Real-Time Orca Call Detection in No Time

OrcaSound || Atreya Majumdar

- **Name: Atreya Majumdar**
- **Timezone: Indian Standard Time - GMT + 5:30**
- **E-Mail: atreyamaj@gmail.com**
- **LinkedIn: https://www.linkedin.com/in/atreyamajumdar/**
- **Slack Name: Atreya Majumdar**

About Me:

I am a sophomore pursuing B.Tech. Electronics and Communication Engineering from the National Institute of Technology Karnataka, India. When I started university, I was exposed to signal processing and the applications of machine learning, in my engineering curriculum, as well as in various technology clubs . I saw how models were planned and built by observing my seniors working on signal classification projects. This helped me to develop an interest in Signal Processing applications using Deep Learning models. I am currently working in the fields of:

1. Deep Learning/Neural Networks
2. Signal and Image Processing
3. Computer Vision
4. Natural Language Processing

Previous Development Experience:

I have worked on the following Signal Processing and Deep Learning projects:

1. **Urban Planning and Landscape Prediction using Deep Learning**, a project initiated by the Indian Space Research Organization (ISRO) under the ISRO-NITK MoU, where I have been selected to work under Prof Shyam Lal, Dept. of Electronics and Communication, NITK. We have currently implemented a few IEEE research papers: *Unsupervised Change detection in satellite images using Principal Component Analysis and K-means clustering*, and *Automatic analysis of Difference images for unsupervised change detection*. We plan on finding the most accurate model and using that to perform the landscape prediction in order to optimize land usage for future generations. The technology used will be DCNN's, RNN's and RCNN's to determine which will be most efficient and accurate and will be further worked upon. As of today, we are building the models using CNN.

2. **Optical Character Recognition Module for the blind** which works in real-time on video using OpenCV's EAST algorithm, a Deep Learning based text detection pipeline. The model takes video input from a camera, extracts text and converts it into speech. I have finished making the model and implementing it. This project's code has been uploaded to Github **HERE:** (https://github.com/atreyamaj/OCR-EAST).

Why am I interested in this project?

I wish to contribute to the conservation and sustainability effort to the best of my ability. I strongly believe that the wielders of cutting edge technology can shape the future of not only the tech-world, but also the planet as a whole by using technology in a constructive manner in order to achieve an ecological balance with nature. The endangerment of Orca whales is a growing concern. I wish to explore how we as a community can use Audio/Signal Processing and Deep Learning in order to locate and protect them. We can create a more sustainable world for not just our future generations, but the future generations of all species on this planet.

I always wanted to be part of the open source community but didn't know where and how to start contributing. Google Summer of Code is a great opportunity to enter and participate in this amazing community. I hope that I can contribute to the open source community to the best of my ability.

Abstract:

The Orca call database is fairly voluminous, therefore there are inevitable instances of label noise being present. I plan on de-noising the original audio signal first, converting it to a spectrogram, and training a detection model on the processed data. After this initial training is done, I plan on further training the model with the same data but augmented with noise. This is done in order to perform binary classification of Orca calls with satisfactory accuracy even with fairly high levels of noise.

The aim of the project is to make an accurate model light enough to run on a personal computer, and not just on a cloud computing platform. The model must be able to classify the input audio signal as fast as possible. Thus it must be as optimized and as lightweight as possible while simultaneously minimizing processing costs. In this project, I have elaborated upon how we can do this and the various tools at our disposal that will aid us in achieving the same.

Technical Details:

Required Packages and relevant work: We will be referring to OrcaCNN from ESIP. Packages that will be used include Keras, Pandas, open-unmix-pytorch [3], OS, sys, io, tensorflow/tfjs, numpy, Pydub, Librosa.

