ENHANCING PRODUCT RELIABILITY: LEVERAGING TRANSFER LEARNING FOR FAULT DETECTION

Project Hand-out, Faculty Development Program – NaanMudhalvan

SmartInternz
www.smartinternz.com

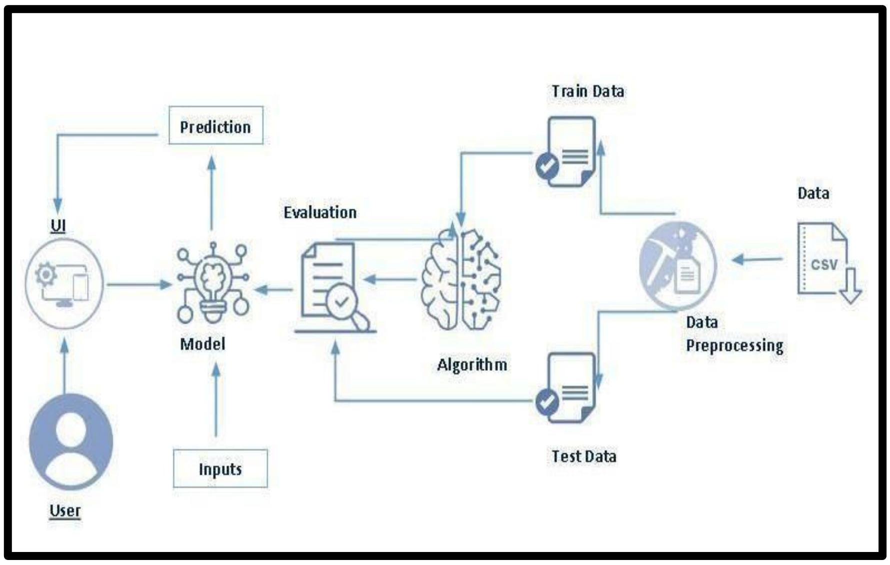
Fault Detection Using Transfer Learning

In modern manufacturing, ensuring product quality and reliability is paramount. Industries such as automotive, aerospace, and electronics rely on stringent quality control processes to prevent defective products from reaching the consumer. A single undetected fault, such as a micro-crack in a casted component, can lead to catastrophic failures, significant financial loss, and damage to brand reputation. Traditional inspection methods, often relying on manual human oversight, are typically slow, costly, and prone to error and inconsistency.

This project addresses this critical challenge by developing an advanced, automated fault detection system using the power of Artificial Intelligence, specifically a Deep Learning technique known as **Transfer Learning**. The system is designed to analyze images of manufactured parts—in this case, submersible pump impellers from a metal casting process—and accurately classify them as either "good" or "defective." By automating this visual inspection, we aim to create a solution that is significantly faster, more reliable, and more scalable than traditional methods, thereby enhancing overall product reliability.

Technical Architecture:

The architecture of this system follows a standard machine learning pipeline, from data ingestion to user prediction. Raw image data is collected and fed into a preprocessing pipeline, which prepares it for the model. The core of the system is a Deep Learning algorithm (VGG16 with a custom classifier) which is trained on labeled data. The performance of the trained model is evaluated, and the best version is saved. This saved model is then integrated into a web application, allowing a user to upload an image through a user interface (UI) and receive an instant prediction about the product's quality



Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with the flask application.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data Collection: Collect or download the dataset that you want to train.
- Data Visualization
- Data pre-processing
    - Splitting data into train and test

- Model building
    - Import the model-building libraries
    - Initializing the model
    - Training and testing the model
    - Evaluating the performance of the model
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code

To successfully build this system, we follow a structured project flow composed of several key milestones:

1. **Problem Definition and Understanding:** Specify the business problem of manufacturing quality control and define the requirements for an automated solution.
2. **Data Collection & Preparation:** Collect a suitable dataset of labeled images from Kaggle and prepare it using techniques like resizing, rescaling, and data augmentation.
3. **Exploratory Data Analysis (EDA):** Perform visual analysis on the dataset to understand the characteristics of the two classes.
4. **Model Building (Transfer Learning):** Select and modify a pre-trained VGG16 model, compile it, and train it on the prepared dataset.
5. **Model Performance Evaluation:** Analyze the training and validation accuracy metrics to evaluate the model's performance.
6. **Model Deployment:** Save the best-performing model and build a Flask web application to serve live predictions.
7. **Project Demonstration:** Demonstrate the end-to-end solution through the web interface.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

.

- **Deep Learning Concepts:** Understanding of Neural Networks and layers.
- **Convolutional Neural Networks (CNNs):** Knowledge of how CNNs process images.
- **Transfer Learning:** Understanding the concept of using pre-trained models like VGG16.
- **Python Programming:** Familiarity with Python syntax.
- **Flask Basics:** Basic knowledge of the Flask web framework.

DL Concepts :

- Neural Networks:: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/
- Deep Learning Frameworks:: https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow
- Transfer Learning: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a
- VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/
- Convolutional Neural Networks (CNNs): https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- Overfitting and Regularization: https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/
- Optimizers: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

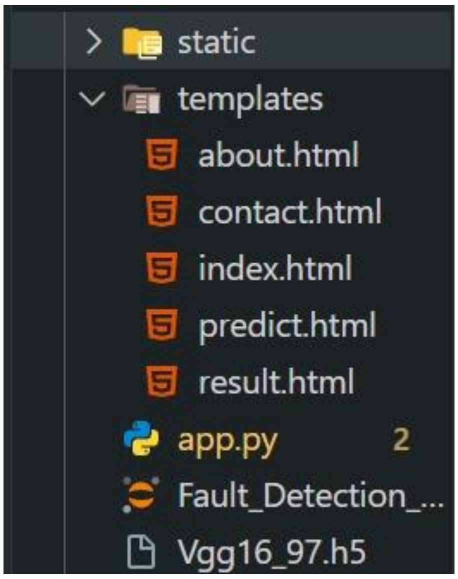
Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create a Project folder that contains files as shown below

- The Data folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the templates.
- folder and a python script app.py for server-side scripting
- we need the model that is saved and the saved model in this content is a Vgg16_97.h5
- templates folder contains index.html, inner-page.html & portfolio-details.html pages.

