EE603: Assignment 2 - Multiple Event Detection

Somya Gupta
somyagupta20@iitk.ac.in
200993

Abstract

The assignment focuses on detecting the sound (multi-label classification) given the numpy arrays as the input. We use various methods such as Convolutional Neural Networks(CNN), Recurrent Neural Networks(RNN) and a combination of the two to classify the test set into appropriate sound labels. We use average f1-score as a means to determine the best model for classification. The results we have obtained show us that CRNN is the best option for constructing the classifier with the given training and validation set.

1 Introduction

Multi-label classification is a generalization of multi-class classification, such that here, a single sample can have several classes associated with it. Several problem transformation methods exist for multi-label classification: converting it into several separate multi-class problems or making it into binary classification, multi-label k-nearest neighbors algorithm (Mlknn) etc. Classification is the basis of many applications, such as text categorization, image classification, automatic annotation for multimedia content, tag recommendation etc.

2 Literature Survey

RNN algorithms process sequences by retaining the memory of the previous value or state in the sequence. RNNs are used for speech recognition, time series prediction etc. While RNNs are suitable for handling temporal or sequential data, CNNs are suitable for handling spatial data (images). The CNN is made up of three types of layers: convolutional layers, pooling layers, and fully-connected (FC) layers. The convolution layer is responsible for the extraction of the different features from the input images. If we try to compare the two, we can say that CNN is more powerful than RNN. That's mainly because RNN has less feature compatibility and it has the ability to take arbitrary output/input lengths which can affect the total computational time and efficiency. On the other hand, CNN takes fixed input and gives a fixed output which allows it to compute the results at a faster pace. The CRNN (convolutional recurrent neural network) involves CNN(convolutional neural network) followed by the RNN(Recurrent neural networks).

3 Method

3.1 Data Pre-processing

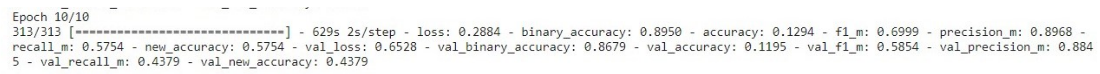
- Importing libraries
- Importing the datasets : Extracting the input data and output labels
- We notice that the given data set has all the numpy arrays (melspec files) have a shape of (1,64,1000) with the label matrices of shape (11,1000)

- We then convert the given eventroll matrices to a multi-hot vector using the given python function and assign the reverse-mapping of labels to indexes (after removing the silence class, we have 10 classes).
- We also assign class weights to each of the 10 classes in order to deal with the data imbalance.
- Reshape the data : We make a stack of the train and validation data and reshape it so as to easily process the model

3.2 Training the model

- **Convolutional Neural Networks (CNN)** : We have used 5 layers of Conv2D and 2 of MaxPooling2D and added some dropout layers to minimise overfitting loss. We have also used BatchNormalization (to make training of artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling). We have then flattened the array and added dense layers with activation functions Relu and sigmoid (for multi-label classification). We have monitored binary cross entropy loss using an adam optimiser.

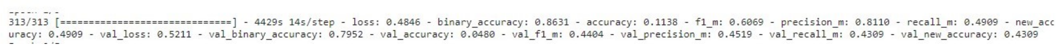
  Obtained training f1-score=(0.69)
  Obtained validation f1-score=(0.58)

```
Epoch 10/10
313/313 [==============================] - 629s 2s/step - loss: 0.2884 - binary_accuracy: 0.8950 - accuracy: 0.1294 - f1_m: 0.6999 - precision_m: 0.8968 -
recall_m: 0.5754 - new_accuracy: 0.5754 - val_loss: 0.6528 - val_binary_accuracy: 0.8679 - val_accuracy: 0.1195 - val_f1_m: 0.5854 - val_precision_m: 0.884
5 - val_recall_m: 0.4379 - val_new_accuracy: 0.4379
```

- **Recurrent Neural Networks (RNN)** : We have used Time Distributed LSTMs(long short-term memory networks) RNN layers along with batch normalization and 2 dense layers. LSTMs are capable of learning long-term dependencies, especially in sequence prediction problems. Here also, we have used BatchNormalization and then flattened the array and used dense layers with activation functions relu and sigmoid. We have again monitored binary cross entropy loss using an adam optimiser.