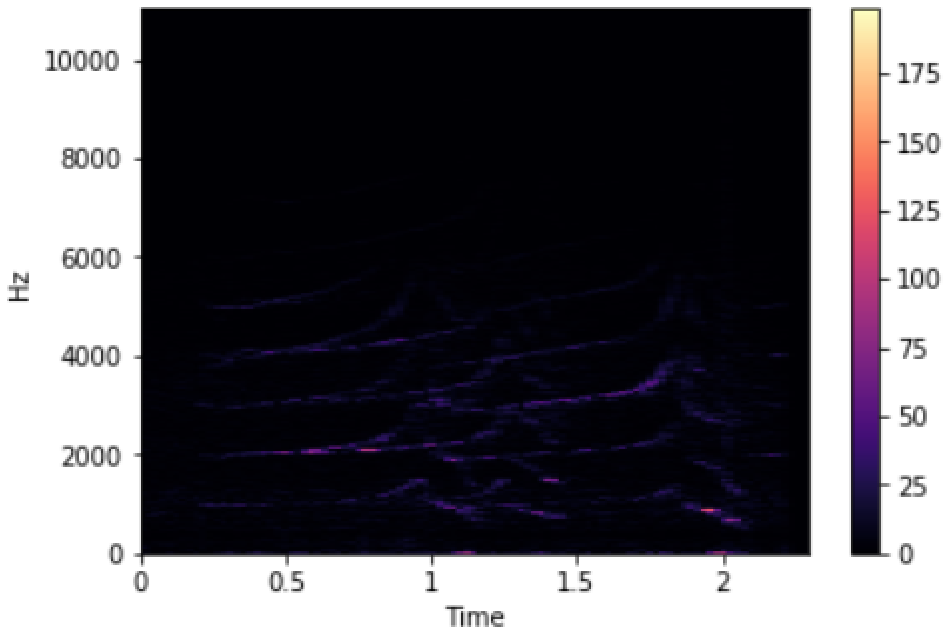
The steps I plan on following during the course of the project are:

1. **Preparing the Audio Data:** In this stage, I will be using the Open-unmix package in order to separate noise (for example: ship sounds, other creatures, background noise etc.) from the given audio signals and will store only the Orca sounds. This will

then be used to compute spectrograms using the Librosa library for each signal which will then be used to train the model.

We are using spectrograms instead of Fourier Transforms because the Fourier Transform plots appear to be pitch black.

The code to compute the Fourier Transform Plot, spectrogram and Mel-spectrogram for a 2 second J-Pod can be found at https://github.com/atreyamaj/GSoC-Proposal-Orcasound-Project-2



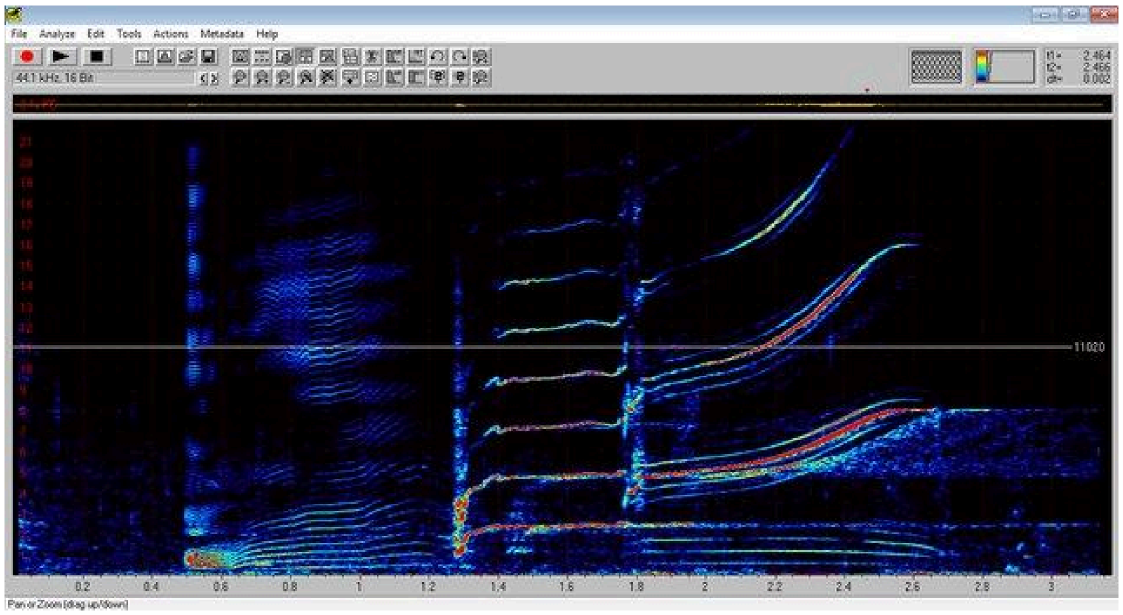
A demonstration of what happens when a Fourier Transform plot is made for an audio recording of a 2 second J Pod Orca call.

