# Report - Dataset Introduction ANN Modeling Analysis Hyperparameter Optimization Analysis

# author:Du SID：6511190010

## Author GitHub:

## <https://github.com/dqswordman/MUT_Neural_Networks_LAB>

## -> [Lab5.rar](https://github.com/dqswordman/MUT_Neural_Networks_LAB/blob/main/Lab5.rar" \o "Lab5.rar) (Part1-3)Code Code Analysis Documentation

## ->[Du\_Hw.ipynb](https://github.com/dqswordman/MUT_Neural_Networks_LAB/blob/main/Du_Hw.ipynb" \o "Du_Hw.ipynb) Chapter 4 details

## ---Please use View->Web Layout to read this report---

# 1. Dataset Introduction

## 1.1 Dataset Content Analysis, Preprocessing, Feature and Label Extraction, Label Encoding, Data Splitting, and Standardization

### In this analysis, we used a network traffic dataset sourced from Kaggle, named ****"Friday-WorkingHours-Morning.pcap\_ISCX.csv"****. The dataset was published by ****SYED ALI HAIDER**** with the purpose of studying and detecting DDoS attacks in the network and their characteristics. Each record in the dataset represents a network flow, containing multiple features and metrics related to network connections. These features are used to build detection models to distinguish between normal traffic and attack traffic.

数据集内容分析：

#### Dataset Content Analysis:

1. **Destination Port**: The destination port number, indicating the target port for the data packet transmission.
2. **Flow Duration**: The duration of the network flow, representing the time span from the start to the end of the flow (milliseconds).
3. **Total Fwd Packets**: Total number of forward (source to destination) packets.
4. **Total Backward Packets**: Total number of backward (destination to source) packets.
5. **Total Length of Fwd Packets**: Total byte length of the forward packets.
6. **Total Length of Bwd Packets**: Total byte length of the backward packets.
7. **Fwd Packet Length Max**: Maximum length of the forward packets.
8. **Fwd Packet Length Min**: Minimum length of the forward packets.
9. **Fwd Packet Length Mean**: Mean length of the forward packets.
10. **Fwd Packet Length Std**: Standard deviation of the forward packet lengths.
11. **Bwd Packet Length Max**: Maximum length of the backward packets.
12. **Bwd Packet Length Min**: Minimum length of the backward packets.
13. **Bwd Packet Length Mean**: Mean length of the backward packets.
14. **Bwd Packet Length Std**: Standard deviation of the backward packet lengths.
15. **Flow Bytes/s**: Number of bytes per second in the network flow.
16. **Flow Packets/s**: Number of packets per second in the network flow.
17. **Flow IAT Mean**: Mean inter-arrival time of the network flows.
18. **Flow IAT Std**: Standard deviation of the inter-arrival time of the network flows.
19. **Flow IAT Max**: Maximum inter-arrival time of the network flows.
20. **Flow IAT Min**: Minimum inter-arrival time of the network flows.
21. **Fwd IAT Total**: Total inter-arrival time for all forward packets.
22. **Fwd IAT Mean**: Mean inter-arrival time for forward packets.
23. **Fwd IAT Std**: Standard deviation of the inter-arrival time for forward packets.
24. **Fwd IAT Max**: Maximum inter-arrival time for forward packets.
25. **Fwd IAT Min**: Minimum inter-arrival time for forward packets.
26. **Bwd IAT Total**: Total inter-arrival time for all backward packets.
27. **Bwd IAT Mean**: Mean inter-arrival time for backward packets.
28. **Bwd IAT Std**: Standard deviation of the inter-arrival time for backward packets.
29. **Bwd IAT Max**: Maximum inter-arrival time for backward packets.
30. **Bwd IAT Min**: Minimum inter-arrival time for backward packets.
31. **Fwd PSH Flags**: Number of PSH flags in the forward packets.
32. **Bwd PSH Flags**: Number of PSH flags in the backward packets.
33. **Fwd URG Flags**: Number of URG flags in the forward packets.
34. **Bwd URG Flags**: Number of URG flags in the backward packets.
35. **Fwd Header Length**: Total header length of the forward packets.
36. **Bwd Header Length**: Total header length of the backward packets.
37. **Fwd Packets/s**: Number of forward packets per second.
38. **Bwd Packets/s**: Number of backward packets per second.
39. **Min Packet Length**: Minimum length of a data packet.
40. **Max Packet Length**: Maximum length of a data packet.
41. **Packet Length Mean**: Mean length of data packets.
42. **Packet Length Std**: Standard deviation of the data packet lengths.
43. **Packet Length Variance**: Variance of data packet lengths.
44. **FIN Flag Count**: Number of FIN flags in the data packets.
45. **SYN Flag Count**: Number of SYN flags in the data packets.
46. **RST Flag Count**: Number of RST flags in the data packets.
47. **PSH Flag Count**: Number of PSH flags in the data packets.
48. **ACK Flag Count**: Number of ACK flags in the data packets.
49. **URG Flag Count**: Number of URG flags in the data packets.
50. **CWE Flag Count**: Number of CWE flags in the data packets.
51. **ECE Flag Count**: Number of ECE flags in the data packets.
52. **Down/Up Ratio**: Ratio of download to upload packets.
53. **Average Packet Size**: Average size of data packets.
54. **Avg Fwd Segment Size**: Average segment size of the forward packets.
55. **Avg Bwd Segment Size**: Average segment size of the backward packets.
56. **Fwd Header Length.1**: Duplicate field for forward packet header length.
57. **Fwd Avg Bytes/Bulk**: Average number of bytes per bulk transfer in the forward direction.
58. **Fwd Avg Packets/Bulk**: Average number of packets per bulk transfer in the forward direction.
59. **Fwd Avg Bulk Rate**: Average bulk transfer rate in the forward direction.
60. **Bwd Avg Bytes/Bulk**: Average number of bytes per bulk transfer in the backward direction.
61. **Bwd Avg Packets/Bulk**: Average number of packets per bulk transfer in the backward direction.
62. **Bwd Avg Bulk Rate**: Average bulk transfer rate in the backward direction.
63. **Subflow Fwd Packets**: Number of forward packets in the subflow.
64. **Subflow Fwd Bytes**: Number of forward bytes in the subflow.
65. **Subflow Bwd Packets**: Number of backward packets in the subflow.
66. **Subflow Bwd Bytes**: Number of backward bytes in the subflow.
67. **Init\_Win\_bytes\_forward**: Initial window size in bytes for forward packets.
68. **Init\_Win\_bytes\_backward**: Initial window size in bytes for backward packets.
69. **act\_data\_pkt\_fwd**: Number of actual forward data packets.
70. **min\_seg\_size\_forward**: Minimum segment size in the forward direction.
71. **Active Mean**: Mean of active time.
72. **Active Std**: Standard deviation of active time.
73. **Active Max**: Maximum active time.
74. **Active Min**: Minimum active time.
75. **Idle Mean**: Mean of idle time.
76. **Idle Std**: Standard deviation of idle time.
77. **Idle Max**: Maximum idle time.
78. **Idle Min**: Minimum idle time.  
    **Label**: Classification label (e.g., "BENIGN" or "DDoS"), indicating whether the network traffic is normal or attack traffic.

### From Lab5\_Code\_Note.ipynb #3.1

*#3.1*

*1.import pandas as pd*

*2.*

*3.# Read data*

*4.df = pd.read\_csv('./Friday-WorkingHours-Afternoon-DDos.pcap\_ISCX.csv')*

*5.*

*6.# Remove the leading and leaving spaces in column names*

*7.df.columns = df.columns.str.strip()*

*8.*

*9.# Display each column of data*

*10.print("Column name of data:")*

*11.print(df.columns)*

*12.*

*13.# Display the first few rows of data*

*14.print("First few rows of data:")*

*15. print(df.head())*

*16.*

*17.# Show the shape of the dataset (number of rows and columns)*

*18.print("Shape of the dataset:")*

*19.print(df.shape)*

*20.*

*21.# Show each row of the data (may output a lot of information, be careful when using)*

*22.print("Each row of the data:")*

*23.for index, row in df.iterrows():*

*24.    print(f"row {index}: {row.to\_dict()}")*

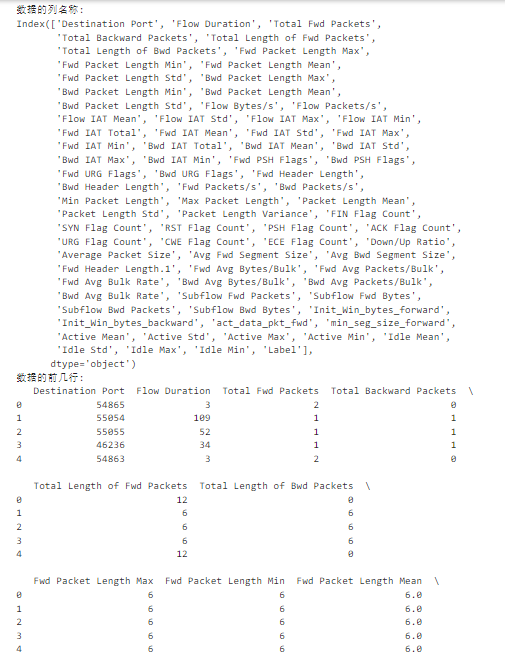
*25.    # Note: This is just an example. If the dataset is very large (such as more than a few hundred rows), this may result in too much information being output.*

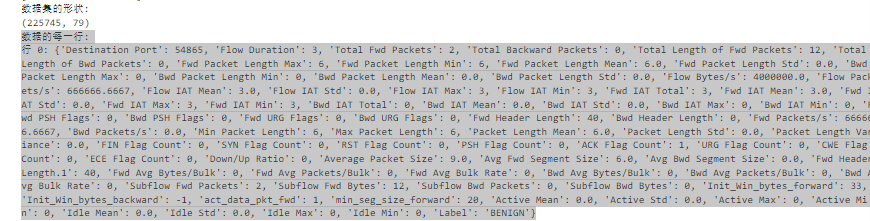
*26.    # You can use conditions to limit the number of rows printed, for example, only print the first 10 rows:*

