Overview

The current task at hand is an acoustic source localization project. This document describes in detail the current steps taken and provides info on the performance of each of researched models.

In the end, as with the performance of each of the models being provided, one can easily observe that convolutional raw models (as to be defined later) are clearly outperforming the other studied models for this task of regression.

I. DATA

The data consists of 2 major datasets obtained from physical measurements, sampled by a period of 10. The datasets were combined into a single one, containing nearly 21000 data points. With the noise filtering stage applied, we got nearly 10000 raw data points, ready to be used for training after a crucial preprocessing step. The details of the filtering and preprocessing stages are provided below.

1.1 - Noise Filtering

The filter used for noise reduction in the raw initial data represents a simple architecture. It compares a real multiple of the mean of the absolute value of a signal, with its mean of the minimum and maximum points along the vertical axis, to filter out data points with a controllably sufficient amount of noise. This is shown in figure 1.

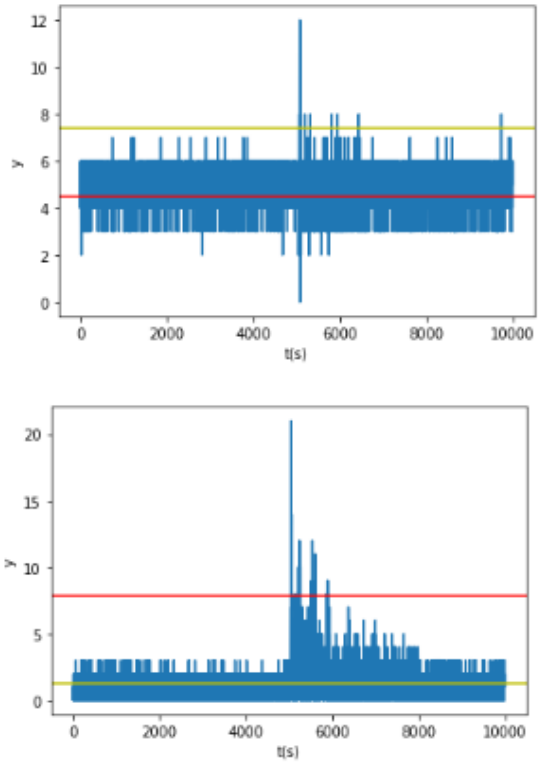

Figure 1. The (top) signal is detected as noise, with the cutoff line being upper than the red baseline. The (bottom) signal is not detected as noise, due to its yellow line being below the base noise line.

The inequality describing the designed filter is as follows, with the left and right sides correspondingly describing the yellow and red lines in figure 1:

$$\frac{E(X)}{\sigma(X)} * 1.15 \geq (X.min + X.max) * 0.375$$

1.2 - Preprocessing

The preprocessing stage entails the data splitting and standardization phases. Clearly worth noting is, that a preprocessing step relevant to each model is done via the dataset generation abstractions. The following summarizes the preprocessing done on the raw data before feeding into the dataset generation stage.

- The input data, consisting of nearly 10000 data points, was split into 0.90, 0.067, and 0.03 shares for the train, validation, and test sets respectively.

- The signals were z-standardized, giving each a mean of 0 and variance of 1 irrespective of other data points in the working set.
- The labels were column-wise standardized respective to the data in the train set only.

1.3 - Data Generation

The *DataGenerator* class instantiates and eases the use of *tf.data.Dataset* objects for model training. This would ultimately ensure low access time, increased capacity, and efficient memory usage. The class also provides some mapped functions, which output the normalized, FFT, and spectrogram outputs out of the original data.

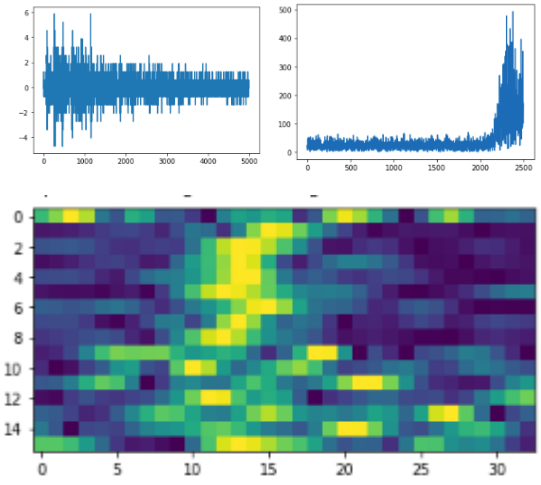

Figure 2. Some raw (top left), FFT (top right), and spectrogram (bottom) outputs of the DataGenerator object.

II. MODELS

Several models were trained on raw, FFT, and spectrogram inputs. The following sections summarize the currently achieved results.