I used Librosa to generate this plot.

Not much is visible as most sounds that humans can hear are concentrated in very small frequency and amplitude ranges, which the Fourier Transform cannot handle.

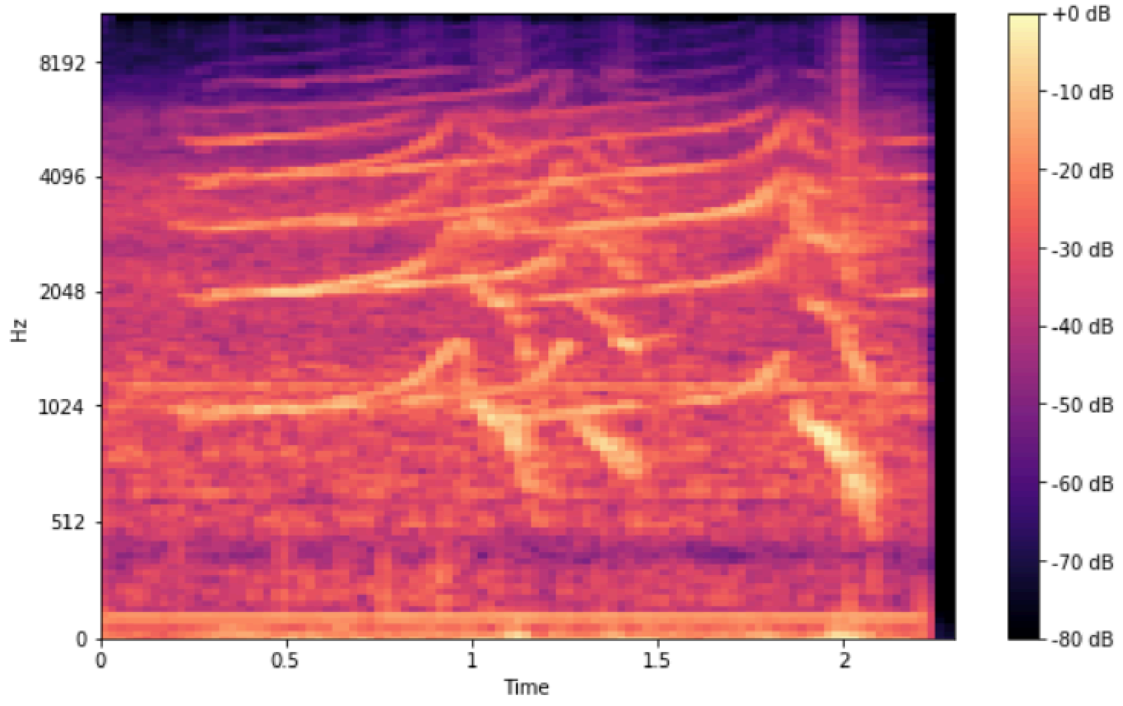
A spectrogram defines a signal in terms of time, frequency and intensity/magnitude. When the Y axis of a spectrogram is the Mel scale, the spectrogram is called a Mel-spectrogram. Mel-spectrograms are used because the Mel-frequency scale is a quasi-logarithmic spacing, which roughly resembles the resolution of the human auditory system. We are converting the audio signals to mel-spectrograms because the mel-spectrogram of an orca sound is visually unique and distinctive.

This is followed by feeding the model the original data which has background noise.



