**EEC4400 Assignment (AY2023/24)**

**Anomaly Detection using Deep Learning**

**Group: 05**

**Task division:**

CNN model—Shen Yixuan

LSTM model—Gao Hao

Hybrid CNN-LSTM model—Zhang Yichi / Peng Ruoyun

**1.** Network Architecture Design

**The reasoning behind the design of the network architectures.**

1) CNN model

In the structure of this CNN model, we have used two 1D convolutional layers because we are dealing with 1D data of a time series and 1D convolutions are more suitable for processing time series data.

This CNN architecture begins with an input layer having a shape specified by input_shape. Subsequently, it captures local patterns in the input data through two convolutional layers, employing the Rectified Linear Unit (ReLU) activation function to introduce non-linearity, facilitating the learning of intricate representations. Dropout layers with a dropout rate of 0.1 are added after each convolutional layer to mitigate overfitting. The spatial dimensions of the data are then reduced, and the model's parameter count is lowered through a MaxPooling layer. The Flatten layer transforms the 2D output from the previous layer into a 1D vector, preparing it for the subsequent fully connected layers. A fully connected layer with 128 units and ReLU activation learns global patterns and dependencies in the data, with an additional Dropout layer following to further regularize the model. The final output layer is a dense layer with a Sigmoid activation function, suitable for binary classification tasks, where the output between 0 and 1 represents the probability of the input belonging to the positive class. The model is compiled using binary crossentropy loss, the Adam optimizer, and accuracy as the evaluation metric.

2) LSTM model

Our model has set 4 LSTM layers in order to guarantee the balance between precision and training time. And after each LSTM layer we also add a dropout layer to prevent overfitting and regularize the model.

3) Hybrid CNN-LSTM model

**Firstly**, we use Conv1D as our CNN layer.

Dimension is actually the representation of information. The use of CNN here is to extract information from the four observed features and get the information that truly determines whether the machine runs normally. And discover the time-series relationship inside the one-dimension sequence. So why not use Conv2D or even higher dimensional convolution layers? In fact, dimension reduction can be achieved by setting stride and pool_size: 1D data is reconstructed to high-dimensional data, where the convolution kernel of (1,n) is used, and reasonable strides are set to achieve the same information extraction of time series. This is equivalent to degenerating to 1D convolution, but consumes more computing power. If the multi-dimensional convolution does not degenerate into a 1D convolution operation, the weights of Con1D will only share weights in one dimension - the time dimension, but the multi-dimensional convolution will share weights in multiple dimensions, which may make the model too complex and have the risk of overfitting.

**Secondly**, we introduce TimeDistributed layer and Flatten layer which helps us to go from Conv1D to LSTM smoothly.

1. CNN only convolves the whole input sequence, while LSTM learns in the time dimension, so CNN is packaged in the TimeDistributed layer so that we can get more exact information in the time dimension.
2. The output of CNN is determined according to the input of the previous layer, and each time_step is ((feature-kernel_size)/strides+1, neuron), while LSTM takes one-dimensional data. Therefore, it is necessary to Flatten the output of CNN to one dimension using the Flatten() layer.

**Thirdly**, we use LSTM to exploit information from long and short time.

Long and short term neural networks are introduced, because CNN emphasizes the extraction of local features, while anomaly detection is considered in a long-term time series, so the model that can store long-term information - LSTM can be introduced.

**At last**, reasoning behind the settings of hyperparameters in our hybrid CNN-LSTM model.

1. The process of determining two-layer CNN and one-layer LSTM:
   Firstly, I tried the structure of models in [1], but found that the performance became worse when the number of CNN increased. This could be attributed to the model becoming overly complex. So I tried to reduce the number of CNN and LSTM. Ultimately, the highest accuracy was achieved with 2 CNN layers and 1 LSTM layer.
2. For the layer's kernel size and number of neurons:
   I initially experimented with an arbitrary size but found that the performance was not satisfactory. Therefore, I considered whether there were related tasks in time series analysis. Based on the hyperparameters in [1], [2], I iteratively adjusted the kernel size and neuron to improve the accuracy of the prediction results.
3. For threshold determination:
   I did not opt for maximizing precision or recall individually. Instead, I aimed to strike a balance between the two, specifically by maximizing the F1-score. Thus, I selected the threshold 'th' that maximizes the F1-score.

