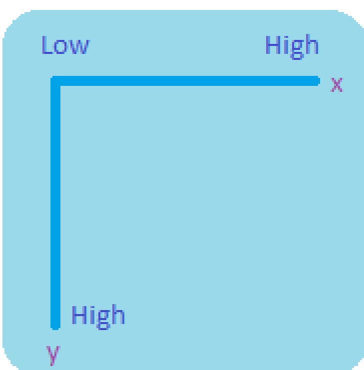
## Opencv:

Opencv was extensively used in the project starting from taking in the video feed from the turtle bot camera and using frame by frame images of the while loop to first resize and reshape to make it compatible with CNN's input layer to classify which object does the image contain to further use Opencv to call Custom build Cascades previously built using opencv and Haarcascade builder GUI to use opencv's *detectMultiScale() function, being the most important function here as it scans the areas and tells us which object is present from the classes and where it is present in the frame. (Coordinate system in Opencv is as follows)*

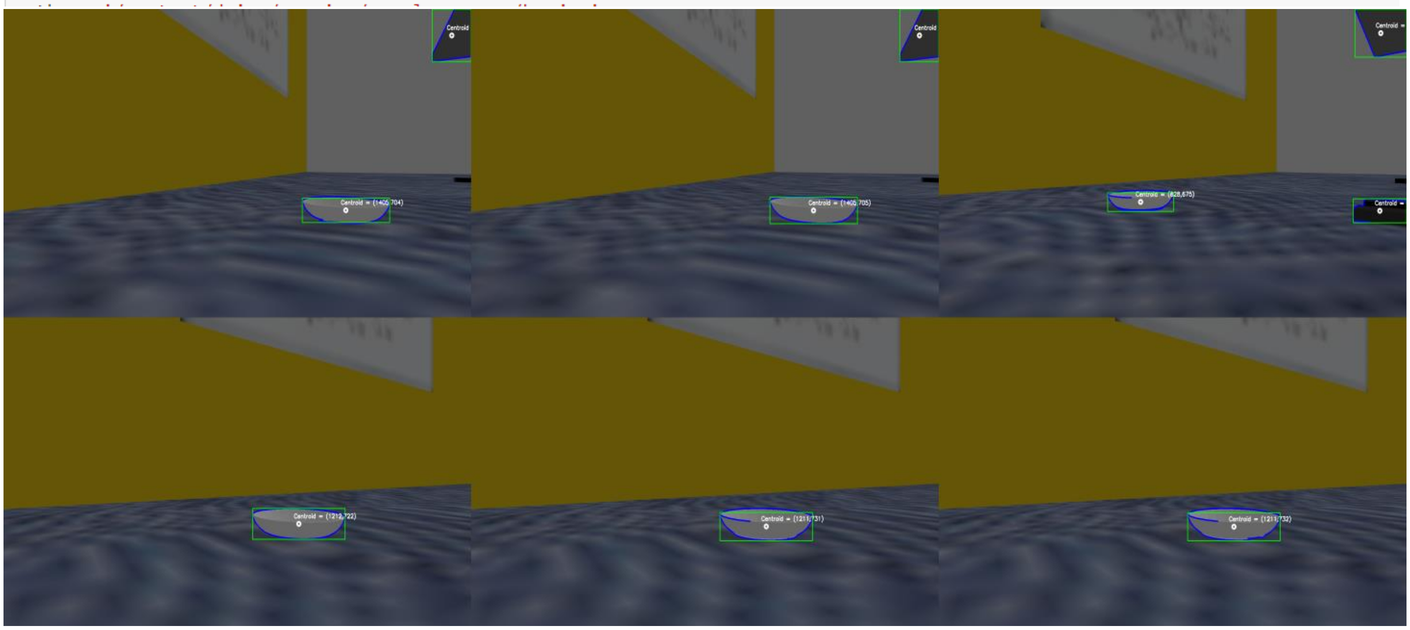



(Where the sign rule is that +ve x axis pertains to right and +ve y axis pertains to down the sheet) the function gives us the x and y coordinate of the top left point of the area detected as well as the width and height of the rectangular area (in that order, x,y,w,h)

