Animal Language Processing

A PROJECT REPORT

Submitted by,

Purushottam Hari (17BCE2182)
S.Mahika Rao (17BCE0944)

Natural Language Processing (EPJ)
Slot- A2

Project Guide

Prof. Sharmila Banu
SCOPE

B-Tech(CSE)
IN
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2019-20

Project GitHub Repo: https://github.com/PurushottamHari/NLP_Project

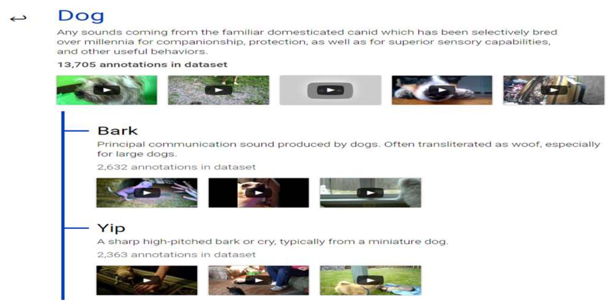
Problem statement

- Our project idea was to firstly gather a dataset consisting of the barks of a particular dog over a substantial period of time.
- Then isolate the barks and the respective timestamps on the video.
- Then train a model using Deep Learning to understand patterns within the barks.
- Cross Verify the results using the timestamps on the video

Dataset collection

- Unfortunately, after looking at many options in our campus, we couldn't collect the required dataset in the format we needed for this project since it required the use of an expensive camera like a GoPro which needed to be attached to a harness like so.

- We tried to hence record bird noises in shops right outside our campus, but their sounds were too feeble to be captured clearly on a Mobile's microphone.
- But gladly after researching on this topic we came across a dataset which consisted of many YouTube Videos of various dogs barking https://research.google.com/audioset/ontology/dog_1.html ......



So, we continued our project with this source so that we could at least understand how the model can be built and the various steps that need to be looked out for if we get a Quality Dataset in the future.

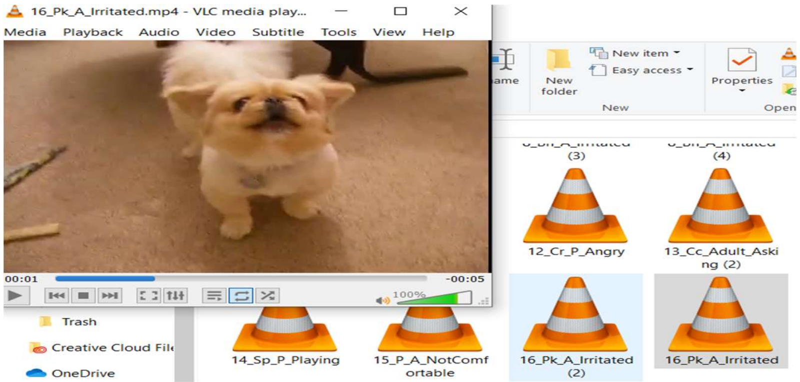
Procedure Adopted

- We first downloaded around 200 videos each using the dataset we found onto our systems.
- Then using a video editor, we sliced the videos to the exact moments where the barks were heard.
- We named the sliced videos with the breed and the assumed emotion of the dog in the video for reference.
- Then using VLC Media Player, we were able to convert these sliced video files into .wav files so that we could train them.
- Now to train the sounds, we found these effective ways of doing them-
    1. Derive the wave form images from the wav files.
    2. Use spectrograms of the wav files.
- We went ahead with the first method and divided the images generated properly into training and testing sets.

