**Description of design choices and Performance evaluation of the model**

## Introduction of RESNET50 :

ResNet50 is a convolutional neural network architecture that was introduced by Microsoft Research . It's part of a family of ResNet architectures, which are known for their deepness and effectiveness in training very deep neural networks.

The "50" in ResNet50 refers to the number of layers in the network. Specifically, it has 48 convolutional layers along with additional layers for downsampling and classification. The core idea behind ResNet (short for Residual Network) is the use of residual blocks. These blocks introduce shortcut connections that skip one or more layers, allowing for the creation of very deep networks while mitigating the vanishing gradient problem.

## Introduction to xgb :

XGBoost, short for eXtreme Gradient Boosting, is a popular and powerful machine learning algorithm that belongs to the class of ensemble learning techniques. It is particularly well-suited for supervised learning problems, including classification and regression tasks. XGBoost has gained widespread popularity and has been widely used in various machine learning competitions and real-world applications due to its effectiveness, scalability, and versatility.

## Aproach :

We will utilize the RESNET50 model for feature engineering and data preprocessing and then train our model with the help of XGBclassifier

# Import the reuire library  
import numpy as np  
import xgboost as xgb  
from keras.applications import ResNet50  
from keras.preprocessing import image  
from keras.applications.resnet50 import preprocess\_input  
from sklearn.model\_selection import train\_test\_split  
import os  
from tqdm import tqdm  
import pickle

## DATA PREPROCESSING AND FEATURE ENGINERRING

Resizing: Images may come in various sizes, and resizing them to a uniform size is often necessary for model compatibility and efficiency.

Normalization: Normalizing pixel values helps in improving model convergence and performance. Typically, this involves scaling pixel values to a range like [0, 1] or [-1, 1].

Color Space Conversion: Converting images to different color spaces (e.g., RGB, grayscale, HSV) can sometimes improve model performance or reduce computational complexity.

Handling Missing Data: Sometimes, images may have missing data or be corrupted. Dealing with such cases might involve image inpainting or removing the corrupted images from the dataset.

Feature Engineering for Image Data:

Feature Extraction: This involves extracting relevant features from images that are informative for the given task. Features can include edges, textures, shapes, or more abstract representations learned through convolutional neural networks (CNNs).

# Load pre-trained ResNet-50 model  
resnet\_model = ResNet50(weights='imagenet', include\_top=False, pooling='avg')

# Load and preprocess images   
# Here we will load the image with help of keras image preprocessing module and reshape it in 224x224 size  
# And process it with the help of Resnet inbuild preprocessing funtion  
def preprocess\_image(img\_path):  
 img = image.load\_img(img\_path, target\_size=(224, 224))  
 x = image.img\_to\_array(img)  
 x = np.expand\_dims(x, axis=0)  
 x = preprocess\_input(x)  
 return x

# Initiate the empty list to store image and label array  
X = []  
y = []

# Get the current directory  
current\_dir = os.getcwd()  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
parent\_dir = os.path.dirname(current\_dir)  
  
# Print the parent directory  
print("Parent Directory:", parent\_dir)

Parent Directory: C:\Users\sonu.a.jain\AppData\Local\miniconda3\Scripts\faultyFinding

# Dataset path  
directory = parent\_dir+"/datasets/raw\_dataset/Digital images of defective and good condition tyres"

# Itertate through Dataset and its subfolder to process image and extract the feature  
for label in os.listdir(directory):  
 label\_dir = os.path.join(directory, label)  
 for filename in tqdm(os.listdir(label\_dir)):  
 img\_path = os.path.join(label\_dir, filename)  
 img\_features = resnet\_model.predict(preprocess\_image(img\_path))  
 X.append(img\_features)  
 y.append(label)

0%| | 0/1028 [00:00<?, ?it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step

100%|██████████████████████████████████████████████████████████████████████████████| 1028/1028 [06:01<00:00, 2.85it/s]  
 0%|

| 0/828 [00:00<?, ?it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step

0%| | 1/828 [00:00<03:49, 3.60it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step

1%|▋ | 7/828 [00:02<04:02, 3.38it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step

1%|▊ | 8/828 [00:02<03:51, 3.54it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step

1%|▉ | 9/828 [00:02<03:47, 3.60it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step

1%|▉ | 10/828 [00:02<03:45, 3.63it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step

1%|█ | 11/828 [00:03<03:42, 3.67it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step

100%|███████████████████████████████████████████████████████████████████████████████▉| 827/828 [03:50<00:00, 3.26it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step

100%|████████████████████████████████████████████████████████████████████████████████| 828/828 [03:50<00:00, 3.59it/s]

