T.E. Mini Project

Quality Control using Machine Learning

Bachelor of Engineering in Mechanical Engineering

By

Raorane Paras	Roll no. 52
Rathod Bhagyesh	Roll no. 53
Raut Shubham	Roll no. 54
Sankpal Sahil	Roll no. 56

Guided by

Prof.(Dr.) Usha C. Pawar

Department of Mechanical Engineering

Datta Meghe College of Engineering

Plot no. 98, Sector 3, Airoli, Navi Mumbai, 400708

2020-21

Datta Meghe College of Engineering

(AICTE & Govt. of Maharashtra Recognized, Affiliated to University of Mumbai)
Sector-3, Airoli, Navi Mumbai – 400 708, (M.S.), INDIA

Department of Mechanical Engineering
T.E. Mini Project

Name of the Candidate	: Raorane Paras
	: Rathod Bhagyesh
	: Raut Shubham
	: Sankpal Sahil
Title of the Project	: "Quality Control using Machine Learning"
Year of Admission	: 2021-22
Name of the Guide	: Prof. (Dr.) Usha C. Pawar

Date of submission of Outline : / / 2022

1.________________2.________________3.________________4.________________
Signature of Candidates

Signature (Guiding Teacher)

Signature (Head of Department)

Content

Abstract

Machine Learning, since the last decade, has been tried and tested to solve a number of complex problems across vast spectrum of fields. Since the gradual rise of computation capabilities due to vast strides in semi-conductor development, Computers have become more powerful than ever. As future Mechanical Engineers, it becomes a responsibility to ensure usage of such ground breaking discoveries in this field to improve existing instruments and methodologies for better efficiency in production.

Different cutting-edge technologies like Machine Learning, Deep Learning, Computer Vision can be of great help in improving the quality of current manufacturing processes. The integration of these new technologies with the existing technologies will possibly be the building blocks for 5th industrial revolution. With the crunch in availability of resources, improvement in manufacturing efficiency/cost becomes the need of hour. The project aims to utilize a subset of Machine Learning called as "Deep Learning" to enhance the process of Quality Control after production.

Identification of need/problem

The manufacturing of mechanical products is a complex industrial process, defects such as internal holes, pits, abrasions, and scratches arise, due to failure in design and machine production equipment as well as unfavorable working conditions. Often it is found that a part or any material manufactured is damaged or has irregular design. Also, manual detection of these parts is very difficult and time consuming. So, this parts or product further when used in machines, may damage and reduce the life of it.

As a part of this project, we are solving a specific problem-Detection of defects during manufacturing of a medicine box. Currently in medium scale manufacturing industries, sensors are used for the detection of these faults and further they are also manually checked by workers in case the sensor fails to detect any damaged part. However, the sensors are quite costly while the manual labor singlehandedly isn’t reliable. The sensors can be replaced by the technology of image capturing devices equipped with specific Machine Learning algorithms which replaces sensors with cameras thereby reducing the cost at almost equal accuracy of the detection of defective manufactured products. This will not only speed up the process but also this will help with the requirement of manual labor.

With this demonstration, the solution to the problem can be scaled sufficiently using the same model. Just replacing the training image data with the images of the required detectable component with correct classification will be enough. Usage of more advanced algorithms on devices with better computational capacity can improve the time requirements for training and classification.

Introduction

"Quality Control using Machine Learning" is our attempt to utilize ground breaking discoveries in computer technology to improve methods of Quality Control in manufacturing industry. Machine Learning can be used to enhance/optimize the methods used for production in the manufacturing industries in present. The integration of Robots which was one of the building blocks of the 3rd Industrial Revolution will now pave a way for the advent of the 4th Industrial Revolution through techniques like Machine Learning, Deep Learning, Computer Vision.

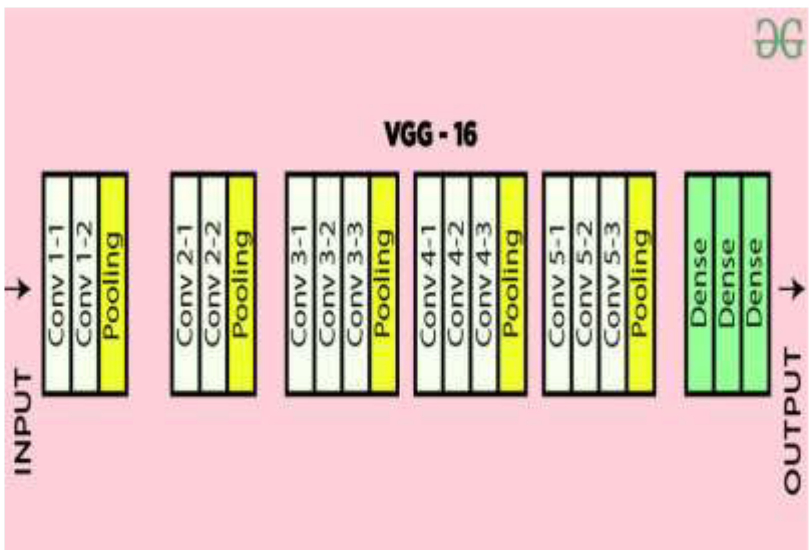
This project is an attempt to use Machine Learning/Computer Vision to improve the efficiency of quality control after production. It tries to provide an alternative to the traditional expensive methodology involving sensors to detect imperfections with image capturing devices equipped with a powerful algorithm connected to the internet.

The project makes use of the concept of "Transfer Learning". It utilizes a VGG-16 model with last 3 fully connected layers added with additional 2 fully connected Dense layers with a Drop Out layer. Trained on an image dataset of 400 images of a medicine box artificially generated using Blender to emulate production line imagery. Training is divided into 3 stages of 200 epochs each. The model gives the probability for classification to each class and on the basis of that we classify the images as "Intact" or "Damaged".

Literature Survey

VGG-16 CNN[1,2]

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks. The first is to detect objects within an image coming from 200 classes, which is called object localization. The second is to classify images, each labeled with one of 1000 categories, which is called image classification. VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University.



This model achieves 92.7% top-5 test accuracy on ImageNet dataset which contains 14 million images. The objectives of the model is the ImageNet dataset contains images of fixed size of 224*224 and have RGB channels. This model process the input image and outputs the a vector of 1000 values. The architecture of the model is as follows the input to the network is image of dimensions (224, 224, 3). The first two layers have 64 channels of 3*3 filter size and same padding. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problem. The configuration of the model is the table below listed different VGG architecture. We can see that there are 2 versions of VGG-16 (C and D). There is not much difference between them except for one that except for some convolution layer there is (3, 3) filter size convolution is used instead of (1, 1). These two contains 134 million and 138 million parameters respectively. Object Localization In Image is to perform localization, we need to replace the class score by bounding box location candidates. A bounding box location is represented by 4-D vector.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

There are two version of localization architecture, one is bounding box is shared among different candidates and other is bounding box is class specific. The paper experimented with both approach on VGG -16 (D) architecture. Here we also need to change loss from classification loss to regression loss functions (such as MSE) that penalize the deviation of predicted loss from ground truth. The Result is VGG-16 was one of the best performing architecture in ILSVRC challenge 2014.It was the runner up in classification task with top-5 classification error of 7.32% (only behind GoogLeNet with classification error 6.66%). It was also the winner of localization task with 25.32% localization error. The size of VGG-16 trained imageNet weights is 528 MB. So, it takes quite a lot of disk space and bandwidth that makes it inefficient.

Predictive model-based quality inspection using Machine Learning and Edge Cloud Computing[3]

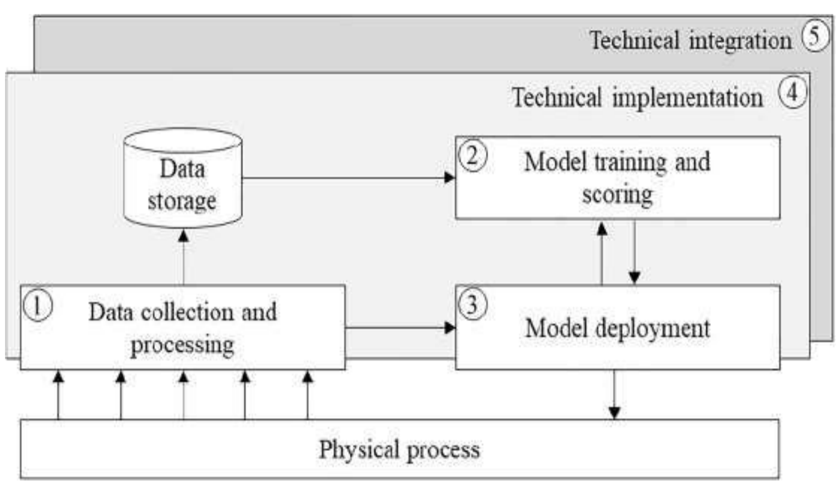
The supply of defect-free, high-quality products is an important success factor for the long-term competitiveness of manufacturing companies. Despite the increasing challenges of rising product variety and complexity and the necessity of economic manufacturing, a comprehensive and reliable quality inspection is often indispensable. InIn this contribution, we investigate a new integrated solution of predictive model-based quality inspection in industrial manufacturing by utilizing Machine Learning techniques and Edge Cloud Computing technology. The results show that by employing the proposed method, inspection volumes can be reduced significantly and thus economic advantages can be generated. As a result of increasing competitive pressure, the supply of high-quality products continues to evolve as an important competitive factor to secure the long-term success of a company. In order to guarantee the delivery and transfer of zero-defect products, it is essential to ensure a constantly high quality for all products.

