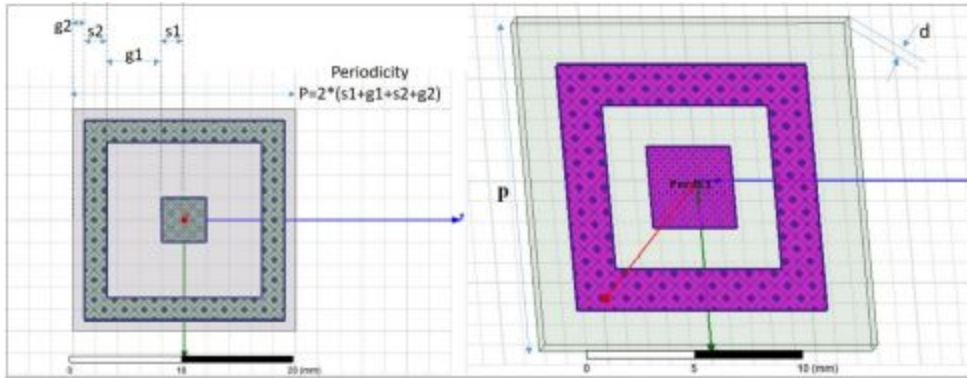


Training ML model for predicting FSS dimensions for given cutoff frequencies



Objective

The objective is to train a machine learning model to predict the dimensions $g1$, $g2$, $s1$ and $s2$ which defines the capacitor and inductor values in an equivalent filter circuit. The training data is obtained using simulations that generate reflective transmission values in dB for frequencies in a given range.

