



In collaboration with



Internship Report on

# Create Tech Prototype

*cargo inspections of Agro-commodities at ports*

Presented by

VISHWESHWAR PARAMESHWAR BHAT

AYUSHMAN HAZARIKA

ROHAN BHAGWAT

T. AISHWARYA

Along with Internal Mentor

Dr. Prashant P Patavardhan

## Contents

<b><i>Sl. No</i></b>	<b>Topic</b>	<b>Page Number</b>
1.	Problem Statement	3
2.	Introduction	3-4
3.	Methodology	4-7
4.	Materials used	7-8
5.	Code	9-16
6.	Results	17-25
7.	Validation	26
8.	Conclusion	27
9.	Future scope	28
10.	Acknowledgement	29
11.	Project Details	30

## Problem Statement

“We would like to reduce the cost and time required to do cargo inspections of Agro-commodities at ports - perishables like onions, and mangoes, and semi-perishables like rice and wheat flour. Currently, these are done manually by experts who charge tens of thousands of rupees a day and are too expensive for smaller exporters.

We would like to see a small-scale prototype of an automated or semi-automated solution. Check for quality of product, accuracy of labelling or both.”

We would want to start by first taking an individual item, and the item that we have considered is **onion**. We aim at creating an automated solution to solve the problem mentioned above and to check the quality of the onions.

## Introduction

Considering the present trends in the cargo inspection industry, it is observed that lots of manpower is needed to do the inspection. This not only wastes the time but also results in inaccuracy, and charges a lot of money. There are various companies which provide cargo inspection services, and they charge very high fees. To eliminate this, we have designed an automated system, which can be used to determine the quality of goods without any hassle and manpower.

Automation improves the nation's quality, production, and economic growth in agriculture science. The selection of fruits and vegetables has an impact on the export market and quality assessment. The most important sensory quality of fruits and vegetables is their look, which influences their market value as well as consumer preference and selection.

Images are the most fundamental approach in the physical classification of food products and the agricultural business for representing conception for the human brain. Visual quantification of the elements affecting fruits and vegetables is possible, but it is time-consuming, expensive, and vulnerable to subjective judgement and physical influence. These inspections and the "best-if-used-before date" help establish market values. The quality assessment was carried out by skilled human investigators using their senses of touch and sight.

This approach is highly erratic, capricious, and rarely yields consistent results across investigators. Machine vision systems are ideally suited for traditional analysis and quality

assurance in this sort of environment since it is a continuous task to analyse fruits and vegetables for various aspect criteria. Computer vision systems and image processing are a rapidly expanding study field in agriculture.

Computer vision-based pattern recognition and image processing are established techniques for safety and quality analysis of many agricultural applications, and information science is a field that is expanding quickly.

## **Methodology**

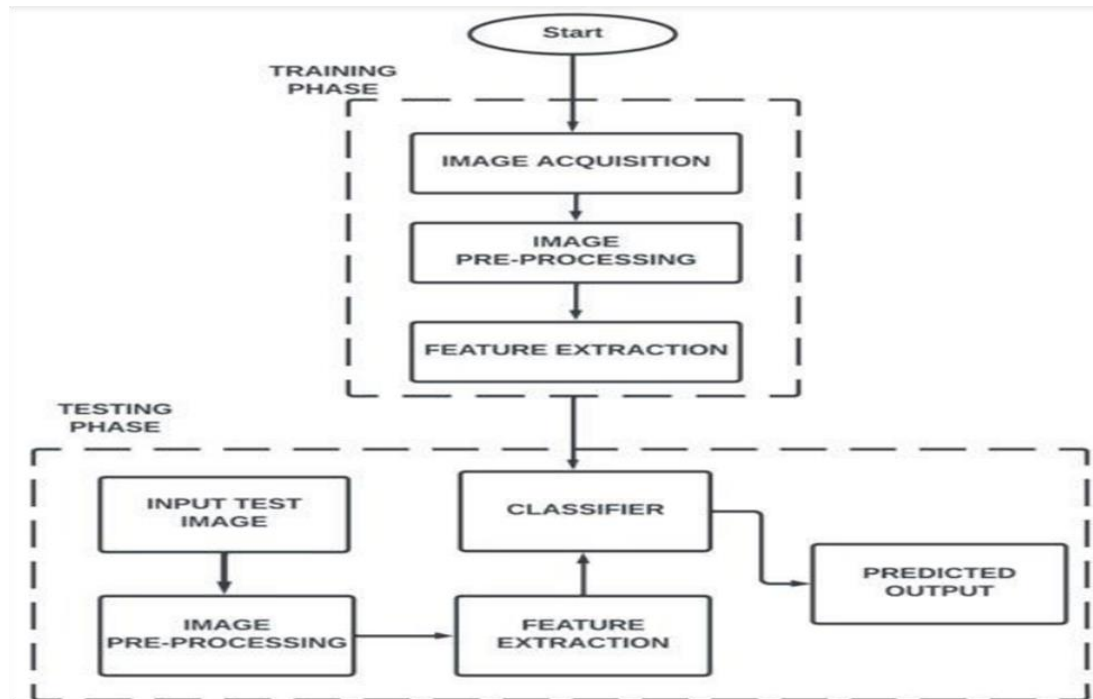
### **Dataset**

1. A dataset of onions was considered for training the model for the project which was created from scratch. The created dataset for the model to train is made using images taken from a light box under accurate conditions.
2. For creating a perfect dataset with minimal noise and disturbance we use the light box [it is a box with translucent sides and white backdrop]. The box is intended to be empty so that we can place the onions inside the box which helps to take a photo and get a result that has professional-quality lighting, with no shadows disrupting the plain, even background.
3. For the model to be trained properly we need both types of onions [good and bad]. We place the onions in different angles to get a proper dataset which can be used in real-time applications.
4. Similarly, we take photos for both the types of onions. The dataset created has around 800 images of onion as input for training the system. The proposed system is implemented using Python and Jupyter notebook and the data-set of images is taken in .jpg format.

### **Model Pre-processing and Training Phase**

1. In the training phase of the model, after creating the dataset for the model to train, the dataset images are pre-processed which is needed for eliminating the unwanted information from the image. Using image processing techniques, higher accuracy can be obtained from the given images.

2. In the pre-processing step, the images are first normalised through normalisation for reducing the data loss of the images, then gray scaling of image to reduce the complexities related to computational requirements, ( grayscale means that the value of each pixel represents only the intensity information of the light).
3. Such images typically display only the darkest black to the brightest white. We use structuring element which is a binary image (consisting of 1's and 0's) that is used to probe an image for finding the region of interest.
4. The pattern of 1's and 0's specifies the shape of structuring element. Then the images are introduced to data augmentation for diversifying and increasing the dataset size when the classifier is trained. After the pre-processing, the **features like colour, shape, size, texture, etc are extracted from the images**. The extracted features are used to train the classifier.
5. The proposed system uses Convolutional Neural Networks (CNN), The model architecture as the consist of **5 convolution layers**, A convolutional layer is the main building block of a CNN.
6. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training. The size of the **filters is usually smaller than the actual image, a drop out layer with 50% drop out ratio** a [the dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others] , and **3 dense layers** [Dense Layer is simple layer of neurons in which each neuron receives input from all the neurons of previous layer] and a **deep learning algorithm**, which consists of a class of neural networks as a classifier for image recognition by a specialised way of **processing on the grid of pixels**.
7. This process trains the system for **classifying and grading**. For the training phase, the images of the dataset consist of fresh and stale classes of onion for which we must train the model.



## Model Testing Phase

1. In the testing phase of the model, a set of testing images are given as input for the system. These images go through the **image-processing and feature extraction processes**. Based on the system classification the given input onion is graded and the result is displayed. The training and testing phases are shown in the above block diagram.
2. CNN network **has three additional hidden layers**, namely an input layer, an output layer, and middle layers that are composed of multiple convolution layers as well as fully connected layers.
3. The CNN uses convolution for extracting the features and flattens the values and then uses functions such as 'ELu' [is a function that tend to converge cost to zero faster and produce more accurate results] for further reducing the matrices. The accuracy of the model is calculated as the ratio between the number of correct predictions to the total number of predictions.
4. This model is then checked to see whether it is fixed for the given application and then the model is used for testing the inputs. With this CNN architecture, the system converts the given input image into an array of pixel values using the convolution and after a certain number of times the convolution is done, then the data is flattened using 'ELu'.

