**Transfer Learning for GSNR Prediction**

# **Understanding Transfer Learning**

## **Definition**

1. Transfer learning is a machine learning technique where a model developed for a task is reused as the starting point for a model on a second task. This is particularly useful when the second task has limited data.
2. Transfer learning is an approach to machine learning where a model trained on one task is used as the starting point for a model on a new task. This is done by transferring the knowledge that the first model has learned about the features of the data to the second model.

## **Benefits**

* **Reduced Training Time**: Since the model is pre-trained, it requires less time to converge.
* **Improved Performance**: Pre-trained models often achieve better performance on the target task.
* **Less Data Requirement**: Useful when the target task has a limited amount of data.

## **Common Techniques**

* **Feature Extraction**: Use the pre-trained model as a fixed feature extractor. Only the final layers are trained on the new task.
* **Fine-Tuning**: Fine-tune some or all layers of the pre-trained model along with the new layers for the new task.

## **Feature Extraction vs. Fine-Tuning**

### **Feature Extraction**

**Method**

* Freeze the weights of the initial layers of the pre-trained model.
* Replace the final classification layer(s) with new layers that are appropriate for the target task.
* Train only the new layers on the new dataset.

**Benefits**

* Faster training as most layers are frozen.
* Useful when the new dataset is small.

**Limitations**

* Less flexible as only the final layers are trained on the new data.
* Might not capture the new task's nuances well if the tasks are significantly different.

### **Fine-Tuning**

**Method**

* Unfreeze some or all of the pre-trained model layers.
* Train the entire model (or a large portion of it) on the new dataset.

**Benefits**

* More flexible and can adapt better to the new task.
* Can achieve higher performance if the new task is similar to the original task.

**Limitations**

* Longer training time.
* Higher risk of overfitting if the new dataset is small.

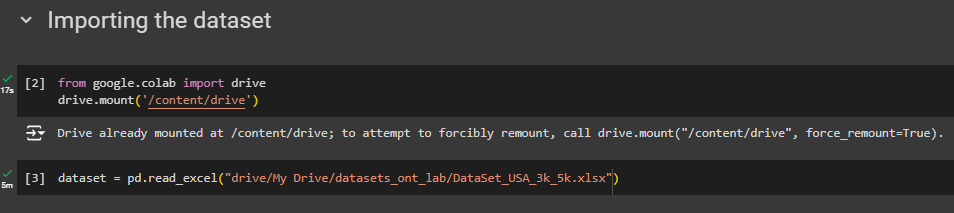
## **References**

1. <https://medium.com/@davidfagb/guide-to-transfer-learning-in-deep-learning-1f685db1fc94#:~:text=Transfer%20learning%20is%20an%20approach,data%20to%20the%20second%20model>.
2. <https://towardsdatascience.com/introduction-to-transfer-learning-c59f6f27e3e>

# **Data Preparation**

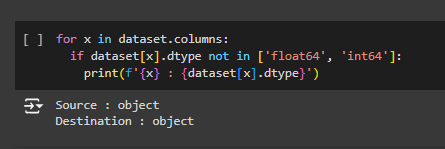
## **Importing the dataset**

First, we loaded the new DataSet\_USA\_3k\_5k.xlsx that we have stored in the Google drive for easy access and maintainability.



