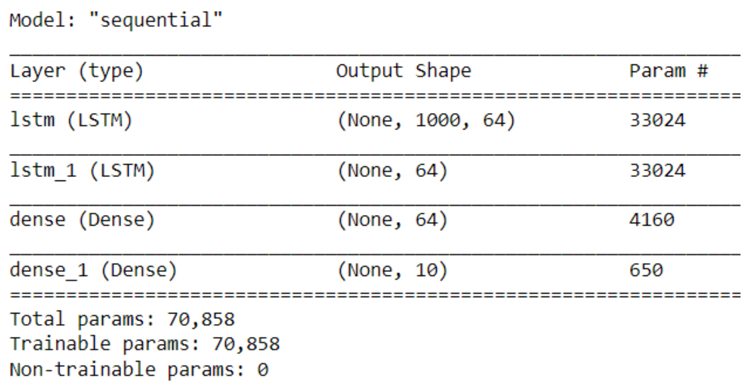
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.488636 | 0.388554 | 0.432886 | 332.0 |
| **Blender** | 0.683486 | 0.454268 | 0.545788 | 328.0 |
| **Cat** | 0.622951 | 0.497817 | 0.553398 | 229.0 |
| **Dishes** | 0.557895 | 0.321212 | 0.407692 | 495.0 |
| **Dog** | 0.760736 | 0.388715 | 0.514523 | 319.0 |
| **Electric_shaver_toothbrush** | 0.849711 | 0.746193 | 0.794595 | 197.0 |
| **Frying** | 0.921429 | 0.638614 | 0.754386 | 202.0 |
| **Running_water** | 0.823529 | 0.428571 | 0.563758 | 196.0 |
| **Speech** | 0.873342 | 0.851876 | 0.862475 | 1546.0 |
| **Vacuum_cleaner** | 0.910569 | 0.589474 | 0.715655 | 190.0 |
| **micro avg** | 0.779994 | 0.610808 | 0.685111 | 4034.0 |
| **macro avg** | 0.749228 | 0.530529 | 0.614516 | 4034.0 |
| **weighted avg** | 0.765004 | 0.610808 | 0.670348 | 4034.0 |
| **samples avg** | 0.695750 | 0.602450 | 0.620591 | 4034.0 |

Figure 8: Various Scores on Validation Set for model 3



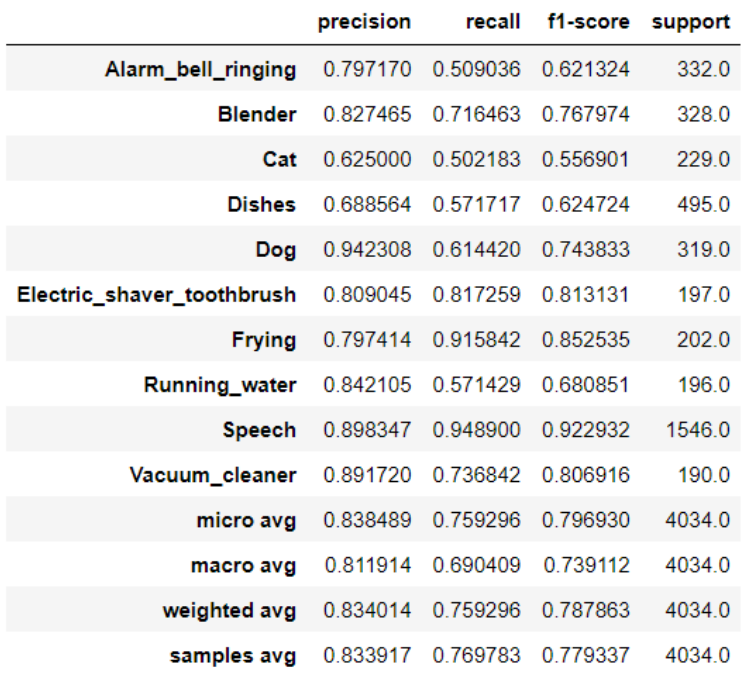
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.501845 | 0.340000 | 0.405365 | 400.0 |
| **Blender** | 0.570513 | 0.334586 | 0.421801 | 266.0 |
| **Cat** | 0.627119 | 0.521127 | 0.569231 | 284.0 |
| **Dishes** | 0.652047 | 0.323657 | 0.432590 | 689.0 |
| **Dog** | 0.445104 | 0.439883 | 0.442478 | 341.0 |
| **Electric_shaver_toothbrush** | 0.779412 | 0.561837 | 0.652977 | 283.0 |
| **Frying** | 0.867159 | 0.623342 | 0.725309 | 377.0 |
| **Running_water** | 0.698113 | 0.241830 | 0.359223 | 306.0 |
| **Speech** | 0.951554 | 0.993257 | 0.971959 | 2373.0 |
| **Vacuum_cleaner** | 0.787037 | 0.677291 | 0.728051 | 251.0 |
| **micro avg** | 0.810442 | 0.671634 | 0.734538 | 5570.0 |
| **macro avg** | 0.687990 | 0.505681 | 0.570898 | 5570.0 |
| **weighted avg** | 0.780671 | 0.671634 | 0.707774 | 5570.0 |
| **samples avg** | 0.827733 | 0.686588 | 0.724909 | 5570.0 |