The model runs through the data many times, these are called **epochs**. As the number of epochs increases, the more the model improves to a certain extent.

5. The accuracy of the model was determined by checking the number of test cases that were returned correctly when tested with a particular input. The developed model returned the value of the onion as good or bad and gave **a testing accuracy of about 96-97%, Precision of 96-97% and Recall factor of 95-96%.**

## **Materials used and Process of obtaining the Dataset**

1. Light Box – for making the data set
2. Camera – 48 mega pixels
3. Tripod Stand
4. Different types of Onions – good, bad etc

### **Software requirements**

1. Python interpreter – Jupyter Notebook

### **Steps followed in obtaining the data set**

1. Set up a well-lit place for the procurement of pictures
2. Set up the light box



3. Fix the tripod stand at 30 CMS from the light box with the camera exactly lying on the parallel axis of the light box's base.



4. Place the onions one after the other and click pictures in different angles maintaining the same distance and lighting conditions.

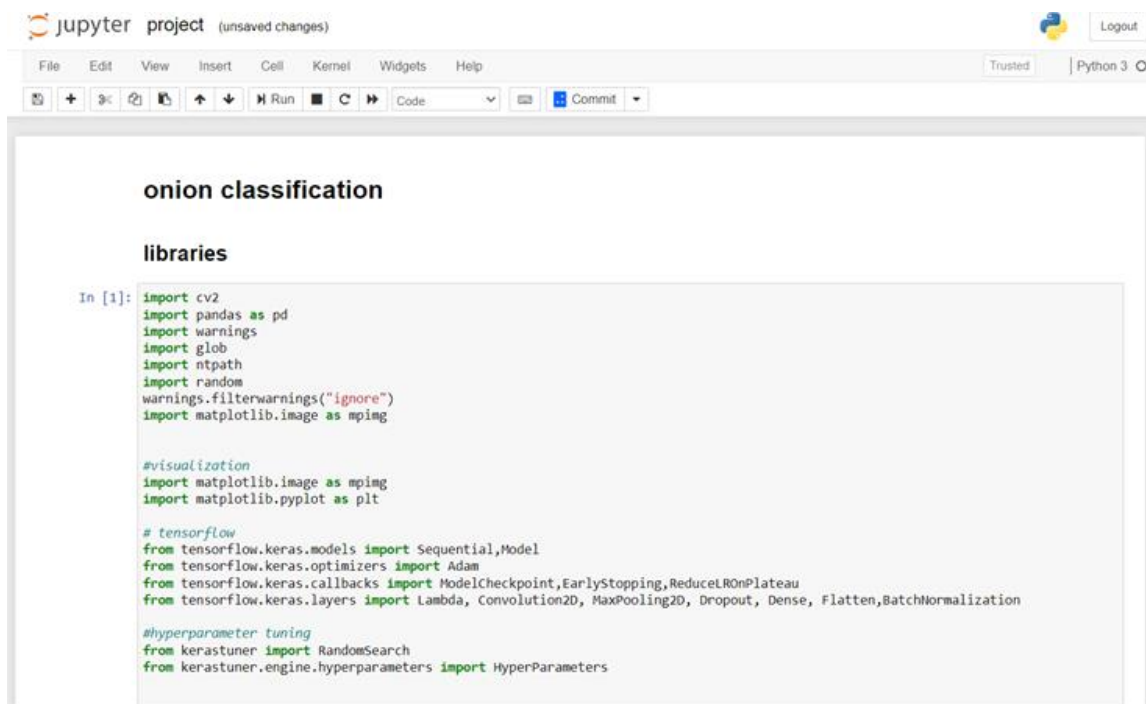


5. Add additional lights from the corners of the light box in order to minimize shadows.





# Code



jupyter project (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Widgets Help Trusted

onion classification

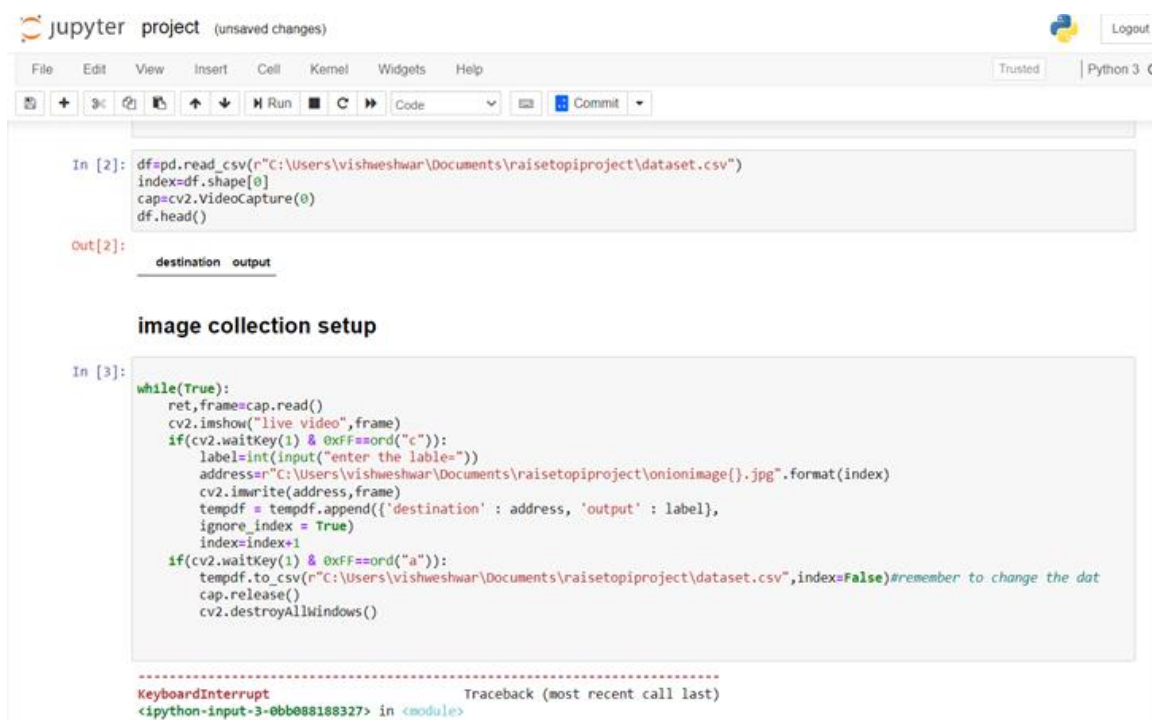
libraries

```
In [1]: import cv2
import pandas as pd
import warnings
import glob
import ntpath
import random
warnings.filterwarnings("ignore")
import matplotlib.image as mpimg

#visualization
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# tensorflow
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Lambda, Convolution2D, MaxPooling2D, Dropout, Dense, Flatten, BatchNormalization

#hyperparameter tuning
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
```



jupyter project (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Widgets Help Trusted

```
In [2]: df=pd.read_csv(r"C:\Users\vishweshwar\Documents\raisetopiproject\dataset.csv")
index=df.shape[0]
cap=cv2.VideoCapture(0)
df.head()
```

Out[2]:

destination	output
-------------	--------

image collection setup

```
In [3]: while(True):
ret,frame=cap.read()
cv2.imshow("live video",frame)
if(cv2.waitKey(1) & 0xFF==ord("c")):
    label=int(input("enter the label="))
    address=r"C:\Users\vishweshwar\Documents\raisetopiproject\onionimage{}.jpg".format(index)
    cv2.imwrite(address,frame)
    tempdf = tempdf.append({'destination' : address, 'output' : label},
    ignore_index = True)
    index=index+1
if(cv2.waitKey(1) & 0xFF==ord("a")):
    tempdf.to_csv(r"C:\Users\vishweshwar\Documents\raisetopiproject\dataset.csv",index=False)#remember to change the dat
    cap.release()
    cv2.destroyAllWindows()
```

