Quaternion Neural Networks for 3D Sound Source Localization in Reverberant Environments: Implementation using First Order Ambisonics

Roberto Aureli, ID 1757131
Riccardo Caprari, ID 1743168
Gianmarco Fioretti, ID 1762135

Neural Networks, Spring 2020

Contents

1 Introduction

This project addresses the **3D Sound Event Localization and Detection Task** in reverberant environments with a quaternion neural network (deep neural network with quaternion input features extracted from the acoustic intensity vector).

As mentioned above, the proposed model performs both sound localization and sound event detection and subsequent classification. In particular it follows the architecture described in paper [1] which allows to estimate the three-dimensional direction of arrivals (DOA), in addition, it has been modified in order to be capable of detecting sound events and estimating the corresponding sources (eleven different classes provided by the development dataset, check for Table 1).

Many recent works have proven that deep quaternion neural networks are able to improve localization performances dramatically, especially in reverberant and noisy environments, thanks to spatial harmonic decomposition which permits to exploit the intrinsic correlation of the ambisonics signal components. One of the main aspect to be considered is the input features to be passed to the network.

Our main work consisted in adapting the provided code to the new dataset, fine-tuning the model's hyper-parameters, applying the three metrics defined in a precedent paper [2] (**SED, DOA, SELD**), implementing the direction of arrival estimation in cartesian coordinates, and computing the confidence intervals on the model's final errors as defined in Perotin et al. paper [3] in order to add also a statistical evaluation on final results.

2 Dataset

The network is trained with the **TAU Spatial Sound Events 2019** dataset which provides four-channel directional microphone recordings of stationary point sources.

It is a balanced dataset, it indeed consists of eleven classes, each with twenty examples that were randomly split into five sets with an equal number of examples for each class, in addition, it was divided into four cross-validation split.

The maximum number of simultaneously overlapping sources are two.
Moreover, in order to improve the performance over new sound events, and to make a more realistic scenario, natural ambient noise collected in the recording locations was added to the synthesized recordings in the dataset such that the average SNR of the sound events was 30 dB.

Table 1: TAU Spatial Sound Events Classes

Sound class	Index
knock	0
drawer	1
clearthroat	2
phone	3
keysDrop	4
speech	5
keyboard	6
pageturn	7
cough	8
doorslam	9
laughter	10

3 Architecture

The network proposed in the paper [1] involves a series of convolutional layers in the quaternion domains, they are composed of several filter kernels which allow learning inter-channel features, with subsequent activation functions and max-pooling functions. The output of this series are properly reshaped and fed to bidirectional quaternion recurrent layers. This first part of the architecture is depicted in Figure 1

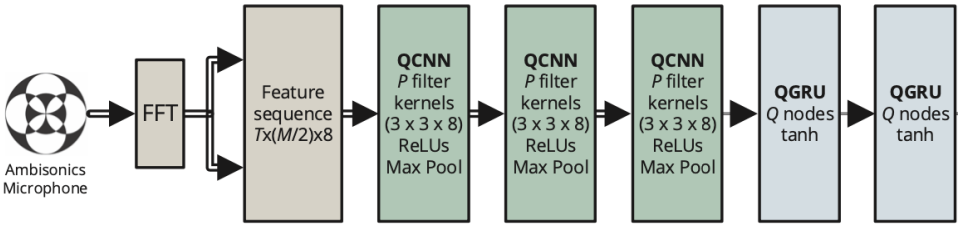

Figure 1: First part of the Network Architecture

As already pointed out on Chapter 1 the network has been modified accordingly, the final part of architecture therefore becomes as in Figure 2, so that network is able to perform a multi-classification for the SED task (one for each class: 1 to indicate detection, 0 otherwise) and multi-regressions for DOA task (which has in input azimuth angle, elevation angle, distance and returns cartesian coordinates).