  Obtained training f1-score=(0.60)
  Obtained validation f1-score=(0.44)

```
313/313 [==============================] - 4429s 14s/step - loss: 0.4846 - binary_accuracy: 0.8631 - accuracy: 0.1138 - f1_m: 0.6069 - precision_m: 0.8110 - recall_m: 0.4909 - new_acc
uracy: 0.4909 - val_loss: 0.5211 - val_binary_accuracy: 0.7952 - val_accuracy: 0.0480 - val_f1_m: 0.4404 - val_precision_m: 0.4519 - val_recall_m: 0.4309 - val_new_accuracy: 0.4309
```

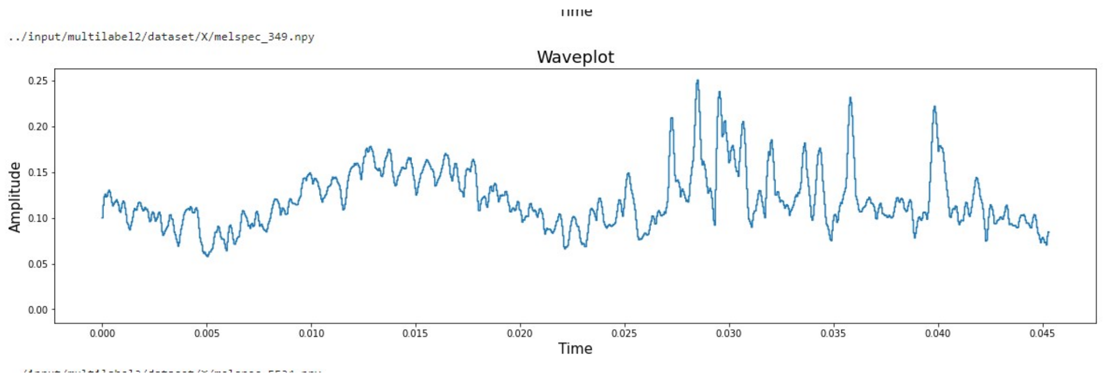
- **Label Powerset** : Label Powerset is a problem transformation approach to multi-label classification that transforms a multi-label problem to a multi-class problem with 1 multi-class classifier trained on all unique label combinations found in the training data.

  Obtained training f1-score=(0.48)
  Obtained validation f1-score=(0.39)

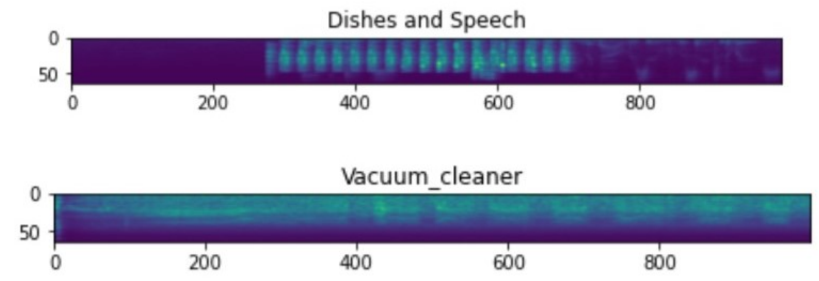
3.3 Data visualisation

I have plotted waveplots for some of the given files to better visualise the sound data given. Along with this, I have also plotted spectrograms containing the data given to us.

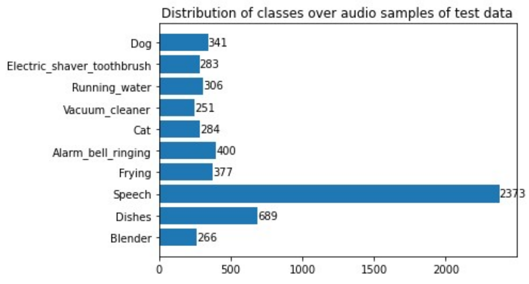
Eg. The following depicts one of the 'Dog' sound:

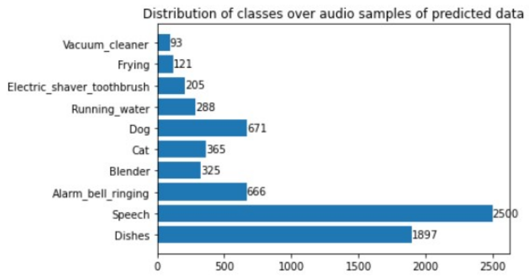


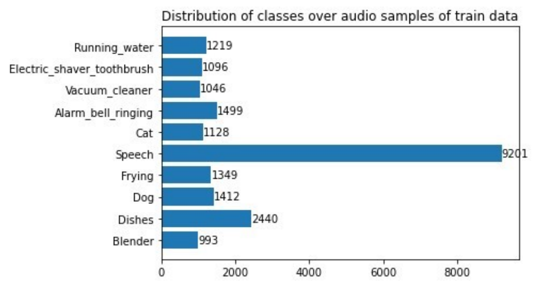
We can observe an intricate pattern of the spectrogram having dishes and speech sound while a uniform and simple pattern of the spectrogram showing vacuum cleaner sound.

