Neural Network Based Fault Detection for the Tennessee Eastman Process

Arthur Dunbar
Department of Computer Science
University of Houston
APDunbar@uh.edu

Oyekunle Oshidele
Department of Computer Science
University of Houston
oooshide@uh.edu

Sai Sriharsha Santosh Vantipalli
Department of Computer Science
University of Houston
svantip2@cougarnet.uh.edu

Abstract

Petrochemical plants are complex processes that are comprised of interconnected equipment and produce products essential for daily human activities. These facilities use various sensors, including those for level, flow, temperature, and pressure, to monitor and control operations. Several process faults can be detected and classified using data from the sensors. Detecting and Classifying Faults helps mitigate production losses, improve product quality, decrease environmental concerns and improve human safety. The intricate nature of petroleum plants makes it difficult to develop fault detection and classification models derived from first principles, and there is a growing need for flexible and modular approaches to solve these problems. Machine Learning emerges as a promising solution, leveraging readily available sensor data to identify process faults. This report performs a large hyperparameter sweep for Convolution Neural Networks (CNNs) and explores their use and the use of Residual Neural Networks (ResNets) for petrochemical plant process faults detection and classification using the Tennessee Eastman Process as a benchmark.

Keywords – Machine learning, Process fault, Petrochemical plant, Manufacturing industries.

1 Introduction

Modern chemical plants are complex facilities that are distributed across large geographical regions with several processing units. Each unit itself is a complex system consisting of many interacting physical and chemical components. To monitor these large operations, sensors are installed at several points of interest to observe and control process variables in the Distributed Control System (DCS) and other controllers such as the Programmable Logic Controller (PLC). Without proper detection of abnormalities in these processes and quick intervention by maintenance and operation technicians, an undesirable event or failure can occur. This could spread to other facilities and cause a significant unplanned plant shutdown, and lead to a substantial financial loss or a safety event.

In the context of Industry 4.0, where chemical facilities are dynamic and complex, modeling the plant from first principles has become more challenging and there is a need to design monitoring systems that can scale to thousands or even millions of sensors. Aside from modeling approaches, other automated approaches, such as data-driven and analytical approaches, have been researched. The analytical approach involves building a detailed model of the infrastructure with which the sensors interface. This is very challenging and almost impossible due to the complexity of chemical

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

plants and other variables such as environmental disturbances, sensor additions and removal during plant improvement projects. Therefore, the data-driven or machine learning approach, which utilizes historical data from sensors to build algorithms to predict process faults, is widespread and promising.

According to (1), the main idea behind the data-driven approach is feature extraction, where the process data is transformed into more informative, lower dimensional data. Several statistical data approaches such as principal component analysis (PCA) and partial least squares (PLS) have been utilized in the past. Significant attention has been on the use of artificial neural network approaches, such as Deep Neural Network (DNN) and Convolutional Neural Network (CNN), in the ever growing field of process faults detection and classification.

As we show in our work, CNNs, Convolutional ResNets and Neural Network Voting Ensembles deliver extremely accurate results for process fault detection and classification.

Our group members contributions are:

Oyekunle Oshidele: Proposed the research topic and formed the team to work on the project. He acted as the lead writer for the team on the proposal and poster. Also, as the subject matter expert (SME) on the research area, he helped the lead programmer interpret the data and collaborated with him on ideas during model development. As the SME, he was also responsible for finding the related works, summarizing them, and writing the abstract and introduction.

Arthur Durbar: Served as the lead programmer throughout the project. He spent extensive time testing various model architectures and hyper-parameter setups to achieving the best performance from the developed models. He also, helped with finishing the proposal and the poster, and took the lead on writing the final report.

Sai Sriharsha Santosh Vantipalli: Programmed along with the lead programmer, and worked extensively on the model architecture design making many attempts to make various model architectures work. Also, helped extensively with writing the final report. He was responsible for the problem setting and formulation section, conclusion and future work section.

2 Related Works

Deep Neural Network: In (1), fault detection and classification problems are tackled using neural networks with deep hidden layers. They test the effects of two hyperparameters, the number of hidden layers and neurons in the final hidden layer. They also examined the effect of input data augmentation on the accuracy of their models.

After tuning, (1) highlights the results of the following neural networks:

1. A Fault Detector MLP with shape 52-52-25-2-2. This model has a Binary Accuracy of 97.85% and a False Alarm Rate of 0.25%. They reported 97.73% fault detection rate, but this metric does not include fault free accuracy, so it is not equivalent to or definition of Binary Accuracy in section 3.
2. A Fault Classifier MLP with shape 102-102-50-40-18. This model has an Accuracy of approximately 90.62% and a False Alarm Rate of approximately 19%. They reported a Fault Classification rate of 91.18%, but this, again, does not count fault free accuracy. Their results for fault free were not reported except for in a histogram, where it looks to be approximately 81%.

We have several setups that beat both models created by (1)'s best results combined. Our ResNet model found in section 4, for instance, has an accuracy of 98.34%, a binary accuracy of 98.49%, and false alarm rate of 0.10%. This beats the best results of any single neural network they created in every category and it does it all in one model. We also have several ensemble neural network systems that do the same.

(1) was the main target of our project, and our methods were specifically designed to beat theirs. (1) has high accuracy and good false alarm rate when compared to the other papers, but it does not make use of faults 3, 9, and 15 because of claims in (10) that these faults are difficult to detect because of no noticeable change in the mean, variance or standard deviation of the sensor data. This paper is the only true apples to apples comparison from this list of related works to ours.

Statistical methods and traditional Artificial Neural Networks (ANNs) are used in (2). This approach uses a Supervised Deep Recurrent Autoencoder Neural Network (Supervised DRAE-NN) and incorporates dynamic process information over time. A hierarchical structure is developed, grouping similar faults into subsets for more effective detection and diagnosis. Additionally, a pseudo-random binary signal (PRBS) is injected into the system to better capture incipient faults. This architecture produced an accuracy of 93% when tested on the Tennessee Eastman Process benchmark.

Our ResNet has an accuracy of 98.34% and we have many other setups that have better accuracy than this 93%; however, the comparison is not exactly apples to apples because they consider faults 3, 9, and 15 and they introduce a PRBS, and we do not.

Order-Invariant and Interpretable Dilated Convolution Neural Network (OIDLCNN-SHAP) were used by (3). CNNs have limitations in handling tabular data and providing interpretable results. To address these issues, (3) proposed a method, called OIDLCNN-SHAP which combines feature clustering, dilated convolutions, and SHAP explanations. This approach improves feature correlation capture and fault detection performance, and it provides interpretability by identifying root-cause features. Their Multiclass accuracy was 96.4% and their binary accuracy was 92.4%.