The final project is organized into a standard Flask web application structure



```
> static
v templates
    about.html
    contact.html
    index.html
    predict.html
    result.html
  app.py        2
  Fault_Detection_...
  Vgg16_97.h5
```

- Fault_Detection_App/
  - static/ (Contains CSS, JavaScript, and image assets)
  - templates/ (Contains all HTML files for the web pages)
  - uploads/ (Temporary folder for user-uploaded images)
  - app.py (The main Flask application script)
  - Vgg16_97.h5 (The saved, trained AI model)
  - Fault_Detection_Model_Training.ipynb (The notebook used for model training)

Milestone 1: Problem Understanding

The core business problem is the inefficiency and potential for error in manual quality control for manufactured goods. This leads to higher operational costs and increased risk of product failure. An automated system is needed to perform visual inspection at scale with high accuracy and speed. The system must be fast, reliable, and provide a user-friendly interface for operators. The impact is significant, leading to reduced costs, enhanced brand reputation, and increased consumer safety in critical industries.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

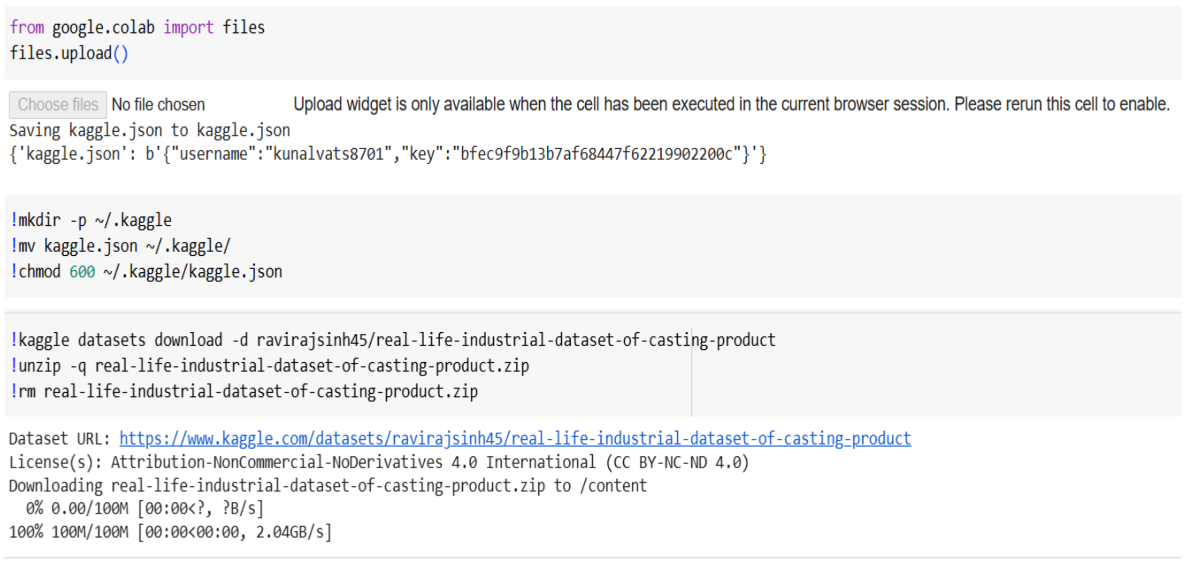
Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

The dataset used is the "Casting product image data for quality inspection" from Kaggle, containing images of submersible pump impellers.

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.



```
from google.colab import files
files.upload()
```

```
Choose files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"kunalvats8701","key":"bfec9f9b13b7af68447f62219902200c"}'}
```

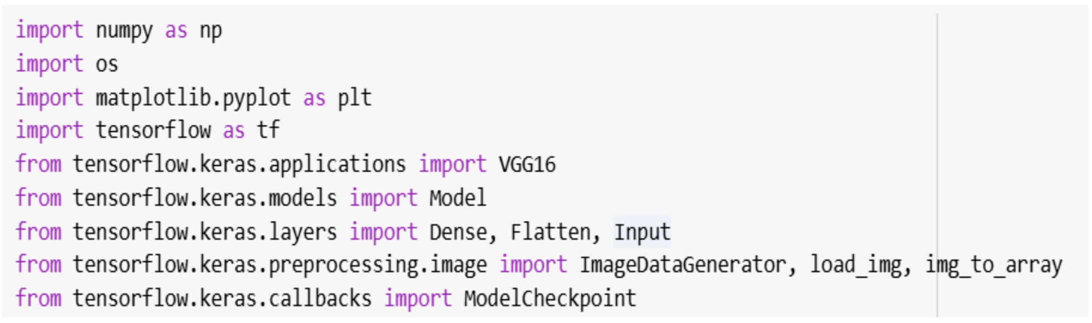
```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d ravirajsinh45/real-life-industrial-dataset-of-casting-product
!unzip -q real-life-industrial-dataset-of-casting-product.zip
!rm real-life-industrial-dataset-of-casting-product.zip
```

```
Dataset URL: https://www.kaggle.com/datasets/ravirajsinh45/real-life-industrial-dataset-of-casting-product
License(s): Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)
Downloading real-life-industrial-dataset-of-casting-product.zip to /content
  0% 0.00/100M [00:00<?, ?B/s]
100% 100M/100M [00:00<00:00, 2.04GB/s]
```