KeyboardInterrupt Traceback (most recent call last)

<ipython-input-3-0bb088188327> in <module>

jupyter project (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Widgets Help Trusted

Code Commit

### automated labelling

```
In [3]: def label_img_folders(folders):
        tempdf=pd.read_csv(r"C:\Users\vishweshwar\Documents\raisetopiproject\dataset.csv")
        for folder in folders:
            for img in glob.glob(r"C:\Users\vishweshwar\Documents\raisetopiproject\{}\*.jpg".format(folder)):
                if folder=="goodonion":
                    label=0
                if folder=="badonion":
                    label=1
                tempdf = tempdf.append({'destination': img, 'output': label},ignore_index = True)
            tempdf.to_csv(r"C:\Users\vishweshwar\Documents\raisetopiproject\dataset.csv",index=False)
        return tempdf
df=label_img_folders(["goodonion","badonion"])
```

```
In [4]: df.head()
```

```
Out[4]:
```

	destination	output
0	C:\Users\vishweshwar\Documents\raisetopiprojec...	0
1	C:\Users\vishweshwar\Documents\raisetopiprojec...	0
2	C:\Users\vishweshwar\Documents\raisetopiprojec...	0
3	C:\Users\vishweshwar\Documents\raisetopiprojec...	0
4	C:\Users\vishweshwar\Documents\raisetopiprojec...	0

### image processing for finding the area

jupyter project (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Widgets Help Trusted

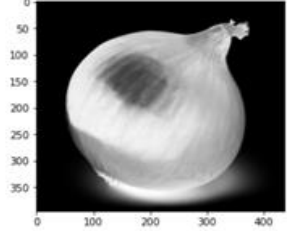
Code Commit

### image processing for finding the area

```
In [5]: img =cv2.imread("img.jpg")
```

```
In [6]: grayimg =cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        inverted=~grayimg
        plt.imshow(inverted,"gray")
```

```
Out[6]: <matplotlib.image.AxesImage at 0x2afb052d970>
```



```
In [7]: def onionArea(image):
        RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        gray=cv2.cvtColor(RGB, cv2.COLOR_RGB2GRAY)
        blur=cv2.medianBlur(gray, 25)
        thresh=cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 27, 6)
        kernel =cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
        close=cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)
        dilate=cv2.dilate(close, kernel, iterations=2)
```

jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

0 100 200 300 400


```
In [7]: def onionArea(image):
        RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        gray=cv2.cvtColor(RGB,cv2.COLOR_RGB2GRAY)
        blur=cv2.medianBlur(gray,25)
        thresh=cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,27,6)
        kernel =cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
        close=cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel,iterations=1)
        dilate=cv2.dilate(close,kernel,iterations=2)

        cnts=cv2.findContours(dilate,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]

        cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:10]
        withcontour=cv2.drawContours(RGB,cnts,0,(255,0,0),5)
        area=cv2.contourArea(cnts[0])
        return (withcontour,area)

        image=cv2.imread("onion4.jpg")
        outputimage,area=onionArea(image)
        plt.imshow(outputimage)
        plt.xticks([])
        plt.yticks([])
        print(f"area of the onion is {area}")

        area of the onion is 160560.0
```



jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

0 100 200 300 400

```
In [8]: image=cv2.imread("onion9.jpg")
        RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        gray=cv2.cvtColor(RGB,cv2.COLOR_RGB2GRAY)
        blur=cv2.medianBlur(gray,25)
        canyedge=cv2.Canny(blur,30,100)

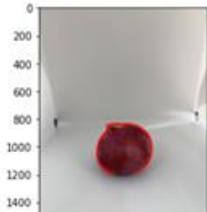
        kernel =cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
        close=cv2.morphologyEx(canyedge,cv2.MORPH_CLOSE,kernel,iterations=1)
        dilate=cv2.dilate(close,kernel,iterations=2)

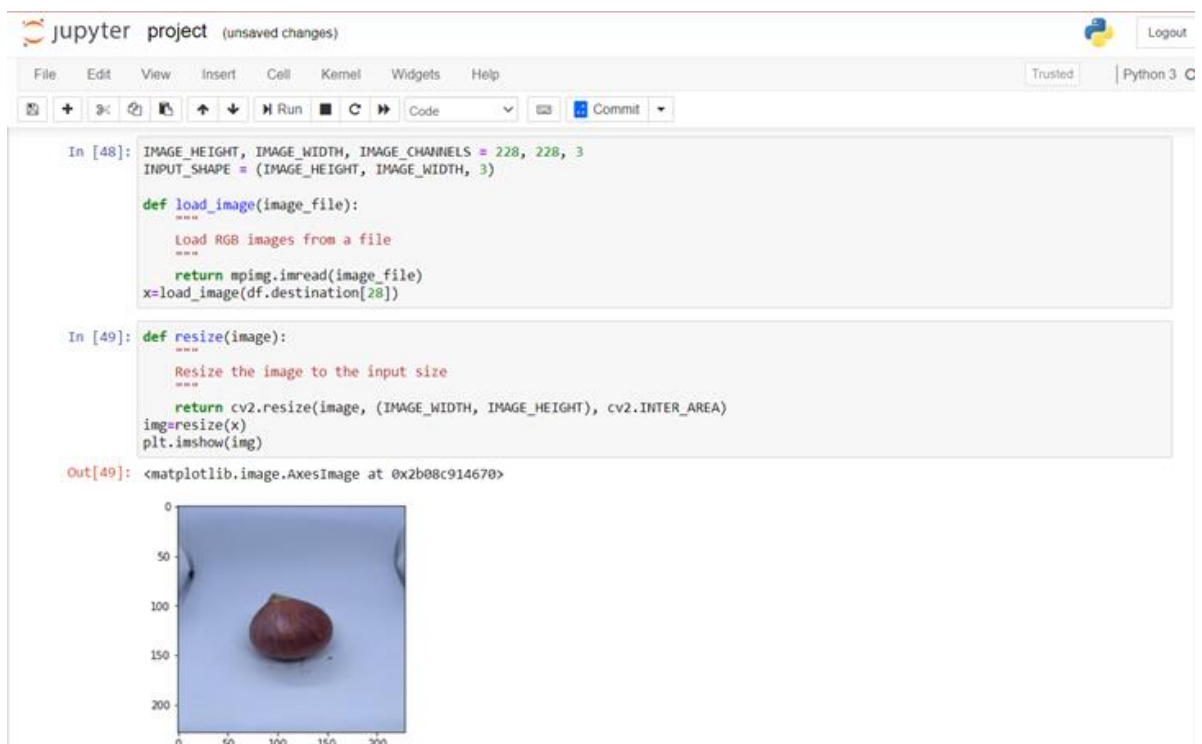
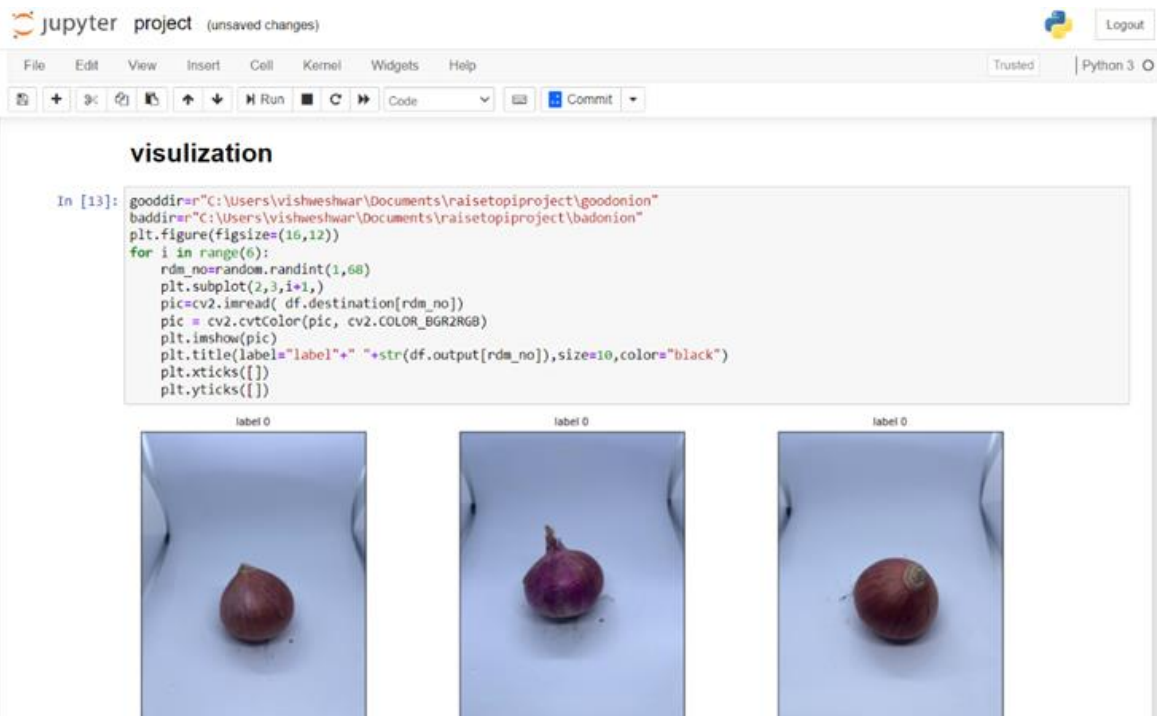
        cnts=cv2.findContours(dilate,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]

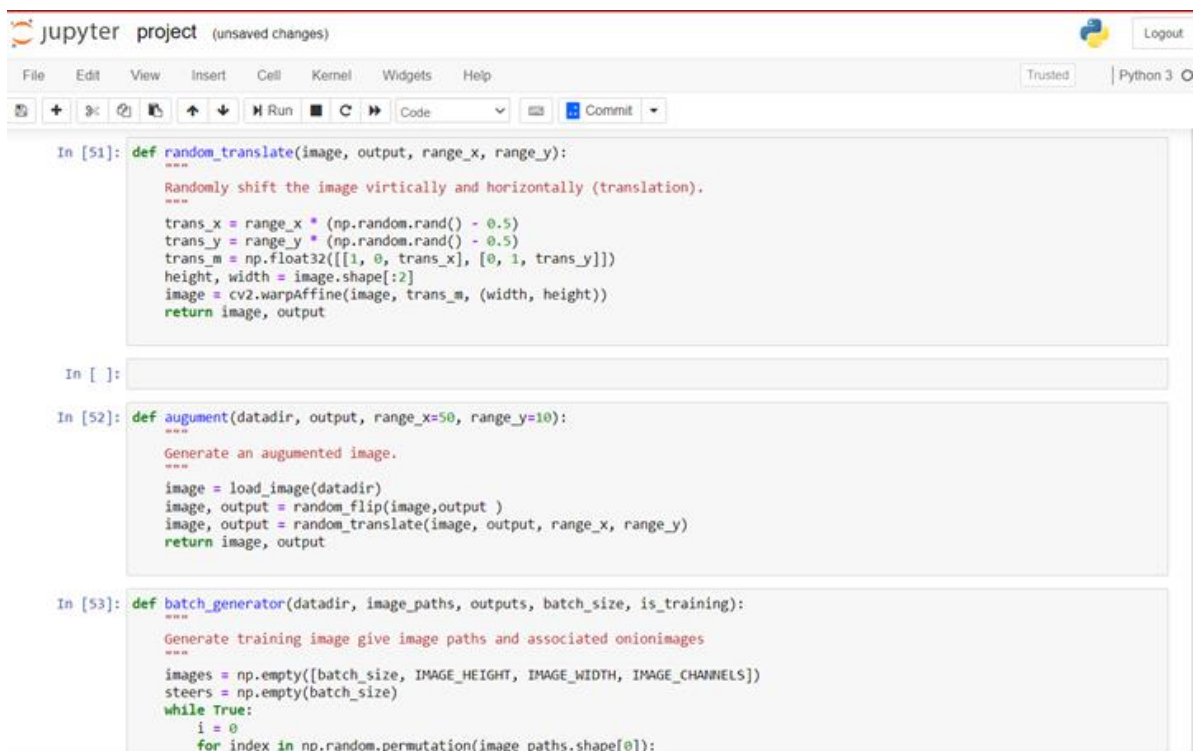
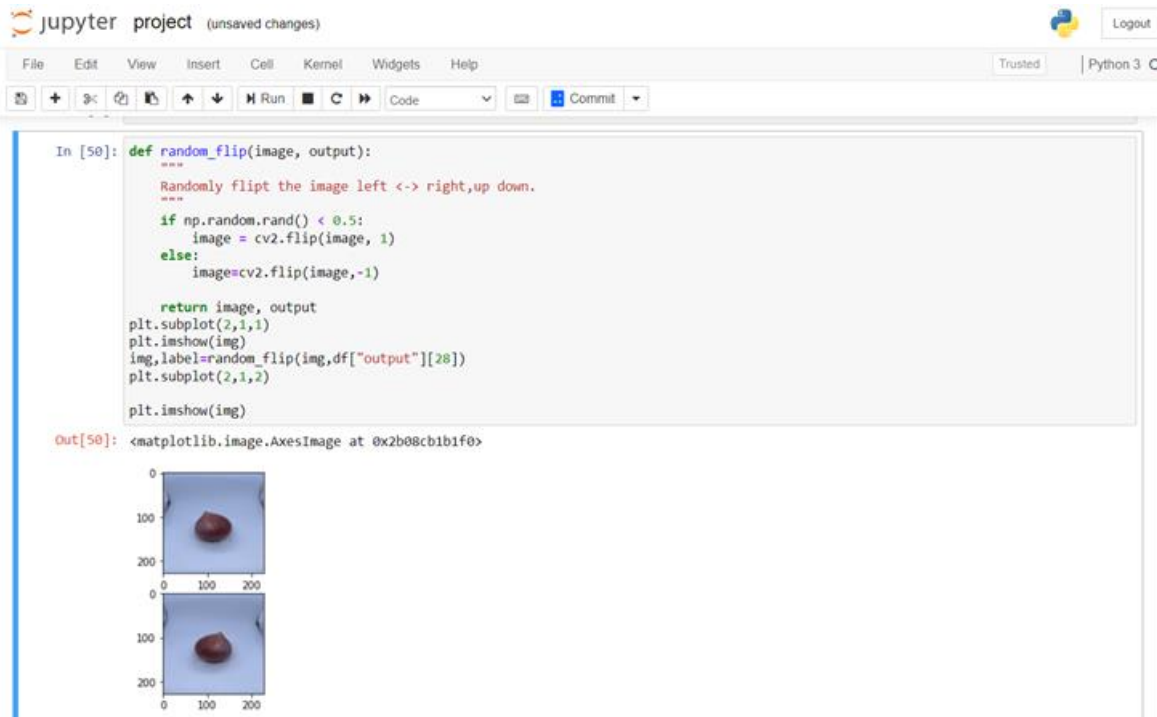
        cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:10]
        withcontour=cv2.drawContours(RGB,cnts,0,(255,0,0),5)

In [9]: plt.imshow(withcontour)
```

Out[9]: <matplotlib.image.AxesImage at 0x2afb3194460>









jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [53]: def batch_generator(datadir, image_paths, outputs, batch_size, is_training):
        """
        Generate training image give image paths and associated onionimages
        """
        images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
        steers = np.empty(batch_size)
        while True:
            i = 0
            for index in np.random.permutation(image_paths.shape[0]):
                center = image_paths[index]
                output = outputs[index]
                # augmentation
                if is_training and np.random.rand() < 0.6:
                    image, output = augment(center, output)
                else:
                    image = load_image(center)
                    images[i] = resize(image)
                    steers[i] = output
                    i += 1
                if i == batch_size:
                    break
            yield images, steers