In recent years, ML has provided advantages in various fields of application, where the success can be credited to the invention of more sophisticated ML models , the availability of large data sets, and the development of software platforms that allow easy employment of vast computational resources for training ML models on large data sets.ML is a subfield of Artificial Intelligence that enables information technology (IT) systems to recognize patterns and laws on the basis of existing data and algorithms and develop solutions autonomously. The knowledge gained from data can then be generalized and used to solve new problems and analyze previously unknown data. They can be categorized according to different learning paradigms into:

•supervised learning,
•unsupervised learning,
•semi-supervised learning,
•reinforcement learning, and
•active learning.

The product quality is essential for the long-term success of a producing company and the economic realization of a comprehensive, reliable quality inspection is therefore of great interest. Manufacturing metrology, conventionally used in quality control, progressively reaches its limits due to the increasing requirements for speed, accuracy, safety, and flexibility.
While the recent state of research contains some literature reviews on general applications of ML in manufacturing, e.g.specific reviews with focus on quality-related applications are rarely found.
Different quality tasks for the application of ML in manufacturing can be distinguished:
▪Description of product/process quality,
▪Classification of quality,
▪Quality prediction, and
▪Parameter optimization

To facilitate the collection, processing, and analyzing of recorded process data, training and deployment of predictive models, as well as their technical implementation and integration, the proposed framework consists of four main elements they are data collection and processing, model training and scoring, model deployment, and technical implementation.
The layout of the proposed framework is shown in below



A Literature Survey on Computer Vision Towards Data Science[4]

Computer Vision is the discipline under a broad area of Artificial Intelligence which teaches machines to see. From the engineering point of view, computer vision aims to build autonomous systems which could perform some of the tasks which the human visual system can perform. We can see examples of such applications are- Amazon Go, Google Lens, Autonomous Vehicles, and Face Recognition. The goal of computer vision is to understand the content of digital images. Typically, this involves developing methods that attempt to reproduce the capability of human vision. The purpose of this paper to brief about the technique behind the computer vision using Data.

Computer vision is an inter disciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. "Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of theoretical and algorithmic basis to achieve automatic visual understanding." As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

Computer vision can be described as finding and telling features from images to help discriminate objects and/or classes of objects. Computer vision was mainly based with image processing algorithms and methods. The main process of computer vision was extracting the features of the image. Detecting the color, edges, corners and objects were the first step to do when performing a computer vision task. These features are human engineered and accuracy and the reliability of the models directly depend on the extracted features and on the methods used for feature extraction. In the traditional vision scope, the algorithms like SIFT (Scale-

Invariant Feature Transform), SURF (Speeded-Up Robust Features), BRIEF (Binary Robust Independent Elementary Features) plays the major role of extracting the features from the raw image.

Deep learning, which is a subset of machine learning has shown a significant performance and accuracy gain in the field of computer vision. Arguably one of the most influential papers in applying deep learning to computer vision, in 2012, a neural network containing over 60 million parameters significantly beat previous state-of-the-art approaches to image recognition in a popular Image Net computer vision competition. The boom started with the convolutional neural networks and the modified architectures of ConvNets.

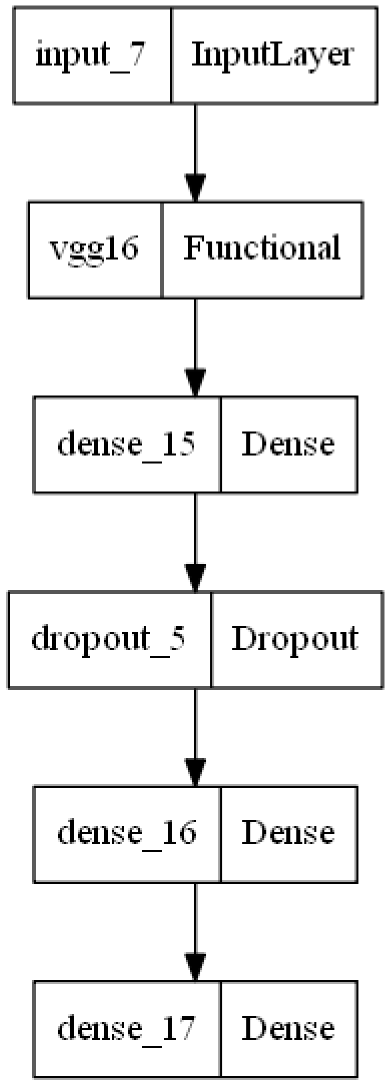
Objectives

The main aim of the project is to demonstrate the advantages, usage feasibility of Machine Learning in Mechanical Engineering. The project is aimed at enhancing the task of Quality Control through usage of alternative ground-breaking technologies.

- To demonstrate usage of Machine Learning in Quality Control.
- To demonstrate usage of Computer Vision in image classification tasks.
- To demonstrate usage of the VGG16 model outside it's intended scope.
- To classify a component using a machine learning model trained on images captured from different angles.
- Provide a cost-efficient alternative to traditional Quality Control methods.
- Improve the quality of products through scalable alternatives to traditional technologies.

Methodology

The project utilizes a pretrained VGG-16 model with the last three layers being trainable connected with a fully connected dense layer of 500 units followed by a drop-out layer (to counter overfitting) followed again by two dense layers to predict the output.



There are about 138 million parameters in the whole model out of which only 510 thousand are trainable. Increasing the number of parameters can improve the output at the cost of a worse training time and requirement of better computing hardware, which comes with a risk of terrible overfitting the dataset.

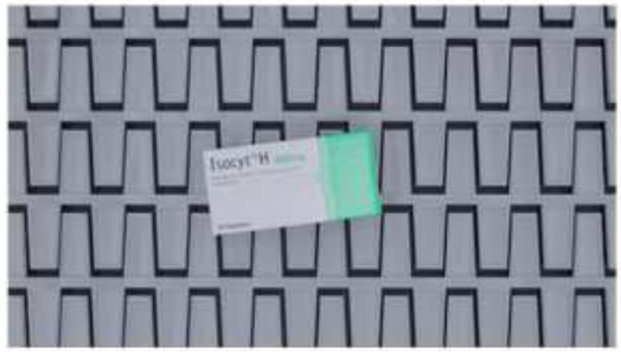
```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 1000)              138357544

 dense_15 (Dense)            (None, 500)               500500

 dropout_5 (Dropout)         (None, 500)               0

 dense_16 (Dense)            (None, 20)                10020

 dense_17 (Dense)            (None, 1)                 21

=================================================================
Total params: 138,868,085
Trainable params: 510,541
Non-trainable params: 138,357,544
_________________________________________________________________
```

Dataset[5] consists of 400 computer generated[8] images of a medicine box, on which the model will train and classify as “Intact” or “Damaged”.

About Dataset

Summary:

This dataset consists of synthetically created images of packages produced in an industrial production line in which the packaging machine occasionally produces faulty packages. One of the goals for the data scientist is to create a model that can identify damaged packages.



Details:

Many companies in the manufacturing industry use packaging machines to wrap their products. These machines usually work fully automatically, but from time to time, faulty packages are produced, for example because small deviations in the position of the packages cause them to be dented or bent. Therefore, in-line quality control inspection points exist across the production line in which different quality measures are taken that should ensure that faulty packages are excluded from the process before they are sent out.

This dataset consists of RGB images taken from a "virtual" production line (meaning, the images were created procedurally, see below) with two classes: damaged and intact. For each class and each package, two camera captures exist: One image taken from a camera mounted above the package, one picture taken from a camera mounted on the side of the inspection belt. Each package is identified by a unique serial number (SN) visible on the side-view of the package. This serial number is also used for naming the files ({sn}*side.png and (sn)*top.png).

The goal of this dataset is to give data scientists the possibility to work on data inspired by industrial manufacturing scenarios. Among others the following questions are interesting:

- Can you train a model that identifies damaged packages using both, top and side view?
- Can you train a model that identifies damaged packages using just the top view?
- Can you train a model that performs OCR to extract the serial numbers (some SN may be blurry due to simulated motion blur)
- Can you create a one-shot computer vision algorithms that can classify packages by just learning from one or two samples?

As mentioned above, the images were created synthetically using Blender. The code to produce these images is open-sourced and can be found at https://github.com/christian-vorhemus/procedural-3d-image-generation

The code for the method followed is attached next.