*27.    if index >= 9:  # Print the first 10 rows*

*28.        break*

The output is as shown



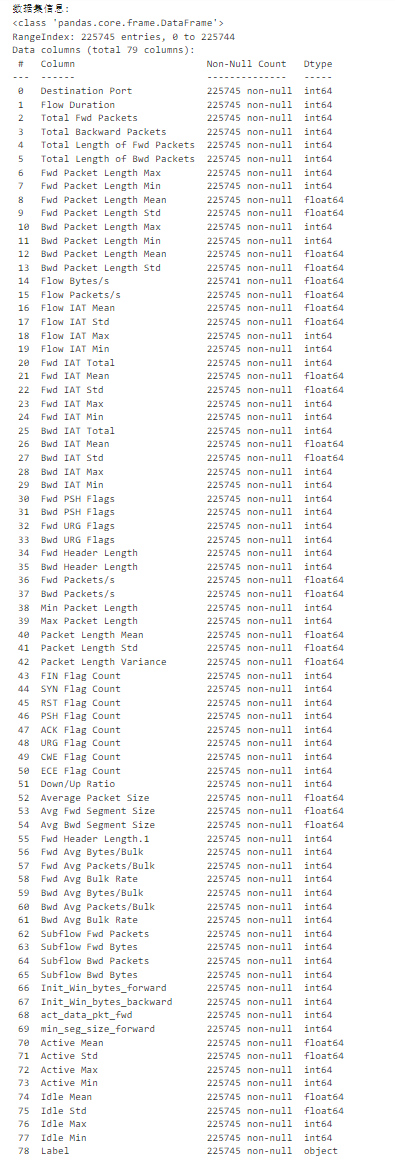


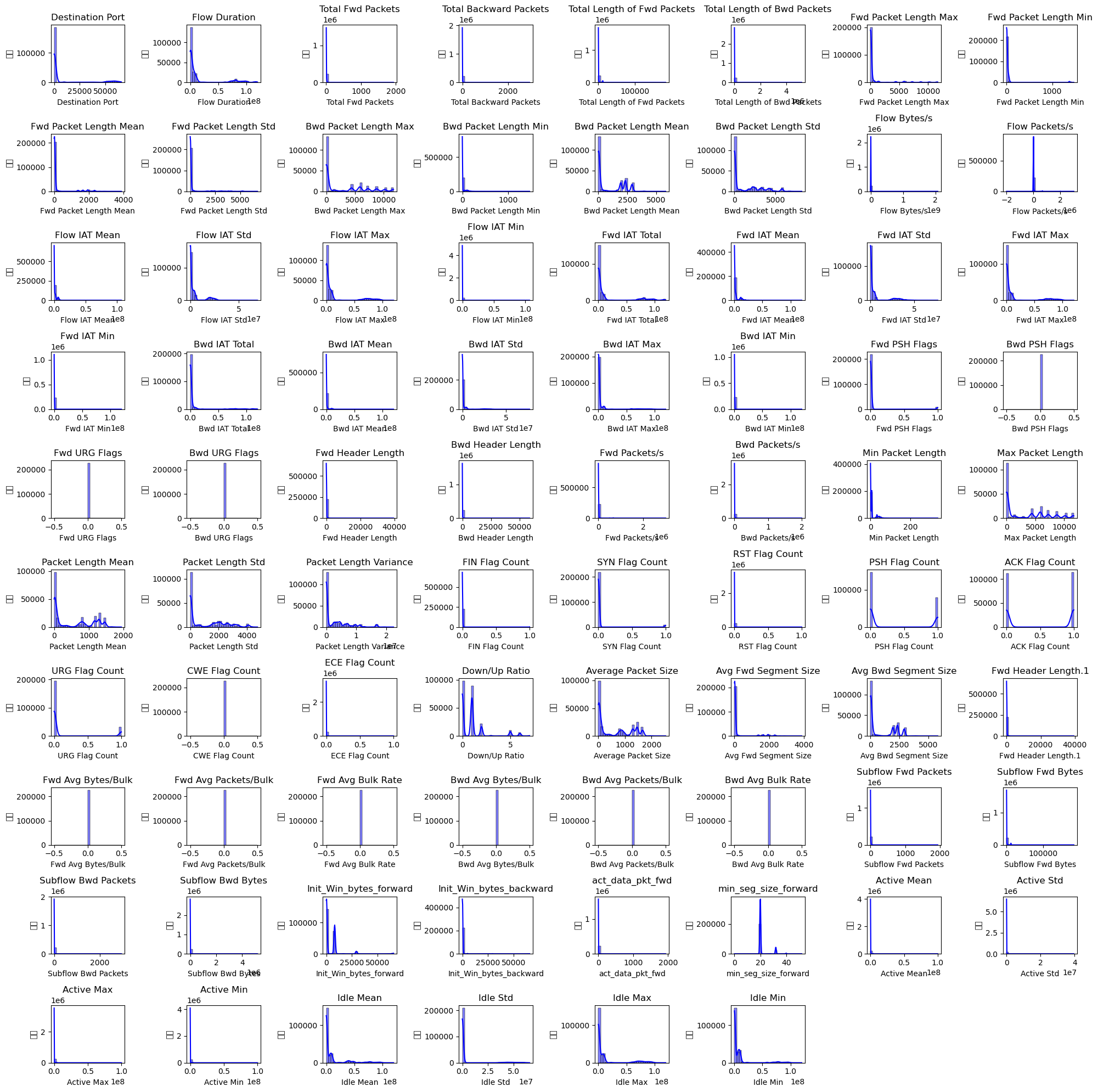
### 1.2 Generate histogram analysis of dataset fields

### From Lab5\_Code\_Note.ipynb #3.2

*#3.2*

1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. *# Reading the dataset*
6. df = pd.read\_csv('./Friday-WorkingHours-Afternoon-DDos.pcap\_ISCX.csv')
7. *# Remove leading and trailing spaces from column names*
8. df.columns = df.columns.str.strip()
9. *# Checking dataset information*
10. print("Dataset Information:")
11. print(df.info())
12. *# Selecting Numeric Features*
13. numeric\_features = df.select\_dtypes(include=[np.number]).columns
14. *# Set the size of the graphic*
15. plt.figure(figsize=(20, 20))
16. *#Generate histogram of numerical features*
17. for i, col in enumerate(numeric\_features):
18. plt.subplot(10, 8, i + 1)  *# Adjust the layout of sub-graphs according to the number of columns*
19. sns.histplot(df[col], bins=30, kde=True, color='blue')
20. plt.title(col)
21. plt.xlabel(col)
22. plt.ylabel('frequency')
23. plt.tight\_layout()
24. plt.show()





For data analysis of histograms, see Lab5\_Code\_Note.ipynb #3.3

## 1.3Extraction of features and labels, numerical encoding of labels, splitting of data sets, and standardization operations

### Extract features and labels From Lab5\_Code\_Note.ipynb #3.4

*#3.4*

1. '''
2. Step 1: Extract features and labels
3. First, we extract features (X) and target labels (y) from the dataset. Assuming the Label column is the target variable (i.e. the outcome to be predicted), we need to separate it from other features.
4. '''
5. *# Extracting features and labels*
6. if 'Label' in df.columns:
7. X = df.drop('Label', axis=1).values  *# Remove the 'Label' column from the dataset, and what remains are the features*
8. y = df['Label'].values               *# The 'Label' column is the target variable*
9. else:
10. raise ValueError("The 'Label' column is not found in the dataset, please check the column name or data structure.")
11. '''
12. Explanation:
13. X is the dataset containing all features (excluding the Label column).
14. y is the dataset containing the target label (i.e. the Label column).
15. If the Label column is not found in the dataset, an error will be thrown to ensure the correctness of the data.
16. '''

### Convert categorical labels to numerical codes From Lab5\_Code\_Note.ipynb #3.5

*#3.5*

1. '''
2. Convert categorical labels to numeric encodings
3. For most machine learning algorithms, labels need to be numeric. We use LabelEncoder to convert labels from text (such as BENIGN and DDoS) to numeric encodings.
4. '''
5. from sklearn.preprocessing import LabelEncoder
6. *# Convert categorical labels to numeric encodings*
7. le = LabelEncoder()
8. y = le.fit\_transform(y)
9. '''
10. LabelEncoder converts text labels into numeric encodings. For example, BENIGN might be encoded as 0 and DDoS might be encoded as 1.
11. This step ensures that the target variable (y) can be correctly processed by the machine learning model
12. '''

### Split the dataset into training and test sets From Lab5\_Code\_Note.ipynb #3.6

*#3.6*

1. '''
2. Split the dataset into training and test sets
3. In order to evaluate the performance of the model, we split the dataset into a training set and a test set. The training set is used to train the model, and the test set is used to evaluate the generalization ability of the model.
4. '''
5. from sklearn.model\_selection import train\_test\_split
6. *# The dataset is split into training and testing sets*
7. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
8. '''
9. Explanation:
10. The train\_test\_split function splits the dataset into training and test sets in a ratio of 80% and 20%.
11. random\_state=42 ensures that the results are reproducible (generates the same split results each time the code is run).
12. '''

### Data Standardization From Lab5\_Code\_Note.ipynb #3.7

*#3.7*

1. '''
2. Data normalization
3. Data normalization refers to scaling all features to the same scale. This is very important for many machine learning algorithms (such as neural networks, support vector machines, etc.) because these algorithms are sensitive to the scale of features.
4. '''
5. from sklearn.preprocessing import StandardScaler
6. *# Data Standardization*
7. scaler = StandardScaler()
8. X\_train = scaler.fit\_transform(X\_train)  *# Use the training set data to calculate the mean and standard deviation and apply it to the training set*
9. X\_test = scaler.transform(X\_test)        *11.# Transform the test set using the same mean and standard deviation*
10. '''
11. Explanation:
12. StandardScaler standardizes the data and scales it to a distribution with mean 0 and variance
13. In the training set, use fit\_transform to calculate and apply the standardization parameters (mean and standard deviation).
14. In the test set, use transform to apply the same standardization parameters (without recalculating the mean and standard deviation) for consistency.
15. '''
16. '''
17. The standardized training set and test set have been prepared and can be used for model training and evaluation. Standardized data helps improve the training efficiency and prediction performance of the model.
18. '''

# ANN Modeling analysis

### Summary: Use Keras to build a simple deep learning model to predict whether network traffic is normal (such as BENIGN) or an attack (such as DDoS)