# Convert to NumPy arrays  
X = np.array(X)  
y = np.array(y)  
  
# Verfiy the shape  
X.shape , y.shape

((1856, 1, 2048), (1856,))

# Creating copy of feature  
X1 = X.copy()

# Reshape the feature to make it compatible with Machine learning models   
X1 = np.reshape(X1 ,(X1.shape[0],X1.shape[1]\*X1.shape[2]))

X1.shape

(1856, 2048)

# Split dataset into training and testing sets in 80-20 ratio  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X1, y, test\_size=0.2, random\_state=42)

y\_test

array(['defective', 'defective', 'good', 'good', 'defective', 'good',  
 'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',  
 'defective', 'good', 'good', 'defective', 'defective', 'good',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'good', 'defective', 'defective', 'good', 'good', 'good', 'good',  
 'defective', 'good', 'good', 'defective', 'good', 'good', 'good',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'good', 'defective', 'good', 'defective', 'good', 'defective',  
 'good', 'good', 'good', 'defective', 'good', 'good', 'good',  
 'good', 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'defective', 'good', 'defective', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'good', 'good',  
 'defective', 'good', 'defective', 'good', 'good', 'defective',  
 'good', 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'good', 'good', 'good', 'defective', 'good', 'defective', 'good',  
 'defective', 'good', 'defective', 'defective', 'defective', 'good',  
 'defective', 'good', 'defective', 'good', 'good', 'good', 'good',  
 'good', 'defective', 'defective', 'defective', 'defective',  
 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'good', 'good', 'good', 'defective', 'defective',  
 'good', 'defective', 'good', 'defective', 'good', 'defective',  
 'defective', 'good', 'good', 'good', 'good', 'defective',  
 'defective', 'good', 'good', 'defective', 'good', 'good', 'good',  
 'good', 'good', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'good', 'good',  
 'good', 'defective', 'defective', 'defective', 'defective', 'good',  
 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'good', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'defective', 'good', 'defective', 'defective', 'good', 'good',  
 'good', 'good', 'defective', 'defective', 'defective', 'good',  
 'good', 'good', 'good', 'defective', 'defective', 'defective',  
 'good', 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'good', 'defective', 'good', 'defective',  
 'good', 'defective', 'good', 'defective', 'good', 'defective',  
 'good', 'defective', 'good', 'defective', 'good', 'defective',  
 'defective', 'good', 'good', 'defective', 'defective', 'defective',  
 'good', 'defective', 'good', 'defective', 'defective', 'good',  
 'defective', 'defective', 'defective', 'good', 'defective', 'good',  
 'defective', 'good', 'defective', 'defective', 'good', 'good',  
 'good', 'good', 'defective', 'good', 'defective', 'defective',  
 'good', 'defective', 'good', 'defective', 'defective', 'defective',  
 'defective', 'defective', 'good', 'good', 'defective', 'defective',  
 'defective', 'good', 'defective', 'defective', 'good', 'good',  
 'good', 'good', 'good', 'defective', 'good', 'defective',  
 'defective', 'good', 'defective', 'good', 'good', 'defective',  
 'good', 'defective', 'good', 'defective', 'good', 'good',  
 'defective', 'good', 'defective', 'good', 'defective', 'good',  
 'defective', 'good', 'defective', 'defective', 'good', 'defective',  
 'defective', 'good', 'defective', 'good', 'good', 'good', 'good',  
 'defective', 'defective', 'good', 'defective', 'good', 'defective',  
 'defective', 'defective', 'defective', 'good', 'defective', 'good',  
 'defective', 'defective', 'good', 'defective', 'defective',  
 'defective', 'defective', 'defective', 'defective', 'good', 'good',  
 'defective', 'good', 'defective', 'defective', 'defective', 'good',  
 'good', 'defective', 'good', 'good', 'defective', 'good', 'good',  
 'good', 'defective', 'defective', 'good', 'defective', 'defective',  
 'good', 'defective', 'good', 'good', 'good', 'defective', 'good',  
 'defective', 'defective', 'defective', 'defective', 'defective',  
 'defective'], dtype='<U9')

# Save the data which will be helpful to train multiple Machine learning models  
with open(parent\_dir+'/datasets/processed\_dataset/resnet\_X\_train.pkl', 'wb') as f:  
 pickle.dump(X\_train, f)  
  
with open(parent\_dir+'/datasets/processed\_dataset/resnetX\_test.pkl', 'wb') as f:  
 pickle.dump(X\_test, f)  
  
with open(parent\_dir+'/datasets/processed\_dataset/resnety\_train.pkl', 'wb') as f:  
 pickle.dump(y\_train, f)  
  
with open(parent\_dir+'/datasets/processed\_dataset/resnety\_test.pkl', 'wb') as f:  
 pickle.dump(y\_test, f)

## 1. Model Selection:

Choose Algorithm: Select an appropriate machine learning algorithm based on the nature of the problem (classification, regression, clustering, etc.), the size of the dataset, and other factors.