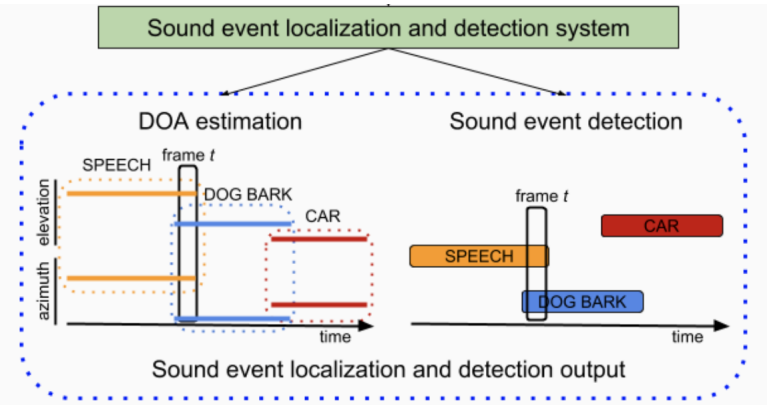

Figure 2: Last part of the Network Architecture

4 Our work

The objective of our work is to build a working SELDnet based network that works with First Order Ambisonics data sets. In particular, we are going to extend SELDnet, adding capabilities to both support pre-existing data sets (ansim, resim, etc.) and the FOA one in a smart, modular, performing way. Other metrics have been added like the SELD score, mainly used in the paper [2] outcomes evaluation, and a tiny library for a graphical representation of the results. Now we'll proceed with a more technical analysis of the work, listing all the features added.

4.1 Project implementation

We decided to mantain a parameters manager file to configure the SELDnet, which contains for example it's input dataset with specific values, training, evaluation and test process combinations and many other settings that can be found in *parameter.py*. We can divide our project in 3 main classical phases: extraction, training and test. We will cover these three parts after highlighting the main changes on input parameters:

- *parameter.py*: extension of *quick_test* parameters supporting now batch and steps size. Support for data sets and extraction output directiories paths. New training management handling different splits also for validation and test evaluation.

4.1.1 Extraction

Before starting with the training, we have to extract features from the new FOA data set. This delicate process makes the difference between a successful training in terms of evaluation metrics. For this reason we decided to introduce a new *FeatureClass*, which will differ from the old one in some parameters and procedures. In fact, as we experienced, introducing a new similar class results in a way more readable, modifiable and modularizable approach. We noticed, in fact, that different data sets can be similar from a structure point of view and totally different with respect to their parameters,

like record settings. For this reason a new full-dedicated extractor results in a wise and faster-implementation choice.

As we can observe in scheme 3, *batch_feature_extraction.py* has the task of managing the extraction process, differing function calls with respect to data set type. The input is divided into two folders, *foa_dev* and *metadata_dev*, which contain audio records stored with a *.wav* format and their corresponding labels.

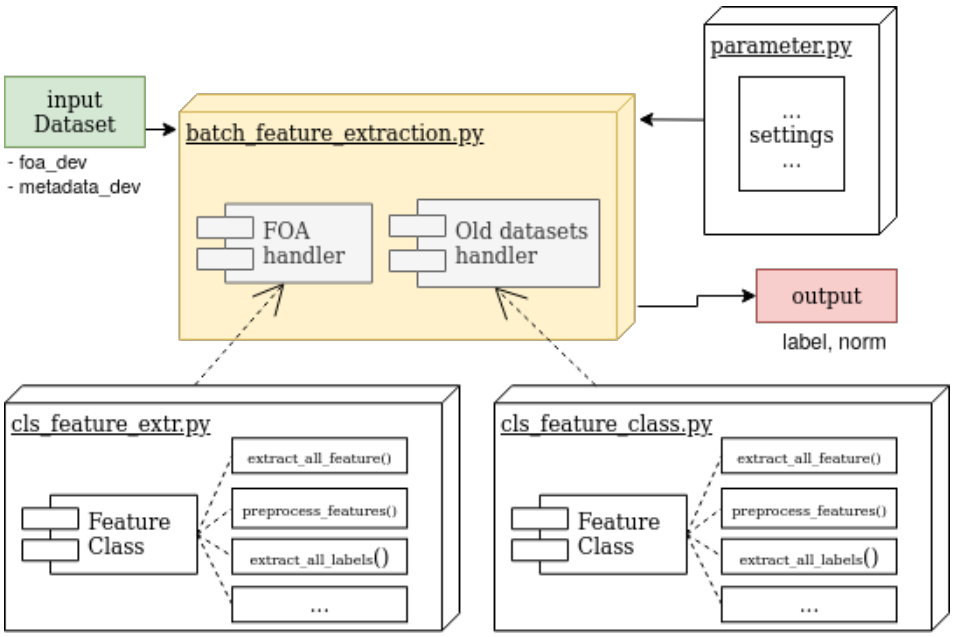

Figure 3: Extraction process.

