Machine Learning in Signal Processing (EE603A) Assignment-II

Saransh Shivhare
Junior Undergraduate
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
`saranshg20@iitk.ac.in`

1 Introduction

Audio classification or sound classification can be referred to as the process of analyzing audio recordings. This amazing technique has multiple applications in the field of AI and data science such as *chatbots, automated voice translators, virtual assistants, music genre identification, and text to speech applications*. Multi-label Audio classification can be of multiple types and forms such as—*Acoustic Data Classification or acoustic event detection, Music Genre classification, Natural Language Classification, and Environmental Sound Classification*.

2 Literature Survey

Referred to several machine learning for audio articles in towardsDataScience website. Also referred to the paper on Sound Event Detection Based on Convolutional Neural Networks with Overlapping Pooling Structure and Polyphonic audio tagging with sequentially labelled data using CRNN with learnable gated linear units.

3 Details about implementation

3.1 Environmental Setup

The code editor used in the assignment is Google's Colab Notebook and Jupyter notebook. The colab editor consist of free GPU that helped in training the models easily.

Install the required packages using 'pip' command. 'pip' is the package installer for Python. You can use it to install packages from the Python Package Index and other indexes.

Packages and Libraries used in the Assignment are:

- **Librosa**: This is a Python package for music and audio analysis. Librosa is basically used when we work with audio data like in music generation(using LSTM's), Automatic Speech Recognition. It provides the building blocks necessary to create the music information retrieval systems.
- **NumPy**: This is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- **Pyplot**: 'matplotlib.pyplot' is a state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.
- **Sklearn**: Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random

forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- **Tensorflow**: TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

3.2 Data Pre-processing

- Imported all the data required for training in a drive folder and initialised all the path variables

```
dir_path = os.getcwd()
x_data = os.path.join(dir_path, "X")
y_data = os.path.join(dir_path, "Y")
mfcc_data = os.path.join(dir_path, "MFCC")
```

- Loaded all the files as mentioned in the from the given 'X' and 'Y' folder in the dataset and saved its MFCC format in a separate folder location.

```
def generateMFCC(x_data, y_data, mfcc_data):
    for file in os.listdir(x_data):
        mel_spectrogram = numpy.load(x_data+'/'+file)
        print(file)
        mfcc = librosa.feature.mfcc(S=mel_spectrogram, sr=16000,n_mfcc=20)
        mean = numpy.average(mfcc)
        std = numpy.std(mfcc)
        if(std == 0):
          std = std + 1e-25
        mfcc = (mfcc - mean)/std
        path = os.path.join(mfcc_data, file)
        numpy.save(path, mfcc)
```

- Following function is used to load data from the desired paths. Here, the training data is converted into tensor form. The distinction between a NumPy array and a tensor is that tensors, unlike NumPy arrays, are supported by accelerator memory such as the GPU, they have a faster processing speed. Therefore all the input training data is converted into tensor data-format.

```
def loadData(x_data, y_data):
    X=[]
    for file in x:
        arr = numpy.load(os.path.join(x_data, file))
        m,n,o = arr.shape
        arr = arr.reshape(n,o)
        mat = arr.reshape((arr.shape[0], arr.shape[1], 1))
        X.append(tf.convert_to_tensor(mat))
    X = numpy.array(X)
    y = []
    for file in y_list:
        arr = numpy.load(os.path.join(y_data, file))
        y.append(eventroll_to_multihot_vector(arr))
    y = numpy.array(y)
    return X,y
```

- As a machine can only understand numbers and cannot understand the text in the first place, this essentially becomes the case with Deep Learning & Machine Learning algorithms. One hot encoding can be defined as the essential process of converting the categorical data variables to be provided to machine and deep learning algorithms which in turn improve predictions as well as classification accuracy of a model.
  One Hot Encoding is a common way of preprocessing categorical features for machine learning models. This type of encoding creates a new binary feature for each possible

category and assigns a value of 1 to the feature of each sample that corresponds to its original category.

