Multiple Audio Event Detection/ Audio Tagging

Happy Pachori
Dept. of Electrical Engineering
Indian Institute of Technology Kanpur, Uttar Pradesh
happyp20@iitk.ac.in

Abstract

Multiple Audio Event Detection/ Audio Tagging is one of the most widely used applications in Audio Deep Learning. It involves learning to classify sounds and to tag the category of that sounds present in that particular clip or datafile. This type of problem can be applied to many practical scenarios e.g. classifying and tagging music clips to identify the events happening in domestic household. In this paper, we conduct application so as to understand the approach used to solve such audio tagging problems. We take advantage of the robust machine learning techniques developed for image classification and apply them on the sound recognition problem. Audio data from the EE603 course assignment 1 Audio-Classification-MLSP train file is converted to a MEL spectrogram which compute dB relative to peak power. We test and compare two approaches using deep convolutional neural networks (CNNs). Artificial Neural Network (ANN), RNN and CRNN using our self-developed architecture, we achieve F1 score, precision, recall, accuracy and weight of our trained model

1 Introduction

Humans are able to identify the occurrences of our near environment using only acoustic information, namely, by hearing the sounds that are produced by those occurrences. For instance, it is sufficient to hear the knocking of a door to understand the underlying event and act in consequence. In this case, the knock on the door would be an example of a sound event. The task that aims to automatize the localization of sound events in time and their classification is called Sound Event Detection (SED), and it is currently a relevant field of research in machine learning and signal processing. Over the last decade, several datasets were released with the objective of developing and evaluating SED systems. Urbansound [1] was proposed as an ontology and a sound dataset containing recordings of 10 event categories of urban outdoors environments.

Sound is one of the five primary senses humans use to understand the world around them. Many of today's autonomous systems are predominantly vision based [1], and do not take advantage of the additional environment information provided from audio. Developing intelligence that is able to process both image and audio data concurrently would provide autonomous systems a deeper understanding of their environment and allow them to interact with their surroundings in a more meaningful way. We look to extract these properties and utilize them for recognizing certain sounds. Audio signals are inherently one-dimensional (amplitude over time), we look to transform these signals into a more descriptive representation that better illustrates the previously mentioned quantities. The train files given to us is already in MEL scale we only compute dB relative to peak power of this data to convert the intensity as human perceive them. Mel spectrogram, a transformation that details the frequency composition of the signal over time [3]. This allows us to make use of well-researched image classification techniques. The convolution neural network (CNN) is a powerful deep learning model that can learn a feature hierarchy for images. Since we are interested in predicting 10 different categories of sound, our model must be able to learn a high number of features in order to recognize specific sounds. In this study, we focus on two approaches to sound recognition using CNNs. In the first, we construct our own CNN architecture and fully train all the layers using our dataset. Secondly we also make model based on CRNN, model based on Artificial Neural Network (ANN) and RNN. The rest of the paper is organized as follows: Literature survey, My method (details), Precision, Recall, F1 score, confusion matrix (tables) Observations and discussion.

2 Literature survey

To complete this problem I have refereed to some websites like **stackoverflow**, **medium** and **towardsdatascience** to understand the concept of CRNN model architecture, LSTM and gru layer. Used papers of Dcase task 4 to undrstand the problem and BInarycrossentropy.

3 My method

In this section we present our two approaches to the audio classification problem. The different model architectures and components are discussed in detail. In my model we are monitoring validation accuracy.

3.1 Prepossessing

In this I make a numpy array of x and y of test and validation data given into a one numpy array file. Silent classes of the data is removed by using the below given code.

```
def eventrolllto_multihot_vector(eventroll):
""" Parameters
———-
eventroll : np.array
Eventroll matrix of shape=(11, 1000).

Returns
——-
np.array
A multihot vector of shape=(10,)
"""

findout active events:
active _events = (eventroll.sum(axis=1) >= 0.5).astype('float')

remove silence class:
return np.delete(active _events, 8)
```

3.2 CNN model

CNN model is one of the most accurate and very successful in various tasks due to its unique layers. It is mainly made by convolution layer and pooling layers. Reason to choose CNN is it is spatial invariant features and using a relatively small number of parameters.

