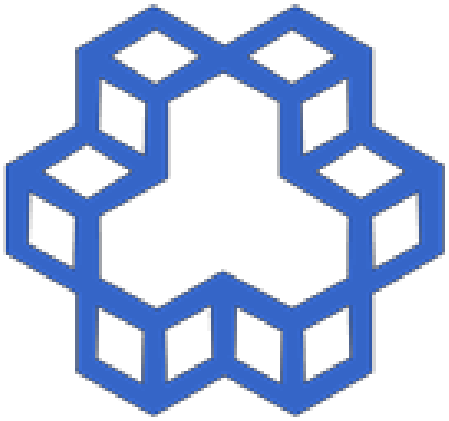


K. N. Toosi
University of Technology

Diagnosis of patients with fat

Challenge 1: Data Characteristics Analysis

In this project, we first need to look at the characteristics of the data. To do this, we need to know which features have the greatest impact on the output. For example, characteristics such as blood sugar, blood pressure, and age can have an important impact on the diagnosis of fatty liver.

Challenge 2: Data Preprocessing

At this stage, we reviewed and modified data that had characteristics with a value of zero. For example, zero blood pressure in a sample has no meaning and must be corrected by conventional preprocessing methods such as mean replacement or sample elimination.

Challenge 3: Data Segmentation and Standardization

The data were divided into two parts: test and training. The reasonable ratio for this task was 80% training and 20% testing. Then, the data were standardized using the StandardScaler model.

Challenge 4: Designing a Neural Network Model

The deep neural network model was designed using TensorFlow and Keras modules. In this model, two hidden layers with different numbers of neurons and a ReLU activation function were used. The output layer with a sigmoid activation function was also used. The hyperparameters of the model were adjusted manually according to various experiments and experiments. For example, the number of neurons in the hidden layers, the learning rate, and the number of layers were selected in such a way that the model achieves the best possible performance. In the following, I have given more detailed explanations about this.

The reason for choosing hyperparameters

- **Number of neurons**: The number of neurons in the hidden layers has been selected based on past experiences and the performance of similar models. 64 neurons in the first layer and 32 neurons in the second layer seem to be a suitable combination for this problem.

- **ReLU Activation Function:** This function is chosen because of its high convergence speed and better performance in deep models.
- **Sigmoid activation function**: Used for the output layer because the output of the model must be between 0 and 1 (due to binary classification).
- **Learning Rate**: Used by default ADAM which performs well on many issues.
- **Number of epoches**: 100 epoches are selected to make the model sufficiently trained.
- **batch_size:** 32 have been selected to strike a balance between the speed of training and the convergence of the model
- **callbacks:** EarlyStopping, to prevent overfitting, and ModelCheckpoint to store the best model.

In the following, I turned to a part of the code in which the value of some of these hyperparameters is known.

```
# Definition of callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')
```

In machine learning problems, the concept **of patience** refers to the discharge of units (neurons). When you set the value of **patience** to equal the number of epochs, the training will only stop if there is no improvement in the performance metric for X periods or consecutive repetitions. In other words, if the performance metric (e.g., validation accuracy) continues for X If the course or successive repetitions do not improve, the training will stop. This amount will help you avoid wasting your time and energy on useless training (by taking a break from the GPU).

```
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint])
```

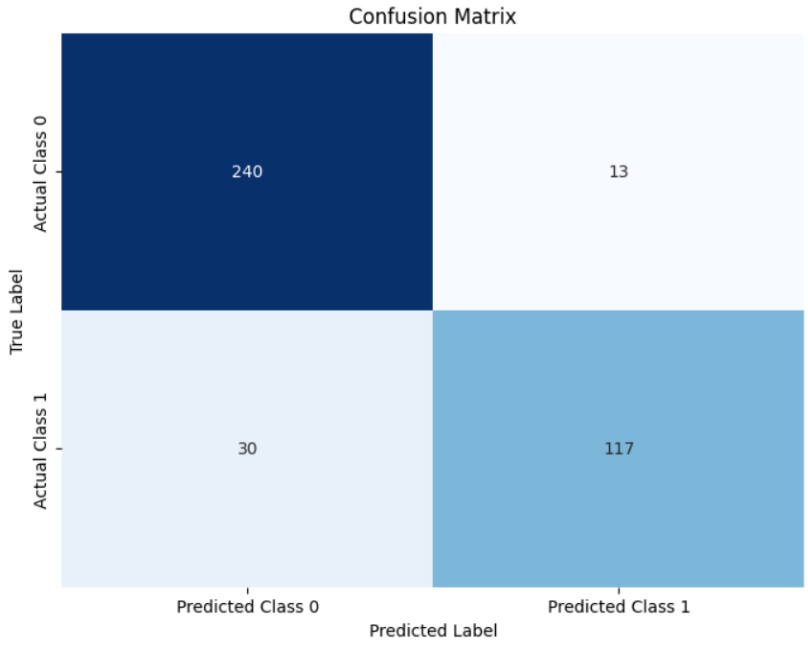
Challenge 5: Training and evaluating the model

The model was trained with training data and evaluated using methods such as the clutter matrix and accuracy and error graphs, which I have given below. The accuracy of the model in this project was 89.25%, which indicates the proper performance of the model. In the following, I will bring an evaluation of my other models. (In another section, the graphs are analyzed.)

```
Accuracy: 0.8925
Confusion Matrix:
[[240  13]
 [ 30 117]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92       253
           1       0.90      0.80      0.84       147

    accuracy                           0.89       400
   macro avg       0.89      0.87      0.88       400
weighted avg       0.89      0.89      0.89       400
```

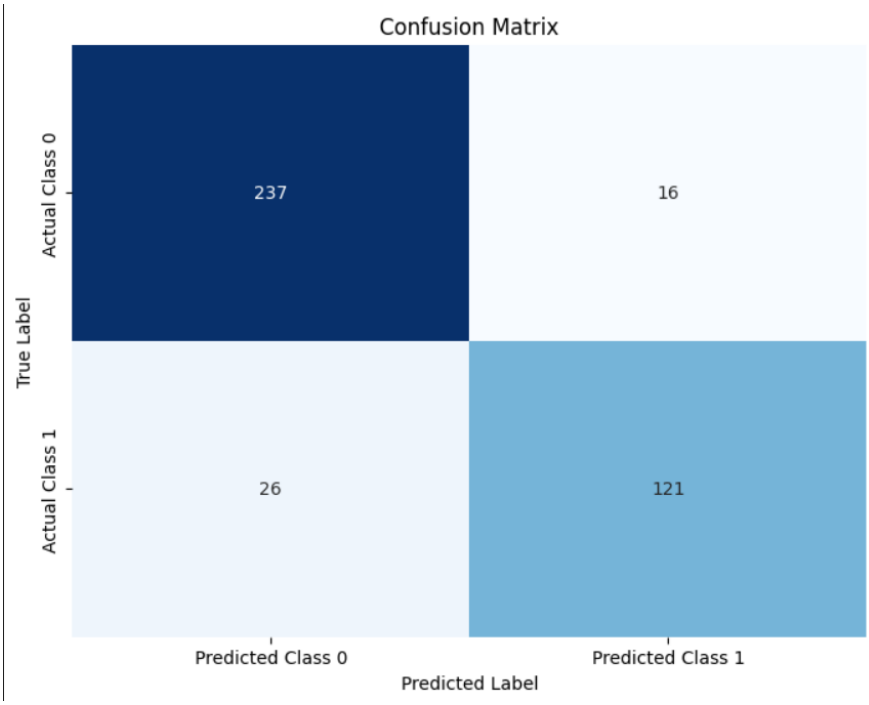


((Figure 1: Initial state))

```
Accuracy: 0.895
Confusion Matrix:
[[237  16]
 [ 26 121]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       253
           1       0.88      0.82      0.85       147

    accuracy                           0.90       400
   macro avg       0.89      0.88      0.89       400
weighted avg       0.89      0.90      0.89       400
```