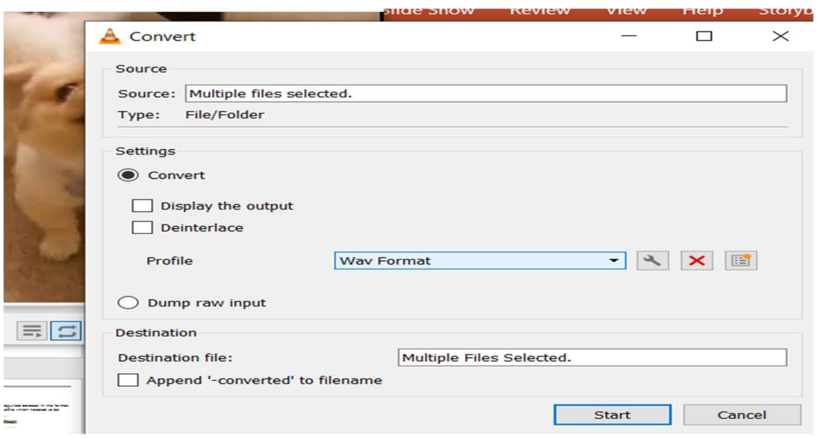
Thereafter we used a Deep CNN Model to train on this data to see if patterns are recognized

Screenshots

- ## Data Processing Phase



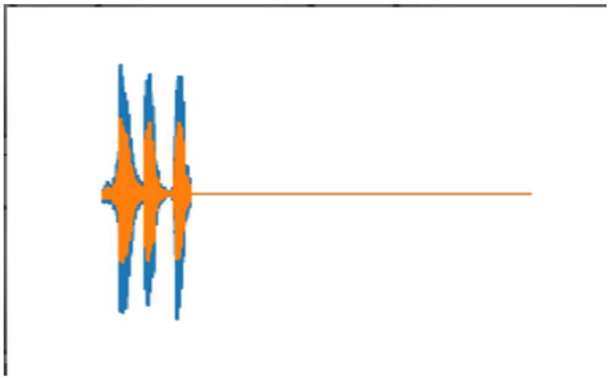
Trimming videos while tagging them.



Converting them into wav format using VLC

```python
# Make the Waveforms
def wav2img_waveform(wav_path, targetdir='', figsize=(4,4)):
    samplerate,test_sound  = wavfile.read(wav_path)
    fig = plt.figure(figsize=figsize)
    plt.plot(test_sound)
    plt.axis('off')
    output_file = wav_path.split('/')[-1].split('.wav')[0]
    output_file = targetdir +'/'+ output_file
    plt.savefig('%s.png' % output_file)
    plt.close()
```

Python Script to generate Wave Forms



A Sample Waveform

- ## Model Setup and Training

```
#WE WILL ADD ANOTHER CONVOLUTION/POOLING LAYER AS FOLLOWS
classifier1 = Sequential()

#Convolution1
classifier1.add(Convolution2D(32,3,3,input_shape = (64,64,3), activation = 'relu'))   # number of feature det

#Pooling1
classifier1.add(MaxPooling2D(pool_size = (2,2)))  #The size of filter is 2X2

#Convolution2
classifier1.add(Convolution2D(32,3,3, activation = 'relu'))   # number of feature detectors,number of rows, n

#Pooling2
classifier1.add(MaxPooling2D(pool_size = (2,2)))  #The size of filter is 2X2

#Flattening
classifier1.add(Flatten())

#classifier1.add(Dense(output_dim=5))

#Full Connection....Starting with ANN
classifier1.add(Dense(output_dim = 128, activation = 'relu'))
classifier1.add(Dense(output_dim = 5, activation = 'softmax'))  #Final Output

#Compiling the Model
classifier1.compile(optimizer = 'adam', loss=tf.keras.losses.CategoricalCrossentropy(), metrics = ['accuracy'
```

CNN Architechure

```
Epoch 1/10
200/200 [==============================] - 60s 301
0.0788 - val_loss: 1.2320 - val_acc: 0.1905
Epoch 2/10
200/200 [==============================] - 58s 290
0.2236 - val_loss: 1.7518 - val_acc: 0.2381
Epoch 3/10
200/200 [==============================] - 58s 289
0.2318 - val_loss: 2.5230 - val_acc: 0.4286
Epoch 4/10
200/200 [==============================] - 58s 290
0.2387 - val_loss: 3.6265 - val_acc: 0.3810
Epoch 5/10
```

CNN Training!