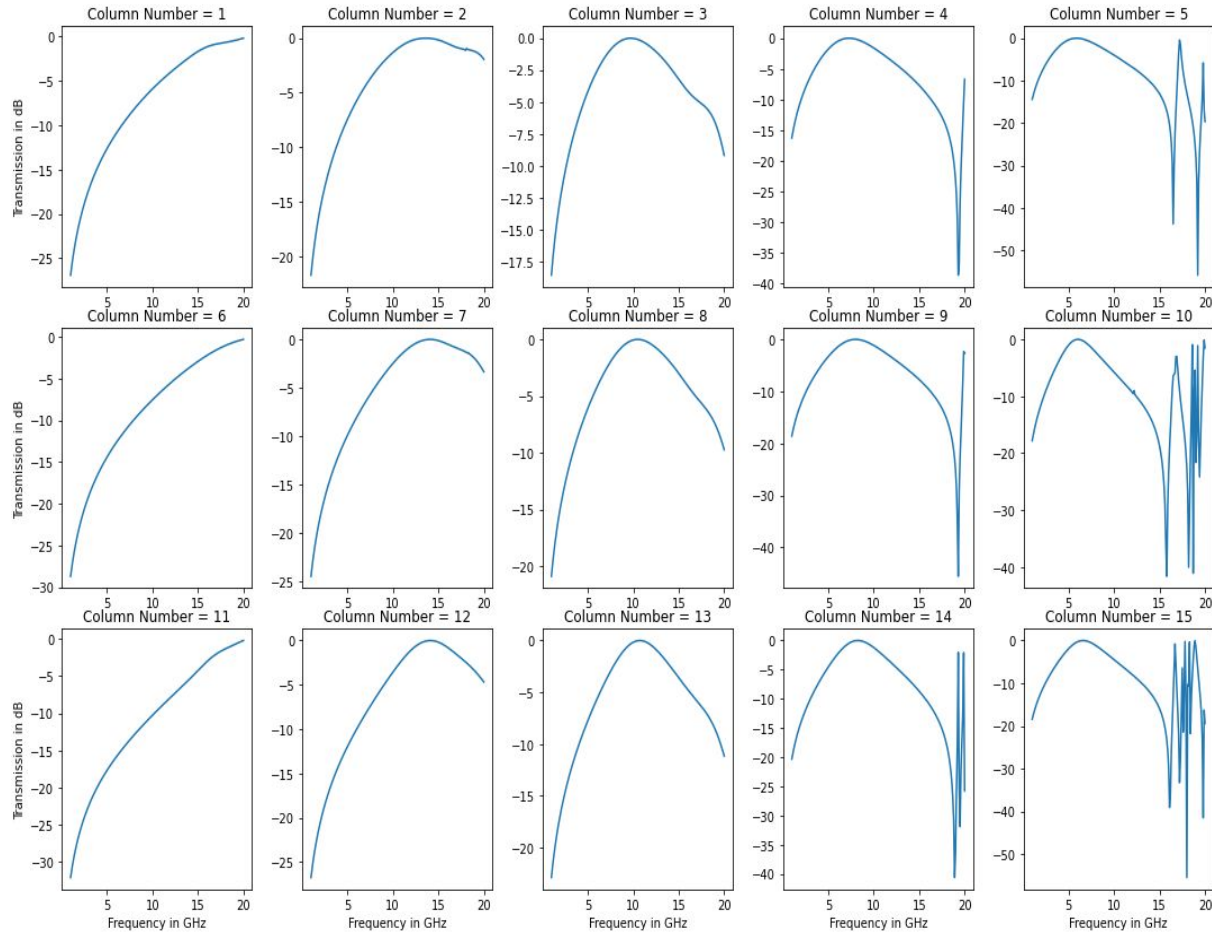
Data

For this project we have a CSV file which has frequencies from 1 GHz to 20 GHz with an increment of 0.1 GHz i.e we have 191 frequency values for all possible combinations $g1$, $g2$, $s1$, $s2$ with given characteristics.

- $g1$ is in the range 0 to 10 mm and increments by 2mm.
 - $g2$ is in the range 0 to 2 mm and increments by 0.5mm.
 - $s1$ is in the range 1 to 5 mm and increments by 1mm.
 - $s2$ is in the range 0.5 to 2 mm and increments by 0.5mm.
-

So in total we have 600 combinations of the parameters. For 20 combinations of these parameters reflective transmission is not defined so effectively there are only 580 useful combinations.

Here are reflective transmission vs frequencies for some of these combinations:



To get the cutoff frequencies from this data we have to isolate the points with transmission more than -0.45 dB but as it is evident from the graph that there can be multiple pairs of cutoff frequencies we will be only focussing on the first pair in this project. As we can see that for some combinations, graph does not cut the -0.45 dB line again so those combinations are also useless for us. So, after removing 251 such combinations only 329 combinations are left.

Implementation

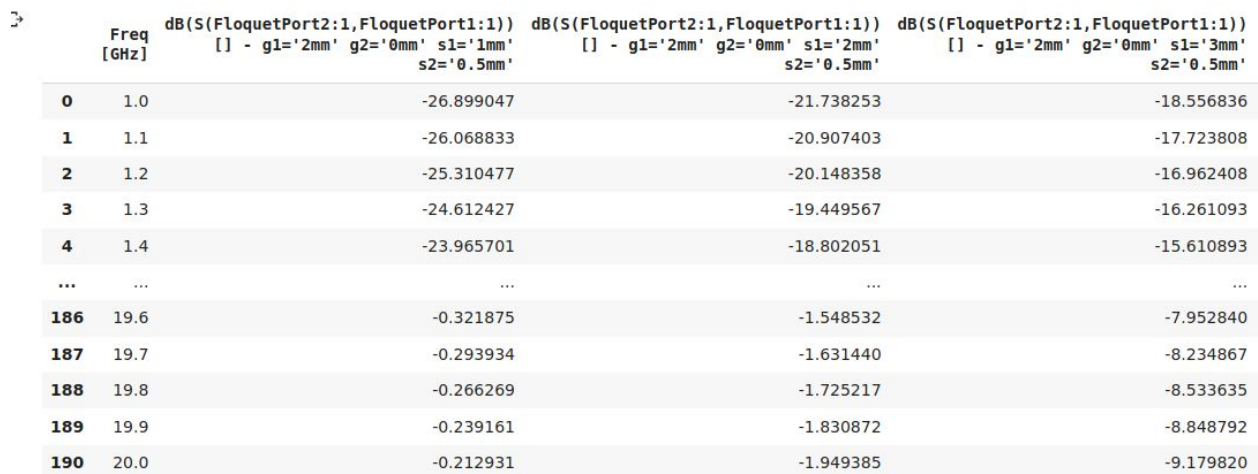
For this project we are using Python 3.6

Python Libraries:

- **Numpy**: For dealing with multi dimensional data matrices
- **Pandas**: For using important data structures like DataFrame
- **Matplotlib**: For plotting and displaying graphs
- **Tensorflow 2.0**: For implementing Neural Network
- **Sklearn**: For additional support

For managing the code smoothly we will also need Jupyter Notebook, a development application and anaconda for managing the python libraries. We can also use Google Colab which provides Jupyter Notebook services remotely.