Firstly we remove the silence class from the dataset given to us using. Then the target value is one hot encoded.

2D Convolutional Layer (Conv2D) unit is the portion that learns the translation invariant spatial patterns and their spatial hierarchies.

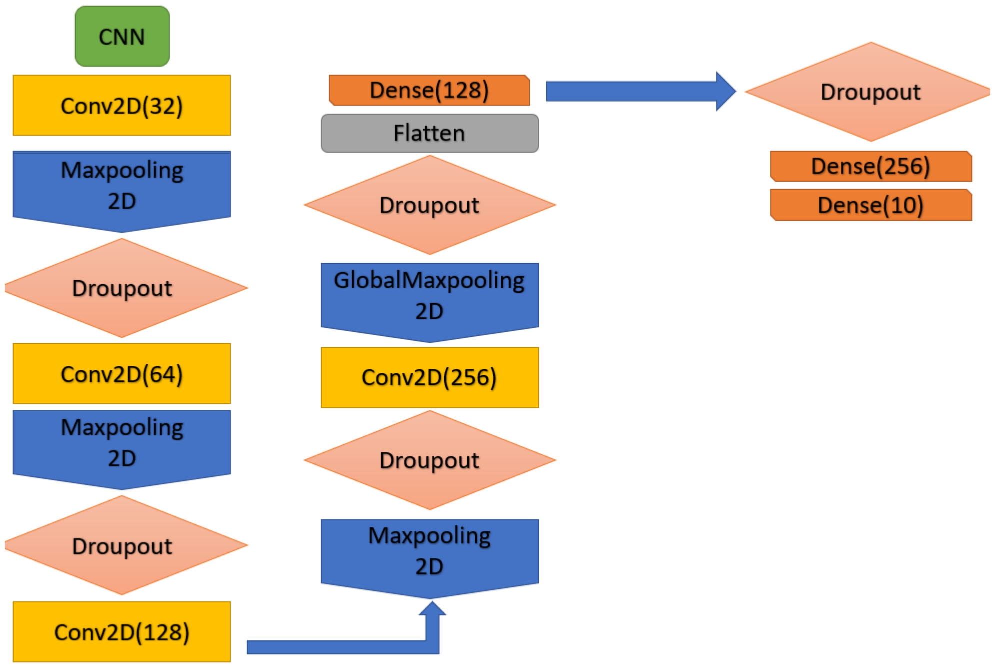
Max Pooling Layer halves the size of the feature maps by downsampling them to the max value inside a window. We use Downsampling because otherwise it would result in a ginormous number of parameters and your computer would blow up and after all that the model would massively overfit the data. This magical layer is the reason that a CNN can handle the huge amounts of data in images. Max Pooling does a model good.

Dropout Layer guards against overfitting by randomly setting the weights of a portion of the data to zero, and the Dense units contain hidden layers tied to the degrees of freedom the model has to try and fit the data. The more complex the data, the more degrees of freedom the model needs. Take care not to add a bunch of these and end up overfitting the data.

Flatten Layer squishes all the feature map information into a single column in order to feed in into a Dense layer, the last of which outputs the 10 species that the model is supposed to classify the audio recordings into.

We also add **Activation function** Here, the **Relu** function is used, which zeros out negative weights. You can read about other activation functions here, but this is a good one to start with. The last Dense layer's activation function type is **sigmoid**, which outputs a probability for class.

The **Adma** optimizer manages the learning rate for you, the loss function is used to evaluate how different the predicted and actual data are and penalizes the model for poor predictions. In this example, the loss function is **BinaryCrossentropy**, Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.



General CNN model architecture.

