## Automatic Classification of Environmental Sounds with Convolutional Neural Networks (CNNs) Using Multi-Feature Channels

**Abstract**

With the advancement of Deep Neural Networks (DNN), the accuracy of sound classification such as Urban Sound Classification, Environmental Sound Classification etc., has been significantly improved. In this project, we propose a model that uses Convolutional Neural Networks (CNN) to identify sound based on the spectrograms for different sound samples collected. The model can be used for detection of deforestation, detection of shooting in urban areas and detection of strange noises at odd hours in streets such as Air Conditioner, Car Horn, Children Playing, Dog bark, Drilling, Engine Idling, Gun Shot, Jackhammer, Siren, Street Music etc.,

**Challenges**

Environmental sound work has two major obstacles, namely the lack of audio data labelled. Previous work focused on audio from carefully produced films or TV tracks from particular environments such as elevators or office spaces and commercial or proprietary datasets.

Lack of fundamental vocabulary in Environmental Sounds work. This means that the classification of sounds in to the semantic groups may vary from study to study, making it difficult to compare results so the goal of this notebook is to address the two challenges mentioned above.

**Dataset**

The dataset is called UrbanSound8K and contains 8732 labelled sound excerpts (<=4s) of urban sounds from 10 classes: - The dataset contains 8732 sound excerpts (<=4s) of urban sounds from 10 classes, namely: Air Conditioner Car Horn Children Playing Dog bark Drilling Engine Idling Gun Shot Jackhammer Siren Street Music The attributes of data are as follows: ID Unique ID of sound excerpt Class type of sound

**Problem statement**

It will show how to apply Deep Learning techniques to environmental recognition sounds, focusing specifically on recognizing unique Environmental sounds. If we give an audio sample of a few seconds duration in a computer-readable format (such as a.wav file), we want to be able to determine whether it contains one of the target Environmental sounds with a corresponding classification accuracy score.

**Note:** Loading audio files and pre-processing takes some times to complete with large dataset. To avoid reload every time reset the kernel or resume works on next day, all loaded audio data will be serialized into a object file. so next round only need to load the seriazed object file.

Optional GPU configuration initialization

In [1]:

*# GPU memory wizardry to avoid out of memory when using tensorflow-GPU as Keras backend*

*# remove this part of using CPU only version of tensorflow.*

import tensorflow as tf

from keras import backend as k

config = tf.ConfigProto() *# Set GPU options for tensorflow GPU*

config.gpu\_options.allow\_growth = True *# Don't pre-allocate memory; allocate as-needed*

config.gpu\_options.per\_process\_gpu\_memory\_fraction = 0.8 *# Only allow a total of half the GPU memory to be allocated*

k.tensorflow\_backend.set\_session(tf.Session(config=config)) *# Create a session with the above options specified.*

Using TensorFlow backend.

**Pre-requisite**

* librosa audio codec which required internet connected turn on under settings

In [2]:

*# If librosa report "no backend error", install audio codec*

!apt-get -y install libav-tools

*# or ffmpeg*

*#!apt-get -y install software-properties-common*

*#!add-apt-repository ppa:mc3man/trusty-media*

*#!apt-get -y install ffmpeg*

*#!apt-get -y install frei0r-plugins*

*#!ffmpeg -version*

Import libraries

In [3]:

import keras

from keras.layers import Activation, Dense, Dropout, Conv2D, \

Flatten, MaxPooling2D

from keras.models import Sequential

from keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,TensorBoard,ProgbarLogger

from sklearn.model\_selection import train\_test\_split

import librosa

import librosa.display

import numpy as np

import pandas as pd

import random

import warnings

warnings.filterwarnings('ignore')

*#object serialization*

import \_pickle as cPickle *#python 3 change*

import os

%matplotlib inline

In [4]:

*#enable memory profiler for memory management usage %%memit*

from memory\_profiler import memory\_usage

%load\_ext memory\_profiler

*#enable garbage collection control*

import gc

gc.enable()

In [5]:

*#progress tracker*

from tqdm import tqdm, tqdm\_notebook

Audio file loading control flag

In [6]:

*# when set to TRUE, training data get loaded from a saved serialized data object file*

*# All audio files data get saved to a serialized object file to save reloading time on training runs*

*#*

*# Note:*

*# On first time run, if serialized file doesn't exist, this flag will get overrident*

*#*

SKIP\_AUDIO\_RELOAD = False

Dataset exploration

In [7]:

*#location of the sound files*

INPUT\_PATH='../input'

TRAIN\_INPUT=INPUT\_PATH+'/train'

TRAIN\_AUDIO\_DIR=TRAIN\_INPUT+'/Train'

TEST\_INPUT=INPUT\_PATH+'/test'

TEST\_AUDIO\_DIR=TEST\_INPUT+'/Test'

In [8]:

def load\_input\_data(pd, filepath):

*# Read Data*

data = pd.read\_csv(filepath)

return data

In [9]:

*# training file*

TRAIN\_FILE=TRAIN\_INPUT+'/train.csv'

*#show info*

train\_input=load\_input\_data(pd,TRAIN\_FILE)

train\_input.head()

Out[9]:

|  | ID | Class |
| --- | --- | --- |
| 0 | 0 | siren |
| 1 | 1 | street\_music |
| 2 | 2 | drilling |
| 3 | 3 | siren |
| 4 | 4 | dog\_bark |

In [10]:

*# training file*

TEST\_FILE=TEST\_INPUT+'/test.csv'

*#show info*

test\_input=load\_input\_data(pd,TEST\_FILE)

test\_input.head()

Out[10]:

|  | ID |
| --- | --- |
| 0 | 5 |
| 1 | 7 |
| 2 | 8 |
| 3 | 9 |
| 4 | 13 |

In [11]:

*#labels*

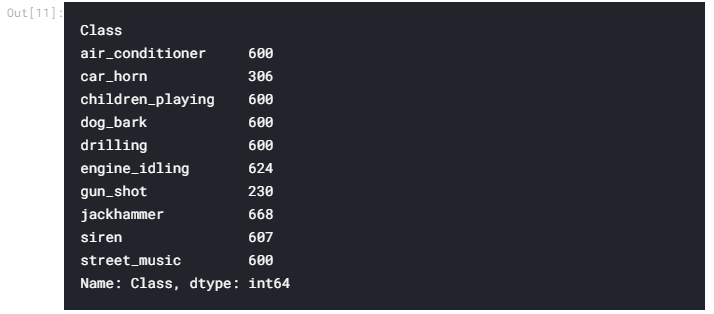
valid\_train\_label = train\_input[['Class']]

*#x=data['label'].unique()*

valid\_train\_label.count()

*#unique classes*

x = train\_input.groupby('Class')['Class'].count()



int64

In [12]:

*# train data size*

valid\_train\_data = train\_input[['ID', 'Class']]

valid\_train\_data.count()



In [13]:

*# test data size*

valid\_test\_data = test\_input[['ID']]

valid\_test\_data.count()



t64

**Check input audio file samples**

In [14]:

*# sample-1 load*

sample1=TRAIN\_AUDIO\_DIR+'/943.wav'

duration=2.97

sr=22050

y, sr = librosa.load(sample1, duration=duration, sr=sr)

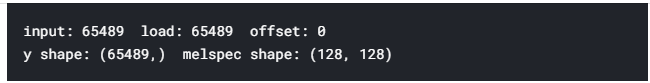
ps = librosa.feature.melspectrogram(y=y, sr=sr)

input\_length=sr\*duration

offset = len(y) - round(input\_length)

print ("input:", round(input\_length), " load:", len(y) , " offset:", offset)

print ("y shape:", y.shape, " melspec shape:", ps.shape)



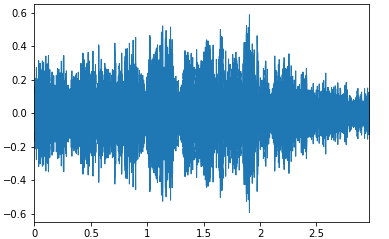
c shape: (128, 128)

In [15]:

*# sample-1 waveplot*

librosa.display.waveplot(y,sr)

on at 0x7f6829081ef0>



In [16]:

*# sample-1: audio*

import IPython.display as ipd

ipd.Audio(sample1)

Out[16]:

In [17]:

*# sample-1: spectrogram*

librosa.display.specshow(ps, y\_axis='mel', x\_axis='time')

In [18]:

*# sample-2 load*

sample2=TRAIN\_AUDIO\_DIR+'/1.wav'

duration=2.97

sr=22050

y2, sr2 = librosa.load(sample2, duration=duration, sr=sr)

ps2 = librosa.feature.melspectrogram(y=y2, sr=sr2)

input\_length=sr\*duration

offset = len(y) - round(input\_length)

print ("input:", round(input\_length), " load:", len(y) , " offset:", offset)

print ("y shape:", y.shape, " melspec shape:", ps2.shape)