Let's now zoom in the new *cls_feature_extr.py* extractor and highlight the main differences with the other one. These are the most significant adjustments:

- *FeatureClass()*: changes on hop length, which are the number of audio samples between adjacent STFT columns; win length, which windows each audio frame by a certain number; nfft has been changed and chosen to fit win length taking the next greater power of 2; new data set output folder management.
- *spectrogram()*: use of *librosa* library for the short-time Fourier trasform

taking a cue from [4]. Then we cycled over the number of channels to get the stft and transpose it to compose the final spectra. After, we loaded each audio file, extracted its spectogram and subsequently saved in a *.npy* format.

- *preprocess_features()*: use of *StandardScaler* which is in charge of standardizing the features. Adding partial fit of the scaler in order to compute mean and std of the precedent computed feature files.
- *extract_all_labels()*: weakness and mode parameters have been remove; slight edit on get labels process.

After the extraction has been done we will deal with three new different folders inside */feat_label_tmp*, which contain *.npy* files. Here is a list of them:

- *foa_dev_label*: storing labels.
- *foa_dev_norm*: which stores normalized features.
- *foa_dev*: which stores unnormalized features.

4.1.2 Training

After the process of features extraction is completed we can proceed with the training. Unlike for the extraction, in this phase we introduced some changes to both manage foa data sets and old ones instead of adding a completely new *seld.py*. In particular, we highlight differences on:

- *cls_data_generator.py*: new split handler system, in *_get_label_filenames_sizes()* we added support for train and validation/test splits which has as objective to create a correct file list. All the files are now taken according to *parameter.py* settings, which specify the number of the splits. Adding check control for circular buffers in generator's *generate()* base function in case pop operations exceed buffer size.
- *seld.py*: new batch and step control for quick tests; changing *unique_name* for model weights data saving; new smart resumption of training if

weights are present in */model* folder; new important **bug-solving** procedure to let *predict_generator()* work correctly, which consists of creating the model, save it and reload it with old weights before a prediction occurs; new confidence interval calculator for DOA and SED errors; adding SELD to the scores and saving all of them in a csv file.

- *evaluation_metrics.py*: adding *compute_confidence()* function to compute 95% confidence interval given an array of data. Implementation based on [5].
- *simple_plotter.py*: new file which can be used for csv saving, metrics 2D plotting and data prediction 3D plotting.

