DEPARTMENT OF

MECHANICAL ENGINEERING

FINAL YEAR PROJECT REPORT

Project Title:

Deep Learning-Based Fault Diagnosis in Nuclear Power Plant

Chinese Title: NIL

Student Name: Leung Ming Haang

Chinese Name: 梁銘桁

Student No.: 55237436

Major: Nuclear and Risk Engineering

Supervisor Name: Prof. Min XIE

Submission Date: (29/04/2022)

Abstract

Loss of coolant flow accidents (LOCA) is a type of failure event for a nuclear reactor, generally initiated by either mechanical or electrical failure. The loss of coolant flow in a nuclear reactor significantly decreases the heat removal performance of the core, reactor core will be damaged and jeopardize tons of people. Detecting failure events in nuclear power plants are always challenging since thousands of parameters are required to keep tracking on. Therefore, forecast the accident through abnormal parameters of the system is crucial for risk and safety analysis.

This study presents a deep learning-based method for fault diagnosis of a Reactor Coolant System in nuclear power plant. With use of neural network model, complicated parameters inside the reactor coolant system become readable for the computer. Distinct events inside the system could understand as multi-class classification problem by the computer.

1. Introduction

First, thanks for the foundation works in the area of convolution neural networks by Yann LeCun and the collaborators in 1998 [1]. At that time, they build the networks for automatically recognizing handwritten digits. Also in 2012, a great breakthrough by Geoffrey E. Hinton with his PHD student created the seven-layer convolutional neural network well-known as SuperVision (AlexNet) in the ImageNet Competition [2]. Their innovation makes fault diagnosis in nuclear power plants system more accurate and efficient. With the advanced technology and increasing demand of choosing cost-effectively with high-capacity factor to operate the power plants, the fault detection and diagnosis has become a popular method to enhance safety, reliability, and availability of nuclear power plants [3]. Plenty of fault and diagnosis methods have been demonstrated in identify the failures of nuclear power plant system over the past several decades. Generally, methods which were widely applicated could be divided into two types. Model-based methods and Model-free methods.

2. Literature Review

A Model-based method describe the power plant system by mathematical model. Faults normally detected by finding the difference between observing behavior and predicting calculation. Research combing fuzzy logic and neural networks have been used to detect fault of steam generator [4]. By evaluate the residuals and simulate the time and space of the fault. The generation of residuals signals is the fundamental question of this approach. Accuracy of the model would be crucial since fault information would lose by low performance. As the traditional model-based methods are using linear systems with less complexity, it is not available for the real system. Non-linear methods such as neural networks become a better choice to operate the simulation.

While Model-free method describe the system by the correlation of the data, analyzing the relationship, training the data for prediction. Research with support vector machine have been used to monitoring machine conditions [5]. As neural network is good at pattern recognition, monitoring real-time condition by neural network would be a good choice undoubtedly. Traditional data-driven learning approach is to minimize error by training the data. While Support vector machine is based on the principle of structural risk minimization, in order to have better generalization of the data. Moreover, it could handle plenty of features due to each dimension of classified vectors do not strongly affecting each other.

In this paper, I propose using Convolutional Neural Network (CNN) to classify the hidden pattern behind the parameters of reactor coolant system. CNN is one of the powerful neural network models and is applicated to abnormality diagnosis in a nuclear power plant system [6]. Moreover, with the help of TensorFlow, CNN application become more user-friendly and available for me to implement training and inference of deep neural network in zero cost. TensorFlow is a free and open-source library for machine learning and artificial intelligence application. It supports a wide variety of programming languages, and I would be using Python as the programming language to do the analysis. Python is a programming language and have been widely applicated in machine learning and artificial intelligence either.

3. Methodology

The deep-learning approach of LOCA classification would be executed on Reactor Coolant System (RCS) of a Pressurized Water Reactor (PWR). GSE GPWR Generic Nuclear Simulator would generate the data. Convolutional Neural Network under the logic of TensorFlow.Keras library are used to recognize the pattern of malfunction of Reactor Coolant System and finish the classification of the malfunction.

3.1. GSE GPWR Generic Nuclear Simulator

The data were generated by GSE GPWR Generic Nuclear Simulator located at LAU-5201, City University of Hong Kong. It is based on a commercial 3-loop Pressurized Water Reactor designed by Westinghouse in the USA, which is similar to the system of Daya Bay Nuclear Power Station [6].

