CME 660 Term Project

Tennessee Eastman Process Data Fault Analysis

Submitted to: Dr. Vinay Prasad

Submitted by: Mohammad Aarooj Yashin Ali

DEPARTMENT OF CHEMICAL AND MATERIALS ENGINEERING

1. Introduction

In the dynamic landscape of process industries, the quest for operational excellence and safety is perpetual. This report digs into a project aimed at enhancing the reliability and efficiency of the Tennessee Eastman process, a cornerstone in chemical manufacturing. The ultimate goal of this initiative is to develop a robust framework for fault detection and classification, a leap towards predictive maintenance and process optimization that could revolutionize standard practices in the industry.

The Tennessee Eastman process, emblematic of complex industrial operations, presents a fertile ground for applying advanced statistical and analytical models. By leveraging techniques such as PCA, Dynamic PCA and Autoencoder for dimensionality reduction and using Logistic Regression, Random Forest Classifier and Deep Neural Networks to achieve fault detection and classification, this project seeks to transform raw process data into actionable insights. The ability to accurately detect and classify faults in real-time is not just an operational imperative but a strategic advantage. It promises to reduce downtime, optimize resource utilization, and significantly mitigate risks, ensuring a safer and more productive industrial environment.

2. Dataset

The dataset considered for this report is the Tennessee Eastman Process (TEP) Data. TEP data refers to a simulated dataset derived from a chemical production process, originally developed by the Eastman Chemical Company. This simulation replicates a real industrial process for producing chemicals, and it is commonly used in academic and research settings for testing process monitoring, fault detection, and control strategies. The TEP dataset is particularly valued for its complexity and realism, encompassing various operational modes, including normal operation, system disturbances, and multiple fault scenarios. This makes it an ideal test bed for developing and evaluating advanced statistical and machine learning models in the context of process optimization and anomaly detection in industrial settings.

The dataset includes 21 process conditions, with one representing normal operation and the remaining 20 simulating various fault types. The training process involves 500 simulation runs for each operational mode, including fault-free and 20 faulty modes, with each simulation run lasting 25 hours and having a 3-minute sampling interval. This results in 500 samples for each simulation run, amounting to a total of 5,250,000 data points in the training dataset. Likewise, for the main test dataset, there are 1,008,000 data points (each simulation runs for 48 hours), and both the training and test datasets consist of 55 features (52 variables, Fault number, simulation run, and samples). The introduction of faults occurs after 1 hour (equivalent to the 20th sample) for the training data and at the 8th hour (equivalent to 160th sample) for the testing data

There are a total of 22 process measurements, 11 manipulated variables and 19 component analysis variables. In industrial settings, component analysis happens asynchronously with the sampling of process variables, which means that 19 of these component analysis variables can be omitted for any data analysis[1]. To sum it up, the dataset at any given moment can be composed of 33 selected variables. The header values for the selected columns would be (xmeas_1 to xmeas_22 and xmv_1 to xmv_11). The cited references [2,3,4] suggested that faults number 3, 9 and 15 are very similar to the fault free condition, which makes it difficult in detecting faults and costs the model heavily in classification. According to this research paper[1] the mutual information of fault types (calculated by the formula proposed by Verron[5] to measure dissimilarity between fault operation and normal operation) conclude to opt out faults types categorized as: 3, 5, 8, 9, 10, 12, 15, 16, and 18. Based on all the information gathered from various sources I have prepared two types of dataset as follows:

2.1 Downsampling of Main Dataset:

As described above, the dataset has too many samples and it becomes difficult in modeling and computations with such a large dataset. So, to tackle this problem I have downsampled my data from the main dataset accordingly:

Sampled Train Dataset: For normal (fault free) operation mode I have taken 40 simulation runs, each consisting of 500 samples (i.e. 25 hours of runtime), making it 20,000 total samples, and likewise for each faulty mode I have taken 25 simulation runs, each consisting of 480 samples. For faulty modes the fault has been introduced at 21st Sample (i.e. after 1 hour), hence it is 480 samples (taking samples from 21st to 500th sample (inclusive)). Total number of rows = 260,000 and columns (Features) = 55

Sampled Cross Validation (CV) Dataset: For normal mode, I have taken the next 20 simulation runs (41st to 60th Simulation run) of the main train dataset (10,000 samples). Each faulty mode is constituted of next 10 simulation runs (i.e. 26th to 36th simulation run). Hence total number of rows = (10,000 + 10*20*480) = 106,000 and 55 columns.

Sampled Test Dataset: For normal mode, I have taken the first 2000 samples of the main test dataset. Each faulty mode is constituted of the first 10 simulation runs of the main test dataset, followed by selection of 161th sample to 640th sample for each simulation. In the test dataset the fault is being introduced at the 8th hour (i.e. 160th sample). Total number of rows = (2000 + 10*20*480) = 98,000 and 55 columns.

2.2 Dataset 1 (Less Faults):

Now the sampled train, CV and test dataset have been used to drop some columns (features) based on the information given in (1). Hence, this dataset consists of 33 process variables in total, and includes the following categories of fault: 0, 1, 2, 4, 6, 7, 11, 13, 14, 17, 19, and 20. The number of rows in train = 152,000, test = 54,800 and CV = 62,800

2.3 Dataset 2 (More Faults):

Now the sampled train, CV and test dataset have been used to drop some columns (features) based on the information given in (1,2,3 and 4). Hence, this dataset consists of 33 process variables in total, and includes the following categories of fault: 0, 1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16, 17, 18, 19 and 20. The number of rows in train = 224,000, test = 83,600 and CV = 91,600

3. Objective:

The objective is to detect and classify the different types of faulty modes and normal mode of the Sampled Test dataset precisely, based on the mathematical models' learning from the Sampled train dataset. For my experiment I have considered the following models:

1. **Logistic Regression for Classification on Dataset 1:**
    a. **Task 1:** First perform an EDA (Exploratory Data Analysis) on the Sampled train dataset, to identify the process variables that are highly correlated and drop them for the analysis along with all the 19 component analysis variables.
    b. **Task 2:** Use PCA for dimensionality reduction to obtain a number of components covering 95% variance. This is followed by training the PCA transformed dataset in a logistic regression model.
    c. **Task 3:** Use dynamic PCA for dimensionality, followed by training the transformed dataset in a logistic regression model for classification.
2. **Random Forest Classifier on Dataset 1:**
    a. **Task 1:** First a gridsearch has been performed for hyperparameter tuning.
    b. **Task 2:** Using the hyperparameters from the previous task, a random forest classifier has been devised for multi-class classification.
3. **Feedforward Neural Network on both Dataset 1 and Dataset 2:**
    a. **Task 1:** Feed the 2 dimensional data of (samples, features) to an encoder to get the latent space.
    b. **Task 2:** Connect the latent space to a Dense network classifier to get the multi-class classification.
4. **LSTM (Long-Term Short Memory) on both Dataset 1 and Dataset 2:**
    a. **Task 1:** First convert each of the dataset into sequences using sliding window technique.
    b. **Task 2:** Feeding the transformed sequential 3D data (samples, timesteps, features) to a basic LSTM model for multi-class classification.
    c. **Task 3:** Build a bidirectional LSTM model integrated with a dense network for classification. The bidirectional LSTM, followed by a unidirectional LSTM transforms the data into a format that can be fed into the neural network classifier.
5. **GRU Based Encoder followed by Dense Neural Network on both Dataset 1 and Dataset 2:**
    a. **Task 1:** First convert each of the dataset into sequences using sliding window technique.
    b. **Task 2:** Build a GRU based encoder, that transforms the input to a latent space, followed by connection to dense neural network that works as a classifier

Using the t-SNE approach and taking 2 components, Figure 1 suggests that a non linear classifier would be preferable over linear classifier. Hence I expected that the precision and accuracy of Logistic regression would be very low if Dataset 2 would have been used, because this has many faulty modes included that have a similar trend to normal mode[1]. The experiment 1 therefore only deals with dataset 1. Also to have a comparison of Dataset 1 with a non linear machine learning model, I chose a Random forest classifier in experiment 2. Besides, experiment 1 also intends to evaluate, logistic regression model, PCA followed by Logistic regression and dynamic PCA followed by logistic regression.

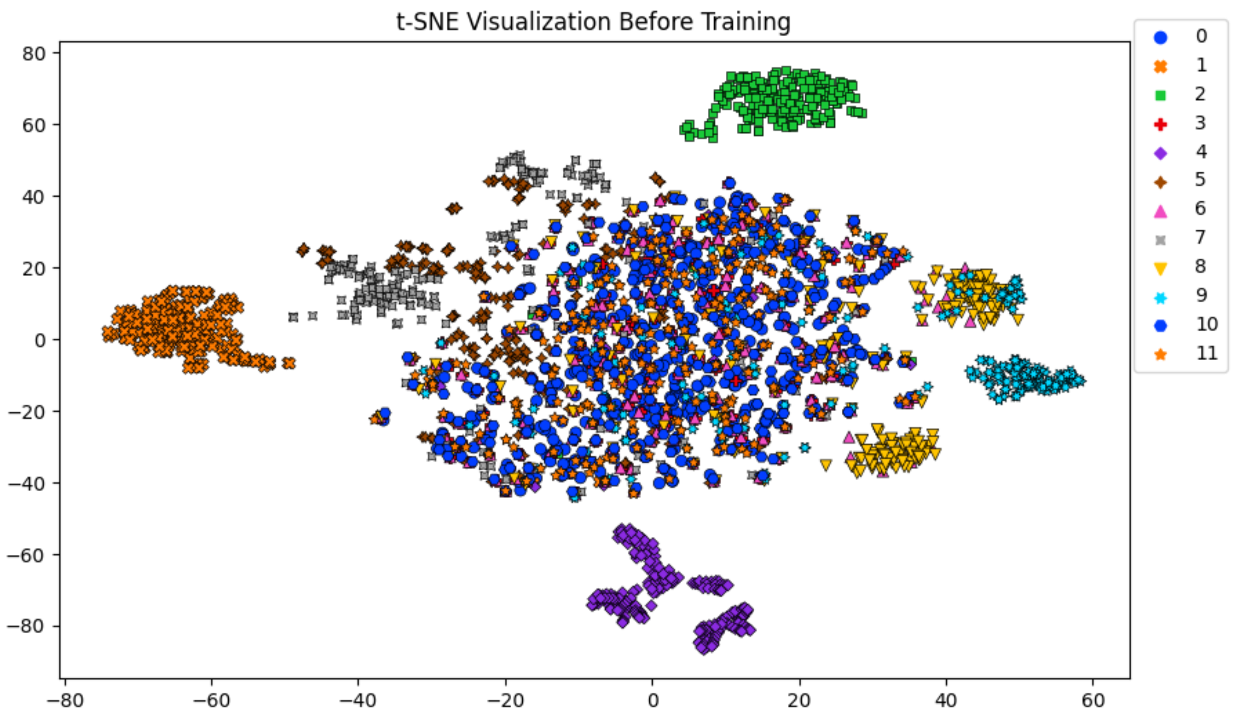


Figure 1: t-SNE Visualization of the training dataset for number of components = 2

For experiment 3 and 5 I have tried to achieve the latent space by two different methods for both datasets 1 and 2, and aim to come to a conclusion on which model is better in terms of precisely identifying and classifying the different categories. In experiment 4, I plan to evaluate the basic LSTM model and a Bidirectional LSTM model integrated with a dense neural network classifier.