Spectrogram of an Orca's call

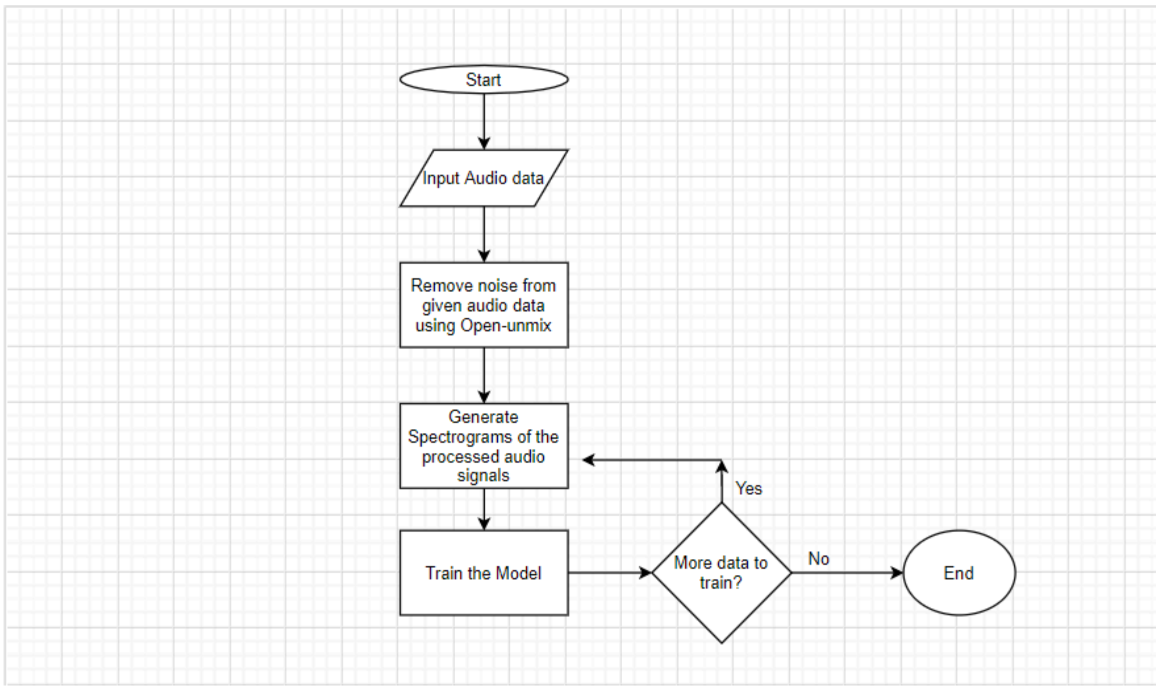
Source:
https://www.researchgate.net/publication/319527386_Vocal_repertoires_of_two_matrilineal_social_whale_species_Long-finned_Pilot_whales_Globicephala_melas_Killer_whales_Orcinus_orca_in_northern_Norway/figures?lo=1



,

What a Mel-spectrogram of a 2 second J-Pod Orca call audio clip looks like.

I used Librosa to generate the mel-spectrogram.


Start
Input Audio data
Remove noise from given audio data using Open-unmix
Generate Spectrograms of the processed audio signals
Train the Model
More data to train?
Yes
No
End

A basic Flowchart describing the workflow of the first stage of the project

2. **Building the Orca Call Binary Classification Model:** In this stage, we will follow the following steps:
   A. First, we will train the model from OrcaCNN and note the results/accuracy on our modified dataset. The parameters governing the accuracy of the model are: Pitch, duration, intensity, audio quality and SNR (Signal to Noise Ratio). Each of these factors contribute to the change in the spectrogram of a signal. I am doing this in order to compare the results with that of the CNN I am building.
   B. Then, we will proceed to make our own model using a Convolutional Neural Network (CNN). I am not using a pre-trained model because I want to augment the dataset with various levels of noise and have a new and larger synthesized dataset before training the model. This will ensure that the model can still work with a significant amount of background noise and classify the signal in no time.