Moving on to the code, we will first import all the data from the CSV file and load it into `pandas.DataFrame`. Here is how it looks:

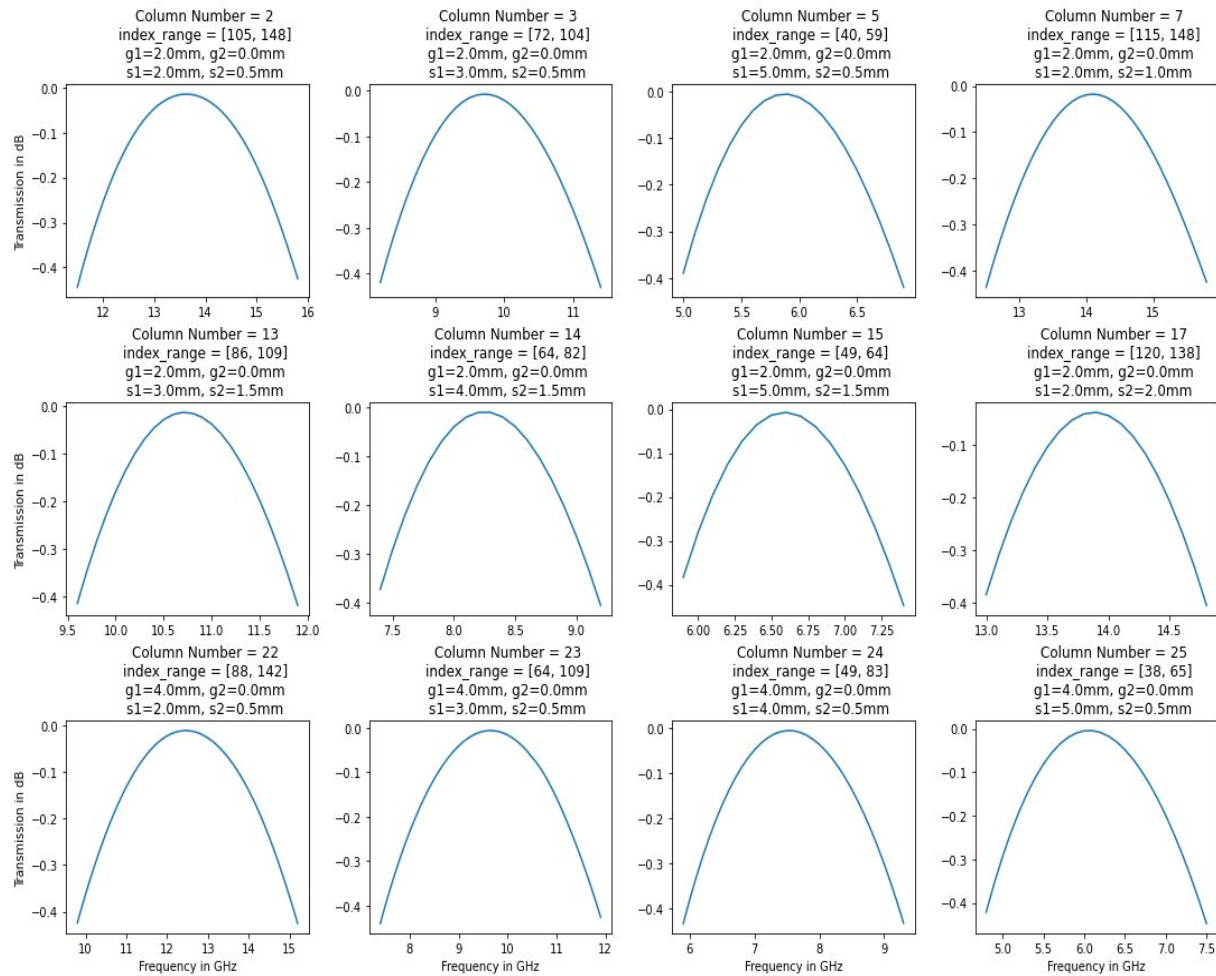


	Freq [GHz]	dB(S(FloquetPort2:1,FloquetPort1:1)) [] - g1='2mm' g2='0mm' s1='1mm' s2='0.5mm'	dB(S(FloquetPort2:1,FloquetPort1:1)) [] - g1='2mm' g2='0mm' s1='2mm' s2='0.5mm'	dB(S(FloquetPort2:1,FloquetPort1:1)) [] - g1='2mm' g2='0mm' s1='3mm' s2='0.5mm'
0	1.0	-26.899047	-21.738253	-18.556836
1	1.1	-26.068833	-20.907403	-17.723808
2	1.2	-25.310477	-20.148358	-16.962408
3	1.3	-24.612427	-19.449567	-16.261093
4	1.4	-23.965701	-18.802051	-15.610893
...
186	19.6	-0.321875	-1.548532	-7.952840
187	19.7	-0.293934	-1.631440	-8.234867
188	19.8	-0.266269	-1.725217	-8.533635
189	19.9	-0.239161	-1.830872	-8.848792
190	20.0	-0.212931	-1.949385	-9.179820

Each column in this DataFrame is a combination of g1, g2, s1, s2. So we will have to drop the useless combinations from this table. To get the cutoff pair of frequencies we will isolate all the set of frequencies corresponding to which the transmission is in the range 0 to -0.45 dB. Since we will be needing only the first pair we will only keep the

first and the last values of the first set. If the second cutoff is not in the range -0.40 dB to -0.45 dB it means the graph doesn't cut the -0.45 dB line so we will remove all such combinations.