## 2. Model Building:

Instantiate Model: Create an instance of the chosen machine learning algorithm.

Fit Model: Train the model on the training data by calling the fit() method. During training, the model learns the patterns and relationships present in the data.

## XGBoost (Extreme Gradient Boosting)

is an advanced implementation of gradient boosting algorithm designed for efficiency, flexibility, and scalability. It is widely used in machine learning competitions and real-world applications due to its state-of-the-art performance and robustness. Here's a detailed description of XGBoost:

## 1. Gradient Boosting Algorithm:

Boosting Ensemble Method: XGBoost belongs to the family of boosting ensemble methods, where multiple weak learners (usually decision trees) are trained sequentially, and each subsequent model corrects the errors made by the previous models.

Gradient Boosting: XGBoost employs the gradient boosting framework, which optimizes a differentiable loss function by iteratively fitting weak learners to the negative gradient of the loss function.

## 2. Key Features of XGBoost:

Tree Ensemble Method: XGBoost builds an ensemble of decision trees, known as a gradient boosted decision tree (GBDT), to make predictions. Each tree is added sequentially to the ensemble, and subsequent trees learn from the residuals (errors) of the previous trees.

Regularization Techniques: XGBoost integrates various regularization techniques to prevent overfitting, including L1 (Lasso) and L2 (Ridge) regularization on leaf weights, and tree pruning to control tree depth and complexity.

Customizable Loss Functions: XGBoost supports customizable loss functions for both regression and classification tasks, allowing users to define their own objectives or use predefined objectives like logistic loss, squared loss, etc.

Parallel and Distributed Computing: XGBoost is highly optimized for parallel and distributed computing, leveraging multiple CPU cores and supporting distributed computing frameworks like Apache Hadoop and Apache Spark.

Optimized Tree Construction: XGBoost employs a number of optimization techniques to speed up tree construction, including approximate tree learning, column block for parallelization, and out-of-core computing for handling large datasets.

## 3. Advantages of XGBoost:

High Performance: XGBoost is known for its high prediction accuracy and efficiency, making it suitable for both small and large-scale datasets.

Flexibility: XGBoost can handle various types of data and tasks, including classification, regression, and ranking, and supports custom loss functions and evaluation metrics.

Feature Importance: XGBoost provides built-in feature importance scores, which help in feature selection and understanding the relative importance of input features in making predictions.

Robustness: XGBoost is robust to overfitting and can handle noisy data and missing values effectively, thanks to its regularization techniques and handling of missing values during tree construction.

## 4. Limitations of XGBoost:

Parameter Tuning: XGBoost requires careful parameter tuning, especially for hyperparameters like learning rate, tree depth, and regularization parameters, to achieve optimal performance.

Computationally Intensive: Training an XGBoost model can be computationally intensive, especially for large datasets or deep trees, requiring substantial computational resources.

Interpretability: While XGBoost provides feature importance scores, the resulting models may not be as interpretable as simpler models like decision trees or linear models.

Overall, XGBoost is a powerful and versatile algorithm that excels in a wide range of machine learning tasks. With its robustness, efficiency, and flexibility, XGBoost has become a popular choice for both practitioners and researchers in the field of machine learning and data science.

# Import the required library   
import xgboost as xgb  
import os  
import pickle

# Get the current directory  
current\_dir = os.getcwd()  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
parent\_dir = os.path.dirname(current\_dir)  
  
# Print the parent directory  
print("Parent Directory:", parent\_dir)

Parent Directory: C:\Users\sonu.a.jain\AppData\Local\miniconda3\Scripts\faultyFinding

preprocessed\_data\_dir = parent\_dir+'/datasets/processed\_dataset/'  
model\_dir = parent\_dir+'/models/'

# Load X\_test from file  
with open(os.path.join(preprocessed\_data\_dir,'resnet\_X\_train.pkl'), 'rb') as f:  
 X\_train = pickle.load(f)  
   
# Load y\_test from file  
with open(os.path.join(preprocessed\_data\_dir,'resnety\_train.pkl'), 'rb') as f:  
 y\_train = pickle.load(f)

# Verifying the shape  
X\_train.shape , y\_train.shape

((1484, 2048), (1484,))

