Orcasound

Proposal- Google Summer of Code 2020

Personal Details

Name- Mohit Saini
Email- sainimohit23@gmail.com
University- Guru Gobind Singh Indraprastha University, New Delhi, India
Telephone- (+91) 8076677127, (+91) 8377810420
LinkedIn- https://www.linkedin.com/in/sainimohit23/
GitHub- https://github.com/sainimohit23
Medium- https://medium.com/@mohitsaini_54300

About Me:

I am a final year computer science engineering student at Ambedkar Institute of Advanced Communication Technologies and Research, Delhi. Currently, I am working as a data science intern with Yatra. My area of interest is deep learning and these days I am more into recent advancements like BERT, Transformers, ELMO, GPT-2, RASA, etc. that are happening in the field of NLP. I am consistently working on expanding my skill set so that I can grow as a developer that can build end-to-end AI solutions.

I have experience of more than two years in developing deep learning models. I have worked on several personal projects. I also bring experience of working in different roles at various startups and companies with me.

Last year I also tried to participate in GSoC through ESIP's project titled- 'OrcaCNN- Detecting and Classifying Killer Whales from Acoustic Data'. For that, I built the whole working project using the ideas that I had learned over time. The project is still available on my GitHub. So, I have prior experience and a little head start that will suit this project.

Skills- Python, C++, DS, Algorithms, Tensorflow, Keras, Computer Vision, NLP, MySQL, Docker, Javascript, Flask, REST APIs, HTML & CSS.

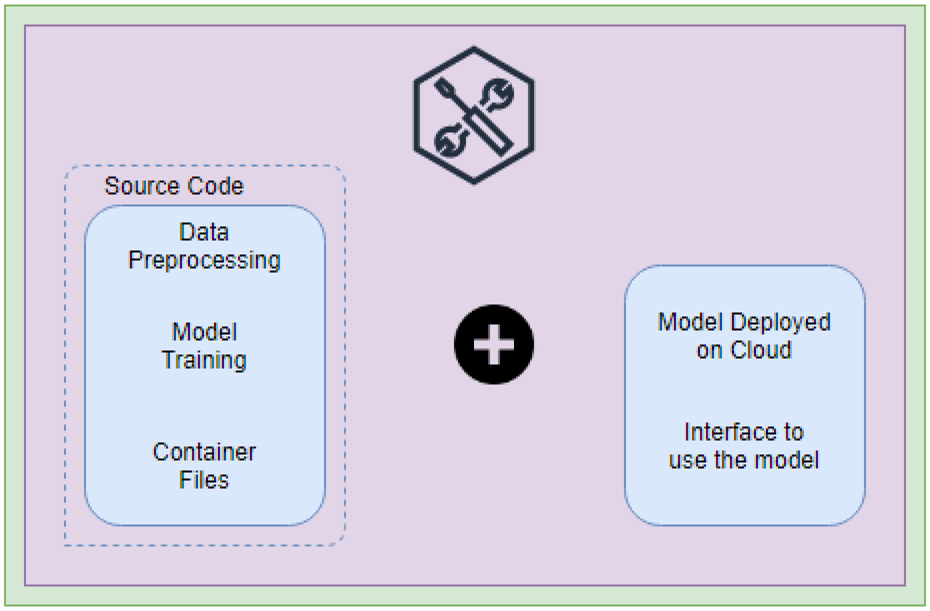
Title: Real-Time Orca Call Detection in No Time

Abstract:

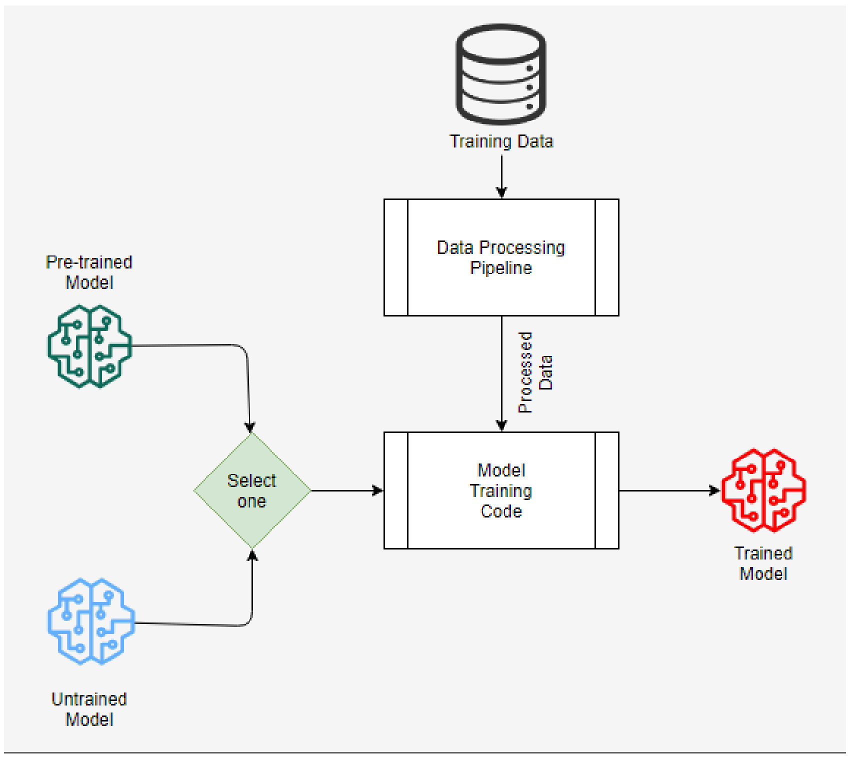
For this project, I am proposing a **real-time orca prediction toolkit** that can be used by researchers around the world with or without programming knowledge. The toolkit will have the following features-

- The toolkit will provide a standard model trained by the organization that users can use to detect the orca sounds in real-time. This model will be deployed using **Flask API** and **Docker** on any cloud service so that the users can access it remotely.
- The source code of the toolkit will provide the functionality to finetune the existing model or train a new model from scratch.
- A data processing pipeline to preprocess the audio data for the training of the model.
- The source code of toolkit will also provide a **Dockerfile** and related scripts for the users who want to create their own containers.
- Users will be able to upload their audio files from their local machines on the server to get the predictions.
- Well written tutorials for every use case scenario. Alternate Jupyter notebook for some Python scripts.

So, according to the above points, we can divide the toolkit into two broad categories. One is the source code related to the training and deployment of the model, and other is model hosted on cloud and it's interface.



I will try to build a CLI based interface for the toolkit that requires no prior programming knowledge. I won't do anything fancy to increase the complexity of the project. Models will be lightweight and users will be able to run them on their local system.



Schematic diagram of the model training pipeline(for users)

Technical Details:

In this section, I will go through the technical aspects of the various parts of the toolkit.

Real-Time Orca Detection from Model Running on Docker

First, let's talk about **Docker**. With the source code of the toolkit, I will provide a **Dockerfile** to create a container that will host a web-based **API** of our model. This **Dockerfile** will have the code to set up the correct Linux environment with the required packages. The **RESTFul Flask** server running on this Docker will host the orca detection model. Containerizing the code will help us in saving the resources and managing the code dependencies.

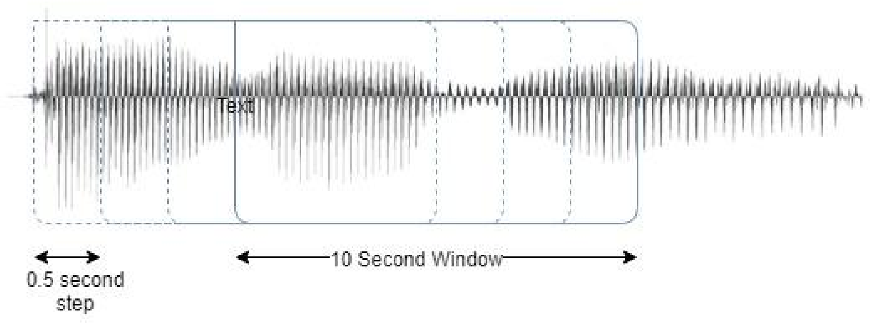
The API running on the cloud service will take an audio file as an input and it will return the start times and end times of orca sounds in the audio.

If a single instance of the Docker container is unable to handle the user requests, then we need to scale up by creating multiple instances of the Docker container, and to manage all this **Kubernetes** will be used.

Now let's take a look at how I will achieve real-time detection of Orca Calls.