C. I am choosing to work with a CNN because:

1. CNNs work well on image classification problems due to their feature extraction and classification.
2. Spectrograms have time as the horizontal axis while the vertical axis corresponds to frequency. This means that a spectrogram is just a time series of frequency measurements. Keeping this in mind, convolution seems to be an appropriate choice to extract the patterns made by Orca calls.
3. I will be using 2D Conv for my CNN with RELU activation. Since spectrograms define signals using 3 quantities: time, frequency and magnitude, but have two directions, it would be better to implement 2D convolution. 1D convolution is usually used if we intend on using audio signals as direct input for the model.

As of this moment, I plan on implementing my CNN architecture in the following manner:

1. Convolution layer with kernel size 3x3
2. Convolution layer with kernel size 3x3
3. Max-pooling layer with pool size 2x2
4. Dropout layer
5. Flattening layer
6. 2 Dense layered neural networks at the end.

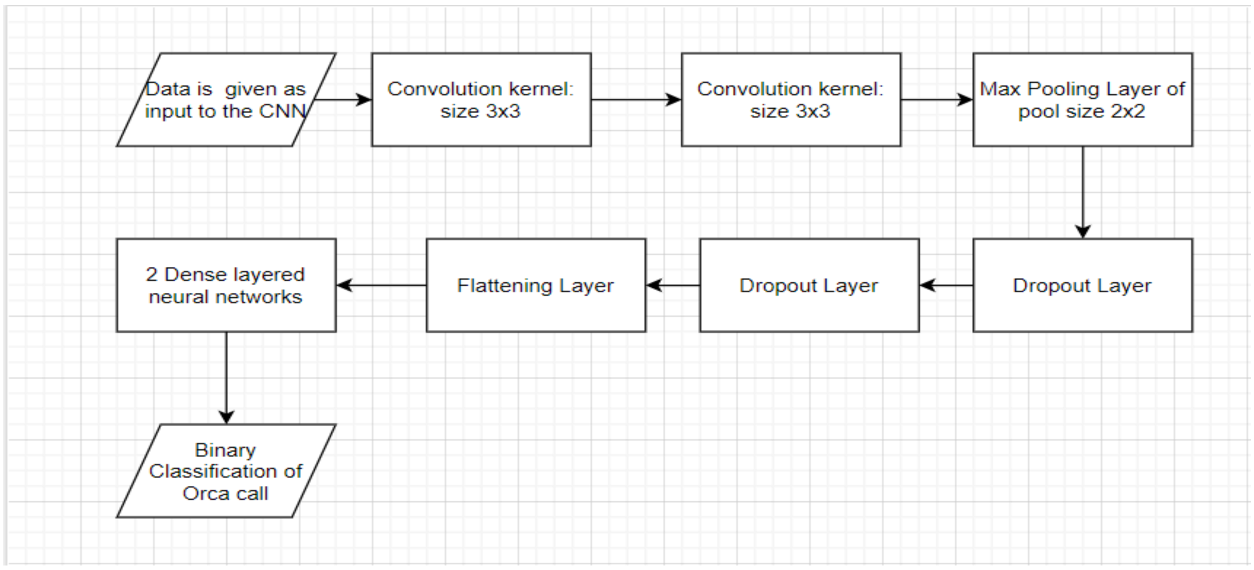
Stacking two smaller convolution layers is better than having a larger convolution layer. This is because even though the original throughput remains constant, it results in a lighter number of parameters, therefore a lesser number of weights are required, and the stacking of layers results in a deeper network. Thus, instead of a regular 5x5 kernel size, I will be stacking two 3x3 convolution layers. The output from the stacked layers is then taken as input for the next layer, which is the max-pooling layer.

The Max-pooling layer gives us some measure of translational invariance. More importantly, pooling is faster to compute than convolutions, thus we increase the learning rate of the model by adding a Max-pooling layer. A standard pool size of 2x2 is good for this. The output of the pooling layer is then taken as input for the dropout layer.

To prevent overfitting, we use a dropout layer. A dropout layer works by randomly setting the outgoing edges of hidden layer neurons to 0 at each update of the training phase. An alternative to this is using multiple neural networks with different model configurations together. However, this is not recommended as this method requires the additional computation expense of training and maintaining multiple models. Therefore, the use of a dropout layer is a better option.