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

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

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

    # remove silence class:
    return numpy.delete(active_events, 8)

def extract_ytrain(y_data):
    y = []
    for file in y_list:
        arr = numpy.load(os.path.join(y_data, file))
        y.append(eventroll_to_multihot_vector(arr))
    y = numpy.array(y)
    return y
```

3.3 ANN-Model and Training

The model in the below code represents the **Artificial Neural Network** implementation for Single Audio Event Detection. From the provided samples, 10000 samples are used for training and 2000 samples are used for validation.

```
from tensorflow.keras import layers

def getModel():
    model = tf.keras.models.Sequential()
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(512, activation = 'relu'))
    model.add(layers.Dense(256, activation = 'relu'))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(128, activation = 'relu'))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(32, activation = 'relu'))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(10, activation = 'sigmoid'))

    return model
```

Model is compiled using

```
input_shape = X_train.shape
model.build(input_shape=input_shape)
model.compile(optimizer = tf.keras.optimizers.Adam(1e-2),
            loss = 'binary_crossentropy', metrics = ['accuracy'])
```

where the Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments, and Binary Cross-entropy is a loss function that is used in multi-label classification tasks. Lastly, model is fitted in the training samples using fit function.

```
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=32,
        epochs=150, verbose = 1, callbacks=[checkpoint])
```

3.4 CNN-Model and Training

The model in the below code represents the **Convolutional Neural Network** implementation for Single Audio Event Detection. This model is trained with 10000 samples and 2000 samples are used for validation.

```
def getModel(X_train):

    pool_size = (2, 2)
    kernel_size = (3, 3)
    input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
    num_classes = 10
    '''CNN1: '''
    model = tf.keras.models.Sequential([
    #first_convolution
    tf.keras.layers.Conv2D(32, kernel_size,
                padding="same", input_shape=input_shape),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.2),
    #second_convolution
    tf.keras.layers.Conv2D(128, kernel_size,
                              padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    #third_convolution
    tf.keras.layers.Conv2D(128, kernel_size,
                              padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    #fifth_convolution
    tf.keras.layers.Conv2D(256, kernel_size,
                              padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.GlobalMaxPooling2D(),
    tf.keras.layers.Dropout(0.2),
    #Fully connected 1st layer
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10, activation="sigmoid")
])

    model.compile(loss = 'binary_crossentropy', optimizer=tf.keras.optimizers.Adam(), metrics=['a

    return model
```

Similar to ANN model, CNN model is also fitted using the command

```
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=32, epochs=100,
        verbose = 1, callbacks=[checkpoint])
```

3.5 RNN-Model and Training

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. The model in the below code represents the **Recurrent Neural Network** implementation for multi-label audio classification.

```
def getModel():

    input_shape = (64, 1000)
    model = tf.keras.models.Sequential()
    model.add(layers.LSTM(units=128, recurrent_dropout=0.2, return_sequences=True,
    input_shape = input_shape))
    model.add(layers.LSTM(units=64, recurrent_dropout=0.2, return_sequences=False))
    model.add(layers.Dense(32, activation = 'relu'))
    model.add(layers.Dense(units=10, activation="sigmoid"))

    # model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
    model.compile(loss='binary_crossentropy',
                  optimizer='adam', metrics=['accuracy'])

    return model
```

3.6 CRNN-Model and Training

CRNN **(Convolutional Recurrent Neural Network)** model that. feeds every window frame by frame into a recurrent layer and. use the outputs and hidden states of the recurrent units in each frame for extracting features from the sequential windows.

