Enhancing Product Reliability: Leveraging Transfer Learning for Fault Detection

The "Enhancing Product Reliability: Leveraging Transfer Learning for Fault Detection" project aims to develop an advanced fault detection system utilizing transfer learning techniques in machine learning. This project addresses the critical need for reliable fault detection in various industries, including manufacturing, automotive, and aerospace. By harnessing transfer learning, the project seeks to improve the accuracy and efficiency of detecting faults in products, thereby enhancing overall product reliability and reducing downtime and maintenance costs.

Scenarios

Scenario 1: Manufacturing Quality Control

Collaboration with Manufacturing Industries: Manufacturing companies partner with the project to integrate the developed fault detection system into their quality control processes. Production lines equipped with high-resolution cameras capture images of products in real-time. The transfer learning model, trained on a vast dataset of defect images, analyzes these images to identify potential faults such as cracks, misalignments, or surface irregularities. This automated inspection process significantly enhances the speed and accuracy of quality control, reducing the need for manual inspection and minimizing the risk of defective products reaching the market. As a result, manufacturing companies achieve higher product reliability, customer satisfaction, and reduced warranty claims.

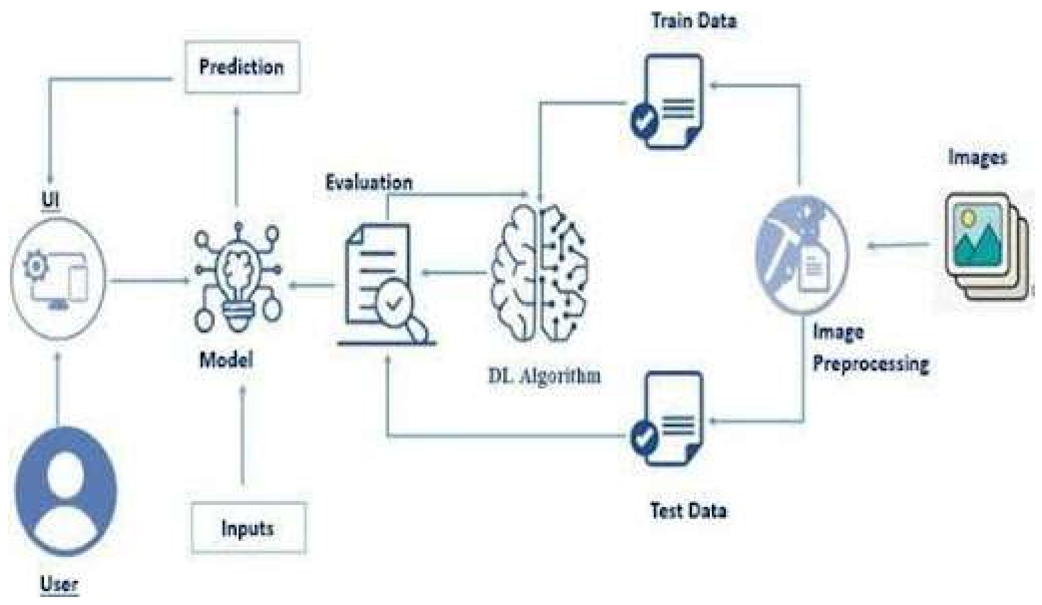
Scenario 2: Automotive Safety Assurance

Enhanced Vehicle Safety Systems: Automotive manufacturers utilize the fault detection system to ensure the safety and reliability of critical vehicle components. During the assembly process, images of engine parts, braking systems, and other essential components are captured and analyzed by the transfer learning model. The system detects anomalies and potential faults that could compromise vehicle safety. By integrating this technology into their production lines, automotive companies can proactively address issues before they escalate, enhancing the overall safety and performance of their vehicles. This leads to improved customer trust, compliance with safety regulations, and a reduction in recalls and associated costs.

Scenario 3: Aerospace Component Inspection

Aerospace Industry Collaboration: Aerospace companies incorporate the fault detection system into their inspection protocols for aircraft components. Given the high stakes in aerospace safety, ensuring the reliability of parts such as turbine blades, fuselage sections, and avionics systems is crucial. The transfer learning model, trained on a diverse set of fault images specific to aerospace components, assists in identifying defects that might otherwise go unnoticed during manual inspections. By leveraging this advanced fault detection technology, aerospace companies can enhance the safety and longevity of their aircraft, reduce maintenance costs, and ensure compliance with stringent industry standards.

Technical Architecture:



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,

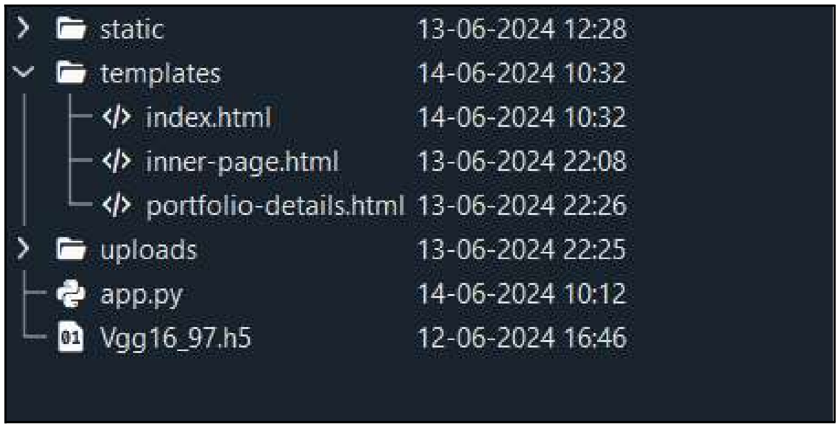
- Data Collection: Collect or download the dataset that you want to train.
- Data Visualization
- Data pre-processing
  - Splitting data into train and test
- Model building
  - Import the model-building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating the performance of the model
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code

Prior Knowledge

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

- 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



```
> static                      13-06-2024 12:28
v templates                   14-06-2024 10:32
    index.html                14-06-2024 10:32
    inner-page.html           13-06-2024 22:08
    portfolio-details.html    13-06-2024 22:26
> uploads                     13-06-2024 22:25
  app.py                      14-06-2024 10:12
  Vgg16_97.h5                 12-06-2024 16:46
```

- 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.

Milestone 1: Define problem

The "Project Initialization and Planning Phase" marks the project's outset, defining goals, scope, and stakeholders. This crucial phase establishes project parameters, identifies key team members, allocates resources, and outlines a realistic timeline. It also involves risk assessment and mitigation planning. Successful initiation sets the foundation for a well-organized and efficiently executed machine learning project, ensuring clarity, alignment, and proactive measures for potential challenges.

Activity 1: Define Problem Statement

Problem Statement: A person working in the casting industry wants to ensure that every product they ship is free from defect, but the current inspection process is manual and inconsistent as it relies on human judgment, which is time-consuming and prone to error which worries them about costly rejections and production delays.

Activity 2: Project Proposal (Proposed Solution)

The proposed project, " Fault detection for enhancing product reliability," aims to leverage transfer learning for more accurate fault detection. Using a comprehensive dataset of images of parts labeled ok or defective, the project seeks to develop a predictive model optimizing fault detection processes. This initiative aligns with our objective to automate the detection of casting defects in manufacturing products, avoiding costly rejection of entire orders and reducing inspection time.

Activity 3: Initial Project Planning

Initial Project Planning involves outlining key objectives, defining scope, and identifying stakeholders for a fault detection system. It encompasses setting timelines, allocating resources, and determining the overall project strategy. During this phase, the team establishes a clear understanding of the dataset, formulates goals for analysis, and plans the workflow for data processing. Effective initial planning lays the foundation for a systematic and well-executed project, ensuring successful outcomes.

Milestone 2: Data Collection and Preprocessing Phase

The Data Collection and Preprocessing Phase involves executing a plan to gather relevant loan application data from Kaggle, ensuring data quality through verification and addressing missing values. Preprocessing tasks include cleaning, encoding, and organizing the dataset for subsequent exploratory analysis and machine learning model development.

Activity 1: Data Collection Plan, Raw Data Sources Identified, Data Quality Report

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

In this project,This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

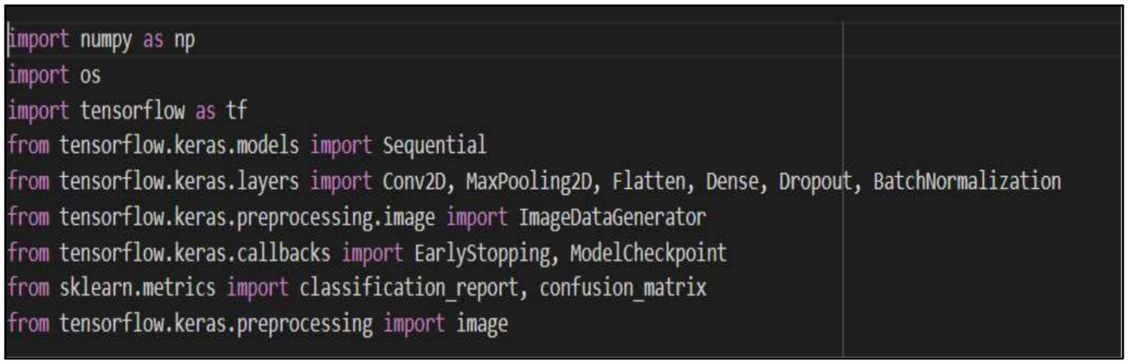
Link:LINK

The dataset comprises images of submersible pump impellers from the casting manufacturing process, aimed at detecting casting defects. It includes 7,348 grayscale images (300x300 pixels, augmented) and 1,300 images (512x512 pixels, non-augmented) categorized into "defective" and "ok" classes.

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

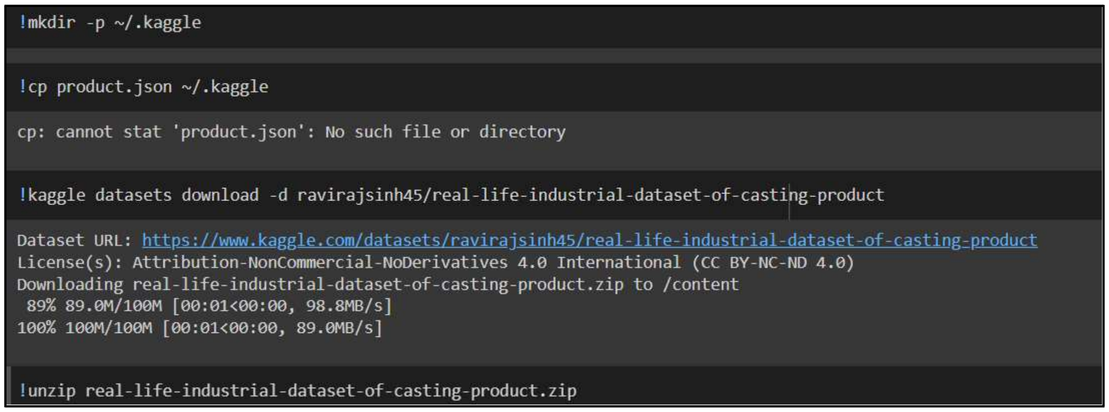
Note: There are many techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques
We are going to build our training model on Google Colab.