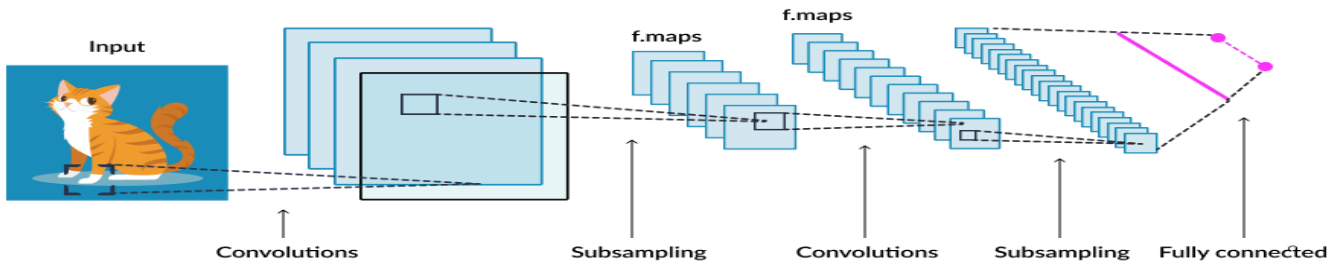
The flattening layer is the layer between the convolutional layers and the final dense/fully connected layer. It ‘flattens’ the output of the convolutional layers by converting the data into a single 1-dimensional long feature vector.

A fully connected layer is also known as a dense layer. In a dense layer each neuron receives input from all the neurons in the previous layer, making the neural network densely connected. This layer produces the final binary output after taking input from the long feature vector that was the output of the flattening layer.


Data is given as input to the CNN
Convolution kernel: size 3x3
Convolution kernel: size 3x3
Max Pooling Layer of pool size 2x2
Dropout Layer
Dropout Layer
Flattening Layer
2 Dense layered neural networks
Binary Classification of Orca call

A flowchart showing the implementation of the proposed CNN model.

D. The input we will be giving to the model will be a spectrogram of fixed length.



An image showing the working of a general CNN

Image reference:https://missinglink.ai/guides/keras/keras-conv2d-working-cnn-2d-convolutions-keras/

3. **Optimization of the Model:** After training the model on the GPU (the GPU I am using is Nvidia 960M) using the large modified dataset, we will start optimizing it. The community can replicate this by using the GPU in their personal computers or by using a cloud GPU.

I will try to avoid using gradient descent and its variants to optimize, due to the presence of a few drawbacks such as:

a) Difficulty in choosing learning rate. If the rate is too small, the convergence is very slow. Parallelly, if the rate is too large the convergence is hindered and the loss function starts fluctuating.
b) The same learning rate applies to all parameter updates.
c) It is possible to get stuck at the saddle points since they are usually surrounded by the same error, making the gradient close to zero in all dimensions.

However, I will also try to implement SGD (Stochastic Gradient Descent)+Nesterov Momentum as according to a research paper by UC Berkeley, SGD has better generalization than adaptive optimizers when the dataset is not that big.

Since I aim to make a light model in order to minimize processing costs, a fast-learning model without saddle point errors is preferred. Thus I am leaning towards implementing adaptive learning optimization algorithms.

I intend on primarily using Adaptive Moment Estimation, or Adam in short. It is an adaptive learning rate optimization algorithm that has been specifically designed for training deep neural networks by leveraging the power of adaptive learning rates to find individual learning rates for each parameter. Adam works well in practice and the speed of learning of the model is fast. This is because ADAM not only stores an exponentially decaying average of past *square* gradients, it also keeps an exponentially decaying average of past gradients M(t), much like Momentum.

$$For\ each\ Parameter\ w^j$$

$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$

$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : $Initial\ Learning\ rate$
ν_t : $Exponential\ Average\ of\ squares\ of\ gradients$
g_t : $Gradient\ at\ time\ t\ along\ \omega^j$

Image source: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

Adam is more efficient and is a better algorithm to optimize with compared to others:

1. It converges very fast, due to which the log function does not fluctuate.

2. Increases the learning speed of the model.

3. Does not have a vanishing learning rate.

Since I will have a decently sized dataset and quite a few parameters to work with, Adam is a better choice than SGD as generalization will not be a concern due to the size of the dataset. Adam is also good for optimizing networks that work with noisy inputs. This will help us train our network better with the noise-augmented Orca call signals in the synthesized dataset. The computational efficiency, low memory requirements and invariancy to diagonal rescale of the gradient makes Adam a good choice for the network so that it can be light and can produce an output as fast as possible, which is needed to classify Real time Orca calls in No Time .