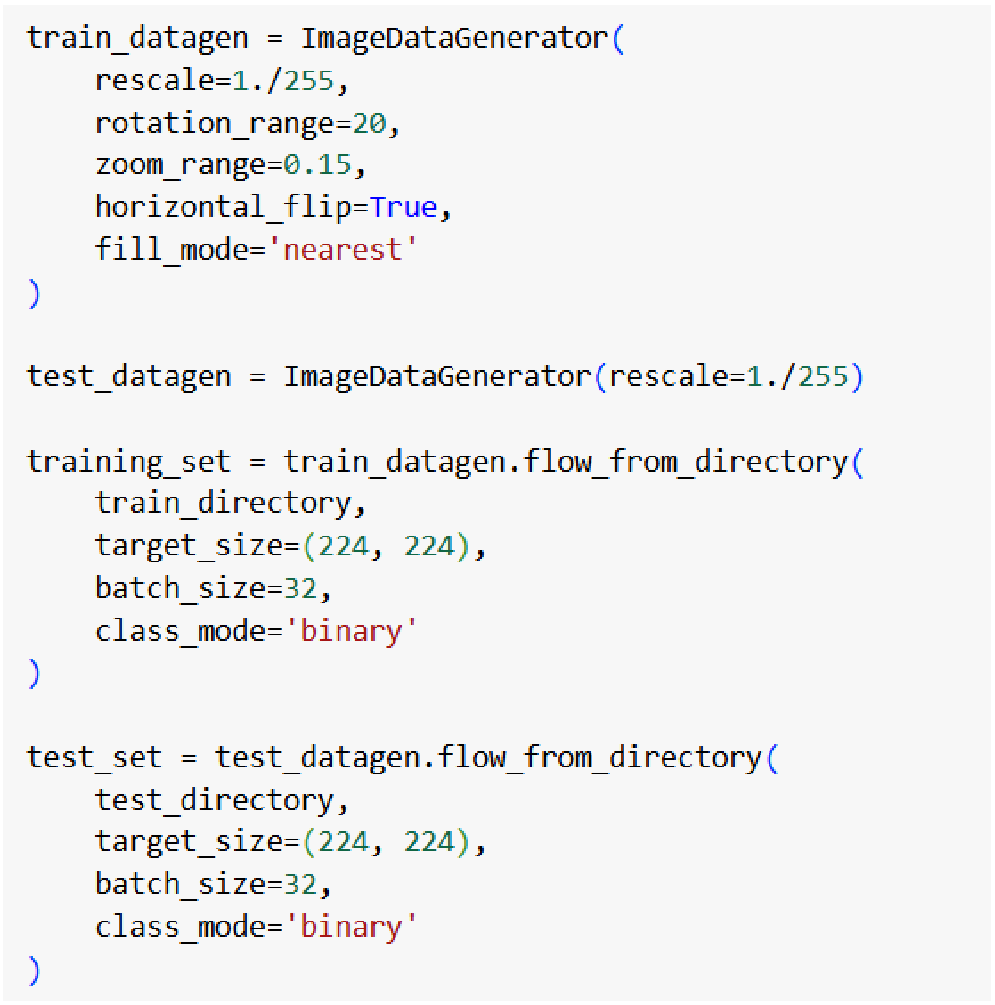
Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.



```
import numpy as np
import os
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.callbacks import ModelCheckpoint
```

Activity 2: Data Preparation using ImageDataGenerator

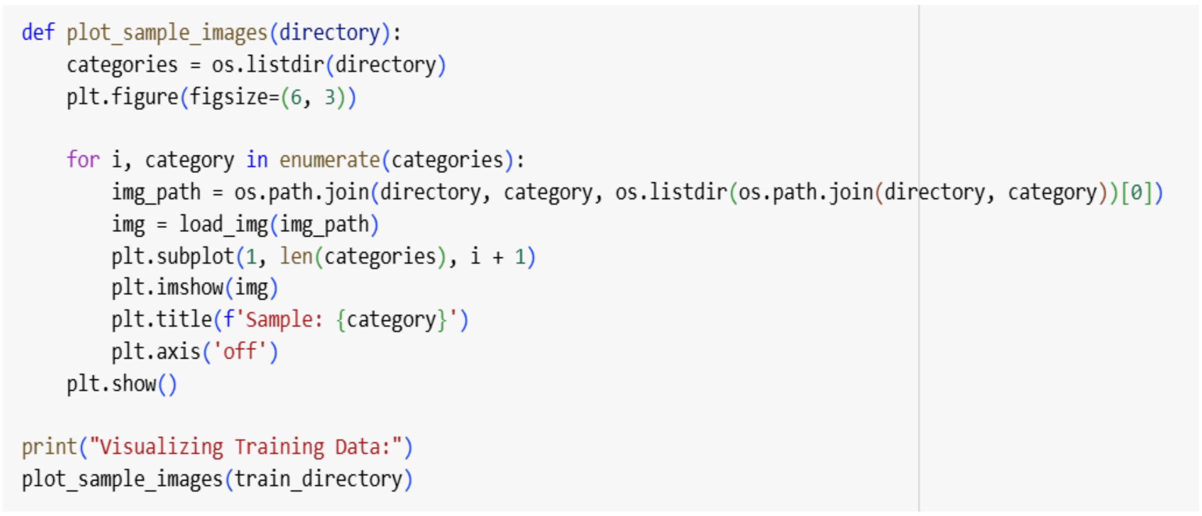


```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory(
    train_directory,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

test_set = test_datagen.flow_from_directory(
    test_directory,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)
```

Milestone 3: Exploratory Data Analysis

EDA for this project involves visualizing sample images to confirm the data is loaded correctly and to understand the classes.

Code: Data Visualization



```
def plot_sample_images(directory):
    categories = os.listdir(directory)
    plt.figure(figsize=(6, 3))

    for i, category in enumerate(categories):
        img_path = os.path.join(directory, category, os.listdir(os.path.join(directory, category))[0])
        img = load_img(img_path)
        plt.subplot(1, len(categories), i + 1)
        plt.imshow(img)
        plt.title(f'Sample: {category}')
        plt.axis('off')
    plt.show()

print("Visualizing Training Data:")
plot_sample_images(train_directory)
```