Evaluation metric: Precision = $\frac{TP}{TP + FP}$ for each class label (here TP = True Positive, FP = False Positive), mean of precision of all class labels, mean of precision of all faulty modes and accuracy of the model.

Hence, the overall objective of this project is to determine the best model for Dataset 1 and Dataset 2 on the basis of the evaluation metrics set above.

4. Methodology:

In this section I will discuss the steps and theory behind each and every model proposed in previous section:

4.1 Logistic Regression on Dataset 1:

1. **Data Preprocessing:** As per the objective the Task 1 and the dimensionality reduction part of Task 2 and Task 3 constitutes the Data Preprocessing section. Using KDE plot and boxplot, I have

first tried to understand how each variable affects all faulty modes. Plot for variables that have curves, clearly delineated distributions, can be considered to be important features. Consequently, I have printed the correlation matrix to figure out features that are highly correlated with each other, and for features that correlated more than 90% I have kept only one of the features in my variable space. Refer section 2.1 for downsampling and 2.2 for definition of Dataset 1. Finally the given columns were dropped from Dataset 1 based on my analysis, 19 component analysis variables (xmeas_23 to xmeas_41), xmeas_11, xmeas_13, xmeas_16, xmv_6, xmv_7, xmv_8, xmv_9, and xmv_11. I have attempted this exploratory data analysis for this model only exclusively because of my understanding that the Figure 1 suggests the dataset is more suitable for non linear classifiers. To get better results for a linear classifier like logistic regression I have tried to omit all those features that can be redundant for the model.

In the next phase (Task 2), I standardized this modified x_train data using the StandardScaler object. It is fit on the training data and then used to transform the CV and test dataset. Following that the y_train, y_test and y_cv data takes the values of the column named 'Fault number', as they denote the category each sample in the x_train belongs to. Thereafter the scaled x_train is taken into consideration for applying PCA to extract the components that capture 95% of the dataset. This results in taking 17 components. Subsequently test and CV data have also been transformed into the new space.

In Task 3, a function has been made to create a lagged feature set for train, test and CV data. The number of lags considered are 15. This data is standardized using object StandardScaler and then has been transformed using PCA to include features that capture 95% variance. The number of components for the lagged feature set is 122. Subsequently test and CV data have also been transformed into the new space.

2. **Modeling Frameworks:** There are 3 kinds of preprocessed data in this scenario, all of them are fed into a logistic regression model to get the classification between normal and different types of faulty modes. The input dimensions of dataset 1 after EDA is (152000, 25), the input dimensions of Dataset 1 after PCA transformation is (152000, 17), and the input dimensions of Dataset 1 after dynamic PCA is (151985, 122).

   In the subsequent stage after the training is carried out on the input, the model uses the test dataset (54800, 25) to predict an output of dimension (54800,1). The output includes the labels for each row in the output dimension. Similarly the PCA transformed test input dimensions (54800,17) gives an output of (54800, 1). And dynamic PCA transformed test input dimensions (54875, 122) gives an output of (54875, 1).

   During model training the Cross-entropy loss function is being optimized. Logistic regression tries to estimate the weights (coefficients) and the intercept (bias). Weights (Coefficients): In logistic regression, each feature of the input data is assigned a weight (or coefficient). These weights determine the impact of each feature on the prediction. The model learns the most appropriate value for these weights during the training process, which involves adjusting them to minimize a loss function (log loss / cross-entropy). Intercept (Bias): This is a constant term in the logistic regression equation. It is like an offset and is also learned from the training data. The intercept represents the log odds of the outcome when all the predictors (features) are held at zero. The logistic regression model essentially learns these parameters such that when a linear combination of the input features, weighted by these learned coefficients, is passed through a logistic (sigmoid) function, it yields the probability of the target variable being in a particular class (for multi-class classification).

In mathematical terms, if my input features are denoted as x and target variables as y, the logistic regression model predicts y using the formula, Y_cap = σ(w . x + b), where Y_cap is the predicted probability, σ is the sigmoid function, w represents the weights, x is the input feature vector and b is the intercept.

4.2 Random Forest Classifier on Dataset 1:

1. **Data Preprocessing:** In Dataset 1 (downsampled as given in section 2.1), the columns belonging to xmeas_23 to xmeas_41 are dropped for this analysis. As a follow up the dataset 1 is standardized (as given in section 4.1) to get the x_train, and x_test. Correspondingly the y_train and y_test are formed by taking the values of a column named 'Fault Number'.

2. **Modeling Framework:** This consists of two tasks, Task 1 is Gridsearch for hyperparameter tuning and Task 2 is performing a random forest classifier on the preprocessed data. Dimensions of X_train and X_test for the dataset are: (152000, 33) and (54800, 33). Accordingly the y_train and y_test are: (152000,1) and (54800, 1).

   Task 1 tries to achieve the appropriate value of the parameters: 'n_estimators' and 'max_depth', where, n_estimators denotes the number of decision trees to be considered and max_depth refers to the depth of the decision tree. It evaluates the performance of various hyperparameter combinations using cross validation and opts the best combination based on the F1 score. N_jobs parameter in the code refers to how many cores of CPU to be used. n_jobs = -1 asks to use all the available cores. After the gridsearch I get n_estimators to be equal to 500 and max_depth to be equal to 35. The loss function is not explicitly mentioned for Random forest, because it does not directly optimize a loss function during training

4.3 Encoder + Neural Classifier on Dataset 1 and Dataset 2:

1. **Data Preprocessing:** Both dataset 1 and dataset 2 are downsampled as given in section 2.1, and the columns belonging to xmeas_23 to xmeas_41 are dropped for this analysis. As a follow up the dataset 1 is standardized (as given in section 4.1) to get the x_train, and x_test. Correspondingly the y_train, y_cv and y_test are changed by using one hot encoding on the values of a column named 'Fault Number'. Here the dataset is in the 2D form of (samples, features). The data has not been formed into sequences to capture the temporal nature.

2. **Modeling Framework:** X_train, X_cv and X_test for dataset 1 and dataset 2 respectively are of following dimensions : (152000, 33) and (224000, 33); (62800, 33) and (91600, 33); (54800, 33) and (83600, 33), and accordingly the y_train, y_cv and y_test for database 1 and dataset 2 after one hot encoding are: (152000, 12) and (224000, 18); (62800, 12) and (91600, 18); (54800, 12) and (83600, 18). After the input is sent through the neural network, it gives an output of the dimensions same as y_test. To explain the architecture of the model, I will consider only dataset 1 here, dataset 2 is likewise.

   Dimensionality Reduction: The input layer is defined using Input(shape=(X_train.shape[1],)). 33 is X_train.shape[1] shows that there should be 33 features in every sample in my input layer. Encoder's Section on Reduction of Dimensionality: Following receipt of the input, the initial dense layer It is transformed into a 64-dimensional space using Dense(64, activation='relu'). This is more of a feature modification than a decline in general. The next layer, Dense(16, activation='relu'), reduces the dimensionality even further to 16. My latent space is represented by this. The latent space, a lower-dimensional representation of my input data, aims to extract the important features and specifics.

   As the dimensionality decreases, the mesh enters this phase. Classifier levels: two complex nodes of 100 and 50, Dense(100, activation='selu') and dense(50, activation='selu'). Deep learning models benefit from the use of 'selu' (Scaled Exponential Linear Unit) activations at these levels as they normalize the output of previous levels and maintain 0 and a variance of 1. These levels go through further processing Latent space characteristics to prepare for final classification. Output Layer: The last layer, Dense(y_train.shape[1], activation='softmax'), is the output layer. Based on the fact that y_train.shape[1] = 12, my model divides the input data into twelve classes. The "softmax" activation function is used to solve multiclass classification problems. For each of my twelve subjects this will produce a probability distribution, with each number representing the probability that a given sample belongs to that class. Model Acquisition: Using the 'Adam' optimizer well known for its sparse gradient and scalable compile the model correctly the number of classes 'categorical_crossentropy ' The loss function is suitable for classification problems with multiple classes The performance of the model in training and testing is evaluated using the 'accuracy' metric. To summarize, my method first reduces the input dimension from 33 items to a hidden space of 16 items to find the most important items for the classification statement and then which way so easily divides the input into twelve categories. In complex classification tasks, this scheme works well because the extraction is simplified by reducing dimensionality.

<u>4.4 LSTM on Dataset 1 and Dataset 2:</u>

1. **Data Preprocessing:** In Dataset 1 and Dataset 2 (downsampled as given in section 2.1), the columns belonging to xmeas_23 to xmeas_41 are dropped for this analysis. As a follow up, both the datasets are converted to sequences using the sliding window technique. The sliding window considers a window size of 20 and a stride of 10. In each window all samples taken behave as an input of the sequence and the last sample of the window's column 'Fault Number' serves as the output of the sequence. I am considering the last row of my window in the sequence generation,

because I aim to do a real time fault prediction. Finally, this converts the initial data into sequences, with a timestep of 20 and 33 features. The 2D data is transformed into a 3D data

The 3D dataset 1 and 2 are standardized by using object Standard Scaler on the Fault free data of training and fitting it to get x_train, x_cv and x_test. Correspondingly the y_train, y_cv and y_test are formed by applying one hot encoding, because this model prefers to take categorical values not at its integral values, rather as a matrix containing features = Types of different categories.

2. **Modeling Framework:** Dataset 1 and 2 after sliding window technique have dimensions of the form (samples, timesteps, features), (where sample refers to number of sequences), for the train dataset as following: (14885, 20, 33) and (21935, 20, 33) respectively. Similarly for CV dataset: (6150, 20, 33) and (8970, 20, 33). And for the Test dataset: (5367, 20, 33) and (8187, 20, 33). The y_train after one hot encoding for Dataset 1 is (14885, 12) and Dataset 2 is (21935, 18). Similarly for y_CV they are, (6150, 12) and (8970, 18) respectively. For y_test the dimensions are, (5367, 12) and (8187, 18) respectively. I have considered two architectures for LSTM; basic LSTM model and Bidirectional LSTM model integrated with densely connected neural classifier. To explain the architecture of the basic LSTM model, here I have only considered the Dataset 1, dataset 2 is likewise.