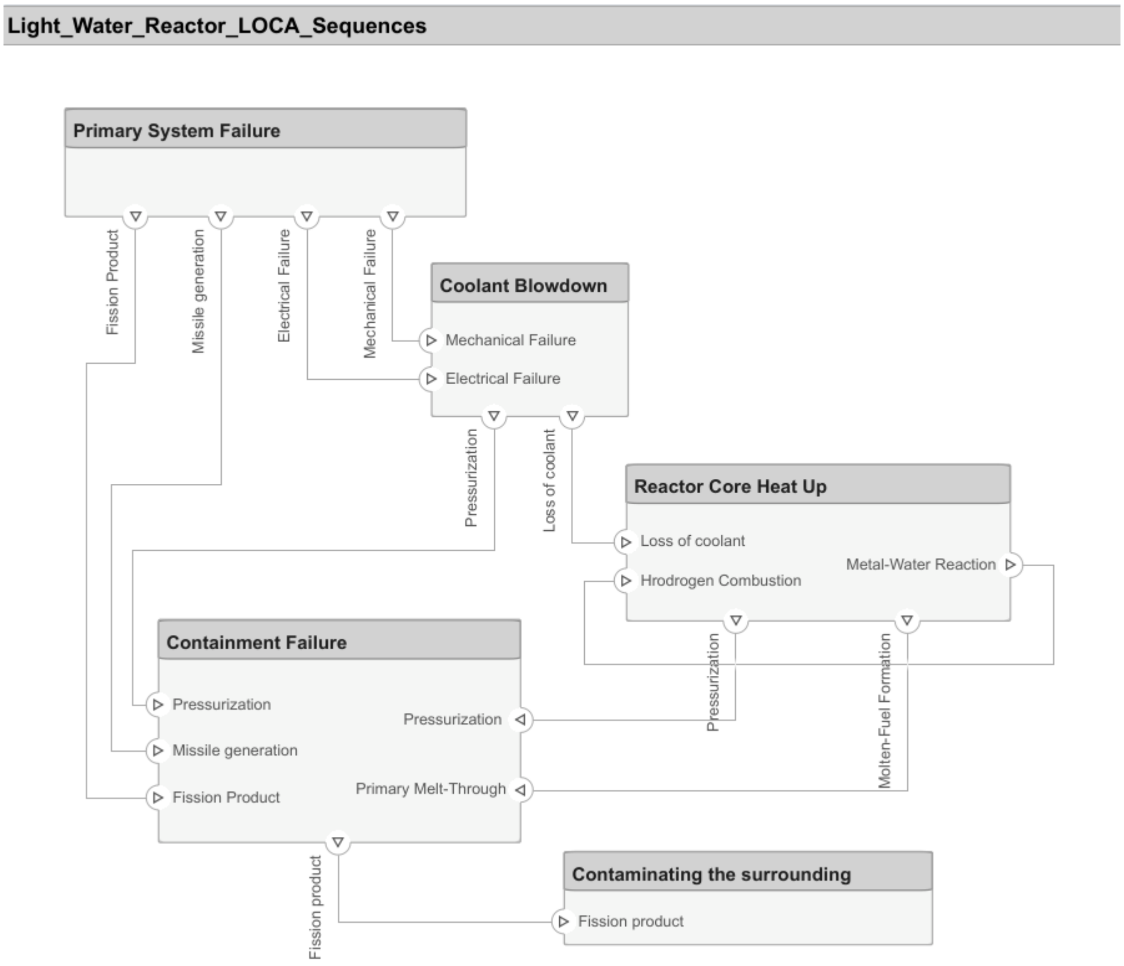


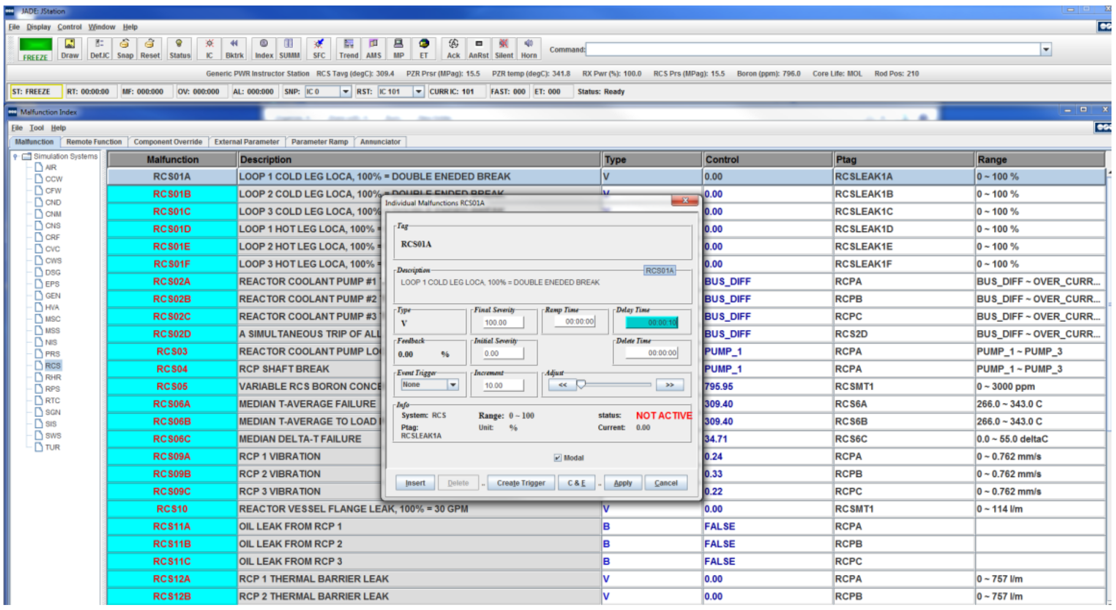
Fig. 1. Loss-of-coolant accident sequences for light-water reactors

3.1.1. Loss-of-coolant accident

Fig. 1 shows the Loss-of-coolant accident sequences for light-water reactors [7]. Loss-of-coolant flow accidents are caused by either mechanical or electrical failure. Normally, the PWR reactor coolant system would carry heat from the reactor core to the secondary coolant system through external primary coolant loops. During the LOCA event, a loss of coolant flow through the reactor will decrease the system ability to remove reactor core heat. Large amount of fluid mass and energy are released through the coolant pipe break into the containment. Tons of parameters are very volatile and providing information for us to track down what it is going on inside the reactor.

3.1.2. Data generation

As there are too many scenarios could possibly during LOCA event, the simulator provided a malfunction index list for us to run through different scenario, with the parameters I have chosen, the simulator is available to export an excel table to monitor all these parameters starting from time = 0.


JADE: JStation
File Display Control Window Help
FREEZE Draw Def.IC Snap Reset Status IC Bktrk Index SUMM SFC Trend AMS MP ET Ack AnRst Silent Horn Command:
Malfunction Index
File Tool Help
Malfunction Remote Function Component Override External Parameter Parameter Ramp Annunciator
Simulation Systems
AIR
CCW
CFW
CND
CNM
CNS
CRF
CVC
CWS
DSG
EPS
GEN
HVA
MSC
MSS
NIS
PRS
RCS
RHR
RPS
RTC
SGN
SIS
SWS
TUR
Insert Delete Create Trigger C & E Apply Cancel

Generic PWR Instructor Station RCS Tavg (degC): 309.4 PZR Prsr (MPag): 15.5 PZR temp (degC): 341.8 RX Pwr (%): 100.0 RCS Prs (MPag): 15.5 Boron (ppm): 796.0 Core Life: MOL Rod Pos: 210

ST: FREEZE RT: 00:00:00 MF: 000:000 OV: 000:000 AL: 000:000 SNP: IC 0 RST: IC 101 CURR IC: 101 FAST: 000 ET: 000 Status: Ready

Malfunction	Description	Type	Control	Ptag	Range
RCS01A	LOOP 1 COLD LEG LOCA, 100% = DOUBLE ENEDED BREAK	V	0.00	RCSLEAK1A	0 ~ 100 %
RCS01B	LOOP 2 COLD LEG LOCA, 100%	V	0.00	RCSLEAK1B	0 ~ 100 %
RCS01C	LOOP 3 COLD LEG LOCA, 100%		0.00	RCSLEAK1C	0 ~ 100 %
RCS01D	LOOP 1 HOT LEG LOCA, 100%		0.00	RCSLEAK1D	0 ~ 100 %
RCS01E	LOOP 2 HOT LEG LOCA, 100%		0.00	RCSLEAK1E	0 ~ 100 %
RCS01F	LOOP 3 HOT LEG LOCA, 100%		0.00	RCSLEAK1F	0 ~ 100 %
RCS02A	REACTOR COOLANT PUMP #1		BUS_DIFF	RCPA	BUS_DIFF ~ OVER_CURR...
RCS02B	REACTOR COOLANT PUMP #2		BUS_DIFF	RCPB	BUS_DIFF ~ OVER_CURR...
RCS02C	REACTOR COOLANT PUMP #3		BUS_DIFF	RCPC	BUS_DIFF ~ OVER_CURR...
RCS02D	A SIMULTANEOUS TRIP OF ALL		BUS_DIFF	RCS2D	BUS_DIFF ~ OVER_CURR...
RCS03	REACTOR COOLANT PUMP LO		PUMP_1	RCPA	PUMP_1 ~ PUMP_3
RCS04	RCP SHAFT BREAK		PUMP_1	RCPA	PUMP_1 ~ PUMP_3
RCS05	VARIABLE RCS BORON CONCE		795.95	RCSMT1	0 ~ 3000 ppm
RCS06A	MEDIAN T-AVERAGE FAILURE		309.40	RCS6A	266.0 ~ 343.0 C
RCS06B	MEDIAN T-AVERAGE TO LOAD		309.40	RCS6B	266.0 ~ 343.0 C
RCS06C	MEDIAN DELTA-T FAILURE		34.71	RCS6C	0.0 ~ 55.0 deltaC
RCS09A	RCP 1 VIBRATION		0.24	RCPA	0 ~ 0.762 mm/s
RCS09B	RCP 2 VIBRATION		0.33	RCPB	0 ~ 0.762 mm/s
RCS09C	RCP 3 VIBRATION		0.22	RCPC	0 ~ 0.762 mm/s
RCS10	REACTOR VESSEL FLANGE LEAK, 100% = 30 GPM	V	0.00	RCSMT1	0 ~ 114 l/m
RCS11A	OIL LEAK FROM RCP 1	B	FALSE	RCPA	
RCS11B	OIL LEAK FROM RCP 2	B	FALSE	RCPB	
RCS11C	OIL LEAK FROM RCP 3	B	FALSE	RCPC	
RCS12A	RCP 1 THERMAL BARRIER LEAK	V	0.00	RCPA	0 ~ 757 l/m
RCS12B	RCP 2 THERMAL BARRIER LEAK	V	0.00	RCPB	0 ~ 757 l/m

Individual Malfunctions RCS01A

Tag: RCS01A

Description: LOOP 1 COLD LEG LOCA, 100% = DOUBLE ENEDED BREAK

Type: V | Final Severity: 100.00 | Ramp Time: 00:00:00 | Delay Time: 00:00:10

Feedback: 0.00 % | Initial Severity: 0.00 | Delete Time: 00:00:00

Event Trigger: None | Increment: 10.00 | Adjust

Info: System: RCS | Range: 0 ~ 100 | Unit: % | status: NOT ACTIVE | Current: 0.00 | Ptag: RCSLEAK1A

✓ Modal

Fig. 2. Starting Menu of RCS01A Malfunction Case

In this report, I have collected 3 malfunction scenarios:

RCS01A	LOOP 1 COLD LEG LOCA, 100% = DOUBLE ENEDED BREAK
RCS01D	LOOP 1 HOT LEG LOCA, 100% = DOUBLE ENEDED BREAK
RCS18A	RCS18 SMALL CL LOCA A LOOP, 100% = 4.5 INCH DIAMETER BREAK

In each scenario, I have to manually run the malfunction and export the data, the following are the steps collecting those data:

Step 1: Input variables required to monitor into simulator and plotting range for them.
Step 2: Go into malfunction index list, find out the scenario I want to simulate and set up the parameter of the scenario. Fig. 2 showing the interface of the set up. We can model the percentage of the malfunction. For example, in RCS18A, 100% = 4.5 INCH means when we set it into 100%, it would have a 4.5 INCH DIAMETER BREAK, while 50% means 2.25 INCH DIAMETER BREAK and so on.
Step 3: Click the top left green button to run the simulator.
Step 4: Stop the simulator when it arrives our desired time frame, in this case time = 60 seconds.
Step 5: Export the data to an excel file and save it, repeat step 1 for next simulation.