)

In [19]:

*# sample-2: audio*

ipd.Audio(sample2)

In [20]:

*# sample-2: spectrogram*

librosa.display.specshow(ps2, y\_axis='mel', x\_axis='time')

ps.shape

**Prepare data file loading**

In [21]:

*#training audio files*

valid\_train\_data['path'] = TRAIN\_AUDIO\_DIR+'/' + train\_input['ID'].astype('str')+".wav"

print ("sample",valid\_train\_data.path[1])

valid\_train\_data.head(5)

sample ../input/train/Train/1.wav

Out[21]:

|  | ID | Class | path |
| --- | --- | --- | --- |
| 0 | 0 | siren | ../input/train/Train/0.wav |
| 1 | 1 | street\_music | ../input/train/Train/1.wav |
| 2 | 2 | drilling | ../input/train/Train/2.wav |
| 3 | 3 | siren | ../input/train/Train/3.wav |
| 4 | 4 | dog\_bark | ../input/train/Train/4.wav |

In [22]:

*#test audio files*

valid\_test\_data['path'] = TEST\_AUDIO\_DIR+'/' + test\_input['ID'].astype('str') +".wav"

print ("sample",valid\_test\_data.path[1])

valid\_test\_data.head(5)

sample ../input/test/Test/7.wav

Out[22]:

|  | ID | path |
| --- | --- | --- |
| 0 | 5 | ../input/test/Test/5.wav |
| 1 | 7 | ../input/test/Test/7.wav |
| 2 | 8 | ../input/test/Test/8.wav |
| 3 | 9 | ../input/test/Test/9.wav |
| 4 | 13 | ../input/test/Test/13.wav |

**Loading audio file and features**

In [23]:

*#*

*# set duration on audio loading to make audio content to ensure each training data have same size*

*#*

*# for instance, 3 seconds audio will have 128\*128 which will be use on this notebook*

*#*

def audio\_norm(data):

max\_data = np.max(data)

min\_data = np.min(data)

data = (data-min\_data)/(max\_data-min\_data+0.0001)

return data-0.5

*#fix the load audio file size*

audio\_play\_duration=2.97

def load\_audio\_file(file\_path, duration=2.97, sr=22050):

*#load 5 seconds audio file, default 22 KHz default sr=22050*

*# sr=resample to 16 KHz = 11025*

*# sr=resample to 11 KHz = 16000*

*# To preserve the native sampling rate of the file, use sr=None*

input\_length=sr\*duration

*# Load an audio file as a floating point time series.*

*# y : np.ndarray [shape=(n,) or (2, n)] - audio time series*

*# sr : number > 0 [scalar] - sampling rate of y*

y, sr = librosa.load(file\_path,sr=sr, duration=duration)

dur = librosa.get\_duration(y=y)

*#pad output if audio file less than duration*

*# Use edge-padding instead of zeros*

*#librosa.util.fix\_length(y, 10, mode='edge')*

if (round(dur) < duration):

offset = len(y) - round(input\_length)

print ("fixing audio length :", file\_path)

print ("input:", round(input\_length), " load:", len(y) , " offset:", offset)

y = librosa.util.fix\_length(y, round(input\_length))

*# y = audio\_norm(y)*

*# using a pre-computed power spectrogram*

*# Short-time Fourier transform (STFT)*

*#D = np.abs(librosa.stft(y))\*\*2*

*#ps = librosa.feature.melspectrogram(S=D)*

ps = librosa.feature.melspectrogram(y=y, sr=sr)

return ps

In [24]:

%%time

%%memit

# Dataset

train\_audio\_data = []

train\_object\_file='saved\_train\_audio\_data.p'

#override the reload flag if serized file doesn't exist

if not os.path.isfile(train\_object\_file):

SKIP\_AUDIO\_RELOAD = False

#load training data

if SKIP\_AUDIO\_RELOAD is True:

print ("skip re-loading TRAINING data from audio files")

else:

print ("loading train audio data, may take more than 15 minutes. please wait!")

for row in tqdm(valid\_train\_data.itertuples()):

ps = load\_audio\_file(file\_path=row.path, duration=2.97)

if ps.shape != (128, 128): continue

train\_audio\_data.append( (ps, row.Class) )

print("Number of train samples: ", len(train\_audio\_data))