   Basic LSTM Model:
   The output layer of my model is a Dense layer with num_classes neurons and a softmax activation function. Given that y_train dimensions, I have 12 categories. This output represents the probability distribution in 12 classes for each sample, due to softmax activation. The characteristics of the output are hierarchical, according to the softmax activation commonly used in multiclass classification problems. The loss function used is categorical_crossentropy. This is a standard loss function for multiclass classification problems where labels are single-hot encoded. The weights and biases of the LSTM layers are the parameters in my LSTM model that I am attempting to learn. These include the weights and biases associated with the input, forget, cell, and output gates of each LSTM unit. Dense layer also has weight and biases associated with it, which are learnt during the training process. Here, Adam optimization technique is used. During training it attempts to get weights and biases in such a way that the loss function is minimal. LSTM 1 has return_sequences set to ‘True’, whereas LSTM 2 has set it to ‘False’, because the output of LSTM 2 is the input for the dense network. The dense layers map the output of LSTM to all class probabilities via softmax function.

   My training data consists of 14885 samples, each with 20 timesteps and 33 features. The output is a 12-class classification problem. Given the architecture of my model, the summary would look something like this:

   LSTM Layer: Input shape: (20, 33) (20 timesteps, 33 features), 64 units. Parameters: Each LSTM unit has 4 parts (input, forget, cell state, output), each with a weight matrix, recurrent weight matrix, and bias vector. The total number of parameters for a single LSTM unit is given by 4 * (num_units * input_dim + num_units^2 + num_units). In this case, it would be 4 * (64 * 33 + 64^2 + 64). Dropout Layer: No parameters, as dropout layers do not alter the input; they only randomly set a fraction of input units to 0 at each update during training time to prevent overfitting. Second LSTM Layer: Since the first LSTM returns sequences, the input to this layer is (20, 64). 32 units Parameters calculation similar to the first LSTM layer, but with the input dimension now being 64 (the output of the previous LSTM layer). The formula is 4 * (32 * 64 + 32^2 + 32). Dense Layer: Input from the second LSTM layer is flattened (or it could be taking the output of the last timestep if return_sequences=False in the second LSTM). 12 units (for 12 classes). Parameters: num_units_in_previous_layer * num_units_in_dense + num_units_in_dense (for biases). Here, it would be 32 * 12 + 12.

Summary:
1st LSTM Layer: Units: 64, 25,088 Params, Dropout Layer 0 params, 2nd LSTM layer, 32 units and 12,416 params, dense layer: 12 units and 396 params. Total params in the model is 37,900.

Bidirectional LSTM Integrated with Dense Neural Classifier:
The output layer of my model is a Dense layer with num_classes neurons and a softmax activation function. Given that y_train dimensions, I have 12 categories. This output represents the probability distribution in 12 classes for each sample, due to softmax activation. The characteristics of the output are hierarchical, according to the softmax activation commonly used in multiclass classification problems. Loss function optimized in training is the categorical cross-entropy.

The framework remains exactly the same as the basic LSTM model, just with a change because of the introduction of Bidirectional LSTM layer. Bidirectional LSTM layer refers to estimating the parameters for both forward and backward passes, effectively doubling the number of parameters compared to the unidirectional LSTM layer. Here the return sequence is set to ‘True’. After this it is fed to the unidirectional LSTM where the return sequence is set to ‘False’, hence the output of this can be an input for the dense layers. The first Dense layer contains the parameters that connect the LSTM output to each of the 300 neurons, and the second Dense layer contains the parameters that connect these 300 neurons to each of the 12 output neurons, one for each class.

Bidirectional LSTM layer parameters: This layer provides double the standard LSTM parameters due to its bidirectional nature. The formula for each instruction is the same as the default LSTM: 4 * (units * input_dim + units^2 + units). In bidirectional LSTM, the outputs are connected for both directions. This means that the input scale of the second LSTM layer is not 128, but double, because the bidirectional LSTM gets output from the forward and backward layers. Second LSTM layer Parameters: This layer will have the parameters calculated is calculated in the same way as standard LSTM: there are 4 * (128 * 128 + 2 * 128^2 + 128). Dense Layer Parameters: The first dense layer will have parameters calculated by input_units * output_units + output_units (for biases). The input unit is 128 (output from the second LSTM), and the output unit is 300. Second dense layer parameters: Similarly, the parameters in this layer are considered as 300 input units (from the previous dense layer) and 12 output units (number of categories).

Summary:

Bidirectional LSTM Layer: Params - 165,888, 2nd LSTM Layer: Params - 197,120, 1st Dense Layer: Params - 38,700, 2nd Dense Layer: Params - 3612, Total Params - 405,320

4.5 GRU Based Encoder followed by Neural Network Classifier on Dataset 1 and Dataset 2:

1. **Data Preprocessing:** Same as data preprocessing discussed in section 4.4.

2. **Modeling Framework:** The input and output dimensions of the dataset, exactly resemble the section 4.4, with changes in the architecture of the model. So here I will discuss the model with respect to Dataset 1. The GRU based encoder is different from the encoder discussed in section 4.3 because this encoder can take the data in 3D format (batch_size (or no. of sequences), timesteps, features), hence capturing the temporal nature unlike encoder in section 4.3.

   Dimensionality Reduction Section: Input Level: The input level is defined using Input(shape=(X_train.shape[1], X_train.shape[2])). For my X_train with dimensions (14885, 20, 33), this translates to an input size of (20, 33), where 20 is the sequence length and 33 is the number of elements per sequence GRU Encoder (Dimensionality Reduction); : First GRU Layer GRU (64). , activation='relu', return_sequences=True) Processes input sequences. It has 64 units and returns the entire series of outputs (one for each time step). The second GRU layer GRU(16, activation='relu') continues to process the output from the first GRU layer. This level has 16 units and does not repeat the entire sequence, which means that output is returned only at the last time step. This is effectively a hidden representation of my space, reducing the sequence to a representation of 16 segments. Flatten Layer: The Flatten() layer is used to give the effect that the second GRU layer is flat. This is important because subsequent dense layers require a 1D array as input, not a 3D array from the GRU layer.

   Phase of classification: After dimensionality reduction, the network enters this phase: Classifier Layers of density: There are two dense layers (Dense(100, activation='selu'); and Dense(50, activation='selu'.)) with 100 and 50 nodes, respectively. These strata additionally handle slender items. For deeper networks, the "selu" (Scaled Exponential Linear Unit) activation function is employed, which can aid in output normalization. Layer of output: The output layer for the Dense(y_train.shape[1], activation='softmax') classification is the final layer. This layer assigns the input to one of the 12 classes since y_train.shape[1] equals 12. The "Softmax" function produces a probability distribution over 12 classes, making it appropriate for class classification. Model compilation: The model uses the ‘Adam’ optimizer, which is suitable for a wide range of problems. The 'categorical_crossentropy' loss function is best for multiclass classification tasks. The performance of the model will be evaluated on the basis of ‘accuracy’ metric.

   This format is particularly suitable for sequential data (such as time series or stories). The GRU layers effectively reduce the dimensionality of the input sequence, taking into account time dependence and critical features. The flat output is then given to dense layers for classification. This model is designed to classify each sequence into one of the 12 categories, making it useful for complex classification tasks where an understanding of the temporal dynamics of the data is important

Summary: 1st GRU Layer: Params - 19,008, 2nd GRU Layer: Params: 3,396, 1st Dense Layer: Params - 1,700, 2nd Dense Layer: 5,050 and Output layer: Params - 612. Total Params - 30,306.

5. Results and Discussions:

This section will present the precision values for each of the fault types, calculated by normalizing each diagonal element of the confusion matrix along the actual label axis. It will also discuss the accuracy values for each model, followed by mean of precision of each classification and mean of precision of only faulty modes. Rest of the analysis is attached in the appendix.