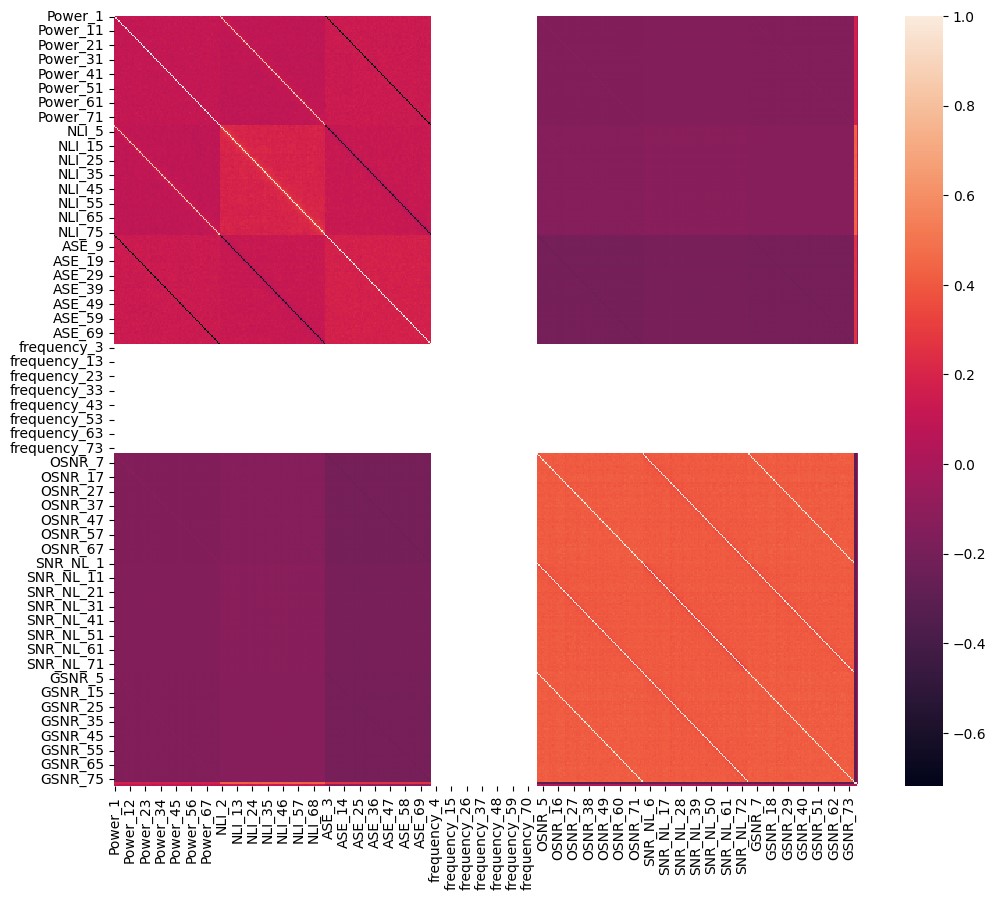
## **Looking at the first few rows**

Using head() function, we found that the last two columns are not numeric columns. Then, we look for the columns with other datatypes and found that there are two columns with object datatype that are Source and Destination.



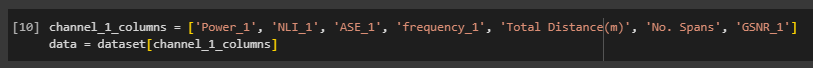
## **Correlation Heatmap**

For understanding the dataset better, we use correlation matrix other than the above two attributes. The heatmap suggest that the frequency attributes have almost zero correlation with other attributes present in the dataset and even with the target variables.

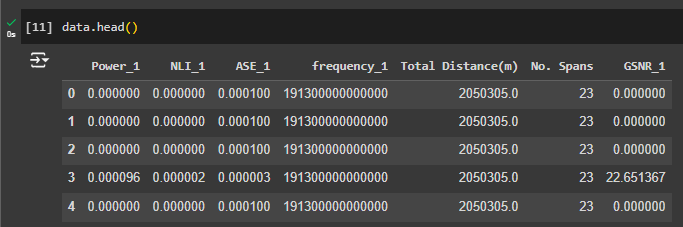


# **Data Preprocessing**

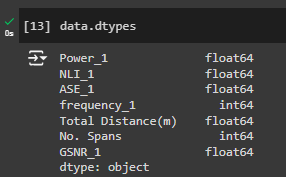
We will be using channel 1 (GSNR\_1) as our target variable. So, first we obtain all the subset of the data columns that are associated with the channel 1.



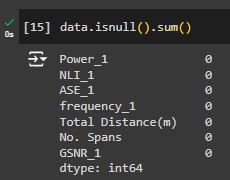
Then we look at the first few rows using head() function.



Then, we look at the datatypes of the columns.



After that we look for any missing values in the dataset.



# **Data Splitting**

To evaluate the performance of our predictive models effectively, we split the dataset into training and test sets. This approach helps us to understand how well the models generalize to unseen data. We used the train\_test\_split function from the scikit-learn library to split the data, with 80% of the data allocated to the training set and 10% each to the validation set and test set. The random state was set to 42 to ensure reproducibility of the results.