Again, just our ResNet has an accuracy of 98.34%, and a binary accuracy of 98.49%, besting their results. However, again this is not a direct comparison because they consider a subset of the faults that are different than the subset we consider for fault classification, and they use 3,9 and 15 for fault detection, which we do not.

An Autoencoder used with an LSTM was used by (4) for anomaly detection to identify rare fault events in chemical plants. LSTM networks are used to classify the detected fault into one or more specific fault types. An autoencoder is trained on normal data to learn low-dimensional representations, enabling it to identify anomalies effectively. The LSTM network leverages its ability to capture temporal dependencies in the predicted fault data to accurately diagnose different fault types. This model gives an accuracy of 88%. They also have a false alarm rate of 0.99%

Our ResNet has an accuracy of 98.34% and a false alarm rate of 0.1% which beats theirs. Our methods are in some way comparable because they consider fault 3, but not 9 and 15, but in other ways we are not that similar because (4) is particularly interested in rare events which comes with unbalanced datasets, and our dataset is balanced for the multiclass problem.

3 Problem setting and formulation

The Tennessee Eastman Process (TEP) is a simulation of complex chemical plant operations, which emulates several key process units. We use the TEP to simulate faults, which enables us to develop and validate systems for fault detection and classification. Fault Detection and Classification enhances the safety and reliability of industrial systems by detecting equipment failure or process disturbances. Our project uses the same dataset as (1). This dataset is from (5). Figures showing the TEP diagram (figure 6) and describing the faults that we are detecting (figure 7) can be found in Appendix E.

3.1 Dataset Description

This dataset consists of 20 distinct fault types as well as one fault free state, making it a 21 class dataset. Each fault type, and the fault free state, contain 500 simulations, which makes a total of 10,500 simulations. Each simulation contains 25 hours of operation, with a sampling time of 3 minutes. This provides 500 data samples in each simulation run. We used the first 300 simulations of each class for training (first 60%) and the final 200 simulations (40%) are used for testing. We made sure to use the exact same breakdown of training and testing examples used in (1) to make our comparisons as close as possible. There are 52 features in this dataset, 41 are process variables and 11 are manipulated variables.

3.1.1 Preparing the Dataset

The TEP dataset from (5) is sampled from an system every 3 minutes, so the data is inherently temporal, but it is not presented in a manner that allows it to be used this way without manipulation. It had to be converted to a form that neural networks, which take advantage of this sort of data, will

recognize. This was done by splitting the data into overlapping chunks of size 'window' with a gap between windows of size 'step.' An explanation of this process can be found in appendix A.

In some ways, this mimics (1)'s idea of 'data augmentation,' where multiple rows of data were added together in order to get an idea of the data's temporal nature, but the version we use here is more sophisticated and allows for more complicated network architecture than MLPs.

Since the fault is introduced 1 hour in and sensor data is taken every 3 minutes, the first 20 entries in each simulation run are from before there are any faults introduced, so they are discarded. As was done in (1), faults 3, 9 and 15 are not considered, making this problem an 18 class problem. They exlucded these faults because of claims in (10) that these faults made no noticable changes to the data's mean, variance and standard deviation.

3.2 Fault Detection and Classification

There are two major goals for this project:

1. Fault Detection: This determines if there is any fault. It is a two-class classification problem where the outputs are labeled as "Fault Free" or "Faulty." Class 0 is the fault free class, and all faults are considered class 1.
2. Fault Classification: This determines precisely what fault, if any, occurred. It is considered a multi-class classification problem with 18 classes.

Neural Networks designed for Fault Classification performed far better than Fault Detection Networks, even in the domain of Fault Detection. We describe how to make Fault Classifiers act like Fault Detector in section 3.3. Fault Classification was easier to train, more stable, converged in fewer epochs, and was more accurate at performing the job of Fault Detecting than any binary Fault Detector we made. It also did this better than the Neural Networks provided in (1). The comparison between binary Fault Detectors and Fault Classifiers was not close.

3.3 Evaluation Metrics

We evaluate each model using the following metrics: accuracy, binary accuracy, and false alarm rate (FAR). The formulas for Accuracy and FAR can be found in appendix B.

The accuracy parameter measures how well the model correctly predicts fault class. We also report binary accuracy, which is the same as normal accuracy, but it considers fault free samples as class 0 and all faulty samples as class 1. Binary accuracy is how we convert the results of a Fault Classifier to the results of a Fault Detector.

The FAR represents the percentage of non faulty samples that are mistaken for being faulty. This is crucial in industrial processes to avoid unnecessary interventions or downtime.

4 Method

4.1 Data Preprocessing

The data is normalized using MinMaxScaler to scale the features between 0 and 1. The formula for doing this, and all others formulas can be found in appendix B.

4.2 Neural Network Architecture Components

The models use 1D Convolutional Neural Networks (1DCNN) and Residual Neural Networks (ResNets) architectures. These neural networks contain several kinds of layers: input layers, 1D convolutional layers, flattening layers, and dense layers, dropout layers, residual layers and output layers. Formulas describing each of these commonly used layers can be fond in appendix B.

4.3 Activation Functions

The activation function for the last layer, i.e., the output layer, is the Softmax activation function. Its formula can be found in appendix B.

To ascertain the best model, we perform a hyper parameter sweep with different activation functions on hidden layers. These activation functions include ReLU, tanh, Leaky-ReLU, GELU, SELU. The formula for describing these can be found in appendix B.

4.4 Batch Normalization

For a mini-batch of data, $x = [x_1, x_2, \ldots, x_m]$, we compute the mean μ_j and variance σ_j^2 for each feature j, and normalize the input features for each example in the batch. We scale and shift the normalized values with learnable parameters γ_j and β_j. The formula for describing batch normalization can be found in appendix B.

4.5 Loss Function

Our models use categorical cross-entropy loss function to measure the difference between predicted values and actual values for fault classification and detection. The formula for categorical cross-entropy loss can be found in appendix B.

4.6 Optimizer

The optimizer we used was Adam. Its formula is in appendix B.

4.7 1D CNN Network and Hyperparameter Sweep

Our solo models presented in this project, save one ResNet, were all 1D CNNs that have the same the general structure as is shown in figure 4 in the appendix. An extensive hyperparameter sweep was performed on these 1D CNN networks, this is described in section 5. All CNNs regardless of hyperparameters had a learning rate of 0.001 and used the Adam optimizer.