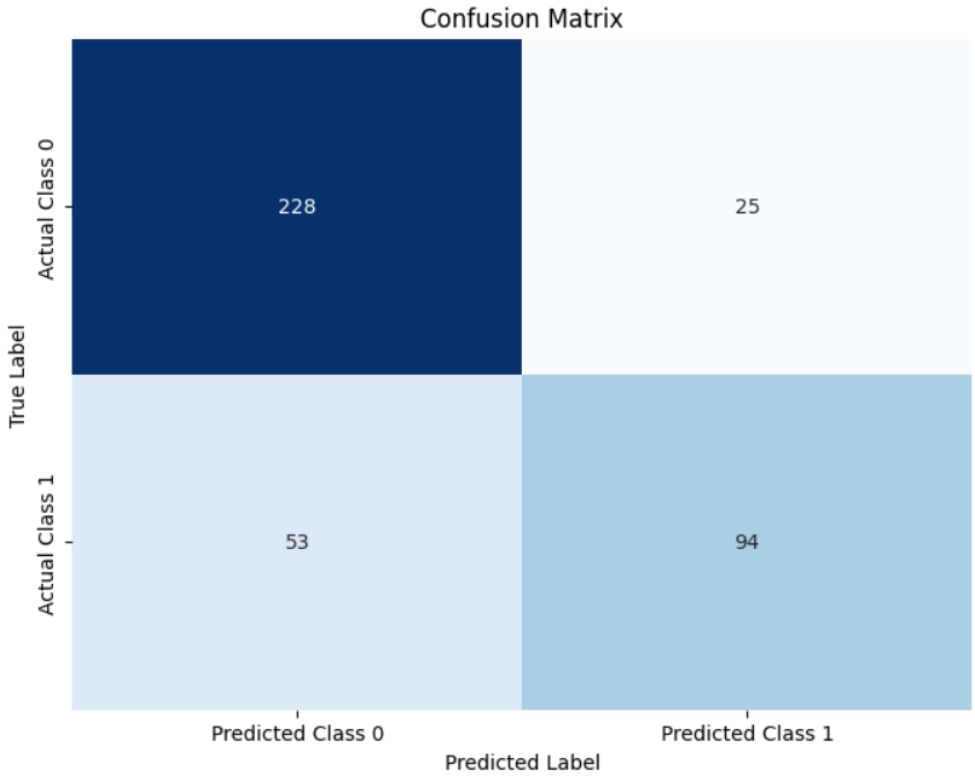


((Figure 2: Initial state + shuffle=True))

```
Accuracy: 0.805
Confusion Matrix:
[[228  25]
 [ 53  94]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       253
           1       0.79      0.64      0.71       147

    accuracy                           0.81       400
   macro avg       0.80      0.77      0.78       400
weighted avg       0.80      0.81      0.80       400
```

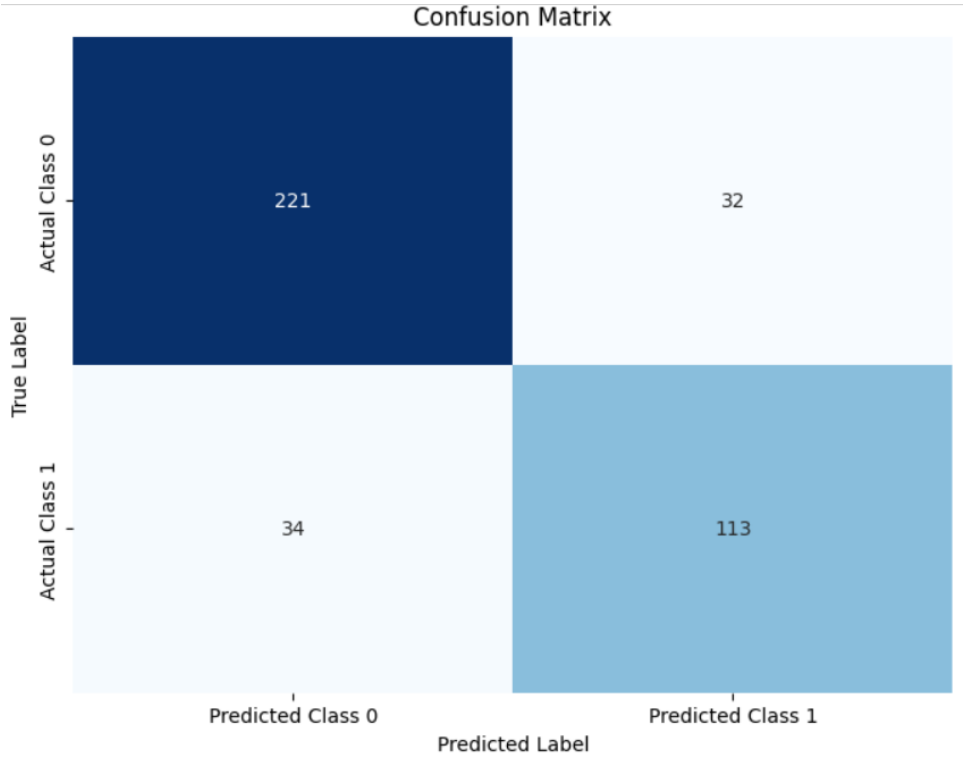


((Figure 3: Initial Mode + shuffle=True + Dropout))

```
Accuracy: 0.835
Confusion Matrix:
[[221  32]
 [ 34 113]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       253
           1       0.78      0.77      0.77       147

    accuracy                           0.83       400
   macro avg       0.82      0.82      0.82       400
weighted avg       0.83      0.83      0.83       400
```