Code

In [1]:

```
#import packages
import pandas as pd,numpy as np
import numpy as np
import pandas as pd
import re
import os
import cv2
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.layers import Dense,Input,Activation,Conv1D,Concatenate
from tensorflow.keras.models import Model
from tensorflow.python.client import device_lib
import datetime
import keras
%load_ext tensorboard

from keras_preprocessing.image import ImageDataGenerator
```

Data Handling

In [2]:

```
temp=os.listdir("damaged")
filename=[]
for i in temp:
    filename.extend(["damaged/"+i+"/"+k for k in os.listdir("damaged/"+i)])

data1=pd.DataFrame(filename)
data1["target"]=[0 for i in range(len(filename))]

temp=os.listdir("intact")
filename=[]
for i in temp:
    filename.extend(["intact/"+i+"/"+k for k in os.listdir("intact/"+i)])

data2=pd.DataFrame(filename)
data2["target"]=[1 for i in range(len(filename))]

data=pd.concat([data1,data2],ignore_index=True)
data.columns=["image","target"]
```

Data Generator

In [3]:

```
ImageFlow = ImageDataGenerator()
TrainGenerator=ImageFlow.flow_from_dataframe(dataframe=data, x_col="image", y_col="target
",shuffle=True,\
                                             class_mode="raw", target_size=(224,224), b
atch_size=10)#,\

z=os.listdir("test_images")

test_data=pd.DataFrame()
test_data["image"]=["test_images/"+i for i in z]
test_data["target"]=[1 for i in z]

TestImageGenerator=ImageFlow.flow_from_dataframe(dataframe=test_data, x_col="image", y_co
```

```
l="target",shuffle=False,\
                                                            class_mode="raw", target_size=(224,224), b
atch_size=1)
```

```
Found 400 validated image filenames.
Found 265 validated image filenames.
```

Pretrained Model

In [12]:

```
basemodel=tf.keras.applications.vgg16.VGG16(
    include_top=True, weights='imagenet',input_shape=(224,224,3))

basemodel.trainable = False
for i in range(0,3):
    basemodel.layers[-i].trainable=True
```

Defining Model Architecture

In [48]:

```
input_layer = tf.keras.Input(shape=(224,224, 3))

layer1=basemodel(input_layer)

layer4=tf.keras.layers.Dense(
    500,
    activation='relu',
    kernel_initializer=tf.keras.initializers.he_uniform(),
)(layer1)

drop=tf.keras.layers.Dropout(0.3)(layer4)
layer5=tf.keras.layers.Dense(
    20,
    activation='relu',
    kernel_initializer=tf.keras.initializers.he_uniform(),
)(drop)

output= Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.he_uniform
())(layer5)

model=Model(input_layer,outputs=output)

optimizer = tf.keras.optimizers.Adam()

model.compile(optimizer=optimizer, loss="BinaryCrossentropy",metrics=['Precision','accur
acy'])
```

In [49]:

```
model.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 1000)              138357544

 dense_15 (Dense)            (None, 500)               500500

 dropout_5 (Dropout)         (None, 500)               0

 dense_16 (Dense)            (None, 20)                10020

 dense_17 (Dense)            (None, 1)                 21
```

```
=================================================================
Total params: 138,868,085
Trainable params: 510,541
Non-trainable params: 138,357,544
_________________________________________________________________
```

Model Training (stage-3)