In [54]: def load_data(datadir):
        """
        Load training data and split it into training and validation set
        """
        data_df = pd.read_csv(datadir)

        X = df["destination"].values
        y = df["output"].values

        X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=0)
```

jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [55]: def build_model():
        """
        building the model
        """
        model = Sequential()
        model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
        model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(228, 228, 3), activation='elu'))
        model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
        model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
        model.add(Convolution2D(64, (3, 3), activation='elu'))
        model.add(Convolution2D(64, (3, 3), activation='elu'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(100, activation='elu'))
        model.add(Dense(50, activation='elu'))
        model.add(Dense(10, activation='elu'))
        model.add(Dense(1))
        model.summary()
        return model

In [56]: def train_model(model, X_train, X_valid, y_train, y_valid, datadir, saving):
        """
        Train the model
        """
        callback=[EarlyStopping(monitor='val_loss', min_delta=0.01, patience=4, verbose=0),
                  ModelCheckpoint(str(saving)+"-model-{epoch:03d}.h5", monitor='val_loss', save_best_only=True, verbose=0),
                  ReduceLROnPlateau(monitor='val_loss', factor=0.5, min_delta=0.001, verbose=0, patience=2)]

        model.compile(loss='mean_squared_error', metrics=["accuracy"], optimizer=Adam(learning_rate=0.0001))
```

jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [56]:

```
def train_model(model,X_train, X_valid, y_train, y_valid,datadir,saving):
    """
    Train the model
    """
    callback=[EarlyStopping(monitor='val_loss',min_delta=0.01, patience=4,verbose=0),
              ModelCheckpoint(str(saving)+"model-{epoch:03d}.h5",monitor='val_loss',save_best_only=True,verbose=0),
              ReduceLROnPlateau(monitor='val_loss',factor=0.5,min_delta=0.001,verbose=0,patience=2)]

    model.compile(loss='mean_squared_error', metrics=["accuracy"],optimizer=Adam(learning_rate=0.0001))

    model.fit(batch_generator(datadir, X_train, y_train, 32, True),batch_size=32,epochs=15,
              callbacks=[callback],steps_per_epoch=32,
              validation_data=batch_generator(datadir, X_valid, y_valid, 32, False),
              verbose=1,validation_steps=32)

    return model
```

In [57]:

```
def Callback(save):
    """
    callbacks to avoid over-fitting and saving the model
    """
    callback=[EarlyStopping(monitor='val_loss',min_delta=0.001, patience=4,verbose=0),
              ModelCheckpoint(str(save)+"model.h5",monitor='val_loss',save_best_only=True,verbose=0),
              ReduceLROnPlateau(monitor='val_loss',factor=0.5,min_delta=0.001,verbose=0,patience=2)]

    return callback
datadir="C:\Users\vishweshwar\Documents\raisetopiproject\dataset.csv"
```

In [58]:

```
X_train, X_valid, y_train, y_valid=load_data(datadir)
#building the model
model=build_model()
#training the model
train=train_model(model,X_train, X_valid, y_train, y_valid,datadir,"old")
```

jupyter project (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [58]:

```
X_train, X_valid, y_train, y_valid=load_data(datadir)
#building the model
model=build_model()
#training the model
train=train_model(model,X_train, X_valid, y_train, y_valid,datadir,"old")
```

Layer (type)	Output shape	Param #
lambda_1 (Lambda)	(None, 228, 228, 3)	0
conv2d_5 (Conv2D)	(None, 112, 112, 24)	1824
conv2d_6 (Conv2D)	(None, 54, 54, 36)	21636
conv2d_7 (Conv2D)	(None, 25, 25, 48)	43248
conv2d_8 (Conv2D)	(None, 23, 23, 64)	27712
conv2d_9 (Conv2D)	(None, 21, 21, 64)	36928
dropout_1 (Dropout)	(None, 21, 21, 64)	0
flatten_1 (Flatten)	(None, 28224)	0
dense_4 (Dense)	(None, 100)	2822500

In [63]:

```
from tensorflow.keras.preprocessing import image
prediction_list = []
for i in X_valid:
    test_image = load_image(i)
    images=resize(test_image)
    images = image.img_to_array(images)
    images = np.expand_dims(images, axis=0)
```

```
prediction_list = []
for i in X_valid:
    test_image = load_image(i)
    images = resize(test_image)
    images = image.img_to_array(images)
    images = np.expand_dims(images, axis=0)

    prediction = train.predict(images)

    if prediction <= 0.5:
        prediction_list.append(0)
    else:
        prediction_list.append(1)
```

```
In [70]: from sklearn.metrics import confusion_matrix
import seaborn as sns
confuse = confusion_matrix(prediction_list, y_valid)
sns.heatmap(confuse, annot=True)
```

Out[70]: <AxesSubplot:>





## Results

1. The model was built using several inbuilt libraries and functions, as shown in **Fig a.** below followed by the overall **architecture of the model** and its various layers defined as depicted in **Fig. b.**

```
In [59]: def build_model():
        """
        building the model
        """
        model = Sequential()
        model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
        model.add(Convolution2D(32, (5, 5), (2, 2), input_shape=(228, 228, 3), activation='elu'))
        model.add(Convolution2D(16, (5, 5), (2, 2), activation='elu'))
        model.add(Convolution2D(16, (5, 5), (2, 2), activation='elu'))
        model.add(Convolution2D(8, (3, 3), activation='elu'))
        model.add(Convolution2D(8, (3, 3), activation='elu'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(32, activation='elu'))
        model.add(Dense(32, activation='elu'))
        model.add(Dense(8, activation='elu'))
        model.add(Dense(1))
        model.summary()
        return model

In [60]: def train_model(model,X_train, X_valid, y_train, y_valid,datadir,saving):
        """
        Train the model
        """
        callback=[EarlyStopping(monitor='val_loss',min_delta=0.01, patience=4,verbose=0),
                  ModelCheckpoint(str(saving)+"_model-{epoch:03d}.h5",monitor='val_loss',save_best_only=True,verbose=0),
                  ReduceLROnPlateau(monitor='val_loss',factor=0.5,min_delta=0.001,verbose=0,patience=2)]
        model.compile(loss='mean_squared_error', metrics=["accuracy"], optimizer=Adam(learning_rate=0.0001))
```

**Fig a. Overall model definition**

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
lambda_1 (Lambda)	(None, 228, 228, 3)	0
conv2d_5 (Conv2D)	(None, 112, 112, 32)	2432
conv2d_6 (Conv2D)	(None, 54, 54, 16)	12816
conv2d_7 (Conv2D)	(None, 25, 25, 16)	6416
conv2d_8 (Conv2D)	(None, 23, 23, 8)	1160
conv2d_9 (Conv2D)	(None, 21, 21, 8)	584
dropout_1 (Dropout)	(None, 21, 21, 8)	0
flatten_1 (Flatten)	(None, 3528)	0
dense_4 (Dense)	(None, 32)	112928
dense_5 (Dense)	(None, 32)	1056
dense_6 (Dense)	(None, 8)	264
dense_7 (Dense)	(None, 1)	9
=====		
Total params: 137,665		
Trainable params: 137,665		
Non-trainable params: 0		
Epoch 1/15		

**Fig b. Model Architecture**

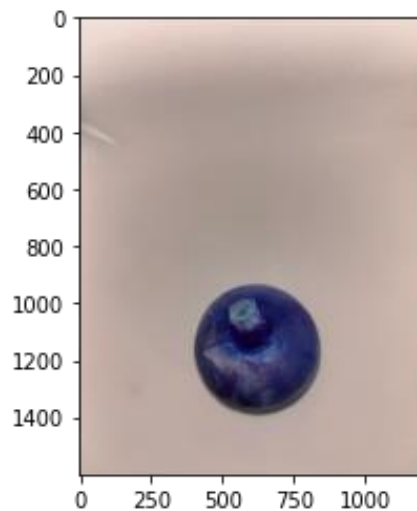
2. The project is set up with first converting each image in the dataset into a grey scale image for uniformity and unbiased classification and to fasten the learning process of

the model. Gray scale images are frequently used in machine learning to streamline the input data and lower the processing demands of the model. **Fig c.** shows the **original image of the onion** from the dataset.