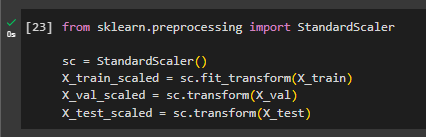
Here are the details of the split:

* **X\_train size**: (28800, 6) – This subset contains 28,800 samples with 6 features each, used to train the models.
* **y\_train size**: (28800,) – This subset contains 28,800 target values corresponding to the training samples.
* **X\_val size**: (3600, 6) – This subset contains 3600 samples with 6 features each, used to validate the models.
* **y\_val size**: (3600,) – This subset contains 3600 target values corresponding to the validation samples.
* **X\_test size**: (3600, 6) – This subset contains 3,600 samples with 6 features each, used to test the models.
* **y\_test size**: (3600,) – This subset contains 3,600 target values corresponding to the test samples.

# **Feature Scaling**

Feature scaling is a crucial preprocessing step in many machine learning workflows. It ensures that all features contribute equally to the model's performance by putting them on a similar scale. This is particularly important for algorithms sensitive to the magnitude of feature values, such as Support Vector Machines, k-Nearest Neighbors, and neural networks.

In this task, we used the StandardScaler from the scikit-learn library to perform feature scaling. The StandardScaler standardizes features by removing the mean and scaling to unit variance, which transforms the data to have a mean of 0 and a standard deviation of 1. By scaling the features, we enhance the performance of our machine learning models, ensuring they converge faster and perform better by treating all features equally, regardless of their original scale.



# **Applying Pre-trained Model**

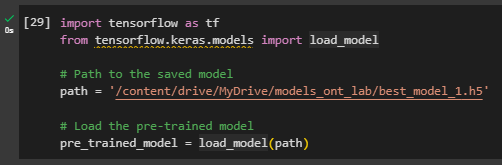
First, we load the pre-trained model that we have used in the task 2 for predicting the GSNR of European dataset. We recompile the model with Adam optimizer, Mean Squared Error (MSE) loss function, and Mean Absolute Error (MAE) as a metric. The summary of the model is that it has an input layer, three hidden layers with dropout layers in between, and an output layer. The total number of trainable parameters is 4129. The model is then trained on the scaled training data for 100 epochs with a batch size of 32. Validation is performed using the scaled validation data. At last the model is evaluated on the scaled test data with a **test loss of 83,898.148** and **test MAE of 59.712**.

# **Implementing Transfer Learning – Feature Extraction**

Since we have a pre-trained model on the European dataset that we have performed in the task 2, we can proceed with the feature extraction method of transfer learning.

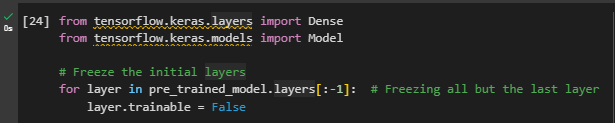
## **Loading the pre-trained model**

The pre-trained model is loaded from a specified path. This model has been previously trained and saved.



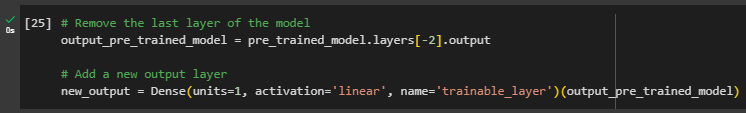
## **Freeze the initial layers**

All layers except the last one are frozen, meaning their weights will not be updated during training. This helps in preserving the learned features from the previous training phase.



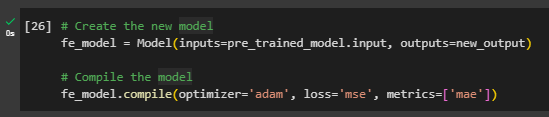
## **Modifying the Model**

The last layer of the pre-trained model is removed and replaced with a new output layer suited for the current task (regression, with a single unit and linear activation).



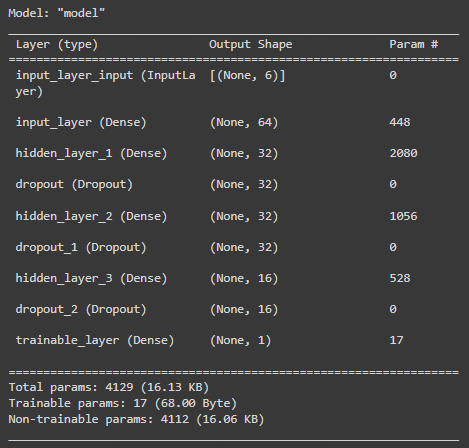
## **Compiling the new Model**

The new model is compiled with the Adam optimizer, Mean Squared Error (MSE) loss, and Mean Absolute Error (MAE) as a metric. The summary of the model shows the structure, including the number of trainable and non-trainable parameters.