4.8 ResNet

We trained a ResNet that used Conv1D layers. This model had trade offs when compared to the 1D CNN networks. On the one hand, this network took much longer to train, and had about the same accuracy as the best 1d CNN models. On the other hand, it had an extremely low FAR. The structure used for this network can be found in figure 5 in the appendix. This network made copious use of dropout layers and had a lower learning rate (0.0001) when compared to our CNNs, contributing to it taking longer to train, but these measures were requried to get the results we achieved.

Unlike the CNNs, there was no hyperparameter sweep performed for this network. The parameters used were: Window size: 15, step size: 1, activation functions: ReLU, kernel size: 3, and Adam optimizer. Descriptions of these parameters are in section 5.

4.9 Voting

All this hyperparameter tuning left us with a large number of Fault Classifiers that all did fairly well. In order to squeeze more accuracy and lower the FAR, simple voting methods were used to combine the results of various groups of models.

5 Results

Our results can be divided into two categories, single model results and ensemble results. Single model results are the result of one neural network, and ensemble results are the results from several neural networks majority voting together.

5.1 Single Model

5.1.1 Hyperparameter Tuning

A large Hyperparameter Sweep was performed on the 1D CNNs described above after they were found to be quick to train and give promising results. These tests were each checkpointed after 100 epochs.

This sweep was initially performed for both Fault Detection and Fault Classification neural networks. However, the Fault Detection network sweep was ended early due to poor results compared to the Fault Classifier sweep, even on its own domain. As a result, this paper almost entirely discusses Fault Classifiers.

The Hyperparemeter Sweep tested the efficacy of the hyperparameters found in table 1. The Results of the Hyper Parameter Tuning can be found in the next subsection.

Activation Functions	Kernel Size	Window Size	Step Size*
ReLU	3	5	1
Leaky ReLU	5	10	10
GELU	7	15	20
SELU	-	-	-
tanh	-	-	-

Table 1: This table shows the Hyperparameters that were tuned. *Step Size was Dropped after it was obvious that 1 was superior. **Not all combinations exist because some are impossible, such as Window Size 5 paired with Kernel Size 7.

Fault Detection while it was still being considered, also used Class Weight, but this hyperparameter was never used for Fault Classifiers. The optimizer and learning rate were not hyperparameter tuned, the optimizer was Adam, and the learning rate was 0.0001.

Table 2 shows the top 4 models from the Hyperparameter Sweep based upon accuracy, the step size was 1 and the window size was 15 for all of the models.

Activation	Kernel Size	Accuracy
tanh	3	98.34%
tanh	5	98.26%
Leaky ReLU	5	98.26%
SELU	3	98.25%

Table 2: The 4 best models from the hyperparameter sweep

The Fault Classification (accuracy) rates here are all better than those found in (1) (their results are given in Section 2). Despite this, (1)'s Fault Detector model still has an advantage when it comes to false alarm rate over these models.

Our best hyperparameter tuned model for FAR, was the same as the best one for accuracy. It had a FAR of 0.44%, which is worse than (1)'s best FAR of 0.25%. However, our model's Binary Accuracy was 98.58%, which beats (1)'s best Binary Accuracy rate of 97.85% because ours was better at detecting faults.

Since this was a hyperparameter sweep to find the best model, we continued training the best model for another 100 epochs with the following results:

Accuracy	Binary Accuracy	False Alarm Rate
98.43%	98.68%	0.93%

Table 3: The best Hyperparameter Tuning Model: Accuracy, Binary Accuracy and False Alarm Rate after 200 epochs of training

This model's results were the single most accurate of any model we made, but the lower false alarm rate was suboptimal and we consider the results of our ResNet discussed in 5.1.3 to be better.

5.1.2 Hyperparameter Tuning Results Explained

Graphs depciting what is described in this section can be found in the appendix.

Activaiton Function: Winner: Tanh, followed closely by SELU and Leaky ReLU.

The following activation functions were tested: ReLU, Leaky ReLU, GELU, SELU, and tanh. See figure 1 for a comparison.

Tanh performed the best. It was better enough that the best two models were both tanh models, one with a kernel size of 3 and one with a kernel size of 5. Some other activation functions were almost as accurate. A Leaky Relu model had the next best accuracy, followed by a SELU model.

There are several possible reasons that tanh performed the best. Firstly, tanh has a self normalizing nature, which limits activations to between -1 and 1, avoiding exploding gradients. This does have a vanishing gradient problem, but it seems to not be a problem here. Secondly, it avoids the dead neurons problem of some activation functions, such as ReLU, because it always provides a gradient. Finally, tanh gets a larger reaction from small changes because it has a steep slope near 0.

Kernel Size: Winner: Tossup. Size 3 for tanh and SELU. 5 for GELU, ReLU and Leaky ReLU.

Kernel Size 3, 5 and 7 were attempted (see figure 2). The absolute best model used kernel size 3, but 3s and 5s were the best for different activation functions, and the 2nd best model had kernel size 5.

There are several reason why kernel size 3 and 5 might have done better than kernel size 7. One particularly good reason is that we did not use padding, except in the ResNet, so there was more data to look over for each point in the smaller kernels. A second reason is that smaller kernel sizes are better suited to learning smaller grained details in features; larger kernel sizes are drown out somewhat by details over larger ranges. Smaller kernels can also preserve the level of sharpness in features better.

Window Size: Winner: As large as the computer can hold in memory, for us 15.

5, 10 and 15 were tried. The results can be seen in the appendix in figure 3.

The efficacy of the models went up with larger window size quite substantially. This makes some degree of sense since it is in effect giving more data for the CNN layers to learn, but the question of overfitting was a concern, but it turns out it was not one we needed to worry about for the memory constraints we have.

Limitation: Window size involves taking advantage of the time series nature of the data, meaning it does not just look at one instance and determine if there is a fault. With a window size of 15 and 3 minutes between each sample, this means, in order for our best models to work, the plant has to run in a state for 45 minutes before we can tell if it has a fault (or 42 if the first measure is immediate). For window size 10, this is 30 minutes, and for window size 5 this is 15 minutes. (1)'s augmented data also had this problem to an extent, but its combinations were of 2 and 3 timestamps, or 6 and 9 minutes respectively.

Also, we noticed that step size being small and window size being large was always better for classification, but we never considered the tradeoff between these two hyperparameters. Window Size 15 created a dataset that was the maximum that could be held in 64 gigs of memory with a step size of 1, but a larger step size would allow for larger window sizes. It is possible that, for instance, a step size of 2 and a window of 30 would give better results.

Other limitations can be found in Appendix D.

