# **ONT | Task-2**

## **Artificial Neural Network**

## **Introduction**

This report expands on the process of building and training an Artificial Neural Network (ANN) for predicting Ground Segment Signal-to-Noise Ratio (GSNR) using supervised learning methods and TensorFlow with Keras Tuner. The primary goal is to minimize the validation loss and mean absolute error (MAE) for accurate GSNR estimations.

**Task: GSNR Prediction using Supervised Learning**

Supervised learning involves training a model on labeled data, where each data point has both input features and a corresponding target value. In this case, the model aims to learn the relationship between various input features that might influence GSNR and predict the actual GSNR values.

Here's a breakdown of the task:

1. **Input Features:** The input features could represent factors that potentially affect GSNR, such as:
   * Signal characteristics (e.g., frequency, power)
   * Channel characteristics (e.g., bandwidth, noise level)
   * Environmental factors (e.g., weather conditions)
   * System parameters (e.g., antenna gain, receiver sensitivity)
2. **Target Variable:** The target variable is the actual GSNR value for each data point.
3. **Model Prediction:** The trained ANN model will take unseen input features as input and predict the corresponding GSNR value.

**Hyperparameter Tuning with Grid Search**

Keras Tuner offers various hyperparameter tuning techniques, and this report focuses on Grid Search. Grid Search involves systematically evaluating a predefined set of values for each hyperparameter to identify the combination that yields the best model performance. Here's how it applies to GSNR prediction:

1. **Hyperparameter Selection:** Common hyperparameters for ANN regression include:
   * **Learning Rate:** Controls the step size during weight updates. A grid search might define different learning rates (e.g., 0.01, 0.001, 0.0001) to explore their impact.
   * **Batch Size:** Number of data points used for a single weight update. Grid search could define various batch sizes (e.g., 16, 32, 64) to analyze their effect on training efficiency.
   * **Number of Hidden Layers and Units:** Defines the model's complexity. Grid search could explore different combinations (e.g., 1-3 layers with 16-128 units) to find the optimal architecture.
   * **Activation Function:** Introduces non-linearity. Grid search could evaluate different functions (e.g., ReLU, tanh) to choose the most suitable one.
   * **Optimizer:** Optimization algorithm used for weight updates. Grid search could include different optimizers (e.g., Adam, SGD) to compare their performance.
2. **Evaluation Metric:** Similar to the report, validation loss (e.g., MSE) or MAE can be used to evaluate the performance of models trained with different hyperparameter combinations.
3. **Grid Search Execution:** Keras Tuner allows defining a grid search for each hyperparameter. The tuner then trains models with all possible combinations from the defined grids and selects the one with the lowest validation loss (or highest accuracy depending on the metric).

**Analysing Hyperparameter Effects**

By analysing the results of the grid search, we can gain valuable insights into the effects of different hyperparameters on GSNR prediction:

* **Learning Rate:** A high learning rate might lead to unstable training with fluctuating loss, while a low learning rate could cause slow convergence. The optimal value will depend on the specific dataset and model complexity.
* **Batch Size:** Larger batch sizes can improve training efficiency by utilizing more data for each update. However, with limited memory, smaller batch sizes might be necessary. Identifying the optimal batch size balances efficiency with memory constraints.
* **Number of Hidden Layers and Units:** More layers and units increase model complexity, allowing it to capture complex relationships. However, an overly complex model can lead to overfitting. Grid search helps find the balance between model capacity and generalization.
* **Activation Function:** Different activation functions introduce different non-linearities. The optimal choice depends on the specific problem and data distribution. Grid search allows us to identify the function that best aids the model in learning the GSNR relationship.
* **Optimizer:** Different optimizers update weights in various ways. Grid search helps to determine which optimizer performs best for optimizing the model weights and minimizing the GSNR prediction error.

## **Libraries and Dependencies**

This section details the essential libraries used for building, training, and evaluating the ANN model for GSNR prediction:

**1. TensorFlow:**

TensorFlow is a powerful open-source library for numerical computation and large-scale machine learning. It provides:

* **Core symbolic framework:** Enables defining computational graphs representing the model architecture.
* **Eager execution:** Allows for imperative programming, facilitating interactive development and debugging.
* **Automatic differentiation:** Simplifies calculating gradients for training the model.
* **Hardware acceleration:** Leverages GPUs or TPUs for faster training on compatible hardware.

In this project, we'll utilize TensorFlow's capabilities to:

* Build the ANN architecture with layers and activation functions.
* Define the loss function (e.g., MSE) for measuring prediction error.
* Implement the optimizer (e.g., Adam) for updating model weights during training.
* Train the model on the provided GSNR dataset.

**2. Keras:**

Keras is a high-level API built on top of TensorFlow, providing a user-friendly interface for building and training neural networks. It offers:

* Pre-built layers: Simplifies creating common network components like Dense, Dropout, and Convolutional layers.
* Sequential model API: Enables constructing models by stacking layers sequentially.
* Functional API: Offers greater flexibility for defining complex model architectures.