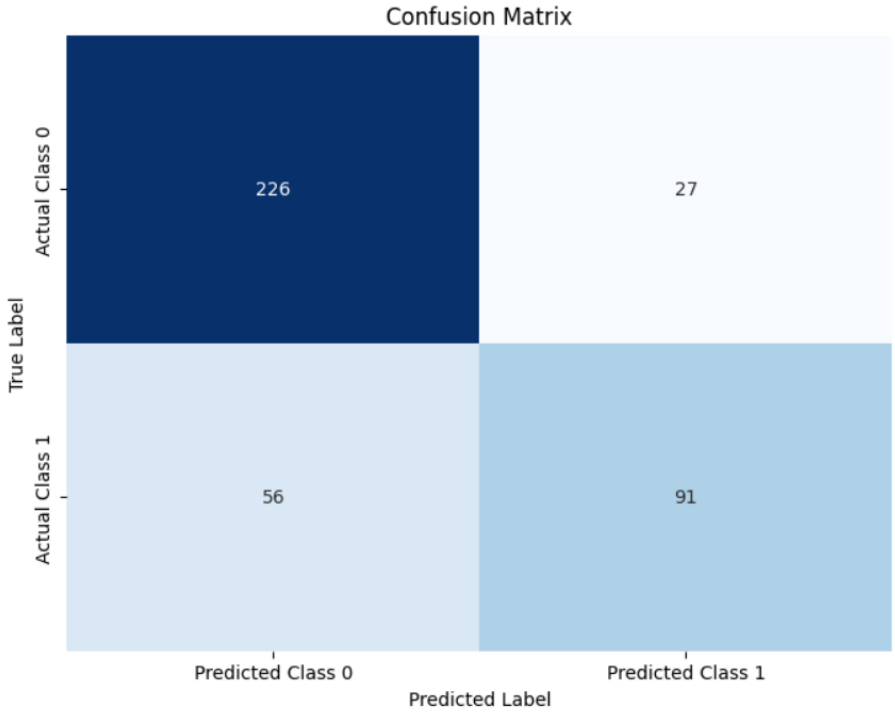


((Figure 4: Initial Mode + shuffle=True + Dropout ++ restore_best_weights=True Add Middle Layer))

```
Accuracy: 0.7925
Confusion Matrix:
[[226  27]
 [ 56  91]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       253
           1       0.77      0.62      0.69       147

    accuracy                           0.79       400
   macro avg       0.79      0.76      0.77       400
weighted avg       0.79      0.79      0.79       400
```



(((Figure 5: The number of AIPACs, layers, and neurons is lower than in the previous state))

```
Test accuracy: 0.8800
Confusion Matrix:
[[232  21]
 [ 27 120]]
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.90      0.92      0.91       253
     Class 1       0.85      0.82      0.83       147

    accuracy                           0.88       400
   macro avg       0.87      0.87      0.87       400
weighted avg       0.88      0.88      0.88       400
```

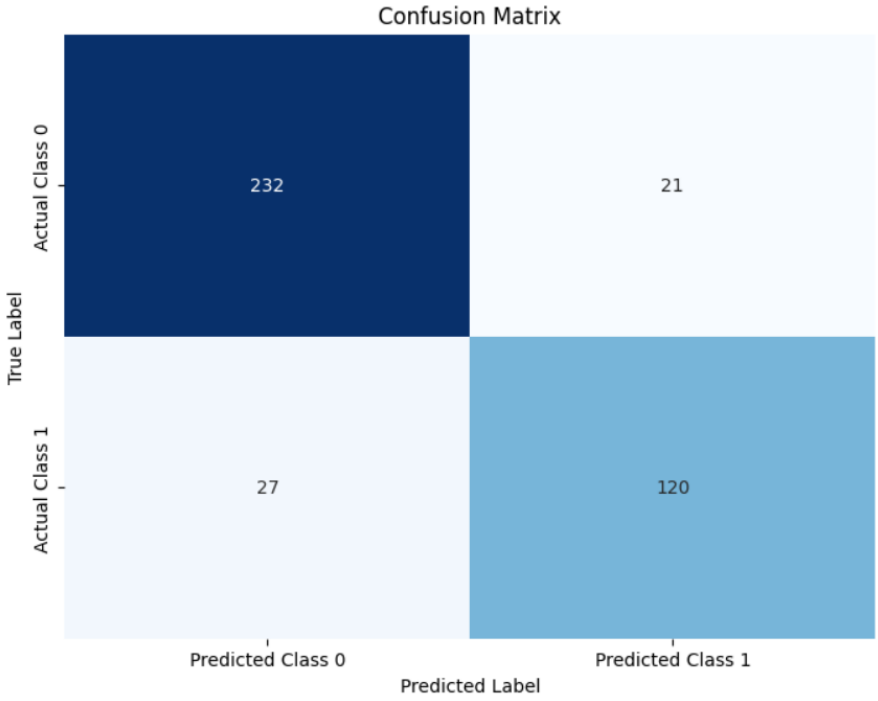


Figure 6: RandomizedSearchCV

Challenge 6: Exploring the Model's Prediction

The trained model was investigated using the test data. The test input was referred to the model and the output of the model was compared with the actual output. One sample of data that was written in the code was guessed correctly and I added two more test samples that they guessed correctly and then I brought the sample and the result of the output that was correct.

Sample with real output:

```
sample = [2, 81, 72, 15, 76, 30.1, 0.547, 25]                  # output: 0 ==> Great! You don't have fatty liver.
# tagir:
sample_2 = [2,  138,  62, 35, 0,  33.6, 0.127,  47]            # output: 1 ==> Oops! You have fatty liver.
# tagir:
sample_3 = [0,  84, 82, 31, 125,  38.2, 0.233,  23]            # output: 0 ==> Great! You don't have fatty liver.
```

Output Prediction:

```
Great! You don't have fatty liver.
Oops! You have fatty liver.
Great! You don't have fatty liver.
```

Challenge 7: Analysis of Model Errors

In disease diagnosis models, there are two main types of errors: false positives and false negatives. In this project, we need to analyze which of these errors will lead to worse events. In my opinion, a false negative error (diagnosing the patient as healthy) can have more risks, because the patient is left untreated. (In the Confusion Matrix, you can find out if the things that are most important to us are better recognized or not?)

To improve the model and reduce these errors, methods such as augmenting data, using more complex models, improving preprocessing techniques, and using regularization adjustment can be used, each of which I have given a brief description of below.

Improving Model Accuracy

To improve the accuracy of the model and reduce false positives and false negatives, the following methods can be used:

- **Data augmentation**: Collecting more data or using data augmentation techniques.
- **Use more complex models**: Experiment with different models, such as deeper neural networks or models based on reinforcement learning.
- **Improved preprocessing techniques**: Using more advanced pre-processing methods such as techniques for filling in missing values or better normalization.
- **Using Regulation Adjustment**: Adding dropout layers or using L1 and L2 ordering techniques to avoid overfitting.

Challenge 8: Studying Keras Documentation

To learn more about the Keras library, we went to its official documentation and looked at two important callbacks, namely EarlyStop and ModelCheckpoint. These two callbacks help us to have a more optimal model and avoid overfitting. I will give a more detailed explanation of it below.

The EarlyStop and ModelCheckpoint methods, which are used as callbacks in the TensorFlow/Keras library, are useful when training neural network models to improve model quality and performance and prevent problems such as overfitting.

1. EarlyStopping:

EarlyStopping is a callback in Keras that is used to stop the model from being trained due to progress during the model training. This callback automatically monitors whether the model's performance is improving on the validation data. If there is no improvement, the model stops to prevent overfitting occurs.