# Creating a copy of labels  
y\_train\_1 = y\_train.copy()

# Replace all occurrences defective as 0 and good as 1  
for i in range(len(y\_train)):  
 if y\_train[i]=='defective' :   
 y\_train[i] = 0  
 else:  
 y\_train[i] = 1

# Conert it into string to int  
y\_train=y\_train.astype(int)

# Model selection  
xgb\_model = xgb.XGBClassifier()

# Train the model  
xgb\_model.fit(X\_train, y\_train)

XGBClassifier(base\_score=None, booster=None, callbacks=None,  
 colsample\_bylevel=None, colsample\_bynode=None,  
 colsample\_bytree=None, device=None, early\_stopping\_rounds=None,  
 enable\_categorical=False, eval\_metric=None, feature\_types=None,  
 gamma=None, grow\_policy=None, importance\_type=None,  
 interaction\_constraints=None, learning\_rate=None, max\_bin=None,  
 max\_cat\_threshold=None, max\_cat\_to\_onehot=None,  
 max\_delta\_step=None, max\_depth=None, max\_leaves=None,  
 min\_child\_weight=None, missing=nan, monotone\_constraints=None,  
 multi\_strategy=None, n\_estimators=None, n\_jobs=None,  
 num\_parallel\_tree=None, random\_state=None, ...)

model\_dir = parent\_dir+'/models/'  
model\_dir

'C:\\Users\\sonu.a.jain\\AppData\\Local\\miniconda3\\Scripts\\faultyFinding/models/'

# Save the trained model to a file  
with open(os.path.join(model\_dir,'RESNET50\_xgbClassifier\_model.pkl'),'wb') as f:  
 pickle.dump(xgb\_model, f)

## Model evaluation :

is a critical step in assessing the performance and reliability of a classifier model. It involves measuring how well the model performs on unseen data and understanding its strengths and weaknesses. Here's a comprehensive description of model evaluation for classifier models:

## 1. Performance Metrics:

Accuracy: The proportion of correctly classified instances out of the total instances. Precision: The proportion of true positive predictions among all positive predictions. Recall (Sensitivity): The proportion of true positive predictions among all actual positives. F1-score: The harmonic mean of precision and recall, which balances between precision and recall. Confusion Matrix: A matrix showing the counts of true positive, true negative, false positive, and false negative predictions. ROC Curve: Receiver Operating Characteristic curve showing the trade-off between true positive rate and false positive rate. AUC-ROC: Area Under the ROC Curve, which measures the model's ability to distinguish between classes. Precision-Recall Curve: A curve showing the trade-off between precision and recall at different classification thresholds.

## 2. Model Evaluation Process:

Model Prediction: Use the trained classifier model to make predictions on the testing set. Calculate Performance Metrics: Compute accuracy, precision, recall, F1-score, and other relevant metrics using the predicted labels and true labels from the testing set. Visualize Results: Plot the confusion matrix, ROC curve, and precision-recall curve to gain insights into the model's performance. Adjust Threshold (if necessary): If the classifier model outputs probabilities, adjust the classification threshold to optimize the desired metric (e.g., precision, recall). Cross-Validation: Perform k-fold cross-validation to assess the model's robustness and generalization ability.

## 3. Interpretation and Improvement:

Interpret Results: Analyze the performance metrics and visualizations to understand the classifier model's strengths and weaknesses. Model Improvement: Iterate on the model by fine-tuning hyperparameters, feature engineering, or trying different algorithms to improve performance. Feature Importance: Assess the importance of input features in making predictions using techniques like permutation importance or feature importance plots.

## 4. Business Impact:

Decision Making: Use the insights from model evaluation to make informed decisions about deploying the model in production or making changes to the data or model architecture. Monitoring and Maintenance: Continuously monitor the model's performance in the production environment and update it as needed to ensure its effectiveness over time.

#Import Required Libaray   
from sklearn.metrics import accuracy\_score ,classification\_report,confusion\_matrix,roc\_auc\_score,precision\_score,recall\_score  
from sklearn.metrics import f1\_score,roc\_curve,precision\_recall\_curve  
   
import os  
import pickle  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline

# Get the current directory  
current\_dir = os.getcwd()  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
parent\_dir = os.path.dirname(current\_dir)  
  
# Print the parent directory  
print("Parent Directory:", parent\_dir)

Parent Directory: C:\Users\sonu.a.jain\AppData\Local\miniconda3\Scripts\faultyFinding

preprocessed\_data\_dir = parent\_dir+'/datasets/processed\_dataset/'  
model\_dir = parent\_dir+'/models/'

# Load X\_test from file  
with open(os.path.join(preprocessed\_data\_dir,'resnetX\_test.pkl'), 'rb') as f:  
 X\_test = pickle.load(f)  
   
