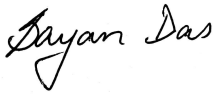
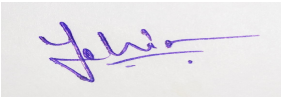
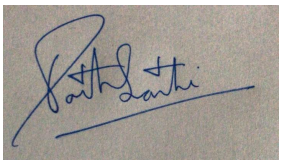
# CS/RBE 549 Computer Vision, Fall 2021
# Project Report

## Team 12: Please Mask My Leprechaun!!

| Member | Signature | Contribution (%) |
|---|---|---|
| Wael Mohammed |  | 25% |
| Sayan Das |  | 25% |
| Febin Fredi |  | 25% |
| Parthsarthi Rawat |  | 25% |

| Grading: | | |
|---|---|---|
| | Approach | _____/15 |
| | Justification | _____/5 |
| | Analysis | _____/15 |
| | Testing & Examples | _____/15 |
| | Documentation | _____/10 |
| | Difficulty | _____/10 |
| | Professionalism | _____/10 |
| | Presentation | _____/20 |
| | Total | _____/100 |

# TABLE OF CONTENTS

# LIST OF FIGURES

## Executive Summary

The aim of this project is to identify and differentiate between the different instances of a class present in an image, namely execute instance segmentation. For this project, we have worked to perform instance segmentation on stuffed toys present in an image and specifically on the leprechaun plushie. The motivation for this is the fact that we received the leprechaun toy in the class to detect and identify. We try to solve this challenge using a traditional method as well as a state of the art method using deep convolutional neural networks.

We begin our traditional approach by first implementing object detection. We detect keypoints in the image with ORB (Oriented FAST and Rotated BRIEF) and match them with a template image. Unfortunately, we find that the matching performed is not robust enough to create a scale and rotation invariant bounding box around the leprechaun. Instead, we calculate the median of the matched keypoints which is generally located on the face of the leprechaun. This gives us a good estimate of where the leprechaun is located in the picture. Next, we try to make our algorithm scale invariant. To that end, we try to find a circle approximating the face of the leprechaun whose radius we can use as a reference to modify the size of the bounding box. We first apply Canny Filter on the Gaussian Filtered image followed by Hough Transform to detect the circles in the image. We then find the circle approximating the face by selecting the circle that is closest to the median. Scaling the size of our bounding box in proportion to the radius of this circle makes our algorithm scale invariant. Now, we focus on making our algorithm rotationally invariant as well. We apply PCA (Principal Component Analysis) within a box centered on the center of the circle we just found. Since the face of the leprechaun is symmetric, one of the eigenvectors found points towards the general upward direction of the leprechaun and we use this angle to orient our bounding box according to the tilt of the leprechaun. For segmentation, we apply HSV thresholding inside the bounding box we have obtained by estimating different HSV masks corresponding to the face, body and beard of the leprechaun in our template image. Our resulting rotation and scale invariant segmentation algorithm is able to successfully segment the leprechaun in different test images successfully. This implies that our algorithm is capable of semantic segmentation. However, we do not extend our algorithm to instance segmentation since we found that it is very difficult to generalize multiple members of a class without using neural networks.

We also implement a state-of-the-art model for instance segmentation of our Leprechaun, namely Mask R-CNN. Here, Mask R-CNN uses Faster RCNN for object detection and FCN (Fully Convolutional Networks) for Segmentation. We annotate members of the class ‘Plushies’ consisting of the leprechaun and a few Pokemons. We utilized Transfer Learning and trained the last 2 layers of the Mask RCNN model with 120 self-annotated images for 200 epochs. The resulting model is able to successfully perform instance segmentation on images and real time videos.

## Abstract

The aim of this project is to identify and differentiate between the different instances of a class present in an image and video. For this project, we have worked to perform instance segmentation on stuffed toys present in an image and specifically on the leprechaun plushie. The motivation for this is the fact that we received the leprechaun toy in the class to detect and identify.

We have tried to solve this challenge using two approaches, namely, a traditional method and a state of the art method using deep convolutional neural networks. The reasoning between choosing these two contrasting approaches was to better understand the benefits and drawbacks between these two methods and apply the different concepts and methodologies taught in class.

## Introduction

In order to start discussing instance segmentation, we need to first discuss semantic segmentation as instance segmentation is an improvement over semantic segmentation. In semantic segmentation, we classify which class each pixel belongs to. This way, we are able to separate the foreground and objects of interest from the background by evaluating and finding exact boundaries in the images. The problem of image recognition and object detection is solved in semantic segmentation but it fails to differentiate between objects belonging to the same class.

For example, consider an image of 3 stuffed toys of pokemon, semantic segmentation would be able to tell the location and exact boundaries of the pokemons but it would classify them all as stuffed toys but wouldn't be able to tell them apart. Although the toys are distinct, the algorithm fails to recognize this feature. To solve this issue, instance segmentation can be used. It would also classify them as stuffed toys but it would realize that these toys are different instances of the class of stuffed toys. Thus toy 1 would be differentiated from toy 2 and toy 3 and would have separate boundaries. Instance segmentation can be thought of as a combination of object detection and semantic segmentation.

The main advantage of using instance segmentation is the presence of more knowledge about the object in the images and their properties. The count and properties of different instances of various classes could be utilized to better decision making capabilities. An example could be taken in the manufacturing industry, where a certain operation of lifting the desired objects and placing them somewhere else is only when their quantity is a dozen. Another use case could be in autonomous vehicles to detect the number of  different humans crossing a road and taking into account their different speeds while deciding when to start moving ahead. Most of the work being done in this domain, if not all, leverages Convolution Neural Networks to solve this challenge effectively and robustly.

**Our Leprechaun**

## Methodology

We have tried to solve this problem using two different types of approaches. In the first approach, we are using computer vision techniques that do not employ neural networks at all and are calling it the traditional approach. In the second approach, we go ahead and use the modern approach of training convolutional neural networks to identify and classify the plushies.

In our traditional approach, we start by detecting features in the query or reference image and the train image and match them to find features on the final train image. A bunch of computer vision techniques are performed to compensate for the short-comings of the feature detection algorithms and then we draw a bounding box around the object which is scale and rotation invariant thus giving us the location of the object within the image. Then we perform thresholding within the bounding box to segment our object of interest from the background.

In the state of the art approach, we use the Mask R-CNN model to train it to classify and detect the leprechaun and other plushies by providing it a labeled data set using transfer learning.

Finally, a comparison between the two methodologies is done to see how both of them perform in different environments and conditions.

# Traditional Approach

The following paragraphs discuss the techniques we used in their order and the reasoning behind using them.

## Object Detection

The first and foremost step we did was to perform feature detection in order to recognize our object, i.e. the leprechaun toy, and localize it in the image by drawing a bounding box. To do this, we took an image of the leprechaun on a plain cardboard background, in a well lit room, in order for the feature detection algorithms to easily detect keypoints on the stuffed toy. We made sure not to use a white background as the color of the face and hands of the leprechaun toy was whitish too. The image was taken such that the focus was on the toy and the background was as less as possible so as to avoid features being detected from the background instead of the toy.