Here are reflective transmission vs frequencies for some of these combinations after cleaning and removing the useless set of frequencies:



Since the neural network works best with normalized values we will normalize all the data using the formula: $x' = (x - \text{mean}) / \text{range}$.

Now the data is ready for feeding into the neural network and here is how it looks:

	g1 (mm)	g2 (mm)	s1 (mm)	s2 (mm)	freq1 (GHz)	freq2 (GHz)
0	-0.178723	-0.516717	-0.260638	-0.487335	0.580009	0.536007
1	-0.178723	-0.516717	-0.010638	-0.487335	0.326163	0.290197
2	-0.178723	-0.516717	0.489362	-0.487335	0.080009	0.038800
3	-0.178723	-0.516717	-0.260638	-0.154002	0.656932	0.536007
4	-0.178723	-0.516717	-0.010638	0.179331	0.433856	0.318130
...
324	0.421277	0.483283	0.239362	0.179331	0.126163	0.117013
325	0.421277	0.483283	-0.510638	0.512665	0.326163	0.239918
326	0.421277	0.483283	-0.260638	0.512665	0.256932	0.195225
327	0.421277	0.483283	0.239362	0.512665	0.133856	0.111426
328	0.621277	0.483283	-0.260638	-0.487335	0.156932	0.133773

329 rows × 6 columns

Now we will divide the data into three sets: training, validation and test in the ratio 3:1:1. Training set is the set from which the neural network learns. Validation set helps us understand how well the neural network is learning the new data. Test set is for the final evaluation of the neural network.

Structure of neural network:

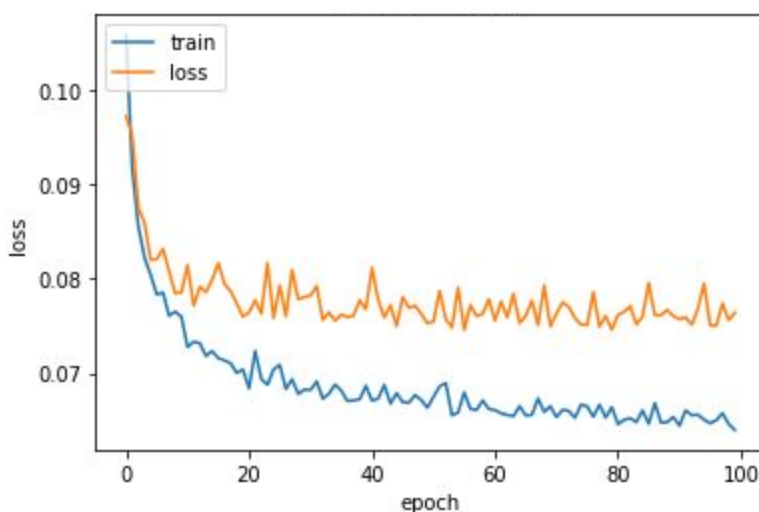
- Input layer: inputs **2** values and produces **16** output values per data point, activation function: **Rectified Linear Unit**
- Hidden layer1 inputs **16** values and produces **128** output values per data point, activation function: **Rectified Linear Unit**
- Hidden layer2 inputs **128** values and produces **512** output values per data point, activation function: **Rectified Linear Unit**
- Hidden layer3 inputs **512** values and produces **128** output values per data point, activation function: **Rectified Linear Unit**

-
- Output layer: inputs **128** values and produces **4** output values per data point, activation function: **Tanh**

The optimization algorithm used in this model is **RMSProp** and the metrics used are **Root Mean Squared Error**.

Results

Here is the training graph for the model:



Since the model is trained using normalized values, to measure the actual RMSE we will have convert the output back to the original form by using the formula:

$$x = (x' * \text{original range}) + (\text{original mean}).$$

Here is the RMSE for each dimension:

- RMSE g1= 1.5mm, increments in training data= 2.0mm
- RMSE g2= 0.5mm, increments in training data= 0.5mm
- RMSE s1= 0.8mm, increments in training data= 1.0mm
- RMSE s2= 0.5mm, increments in training data= 0.5mm

Suggestion

From the results it seems that the error is proportional to the increments in training data for the given parameter. So instead of having a regular data with all possible combinations of parameter values with huge increments it might be better to have random data that might not have all possible combinations of the parameters but has at least few data points with almost every possible value of parameters with the increment $=0.5\text{mm}$ which is the precision required for each parameter.