In each scenario I have collected 10 datasets, total 30 datasets, time from 0 to 60 seconds, with 88 parameters. Indeed only 30 raw datasets are not enough for deep-learning models to learn in general deep-learning case. This report is just a demonstration showing the pipeline of deep-learning approach dealing with fault diagnosis in Nuclear Power Plant. Also due to limited hardware to process large amount of data, doing a demonstration is a possible way for me to show what I have done.

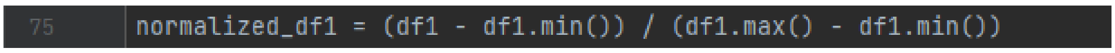
3.1.3. Data transformation

Each data from the previous steps containing total 88 parameters from time 0 to 60 seconds, providing 61x88 = 5368 data points in each excel file. These data points showing the behavior during malfunction. Therefore, it could help us to build up the classification model. To transform our dataset into readable data for python and our neural network. Normalize all the datapoints are necessary. Several recent studies tried to transform their high dimensional data into images and take it as inputs to the CNN models [8]. This approach could reduce the computational cost and learning time for each epoch. The following are the steps of transforming our data from excel format into neural network model accepted data:

Step 1: Using pd.read_excel from pandas to read the excel file and input it into python

Step 2: Transform the data into DataFrame format and normalize it using min-max normalization. This is one of the most common ways to normalize data in deep-learning approach for multi-class classification, the minimum value of each datapoints would be transform to 0, and the maximum value of each datapoints would be transform to 1. Therefore, we would get all the datapoints into a decimal between 0 to 1:

$$X_{normalised} = \frac{X - X_{min}}{X_{max} - X_{min}}$$



```
normalized_df1 = (df1 - df1.min()) / (df1.max() - df1.min())
```

Fig. 3. Normalize DataFrame in code

It is very simple to normalize single DataFrame in python, Fig. 3 shows it could be done by one line of code.

Step 3: Transform every second of the parameters into images. In each scenario, every second we have total 88 parameters in our model. By transforming these 88 parameters into 11x8 2D arrays, we can then plot it as an image. Fig. 4 is showing that how is it look like of the parameters after normalized and reshape into an image. Each little box inside the heatmap is representing one of the parameters we are currently monitoring. Darker the color means it is near 0 and brighter the color means it is near 1.

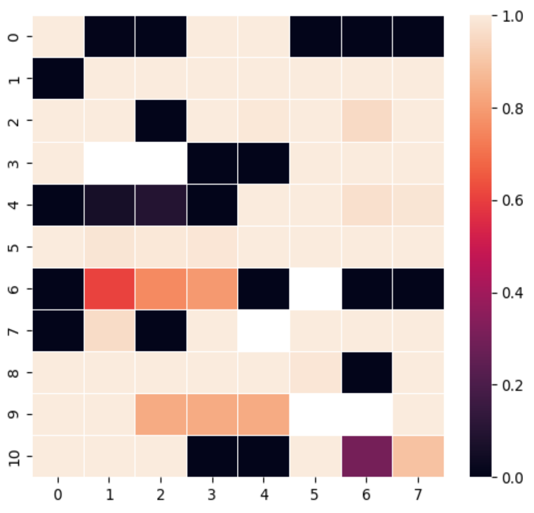


Fig. 4.1. RCS01A Dataset 1 time = 0

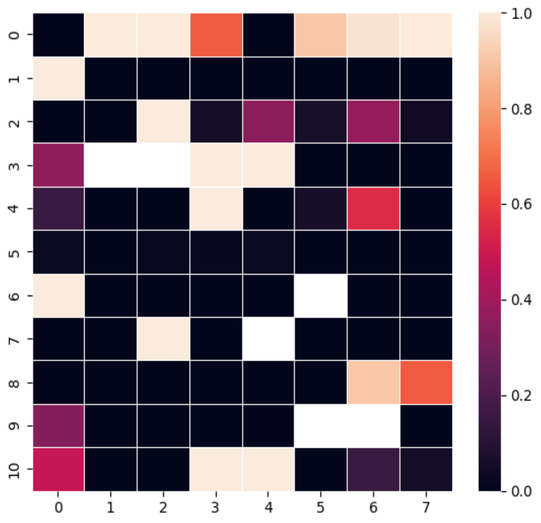


Fig. 4.2. RCS01A Dataset 1 time = 60

Step 4: Before putting these images into our neural network, it is necessary for TensorFlow to split our data into training data, validation data and testing data. Then we can label our dataset which malfunction scenario corresponding to which dataset, and we are ready to feed it to our neural network.

3.2. Convolutional Neural Network (CNN)

Neural network is a model inspired by our human brain. It consists of connected unit called neuron. Neuron is a node in a neural network, each neuron receives a set of values from the previous layer as input, combined with parameters weights (***w***) and bias (***b***), then put the value into an activation function (***σ***), and it is the output value of a neuron. CNN is a neural network with at least one minimum of convolutional layer. A general CNN structure consists of convolutional layers, pooling layers, flatten layers, and fully connected layers.

3.2.1. Convolutional layer (Conv2D)

There are a convolutional filter and an input matrix in convolutional layer. Data structure of a convolutional filter is a N-dimensional matrix, which could not be larger than input matrix. TensorFlow separated the convolutional layer calculation into two-step mathematical operation.

Fig. 5 shows each convolutional operation consists of a single P x Q slice of the convolutional filter multiply with input matrix according to strides. Therefore, size of output Matrix could be determined by the size of both input matrix and convolutional filter: I = M – P + 1, J = N – Q + 1, where I and J are the column and row of output matrix, M and N are the column and row of input matrix, P and Q are the column and row of convolutional filter.

A normal neural network learns a separate weight for every neuron inside the matrix, with the help of convolutional layer, our model could learn the weights of every cell in the convolutional filter only, which reducing the required memory to train and bring high efficiency [9].

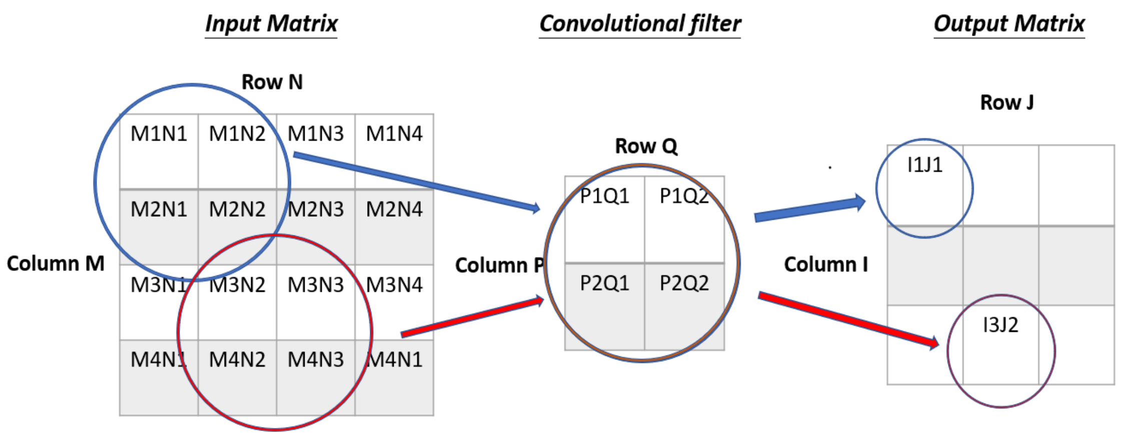


Fig. 5. Operation example of convolutional layer

In order to model a nonlinear problem, it contains an activation function which is applicated for getting the correct probabilities for classification [9]. Activation function would get the weighted sum of every input from the previous layer, and then output a new value for the next layer of the neural network. The value of a neuron in a network could be defined as: $\sigma(wx + b)$,where σ is the activation function, w is weight matrix, x is the input vector and b are the bias vector. A common activation function "relu" would be applied into Conv2D layer:

$$f(x) = max\ (0, x)$$

Strides would be set to (3, 3), it could be set inside the Conv2D layer.

3.2.2. Pooling and Flatten layer (MaxPooling2D, Flatten)

Pooling operation is similar with convolutional operation, which consist of two elements. Input matrix and output matrix. By dividing input matrix into several slides, then slides it into output matrix following the correct orders according to strides. During the pooling operation, maximum values would be taken from the input matrix, each value then would be placed in a new matrix, and this is the output matrix of the pooling layer [10].

Fig. 6 shows an example of pooling layer, the output matrix from the convolutional layer become the input matrix during the pooling operation.

