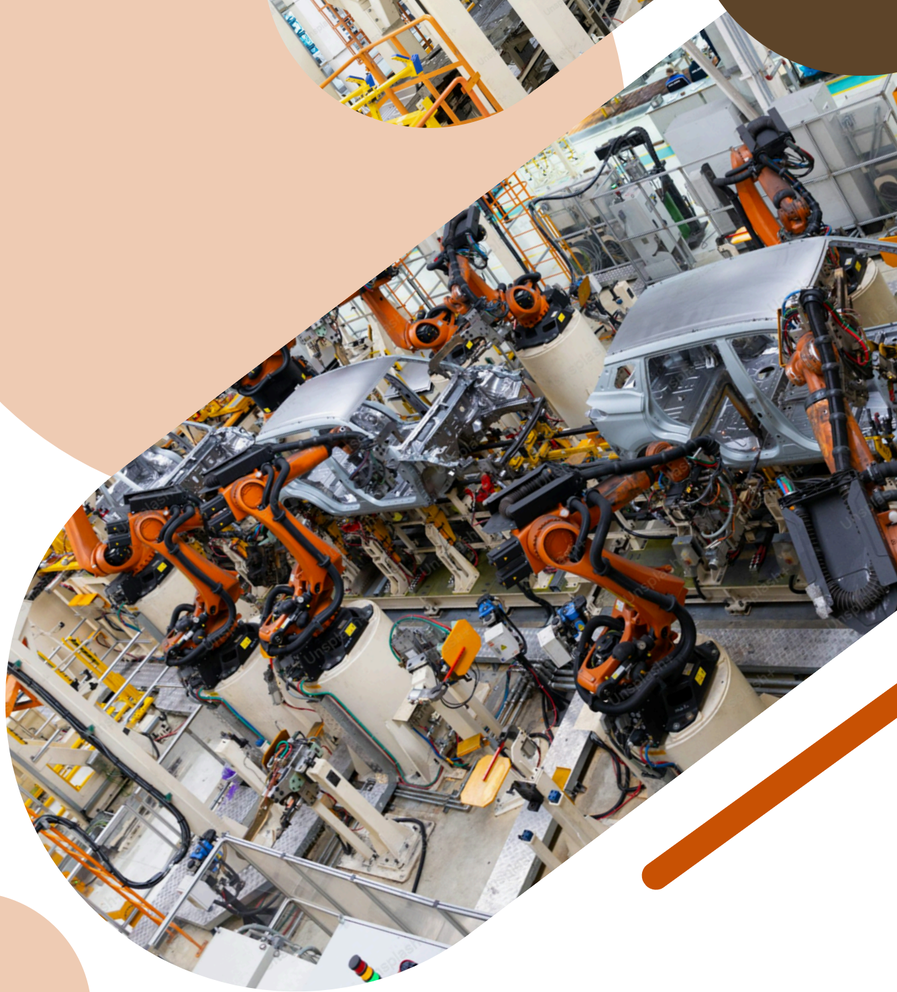
Department of Computer Science , Faculty of Sciences and Technologie , Tangier



ETUDE DE CAS
FST-INDUSTRIE

Prepared by: **Abdelmajid Benjelloun**

Supervized by: **Pr. Faouzi TAYALATI**

Department of Computer Science , Faculty of Sciences and Technologie , Tangier

PART 1 : Analyse & Modelisation

Introduction

This report investigates the use of Artificial Neural Networks (ANN) for classifying Total Effective Equipment Performance (TRS) in manufacturing machines. Utilizing a dataset of operational metrics such as functioning time and required time, the model aims to identify machine performance levels. Additionally, interpretability techniques like LIME and SHAP are applied to uncover key variables influencing predictions, thereby highlighting both top-performing machines and factors that may impact efficiency.

Calculations for Operational Metrics

CALCULATION OF OPERATING TIME

Operating Time is defined as the total available time during which the machine is open, minus the planned downtime and unplanned downtime

Operating Time=Opening Time−Planned Downtime−Downtime Duration.

CALCULATION OF REQUIRED TIME

Required Time reflects the amount of time necessary to achieve the production goals, accounting for any discrepancies in production speed.

Required Time=Operating Time−Cycle Time Deviation

CALCULATION OF USEFUL TIME

Useful Time quantifies the effective production time based on the number of good parts produced, adjusted by the production rate.