Results

The CNN output turned out to give a bad predictive response (which was expected to be fair), since our dataset included barks of dogs of various species.

That meant that a lot of dogs had naturally high/low frequencies and hence the model was not able to categorize the input properly. We require a dataset of a single animal over a period of time to classify emotions/responses based on that animals particular fluctuations.

Through this project, we have laid out a framework for future work wherein only the data needs to be collected and can be directly fed into this model.

The steps of:

- Making the videos
- Extracting the Data
- Splitting and tagging the barks
- Preprocessing the data
- Converting them into viable image formats
- Then finally training the images

have been explained and implemented in detail in this project.

Challenges Faced

- As mentioned before, collection of a quality dataset was the biggest issue we faced in this project
- Pre-Processing and Manually sorting the dataset was a very cumbersome task
- Some methods like MFCC and spectrograms demand that the wav file be of 1 second or less…which failed in our case since the video editor I used attached a “Brand” snippet of 2 seconds after each video.

But gladly, waveform representation can still be executed so we chose that format.

References

[1] https://www.kaggle.com/davids1992/speech-representation-and-data-exploration

[2] https://research.google.com/audioset/ontology/dog_1.html

Appendix

Full Code:

```
import matplotlib.pyplot as plt
from matplotlib.backend_bases import RendererBase
from scipy import signal
from scipy.io import wavfile
#import soundfile as sf
import os
import numpy as np
from PIL import Image
from scipy.fftpack import fft

%matplotlib inline

audio_path = 'C:/Users/PURUSHOTTAM/Desktop/Barks/WavFiles/Train/'
pict_Path = 'C:/Users/PURUSHOTTAM/Desktop/Barks/PicsGenerated/Train/'
test_pict_Path = 'C:/Users/PURUSHOTTAM/Desktop/Barks/PicsGenerated/Test/'
test_audio_path = 'C:/Users/PURUSHOTTAM/Desktop/Barks/WavFiles/Test/'
samples = []

#-----------------------------------------------------------------------------------
#CODE TO MAKE DIRECTORES AND FORM WAVE FORM IMAGES FROM THE AUDIO FILES GATHERED (WAV FORMAT)

subFolderList = []
for x in os.listdir(audio_path):
   if os.path.isdir(audio_path + x):
      subFolderList.append(x)
      if not os.path.exists(pict_Path  + x):
```

```
            os.makedirs(pict_Path + x)
        if not os.path.exists(test_pict_Path  + x):
            os.makedirs(test_pict_Path + x)



sample_audio = []
total = 0

for x in subFolderList:
    # get all the wave files
    all_files = [y for y in os.listdir(audio_path +x) if '.wav' in y]
    total += len(all_files)
    # collect the first file from each dir
    sample_audio.append(audio_path  + x +'/'+ all_files[0])

    # show file counts
    print('count: %d : %s' % (len(all_files), x ))
print(total)

# Make the Waveforms
def wav2img_waveform(wav_path, targetdir='', figsize=(4,4)):
    samplerate,test_sound  = wavfile.read(wav_path)
    fig = plt.figure(figsize=figsize)
    plt.plot(test_sound)
    plt.axis('off')
    output_file = wav_path.split('/')[-1].split('.wav')[0]
    output_file = targetdir +'/'+ output_file
    plt.savefig('%s.png' % output_file)
    plt.close()

#Build the training pics
for i, x in enumerate(subFolderList[:3]):
    print(i, ':', x)
    # get all the wave files
    all_files = [y for y in os.listdir(audio_path + x) if '.wav' in y]
    for file in all_files[:]:
        wav2img_waveform(audio_path + x + '/' + file, pict_Path+ x)
```

```
#Build the testing pics
for i, x in enumerate(subFolderList[:3]):
    print(i, ':', x)
    # get all the wave files