Activity 1.1: Importing the libraries:



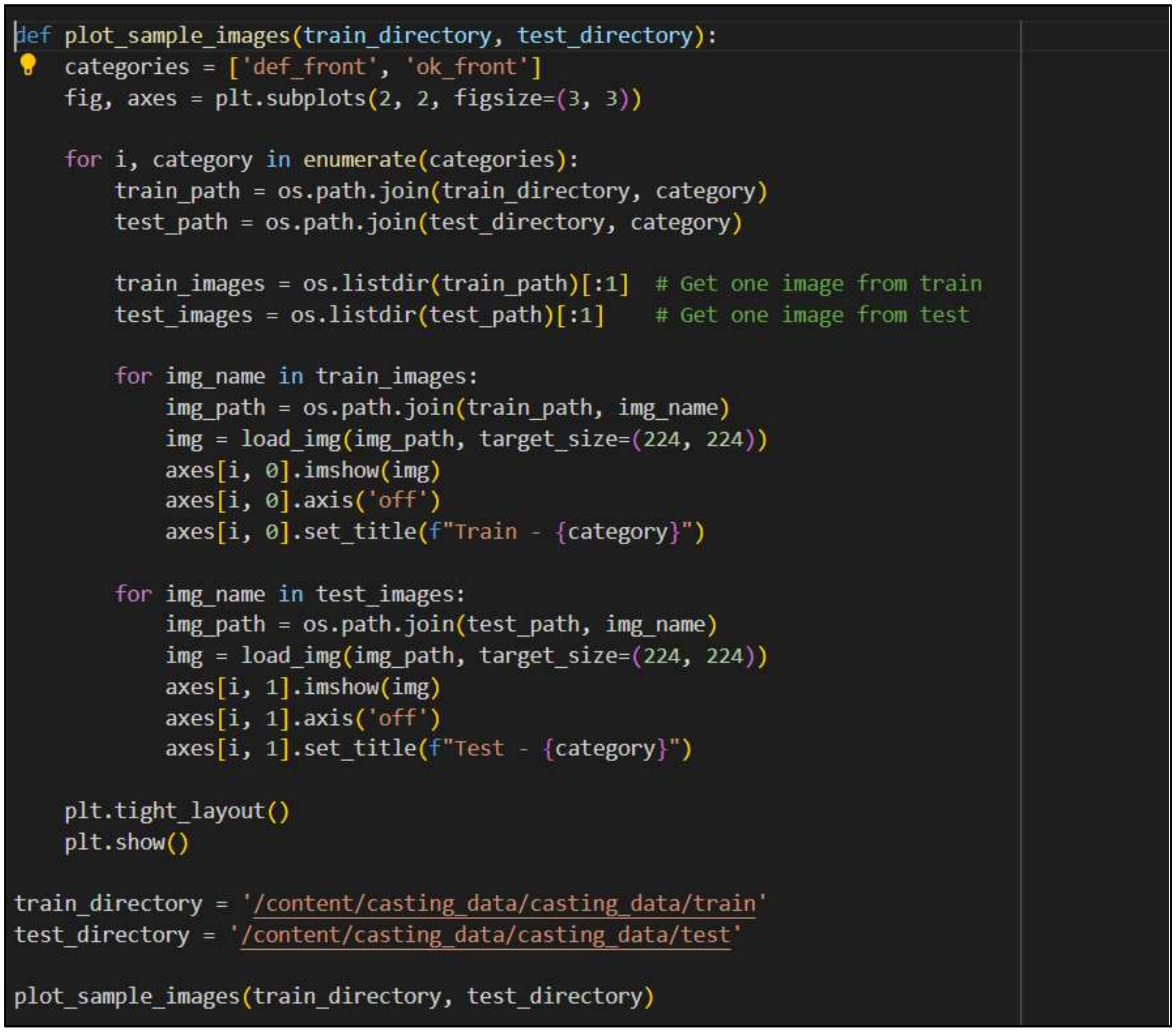
```
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing import image
```

Activity 1.2:Reading Dataset



```
!mkdir -p ~/.kaggle

!cp product.json ~/.kaggle

cp: cannot stat 'product.json': No such file or directory

!kaggle datasets download -d ravirajsinh45/real-life-industrial-dataset-of-casting-product

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
 89% 89.0M/100M [00:01<00:00, 98.8MB/s]
100% 100M/100M [00:01<00:00, 89.0MB/s]

!unzip real-life-industrial-dataset-of-casting-product.zip
```

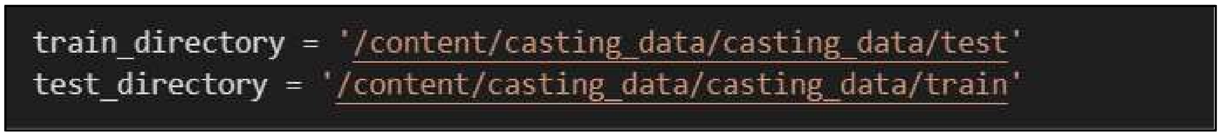
Activity 2: Data Exploration and Preprocessing



```python
def plot_sample_images(train_directory, test_directory):
    categories = ['def_front', 'ok_front']
    fig, axes = plt.subplots(2, 2, figsize=(3, 3))

    for i, category in enumerate(categories):
        train_path = os.path.join(train_directory, category)
        test_path = os.path.join(test_directory, category)

        train_images = os.listdir(train_path)[:1]  # Get one image from train
        test_images = os.listdir(test_path)[:1]    # Get one image from test

        for img_name in train_images:
            img_path = os.path.join(train_path, img_name)
            img = load_img(img_path, target_size=(224, 224))
            axes[i, 0].imshow(img)
            axes[i, 0].axis('off')
            axes[i, 0].set_title(f"Train - {category}")

        for img_name in test_images:
            img_path = os.path.join(test_path, img_name)
            img = load_img(img_path, target_size=(224, 224))
            axes[i, 1].imshow(img)
            axes[i, 1].axis('off')
            axes[i, 1].set_title(f"Test - {category}")

    plt.tight_layout()
    plt.show()

train_directory = '/content/casting_data/casting_data/train'
test_directory = '/content/casting_data/casting_data/test'

plot_sample_images(train_directory, test_directory)
```

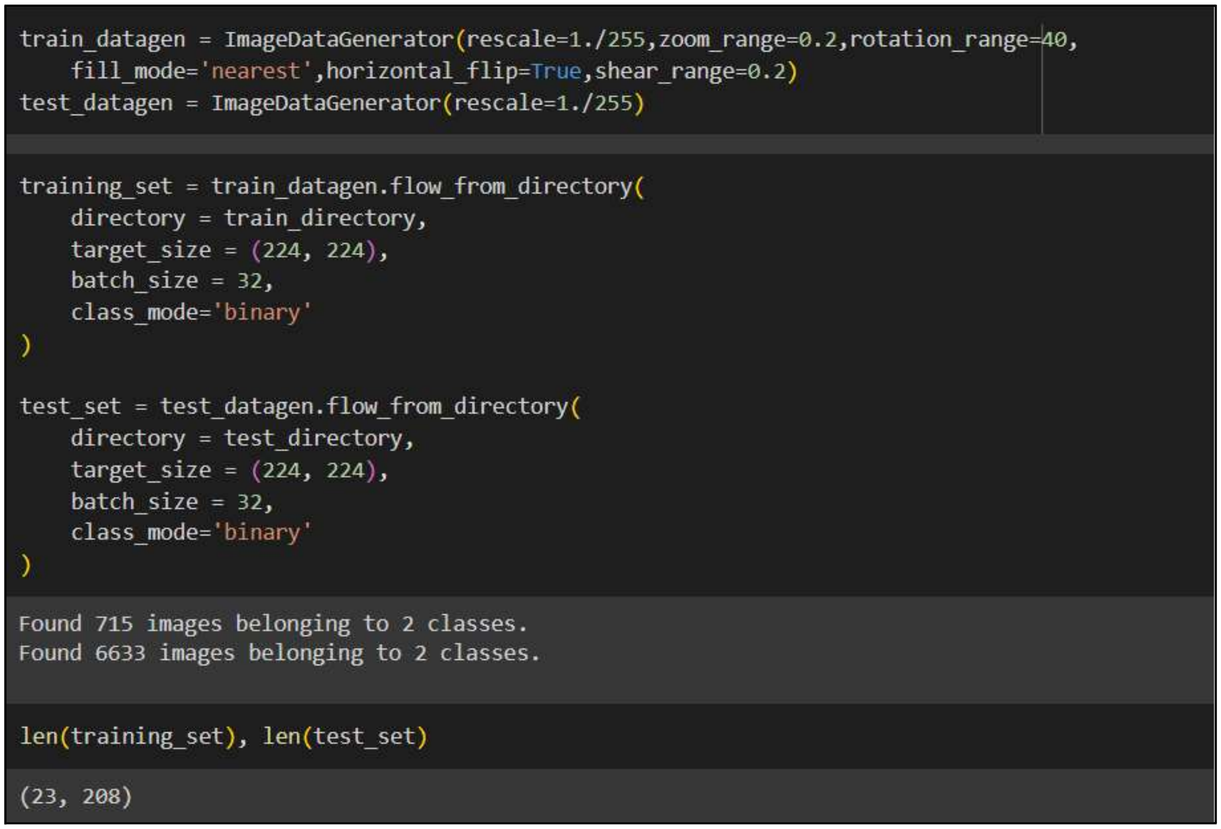
The function `plot_sample_images` is designed to visualize sample images from the training and testing directories for a transfer learning project involving casting defect detection. The function takes two arguments: `train_directory` and `test_directory`, each containing subdirectories for different categories of images (in this case, 'def_front' and 'ok_front'). The function sets up a 2x2 grid of subplots, where each row corresponds to a category and each column corresponds to the train or test dataset. It selects one image from each category's train and test directories, loads these images, resizes them to 224x224 pixels, and displays them in the appropriate subplot with titles indicating their source. This visualization helps in quickly verifying the content and quality of the images used for training and testing the model.

Milestone 3: Model Development Phase



```
train_directory = '/content/casting_data/casting_data/test'
test_directory = '/content/casting_data/casting_data/train'
```

The code snippet defines two variables, `train_directory` and `test_directory`, which specify the file paths for the training and testing datasets, respectively. The `train_directory` points to the location where the test images for the casting products are stored, while the `test_directory` points to the location of the training images. These directories contain images of casting products, categorized into two main classes: defective and okay. By organizing the data into these directories, the dataset is set up for training a deep learning model to automatically detect defects in casting products, streamlining the quality inspection process.



```
train_datagen = ImageDataGenerator(rescale=1./255,zoom_range=0.2,rotation_range=40,
    fill_mode='nearest',horizontal_flip=True,shear_range=0.2)
test_datagen = ImageDataGenerator(rescale=1./255)
```

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

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

```
Found 715 images belonging to 2 classes.
Found 6633 images belonging to 2 classes.
```

```
len(training_set), len(test_set)
```

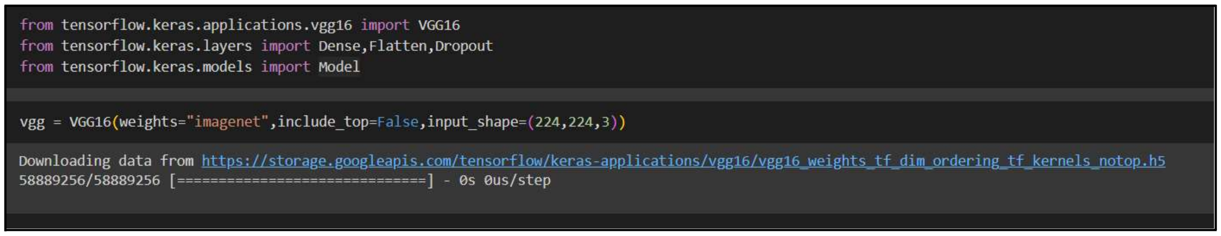
```
(23, 208)
```

The provided code initializes train_datagen and test_datagen using ImageDataGenerator from TensorFlow's Keras library for data augmentation and normalization. The train_datagen includes various augmentation techniques such as zooming, rotation, horizontal flipping, and shearing, which enhance the diversity of training images and improve model generalization. Conversely, test_datagen only rescales the pixel values of the images to maintain consistency during model evaluation.

