CLICK LEARN

AUTOMATED DOLPHIN CLICK DETECTION IN MARINE ENVIRONMENT WITH DEEP LEARNING

ABSTRACT

We present an audio classifier capable of recognizing dolphin sounds. The model can be deployed on buoys for the real time monitoring of dolphins to prevent bycatching by commercial fishing boats.

Arnaud Felin, Benjamin Bernard, Benoit Mialet, Deniz Pekin, Manuel Gawert

Machine Learning with Python Labs – DSTI

Project supervised by Assan Sanogo

1- Project overview

Background:

Dolphins are highly threatened in French waters due to extensive fishing activity. Up to 10 different species of dolphins could be extinct within the next 10 years which pushes the European Union to pressure fisheries for action [1]. Particularly, since January 1st 2021 in France, the approach within 100 meters of cetaceans (including dolphins) is prohibited in marine protected areas.

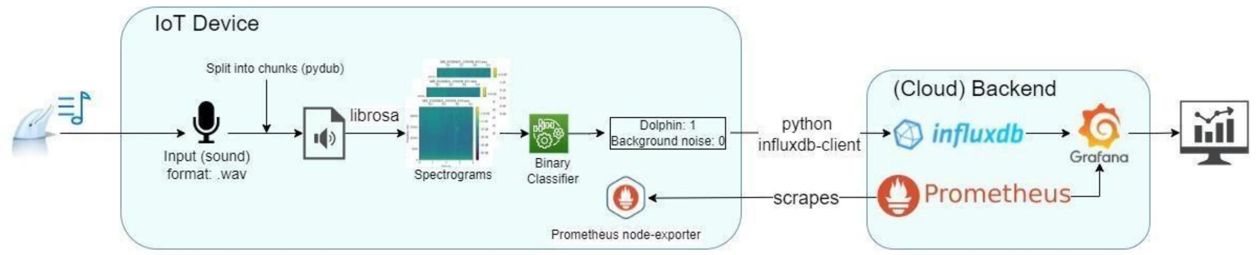
This brings the need for automated solutions for real-time detection of the presence of dolphins and divert the fishing boats away from these areas.

ClickLearn is a R&D school project in partnership with DSTI and two companies located in La Rochelle, France:

Systel is a company developing solutions for the management of major territorial and national high-risk situations. Their products meet both the operational and technical requirements of the civil security forces with an emphasis placed on the operational simplicity.

Ubidreams is a digital service company for the creation of customized mobile and software solutions. Experts in the creation of web and mobile apps also create connected devices and provide advice and technical expertise in intersecting fields of business software and application development.

Project diagram:



2- Problem statement:

The essence of the problem we are addressing is in two levels:

(i) How can we build a system that will "listen" to the underwater sounds and recognize dolphin sounds?

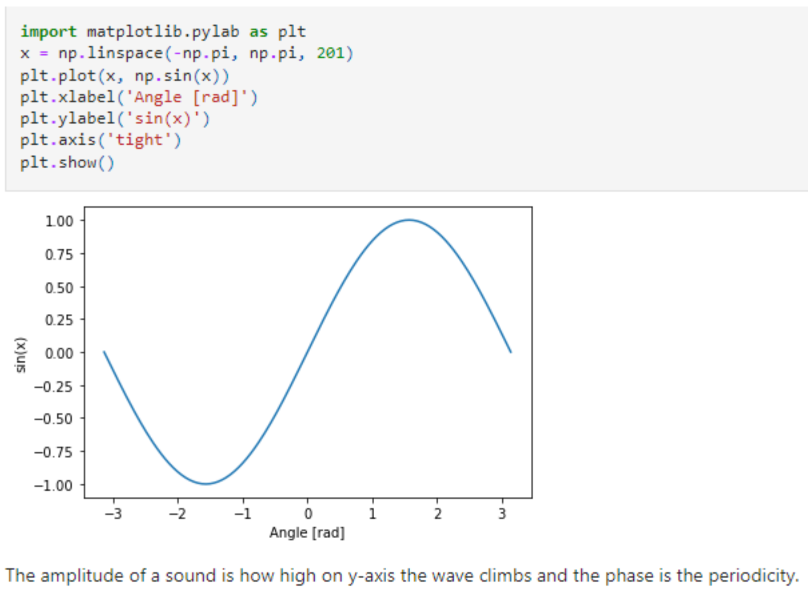
(ii) How can we deploy this system "in the middle of the ocean"?