    all_files = [y for y in os.listdir(test_audio_path + x) if '.wav' in y]
    for file in all_files:
        wav2img_waveform(test_audio_path + x + '/' + file, test_pict_Path + x)


#------------------------------------------------------------------------------------

#BEGINNING THE TRAINING

import tensorflow as tf
import keras
config = tf.ConfigProto( device_count = {'GPU': 1 , 'CPU': 5} )
sess = tf.Session(config=config)
keras.backend.set_session(sess)

#CNN Building
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import model_from_json


#WE WILL ADD ANOTHER CONVOLUTION/POOLING LAYER AS FOLLOWS
classifier1 = Sequential()

#Convolution1
classifier1.add(Convolution2D(32,3,3,input_shape = (64,64,3), activation = 'relu'))   # number of feature detectors,number of rows,
number of columns, input shape{for theano backend=>(3,64,64)}

#Pooling1
classifier1.add(MaxPooling2D(pool_size = (2,2)))  #The size of filter is 2X2
```

```
#Convolution2

classifier1.add(Convolution2D(32,3,3, activation = 'relu'))   # number of feature detectors,number of rows, number of columns, {NO
NEED HERE=>}input shape{for theano backend=>(3,64,64)}


#Pooling2

classifier1.add(MaxPooling2D(pool_size = (2,2)))  #The size of filter is 2X2


#Flattening

classifier1.add(Flatten())


#classifier1.add(Dense(output_dim=5))


#Full Connection....Starting with ANN

classifier1.add(Dense(output_dim = 128, activation = 'relu'))

classifier1.add(Dense(output_dim = 5, activation = 'softmax'))  #Final Output


#Compiling the Model

classifier1.compile(optimizer = 'adam', loss=tf.keras.losses.CategoricalCrossentropy(), metrics = ['accuracy'])


#Fitting the image


from keras.preprocessing.image import ImageDataGenerator


#Code taken from Keras Documentation

train_datagen = ImageDataGenerator(

    rescale=1./255,

    shear_range=0.2,

    zoom_range=0.2,

    horizontal_flip=True)


test_datagen = ImageDataGenerator(rescale=1./255)


training_set = train_datagen.flow_from_directory(

    'C:/Users/PURUSHOTTAM/Desktop/Barks/PicsGenerated/Train',

    target_size=(64, 64),

    batch_size=32,

    class_mode='binary')
```

```
test_set = test_datagen.flow_from_directory(
        'C:/Users/PURUSHOTTAM/Desktop/Barks/PicsGenerated/Test',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

classifier1.fit_generator(
        training_set,
        steps_per_epoch=200,    #Total images we have
        epochs=10,
        validation_data=test_set,
        validation_steps=1000
        )   #Total images we have
```

```
#Saving the model
# serialize model to JSON
model_json = classifier1.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
classifier1.save_weights("model.h5")
print("Saved model to disk")
```

```
#Loading the Model
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
classifier1 = model_from_json(loaded_model_json)
# load weights into new model
classifier1.load_weights("model.h5")
print("Loaded model from disk")
```

```
#-------------------------------------------------------------------------------
# Let's try with the spectogram representation

import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile

sample_rate, samples = wavfile.read('C:/Users/PURUSHOTTAM/Desktop/Barks/WavFiles/Test/Angry/28_Sd_A_Angry-converted.wav')
frequencies, times, spectrogram = signal.spectrogram(samples, sample_rate)

plt.pcolormesh(times, frequencies, spectrogram)
plt.imshow(spectrogram)
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()

#-----------------------------------------------------------------------------------
import librosa
sample_rate, samples = wavfile.read('C:/Users/PURUSHOTTAM/Desktop/Barks/WavFiles/Test/Angry/28_Sd_A_Angry-converted.wav')
S = librosa.feature.melspectrogram(samples, sr=sample_rate, n_mels=128)
```