The flow_from_directory method is then used to generate batches of augmented images from the specified train_directory and test_directory. For both datasets, the images are resized to 224x224 pixels and batched into groups of 32 for efficient training and evaluation. The class_mode='binary' parameter indicates that the model is trained for a binary classification task, distinguishing between 'def_front' and 'ok_front' categories.

The function outputs confirm that the training dataset contains 715 images distributed across the two classes, while the test dataset consists of 6633 images. These datasets are divided into 23 batches for training and 208 batches for testing, ensuring comprehensive coverage during model training and evaluation phases.

Model Building:

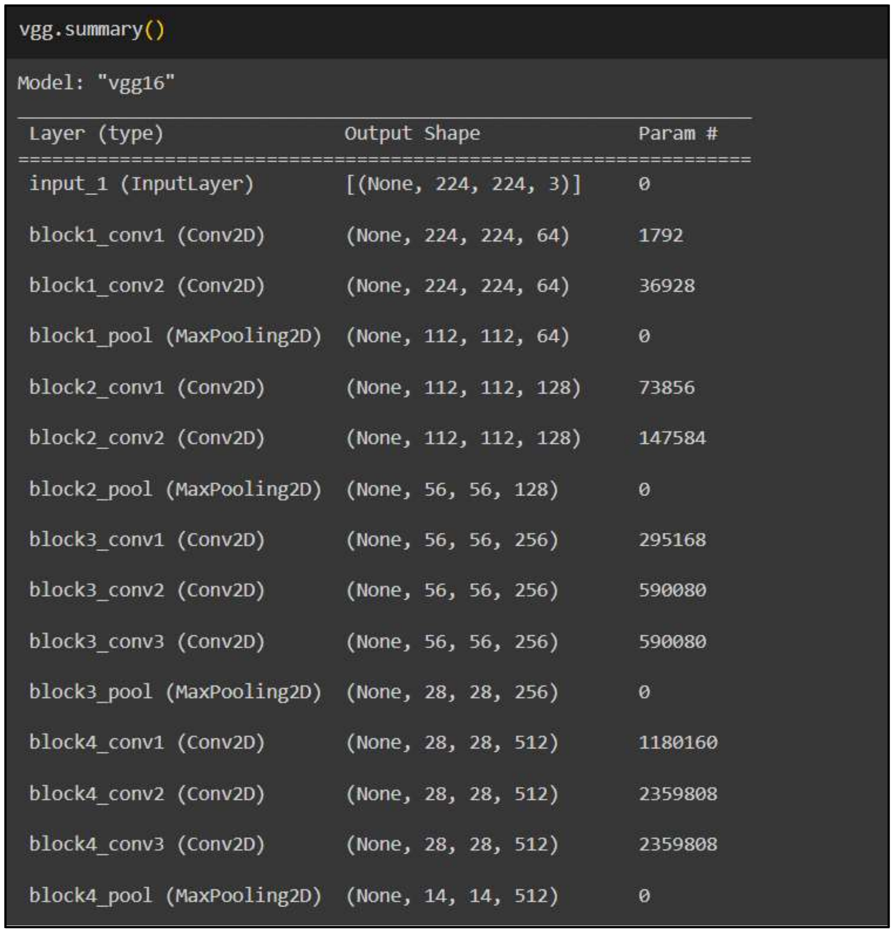


```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense,Flatten,Dropout
from tensorflow.keras.models import Model
```

```
vgg = VGG16(weights="imagenet",include_top=False,input_shape=(224,224,3))
```

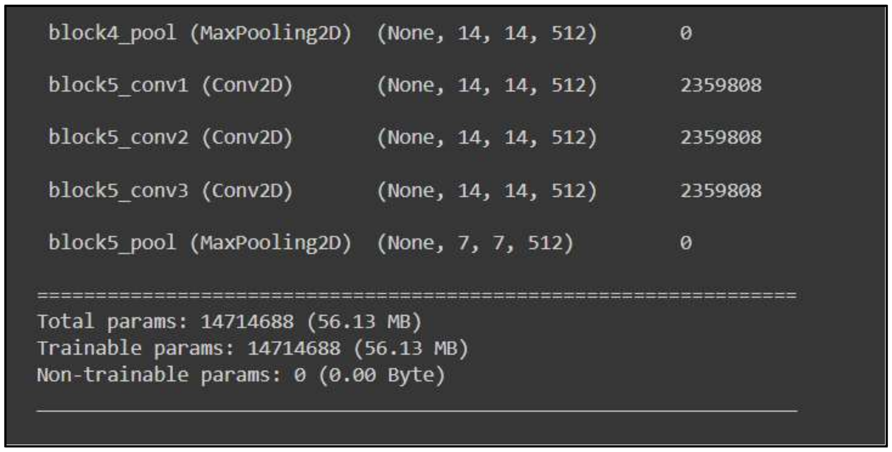
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

The code imports VGG16 from TensorFlow's Keras applications with pre-trained weights from ImageNet, excluding the top layers. This setup enables it to be used as a feature extractor for transfer learning in image-related tasks.

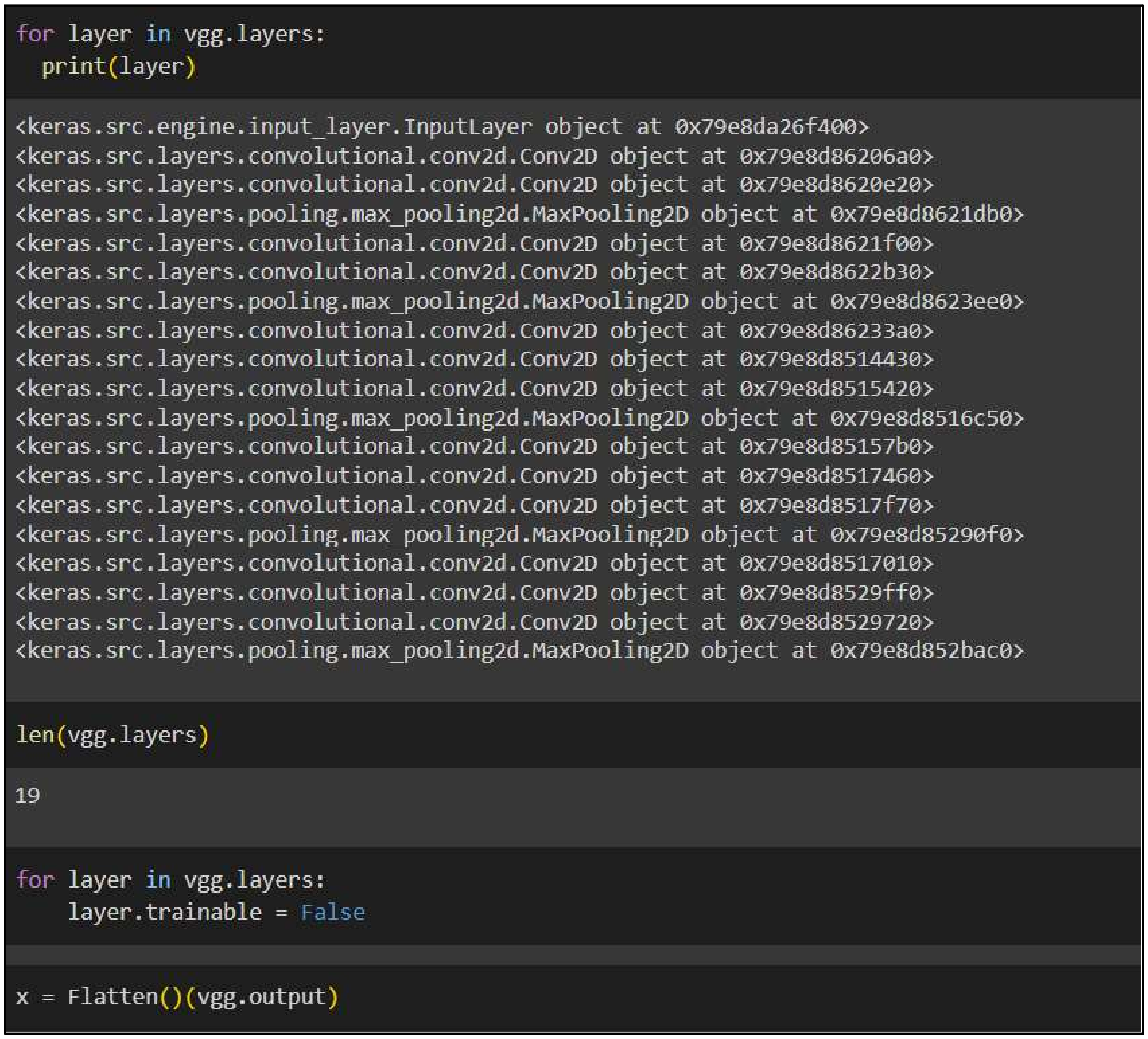


```
vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
```


```
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The VGG16 model summary shows a deep convolutional neural network architecture with five blocks, each consisting of convolutional layers followed by max-pooling layers. It uses ImageNet weights and has 14.7 million trainable parameters, which are fixed unless specified otherwise during transfer learning.