Pooling operation could reduce the number of parameters from the input matrix. Enhancing the ability of the algorithm to classify pattern of between plenty of parameters inside the reactor coolant system. While flatten layer helps to convert the output of our convolutional into a 1D feature vector, those features then input to the dense layer to finish the classification.

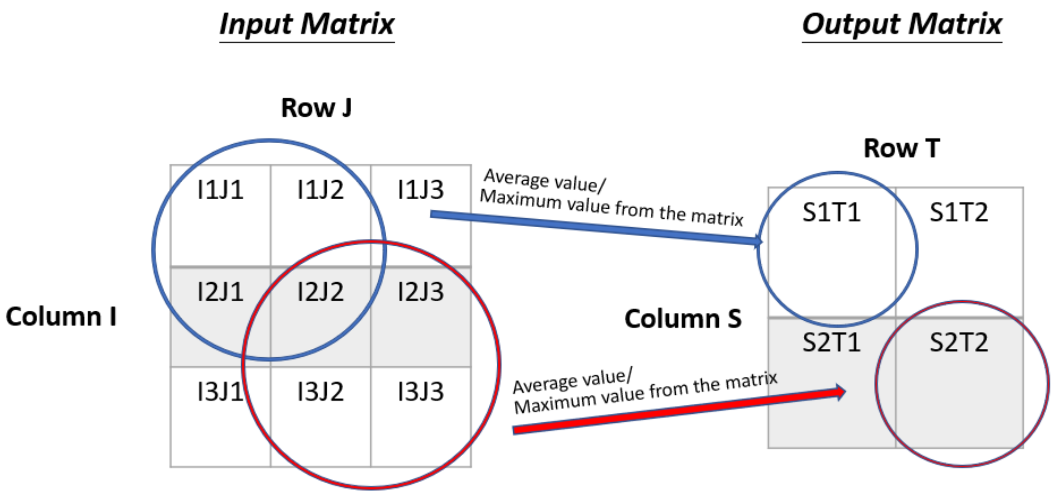


Fig. 6. Operation example of pooling layer

3.2.3. Fully connected layer (Dense)

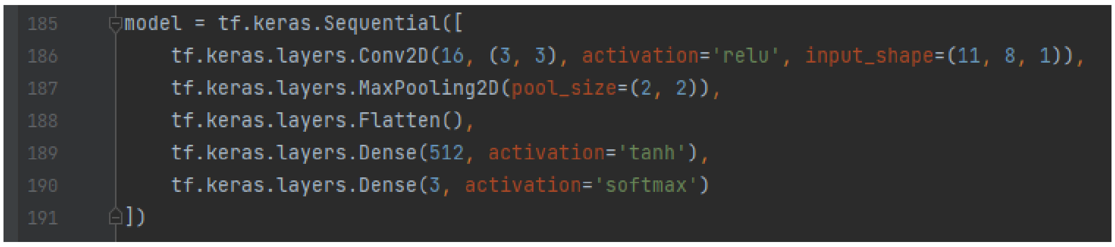
Fully connected layer is the last few layers of the neural network. After the initial matrix passing through convolutional layer and pooling layer, it would reach fully connected layer. It is a hidden layer between input layer and output layer, with each node fully connected to every node in the subsequent hidden layer. First part of the dense layer contains 512 units, describing the dimensionality of the output space, with activation function "tanh":

$$f(x) = tanh\,(x)$$

Second part of the dense layer contains only 3 units, which describing the ultimate goal of our deep learning approach. To classify 3 different classes of malfunction in the reactor coolant system during LOCA event. Activation function "SoftMax" would be applied on this final layer:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

SoftMax function would assigns decimal probabilities to those 3 classes and the final probability adding up must equals to 1 [10]. Our neural network would output the class with the largest probability as the result.



```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(11, 8, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='tanh'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Fig. 7. The Convolution neural network structure in code

3.2.4. Performance evaluation

To consider our performance during training, loss function is required for the model to determine current performance. For multi-class classification, categorical cross-entropy loss function is commonly used:

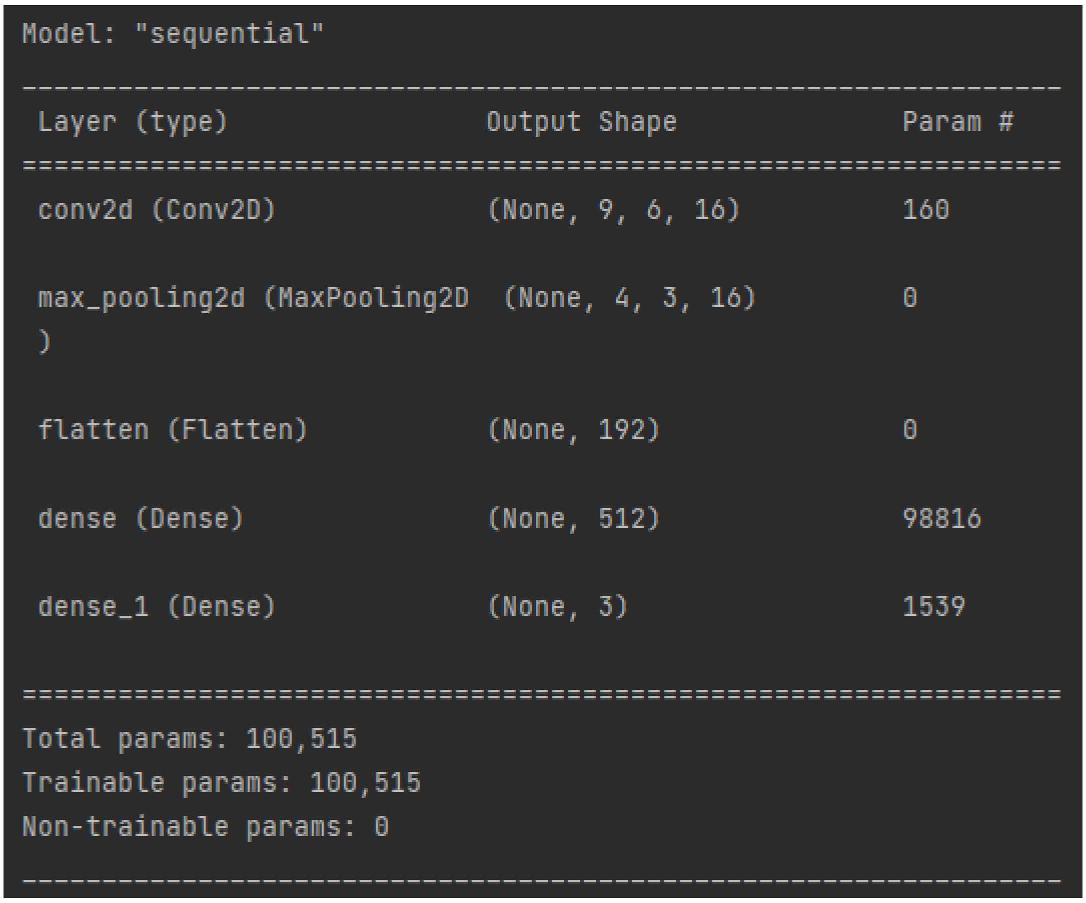
$$CE = -\log\left(\frac{e^{s_p}}{\sum_{j}^{C} e^{s_j}}\right)$$

By using this type of loss function, our label should transform to one-hot label. The loss function would calculate the gradient respect to the output neurons of our model, then backpropagate it and optimize the parameters. Backpropagation Algorithm is modifying the weights ($\boldsymbol{w}$) of an input signal, then release an expected output signal. The error between model prediction and the labels will then input back to weights ($\boldsymbol{w}$) [11].

TensorFlow also provided optimizer for our model to optimize. Optimizer is a stochastic gradient descent procedure to update parameters weights ($\boldsymbol{w}$) based on training data. It is an algorithm for the model to estimates the error gradient at a moment of point.

Performance evaluation metrics would be using Categorical Accuracy, it can help to modulate the contribution of each output, estimate the final loss of the model [9].

Fig.8 is the model summary from first layer to the final layer, with the output shape of each layer showing all parameters and features have been extracted by the neural network. There is total 100,515 parameters during the training process.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 9, 6, 16)          160

 max_pooling2d (MaxPooling2D  (None, 4, 3, 16)         0
 )

 flatten (Flatten)           (None, 192)               0

 dense (Dense)               (None, 512)               98816

 dense_1 (Dense)             (None, 3)                 1539

=================================================================
Total params: 100,515
Trainable params: 100,515
Non-trainable params: 0
_________________________________________________________________
```

Fig. 8. Model Summary

4. Experimental setting

4.1. Data setting

GSE GPWR Generic Nuclear Simulator would be using to simulate the data. It is an ANS 3.5 certified full scope Pressurized Water Reactor training simulator.