The limitation of deep learning models that I will be using is that they only work on audio data with a fixed shape or length. But, the input audio is going to be of variable length. So, to tackle this problem, the code will break down the input audio into smaller fixed-length chunks at regular intervals. If the length of the input audio is smaller than the chunk size, then the program will pad it with silence. If the length of audio is larger than the chunk size then the program will trim the audio in smaller sized chunks.

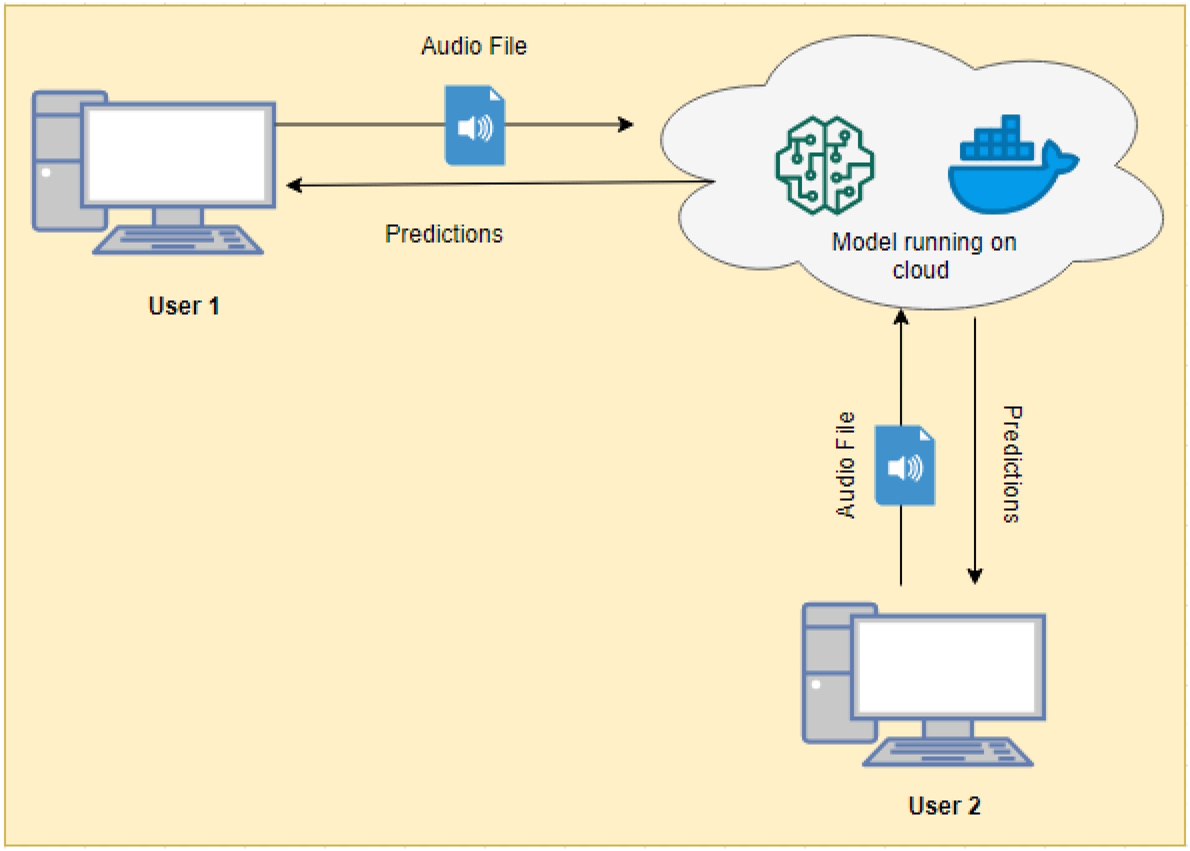
Just breaking the audio at regular intervals of time is not a good idea because we might encounter a case where orca sounds will get divided into two separate chunks and our model will miss it. To avoid this we will create a moving window of 10 seconds and it will be updated every fixed small interval of time(let's say 0.5 seconds). Meaning for every 0.5 second, the oldest 0.5 second chunk of audio will be discarded and the fresh 0.5 second audio will be shifted in.



We will feed these chunks one by one to our orca detection model to get the outputs. Now, another problem arises. The forward chunks created by the sliding window will overlap with previous chunks, thus their outputs will also overlap. So, the final output will be generated by stitching up the predictions of individual chunks. Here is an **Ipython notebook** for reference- https://github.com/sainimohit23/OrcaCNN-Demo/blob/master/4%20Model%20Pipeline/FullModel.ipynb.