And model.summary() is as follows(when seq_len=30):

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_2 (InputLayer) | [(None, 30, 4, 1)] | 0 |
| time_distributed (TimeDistri | (None, 30, 3, 64) | 192 |
| time_distributed_1 (TimeDist | (None, 30, 2, 64) | 0 |
| dropout_7 (Dropout) | (None, 30, 2, 64) | 0 |
| time_distributed_2 (TimeDist | (None, 30, 1, 128) | 16512 |
| time_distributed_3 (TimeDist | (None, 30, 128) | 0 |
| dropout_8 (Dropout) | (None, 30, 128) | 0 |
| lstm_4 (LSTM) | (None, 100) | 91600 |
| dropout_9 (Dropout) | (None, 100) | 0 |
| dense_3 (Dense) | (None, 1) | 101 |

Total params: 108,405
Trainable params: 108,405
Non-trainable params: 0

For time_distributed_4 layer, it's a TimeDistributed Conv1D layer with kernel_size=2. Param#=kernel_size* units#+units#=2* 64* 1+64=192. For time_distributed_6 layer, it's similar with the previous, Param#= kernel_size* units# *pre_units#+units#=2* 128* 64+128 = 16521

For lstm_1 layer with 100 neurons, Param# = 4* ((pre_units#+units#)* units#+units#)= 4* ((128+100)* 100+100)=91600. For dense layer, Param# = pre_units#* units#+units# = 100*1 +1=101

**Performance of the network architectures. (Including performance metrics and TensorBoard visualizations of training data and test data)**

1) CNN model

Based on performance metrics, our model has demonstrated excellent performance on the test data. Taking seq = 30 as an example, here is an analysis for each metric:

Accuracy: The accuracy on the test data is 99.94%, indicating exceptionally high overall predictive performance of the model.

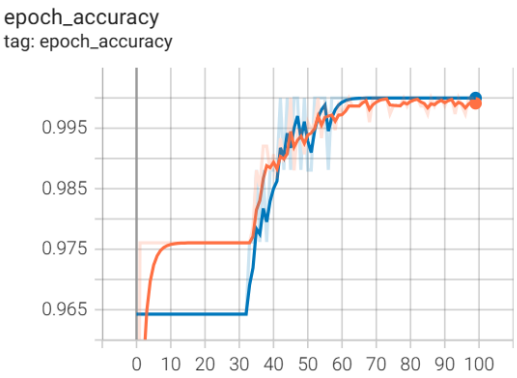
Confusion Matrix: The confusion matrix reveals that out of 1688 instances of the negative class, the model correctly predicted all instances. For the 33 instances of the positive class, the model accurately predicted 32 of them. There is one instance of the negative class misclassified as positive.

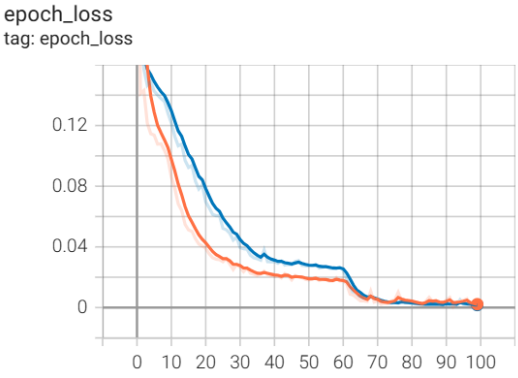
Precision: With a precision of 0.9994, it suggests that when the model predicts the positive class, it is correct 99.94% of the time.

Recall: A recall of 0.9994 indicates that the model is able to correctly identify almost all true positive instances.

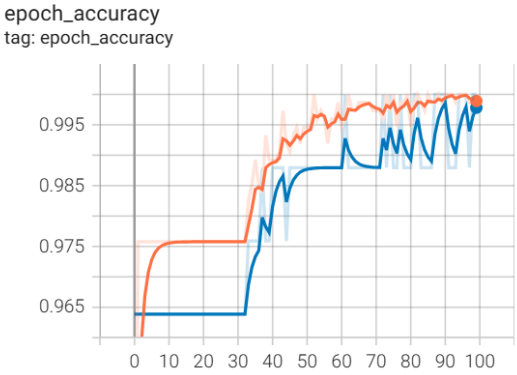
F1 Score: The F1 score, at 0.9994, signifies that the model has achieved a balance between precision and recall.

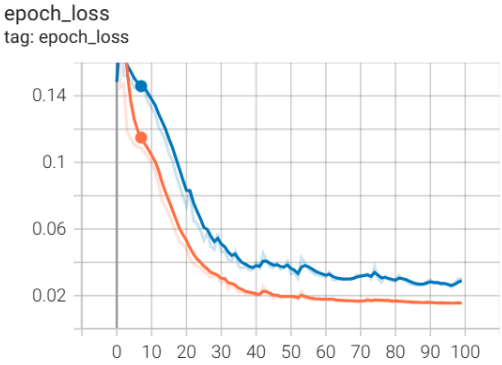
- **seq_len =30**




- **seq_len =40**




2) LSTM model

Based on the performance metrics, our model has demonstrated excellent performance on the test data. Here's an analysis of each metric:

Accuracy: The accuracy on the test data is 99.94%, indicating that the model's overall predictive performance is extremely high.