# Load y\_test from file  
with open(os.path.join(preprocessed\_data\_dir,'resnety\_test.pkl'), 'rb') as f:  
 y\_test = pickle.load(f)

# Load the saved model from file  
with open(os.path.join(model\_dir,'RESNET50\_xgbClassifier\_model.pkl'), 'rb') as f:  
 xgb\_clf = pickle.load(f)

# Prediction on test data with the help of trained model  
y\_pred = xgb\_clf.predict(X\_test)

y\_test\_n = y\_test.copy()

# Replace all occurrences defective as 0 and good as 1  
for i in range(len(y\_test)):  
 if y\_test\_n[i]=='defective' :   
 y\_test\_n[i] = 0  
 else:  
 y\_test\_n[i] = 1

y\_test\_n = y\_test\_n.astype(int)

# Calculate performance metrics  
  
accuracy = accuracy\_score(y\_test\_n, y\_pred)  
precision = precision\_score(y\_test\_n, y\_pred)  
recall = recall\_score(y\_test\_n, y\_pred)  
f1 = f1\_score(y\_test\_n, y\_pred)  
conf\_matrix = confusion\_matrix(y\_test\_n, y\_pred)  
roc\_auc = roc\_auc\_score(y\_test\_n, y\_pred)  
fpr, tpr, \_ = roc\_curve(y\_test\_n, y\_pred)  
precision, recall, \_ = precision\_recall\_curve(y\_test\_n, y\_pred)

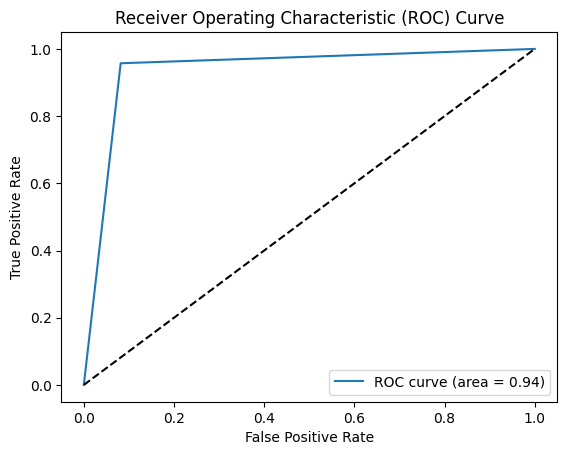
# Print performance metrics  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)  
print("Confusion Matrix:\n", conf\_matrix)  
print("ROC AUC Score:", roc\_auc)

Accuracy: 0.9354838709677419  
Precision: [0.44086022 0.90229885 1. ]  
Recall: [1. 0.95731707 0. ]  
F1-score: 0.9289940828402367  
Confusion Matrix:  
 [[191 17]  
 [ 7 157]]  
ROC AUC Score: 0.9377931519699811

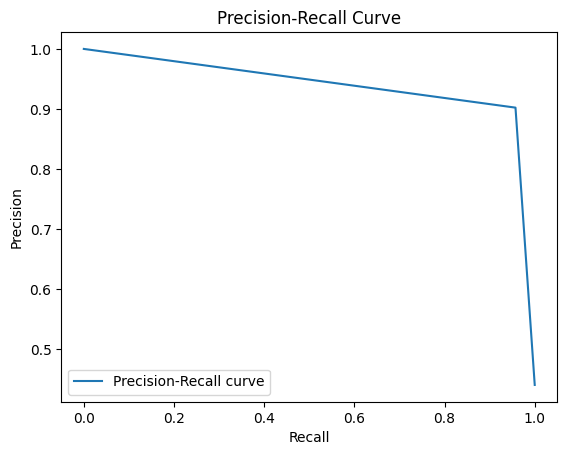
print(classification\_report(y\_test\_n, y\_pred))

precision recall f1-score support  
  
 0 0.96 0.92 0.94 208  
 1 0.90 0.96 0.93 164  
  
 accuracy 0.94 372  
 macro avg 0.93 0.94 0.93 372  
weighted avg 0.94 0.94 0.94 372

# Plot ROC Curve  
plt.figure()  
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc\_auc)  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver Operating Characteristic (ROC) Curve')  
plt.legend(loc="lower right")  
plt.savefig(parent\_dir+'\\visuals\\Resnet50\_XGB\_ROC\_Curve')  
plt.show()