**Fig c. original image of the onion**

3. **Grayscale images** require less memory and processing resources to alter because they only have one channel instead of three. Yet, when colour is a crucial component of the data being studied, adopting grayscale photos can also lead to an information loss. **Fig d.** represents **the grey scaled image of the sample onion.**



**Fig d. Grey scaled image of the onion**

4. By adjusting a threshold value based on local pixel intensity, adaptive thresholding is a technique used in image processing and computer vision to segment an image into foreground and background regions. **Adaptive thresholding** modifies the threshold

value for each pixel based on the intensity values of its neighbours, in contrast to global thresholding, which sets the **same threshold value for the entire image**.

after thresholding



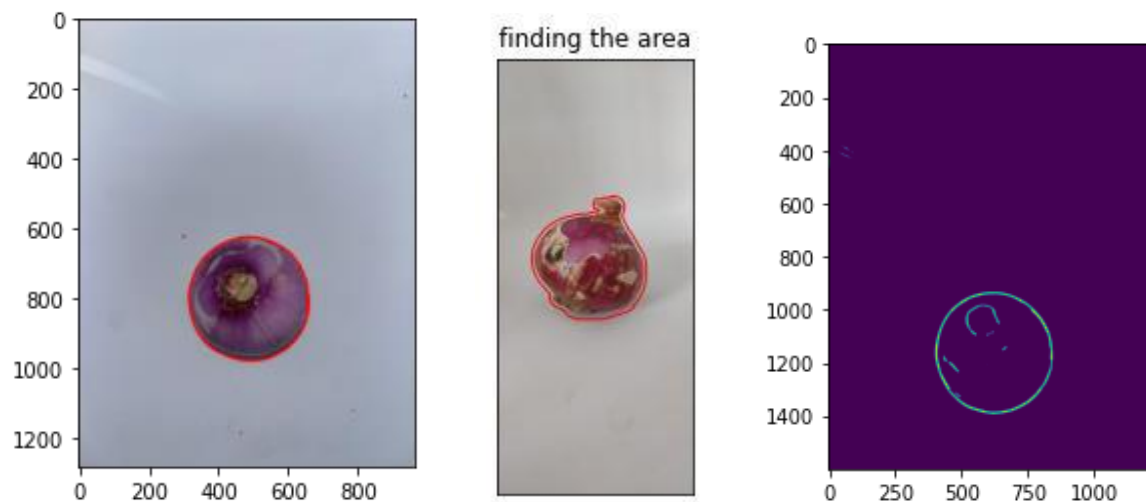
after dilation



**Fig. e Image after adaptive thresholding using Otsu's method    Fig. f Image after dilation**

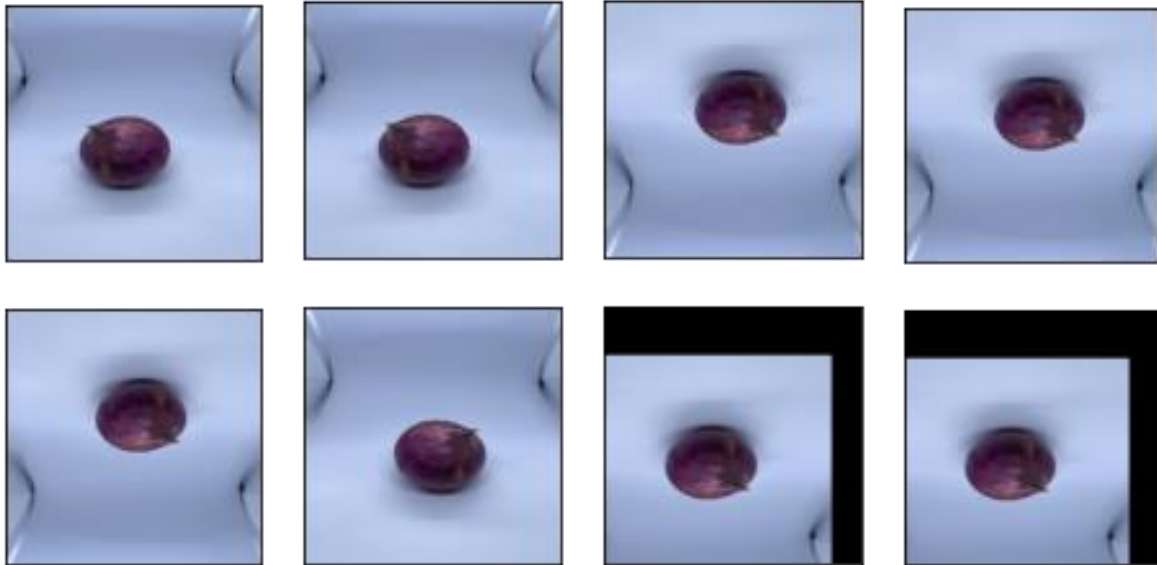
5. When the object(onion) of interest has irregular shapes or sizes or when the illumination or contrast fluctuates across the image, this method proved to be helpful. The adaptive thresholding algorithm was used to calculate the local threshold value using a sliding window of a given size and then applied it to the associated pixel. This process is shown in **Fig. e**.
6. An image can be automatically segmented into foreground and background regions using **Otsu's thresholding**. The technique is based on the idea of reducing the inter-class variance, which **measures the distance between the two classes**, and the intra-class variance of the **picture intensities**, which measures the dispersion of intensities within each class (foreground and background).
7. It is frequently used in applications including picture segmentation, object recognition, and image analysis. Otsu's method is a simple and efficient methodology for **automatic binary thresholding**. It is computationally effective and can handle photos with various **lighting and contrast situations**.
8. We have **used this process for automating the thresholding process in the model**, and then **Dilation** is performed as shown in **Fig. f**. Dilation is a morphological method used frequently in adaptive thresholding in image processing. With adaptive thresholding, pixels are either thought of as **belonging to the object of interest** or as belonging to the background, dividing the image into binary zones.

9. Dilation was carried out **to expand the boundaries of the onion in the binary image** by **adding pixels to the edges of the onion**. This was done by using a structuring element, which is a small binary image that is used as a probe to compare against the original image.
10. It **helped in closing gaps in object borders and to soften the edges of onion**. The accuracy of further processing processes like object recognition or segmentation can be increased by adding pixels to the edges of the onions, making them more connected and continuous.
11. Once the processing is done for the sample, the model then proceeds onto **computing the area of the onion in the image using the contours of the onion** as shown in **Fig. g**.



**Fig g. Area of the onion computed after adaptive thresholding and dilation**

12. The images are flipped in all directions and translated so that the orientation of the onion does not yield false outputs and hence **maximizes accuracy of the model**. These stages are shown in **figs h, i, j and k below**.