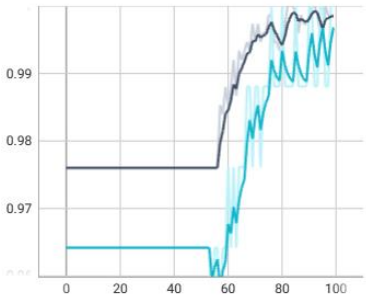
Confusion Matrix: The confusion matrix shows that out of 1688 instances of the negative class, the model correctly predicted 1687 of them, and out of 33 instances of the positive class, the model correctly predicted all of them. There was one instance of the negative class incorrectly predicted as the positive class.

Precision: The precision of 0.9706 suggests that when the model predicts a positive class, it is correct 97.06% of the time.

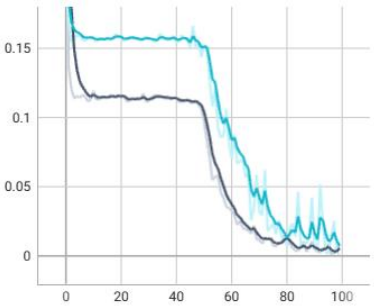
Recall: The recall of 1.0 indicates that the model is able to correctly identify all true positive instances.

F1-score: The F1-score of 0.9851 suggests that the model achieves a balance between precision and recall.

epoch_accuracy:



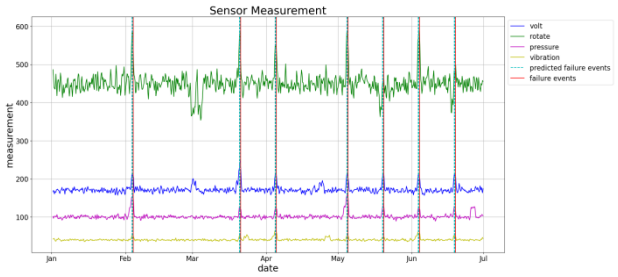

epoch_loss:




3) Hybrid CNN-LSTM model

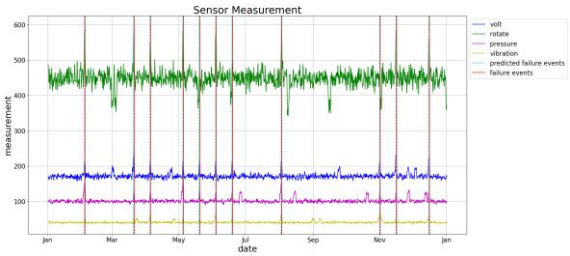
i. seq_len=30

Training data Prediction:



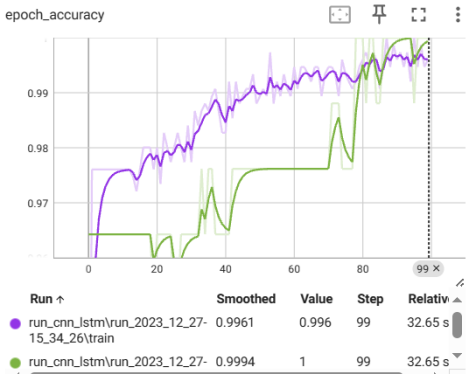

```
Accuracy:  1.0
Confusion Matrix:
 [[805   0]
 [  0  21]]
Precision :  1.0
 Recall :  1.0
 F-score :  1.0
```

Test data Prediction:

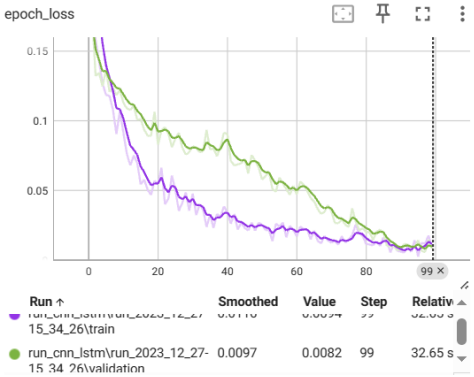



```
Accuracy:  1.0
Confusion Matrix:
 [[1678    0]
 [   0   33]]
Precision :  1.0
 Recall :  1.0
 F-score :  1.0
```

epoch_accuracy:



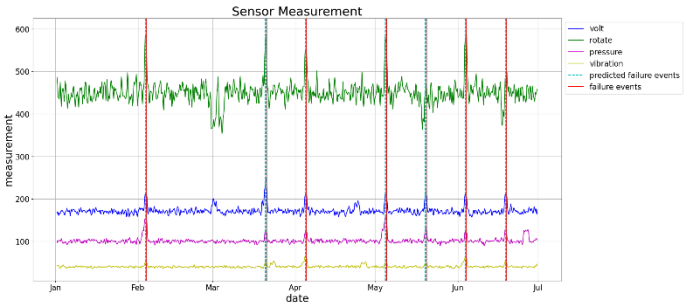

epoch_loss:




purple is training test and green is validation test

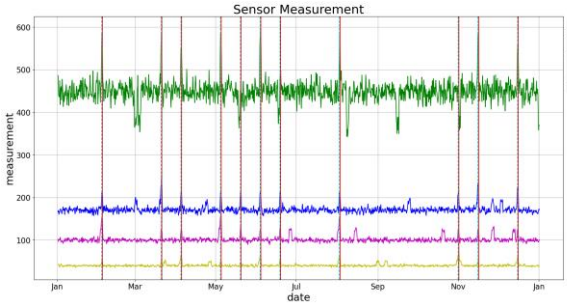
ii. seq_len=40

Training data Prediction:



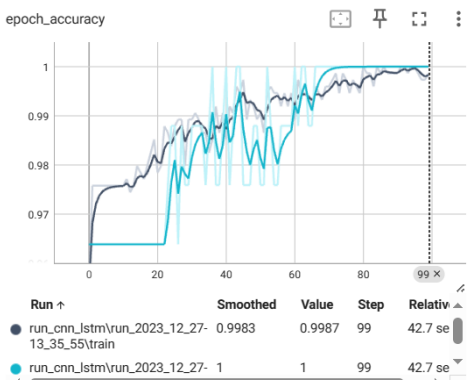

```
Accuracy:  1.0
Confusion Matrix:
 [[805   0]
 [  0  21]]
Precision :  1.0
 Recall :  1.0
 F-score :  1.0
```

Test data Prediction:

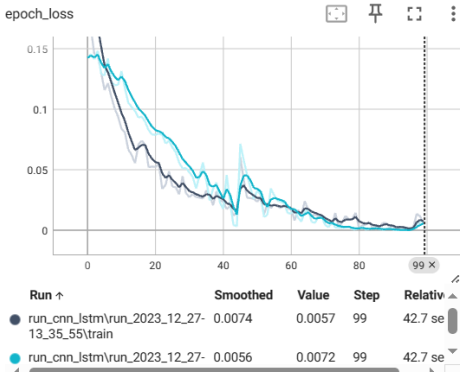



```
Accuracy:  0.9976621858562245
Confusion Matrix:
 [[1674    4]
 [   0   33]]
Precision :  0.8918918918918919
 Recall :  1.0
 F-score :  0.9428571428571428
```

epoch_accuracy:




epoch_loss:

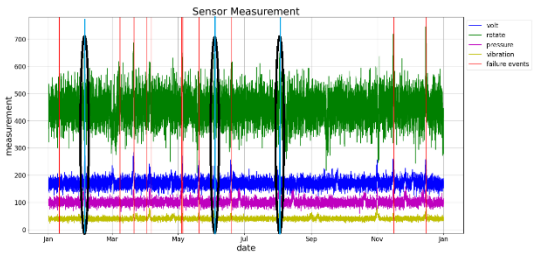


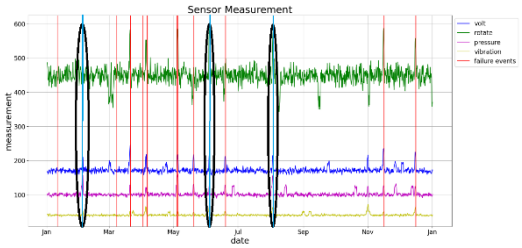

black is training test and blue is validation test

**2.** Answers to the questions in Section 7.1.

**7.1.1** We suppose the most thing is to improve accuracy. Noise in the raw data can lead to **inaccurate and unreliable results**. By removing noise, the overall accuracy and reliability of the data can be improved, leading to more accurate analysis and predictions. **What's more**, noise can interfere with the learning process of machine learning models, leading to **overfitting or underfitting**. By removing noise, the model can focus on the relevant patterns and relationships in the data, thus improving its performance. Finally, noise data can **hinder the generalization ability of models**, making it a challenge to apply the model to new or unseen data. And removing noise in the pre-processing stage can help the model generalize better to new data and make more accurate predictions.