## 2.1Building the ANN model

### From Lab5\_Code\_Note.ipynb #3.8

*#3.8*

1. '''
2. Build an ANN model
3. First, we use the Sequential model of Keras to build a multi-layer neural network.
4. The following is the specific code and its explanation:
5. '''
6. from tensorflow.keras.models import Sequential
7. from tensorflow.keras.layers import Dense, Dropout
8. *# Building an ANN model*
9. model = Sequential()
10. *# The first layer: a fully connected layer, containing 64 neurons, the input dimension is equal to the number of features in the training set, and the activation function uses ReLU*
11. model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))
12. model.add(Dropout(0.5))  *# Use Dropout regularization to randomly discard 50% of neurons to prevent overfitting*
13. *# The second layer: fully connected layer, containing 32 neurons, the activation function uses ReLU*
14. model.add(Dense(32, activation='relu'))
15. model.add(Dropout(0.5))  *# Using Dropout Regularization*
16. *# utput layer: 1 neuron, using sigmoid activation function, suitable for binary classification problems*
17. model.add(Dense(1, activation='sigmoid'))
18. '''
19. Explanation:
20. First layer (input layer and hidden layer): Use Dense layer, containing 64 neurons, and the activation function is ReLU (rectified linear unit). The input dimension (input\_dim) is the number of features of X\_train.
21. Dropout layer: Add a Dropout layer after each hidden layer to randomly drop neurons with a probability of 50%, which helps prevent overfitting.
22. Second layer (hidden layer): Use Dense layer again, containing 32 neurons, and the activation function is ReLU.
23. Output layer: Use Dense layer, containing 1 neuron, and the activation function is sigmoid, which is suitable for binary classification problems (the output range is between 0 and 1)
24. '''

## 2.2 Compiling the Model

### From Lab5\_Code\_Note.ipynb #3.9

*#3.9*

1. '''
2. Compile the model
3. After building the model, we need to compile it to specify the optimizer, loss function, and evaluation metric for training.'''
4. *# 编译模型*
5. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
6. '''
7. Explanation:
8. Optimizer (adam): Adam is an adaptive learning rate optimization algorithm that usually performs well in deep learning tasks.
9. Loss function (binary\_crossentropy): The binary cross entropy loss function is suitable for binary classification problems. It measures the distance between the probability distribution predicted by the model and the probability distribution of the true label.
10. Evaluation metric (accuracy): During each training process, we record the accuracy to monitor the performance of the model.
11. '''

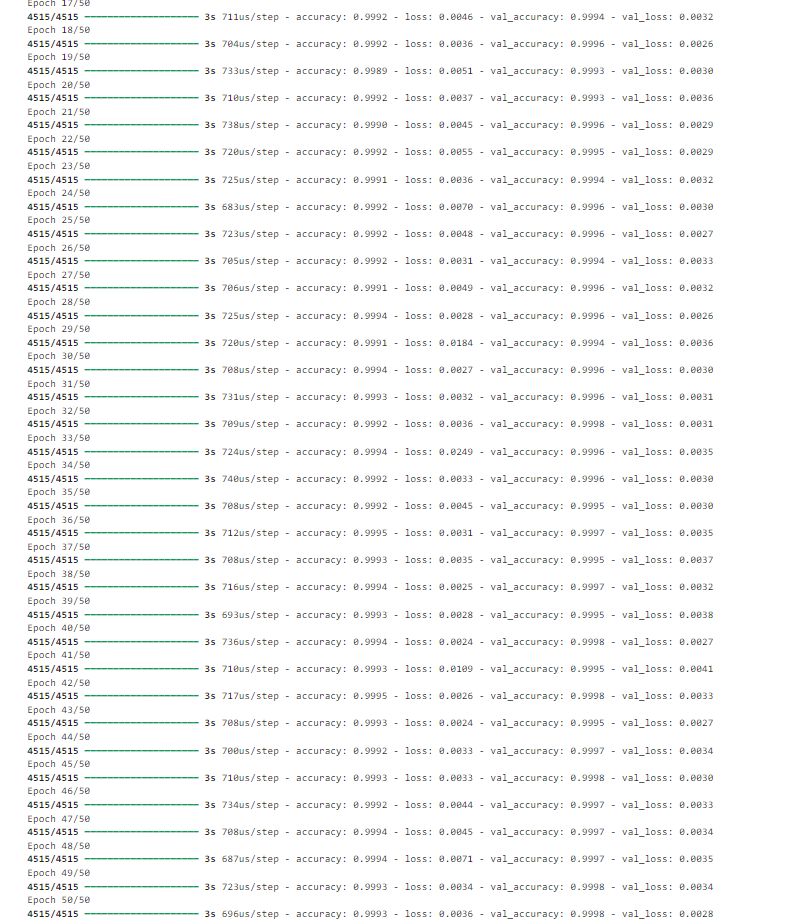
## 2.3 Training the model

### From Lab5\_Code\_Note.ipynb #3.10

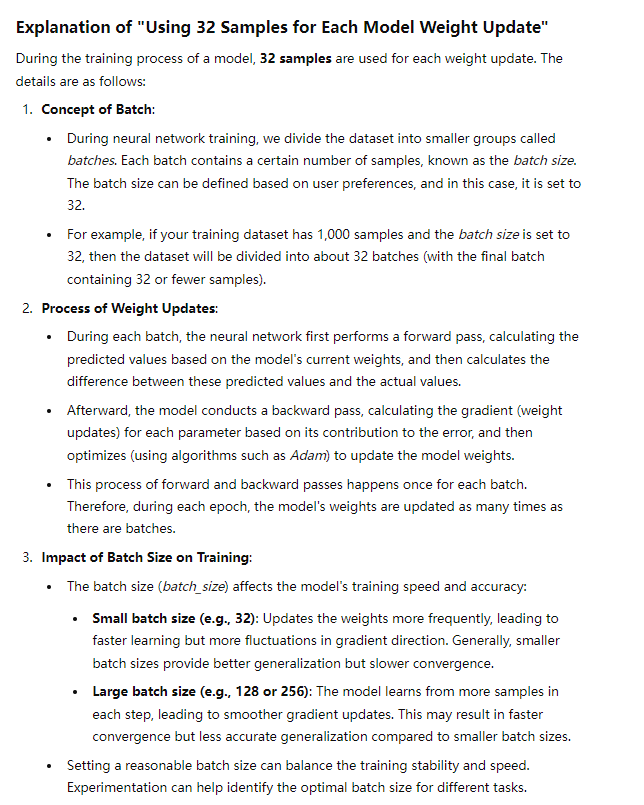
*#3.9*

1. '''
2. Next, we train the model using the training set data.
3. '''
4. *# Training the model*
5. history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)
6. '''
7. Explanation:
8. Training data: X\_train and y\_train.
9. epochs=50: The model will go through the entire training dataset 50 times.
10. batch\_size=32: 32 samples will be used each time the model weights are updated.
11. validation\_split=0.2: 20% of the data in the training set will be used for validation to monitor the performance of the model on unseen data (to prevent overfitting).
12. '''

### 训练过程截图：



### FAQ:



## 2.4 Model Evaluation:

### Generate predicted values From Lab5\_Code\_Note.ipynb #3.11

*#3.11*

1. '''
2. Model Evaluation
3. Generating Predictions
4. In model evaluation, we first use the trained model to make predictions on the test set.'''
5. from sklearn.metrics import classification\_report, confusion\_matrix
6. *# Generate predictions*
7. y\_pred = (model.predict(X\_test) > 0.5).astype("int32")
8. '''
9. Explanation:
10. model.predict(X\_test): Use the model to predict the test set X\_test. The output is a probability value between 0 and 1, indicating the probability of predicting the positive class (such as 1).
11. (model.predict(X\_test) > 0.5).astype("int32"): Convert the probability value to a binary label. If the probability is greater than 0.5, the prediction is 1, otherwise it is 0. Use astype("int32") to convert the result to an integer.
12. '''

### 打印分类报告 From Lab5\_Code\_Note.ipynb #3.12

*#3.12*

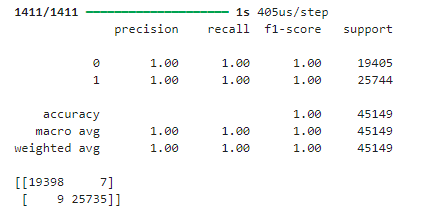
1. '''
2. Model evaluation
3. Print classification report
4. Use the classification\_report function to generate a classification report, which includes important indicators such as precision, recall, F1-score and support.
5. '''
6. *# 打印分类报告*
7. print(classification\_report(y\_test, y\_pred))
8. '''
9. Explanation:
10. Precision: The ratio of the number of positive samples correctly predicted by the model to the number of samples predicted as positive. It reflects the accuracy of the model when it does not misjudge negative samples as positive.
11. Recall: The ratio of the number of positive samples correctly predicted by the model to the number of samples that are actually positive. It reflects the ability of the model when it does not miss positive samples.
12. F1-score: The harmonic mean of precision and recall, which is used to weigh the importance of both. The higher the F1-score, the better the overall performance of the model.
13. Support: The number of samples that actually appear in each category in the test set'''

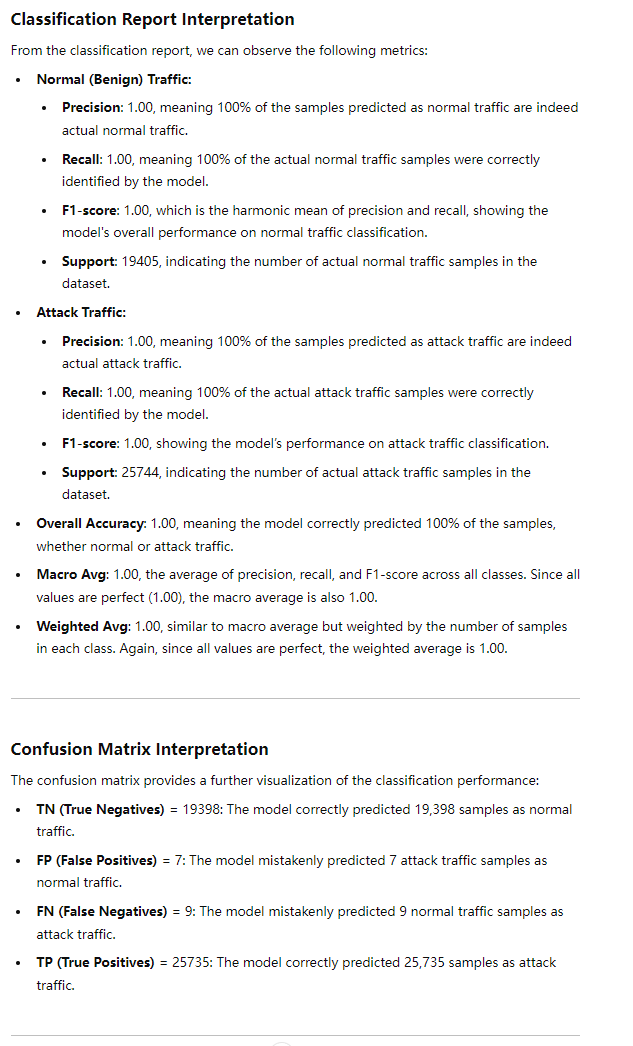
### 打印分类报告 From Lab5\_Code\_Note.ipynb #3.13