As mentioned before, total 3 of the abnormal events have been chosen from the malfunction index list for the classification task.

RCS01A, it represents the cold leg double ended break LOCA in Loop 1.

RCS01D, it presents the hot leg double ended break LOCA in Loop 1.

RCS18A, it represents the small cold leg LOCA in Loop A.

Cold leg accumulators are pressure vessels filled with boron solutions and pressurized nitrogen. It is an electricity-free system, which means it does not require any electrical power to run it. During the normal operation, each cold leg accumulators would be isolated from the reactor coolant system by two check valves in series. It is designed to provide water to the reactor coolant system when it is overheated. During the malfunction, the pressure of the primary system would drop significantly and causing large primary break. Also, the nitrogen would causing the borated water flow into the reactor coolant system. The cold leg accumulator is sized to reflood the reactor core adequately. Therefore, by monitoring the reactor vessel level (brvls2a), Core pressure (pcore_mpag) etc. The characteristic of the malfunction could be visualized and analyzed manually or by computers.

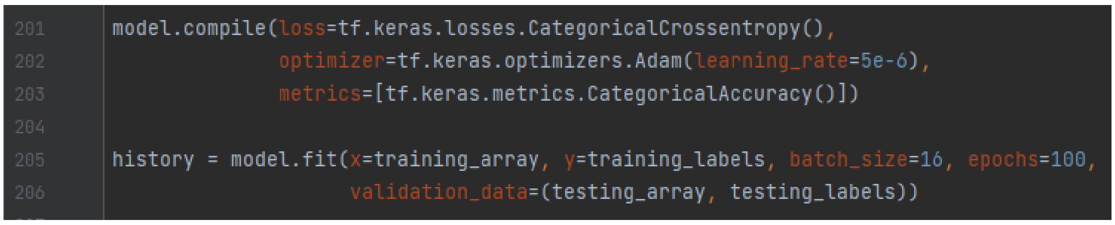
In this report I would be focusing on classify the reactor coolant system malfunction. However, as an undergraduate level student, it is impossible for me to understand all the components from the nuclear reactor coolant system, correctly recognize them and input those parameters into the simulator. Therefore, some of the parameters I am monitoring are come from its default setting, while 20-30 parameters are inspired by our nuclear simulator lab teaching section. The chosen parameters are crucial parameters for determine the situation of LOCA events. Including Reactor Coolant System Loop A Flow (ft:414), RCS Loop A cold leg temperature (tt:410_si), Containment vessel pressure as MPaa (pcnm_si) etc.

For each scenario, the severity rate of the malfunction is different. I intended averagely distribute the severity rate from 10% to 100% for each case. With delayed time set to t = 10 second, which means all of the scenario would starting its malfunction at exactly t = 10s. For each malfunction, it contains the same number of datasets, it could help to avoid imbalance data lead to ruining our model with bias.

4.2. Hyper-parameter selection

Hyper-parameter representing the values which are controlling the learning process, in order to find out a good learning curve for the neural network model, retesting the model by fitting different value of hyper-parameters is necessary. It could help to identify the feature extraction and do a better classification. In TensorFlow, the number of the output convolution filters, pooling size of the pooling layer and the stride value could be tune on the layer code. While input shape of the first layer required to calculate by ourselves. Input a wrong dimensional shape would causing the program error. Tiny changes of the hyper-parameter would cause a totally different result, and the following shows the changing of hyper-parameter in our simulation:

Number of the output convolution filters	8	16
Pooling Size of pooling layer	(2, 2)	(3, 3)
Dimensionality of the Dense output	128	512
Optimizer	Adam	RMSprop
Learning rate	Using LearningRateScheduler Callback to find the optimal learning rate for our neural network: 5e-6	



```
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(learning_rate=5e-6),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])

history = model.fit(x=training_array, y=training_labels, batch_size=16, epochs=100,
                    validation_data=(testing_array, testing_labels))
```

Fig. 9. Model Compile in code

Fig. 9 showing how to input some of the hyper-parameters and compile the model in TensorFlow.

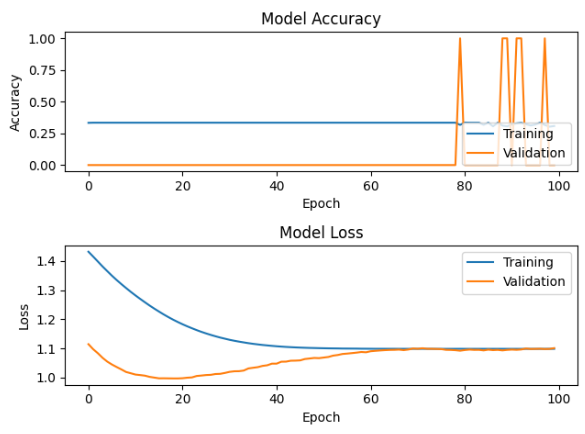
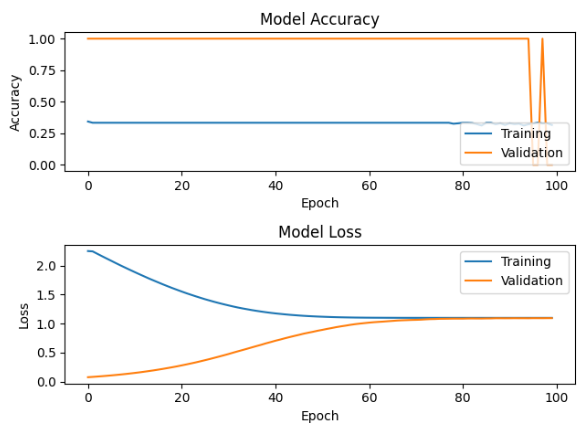
As our datasets is not in large scale, so I decided the number of batch size and epochs would remain as constant.

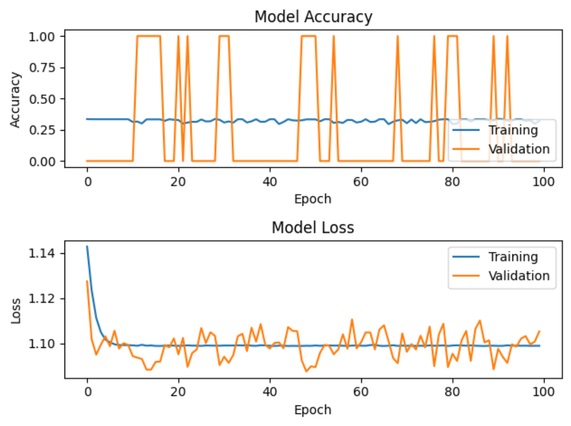
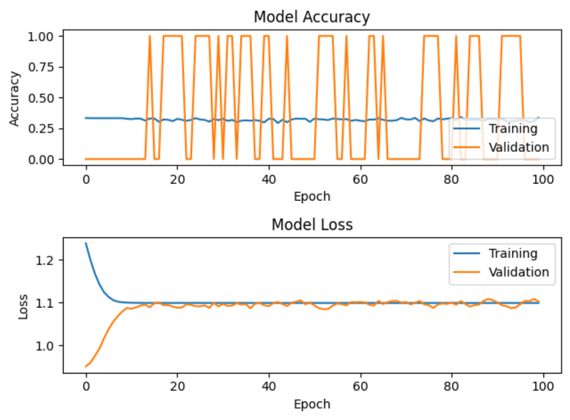
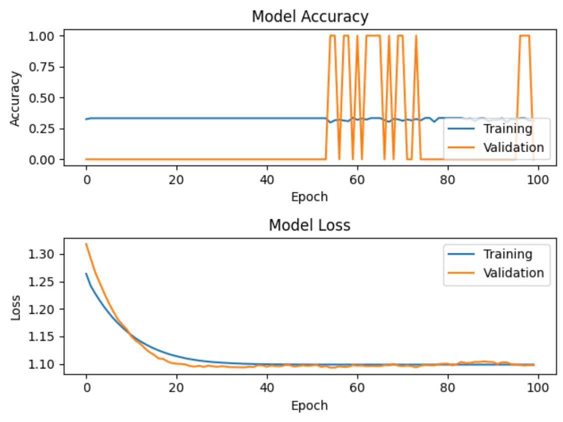
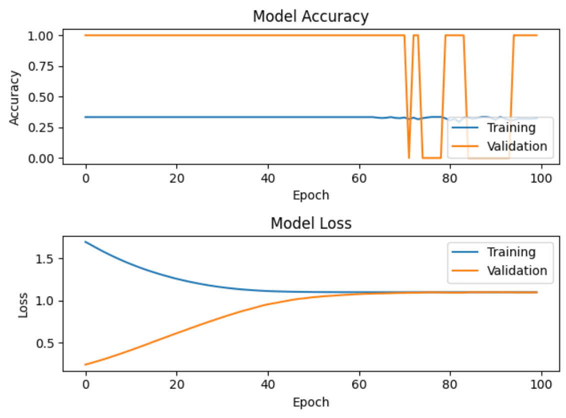
Adam and RMSprop optimizer would be chosen as the optimizer. Adam could maintain a per-parameter learning rate and improve the performance [12]. While RMSprop using moving average of gradient square to optimize the gradient [13].

