Introduction:

The aim of the project is to detect the production plant shutdown using Transformer Model which is trained using Historical production plant data which has 5 different folders of train set data and test set data. The transformer model is superior in quality while being more parallelizable and requiring significantly less time to train. The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution. This Transformer model has been applied to different data by changing the Time window and resampling period of original DSM data which is taken from the plant between the end of 2017 and September 2020.

Model Architecture:

The encoder-decoder structure is found in the majority of competitive neural sequence transduction models [5, 2, 35]. The encoder converts a sequence of symbol representations (x1,..., xn) into a sequence of continuous representations (z =). (z1, ..., zn). The decoder then outputs a symbol output sequence (y1,..., ym) one element at a time, given z. The model is auto regressive at each step [10], using previously generated symbols as extra input to generate the next. For both the encoder and decoder, the Transformer uses layered self-attention and pointwise, completely linked layers, as seen in the left and right half of Figure 1, respectively. Here, the encoder and decoder haven’t played a big role in DSM work. However, there are 2 different multi-Head attention layers specified with 12 different blocks. The first attention layers return of x and y. but, the 2nd attention layer doesn’t return y and only returns x, because it doesn’t return attention score.

Diagram

Description automatically generated

Figure 1: The Transformer - model architecture.

Analysis of the Data:

The Transformer model was applied to the Time window of 6 hours and resample period of 10 minutes initially which is the default given by DSM team. However, we have planned to cross verify the performance in the moto of increasing the accuracy and to compare with other time windows to choose a best time window to apply to model. So, we have implemented the Transformer model in 5-hour 12 minutes data, 7-hour 10 minutes data, 12-hour 10 minutes data. All the trained models have been evaluated by combining the models obtained from folder 0 to folder 4 which is K fold cross validation. From this analysis, we have confirmed that, the time window 6 hours and 10 minutes resampling period perform well and have the most True positive predictions.

Parameter Tuning:

The default configuration of the Transformer model has been tested with the 6 Hours 10 minutes data, but it produced poor result, so we have planned to tune the parameters of the Transformer model. The original configuration of the model is shown in the Figure 2.

Text

Description automatically generated

Figure: 2

Result Obtained with the original model configuration is shown in the Figure 3. The figure illustrates the training loss 17%, training accuracy 95%, validation loss 0.41% and validation accuracy 90%.

 Figure: 3

Then, the configuration of the model has been tested with different combination of values such as,

* head\_size: 64, 128, 256
* ff\_dim: 4, 8, 16, 32
* transformer\_blocks: 2, 3, 4, 5, 6, 7, 8

By testing these combination, we have found the best combination which gives the validation loss of 33% and validation accuracy of 91%.

* Head\_size = 128,
* Num\_heads = 4,
* Ff\_dim = 4,
* num\_transformer\_blocks = 6,
* Mlp\_units = [128],
* Mlp\_dropouts = 0.4,
* Dropout = 0.25

Still, we have planned to enhance the accuracy of the model by tuning the channels.

Testing the Channels:

what is Channel\_first and Channel\_last?

When represented as [three-dimensional arrays](https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/), the channel dimension for the data is last by default, but may be moved to be the first dimension, often for performance-tuning reasons.

* **Channels Last**. Image data is represented in a three-dimensional array where the last channel represents the color channels, e.g. *[rows][cols][channels]*.
* **Channels First**. Image data is represented in a three-dimensional array where the first channel represents the color channels, e.g. *[channels][rows][cols]*.

But the default channel of the Transformer is “Channel\_first”. We have obtained the result (Figure 4) with that channel only. So, we thought that, by changing the channel, we can get some different results. Then we have implemented a model for all the 5 folders with “Channel\_last” and evaluated it by combining all the models and visualized the result with normalized confusion matrix. Here, the prediction of the True Positive in Channel\_last is increased. Unfortunately, the prediction of True Negative is decreased. Thus, it is a poor result (Figure 5) while compared with “Channel\_first” prediction. Both the results have been presented below.

Chart, treemap chart

Description automatically generated Chart, treemap chart

Description automatically generated

Figure 4: Channel\_first Figure 5: Channel\_last

Combining the Channels:

Then, we have planned to change the channels of the pooling layer and combine the layers of Channel\_first and Channel\_last together and pass it to the Dense layer subsequently. Furthermore, the model has been trained with the same configuration as mentioned above with the combination of Channel\_first and Channel\_last. Here the layers have been merged.

This method produced slightly better results while compared with all the evaluated models. The normalized confusion matrix figure is presented below (Figure 6).

Chart, treemap chart

Description automatically generated

Figure: 6, Combined channels

Dropout layer Reduction:

While compared with the Figure 5 and Figure 6, the prediction of True Positive and True Negative has raised but this method doesn’t make a big change. However, we are really interested in brining more correct predictions of True Negative. But the prediction of True Negative is quite lower. This happens because of the imbalance in the data. The term imbalance denotes that, we have a greater number of samples in the class 1, but the class 2 samples are too low. To be more specific, we have 1616 samples of class 1 and 231 samples of class 2. Here, the class 1 has 8x times more samples than class 2. From this, we can understand that we need a greater number of samples to overcome this issue. So, we have tried reducing the dropout layers in the code, but it doesn’t make a huge difference because the dropout layers also drop the class 1, and we have a greater number of samples in class 1 and low number of samples in class 2. So, it brought poor result (Figure 7) once again while compared with Figure 6.

Chart, treemap chart

Description automatically generated

Figure: 7, reduced dropout layer

From the Figure 7, we can get to know that, even after reducing the dropout layer, we haven’t got much better results. it actually reduced the True Positive prediction rate as well. so, having dropout is a better way to enhance the accuracy.

Dropout comparison:

Table

Description automatically generated

Figure: 8, Comparison Chart

Return attention scores:

We are interested in getting attention scores for the transformer block 1 and 6. so except the block 1 and 6 we have left the attention parameter as default and for 1 and 6, it is switched to TRUE. Additionally, the attention score requires x and y in the transformer\_encoder block. so the 1st part of the encoder part only works for the transformer block 1 and 6, and for other blocks such as 2, 3, 4, 5 the else part will work which has only the x argument.

Chart, treemap chart

Description automatically generated

Figure: 9, Return attention score with Additional Dropout layer.

References:

* Ashish Vaswani., Noam Shazeer., Niki Parmar., Jakob Uszkoreit., Llion Jones., Aidan N. Gomez., Lukasz Kaiser., Illia Polosukhin, 2017. Attention Is All You Need. *Computation and Language.* Arxiv [Online]. Available from: <https://arxiv.org/abs/1706.03762>
* Theodoros Ntakouris, 2021. Timeseries classification with a Transformer model. Available from: <https://keras.io/examples/timeseries/timeseries_classification_transformer/>