Milestone 4: Model Building

Activity 1: Training the model

Code: Defining the Transfer Learning Model

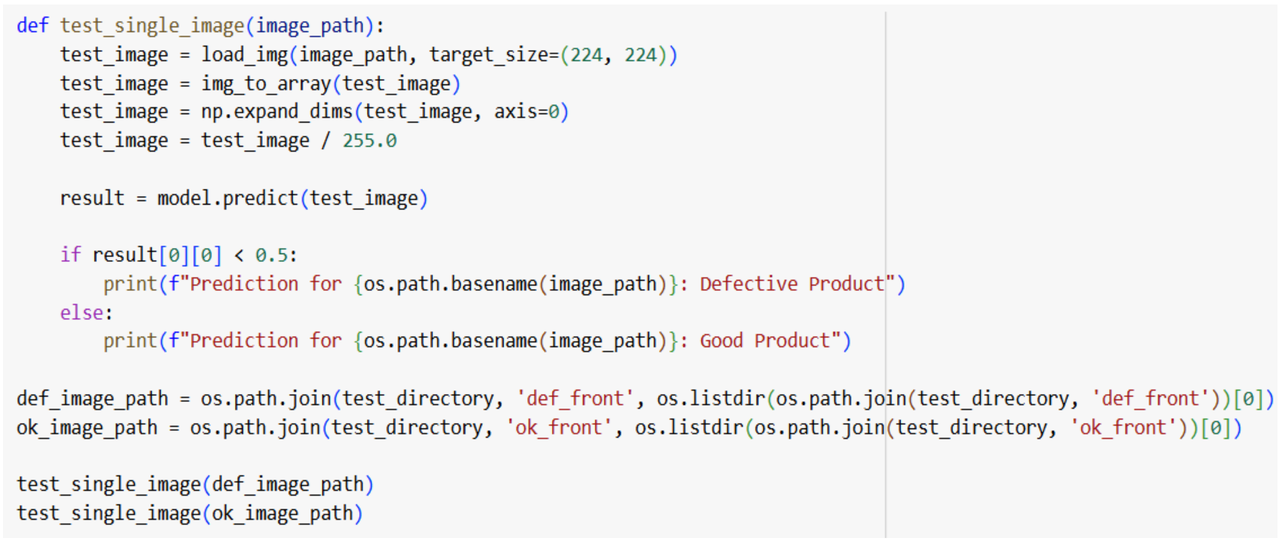
```
base_model = VGG16(
    weights='imagenet',
    include_top=False,
    input_tensor=Input(shape=(224, 224, 3))
)

for layer in base_model.layers:
    layer.trainable = False

x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

Code: Compiling and Training the Model

```
checkpoint = ModelCheckpoint(
    'Vgg16_97.h5',
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    training_set,
    validation_data=test_set,
    epochs=10,
    callbacks=[checkpoint]
)
```

Activity 2: Testing the model

Code: Single Image Test



```python
def test_single_image(image_path):
    test_image = load_img(image_path, target_size=(224, 224))
    test_image = img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    test_image = test_image / 255.0

    result = model.predict(test_image)

    if result[0][0] < 0.5:
        print(f"Prediction for {os.path.basename(image_path)}: Defective Product")
    else:
        print(f"Prediction for {os.path.basename(image_path)}: Good Product")

def_image_path = os.path.join(test_directory, 'def_front', os.listdir(os.path.join(test_directory, 'def_front'))[0])
ok_image_path = os.path.join(test_directory, 'ok_front', os.listdir(os.path.join(test_directory, 'ok_front'))[0])