```
for layer in vgg.layers:
  print(layer)
```

```
<keras.src.engine.input_layer.InputLayer object at 0x79e8da26f400>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d86206a0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8620e20>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79e8d8621db0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8621f00>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8622b30>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79e8d8623ee0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d86233a0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8514430>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8515420>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79e8d8516c50>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d85157b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8517460>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8517f70>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79e8d85290f0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8517010>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8529ff0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79e8d8529720>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79e8d852bac0>
```

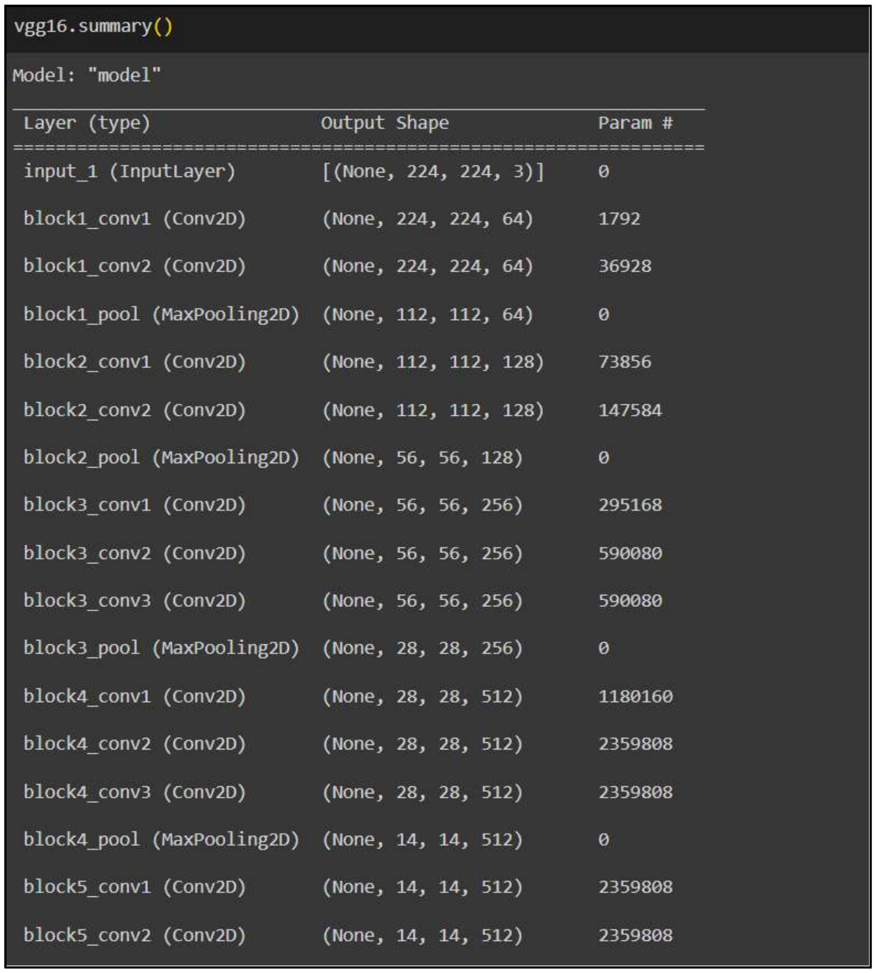
```
len(vgg.layers)
```

```
19
```

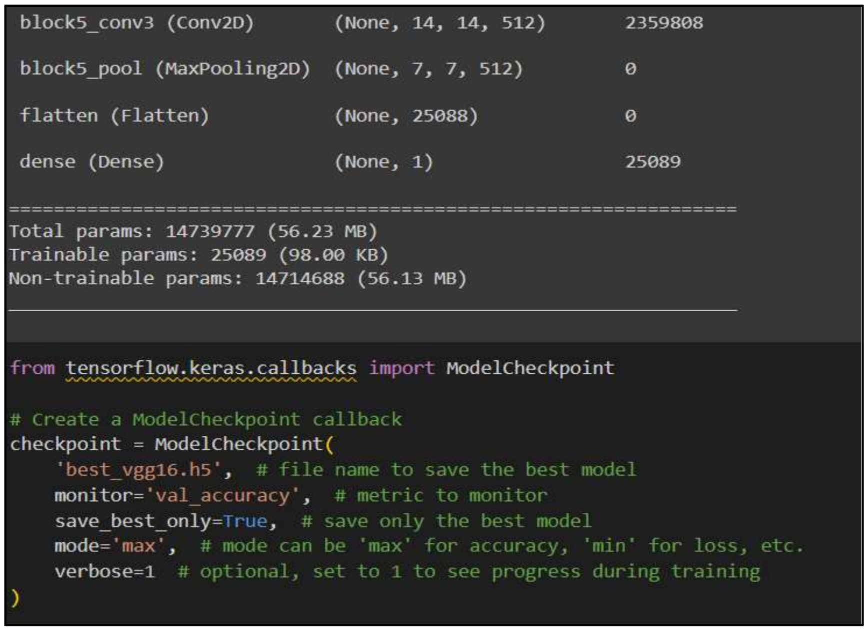
```
for layer in vgg.layers:
    layer.trainable = False
```

```
x = Flatten()(vgg.output)
```

The code inspects and freezes the layers of VGG16, a pre-trained convolutional neural network, by iterating through its 19 layers and setting them as non-trainable (`trainable = False`). Additionally, it appends a `Flatten` layer to the model's output, preparing the feature maps for subsequent classification tasks or further neural network layers.



```
vgg16.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
```

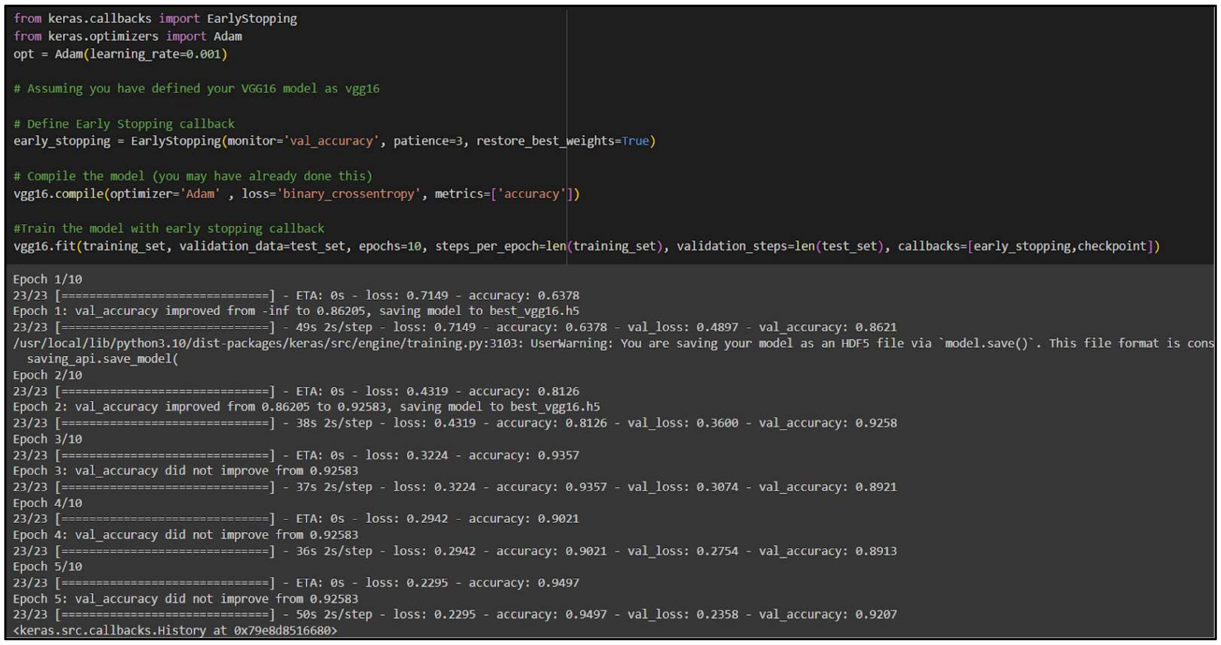


```
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 1)                 25089

=================================================================
Total params: 14739777 (56.23 MB)
Trainable params: 25089 (98.00 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```python
from tensorflow.keras.callbacks import ModelCheckpoint

# Create a ModelCheckpoint callback
checkpoint = ModelCheckpoint(
    'best_vgg16.h5',  # file name to save the best model
    monitor='val_accuracy',  # metric to monitor
    save_best_only=True,  # save only the best model
    mode='max',  # mode can be 'max' for accuracy, 'min' for loss, etc.
    verbose=1  # optional, set to 1 to see progress during training
)
```

The `vgg16.summary()` displays the VGG16 model structure, showing layers like convolutional and pooling with their output shapes and parameters. The

`ModelCheckpoint` callback `checkpoint` saves the best model based on validation accuracy (`val_accuracy`). It stores the model as 'best_vgg16.h5', updating only when validation accuracy improves (`save_best_only=True`) and provides progress updates during training (`verbose=1`).



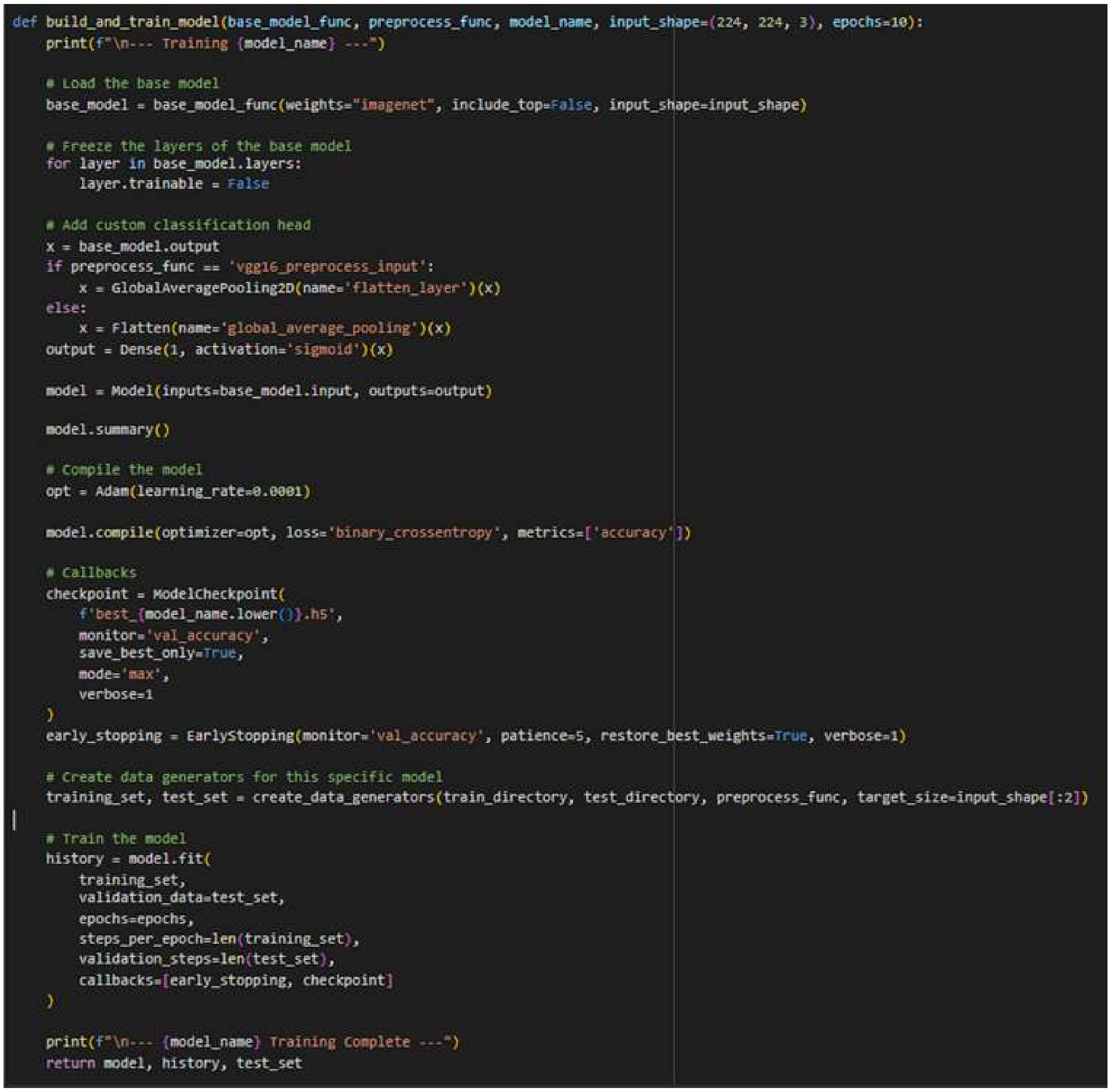
```
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
opt = Adam(learning_rate=0.001)