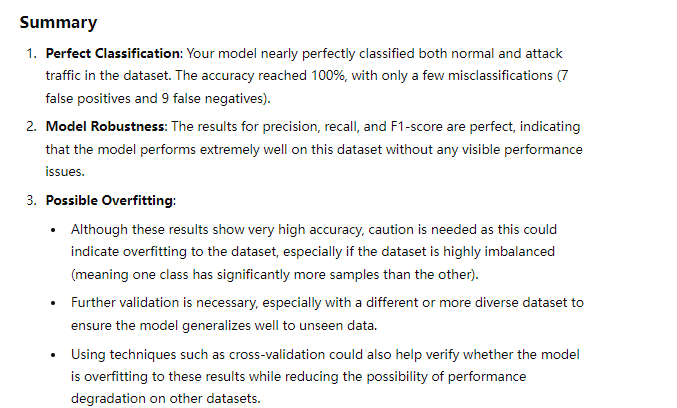
*#3.13*

1. '''
2. Model Evaluation
3. Printing Confusion Matrix
4. The confusion matrix is ​​used to show the match between the model's predictions and the true labels.'''
5. *# Printing confusion matrix*
6. print(confusion\_matrix(y\_test, y\_pred))
7. '''
8. Explanation:
9. The confusion matrix is ​​a visualization tool used to describe the performance of a classification model. It shows the number of true positives (True Positive, TP), false positives (False Positive, FP), true negatives (True Negative, TN), and false negatives (False Negative, FN).
10. The structure of the confusion matrix is ​​as follows:
11. Predicted as negative class    Predicted as positive class
12. Actual negative class  TN (True negative)    FP (False positive)
13. Actual positive class  FN (False negative)    TP (True positive)
14. Through the confusion matrix, you can intuitively see the types of classification errors made by the model (such as misclassification rate and missed classification rate)
15. '''

### 混沌矩阵结果：







## 2.5 Expanded content Visualization: Validation/Training Accuracy and loss function images Confusion matrix images

### Verification/training accuracy and loss function images From Lab5\_Code\_Note.ipynb #3.14

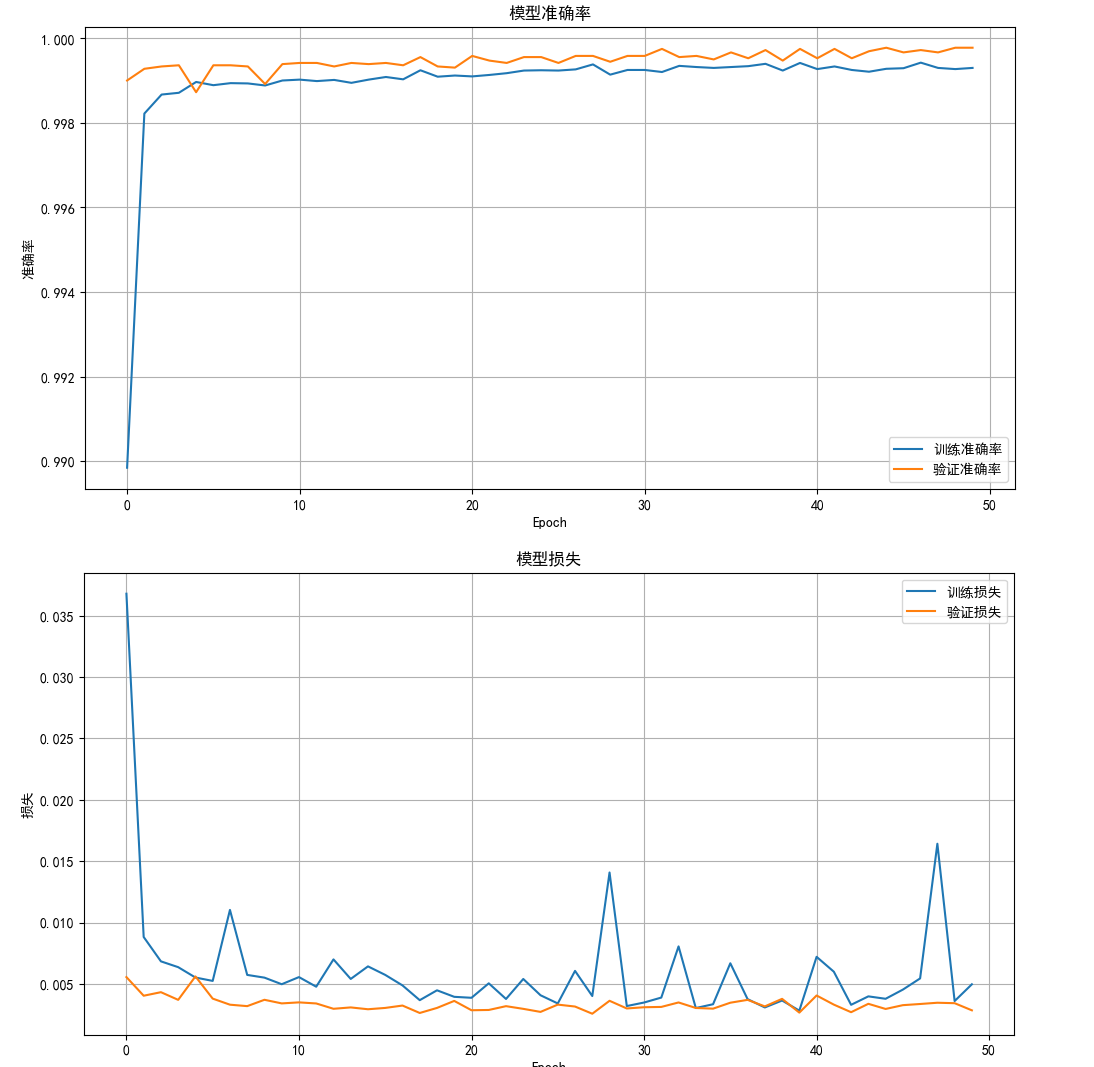
*#3.14 Visualization Expanded Content*

1. import matplotlib.pyplot as plt
2. import matplotlib.font\_manager as fm
3. *# Set the font to SimHei (bold) to display Chinese characters normally.*
4. plt.rcParams['font.sans-serif'] = ['SimHei']  *# Specify the default font*
5. plt.rcParams['axes.unicode\_minus'] = False  *# Solve the minus sign display problem*
6. *# Plot the changes in accuracy of training and validation*
7. plt.figure(figsize=(12, 6))
8. plt.plot(history.history['accuracy'], label='训练准确率')
9. plt.plot(history.history['val\_accuracy'], label='验证准确率')
10. plt.title('模型准确率')
11. plt.xlabel('Epoch')
12. plt.ylabel('准确率')
13. plt.legend()
14. plt.grid(True)
15. plt.show()
16. *# Plot the changes in training and validation loss*
17. plt.figure(figsize=(12, 6))
18. plt.plot(history.history['loss'], label='训练损失')
19. plt.plot(history.history['val\_loss'], label='验证损失')
20. plt.title('模型损失')
21. plt.xlabel('Epoch')
22. plt.ylabel('损失')
23. plt.legend()
24. plt.grid(True)
25. plt.show()

### 混淆矩阵图像 From Lab5\_Code\_Note.ipynb #3.15

1. *#3.15 Visualization Expanded Content*
2. from sklearn.metrics import ConfusionMatrixDisplay
3. *# Plotting the confusion matrix*
4. ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred)
5. plt.title('混淆矩阵')
6. plt.show()

## 可视图：



### Ps:Blue is training and orange is validation

## 1726488050977

# Extension: Hyperparameter Optimization Using Keras Tuner

## 3.1 Import the required libraries Model building function for hyperparameter optimization From Lab5\_Code\_Note.ipynb #3.16