We read about the different types of feature detection algorithms and tried out SIFT, SURF and ORB .We used the SIFT (Scale Invariant Feature Transform) algorithm first by downloading the opencv-contrib module. Initially, we implemented the difference of gaussians on our object, which is the first step of SIFT and observed the noise in the image getting blurred and the key features being more defined. Later, we decided on using the in-built functions due to their robustness and the ability to try out different feature detection algorithms quickly.

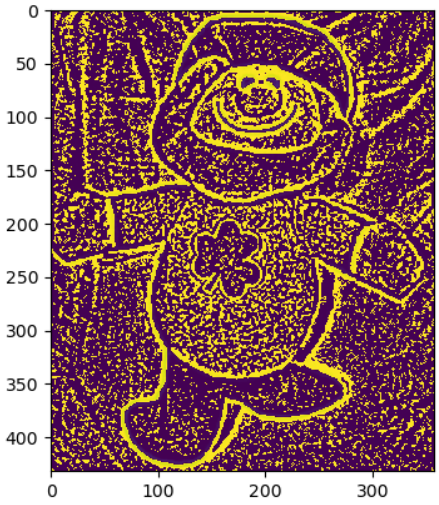



**Figure 1.0** Difference of two images after applying gaussian blur on both

Before we used SIFT, we converted the images to grayscale and then on applying it, we found 2318 features on the training image and 764 features on the second image. The matching of these features was done using the Brute Force matching algorithm. It took a keypoint and its descriptor from the first image and tried to match it with all of the keypoints and descriptors of

the second image while computing the distance if a match was made. On trying different images, we felt the matches made were not good and no suitable bounding box could be drawn in a robust fashion to contain all the keypoints and retain the shape of the leprechaun in it.

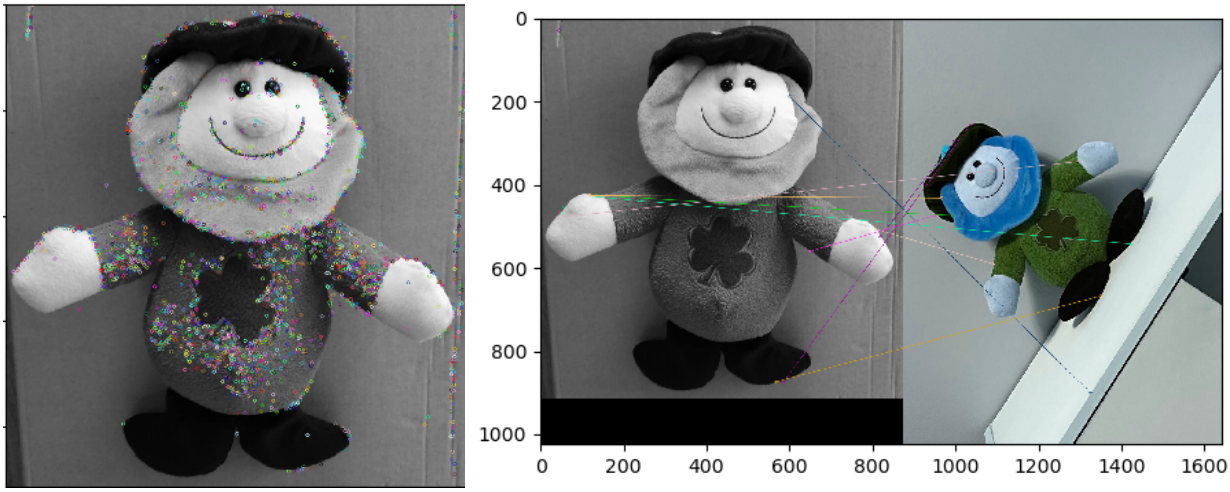



Figure 2.0 Keypoints (left image) and matching of keypoints on query and train image (right image)

Next, we used the Speeded Up Robust Features (SURF) to detect the keypoints and found that the detected features were slightly more. In the training image, we found 2254 features and in the other image we got 1827 features. We perform the feature matching using the Flann knnmatch algorithm that finds the best k matches for each descriptor and the keypoint in the training image. But after applying the Lowe distance ratio test to eliminate false matches we got only 32 suitable ones. The drawback we saw in this algorithm was that there were not enough good matches to draw a suitable bounding box across the leprechaun.

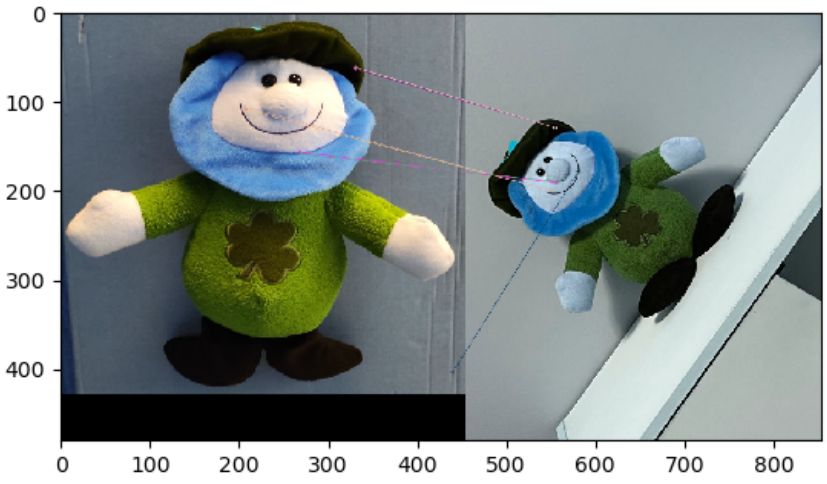



**Figure 3.0** Matches were very low with Flann KNNmatch algorithm

Finally, we used the Oriented FAST and Rotated BRIEF (ORB) algorithm for feature detection. The ORB algorithm is an improvement over the FAST and BRIEF algorithms by performing a

combination of them. It is as good and robust as the SIFT algorithm in detecting keypoints and is twice as fast. It performs better in comparison to SURF too. Another crucial information that helped us choose ORB was the fact it was open source and not patented. The other important reason being the presence of a large number of features being detected around the face of the leprechaun consistently regardless of the orientation of the leprechaun. The brute force matcher was used again to do the matching between the key points of the training and the target images.