Now if both the layers of detection of the object fail assuming the CNN classifies the object incorrectly and so does the haarcascade (cause as we know the haarcascade is known to classify the Mars Rover and Quadcopter as the same object or detecting an object that’s not there by the CNN) then we use the tracker function in Opencv along with the bounding box option and taking the video feed from the camera and increasing brightness and save the images in a folder to create a dataset to retrain the model to increase its accuracy.

We also used Opencv to calculate the centroid of each object by using contour detection and Also very extensively by drawing on the images.

```python
def getContours(img,imgContour,i):
    print("Image"+str(i)+"\n\n")
    print("Image Shape: ",img.shape)


    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray,(7,7),1)
    imgCanny = cv2.Canny(imgBlur,50,50)
    img=imgCanny
    contours,hierarchy = cv2.findContours(img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area>20:
            cv2.drawContours(imgContour, cnt, -1, (255, 0, 0), 3)
            peri = cv2.arcLength(cnt,True)
            approx = cv2.approxPolyDP(cnt,0.02*peri,True)
            objCor = len(approx)
            x, y, w, h = cv2.boundingRect(approx)
            cv2.circle(imgContour,(int(x+w/2),int(y+h/2)),7,(255,255,255),5)
            text="Centroid = ("+str(int(x+w/2))+","+str(int(y+h/2))+")"
            print(text)
            cv2.putText(imgContour, text, (int(x+w/2+-20),int(y+h/2+-20)), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)



            cv2.rectangle(imgContour,(x,y),(x+w,y+h),(0,255,0),2)
    print("\n\n\n")
```



## Machine learning framework:

In this project what we use concepts of Semi Supervided Learning and Reinforcement learning to strengthen our models understanding of the object we are feeding into it.

As a start we started to observe the classes that should be in our Neural Network as in all the objects that we are going to need to be detected which we settled on to be:

1. Mars Rover
2. Quadcopter
3. Bowl
4. Boxes

So first things first we collected a lot of images from our virtual environment and manually labelled to make our training set it was only a set 60-70 images that has very less variation but seemed appropriate for the goal we were trying to achieve in unsupervised learning.

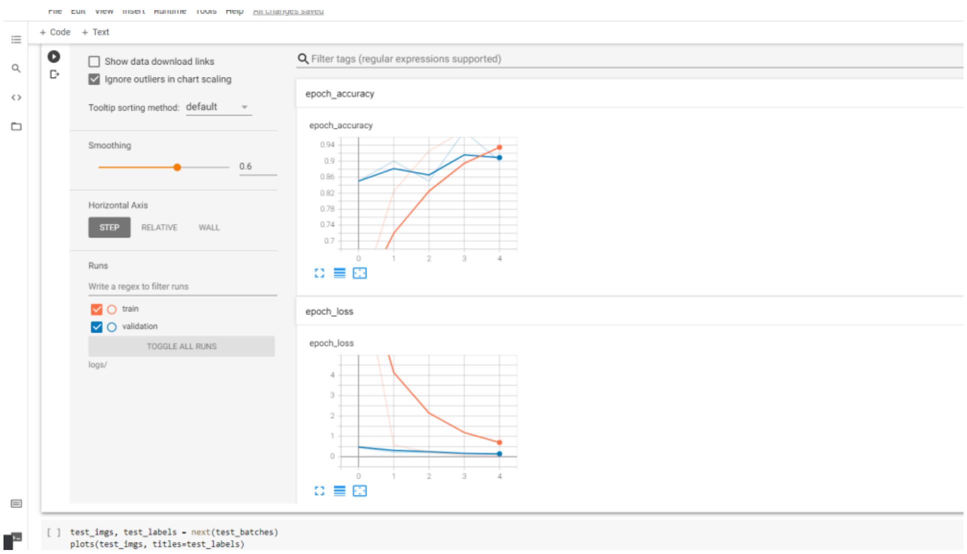
We then started making our custom CNN which could classify which object is present in our frame. So after a lot of trial and error with neural network architecture (where we first made a model too complex for the images which saw massive overfitting and further we reduced the complexity so that there is just enough overfitting that can be countered by adding more varied data later by our scanner which I will explain further). So once we got the 96% accuracy on our training data we stopped.