```
from tensorflow import keras
from keras import Sequential
from keras.layers import Conv2D, MaxPool2D, BatchNormalization, Flatten,
        Dense, Dropout, GRU, Reshape

def getModel():

    input_shape = (64,1000,1)
    model = Sequential()

    # Convolutional Block 1
    model.add(Conv2D(filters=16, kernel_size=(3,3), input_shape=input_shape))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    # Convolutional Block 2
    model.add(Conv2D(filters=32, kernel_size=(3,3)))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    # Convolutional Block 3
    model.add(Conv2D(filters=64, kernel_size=(3,3)))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    # Convolutional Block 4
    model.add(Conv2D(filters=128, kernel_size=(3,3)))
    model.add(MaxPool2D(pool_size=(1,5)))
    model.add(BatchNormalization())

    # Reshape Layer (effectively squeezes the frequency dimension away)
    model.add(Reshape((96,128)))

    # Recurrent Layer
    model.add(GRU(256, return_sequences=False, activation="tanh"))

    # Dense Layer
    model.add(Dense(64))
    model.add(Dropout(0.4))

    # Output Layer
    model.add(Dense(10, activation="sigmoid"))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

The recurrent layer of the recommended CRNN has the same advantages as RNN, and it has the ability to mine timing related information. Following code is used for fitting the model on the training dataset.

```
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=32,
epochs=150, verbose = 1, callbacks=[checkpoint])
```

4 Model Evaluation Metrics

The evaluation metrices depends on the threshold set to multi hot encodings the predicted values. Following function is used to modify the predicted probabilities into multi hot encodings.

```
def modify_predictions(y_pred, threshold):
  for arr in y_pred:
    i = 0
    for _ in arr:
      if(_>=threshold):
        arr[i] = 1
      else:
        arr[i] = 0
      i=i+1
  return y_pred
```

The evaluation results depends on the selected threshold value. As per the observations, 0.1 gives fairly good results. Hence for all the evaluations made in the assignment the threshold is set to 0.1 value.
It should be noted that in the evaluation metrics data presented below the measurement takes into account true-positive, false-positive, true-negative and false-negative predictions.

4.1 Artificial Neural Network

ANN Evaluation Metrics (Validation Set)

Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.18	0.17	0.29	0.18
Recall	0.88	0.90	0.88	0.87
F1 Score	0.30	0.26	0.40	0.30

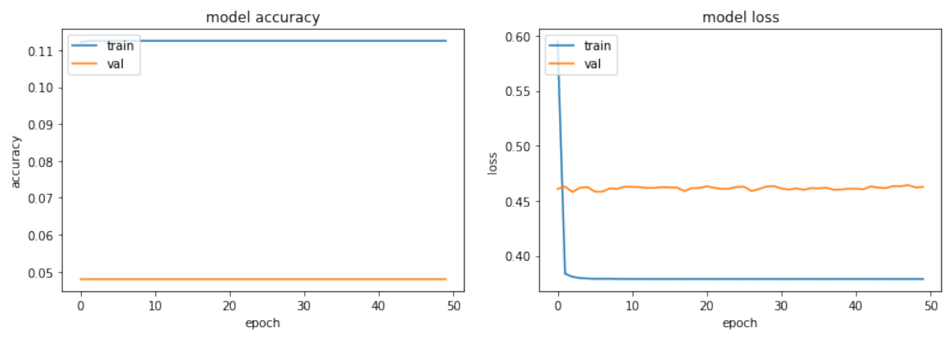


Figure 1: Accuracy and loss curve for validation data

ANN Evaluation Metrics (Test Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision 0.43	0.10	1.0	0.44	
Recall 0.95	0.12	0.95	0.95	
F1 Score 0.59	0.10	0.97	0.59	

4.2 Convolution Neural Network

CNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.90	0.88	0.91	0.92
Recall	0.84	0.84	0.84	0.87
F1 Score	0.87	0.85	0.87	0.87

