Assignment - 2

Q1 N-Cuts

- **N-Cut for segmenting brightness Images**

- Feature vector of a pixel used for calculating weight

  F(i) = (I_red(i) + I_green(i) + I_blue(i)) / 3

  I_red(i) -> Intensity of pixel in red channel
  I_green(i) -> Intensity of pixel in green channel
  I_blue(i) -> Intensity of pixel in blue channel
- Each element (W[i][j]) in the weight matrix is calculated using feature proximity and spatial proximity terms.

  Spatial_proximity [i][j] = `exp(-(distance(i,j)**2)/sigma_x)`
  Feature_proximity [i][j] = `exp(-(F(i) - F(j)**2)/sigma_i)`

  if **`distance(i,j) < r`**
  `W[i][j] =` **Feature_proximity [i][j] * (Spatial_proximity [i][j])**

  else
  W[i][j] = 0

  r-> cutoff distance
- Calculated degree matrix (D) as follows

  `D = np.diag(W.sum(axis = 1))`
- `Calculated eigen vector corresponding to the second smallest eigenvalue for the following generalised eigen system-`

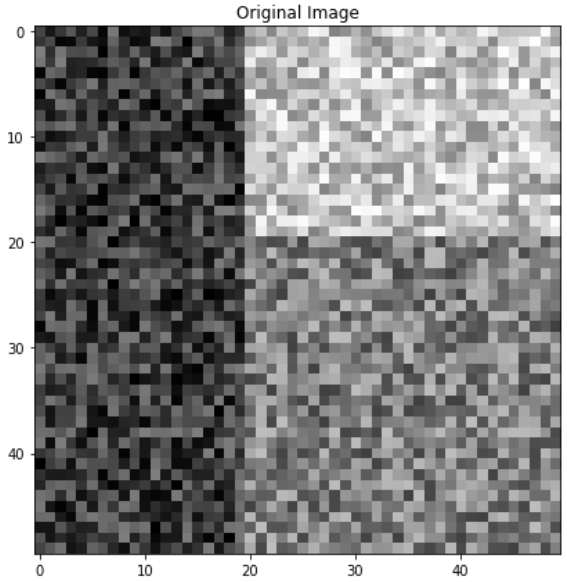
  `(D-W)*y = lambda*D*y`
- `Used threshold = 0 for eigen vector`
- `Parameters used-`

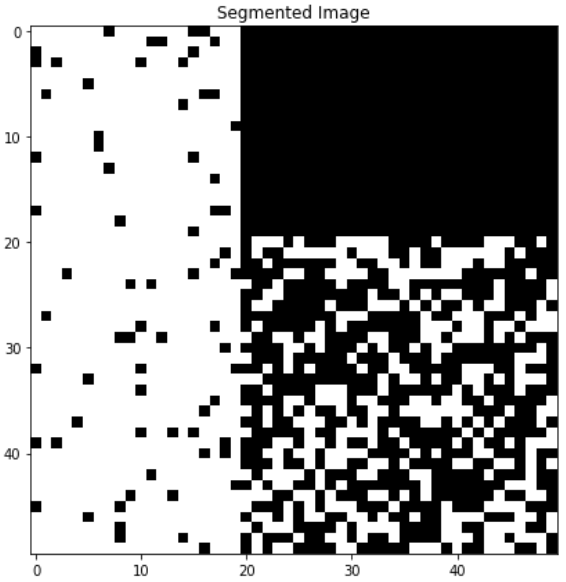
  `R = 50`
  `Sigma_i = 10`
  `Sigma_x = 1e7`
- `Downscaled the images to 50*50 for faster computation`

Results:

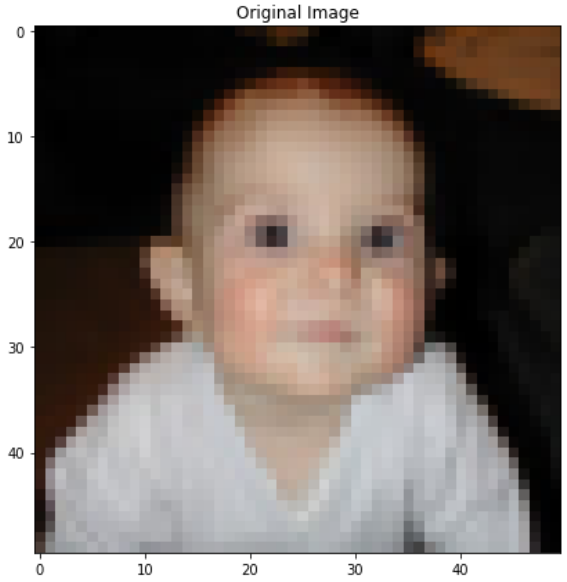
i) **N-Cut on Original Image (Downscaled)**

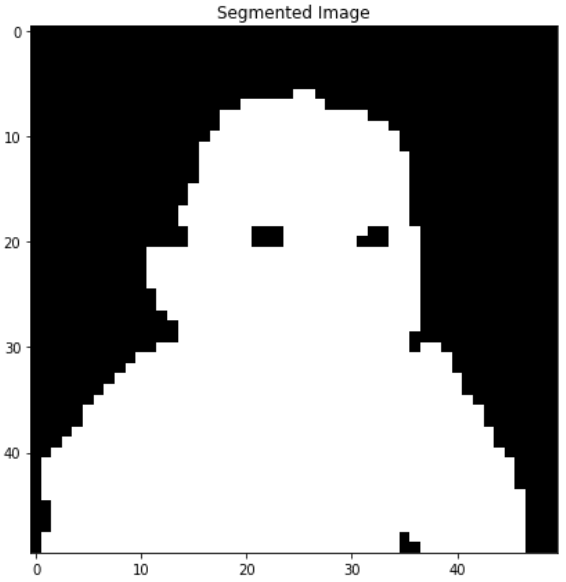
test1.png (downscaled - (50*50))



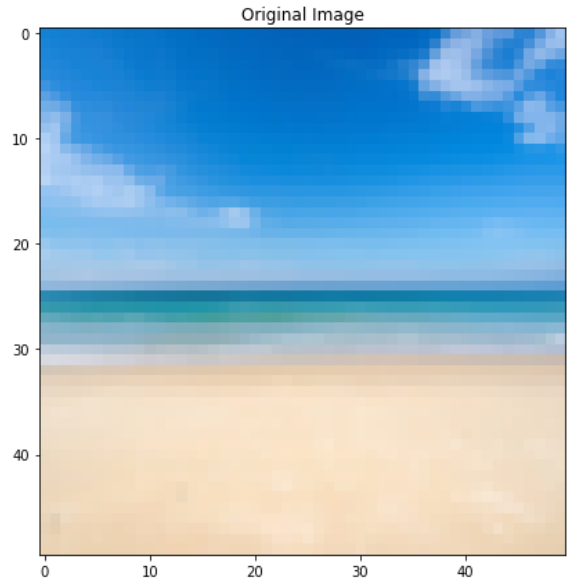


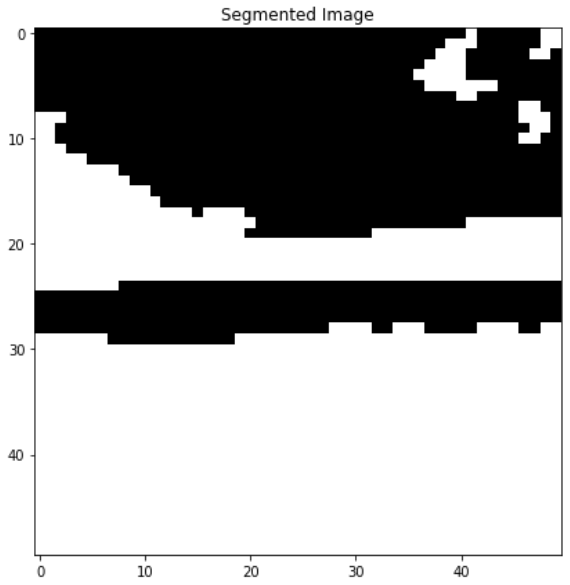
test2.jpg (downscaled - (50*50))



