Machine Learning in Signal Processing (EE603A) Assignment-I

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*. Audio classifications can be of multiple types and forms such as—*Acoustic Data Classification or acoustic event detection, Music 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 Deep Learning for Audio Signal Processing

3 Details about implementation

3.1 Environmental Setup

The code editor used in the assignment is Google's Colab Notebook. The 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.
- **Pandas**: Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **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

```
path = '/content/drive/MyDrive/Audio_Classification-MLSP'
train_path=os.path.join(path, 'train')
label_data=pd.read_csv(os.path.join(path, 'annotations.csv'))
```

- Load all the files as mentioned in the 'fname' column of the data-frame in a list. The loaded spectrograms has 128-mel frequencies and variable time. Therefore, padding is required in the time axis to make the training data uniform. The list needs to be converted into 2-dimensional form to add padding using numpy.pad function.

```
spectrograms=[]
for file in label_data['fname']:
    arr=numpy.load(os.path.join(train_path, file))
    m,n,o=arr.shape
    arr.resize(n,o)
# Convert a power spectrogram (amplitude squared) to decibel (dB) units
    mel_spect = librosa.power_to_db(arr, ref=numpy.max)
    spectrograms.append(mel_spect)

# get the max duration numpy array
max_duration=0
for spectrogram in spectrograms:
    m,n=spectrogram.shape
    if max_duration<n:
        max_duration=n

spectrograms_padded=[]
for spectrogram in spectrograms:
# add zero padding
    temp=numpy.pad(spectrogram,[(0, 0),(0, max_duration-spectrogram[0].size)],
    mode='constant', constant_values=0)

    temp=temp.reshape((temp.shape[0], temp.shape[1], 1))
    spectrograms_padded.append(temp)
#convert from list to numpy array type
spectrograms_padded=numpy.array(spectrograms_padded)
```

- 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.

```
i=0
temp2=[]
for _ in spectrograms_padded:
# flatten the numpy array for ANN and KNN Model only
    arr = tf.convert_to_tensor(spectrograms_padded[i].flatten())
    # arr = tf.convert_to_tensor(spectrograms_padded[i])
    temp2.append(arr)
    i=i+1
```

```
X=numpy.array(temp2)
y=label_data['label']
```

- Splitting the dataset into training and validation set using sklearn package.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
= train_test_split(X,y,test_size=0.2, random_state=40, shuffle='true')
```

Dataset is split into 80:20 for training and validation respectively. Here, random state is a model hyper-parameter used to control the randomness involved in machine learning models. In the above code, random state of 40 gives the evenly distributed training data for all classes.

- 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.

```
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

#convert y_train label data into one_hot vector
train_encoder = LabelEncoder()
train_ = train_encoder.fit_transform(y_train)
to_onehot=to_categorical(train_)
y_train=to_onehot
#convert y_test label data into one_hot vector
test_encoder = LabelEncoder()
test_ = test_encoder.fit_transform(y_test)
to_onehot=to_categorical(test_)
y_test=to_onehot
```

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 1000 samples, 800 samples are used for training and 200 samples are used for validation.

```
pool_size = (2, 2)
kernel_size = (3, 3)
input_shape = X_train.shape
num_classes = 10

model = tf.keras.models.Sequential([
    #Fully connected 1st layer
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Model is compiled using

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
          loss=tf.keras.losses.CategoricalCrossentropy(),
          metrics=['accuracy', f1_m, precision_m, recall_m])
```

where the Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments, and Categorical Crossentropy is a loss function that is used in multi-class classification tasks. Lastly, model is fitted in the training samples using fit function.
The definition for f1, precision and recall used while compiling the model is given below (reference):

```
from keras import backend
def recall_m(y_true, y_pred):
    true_positives = backend.sum(backend.round(backend.clip(y_true * y_pred, 0, 1)))
    possible_positives = backend.sum(backend.round(backend.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + backend.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = backend.sum(backend.round(backend.clip(y_true * y_pred, 0, 1)))
    predicted_positives = backend.sum(backend.round(backend.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + backend.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+backend.epsilon()))

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 too is trained with 800 samples and the remaining samples are used for validation.

```
pool_size = (2, 2)
kernel_size = (3, 3)
input_shape = (128, 2584, 1)
num_classes = 10

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),
    #second_convolution
    tf.keras.layers.Conv2D(64, kernel_size,
                                padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),
    #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(0.25),
    #fourth_convolution
```

```
    tf.keras.layers.Conv2D(256, kernel_size,
                                    padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.GlobalMaxPooling2D(),
    #Fully connected 1st layer
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

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

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
          loss=tf.keras.losses.CategoricalCrossentropy(),
          metrics=["accuracy", f1_m, precision_m, recall_m])
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=32, epochs=160,
```