Useful Time=(Number of Good Parts)/Production Rate×60

CALCULATION OF TOTAL EFFECTIVE EQUIPMENT PERFORMANCE (TRS)

Finally, TRS is calculated to provide a comprehensive measure of equipment effectiveness, showing the proportion of Useful Time to Operating Time.

TRS=(Useful Time)/(Operating Time)×100

Overview of the Resulted Dataset

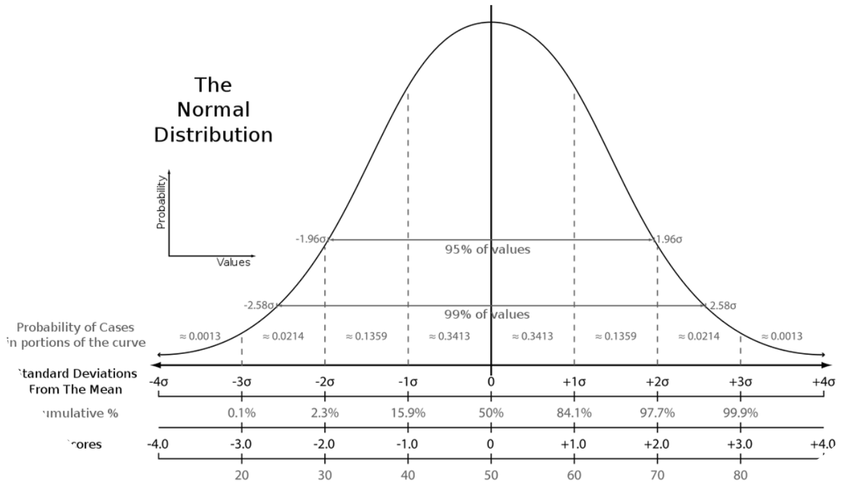
The resulting dataset contains key operational metrics, including Operating Time, Required Time, Net Time, Useful Time, and Total Effective Equipment Performance (TRS) for various machines. Each metric is essential for evaluating the efficiency and performance of equipment in a production environment.

	Machine	Temps_Fonctionnement	Temps_Requis	Temps_Utile	TRS
0	M1	12.743276	12.080601	0.210278	1.650108
1	M2	12.933940	12.616348	0.227124	1.756032
2	M3	11.791588	11.361981	0.227083	1.925808
3	M4	13.449393	13.039580	0.205346	1.526804
4	M1	13.828969	13.648769	0.211765	1.531312
...	...	...	...	...	...
995	M4	11.964955	11.623768	0.264506	2.210674
996	M1	13.156786	12.831901	0.231667	1.760815
997	M2	11.574963	11.339417	0.247458	2.137870
998	M3	12.815695	12.166345	0.199123	1.553742
999	M4	13.566240	13.007751	0.219492	1.617925

Standardizing the dataset is crucial because it ensures that all features contribute equally to the analysis. Standardization transforms the data to have a mean of zero and a standard deviation of one, eliminating bias due to varying scales and units.

More Information

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{mi}}$$



Why Min-Max ?

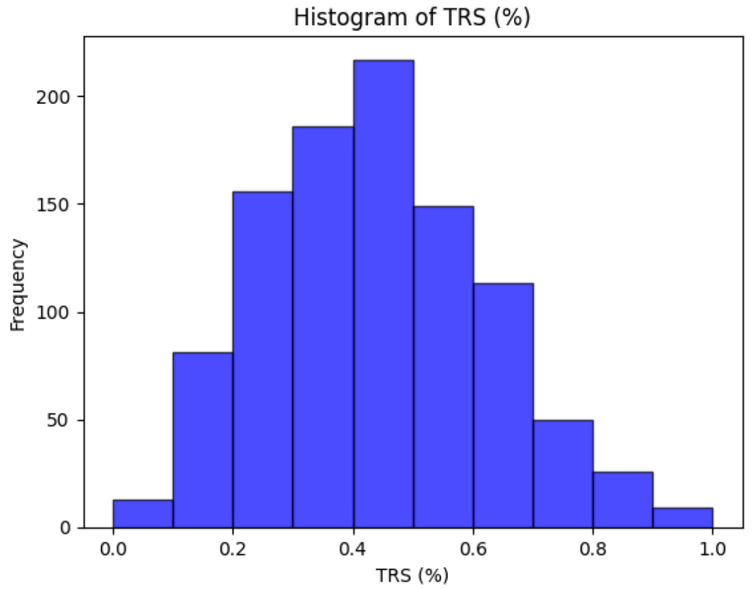
We chose Min-Max scaling for our dataset because it effectively transforms features to a uniform range, typically between 0 and 1. This is particularly beneficial for algorithms like Artificial Neural Networks, which can be sensitive to the scale of input data. By ensuring that all features are on the same scale, Min-Max scaling helps improve model convergence speed and performance, preventing any feature from dominating the learning process due to its scale.