# Assuming you have defined your VGG16 model as vgg16

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)

# Compile the model (you may have already done this)
vgg16.compile(optimizer='Adam' , loss='binary_crossentropy', metrics=['accuracy'])

#Train the model with early stopping callback
vgg16.fit(training_set, validation_data=test_set, epochs=10, steps_per_epoch=len(training_set), validation_steps=len(test_set), callbacks=[early_stopping,checkpoint])
```

```
Epoch 1/10
23/23 [==============================] - ETA: 0s - loss: 0.7149 - accuracy: 0.6378
Epoch 1: val_accuracy improved from -inf to 0.86205, saving model to best_vgg16.h5
23/23 [==============================] - 49s 2s/step - loss: 0.7149 - accuracy: 0.6378 - val_loss: 0.4897 - val_accuracy: 0.8621
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is cons
  saving_api.save_model(
Epoch 2/10
23/23 [==============================] - ETA: 0s - loss: 0.4319 - accuracy: 0.8126
Epoch 2: val_accuracy improved from 0.86205 to 0.92583, saving model to best_vgg16.h5
23/23 [==============================] - 38s 2s/step - loss: 0.4319 - accuracy: 0.8126 - val_loss: 0.3600 - val_accuracy: 0.9258
Epoch 3/10
23/23 [==============================] - ETA: 0s - loss: 0.3224 - accuracy: 0.9357
Epoch 3: val_accuracy did not improve from 0.92583
23/23 [==============================] - 37s 2s/step - loss: 0.3224 - accuracy: 0.9357 - val_loss: 0.3074 - val_accuracy: 0.8921
Epoch 4/10
23/23 [==============================] - ETA: 0s - loss: 0.2942 - accuracy: 0.9021
Epoch 4: val_accuracy did not improve from 0.92583
23/23 [==============================] - 36s 2s/step - loss: 0.2942 - accuracy: 0.9021 - val_loss: 0.2754 - val_accuracy: 0.8913
Epoch 5/10
23/23 [==============================] - ETA: 0s - loss: 0.2295 - accuracy: 0.9497
Epoch 5: val_accuracy did not improve from 0.92583
23/23 [==============================] - 50s 2s/step - loss: 0.2295 - accuracy: 0.9497 - val_loss: 0.2358 - val_accuracy: 0.9207
<keras.src.callbacks.History at 0x79e8d8516680>
```

The code uses the VGG16 model pretrained on ImageNet and fine-tunes it for binary classification of casting product images. It employs Adam optimizer with a learning rate of 0.001 and sets up Early Stopping with a patience of 3 epochs to prevent overfitting. The model is compiled with binary crossentropy loss and accuracy metrics. During training, it runs for 10 epochs on training and validation sets (training_set and test_set), saving the best model based on validation accuracy to 'best_vgg16.h5'. The training progress and validation results are displayed for each epoch, showing improvements in validation accuracy from 0.862 to 0.926 across epochs.

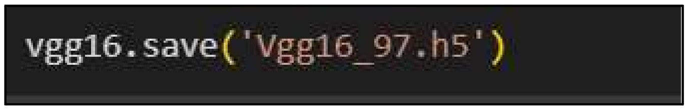
Milestone 4: Model Optimization and Tuning Phase

Input shape: (224, 224, 3)
Epochs: 20
Learning rate: .0001
Optimizer: adam
Loss: binary crossentropy
Preprocessing: vgg16_preprocess_input
EarlyStopping: patience = 5
Additional layers: flatten and dense layer with sigmoid activation function



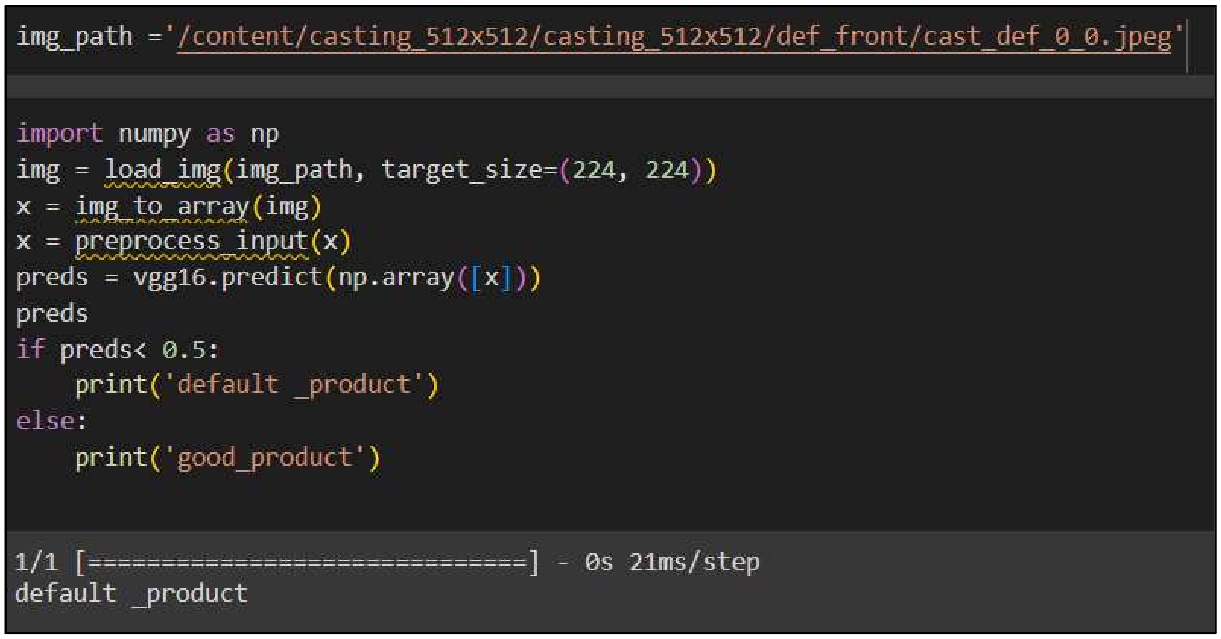
```
def build_and_train_model(base_model_func, preprocess_func, model_name, input_shape=(224, 224, 3), epochs=10):
    print(f"\n--- Training {model_name} ---")

    # Load the base model
    base_model = base_model_func(weights="imagenet", include_top=False, input_shape=input_shape)

    # Freeze the layers of the base model
    for layer in base_model.layers:
        layer.trainable = False

    # Add custom classification head
    x = base_model.output
    if preprocess_func == 'vgg16_preprocess_input':
        x = GlobalAveragePooling2D(name='flatten_layer')(x)
    else:
        x = Flatten(name='global_average_pooling')(x)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=output)

    model.summary()

    # Compile the model
    opt = Adam(learning_rate=0.0001)

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

    # Callbacks
    checkpoint = ModelCheckpoint(
        f'best_{model_name.lower()}.h5',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max',
        verbose=1
    )
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True, verbose=1)

    # Create data generators for this specific model
    training_set, test_set = create_data_generators(train_directory, test_directory, preprocess_func, target_size=input_shape[:2])

    # Train the model
    history = model.fit(
        training_set,
        validation_data=test_set,
        epochs=epochs,
        steps_per_epoch=len(training_set),
        validation_steps=len(test_set),
        callbacks=[early_stopping, checkpoint]
    )

    print(f"\n--- {model_name} Training Complete ---")
    return model, history, test_set
```

Save the Model



```
vgg16.save('Vgg16_97.h5')
```

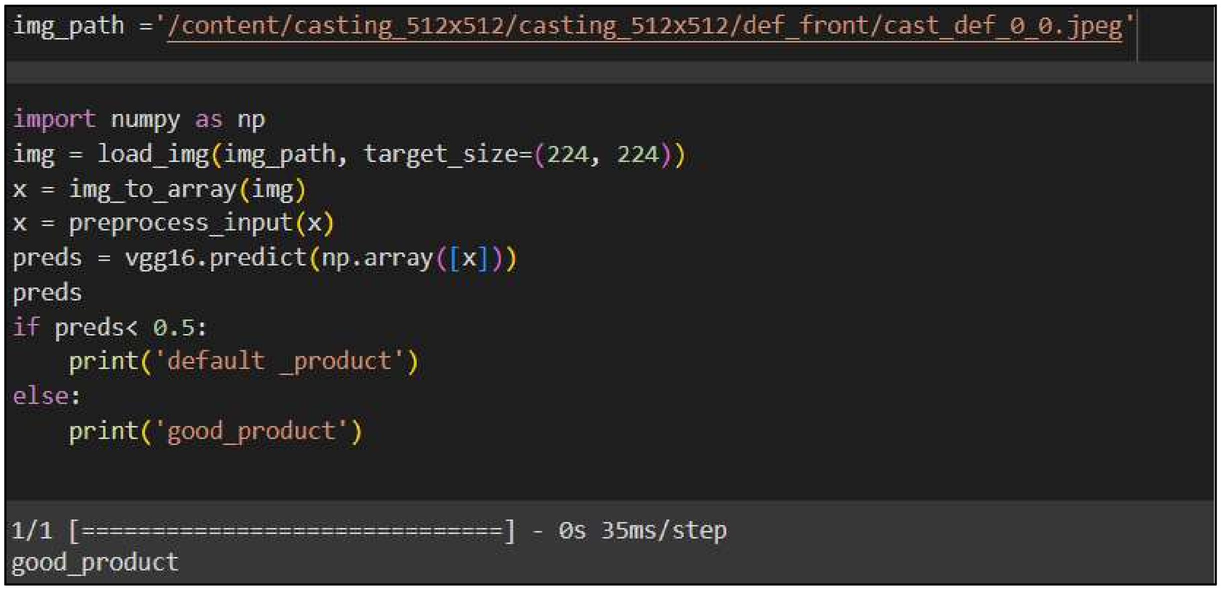
Activity 3.1: Testing the Model



```
img_path ='/content/casting_512x512/casting_512x512/def_front/cast_def_0_0.jpeg'

import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
if preds< 0.5:
    print('default _product')
else:
    print('good_product')
```

```
1/1 [==============================] - 0s 21ms/step
default _product
```

The code uses a pre-trained VGG16 model to classify a casting product image (`cast_def_0_0.jpeg`) as either 'def_front' or 'ok_front'. After loading and preprocessing the image, the model predicts its class probabilities (`preds`). If `preds` is less than 0.5, it indicates a 'default_product' (defective); otherwise, it suggests a 'good_product'. This demonstrates automated defect detection in manufacturing, enhancing quality control processes.