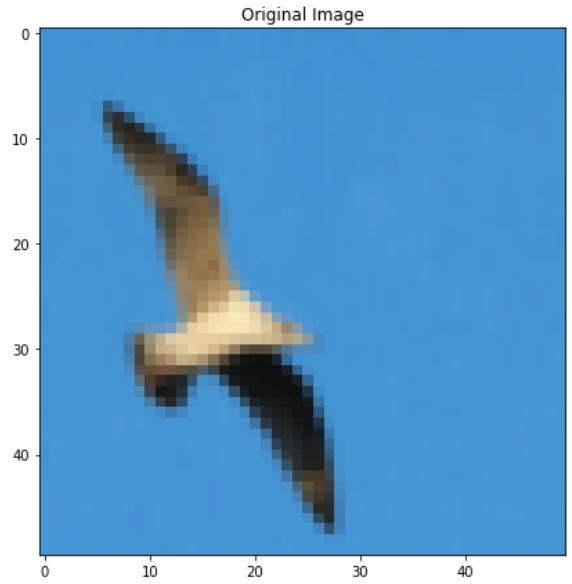


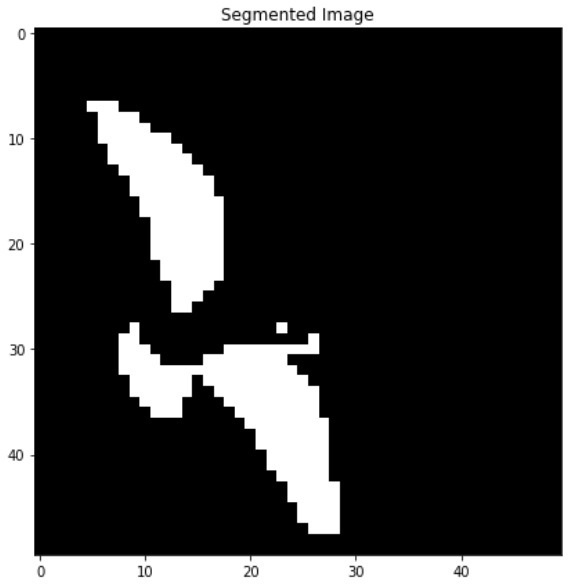
test3.jpg (downscaled - (50*50)



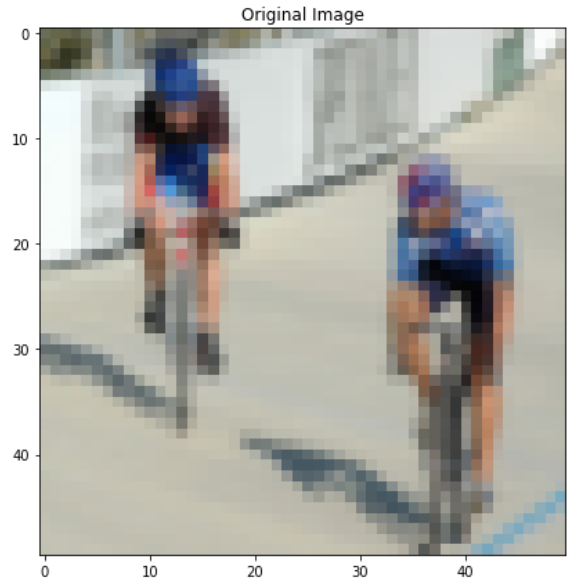


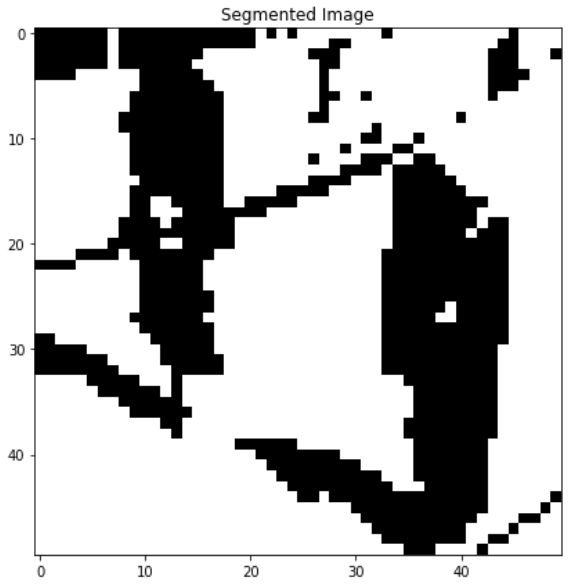
test4.jpg (downscaled - (50*50)