More Details

Min-Max scaling works by taking each value in a feature and adjusting it using a simple formula. The original value is subtracted by the minimum value of that feature, and then this result is divided by the difference between the maximum and minimum values of the feature. This way, all values are rescaled to fit between 0 and 1, while keeping the relationships between the data points intact. This helps the model learn more effectively.

Overview of the standarized Dataset

	Machine	Temps_Fonctionnement	Temps_Requis	Temps_Utile	TRS
0	M1	0.384689	0.317476	0.392376	0.44008
1	M2	0.429390	0.434432	0.515914	0.51465
2	M3	0.161564	0.160598	0.515614	0.63419
3	M4	0.550239	0.526826	0.356210	0.35327
4	M1	0.639231	0.659814	0.403280	0.35644
...	...	...	...	...	...
995	M4	0.202210	0.217747	0.790043	0.83475
996	M1	0.481637	0.481488	0.549225	0.51802
997	M2	0.110776	0.155672	0.665023	0.78349
998	M3	0.401668	0.336194	0.310575	0.37223
999	M4	0.577634	0.519877	0.459942	0.41742



Machine with Highest TRS

Explanation

he machine with the highest TRS (Taux de Rendement Synthétique) is identified as [Machine Name], achieving a maximum TRS value of [TRS Value]. This performance highlights its efficiency and effectiveness in operation compared to other machines in the dataset.

Machine	M3
Temps_Fonctionnement	0.132644
Temps_Requis	0.093005
Temps_Utile	0.942772
TRS	1.0

Name: 882, dtype: object

Department of Computer Science , Faculty of Sciences and Technologie , Tangier

PART 2 : AI Application with ANN

Simulation of The Major Steps for Classification

Here how the ANN will simulate the work

- Data Preparation
- Train-Test Split
- Model Construction
- Model Training
- Model Evaluation

Simulation Steps

Train-Test Split

The dataset is again split into training and testing sets, maintaining the same proportions as before to ensure model validity.

Model Construction

An ANN architecture is designed with input neurons for our features, multiple hidden layers, and an output layer consisting of neurons equal to the number of classes (e.g., 7 for our TRS categories).
Activation functions such as ReLU for hidden layers and softmax for the output layer are utilized to classify the TRS into distinct categories.

Model Training

The model is trained using a categorical cross-entropy loss function, which is suitable for multi-class classification tasks. We monitor training progress to avoid overfitting and adjust hyperparameters as needed.

Simulation Steps

Model Evaluation

After training, we evaluate the model's performance on the test dataset using metrics like accuracy, precision, recall, and F1-score to understand how well the model classifies the TRS categories.

Predictions and Analysis

Predictions are made on the test set, with results compared to actual class labels. We analyze these results to identify trends and areas for improvement.

In Code !

The code splits the dataset into training (80%) and testing (20%) sets. It constructs an ANN with three hidden layers using ReLU activation and a linear output layer for regression. The model is compiled with the Adam optimizer and trained for 100 epochs. Finally, it evaluates performance on the test set and makes predictions.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

history= model.fit(X_train, Y_train, epochs=100, batch_size=10, validation_split=0.2)