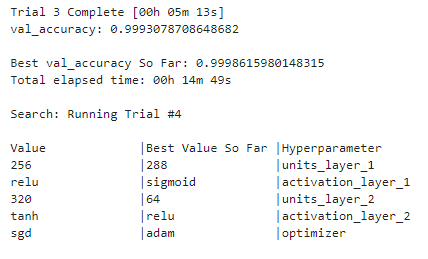
1. *#3.16*
2. '''
3. 3. Import libraries and define model building functions
4. '''
5. from keras\_tuner import RandomSearch
6. from tensorflow.keras.models import Sequential
7. from tensorflow.keras.layers import Dense, Input
8. import tensorflow as tf
9. '''
10. keras\_tuner: This is an open source library for hyperparameter search. We imported RandomSearch, which is a hyperparameter tuning method based on random search strategy.
11. tensorflow.keras.models.Sequential: Imported the Sequential model in Keras, which is a linear stacked neural network model.
12. tensorflow.keras.layers.Dense and tensorflow.keras.layers.Input: Imported the layer module of Keras, Dense is used for fully connected layer, and Input is used to define input layer.
13. tensorflow: TensorFlow is an open source deep learning framework, we use it to build and train neural networks.'''
14. '''
15. Define the model building function build\_model
16. Keras Tuner requires a model building function to dynamically build different model structures. This function accepts a HyperParameters object (ie hp) as a parameter to use different hyperparameters in the model.
17. '''
18. *# Model building function for hyperparameter optimization*
19. def build\_model(hp):
20. model = Sequential()
21. # Use the Input layer to define the shape of the input
22. model.add(Input(shape=(X\_train.shape[1],)))
23. '''
24. model = Sequential()：Creates a Sequential model instance.Sequential is a simple sequential model type provided by Keras and is suitable for linear stacked layers from input to output.
25. Input(shape=(X\_train.shape[1],))：Adds an input layer and defines the shape of the input data, where X\_train.shape[1] represents the number of input features.
26. Using Input layer explicitly defines the shape of the input for easy model visualization and debugging. '''
28. *# 第一层隐藏层，超参数优化层数和激活函数*
29. model.add(Dense(
30. units=hp.Int('units\_layer\_1', min\_value=32, max\_value=512, step=32),  *# 随机选择 32 到 512 个神经元*
31. activation=hp.Choice('activation\_layer\_1', values=['relu', 'tanh', 'sigmoid'])  *# 随机选择激活函数*
32. ))
33. '''
34. Dense: represents a fully connected layer (or dense layer). Each neuron in this layer is connected to all neurons in the previous layer.
35. units parameter:
36. hp.Int('units\_layer\_1', min\_value=32, max\_value=512, step=32): Here, the hyperparameter units\_layer\_1 is defined using the hp.Int method, which represents the number of neurons.
37. min\_value=32 and max\_value=512: Indicates that the number of neurons ranges from 32 to 512, with each increment being 32. Keras Tuner will randomly select a value within this range as the number of neurons in this layer.
38. activation parameter:
39. hp.Choice('activation\_layer\_1', values=['relu', 'tanh', 'sigmoid']): Here, the hyperparameter activation\_layer\_1 is defined using the hp.Choice method, which represents the type of activation function.
40. values=['relu', 'tanh', 'sigmoid']: indicates that the choice of activation function can be ReLU, Tanh or Sigmoid. Keras Tuner will randomly pick an activation function from these choices to build the model.'''
42. *# Second hidden layer, hyperparameter optimization of number of layers and activation function*
43. model.add(Dense(
44. units=hp.Int('units\_layer\_2', min\_value=32, max\_value=512, step=32), *#*  Randomly select 32 to 512 neurons
45. activation=hp.Choice('activation\_layer\_2', values=['relu', 'tanh', 'sigmoid'])  *# Randomly select activation function*
46. ))
47. '''
48. The usage of the units and activation parameters is similar to the first layer:
49. This layer also uses the hp.Int and hp.Choice methods to define adjustable hyperparameters. Keras Tuner will randomly select the number of neurons and activation function within the specified range and options.
50. '''
52. *# Output layer, sigmoid activation function is suitable for binary classification problems*
53. model.add(Dense(1, activation='sigmoid'))
54. '''
55. Dense: A fully connected output layer.
56. units=1: The output layer has only one neuron because we are solving a binary classification problem (normal traffic and attack traffic).
57. activation='sigmoid': Use the Sigmoid activation function to limit the output value between 0 and 1, indicating the probability that the sample belongs to a certain category.'''
58. *# 编译模型，优化器、损失函数和评估指标也作为超参数*
59. model.compile(
60. optimizer=hp.Choice('optimizer', values=['adam', 'rmsprop', 'sgd']),  *# 随机选择优化器*
61. loss='binary\_crossentropy',
62. metrics=['accuracy']
63. )
64. '''
65. optimizer parameters:
66. hp.Choice('optimizer', values=['adam', 'rmsprop', 'sgd']) : Here the hp.Choice method is used to define the choice of optimizer. The optimizer can be Adam, RMSprop or SGD, and Keras Tuner will randomly select one among these options.
67. loss='binary\_crossentropy' : Binary cross entropy loss function, suitable for binary classification problems.
68. metrics=['accuracy'] : Use accuracy as the metric for model evaluation.
69. '''
70. return model
71. '''
72. hp.Int: used to select integer hyperparameters. Here, the range is 32 to 512 (increases by 32 per step), which represents the number of neurons in the hidden layer.
73. hp.Choice: used to select discrete hyperparameters. Here, it is used to select activation functions (relu, tanh, sigmoid) and optimizers (adam, rmsprop, sgd).
74. We have defined an adjustable model architecture, and Keras Tuner will select the optimal hyperparameter combination through random search method.'''

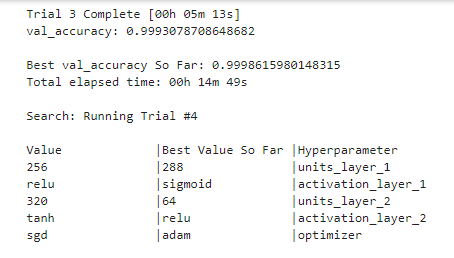
## 3.2 Start hyperparameter optimization From Lab5\_Code\_Note.ipynb #3.17

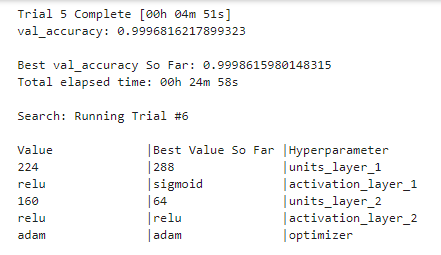
*#3.17*

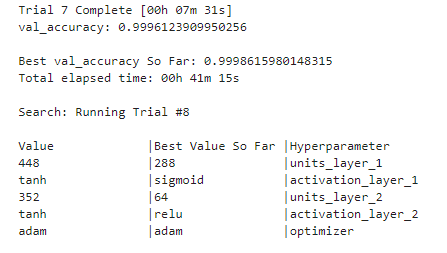
1. *# Hyperparameter optimization using Keras Tuner*
2. tuner = RandomSearch(
3. build\_model,  *# 模型构建函数*
4. objective='val\_accuracy',  *# 目标是验证集上的准确率*
5. max\_trials=10,  *# 最多尝试 10 次不同的超参数组合*
6. executions\_per\_trial=2  *# 每次试验的模型重复执行两次*
7. )
8. '''
9. build\_model: Model building function, which will dynamically create models with different hyperparameter combinations.
10. objective='val\_accuracy': Our optimization objective is the validation set accuracy (val\_accuracy). Keras Tuner will try to maximize this metric.
11. max\_trials=10: Specifies that at most 10 different hyperparameter combinations will be tried. Each attempt will build a model based on the build\_model function.
12. executions\_per\_trial=2: Specifies that the model for each trial will be repeated twice to ensure the stability of the results. The evaluation results of each hyperparameter combination will be taken as the average of two trials.
13. '''
14. *# Search for optimal model parameters*
15. '''
16. Search for optimal model parameters
17. Use the search method of Keras Tuner to perform hyperparameter optimization.'''
18. tuner.search(X\_train, y\_train, epochs=50, validation\_split=0.2)
19. '''
20. X\_train 和 y\_train: 训练数据集的特征和标签。
21. epochs=50: 每个模型训练的最大迭代次数为 50 次。通过这个设置，所有超参数组合的模型都会在 50 个 epochs 内进行训练。
22. validation\_split=0.2: 将训练数据中的 20% 用于验证。Keras Tuner 将使用这部分数据来评估模型在每个 epoch 后的表现，从而找到最优的超参数组合。
23. Keras Tuner 将使用随机搜索（Random Search）策略在指定的超参数空间中进行搜索，以找到最佳的模型配置
24. '''
25. *# Get the best model from the search results*
26. '''
27. Get the best model from the search results Once the search is complete, we can get the best model found.'''
28. best\_model = tuner.get\_best\_models(num\_models=1)[0]
29. '''
30. tuner.get\_best\_models(num\_models=1): Returns the num\_models models with the best performance. Here we take the first model ([0]), which is the best model.
31. best\_model: This model is trained with the best hyperparameter combination based on the accuracy (val\_accuracy) on the validation set.
32. '''
33. *# Train the best model*
34. history = best\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)
35. '''
36. X\_train and y\_train: features and labels of the training dataset.
37. epochs=50: the maximum number of iterations for retraining is 50. We can adjust this value as needed.
38. batch\_size=32: the number of samples used each time when updating the model weights.
39. validation\_split=0.2: 20% of the training data is used for validation to help monitor the model performance during training.
40. '''
41. '''
42. Hyperparameter optimization: Using Keras Tuner to search hyperparameters to find the optimal model configuration.
43. Search process: Keras Tuner selects the optimal hyperparameter combination based on the validation set accuracy, and trying different hyperparameter combinations through a random search method.
44. Best model training: Based on the best hyperparameter combination found, the model is further trained to fulfill its potential
45. '''

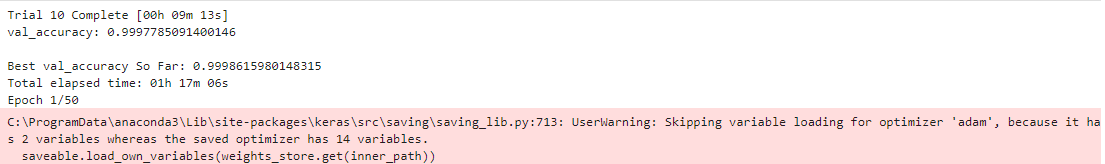
### After 10 random steps, the best parameters are found:











### The current best hyperparameter combination (after the second attempt) is still:

### Number of neurons in the first layer: 288

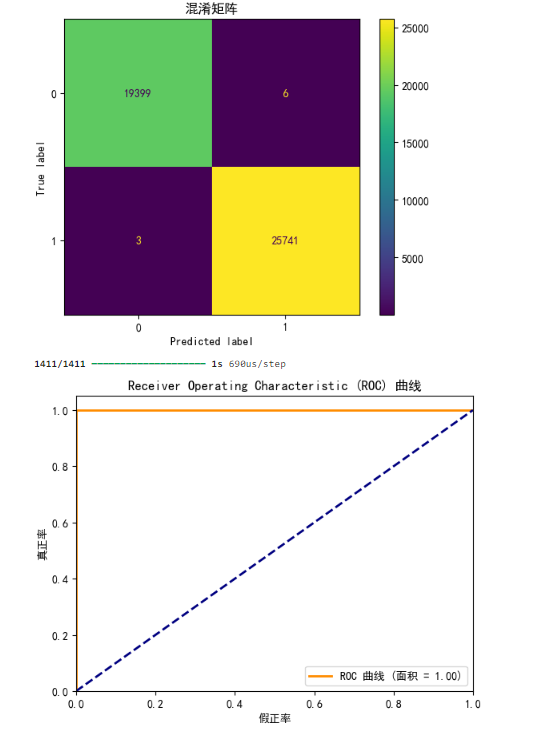
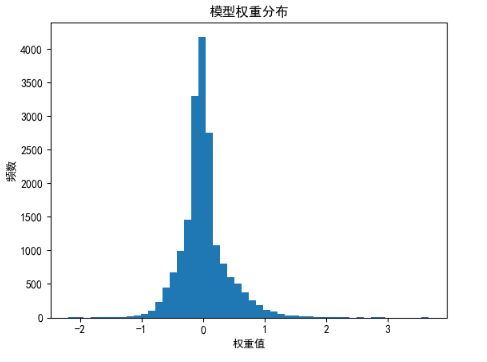
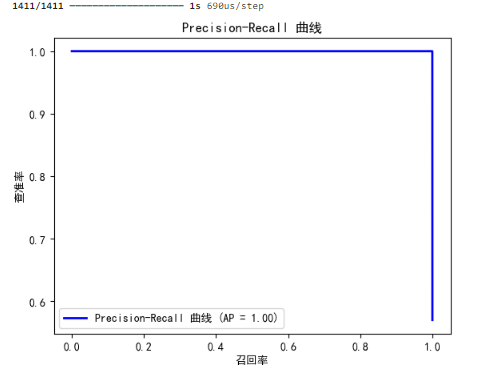
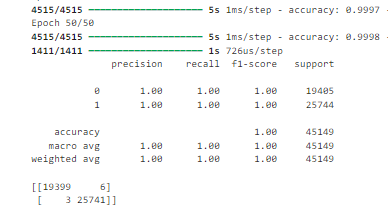
### Activation function in the first layer: sigmoid