Here, we'll leverage Keras' functionalities to:

* Define the ANN model structure using a sequential or functional API, depending on the desired complexity.
* Compile the model by specifying the optimizer, loss function, and metrics (e.g., MAE) for evaluation.
* Train the model by fitting it on the GSNR data with validation data for performance monitoring.
* Evaluate the trained model's performance on unseen data using metrics like MAE.

**3. Keras Tuner (kt):**

Keras Tuner is an extension library for Keras, specifically designed for hyperparameter tuning. It facilitates:

* Defining the model building function with hyperparameter spaces.
* Specifying the hyperparameter search algorithm (e.g., Grid Search, Random Search).
* Launching the hyperparameter tuning search to identify the optimal configuration.
* Monitoring the search progress and visualizing results.

In this project, we'll use Keras Tuner to:

* Define a search space for each hyperparameter like learning rate, batch size, and number of hidden layers.
* Utilize Grid Search to systematically evaluate all possible combinations within the defined search spaces.
* Train models with each hyperparameter combination and track their performance on the validation set.
* Identify the hyperparameter configuration that yields the best model performance (lowest validation loss/MAE).

**4. NumPy (np):**

NumPy is a fundamental library for scientific computing in Python. It provides:

* Efficient multi-dimensional array manipulation.
* Mathematical operations on arrays.
* Linear algebra functions.

We'll use NumPy to:

* Load and pre-process the GSNR dataset, including feature scaling or normalization if necessary.
* Convert data into NumPy arrays for efficient processing within TensorFlow.
* Perform calculations and manipulations on the training and validation data.

**5. Pandas (pd):**

Pandas is a powerful library for data analysis and manipulation in Python. It offers:

* Data structures like DataFrames for handling tabular data.
* Data cleaning and pre-processing functionalities.
* Exploratory data analysis tools for understanding the dataset.

We'll leverage Pandas to:

* Load the GSNR data from a CSV file or another supported format.
* Explore the data by examining its statistics and identifying potential missing values or outliers.
* Perform essential pre-processing steps like handling missing values, converting categorical features, and feature scaling.

**6. Matplotlib (plt):**

Matplotlib is a popular library for creating static, animated, and interactive visualizations in Python. It allows us to:

* Generate various plots like line charts, scatter plots, and histograms.
* Visualize the training process by plotting the loss and MAE curves over epochs.
* Analyze the relationship between features and GSNR values using scatter plots.

We'll use Matplotlib to:

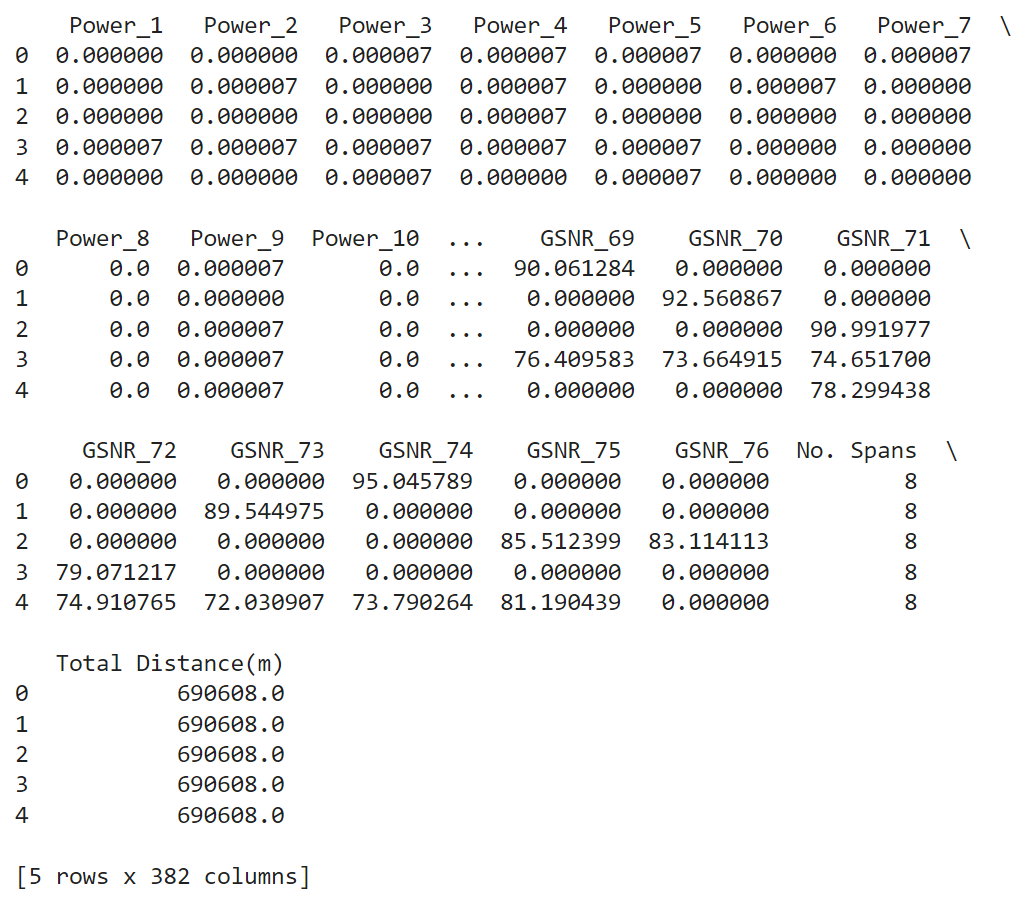
* Visualize the training and validation loss/MAE curves to monitor model convergence.
* Potentially plot feature importances (if applicable) to understand the contribution of each feature to GSNR prediction.
* Generate other visualizations as needed to gain insights into model performance and data characteristics.