5.1.3 ResNet

The ResNet results are as follows:

Accuracy	Binary Accuracy	False Alarm Rate
98.34%	98.49%	0.10%

Table 4: ResNet Accuracy, Binary Accuracy, and False Alarm Rate

The ResNet had a fairly good Accuracy and Binary Accuracy, but it had an amazing false alarm rate of 0.10%, finally beating (1)'s false alarm rate of 0.25%. Despite having a slightly lower accuracy than some other models, this model was our first one to beat every model from (1) in terms of accuracy, binary accuracy, and FAR all at the same time, and the first one to do it for FAR period.

5.2 Ensemble Voting

Our results for accuracy, binary accuracy, and false alarm rate were quite good, but they could be better. The ResNet beat (1)'s false alarm rate, accuracy, and binary accuracy all at once for all the best results from all their models, but we wanted to beat it by even more.

In order to increase both accuracies and lower the false alarm rate even further, various models we trained from the hyperparameter sweep, as well as the ResNet, were set up in various voting groups, where the class identification would be determined by a simple majority vote (where a tie goes to the lowest class number). The results can be found in table 5.

Voting Group	Accuracy	Binary Accuracy	False Alarm Rate
Top 3 Hyp. Tuning Models	98.31%	98.48%	0.09%
Top 4 Hyp. Tuning Models + ResNet	98.39%	98.52%	0.03%
All Window 15 Hyp. Models	98.37%	98.49%	0.02%
All Window 10 Hyp. Models	97.69%	97.90%	0.24%
All Window 5 Hyp. Models	97.01%	97.34%	0.50%

Table 5: Model Voting Results. Top 3 and 4 Hyperparameter Tuning Models can be found in table 2. The Resnet can be found in table 4. All Window X Hyp. Models refers to all models trained that used the same window size including the ResNet for Window Size 15. We did not cross window size because the dataset is differently shaped.

The voting structure with the best results for both accuracy and binary accuracy was the group that contained the top 4 hyperparameter tuning models and the ResNet. It also had an amazing FAR of 0.03%, but all window size 15 models working in tandem had an even better FAR of 0.02%.

The reason we showed the results for Window size 10 and Window size 5 methods is because they mitigate the time related limitations expressed in 5.1.2. The ensemble of Window Size 10 models outperformed all neural networks in all categories from (1), and the window size 5 models were better for accuracy, but slightly less good for binary accuracy and FAR.

6 Conclusions and Future Work

The main goal of our project is to perform fault detection and classification for the Tennessee Eastman Process. We built several models such as CNNs, ResNets and ensemble models that were able to beat the results of (1) in terms of multiclass accuracy, binary accuracy and false alarm rate. For our 1D-CNN models, we performed a hyperparameter sweep that used different activation functions, kernel sizes and window sizes. With the parameters, tanh for activation function, kernel size 3 and window size of 15, the 1D-CNN models showed an multiclass accuracy of 98.43%, a binary accuracy of 98.68% and a FAR of 0.93%. The 1D-CNN model was able to beat (1)'s results in terms of multiclass accuracy and binary accuracy. The ResNet model achieved an multiclass accuracy of 98.34%, a binary accuracy of 98.49% and a FAR of 0.10%, which was able to beat (1)'s results in all evaluation metrics. The 1D-CNN and ResNET models performed well individually but they performed even better when gathered together in an ensemble with the rest of them.

There are several way to expand upon our work. First, we could make a predictor that doesn't take advantage of time series data, so that faults can be predicted from one data sample. Second, our approach was laser focused on beating the results of (1), this means we did not consider faults 3, 9 and 15, nor did we consider many of the concerns the other papers had such as detecting rare events. In the future we could more thoroughly contend with these works as well. Finally, we could try other neural network architectures such as Transformers.

References

[1] Seongmin Heo and Jay H. Lee "Fault detection and classification using artificial neural networks" In 2018 IFAC (International Federation of Automation Control).

[2] Agarwal et al "Hierarchical Deep Recurrent Neural Network based Method for Fault Detection and Diagnosis"

[3] Li et al "An Order-Invariant and Interpretable Dilated Convolution Neural Network for Chemical Process Fault Detection and Diagnosis"

[4] Park et al "Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network"

[5] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1 (it comes from this paper Rieth, C., Amsel, B., Tran, R., and Cook, M. (2017). Additional Tennessee Eastman process simulation data for anomaly detection evaluation. but it is not referenced other than being the source of the dataset)

[6] Danfeng Xie and Li Bai "A hierarchical Deep Neural Network for Fault Diagnosis on Tennesse-Eastman Process".

[7] Zhang and Zhao "A Deep Belief Network Based Fault Diagnosis Model for Complex Chemical Process" in Computers and Chemical Engineering 107(2017)

[8] Zarch and Soltani "An Artificial Intelligent Approach to Fault Isolation Based on Sensor Data in Tennesse Eastman Process"

[9] Y. lecun, Y Bengio and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436-444. 2015.

[10] Zhang, Y. (2009). Enhanced statistical analysis of non-linear processes using KPCA, KICA and SVM. Chem. Eng. Sci., 64(5), 801–811.

A Dataset Alteration Description

Elementas in the dataset are combined as follows:

Each data point takes the form of a 2d matrix where temporally consecutive data is combined in groups of size window size with a gap between their starting point of size step size. Say we have a window size of 3 and a step size of 1, then the data would look like the following:

data entry 1:

$x_{1,1}$	...	$x_{52,1}$
$x_{1,2}$	...	$x_{52,2}$
...	...	...
$x_{1,1+window-1}$	...	$x_{52,1+window-1}$

data entry 2:

$x_{1,2}$	...	$x_{52,2}$
$x_{1,3}$	...	$x_{52,3}$
...	...	...
$x_{1,2+window-1}$	...	$x_{52,2+window-1}$

etc

Where in $x_{52,2}$, the 52 is the feature number (there are 52), and 2 is the original row number the features were taken from.

There are:

$$floor(\frac{n-w}{s}) + 1$$

2d matrices where n is the number of data points, w is the window size, and s is the step.

B Formulas

B.1 Evaluation Metrics

$$\text{Accuracy} = \frac{\text{No. of correctly predicted samples}}{\text{No. of total samples}}$$

$$\text{FAR} = \frac{\text{No. of fault free samples predicted as fault}}{\text{No. of fault free samples}}$$

B.2 MinMaxScaling

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

B.3 Activation Functions

B.3.1 Softmax

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

where N is the number of classes, and z_i is the score for class i.