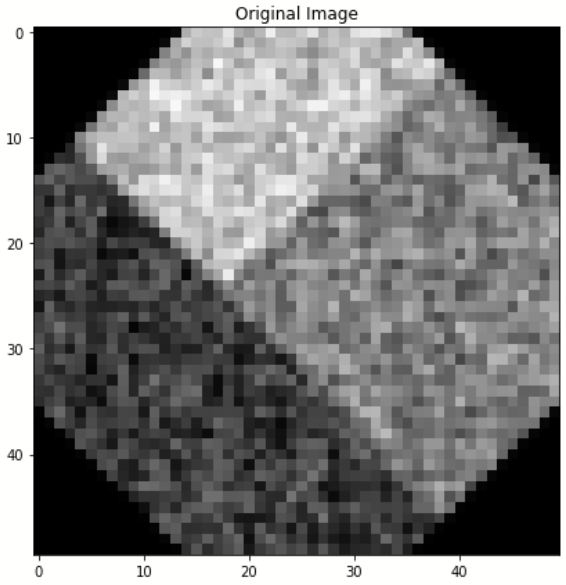
test5.jpg (downscaled - (50*50))

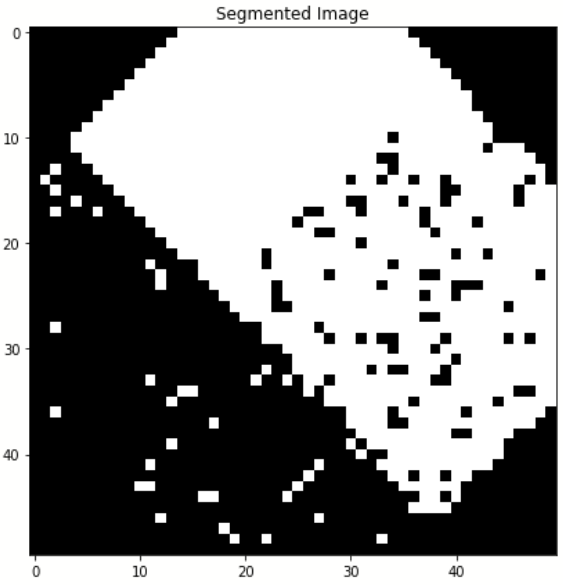




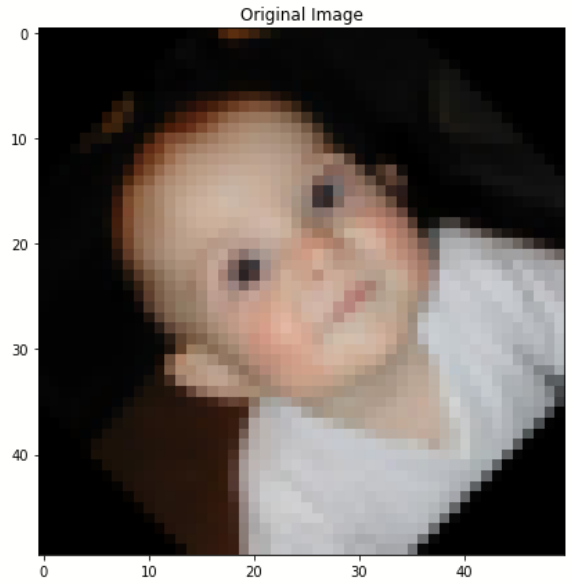
ii) **N-Cut on Rotated Image (Downscaled)**