In [ ]:

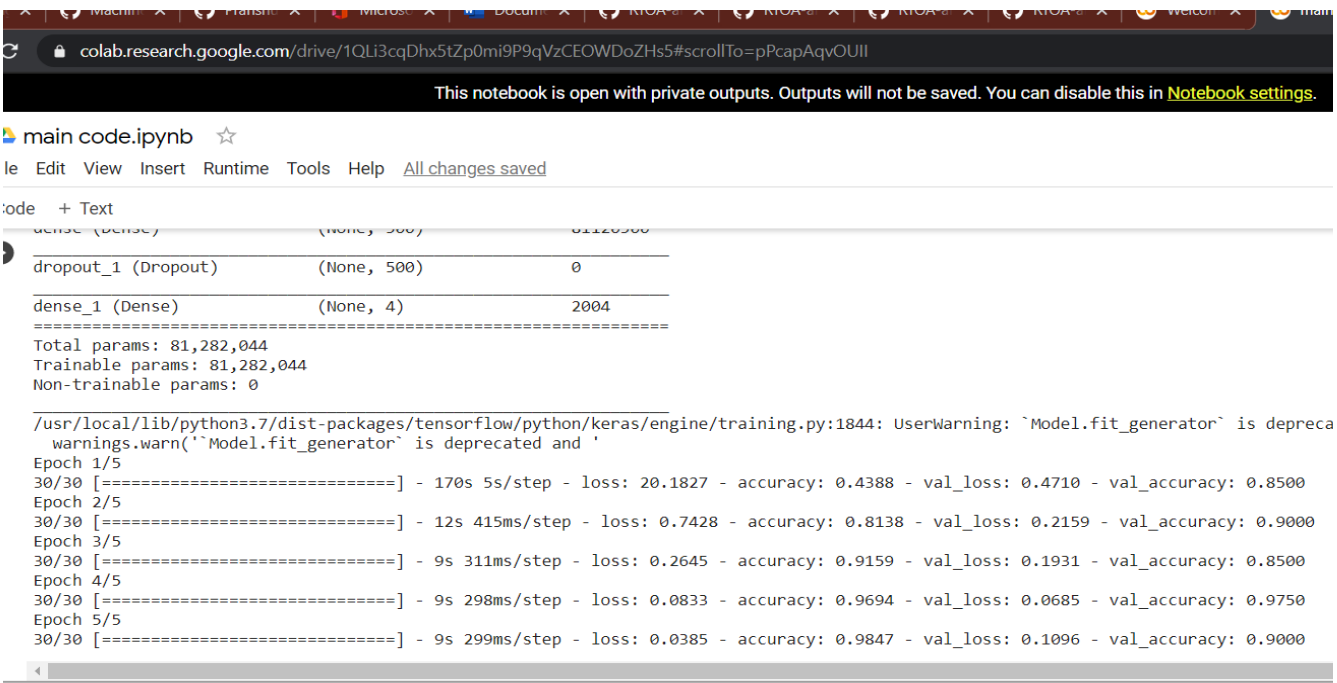
```python
batch_size_val=50
steps_per_epoch_val=2000
epochs_val=30

def myModel():
    no_of_Filters=60
    size_of_Filter=(5,5)
    size_of_Filter2=(3,3)
    size_of_pool=(2,2)
    no_of_Nodes=500
    model=Sequential()
    model.add((Conv2D(no_of_Filters,size_of_Filter,input_shape=(224,224,3),activation='relu')))
    model.add((Conv2D(no_of_Filters,size_of_Filter,activation='relu')))
    model.add(MaxPool2D(pool_size=size_of_pool))

    model.add((Conv2D(no_of_Filters,size_of_Filter2,activation='relu')))
    model.add((Conv2D(no_of_Filters,size_of_Filter2,activation='relu')))
    model.add(MaxPool2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(no_of_Nodes,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4,activation='softmax'))

    model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 220, 220, 60)      4560
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 216, 216, 60)      90060
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 108, 108, 60)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 106, 106, 60)      32460
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 104, 104, 60)      32460
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 52, 52, 60)        0
_________________________________________________________________
dropout (Dropout)            (None, 52, 52, 60)        0
_________________________________________________________________
flatten (Flatten)            (None, 162240)            0
_________________________________________________________________
dense (Dense)                (None, 500)               81120500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2004
=================================================================
Total params: 81,282,044
Trainable params: 81,282,044
Non-trainable params: 0
_________________________________________________________________
```