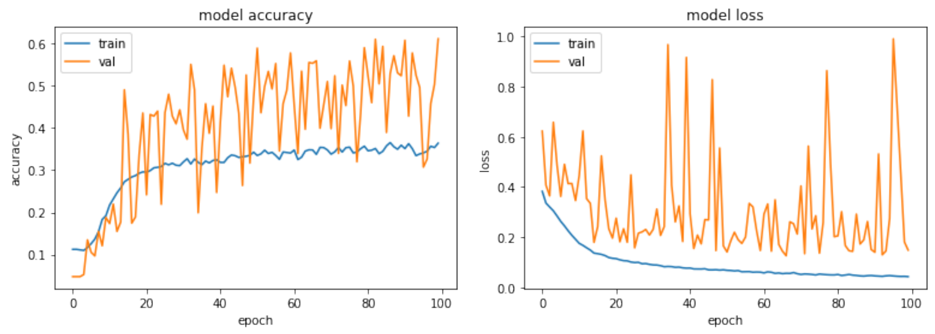


Figure 2: Accuracy and loss curve for validation data

CNN Evaluation Metrics (Test Set Accuracy: %)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.53	0.28	0.82	0.55
Recall	0.47	0.16	0.47	0.49
F1 Score	0.50	0.18	0.53	0.50

4.3 Recurrent Neural Network

RNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.33	0.27	0.37	0.33
Recall	0.78	0.63	0.78	0.77
F1 Score	0.46	0.37	0.49	0.45

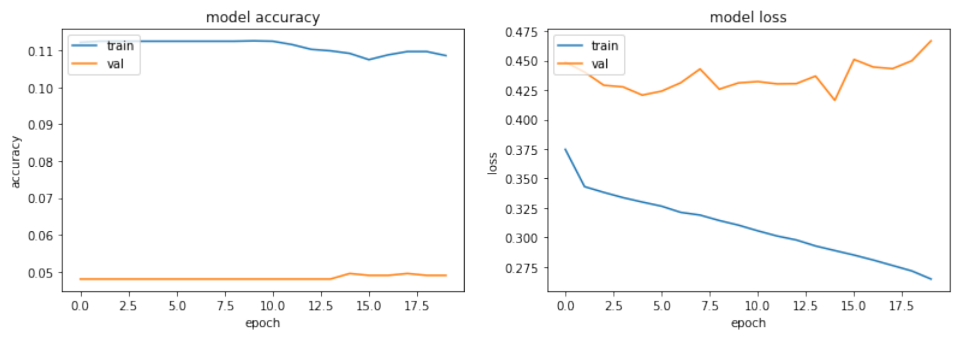


Figure 3: Accuracy and loss curve for validation data

RNN Evaluation Metrics (Test Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.78	0.62	0.95	0.79
Recall	0.30	0.22	0.30	0.30
F1 Score	0.43	0.27	0.39	0.43

4.4 Convolution-Recurrent Neural Network

CRNN Evaluation Metrics (Validation Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.77	0.81	0.78	0.79
Recall	0.84	0.76	0.84	0.85
F1 Score	0.80	0.77	0.80	0.79