2.1 - Raw Models

Linear, dense, convolutional, and sequential networks were trained on the raw data points. With the prediction quality of the linear and dense models being obviously low, the results related to 2 convolutional networks, 1 convolutional residual network, and 1 stacked LSTM model are described in detail below.

2.1.1 - Convolutional Models

2.1.1.1 - Conv 1

This network's architecture was majorly inspired by [1]. Figure 3 summarizes the overall architecture of this model.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 5000, 1)]	0
conv1d_5 (Conv1D)	(None, 4994, 96)	768
batch_normalization_6 (BatchNormalization)	(None, 4994, 96)	384
activation_6 (Activation)	(None, 4994, 96)	0
max_pooling1d_3 (MaxPooling1D)	(None, 1664, 96)	0
conv1d_6 (Conv1D)	(None, 1664, 96)	64608
batch_normalization_7 (BatchNormalization)	(None, 1664, 96)	384
activation_7 (Activation)	(None, 1664, 96)	0
conv1d_7 (Conv1D)	(None, 1660, 128)	61568
batch_normalization_8 (BatchNormalization)	(None, 1660, 128)	512
activation_8 (Activation)	(None, 1660, 128)	0
max_pooling1d_4 (MaxPooling1D)	(None, 553, 128)	0
conv1d_8 (Conv1D)	(None, 553, 128)	82048
batch_normalization_9 (BatchNormalization)	(None, 553, 128)	512
activation_9 (Activation)	(None, 553, 128)	0
max_pooling1d_5 (MaxPooling1D)	(None, 110, 128)	0
conv1d_9 (Conv1D)	(None, 110, 128)	49280
batch_normalization_10 (BatchNormalization)	(None, 110, 128)	512
activation_10 (Activation)	(None, 110, 128)	0
flatten_1 (Flatten)	(None, 14080)	0
dense_2 (Dense)	(None, 128)	1802368
batch_normalization_11 (BatchNormalization)	(None, 128)	512
activation_11 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 2)	258

Total params: 2,063,714
Trainable params: 2,062,306
Non-trainable params: 1,408

Figure 3. Conv 1 architecture

A Glorot uniform [2] initialization, along with a dropout layer with a rate of 0.5 were used. Also for the learning rate, a step reduction approach was taken, setting the learning rate

to 0.1 the previous value on every plateau down to a minimum value of 1e-7. Also for the optimization, the standard Adam [3] optimizer was used. Also, BN [4] was applied before every activation in the network.

2.1.1.2 - Conv 2

The second major convolutional network used for the current project was a network with a different convolutional base block. Figure 4 shows the structure of this model.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 5000, 1)]	0
conv1d (Conv1D)	(None, 4994, 96)	768
batch_normalization (BatchNormalization)	(None, 4994, 96)	384
activation (Activation)	(None, 4994, 96)	0
max_pooling1d (MaxPooling1D)	(None, 1664, 96)	0
conv1d_1 (Conv1D)	(None, 1664, 96)	64608
batch_normalization_1 (BatchNormalization)	(None, 1664, 96)	384
activation_1 (Activation)	(None, 1664, 96)	0
conv1d_2 (Conv1D)	(None, 1660, 128)	61568
batch_normalization_2 (BatchNormalization)	(None, 1660, 128)	512
activation_2 (Activation)	(None, 1660, 128)	0
max_pooling1d_1 (MaxPooling1D)	(None, 553, 128)	0
conv1d_3 (Conv1D)	(None, 553, 128)	82048
batch_normalization_3 (BatchNormalization)	(None, 553, 128)	512
activation_3 (Activation)	(None, 553, 128)	0
max_pooling1d_2 (MaxPooling1D)	(None, 110, 128)	0
conv1d_4 (Conv1D)	(None, 110, 128)	49280
batch_normalization_4 (BatchNormalization)	(None, 110, 128)	512
activation_4 (Activation)	(None, 110, 128)	0
flatten (Flatten)	(None, 14080)	0
dense (Dense)	(None, 128)	1802368
batch_normalization_5 (BatchNormalization)	(None, 128)	512
activation_5 (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

Total params: 2,063,714
Trainable params: 2,062,306
Non-trainable params: 1,408

Figure 4. Conv 2 architecture

This model also used the same training properties as Conv 1, but with dropout on the last layer's input set to 0.0.

2.1.2 - Residual Convolutional Models

Residual models with skip connections have been proved pretty powerful for image classification tasks. We investigate whether the same properties hold for our training as well.

2.1.2.1 - ResConv 1

This model was built on top of the models introduced in [5]. It uses skip connections to make generating the response of a shallower network trivial for a quite deep network. It's important to note that, we're using end-to-end 1D convolutions in this network.