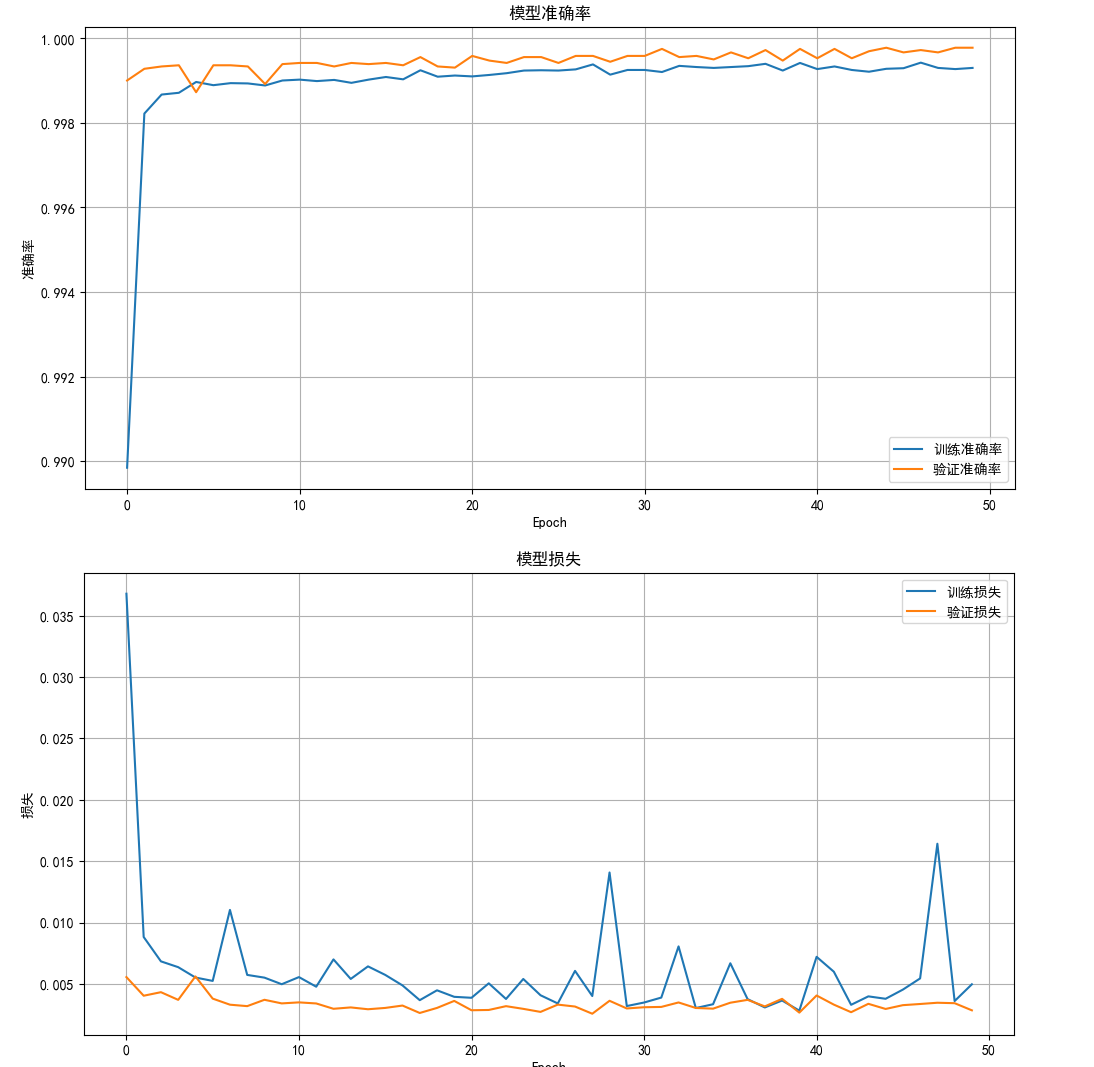
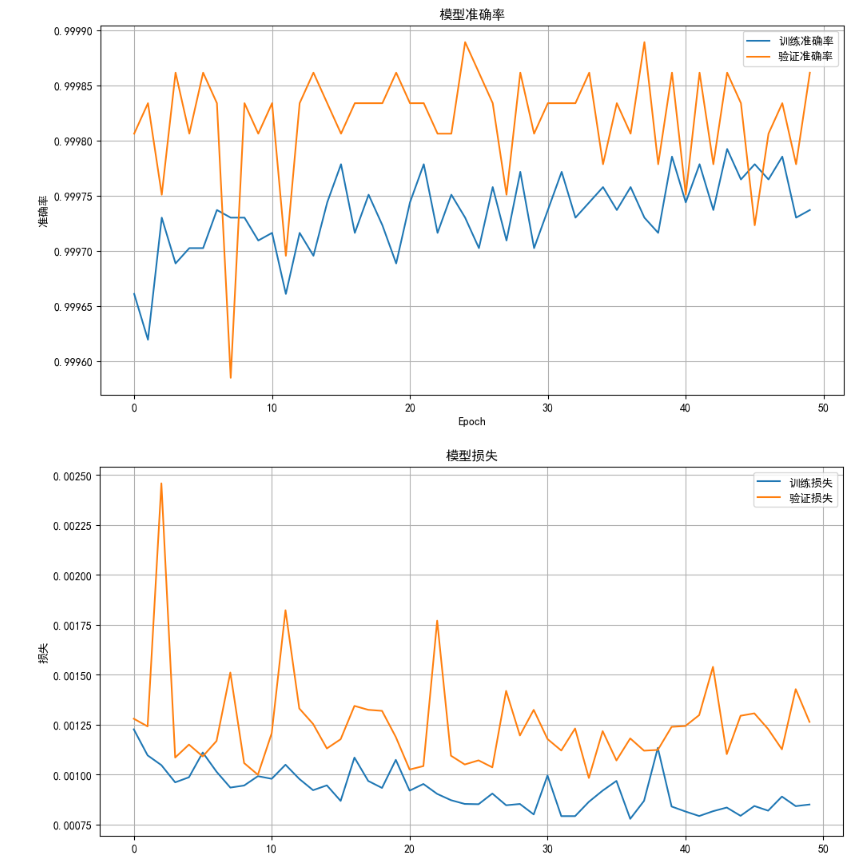
### Number of neurons in the second layer: 64

### Activation function in the second layer: relu

### Optimizer: adam

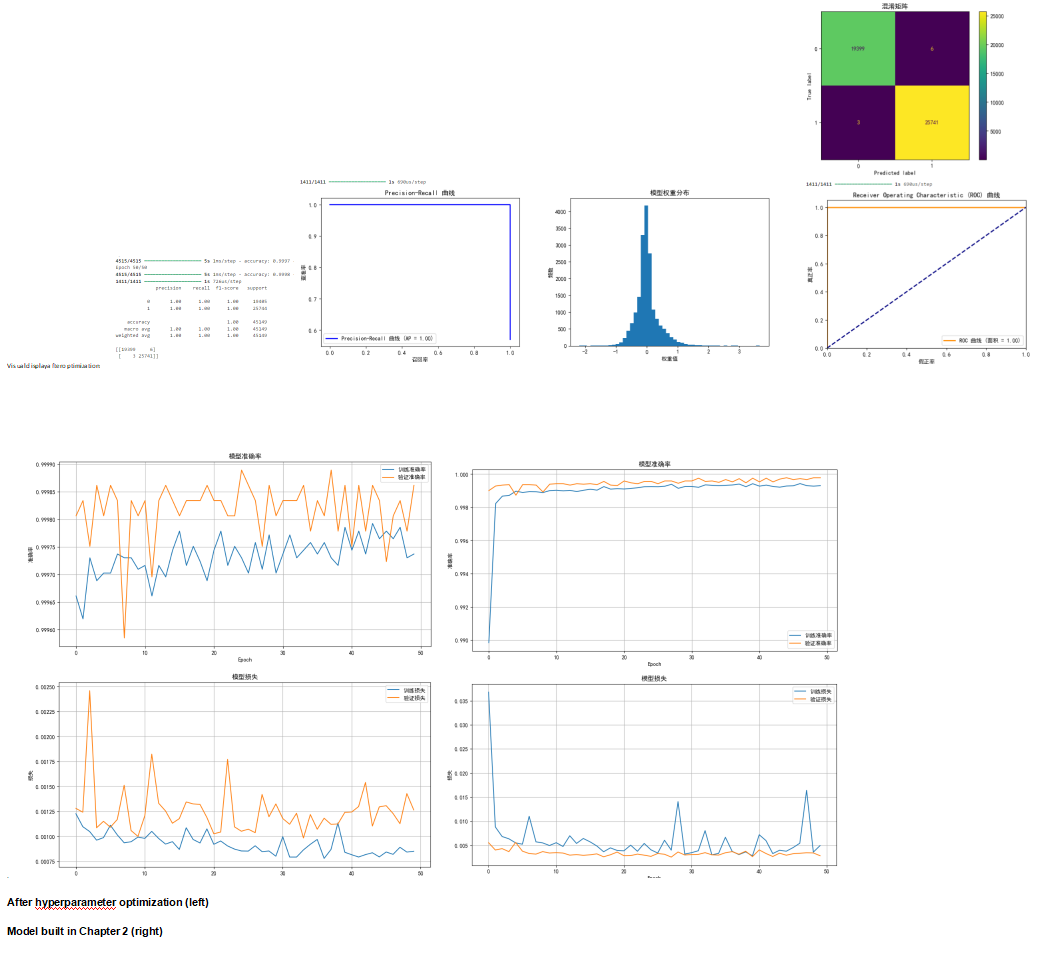
### After running with the best parameters, the visualization code is the same as the code in Chapter 2

Visual display after optimization:

.