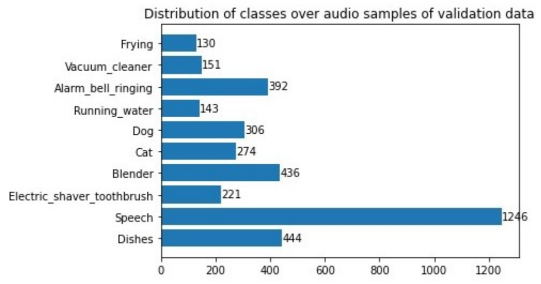


We can also visualise the data imbalance in the train and validation data using the plots showing number of labels of each type in the given data. The given training and validation data was hugely imbalanced with 'Speech' appearing the maximum number of times. Attached you can see the data distribution for the train, validation and test set. To deal with the data imbalance, there are several methods such as undersampling, oversampling, deciding on separate thresholds for each class etc. One such way is using 'class weights' to assign each class/label a certain weight which is then used to classify the data accordingly.
In the test and the predicted data, the numbers look like this:





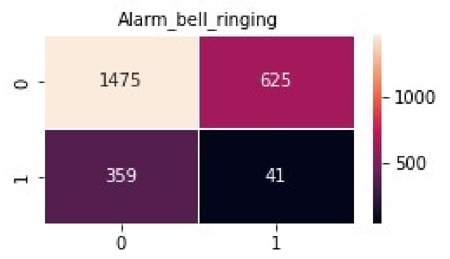


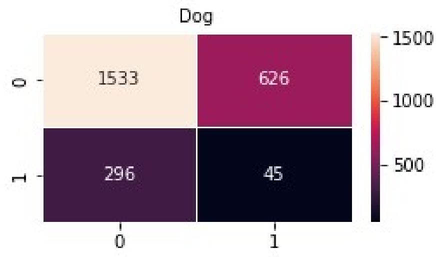


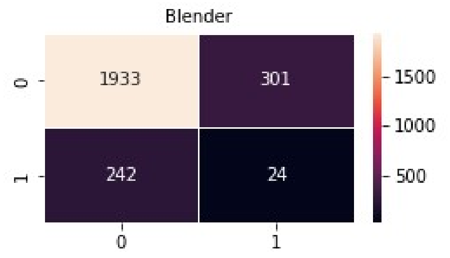
4 Measures of Result Relevancy

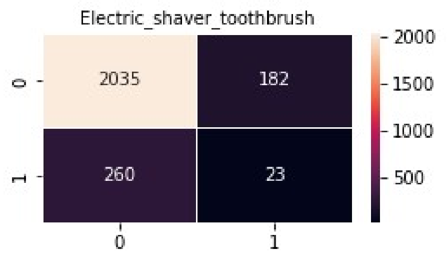
4.1 Confusion matrix

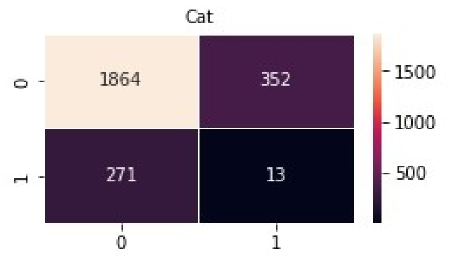
Table 3: For CNN The confusion matrices per label:

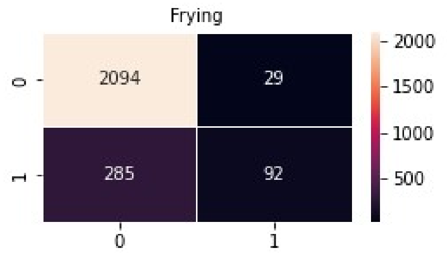


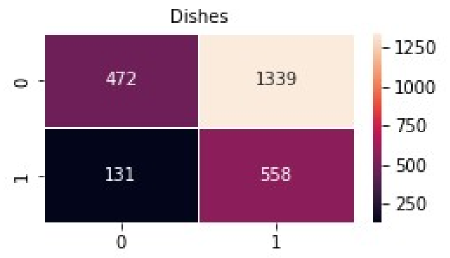


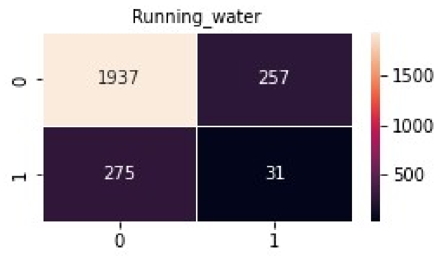


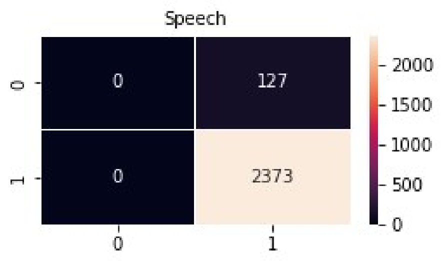












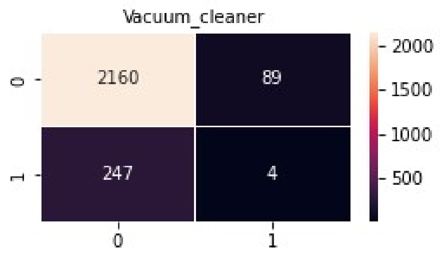
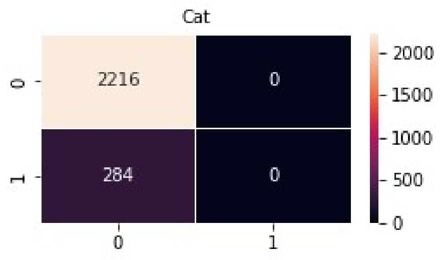
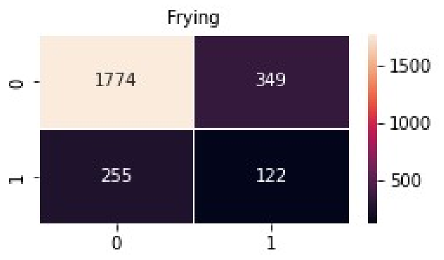
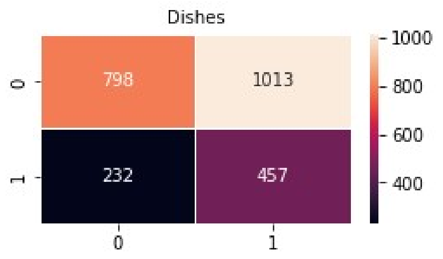
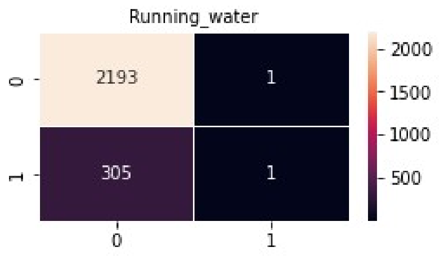


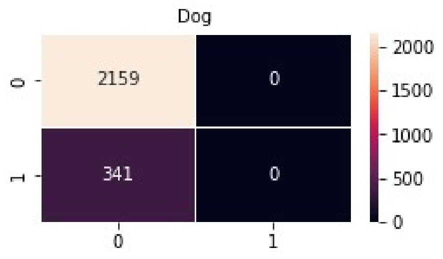
Table 3: For RNN Now for RNN, the following are the confusion matrices per label:

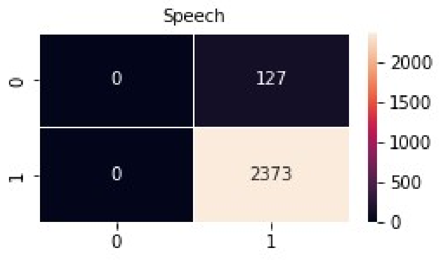


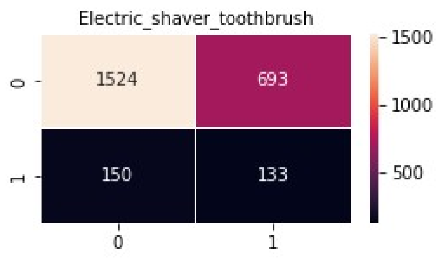


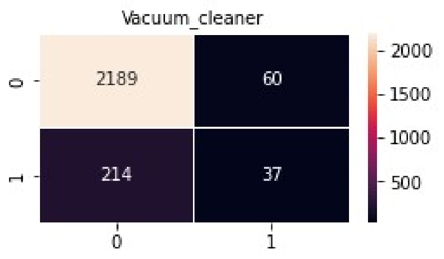












Now, moving forward we use the samples and micro averages of precision, recall and f1-scores. "Samples" says the function to compute f1 for each instance, and returns the average. Use it for multilabel classification. "Micro" says the function to compute f1 by considering total true positives, false negatives and false positives (no matter of the prediction for each label in the dataset) and is the best for imbalanced data.