3- State of the art:

3.1- Sound as data:

Sound is a signal that is produced by variations in air pressure. The intensity of the pressure variations over time resembles a sin(x) function (Figure 1). The time taken for the signal to complete one full wave is the period. The number of waves made by the signal in one second is called the frequency, which can also be defined as the inverse of a period, therefore the unit is 1/seconds (defined as Hertz).

The signals that we encounter, however, are rarely pure sine waves. They are a blending of different frequencies. Figure 2 shows real cello sound recorded with a microphone. You can imagine that this waveform is made up of pure sine waves added together (the fundamental plus all the harmonics). Mathematically, any periodic (repetitious) waveform can be generated by adding an infinite number of pure sine waves, which is called a Fourier series.



```python
import matplotlib.pylab as plt
x = np.linspace(-np.pi, np.pi, 201)
plt.plot(x, np.sin(x))
plt.xlabel('Angle [rad]')
plt.ylabel('sin(x)')
plt.axis('tight')
plt.show()
```

The amplitude of a sound is how high on y-axis the wave climbs and the phase is the periodicity.

Figure 1: An example of a simple repeating signal is the sin(x) function. Image source: Deniz's github (https://github.com/TacticalNuclearRaccoon)

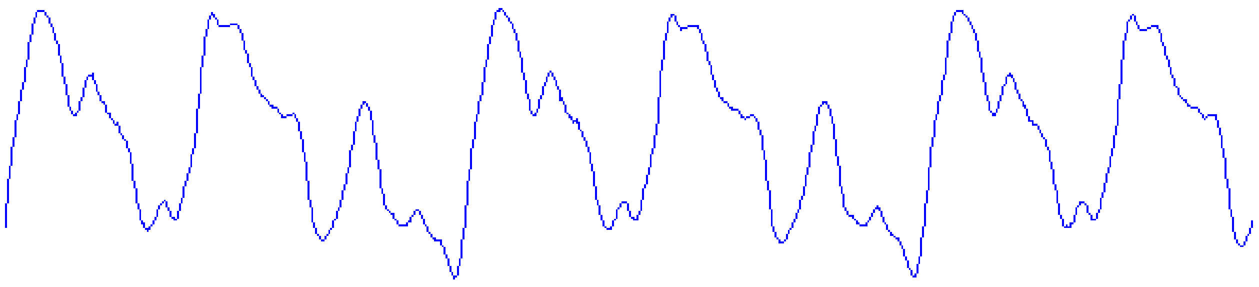


Figure 2: The waveform obtained by playing G# on the D string of a cello (disclaimer: it is a difficult note to play). The graph was obtained from the commercial Vobarian software (https://www.vobarian.com/index.html).

Digital representation of sound and the concept of sampling rate:

Much like we transform images into numpy arrays to be handles by models, to digitize a sound wave, we need to sample it. This is done by measuring the amplitude of the sound at fixed intervals of time. Each such measurement is called a sample, and the sample rate is the number of samples per second. For instance, a common sampling rate is for music is about 44,100 samples per second. That means that a 10-second music clip would have 441,000 samples (Figure 3).

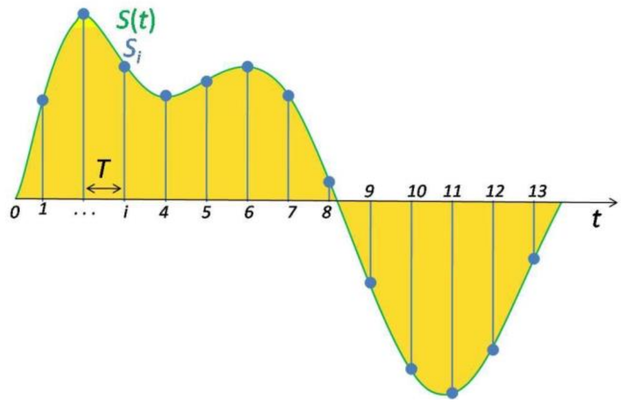


Figure 3: The concept of sampling rate where the signal is divided (sampled) into segments (Image source: https://commons.wikimedia.org/wiki/File:Signal_Sampling.png)

In the modern deep learning strategies, the audio signal is transformed into an image so that in can easily be handled by a standard CNN architecture.