We can spot that when **volt and rotate went high, failures occured**(may be there are some inner relations between those parameters). And when can mark some spots based on the original sensor measurements plot as shown below(left). From the sliding window averaged data plot we can get the similar law and also the spots are marked below(right):




Blue lines are potential failures we have spotted.

**7.1.2** Based on the data calculated out by the function, we can see that training accuracy and validation accuracy are quite high. There are some reasons. **First, overfitting.** The model may have overfit the training data, meaning it has learned to perform well on the specific examples in the training set but does not generalize well to unseen data in the validation set. But the test accuracy is also high and acceptable so it doesn't overfit. **Second, data imbalance.** If the training data and validation data have a similar distribution, the model may perform well on both, leading to high accuracy scores for both sets. **Third, model complexity.** The LSTM model is complex enough to capture the underlying patterns in the training data and generalize well to the validation data, resulting in high accuracy which means our models are good enough. **Fourth, proper hyperparameter tuning**. It means that we have chosen and set up optimal hyperparameters to train model and get high accuracy both on training and validation datasets.

Not really. As stated above, we may get overfitted model and when we change the datasets, the generalization gets worse.

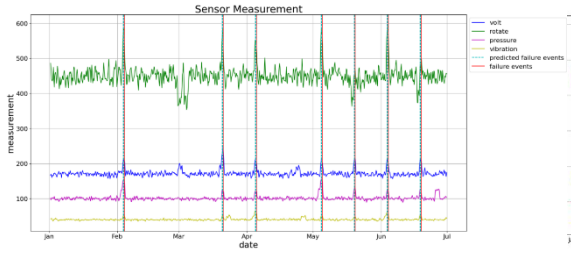
We think **precision** and **recall metric** are more important. The reason **precision** is often prioritized in anomaly detection is that incorrectly flagging normal instances as anomalies (false positives) can lead to unnecessary alerts or actions, which could be costly or disruptive in real-world applications. Therefore, minimizing false positives is often a key concern in anomaly detection. **The recall metric** (the proportion of true positives out of all actual positives) is also important in anomaly detection, as it measures the model's ability to identify all true anomalies. However, in many

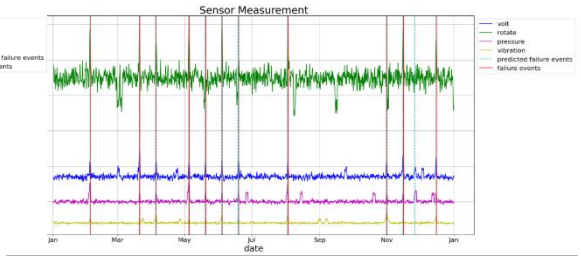
anomaly detection scenarios, precision is given greater emphasis due to the potential impact of false positives. (We give this answer most based on the background.)

### 7.1.3 Model Comparison (CNN)

- **seq_len =30**

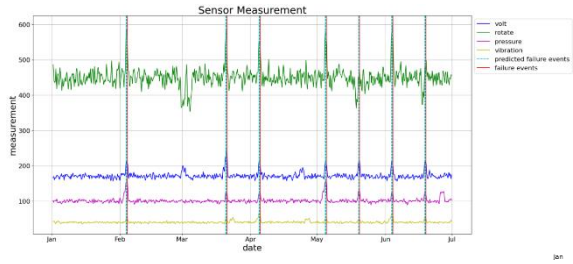
train_predicted: test_predicted:

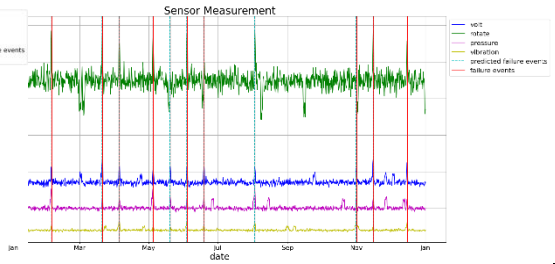



- **seq_len =40**

train_predicted: test_predicted:




| CNN_Model | State | Accuracy | Confusion | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| **seq_len=30** | train_pred | 1.0 | [[815 0] , [ 0 21]] | 1.0 | 1.0 | 1.0 |
| | test_pred | 0.999418 | [[1688 0] , [ 1 32]] | 0.999419 | 0.999418 | 0.999414 |
| **seq_len=40** | train_pred | 1.0 | [[805 0] , [ 0 21]] | 1.0 | 1.0 | 1.0 |
| | test_pred | 0.997662 | [[1678 0] , [ 4 29]] | 0.997667 | 0.997662 | 0.997588 |

We can see that the performance of the CNN model when predicting after observing a longer past sequence pattern has a lower performance than the shorter past sequence pattern.