The same approach can be used in live audio streaming of orca sounds. But, there is another problem with the live audio streaming. Prediction models take some time to generate the output. So, by just dividing the audio stream into chunks will create gaps in the detection process.

Pyaudio library has the ability to read audio streams asynchronously. This will help us to record the audio in one thread and when a new fixed length of audio data is available, it will notify our model to process it in the main thread.



Orca Detection Model

Depending on whether we are concerned with the orca pod classification, models can be divided into two broad categories-

1. If we want to detect the presence of orca in audio, then we can use a binary classification model that detects the presence and absence of orca sounds.
2. Along with the presence of orca, if we also want to classify the orca pod then we can use a model with a slightly different architecture.

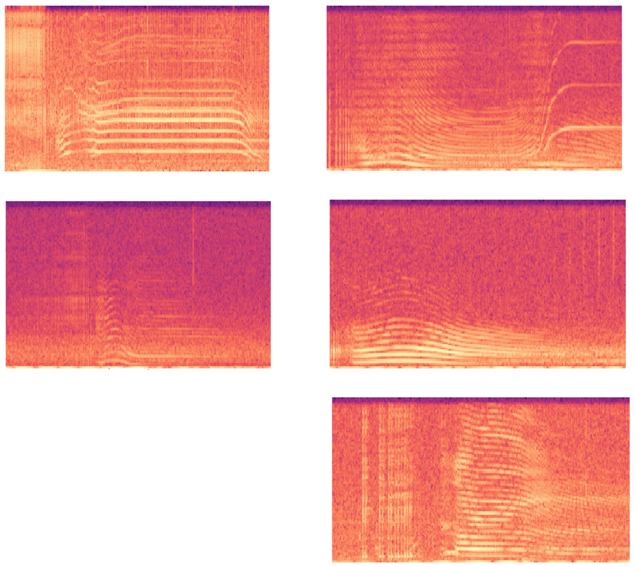
Based on the requirement of the temporal start and end time of the orca calls, we can further categorize our models.

Now, I will explain the approaches for the orca detection models. One thing I want to make clear is that these deep learning models work on static data. So, the detection in real-time with audio streaming is handled separately.

1. Binary Classification models

If we just want to detect the presence of orca then we can opt for the following approaches.

a. CNN Model- The Mel-spectrogram of orca sounds are visually very distinctive. We can use this property of orca sounds to train a Convolutional Neural Network for the task of binary classification of orca sounds. This method is very effective if we just want to detect the orca sounds. Precise starting and ending time of orca sounds can't be estimated using this method.



Spectrograms of orca sounds

A simple lightweight CNN model with a sigmoid output layer would be enough for this task. The input of the model will be a spectrogram on fixed-length audio and the output will be a number between 0 and 1, showing the probability of an orca call.

b. RCNN Based Model- If the precise time of orca sounds is also required then we can use this technique. I used this model for my last year's GSoC proposal. In this, we will train our model for the problem of **Wake Word Detection**. The models trained on this problem statement are used to detect the trigger words.

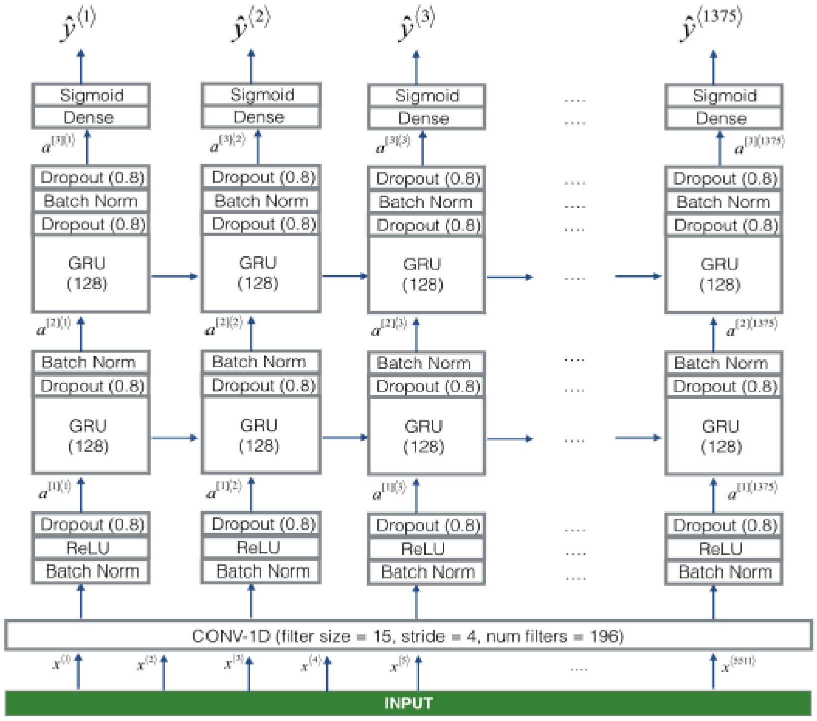
At an abstract level, the model takes a Mel-spectrogram of fixed length audio as an input and it returns a stream of numbers between 0-1 as output. Each number of the output stream represents a certain timestamp. If we get a continuous stream of ones counting higher than a certain threshold (let's say 25), then we say that we have detected an Orca.