In [57]:

```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)

model.fit(TrainGenerator,batch_size=10,epochs=200,callbacks=tensorboard_callback)
```

```
Epoch 1/200
40/40 [==============================] - 19s 447ms/step - loss: 0.2140 - precision: 0.906
9 - accuracy: 0.9150
Epoch 2/200
40/40 [==============================] - 18s 447ms/step - loss: 0.2397 - precision: 0.873
8 - accuracy: 0.8850
Epoch 3/200
40/40 [==============================] - 17s 431ms/step - loss: 0.2047 - precision: 0.915
0 - accuracy: 0.9150
Epoch 4/200
40/40 [==============================] - 18s 437ms/step - loss: 0.2085 - precision: 0.898
6 - accuracy: 0.9125
Epoch 5/200
40/40 [==============================] - 17s 424ms/step - loss: 0.2084 - precision: 0.900
5 - accuracy: 0.9025
Epoch 6/200
40/40 [==============================] - 17s 429ms/step - loss: 0.2268 - precision: 0.893
2 - accuracy: 0.9050
Epoch 7/200
40/40 [==============================] - 18s 437ms/step - loss: 0.2232 - precision: 0.903
6 - accuracy: 0.8975
Epoch 8/200
40/40 [==============================] - 17s 430ms/step - loss: 0.2014 - precision: 0.879
1 - accuracy: 0.9075
Epoch 9/200
40/40 [==============================] - 17s 427ms/step - loss: 0.1896 - precision: 0.908
7 - accuracy: 0.9250
Epoch 10/200
40/40 [==============================] - 18s 443ms/step - loss: 0.2016 - precision: 0.919
2 - accuracy: 0.9150
Epoch 11/200
40/40 [==============================] - 17s 424ms/step - loss: 0.2142 - precision: 0.884
1 - accuracy: 0.8975
Epoch 12/200
40/40 [==============================] - 17s 425ms/step - loss: 0.2135 - precision: 0.893
2 - accuracy: 0.9050
Epoch 13/200
40/40 [==============================] - 17s 428ms/step - loss: 0.2153 - precision: 0.901
5 - accuracy: 0.9075
Epoch 14/200
40/40 [==============================] - 16s 392ms/step - loss: 0.2034 - precision: 0.890
0 - accuracy: 0.9075
Epoch 15/200
40/40 [==============================] - 17s 416ms/step - loss: 0.2227 - precision: 0.889
4 - accuracy: 0.9050
Epoch 16/200
40/40 [==============================] - 17s 414ms/step - loss: 0.2305 - precision: 0.873
8 - accuracy: 0.8850
Epoch 17/200
40/40 [==============================] - 17s 421ms/step - loss: 0.1922 - precision: 0.915
4 - accuracy: 0.9175
Epoch 18/200
40/40 [==============================] - 18s 461ms/step - loss: 0.2493 - precision: 0.887
3 - accuracy: 0.8950
Epoch 19/200
```

```
Epoch 19/200
40/40 [==============================] - 19s 472ms/step - loss: 0.2365 - precision: 0.888
3 - accuracy: 0.9000
Epoch 20/200
40/40 [==============================] - 10s 246ms/step - loss: 0.2312 - precision: 0.881
2 - accuracy: 0.8850
Epoch 21/200
40/40 [==============================] - 6s 145ms/step - loss: 0.2309 - precision: 0.8841
- accuracy: 0.8975
Epoch 22/200
40/40 [==============================] - 6s 145ms/step - loss: 0.2371 - precision: 0.8660
- accuracy: 0.8825
Epoch 23/200
40/40 [==============================] - 6s 145ms/step - loss: 0.2235 - precision: 0.8685
- accuracy: 0.8925
Epoch 24/200
40/40 [==============================] - 6s 146ms/step - loss: 0.2083 - precision: 0.9059
- accuracy: 0.9100
Epoch 25/200
40/40 [==============================] - 6s 147ms/step - loss: 0.2364 - precision: 0.8775
- accuracy: 0.8850
Epoch 26/200
40/40 [==============================] - 6s 148ms/step - loss: 0.2165 - precision: 0.8850
- accuracy: 0.8850
Epoch 27/200
40/40 [==============================] - 6s 151ms/step - loss: 0.2059 - precision: 0.8981
- accuracy: 0.9100
Epoch 28/200
40/40 [==============================] - 6s 150ms/step - loss: 0.2166 - precision: 0.8835
- accuracy: 0.8950
Epoch 29/200
40/40 [==============================] - 6s 152ms/step - loss: 0.2050 - precision: 0.8966
- accuracy: 0.9025
Epoch 30/200
40/40 [==============================] - 7s 160ms/step - loss: 0.1989 - precision: 0.8883
- accuracy: 0.9000
Epoch 31/200
40/40 [==============================] - 7s 181ms/step - loss: 0.1997 - precision: 0.9118
- accuracy: 0.9200
Epoch 32/200
40/40 [==============================] - 8s 189ms/step - loss: 0.2027 - precision: 0.9010
- accuracy: 0.9050
Epoch 33/200
40/40 [==============================] - 8s 193ms/step - loss: 0.2526 - precision: 0.8599
- accuracy: 0.8725
Epoch 34/200
40/40 [==============================] - 7s 172ms/step - loss: 0.2065 - precision: 0.8883
- accuracy: 0.9000
Epoch 35/200
40/40 [==============================] - 7s 167ms/step - loss: 0.1873 - precision: 0.9043
- accuracy: 0.9225
Epoch 36/200
40/40 [==============================] - 7s 169ms/step - loss: 0.2080 - precision: 0.8762
- accuracy: 0.8950
Epoch 37/200
40/40 [==============================] - 7s 165ms/step - loss: 0.2052 - precision: 0.9158
- accuracy: 0.9200
Epoch 38/200
40/40 [==============================] - 7s 162ms/step - loss: 0.2062 - precision: 0.9055
- accuracy: 0.9075
Epoch 39/200
40/40 [==============================] - 7s 162ms/step - loss: 0.1792 - precision: 0.9208
- accuracy: 0.9250
Epoch 40/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1880 - precision: 0.9118
- accuracy: 0.9200
Epoch 41/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1872 - precision: 0.9126
- accuracy: 0.9250
Epoch 42/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1827 - precision: 0.9216
- accuracy: 0.9300
Epoch 43/200
```

```
Epoch 43/200
40/40 [==============================] - 7s 171ms/step - loss: 0.2019 - precision: 0.8986
- accuracy: 0.9125
Epoch 44/200
40/40 [==============================] - 7s 174ms/step - loss: 0.2077 - precision: 0.9005
- accuracy: 0.9025
Epoch 45/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1790 - precision: 0.9293
- accuracy: 0.9250
Epoch 46/200
40/40 [==============================] - 7s 166ms/step - loss: 0.1877 - precision: 0.9171
- accuracy: 0.9275
Epoch 47/200
40/40 [==============================] - 7s 166ms/step - loss: 0.1964 - precision: 0.8883
- accuracy: 0.9000
Epoch 48/200
40/40 [==============================] - 8s 200ms/step - loss: 0.1924 - precision: 0.9126
- accuracy: 0.9250
Epoch 49/200
40/40 [==============================] - 8s 202ms/step - loss: 0.2096 - precision: 0.9055
- accuracy: 0.9075
Epoch 50/200
40/40 [==============================] - 10s 248ms/step - loss: 0.2083 - precision: 0.906
4 - accuracy: 0.9125
Epoch 51/200
40/40 [==============================] - 7s 182ms/step - loss: 0.2195 - precision: 0.8986
- accuracy: 0.9125
Epoch 52/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1886 - precision: 0.9109
- accuracy: 0.9150
Epoch 53/200
40/40 [==============================] - 8s 189ms/step - loss: 0.2125 - precision: 0.8632
- accuracy: 0.8850
Epoch 54/200
40/40 [==============================] - 7s 180ms/step - loss: 0.1597 - precision: 0.9406
- accuracy: 0.9450
Epoch 55/200
40/40 [==============================] - 8s 186ms/step - loss: 0.1719 - precision: 0.9250
- accuracy: 0.9250
Epoch 56/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1856 - precision: 0.9064
- accuracy: 0.9125
Epoch 57/200
40/40 [==============================] - 8s 184ms/step - loss: 0.1771 - precision: 0.9118
- accuracy: 0.9200
Epoch 58/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1780 - precision: 0.9163
- accuracy: 0.9225
Epoch 59/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1824 - precision: 0.9078
- accuracy: 0.9200
Epoch 60/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1869 - precision: 0.9038
- accuracy: 0.9200
Epoch 61/200
40/40 [==============================] - 7s 179ms/step - loss: 0.2115 - precision: 0.8932
- accuracy: 0.9050
Epoch 62/200
40/40 [==============================] - 7s 178ms/step - loss: 0.2323 - precision: 0.8911
- accuracy: 0.8950
Epoch 63/200
40/40 [==============================] - 7s 178ms/step - loss: 0.2048 - precision: 0.8960
- accuracy: 0.9000
Epoch 64/200
40/40 [==============================] - 7s 172ms/step - loss: 0.2233 - precision: 0.9064
- accuracy: 0.9125
Epoch 65/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1997 - precision: 0.8947
- accuracy: 0.9125
Epoch 66/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1942 - precision: 0.9024
- accuracy: 0.9125
Epoch 67/200
```

```
Epoch 67/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1861 - precision: 0.8981
- accuracy: 0.9100
Epoch 68/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1804 - precision: 0.9208
- accuracy: 0.9250
Epoch 69/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1863 - precision: 0.8971
- accuracy: 0.9050
Epoch 70/200
40/40 [==============================] - 7s 174ms/step - loss: 0.2001 - precision: 0.8957
- accuracy: 0.9175
Epoch 71/200
40/40 [==============================] - 7s 180ms/step - loss: 0.1849 - precision: 0.9171
- accuracy: 0.9275
Epoch 72/200
40/40 [==============================] - 7s 173ms/step - loss: 0.2066 - precision: 0.8981
- accuracy: 0.9100
Epoch 73/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1853 - precision: 0.9034
- accuracy: 0.9175
Epoch 74/200
40/40 [==============================] - 7s 166ms/step - loss: 0.1942 - precision: 0.9104
- accuracy: 0.9125
Epoch 75/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1986 - precision: 0.8868
- accuracy: 0.9100
Epoch 76/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1761 - precision: 0.9303
- accuracy: 0.9325
Epoch 77/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1725 - precision: 0.9118
- accuracy: 0.9200
Epoch 78/200
40/40 [==============================] - 8s 186ms/step - loss: 0.1735 - precision: 0.9135
- accuracy: 0.9300
Epoch 79/200
40/40 [==============================] - 8s 203ms/step - loss: 0.1766 - precision: 0.9171
- accuracy: 0.9275
Epoch 80/200
40/40 [==============================] - 8s 186ms/step - loss: 0.1828 - precision: 0.9130
- accuracy: 0.9275
Epoch 81/200
40/40 [==============================] - 7s 180ms/step - loss: 0.2129 - precision: 0.9015
- accuracy: 0.9075
Epoch 82/200
40/40 [==============================] - 8s 190ms/step - loss: 0.1866 - precision: 0.8976
- accuracy: 0.9075
Epoch 83/200
40/40 [==============================] - 7s 181ms/step - loss: 0.1881 - precision: 0.8990
- accuracy: 0.9150
Epoch 84/200
40/40 [==============================] - 7s 167ms/step - loss: 0.1740 - precision: 0.9175
- accuracy: 0.9300
Epoch 85/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1868 - precision: 0.9034
- accuracy: 0.9175
Epoch 86/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1611 - precision: 0.9183
- accuracy: 0.9350
Epoch 87/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1839 - precision: 0.9020
- accuracy: 0.9100
Epoch 88/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1603 - precision: 0.9261
- accuracy: 0.9325
Epoch 89/200
40/40 [==============================] - 7s 183ms/step - loss: 0.1978 - precision: 0.8976
- accuracy: 0.9075
Epoch 90/200
40/40 [==============================] - 8s 193ms/step - loss: 0.1678 - precision: 0.9310
- accuracy: 0.9375
Epoch 91/200
```

```
Epoch 91/200
40/40 [==============================] - 9s 218ms/step - loss: 0.1781 - precision: 0.8962
- accuracy: 0.9200
Epoch 92/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1934 - precision: 0.9078
- accuracy: 0.9200
Epoch 93/200
40/40 [==============================] - 8s 187ms/step - loss: 0.1971 - precision: 0.9055
- accuracy: 0.9075
Epoch 94/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1719 - precision: 0.9171
- accuracy: 0.9275
Epoch 95/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1667 - precision: 0.9220
- accuracy: 0.9325
Epoch 96/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1857 - precision: 0.8905
- accuracy: 0.9100
Epoch 97/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1892 - precision: 0.8952
- accuracy: 0.9150
Epoch 98/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1776 - precision: 0.8990
- accuracy: 0.9150
Epoch 99/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1675 - precision: 0.9118
- accuracy: 0.9200
Epoch 100/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1695 - precision: 0.9034
- accuracy: 0.9175
Epoch 101/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1572 - precision: 0.9257
- accuracy: 0.9300
Epoch 102/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1981 - precision: 0.9020
- accuracy: 0.9100
Epoch 103/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1818 - precision: 0.9208
- accuracy: 0.9250
Epoch 104/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1941 - precision: 0.9150
- accuracy: 0.9150
Epoch 105/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1705 - precision: 0.9091
- accuracy: 0.9275
Epoch 106/200
40/40 [==============================] - 7s 168ms/step - loss: 0.1797 - precision: 0.9082
- accuracy: 0.9225
Epoch 107/200
40/40 [==============================] - 7s 179ms/step - loss: 0.1737 - precision: 0.9052
- accuracy: 0.9275
Epoch 108/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1996 - precision: 0.9091
- accuracy: 0.9050
Epoch 109/200
40/40 [==============================] - 7s 172ms/step - loss: 0.2002 - precision: 0.9113
- accuracy: 0.9175
Epoch 110/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1945 - precision: 0.9158
- accuracy: 0.9200
Epoch 111/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1861 - precision: 0.8976
- accuracy: 0.9075
Epoch 112/200
40/40 [==============================] - 7s 172ms/step - loss: 0.2106 - precision: 0.8846
- accuracy: 0.9000
Epoch 113/200
40/40 [==============================] - 7s 174ms/step - loss: 0.2148 - precision: 0.8857
- accuracy: 0.9050
Epoch 114/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1984 - precision: 0.8942
- accuracy: 0.9100
Epoch 115/200
```

```
Epoch 115/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1738 - precision: 0.9073
- accuracy: 0.9175
Epoch 116/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1643 - precision: 0.9163
- accuracy: 0.9225
Epoch 117/200
40/40 [==============================] - 7s 181ms/step - loss: 0.1628 - precision: 0.9095
- accuracy: 0.9300
Epoch 118/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1756 - precision: 0.9212
- accuracy: 0.9275
Epoch 119/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1525 - precision: 0.9353
- accuracy: 0.9375
Epoch 120/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1812 - precision: 0.8995
- accuracy: 0.9175
Epoch 121/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1620 - precision: 0.9216
- accuracy: 0.9300
Epoch 122/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1738 - precision: 0.9158
- accuracy: 0.9200
Epoch 123/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1925 - precision: 0.9043
- accuracy: 0.9225
Epoch 124/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1611 - precision: 0.9087
- accuracy: 0.9250
Epoch 125/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1933 - precision: 0.9087
- accuracy: 0.9250
Epoch 126/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1594 - precision: 0.9303
- accuracy: 0.9325
Epoch 127/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1485 - precision: 0.9216
- accuracy: 0.9300
Epoch 128/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1547 - precision: 0.9187
- accuracy: 0.9375
Epoch 129/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1578 - precision: 0.9179
- accuracy: 0.9325
Epoch 130/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1774 - precision: 0.9048
- accuracy: 0.9250
Epoch 131/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1571 - precision: 0.9212
- accuracy: 0.9275
Epoch 132/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1625 - precision: 0.9272
- accuracy: 0.9400
Epoch 133/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1911 - precision: 0.9135
- accuracy: 0.9300
Epoch 134/200
40/40 [==============================] - 7s 170ms/step - loss: 0.2117 - precision: 0.8867
- accuracy: 0.8925
Epoch 135/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1899 - precision: 0.9200
- accuracy: 0.9200
Epoch 136/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1566 - precision: 0.9220
- accuracy: 0.9325
Epoch 137/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1793 - precision: 0.9126
- accuracy: 0.9250
Epoch 138/200
40/40 [==============================] - 7s 181ms/step - loss: 0.1952 - precision: 0.9034
- accuracy: 0.9175
Epoch 139/200
```

```
Epoch 139/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1767 - precision: 0.9183
- accuracy: 0.9350
Epoch 140/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1605 - precision: 0.9300
- accuracy: 0.9300
Epoch 141/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1636 - precision: 0.9212
- accuracy: 0.9275
Epoch 142/200
40/40 [==============================] - 7s 169ms/step - loss: 0.1715 - precision: 0.9073
- accuracy: 0.9175
Epoch 143/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1698 - precision: 0.9223
- accuracy: 0.9350
Epoch 144/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1586 - precision: 0.8962
- accuracy: 0.9200
Epoch 145/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1712 - precision: 0.9261
- accuracy: 0.9325
Epoch 146/200
40/40 [==============================] - 7s 170ms/step - loss: 0.1690 - precision: 0.9220
- accuracy: 0.9325
Epoch 147/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1696 - precision: 0.9109
- accuracy: 0.9150
Epoch 148/200
40/40 [==============================] - 7s 172ms/step - loss: 0.1494 - precision: 0.9143
- accuracy: 0.9350
Epoch 149/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1645 - precision: 0.9130
- accuracy: 0.9275
Epoch 150/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1592 - precision: 0.9122
- accuracy: 0.9225
Epoch 151/200
40/40 [==============================] - 7s 182ms/step - loss: 0.1785 - precision: 0.9158
- accuracy: 0.9200
Epoch 152/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1866 - precision: 0.8905
- accuracy: 0.9100
Epoch 153/200
40/40 [==============================] - 7s 173ms/step - loss: 0.1834 - precision: 0.9109
- accuracy: 0.9150
Epoch 154/200
40/40 [==============================] - 7s 171ms/step - loss: 0.1527 - precision: 0.9265
- accuracy: 0.9350
Epoch 155/200
40/40 [==============================] - 7s 183ms/step - loss: 0.1722 - precision: 0.8957
- accuracy: 0.9175
Epoch 156/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1724 - precision: 0.9254
- accuracy: 0.9275
Epoch 157/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1687 - precision: 0.9126
- accuracy: 0.9250
Epoch 158/200
40/40 [==============================] - 8s 194ms/step - loss: 0.1538 - precision: 0.9356
- accuracy: 0.9400
Epoch 159/200
40/40 [==============================] - 10s 236ms/step - loss: 0.1657 - precision: 0.912
6 - accuracy: 0.9250
Epoch 160/200
40/40 [==============================] - 8s 199ms/step - loss: 0.1474 - precision: 0.9087
- accuracy: 0.9250
Epoch 161/200
40/40 [==============================] - 9s 228ms/step - loss: 0.1536 - precision: 0.9100
- accuracy: 0.9325
Epoch 162/200
40/40 [==============================] - 8s 203ms/step - loss: 0.1431 - precision: 0.9310
- accuracy: 0.9375
Epoch 163/200
```

```
Epoch 163/200
40/40 [==============================] - 8s 189ms/step - loss: 0.1497 - precision: 0.9208
- accuracy: 0.9250
Epoch 164/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1480 - precision: 0.9320
- accuracy: 0.9450
Epoch 165/200
40/40 [==============================] - 8s 188ms/step - loss: 0.1493 - precision: 0.9087
- accuracy: 0.9250
Epoch 166/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1716 - precision: 0.9261
- accuracy: 0.9325
Epoch 167/200
40/40 [==============================] - 8s 194ms/step - loss: 0.1485 - precision: 0.9231
- accuracy: 0.9400
Epoch 168/200
40/40 [==============================] - 8s 190ms/step - loss: 0.1494 - precision: 0.9143
- accuracy: 0.9350
Epoch 169/200
40/40 [==============================] - 7s 182ms/step - loss: 0.1765 - precision: 0.9208
- accuracy: 0.9250
Epoch 170/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1701 - precision: 0.9118
- accuracy: 0.9200
Epoch 171/200
40/40 [==============================] - 7s 179ms/step - loss: 0.1405 - precision: 0.9307
- accuracy: 0.9350
Epoch 172/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1675 - precision: 0.9087
- accuracy: 0.9250
Epoch 173/200
40/40 [==============================] - 7s 180ms/step - loss: 0.1907 - precision: 0.8952
- accuracy: 0.9150
Epoch 174/200
40/40 [==============================] - 8s 199ms/step - loss: 0.1546 - precision: 0.9337
- accuracy: 0.9250
Epoch 175/200
40/40 [==============================] - 8s 185ms/step - loss: 0.1441 - precision: 0.9257
- accuracy: 0.9300
Epoch 176/200
40/40 [==============================] - 7s 183ms/step - loss: 0.1530 - precision: 0.9455
- accuracy: 0.9500
Epoch 177/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1560 - precision: 0.9118
- accuracy: 0.9200
Epoch 178/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1722 - precision: 0.9363
- accuracy: 0.9450
Epoch 179/200
40/40 [==============================] - 7s 180ms/step - loss: 0.1662 - precision: 0.9227
- accuracy: 0.9375
Epoch 180/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1512 - precision: 0.9296
- accuracy: 0.9275
Epoch 181/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1625 - precision: 0.9048
- accuracy: 0.9250
Epoch 182/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1446 - precision: 0.9187
- accuracy: 0.9375
Epoch 183/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1541 - precision: 0.9095
- accuracy: 0.9300
Epoch 184/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1479 - precision: 0.9257
- accuracy: 0.9300
Epoch 185/200
40/40 [==============================] - 8s 189ms/step - loss: 0.1571 - precision: 0.9212
- accuracy: 0.9275
Epoch 186/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1771 - precision: 0.9059
- accuracy: 0.9100
Epoch 187/200
```

```
Epoch 187/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1582 - precision: 0.9135
- accuracy: 0.9300
Epoch 188/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1391 - precision: 0.9320
- accuracy: 0.9450
Epoch 189/200
40/40 [==============================] - 8s 202ms/step - loss: 0.1572 - precision: 0.9223
- accuracy: 0.9350
Epoch 190/200
40/40 [==============================] - 8s 186ms/step - loss: 0.1530 - precision: 0.9135
- accuracy: 0.9300
Epoch 191/200
40/40 [==============================] - 7s 179ms/step - loss: 0.1668 - precision: 0.9135
- accuracy: 0.9300
Epoch 192/200
40/40 [==============================] - 8s 187ms/step - loss: 0.1679 - precision: 0.9163
- accuracy: 0.9225
Epoch 193/200
40/40 [==============================] - 7s 180ms/step - loss: 0.1583 - precision: 0.9130
- accuracy: 0.9275
Epoch 194/200
40/40 [==============================] - 7s 174ms/step - loss: 0.1392 - precision: 0.9227
- accuracy: 0.9375
Epoch 195/200
40/40 [==============================] - 8s 191ms/step - loss: 0.1512 - precision: 0.9087
- accuracy: 0.9250
Epoch 196/200
40/40 [==============================] - 7s 175ms/step - loss: 0.1815 - precision: 0.9038
- accuracy: 0.9200
Epoch 197/200
40/40 [==============================] - 7s 178ms/step - loss: 0.1844 - precision: 0.9020
- accuracy: 0.9100
Epoch 198/200
40/40 [==============================] - 7s 177ms/step - loss: 0.1734 - precision: 0.9122
- accuracy: 0.9225
Epoch 199/200
40/40 [==============================] - 7s 179ms/step - loss: 0.1692 - precision: 0.9126
- accuracy: 0.9250
Epoch 200/200
40/40 [==============================] - 7s 176ms/step - loss: 0.1991 - precision: 0.8807
- accuracy: 0.9150
```