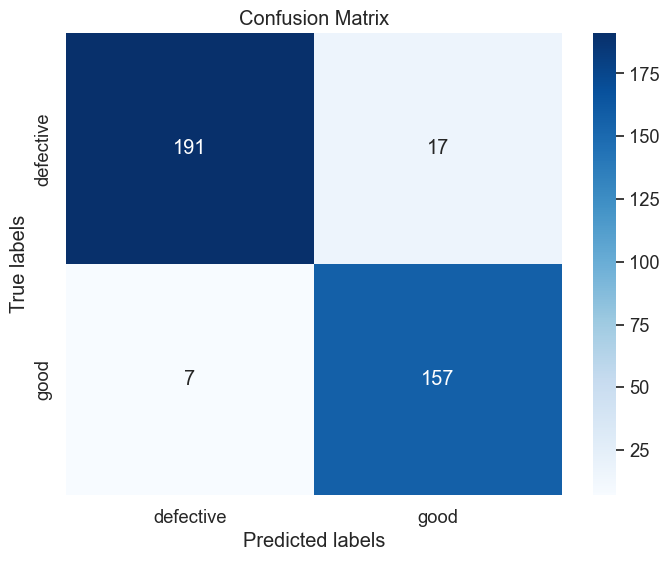


# Plot Precision-Recall Curve  
plt.figure()  
plt.plot(recall, precision, label='Precision-Recall curve')  
plt.xlabel('Recall')  
plt.ylabel('Precision')  
plt.title('Precision-Recall Curve')  
plt.legend(loc="lower left")  
plt.savefig(parent\_dir+'\\visuals\\Resnet50\_XGB\_Precision-Recall Curve')  
plt.show()



class\_names = ['defective','good']

# Plot confusion matrix  
plt.figure(figsize=(8, 6))  
sns.set(font\_scale=1.2) # Adjust font size  
sns.heatmap(conf\_matrix, annot=True, cmap="Blues", fmt='g', xticklabels=class\_names, yticklabels=class\_names)  
plt.xlabel('Predicted labels')  
plt.ylabel('True labels')  
plt.title('Confusion Matrix')  
plt.savefig(parent\_dir+'\\visuals\\RESNET50\_XGB\_Confusion Matrix')  
plt.show()



## Interpretation and Improvement:

We can choose this model as final model as it accuracy and other perfomance terms are good .

Accuracy: 0.9354838709677419 Precision: [0.44086022 0.90229885 1. ] Recall: [1. 0.95731707 0. ] F1-score: 0.9289940828402367 Confusion Matrix: [[191 17] [ 7 157]] ROC AUC Score: 0.9377931519699811

Business Impact : Identifying the Good and Defective tyres are very important aspect of currrent era as these days teh vehicles are increased and most of the people are used it for transport , so in term of safty of human being its good to identity the issues on time and get rid from any bad incidents.

Good tires are critical for safe and efficient operation of vehicles. Here are some key reasons why having good tires is important:

Safety: Tires are the only contact point between the vehicle and the road surface. Good tires provide optimal traction, which is essential for maintaining control of the vehicle, especially during braking, accelerating, and cornering. Tires with sufficient tread depth and proper inflation help prevent skidding, hydroplaning, and loss of control, reducing the risk of accidents.

Handling and Stability: Quality tires contribute to better handling and stability of the vehicle. They provide responsive steering and improved cornering performance, allowing drivers to maneuver safely and confidently, especially in challenging road conditions such as wet or slippery surfaces.

Braking Distance: Tires play a crucial role in braking performance. Good tires with adequate tread depth and optimal grip reduce the braking distance, enabling the vehicle to stop more quickly and effectively in emergency situations, thereby enhancing overall safety on the road.

Fuel Efficiency: Properly maintained tires can improve fuel efficiency. Low rolling resistance tires reduce the energy required to propel the vehicle forward, resulting in lower fuel consumption and reduced carbon emissions. By ensuring tires are properly inflated and aligned, drivers can maximize fuel efficiency and save money on fuel costs.

Comfort and Ride Quality: Quality tires contribute to a smoother and more comfortable ride. They help dampen road vibrations and reduce noise, providing a quieter and more pleasant driving experience for occupants. Additionally, tires with good shock absorption properties enhance ride quality by minimizing bumps and jolts on uneven road surfaces.

Longevity and Durability: Investing in high-quality tires can result in longer tread life and extended tire longevity. Quality tires are designed to withstand wear and tear, punctures, and road hazards, resulting in fewer tire replacements and maintenance costs over time.

All-Weather Performance: Good tires are designed to perform well in various weather conditions, including dry, wet, and snowy conditions. All-season or winter tires with specialized tread patterns and rubber compounds provide enhanced traction and grip, ensuring safe driving regardless of weather conditions.