# this step took sometime to finish 5382

#peak memory: 1141.30 MiB, increment: 642.16 MiB

#CPU times: user 15min 41s, sys: 14min 57s, total: 30min 39s

In [25]:

*# load saved audio object*

if SKIP\_AUDIO\_RELOAD **is** True:

train\_audio\_data = cPickle.load(open(train\_object\_file, 'rb'))

print ("loaded train data [**%s**] records from object file" % len(train\_audio\_data))

else:

cPickle.dump(train\_audio\_data, open(train\_object\_file, 'wb'))

print ("saved loaded train data :",len(train\_audio\_data))

saved loaded train data : 5382

In [26]:

%%time

%%memit

#load test data

test\_audio\_data = []

test\_object\_file='saved\_test\_audio\_data.p'

#override the reload flag if serized file doesn't exist

if not os.path.isfile(test\_object\_file):

SKIP\_AUDIO\_RELOAD = False

if SKIP\_AUDIO\_RELOAD is True:

print ("skip re-loading TEST data from audio files")

else:

print ("loading test audio data, may take more than 15 minutes. please wait!")

for row in tqdm(valid\_test\_data.itertuples()):

ps = load\_audio\_file(file\_path=row.path, duration=2.97)

if ps.shape != (128, 128):

print ("\*\*\*data shape is wrong, replace it with zeros ", ps.shape, row.path)

ps = np.zeros([128, 128])

#continue

test\_audio\_data.append( (ps, row.ID) )

print("Number of train samples: ", len(train\_audio\_data))

# this step took sometime to finish 3251

#peak memory: 1586.96 MiB, increment: 445.65 MiB

#CPU times: user 9min 32s, sys: 9min 37s, total: 19min 10s

0iU times: user 12min 34s, sys: 4min 42s, total: 17min 16s

Wall time: 16min 34s

In [27]:

*# load saved data*

if SKIP\_AUDIO\_RELOAD **is** True:

test\_audio\_data = cPickle.load(open(test\_object\_file, 'rb'))

print ("loaded test data [**%s**] records from object file" % len(test\_audio\_data))

else:

cPickle.dump(test\_audio\_data, open(test\_object\_file, 'wb'))

print ("save loaded test data :", len(test\_audio\_data))

save loaded test data : 3297

**Prepare data for training**

**Encode labels**

In [28]:

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import OneHotEncoder

from keras.utils import to\_categorical

from numpy import argmax

*# get a set of unique text labels*

list\_labels = sorted(list(set(valid\_train\_data.Class.values)))

print ("unique text labels count: ",len(list\_labels))

print ("labels: ",list\_labels)

*# integer encode*

label\_encoder = LabelEncoder()

label\_integer\_encoded = label\_encoder.fit\_transform(list\_labels)

print("encoded labelint values", label\_integer\_encoded)

*# one hot encode*

encoded\_test = to\_categorical(label\_integer\_encoded)

inverted\_test = argmax(encoded\_test[0])

*#print(encoded\_test, inverted\_test)*

*#map filename to label*

file\_to\_label = {k:v for k,v **in** zip(valid\_train\_data.path.values, valid\_train\_data.ID.values)}

*# Map integer value to text labels*

label\_to\_int = {k:v for v,k **in** enumerate(list\_labels)}

*#print ("test label to int ",label\_to\_int["Applause"])*

*# map integer to text labels*

int\_to\_label = {v:k for k,v **in** label\_to\_int.items()}

unique text labels count: 10

labels: ['9]

**Split Up Data Into Train, Test And Validation**

In [29]:

*#full dataset*

dataset = train\_audio\_data

random.shuffle(dataset)

RATIO=0.9

train\_cutoff= round(len(dataset) \* RATIO)

train = dataset[:train\_cutoff]

test = dataset[train\_cutoff:]

X\_train, y\_train = zip(\*train)

X\_test, y\_test = zip(\*test)

*# Reshape for CNN input*

X\_train = np.array([x.reshape( (128, 128, 1) ) for x **in** X\_train])

X\_test = np.array([x.reshape( (128, 128, 1) ) for x **in** X\_test])

print ("train ",X\_train.shape, len(y\_train))

print ("test ", X\_test.shape, len(y\_test))

train (4844, 128, 128, 1) 4844

test (538, 128, 128, 1) 538

In [30]:

*# Apply sck-learn label text encoding to integer*

label\_encoder = LabelEncoder()

y\_train\_integer\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_integer\_encoded = label\_encoder.fit\_transform(y\_test)