3.5 KNN-Model and Training

The model in the below code represents the **K-Nearest Classifier** implementation using Sklearn package for Single Audio Event Detection.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
import numpy as np
grid_params = {
    'n_neighbors': [3, 5],
    'weights':['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

KNN uses GridSearchCV function from Sklearn package where KNearestClassifier() is passed as an estimator. It runs through all the different parameters that is fed into the parameter grid and produces the best combination of parameters, based on a scoring metric of your choice (accuracy, f1, etc).

```
model = GridSearchCV(KNeighborsClassifier(), param_grid = grid_params,scoring = 'accuracy',
cv = 5, verbose = 1, n_jobs = -1)
model.fit(X_train, y_train)
```

4 Model Evaluation Metrics

4.1 Artificial Neural Network

ANN Evaluation Metrics		
Metrics	Score(Validation Set)	Score(Test Samples)
Accuracy	0.71499	0.35820
Precision	0.75841	0.35876
Recall	0.70089	0.33085
F1 Score	0.72705	0.34406

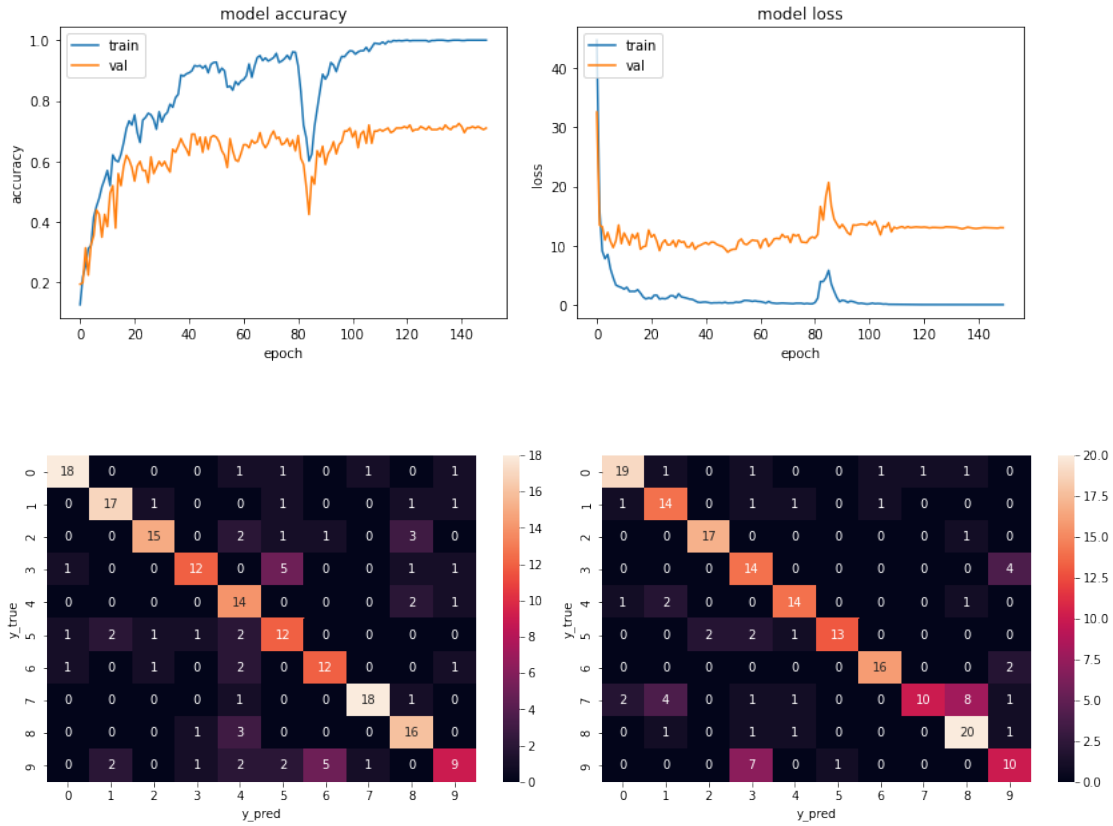
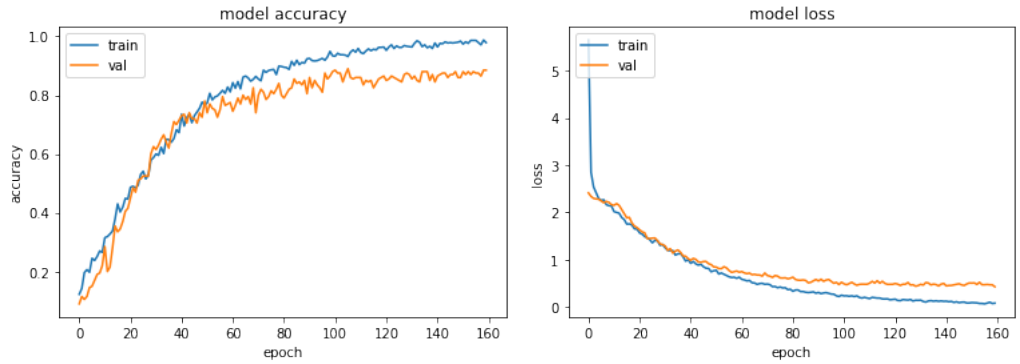