Spectrum and Spectrogram:

The Spectrum is the set of frequencies that are combined to produce a signal. Figure 4 shows the spectrum of a cello tone.

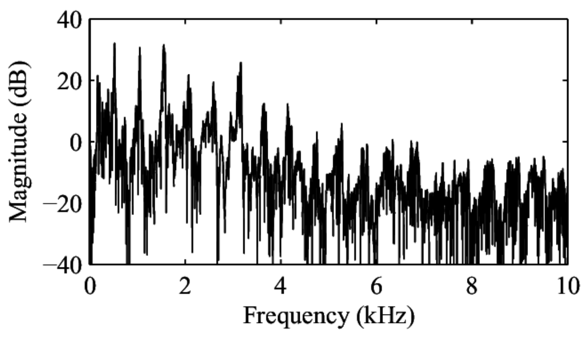


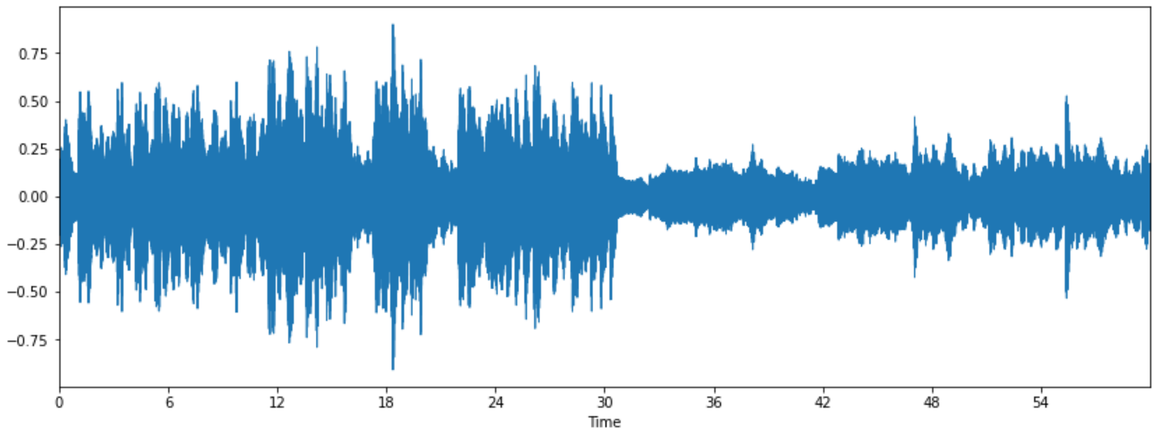
Figure 4: Spectrum of the C5 note on a cello (image source: DOI 10.1109/NORSIG.2004.250163)

The Spectrum plots all the frequencies that are present in the signal along with the strength or amplitude of each frequency.

A Spectrogram of a signal plots its Spectrum over time, with Time on the x-axis and Frequency on the y-axis. A color code is used to indicate the Amplitude or strength of each frequency (which can be considered as the 3rd dimension of the plot).

In the example below (Figure 5), we see the Amplitude vs Time of a sound signal from a recording. This kind of can provide a sense of how loud or quiet a clip is at any point in time, but it provides very little information about which frequencies are present. This information is completed by the spectrogram.

ut[13]:



n [14]:

```
#Converting to Spectrogram
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

ut[14]:

```
<matplotlib.colorbar.Colorbar at 0x7f523b0c4640>
```

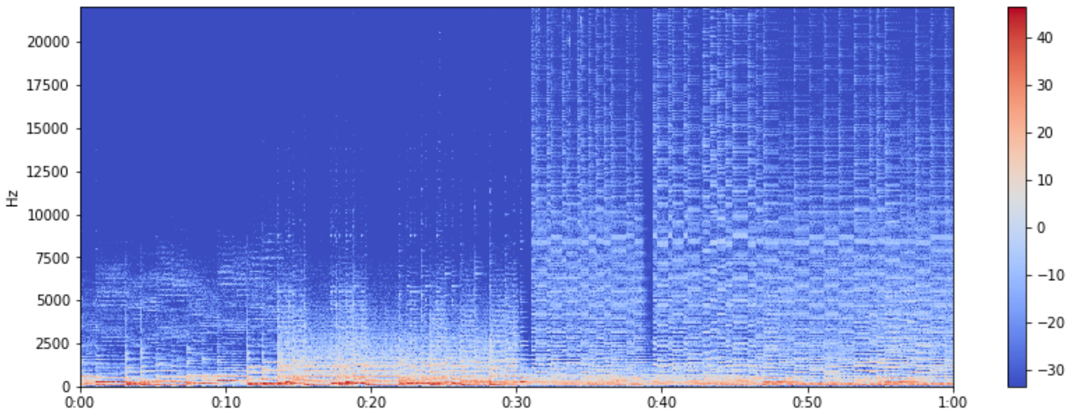


Figure 5: The information content of a wave plot versus a spectrogram. The middle panel shows the python code used to obtain the spectrogram from the audio wave data.

Deep learning models for audio classification:

Most deep learning models we have encountered use a similar workflow:

1. Start with raw audio wave file
2. Convert the audio to spectrogram
3. Optional data augmentation (frame shift of frequency shift – usually semitones)
4. Use a standard CNN to extract features maps
5. Pass to output through a classifier

3.2- Deep Learning for IoT

In a general sense (for audio classification applications but also beyond) implementing deep neural networks to be deployed on IoT devices brings up several considerations, such as:

- What deep neural network structures can effectively process and fuse input data for diverse IoT applications?
- How can resource consumption of deep learning models be reduced such that they can be efficiently deployed on resource constrained IoT devices?
- How can confidence measurements be computed correctly in deep learning predictions for IoT applications?
- How to minimize the information to send from the IoT to a server?
- Finally, how can the need for labeled data be minimized in learning?

In short, we need to be capable of collecting data from IoT devices, as well as be energy-efficient, accurate, and able to function with minimal data labels.

Project workflow

In this project we have worked with 2 types of data:

- Sound recordings of North Atlantic right whales (NARW)
- Sound recordings of dolphins.