Other algorithms that I will be implementing and comparing with Adam include SGD, AdaGrad and RMSprop.

4. **End Result:** We have a binary classifier that has been optimized with Adam to have a fast learning rate, due to which we can train and test on a personal computer. It can detect Orca call sounds in the presence of quite a bit of noise and works with good accuracy. I will be splitting the larger synthesized dataset in a 4:1 ratio for training and testing respectively to ensure that we have a large enough set for both training and testing.

5. **Deploying the Model:** I will be deploying the model as an API with Flask on a suitable cloud computing platform such as Google Compute Engine. Furthermore, I will try to decide which cloud computing platform to use after consulting with the community during the community bonding period. I will productionize the Flask API and get it ready for deployment using Docker.

   I will be splitting the larger synthesized dataset in a 80:20 ratio for training the model and testing it post deployment respectively.

   Flask Restful is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with my existing libraries. It requires minimal setup and thus is a good choice for deployment.

For the production server, I will be using NGINX, which is a free open-source high performance HTTP server. It is known for its high performance, stability, rich feature set, simple configuration and low resource consumption.

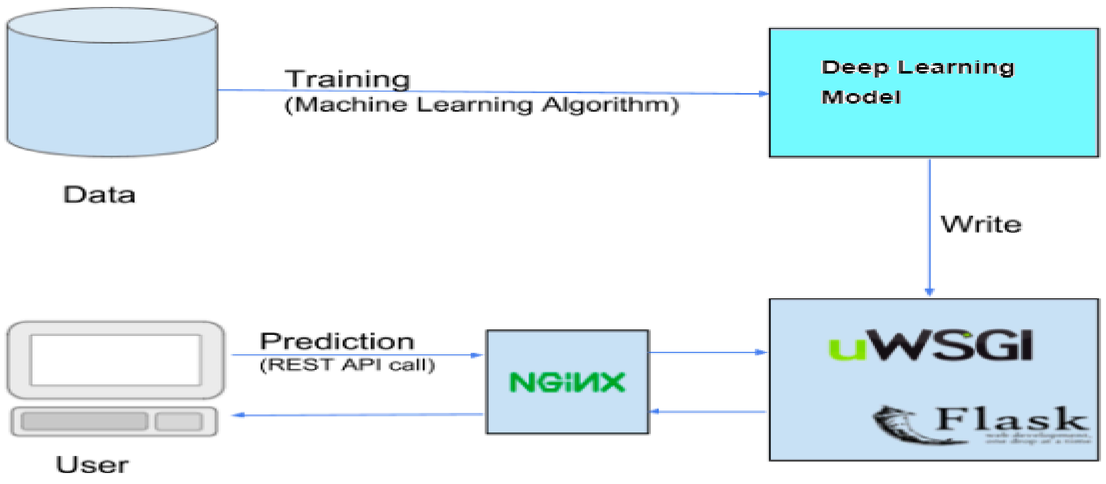


Image taken and modified from: https://cloudxlab.com/blog/deploying-machine-learning-model-in-production/

Deliverables:

Phase 1:

- Development and training of a binary Orca whale call classifier that can work with significant noise in the background.
- Comprehensive documentation and blog posts on the Orcasound blog for the same along with the preprocessing phase for the dataset.

Phase 2:

- Deployment of the model to the cloud using Flask Restful API.
- Subsequent testing of the aforementioned model.
- Documentation and blog post on the Orcasound Blog for the same.

Final submission:

- Final model fully deployed and working.
- Testing on real time data streams coming in from the hydrophones.
- Blog posts on the Orcasound Blog about my progress, the model and my experience with the organization.