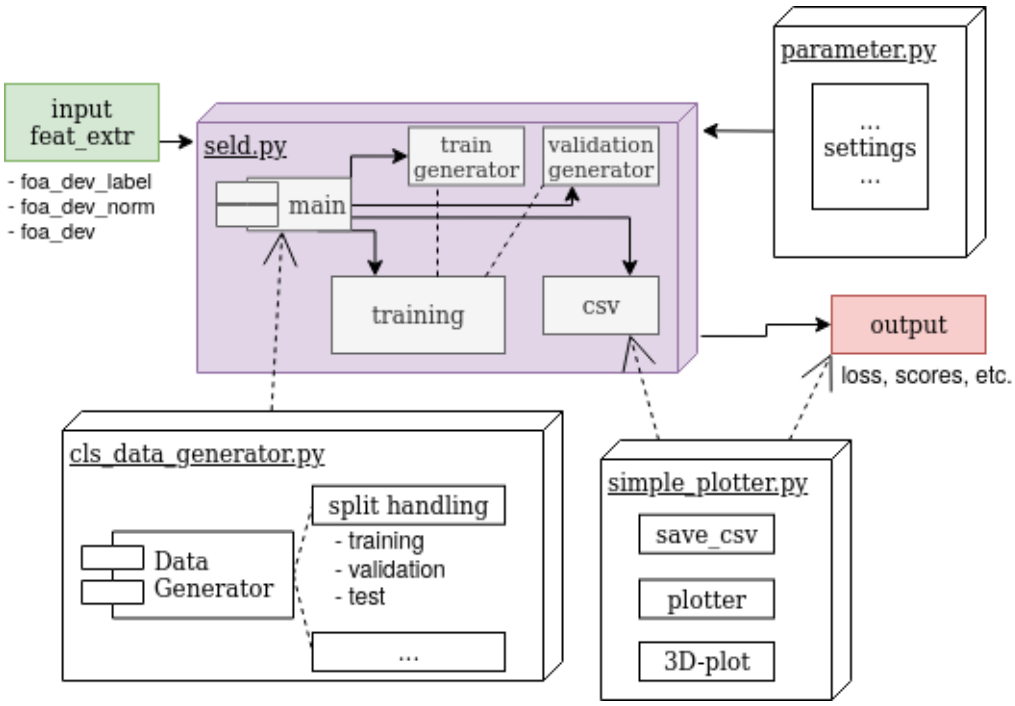

Figure 4: Training process.

In figure 4 it's possible to have a general view of *seld.py* in action. During the training we fit the model using a data train set and then, for each epoch, we use another smaller set as validation to compute all the necessary metrics. These metrics are then saved in a csv file in a consecutive way, meaning that each line corresponds to an epoch. Then, apart from *seld.py*, we generate all the plots which will be discussed in Chapter 5.

4.1.3 Test

Finally, we end up with test phase. This part wasn't really handled from the original code so we decided to add it to have a full final evaluation of a different set. In fact, by adding a new split in *parameter.py*, we are able to load weights of a pre-trained model and then make a full test using it. This part, like for the training one, includes all the evaluation metrics, confidence intervals and plots making.

In particular, the new included file for testing is *seld_test.py*. This program involves the creation of a new model whose weights are loaded from a trained one, two data generators (train + test) and a single prediction using *Keras* library.

4.1.4 Summary

Recapping, this is an overview of the introduced and changed files:

New:

+	batch_feature_extr.py
+	simple_plotter.py
+	seld_test.py

Changed:

~	parameter.py
~	batch_feature_extraction.py
~	cls_data_generator.py
~	seld.py
~	evaluation_metrics.py

Removed:

– – –

5 Results

The First Order Ambisonics data set we are going to use is divided into four different splits. We decided to use splits 3 and 4 for the training process, split 2 for the validation and split 1 for a final test. In order to obtain substantial results we trained the model for 100 epochs with a *batch_size* equal to 16 using *Tesla K80* GPU from Google Colab.

Initially, we were obtaing results with a really low accuracy even after plenty of epochs. This was caused by a Keras bug related to the use of *fit_generator()* and *predict_generator()* functions. However, we managed to solve this as explained in training part (4.1.2).

We present, through different plots, the results obtained using this specific combination of splits and the new implementation of the project. The first analysis we are going to conduct is the loss function. In Fig. 5 it's possible to see how training and validation loss decrease almost monotonically. The gap between the two functions is really small in the first 30/40 epochs and tends to grow in the second part. However, we can say that overfitting is relatively small and doesn't influence results in a particular way.

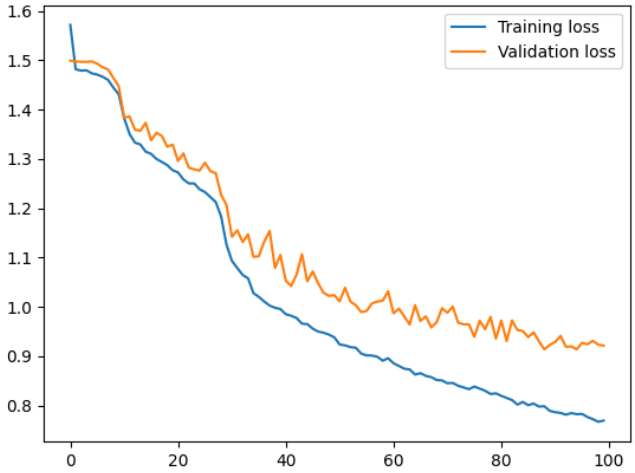

Figure 5: Loss function over 100 epochs.

A second plot we decided to produce and comment out is the one showing the main scores over the epochs. Specifically, these scores are DOA (Direction Of Arrival), SED (Sound Event Detection) and SELD (Sound Event Localization Detection). We compute them as:

$$S_{SED} = \frac{ER+(1-F)}{2} \quad S_{DOA} = \frac{\frac{DOA_{err}}{180}+(1-K)}{2} \quad S_{SELD} = \frac{S_{SED}+S_{DOA}}{2}$$

in which ER is the SED error rate and F is the f1-score; DOA_{err} is the DOA estimation error and K is the recall which takes into account true positives.