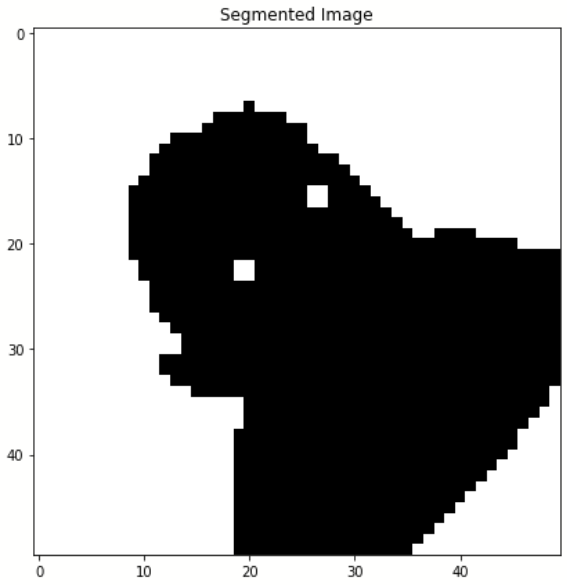
test1.png (downscaled - (50*50))



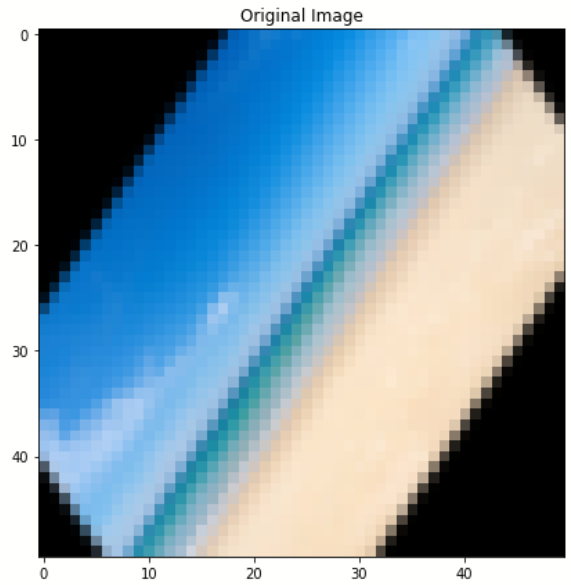


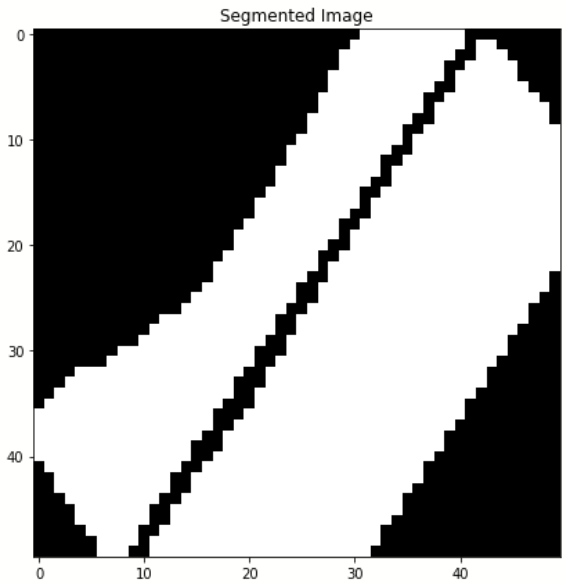
test2.jpg (downscaled - (50*50)