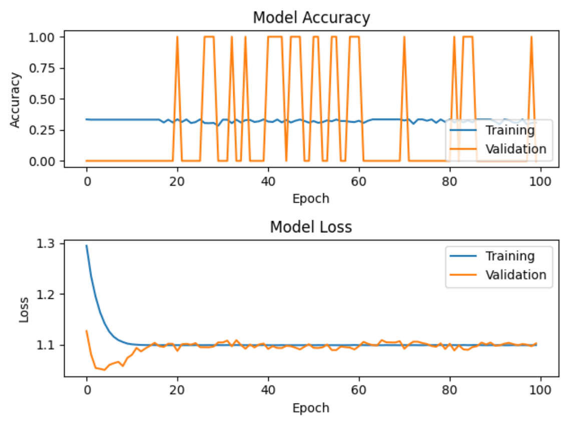
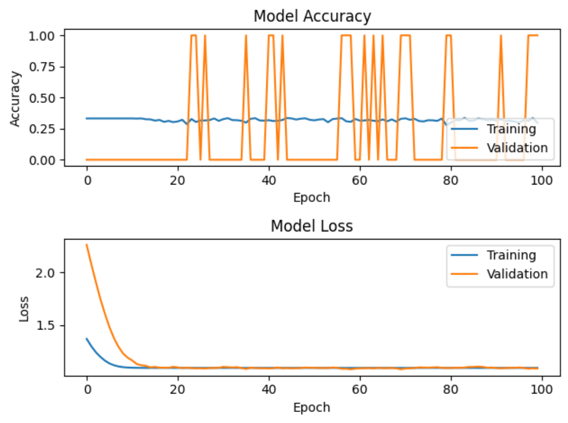
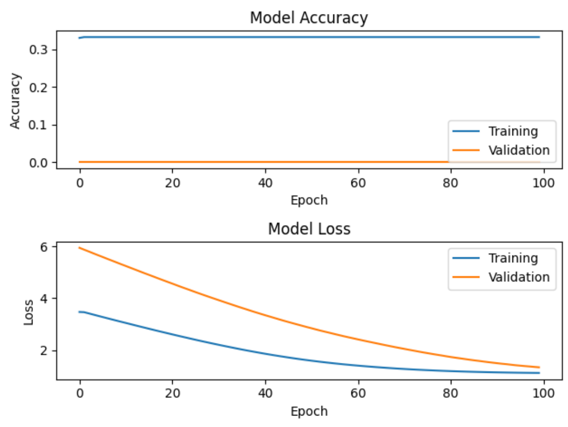
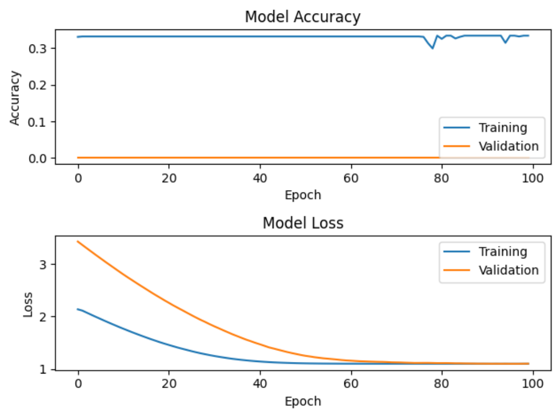
During the backpropagation process mentioned in section 3.2.4. Performance evaluation, weights (***w***) would be updated during training. The step size of training process is called learning rate. The learning rate of our training are determined by learning rate callback function. It allows the model to return the performance of different learning rate during each epochs. Therefore, we can find the optimal learning rate through the graph plotted by callback function.

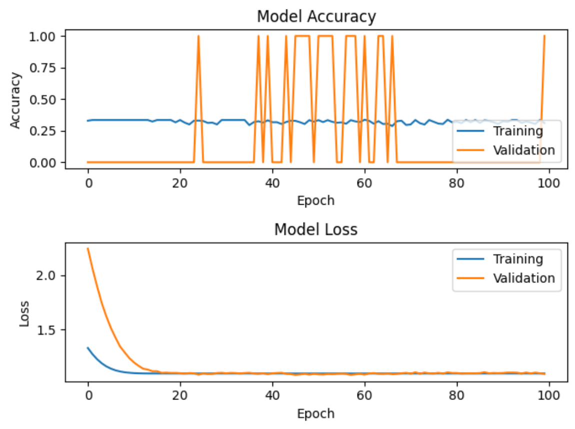
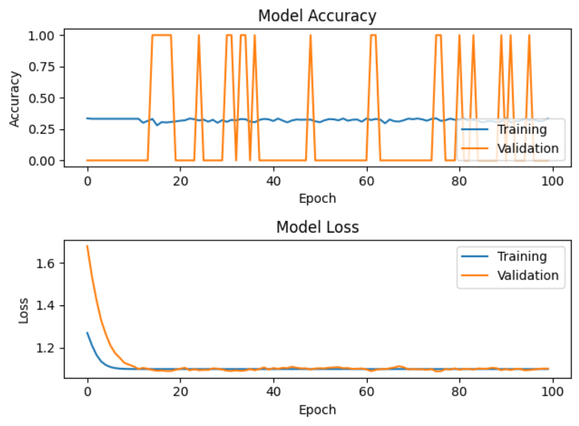
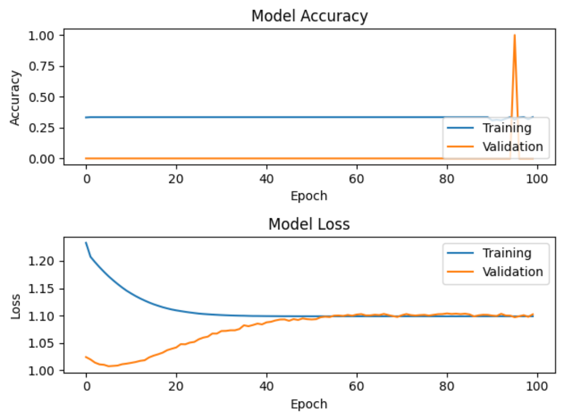
5. Results

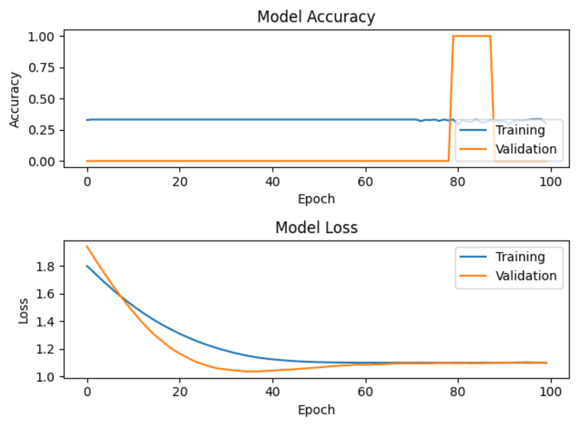
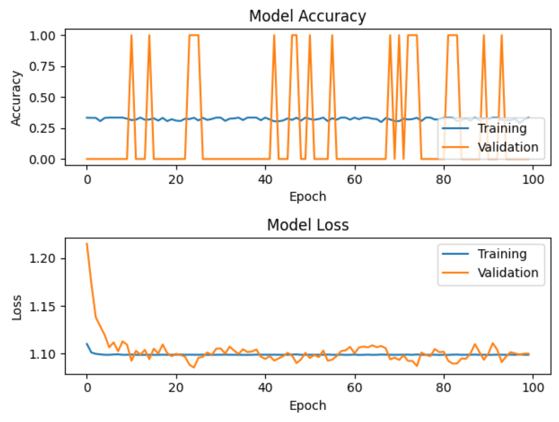
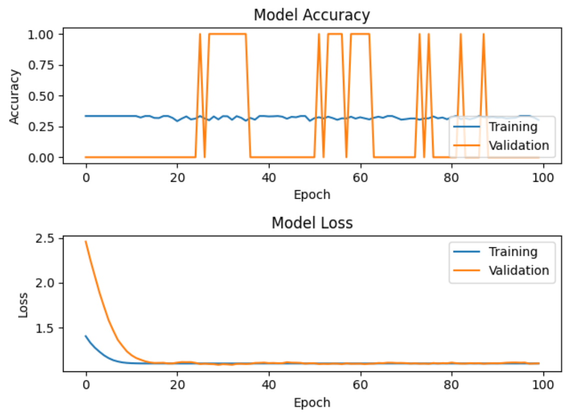
5.1. Performance table of the model with different hyper-parameter