Observing Fig. 6 and Fig. 7 it's possible to see the behavior of the scores with the addition of F and ER SED's terms over 100 epochs. In the almost first 10 epochs the error rate ER is fixed to 1 which means that the model is predicting wrong with a percentage of 100%. On the other hand, f1-score is obviously fixed to zero because of the completely wrong predictions. These two terms results in an overall SED score fixed to one. In fact, the smaller these scores results the better they are. We can notice that as soon ER and F moves respectively from 1 and 0, SED scores starts to decrease rapidly.

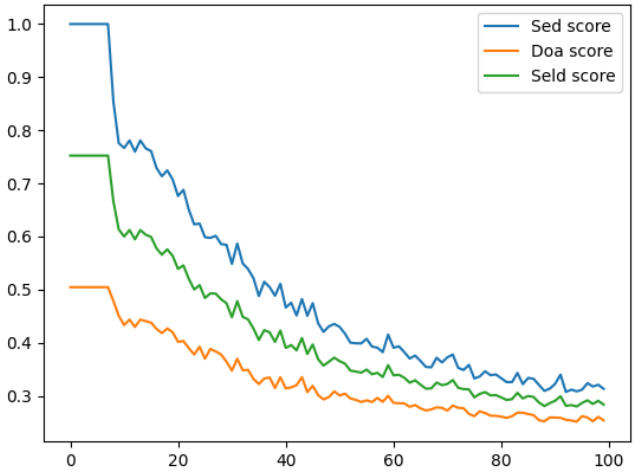

Figure 6: Sed, Doa and Seld scores over 100 epochs.

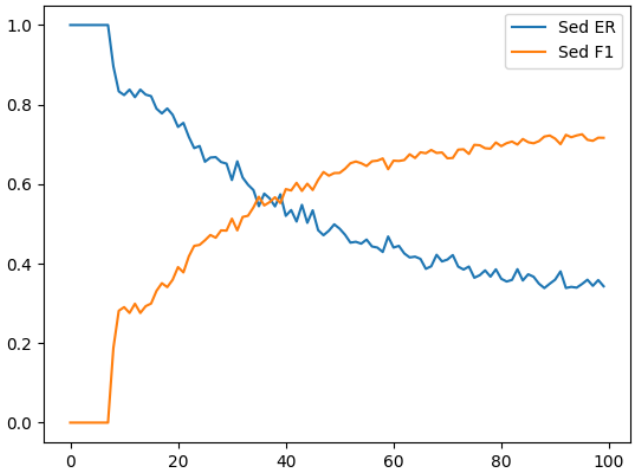

Figure 7: Sed error rate and f1-score over 100 epochs.

This overview is the same for DOA score which starts with a value of 0.5 and starts to decrease in a same fashion. Due to that SELD will behave in the same way, being an average of the precedent two scores.

We finally ended up with these training scores:

$$S_{DOA} = 0.31 \quad S_{SED} = 0.25 \quad S_{SELD} = 0.28$$

After a full training we decided then to test the remaining split (1) using the new *seld_test.py* program. Here we report outcomes evaluated with several evaluation metrics:

$$precision = 0.80 \quad recall = 0.68$$
$$S_{DOA} = 0.24 \quad S_{SED} = 0.29 \quad S_{SELD} = 0.26$$

Error confidence intervals (95%):
SED: $[0.014039, 0.014361]$, median 0.014179, displacement ± 0.000181
DOA: $[0.000000, -0.001624]$, median -0.001689, displacement ± 0.000065

With the precedent results we decided to introduce a new tool to better visualize and understand in a geometric way what is really happening: 3D-plotting. Using predictions from *predict_generator()*, we made a comparison with ground truth values, finding a spatial meaning for our results. In particular, we used random sampling in both pred and gt sets to give a better look on what's happening (plotting thousand of points may look terrible and counterintuitive). To be exact, we chose 200 points per class.

In Fig. 8 there is a first 3D-plot for DOA predictions, which are placed on the left. On the right it's possible to see how ground truth points are affected by the presence of outliers, which are not always well predicted by the model and, by the fact, suffers from. Further observing, it seems that the number of points in the two frames is different: this is just a perception due to the fact that many ground truth points overlap while those on the left are badly predicted.