Model and its Summary




At Every trial and error situation we used Google Colab's free GPU and Tensorboard to analyze the strength of our model



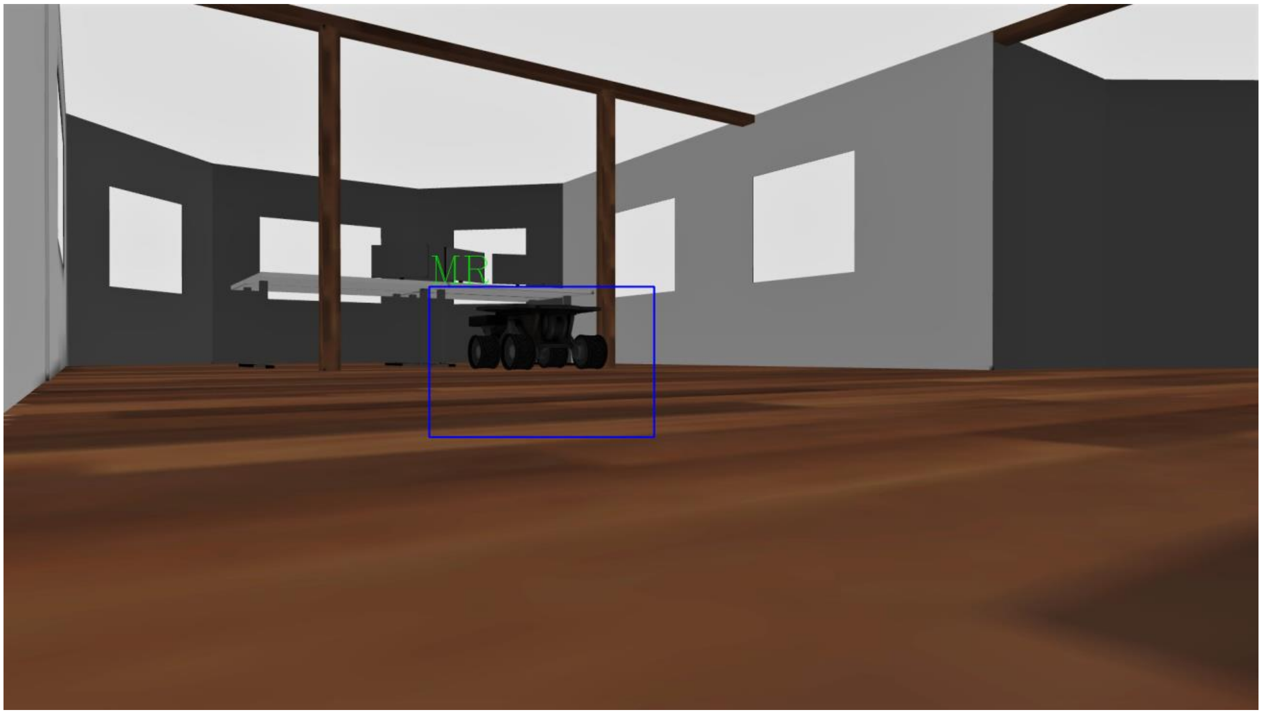

```
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2004
=================================================================
Total params: 81,282,044
Trainable params: 81,282,044
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is depreca
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/5
30/30 [==============================] - 170s 5s/step - loss: 20.1827 - accuracy: 0.4388 - val_loss: 0.4710 - val_accuracy: 0.8500
Epoch 2/5
30/30 [==============================] - 12s 415ms/step - loss: 0.7428 - accuracy: 0.8138 - val_loss: 0.2159 - val_accuracy: 0.9000
Epoch 3/5
30/30 [==============================] - 9s 311ms/step - loss: 0.2645 - accuracy: 0.9159 - val_loss: 0.1931 - val_accuracy: 0.8500
Epoch 4/5
30/30 [==============================] - 9s 298ms/step - loss: 0.0833 - accuracy: 0.9694 - val_loss: 0.0685 - val_accuracy: 0.9750
Epoch 5/5
30/30 [==============================] - 9s 299ms/step - loss: 0.0385 - accuracy: 0.9847 - val_loss: 0.1096 - val_accuracy: 0.9000
```