Figure 9: Various Scores on Test Set for model 3



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1000, 64)          33024
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 70,858
Trainable params: 70,858
Non-trainable params: 0
```

Figure 10: Model Architecture for model 4



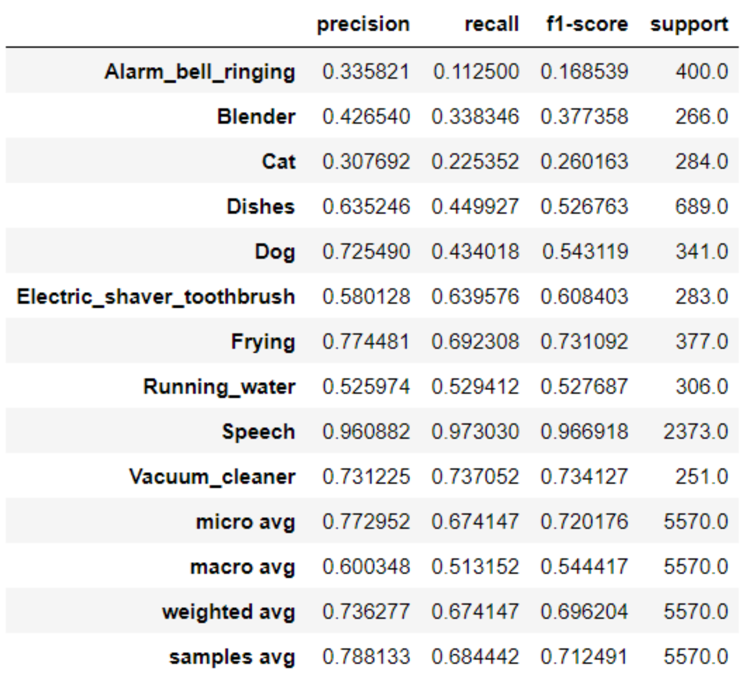
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.797170 | 0.509036 | 0.621324 | 332.0 |
| **Blender** | 0.827465 | 0.716463 | 0.767974 | 328.0 |
| **Cat** | 0.625000 | 0.502183 | 0.556901 | 229.0 |
| **Dishes** | 0.688564 | 0.571717 | 0.624724 | 495.0 |
| **Dog** | 0.942308 | 0.614420 | 0.743833 | 319.0 |
| **Electric_shaver_toothbrush** | 0.809045 | 0.817259 | 0.813131 | 197.0 |
| **Frying** | 0.797414 | 0.915842 | 0.852535 | 202.0 |
| **Running_water** | 0.842105 | 0.571429 | 0.680851 | 196.0 |
| **Speech** | 0.898347 | 0.948900 | 0.922932 | 1546.0 |
| **Vacuum_cleaner** | 0.891720 | 0.736842 | 0.806916 | 190.0 |
| **micro avg** | 0.838489 | 0.759296 | 0.796930 | 4034.0 |
| **macro avg** | 0.811914 | 0.690409 | 0.739112 | 4034.0 |
| **weighted avg** | 0.834014 | 0.759296 | 0.787863 | 4034.0 |
| **samples avg** | 0.833917 | 0.769783 | 0.779337 | 4034.0 |

Figure 11: Various Scores on Validation Set for model 4

training can also be done using 4 one hot vectors. But, as this was a 10 class problem and there was a huge variability in the lables extracted. There were more than 139 different classes for training dataset, making each one hot vector of dimensions (139,1) also, validation dataset had 79 classes. The figure 13 shows a printed list of number of classes for each one hot vector to train and validate. As we can see, many of these classes have just 1 sample belonging to them. So data would be insufficient to train even if we augment new data samples. So I dropped this technique and did not make any model using this. But this technique can work out well if labels are less and there is sufficient data to do so.

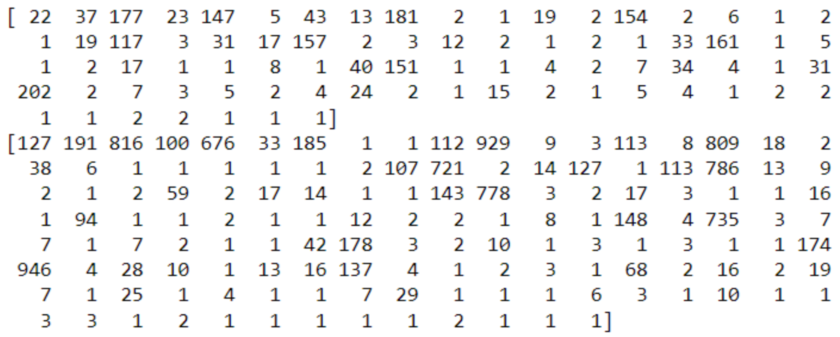
## 4 Observations and Discussion:

We observe that all these models seemed to work well and had comparable performance. The LSTM model was supposed to do a little bit better looking at the kind of training and the way the training and validation loss converged. In future, for this kind of projects, we can consider data augmentation to increase the number of samples appropriately and it should result in better performance. We can



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Alarm_bell_ringing** | 0.335821 | 0.112500 | 0.168539 | 400.0 |
| **Blender** | 0.426540 | 0.338346 | 0.377358 | 266.0 |
| **Cat** | 0.307692 | 0.225352 | 0.260163 | 284.0 |
| **Dishes** | 0.635246 | 0.449927 | 0.526763 | 689.0 |
| **Dog** | 0.725490 | 0.434018 | 0.543119 | 341.0 |
| **Electric_shaver_toothbrush** | 0.580128 | 0.639576 | 0.608403 | 283.0 |
| **Frying** | 0.774481 | 0.692308 | 0.731092 | 377.0 |
| **Running_water** | 0.525974 | 0.529412 | 0.527687 | 306.0 |
| **Speech** | 0.960882 | 0.973030 | 0.966918 | 2373.0 |
| **Vacuum_cleaner** | 0.731225 | 0.737052 | 0.734127 | 251.0 |
| **micro avg** | 0.772952 | 0.674147 | 0.720176 | 5570.0 |
| **macro avg** | 0.600348 | 0.513152 | 0.544417 | 5570.0 |
| **weighted avg** | 0.736277 | 0.674147 | 0.696204 | 5570.0 |
| **samples avg** | 0.788133 | 0.684442 | 0.712491 | 5570.0 |

Figure 12:  Various Scores on Test Set for model 4