B.3.2 Hyper parameter tuned activation functions

These are the formulae for the activation functions we tuned over:

$$\text{ReLU}(z) = \max(0, z)$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\text{LeakyReLU}(z) = \max\{z, \gamma z\}, \quad \gamma \in (0, 1)$$

$$\text{GeLU}(z) = \frac{z}{2}\left[1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right)\right]$$

$$\text{SeLU}(z) = \begin{cases} \lambda z, & \text{if } z > 0 \\ \lambda\alpha(e^z - 1), & \text{if } z \leq 0 \end{cases}$$

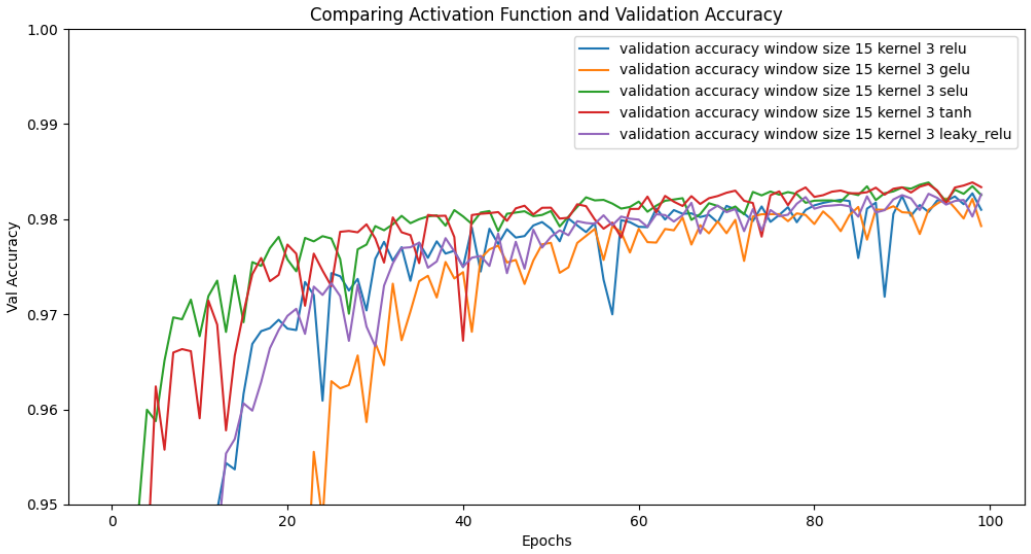

Figure 1: Activation Function Comparison

B.4 Neural Network Layer Equations

The formula for convolutional layers is:

$$Y_i = \sigma\left(X_{i-1} * K_i + b_i\right)$$

Where: X_{i-1} is the input to the i-th layer, K_i is the set of filters (weights) applied at the i-th layer, b_i is the bias at the i-th layer, $*$ denotes the convolution operation, σ is the activation function, and Y_i is the set of feature maps after the convolution and activation. After convolutional layers end, we apply a flattening function to continue the network.

$$Z_{\text{flat}} = \text{Flatten}(Y_i)$$

where Z_{flat} is the 1D vector obtained by flattening the feature map Y_i.

$$Z = W_f \cdot Z_{\text{flat}} + b_f$$

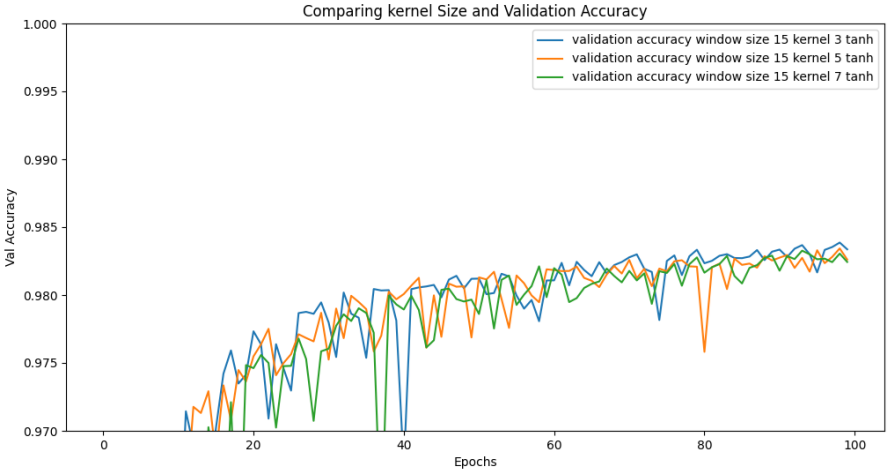

Figure 2: Kernel Size Comparison

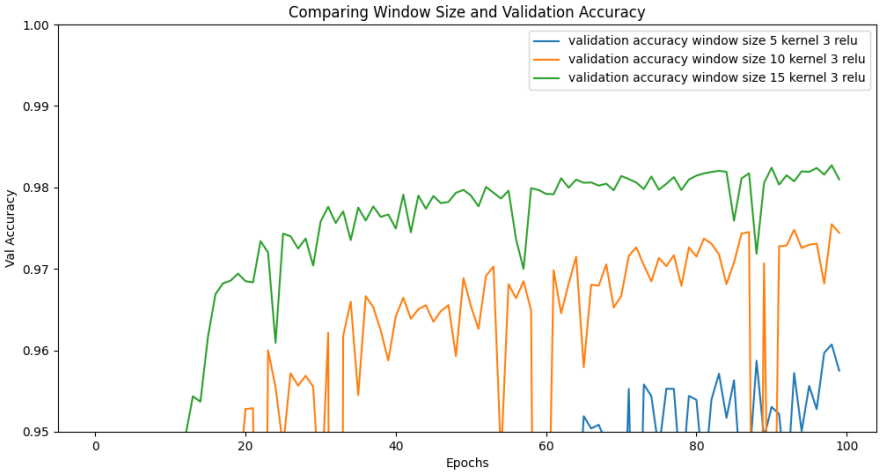

Figure 3: Window Size Comparison

A fully connected layer (also known as dense layer) is applied on the flattened output. W_f and b_f are the weights and bias of the fully connected layer. Z is the output vector of the fully connected layer, which is then passed through an activation function.

The residual connection is defined as:

$$y = F(x, \{W_i\}) + x$$

Where x is the input to the residual block, $F(x, \{W_i\})$ is the residual function computed using convolutional layers, batch normalization, and activation functions, where $\{W_i\}$ are the weights of the layers in the residual block (convolutions and batch norm layers).

B.5 Batch Normalization

For a mini-batch of data, $x = [x_1, x_2, \ldots, x_m]$, we compute the mean μ_j and variance σ_j^2 for each feature j, and normalize the input features for each example in the batch. We scale and shift the normalized values with learnable parameters γ_j and β_j.