1. Only Logistic Regression:

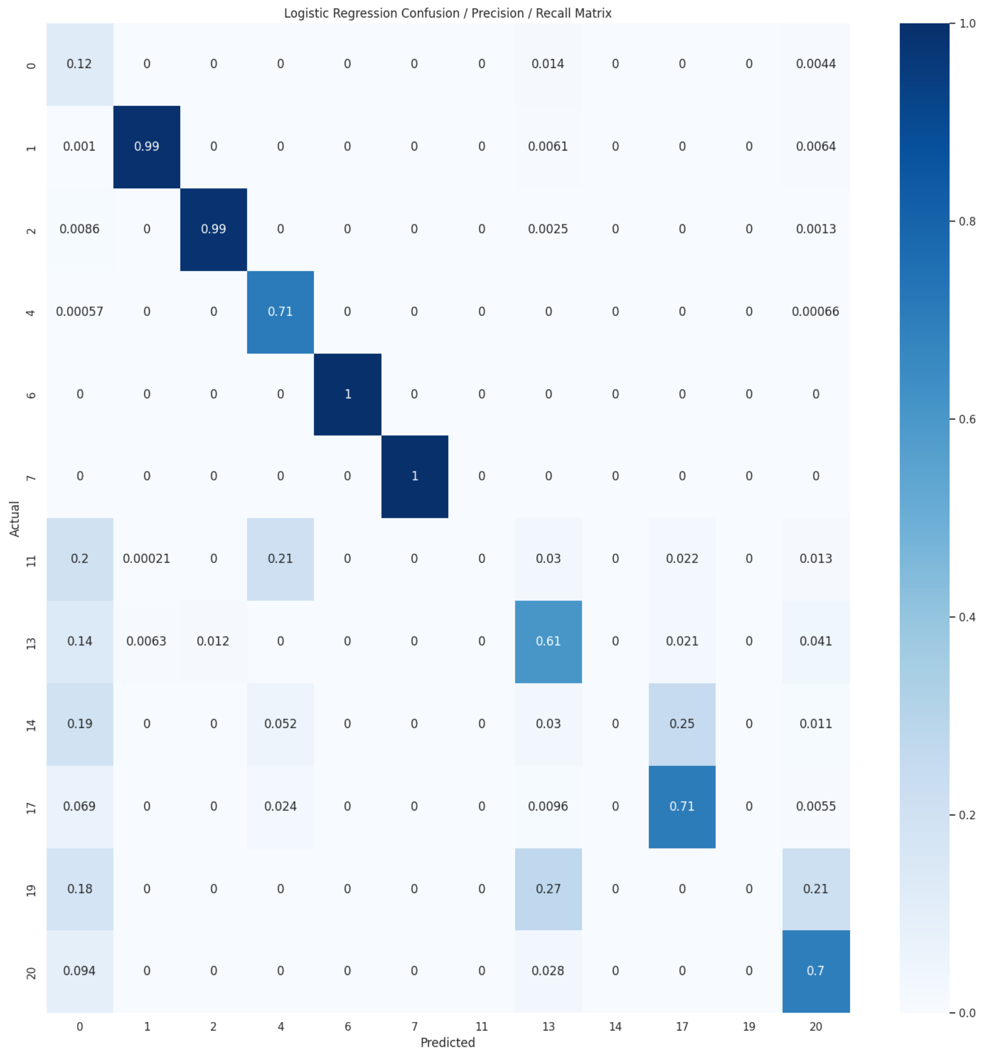


Fig 2: Precision Matrix for Logistic Regression on Dataset 1

2. PCA followed by Logistic Regression

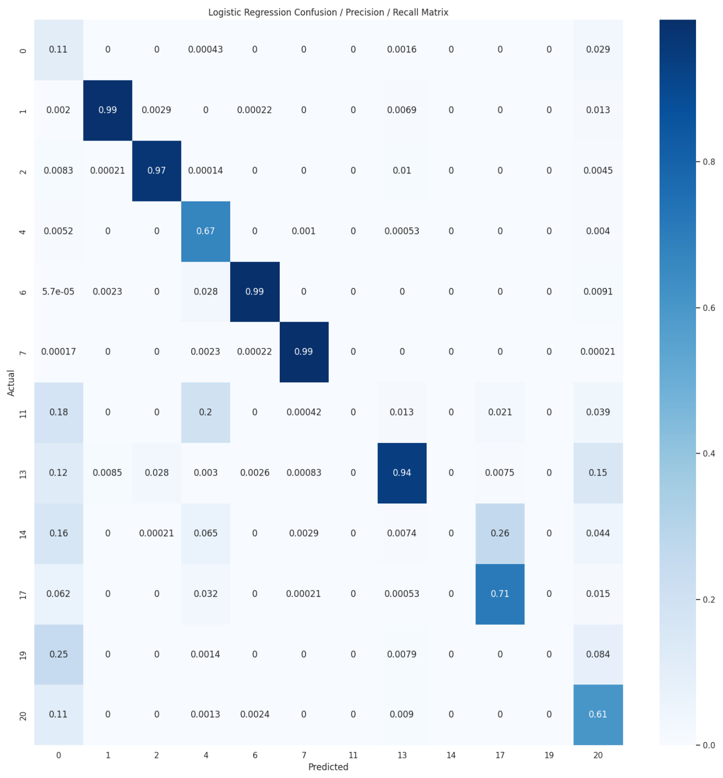


Fig 3: Precision Matrix for PCA Followed by Logistic Regression on Dataset 1

3. Dynamic PCA followed by Logistic Regression

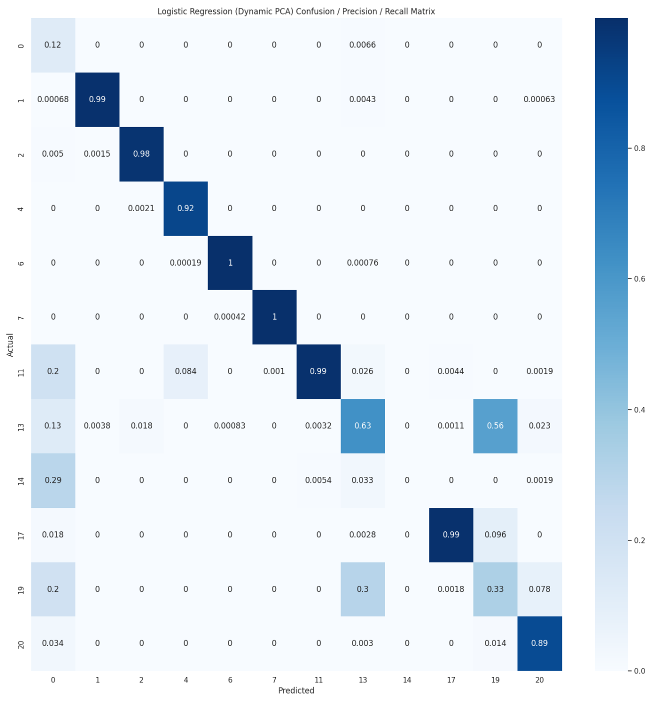


Fig 4: Precision Matrix for dynamic PCA Followed by Logistic Regression on Dataset 1

4. Random Forest Classifier:

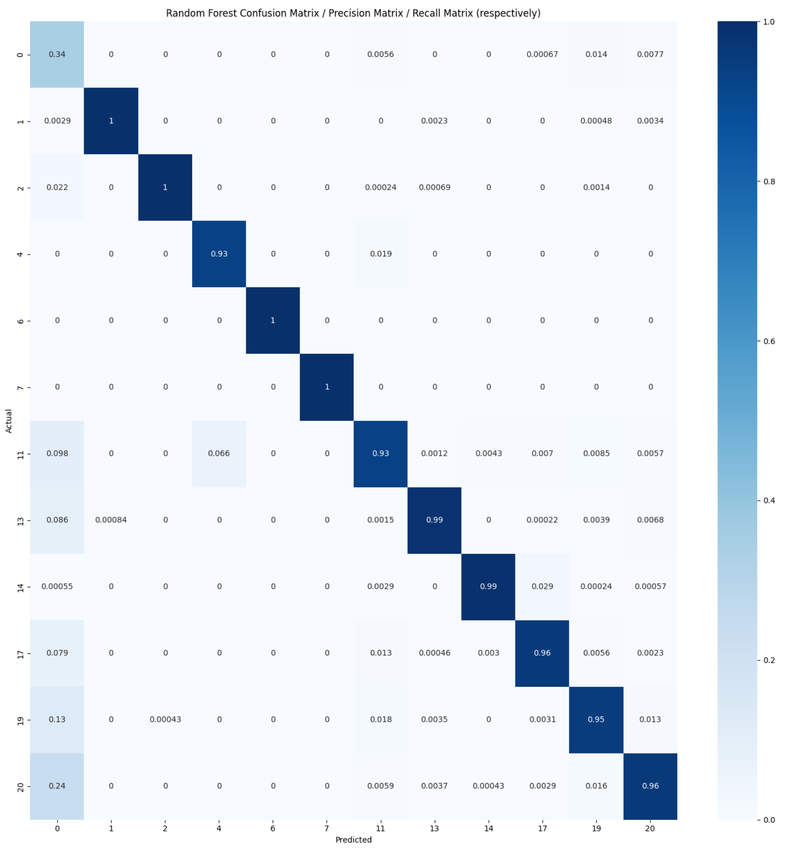
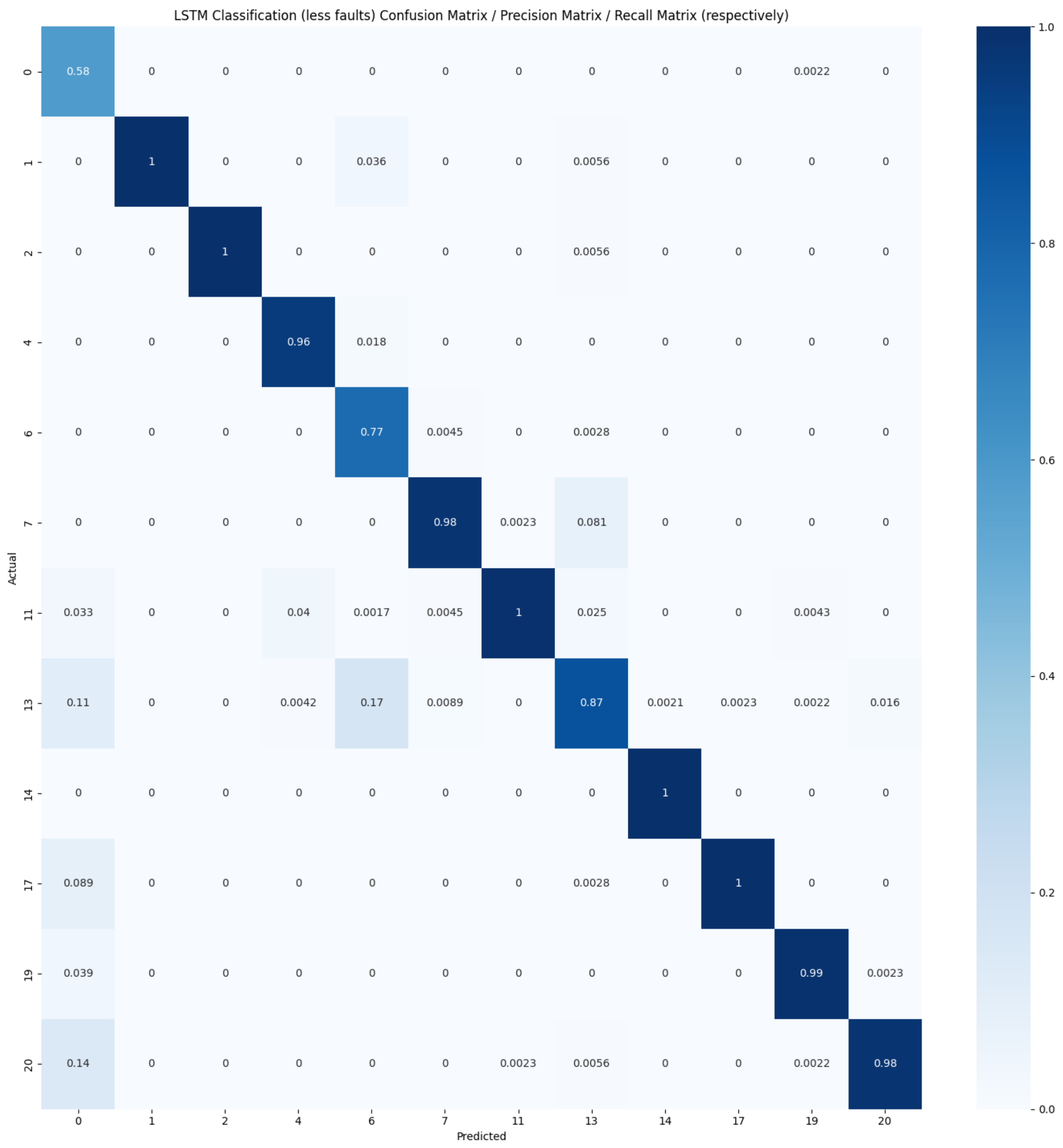
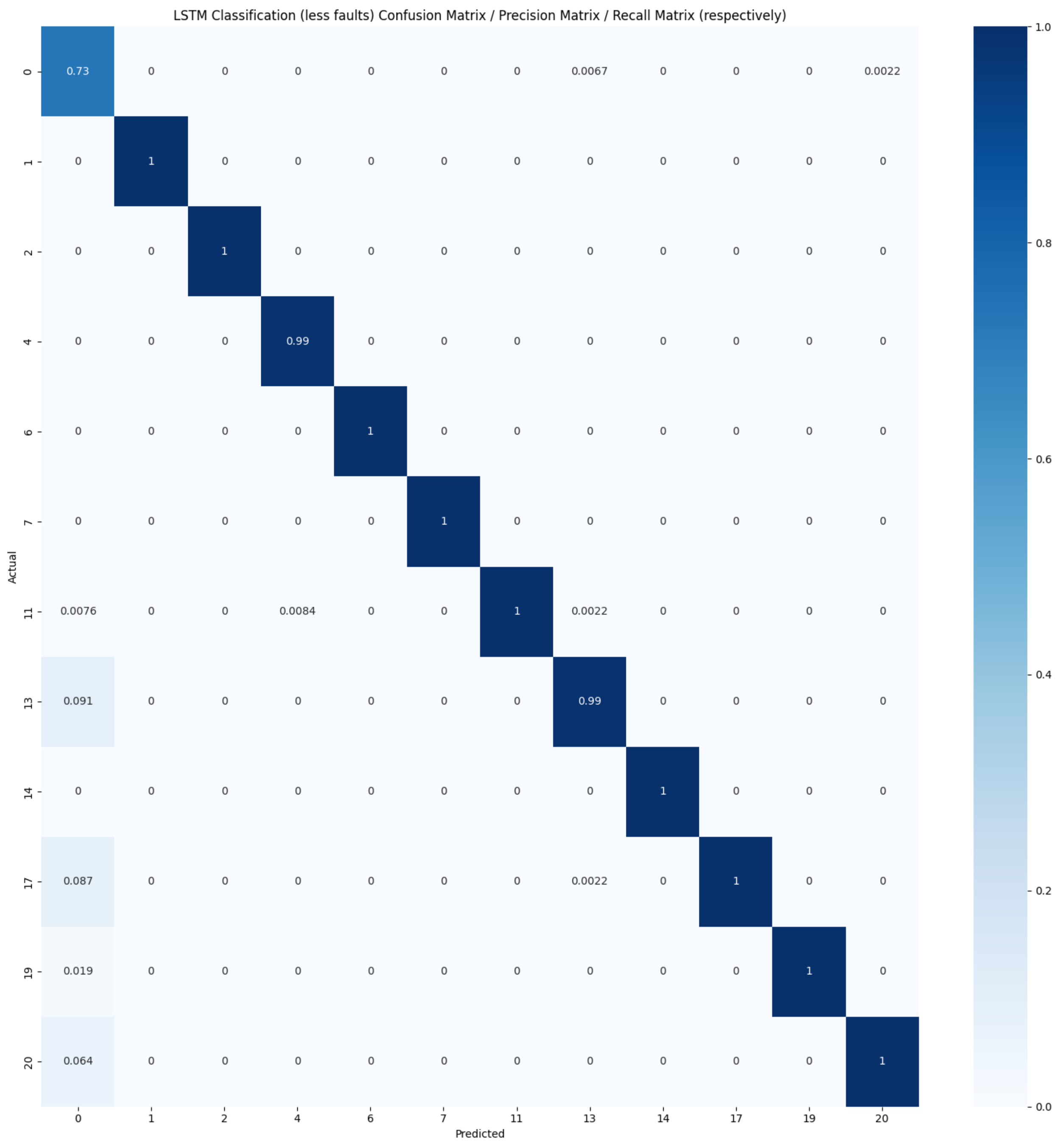