```
img_path ='/content/casting_512x512/casting_512x512/def_front/cast_def_0_0.jpeg'

import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
if preds< 0.5:
    print('default _product')
else:
    print('good_product')
```

```
1/1 [==============================] - 0s 35ms/step
good_product
```

The code snippet utilizes a pre-trained VGG16 model to classify an image (`cast_ok_0_1028.jpeg`) of a casting product as either 'ok_front' or 'def_front'. After loading, converting to an array, and preprocessing the image, the model predicts the class probabilities (`preds`). Since `preds` exceeds 0.5, the output is 'good_product', indicating that the casting is deemed acceptable according to the model's classification. This demonstrates how deep learning can automate quality assessment in manufacturing, improving efficiency and accuracy in identifying defective products.

Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Activity1: Building HTML Pages:

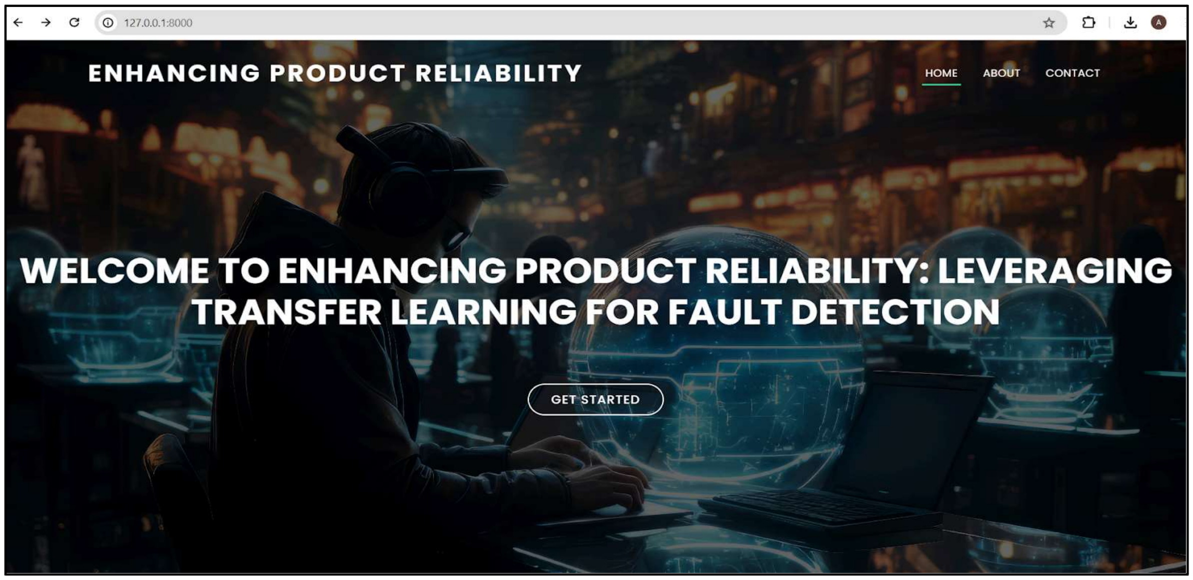
For this project create three HTML files namely

- index.html

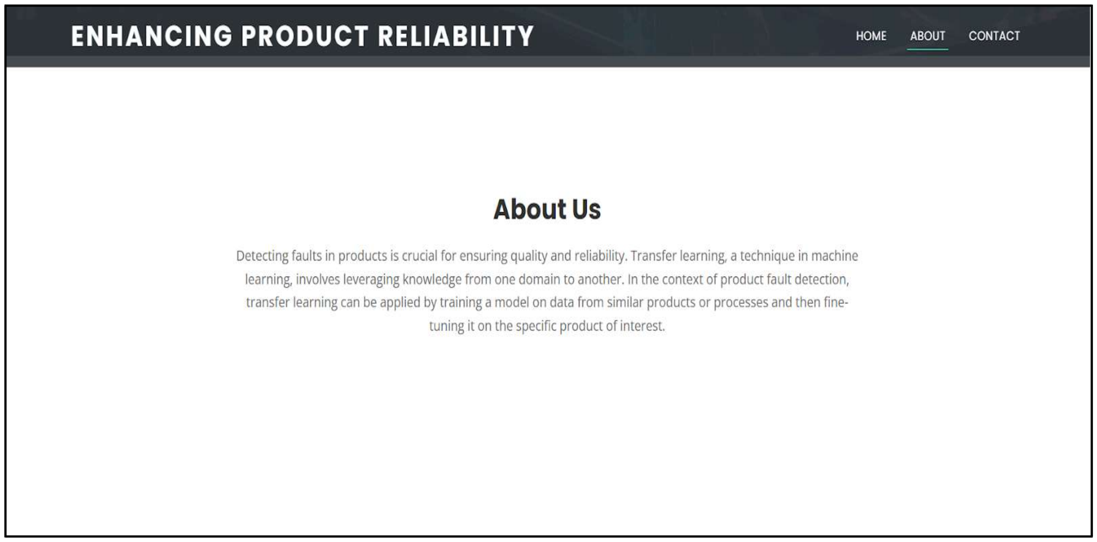
And save them in the templates folder.

UI Image preview:

Let’s see what our index.html page looks like:


127.0.0.1:8000
ENHANCING PRODUCT RELIABILITY
HOME
ABOUT
CONTACT
WELCOME TO ENHANCING PRODUCT RELIABILITY: LEVERAGING TRANSFER LEARNING FOR FAULT DETECTION
GET STARTED

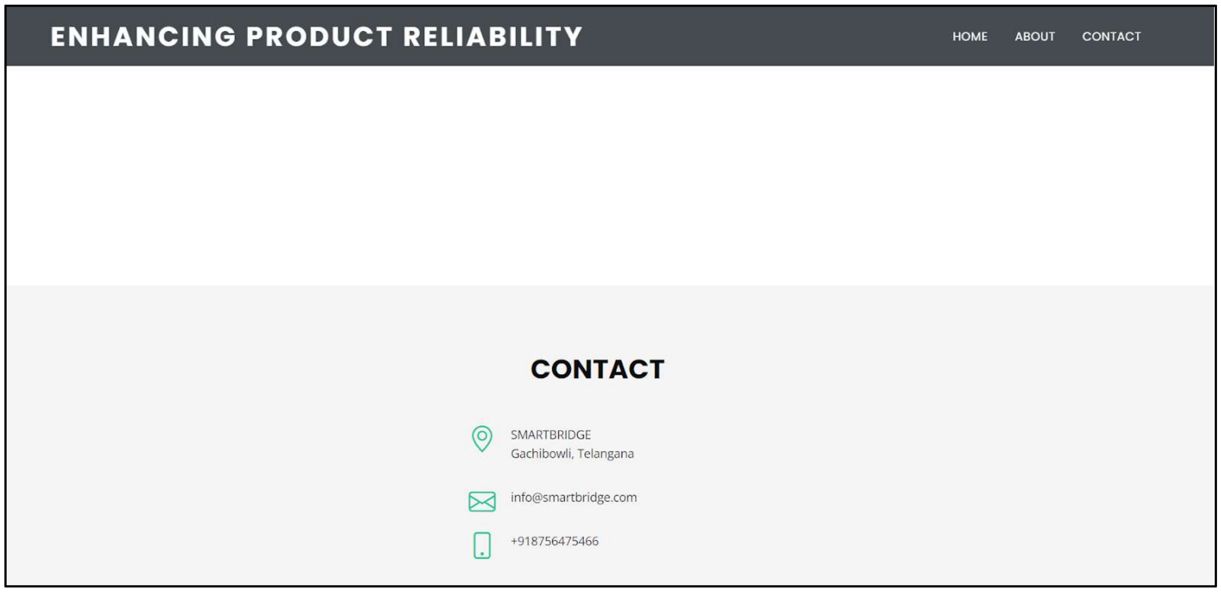
When you click on the About button on the top, you will be redirected to the following page



About Us

Detecting faults in products is crucial for ensuring quality and reliability. Transfer learning, a technique in machine learning, involves leveraging knowledge from one domain to another. In the context of product fault detection, transfer learning can be applied by training a model on data from similar products or processes and then fine-tuning it on the specific product of interest.

When you click on the contact button, it will redirect you to the below page



ENHANCING PRODUCT RELIABILITY

CONTACT

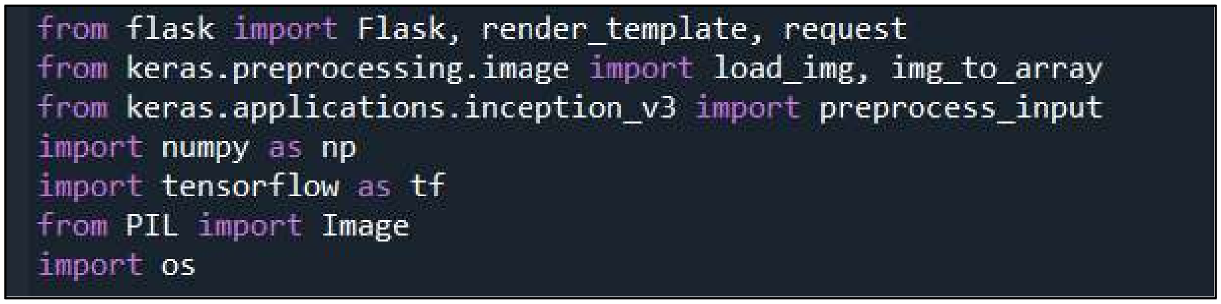
SMARTBRIDGE
Gachibowli, Telangana

info@smartbridge.com

+918756475466

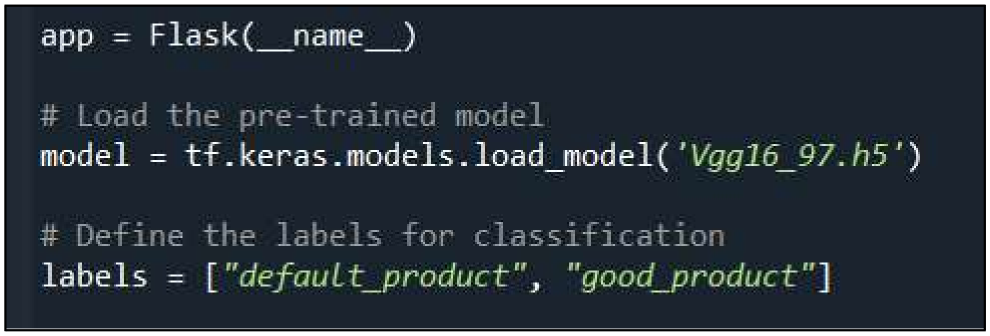
Build Python code:

Import the libraries



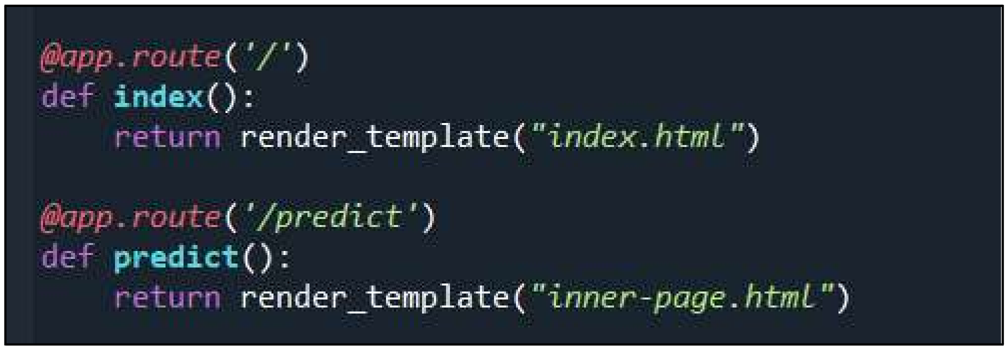
```python
from flask import Flask, render_template, request
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.inception_v3 import preprocess_input
import numpy as np
import tensorflow as tf
from PIL import Image
import os
```

Loading the saved model and initializing the Flask app



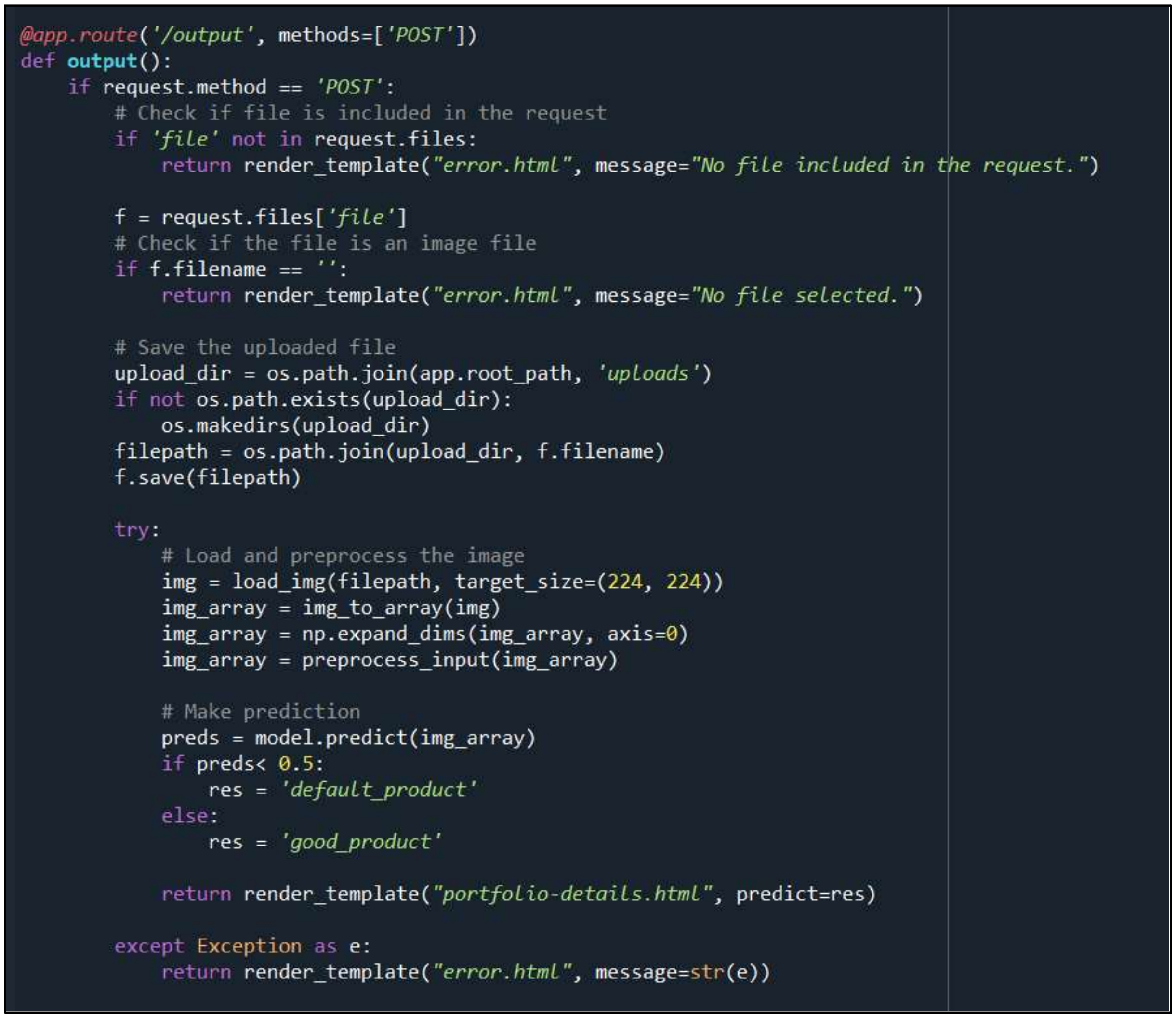
```python
app = Flask(__name__)