```
[ 22  37 177  23 147   5  43  13 181   2   1  19   2 154   2   6   1   2
   1  19 117   3  31  17 157   2   3  12   2   1   2   1  33 161   1   5
   1   2  17   1   1   8   1  40 151   1   1   4   2   7  34   4   1  31
 202   2   7   3   5   2   4  24   2   1  15   2   1   5   4   1   2   2
   1   1   2   2   1   1   1]
[127 191 816 100 676  33 185   1   1 112 929   9   3 113   8 809  18   2
  38   6   1   1   1   1   1   2 107 721   2  14 127   1 113 786  13   9
   2   1   2  59   2  17  14   1   1 143 778   3   2  17   3   1   1  16
   1  94   1   1   2   1   1  12   2   2   1   8   1 148   4 735   3   7
   7   1   7   2   1   1  42 178   3   2  10   1   3   1   3   1   1 174
 946   4  28  10   1  13  16 137   4   1   2   3   1  68   2  16   2  19
   7   1  25   1   4   1   1   7  29   1   1   1   6   3   1  10   1   1
   3   3   1   2   1   1   1   1   1   2   1   1   1]
```

Figure 13:  Various Scores on Test Set for model 4

also test LSTM even further by increasing number of layers and including more non-linearity. We observed that the results also depends upon the data samples somewhat. Some data samples like speech had good results for all the models. One reason being there were a lot more number of samples for training them. We can also try giving proper weights to each sample depending upon how much they occur. Also, mfcc did work well with the data. We can also try extracting different kinds of features for the same problem. Also, for audio classification, CNN along with LSTM and RNN can be tried as a future project, which preserves sequential information also, which can be useful in case of audio data or spectrograms in this case. Mostly we may have to use 1d CNN followed by LSTM, GRU or RNN layers.