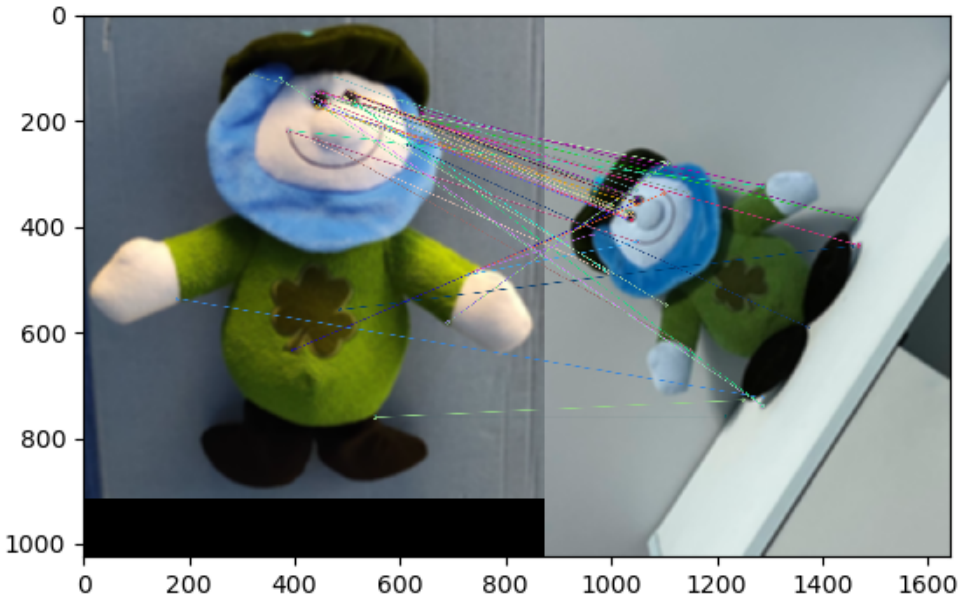



**Figure 4.0** Keypoints found using ORB and matched using brute force matcher

## Using median of matched keypoints

Although there are some bad matches, the presence of a fair number of matched keypoints around the face of the leprechaun gave us the idea of using the median to find an anchor point on the face, from where we plan to draw the bounding box from.

As mentioned above, the median of the matched keypoints was guaranteed to lie on the face of the leprechaun. This helped us compensate for the bad matches present above. On trying this idea on different scales and orientations of the leprechaun, we realized that it works and selected it as a point to help in detecting and distinguishing the face of the leprechaun from its body.


**Figure 5.0** Median (pink dot) of matched keypoints

## SCALE INVARIANCE:

We can draw a bounding box around the leprechaun based on the location of the median of the matched keypoints we just found. We would also prefer to have the size of the bounding box scale with the size of the leprechaun in the image. To that end, we try to find a circle approximating the face of the leprechaun which we can use as a reference to modify the size of the bounding box.

We first apply Canny Filter on the Gaussian Filtered image followed by Hough Transform to detect the circles in the image. The efficiency of Hough Transform relies on the group of accumulated pixels being distinct, i.e. a direct contrast between a pixel and its surrounding neighbors or if using a mask region, between a pixel region and its surrounding regions. If all pixels had similar accumulated values nothing would stand out as a line or circle. This is the reason why a Canny edge detection filter is applied prior to applying Hough Transform. An example of the circles found in the image after finding Hough Transform can be seen in Fig. (6). We select the circle that is closest to the median as seen in Fig. (6) and that gives us the circle approximating the face. Scaling the size of our bounding box in proportion to the radius of this circle makes our algorithm scale invariant.

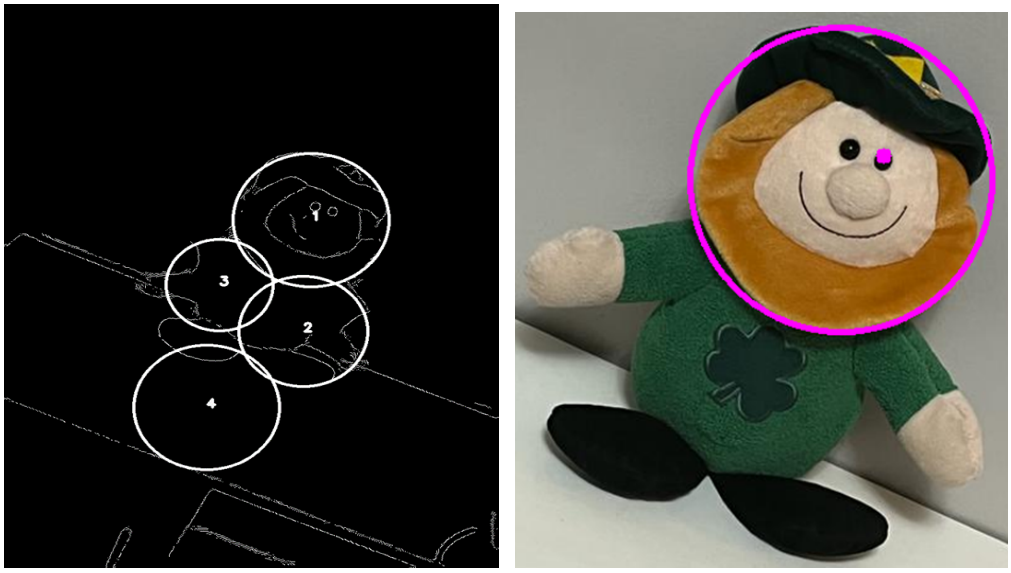



**Figure 6.0** Hough circles (left) and selected hough circle approximating the face (right)

Here is the code which does scaling:

***scale_top = 1.5*2***
***scale_bottom = 4.5*3***
***scale_side = 8***
***A = [c1+(scale_top*radius), c2+(scale_side*radius)]***
***B = [c1-(scale_bottom*radius), c2+(scale_side*radius)]***
***C = [c1-(scale_bottom*radius), c2-(scale_side*radius)]***
***D = [c1+(scale_top*radius), c2-(scale_side*radius)]***

Scale_top, scale_bottom and scale_side are the scaling factors which determine how rectangular the box will be and c1 and c2 are the center coordinates of the Hough circle.

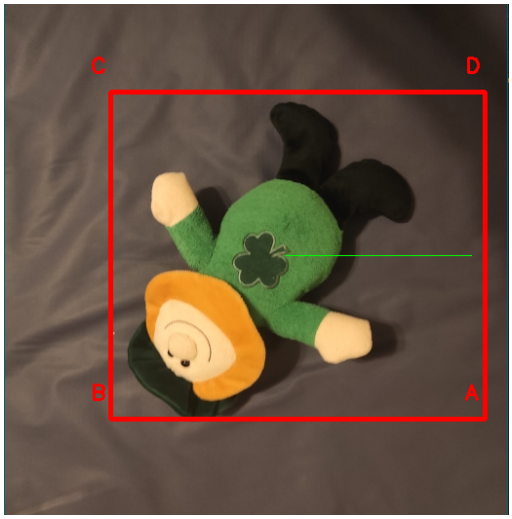



**Figure 7.0** Scaled and rotated bounding box around the object

Here we can see the box which is at zero degrees (The green line shows the 0 degrees reference line) but has been scaled according to the scale of the object.

## ROTATION INVARIANCE:

Once the scaling is done, we need to rotate the bounding box according to the orientation of our object.

This is done by finding the angle using Principal Component Analysis (PCA). We use PCA to find the direction in which data varies the most, that is to find the covariance of different features in an image and these directions are given by the eigenvectors which are the principal components. We apply PCA inside a box on the canny image. The center of the box is the centre of the circle which we found earlier and we try to keep the box over the face since it is symmetric to get the best possible covariance matrix and thus giving us accurate eigenvectors. The eigenvectors give us the directions of the feature axes and using these eigenvectors we find the angles by which both eigenvectors are rotated. Then we use one of these angles as the reference angle by which we need to rotate the bounding box.

Once we get the angle from PCA, we use trigonometry and geometric calculations to find the corner points for the new rotated bounding box.