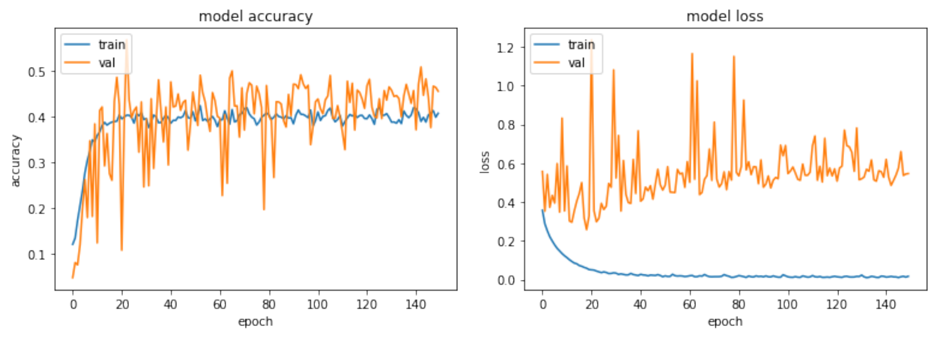
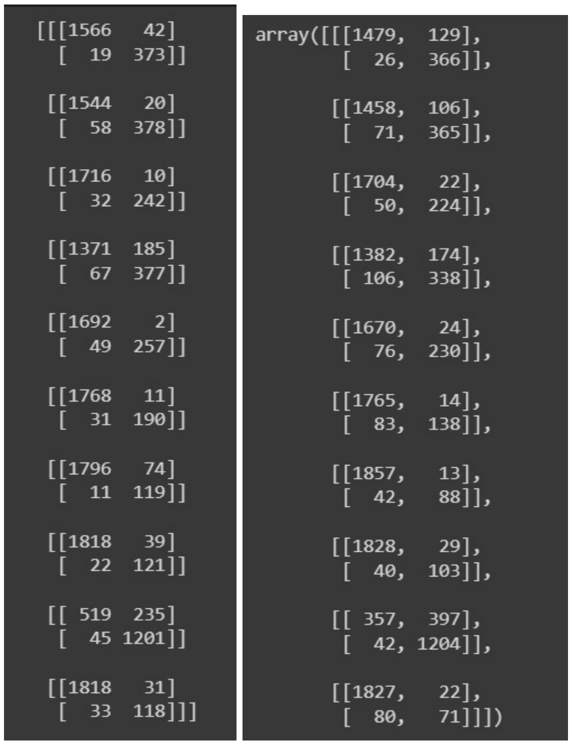


Figure 4: Accuracy and loss curve for validation data

CRNN Evaluation Metrics (Test Set)				
Metrics	Micro-Avg	Macro-Avg	Weighted-Avg	Samples-Avg
Precision	0.68	0.43	0.75	0.67
Recall	0.35	0.19	0.35	0.38
F1 Score	0.47	0.24	0.43	0.47



```
[[[1566   42]
  [  19  373]]

 [[1544   20]
  [  58  378]]

 [[1716   10]
  [  32  242]]

 [[1371  185]
  [  67  377]]

 [[1692    2]
  [  49  257]]

 [[1768   11]
  [  31  190]]

 [[1796   74]
  [  11  119]]

 [[1818   39]
  [  22  121]]

 [[ 519  235]
  [  45 1201]]

 [[1818   31]
  [  33  118]]]
```

```
array([[[1479,  129],
        [  26,  366]],

       [[1458,  106],
        [  71,  365]],

       [[1704,   22],
        [  50,  224]],

       [[1382,  174],
        [ 106,  338]],

       [[1670,   24],
        [  76,  230]],

       [[1765,   14],
        [  83,  138]],

       [[1857,   13],
        [  42,   88]],

       [[1828,   29],
        [  40,  103]],

       [[ 357,  397],
        [  42, 1204]],

       [[1827,   22],
        [  80,   71]]])
```

Figure 5: Confusion matrix(Validation) for CNN and CRNN models respectively

5 Observation and Discussion

- Convolutional-Recurrent Neural Network model is much better in performance as compared Convolutional Neural Network, Artificial Neural Network and Recurrent Neural Network models.
- If model is trained more than the required epochs then the predicted probabilities are very close to one, i.e. kind of hard classification.
- The model performance is nearly same for differently pre-processed dataset. Performance is better for log-scaled melspectrograms whereas MFCC and mel spectrograms performs nearly the same.
- MFCC dataset may be preferred over log-scaled melspectrograms to reduce computations while training the model. This trade-off helps to get the trained model in less time with minor decrement in performance.
- In case of CRNN model, it is important to select an **optimal threshold value to scale the predictions into multi-hot encoding**.
- There is a high correlation between **the learning rate** and **the batch size**, when the learning rates are high, the large batch size performs better than with small learning rates.
- Though CNN and CRNN models performance was well enough on the validation dataset but their performance is far from the expected results in the test data. It seem like the model is slightly overtrained on the training dataset.
- The RNN and ANN models does not perform well in case of multi-label classification.

6 Scope of improvement

- Multiple CNN models can be trained for the individual classes. This will slightly increase the pre-processing section. Though, this might not be a good option for dataset with more number of classes.