The slightly lower performance of the model at sequence length 40 compared to sequence length 30 may have several reasons:

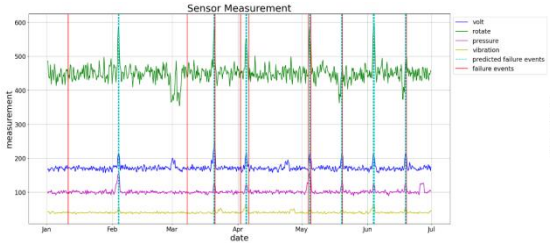
Increased complexity: Longer sequences contain more information and may be more complex to model and analyze accurately. The model may struggle to capture all the relevant patterns and dependencies within the longer sequences, leading to a slight decrease in performance.

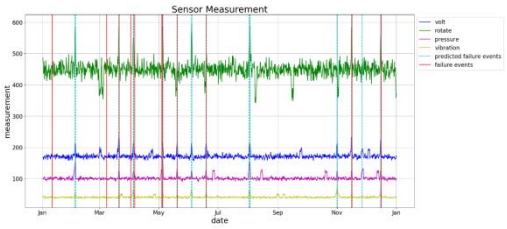
Overfitting: With longer sequences, the model may have a higher chance of overfitting the training data, especially if the dataset is limited. Overfitting occurs when the model becomes too specialized in the training data and fails to generalize well to unseen data. This can result in lower performance on the test or validation data.

### 7.1.4 Model Comparison (LSTM)

- **seq_len =30**

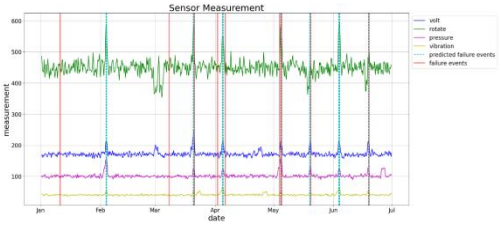
train_predicted: test_predicted:



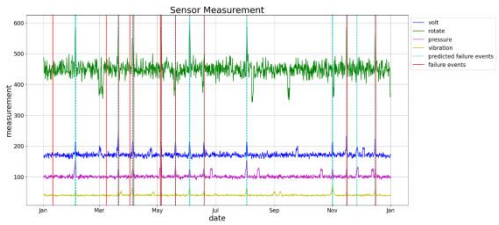




· **seq_len =40**

train_predicted:




test_predicted:




| Length | State | Accuracy | Confusion | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| **seq_len=30** | train_pred | 1.0 | [[815 0] , [ 0 21]] | 1.0 | 1.0 | 1.0 |
| | test_pred | 0.998 | [[1684 4] , [ 0 33]] | 0.892 | 1.0 | 0.943 |
| **seq_len=40** | train_pred | 0.999 | [[805 0] , [ 1 20]] | 1.0 | 0.952 | 0.976 |
| | test_pred | 0.997 | [[1676 2] , [ 3 30]] | 0.9375 | 0.909 | 0.923 |

As we can see the table above, when moving from a sequence length of 30 to 40 in the LSTM model for anomaly detection, the increase in the past window can affect the performance metrics such as **precision, recall, and F-score**. The **higher precision** but **lower recall and F-score** can be attributed to the **longer sequence length providing more context and historical information** for the model to learn from. This can lead to the model being better at correctly identifying true anomalies (hence the **higher precision**), as it has more historical information to base its predictions on. However, the longer sequence length can also lead to a **lower recall and F-score**, as the model may become more **conservative** in predicting anomalies. With a longer sequence, the model may **become more sensitive to small fluctuations and variations** in the data, leading to being less effective at capturing all true anomalies (**leading to a lower recall**). The lower F-score is a result of the trade-off between precision and recall. The increase in precision and decrease in recall create a lower F-score as it indicates the model is now biased towards correctly identifying true anomalies while potentially missing some of them.