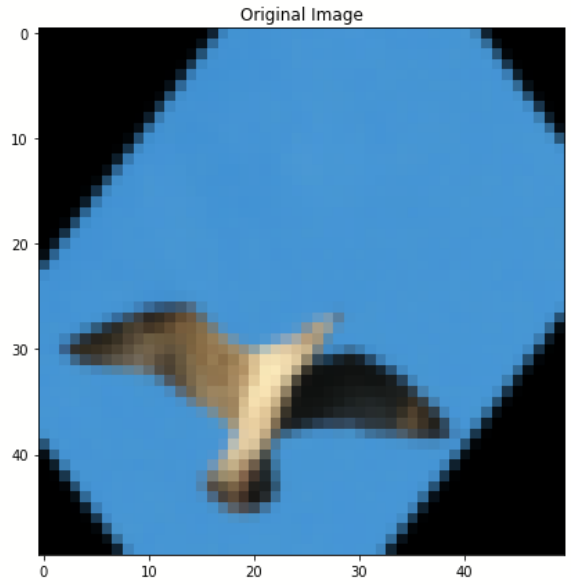


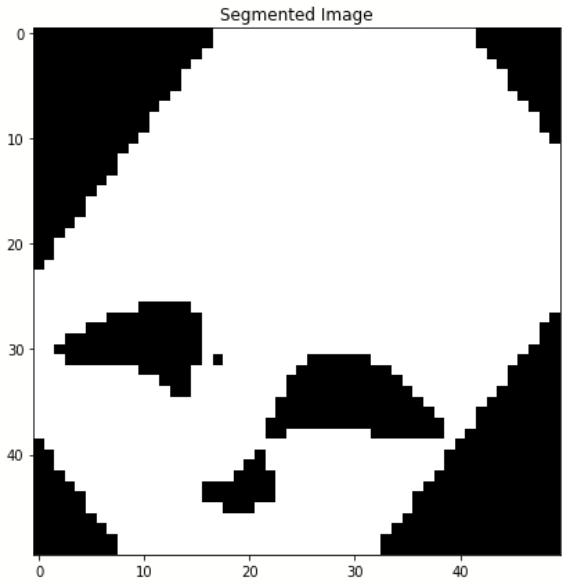
test3.jpg (downscaled - (50*50))



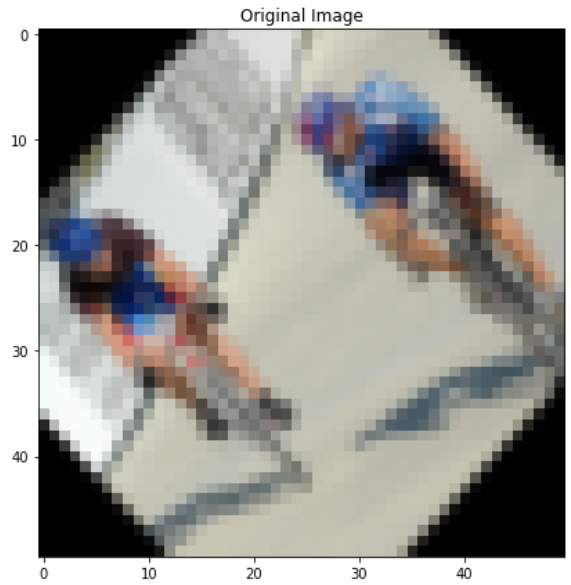


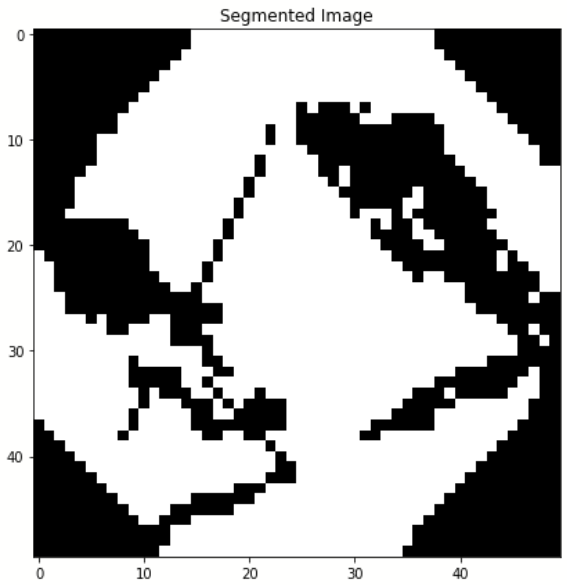
test4.jpg (downscaled - (50*50))