Number of the output convolution filters	Pooling Size of pooling layer	Dimensionality of the Dense output	Optimizer	Accuracy	Model Accuracy and Model Loss during training
8	(2, 2)	128	Adam	0.3082	
8	(2, 2)	128	RMSprop	0.3158	

8	(2, 2)	512	Adam	0.3202	
8	(2, 2)	512	RMSprop	0.3355	
8	(3, 3)	128	Adam	0.3344	
8	(3, 3)	128	RMSprop	0.3235	

8	(3, 3)	512	Adam	0.3082	
8	(3, 3)	512	RMSprop	0.2973	
16	(2, 2)	128	Adam	0.3322	
16	(2, 2)	128	RMSprop	0.3344	

16	(2, 2)	512	Adam	0.3104	
16	(2, 2)	512	RMSprop	0.3322	
16	(3, 3)	128	Adam	0.3344	

16	(3, 3)	128	RMSprop	0.3005	
16	(3, 3)	512	Adam	0.3344	
16	(3, 3)	512	RMSprop	0.3027	

5.2. Observation of the result

First, it is obviously that our convolution neural network does not working well, the training accuracy in all 16 cases are stuck in 0.3344 below. While the validation accuracy was jumping between 0 and 1. Validation loss of different model are different depending on their hyper-parameter setting, but the final training loss of all models coincidentally reached 1.099 and stuck in there. Seems like 1.099 is the limit of our model. Since the model could not classify well which kind of malfunction were happening. The deep-learning approach was failed.

Although the result is not ideal, analyzing the changing of hyper-parameter could be useful for improve our performance for the next research. Therefore, the following would be describing how the hyper-parameter affecting the model output.

Number of the output convolution filters: There is not significantly different between 8 and 16 convolutional filters

Pooling Size of pooling layer: There is not significantly different between (2, 2) and (3, 3) pooling size

Dimensionality of the Dense output: the training loss with 128 dense outputs mostly dropping from epoch 0 to 50, then convergence with the validation loss. While 512 dense outputs training loss drop significantly from epochs 0 to 15, showing dense output with 512 could enhance the model performance, causing the model decreasing its loss quicker.

Optimizer: There is not significantly different between two optimizers

After comparing all the training results, only dimensionality of the dense output did enhance the performance of our model. While others seem not doing well.

6. Discussion

In part 5, I only describing the pattern of our model result, and all the explanation parts would be discussed in this section.

6.1. Explanation of the result

The reason why there is no description of validation accuracy is because it is unsolved bug for me. As we can observe that, the validation accuracy of the model is keep jumping between 0 and 1, which means the model having bugs dealing with new datasets. A pattern could be observed is that the validation accuracy will only jumping after the training loss and validation loss convergence. Meaning that validation accuracy at that moment of point is low enough for model to classify the malfunction of the reactor coolant system. However, it is weird that the training accuracy are stuck in 0.3344 below no matter the training loss is dropping or not. After doing some research, I find out our model could be stuck in a local minima due to not enough training data. It also explained why the training loss and validation loss stuck, the model could no longer minimize the loss and generate our desired output. Leading us that putting a small datasets into a convolutional neural network is not a smart choice.

Secondly, the changing of the hyper-parameter does not show significant pattern mostly. At first, I expected that changing the hyper-parameter would lead me to find a better performance model to do our classification. However, I did not expect 3 out the 4 hyper-parameter does not show the correlation of the performance.

The only good result is that the dimensionality of the Dense output did affect our model. The function of the second Dense layer is to extract features from the precious layer. Therefore, the result is telling us the features of our malfunction scenario were captured by the model. Due to there are too many features inside the reactor coolant system, 512 dimensionality is better than 128.

6.2. Bug solving

The original architecture of the neural network model is not same as the current version. For instance, the activation function of first dense layer is not "tanh" at first. I had chosen "relu" function for it. After running the model, I discovered that the training loss of the model are remaining NaN. I considered it as too high learning rate causing the loss begins to increase and diverges to infinity. Then try to reduce others hyper-parameter, the learning rate. However, the loss value is remaining at NaN.

As mentioned, the NaN value is solved by charging activation function "relu" to "tanh". It is because the "relu" function has a range of $[0, \infty]$, when it deals with the categorical cross entropy loss function, any activation value with 0 or 1 will turn to NaN value due to its mathematical property [14]

7. Conclusion

"Garbage in , garbage out", it is a common concept of computer science meaning nonsense input data would also produce nonsense result. The failure of the deep-learning approach may result in choosing not accurate parameter from the simulator. Causing the reactor coolant system malfunction features could not be reflected on our model. Also, small quantity of datasets also limited the training process. It is pity that the data from the nuclear simulator lab have to export the excel file manually. It could be more effective if I know the way of exporting it through program, large amount of data could then be trained.

Classifying the malfunction event of the reactor coolant reactor is a multi-class classification problem. Therefore, knowledge of machine learning is crucial for this project. By demonstrating the pipeline of how deep learning could apply to nuclear science, it really helps me to understand the fundamental of both nuclear and computer science. Although, the result is undoubtedly failed, it is a great experience for me to learn.

8. References

----

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional Neural Networks,” Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017.

[3] J. Ma and J. Jiang, “Applications of fault detection and diagnosis methods in nuclear power plants: A Review,” Progress in Nuclear Energy, vol. 53, no. 3, pp. 255–266, 2011.

[4] R. Razavi-Far, H. Davilu, V. Palade, and C. Lucas, “Model-based fault detection and isolation of a steam generator using neuro-fuzzy networks,” Neurocomputing, vol. 72, no. 13-15, pp. 2939–2951, 2009.

[5] A. Widodo and B.-S. Yang, “Support Vector Machine in machine condition monitoring and fault diagnosis,” Mechanical Systems and Signal Processing, vol. 21, no. 6, pp. 2560–2574, 2007.

[6] GSE GPWR Generic Nuclear Simulator, Westinghouse, USA

[7] R. A. Knief, in Nuclear engineering: Theory and technology of commercial nuclear power, LaGrange Park, IL: American Nuclear Society, 2008, pp. 348–348.

[8] G. Lee, S. J. Lee, and C. Lee, “A convolutional neural network model for abnormality diagnosis in a nuclear power plant,” Applied Soft Computing, vol. 99, p. 106874, 2021.

[9] J. Brownlee, “How to develop a CNN from scratch for CIFAR-10 photo

classification," Machine Learning Mastery, 27-Aug-2020. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/. [Accessed: 25-Nov-2021].

[10] D. Google, "Machine learning glossary | google developers," Google, 2021. [Online]. Available: https://developers.google.com/machine-learning/glossary/#convolutional_operation. [Accessed: 26-Nov-2021].

[11] J. Brownlee, "How to code a neural network with backpropagation in Python (from scratch)," Machine Learning Mastery, 21-Oct-2021. [Online]. Available: https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/. [Accessed: 25-Apr-2022].

[12] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," Machine Learning Mastery, 12-Jan-2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed: 25-Apr-2022].

[13] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu, "A sufficient condition for convergences of Adam and rmsprop," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[14] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of Mislabeling," IEEE Access, vol. 8, pp. 4806–4813, 2020.