^^^Accuracy with each epoch

Our next job was the predictions, which happened in the RTOA section of the code where once after importing our libraries we imported our Haarcascades

```python
#IMPORTING ALL THE PRETRAINED HAARCASCADES
bowl_cascade = cv2.CascadeClassifier('/content/drive/MyDrive/Haarcascades/bowl_cascade.xml')
MR_cascade = cv2.CascadeClassifier('/content/drive/MyDrive/Haarcascades/MR_cascade.xml')
QC_cascade = cv2.CascadeClassifier('/content/drive/MyDrive/Haarcascades/QC_cascade.xml')
wheel_cascade = cv2.CascadeClassifier('/content/drive/MyDrive/Haarcascades/Wheel_cascade.xml')

#LOADING THE PRETRAINED MODEL
from tensorflow.keras.models import load_model
model= load_model('/content/drive/MyDrive/models/CNN for RTOA.h5')
```

and first ran the predictions from our CNN using model.predict() function after importing the model architecture and saved weights that we did earlier in the “Main code” of the CNN and classified which object is present in the frame or if there is none, further after finding out which object there is we run the that specific object’s Haarcascade to for us to return the coordinates of the top left corner of the object area and it’s weight and height as mentioned in the Opencv part of this report, and we return the x,y,w,h and the object code of the individual object from the function.




Now assuming the CNN fails to identify the object (it may show none, or a different object due the limited capability of ours to collect large amounts of data and manually label it Thus the model has not been trained on Varied data) so the code asks us if we are satisfied with the predictions or not.

And if we input ‘n’ that means we are not and it gives us an option to create a bounding box around the object we think is the object and the Turtle bot takes all the images from the camera feed and puts it in a folder to create a new data which it inputs in the CNN to update the weights and strengthen the CNN’s accuracy. And further save the model in it’s place to later be called.