Vehicle Performance and Handling: Tires influence the overall performance and dynamics of the vehicle. They affect acceleration, braking, cornering, and stability, contributing to the overall driving experience. Choosing tires that match the vehicle's specifications and intended use (e.g., passenger car, SUV, truck, performance vehicle) ensures optimal performance and handling characteristics.

# Implement a feedback loop to update the model periodically with fresh manufacturing data.

# Import the reuire library  
import numpy as np  
import xgboost as xgb  
from keras.applications import ResNet50  
from keras.preprocessing import image  
from keras.applications.resnet50 import preprocess\_input  
from sklearn.model\_selection import train\_test\_split  
import os  
from tqdm import tqdm  
import pickle  
from sklearn.metrics import accuracy\_score ,classification\_report,confusion\_matrix,roc\_auc\_score,precision\_score,recall\_score  
from sklearn.metrics import f1\_score,roc\_curve,precision\_recall\_curve  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline

## DATA PREPROCESSING AND FEATURE ENGINERRING

# Load pre-trained ResNet-50 model  
resnet\_model = ResNet50(weights='imagenet', include\_top=False, pooling='avg')

# Load and preprocess images   
# Here we will load the image with help of keras image preprocessing module and reshape it in 224x224 size  
# And process it with the help of Resnet inbuild preprocessing funtion  
def preprocess\_image(img\_path):  
 img = image.load\_img(img\_path, target\_size=(224, 224))  
 x = image.img\_to\_array(img)  
 x = np.expand\_dims(x, axis=0)  
 x = preprocess\_input(x)  
 return x

# Initiate the empty list to store image and label array  
X = []  
y = []

# Get the current directory  
current\_dir = os.getcwd()  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
parent\_dir = os.path.dirname(current\_dir)  
  
# Print the parent directory  
print("Parent Directory:", parent\_dir)

Parent Directory: C:\Users\sonu.a.jain\AppData\Local\miniconda3\Scripts\faultyFinding

# Dataset path  
directory = parent\_dir+"/datasets/feedbackLoopData/Digital images of defective and good condition tyres"

# Itertate through Dataset and its subfolder to process image and extract the feature  
for label in os.listdir(directory):  
 label\_dir = os.path.join(directory, label)  
 for filename in tqdm(os.listdir(label\_dir)):  
 img\_path = os.path.join(label\_dir, filename)  
 img\_features = resnet\_model.predict(preprocess\_image(img\_path))  
 X.append(img\_features)  
 y.append(label)

0%| | 0/4 [00:00<?, ?it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step

25%|█████████████████████ | 1/4 [00:02<00:08, 2.92s/it]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step

50%|██████████████████████████████████████████ | 2/4 [00:03<00:02, 1.33s/it]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step

75%|███████████████████████████████████████████████████████████████ | 3/4 [00:03<00:00, 1.18it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step

100%|████████████████████████████████████████████████████████████████████████████████████| 4/4 [00:03<00:00, 1.11it/s]  
 0%| | 0/4 [00:00<?, ?it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step

25%|█████████████████████ | 1/4 [00:00<00:00, 4.10it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step

50%|██████████████████████████████████████████ | 2/4 [00:00<00:00, 4.39it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step

75%|███████████████████████████████████████████████████████████████ | 3/4 [00:00<00:00, 4.31it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step

100%|████████████████████████████████████████████████████████████████████████████████████| 4/4 [00:01<00:00, 3.98it/s]

# Convert to NumPy arrays  
X = np.array(X)  
y = np.array(y)

# Creating copy of feature  
X1 = X.copy()

# Reshape the feature to make it compatible with Machine learning models   
X1 = np.reshape(X1 ,(X1.shape[0],X1.shape[1]\*X1.shape[2]))

# Creating a copy of labels  
y1 = y.copy()

# Replace all occurrences defective as 0 and good as 1  
for i in range(len(y1)):  
 if y1[i]=='defective' :   
 y1[i] = 0  
 else:  
 y1[i] = 1

# Conert it into string to int  
y1=y1.astype(int)

# Split dataset into training and testing sets in 80-20 ratio  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X1, y1, test\_size=0.2, random\_state=42)

model\_dir = parent\_dir+'/models/'  
model\_dir

'C:\\Users\\sonu.a.jain\\AppData\\Local\\miniconda3\\Scripts\\faultyFinding/models/'

# Load the saved model from file  
with open(os.path.join(model\_dir,'RESNET50\_xgbClassifier\_model.pkl'), 'rb') as f:  
 xgb\_model = pickle.load(f)

# Trained the PreTrained Model with new dataset

model = xgb.XGBClassifier()  
# Train the model  
model = model.fit(X\_train, y\_train,xgb\_model = xgb\_model)

# Save the trained model to a file  
with open(os.path.join(model\_dir,'RESNET50\_xgbClassifier\_model.pkl'),'wb') as f:  
 pickle.dump(model, f)

# I don't have much fresh dataset , so i just uploaded 4-4 picture in defective and good , so accuracy can't be calculaed in much efficient way due to lack of train and test data.So i tested with existing test dataset and its working upto the mark and commented the code.