Here is the code which finds the new corner points:

```
hyp_top = round(abs(math.sqrt((A[0]-c1)**2 + (A[1]-c2)**2)))
hyp_bottom = round(abs(math.sqrt((B[0]-c1)**2 + (B[1]-c2)**2)))

A_offset = math.degrees(math.atan(scale_side/scale_top))
B_offset = math.degrees(math.atan(scale_bottom/scale_side))
C_offset = 90-B_offset
D_offset = 90-A_offset

A_new = [round(c1+hyp_top*math.cos(math.radians(A_offset+theta))),
      round(c2+hyp_top*math.sin(math.radians(A_offset+theta)))]
B_new = [round(c1-hyp_bottom*math.sin(math.radians(B_offset+theta))),
      round(c2+hyp_bottom*math.cos(math.radians(B_offset+theta)))]
C_new = [round(c1-hyp_bottom*math.cos(math.radians(C_offset+theta))),
      round(c2-hyp_bottom*math.sin(math.radians(C_offset+theta)))]
D_new = [round(c1+hyp_top*math.sin(math.radians(D_offset+theta))),
      round(c2-hyp_top*math.cos(math.radians(D_offset+theta)))]
```

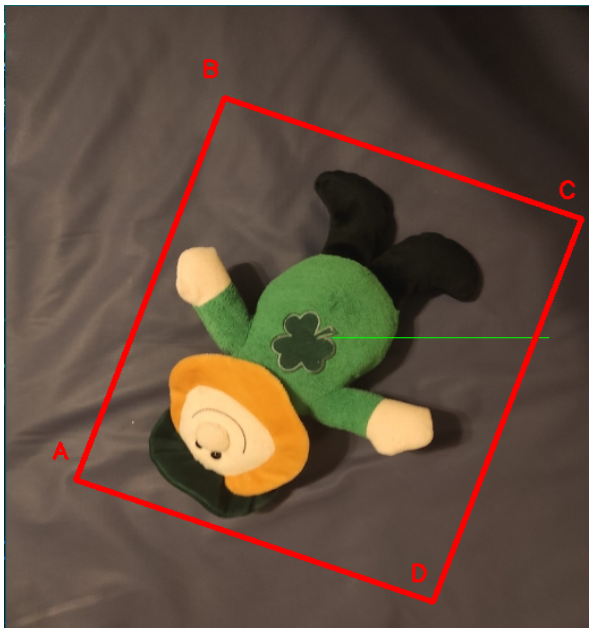



**Figure 8.0** Scaled and rotated bounding box around the object

Here we can see the scaled bounding box has been rotated by an angle theta ≈ 115° which was found using PCA.

Thus we get a bounding box which is scale and rotation invariant and is pretty accurate.

## Thresholding

After detecting our test subject i.e. leprechaun, we segmented it from its background. Creating a rotational and scale invariant bounding box gave us the exact part of the image where the thresholding needs to be applied. This approach makes our idea robust because even if our leprechaun is surrounded by similarly colored objects, we will still be able to segment it from the background. We take the image with the detected bounding box as the input Fig 9.0. Then a lower and upper bound is decided for the torso, face and beard of the leprechaun and combined using BITWISE OR. Using these we created a binary mask for the input image Fig.10.0. This mask and inverse mask is then applied on the input image to segment the leprechaun and the background into 2 different images shown in Fig.11.0 and Fig.12.0. Finally we take a weighted addition of these two images to fade out the background as shown in the results section. Here is the pseudocode for thresholding:

***Lower_Bound_Body=set of values in HSV***
***Upper_Bound_Body=set of values in HSV***
***Lower_Bound_Face=set of values in HSV***
***Upper_Bound_Face=set of values in HSV***
***Lower_Bound_Beard=set of values in HSV***

***Upper_Bound_Beard=set of values in HSV***

***Mask1= In Range of Body Parameters***
***Mask2= In Range of Face Parameters***
***Mask3= In Range of Beard Parameters***

***Total Mask= (Mask1) BITWISE OR (Mask2) BITWISE OR (Mask3)***
***Inverse Mask= BITWISE NOT(Total Mask)***

***Apply Total Mask to Image to get segmented Leprechaun***
***Apply Inverse Mask to Image to get segmented Background***

***Segmented Image= Weighted addition(0.8 x Segmented Object + 0.2 x Segmented Background)***

**Figure 9.0** Image with only active pixels inside the bounding box

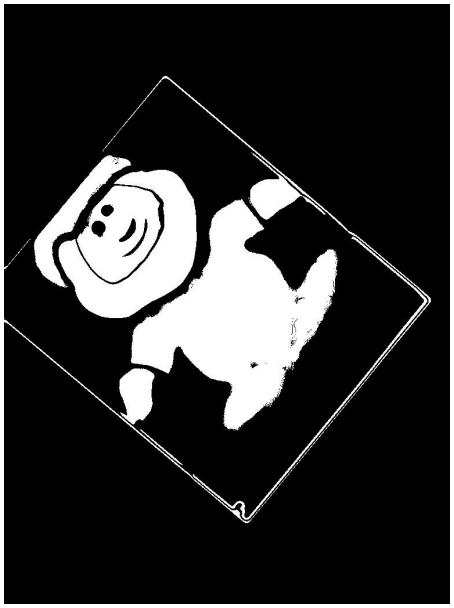


**Figure 10.0** Image Binary mask of the ROI (Region of Interest)

**Figure 11.0** Object Extraction from the image

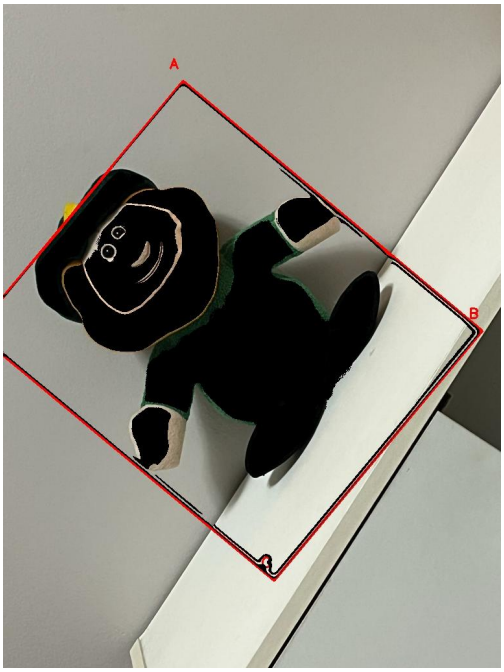



**Figure 12.0** Background Extraction from the image

## State-of-the-Art Approach

To achieve better robustness and accuracy we used a state-of-the-art model for instance segmentation of our Leprechaun, namely Mask R-CNN. Mask R-CNN is a computer vision model used for instance segmentation. It detects objects and segments it i.e. apply a mask over it simultaneously. As discussed in the paper we can use any backbone architecture ResNets (Residual Networks) or FPN (Feature Pyramid Network) with Faster RCNN for object detection and FCN (Fully Convolutional Networks) for Segmentation. We use ResNet-101 as the backbone pipeline for our implementation Fig.13.0.