Implementation of CNN in Tensorflow

$CNNmodel = models.Sequential()$
$CNNmodel.add(layers.Conv2D(32, (3, 3), activation =' relu', inputshape = inputshape))$
$CNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CNNmodel.add(layers.Dropout(0.2))$
$CNNmodel.add(layers.Conv2D(64, (3, 3), activation =' relu'))$
$CNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CNNmodel.add(layers.Dropout(0.2))$
$CNNmodel.add(layers.Conv2D(128, (3, 3), activation =' relu')$
$CNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CNNmodel.add(layers.Dropout(0.2))$
$CNNmodel.add(layers.Conv2D(256, (3, 3), activation =' relu'))$
$CNNmodel.add(layers.GlobalMaxPooling2D())$
$CNNmodel.add(layers.Dropout(0.2))$
$CNNmodel.add(layers.Flatten())$

$CNNmodel.add(layers.Dense(256, activation =' relu')$
$CNNmodel.add(layers.Dense(10, activation =' sigmoid'))$

3.2.1 ANN model

The number of classes is 10, which is our output shape(number of classes), and we will create ANN with 4 dense layers and architecture The first layer has 256 neurons. Input shape according to the number of features with activation function as Relu, and to avoid any overfitting, we'll use the Dropout layer at a rate of 0.5. The second layer has 128 neurons with activation function as Relu and the drop out at a rate of 0.5. The third layer again has 64 neurons with activation as Relu and the drop out at a rate of 0.5. The fourth layer again has 32 neurons with activation as Relu and the drop out at a rate of 0.5.

Adam is used as an optimize and BinaryCrossentropy as loss

Implementation of ANN in Tensorflow
$ANNmodel = models.Sequential()$
$ANNmodel.add(layers.Flatten())$
$ANNmodel.add(layers.Dense(256, activation =' relu'))$
$ANNmodel.add(layers.Dropout(0.5))$
$ANNmodel.add(layers.Dense(128, activation =' relu'))$
$ANNmodel.add(layers.Dropout(0.5))$
$ANNmodel.add(layers.Dense(64, activation =' relu'))$
$ANNmodel.add(layers.Dropout(0.5))$
$ANNmodel.add(layers.Dense(32, activation =' relu'))$
$ANNmodel.add(layers.Dropout(0.5))$
$ANNmodel.add(layers.Dense(10, activation =' sigmoid'))$

3.2.2 RNN model

Long Short-Term Memory (LSTM) unit is the portion that does the remembering, the Dropout randomly sets the weights of a portion of the data to zero to guard against overfitting, and the Dense units contain hidden layers tied to the degrees of freedom the model has to try and fit the data. The more complex the data, the more degrees of freedom the model needs all the while taking care to avoid overfitting (more on this later). The last Dense layer outputs the 10 species that the model is supposed to classify the audio recordings into.

In my RNN model i am using two **LSTM** layer of units 128 and 64 respectively and then dense layers and dropout are added in the following manner,the first dense layer has 256 neurons. Input shape according to the number of features with activation function as Relu, and to avoid any overfitting, we'll use the Dropout layer at a rate of 0.5. The second dense layer has 128 neurons with activation function as Relu and the drop out at a rate of 0.5. the third dense layer again has 64 neurons with activation as Relu and the drop out at a rate of 0.5. The fourth layer dense again has 32 neurons with activation as Relu and the drop out at a rate of 0.5. Adam is used as an optimize and BinaryCrossentropy as loss.

Implementation of ANN in Tensorflow
$RNNmodel = models.Sequential()$
$RNNmodel.add(LSTM(128, input_shape = inputshape))$
$RNNmodel.add(LSTM(64))$
$RNNmodel.add(layers.Dense(256, activation =' relu'))$
$RNNmodel.add(layers.Dropout(0.5))$
$RNNmodel.add(layers.Dense(128, activation =' relu'))$
$RNNmodel.add(layers.Dropout(0.5))$
$RNNmodel.add(layers.Dense(64, activation =' relu'))$
$RNNmodel.add(layers.Dropout(0.5))$
$RNNmodel.add(layers.Dense(32, activation =' relu'))$

$RNNmodel.add(layers.Dropout(0.5))$
$RNNmodel.add(layers.Dense(10, activation =' sigmoid'))$

Notice here no flatten layer is used as we are using LSTM layer to connect the the model to dense layer.