# Load the pre-trained model
model = tf.keras.models.load_model('Vgg16_97.h5')

# Define the labels for classification
labels = ["default_product", "good_product"]
```

Render HTML pages:



```python
@app.route('/')
def index():
    return render_template("index.html")

@app.route('/predict')
def predict():
    return render_template("inner-page.html")
```

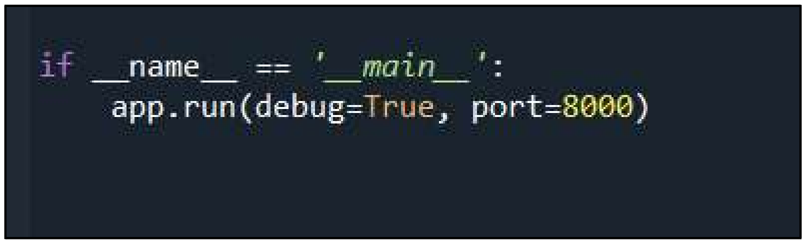
This Flask route /output processes POST requests containing image files. It checks if a file is present and saves it to a designated directory. The uploaded image is then loaded, resized, converted to an array, and preprocessed for model input. Using a pre-trained model, predictions are made on the preprocessed image array. The predicted class is determined based on the highest probability in the prediction array. Finally, the predicted class is passed to an HTML template for rendering, allowing users to see the predicted result. Error handling is implemented to display appropriate messages if no file is included or if the selected file is not an image.



```python
@app.route('/output', methods=['POST'])
def output():
    if request.method == 'POST':
        # Check if file is included in the request
        if 'file' not in request.files:
            return render_template("error.html", message="No file included in the request.")

        f = request.files['file']
        # Check if the file is an image file
        if f.filename == '':
            return render_template("error.html", message="No file selected.")

        # Save the uploaded file
        upload_dir = os.path.join(app.root_path, 'uploads')
        if not os.path.exists(upload_dir):
            os.makedirs(upload_dir)
        filepath = os.path.join(upload_dir, f.filename)
        f.save(filepath)

        try:
            # Load and preprocess the image
            img = load_img(filepath, target_size=(224, 224))
            img_array = img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)
            img_array = preprocess_input(img_array)

            # Make prediction
            preds = model.predict(img_array)
            if preds< 0.5:
                res = 'default_product'
            else:
                res = 'good_product'

            return render_template("portfolio-details.html", predict=res)

        except Exception as e:
            return render_template("error.html", message=str(e))
```

Here we are routing our app to res function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. This prediction value will rendered to the text that we have mentioned in the index.html page earlier.

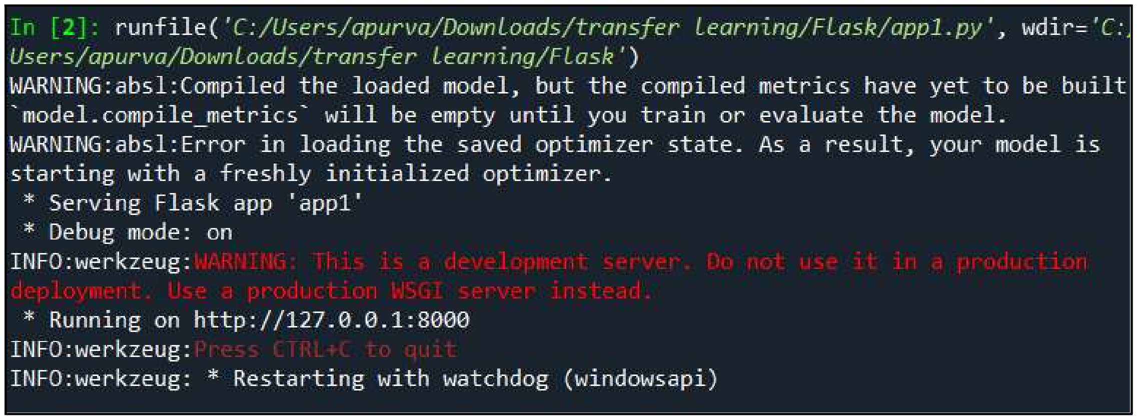
Main Function:



```python
if __name__ == '__main__':
    app.run(debug=True, port=8000)
```

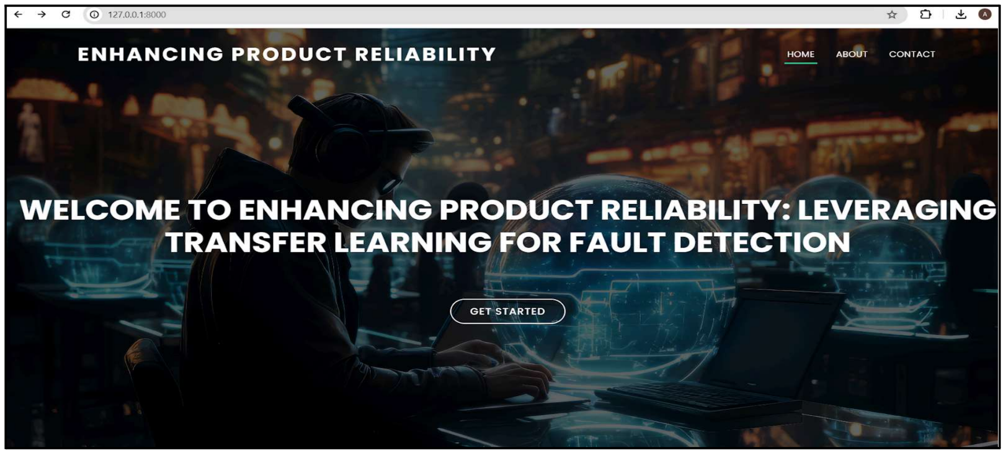
Run the web application

- Open the Anaconda prompt from the start menu.
- Navigate to the folder where your Python script is.
- Now type the “app1.py” command.
- Navigate to the localhost where you can view your web page.

- 

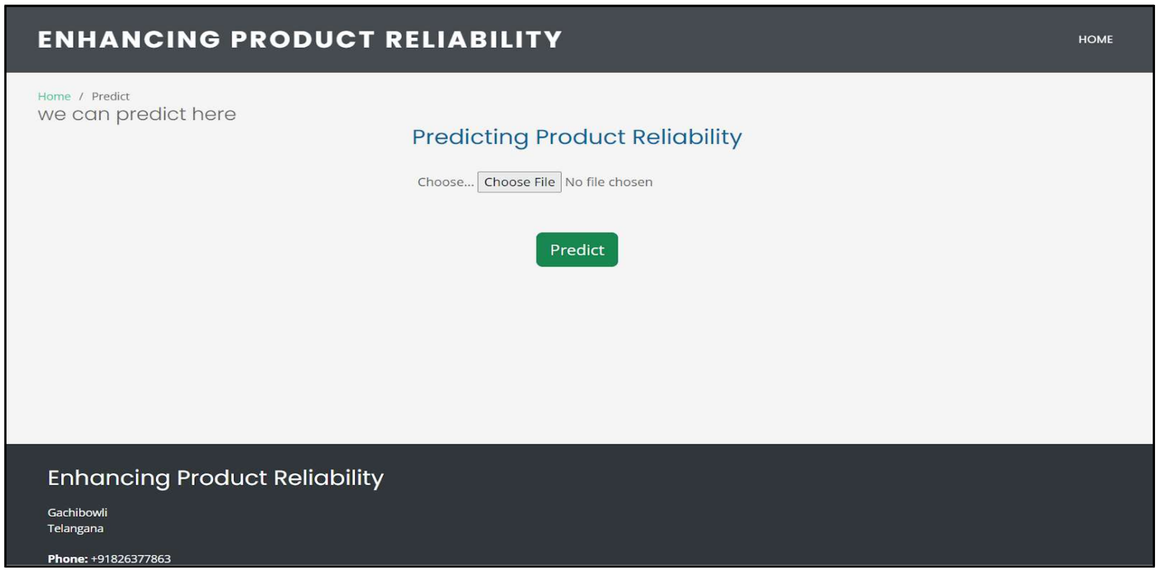
```
In [2]: runfile('C:/Users/apurva/Downloads/transfer learning/Flask/app1.py', wdir='C:/
Users/apurva/Downloads/transfer learning/Flask')
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built
`model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is
starting with a freshly initialized optimizer.
 * Serving Flask app 'app1'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with watchdog (windowsapi)
```

  - Click on the Get Start button, enter the inputs, click on the submit button, and see the result/prediction on the web.
- The home page looks like this. When you click on the get started “Drop in the image you want to validate!”, you’ll be redirected to the predict section


127.0.0.1:8000
ENHANCING PRODUCT RELIABILITY
HOME
ABOUT
CONTACT
WELCOME TO ENHANCING PRODUCT RELIABILITY: LEVERAGING TRANSFER LEARNING FOR FAULT DETECTION
GET STARTED

click on the Get Started button

  - The prediction page looks like this.
-



ENHANCING PRODUCT RELIABILITY

HOME

Home / Predict

we can predict here

Predicting Product Reliability

Choose... Choose File No file chosen

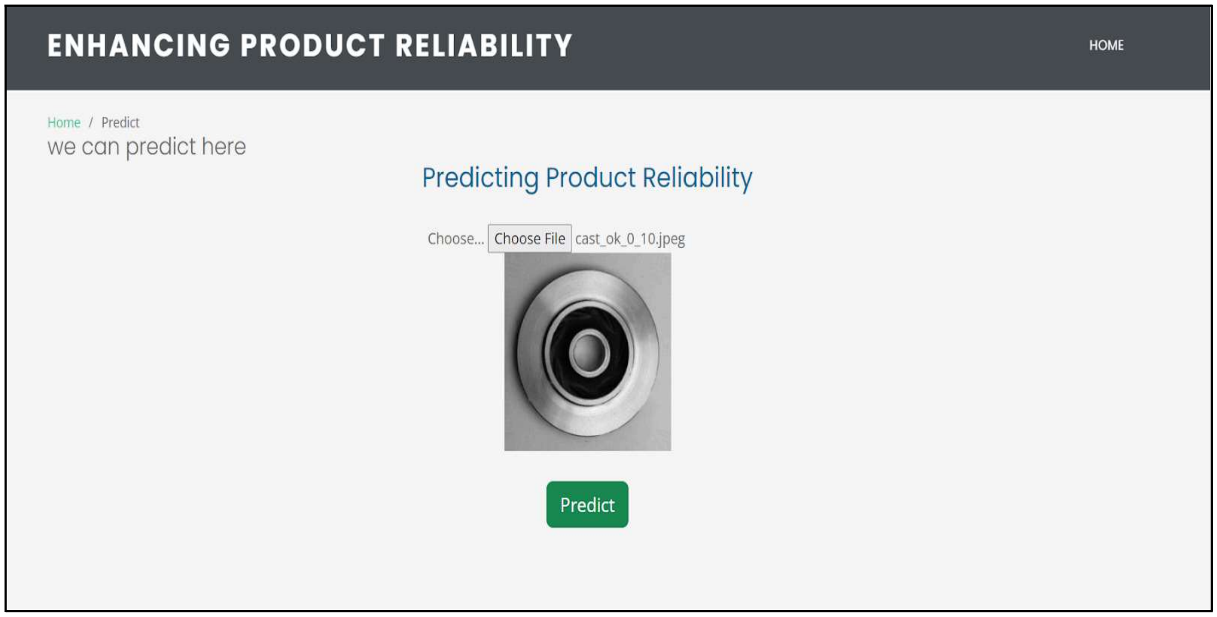
Predict

Enhancing Product Reliability

Gachibowli
Telangana

Phone: +91826377863

INPUT1:



ENHANCING PRODUCT RELIABILITY

HOME

Home / Predict

we can predict here

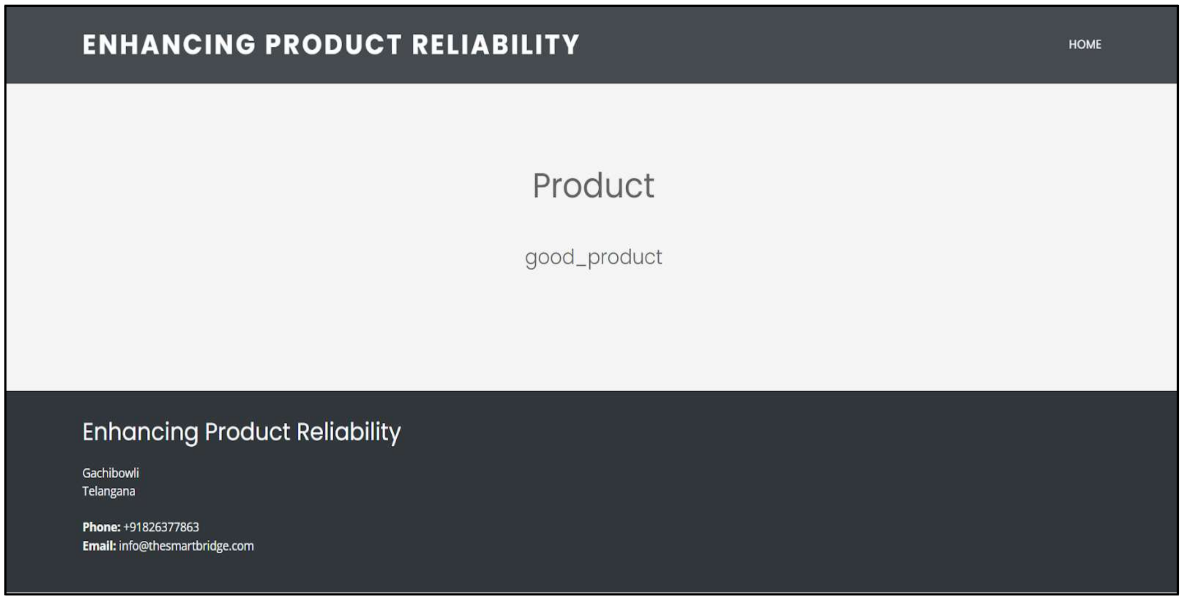
Predicting Product Reliability

Choose... Choose File cast_ok_0_10.jpeg

Predict

Once you upload the image and click on the predict button, the output will be displayed

Output:1



ENHANCING PRODUCT RELIABILITY

HOME

Product

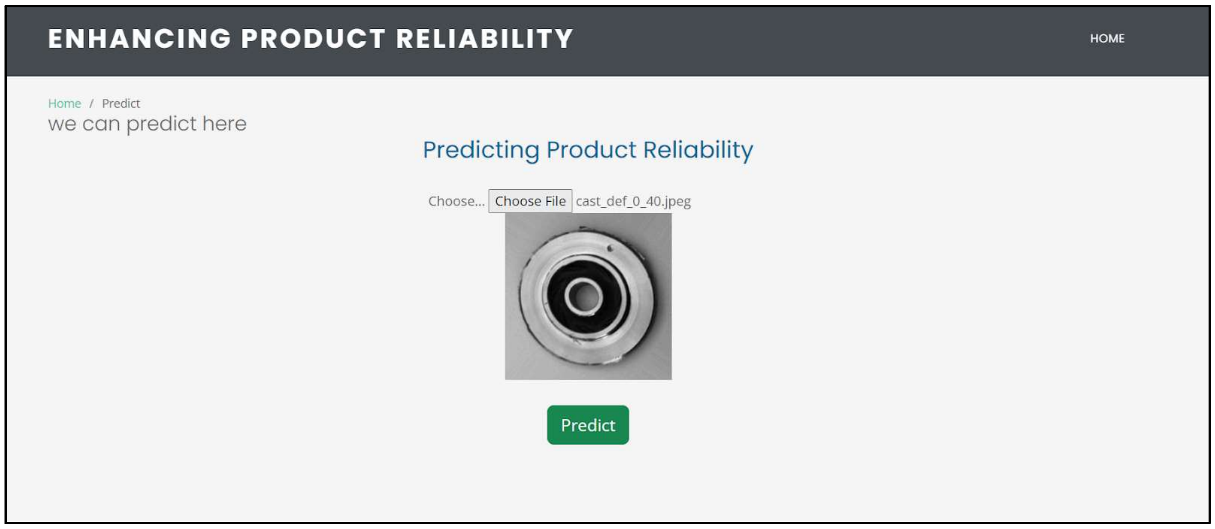
good_product

Enhancing Product Reliability

Gachibowli
Telangana

Phone: +91826377863
Email: info@thesmartbridge.com

INPUT 2:



ENHANCING PRODUCT RELIABILITY

HOME

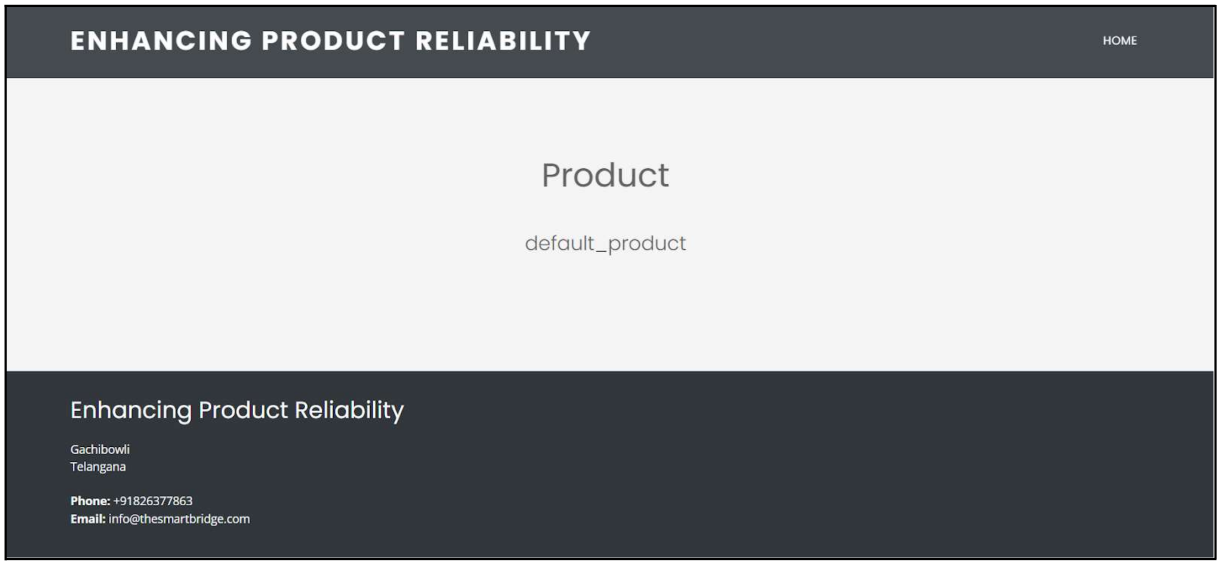
Home / Predict
we can predict here

Predicting Product Reliability

Once you upload the image and click on the predict button, the output will be displayed

Output-2:

Once you upload the image and click on the prediction



Product

default_product

Enhancing Product Reliability

Gachibowli
Telangana

Phone: +91826377863
Email: info@thesmartbridge.com