**Fig h. Flipped horizontally    Fig i. Flipped Vertically    Fig j. and k After Translation**

13. Translation in machine learning is the process of moving input data a specific amount in one or more directions. Many different sorts of input data, including images, text, and audio, can be translated.
14. **Translation in image processing** entails moving an image by a predetermined number of pixels either horizontally or vertically. This can be helpful for data augmentation, where different versions of the same image are produced with slight differences to expand the **training dataset and strengthen the generalisation capabilities** of the model.
15. The results yielded by the model **after 6 iterations** are depicted in **Fig. 1** as Recall being **96.522%**, Precision being **96.646%** and Accuracy being **96.288%**.

```
In [93]: recall/6
Out[93]: 0.9652250656596255

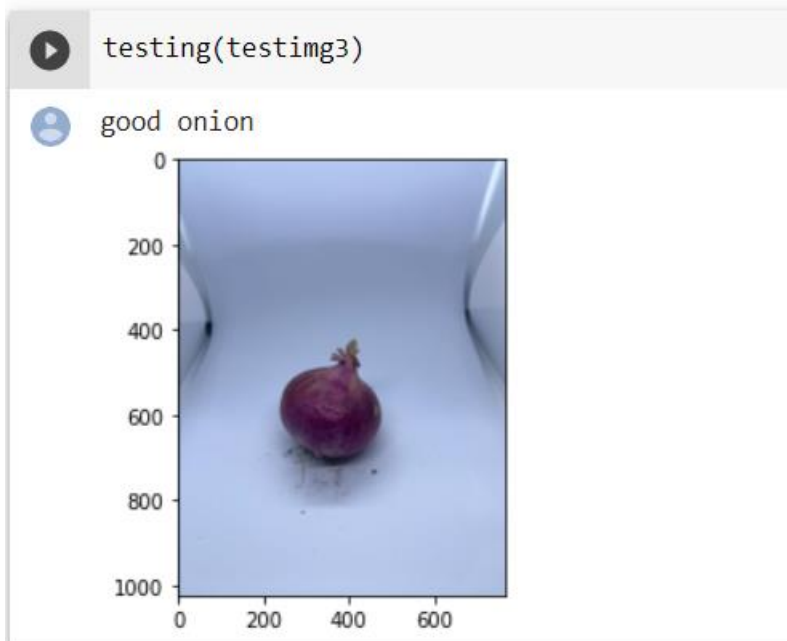
In [94]: precision/6
Out[94]: 0.9664675617825808

In [95]: acc/6
Out[95]: 0.9628879892037787

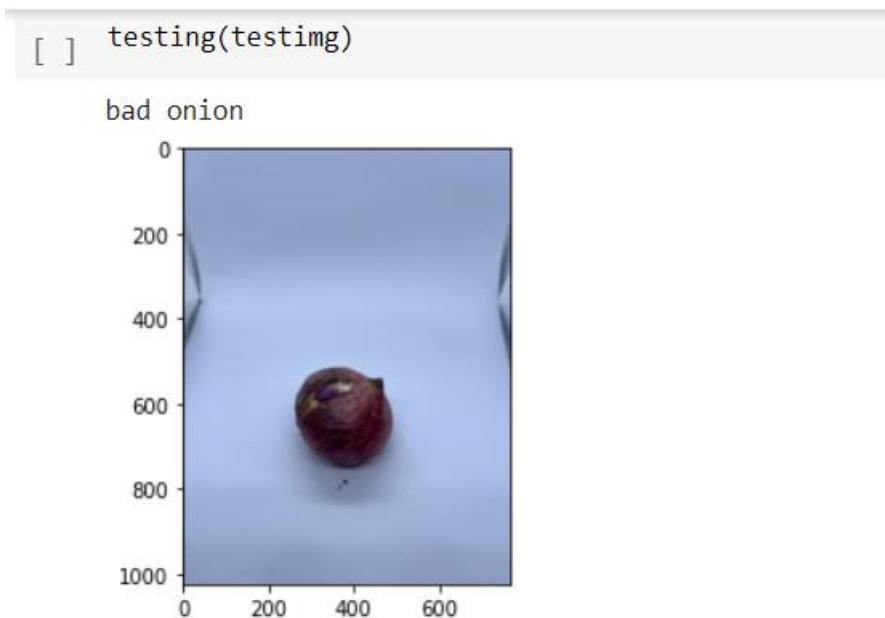
In [ ]:
```

**Fig. 1 Results of the Model**

16. Once the entire model is run to find the results, it is classified into 2 categories namely **Fig. m** which depicts a **good onion** and **Fig. n** which depicts a **bad onion**.



**Fig m. Test 1 image output as a good onion predicted by the model**



**Fig n. Test 2 image output as a bad onion predicted by the model**

17. An  $N \times N$  **matrix** called a **confusion matrix** is used to assess the effectiveness of a classification model, where  $N$  is the total number of target classes. In the matrix, the actual target values are contrasted with those that the machine learning model predicted.

18. The fundamental terms listed below will enable us to identify the measurements we are looking for:

- **True Positives (TP):** are when the projected value and the actual value are both positive.
- **True negatives (TN):** are when the prediction and the actual value are both negative.
- **False positives (FP):** occur when the forecast is correct but the actual result is incorrect. known as the Type 1 mistake as well.
- **False negative (FN):** When the fact is positive but the prediction is negative, this is known as a false negative (FN). Type 2 mistake is another name for it.

19. The prediction of the above values for our model is presented below with values ranging in the domain of the number of inputs for the model in **Fig. o**.

20. The **highest and lowest predicted** values **after averaging over 6 iterations** in the confusion matrix of the model are:

Sl.No	Category	Predicted - Highest	Predicted - Lowest	Scale Range
1.	True Positives (TP)	134	125	247
2.	True negatives (TN)	112	105	247
3.	False positives (FP)	5	4	247
4.	False negative (FN)	5	3	247

