Neural Network Design Report

CLASSIFICATION OF FAULTY 90-DEGREE HYBRID COUPLERS

MOUSTAFA ELNEANAEY, YOUSSEF ABAZA

1. Introduction

In high-frequency microwave design, ensuring correct performance of components like 90-degree hybrid quadrature couplers is critical. These couplers are sensitive to physical parameters such as metal width, metal length. Small deviations in these can cause failures in meeting required specifications for scattering parameters (S11, S12, S13, S14, ... S44).

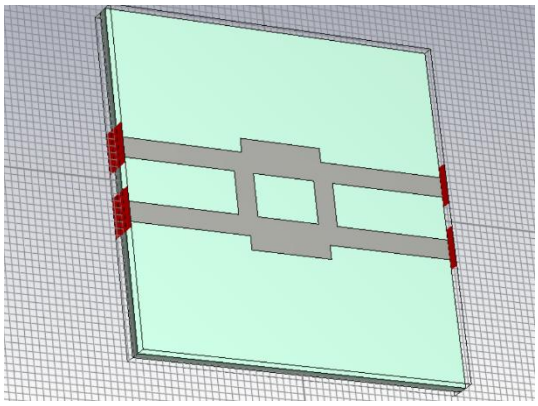
To assist in automated design validation, we developed a neural network that takes S-parameter sweeps as input and predicts whether the design is:

- Correct
- Or has issues related to:
    - Transmission lines lengths
    - Side TLs width
    - Upper and lower TLs widths

This model acts as a classification tool for identifying specific fabrication/design issues based on electromagnetic simulation data from CST.

2. Dataset extraction

With the top and bottom transmission lines having the same Z0 as the system and with the right and left ones having a characteristic impedance of $Z0/\sqrt{2}$ and with a horizontal and vertical separation of lambda/4 (with target frequency being at 4 GHz), the coupler was designed in CST.



L	= 50
W	= 50
H	= 1.5
Z0	= 2.94
lamda	= 49.029
Z0root2	= 5

A parametric sweep was then conducted on the transmission line width, length and Dielectric thickness. 1000 ,495, 499 sample results were obtained for the change of the length, width of the side transmission lines, and width of the top and bottom transmission lines respectively. The results were exported as touchstone files and were then read out using **Dataextract.py** and the data was then stored in **lambdasweep.npy**, **Z0root2sweep.npy**, **Z0sweep.npy respectively**.

3. Dataset Categorization

The file Categorization.py was used to label our S-parameter data by automatically assigning each sample to one of **four categories**:

1. **Correct design**
2. **Faulty due to metal length (λ)**
3. **Faulty due to width of the upper/lower transmission lines (Z0root2)**
4. **Faulty due to width of the side transmission lines (Z0)**

This was done by comparing each sample to a **reference S-parameter vector**. If all 16 S-parameters were within a small **tolerance (±0.04)** of the reference, the sample was labeled as **correct**. If not, the fault was classified based on which design parameter the sample was testing (λ, Z0root2, or Z0).

The result was saved as a **one-hot encoded label** — a 4-element array where only one entry is "1" to indicate the class. These labeled results were then combined and shuffled to be used for training the neural network.

4. Neural Network Inputs/Outputs Description

- Input: `**shuffledsweep**` — Array of real 16 S-parameter values sampled at 4GHz.

- Output: `**shuffledresult**` — One-hot encoded labels for the 4 output categories.

The shuffled data was split into three categories:

1. **Training Data:** **1196 samples (about 60%)**
2. **Validation Data:** **398 samples (about 20%)**
3. **Test Data:** **400 samples (about 20%)**

As the name suggests, **training data** is used to directly train the model — the neural network adjusts its internal weights based on the input and the known correct output (label). **Validation data**, on the other hand, is also used during the training process, but not for weight updates. Instead, it serves as a way to evaluate the model's performance on unseen data after each training cycle (epoch). The results from the validation data are "hidden" from the model during training, meaning the model doesn't learn from them — they are only used to monitor whether the model is improving and to detect issues like overfitting. And **the test data** is finally inputted to evaluate the **accuracy** of the neural network.

5. Model Architecture

We used a feed-forward neural network with two hidden layers. The architecture is as follows:

```
model = Sequential([
    Dense(64, activation='relu'),      # Hidden layer with 64 neurons
    Dense(64, activation='relu'),       # Hidden layer with 64 neurons
    Dense(4, activation='softmax')     # Output layer for 4 classes
])
```

Two hidden layers allow the network to learn both low-level and high-level patterns from S-parameter data.

64 neurons per layer provide a good balance of learning capacity and computational efficiency.

ReLU activation introduces non-linearity while keeping training efficient.

Softmax activation in the final layer outputs a probability distribution over the four design categories and take the highest probability.

6. Training Strategy

Compilation:

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

- **Adam Optimizer:** Chosen for its adaptive learning rate and robust convergence behavior.
- **Categorical Crossentropy:** Appropriate for one-hot encoded multi-class classification.

Training Process:

```
early_stop = EarlyStopping(monitor='val_loss', patience=5)
model.fit(
    x_train, y_train,
    validation_data=(x_validation, y_validation),
    epochs=100,
    callbacks=[early_stop],
    batch_size=32,
    verbose=1,)
```

Epochs = 100 in addition to callbacks=[early_stop]: Allows the model to converge to a stable solution without significant overfitting by stopping early if validation accuracy does not increase after 5 epochs.

Batch size = 32: Provides a good trade-off between computational load and gradient stability.

Separate validation set: Ensures unbiased evaluation during training.

6. Evaluation and Metrics

After training, the model was evaluated on a hold-out test set to assess its generalization:

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
```

- **test_acc** indicates the model's ability to classify unseen designs correctly.

An accuracy of 96% was achieved, demonstrating that the model is highly effective at correctly classifying the different design conditions. This level of performance reflects the model's strong ability to learn meaningful patterns from the S-parameter data and generalize well to unseen cases.

7. Interpretability & Engineering Relevance

- The network learns patterns in the S-parameter frequency response that correspond to physical design flaws.
- These patterns are often too subtle or nonlinear for traditional rule-based systems.
- By classifying based on learned electromagnetic behavior, the model helps:
    - Identify where tolerances are violated
    - Accelerate verification in design cycles
    - Suggest corrective actions (e.g., adjust metal width)

8. Future Work

- Use real and phase information of S parameters for more accurate classification.
- Optimize the neural network topology
- Add more parameters to classify like Dielectric thickness, and faults like disconnected transmission lines

9. Conclusion

This neural network is a fast, accurate tool for automatically classifying quadrature hybrid designs based on their S-parameter behavior. Its design balances complexity, training efficiency, and domain relevance — making it a useful assistant for microwave circuit designers in both simulation and post-fabrication testing stages.