### After hyperparameter optimization (left)

### Model built in Chapter 2 (right)



# 4.Comparative analysis of the original model and the randomly generated best model after modifying the number of hidden layers and their response to the newly added data set

## Code：<https://github.com/dqswordman/MUT_Neural_Networks_LAB/blob/main/Du_Hw.ipynb>

## 4.1 Generate new data

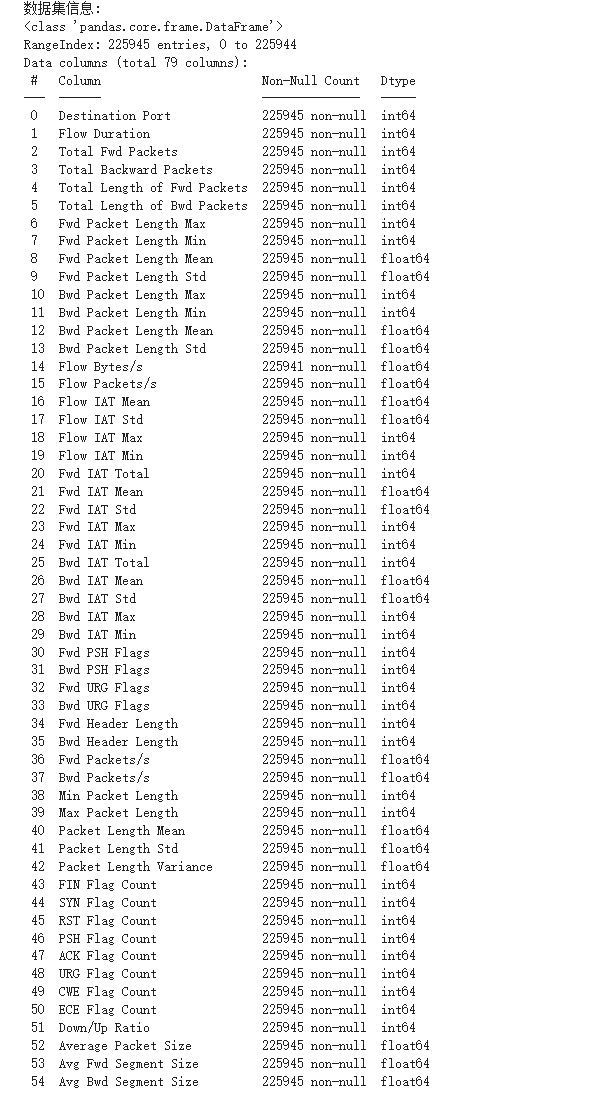
1. *#GitHub https://github.com/dqswordman/MUT\_Neural\_Networks\_LAB/blob/main/Du\_Hw.ipynb*
2. '''
3. dont run this part in jupyter!!!! broken the page
4. python code to add random data
5. import pandas as pd
6. import numpy as np
7. # Load original data set
8. file\_path = './Friday-WorkingHours-Afternoon-DDos.pcap\_ISCX.csv'
9. data = pd.read\_csv(file\_path)
10. # Create a new dataframe with 200 values ​​per column randomly drawn from the original dataset
11. new\_rows = pd.DataFrame({col: np.random.choice(data[col], size=200, replace=True) for col in data.columns})
12. # Add the newly generated 200 pieces of data to the original data set
13. augmented\_data = pd.concat([data, new\_rows], ignore\_index=True)
14. # Save the dataset after adding new data
15. augmented\_file\_path = './augmented\_data\_with\_200\_rows.csv'
16. augmented\_data.to\_csv(augmented\_file\_path, index=False)
17. !!!!!!!!by the way  from here to next note part  all dataset I used is wrong because I use "random" in lable ...!!!!!!!
18. '''

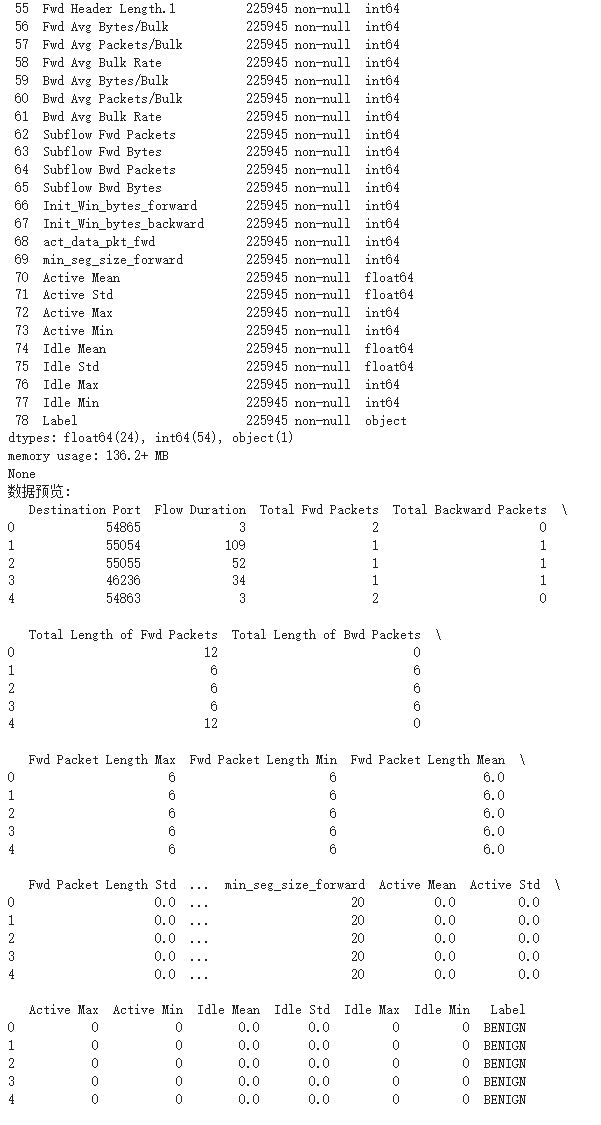
### Note：The label was also randomly generated by mistake and was manually modified later.

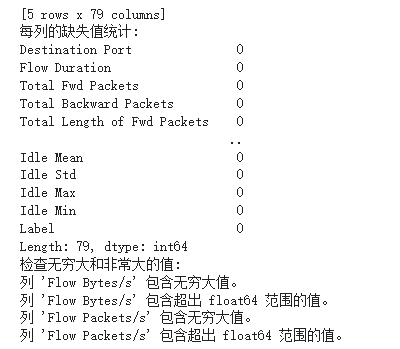
## 4.2 Dataset analysis, initialization

1. import pandas as pd
2. import numpy as np
3. from sklearn.model\_selection import train\_test\_split
4. from sklearn.preprocessing import StandardScaler, LabelEncoder
5. *# Read data*
6. df = pd.read\_csv('./corrected\_augmented\_data.csv')
7. *# Remove leading and trailing spaces from column names*
8. df.columns = df.columns.str.strip()
9. *# Check data overview*
10. print("数据集信息:")
11. print(df.info())
12. *# Display the first few rows of data to make it easier to understand the data content*
13. print("数据预览:")
14. print(df.head())
15. *# Check for missing values*
16. print("每列的缺失值统计:")
17. print(df.isnull().sum())
18. *# Handle missing values ​​(if there are missing values, we fill numeric features with the mean)*
19. numeric\_features = df.select\_dtypes(include=[np.number]).columns
20. df[numeric\_features] = df[numeric\_features].fillna(df[numeric\_features].mean())
21. *# Check for infinities and very large values ​​in your data*
22. print("检查无穷大和非常大的值:")
23. for col in numeric\_features:
24. if np.isinf(df[col]).any():
25. print(f"列 '{col}' 包含无穷大值。")
26. if (df[col] > np.finfo(np.float64).max).any():
27. print(f"列 '{col}' 包含超出 float64 范围的值。")
28. *# Replace infinity and very large values ​​with the mean of the column*
29. df.replace([np.inf, -np.inf], np.nan, inplace=True)
30. df[numeric\_features] = df[numeric\_features].fillna(df[numeric\_features].mean())
31. *# Extract features and labels*
32. if 'Label' in df.columns:
33. X = df.drop('Label', axis=1).values  *# 假设 'Label' 列是目标变量*
34. y = df['Label'].values
35. else:
36. raise ValueError("数据集中未找到 'Label' 列，请检查列名或数据结构。")
37. *# Convert categorical labels to numerical encodings (assuming 'Label' is a categorical variable)*
38. le = LabelEncoder()
39. y = le.fit\_transform(y)
40. *# The data set is split into training set and test set*
41. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
42. *# Data normalization*
43. scaler = StandardScaler()
44. X\_train = scaler.fit\_transform(X\_train)
45. X\_test = scaler.transform(X\_test)

### Output：





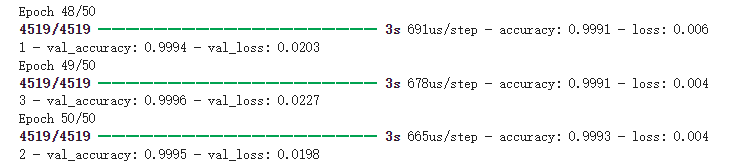


## 4.3 Extract features and labels & Convert categorical labels to numerical encodings

1. *# Extract features and labels*
2. if 'Label' in df.columns:
3. X = df.drop('Label', axis=1).values  Assume the 'Label' column is the target variable
4. y = df['Label'].values
5. else:
6. raise ValueError("'Label' column not found in dataset, please check column name or data structure.")
7. *# Convert categorical labels to numerical encodings (assuming 'Label' is a categorical variable)*
8. le = LabelEncoder()
9. y = le.fit\_transform(y)
10. *# Split the data set into training set and test set*
11. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
12. *# Data normalization*
13. scaler = StandardScaler()
14. X\_train = scaler.fit\_transform(X\_train)
15. X\_test = scaler.transform(X\_test)