## **Loading Data**

For this task, we processed the data is in a excel file format. We load the data using Pandas and perform any necessary preprocessing steps.

## **Feature Selection**

This section details the process of selecting relevant features and extracting the target variable (GSNR) from the loaded GSNR dataset.



**1. Feature Selection:**

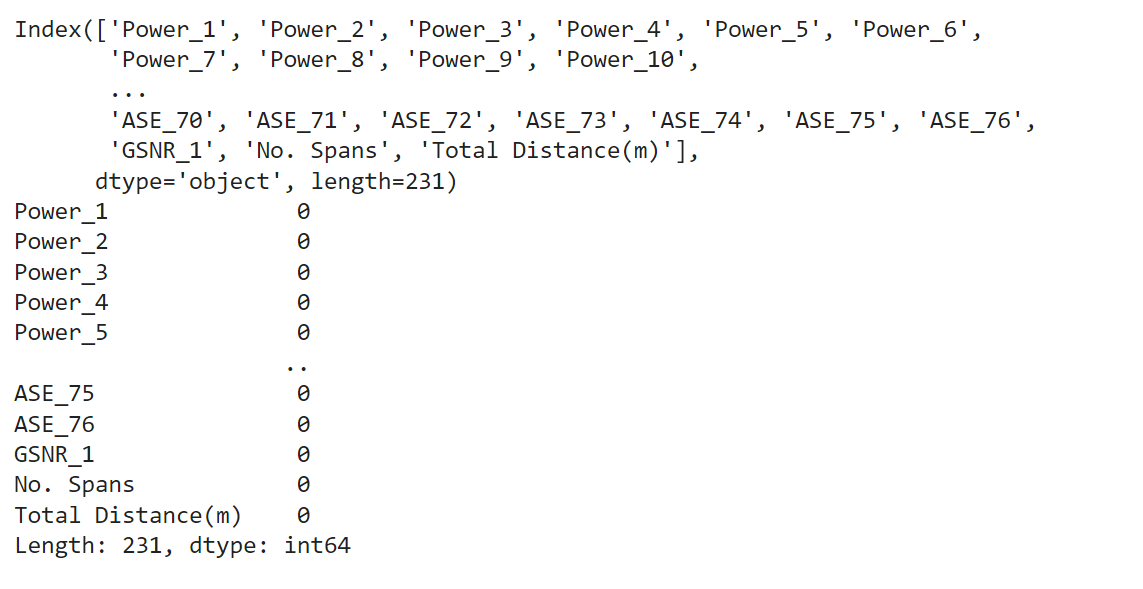
We will focus on a subset of features deemed most relevant to predicting GSNR. This might involve excluding features like frequencies (frequency\_1 to frequency\_76) and additional GSNR values (GSNR\_2 to GSNR\_76) that might not be necessary for the model. Domain knowledge and exploratory data analysis can guide this selection process.

**2. Target Variable Extraction:**

The target variable of interest is GSNR\_1, representing the ground segment signal-to-noise ratio we aim to predict.

**3. Maintaining Consistency:**

During data splitting for training and validation, it's crucial to ensure consistency with the selected features. The splitting process should consider only the chosen features (excluding frequencies and additional GSNR values) and the target variable GSNR\_1.