3.2.3 CRNN model

In my CRNN mdoel I first implement the CNN part of my code according to the architecture given below.
$CRNNmodel = models.Sequential()$
$CRNNmodel.add(layers.Conv2D(32, (3, 3), activation =' relu', inputshape = inputshape))$
$CRNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CRNNmodel.add(layers.Dropout(0.2))$
$CRNNmodel.add(layers.Conv2D(64, (3, 3), activation =' relu'))$
$CRNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CRNNmodel.add(layers.Dropout(0.2))$
$CRNNmodel.add(layers.Conv2D(128, (3, 3), activation =' relu'))$
$CRNNmodel.add(BatchNormalization())$
$CRNNmodel.add(layers.MaxPooling2D((2, 2)))$
$CRNNmodel.add(layers.Dropout(0.2))$
$CRNNmodel.add(layers.Conv2D(256, (3, 3), activation =' relu'))$
$CRNNmodel.add(BatchNormalization())$
$CRNNmodel.add(layers.Dropout(0.2))$
Then this CNN side is connected to RNN side by reshaping it the output shape of this CNN into (484, 256).
Now this is connected by adding one GRU layer of unit= 32, followed by droupout of 0.2, then flatten it and pass through the dense layer to make fully connected CRNN model.

Connecting the CNN with RNN to pass it through the dense layer
$CRNNmodel.add(GRU(32))$
$CRNNmodel.add(layers.Dropout(0.2))$
$CRNNmodel.add(layers.Flatten())$
$CRNNmodel.add(layers.Dense(256, activation =' relu'))$
$CRNNmodel.add(layers.Dense(10, activation =' sigmoid'))$

4 Precision, Recall, F1 score, confusion matrix (tables)

4.1 CNN model

The Deep CNN has the best performance in terms of validation accuracy and f1 score. It is a result of iterating through hyperparameters to find the balance between complexity and overtraining. As seen above in the model introduction section, our deep CNN model is relatively small compared to popular structures that have been implemented successfully in various tasks (e.g. Resnet, Inception. . .) This is because of the main difficulty for this project, getting the best performing model with limited number of data. **Validation accuracy** of this model is **0.59150**

CNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.84	0.87	0.85	0.85
Recall	0.87	0.82	0.87	0.89
F1 Score	0.85	0.84	0.85	0.84

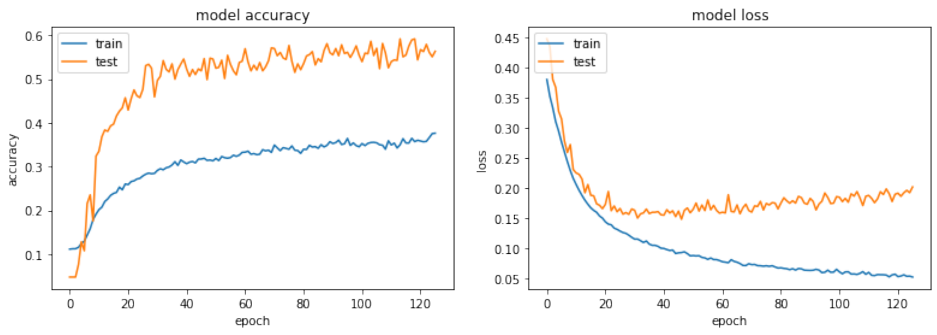
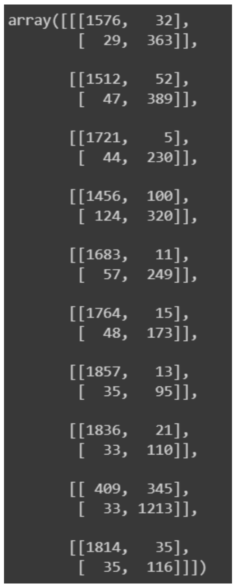