**7.1.5 Model Comparison(Hybrid CNN-LSTM)**

When th=0.5, the performance of seq_len=30 is perfect(confusion matrix is all right), but seq_len=40 is bad. For training data and test data with seq_len=40, the performance metrics are as follows(right is for training, and left is for test):

```
Accuracy:  0.9782082324455206
Confusion Matrix:
 [[801   4]
 [ 14   7]]
Precision :  0.6363636363636364
 Recall :  0.3333333333333333
 F-score :  0.43749999999999994
```

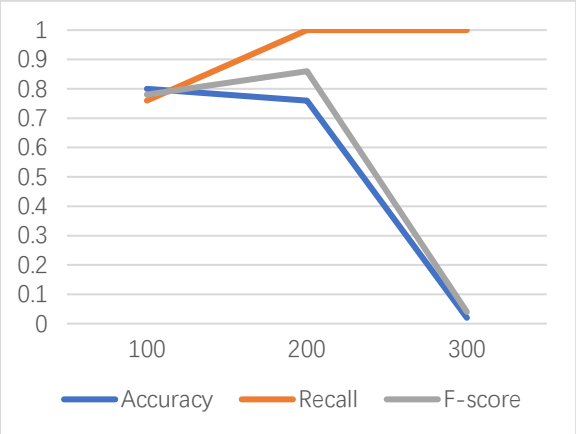
```
Accuracy:  0.983635300993571
Confusion Matrix:
 [[1673    5]
 [  23   10]]
Precision :  0.6666666666666666
 Recall :  0.30303030303030304
 F-score :  0.41666666666666663
```

But after we choosed the best th for the hybrid model, the performance has been imporoved(as the previous pictures of part 1). We think there are some reasons for worse performance of a longer past sequence pattern: **1). Longer Temporal Dependency:** As the pdf said, abnormal sensor measurements occurs 24 hours before each failure are labelled with '1', with seq_len=40, the model is provided with more historical information. The longer temporal

dependency might introduce more noise and irrelevant information into the model, which could potentially outweight the useful signal in the data. **2) Vanishing or Exploding Gradients:** Longer sequence lengths can increase the difficulty of training the model due to the vanishing or exploding gradient problem. And because we include the CNN and LSTM layers, but there's no batch normalization or layer normalization, so there potentially exits gradients vanishing or exploding, that makes the performance worse.

More comments on this problem:

There are two hypotheses on the instability, one is model underfitting and another is the current model is not optimal. For the first one: We increase the epoch(100 ,200, 300) and get a result on the test set to observe whether the model is underfitting. The results are as follows:




We found that we can not get the same great performance even if we increase the epoch. It can be inferred that the issue is not related to underfitting of the model.

For the second one: After making the network more complex or simpler, I observed that the performance of the model deteriorated. This indicates that the current model is the optimal network for predicting this problem, suggesting that the issue may not lie with the model itself.

So, seq_len=40 is not the best choice for this problem.

### 7.1.6 Effect Facts

| | accuracy | precision | recall | F-score |
|---|---|---|---|---|
| CNN | 0.999418942475305 | 0.9994360324025019 | 0.999418942475305 | 0.999423192642893 |
| LSTM | 0.9988378849506101 | 0.9428571428571428 | 1.0 | 0.9705882352941176 |
| Hybrid CNN-LSTM | 1.0 | 1.0 | 1.0 | 1.0 |

As the result shown above wo can see several facts:

**LSTM:**

Effect: LSTMs are well-suited for processing and making predictions on time sequences of data. LSTM can effectively capture temporal dependencies and long-term dependencies in the input sequences due to their memory cell structure.

Advantages: LSTMs can handle input sequences of varying lengths, handle time-series data with long-term dependencies. This makes them suitable for detecting anomalies in sequential data with complex patterns.

**CNN:**

Effect: CNN can be adapted to process 1D sequential data and can learn local dependencies within the input data.

Advantages: CNNs are effective at automatically learning hierarchical representations of input data, capturing features at different levels of abstraction. In our task, CNNs can identify local patterns and anomalies within the sequential data.

**Hybrid CNN-LSTM Model:**

Effect: The combination of CNN and LSTM in a hybrid model leverages the strengths of both architectures. CNN can be used for feature extraction and capturing local patterns, while LSTM can process the features extracted by the CNN and capture long-term dependencies in the sequential data.

Advantages: The hybrid model can benefit from the local feature learning capabilities of CNN while also capturing temporal dependencies and context using LSTM, resulting in a more comprehensive representation of the sequential data and potentially improved anomaly detection performance.