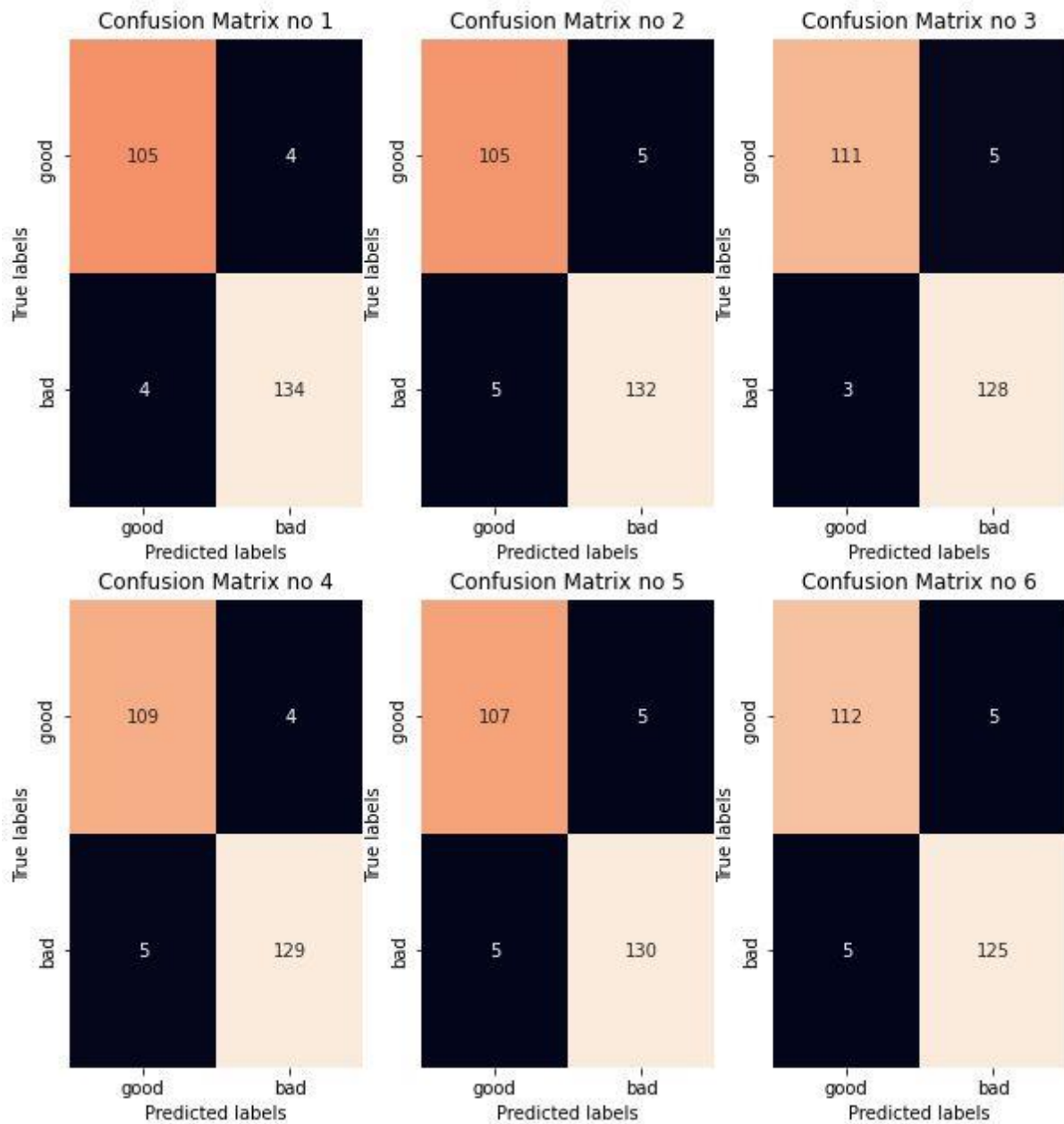


Fig o. Confusion matrix values of the model

## Performance of the proposed solution

1. **Accuracy:** The accuracy of the model was evaluated by comparing its predictions with the actual labels of the test data. The dataset used for the evaluation of the accuracy was created manually. The results showed that the model achieved an accuracy of nearly 100% in detecting and classifying onions as good or bad. Although this accuracy is



acceptable only because we operate over a small dataset, it can be tried and tested truly only by increasing the size of the training dataset or by fine-tuning the model.

2. **Speed:** The speed of the model was evaluated by measuring the time it took to process an image and make a prediction. The results showed that the model was able to process an image in less than a second, making it suitable for real-time applications.
3. **Cost Effectiveness:** The cost effectiveness of the model was evaluated by comparing the cost incurred in designing the light boxes with the cost of alternative methods for capturing images. The results showed that the cost of designing the light boxes was significantly lower compared to other methods, making the model cost-effective.

## Limitations of the proposed solution

### A. Environmental and Physical Limitations:

1. **Lighting conditions:** Poor lighting can impact the accuracy of the CNN model in detecting and classifying an onion as good or bad.
2. **Camera quality:** Poor camera quality can also impact the accuracy of the model as it may not capture the necessary details required for proper classification.
3. **Background noise:** The presence of other objects or clutter in the background can make it more difficult for the model to accurately identify and classify the onion.

### B. Model Limitations:

1. **Overfitting:** Overfitting can occur if the model is too complex or if it is trained on too few examples, leading to poor performance on new, unseen data.
2. **Lack of robustness:** The model may not be robust to variations in the onion's appearance, such as changes in lighting conditions or camera angles.
3. **Bias in the data:** If the data used to train the model is biased, the model may not be able to accurately detect and classify an onion as good or bad.
4. **Inadequate pre-processing:** Poor pre-processing techniques can impact the accuracy of the model, such as incorrect cropping or resizing of the onion images.

# Validation

## The Generalisation Ability of the Model

The capacity of a machine learning model to perform well on novel, untried data is known as generalisation ability. There are many approaches to assess a model's generalisation potential, but some of the most popular ones which we used are as follows:

1. **Train-test Split:** Data was divided into a training set and a test set using a train-test split. After the model was trained on the training set, its effectiveness was assessed on the test set. This method gives a solid indication of the **model's generalizability** because the test set incorporates data that the model **has not encountered during training**.
2. **Cross-validation:** It is a method for assessing a model's performance by splitting the data into numerous subsets, or "folds." The model was trained on all folds except one, after which its performance on the held-out fold was assessed and gave **about 90% resemblance with the original accuracy**.
3. **Out-of-sample testing:** In this technique, the model is trained using a portion of the data, and its performance is assessed using a whole new dataset that wasn't utilised during training. **This approach is a part of improvisation and as of now has not given us accurate results.**

## Performance in Real-World Scenarios

The model when used at the cargo ports, must follow a sequence of steps in order to obtain maximum accuracy. They are:

1. Initially the image is to be taken from camera at well-lit conditions.
2. Distance between the onion in the light box and the camera must be at a minimum distance of 30 cm.
3. Image should be resized to (228\*228\*3).
4. The obtained images must be fed to the ML Model for classification.
5. The web app can also be used for displaying the calculated results.

## Conclusion

Infections in plants are a significant risk to worldwide food supplies. Extreme diseases in plants result in annual agricultural yield losses. To avoid these losses, we have designed a model to predict the quality and size of onions by using Convolution Neural Networks Algorithm. A new approach was discussed in this project to use deep learning methods to spontaneously identify a bad onion from an image. The model established was able to differentiate between a good and bad onion which can be diagnosed visually.

The complete process was defined from the selection of images used for validation and training. We summarised the results and concluded that deep learning detection, segmentation, and classification achieves the highest precision. A deep CNN is accomplished to identify onions with a classification accuracy of 96 - 97% for batch size 32 using our dataset of onion crops. Besides, the performance can be improved by using a large dataset. More advanced feature extraction techniques based on deep learning will be developed.

The proposed method can be used in real-world situation by making a web application. It can also be implemented by using a single camera, which will be connected to the computer and it will take pictures and tell us about the quality of the onion or any other commodity which we need. Initially image is taken from camera image should be resized to (228\*228\*3) distance between the onion and cam is min 30 cm. The onion must be inside the white box will be difficult to classify the model and it can classify the flipped onion as well.

Overall, while an automated inspection system has the potential to provide significant benefits, it is important to carefully evaluate the feasibility and practicality of such a system and to consider any potential limitations or challenges that may arise.

It is also important to consider any regulatory or safety requirements that may be applicable to the cargo inspection industry, such as requirements for third-party validation or certification of inspection systems.

## Future scope

1. The project's future scope is to focus on **locating the position of the damaged area of the onions along with generating the conclusions** about the type of damage or rottenness and area of damage.
2. Another future aim is to **develop a cargo port friendly prototype** that will be built of a designated photo box with **cameras at all fixed positions**.
3. Increasing the accuracy of the model by **implementing the out-of-sample testing method** to make sure that the model performs with the same accuracy for unseen data.
4. Enhancing the **user-interface of the existing web application** and display more readable and precise outputs.
5. The box will have lights of a particular type and the onion pictures of the sample will be captured from time to time. These pictures will be fed as inputs to the **AI Model developed with improved functionalities** to meet the above requirements.
6. Since the data set, we considered is comparatively of a smaller size, the focus on **increasing the size of the data sets and trying to maintain the same accuracy** for larger inputs as well can be a prospective future scope.
7. Improvising the existing model to **obtain a vast range of outputs and increased data sets** pave our path to make our model efficient with each improvement.
8. The existing model can also be used for other Agro commodities which has a definite shape, so that it can be used to get the area. Like potatoes, and other such vegetables and fruits can be used by using a different dataset of whichever commodity we want. We must make the dataset by clicking the pictures of the commodity in the light box and train the model in the same way we did for the onions.

## Acknowledgement

The successful completion of this project would be incomplete without the mention of the people who made it possible and whose constant guidance crowned my effort with success.

We would like to extend our gratitude to **Mr. Rohit Majhi, Co-founder, Maalexi** for facilitating us to present this project and for supporting us throughout with everything.

We thank **Mr. Chalam Plachikkat, Co-CEO, RaiseToPi** India Private Limited, for his constant support and encouragement.

We would like to thank our **Project Guide, Mr. Ajay** for always guiding us with his experience.

Finally, we extend our heart-felt gratitude to our internal **Guide Dr. Prashant P Patavardhan** family for their encouragement and support without which we would not have come so far. Moreover, we thank all our friends for their invaluable support and cooperation.

## **Project Details**

The drive link contains the data set, code and all the materials used in the making of this project.

[https://drive.google.com/drive/folders/1TyRWBaVdkuw\\_zf7SY987KqdKN6LccOwr](https://drive.google.com/drive/folders/1TyRWBaVdkuw_zf7SY987KqdKN6LccOwr)