Important parameters of EarlyStop include:

- ● The metric used for monitoring, such as val_loss or val_accuracy.
- **:p atience** the number of epochs that the model must continue without improvement before it stops.
- **:restore_best_weights**If this parameter is true, the model weights will be returned to when it performed best.

: ModelCheckpoint.2

ModelCheckpoint is also a callback in Keras that is used to store model weights during training. This callback periodically monitors whether the model's performance is improved on validation data. If an improvement is seen, the model weights are stored.

Important parameters of ModelCheckpoint include:

- **filepath:** file path to save model weights.
- ● The metric used for monitoring, such as val_loss or val_accuracy.
- **:save_best_only** If it is true, only the weights of the best model based on the monitored metric will be saved.

Usage : callbacks

To use these callbacks, you add them as a list to the callbacks parameter in your model's fit function. Monitoring the model's progress over training time and storing the best weights can have major improvements in your model's performance and quality.

Code Description:

I put part of the code description in the code in the form of comments, and at the end of all the codes, I wrote a general explanation, which I will also include here, which is as follows:

```
Code Description

Importing libraries:
:p andas for data management.
:train_test_split for splitting data.
StandardScaler for data standardization.
Sequential, :D ense for building neural network models.
EarlyStop and ModelCheckpoint for callbacks.
confusion_matrix, classification_report, accuracy_score: to evaluate the model.
matplotlib.pyplot ::for drawing graphs.
Adam : To optimize the model.

Data Uploading:
Upload data from CSV file.
Separate the attributes (X) and tags .( y)

Dividing the data into training and testing:
The data were divided into 80% training and 20% testing.

Data standardization:
Standardization of data with . StandardScaler.

Definition of model build function to adjust hyperparameters:
Building a Neural Network Model with Variable Layers and the .relu Activation
Function
Adjust the learning rate using . Adam
```

```
Definition of callbacks:
Definition of EarlyStopping and . ModelCheckpoint

Model Training:
Model training using training and validation data.

Model Evaluation:
Forecasting with model and calculation accuracy, clutter matrix, and classification
reporting.

Drawing Diagrams of Education History:
Plotting Epoch-Loss and . Epoch-Accuracy
```

In a few cases, we evaluate the model to arrive at the best case.

i. The hyperparameters are as follows:

```
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint])
```

Neural Network Model Design:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Designing a Neural Network Model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

This code defines a neural network model using the TensorFlow and Keras library. I will explain more below.

1. First layer (Dense(64, input_dim=X_train.shape[1], activation='relu')):

- **Number of Neurons :(64)** The choice of 64 neurons in the first layer is based on experience and the prevalence of this number in many similar issues. This

number is usually enough to begin with and can well learn input characteristics.

- **Input: (input_dim=X_train.shape[1])** This parameter specifies the number of input features of the data, which is determined based on the shape of the training data.
- **Activation :(activation='relu')** The use of the ReLU activation function is due to its useful features such as preventing the gradient vanishing problem and speeding up learning.

2. Second layer :(Dense(32, activation='relu'))

- **Number of neurons :(32)** Reducing the number of neurons in the second layer relative to the first layer can help slightly reduce the complexity of the model and also help prevent overfitting.
- **Activation :(activation='relu')** The use of ReLU in this layer continues to maintain consistency and consistency between layers.

3. Output layer (Dense(1, activation='sigmoid')):

- **Number of neurons: (1)** Because your problem is a binary classification problem, one neuron in the output layer is enough.
- **Activation: (activation='sigmoid')** The use of the sigmoid function for the output is because it limits the output value to a range between 0 and 1, which is suitable for binary categorization issues.

Why were so many neurons chosen?

The number of neurons in each layer is usually selected based on previous experiences as well as trial and error. In many cases, a simple structure with fewer neurons is selected first, and then the number of neurons and layers is optimized using cross-validation and model adjustment.

Model Improvement

If you want to improve the model, you can change the number of layers and neurons, use regularization techniques such as dropout, or explore even more complex architectures such as deep neural networks. To do this, you can use tools such as GridSearchCV or RandomizedSearchCV for optimal model settings.

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

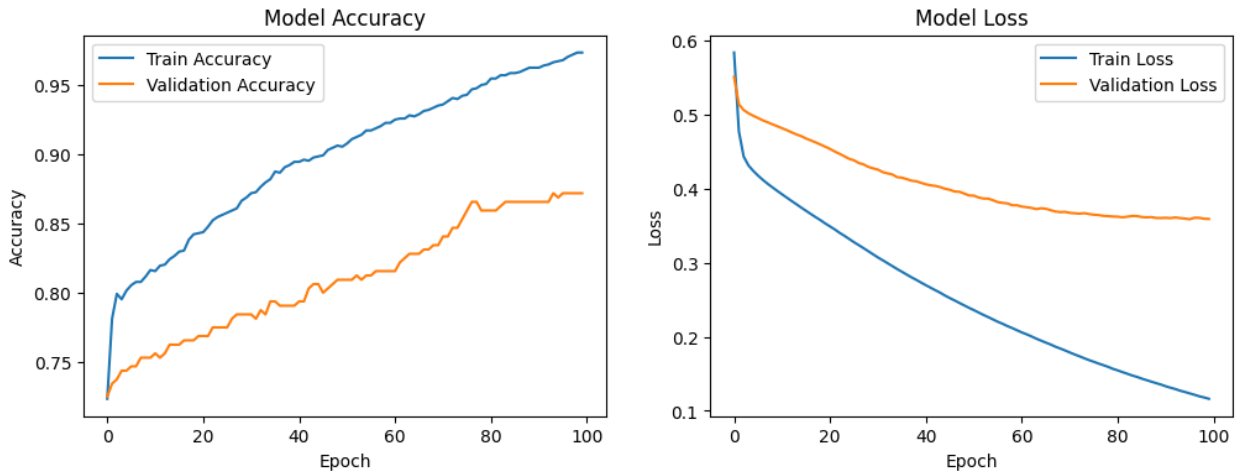


Figure 7: Epoch-Accuracy and Epoch-Loss Modulation for Mode 1

Analysis of Charts:

The graph provided includes two sub-charts: one for the accuracy of the model and the other for the losses during the 100 epoch training.

Model Accuracy Chart:

Training accuracy (blue line): This graph shows that the accuracy of the model is steadily increasing, and the model's performance on the training data is gradually improving. By the end of the training (Epok 100), the training accuracy is approaching around 0.98.

Validation accuracy (orange line): This accuracy also increases as the epochs pass but at a slower rate than the training accuracy. The validation accuracy of about Epok 80 reaches the plateau state, indicating that the model is approaching convergence. By the end of the training, the validation accuracy is about 0.88.

Brief definition of plateaus: refers to the smooth areas in the error function that slow down the convergence rate in optimization, or in other

words, when the error function remains almost constant in a large parameter space, gradient-based optimization methods fail to make meaningful updates.

Model Loss Chart:

Training loss (blue line): The training loss is continuously decreasing, which is a good sign that the model is learning and reducing errors on the training data. By the end of the training, the training loss is very small, which indicates the model's good fit to the training data.