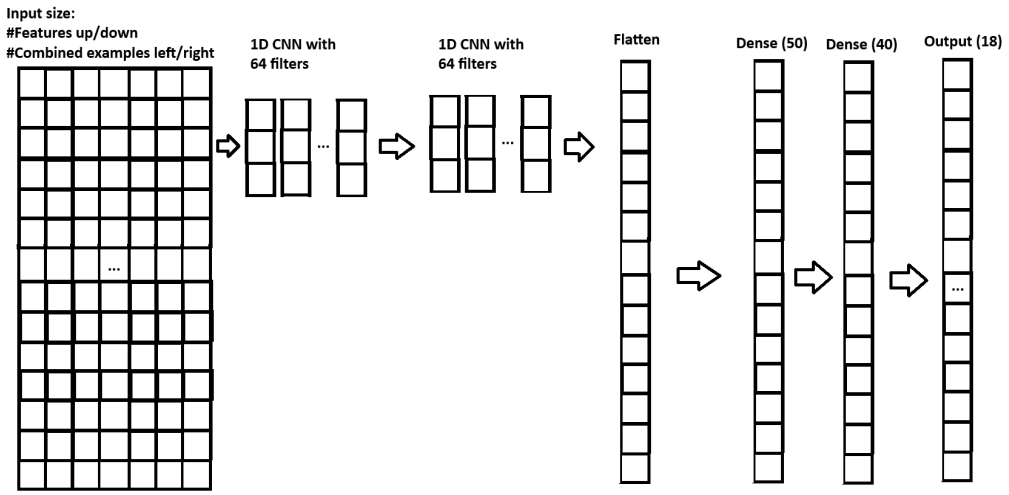

Figure 4: Our 1D Convolutional Neural Network Structure. It feeds the data into a convolutional layer, which feeds it into another convolutional layer, which flattens the data, which then feeds it into 2 fully connected dense layers which then feeds it into the output layer.

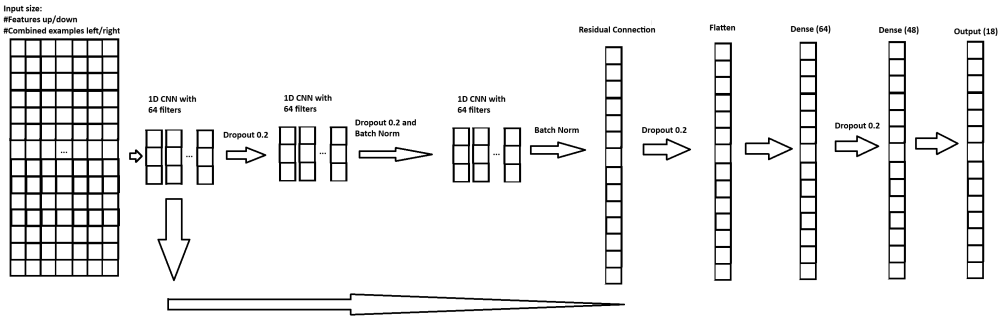

Figure 5: Our ResNet Structure. The data is fed into the first convolutional layer, which sends it with a 0.2 dropout into the next layer, which using batch normalization and dropout 0.2 sends it into the next convolutional layer. This feeds into a residual layer which also takes input from the first convolutional layer and applies batch normalizion. After that, the data is flattened and a dropout of 0.2 is applied and given to a set of dense feed forward layers that have a dropout layer of 0.2 between them and after that the output is predicted

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \quad y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

B.6 Cross Entropy Loss

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Where y_i is the actual probability that the data is for a certain class (typically 1 or 0) and $\hat{y}_i$ is the predicted probability of that class.

B.7 Adam Optimizer

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where θ is a parameter, $\hat{m}_t$ is the exponentially decaying average of past gradients, $\hat{v}_t$ is the exponentially decaying average of past gradients squared, and epsilon is a small constant.

C Other Architectures attempted

Many other architectures were tried, but the general Conv1D schema, found in figure 4, and ResNet network structure, found in figure 5, worked the best. Other architectures were tried, but either took too long to train or did not give any benefit, or both.

Conv2D: Conv2D neural networks were the closest of the ones tried to matching Conv1D's results, and are almost as accurate, but not quite.

This slight lack in accuracy could be because there is a bit of a problem with the idea of convolving over the non-temporal axis. Convolutions are typically used for domains where the data is all similar, like pixels in an image. A dog in the upper left corner is a dog as much as one in the bottom left corner because the pixel locations do not really matter.

The Tennessee Eastman dataset, by contrast, comes from the results of multiple sensors from across the simulated plant, and there is no guarantee that the 'left right' axis of the data (eg. features in the same timestamp) would have any pattern that generalizes to different parts of the dataset, and even if they did, would they generalize from feature 1 to feature 2 in the same way that they did for feature 15 to feature 16 etc. Comparatively, the 'up down' axis (eg. across time stamps) has obvious reasons why it would be useful.

In order to test this theory, a version of the convolutional dataset was made where each matrix in each data point was flipped on its diagonal and a Conv1D neural network was run on it that way instead. Though the network did beat chance accuracy, its accuracy never got above 75%. This confirmed the suspicion that the non-temporal axis is not nearly as useful for convolving over.

In general Conv2D results had approximately 1% less accuracy on the validation set than Conv1D results, and so their results were not published in this project.

LSTM: LSTMs were attempted, but the accuracy had trouble climbing higher than about 90%. It is possible we never hit the 'magic combination' for an appropriate shape to the network, but it was deemed relatively early on that this avenue was not as fruitful as others.

CONV LSTMS (Both 1D and 2D): This network structure might work well, it had fairly strong accuracy for only a few epochs (about 91% after 2 epochs or so), but it is difficult to say if this would continue because the network completed epochs very slowly. A very basic CONVLSTM1D network took about 1 hour to train per epoch. If you have a spare supercomputer lying around we would love to try this on it because it was somewhat promising.

Simpler architectures with less optimal settings (lower window size, higher step size, fewer layers etc) were tried, but though they were tractable, they weren't nearly as good as Conv1Ds, and were abandoned.

Multi Layer Perceptron: (1) used MLPs. We were able to match and exceed their accuracy for fault classification using an MLP as part of our attempts to copy their results, but we had trouble matching fault detection using the methods they described. Regardless, Conv1Ds beat these method quite convincingly in our experiments, so we stopped pursuing them.