### 7.1.7 Chaning sliding_window_width and sliding_step_size (Taking LSTM as an example)

| Length | Window_width=10 | Accuracy | Confusion | Precision | Recall | F-score |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| **step_size=1** | train_pred | 0.999 | [[4162 4] , [ 0 133]] | 0.971 | 1.0 | 0.985 |
| | test_pred | 0.996 | [[8477 35] , [ 0 209]] | 0.856 | 1.0 | 0.923 |
| **step_size=3** | train_pred | 0.998 | [[1371 0] , [ 2 40]] | 0.996 | 0.952 | 0.975 |
| | test_pred | 0.997 | [[2816 5] , [ 4 62]] | 0.925 | 0.939 | 0.932 |
| **step_size=5** | train_pred | 0.999 | [[805 0] , [ 1 20]] | 1.0 | 0.952 | 0.976 |
| | test_pred | 0.997 | [[1676 2] , [ 3 30]] | 0.892 | 1.0 | 0.943 |
| **step_size=7** | train_pred | 0.984 | [[564 5] , [ 4 16]] | 0.762 | 0.8 | 0.780 |
| | test_pred | 0.982 | [[1676 2] , [ 3 30]] | 0.618 | 0.7 | 0.656 |

| Length | step_size=5 | Accuracy | Confusion | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| **window_width=1** | train_pred | 0.984 | [[800 3] , [ 10 25]] | 0.893 | 0.714 | 0.793 |
| | test_pred | 0.980 | [[1654 13] , [ 21 34]] | 0.723 | 0.618 | 0.667 |
| **window_width=5** | train_pred | 0.996 | [[807 2] , [ 5 23]] | 0.920 | 0.821 | 0.868 |
| | test_pred | 0.991 | [[1673 5] , [ 10 34]] | 0.871 | 0.772 | 0.819 |
| **window_width=10** | train_pred | 0.999 | [[805 0] , [ 1 20]] | 1.0 | 0.952 | 0.976 |
| | test_pred | 0.997 | [[1676 2] , [ 3 30]] | 0.892 | 1.0 | 0.943 |
| **window_width=20** | train_pred | 0.998 | [[813 0] , [ 1 20]] | 1.0 | 0.952 | 0.976 |
| | test_pred | 0.982 | [[1676 2] , [ 3 30]] | 0.969 | 0.939 | 0.954 |

We can set $\mathrm{k} = \frac{\text{window\_width}}{\text{step\_size}}$, and as the table shown above, when k decreases, all the performance metrics decline and when k increases dramaly, the performance metrics also get worse. By analyzing the data, when k equals around 4 to 5, our model performs well on both training sets and testing sets.

**Explanation:**

When the **sliding_window_width** and **sliding_step_size** are too high or too low, it can negatively impact the performance of the model for anomaly detection. If the **sliding_window_width** is **too high**, it may lead to **oversmoothing of the data**, resulting in the **loss of important details or variations** in the sensor measurements. This can lead to the model not capturing small changes or anomalies in the data, thereby causing a decrease in performance. On the other hand, if the **sliding_window_width** is **too low**, it may lead to **excessive noise** in the data. The model may then be trained on **noisy data**, leading to decreased performance in anomaly detection. Similarly, if the **sliding_step_size** is **too high**, it may result in a **loss of data points and information**, which can affect the model's ability to capture patterns and variations in the data, potentially leading to decreased performance. Conversely, if the **sliding_step_size** is **too low**, it may lead to the model being trained on **overly redundant or similar data**, which can

affect the model's ability to **generalize to unseen data** and cause a decrease in performance. Our goal is to seek the proper k to keep the balance.

**7.1.8**

Based on the comparison of the Weight Distribution plots for different models, the following commonalities can be observed:

**For Conv1D:**

The weight distribution exhibits six peaks, indicating that the model is particularly sensitive to these features.

During the learning process, there is noticeable variation in biases, suggesting that the model is learning the correct bias values.

**For LSTM:**

Biases are predominantly distributed around 0+ and 1-.

The kernel_0 shows six peaks, and values are mainly distributed in the range [-0.1, 0.1].

Recurrent_kernel_0 also exhibits six peaks.

**For Dense:**

Kernel_0 displays six peaks, and the values are symmetric around 0.

**3.** Explanations and/or hypotheses on the relative performance of the different models.

After fine-tuning hyperparameters, both CNN and LSTM did not achieve better performance on the test set compared to the Hybrid model. I provide the following explanations:

As mentioned before, CNN learns hierarchical information of the data, while LSTM can effectively explore relationships between data in long-term sequential data. The problem we are dealing with involves time series sequences with multiple levels (4 features), so using a CNN+LSTM model can better explore the relationships within the data.

**4.** References

[1] Bagde A M. Predicting Stock Market Time-Series Data using CNN-LSTM Neural Network Model[J]. arXiv preprint arXiv:2305.14378, 2023.

[2] Yang R, Singh S K, Tavakkoli M, et al. CNN-LSTM deep learning architecture for computer vision-based modal frequency detection[J]. Mechanical Systems and signal processing, 2020, 144: 106885.