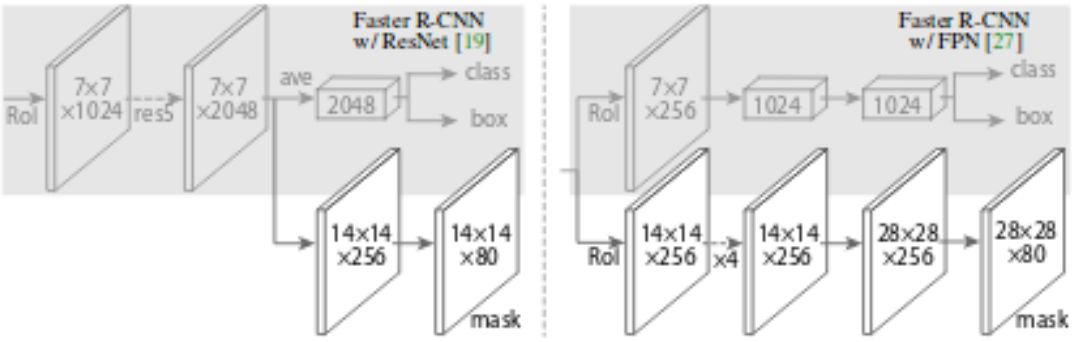



**Figure 13.0** Mask RCNN framework with Faster RCNN backend w/ ResNets**(left)** and FPN**(right)**

As we wanted to maintain the accuracy of the existing model but wanted to implement it to our test case i.e. a leprechaun, we performed Transfer Learning on the pre-trained model.

## Transfer Learning

Transfer Learning is a technique in which we take pre-trained weights of a model, unfreeze the last few layers, i.e. we re-learn the weights of the last few layers to fit our data to the new subject. This is done because a Neural Net learns high level features in the starting layers and as the model gets deeper it starts to learn the specific features of the data provided. By using Transfer Learning we achieve to maximum upper hands:

- Great Accuracy
- Faster Learning

We annotated our leprechaun along with some Pokemon plushes and labeled their class as “Plushies”. We unfroze the last 2 layers of the Mask RCNN model and trained our model on 120 self-annotated images for 200 epochs. The training was done on a 8GB GTX 1070 with 32 RAM. The results we got were quite impressive and can be seen in Fig. 14.0, 15.0 and 16.0 for only Leprechaun, Leprechaun and Pokemons (uncluttered) and Leprechaun and Pokemons (cluttered) respectively.


**Figure 14.0** Test Leprechaun image segmented

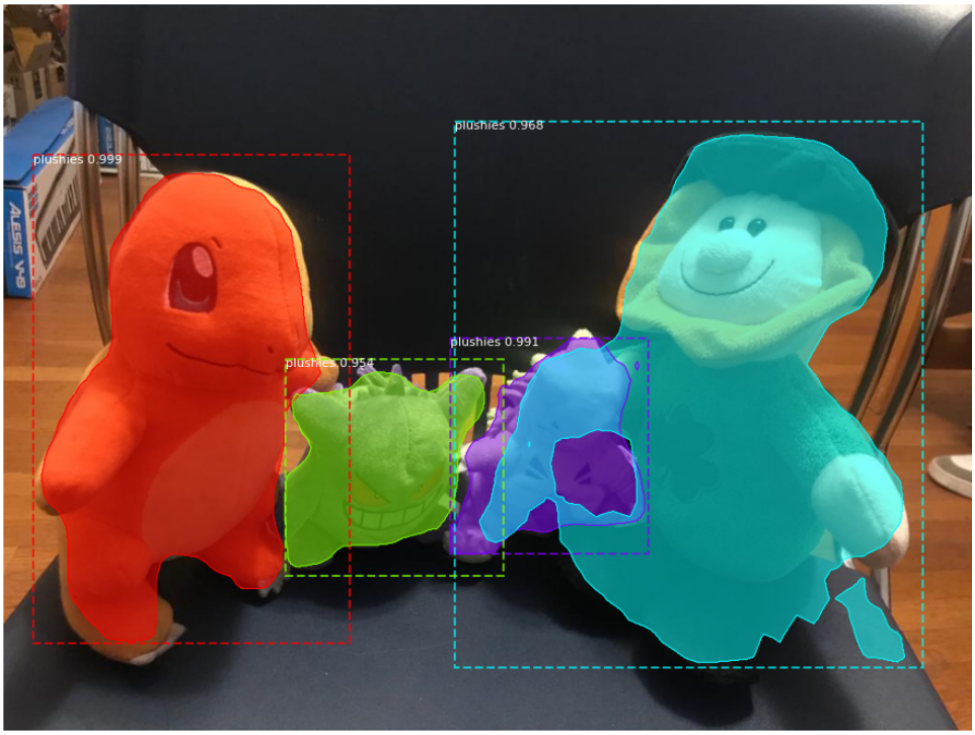



**Figure 15.0** Instance Segmented in an uncluttered environment

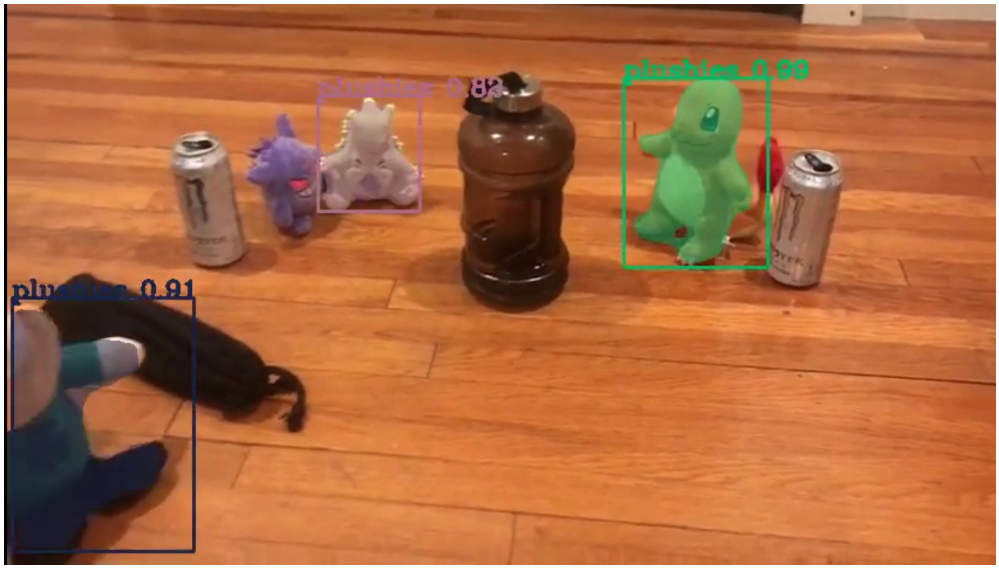


**Figure 16.0** InstanceSegmented in an cluttered environment

As we can see in Fig. 16.0 our trained model does not mask one of the Pokemons(Gengar) showing that though transfer learning gives us a model with acceptable accuracy with low data, we still need loads of data to get a super accurate model. If we had trained on more than 120 pictures(as stated earlier) we would have been able to mask even the “Gengar” plushie (purple plushie in the image) in Fig.16.0.