The architecture of this model consists of conv_blocks and identity_blocks. The identity blocks have no trainable parameters on the skip connection, while the conv_blocks have a convolutional layer added back to the skip connection as well, to account for initial model complexity. They also make shape transitioning smooth for assembling identity_blocks in the network with different input and output shapes. Each of the general block types also features a customizable stride length and uses BN for accelerated training.

This model is just a shallower counterpart to the 2D, 50 layer model introduced in [5]. It uses 64 initial convolutional filters with kernel_size of 7, an initial max-pooling layer, and 3 levels of conv_blocks and identity_blocks used as shown below.

[Conv_block [64, 64, 256] / 1
Identity_block [64, 64, 256] / 1
Identity_block [64, 64, 256] / 1
Conv_block [128, 128, 512] / 2
Conv_block [256, 256, 1024] / 2
Average_pooling [2] / 1]

2.1.3 - Sequential Models

Here, we've only used stacked LSTM models for recurrent inference so far, due to efficiency, and the observed performance of such networks for our regression task. Note that we used initial average_pooling layers to lower the cardinality of the input data to be fed into the stacked LSTM network. Figure 5 shows a view of this model's architecture.

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 5000, 1)	0
average_pooling1d_3 (AveragePooling1D)	(None, 1666, 1)	0
average_pooling1d_4 (AveragePooling1D)	(None, 555, 1)	0
lstm_2 (LSTM)	(None, 555, 64)	16896
lstm_3 (LSTM)	(None, 555, 64)	33024
lstm_4 (LSTM)	(None, 555, 64)	33024
flatten_2 (Flatten)	(None, 35520)	0
dense_2 (Dense)	(None, 2)	71042

Total params: 153,986
Trainable params: 153,986
Non-trainable params: 0

Figure 5. The stacked LSTM network's structure

2.2 - FFT Models

The FFT models take as input the FFT of a raw signal and are trained to label the input signals with the regressive source positions in the x-y plane. Here, we trained the exact same Conv 1 and Conv 2 models on FFT inputs, and the results are summarized in the performance section.

2.3 - Spectrogram Models

The spectrogram models take as input the spectrogram of a signal and do 2D vision processing to infer the label output values. Here for spectrogram calculations, we used a frame length of 64, a frame step of 32, and a fft_length of 128, giving us a pretty wide window to apply 2D convolutions on.

The main model we used for our spectrogram processing, is based on the model developed in [6]. As a CRNN model, this model first used 2D convolutions to lower the dimensionality of the input, and later apply stacked sequential layers to the outputs of the initial layers. BN was used among the convolutional layers, and dropout was used after each LSTM layer in the developed model. Figure 6 shows the structure of this model.

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 16, 33, 1)]	0
conv2d (Conv2D)	(None, 8, 17, 32)	14464
batch_normalization_116 (BatchNormalization)	(None, 8, 17, 32)	128
activation_116 (Activation)	(None, 8, 17, 32)	0
conv2d_1 (Conv2D)	(None, 8, 9, 32)	236576
batch_normalization_117 (BatchNormalization)	(None, 8, 9, 32)	128
activation_117 (Activation)	(None, 8, 9, 32)	0
conv2d_2 (Conv2D)	(None, 8, 5, 64)	473152
batch_normalization_118 (BatchNormalization)	(None, 8, 5, 64)	256
activation_118 (Activation)	(None, 8, 5, 64)	0
conv2d_3 (Conv2D)	(None, 8, 5, 1)	65
lambda (Lambda)	(None, 8, 5)	0
bidirectional (Bidirectional)	(None, 8, 256)	137216
dropout_1 (Dropout)	(None, 8, 256)	0
bidirectional_1 (Bidirectional)	(None, 8, 256)	394240
dropout_2 (Dropout)	(None, 8, 256)	0
bidirectional_2 (Bidirectional)	(None, 8, 256)	394240
dropout_3 (Dropout)	(None, 8, 256)	0
bidirectional_3 (Bidirectional)	(None, 8, 256)	394240
dropout_4 (Dropout)	(None, 8, 256)	0
bidirectional_4 (Bidirectional)	(None, 8, 256)	394240
flatten_3 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 256)	524544
dense_5 (Dense)	(None, 64)	16448
dense_6 (Dense)	(None, 2)	130

Total params: 2,980,067
Trainable params: 2,979,811
Non-trainable params: 256

Figure 6. The CRNN model used for spectrogram regression

III. PERFORMANCE

Each of the models was trained and got their hyperparameters tuned along the way. The final performance for each of the models is summarized in table 1. Also, figure 7

visualizes the results of the raw models in evaluation against the training and validation sets.