Even though our target was to classify dolphin sounds, the sound data from dolphins was not readily available at the beginning of the project. These recordings come from a scientific research project, and we had to wait for the recording program to be completed.

As we decided to build a complete workflow, from sound recording to detection visualization from a development kit, we started to work by training a model with NARW Data as a first step. As a second step, we trained a model with dolphin Data (as soon as they were available) and then implemented it in the workflow.

The concept of "Curse of dimensionality" in machine learning which emanates when the number of input features is too high, cases the models to be over complicated. Since raw audio data contains a large number of unnamed features, it is wiser to use a deep neural network with the capability of automatic feature learning. Thus, we have tested 2 frameworks:

Ketos: Ketos is an open-source Python package for acoustic data analysis with neural networks, which has been developed by MERIDIAN. We have used this framework because it was chosen by the project owner company (Ubisea) and it is a framework that has been adapted for marine life detection, and therefore is well known by the researchers. To learn more about Ketos, check out the GitLab repository at *gitlab.meridian.cs.dal.ca/public_projects/ketos* or explore the documentation pages at *docs.meridian.cs.dal.ca/ketos*.

A regular CNN network: This part of the project will be completed by our colleague Vincent Swiderski on the following cohort.

This report will focus on the Ketos framework.

4.1- Audio transformation and preprocessing

The information on sound processing can be found in the "**Sound processing**" repository and in our Confluence documentation: https://ubidreams.atlassian.net/wiki/spaces/UBISEA/overview?homepageId=65900.