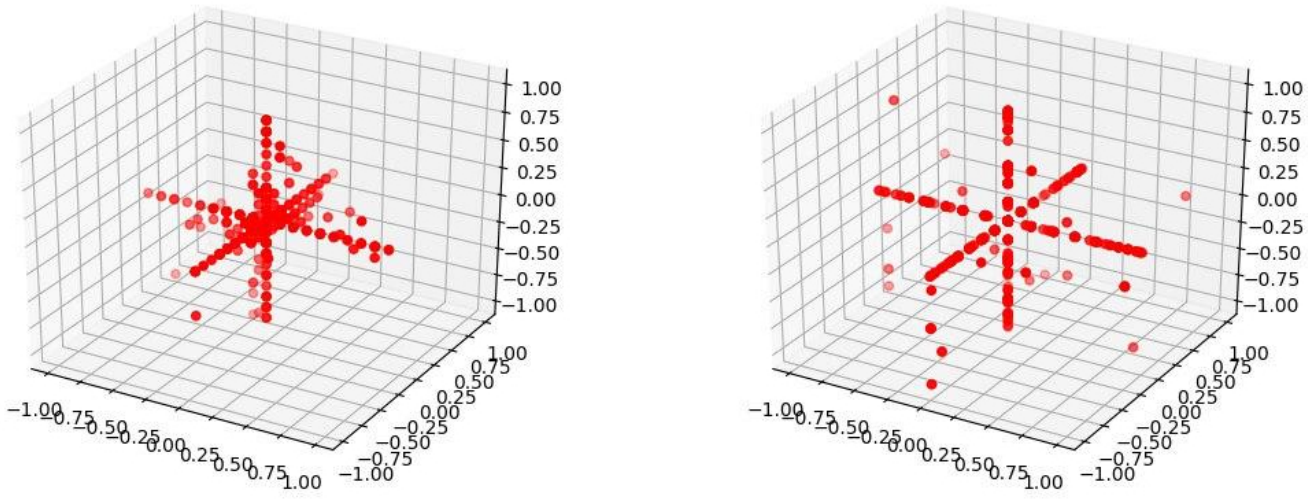

Figure 8: (Left) DOA prediction points
(Right) DOA ground truth points

We made then other two plots, Fig. 9 and Fig. 10, which are the combination of DOA and SED. In the first one all the classes have been plotted to get a total overview and behaviour of the network. The second one refers to 3 classes only to better understand predictions in a smaller set of points.