4.2 Precision

$P = \frac{TP}{TP+FP}$

- For CNN:
  Micro avg: 0.60
  Samples avg: 0.45
- For RNN:
  Micro avg: 0.66
  Samples avg: 0.58
- For Label Powerset:
  Micro avg: 0.53
  Samples avg: 0.53

4.3 Recall

$R = \frac{TP}{TP+FN}$

- For CNN:
  Micro avg: 0.56
  Samples avg: 0.58
- For RNN:
  Micro avg: 0.56
  Samples avg: 0.56
- For Label Powerset:
  Micro avg: 0.50
  Samples avg: 0.51

4.4 F1-score

$F1 = 2 * \frac{P*R}{P+R}$

- For CNN:
  Micro avg: 0.53
  Samples avg: 0.50
- For RNN:
  Micro avg: 0.57
  Samples avg: 0.57
- For Label Powerset:
  Micro avg: 0.51
  Samples avg: 0.51

4.5 Accuracy

Since we're dealing with multilabel classification, this will not be a good measure of evaluation since it returns a true positive only if for a particular npy array, the model has predicted all the labels correctly. Here accuracy=exact match ratio (ignores partially correct and extends the accuracy used in single-label case for multi-label prediction. Clearly, a disadvantage of this measure is that it does not distinguish between completely incorrect and partially correct which might be considered harsh.)
$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

- For CNN: 0.10
- For RNN: 0.8
- For Label Powerset: 0.8

Binary Accuracy calculates the percentage of predicted values that match with actual values for binary labels but this is also a bad measure since most of our data has 0's and here we count every matching 0 as well, thus returning a high binary accuracy.

- For CNN: 0.84
- For RNN: 0.80

4.6 Other evaluation metrics

I used a few other ways to judge my model, including hamming loss, new definition of accuracy and precision.

- For CNN:
  Alternate accuracy, threshold:0.2 = (0.44)
  Hamming loss, threshold:0.2=(0.14)
  Alternate accuracy, threshold:0.15 = (0.40)
  Hamming loss, threshold:0.15=(0.24)
- For RNN:
  Alternate accuracy, threshold:0.45 = (0.43)
  Hamming loss, threshold:0.45=(0.18)
  Precision, threshold:0.45=(0.56)

5 Observation

- Data imbalance The data had a huge imbalance with the number of samples having 'speech' label maximum. Out of the many ways to handle this, we used class weights to balance the data. The above-attached graphs show our observations as well.
- I observed that the sound event patterns are frequency-dependent: the same time-frequency pattern sounds different on different frequency regions but almost same when shifted only along the time axis.
- Construction of CNN: If we use MaxPooling with kernel size 2, we will get lower accuracy and f1-score. Using a larger kernel gives better results.
- Label Powerset method is strong and converts our problem into multi-class classification but is not the optimal way of dealing with given problem.
- Sound event classes: alarm-bell-ringing, cat, dishes, dog, electric shaver/toothbrush, running water and speech, these classes are non-stationary sound events, that is they keep changing along the time axis, thus resulting in intricate time-frequency patterns as can be observed in the attached melspectrograms.
- Sound events such as blender, frying, and vacuum cleaner are relatively almost stationary over time thus resulting in simple time-frequency patterns (can be seen in the melspectrograms)
- Hamming loss is a good measure of predicting the result relevancy of the model. for multi-label classification.

6 Conclusion

- Conventional 2D convolution takes into account the translational movement along the time and frequency axis of audios but doesn't work with the frequency-dependent patterns of audios.
- For multilabel classification, accuracy and binary accuracy are a bad measure of evaluation as described earlier. We can use averages(micro or samples) of the f1-score or define our own metrics for measuring result relevancy.

- CNN and RNN both can work on our multi-label dataset, providing almost similar results in terms of f1-score and hamming loss. A combination of the two takes up a lot of space and time but would be ideal.
- We can conclude that a mixture of **Convolutional Neural Networks(CNN)** and **Recurrent Neural Networks(CNN)** is the best model to classify the given data. It gives the best averaged f1-score even on the test data.