Model details

Before diving into further details, first, let's define certain variables that will help me to explain the mathematics and technical aspects of the model.

- Sampling rate of input audio: 44100 samples/sec.
- Size of input audio: 10 seconds (i.e. 441000 samples).
- Tx = 5511 (samples of input spectrogram).
- Ty = 1375 (size of the array of 0's and 1's returned by the RNN model).

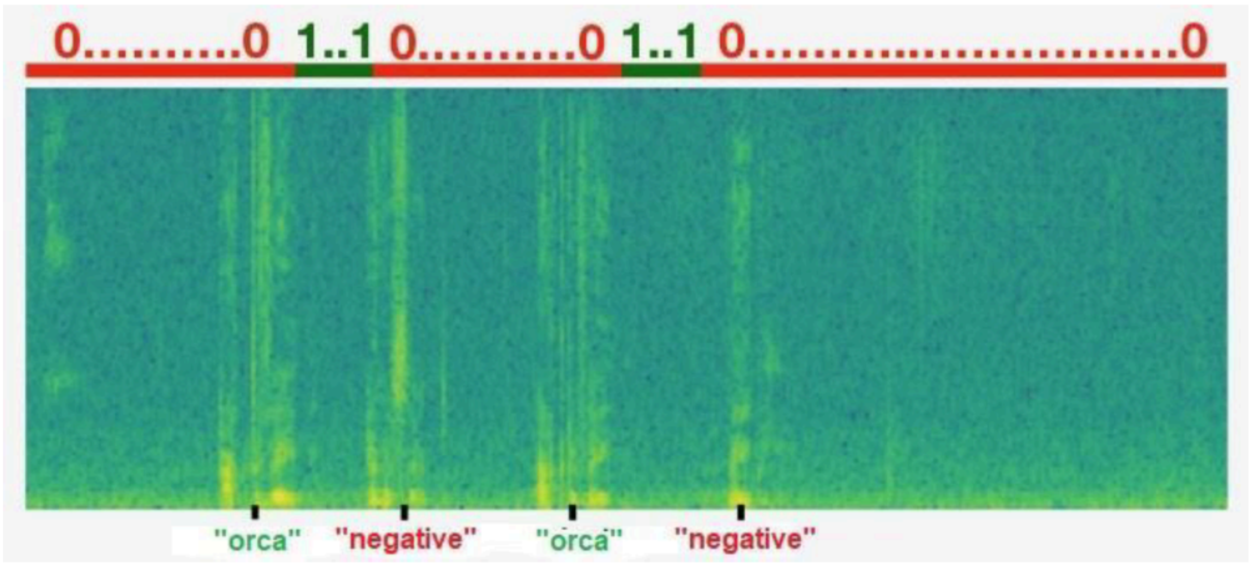
Each step of raw audio represents 10/441000 = 0.000023 seconds.
Each step of Tx represents 10/5511 = 0.0018 seconds.
Each step of Ty represents 10/1375 = 0.0072 seconds.