In [31]:

*# Apply Keras One-Hot encoding for classes*

y\_train = np.array(keras.utils.to\_categorical(y\_train\_integer\_encoded, len(list\_labels)))

y\_test = np.array(keras.utils.to\_categorical(y\_test\_integer\_encoded, len(list\_labels)))

In [32]:

*#split up test into test and validation*

X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_test, y\_test, test\_size=0.30, random\_state=42)

print ("test ",X\_test.shape, len(y\_test))

print ("valid ", X\_val.shape, len(y\_val))

test (376, 128, 128, 1) 376

valid (162, 128, 128, 1) 162

In [33]:

*# build convolution model*

*# input shape = (128, 128, 1)*

model = Sequential()

input\_shape= X\_train.shape[1:]

model.add(Conv2D(24, (5, 5), strides=(1, 1), input\_shape=input\_shape))

model.add(MaxPooling2D((4, 2), strides=(4, 2)))

model.add(Activation('relu'))

model.add(Conv2D(48, (5, 5), padding="valid"))

model.add(MaxPooling2D((4, 2), strides=(4, 2)))

model.add(Activation('relu'))

model.add(Conv2D(48, (5, 5), padding="valid"))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dropout(rate=0.5))

model.add(Dense(64))

model.add(Activation('relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(len(list\_labels)))

model.add(Activation('softmax'))

model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [34]:

%%time

%%memit

# NOTE:

# Increase number if epochs from 1 to 60 or 100 for higher prediction accuracy

# default is set to 1 for faster commit

MAX\_EPOCHS=3

MAX\_BATCH\_SIZE=23

# learning rate reduction rate

MAX\_PATIENT=2

# saved model checkpoint file

best\_model\_file="./best\_model\_trained.hdf5"

# callbacks

# removed EarlyStopping(patience=MAX\_PATIENT)

callback=[ReduceLROnPlateau(patience=MAX\_PATIENT, verbose=1),

ModelCheckpoint(filepath=best\_model\_file, monitor='loss', verbose=1, save\_best\_only=True)]

#compile

model.compile(optimizer="Adam",loss="categorical\_crossentropy",metrics=['accuracy'])

#train

print('training started.... please wait!')

history = model.fit(x=X\_train, y=y\_train,

epochs=MAX\_EPOCHS,

batch\_size=MAX\_BATCH\_SIZE,

verbose=0,

validation\_data= (X\_val, y\_val),

callbacks=callback)

print('training finished')

# quick evaludate model

print('Evaluate model with test data')

score = model.evaluate(x=X\_test,y=y\_test)

print('test loss:', score[0])

print('test accuracy:', score[1])

training started.... please wait!

In [35]:

%%time

%%memit

import matplotlib.pyplot as plt

#Plot loss and accuracy for the training and validation set.

def plot\_history(history):

loss\_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]

val\_loss\_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]

acc\_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]

val\_acc\_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]

if len(loss\_list) == 0:

print('Loss is missing in history')

return

plt.figure(figsize=(22,10))

## As loss always exists

epochs = range(1,len(history.history[loss\_list[0]]) + 1)

## Accuracy

plt.figure(221, figsize=(20,10))

## Accuracy

# plt.figure(2,figsize=(14,5))

plt.subplot(221, title='Accuracy')

for l in acc\_list:

plt.plot(epochs, history.history[l], 'b', label='Training accuracy (' + str(format(history.history[l][-1],'.5f'))+')')

for l in val\_acc\_list:

plt.plot(epochs, history.history[l], 'g', label='Validation accuracy (' + str(format(history.history[l][-1],'.5f'))+')')

plt.title('Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

## Loss

plt.subplot(222, title='Loss')

for l in loss\_list:

plt.plot(epochs, history.history[l], 'b', label='Training loss (' + str(str(format(history.history[l][-1],'.5f'))+')'))

for l in val\_loss\_list:

plt.plot(epochs, history.history[l], 'g', label='Validation loss (' + str(str(format(history.history[l][-1],'.5f'))+')'))

plt.title('Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# plot history

plot\_history(history)

<Figure size 1584x720 with 0 Axes>

**Model Evaluation**

In [36]:

*#Evaludate model use Keras reported accuracy:*

score = model.evaluate(X\_train, y\_train, verbose=0)

print ("model train data score : ",round(score[1]\*100) , "%")

score = model.evaluate(X\_test, y\_test, verbose=0)

print ("model test data score : ",round(score[1]\*100) , "%")

score = model.evaluate(X\_val, y\_val, verbose=0)

print ("model validation data score : ", round(score[1]\*100), "%")

model train data score : 44.0 %

model test data score : 41.0 %

model validation data score : 43.0 %

**Test Prediction**

In [37]:

print ("Prediction with [train] data")

y\_pred = model.predict\_classes(X\_train)

missed=[]

matched=[]

for i **in** range(len(y\_pred)):

y\_val\_label\_int = argmax(y\_train[i])

if (y\_pred[i]!=y\_val\_label\_int):

missed.append( (y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int] ))

else:

matched.append((y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int]))

print (" |\_\_match :", len(matched))

print (" |\_\_miss :", len(missed))

print (" |\_\_accuracy :", round((len(matched)-len(missed))/len(matched)\*100,2), "%")

print ("")

*#print ("Value missed : \n",missed)*

*# show sample results*

print ("---samples---")

for i **in** range(5):

print (i,"predict =", int\_to\_label[y\_pred[i]])

print (i,"original=", int\_to\_label[argmax(y\_train[i])])

print ("")

Prediction with [train] data= drilling

In [38]:

*# prediction class*

print ("Prediction with [test] data")

y\_pred = model.predict\_classes(X\_test)

missed=[]

matched=[]

for i **in** range(len(y\_pred)):

y\_val\_label\_int = argmax(y\_test[i])

if (y\_pred[i]!=y\_val\_label\_int):

missed.append( (y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int] ))

else:

matched.append((y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int]))

print (" |\_\_match :", len(matched))

print (" |\_\_miss :", len(missed))

print (" |\_\_accuracy :", round((len(matched)-len(missed))/len(matched)\*100,2), "%")

print ("")

*#print ("Value missed : \n",missed)*

*# show sample results*

print ("---samples---")

for i **in** range(8):

print (i,"predict =", int\_to\_label[y\_pred[i]])

print (i,"original=", int\_to\_label[argmax(y\_test[i])])

print ("")

Prediction with [test] data

|\_\_match : 154

|\_\_miss

In [39]:

*# prediction class*

print ("Prediction with [validation] data")

y\_pred = model.predict\_classes(X\_val)

missed=[]

matched=[]

for i **in** range(len(y\_pred)):

y\_val\_label\_int = argmax(y\_val[i])

if (y\_pred[i]!=y\_val\_label\_int):

missed.append( (y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int] ))

else:

matched.append((y\_pred[i], "-", int\_to\_label[y\_pred[i]], " - ", int\_to\_label[y\_val\_label\_int]))

print (" |\_\_match :", len(matched))

print (" |\_\_miss :", len(missed))

print (" |\_\_accuracy :", round((len(matched)-len(missed))/len(matched)\*100,2), "%")

print ("")

*#print ("Value missed : \n",missed)*

*# show sample results*

print ("---samples---")

for i **in** range(8):

print (i,"predict =", int\_to\_label[y\_pred[i]])

print (i,"original=", int\_to\_label[argmax(y\_val[i])])

print ("")

Prediction with [validation] data

|\_\_mat

Prepcare Submission

In [40]:

print ("test data size ",len(test\_audio\_data))

sub\_test = test\_audio\_data[1:22]

tx\_test, ty\_test = zip(\*test\_audio\_data)

*# make prediction*

tx\_test2 = np.array([x.reshape((128, 128, 1)) for x **in** tx\_test])

print ("test data shape ", tx\_test2.shape)

test data size 3297

test data shape (3297, 128, 128, 1)

In [41]:

*# run prediction data*

y\_pred = model.predict\_classes(tx\_test2, batch\_size=1)

print ( len(y\_pred), len(tx\_test2))

3297 3297

In [42]:

*# save result for submission*

prediction\_output\_file='prediction\_result\_1.csv'

with open(prediction\_output\_file,"w") as file:

file.write("ID,Prediction**\n**")

i=0

for i **in** range( (len(valid\_test\_data)-1)) :

*#print(i, y\_pred[i])*

file.write(str(valid\_test\_data['ID'][i])+','+ int\_to\_label[y\_pred[i]])

file.write('**\n**')

i=i+1

print (len(y\_pred))

output = pd.read\_csv(prediction\_output\_file)

output.head(20)

3297

Automatic Classification of Environmental Sounds with Convolutional Neural Networks (CNNs) Using Multi-Feature Channels is the model prediction accuracy has increased and will give best results .