D Limitations

D.1 Synthetic Data

This is true of many papers in the field including the one we emulate, but the Tennessee Eastman process is a synthetic simulation of a plant, rather than a real one. We relying upon the TEP being actually representative of real world plants.

D.2 Lack of Tuning on Certain Hyperparameters

Certain hyperparameters that make sense to tune on were left out for time. These include:

Optimizer: All results given were performed on the Adam optimizer. Adadelta was tried, but the results were so abysmal that we decided it was likely we did not use it correctly. Other optimizers would also be interesting to attempt.

Dense Layer Neuron Counts: Almost all models provided here are of the shape x-x-flatten-50-40-18. Though the activation functions were tuned for, the actual structure of the end of the networks was not. If we had more time, we would tun those variables as well.

Some testing was done with this early on, and the results did not change much, but if we had more time we would test this more extensively.

Structure: Certain structures were attempted, like LSTM and Conv2D, but there was not much tuning work put into things like how many layers the network has.

Filters: Our convolutional layers all used 64 filters. Other numbers would be good to test.

Also, some model setups were accidentally ran twice, and these second runs were included in the ensemble votes discussed in 5.2. The ones that were ran twice were GELU and ReLU with window size 15, and with kernels 3, 5 and 7. These were ran on two different computers that accidentally overlapped in their sweeps, but it felt like a waste to throw them out entirely. The results reported in Appendix A were the results for the first run to complete so that the results there were not biased by cherry picking the best result.

E Tennessee Eastman Process Diagrams and Fault Description

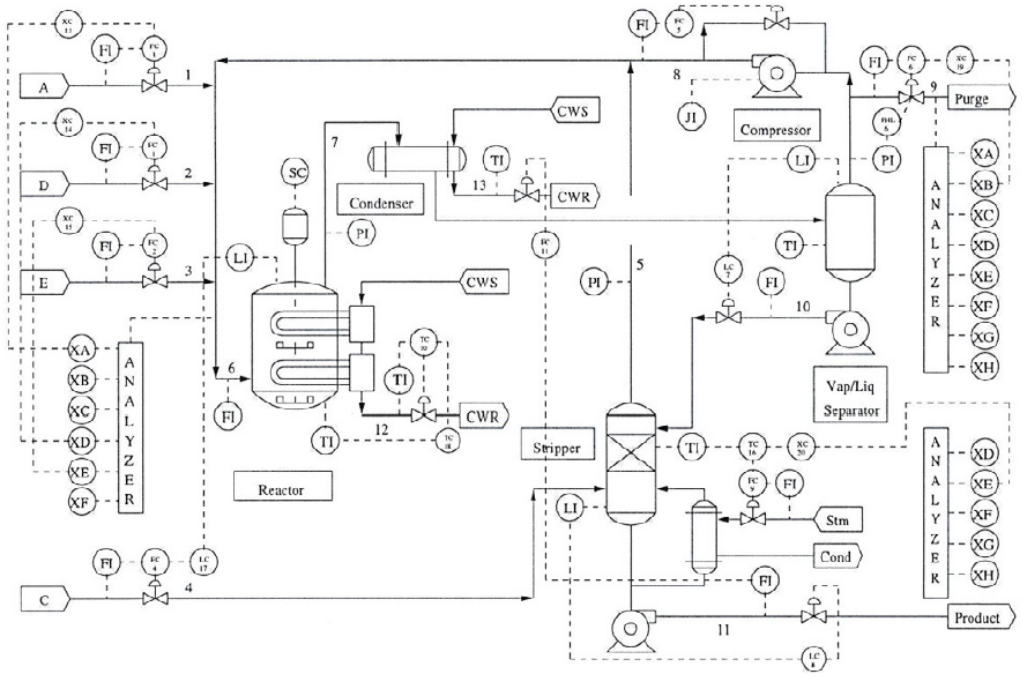

Figure 6: TE Process Process Flow Diagram

Fault ID	Process variable	Type
Fault 1	A/C feed ratio, B composition constant (stream 4)	Step
Fault 2	B composition, A/C ratio constant (stream 4)	Step
Fault 3	D feed temperature (stream 2)	Step
Fault 4	Reactor cooling water inlet temperature	Step
Fault 5	Condenser cooling water inlet temperature	Step
Fault 6	A feed loss (stream 1)	Step
Fault 7	C header pressure loss - reduced availability (stream 4)	Step
Fault 8	A, B, C feed composition (stream 4)	Random variation
Fault 9	D feed temperature (stream 2)	Random variation
Fault 10	C feed temperature (stream 4)	Random variation
Fault 11	Reactor cooling water inlet temperature	Random variation
Fault 12	Condenser cooling water inlet temperature	Random variation
Fault 13	Reaction kinetics	Slow drift
Fault 14	Reactor cooling water valve	Sticking
Fault 15	Condenser cooling water valve	Sticking
Fault 16	Unknown	Unknown
Fault 17	Unknown	Unknown
Fault 18	Unknown	Unknown
Fault 19	Unknown	Unknown
Fault 20	Unknown	Unknown

Figure 7: TE Process Faults Description

NEXT PAGE STARTS THE PROPOSAL!!!!!!

F Summary of the original paper

Our project is an adaptation and improvement on the methods found in [1]. In that paper they use artificial neural networks to detect and classify faults in the Tennessee Eastmen simulation dataset [5]. They also investigate the effect of two hyper-parameters, the number of hidden layers, and the number of perceptrons in the final hidden layer on this problem.

F.1 Methods of the original paper

First, [1] formulates fault detection and fault classification as neural network based classification problems. Fault detection is treated as a binary classification problem with the labels normal and fault. Fault classification is a multiclass classificatoin problem. Then they create several neural networks with different hyperparameters for the number of hidden layers and the number of perceptrons in the last hidden layer, and then they see how well these different set ups perform on the tasks.

Fault Detection [1] is specific about the shape of the neural network they use and what optimizations they use with them. In the section Fault Detection Results, they report their results when using networks with 0 through 4 hidden layers. These networks use Xavier initialization and the Adam optimizer. They divide the data into 50 batches and train for 400 epochs. They try several numbers of hidden layers (fig 1). They then perform data augmentation in the form of combining 2 and 3 consecutive samples to perform an approximation of Dynamic Principle Component Analysis (DPCA), a sort of time series combining of data, and they use this on the 3 hidden layers 52-52-25-2-2 neural network because they deemed more layers unnecessary.

Number of hidden layers	Network structure
0	52-2
1	52-25-2
2	52-25-12-2
3	52-52-25-2-2
4	52-52-25-12-2-2

Figure 8: Network structure of neural networks with different number of layers

Fault Classification They use the approximation of DPCA where they combine two consecutive samples to make a neural network with 102 input perceptrons that has the shape 102-102-50-40-18 that is trained alongside a 52-52-52-40-18 neural network that doesn't use the DPCA.