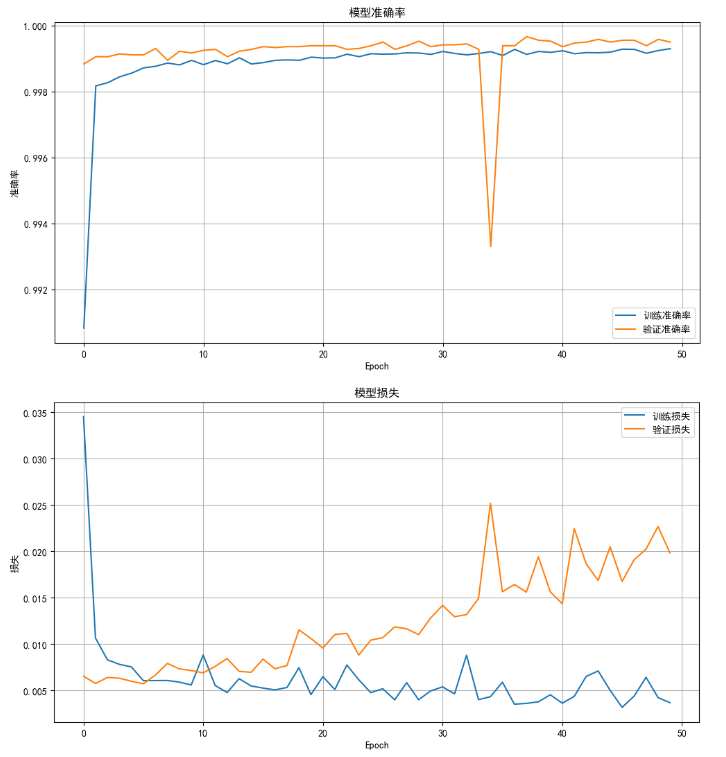
## 4.4 Build Original ANN model

1. # Build Original ANN model
2. model = Sequential()
3. model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))
4. model.add(Dropout(0.5))
5. model.add(Dense(32, activation='relu'))
6. model.add(Dropout(0.5))
7. model.add(Dense(1, activation='sigmoid'))  # It is a two-classification problem
8. # Compile model
9. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
10. # Training model
11. history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

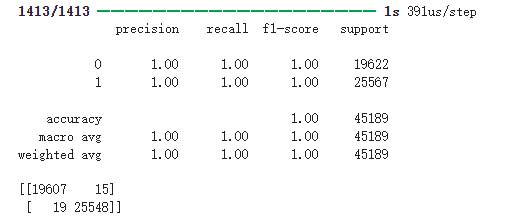


## 4.5 Visual analysis of original code

1. *#forget to change picture language sry  yellow is training data*
2. import matplotlib.pyplot as plt
3. import matplotlib.font\_manager as fm
4. *# Set the font to SimHei (black body) so that Chinese can be displayed normally*
5. plt.rcParams['font.sans-serif'] = ['SimHei']  *# Specify default font*
6. plt.rcParams['axes.unicode\_minus'] = False  *# Solve the problem of negative sign display*
7. *# 绘制训练和验证的准确率变化*
8. plt.figure(figsize=(12, 6))
9. plt.plot(history.history['accuracy'], label='训练准确率') *#training accuracy*
10. plt.plot(history.history['val\_accuracy'], label='验证准确率') *#Verification accuracy*
11. plt.title('模型准确率') *#Model accuracy*
12. plt.xlabel('Epoch')
13. plt.ylabel('准确率') *#Accuracy*
14. plt.legend()
15. plt.grid(True)
16. plt.show()
17. *# 绘制训练和验证的损失变化*
18. plt.figure(figsize=(12, 6))
19. plt.plot(history.history['loss'], label='训练损失') *#training loss*
20. plt.plot(history.history['val\_loss'], label='验证损失') *#Validation loss*
21. plt.title('模型损失') *#model loss*
22. plt.xlabel('Epoch')
23. plt.ylabel('损失') *#Loss*
24. plt.legend()
25. plt.grid(True)
26. plt.show()

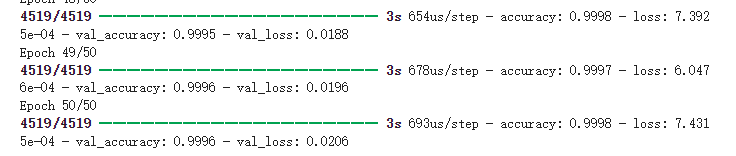


1. *# Model evaluation*
2. y\_pred = (model.predict(X\_test) > 0.5).astype("int32")
3. *# Print classification report and confusion matrix*
4. print(classification\_report(y\_test, y\_pred))
5. print(confusion\_matrix(y\_test, y\_pred))

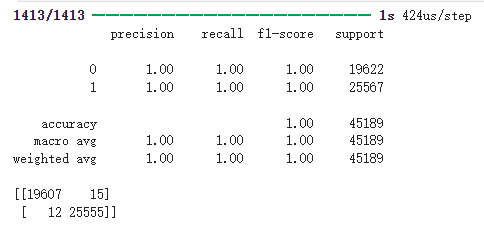


## 4.6 Build the best code ANN model found by hyperparameters

1. from keras.models import Sequential
2. from keras.layers import Dense, Dropout
3. *# Build new ANN models*
4. model = Sequential()
5. *# The first layer: 288 neurons, the activation function is sigmoid*
6. model.add(Dense(288, input\_dim=X\_train.shape[1], activation='sigmoid'))
7. *# Second layer: 64 neurons, activation function is relu*
8. model.add(Dense(64, activation='relu'))
9. *# Output layer: 1 neuron, activation function is sigmoid (for binary classification problems)*
10. model.add(Dense(1, activation='sigmoid'))
11. *# Compile the model, the optimizer is adam, the loss function is binary\_crossentropy, and the evaluation index is accuracy*
12. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
13. *# Training model*
14. history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

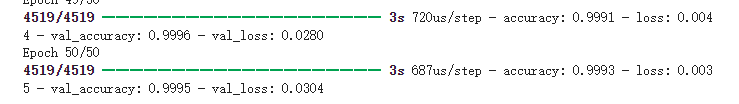


1. *# Model evaluation*
2. y\_pred = (model.predict(X\_test) > 0.5).astype("int32")
3. *# Print classification report and confusion matrix*
4. print(classification\_report(y\_test, y\_pred))
5. print(confusion\_matrix(y\_test, y\_pred))

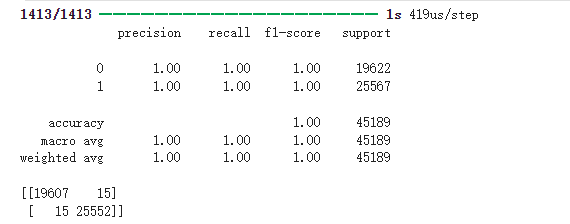


## 4.7 Constructing an ANN model with an additional hidden layer

1. #new model
2. from keras.models import Sequential
3. from keras.layers import Dense, Dropout
4. # Build an ANN model with added hidden layers
5. model = Sequential()
6. model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))
7. model.add(Dropout(0.5))
8. model.add(Dense(48, activation='relu'))  # New hidden layer, 48 neurons
9. model.add(Dropout(0.5))
10. model.add(Dense(32, activation='relu'))
11. model.add(Dropout(0.5))
12. model.add(Dense(1, activation='sigmoid'))
13. # Compile model
14. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
15. # Training model
16. history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)



1. *# Model evaluation*
2. y\_pred = (model.predict(X\_test) > 0.5).astype("int32")
3. *# Print classification report and confusion matrix*
4. print(classification\_report(y\_test, y\_pred))
5. print(confusion\_matrix(y\_test, y\_pred))



## 4.8 Summaary

1. '''
2. After correcting the dataset it seems to be correct
3. 1. Original model
4. False Positive = 15
5. False Negative = 19
6. Accuracy ≈ 9994025094602669
7. Error Rate ≈ 0.0752%
9. 2. Modify the model of hidden layers and parameters
10. False Positive = 15
11. False Negative = 15
12. Accuracy ≈ 99.93%
13. Error Rate ≈ 0.0664%。
15. 3. The best model based on the original model
16. False Positive = 15
17. False Negative = 12
18. Accuracy ≈ 99.94 %
19. Error Rate ≈ 0.0597%

## 4.9 conclusion

### Adding hidden layers improves model performance:

### - By adding hidden layers, the model performs better than the original code, which indicates that increasing the network depth allows the model to capture more complex features, thereby improving the ability of classification or regression.

### - This is in line with the general rule of deep learning, that is, deeper networks can learn richer feature expressions, thereby improving the performance of the model.

### Compared with the best model after hyperparameter optimization, there is still a gap in performance:

### - Although the performance is improved after adding hidden layers, there is still a gap in performance compared with the best model found after hyperparameter optimization based on the original code. This shows that simply increasing the network depth cannot completely make up for the performance deficiencies caused by improper model structure or hyperparameter selection.

### - Tuning hyperparameters (such as learning rate, optimizer, etc.) is crucial in improving model performance. Although increasing the depth of hidden layers is effective, it may not be enough to make up for poor hyperparameter selection.

### Increased computational cost:

### - As the number of hidden layers increases, the training time becomes longer, which indicates that the increase in model complexity brings higher computational costs. This shows that the efficiency and performance of the model are not linearly related. Increasing the network depth will increase the computational burden, especially when the model is not fully optimized, the increase in time cost may not be worth it.

### Conclusion

### - Increasing network depth can improve model performance, but the effect is limited: after adding hidden layers, the model has a stronger ability to learn complex features, but it cannot replace fine hyperparameter optimization. Deep networks are important, but appropriate hyperparameter adjustment is still the key.

### - Weighing performance and computational cost: more complex models require more computing resources and time, so in practical applications, while improving performance, we should pay attention to the computational cost of the model. An ideal model should find a balance between performance and efficiency, rather than blindly pursuing depth.

### - Hyperparameter optimization is still the key: increasing depth can effectively improve the model, but the final model performance still depends on the optimization of hyperparameters. Therefore, when improving model performance, we should not only pay attention to the adjustment of network structure, but also pay attention to the selection and adjustment of hyperparameters.