9. Attachment

The google drive link contains all 30 raw data csv file from the nuclear simulator
https://drive.google.com/drive/folders/1J49ISMhRI5wD1hLB9YRBvks2Cj6fglIV

The following is the code for this report:

```
import csv
import pandas as pd
import numpy as np
import openpyxl
import seaborn as sns
import matplotlib.pyplot as plt
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
import tensorflow as tf

# Read the excel file
data_1 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A1D.xlsx')
data_2 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A2D.xlsx')
data_3 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A3D.xlsx')
data_4 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A4D.xlsx')
data_5 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A5D.xlsx')
data_6 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A6D.xlsx')
data_7 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A7D.xlsx')
data_8 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A8D.xlsx')
data_9 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A9D.xlsx')
data_10 =
pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01A10D.xlsx')
data_11 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D1D.xlsx')
data_12 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D2D.xlsx')
data_13 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D3D.xlsx')
data_14 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D4D.xlsx')
data_15 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D5D.xlsx')
data_16 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D6D.xlsx')
data_17 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D7D.xlsx')
```

```
data_18 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D8D.xlsx')
data_19 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D9D.xlsx')
data_20 =
pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS01D10D.xlsx')
data_21 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A1D.xlsx')
data_22 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A2D.xlsx')
data_23 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A3D.xlsx')
data_24 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A4D.xlsx')
data_25 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A5D.xlsx')
data_26 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A6D.xlsx')
data_27 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A7D.xlsx')
data_28 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A8D.xlsx')
data_29 = pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A9D.xlsx')
data_30 =
pd.read_excel(r'C:\Users\JohnLeung\Documents\fypdata\RCS18A10D.xlsx')

# Transform all data into DataFrame
df1 = pd.DataFrame(data_1)
df2 = pd.DataFrame(data_2)
df3 = pd.DataFrame(data_3)
df4 = pd.DataFrame(data_4)
df5 = pd.DataFrame(data_5)
df6 = pd.DataFrame(data_6)
df7 = pd.DataFrame(data_7)
df8 = pd.DataFrame(data_8)
df9 = pd.DataFrame(data_9)
df10 = pd.DataFrame(data_10)
df11 = pd.DataFrame(data_11)
df12 = pd.DataFrame(data_12)
df13 = pd.DataFrame(data_13)
df14 = pd.DataFrame(data_14)
df15 = pd.DataFrame(data_15)
df16 = pd.DataFrame(data_16)
df17 = pd.DataFrame(data_17)
df18 = pd.DataFrame(data_18)
df19 = pd.DataFrame(data_19)
df20 = pd.DataFrame(data_20)
df21 = pd.DataFrame(data_21)
```

```
df22 = pd.DataFrame(data_22)
df23 = pd.DataFrame(data_23)
df24 = pd.DataFrame(data_24)
df25 = pd.DataFrame(data_25)
df26 = pd.DataFrame(data_26)
df27 = pd.DataFrame(data_27)
df28 = pd.DataFrame(data_28)
df29 = pd.DataFrame(data_29)
df30 = pd.DataFrame(data_30)

# Normalize all the DataFrame
normalized_df1 = (df1 - df1.min()) / (df1.max() - df1.min())
normalized_df2 = (df2 - df2.min()) / (df2.max() - df3.min())
normalized_df3 = (df3 - df3.min()) / (df3.max() - df3.min())
normalized_df4 = (df4 - df4.min()) / (df4.max() - df4.min())
normalized_df5 = (df5 - df5.min()) / (df5.max() - df5.min())
normalized_df6 = (df6 - df6.min()) / (df6.max() - df6.min())
normalized_df7 = (df7 - df7.min()) / (df7.max() - df7.min())
normalized_df8 = (df8 - df8.min()) / (df8.max() - df8.min())
normalized_df9 = (df9 - df9.min()) / (df9.max() - df9.min())
normalized_df10 = (df10 - df10.min()) / (df10.max() - df10.min())
normalized_df11 = (df11 - df11.min()) / (df11.max() - df11.min())
normalized_df12 = (df12 - df12.min()) / (df12.max() - df12.min())
normalized_df13 = (df13 - df13.min()) / (df13.max() - df13.min())
normalized_df14 = (df14 - df14.min()) / (df14.max() - df14.min())
normalized_df15 = (df15 - df15.min()) / (df15.max() - df15.min())
normalized_df16 = (df16 - df16.min()) / (df16.max() - df16.min())
normalized_df17 = (df17 - df17.min()) / (df17.max() - df17.min())
normalized_df18 = (df18 - df18.min()) / (df18.max() - df18.min())
normalized_df19 = (df19 - df19.min()) / (df19.max() - df19.min())
normalized_df20 = (df20 - df20.min()) / (df20.max() - df20.min())
normalized_df21 = (df21 - df21.min()) / (df21.max() - df21.min())
normalized_df22 = (df22 - df22.min()) / (df22.max() - df22.min())
normalized_df23 = (df23 - df23.min()) / (df23.max() - df23.min())
normalized_df24 = (df24 - df24.min()) / (df24.max() - df24.min())
normalized_df25 = (df25 - df25.min()) / (df25.max() - df25.min())
normalized_df26 = (df26 - df26.min()) / (df26.max() - df26.min())
normalized_df27 = (df27 - df27.min()) / (df27.max() - df27.min())
```

```
normalized_df28 = (df28 - df28.min()) / (df28.max() - df28.min())
normalized_df29 = (df29 - df29.min()) / (df29.max() - df29.min())
normalized_df30 = (df30 - df30.min()) / (df30.max() - df30.min())

# Transform all the DataFrame into Numpy Array
normalized_numpy1 = normalized_df1.to_numpy()
normalized_numpy2 = normalized_df2.to_numpy()
normalized_numpy3 = normalized_df3.to_numpy()
normalized_numpy4 = normalized_df4.to_numpy()
normalized_numpy5 = normalized_df5.to_numpy()
normalized_numpy6 = normalized_df6.to_numpy()
normalized_numpy7 = normalized_df7.to_numpy()
normalized_numpy8 = normalized_df8.to_numpy()
normalized_numpy9 = normalized_df9.to_numpy()
normalized_numpy10 = normalized_df10.to_numpy()
normalized_numpy11 = normalized_df11.to_numpy()
normalized_numpy12 = normalized_df12.to_numpy()
normalized_numpy13 = normalized_df13.to_numpy()
normalized_numpy14 = normalized_df14.to_numpy()
normalized_numpy15 = normalized_df15.to_numpy()
normalized_numpy16 = normalized_df16.to_numpy()
normalized_numpy17 = normalized_df17.to_numpy()
normalized_numpy18 = normalized_df18.to_numpy()
normalized_numpy19 = normalized_df19.to_numpy()
normalized_numpy20 = normalized_df20.to_numpy()
normalized_numpy21 = normalized_df21.to_numpy()
normalized_numpy22 = normalized_df22.to_numpy()
normalized_numpy23 = normalized_df23.to_numpy()
normalized_numpy24 = normalized_df24.to_numpy()
normalized_numpy25 = normalized_df25.to_numpy()
normalized_numpy26 = normalized_df26.to_numpy()
normalized_numpy27 = normalized_df27.to_numpy()
normalized_numpy28 = normalized_df28.to_numpy()
normalized_numpy29 = normalized_df29.to_numpy()
normalized_numpy30 = normalized_df30.to_numpy()

# Split the data into training set and validation set
training_array = np.stack(
    [normalized_numpy1, normalized_numpy3, normalized_numpy5,
normalized_numpy7, normalized_numpy9,
     normalized_numpy11, normalized_numpy13, normalized_numpy15,
normalized_numpy17, normalized_numpy19,
     normalized_numpy21, normalized_numpy23, normalized_numpy25,
normalized_numpy27, normalized_numpy29,
     ])

testing_array = np.stack(
    [normalized_numpy2, normalized_numpy4, normalized_numpy6,
normalized_numpy12,
     normalized_numpy14, normalized_numpy16, normalized_numpy22,
normalized_numpy24, normalized_numpy26])

# Reform the training data in order to fit the neural network
# Setting the labels for the network
train_list = []
test_list = []
counter = 0
num = 0
counter2 = 0
num2 = 0

for i in range(915):
    if counter < 305:
        train_list.append(num)
        counter += 1
    else:
        num += 1
        train_list.append(num)
        counter = 0

for j in range(549):
    if counter2 < 183:
        test_list.append(num2)
        counter2 += 1
    else:
        num += 1
```

```
          test_list.append(num2)
          counter2 = 0

training_labels = np.asfarray(train_list, float)
testing_labels = np.asfarray(test_list, float)

training_labels = tf.keras.utils.to_categorical(training_labels, num_classes=3)
testing_labels = tf.keras.utils.to_categorical(testing_labels, num_classes=3)

training_array = training_array.reshape(915, 11, 8)
testing_array = testing_array.reshape(549, 11, 8)

training_array = tf.expand_dims(training_array, axis=3)
testing_array = tf.expand_dims(testing_array, axis=3)

# CNN model structure
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(11, 8, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='tanh'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Print the model summary
model.summary()

# Model compiler
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=5e-6),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])

# Fit the model and print out the result
history = model.fit(x=training_array, y=training_labels, batch_size=16, epochs=100,
                    validation_data=(testing_array, testing_labels))

plt.subplot(211)
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.tight_layout()

plt.show()
```