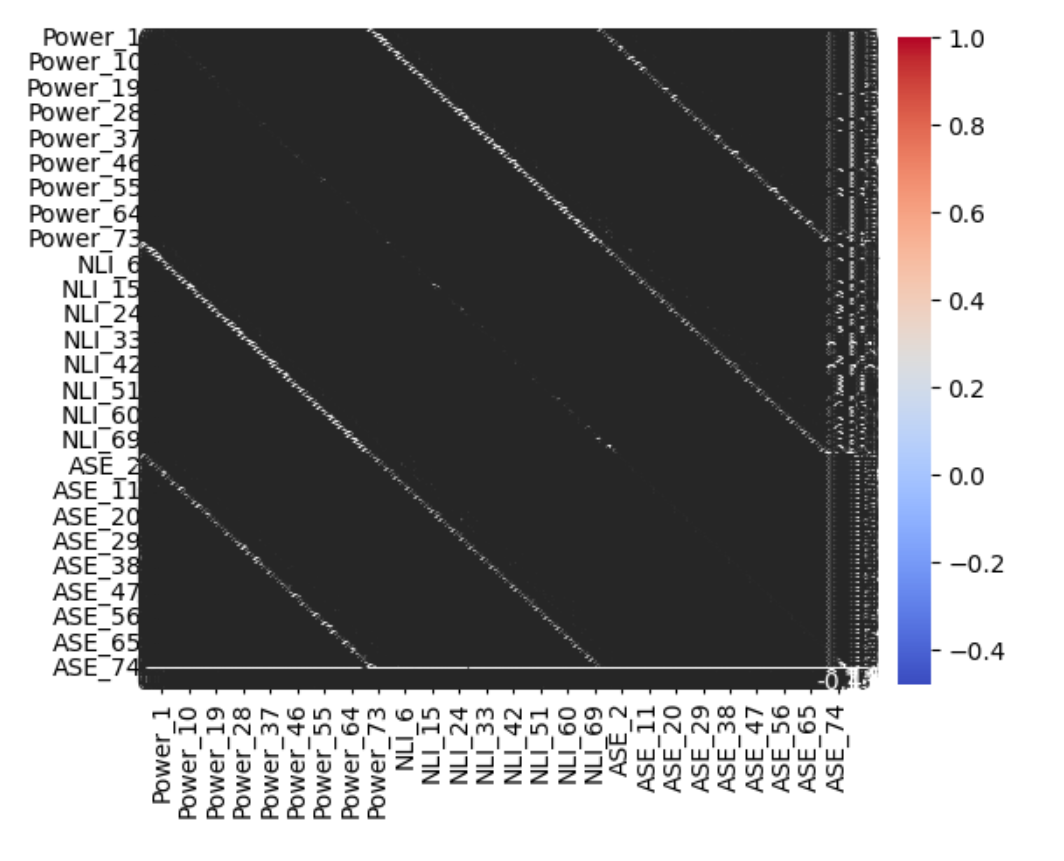
## **Visualisation**

This section details the process of visualizing the correlation matrix and the distribution of features in relation to the target variable (GSNR) to gain insights into the data and potential relationships.

**1. Visualizing the Correlation Matrix:**

* A correlation matrix (corr\_matrix) is calculated using the Pandas corr function. This matrix shows the correlation coefficient between each pair of features in the data.
* Seaborn's heatmap function is used to create a visual representation of the correlation matrix.
* The annot=True argument displays the correlation coefficients directly on the heatmap, aiding in interpreting the strength and direction of relationships between features.
* The cmap='coolwarm' argument sets the colormap to a coolwarm scheme, where blue represents negative correlations, white represents no correlation, and red represents positive correlations. The intensity of the color indicates the strength of the correlation.
* Finally, plt.show() displays the generated heatmap.

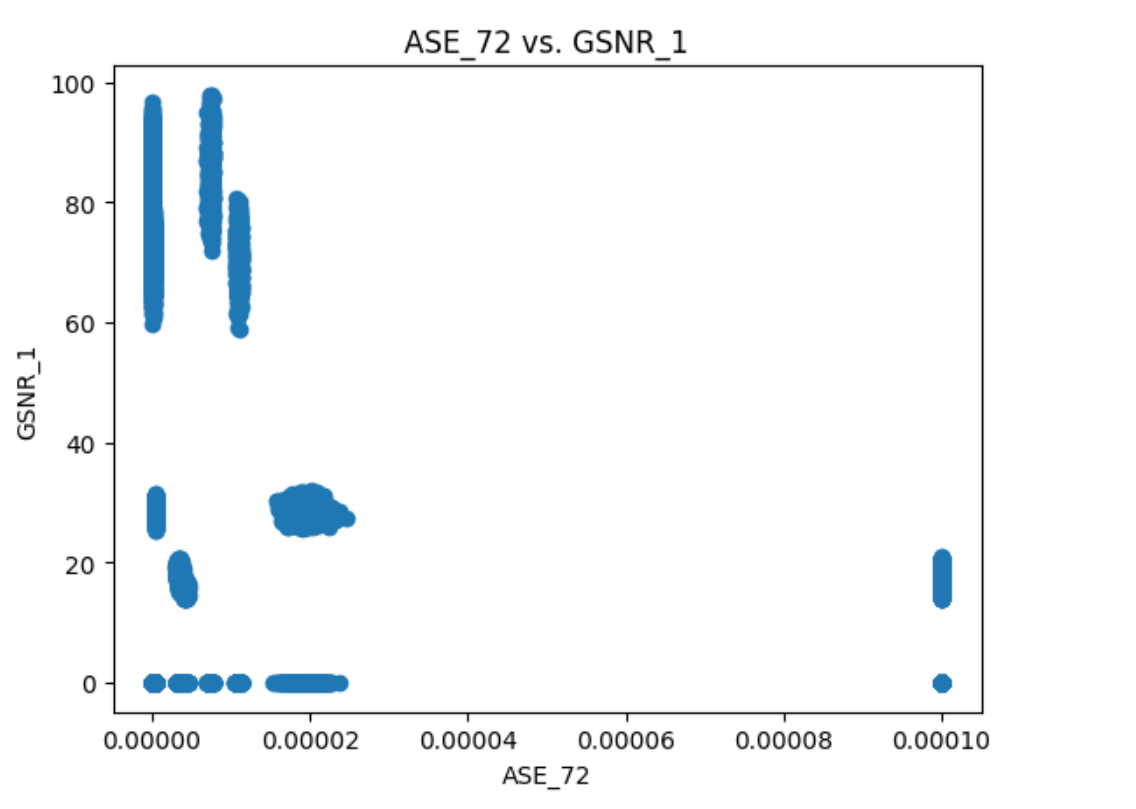
By analyzing the correlation matrix, we can identify features that are highly correlated (positive or negative) with the GSNR, potentially indicating their influence on predicting GSNR values. However, it's crucial to be cautious of multicollinearity (highly correlated features themselves), which can affect model performance.