$\hat{y}^{\langle 1 \rangle}$
$\hat{y}^{\langle 2 \rangle}$
$\hat{y}^{\langle 3 \rangle}$
$\hat{y}^{\langle 1375 \rangle}$
Sigmoid
Dense
Dropout (0.8)
Batch Norm
GRU (128)
ReLU
CONV-1D (filter size = 15, stride = 4, num filters = 196)
INPUT

The input audio will be first converted to a spectrogram. The spectrogram will have the shape: **(101, 5511) [parameters: nfft=200, noverlap=120]**. This spectrogram will act as an input for the model. The model will first perform a 1D convolution on the spectrogram. The **1375 dimensional** output array of the **Conv1D** layer will be further processed by the multiple layers of **Gated RecurrentUnit (GRU)** cells to get the final **Ty=1375** step output. The 1D convolutional layer plays a role similar to the 2D convolutions of extracting low-level features and then possibly generating an output of a smaller dimension. Computationally, the 1D convolution layer also helps speed up the model because now the GRU has to process only 1375 timesteps rather than 5511 timesteps. The two GRU layers read the sequence of inputs from left to right, then ultimately use a **{dense + sigmoid}** layer to make a prediction for Yt. Because Y is binary-valued (0 or 1), we use a sigmoid output at the last layer to estimate the chance of the output being 1.

Let's assume we get a stream of '1s' starting at the position 850 in the output array. Then the precise time of the ending of the Orca call will be: (10/1375) * 850 = 6.18 second.



The model that I have explained above gives just the ending time of the orca sound. With slight modifications in the training procedure. The above architecture can be used to get both the starting and ending times of orca sounds.

Since the input to the model is the spectrogram of fixed length audio. We can further optimize the performance of the model by using bidirectional GRUs.

2. Models that can detect as well as classify the orca pods

The two architectures that I have explained above can also be tweaked to classify the orca sounds into their respective pods. So, let's discuss both architectures one by one.

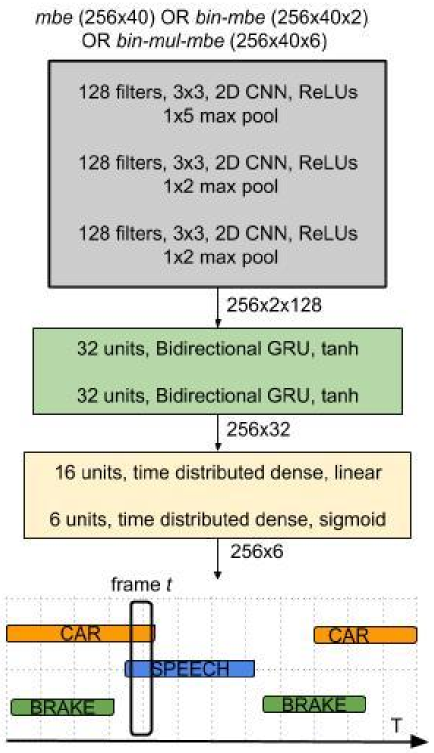
a. CNN Model- For CNN classifier it is actually very simple. We just have to change the output layer from sigmoid to softmax and we have to train the model for the classes: (number of orca pods) + 1 using the **categorical cross-entropy** loss function. The extra class is for other types of sounds.

b. RCNN Model- A modified version of the RCNN model that we discussed earlier is presented in the paper- Sound Event Detection using Spatial Features and Convolutional Recurrent Neural Networks. In this model, the inputs will be the same but the output layer will be a softmax layer. The model will output a stream of 0s and 1s for each class, signifying the presence of each class in the input audio.

In the paper, it is also described that by employing the same RNN layers on different channels of the input audio and concatenating their outputs, the model can learn the inter-channel features.

According to the authors, the model is tested for the following problem statements-

- Sound event detection
- SED with weak labels
- Bird audio detection
- Music emotion recognition



mbe (256x40) OR *bin-mbe* (256x40x2) OR *bin-mul-mbe* (256x40x6)
128 filters, 3x3, 2D CNN, ReLUs
1x5 max pool
128 filters, 3x3, 2D CNN, ReLUs
1x2 max pool
128 filters, 3x3, 2D CNN, ReLUs
1x2 max pool
256x2x128
32 units, Bidirectional GRU, tanh
32 units, Bidirectional GRU, tanh
256x32
16 units, time distributed dense, linear
6 units, time distributed dense, sigmoid
256x6
frame *t*
CAR
CAR
SPEECH
BRAKE
BRAKE
T

Image is taken from the paper

Depending on the requirements of the organization. I can build the toolkit around any of the architectures mentioned above.

NOTE: The evaluation of the performance of the models will be done using the SED EVAL python library that Scott Veirs shared on the GitHub issue page. It provides the implementation of metrics like Precision, Recall, F-Score, Sensitivity, etc.