4.2- Model Training

We have trained a NARW whale classifier (named Mobydick) and a Dolphin sound classifier (named Flipper) using the ketos framework.

Both models have a ResNet architecture (Figure 6). The whale classifier was trained using audio recordings from the Gulf of Saint Lawrence. The training data set consisted of approximately 6,000 spectrograms, each of 3 seconds duration and covering the frequency range 0-500 Hz, with about half of the spectrograms containing an upcall. These calls are characterized by an upsweep frequency from about 50 Hz to 350 Hz and a duration of about 1 second. The data was provided by MERIDIAN.

The Notebooks describing the workflow can be found in the "Models" repository.

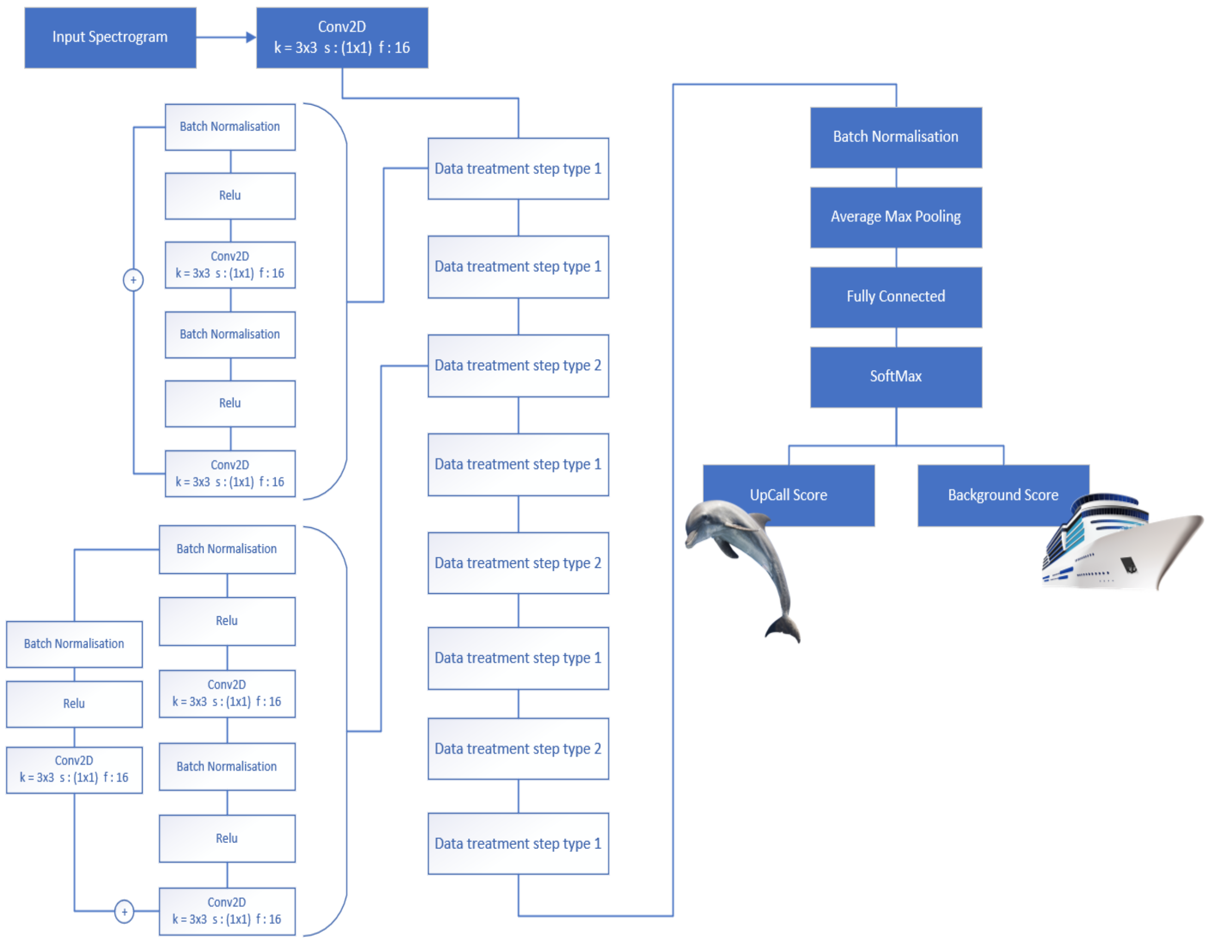


Figure 6: the ketos model architecture based on ResNet.

The NARW classifier was trained on open source acoustic data collected between 2015 and 2019 in the southern Gulf of St. Lawrence. Repository: Acoustic recordings of North Atlantic right whale upcalls in the Gulf of St. Lawrence

The dolphin sound classifier was trained on acoustic data provided by the Marine Conservation Research International (MCR International) in partnership with Observatoire Pelagis (Université de La Rochelle) for research purposes on marine mammal sound detection and classification. Sound samples of 15 min (192kHz subsampling) were collected during the ASI scientific survey conducted in the Mediterranean Sea in 2018. Labels were provided by Observatoire Pelagis marine mammal specialists. The details on database creation can be found in our Confluence page in the section "Software and model".

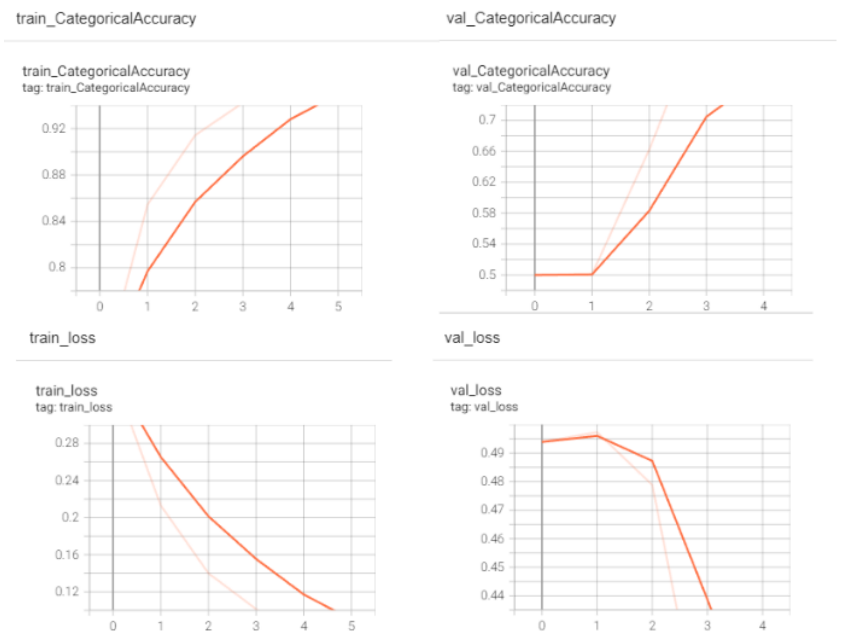