## Results

Thus, using the traditional approach, we have made our code capable of segmenting the leprechaun plushie from the background regardless of the orientation and scaling. Our code can also handle some clutter in its environment. As seen in the below images, we are able to detect and segment the leprechaun stuffed toy at different angles and scaling.

The cases which we weren't able to handle were when the leprechaun was turned around its body or there is too much clutter in the environment. Also the drawback of the traditional method is that we couldn't generalize a class of the object to detect different instances of the object. This shortcoming necessitates the usage of neural networks to fine tune the weights.

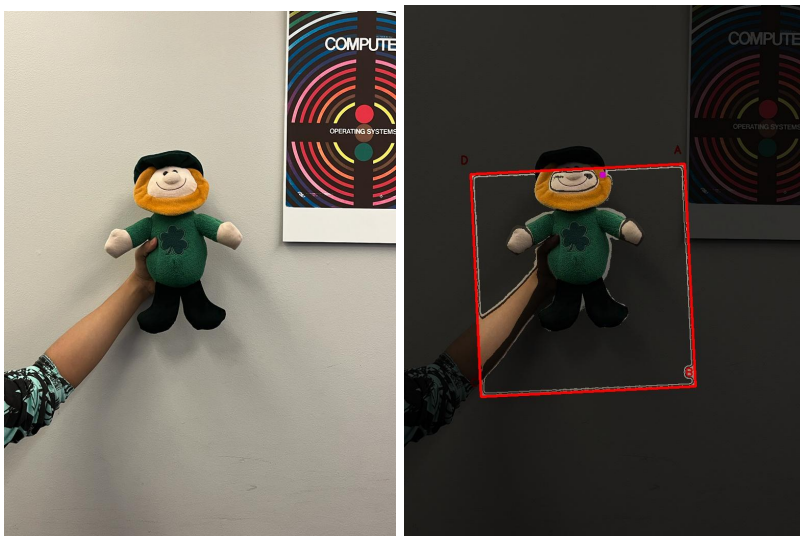



**Figure 17.0** Input and Output with some clutter in the background

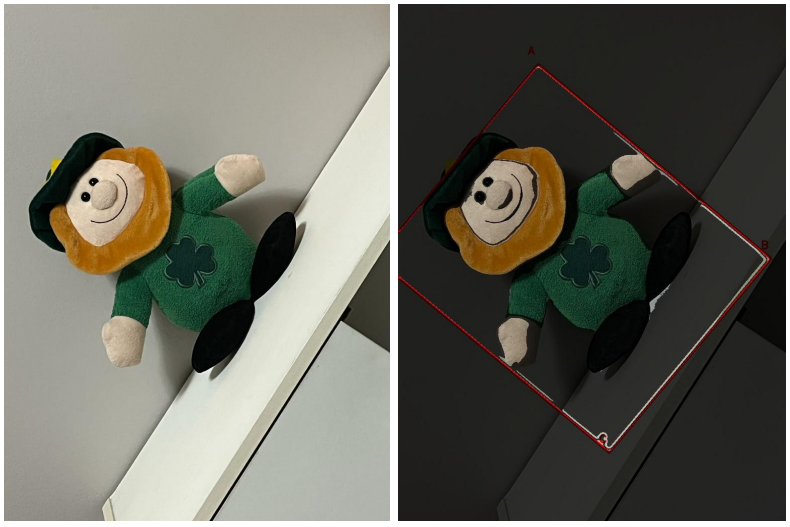


**Figure 18.0** Input and Output with rotation and scaling 1

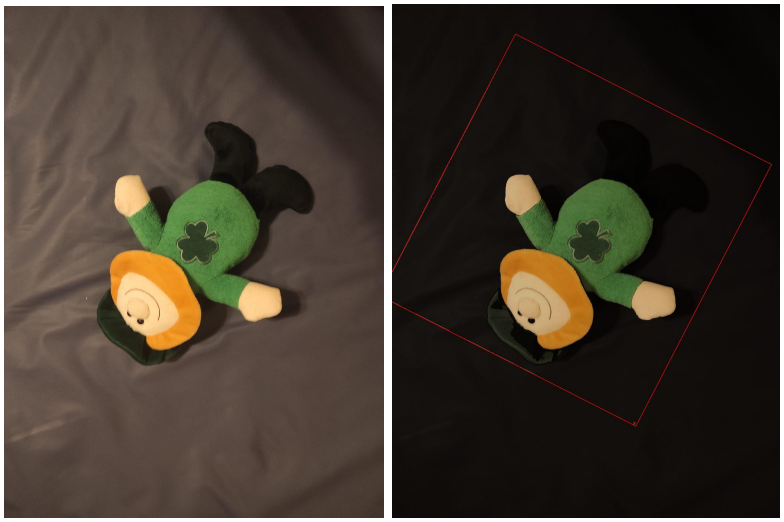


**Figure 19.0** Input and Output with rotation and scaling 2

On utilizing the Mask R-CNN network, we were able to handle multiple instances of the same class. Although sometimes, this network also fails when there is a huge overlap between the objects. Another drawback of the state of the art methodology is that it is very heavily dependent on the amount of data provided for training the network.

A real time video implementation of instance segmentation was also done using the Mask R-CNN to detect the plushies and its recording is available at the below youtube link.

**https://youtu.be/XR-_cNjNDcs**

## Discussion

Overall, our final project tries to draw comparison between the traditional methodology and the state of the art methodology and realize the drawbacks and use cases of both. The traditional methodology though lacking in robustness works quite well in simple use case scenarios of semantic segmentation where using a neural network would be an overkill.

Another aim of this final project was to try to utilize the material covered in class and having seen both approaches and their usefulness, we believe that we have learned a lot about the complexities involved in designing a computer vision system to solve challenging tasks like semantic and instance segmentation.

## References


- https://nanonets.com/blog/semantic-image-segmentation-2020/
- https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation
- https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/#h2_7
- https://viso.ai/deep-learning/mask-r-cnn/
- https://docs.opencv.org/3.4/d1/dee/tutorial_introduction_to_pca.html
- https://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf
- https://docs.opencv.org/4.x/d9/df8/tutorial_root.html
- He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In *Proceedings of the IEEE international conference on computer vision*, pp. 2961-2969. 2017.