loss = model.evaluate(X_test, Y_test)
predictions = model.predict(X_test)
```

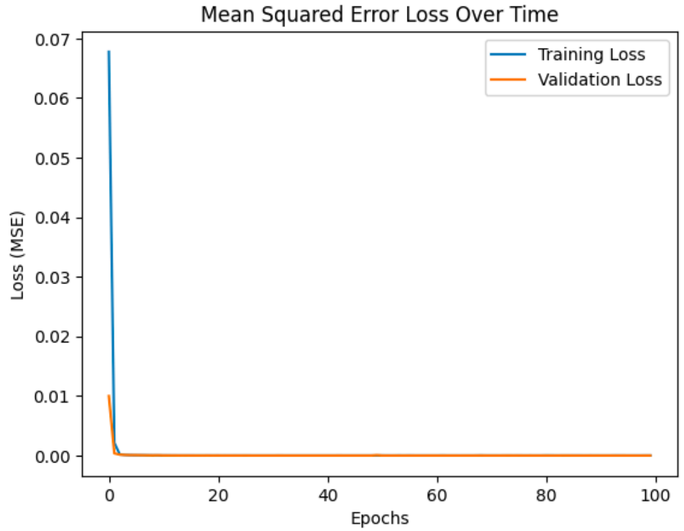
In Numbers !

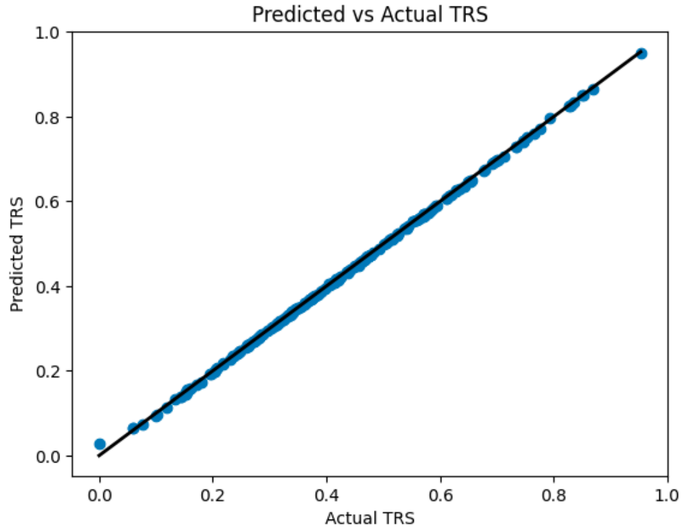
The training logs show the model's performance across 100 epochs, with loss values decreasing steadily, indicating improved accuracy. The training loss reaches a minimum of approximately 0.0000061 by the final epoch, while the validation loss fluctuates, ending at around 0.0000114. Overall, the low loss values suggest effective model training.

```
Epoch 94/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.4789e-05 - val_loss: 3.2001e-06
Epoch 95/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3.2287e-06 - val_loss: 1.1702e-05
Epoch 96/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3.7945e-06 - val_loss: 4.3485e-06
Epoch 97/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 5.3965e-06 - val_loss: 2.9763e-06
Epoch 98/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3.9831e-06 - val_loss: 2.7764e-06
Epoch 99/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.4367e-06 - val_loss: 4.4243e-06
Epoch 100/100
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 6.0746e-06 - val_loss: 1.1410e-05
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 2.1065e-05
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

In Plots !

The code generates two plots to evaluate model performance. The first plot displays the training and validation loss over epochs, illustrating the Mean Squared Error (MSE) trend throughout the training process. The second plot compares the predicted TRS values against the actual TRS values, with a diagonal line representing perfect predictions. This scatter plot helps visualize the model's accuracy, showing how closely the predictions align with the true values.





Predicting Best Machine

The code predicts TRS values for the entire standardized dataset using the trained model and adds these predictions to the DataFrame as a new column called TRS_Predit. It identifies the machine with the highest predicted TRS and retrieves both the actual and predicted TRS values for that machine. Finally, it displays the best-performing machine along with its actual and predicted TRS values and outputs a comparison table of the actual and predicted TRS for all machines.