# In Real world , We will be having suffiecient freash data to train and test.

# preprocessed\_data\_dir = parent\_dir+'/datasets/processed\_dataset/'

# # Load X\_test from file  
# with open(os.path.join(preprocessed\_data\_dir,'resnetX\_test.pkl'), 'rb') as f:  
# X\_test = pickle.load(f)  
   
# # Load y\_test from file  
# with open(os.path.join(preprocessed\_data\_dir,'resnety\_test.pkl'), 'rb') as f:  
# y\_test = pickle.load(f)

# # Replace all occurrences defective as 0 and good as 1  
# for i in range(len(y\_test)):  
# if y\_test[i]=='defective' :   
# y\_test[i] = 0  
# else:  
# y\_test[i] = 1  
# y\_test = y\_test.astype(int)

# Prediction on test data with the help of trained model  
y\_pred = model.predict(X\_test)

# Calculate performance metrics  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
roc\_auc = roc\_auc\_score(y\_test, y\_pred)  
fpr, tpr, \_ = roc\_curve(y\_test, y\_pred)  
precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred)

# Print performance metrics  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)  
print("Confusion Matrix:\n", conf\_matrix)  
print("ROC AUC Score:", roc\_auc)

Accuracy: 0.9354838709677419  
Precision: [0.44086022 0.90229885 1. ]  
Recall: [1. 0.95731707 0. ]  
F1-score: 0.9289940828402367  
Confusion Matrix:  
 [[191 17]  
 [ 7 157]]  
ROC AUC Score: 0.9377931519699811

## Model Test with some Random samples

#Import the required library   
import numpy as np  
from keras.applications import ResNet50  
from keras.preprocessing import image  
from keras.applications.resnet50 import preprocess\_input  
import os  
import pickle  
import matplotlib.pyplot as plt  
%matplotlib inline

# Get the current directory  
current\_dir = os.getcwd()  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
current\_dir = os.path.dirname(current\_dir)  
  
# Get the parent directory (one level up)  
parent\_dir = os.path.dirname(current\_dir)  
  
# Print the parent directory  
print("Parent Directory:", parent\_dir)

Parent Directory: C:\Users\sonu.a.jain\AppData\Local\miniconda3\Scripts\faultyFinding

#Build the path for required directories  
random\_samples\_to\_test\_model\_dir = parent\_dir+'/datasets/random\_samples\_to\_test\_model/'  
model\_dir = parent\_dir+'/models/'

# Load the saved model from file  
with open(os.path.join(model\_dir,'RESNET50\_xgbClassifier\_model.pkl'), 'rb') as f:  
 xgb\_clf = pickle.load(f)

# create model for feature engineering on test data to make it feasible for trained model  
resnet\_model = ResNet50(weights='imagenet', include\_top=False, pooling='avg')

#create empty list to store the predictions  
result =[]

# Function to preprocess and predict the data  
def preprocess\_and\_predict(img\_path):  
 img = image.load\_img(img\_path, target\_size=(224, 224))  
 x = image.img\_to\_array(img)  
 x = np.expand\_dims(x, axis=0)  
 x = preprocess\_input(x)  
 img\_features = resnet\_model.predict(x)  
 img\_features = np.array(img\_features)  
 y\_pred = xgb\_clf.predict(img\_features)  
 result.append(y\_pred)

for filename in (os.listdir(random\_samples\_to\_test\_model\_dir)):  
 img\_path = os.path.join(random\_samples\_to\_test\_model\_dir, filename)  
 preprocess\_and\_predict(img\_path)

1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step  
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step  
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step  
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step

class\_name = ['defective','good']

numberOfFiles = len(os.listdir(random\_samples\_to\_test\_model\_dir))

#Plot the test data and its predictions :  
for i, filename in enumerate(os.listdir(random\_samples\_to\_test\_model\_dir)):  
 img\_path = os.path.join(random\_samples\_to\_test\_model\_dir, filename)  
 img = image.load\_img(img\_path, target\_size=(224, 224))  
 ax = plt.subplot(numberOfFiles//2,numberOfFiles-(numberOfFiles//2),i+1)  
 plt.imshow(img)  
 plt.title(class\_name[result[i].item()])   
 plt.xticks([])  
 plt.yticks([])