Out[57]:

```
<keras.callbacks.History at 0x243586e4eb0>
```

Loading Model

In [4]:

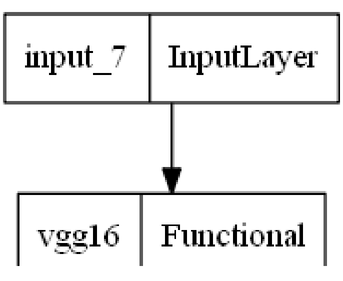
```
model = keras.models.load_model("finalmodel")
```

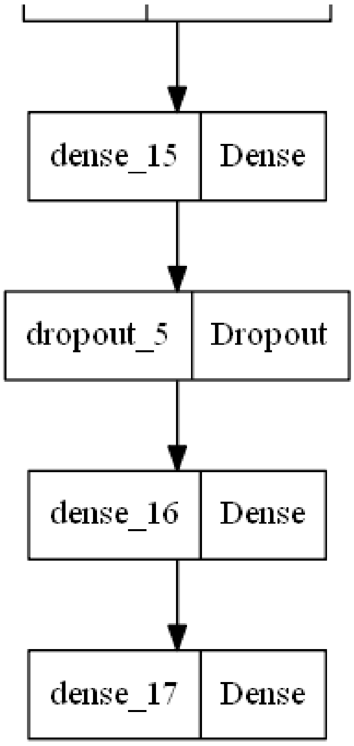
Plotting Model Architecture

In [5]:

```
tf.keras.utils.plot_model(model)
```

Out[5]:





LIVE DEMO

In [6]:

```
def predict(array):
    x=model.predict(array.reshape(1,array.shape[0],array.shape[1],array.shape[2]))
    if x<0.5:
        print("According to the Model, the object is Damaged with a probability of ",roun
d(100*(1-x[0][0]),4),"%.")
    else:
        print("According to the Model, the object is Intact with a probability of ",roun
d(100*(x[0][0]),4),"%.")

def model_predict(num):

    test_data=pd.DataFrame()
    test_data["image"]=["test_images/"+str(num)+".png"]
    test_data["target"]=[1]

    TestImageGenerator=ImageFlow.flow_from_dataframe(dataframe=test_data, x_col="image",
y_col="target",shuffle=False,\
                                                     class_mode="raw", target_size=(224,224), b
atch_size=1)

    i=TestImageGenerator[0][0][0]
    plt.title("Input to the model")
    plt.imshow(i.astype('uint8'))

    plt.figure(figsize=(10,10))
    z=cv2.imread("test_images/"+str(num)+".png")
    plt.title("Real Image")
    plt.imshow(z)
    plt.show()
    predict(i)
```