# Appendix

## Code for traditional approach:

```
import numpy as np
import cv2
import math
from matplotlib import pyplot as plt
import tkinter as tk
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification

# function for resizing image
def resize_image(img, area=0.0, window_h=0, window_w=0):
        h, w = img.shape[:2]
        root = tk.Tk()
        screen_h = root.winfo_screenheight()
        screen_w = root.winfo_screenwidth()

        if area != 0.0:
        vector = math.sqrt(area)
        window_h = screen_h * vector
        window_w = screen_w * vector

        if h > window_h or w > window_w:
        if h / window_h >= w / window_w:
        multiplier = window_h / h
        else:
        multiplier = window_w / w
        img = cv2.resize(img, (0, 0), fx=multiplier, fy=multiplier)

        return img


nearest_circle = []
curr_dist = 0

img1 = cv2.imread(r"C:\Users\hp\Downloads\base_1_resize.jpeg")  # query Image
img2 = cv2.imread(r"C:\Users\hp\Downloads\mag5_resize.jpeg") # train Image
```

```
img_name = "mag_1.jpeg"
img2=resize_image(img2,area=0.8)
img_final = img2
img1=cv2.GaussianBlur(img1,(11,11),cv2.BORDER_DEFAULT)
img2=cv2.GaussianBlur(img2,(11,11),cv2.BORDER_DEFAULT)

# Create an ORB object
orb = cv2.ORB_create()

# find the keypoints and descriptors using ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
good_matches = matches[:50]

img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches,None, flags=2)

# Show all the matches
plt.imshow(img3),plt.show()

#  Source points (from query image)
src_pts = np.float32([ kp1[m.queryIdx].pt for m in good_matches ]).reshape(-1,1,2)
#  Destination points (from train image)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good_matches ]).reshape(-1,1,2)
#  Homography
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
matchesMask = mask.ravel().tolist()
h,w = img1.shape[:2]
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)

dst = cv2.perspectiveTransform(pts,M)
dst += (w, 0)  # adding offset
```

```
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   # matchesMask = matchesMask, # draw only inliers
                   flags = 2)
t1=np.reshape(dst_pts,(len(dst_pts),2))

# find median of matched keypoint
point=np.median(t1,axis=0)
k,l=int(point[0]),int(point[1])

# Draw keypoints matches
img3 = cv2.drawMatches(img1,kp1,img2,kp2,good_matches, None,**draw_params)

# Draw bounding box on target image and show along with train image
img3 = cv2.polylines(img3, [np.int32(dst)], True, (0,0,255),3, cv2.LINE_AA)
img3=cv2.resize(img3,(854,480))

# Print location of median
print(f"median: {k}, {l}")

# Applying canny
gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)
center=(0,0)
radius=0
gray=cv2.GaussianBlur(gray,(25,25),cv2.BORDER_DEFAULT)
gray=cv2.Canny(gray,20,30) #20,30
# cv2.imshow("g",gray)
rows = gray.shape[0]

# Applying hough
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, rows / 8,
                           param1=100, param2=30,
                           minRadius=10, maxRadius=250)

if circles is not None:
        circles = np.uint16(np.around(circles))
        ctr = 0
        for i in circles[0, :]:
```

```
        ctr+=1
        # Finding nearest circle to the median
        print(f"circle {ctr} points: " + str(i), end="")
        dist = abs(math.sqrt((k-i[0])**2 + (l-i[1])**2))
        print(" dist: " + str(dist))
        if dist<curr_dist or curr_dist == 0:
        nearest_circle = i
        curr_dist = dist
        dist = 0
        # numbering circle
        cv2.putText(gray, str(ctr), (i[0],i[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(246,255,12), 3)
        # draw circle
        radius = i[2]
        #cv2.circle(gray, (i[0],i[1]), radius, (255, 0, 255), 3)

print(f"nearest circle: {nearest_circle}")

# c1 and c2 are the center of the hough circle
c1,c2=nearest_circle[0], nearest_circle[1]
side=(nearest_circle[2])*2.5
side=int(side)

# draw median on final image
# cv2.circle(img_final, (k,l), 1, (255, 0, 255), 10)
# draw circle on final image
# cv2.circle(img_final, (c1,c2), nearest_circle[2], (255, 0, 255), 3)

p1=(c1-side,c2-side)
p2=(c1+side,c2-side)
p3=(c1+side,c2+side)
p4=(c1-side,c2+side)

a,b=gray.shape
l=[]
for i in range(a):
        for j in range(b):
        if gray[i][j]==255 and i>(c1-side) and i<(c1+side) and j>(c2-side) and j<(c2+side):
        l.append([i,j])
```

```
# cv2.imshow("matches", img3)
# cv2.imshow("detected circles", gray)
# cv2.waitKey(0)

# PCA
X, y = make_classification(n_samples=1000)
X=np.array(l)
n_samples = X.shape[0]

# create a PCA object
pca = PCA()
X_transformed = pca.fit_transform(X)

# We center the data and compute the sample covariance matrix.
X_centered = X - np.mean(X, axis=0)
cov_matrix = np.dot(X_centered.T, X_centered) / n_samples
eigenvalues = pca.explained_variance_
eigenvectors=[]
for eigenvalue, eigenvector in zip(eigenvalues, pca.components_):
        # print(np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)))
        # print(f"eigenValue: {eigenvalue}")
        eigenvectors.append(eigenvector)

print(f"eigenVectors: {eigenvectors}")
tampp = 45
angle2=math.atan2(eigenvectors[1][1],eigenvectors[1][0])*180/math.pi+tampp
angle1=math.atan2(eigenvectors[0][1],eigenvectors[0][0])*180/math.pi+tampp

# for drawing rotation angle as found from PCA
centre_tuple = (c1,c2)
endpoint1 = (c1 + np.uint16(eigenvectors[0][0]*100), c2 +
np.uint16(eigenvectors[0][1]*100))
# cv2.line(img_final,centre_tuple, endpoint1, (0, 0, 255))
# cv2.line(img2,centre_tuple, (c1+side,c2), (0, 255, 0))

angle_deg=min(angle1,angle2)
print(f"angle_deg: {angle_deg}")

if angle_deg<180:
        theta = angle_deg+180
```

```
else:
        theta = angle_deg

# scaling for bounding box
scale_top = 1.5*2
scale_bottom = 4.5*3
scale_side = 8

# Find bounding box corner points which is scaled and rotated
A = [c1+(scale_top*radius), c2+(scale_side*radius)]
B = [c1-(scale_bottom*radius), c2+(scale_side*radius)]
C = [c1-(scale_bottom*radius), c2-(scale_side*radius)]
D = [c1+(scale_top*radius), c2-(scale_side*radius)]

hyp_top = round(abs(math.sqrt((A[0]-c1)**2 + (A[1]-c2)**2)))
hyp_bottom = round(abs(math.sqrt((B[0]-c1)**2 + (B[1]-c2)**2)))

A_offset = math.degrees(math.atan(scale_side/scale_top))
B_offset = math.degrees(math.atan(scale_bottom/scale_side))
C_offset = 90-B_offset
D_offset = 90-A_offset

# four corner points of bounding box after scaling and rotating
A_new = [round(c1+hyp_top*math.cos(math.radians(A_offset+theta))),
        round(c2+hyp_top*math.sin(math.radians(A_offset+theta)))]
B_new = [round(c1-hyp_bottom*math.sin(math.radians(B_offset+theta))),
        round(c2+hyp_bottom*math.cos(math.radians(B_offset+theta)))]
C_new = [round(c1-hyp_bottom*math.cos(math.radians(C_offset+theta))),
        round(c2-hyp_bottom*math.sin(math.radians(C_offset+theta)))]
D_new = [round(c1+hyp_top*math.sin(math.radians(D_offset+theta))),
        round(c2-hyp_top*math.cos(math.radians(D_offset+theta)))]

# labeling four corners of bounding box
cv2.putText(img_final, "A", (A_new[0]-20,A_new[1]-20),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
cv2.putText(img_final, "B", (B_new[0]-20,B_new[1]-20),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
cv2.putText(img_final, "C", (C_new[0]-20,C_new[1]-20),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
```

```
cv2.putText(img_final, "D", (D_new[0]-20,D_new[1]-20),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)

# Draw bounding box
contour = np.array([A_new, B_new, C_new, D_new])
cv2.drawContours(img_final,[contour],0,(0,0,255),10)
cv2.imwrite("Rot.jpeg",img_final)

# creating an image copy for mask
img_ = np.copy(img_final)

# all locations outside bounding box made black
for i in range(img_final.shape[0]):
        for j in range(img_final.shape[1]):
        if cv2.pointPolygonTest(contour, (j, i), False) == -1.0:
        #print(f"{i} {j}")
        img_[i][j] = np.array([0, 0, 0])


# cv2.imshow("detected circles: ", gray)
cv2.imshow("result", img_final)
cv2.imwrite("result_"+img_name, img_final)
cv2.imshow("contour_space", img_)
cv2.imwrite("contour_space_"+img_name, img_)

# Applying GaussianBlur and Converting Image into HSV colorspace
gaus=cv2.GaussianBlur(img_,(25,25),cv2.BORDER_DEFAULT)
hsv_conv=cv2.cvtColor(gaus,cv2.COLOR_BGR2HSV)

# threshold values for HSV
lower_bound_body = np.array([31,1,1])
upper_bound_body = np.array([180,255,255])
lower_bound_face = np.array([0,60,68])
upper_bound_face = np.array([18,88,255])
lower_bound_beard = np.array([0,115,115])
upper_bound_beard = np.array([35,255,255])

# create masks
mask1 = cv2.inRange(hsv_conv,lower_bound_body,upper_bound_body)
mask2 = cv2.inRange(hsv_conv,lower_bound_face,upper_bound_face)
```

```
mask3 = cv2.inRange(hsv_conv,lower_bound_beard,upper_bound_beard)

# bitwise OR to get the overall mask
final_mask = mask1|mask2|mask3

#Inverse mask for the background
inv_mask=cv2.bitwise_not(final_mask)

#Segment Object Using Mask
masked_img_object=cv2.bitwise_and(img_final,img_final,mask=final_mask)
#Segment Background Using Inverse Mask
masked_img_background=cv2.bitwise_and(img_final,img_final,mask=inv_mask)

#Show all the masked images
cv2.imshow("segment1",masked_img_object)
cv2.imwrite("segment1_"+img_name,masked_img_object)
cv2.imshow("segment",masked_img_background)
cv2.imwrite("segment_"+img_name,masked_img_background)

#Add the only object and only background image in a weighted ration of 8:2
img_segment=cv2.addWeighted(masked_img_object, 0.8, masked_img_background,
0.2,0)

# Draw the final bounding box using red color
cv2.drawContours(img_segment,[contour],0,(0,0,255),4)
# Show the final image
cv2.imshow("Final Answer",img_segment)
#Write the final image to file
cv2.imwrite("Answer_"+img_name,img_segment)

cv2.imshow("final mask",final_mask)
cv2.imwrite("final_mask_"+img_name,final_mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

**Code for Mask R-CNN**