test_single_image(def_image_path)
test_single_image(ok_image_path)
```

Milestone 5: Performance Evaluation

The model's performance was evaluated by monitoring the validation accuracy during training. The final saved model achieved an accuracy of over 99% on the validation set, indicating that it is highly effective at distinguishing between defective and good products. The close tracking of the training and validation accuracy curves suggests the model has generalized well and is not overfitted.

Milestone 6: Model Deployment

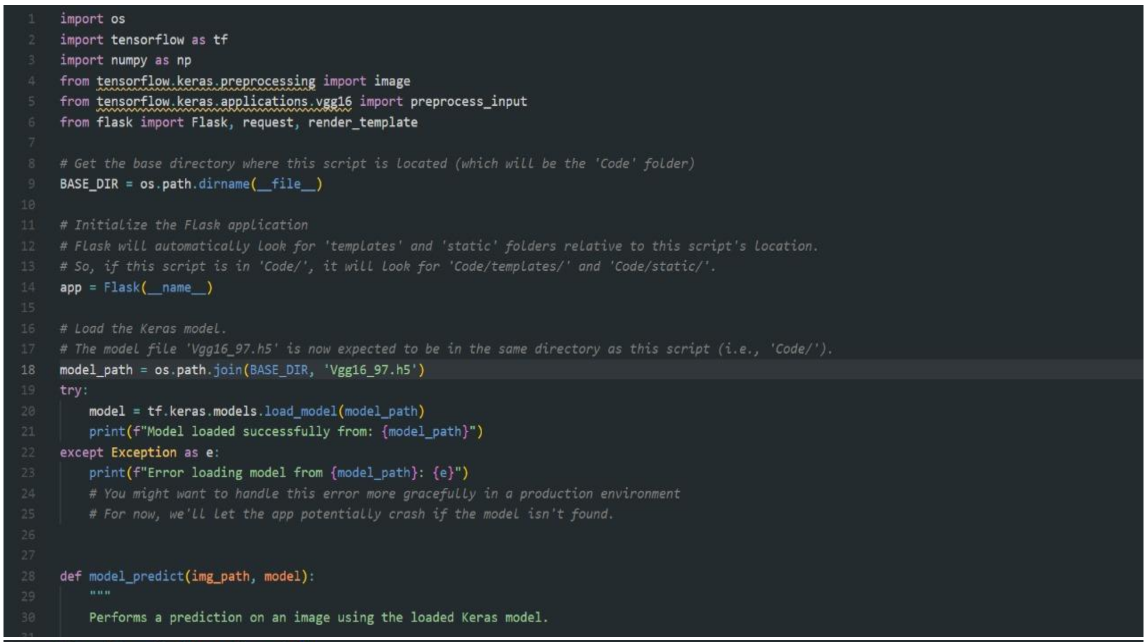
Activity 1: Save the best model

The best model was automatically saved during training as **Vgg16_97.h5**.

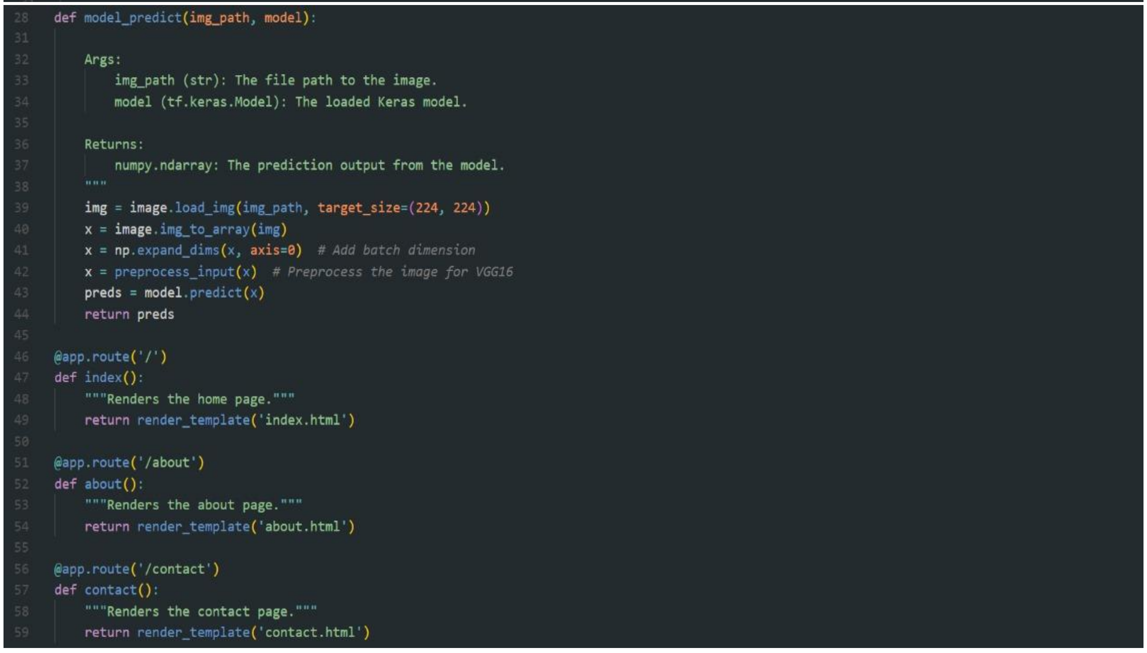
Activity 2: Integrate with Web Framework

A web application was built using Flask. The app.py script loads the saved model and serves the HTML pages.

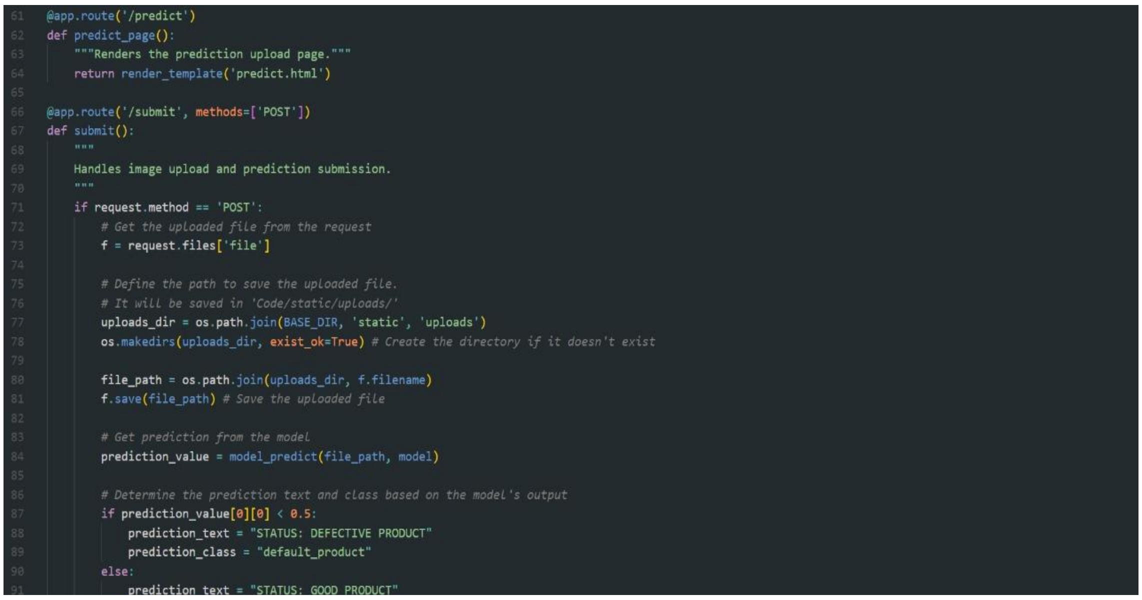
Code: Flask Application (app.py)



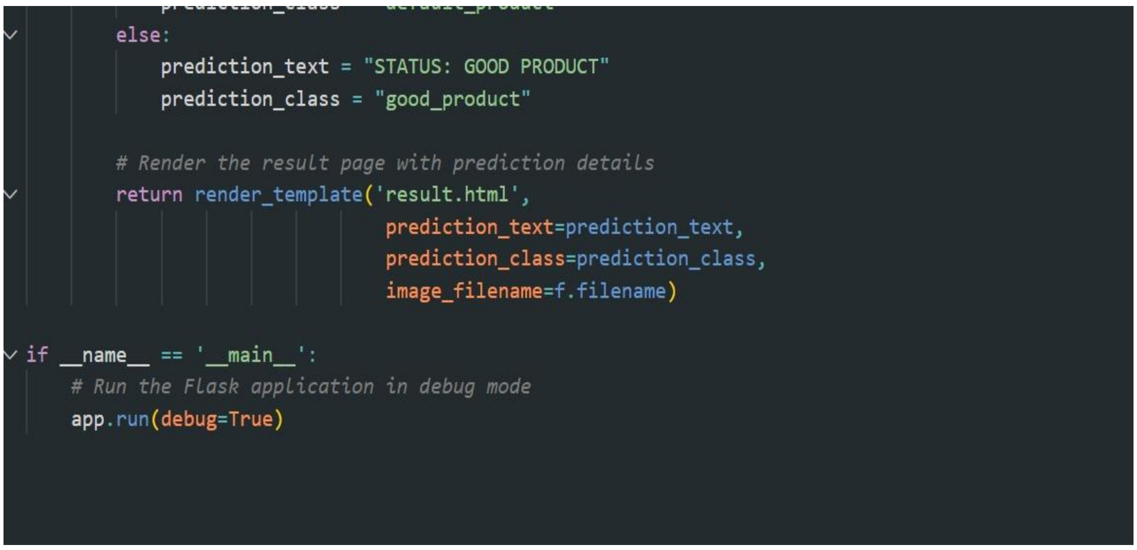
```python
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from flask import Flask, request, render_template