Validation Loss (Orange Line): The validation loss decreases first but after Epok 60 it reaches a plateau state and increases slightly. This suggests that the model may start to overfitting the training data, as it performs well on the training set but does not perform more effectively on the validation set.

Analysis:

Overfitting: The large distance between the training accuracy and the validation accuracy, as well as the large distance between the training loss and the validation loss, increases the probability of overfitting. The training accuracy is higher than the validation accuracy, and the validation loss has reached the plateau state, which is a common symptom of overfitting.

Model performance: Despite overfitting, the model represents good generalization with a validation accuracy of around 0.88 and a low validation loss. To further improve the model, techniques such as dropout, early stop, or regularization can be used.

ii. The hyperparameters are as follows. (shuffle=True added.)

```
# Definition of callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint], shuffle=True)
```

shuffle=True means **randomizing the data**. When we enable this parameter, our model will view the training data in random order per training period (epoch). This can improve the accuracy of the model and possibly hide some conflicts in the data. In other words, it prevents the model from learning the order of the data.

Neural Network Model Design:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Designing a Neural Network Model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

1First layer (Dense(64, input_dim=X_train.shape[1], activation='relu')):

- **Number of Neurons :(64)** The choice of 64 neurons in the first layer is based on experience and the prevalence of this number in many similar issues. This number is usually enough to begin with and can well learn input characteristics.
- **Input: (input_dim=X_train.shape[1])** This parameter specifies the number of input features of the data, which is determined based on the shape of the training data.
- **Activation :(activation='relu')** The use of the ReLU activation function is due to its useful features such as preventing the gradient vanishing problem and speeding up learning.

2. Second layer :(Dense(32, activation='relu'))

- **Number of neurons :(32)** Reducing the number of neurons in the second layer relative to the first layer can help slightly reduce the complexity of the model and also help prevent overfitting.
- **Activation :(activation='relu')** The use of ReLU in this layer continues to maintain consistency and consistency between layers.

3. Output layer (Dense(1, activation='sigmoid')):

- **Number of neurons: (1)** Because your problem is a binary classification problem, one neuron in the output layer is enough.
- **Activation: (activation='sigmoid')** The use of the sigmoid function for the output is because it limits the output value to a range between 0 and 1, which is suitable for binary categorization issues.

Why were so many neurons chosen?

The number of neurons in each layer is usually selected based on previous experiences as well as trial and error. In many cases, a simple structure with fewer neurons is selected first, and then the number of neurons and layers is optimized using cross-validation and model adjustment.

Model Improvement

If you want to improve the model, you can change the number of layers and neurons, use regularization techniques such as dropout, or explore even more complex architectures such as deep neural networks. To do this, you can use tools such as GridSearchCV or RandomizedSearchCV for optimal model settings.

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

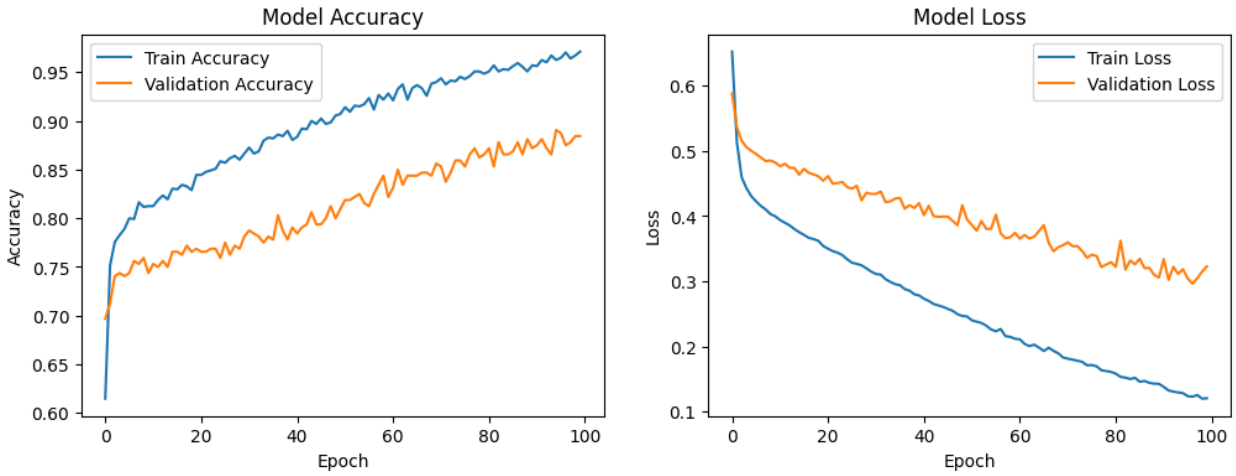


Figure 8: Epoch-Accuracy and Epoch-Loss Modulus for Mode 2

Analysis of Charts:

In the Accuracy Chart :(Model Accuracy)

Train Accuracy increases steadily as the number of epochs increases, reaching about 95%.

Validation accuracy also increases gradually, but is fixed at around 85%.

This difference between training accuracy and validation accuracy indicates some overfitting , which means that the model responds very well to training data but performs less on validation data.

Model Loss:

The Train Loss is steadily decreasing to about 0.1.

The validation loss is also reduced, but fixed around 0.3-0.4.

The difference between the training loss and the validation loss also indicates some overfitting.

By adding shuffle=True, it has a better model for training the data, which leads to increased stability and accuracy of the results. However, the model still has some overfitting that can be mitigated by using more regularization techniques, such as Dropout.

iii. The hyperparameters are as follows: (Dropout added.)

```
# Definition of callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint], shuffle=True)
```

Neural Network Model Design:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout


# Designing a Neural Network Model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

First, I'll give a brief description of Dropout, which is as follows:

In neural networks, the concept **of dropout** refers to the discharge of units (neurons). Simply put, dropout means that randomly selected neurons are ignored by units (neurons). In other words, in every forward or backward passage, these units are not considered. This method helps to solve the problem of overfitting and ensures that no units are dependent on each other.

As it turns out, this model is an improvement over the previous one and includes dropout layers to prevent overfitting. Below are the reasons to choose each part of the model:

1. First layer (Dense(64, input_dim=X_train.shape[1], activation='relu')):

- **Number of neurons (64): The** choice of 64 neurons in the first layer is based on experience and the prevalence of this number in many similar issues. This number is usually enough to begin with, and one can learn the input characteristics well.
- **Input (input_dim=X_train.shape[1]):** This parameter specifies the number of input properties of the data, which is determined based on the shape of the training data.
- **Activation (activation='relu'):** The use of the ReLU activation function because of its useful features such as preventing gradient vanishing and speeding up learning.

2. Dropout Layer (Dropout(0.5)):

- **Dropout:** Using Dropout at a rate of 0.5 is meant to deactivate 50% of neurons randomly in each training session. This helps prevent overfitting.

3. The second layer (Dense(32, activation='relu')):

- **Number of Neurons (32):** Reducing the number of neurons in the second layer relative to the first layer can help slightly reduce the complexity of the model and also help prevent overfitting.
- **Activation (activation='relu'):** ReLU continues to be used in this layer to maintain consistency between layers.