Figure 1: Heatmaps for validation data and test data respectively

4.2 Convolution Neural Network

CNN Evaluation Metrics		
Metrics	Score(Validation Set)	Score(Test Samples)
Accuracy	0.88499	0.73134
Precision	0.91753	0.76814
Recall	0.89732	0.71130
F1 Score	0.90706	0.73850



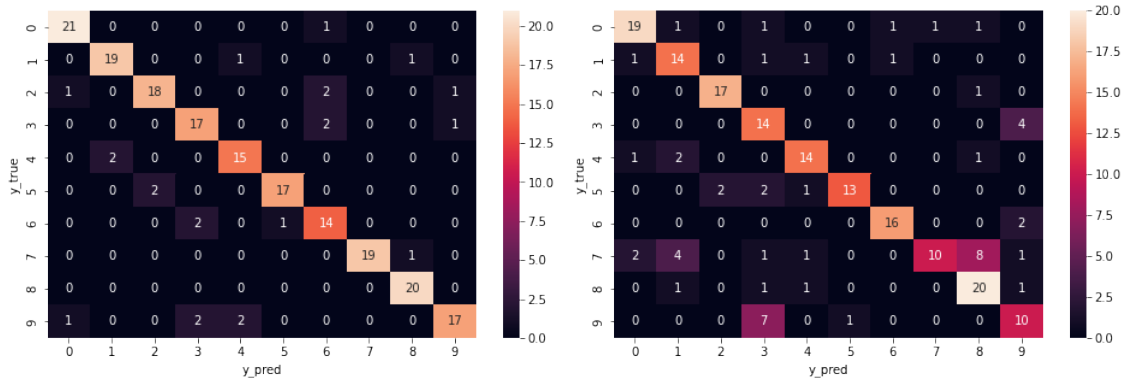


Figure 2: Heatmaps for validation data and test data respectively

4.3 K-Nearest Neighbours

KNN Evaluation Metrics		
Metrics	Score(Validation Set)	Score(Test Samples)
Accuracy	0.615	0.521
Precision	0.728	0.494
Recall	0.615	0.547
F1 Score	0.667	0.519

```
200 test samples
 [[11  0  1  0  0  1  1  1  3  1  3]
 [ 0 17  0  0  0  1  0  1  0  1  1]
 [ 0  0 15  0  0  0  2  2  0  0  3]
 [ 0  0  1 10  0  1  1  1  1  0  5]
 [ 0  0  0  0 11  1  0  0  0  0  5]
 [ 0  0  1  1  0 14  1  0  2  0  0]
 [ 0  0  1  0  2  1  8  0  0  0  5]
 [ 0  0  0  0  2  0  0 18  0  0  0]
 [ 0  0  1  0  0  1  0  1 13  0  4]
 [ 0  1  1  3  2  0  0  1  3  6  5]
 [ 0  0  0  0  0  0  0  0  0  0  0]]
```

Confusion Matrix for Validation Set

Best Parameters for KNN model built through GridSearchCV function are

```
model.best_params_ = {'metric': 'euclidean', 'n_neighbors':'3', 'weights': 'distance'}
```

5 Observation and Discussion

- Convolutional Neural Network model is much better in performance as compared K-Nearest Neighbors and Artificial Neural Network models. Observed that CNN model performs best when number of parameters are near to 50-million. Denser and Deep Networks tend to overfit.
- 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.
- In KNN, output completely relies on nearest neighbors, which may or may not be good choice. Also it is sensitive to distance metrics.

- The major difference between a traditional Artificial Neural Network (ANN) and CNN is that only the last layer of a CNN is fully connected whereas in ANN, each neuron is connected to every other neurons.
- ANN performance is non-uniform. Results widely vary in terms of accuracy and F1 score.