Fig 5: Precision Matrix for Random Forest Classifier on Dataset 1

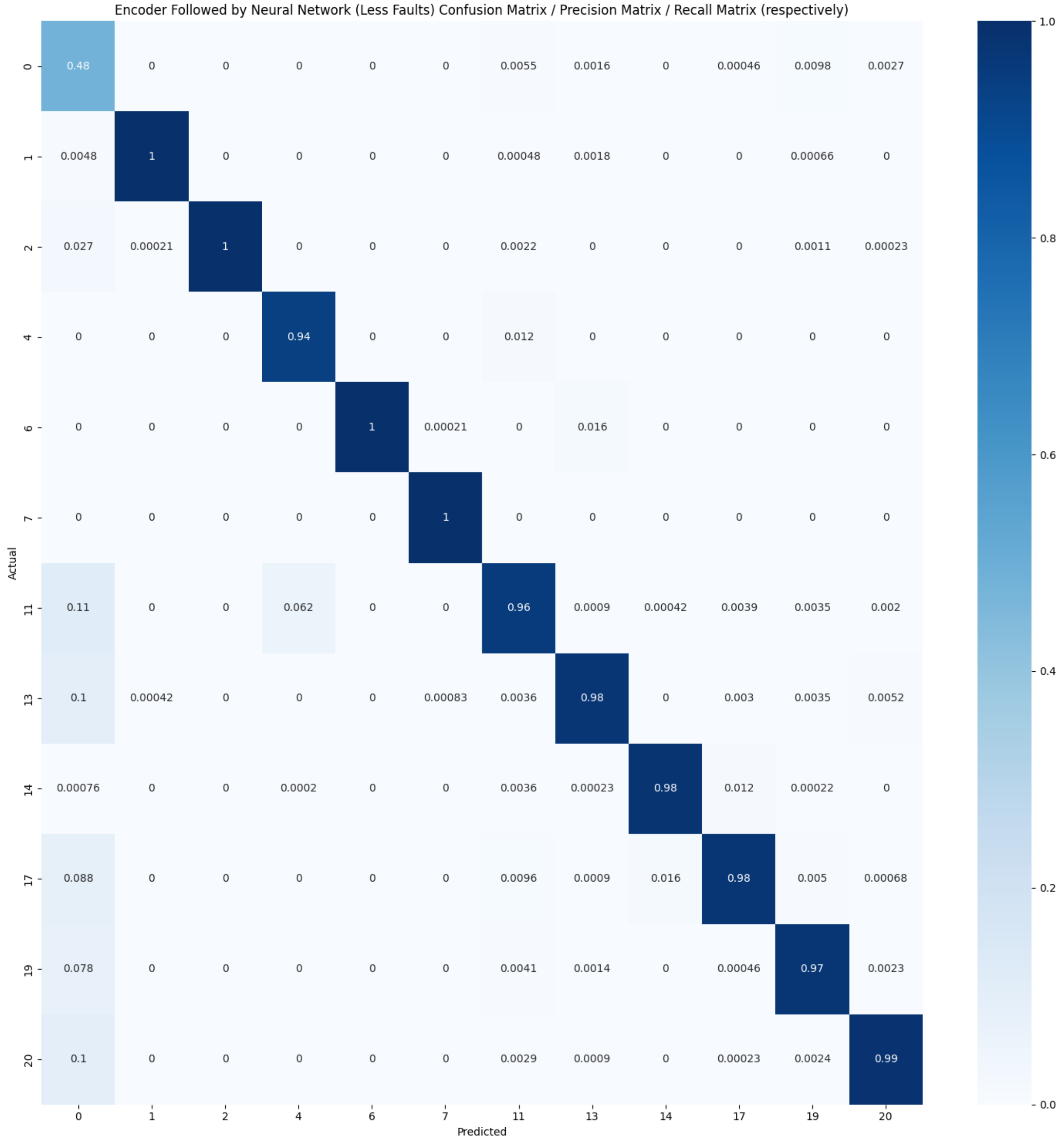
5. Various Deep Learning Methods on Dataset 1:

Following figures are attached in the order: Basic LSTM model, Bidirectional LSTM integrated with dense neural network, Encoder followed by Neural Network and GRU Based encoder with dense neural classifier

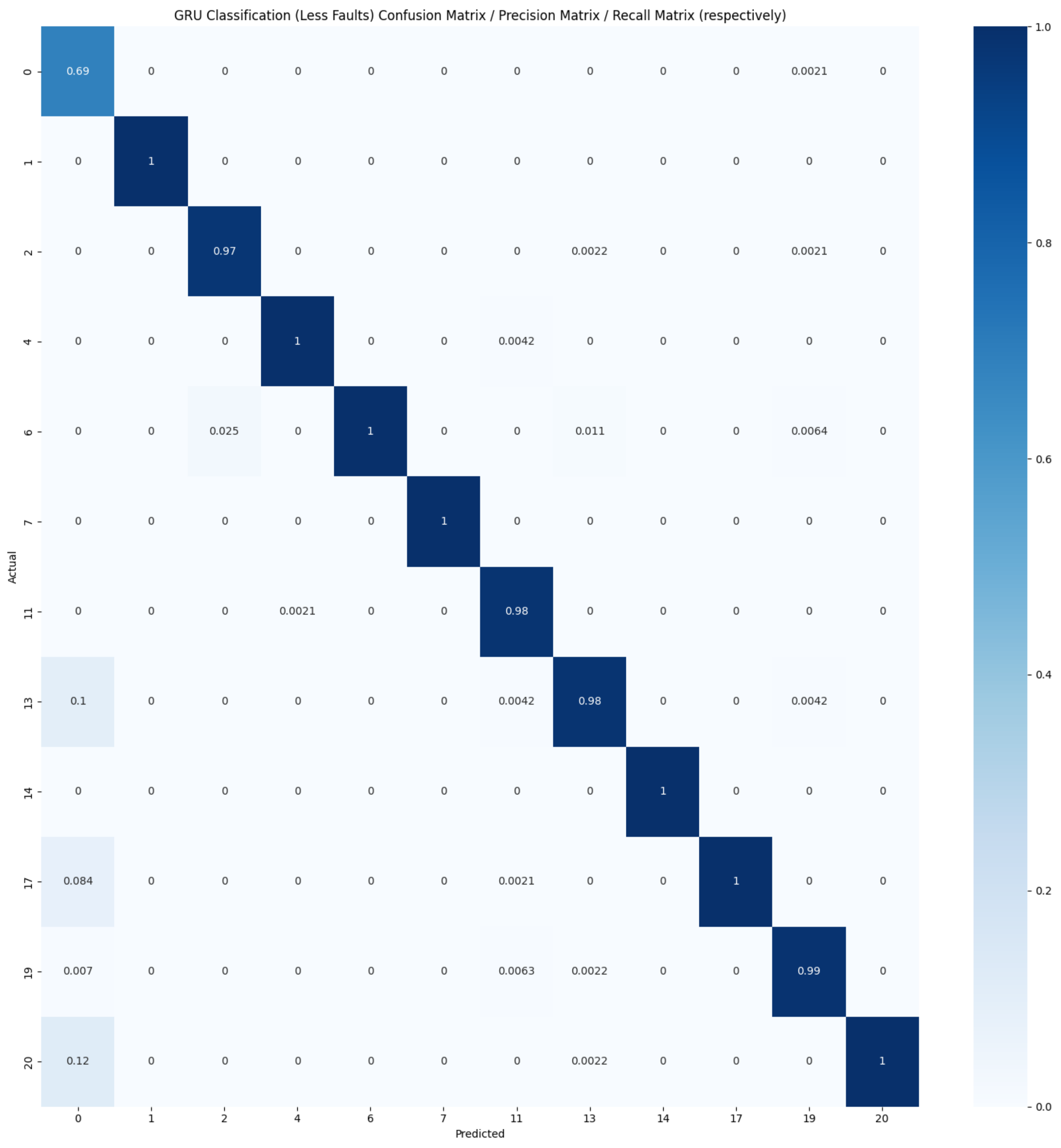

LSTM Classification (less faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)
Actual
Predicted
0 1 2 4 6 7 11 13 14 17 19 20
0: 0.58 0 0 0 0 0 0 0 0 0 0.0022 0
1: 0 1 0 0 0.036 0 0 0.0056 0 0 0 0
2: 0 0 1 0 0 0 0 0.0056 0 0 0 0
4: 0 0 0 0.96 0.018 0 0 0 0 0 0 0
6: 0 0 0 0 0.77 0.0045 0 0.0028 0 0 0 0
7: 0 0 0 0 0 0.98 0.0023 0.081 0 0 0 0
11: 0.033 0 0 0.04 0.0017 0.0045 1 0.025 0 0 0.0043 0
13: 0.11 0 0 0.0042 0.17 0.0089 0 0.87 0.0021 0.0023 0.0022 0.016
14: 0 0 0 0 0 0 0 0 1 0 0 0
17: 0.089 0 0 0 0 0 0 0.0028 0 1 0 0
19: 0.039 0 0 0 0 0 0 0 0 0 0.99 0.0023
20: 0.14 0 0 0 0 0 0.0023 0.0056 0 0 0.0022 0.98
1.0
0.8
0.6
0.4
0.2
0.0