4. Dropout Layer (Dropout(0.5)):

- **Dropout:** Reusing Dropout at a rate of 0.5 is meant to reduce overfitting and increase the ability to generalize the model.

5. Output layer (Dense(1, activation='sigmoid')):

- **Number of Neurons (1):** Because your problem is a binary classification problem, one neuron in the output layer is enough.
- **Activation (activation='sigmoid'):** The use of the sigmoid function for output is because it limits the output value to a range between 0 and 1, which is suitable for binary categorization issues.

Why were so many neurons chosen?

The number of neurons in each layer is usually chosen based on previous experiences as well as trial and error. The choice of 64 and 32 neurons is because these numbers generally work well for many issues. Also, using a dropout rate of 0.5 helps the model avoid overfitting and perform better on new data.

Model Improvement

If you want to improve the model, you can change the number of layers and neurons, adjust the dropout rate, or even use more advanced techniques such as deeper neural networks or more complex architectures. You can also use tools like GridSearchCV or RandomizedSearchCV for optimal model adjustments.

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

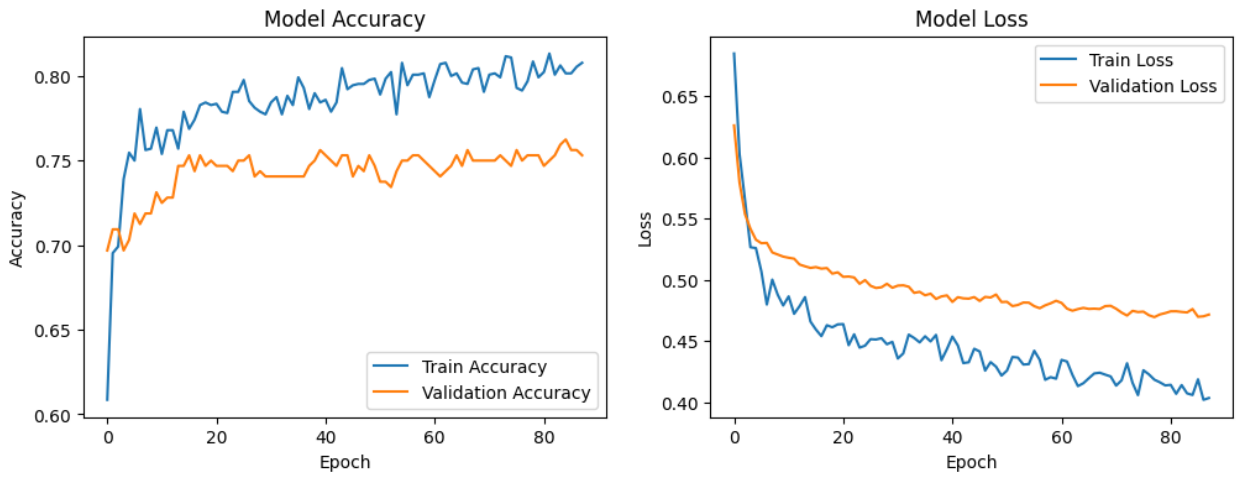


Figure 9: Epoch-Accuracy and Epoch-Loss Modulation for Mode 3

On the accuracy chart (left):

- **The accuracy of training** (blue line) has steadily increased, eventually reaching around 0.80. This shows that the model can learn training data very well.

- **Validation accuracy** (orange line) also increased but remained stable at around 0.75. This suggests that the model has been able to perform well on validation data, but the difference between training accuracy and validation accuracy suggests that the model has not yet been fully generalized to the new data.

On the loss chart (right):

- **The training** loss (blue line) has steadily decreased to around 0.40. This shows that the model is learning well from the training data and is being optimized.
- **The validation loss** (orange line) has also decreased but has remained steady at around 0.50. This suggests that the model performs well on validation data but can still improve.

Conclusion:

1. The model is learning well from the training data, but it has not yet been fully generalized to the new data.
2. There is a need for further investigation and perhaps better model adjustments, such as reduced learning rates or the use of data augmentation techniques, to improve model performance on new data.

iv. We're still looking to reduce overfitting, so we're going to make some changes as follows.

The hyperparameters are as follows:

```
# Definition of callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)  # add restore_best_weights=True
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint], shuffle=True)
```

Neural Network Model Design:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2

# Designing a Neural Network Model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu',
kernel_regularizer=l2(0.01)))         # add kernel_regularizer
model.add(BatchNormalization())  # add Batch Normalization
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01))) # add
kernel_regularizer
model.add(BatchNormalization())  # add Batch Normalization
model.add(Dropout(0.5))

model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.01)))  add
kernel_regularizer
model.add(BatchNormalization())  # add Batch Normalization
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

restore_best_weights=True:

This means that after an early stopping, the model's weights are returned to their best condition (corresponding to the lowest value val_loss). This option is very useful because:

Description

1. **Avoiding Overfitting**: During model training, the model may reach the overfitting state after a few epochs. Using restore_best_weights=True, the model weights are returned to the state where they performed best on the validation data.
2. **Maintain Best Performance: This** setting ensures that after an early stop, the model is stored with the weights that had the best accuracy or the least

error on the validation data. This way, weight that may have been worsened due to overfitting is avoided.

Example

For example, suppose that EarlyStop stops training the model after 50 epoch and the best weights are obtained at epoch 40. If restore_best_weights=True is set, the model weights will be returned to epoch 40. Otherwise, the model will be left with 50 epoch weights, which may be less accurate due to overfitting.

These settings help improve model performance in validation data and prevent overfitting.

Network Model Explanation:

The first layer:

- : **Dense** It is a fully connected layer with 64 neurons.
- **: input_dim=X_train.shape[1]** specifies the number of input features. This value is equal to the number of features in our training data .
- **: activation='relu'** is used for the ReLU activation function.
- **: kernel_regularizer=L2(0.01) The** L2 regulator is employed to reduce overfitting. This is the penalty amount applied to large weights.

:**BatchNormalization()**

- This layer is used to normalize the inputs to the next layer, making the learning speed increase and make the model more stable.

:**Dropout(0.5)**

- This layer randomly disables 50% of the neurons per update to prevent overfitting.

The second layer:

- Use Dense with 32 neurons, relu activation, and L2 regularizer with a coefficient of 0.01.

- The use of BatchNormalization is used to normalize the inputs to the next layer, increasing the speed of learning and making the model more stable.)
- Using Dropout at a rate of 0.5 (randomly disabling 50% of neurons and preventing overfitting)

The third layer:

- Use Dense with 16 neurons, relu activation, and L2 regularizer with a coefficient of 0.01.
- The use of BatchNormalization is used to normalize the inputs to the next layer, increasing the speed of learning and making the model more stable.)
- Using Dropout at a rate of 0.5 (randomly disabling 50% of neurons and preventing overfitting)

Output Layer:

- Using Dense with 1 neuron as output (for binary categorization), activating the sigmoid to generate an output probability between 0 and 1.