F.2 Dataset

The dataset is from another paper and the dataset can be found here [5]. This dataset models a system that has 5 process units, the reactor, condenser, compressor, separator and stripper. This system produces products G and H and byproduct F from four reactants A, C, D and E. There is also an inert compound B. There are 52 measurements, of which 41 are process variables and 11 are measurements of manipulated variables. This 52 is where the number of input perceptrons comes from. There are 20 different fault types, fault 1 through fault 20.

The dataset provides 25 hours of simulation split in to 3 minute intervals for 500 simulation runs for each normal and fault state (10500 total runs). A fault is introduced 1 hour in to the run of each simulation that has a fault. To train they used 300 of the simulation runs from each type and 200 were used to test.

F.3 Results of the original paper

These are there results for fault detection (fig 2, the ours column). Faults 3, 9 and 15 were deemed to hard to classify by this paper, so they didn't bother. Also, though the problem was a binary classification problem they report their accuracy for detecting any fault for each fault.

Fault ID	DPCA	MPLS	ICA	DBN (s)	DBN (G)	Ours
1	99.88	100	99.88	100	98	100
2	99.38	98.88	98.75	97	95	99.51
4	100	100	100	100	100	100
5	43.25	100	100	87	79	100
6	100	100	100	100	100	100
7	100	100	100	100	100	100
8	98	98.63	97.88	77	89	98.06
10	72	92.75	89	0	98	93.96
11	91.5	83.25	79.75	12	91	97.20
12	99.25	99.88	99.88	1	72	98.69
13	95.38	95.5	95.38	60	91	95.78
14	100	100	100	5	91	99.97
16	67.38	94.38	80.13	0	0	95.41
17	97.25	97.13	96.88	100	100	95.93
18	90.88	91.25	90.5	100	78	94.15
19	87.25	94.25	93.13	13	98	99.18
20	73.75	91.5	90.88	93	93	93.62
Overall	89.13	96.32	94.83	61.47	86.65	97.73

Table 5. Fault detection rates (%) of different data-driven methods

Figure 9: Fault Detection Accuracy

These are the results for fault classification (fig 3, the ours column). Again faults 3, 9 and 15 are not included.

Fault ID	SNN	HNN	SAE	Ours
1	81.19	97.51	98.75	99.31
2	81.97	98.26	98.33	98.16
4	80.02	95.89	93.10	98.50
5	73.33	97.13	99.79	98.46
6	83.31	99.38	97.70	100
7	81.49	100	99.37	99.86
8	46.65	60.15	53.98	91.55
10	10.96	46.33	63.39	81.19
11	17.30	47.32	62.97	86.32
12	24.12	45.21	66.74	89.30
13	16.93	31.51	29.29	87.35
14	29.11	66.75	94.56	99.66
16	15.83	36.61	66.53	81.76
17	51.15	70.74	88.70	91.76
18	75.76	94.89	88.29	86.57
19	14.50	51.18	82.64	79.40
20	46.16	66.25	77.20	80.92
Overall	48.83	70.89	80.08	91.18

Table 7. Fault classification rates (%) of different neural network models

Figure 10: Fault Classification Accuracy

They also have a significantly lower false alarm rate than their competition. Their false alarm rate is 0.25% versus 2.63 to 15.13% in their competition.

G Literature Review

Deep Recurrent Neural Network In [2], a Deep Neural Network (DNN)-based algorithm is introduced for fault detection and classification in industrial plants. This approach excels at identifying faults, particularly incipient ones, which are often challenging to detect using conventional threshold-based statistical methods or traditional Artificial Neural Networks (ANNs). This algorithm leverages a Supervised Deep Recurrent Autoencoder Neural Network (Supervised DRAE-NN), incorporating dynamic process information over time. A hierarchical structure is developed, grouping similar faults into subsets for more effective detection and diagnosis. Additionally, a pseudo-random binary signal (PRBS) is injected into the system to better capture incipient faults. This architecture produced an accuracy of 93 percent when tested on the Tennessee Eastman Process benchmark.

Order-Invariant and Interpretable Dilated Convolution Neural Network (OIDLCNN-SHAP) CNNs have limitations in handling tabular data and providing interpretable results. To address these issues, [3] proposed a method, called OIDLCNN-SHAP, combines feature clustering, dilated convolutions, and SHAP explanations. This approach improves feature correlation capture, fault detection performance, and provides interpretability by identifying root-cause features.

Autoencoder used in an LSTM An autoencoder approach has also been used for anomaly detection, identifying rare fault events in chemical plants, then LSTM networks are then used to classify the detected fault into one or more specific fault types. An autoencoder is trained on normal data to learn low-dimensional representations, enabling it to identify anomalies effectively. The LSTM network leverages its ability to capture temporal dependencies in the predicted fault data to accurately diagnose different fault types [4].

H Potential Solutions

The neural networks that are implemented in [1] are standard MLP Neural Networks with a data augmentation step called DPCA. Though it was effective for them, it is not a highly technical solution. One potential way to improve upon their work is to implement a more modern neural network such as a Convolutional Neural Network, a Recurrent Neural Network or a Transformer. The DPCA was performed to take advantage of time related data, so this seems like the kind of problem that a CNN, RNN or Transformer would be able to help with.

I References

[1] Seongmin Heo and Jay H. Lee "Fault detection and classification using artificial neural networks" In 2018 IFAC (International Federation of Automation Control).

[2] Agarwal et al "Hierarchical Deep Recurrent Neural Network based Method for Fault Detection and Diagnosis"

[3] Li et al "An Order-Invariant and Interpretable Dilated Convolution Neural Network for Chemical Process Fault Detection and Diagnosis"

[4] Park et al "Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network"

[5] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1 (it comes from this paper Rieth, C., Amsel, B., Tran, R., and Cook, M. (2017). Additional Tennessee Eastman process simulation data for anomaly detection evaluation. but it is not referenced other than being the source of the dataset)

[6] Danfeng Xie and Li Bai "A hierarchical Deep Neural Network for Fault Diagnosis on Tennesse-Eastman Process".

[7] Zhang and Zhao "A Deep Belief Network Based Fault Diagnosis Model for Complex Chemical Process" in Computers and Chemical Engineering 107(2017)

[8] Zarch and Soltani "An Artificial Intelligent Approach to Fault Isolation Based on Sensor Data in Tennesse Eastman Process"