Training Data Preparation

In my opinion data preparation is going to be the most rigorous and time-consuming task of this project. Although the input formats of data and labels of both the model architectures are quite different. But there are some common processing steps that are needed to be done. So, let's first talk about the steps that are common.

Based on last year's experience, I am assuming that the data is not going to be structured or labeled. So, to prepare the training data. First, I will collect positive and negative audio samples manually. Then I will do the manual trimming of the audio clips so that the clips just have the orca sound and nothing else. If required, the audio clips will be classified according to orca pods.

Apart from positive and negative orca samples. 10 second long audio clips of background audio will be required. Reason? I will tell you in a moment.

After the manual work, now following are the preprocessing steps that are needed to be performed to make the data consistent-

- Standardizing the frequency of audio.
- Standardizing the number of channels (it will affect the architecture of the model).
- Standardizing the format of audio i.e. .wav, .mp3 etc.

I will use the **Pydub or Librosa** library to perform the above transformations. Now, let's see the model-specific data preparation steps-

CNN Model- We Don't have much to do in the case of the CNN model. We just have to fix the size of input audio clips and generate the spectrogram. The length of the audio should be fixed so that the generated spectrograms are consistent. After all this, we just need to prepare the labels for each class of the input data.

RCNN Model- The input data of RCNN is going to be a bit different from the CNN model. To prepare the data of the RCNN model, I will write a python code that will do the following operations to generate the training samples-

- Randomly pick up a background audio clip of 10 seconds in size.
- Randomly select positive and negative audio samples, and superimpose them on the background audio clip.
- Generate a label file corresponding to each training audio sample.

The audio data generated by the above steps will be converted to the spectrogram in the final processing step.

These are the exact preprocessing steps that I used last year-
https://github.com/sainimohit23/OrcaCNN-Demo/tree/master/1%20Input%20Standardizer

Tech Stack and References

Tech Stack-

- Python
- Keras
- Docker
- Flask

References-

- https://www.coursera.org/learn/nlp-sequence-models
- https://github.com/jaimeps/whale-sound-classification
- https://github.com/axiom-data-science/OrcaCNN
- https://github.com/Tony607/Keras-Trigger-Word/blob/master/trigger_word_real_time_demo.ipynb
- https://frontendmasters.com/courses/complete-intro-containers/
- https://medium.com/@shamir.stav_83310/lets-create-a-cli-with-python-part-1-ae4fe9e0258b (found this through Abhivav's proposal)
- https://tut-arg.github.io/sed_eval/sound_event.html
- https://arxiv.org/pdf/1706.02291.pdf

Schedule of Deliverables:

Community bonding period	May 5 - May 31	- Discussing the project roadmap with the mentors - Exploring the possibilities - Read DSP literature - Setting up the development environment
Phase 1	June 1 - June 14	- Setting up the data processing pipeline - Preparing data for different models - Training different models
	June 15 - June 28	- Evaluating the performance of models - Finalizing the model for the project - Preparing OOPs based modular code around the selected model and data processing pipeline
Phase 1 Evaluation	June 29 - 3 July	
Phase 2	July 4 - July 18	- Improve on evaluation - Preparing Flask API - Code the script to stream the data on flask API
	July 18 - July 26	- Setting up a Docker environment and Dockerfile - Deploying model on the cloud - Writing the documentation and tutorials
Phase 2 Evaluation	July 27 - July 31	
Final Work Period	Aug 1 - Aug 24	- Improve on evaluation - Extensive testing of the toolkit - Fixing bugs - Submitting the final code and documentation.
Final Week	Aug 24 - Aug 31	- Submit the final work

Development Experience

Internships:

Yatra Online Pvt. Ltd. | Data Science Intern

Feb 2020 - Present | Gurugram

- Wrote a python script to automatically update the database daily on AWS redshift using crontab.
- Development of YATRA B2E email chatbot and YATRA FAQ.
- Worked on Flask APIs for models and created their docker containers.

CRIO.DO | Crio Launch Micro experience

Jan 2020 - Mar 2020| Virtual

- Internships like micro-experience by working on real products.
- Worked on the development of Qbox- a secured file-sharing solution for enterprise networks, Qmoney- a personal stock portfolio analysis and recommendations platform and Qcharm- a code editor.

Metvy | Machine Learning Intern

Nov 2019 - Jan 2020 | Gurugram

- Worked on the development and improvement of existing recommendation systems.
- Developed an NLP based job recommender system.
- Time series prediction using RNNs.