# Get the base directory where this script is located (which will be the 'Code' folder)
BASE_DIR = os.path.dirname(__file__)

# Initialize the Flask application
# Flask will automatically look for 'templates' and 'static' folders relative to this script's location.
# So, if this script is in 'Code/', it will look for 'Code/templates/' and 'Code/static/'.
app = Flask(__name__)

# Load the Keras model.
# The model file 'Vgg16_97.h5' is now expected to be in the same directory as this script (i.e., 'Code/').
model_path = os.path.join(BASE_DIR, 'Vgg16_97.h5')
try:
    model = tf.keras.models.load_model(model_path)
    print(f"Model loaded successfully from: {model_path}")
except Exception as e:
    print(f"Error loading model from {model_path}: {e}")
    # You might want to handle this error more gracefully in a production environment
    # For now, we'll let the app potentially crash if the model isn't found.


def model_predict(img_path, model):
    """
    Performs a prediction on an image using the loaded Keras model.
```



```python
def model_predict(img_path, model):

    Args:
        img_path (str): The file path to the image.
        model (tf.keras.Model): The loaded Keras model.

    Returns:
        numpy.ndarray: The prediction output from the model.
    """
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)  # Add batch dimension
    x = preprocess_input(x)  # Preprocess the image for VGG16
    preds = model.predict(x)
    return preds

@app.route('/')
def index():
    """Renders the home page."""
    return render_template('index.html')

@app.route('/about')
def about():
    """Renders the about page."""
    return render_template('about.html')

@app.route('/contact')
def contact():
    """Renders the contact page."""
    return render_template('contact.html')
```



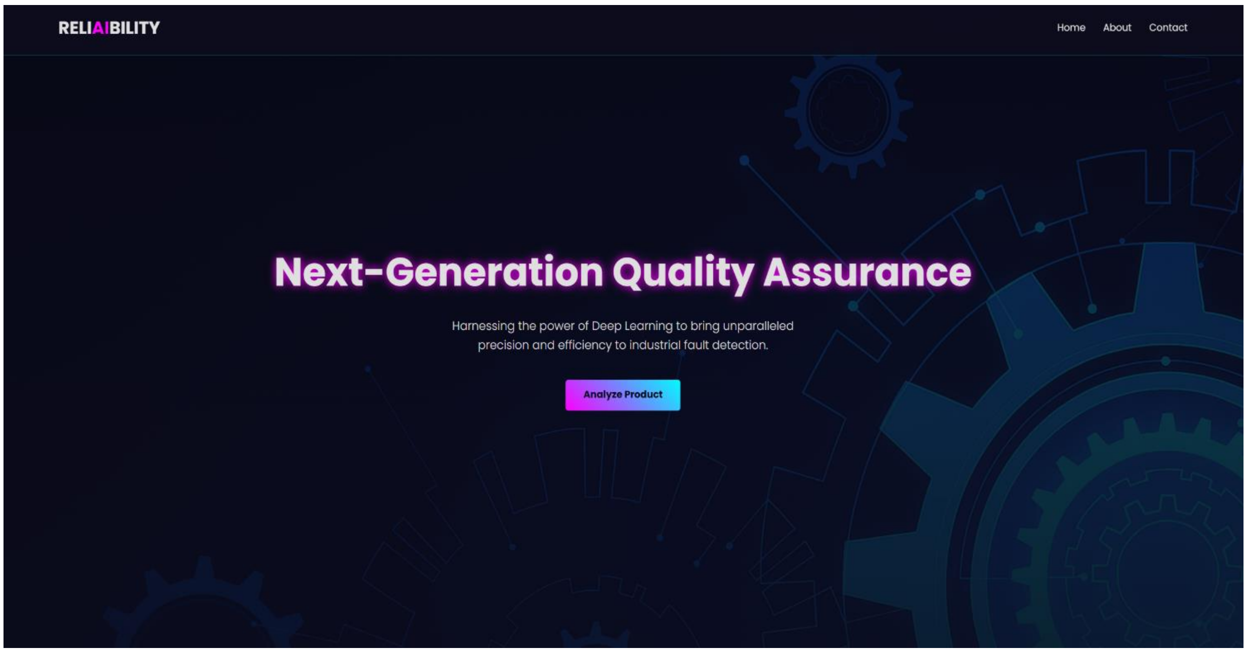
```python
@app.route('/predict')
def predict_page():
    """Renders the prediction upload page."""
    return render_template('predict.html')

@app.route('/submit', methods=['POST'])
def submit():
    """
    Handles image upload and prediction submission.
    """
    if request.method == 'POST':
        # Get the uploaded file from the request
        f = request.files['file']

        # Define the path to save the uploaded file.
        # It will be saved in 'Code/static/uploads/'
        uploads_dir = os.path.join(BASE_DIR, 'static', 'uploads')
        os.makedirs(uploads_dir, exist_ok=True) # Create the directory if it doesn't exist

        file_path = os.path.join(uploads_dir, f.filename)
        f.save(file_path) # Save the uploaded file

        # Get prediction from the model
        prediction_value = model_predict(file_path, model)

        # Determine the prediction text and class based on the model's output
        if prediction_value[0][0] < 0.5:
            prediction_text = "STATUS: DEFECTIVE PRODUCT"
            prediction_class = "default_product"
        else:
            prediction_text = "STATUS: GOOD PRODUCT"
```