Conclusion

The designed model consists of several Dense layers, each equipped with an L2 regulator to prevent overfitting and a Batch Normalization layer to normalize and improve the speed of learning. Also, the Dropout layer is employed to prevent overfitting.

This model can learn well by combining these layers and different settings and avoid overfitting.

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

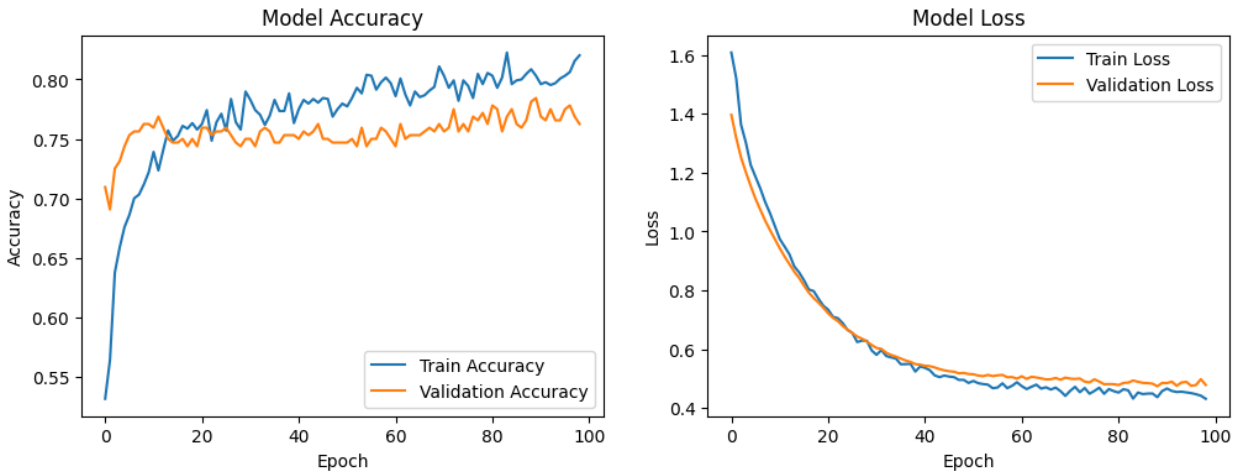


Figure 10: Epoch-Accuracy and Epoch-Loss Scale for Mode 4

Chart Analysis:

1. **Model Accuracy :**
    - At the beginning of the training, the model's accuracy on the training data (Train Accuracy) increases rapidly from 0.55 to about 0.75 over 20 cycles (epoch). This indicates that the model has quickly learned the basic patterns.
    - The accuracy of the model on the validation accuracy data increases in the same way, but after about 20 periods it reaches a relatively stable state of about 0.73.
    - After 50 periods, the accuracy of the model on the training data reaches about 0.80 and finally closes to 0.82. However, the accuracy of the model on the validation data fluctuates from 0.73 to 0.75.
    - This difference between training accuracy and validation accuracy represents some overfitting, as the model performs better on training data but fails to generalize as well to new data (validation).
2. **Model Loss :**

- At the beginning of the training, the model's error on the training data (Train Loss) quickly decreases from about 1.6 to about 0.6 over 20 periods.
- The model error on the validation loss data is similarly reduced, eventually reaching about 0.5.
- After 50 periods, the model error on the training data decreases to about 0.4 and eventually closes to 0.35, while the model error on the validation data fluctuates around 0.45 to 0.55.
- This discrepancy between the training error and the validation error is also indicative of overfitting, as the model has fewer errors on the training data but has not been able to perform as well on the new data (validation).

Conclusion

The diagrams show that the model learns well on the training data, and its error is reduced and its accuracy increased. Although the accuracy and error of the model on the validation data stabilize after a few periods, there is still some overfitting in the model. To reduce overfitting, the following measures can be useful:

1. **Increased Dropout Rate:** Increasing the dropout rate to 0.6 or 0.7 can randomly disable more neurons and reduce overfitting.
2. **Reduced number of neurons:** Reducing the number of neurons in each layer relative to other layers can reduce the complexity of the model and reduce overfitting.
3. **Data augmentation:** Increasing the volume of training data using techniques such as Data Augmentation can help the model learn better patterns and increase its accuracy on validation data.
4. **Use more regularization techniques:** Using L1 or L2 regularization with higher amounts can help reduce overfitting.
5. **Early Stopping:** Using Early Stopping with more precise parameters can help prevent overfitting.

By implementing these suggestions, it can help reduce overfitting and improve the accuracy of the model on validation data.

v. Following the fixing of the overfitting issue, we will make changes that are as follows.

The hyperparameters are as follows:

```
# Definition of callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)  # restore_best_weights=True ro EZAFE
KARDAM!!!!!!!!!!!!!!!!!!!!!!
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')
```

```
history = model.fit(X_train, y_train, epochs=25, batch_size=32,
validation_split=0.2, callbacks=[early_stopping, model_checkpoint], shuffle=True)
```

As you can see, we put epochs=25 instead of 100.

Neural Network Model Design:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2

# Designing a Neural Network Model
model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu',
kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.6))

model.add(Dense(8, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.6))