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
La machine avec la meilleure performance est : M3
TRS réelle: 0.9999999999999999
TRS prédite: 0.9974409341812134
    Machine       TRS  TRS_Predit
0        M1  0.440082    0.438807
1        M2  0.514659    0.511719
2        M3  0.634190    0.632503
3        M4  0.353270    0.351992
4        M1  0.356444    0.353355
..      ...       ...         ...
995      M4  0.834751    0.833553
996      M1  0.518026    0.515500
997      M2  0.783493    0.784280
998      M3  0.372235    0.369074
999      M4  0.417424    0.415035
```

LIME

The LIME analysis indicates that the model's prediction is heavily influenced by the input features. The intercept is approximately 0.44, and the local prediction for the given input is about 0.41, closely aligning with the actual value of 0.44. Among the features, Temps_Utile is the most influential, contributing 0.39 to the prediction, followed by Temps_Fonctionnement at 0.38, and Temps_Requis with a contribution of 0.32. This suggests that Temps_Utile has the greatest impact on the model's prediction, underscoring its importance in determining performance outcomes.

What’s About Classification ?

Simulation of The Major Steps for Classification

Here how the ANN will simulate the work

- Data Preparation
- Feature and Label Extraction
- Model Creation
- Model Evaluation and Visualization

Simulation Steps

Data Preparation

In the initial step, the dataset is loaded into a DataFrame. The classify_trs function is applied to the TRS column to categorize the values into discrete classes ranging from 0 to 6. This classification is based on predefined thresholds, converting continuous TRS measurements into discrete categories for further analysis.

Feature and Label Extraction

After classifying the TRS values, the next step involves extracting the features and labels for model training. The features **X** are selected from the relevant columns: Temps_Fonctionnement, Temps_Requis, and Temps_Utile. The target labels **Y** are set as the newly created TRS_Class, which reflects the classified TRS values.

Simulation Steps

Model Creation

A Sequential neural network model is constructed using Keras for the classification task. The model consists of an input layer with 10 neurons activated by the ReLU function, followed by a second hidden layer with another 10 neurons. The output layer comprises 7 neurons, corresponding to the 7 classes of TRS classification, and utilizes the softmax activation function to generate class probabilities.

Model Evaluation and Visualization

After training, the model's accuracy is evaluated on the test set to determine its performance in classifying TRS values. Additionally, accuracy trends are visualized through a plot of training and validation accuracy over epochs. This visualization helps identify potential overfitting and the model's learning progression throughout the training process.

In Code !

The classify_trs function sorts Total Running Time (TRS) values into seven categories, ranging from 0 to 6, based on specific cut-off points. For example, TRS values below 0.15 are labeled as 0, while those at 0.90 or higher are labeled as 6. This function is applied to the TRS column in the DataFrame df, creating a new column called TRS_Class that contains these classifications. The feature set XXX includes Temps_Fonctionnement, Temps_Requis, and Temps_Utile, while the target variable YYY consists of the classified TRS values.

```python
def classify_trs(trs_value):
    if trs_value < 0.15:
        return 0
    elif trs_value < 0.30:
        return 1
    elif trs_value < 0.45:
        return 2
    elif trs_value < 0.60:
        return 3
    elif trs_value < 0.75:
        return 4
    elif trs_value < 0.90:
        return 5
    else:
        return 6

# Apply classification to the TRS column
df['TRS_Class'] = df['TRS'].apply(classify_trs)

# Prepare features and labels
X = df[['Temps_Fonctionnement', 'Temps_Requis', 'Temps_Utile']].values
Y = df['TRS_Class'].values
```

In Code !

This code sets up a simple artificial neural network using Keras. It features two hidden layers with ReLU activation to learn complex patterns and an output layer that predicts one of seven classes using softmax. The model is compiled with sparse categorical crossentropy as the loss function and uses the Adam optimizer for efficient training.

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

# Create the ANN model
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))  # First hidden layer
model.add(Dense(10, activation='relu'))  # Second hidden layer
model.add(Dense(7, activation='softmax'))  # Output layer (7 classes)

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

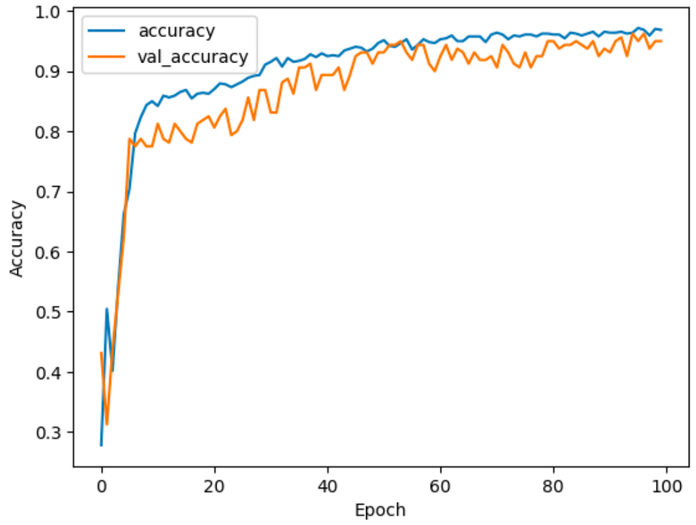
In Numbers !

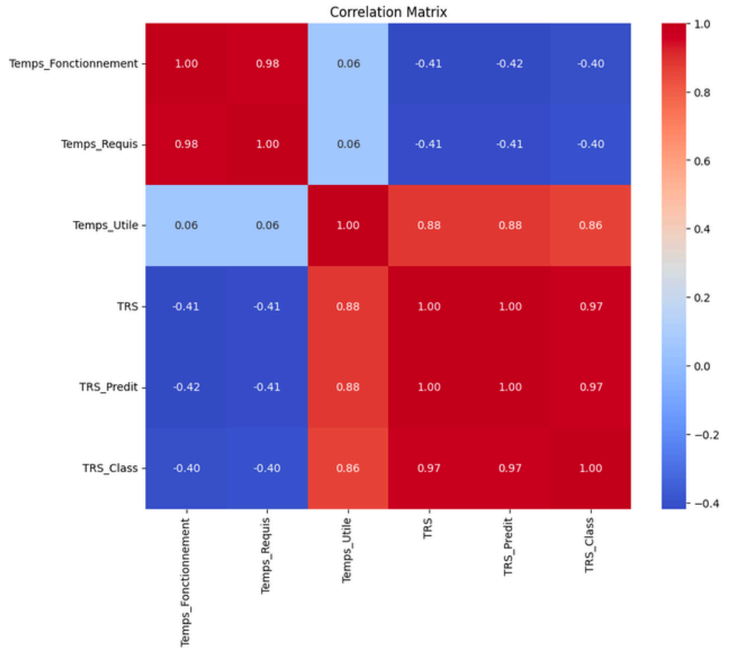
The training results show impressive performance, with the model achieving an accuracy of 98.36% and a loss of 0.1024 at the end of the 100 epochs. Validation accuracy also remains high, reaching 95.00%, with a validation loss of 0.1525. Throughout the epochs, the model consistently improved, demonstrating a strong ability to generalize on unseen data.