Figure 1: Accuracy and loss curve for validation data



```
array([[[1576,   32],
        [  29,  363]],

       [[1512,   52],
        [  47,  389]],

       [[1721,    5],
        [  44,  230]],

       [[1456,  100],
        [ 124,  320]],

       [[1683,   11],
        [  57,  249]],

       [[1764,   15],
        [  48,  173]],

       [[1857,   13],
        [  35,   95]],

       [[1836,   21],
        [  33,  110]],

       [[ 409,  345],
        [  33, 1213]],

       [[1814,   35],
        [  35,  116]]])
```

Confusion matrix

4.2 CRNN model

CRNN model performs the second best in terms of validation accuracy and f1 score. **Validation accuracy** of this model is **0.60250**

CRNN Evaluation Metrics (Test Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.45	0.13	0.75	0.46
Recall	0.72	0.18	0.72	0.79
F1 Score	0.55	0.13	0.73	0.56

CRNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.85	0.86	0.85	0.85
Recall	0.82	0.75	0.82	0.84
F1 Score	0.83	0.80	0.83	0.83

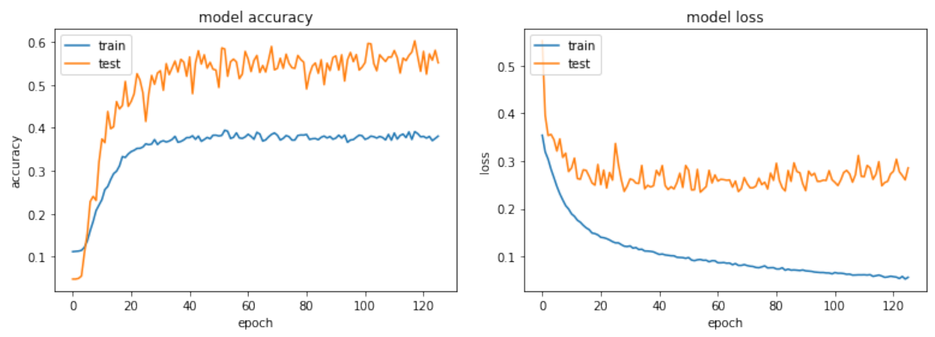
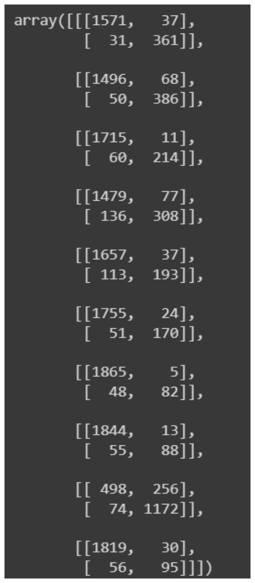


Figure 2: Accuracy and loss curve for validation data



```
array([[[1571,   37],
        [  31,  361]],

       [[1496,   68],
        [  50,  386]],

       [[1715,   11],
        [  60,  214]],

       [[1479,   77],
        [ 136,  308]],

       [[1657,   37],
        [ 113,  193]],

       [[1755,   24],
        [  51,  170]],

       [[1865,    5],
        [  48,   82]],

       [[1844,   13],
        [  55,   88]],

       [[ 498,  256],
        [  74, 1172]],

       [[1819,   30],
        [  56,   95]]])
```

Confusion matrix

4.3 RNN model

Validation accuracy of this model is **0.05750**

RNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.52	0.22	0.21	0.54
Recall	0.38	0.19	0.38	0.34
F1 Score	0.44	0.19	0.32	0.39