model.add(Dense(1, activation='sigmoid'))
```

Explanation of the neural network model:

1. **Input Layer**:
    - model.add(Dense(16, input_dim=X_train.shape[1], activation='relu', kernel_regularizer=l2(0.01)))
    - This layer contains 16 neurons and uses the ReLU activation function.
    - :input_dim=X_train.shape[1] The input size is equal to the number of input properties (equal to the number of X_train columns).
    - L2:kernel_regularizer=(0.01) This section shows that the L2 regularizer method was used to reduce the size of the weights.
2. **Batch Normalization Layer**:
    - model.add(BatchNormalization())
    - This layer is used to normalize the output values of the previous layers (such as the values of weights and biases), which helps to make neural network training faster and more stable.
3. **Dropout Layer**:
    - model.add(Dropout(0.6))
    - This layer disables the neurons in the previous layer with a 0.6 (or 60%) probability. This helps to avoid overfitting and improve the model's performance on new data.
4. **Hidden Layer**:
    - model.add(Dense(8, activation='relu', kernel_regularizer=l2(0.01)))
    - This layer contains 8 neurons and uses the ReLU activation function.
    - kernel_regularizer=L2(0.01): L2 regularizer is still used to reduce weight size.
5. **Output Layer**:
    - model.add(Dense(1, activation='sigmoid'))

- This layer consists of a neuron that uses the sigmoid activation function to calculate the probability of output, which is suitable for two-batch categorization problems.

This model uses the L2 regularizer to reduce overfit and tries to improve the performance and performance of the model and prevent overfitting by using Dropout and Batch Normalization.

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

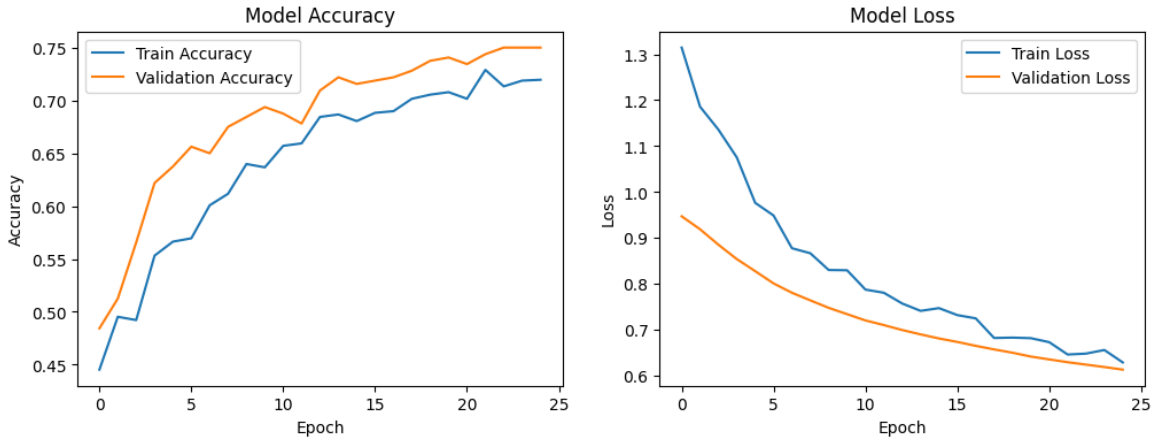


Figure 11: Epoch-Accuracy and Epoch-Loss Modulus for Mode 5

Analysis of Model Accuracy and Error Graphs

The graphs above show the performance of the neural network model during the training and validation process. The graph on the left shows the changes in accuracy and the graph on the right shows the changes in error (Loss) during the training periods (Epochs). Both graphs have two lines for the training data and the validation data.

Diagram on the Left: Model Accuracy

- **Train Accuracy**
  - This graph shows a steady increase in accuracy throughout the training periods.
  - At the beginning of the training, the accuracy of the training was about 0.45 and gradually increased and finally reached about 0.72.

- Increasing the accuracy of training indicates an improvement in the model's ability to learn patterns of training data.

- **Validation Accuracy**:(
  - The accuracy of validation also shows a similar increase.
  - Initially, the validation accuracy was around 0.48, and it gradually increased, eventually reaching around 0.75.
  - The convergence of validation accuracy with training accuracy indicates the absence of overfitting in the model.

Diagram on the Right: Model Error

- **Train Loss Training Error**:(
  - This graph shows a steady decrease in error during the training periods.
  - At the beginning of the training, the training error was about 1.35 and gradually decreased, finally reaching about 0.63.
  - Error reduction indicates improved model performance in learning training data patterns.

- **Validation Loss:**
  - The validation error also shows a similar decrease.
  - Initially, the validation error was around 1.05 and gradually decreased, eventually reaching around 0.61.
  - The convergence of the validation error with the training error indicates the absence of overfitting in the model.

Conclusion

- **The continuous reduction of training and validation error** indicates an improvement in the model's performance in learning the patterns of training and validation data.
- **The continuous increase in the accuracy of training and validation** indicates the model's ability to generalize and predict correctly.

- **The convergence of error and accuracy of training and validation** indicates the absence of overfitting, which means that the model has been able to generalize well and has good performance on unseen data (validation).

These results show that your neural network model is well-trained and can make accurate predictions with good accuracy.

vi. In the previous case, we saw that the overfitting problem was solved, but the accuracy of the model was also reduced. Here we find the best hyperparameter using RandomizedSearchCV.

```
best_params = random_search_result.best_params_

final_model = create_model(
    learning_rate=best_params['model__learning_rate'],
    dropout_rate=best_params['model__dropout_rate'],
    batch_norm=best_params['model__batch_norm']
)

early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_loss')

history = final_model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=best_params['epochs'],
    batch_size=best_params['batch_size'],
    callbacks=[early_stopping, model_checkpoint],
    verbose=1
)
```

Neural Network Model Design:

A neural network model was defined using Keras. This model includes Dense, Dropout, and BatchNormalization layers.

```
# Creating a Neural Network Model
```

```
def create_model(learning_rate=0.001, dropout_rate=0.5, batch_norm=False):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    if batch_norm:
        model.add(BatchNormalization())
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    if batch_norm:
        model.add(BatchNormalization())
    model.add(Dropout(dropout_rate))
    model.add(Dense(16, activation='relu'))
    if batch_norm:
        model.add(BatchNormalization())
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Search Hyperparameters: To find the best hyperparameters, RandomizedSearchCV was used.

```
model = KerasClassifier(
    model=create_model,
    verbose=0
)

# Define the search space for hyperparameters
param_dist = {
    'model__learning_rate': [0.001, 0.01, 0.1],
    'model__dropout_rate': [0.3, 0.5, 0.7],
    'model__batch_norm': [False, True],
    'batch_size': [32, 64, 128],
    'epochs': [50, 100, 200],
}

# Create RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_dist, n_iter=10, cv=3, verbose=1, n_jobs=-1)
```

```
#  RandomizedSearchCV
random_search_result = random_search.fit(X_train, y_train)

# Show the best hyperparameters
print("Best parameters found: ", random_search_result.best_params_)
```

Related Chart:

The Epoch-Accuracy and Epoch-Loss chart is as follows.

Figure 12: Epoch-Accuracy and Epoch-Loss Scores for Model 6

Analysis of Training Charts and Validation

The graphs above show the performance of the neural network model during the training and validation process. The graph on the left shows the changes in error (Loss) and the graph on the right shows the changes in accuracy during the training periods (Epochs). Both graphs have two lines for the training data and the validation data.

Diagram on the Left: Training and Validation Error

- **Training Loss :**

- This graph shows a steady decrease in error during the training periods.
- At the beginning of the training, the training error was about 0.60 and gradually decreased until it finally reached about 0.30.
- Error reduction indicates improved model performance in learning training data patterns.

- **Validation Loss:**
  - The validation error also shows a similar decrease.
  - Initially, the validation error was around 0.55 and gradually decreased, eventually reaching around 0.30.
  - The convergence of the validation error with the training error indicates the absence of overfitting in the model.

Diagram on the right: Training accuracy and validation

- **Training Accuracy:**
  - This graph shows a steady increase in accuracy throughout the training periods.
  - The accuracy of the model was about 0.65 at the beginning of the training, and it gradually increased and finally reached about 0.87.
  - Increasing the accuracy of training indicates the model's ability to learn patterns of training data.

- **Validation Accuracy:**
  - The accuracy of validation also shows a similar increase.
  - Initially, the validation accuracy was around 0.75 and gradually increased, eventually reaching around 0.88.
  - The convergence of validation accuracy with training accuracy indicates the absence of overfitting in the model.

Conclusion

- **The continuous reduction of training and validation error** indicates an improvement in the model's performance in learning the patterns of training and validation data.
- **The continuous increase in the accuracy of training and validation** indicates the model's ability to generalize and predict correctly.
- **The convergence of error and accuracy of training and validation** indicates the absence of overfitting, which means that the model has been able to generalize well and has good performance on unseen data (validation).

These results show that your neural network model is well-trained and can make accurate predictions with good accuracy.