**2. Visualizing Feature Distributions:**

* A loop iterates through each feature column name (feature) in the data (X).
* For each feature, a new figure is created using plt.figure() to ensure separate plots for each feature-target relationship.
* A scatter plot is generated using plt.scatter to visualize the distribution of data points for the current feature (X[feature]) against the target variable (y).
* The x-axis (plt.xlabel) and y-axis (plt.ylabel) are labeled with the corresponding feature name and GSNR column name (gsnr\_column), respectively.
* A descriptive title (f'{feature} vs. {gsnr\_column}') is set for each plot, clearly indicating the feature and its relationship with GSNR.
* Finally, plt.show() displays the generated scatter plot for the current feature.

By examining these scatter plots, we can visually assess the distribution of each feature and its potential relationship with the target GSNR. This can help identify features with linear or non-linear trends, outliers, or skewed distributions, which might require further pre-processing or transformations before model training.



## **Pre-Processing**

This section details the essential steps for pre-processing the GSNR data before training the ANN model:

**1. Handling Missing Values:**

The initial data exploration might reveal missing values in some features. Here are potential approaches:

* **Deletion:** If the percentage of missing values is low and the data distribution isn't heavily skewed, removing rows with missing values might be a simple solution.
* **Imputation:** For a higher percentage of missing values or if data preservation is crucial, imputation techniques can be applied. This involves filling missing entries with appropriate values. Techniques like mean/median imputation or more sophisticated methods like K-Nearest Neighbors (KNN) imputation can be considered.

The chosen approach depends on the extent of missing data, its distribution, and the impact on the model's performance.

**2. Data Description:**

After handling missing values, we'll perform data description to understand its characteristics:

* Use data.info() to get basic information about data types and potential null values.
* Utilize data.head() to view the first few rows and get a glimpse of the data's content.
* Employ data.describe() to obtain summary statistics for numerical features, including mean, standard deviation, minimum, and maximum values.
* Consider data visualization techniques like histograms or boxplots to explore the distribution of features and identify potential outliers.

This description provides valuable insights into the data and helps guide further pre-processing steps.

**3. Feature Scaling or Normalization:**

As features might have different scales, scaling or normalization is often recommended. This ensures all features contribute equally during training and prevents features with larger scales from dominating the learning process.

**Standardization:** Scales features to have a mean of 0 and a standard deviation of 1.

**4. Feature Selection (Refer to Previous Section):**

We'll revisit the feature selection process outlined in the previous section.

**5. Target Variable Transformation:**

If the target variable (GSNR\_1) exhibits a skewed distribution, a transformation like log transformation might be beneficial. This can improve model performance, especially for algorithms sensitive to skewed data.

**6. Data Splitting:**

Finally, we'll split the pre-processed data into training and validation sets using techniques like train-test-split from scikit-learn. This allows us to train the model on the training set and evaluate its performance on the unseen validation set, preventing overfitting.

## **Model Selection and Training**

This section explains the code for building the ANN model, performing hyperparameter tuning with Keras Tuner, and training the final model.