]:
```
#SIMPLE CODE TO DRAW BOX AT THE REQUIRED AREA
def drawBox(img,bbox):
    x, y, w, h = int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3])
    cv2.rectangle(img, (x, y), ((x + w), (y + h)), (255, 0, 255), 3, 3 )
    cv2.putText(img, "Tracking", (100, 75), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

myPath = 'data/images/train'
cameraNo = 0
cameraBrightness = 180
moduleVal = 10  # SAVE EVERY ITH FRAME TO AVOID REPETITION
minBlur = 500  # SMALLER VALUE MEANS MORE BLURRINESS PRESENT
grayImage = False # IMAGES SAVED COLORED OR GRAY
saveData = True   # SAVE DATA FLAG
showImage = True  # IMAGE DISPLAY FLAG
imgWidth = 224
imgHeight = 224
areamin=224*224
status=1

val= input('are you satisfied with detection (y/n)?')
#ASKING USER IF THE TWO LAYERS OF SECURITY WORKED WELL...IF THEY DIDN'T WE NEED TO MANUALLY ADD NEWER IMAGES OF THE OBJECT
#AS WELL AS COMPLETE THE TASK
tracker = cv2.TrackerMOSSE_create()
#PRE-BUILT TRACKER PROVIDED BY OPENCV
global countFolder
cap = cv2.VideoCapture(cameraNo)
cap.set(3, 640)
cap.set(4, 480)
cap.set(10,cameraBrightness)


count = 0
```

Here we are writing the prerequires for our code ahead, including a function to draw a box using the parameters that might be given to it, our path of our train data folder where we want it to save the data for training further

Our camera number, brightness, moduleVal to take every 10 frame and not all, setting a blur value, and most of all asking “are you satisfied with the detection:” and going ahead only on “n” input.

Followed by importing the aforementioned tracker and starting to get the video feed....**Over here to isolate just the Image Processing and Machine Learning part of the code it is being run in colab and not in the ROS Environment**

```
    #if count % moduleval ==0 and blur > minBlur:
    #if 4*w*h>areamin:

    if success:
        status=1
    else:
        status=0

    if status is 1:
        #SAVING THE IMAGE FOR TRAINIGNG THE MODEL AGAIN IF STATUS OF SUCCESS IS 1 AS IN IF SUCCESSFUL
        nowTime = time.time()
        cv2.imwrite(myPath+'/'+class_index+'/'+'img'+str(countSave)+"_"+"_"+str(nowTime)+".png", img)
        countSave+=1
    #count += 1

    if success:
        drawBox(img,bbox)
        status=1
    else:
        cv2.putText(img, "Lost", (100, 75), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        status=0


    cv2.imshow("Tracking", img)
    if cv2.waitKey(1) & 0xff == ord('q'):
        break
```

Saving the file^^^^

```
train_batches=ImageDataGenerator().flow_from_directory(train_path, target_size=(224,224), classes=['0','1','2','3'], batch_size=10)
valid_batches=ImageDataGenerator().flow_from_directory(valid_path, target_size=(224,224), classes=['0','1','2','3'], batch_size=10)
```

```
Found 107 images belonging to 4 classes.
Found 0 images belonging to 4 classes.
```

```
def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_batches)
plots(imgs, titles=labels)
print(imgs.shape,'\n')
```

```
(10, 224, 224, 3)
```

^^^Data Augmenting our current newly made data set and plotting it out to confirm

In [ ]:
```
batch_size_val=50
steps_per_epoch_val=2000
epochs_val=30
imageDimensions=(32,32,3)
#CALLING MODEL
from tensorflow.keras.models import load_model
model= load_model('models/CNN for RTOA.h5')
```

In [ ]:
```
model.summary()
history=model.fit_generator(train_batches, steps_per_epoch=30, validation_data=valid_batches, validation_steps=4, epochs=10, verbose=1)
#bhai steps per epoch number of images me 10 divide krke likhna hoga but count nhi ho paa rhe
```

In [ ]:
```
test_imgs, test_labels = next(test_batches)
plots(test_imgs, titles=test_labels)
```

In [ ]:
```
#INCASE OF NO MODEL WE ADD A MODEL OR REPLACE IT
import os.path
if os.path.isfile('/content/drive/MyDrive/models/CNN for RTOA.h5') is False:
    model.save('/content/drive/MyDrive/models/CNN for RTOA.h5')
else:
    os.remove('/content/drive/MyDrive/models/CNN for RTOA.h5')
    model.save('/content/drive/MyDrive/models/CNN for RTOA_updated.h5')
```

Updating the model further on our new data and saving the model in its original place and deleting the previous one