The Research Nest | AI Blogger

Nov 2019 - Jan 2020 | Work From Home

- Analyze the latest research and developments in the field of AI and derive useful insights.
- Wrote well-researched blogs on topics like parking space detection, GPT-2 and voice cloning. Medium Link- https://medium.com/@mohitsaini_54300

Athancare | Deep Learning Research Intern

Feb 2020 - Present | Gurugram

- Worked on the development of the handwriting OCR pipeline using siamese convolutional neural networks.
- Implemented research papers on line segmentation. Implemented SigNet paper.

Personal Projects:

Parking Space Detection System- Developed a real-time parking space detection system pipeline based on NVIDIA's research blogs using tools like matplotlib, Mask-RCNN, and Intersection over union. Github Link- https://github.com/sainimohit23/parking.

Real-Time face Recognition using Facenet- Developed a pipeline for the training of facenet model on triplet loss. The program uses a webcam to recognize faces in real-time. Github Link- https://github.com/sainimohit23/FaceNet-Real-Time-face-recognition.

AI Chatbot- Developed a chatbot using seq2seq transformers, flask and react on papaya dataset for final year minor group project. The chatbot can remember names in sessions and it can also perform simple calculations and tell jokes and stories.

Orca detection and classification from acoustic data- Curated training data from raw audio samples. Developed RNN based orca detection model using the concept of wake word detection. Developed RNN+CNN based orca pod classification model. Worked on this to secure a slot in GSoC 2019. GitHub Link- https://github.com/sainimohit23/OrcaCNN-Demo.

Yolov3 on CSGO- Trained YOLOv3 object detection model to recognize Terrorists and Counter-Terrorists in CSGO gameplay. I prepared the whole training dataset by myself from scratch. I also made an image labeling tool to prepare the dataset for training. Github Link- https://github.com/sainimohit23/YOLOv3-Counter-Strike-Global-Offensive.

ECG Image Classification- A CNN classification model for the ECG data. The model takes the images of spectrograms of the ECG and then classifies it into 4 categories. Github Link- https://github.com/sainimohit23/ECG-image-classification/tree/master.

Text Summarization- Developed a seq2seq encoder-decoder model for the task for text summarization. Github Link- https://github.com/sainimohit23/Text-Summarization.

Image Captioning- Built a hybrid model for image caption generation using VGG-16 and RNN layers. Github Link- https://github.com/sainimohit23/Image-Captioning.

Other

Other Experiences:

- Google Code-In Mentor for Tensorflow organization (Nov 2019 - Feb 2020)- I was selected by the **Tensorflow** organization to mentor the pre-university students in the google code-in program. I helped around 15 students during the program.
- Completed Hacktoberfest 2019.
- Runner up in Coding Blocks DS-Algo hackathon.
- Contributed to CLTK.

Certifications and Courses:

- Algorithms Design and Analysis (Stanford Lagunita)
- Data Structures and Algorithms in C++ (Coding Blocks, Delhi)
- Stanford Machine Learning (Coursera)
- Deep Learning Specialization (Coursera)
- Tensorflow Specialization (Coursera)
- Building Recommender Systems using ML & AI (Udemy)
- Web Developer Bootcamp (Udemy)
- Flask Tutorial (Miguel Grinberg)
- Complete Introduction to Containers (FrontendMasters)

Personal goals for the upcoming weeks:

- I feel that my OS and Networking Concepts are a little weak. So, I will be reading these two books-
  - Computer Networking: A top-down approach.
  - Operating System Concepts

Activities/Hobbies:

- I have been part of my college and school basketball teams.
- Sometimes I like solving problems on coding platforms. I try to participate in leetcode weekly challenges.
- I like to play competitive games like CSGO and Rainbow Six Siege a lot.
- Up until January, I was regular in the gym for about 2 years.

Why this project?

As I have mentioned earlier, my connection with the task of orca detection goes back to last year's GSoC. Since I have a huge interest in the field of AI, therefore I have been doing internships and working on my personal projects in this field for the past two years. In the industry, most of the time I am working on generic tasks like chatbot development, recommendation systems, image classification, etc. and there's isn't much to explore in terms of the applications of AI. This is the prime reason I am interested in this project. The whole idea of applying machine learning to detect orcas is really fascinating for me and it would be an honor to work for the research community.