Table I. Overall performance of models

input mapping	model	#trainable params	training r_square	Validation r_square
raw	Conv 1	2,062,306	0.9129	0.4910
raw	Conv 2	2,062,306	0.9367	0.5318
raw	ResConv 1	1,806,596	0.9	0.2638
raw	LSTM 1	153,986	0.75	0.3029
FFT	Conv 1	1,161,186	0.0537	-0.06
FFT	Conv 2	1,161,186	0.89	-0.88
spectrogram	CRNN 1	2,979,811	0.33	0.09

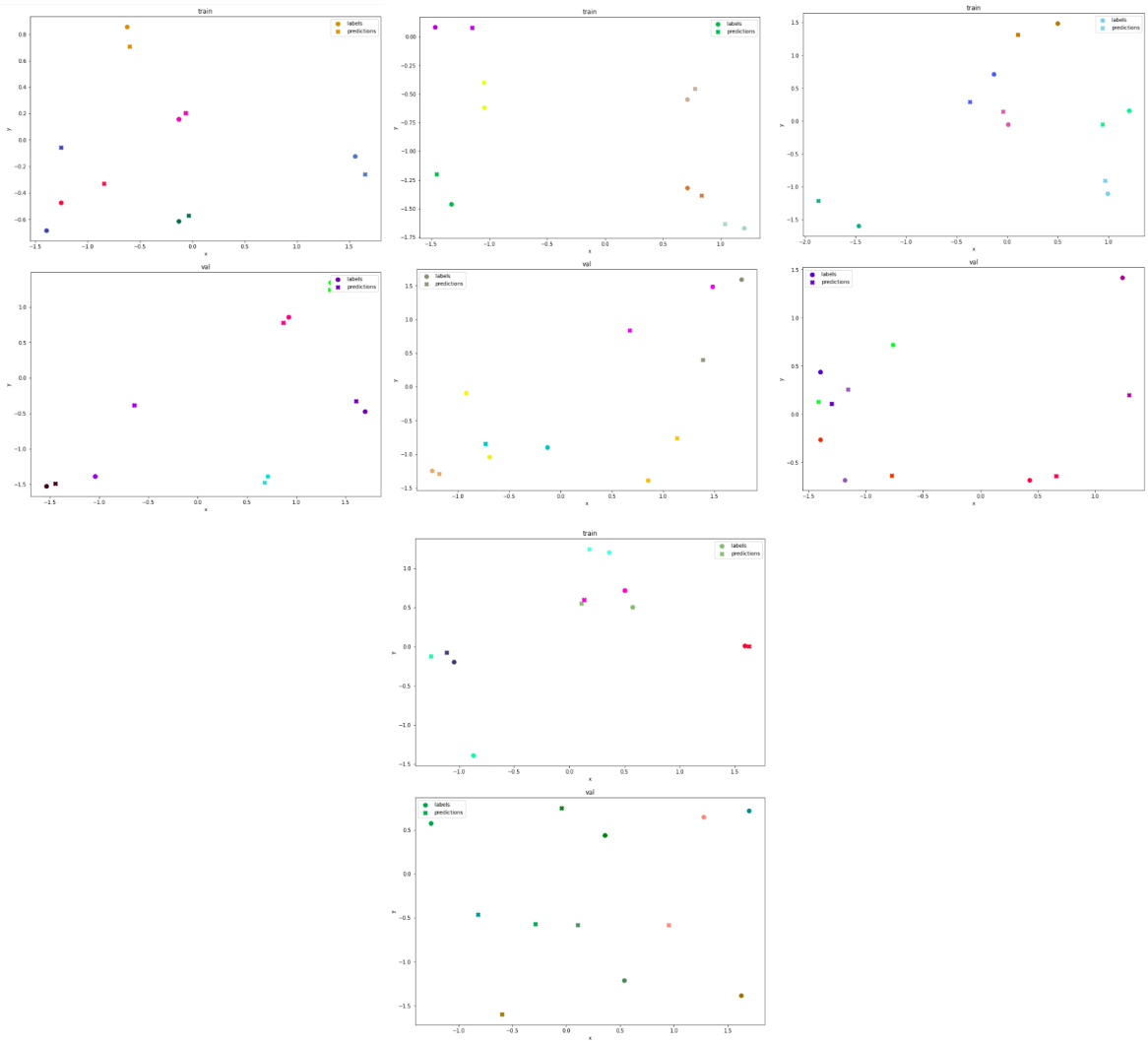

Figure 7. The training and validation inference visualization for Conv 1 (top left), Conv 2 (top center), ResConv1 (top right), and stacked LSTM (bottom) models given the raw inputs

References

[1] J. M. Vera-Diaz, D. Pizarro, and J. Macias-Guarasa, “Towards endto-end acoustic localization using deep learning: from audio signal to source position coordinates,” Sensors, vol. 18, no. 10, p. 3418, 2018.

[2] Glorot, Xavier & Bengio, Y.. (2010). Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track. 9. 249-256.

[3] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[4] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[6] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., ... & Zhu, Z. (2016, June). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning* (pp. 173-182). PMLR.