LSTM Classification (less faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)
Actual
Predicted
0 1 2 4 6 7 11 13 14 17 19 20
0: 0.73 0 0 0 0 0 0 0.0067 0 0 0 0.0022
1: 0 1 0 0 0 0 0 0 0 0 0 0
2: 0 0 1 0 0 0 0 0 0 0 0 0
4: 0 0 0 0.99 0 0 0 0 0 0 0 0
6: 0 0 0 0 1 0 0 0 0 0 0 0
7: 0 0 0 0 0 1 0 0 0 0 0 0
11: 0.0076 0 0 0.0084 0 0 1 0.0022 0 0 0 0
13: 0.091 0 0 0 0 0 0 0.99 0 0 0 0
14: 0 0 0 0 0 0 0 0 1 0 0 0
17: 0.087 0 0 0 0 0 0 0.0022 0 1 0 0
19: 0.019 0 0 0 0 0 0 0 0 0 1 0
20: 0.064 0 0 0 0 0 0 0 0 0 0 1
1.0
0.8
0.6
0.4
0.2
0.0

Encoder Followed by Neural Network (Less Faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)

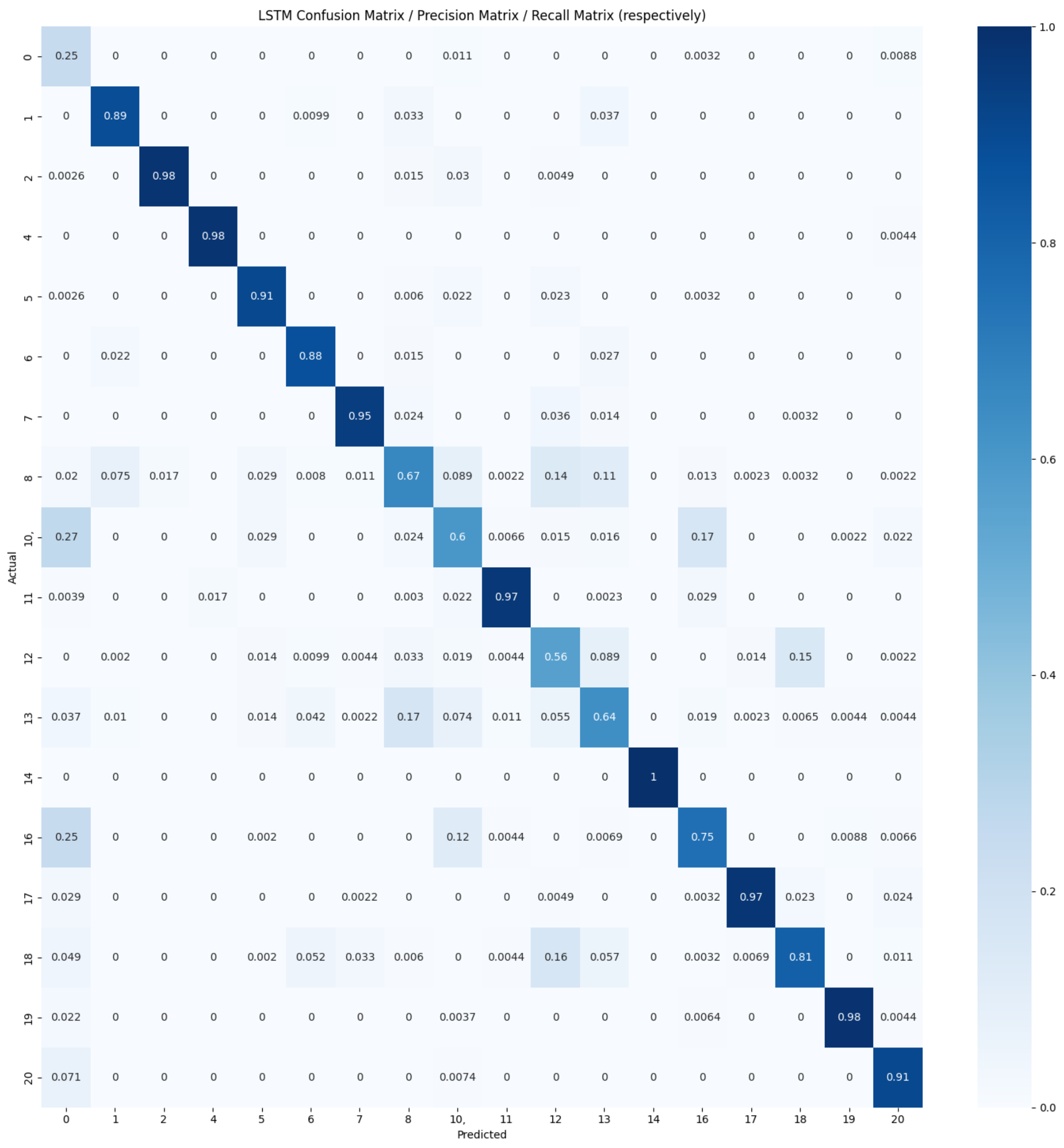


Actual \ Predicted	0	1	2	4	6	7	11	13	14	17	19	20
0	0.48	0	0	0	0	0	0.0055	0.0016	0	0.00046	0.0098	0.0027
1	0.0048	1	0	0	0	0	0.00048	0.0018	0	0	0.00066	0
2	0.027	0.00021	1	0	0	0	0.0022	0	0	0	0.0011	0.00023
4	0	0	0	0.94	0	0	0.012	0	0	0	0	0
6	0	0	0	0	1	0.00021	0	0.016	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0
11	0.11	0	0	0.062	0	0	0.96	0.0009	0.00042	0.0039	0.0035	0.002
13	0.1	0.00042	0	0	0	0.00083	0.0036	0.98	0	0.003	0.0035	0.0052
14	0.00076	0	0	0.0002	0	0	0.0036	0.00023	0.98	0.012	0.00022	0
17	0.088	0	0	0	0	0	0.0096	0.0009	0.016	0.98	0.005	0.00068
19	0.078	0	0	0	0	0	0.0041	0.0014	0	0.00046	0.97	0.0023
20	0.1	0	0	0	0	0	0.0029	0.0009	0	0.00023	0.0024	0.99