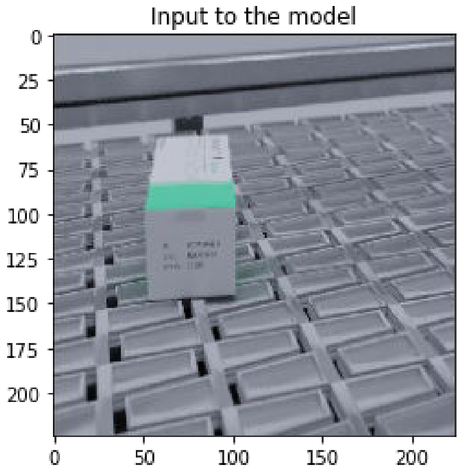
Classification given the image number

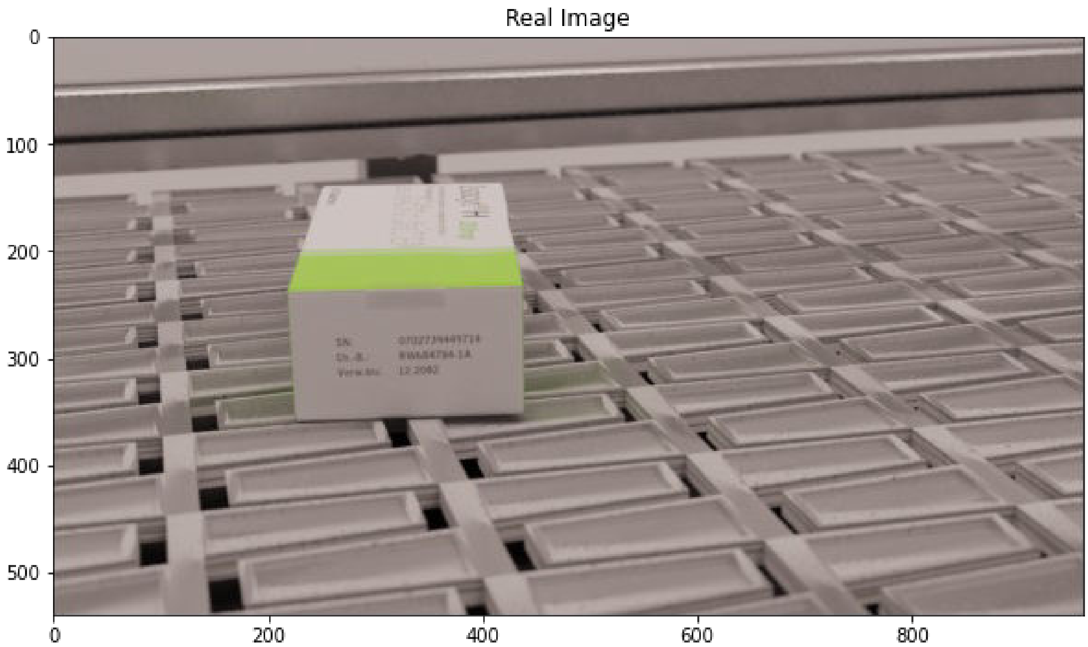
In [19]:

```
num=int(input())
model_predict(num)
```

35

```
Found 1 validated image filenames.
```





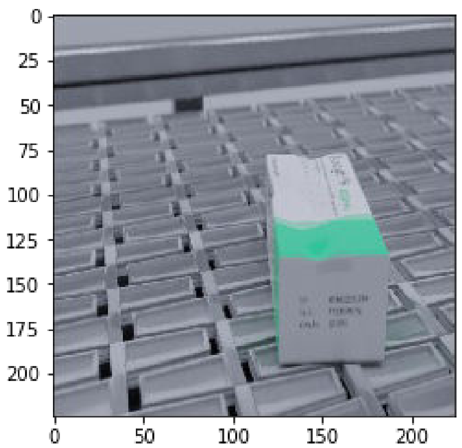

```
According to the Model, the object is Intact with a probability of  99.3404 %.
```

Random Predictor

- **Takes a random image and predicts the class for that image**

In [7]:

```
i=TrainGenerator[np.random.randint(0,40)][0][np.random.randint(0,10)]
plt.imshow(i.astype('uint8'))
plt.show()
predict(i)
```



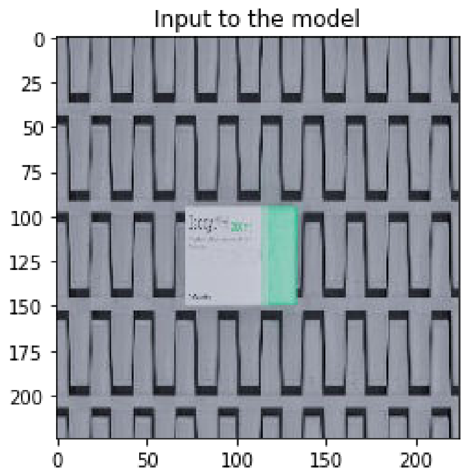

```
According to the Model, the object is Damaged with a probability of  100.0 %.
```

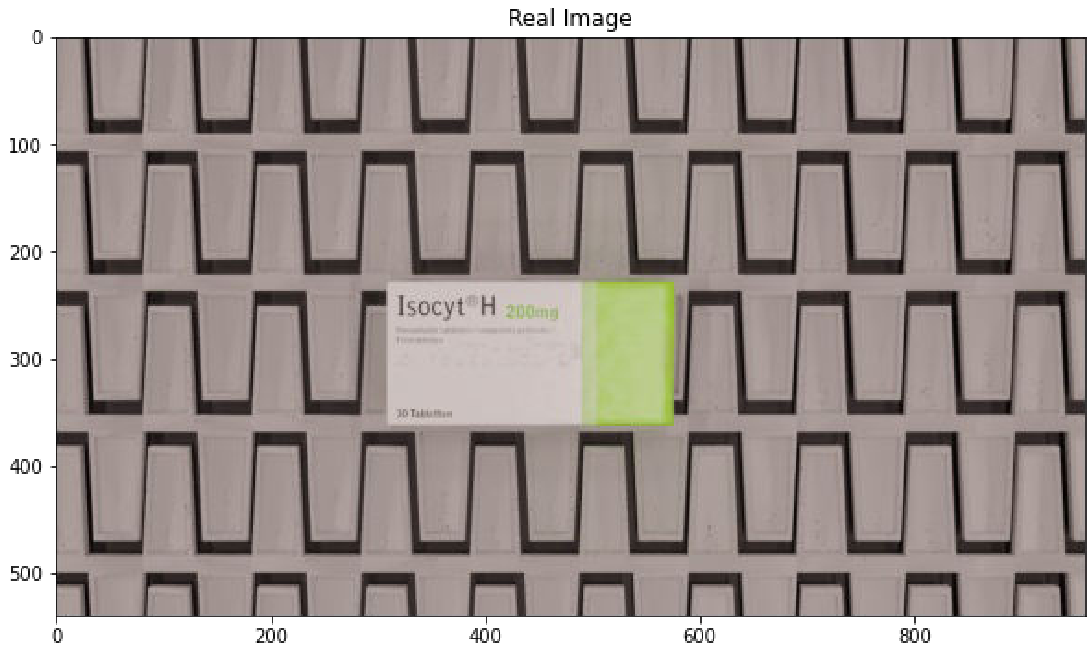
- **With preview**

In [11]:

```
num=np.random.randint(0,265)
print("Random Number generator is ",num)
model_predict(num)
```

```
Random Number generator is  65
Found 1 validated image filenames.
```






```
According to the Model, the object is Intact with a probability of  99.883 %.
```

Components of System

The project utilizes a combination of frameworks and libraries to comprehensively perform the required task. These include:

- Python

Python is the basic building block of the whole project. With booming popularity and it's Opensource nature, python is the first choice for any future-proof technology.

- Pandas[10]

Pandas is a library used for accessing data in a time and storage efficient way.

- Numpy[9]

Numpy is a library used for efficient data storage in form of arrays which aren't available in Python. Python uses lists which aren't as flexible as Numpy Arrays.

- Tensorflow[11]

Tensorflow is the most important framework for Machine Learning in recent times. Developed by Google, it's one of the most used technologies for Machine Learning

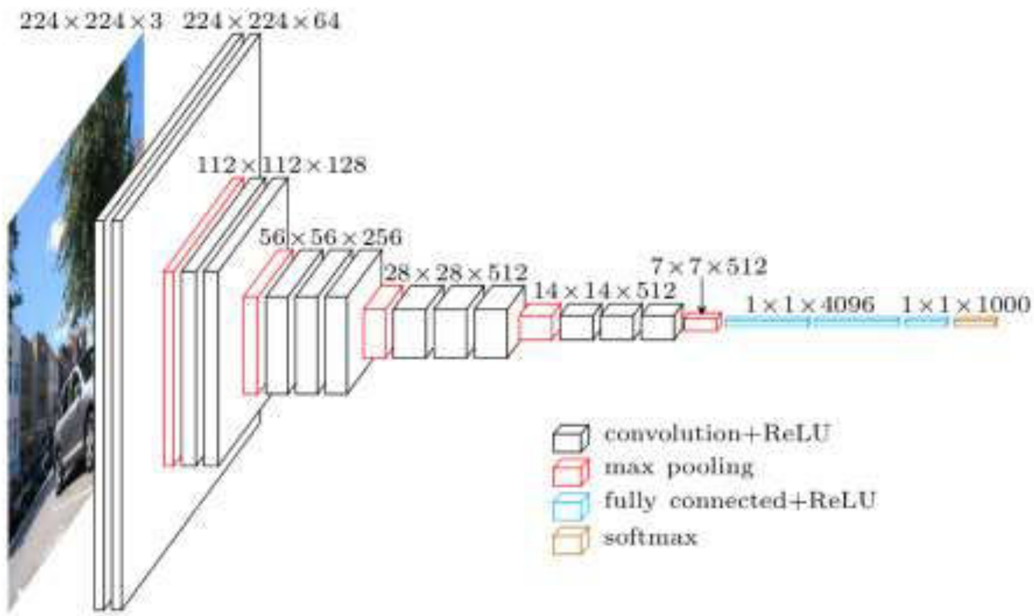
- Keras[7]

Keras is a framework built on top of Tensorflow to provide easy interface for better utilization of Machine Learning algorithms which are hard to understand in pure Tensorflow code.