Timeline:

Community Bonding	Week 1- 5th to 11th May,	Start interacting with the Orcasound community, start reading the documentations for the aforementioned packages and install all necessary softwares.
	Week 2- 11th to 17th May	Continue to do the same and also go through all necessary packages and patchnotes once to avoid issues in the future.
	Week 3- 18th to 24th May	Discuss and map my plan of action and write blog posts on the Orcasound blog about the techniques and architecture used, decide which cloud computing service to use.
	Week 4- 25th to 31st May	Confirm the roadmap and the plan of action with the mentors and get ready to start coding, finish any leftover installations.
Phase 1	Week 5- 1st June to 7th June	Preprocess the audio data, convert it into spectrograms and start building the basic model in tandem. Try to see

		if other architectures might be better suited to the task at hand.
	Week 6- 9th to 15th June	Develop the model and test/train it on the preprocessed audio data converted to spectrograms.
	Week 7 - 17th June to 22nd June	Solidify the model and start working on uploading the model to a suitable cloud computing platform using Flask API.
	Week 8 - 23rd June to 29th June +30th June	Keep working on uploading the model to the cloud computing platform.
Phase 1 Evaluations	**1rst- 3rd July**	
Phase 2	Week 8- 4th July to 10th July	Documentation, buffer week.
	Week 9- 11th July to 17th July	Finish uploading the model to the chosen cloud computing platform and use cloud GPU's to make it run.
	Week 10- 8th July to 14th July	Work on implementing feedback from evaluations and solidifying the model more.
	Week 11- 15th July to 21st July	Start optimizing the model by using ADAM, try minimizing cost.
	Week 12- 22nd July to 26th July	Keep working on optimizing the model, make changes if necessary.
Phase 2 Evaluations	27th to 31st July	
	Week 13- 1st to 4th August	Start writing the blog, finishing optimizing the model and start working on

		deploying the model online using Docker
	Week 14- 4th August to 10th August	Keep working on online deployment so that users can detect Orca calls, write blog posts
	Week 15- 14th August to 20th August	Finish deployment using Docker. Work on troubleshooting any bugs or errors that may occur.
Final Submission	21st to 31st August	Documentation and Mentor Suggestions, Submission
Final Evaluation	31st August to September 7th	

I can work 4 hours on weekdays and 8 hours on weekends. These hours are flexible and can be changed according to the availability of the mentors, time constraints of the project and any unforeseen developments.

I will be on my summer break till June 1st and will be following the weekend work schedule mentioned above for all the days of the week.

From 1st to 28th June, university classes will resume, therefore I will be following the timings mentioned above for weekdays and weekends.

Due to COVID-19, I will be having my examinations at this time and will be unavailable to work from June 29 to July 13. I will still keep working on the project but I will not be able to follow a strict work schedule for this duration. I will compensate for the time lost during holidays and weekends.

I will be having end semester vacations from 13th to 29th July and will be able to work 8 hours a day.

From 29th July, the new semester will begin and I will be working 5 hours on weekdays and 8 hours on weekends, as mentioned above.

References:

1. Coursera- Deep Learning and Neural Networks by deeplearning.ai
2. https://github.com/sigsep/open-unmix-pytorch
3. https://github.com/axiom-data-science/OrcaCNN
4. https://towardsdatascience.com/deploying-a-custom-ml-prediction-service-on-google-cloud-ae3be7e6d38f
5. https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f
6. https://medium.com/x8-the-ai-community/audio-classification-using-cnn-coding-example-f9cbd272269e
7. https://towardsdatascience.com/deploying-machine-learning-models-with-docker-5d22a4dacb5
8. https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/
9. https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel
10. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
11. https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0
12. https://www.kaggle.com/michael422/spectrogram-convolution
13. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
14. https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab
15. https://missinglink.ai/guides/keras/using-keras-flatten-operation-cnn-models-code-examples/

16. The Orca call sound used to demonstrate why the Fourier Transform doesn't work was taken from: https://www.youtube.com/watch?v=JTEvEE6fq6k
17. https://arxiv.org/abs/1705.08292