```python
        else:
            prediction_text = "STATUS: GOOD PRODUCT"
            prediction_class = "good_product"

        # Render the result page with prediction details
        return render_template('result.html',
                               prediction_text=prediction_text,
                               prediction_class=prediction_class,
                               image_filename=f.filename)

if __name__ == '__main__':
    # Run the Flask application in debug mode
    app.run(debug=True)
```

Milestone 7: Project Demonstration & Documentation

The final application provides a simple and beautiful user interface for real-time predictions. The user navigates from the homepage to the prediction page, uploads an image, and receives a clear, color-coded result indicating the product's quality status.



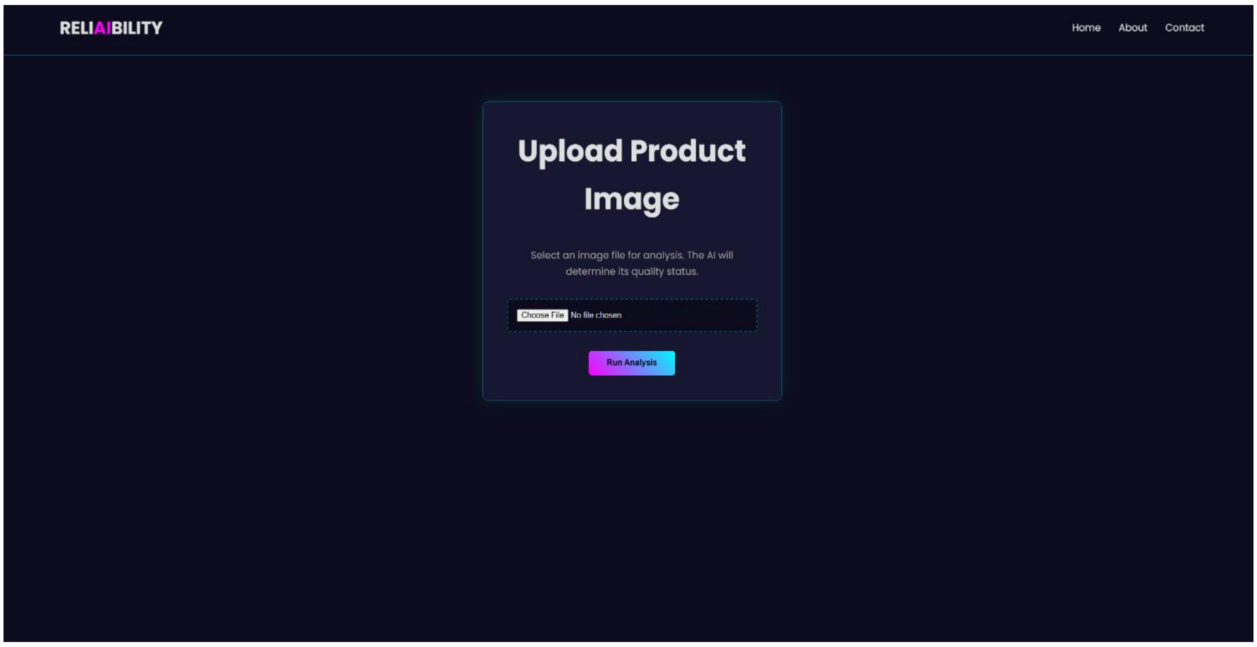
RELIAIBILITY

Home About Contact

Next-Generation Quality Assurance

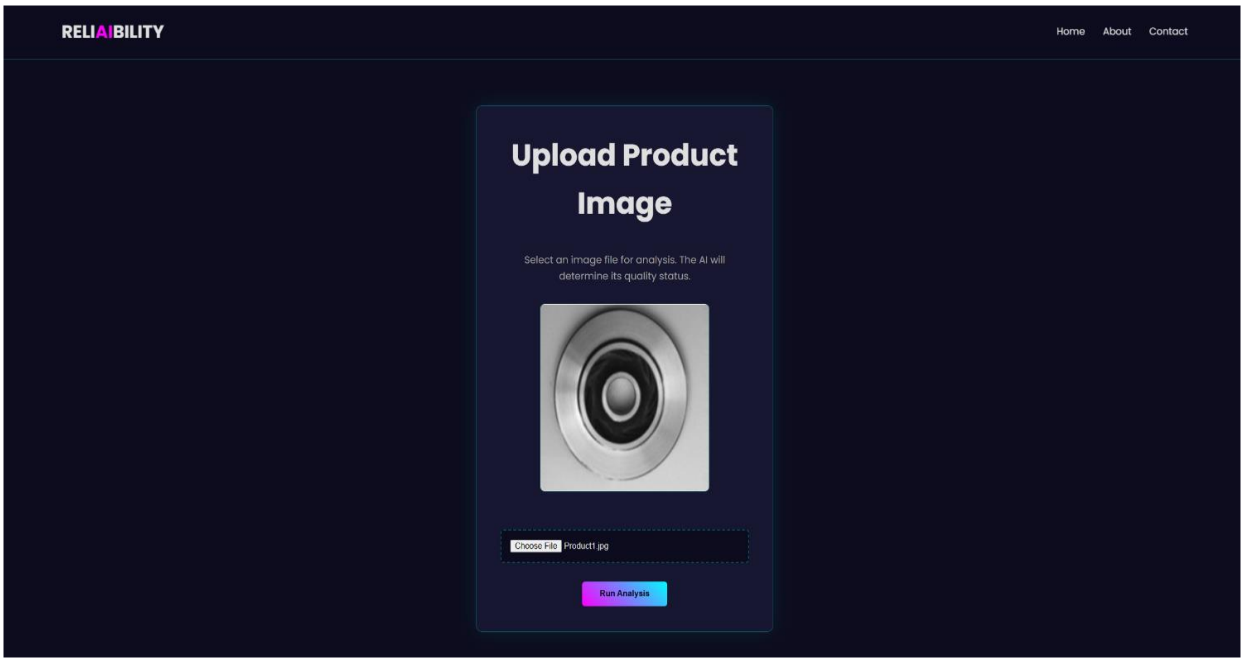
Harnessing the power of Deep Learning to bring unparalleled precision and efficiency to industrial fault detection.

Analyze Product



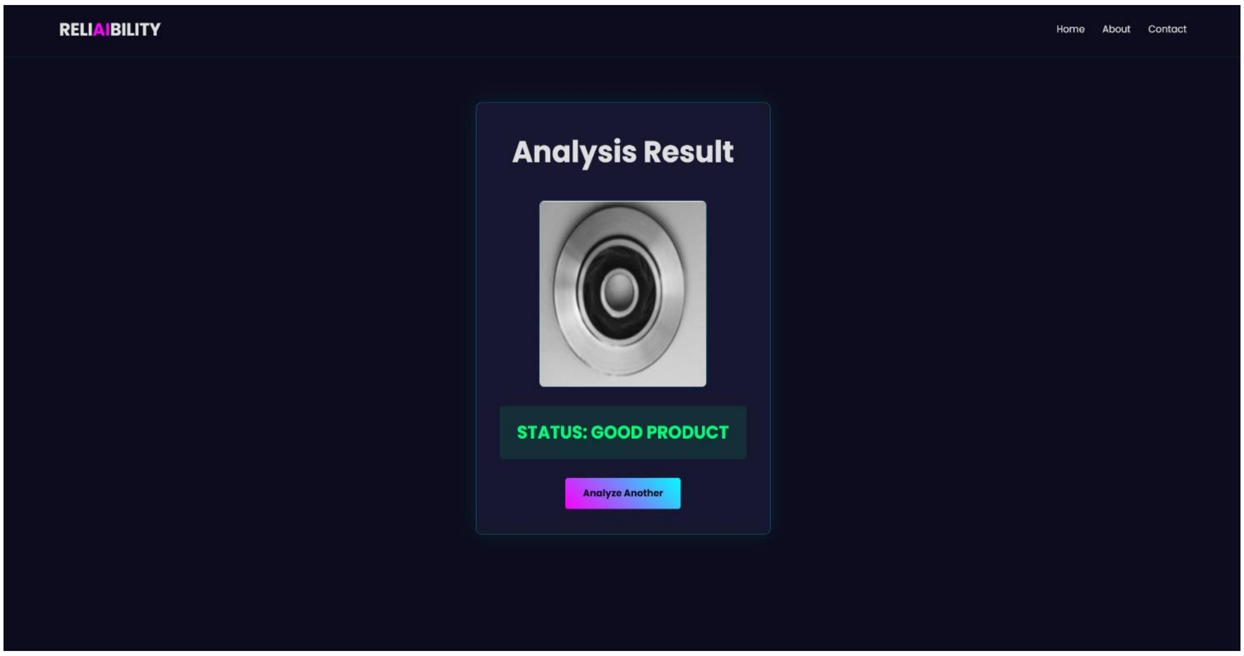
Upload Product Image

Select an image file for analysis. The AI will determine its quality status.



Upload Product Image

Select an image file for analysis. The AI will determine its quality status.

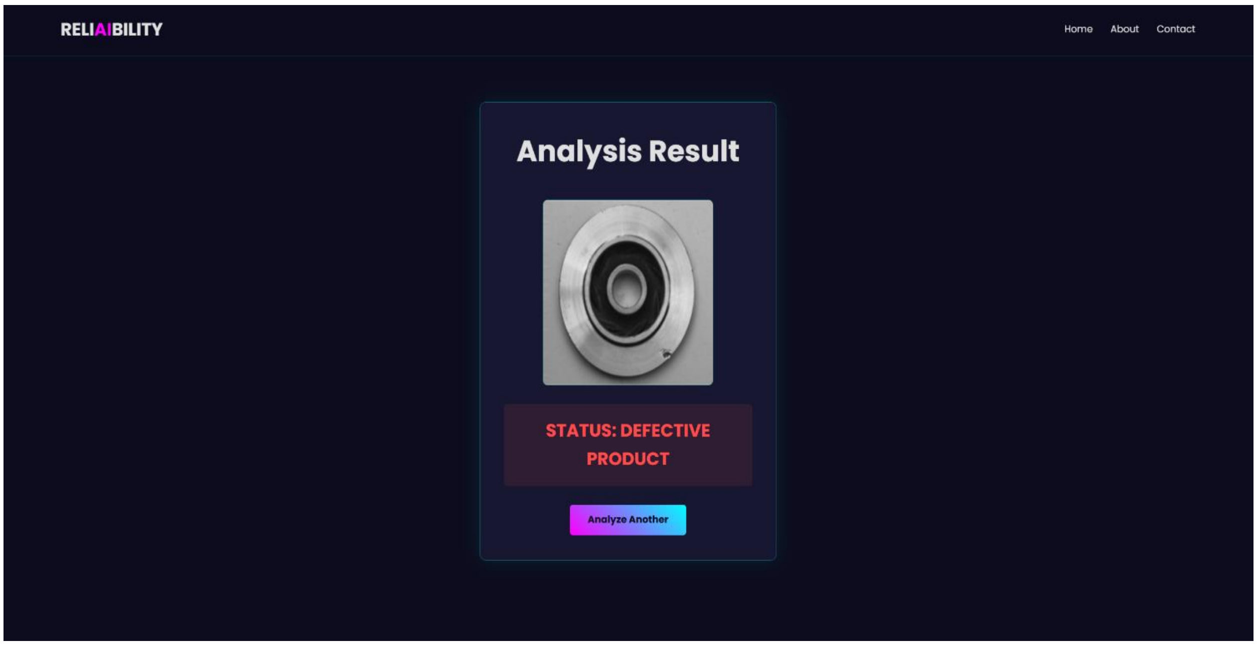

RELIAIBILITY

Home About Contact

Analysis Result

STATUS: GOOD PRODUCT

Analyze Another


RELIAIBILITY

Home About Contact

Analysis Result

STATUS: DEFECTIVE PRODUCT

Analyze Another