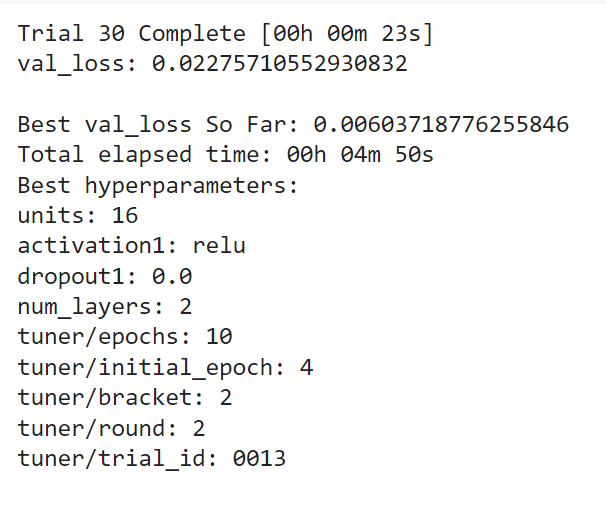
**1. Defining the Model Building Function (build\_model):**

This function takes a HyperBand object (hp) as input and constructs the ANN model:

* **Input Layer:** The first layer uses Dense with 64 units, ReLU activation, and an input shape specified by input\_shape. input\_shape is dynamically determined from the training data using X\_train\_scaled.shape[1]. This ensures the model can handle the specific number of features present in the data.
* **Hyperparameter Tuning with Keras Tuner:**
  + **Choice for Units and Activation:** The function uses hp.Choice to define search spaces for the number of units (8, 16, or 32) and activation function (ReLU or tanh) in hidden layers. This allows the tuner to explore different configurations.
  + **Dropout Rate:** A dropout layer with a rate between 0.0 and 0.5 (adjustable in steps of 0.1) is added using hp.Float. Dropout helps prevent overfitting.
  + **Number of Layers:** The hp.Int function defines a search space for the number of hidden layers (1 to 5). The model can have a varying number of layers based on the hyperparameter search.
* **Output Layer:** The final layer uses Dense with 1 unit and a linear activation for regression.
* **Compiling the Model:** The model is compiled with the Adam optimizer, mean squared error (MSE) loss function, and mean absolute error (MAE) metric.

**2. Hyperparameter Tuning with Keras Tuner (HyperBand):**

* A HyperBand tuner is created, specifying the build\_model function, val\_loss (validation loss) as the objective to minimize, and other parameters:
  + max\_epochs: Maximum number of epochs for hyperparameter trials (set to 10).
  + factor: HyperBand stopping factor (set to 3).
  + directory: Directory to store hyperparameter tuning results (set to 'my\_dir').
  + project\_name: Project name for hyperparameter tuning results (set to 'ONTTask2').



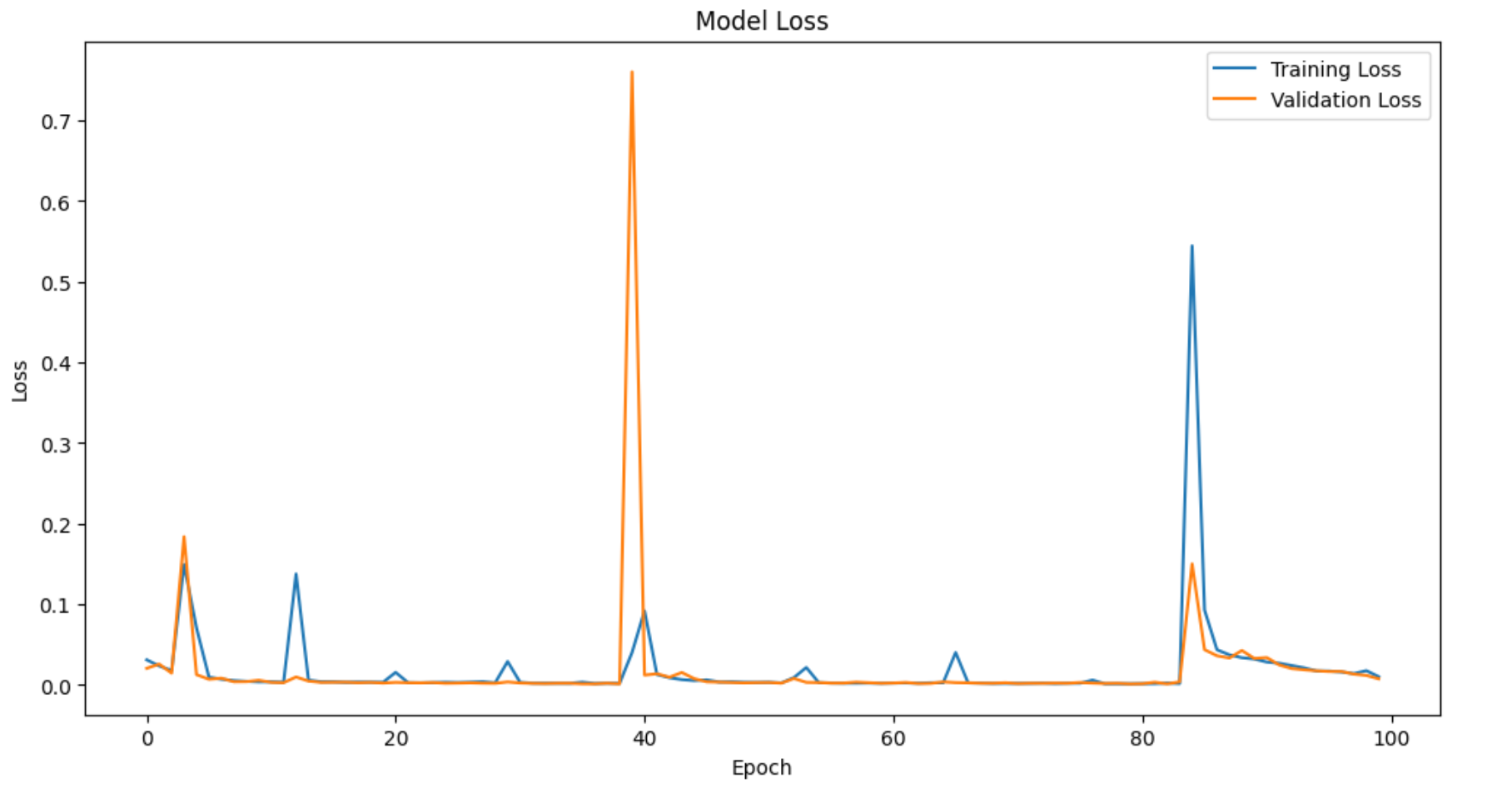
* The tuner.search method initiates the hyperparameter search process. It trains the model with different hyperparameter combinations using the training data (X\_train\_scaled, y\_train\_scaled) and monitors performance on the validation data (X\_val\_scaled, y\_val\_scaled) for 5 epochs.
* The tuner.get\_best\_models method retrieves the best model based on the defined objective (lowest validation loss).
* The tuner.get\_best\_hyperparameters method retrieves the hyperparameter configuration associated with the best model.
* The retrieved best hyperparameters are printed for analysis.