Figure 7: Tensorboard output for the training of the whale audio classifier.

Early stopping was used during the training of both models to avoid overfitting. The whale classifier showed comparable accuracy and performance (loss) on both training and test datasets (Figure 7).

The accuracy and performance metrics of the dolphin classifier are shown on Figure 8.

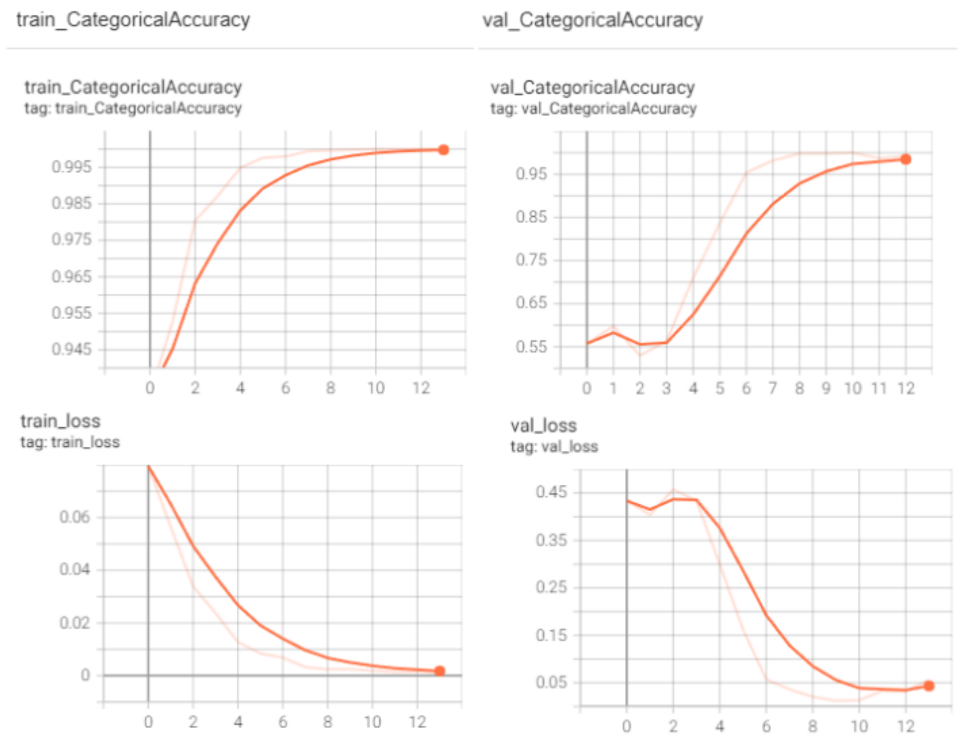


Figure 8: Tensorboard output from the training of the dolphin audio classifier.

4.3- Model evaluation and validation

Already annotated test data was available for the whales. We have prepared a graphical user interface with Streamlit in order to automatize the workflow. A demo recording can be found in our github page. The binary classifier trained on whale upcalls (that we have named Mobydick) performed at 0.68 accuracy on a set of 30 second .wav whales.

For the dolphins, such data was not available therefore we have produced a 2min long test .wav file by concatenating 1min of background noise with 1 min of a recording which was heavy with dolphin click trains.

The dolphin binary classifier (that we have named Flipper) performed well on this file (see Figure 9)

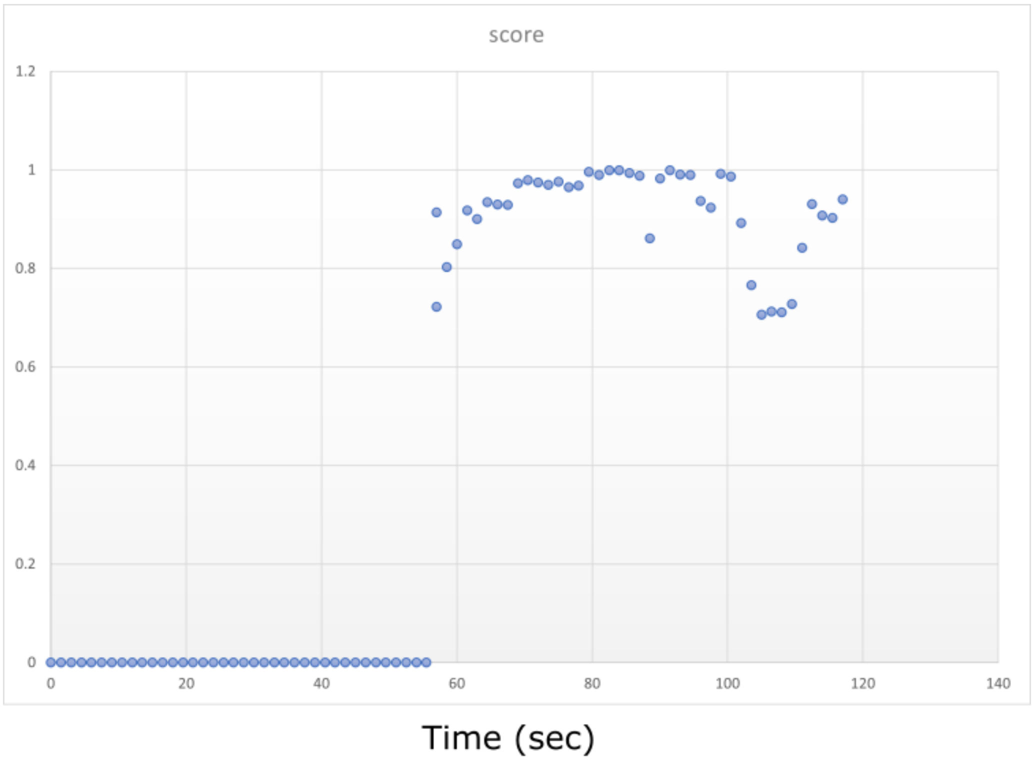


Figure 8: We have tested the performance of the dolphin sound classifier on a 2min file where the 1st minute consists on background noise only and the second half of the file has click trains. The y axis shows the score given by the model to a specific sound being a dolphin click. The model was able to detect the click train.

4- Buoy monitoring