.

- VGG-16 model[6]

VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University as a part of the "The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" for Computer Vision. The model architecture is as follows:

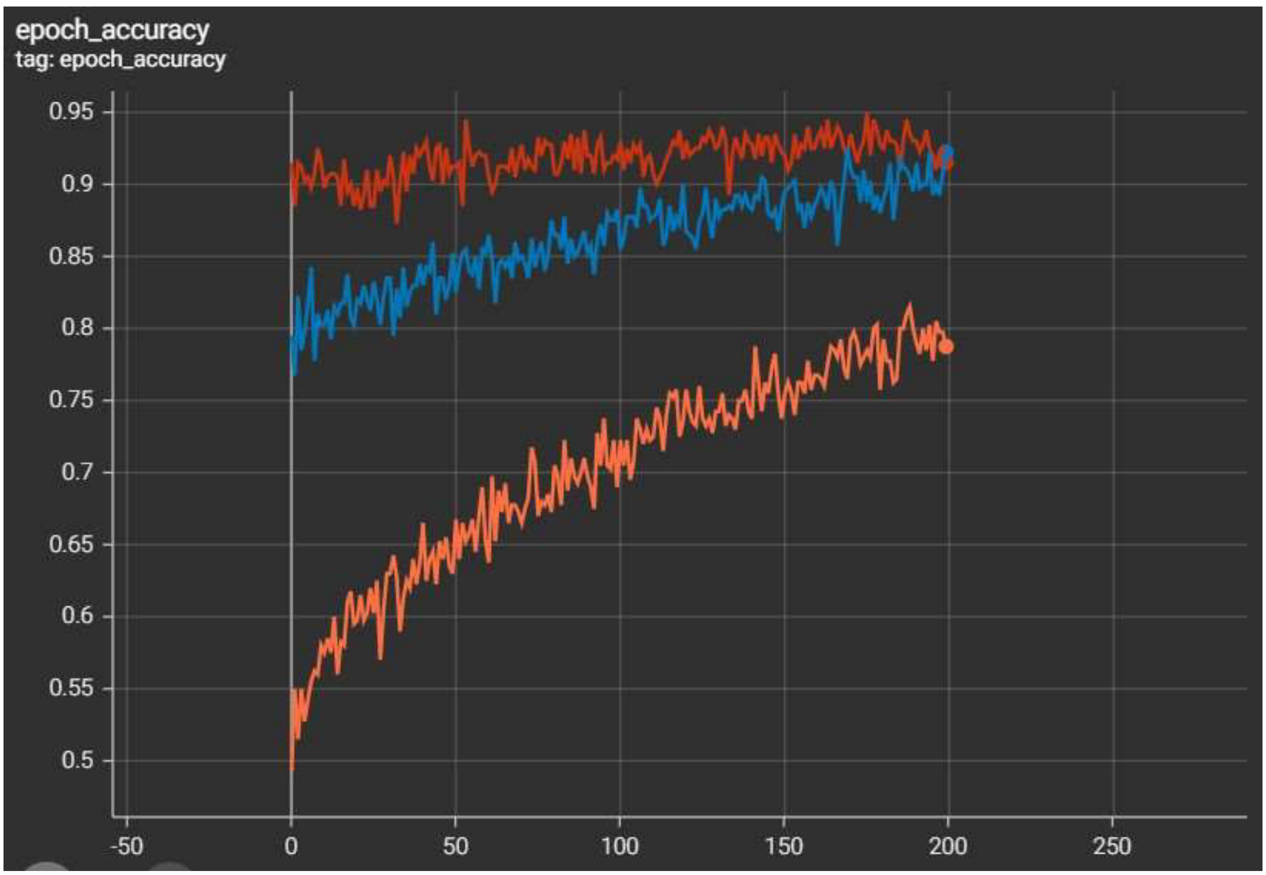


The model developed as a part of this project utilizes this without training capabilities for whole except the last 3 fully connected layers.

Testing & Analysis

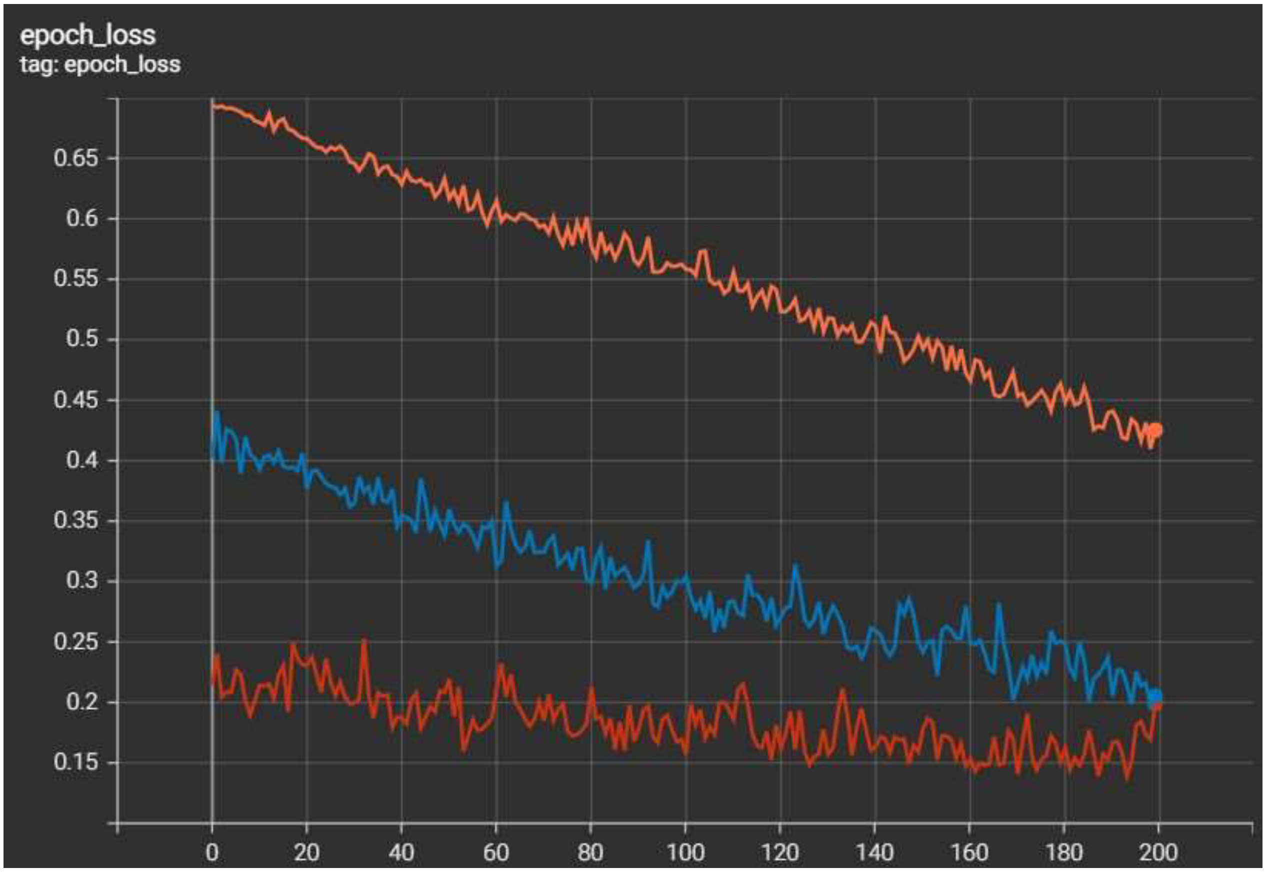
Analyzing Training:
On a system of NVIDIA GTX 1650 with Intel i5 processor, it takes about 70 minutes. So, training is divided into 3 separate stages of 200 epochs(steps) each. In order to limit the contents, the code files contain the data of only the third epoch. The accuracy of result and loss(binary-crossentropy) with each epoch is given as follows:

Accuracy vs Epoch:



Here:
Orange represents Stage:1Training
Blue represents Stage:2 Training
Red represents Stage:3 Training

Loss vs Epoch:

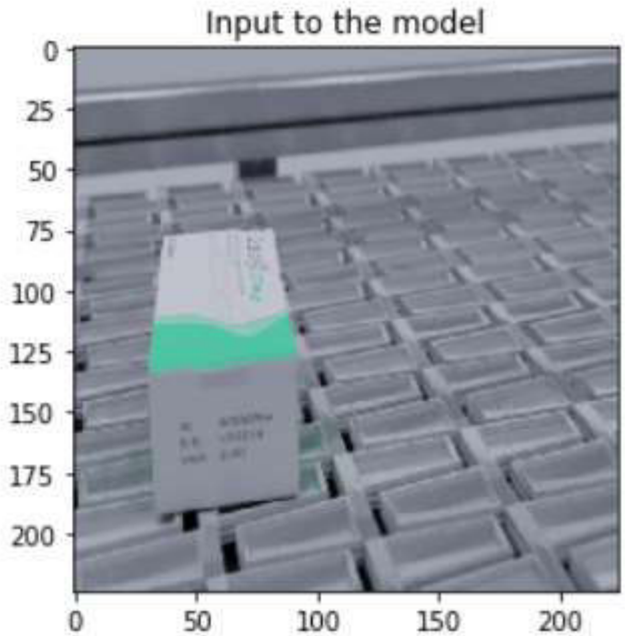


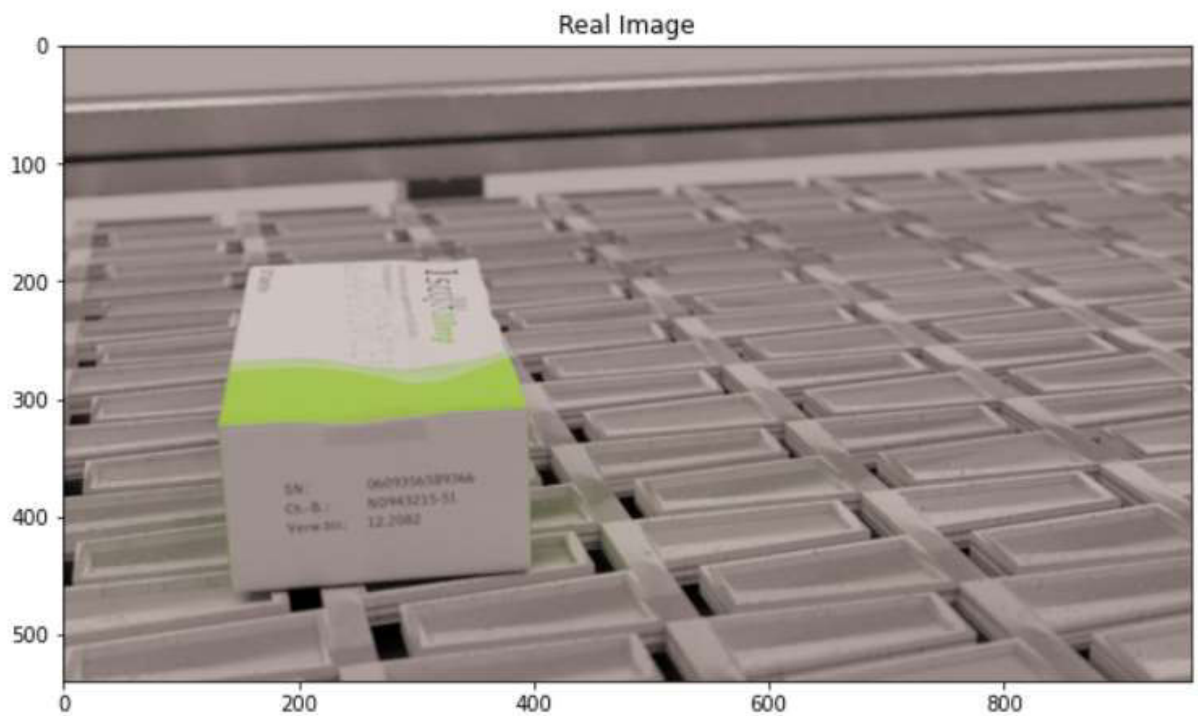
Here:
Orange represents Stage:1 Training
Blue represents Stage:2 Training
Red represents Stage:3 Training

Testing

Picking 2 random images and use the model to predict it's condition.

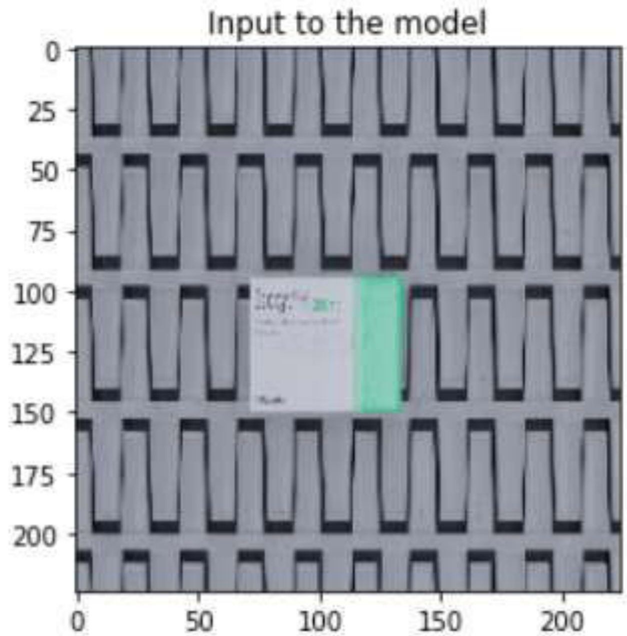
```
Random Number generator is  338
Found 1 validated image filenames.
```

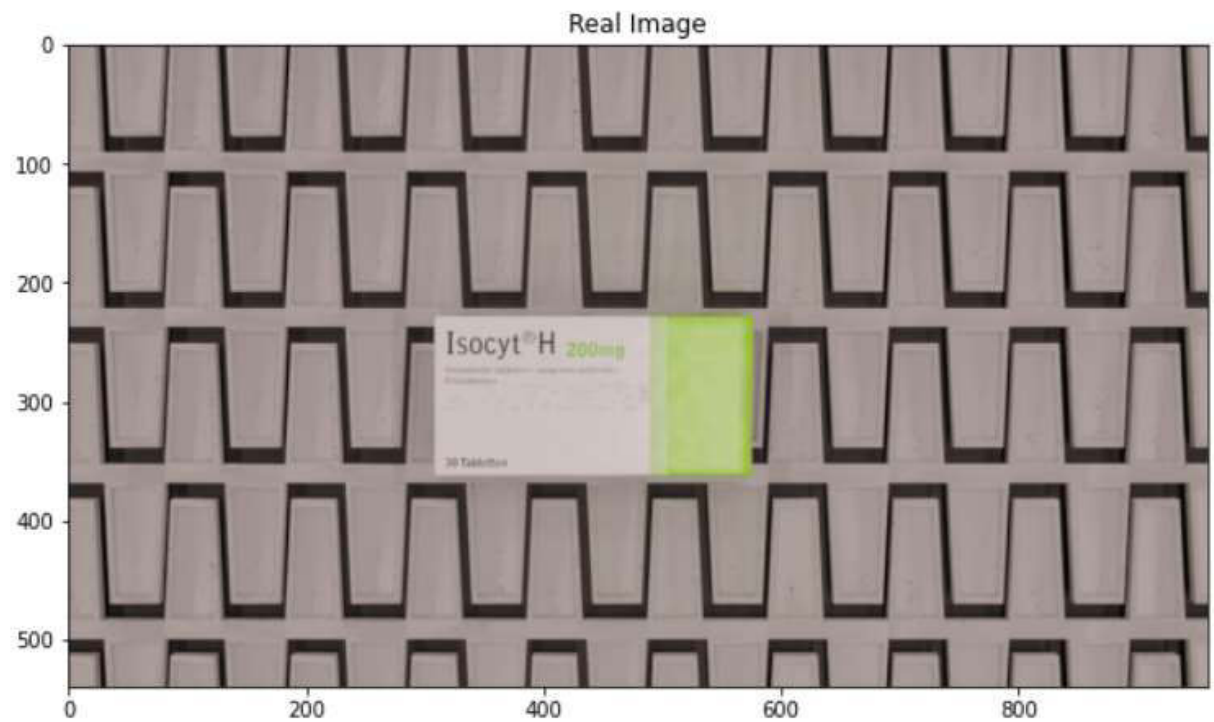





```
According to the Model, the object is Damaged with a probability of  95.8418 %.
```

```
Random Number generator is  65
Found 1 validated image filenames.
```






```
According to the Model, the object is Intact with a probability of  99.883 %.
```

Results

With the successful demonstration of accessibility and practicality of a machine learning algorithm to detect damage to an object, we have also successfully proposed a more efficient solution for quality control in manufacturing industries.

Like the detection of damage to a medicine box, damage to almost every produced component can be detected using the same machine learning algorithm trained on different images of respective components. The scalability of this algorithm is more compared to the traditional method. Image capturing devices connected to a processing unit with an internet connection is all that's required for a scalable solution. These devices can be used to detect quality of multiple components of any given material simultaneously. This minimizes need for a technical operator and the whole process of feedback loop, repair and improvement with regards to the feedback is efficiently automated.

Machine Learning models trained on a particular component can accurately predict it's condition. Training a model on a new component and using it in production instantly is possible using IOT. Combining multiple models makes detecting conditions of multiple objects in a sequence or simultaneously possible.

References

1. https://www.geeksforgeeks.org/vgg-16-cnn-model/
2. https://arxiv.org/pdf/1409.1556.pdf
3. https://www.sciencedirect.com/science/article/pii/S1474034620300707
4. https://ijcrt.org/papers/IJCRT2006458.pdf
5. https://www.kaggle.com/datasets/christianvorhemus/industrial-quality-control-of-packages
6. https://keras.io/api/applications/vgg/
7. https://keras.io/
8. https://github.com/christian-vorhemus/procedural-3d-image-generation
9. https://numpy.org/doc/stable/
10. https://pandas.pydata.org/pandas-docs/stable/
11. https://www.tensorflow.org/api_docs

Group Members Details

Sr. No.	Group. No.	Name of the Students	Email id	Mobile No.
01	5	Raorane Paras	parasganeshraorane@gmail.com	8424024587
02		Rathod Bhagyesh	bhagyeshr9860@gmail.com	7028959240
03		Raut Shubham	sraut3349@gmail.com	7666501152
04		Sankpal Sahil	sankpalsahil02@gmail.com	9545082438