```
import cv2
import numpy as np
import os
import sys
import random
import math
import numpy as np
import skimage.io
import matplotlib
import matplotlib.pyplot as plt
import keras
import tensorflow as tf

%matplotlib inline

def random_colors(N): #Random Colors of Masks
        np.random.seed(1)
        colors = [tuple(255 * np.random.rand(3)) for _ in range(N)]
        return colors


def apply_mask(image, mask, color, alpha=0.5):# Apply the mask to the results
        """apply mask to image"""
        for n, c in enumerate(color):
        image[:, :, n] = np.where(
        mask == 1,
        image[:, :, n] * (1 - alpha) + alpha * c,
        image[:, :, n]
        )
        return image


def display_instances(image, boxes, masks, ids, names, scores):
        """
        take the image and results and apply the mask, box, and Label
        """
        n_instances = boxes.shape[0]
        colors = random_colors(n_instances)
```

```
        if not n_instances:
        print('NO INSTANCES TO DISPLAY')
        else:
        assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]

        for i, color in enumerate(colors):
        if not np.any(boxes[i]):
        continue
        y1, x1, y2, x2 = boxes[i]
        label = names[ids[i]]
        score = scores[i] if scores is not None else None
        caption = '{} {:.2f}'.format(label, score) if score else label
        mask = masks[:, :, i]

        image = apply_mask(image, mask, color)
        image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
        image = cv2.putText(image, caption, (x1, y1), cv2.FONT_HERSHEY_COMPLEX,
0.7, color, 2)

        return image


if __name__ == '__main__':
        """
        test everything
        """
        # Root directory of the project
        ROOT_DIR = os.path.abspath("../")

        # Import Mask RCNN
        sys.path.append(ROOT_DIR)  # To find local version of the library
        from mrcnn import utils
        import mrcnn.model as modellib
        from mrcnn import visualize
        # Import config
        sys.path.append(os.path.join(ROOT_DIR, "samples/"))  # To find local version

        # Directory to save logs and trained model
        MODEL_DIR = os.path.join(ROOT_DIR, "logs")
```

```
        # Local path to trained weights file
        COCO_MODEL_PATH = os.path.join(ROOT_DIR, "plushies.h5")# load the
weights
        # Directory of images to run detection on
        IMAGE_DIR = os.path.join(ROOT_DIR, "images") #image directory if testing on
images

        class InferenceConfig(plushie.Config):
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1   # As we have only 8GB VRAM on GTX 1070 it is set
to 1

        config = tf.ConfigProto()
        config.gpu_options.per_process_gpu_memory_fraction = 0.6  # 0.6 Use only
60% VRAM
        keras.backend.tensorflow_backend.set_session(tf.Session(config=config))
        config = InferenceConfig()
        config.display()
        #Run MaskRCNN model in Inference Mode
        model = modellib.MaskRCNN(
        mode="inference", model_dir=MODEL_DIR, config=config
        )
        model.load_weights(MODEL_PATH, by_name=True)
        class_names = ['plushies']



        capture = cv2.VideoCapture(r"E:\cvproject\2021-12-13 05-19-55.mp4")
        size =
(int(capture.get(cv2.CAP_PROP_FRAME_WIDTH)),int(capture.get(cv2.CAP_PROP_F
RAME_HEIGHT)))
        codec = cv2.VideoWriter_fourcc(*'DIVX')
        output =
cv2.VideoWriter(r'E:\cvproject\Mask_RCNN\samples\videofile_masked.avi', codec, 60.0,
size)
        while (capture.isOpened()):
        ret, frame = capture.read()#Read the video frame by frame
        if ret:
        results = model.detect([frame], verbose=0)
        r = results[0]
```

```
        frame = display_instances(
                frame, r['rois'], r['masks'], r['class_ids'], class_names, r['scores']# Perform
Instance Segmentation on each image
        )
        output.write(frame) # Write out the frame into a video file
        else:
        break
        capture.release()
        cv2.destroyAllWindows()
```