The end goal of our project is to the deploy the model on buoys, in the ocean. As these Buoys will be difficult to access physically, it is crucial that they are monitored properly, and failures can be detected proactively. That's why all the buoys will be equipped with the Prometheus Node exporter. The Exporter will expose metrics that are then scraped by Prometheus installed on the “backend” environment. For Visualization, a dashboard was created in Grafana (Figure 9).

Monitoring with Influx DB and Grafana:

After the Buoys detect Dolphins, the result of the detections will be sent to an Influx Timeseries Database via the InfluxDB python client. This Database will serve as a datasource for our Grafana Dashboard. Grafana is also used as part of the Buoy Monitoring. Grafana and InfluxDB will both be deployed as docker containers. Ideally on a cloud backend for easy maintenance and availability. However, for the ClickLearn University Project, there was no Cloud budget and access to a University Server was not granted, so that both containers were deployed on one of the students Linux Homeserver.

Please refer to our Confluence page for the details of the installations of Node exporter and Prometheus.

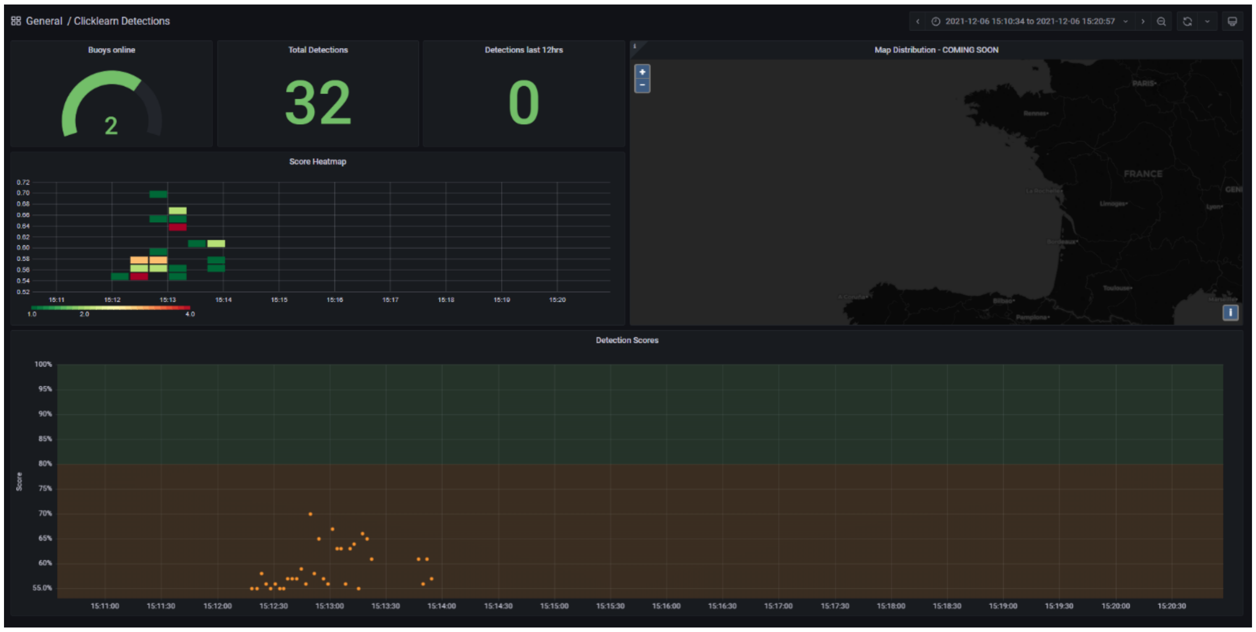

General / Clicklearn Detections
2021-12-06 15:10:34 to 2021-12-06 15:20:57
Buoys online
2
Total Detections
32
Detections last 12hrs
0
Map Distribution - COMING SOON
Score Heatmap
Detection Scores
Score

Figure 9: The Dashboard and fields will be auto populated from the data scraped from the different Prometheus targets, which are in fact the node exporters on the buoys.

The dashboard is also stored in our Github:

“https://github.com/DSTI-ClickLearn/ClickLearn_DSTI_Project/tree/main/Hardware”

5- End to end workflow

To test our project’s end to end workflow we have recorded a 30 minutes long audio file and processed. A demo can be found on our Confluence page as well as in our github.