## **Summary of the new Model**

The summary of the model shows the structure, including the number of trainable and non-trainable parameters.



## **Training and Evaluating the new Model**

The model is trained on the scaled training data for 100 epochs with a batch size of 32. Validation data is used to monitor performance during the training. The trained model is evaluated on the scaled test data with the resulting **test loss of 84066.969** and **test MAE of 59.708**.

# **Implementing Transfer Learning – Fine Tuning**

Similarly, like the previous step for feature extraction we load the pre-trained model. Then we determine the number of layers to freeze. The total number of layers in the model is calculated, and half of these layers are determined for freezing.

## **Freezing and Unfreezing Layers**

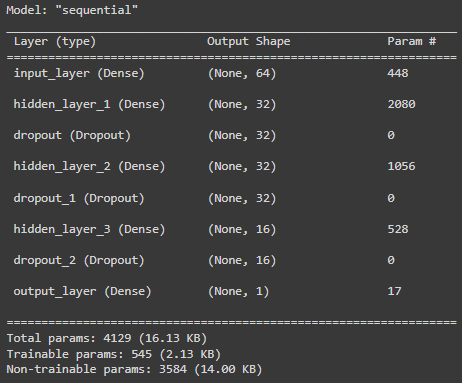
The first half of the layers in the model are set to non-trainable, meaning their weights will not be updated during training. The second half of the layers are set to trainable, allowing their weights to be updated during training.

## **Compiling the modified model**

The modified model is compiled with the Adam optimizer, Mean Squared Error (MSE) loss, and Mean Absolute Error (MAE) as a metric.

## **Summary of the modified model**

The summary of the model shows the structure, including the number of trainable and non-trainable parameters.



## **Training the modified model**

The model is trained on the scaled training data for 100 epochs with a batch size of 32. Validation data is used to monitor performance during training.

## **Evaluating the Model**

The trained model is evaluated on the scaled test data resulting **test loss of 78149.438** and **test MAE of 62.323**.

# **Conclusion**

The experiments conducted aimed to evaluate the performance of different training approaches using a pre-trained model for predicting the GSNR of the USA dataset, focusing on the following three methods: using the pre-trained model directly, transfer learning through feature extraction, and transfer learning through fine-tuning.

## **Using Pre-trained Model:**

The initial approach utilized a pre-trained model, which was recompiled and trained on the new dataset. The model, which consisted of an input layer, three hidden layers with dropout layers, and an output layer, had a **total of 4129 trainable parameters**. After training for 100 epochs with a batch size of 32, the model was evaluated on the test data, resulting in a **test loss of 83,898.148** and a **test MAE of 59.712**. These metrics serve as a baseline for comparing the effectiveness of transfer learning methods.

## **Using Transfer Learning - Feature Extraction Model:**

In this approach, transfer learning was implemented by freezing the initial layers of the pre-trained model and training a new output layer on the new dataset. This method aimed to leverage the learned features from the pre-trained model while adapting to the new data. After training for 100 epochs with a batch size of 32, the model achieved a **test loss of 84,066.969** and a **test MAE of 59.708**. While the test loss was slightly higher compared to the pre-trained model, the test MAE was marginally lower, indicating a comparable performance.

## **Using Transfer Learning - Fine Tuning Model:**

The final approach involved fine-tuning the pre-trained model by unfreezing half of its layers and allowing them to be trained on the new dataset. This method aimed to further adapt the model to the new data by updating a larger portion of its weights. The evaluation on the test data yielded a **test loss of 78,149.438** and a **test MAE of 62.323**. The fine-tuning approach resulted in a lower test loss compared to the other methods, indicating an improvement in the overall error reduction. However, the test MAE was higher, suggesting that while the model had lower overall error, its absolute deviations were greater.

# **Summary**

In summary, the experiments demonstrated that transfer learning methods can effectively leverage a pre-trained model to adapt to a new dataset. The feature extraction approach provided a performance comparable to the pre-trained model, while the fine-tuning method showed a reduction in test loss but an increase in test MAE. These results highlight the potential of fine-tuning for improving model performance but also indicate the need for further optimization to balance the trade-offs in different performance metrics.