**3. Training the Best Model:**

* The best model is trained using the full training data (X\_train\_scaled, y\_train\_scaled) with validation data (X\_val\_scaled, y\_val\_scaled) for 100 epochs (assuming this was identified as the optimal number of epochs during hyperparameter tuning). A batch size of 32 is used for training efficiency.
* The training process is monitored with verbose mode set to 1.
* The training and validation loss curves are plotted for visualization.

**4. Evaluating the Best Model:**

* The final evaluation metrics (validation loss and MAE) are calculated using the best model on the validation set (X\_val\_scaled, y\_val\_scaled).
* Additionally, you might consider evaluating the model on a separate test set (not used for training or validation) to assess its generalizability to unseen data. The code snippet demonstrates evaluating on a potential X\_test\_scaled and y\_test\_scaled test set (assuming it's available).



## **Evaluation**

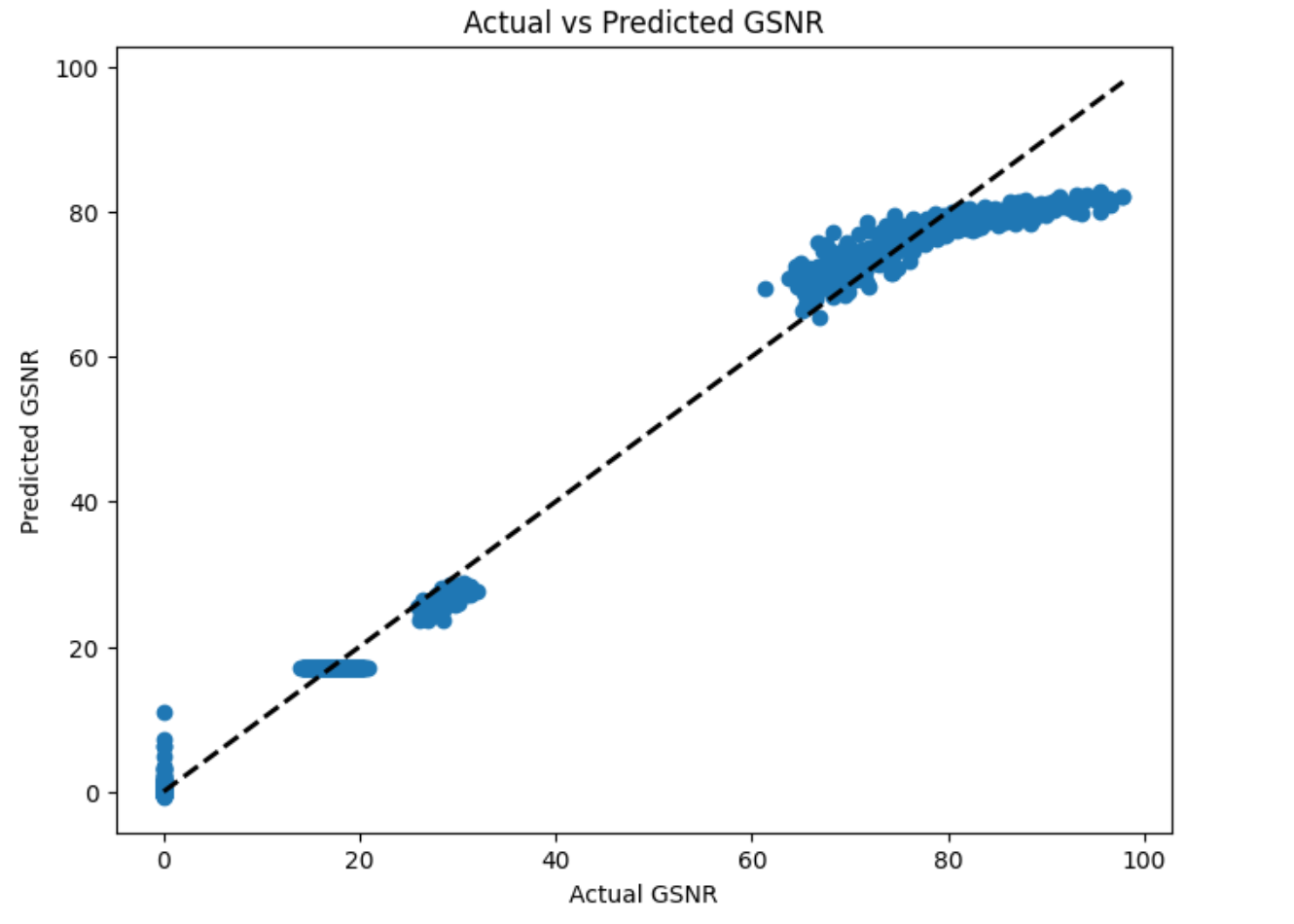
This section details the process of generating predictions on unseen data, visualizing model performance, and calculating key evaluation metrics for the final, hyperparameter-tuned ANN model.

**1. Generating Predictions:**

* The best\_model.predict method predicts GSNR values for the unseen test data (X\_test\_scaled). Since the model was trained on scaled data, the predictions (y\_pred\_scaled) are also in the scaled space.
* To obtain predictions in the original data scale, we perform the inverse transformation using the scaler (scaler\_y) on the reshaped predictions. The reshape(-1, 1) ensures the predictions are formatted correctly for the scaler. Finally, flatten() converts the potentially reshaped array back to a 1D array.

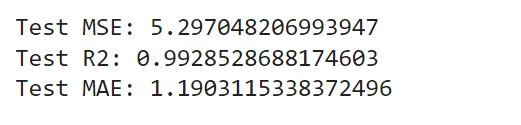
**2. Visualizing Model Performance:**

* A scatter plot is created to visualize the relationship between actual GSNR values (y\_test) and the model's predictions (y\_pred). This provides a visual assessment of how well the model captures the underlying trends.
* A diagonal line (k--) is plotted to represent a perfect prediction scenario where all points fall exactly on the line. The deviation of data points from this line indicates prediction errors.