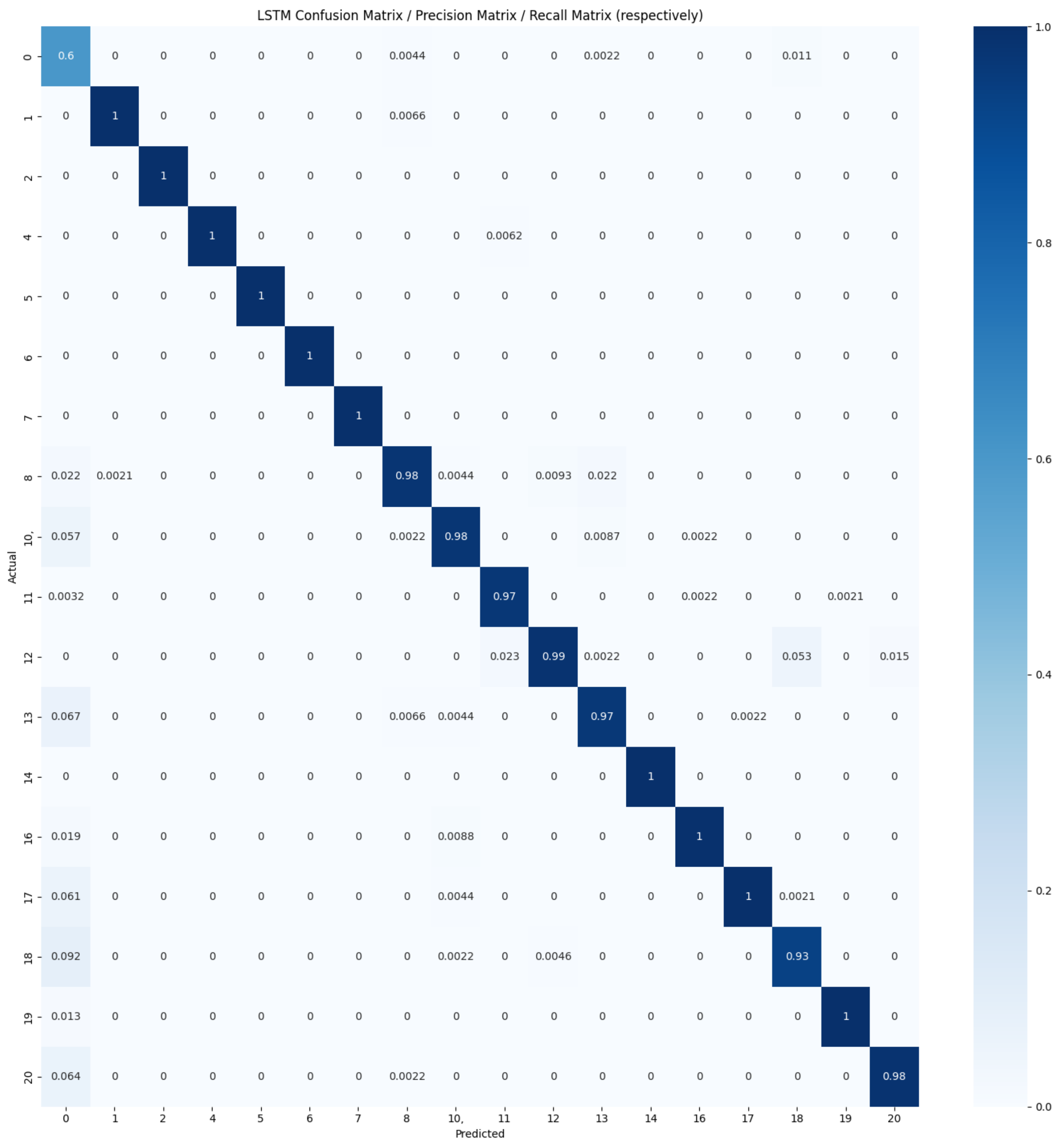

GRU Classification (Less Faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)
Actual
Predicted
0 1 2 4 6 7 11 13 14 17 19 20
0: 0.69 0 0 0 0 0 0 0 0 0 0.0021 0
1: 0 1 0 0 0 0 0 0 0 0 0 0
2: 0 0 0.97 0 0 0 0 0.0022 0 0 0.0021 0
4: 0 0 0 1 0 0 0.0042 0 0 0 0 0
6: 0 0 0.025 0 1 0 0 0.011 0 0 0.0064 0
7: 0 0 0 0 0 1 0 0 0 0 0 0
11: 0 0 0 0.0021 0 0 0.98 0 0 0 0 0
13: 0.1 0 0 0 0 0 0.0042 0.98 0 0 0.0042 0
14: 0 0 0 0 0 0 0 0 1 0 0 0
17: 0.084 0 0 0 0 0 0.0021 0 0 1 0 0
19: 0.007 0 0 0 0 0 0.0063 0.0022 0 0 0.99 0
20: 0.12 0 0 0 0 0 0 0.0022 0 0 0 1
1.0
0.8
0.6
0.4
0.2
0.0

6. Various Deep Learning Methods on Dataset 2:

Following figures are attached in the order: Basic LSTM model, Bidirectional LSTM integrated with dense neural network, Encoder followed by Neural Network and GRU Based encoder with dense neural classifier

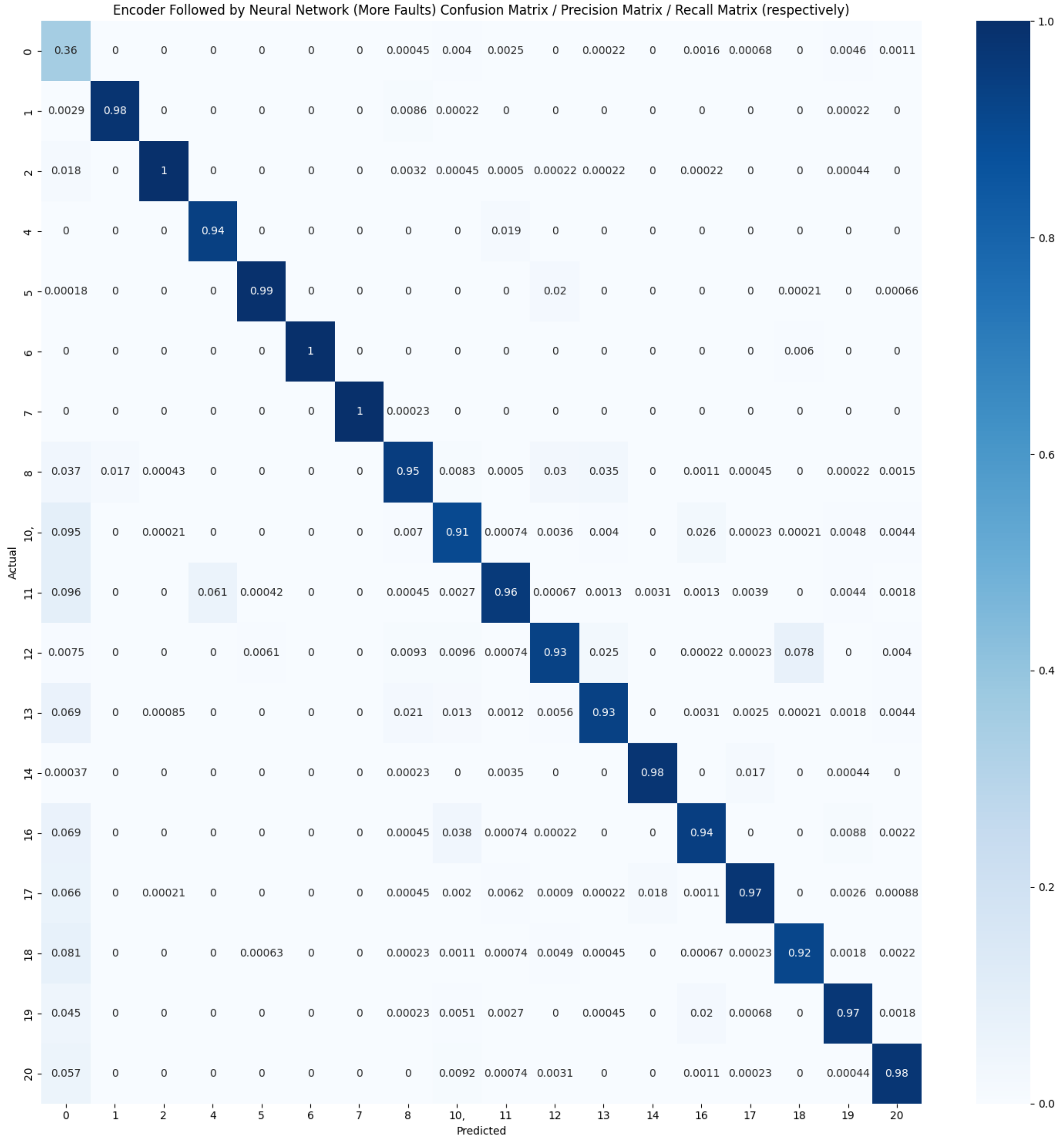


	0	1	2	4	5	6	7	8	10,	11	12	13	14	16	17	18	19	20
0	0.25	0	0	0	0	0	0	0	0.011	0	0	0	0	0.0032	0	0	0	0.0088
1	0	0.89	0	0	0	0.0099	0	0.033	0	0	0	0.037	0	0	0	0	0	0
2	0.0026	0	0.98	0	0	0	0	0.015	0.03	0	0.0049	0	0	0	0	0	0	0
4	0	0	0	0.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0044
5	0.0026	0	0	0	0.91	0	0	0.006	0.022	0	0.023	0	0	0.0032	0	0	0	0
6	0	0.022	0	0	0	0.88	0	0.015	0	0	0	0.027	0	0	0	0	0	0
7	0	0	0	0	0	0	0.95	0.024	0	0	0.036	0.014	0	0	0	0.0032	0	0
8	0.02	0.075	0.017	0	0.029	0.008	0.011	0.67	0.089	0.0022	0.14	0.11	0	0.013	0.0023	0.0032	0	0.0022
10,	0.27	0	0	0	0.029	0	0	0.024	0.6	0.0066	0.015	0.016	0	0.17	0	0	0.0022	0.022
11	0.0039	0	0	0.017	0	0	0	0.003	0.022	0.97	0	0.0023	0	0.029	0	0	0	0
12	0	0.002	0	0	0.014	0.0099	0.0044	0.033	0.019	0.0044	0.56	0.089	0	0	0.014	0.15	0	0.0022
13	0.037	0.01	0	0	0.014	0.042	0.0022	0.17	0.074	0.011	0.055	0.64	0	0.019	0.0023	0.0065	0.0044	0.0044
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
16	0.25	0	0	0	0.002	0	0	0	0.12	0.0044	0	0.0069	0	0.75	0	0	0.0088	0.0066
17	0.029	0	0	0	0	0	0.0022	0	0	0	0.0049	0	0	0.0032	0.97	0.023	0	0.024
18	0.049	0	0	0	0.002	0.052	0.033	0.006	0	0.0044	0.16	0.057	0	0.0032	0.0069	0.81	0	0.011
19	0.022	0	0	0	0	0	0	0	0.0037	0	0	0	0	0.0064	0	0	0.98	0.0044
20	0.071	0	0	0	0	0	0	0	0.0074	0	0	0	0	0	0	0	0	0.91