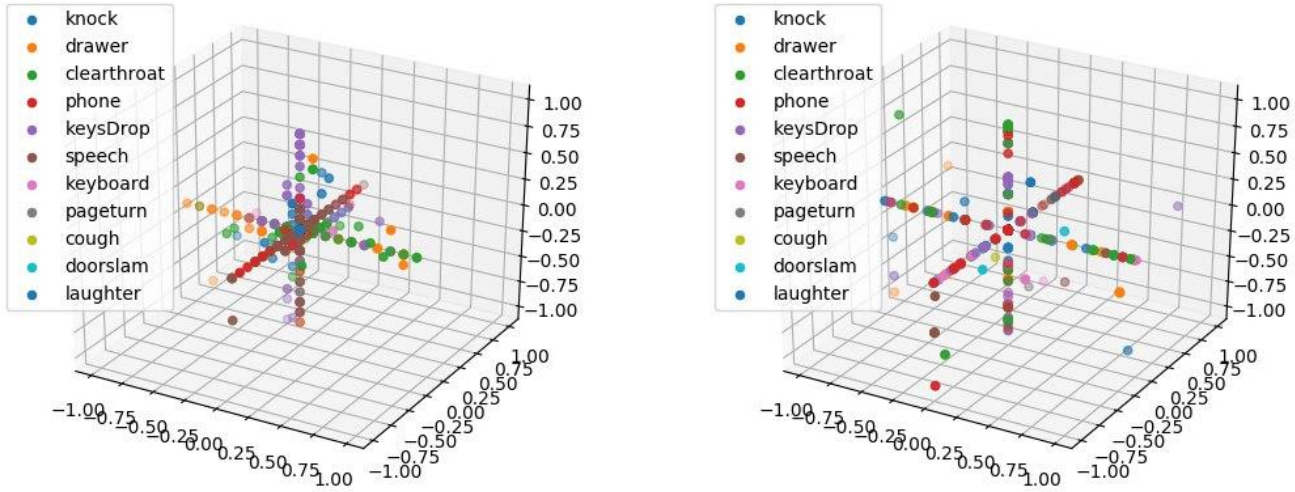

Figure 9: (Left) SELD prediction points
(Right) SELD ground truth points

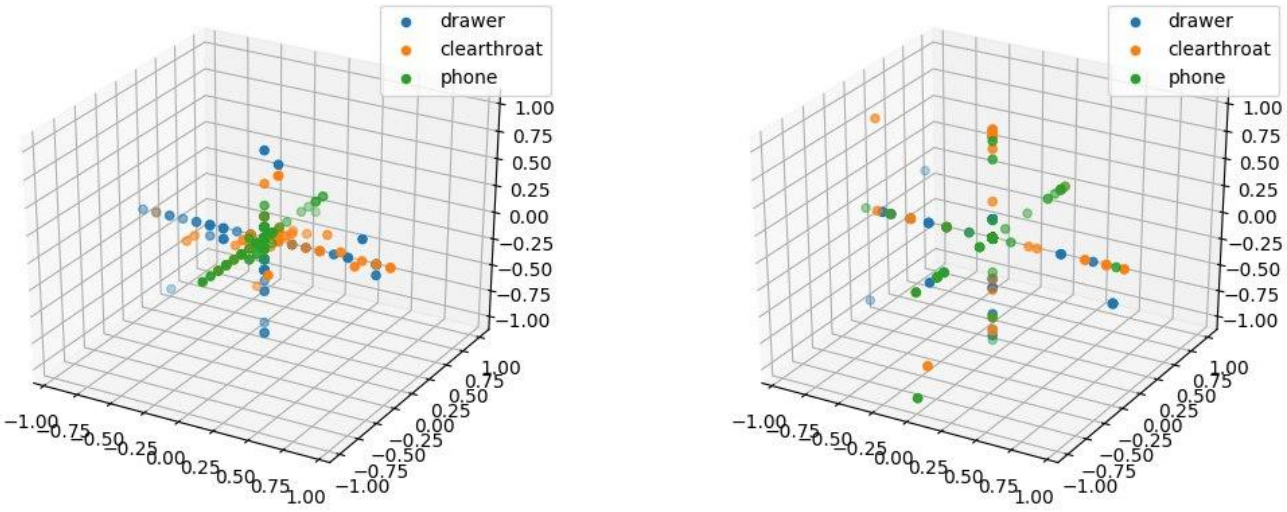

Figure 10: (Left) SELD prediction points for 3 classes
(Right) SELD ground truth points for 3 classes

6 Conclusion

Main objective of this work was to let the SELDnet project work with different type of data sets, in particular the First Order Ambisonics TAU Spatial Sound Events 2019. We managed to deal with it starting from three main papers ([1], [2], [3]) and understanding how SELDnet works.

We tried not to find the easiest solution but one of the most optimized and better-performing ones. We proceeded with a technical analysis of the code and a training process to allow us to produce valid results. In the second part, we presented different plots of our outcomes to realize and comment out how the network is performing.

We consider ourselves satisfied with the results obtained. We didn't deal with a specific study of the network and its architecture but we leave this as future work.

References

[1] Michela Ricciardi Celsi, Simone Scardapane and Danilo Comminiello, *Quaternion Neural Networks for 3D Sound Source Localization in Reverberant Environments*, DIET Dept., Sapienza University of Rome, 2020.

[2] Danilo Comminiello, Marco Lella, Simone Scardapane, and Aurelio Uncini, *Quaternion Convolutional Neural Networks For Detection And Localization Of 3D Sound Events*, DIET Dept., Sapienza University of Rome, 2019.

[3] Lauréline Perotin, Romain Serizel, Emmanuel Vincent and Alexandre Guérin, *CRNN-based multiple DoA estimation using Ambisonics acoustic intensity features*, Submited to the IEEE Journal of Selected Topics in Signal Processing, Special Issue on Acoustic S.. 2018.

[4] Seld-dcase2019 (Link)

[5] Confidence Intervals for Machine Learning (Link)