```
Epoch 94/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9660 - loss: 0.1387 - val_accuracy: 0.9250 - val_loss: 0.1676
Epoch 95/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9703 - loss: 0.1287 - val_accuracy: 0.9625 - val_loss: 0.1642
Epoch 96/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9761 - loss: 0.1255 - val_accuracy: 0.9500 - val_loss: 0.1680
Epoch 97/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9643 - loss: 0.1229 - val_accuracy: 0.9625 - val_loss: 0.1568
Epoch 98/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9532 - loss: 0.1399 - val_accuracy: 0.9375 - val_loss: 0.1631
Epoch 99/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9716 - loss: 0.1303 - val_accuracy: 0.9500 - val_loss: 0.1623
Epoch 100/100
128/128 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9836 - loss: 0.1024 - val_accuracy: 0.9500 - val_loss: 0.1525
```

In Plots !

The code creates a line plot to display the training and validation accuracy of the model across epochs, helping to visualize its performance over time. It also generates a heatmap for the correlation matrix, illustrating the relationships between different features. This heatmap includes annotations for clarity and is sized appropriately for better readability.





Predicting Best Machine

The code predicts class labels for a dataset and compares them with the actual labels. It first uses the model to get predictions based on the feature set X. Then, it creates a DataFrame called comparison_df that includes the machine names, actual class labels, and predicted class labels. The code groups the data by machine name to calculate accuracy and finds the best-performing machine with the highest accuracy. Finally, it prints the comparison results and shows a bar plot to visualize how well each machine performed.

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Comparison of Actual vs Predicted TRS Classes:
    Machine  Actual_TRS_Class  Predicted_TRS_Class
0        M1                 2                    2
1        M2                 3                    3
2        M3                 4                    4
3        M4                 2                    2
4        M1                 2                    2
..      ...               ...                  ...
995      M4                 5                    5
996      M1                 3                    3
997      M2                 5                    5
998      M3                 2                    2
999      M4                 2                    2

[1000 rows x 3 columns]

Best Performing Machine:
Machine        M2
Accuracy    0.984
```

Conclusion

In conclusion, the implementation of the model demonstrated impressive performance in classifying TRS classes, achieving high accuracy rates during training and validation. The use of LIME provided valuable insights into the decision-making process of the model, allowing us to understand how different features contributed to the predictions.

Utilizing Kaggle's GPU resources significantly improved the training time, enabling efficient handling of larger datasets and more complex models. This accessibility to computational power facilitates faster experimentation and iteration, making it an excellent choice for data science projects.

For a comprehensive overview of the functionality and methodologies employed, the full details are available in the provided code. Please feel free to reach out with any questions or for further clarifications.

Thank you,