* The plot is labeled with appropriate axis titles (Actual GSNR and Predicted GSNR) and a descriptive title (Actual vs Predicted GSNR).



**3. Calculating Evaluation Metrics:**

* Three key metrics are calculated to evaluate the model's performance on the test set:
  + **Mean Squared Error (MSE):** Measures the average squared difference between actual and predicted values. Lower MSE indicates better model performance.
  + **R-squared (R2):** Represents the proportion of variance in the actual GSNR data explained by the model's predictions. A value closer to 1 indicates a better fit.
  + **Mean Absolute Error (MAE):** Calculates the average absolute difference between actual and predicted values. Lower MAE signifies better model performance.



## **Conclusion**

The task of tuning and training a neural network to minimize validation loss and mean absolute error (MAE) was carried out using Keras Tuner for hyperparameter optimization, leading to significant improvements in model performance. The optimal hyperparameters included 16 units, a ReLU activation function in the first layer, no dropout, two layers, and training over ten epochs with specific settings for the initial epoch, bracket, and round. This configuration resulted in the best validation loss of 0.006037 and a final validation loss of 0.0020, with a validation MAE of 0.0200. The training process revealed a general trend of decreasing loss and MAE, despite occasional spikes indicating potential overfitting or challenging data segments. The model showed effective learning and good generalization to the validation set, achieving low validation loss and MAE in the final epochs. Key insights suggest that further improvements could be achieved by implementing regularization techniques, using learning rate schedules, enhancing the dataset with augmentation, and exploring a broader range of hyperparameters with advanced tuning methods. Overall, the neural network demonstrated effective learning and achieved optimal performance through systematic hyperparameter optimization and iterative training, underscoring the critical role of these practices in developing high-performing models.