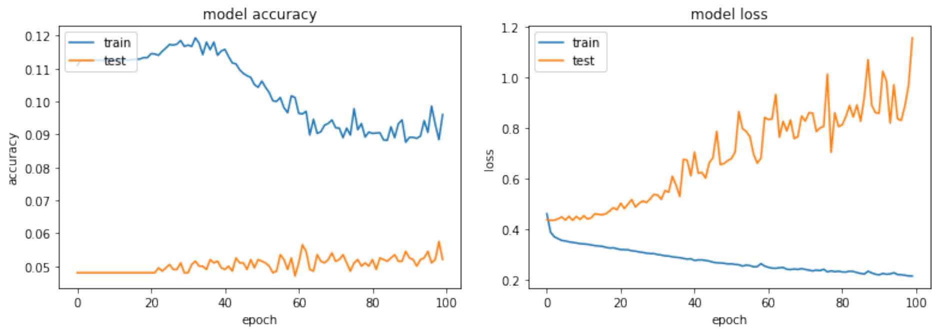
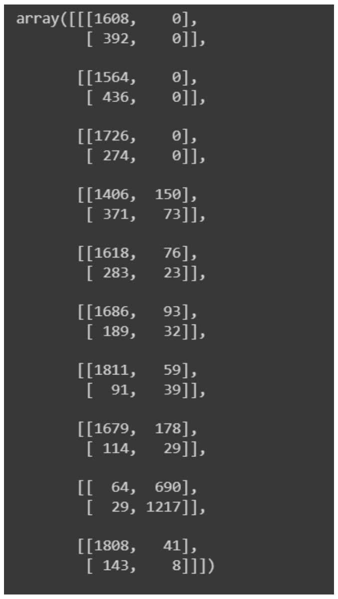


Figure 3: Accuracy and loss curve for validation data



```
array([[[1608,    0],
        [ 392,    0]],

       [[1564,    0],
        [ 436,    0]],

       [[1726,    0],
        [ 274,    0]],

       [[1406,  150],
        [ 371,   73]],

       [[1618,   76],
        [ 283,   23]],

       [[1686,   93],
        [ 189,   32]],

       [[1811,   59],
        [  91,   39]],

       [[1679,  178],
        [ 114,   29]],

       [[  64,  690],
        [  29, 1217]],

       [[1808,   41],
        [ 143,    8]]])
```

Confusion matrix

4.4 ANN model

Validation accuracy of this model is **0.05750**

ANN Evaluation Metrics (Validation Set)

Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.13	0.19	0.32	0.19
Recall	1.0	1.0	1.0	1.0
F1 Score	0.32	0.29	0.44	0.31

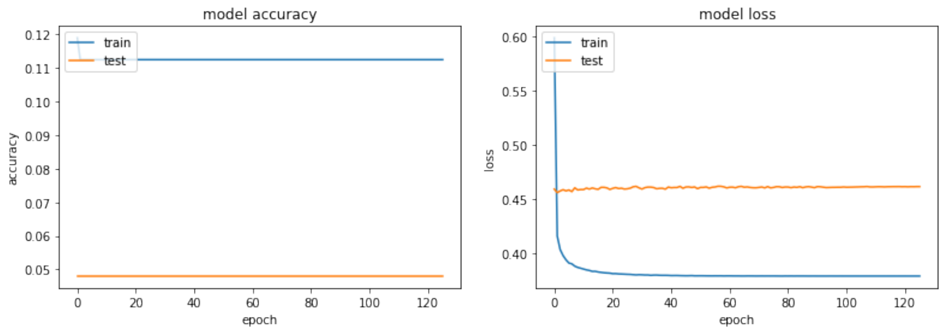
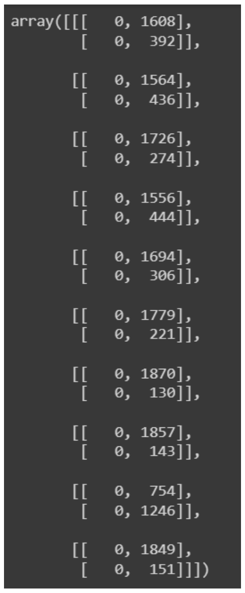


Figure 4: Accuracy and loss curve for validation data



```
array([[[   0, 1608],
        [   0,  392]],

       [[   0, 1564],
        [   0,  436]],

       [[   0, 1726],
        [   0,  274]],

       [[   0, 1556],
        [   0,  444]],

       [[   0, 1694],
        [   0,  306]],

       [[   0, 1779],
        [   0,  221]],

       [[   0, 1870],
        [   0,  130]],

       [[   0, 1857],
        [   0,  143]],

       [[   0,  754],
        [   0, 1246]],

       [[   0, 1849],
        [   0,  151]]])
```

Confusion matrix