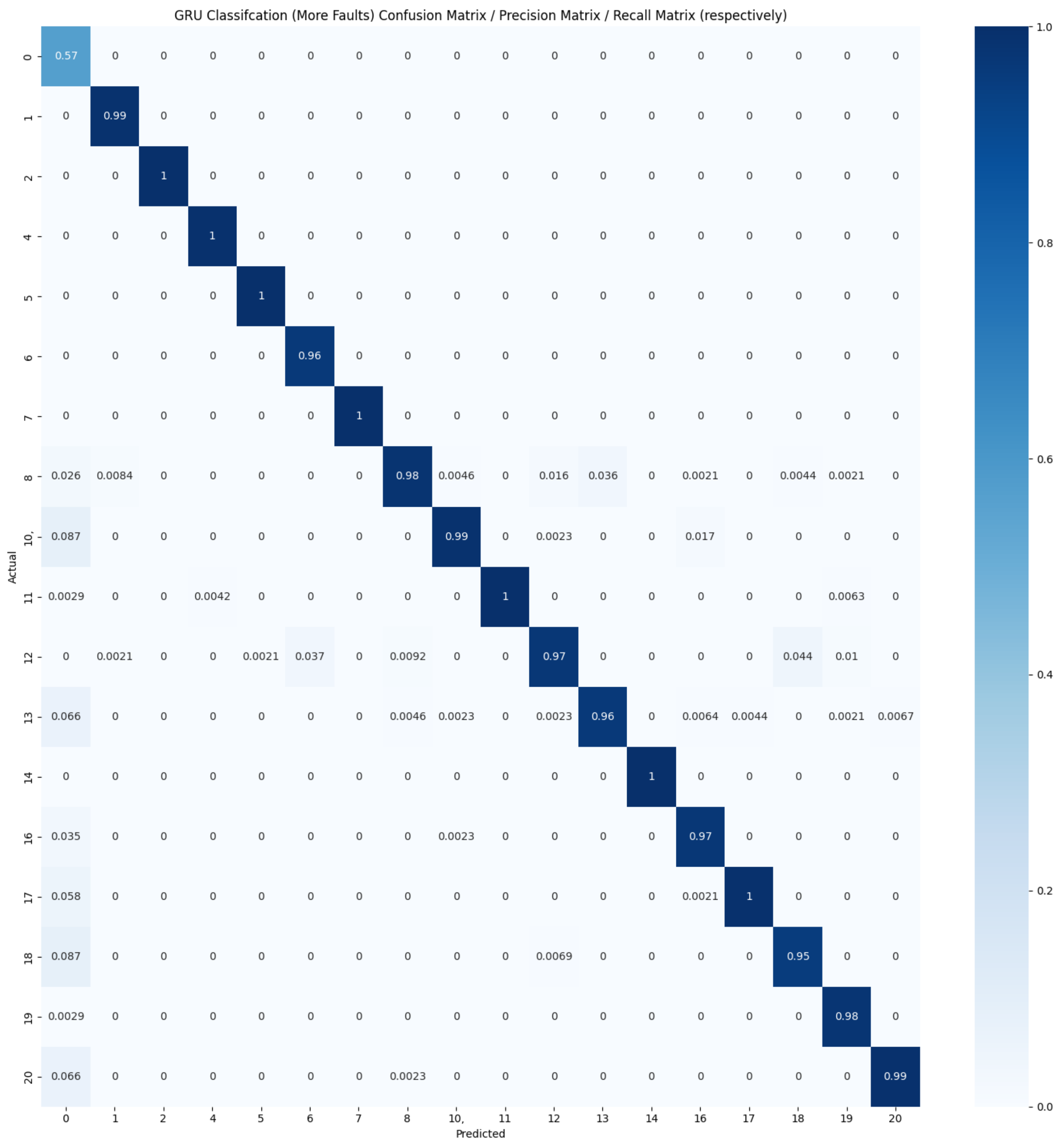


	0	1	2	4	5	6	7	8	10,	11	12	13	14	16	17	18	19	20
0	0.6	0	0	0	0	0	0	0.0044	0	0	0	0.0022	0	0	0	0.011	0	0
1	0	1	0	0	0	0	0	0.0066	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0.0062	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
8	0.022	0.0021	0	0	0	0	0	0.98	0.0044	0	0.0093	0.022	0	0	0	0	0	0
10,	0.057	0	0	0	0	0	0	0.0022	0.98	0	0	0.0087	0	0.0022	0	0	0	0
11	0.0032	0	0	0	0	0	0	0	0	0.97	0	0	0	0.0022	0	0	0.0021	0
12	0	0	0	0	0	0	0	0	0	0.023	0.99	0.0022	0	0	0	0.053	0	0.015
13	0.067	0	0	0	0	0	0	0.0066	0.0044	0	0	0.97	0	0	0.0022	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
16	0.019	0	0	0	0	0	0	0	0.0088	0	0	0	0	1	0	0	0	0
17	0.061	0	0	0	0	0	0	0	0.0044	0	0	0	0	0	1	0.0021	0	0
18	0.092	0	0	0	0	0	0	0	0.0022	0	0.0046	0	0	0	0	0.93	0	0
19	0.013	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
20	0.064	0	0	0	0	0	0	0.0022	0	0	0	0	0	0	0	0	0	0.98

Encoder Followed by Neural Network (More Faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)



Actual \ Predicted	0	1	2	4	5	6	7	8	10,	11	12	13	14	16	17	18	19	20
0	0.36	0	0	0	0	0	0	0.00045	0.004	0.0025	0	0.00022	0	0.0016	0.00068	0	0.0046	0.0011
1	0.0029	0.98	0	0	0	0	0	0.0086	0.00022	0	0	0	0	0	0	0	0.00022	0
2	0.018	0	1	0	0	0	0	0.0032	0.00045	0.0005	0.00022	0.00022	0	0.00022	0	0	0.00044	0
4	0	0	0	0.94	0	0	0	0	0	0.019	0	0	0	0	0	0	0	0
5	0.00018	0	0	0	0.99	0	0	0	0	0	0.02	0	0	0	0	0.00021	0	0.00066
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.006	0	0
7	0	0	0	0	0	0	1	0.00023	0	0	0	0	0	0	0	0	0	0
8	0.037	0.017	0.00043	0	0	0	0	0.95	0.0083	0.0005	0.03	0.035	0	0.0011	0.00045	0	0.00022	0.0015
10,	0.095	0	0.00021	0	0	0	0	0.007	0.91	0.00074	0.0036	0.004	0	0.026	0.00023	0.00021	0.0048	0.0044
11	0.096	0	0	0.061	0.00042	0	0	0.00045	0.0027	0.96	0.00067	0.0013	0.0031	0.0013	0.0039	0	0.0044	0.0018
12	0.0075	0	0	0	0.0061	0	0	0.0093	0.0096	0.00074	0.93	0.025	0	0.00022	0.00023	0.078	0	0.004
13	0.069	0	0.00085	0	0	0	0	0.021	0.013	0.0012	0.0056	0.93	0	0.0031	0.0025	0.00021	0.0018	0.0044
14	0.00037	0	0	0	0	0	0	0.00023	0	0.0035	0	0	0.98	0	0.017	0	0.00044	0
16	0.069	0	0	0	0	0	0	0.00045	0.038	0.00074	0.00022	0	0	0.94	0	0	0.0088	0.0022
17	0.066	0	0.00021	0	0	0	0	0.00045	0.002	0.0062	0.0009	0.00022	0.018	0.0011	0.97	0	0.0026	0.00088
18	0.081	0	0	0	0.00063	0	0	0.00023	0.0011	0.00074	0.0049	0.00045	0	0.00067	0.00023	0.92	0.0018	0.0022
19	0.045	0	0	0	0	0	0	0.00023	0.0051	0.0027	0	0.00045	0	0.02	0.00068	0	0.97	0.0018
20	0.057	0	0	0	0	0	0	0	0.0092	0.00074	0.0031	0	0	0.0011	0.00023	0	0.00044	0.98


GRU Classifcation (More Faults) Confusion Matrix / Precision Matrix / Recall Matrix (respectively)
Actual
Predicted
0 1 2 4 5 6 7 8 10, 11 12 13 14 16 17 18 19 20
0: 0.57 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1: 0 0.99 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
6: 0 0 0 0 0 0.96 0 0 0 0 0 0 0 0 0 0 0 0
7: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
8: 0.026 0.0084 0 0 0 0 0 0.98 0.0046 0 0.016 0.036 0 0.0021 0 0.0044 0.0021 0
10,: 0.087 0 0 0 0 0 0 0 0.99 0 0.0023 0 0 0.017 0 0 0 0
11: 0.0029 0 0 0.0042 0 0 0 0 0 1 0 0 0 0 0 0 0.0063 0
12: 0 0.0021 0 0 0.0021 0.037 0 0.0092 0 0 0.97 0 0 0 0 0.044 0.01 0
13: 0.066 0 0 0 0 0 0 0.0046 0.0023 0 0.0023 0.96 0 0.0064 0.0044 0 0.0021 0.0067
14: 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
16: 0.035 0 0 0 0 0 0 0 0.0023 0 0 0 0 0.97 0 0 0 0
17: 0.058 0 0 0 0 0 0 0 0 0 0 0 0 0.0021 1 0 0 0
18: 0.087 0 0 0 0 0 0 0 0 0 0.0069 0 0 0 0 0.95 0 0
19: 0.0029 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.98 0
20: 0.066 0 0 0 0 0 0 0.0023 0 0 0 0 0 0 0 0 0 0.99
1.0
0.8
0.6
0.4
0.2
0.0

Tabulated Results:

Here by in Table 1 and Table 2 I have discussed the mentioned metrics for Dataset 1 (Less Faults) and Dataset 2 (More faults) respectively:

DATASET 1								
Metrics	**Logistic Regression**	**PCA Followed by Logistic Regression**	**Dynamic PCA Followed by Logistic Regression**	**Random Forest Classifier**	**Basic LSTM**	**Encoder + ANN**	**GRU Based Encoder with Dense Neural Classifier**	**Bidirectional LSTM + Dense Neural Classifier**
Accuracy	0.6312043796	0.6059124088	0.6929816556	0.9115510949	0.9310601826	0.9452919708	0.9767095211	0.9849077697
Mean Precision of all Categories	0.57	0.582	0.737	0.922	0.927	0.94	0.968	0.976
Mean of precision of faulty modes only	0.61	0.625	0.793	0.974	0.958	0.981	0.993	0.998
Precision of normal mode only	0.12	0.11	0.12	0.34	0.58	0.48	0.69	0.73

Table 1: Results of various metrics for DATASET 1

DATSET 2				
Metrics	**Basic LSTM**	**Encoder + ANN**	**GRU Based Encoder with Dense Neural Classifier**	**Bidirectional LSTM + Dense Neural Classifier**
Accuracy	0.8012703066	0.9226196172	0.967753756	0.9725174056
Mean Precision of all Categories	0.817	0.928	0.962	0.966
Mean of precision of faulty modes only	0.851	0.962	0.985	0.987
Precision of normal mode only	0.25	0.36	0.57	0.6

Table 2: Results for various metrics for DATASET 2

After the analysis the best model for both DATASET 1 and DATASET 2 is Bidirectional LSTM + Dense Neural Classifier, and the worst is logistic regression. These results are in line with the understanding from Figure 1, that suggested non linear classifiers would be better than linear classifiers. Between dynamic PCA followed by logistic regression and PCA followed by logistic regression, the former shows better results because the lagged feature matrix captures the temporal nature of this time series dataset.

Subsequently it can be observed that GRU based encoder is a better option than 2D input encoder, because GRU based encoder takes the data in the format of (samples or number of sequence, timesteps, features). This format enables the model to take the temporal nature of the data into consideration. Random forest classifier works better than logistic regression because of its non linear nature and capacity to divide the space according to the features using decision tree algorithms.

Overall it can also be observed for all the Deep learning methods that DATASET 1 has better results than DATASET 2, because DATASET 1 excludes many faults that have a similar trend like the normal mode[1]. DATASET 2 including those faults that have a similar trend like the normal mode, costs it in the precision of 'Normal Mode' classification.

From the precision matrix it can be concluded that each model has a low precision in classifying the normal mode compared to the faulty modes. This has a positive and negative to it. Being able to precisely classify faulty modes results in overcoming the dangers corresponding to the faults, but at the same time having a less precision for 'Normal mode' means there could be many false alarms, as the true value of 'Normal Mode' is being classified into some other faulty type.

<u>6. References:</u>

1. S. Zheng and J. Zhao, "A new unsupervised data mining method based on the stacked autoencoder for chemical process fault diagnosis," Computers & Chemical Engineering, vol. 135, p. 106755, 2020.
2. Mihajlo Grbovic et.al "Cold start approach for data — driven fault detection", IEEE Transactions on Industrial Informatics, November 2013.
3. Mohamed Bin Shams et .al "Fault Detection using CUMSUM based techniques with application to the Tennessee Eastman Process", Proceedings of the 9th International Symposium on Dynamics and Control of Process Systems.
4. Danfeng Xie et .al " Deep Learning in Visual Computing and Signal Processing", Review Article, Applied Computation Intelligence and Soft computing, Volume 2017,Article ID 1320780.

<u>7. Appendix:</u>

1. The link includes the pdf file for 5 google colab notebooks, that discusses Downsampling, Logistic Regression, Random Forest Classifier, LSTM, GRU and Encoder + Neural Network: https://drive.google.com/drive/folders/1gZc-OMWUyhzMAywhRls2ukbf2_I6ynSp?usp=sharing