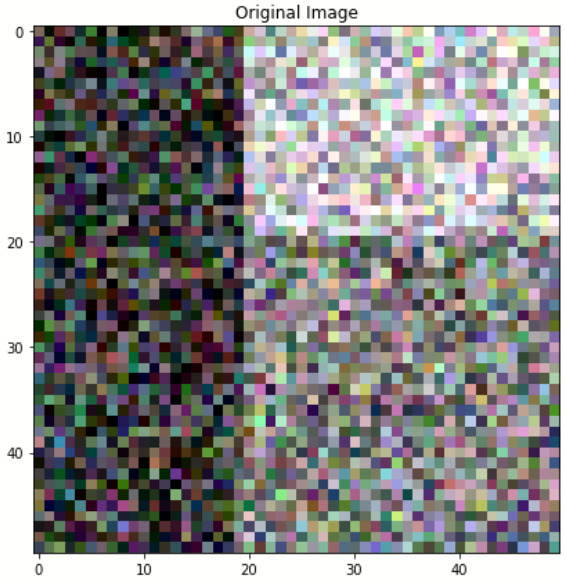
test5.jpg (downscaled - (50*50))

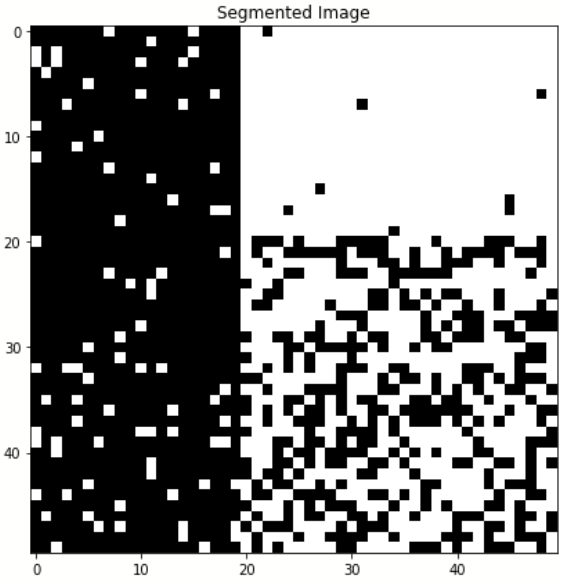




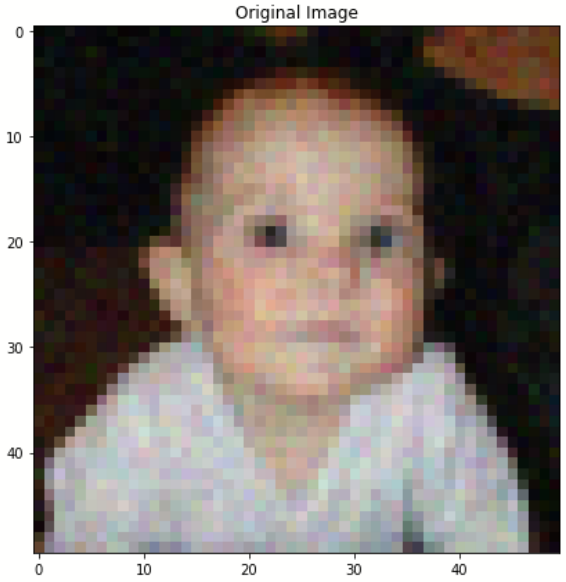
iii) **N-Cut on Gaussian Blurred Image (Downscaled)**

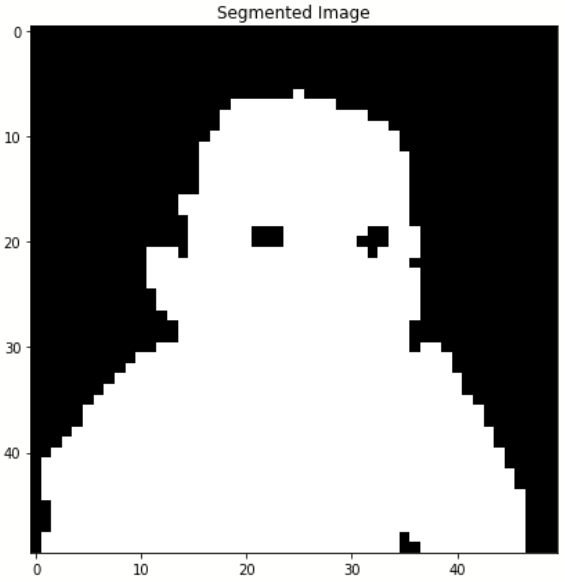
test1.png (downscaled - (50*50))



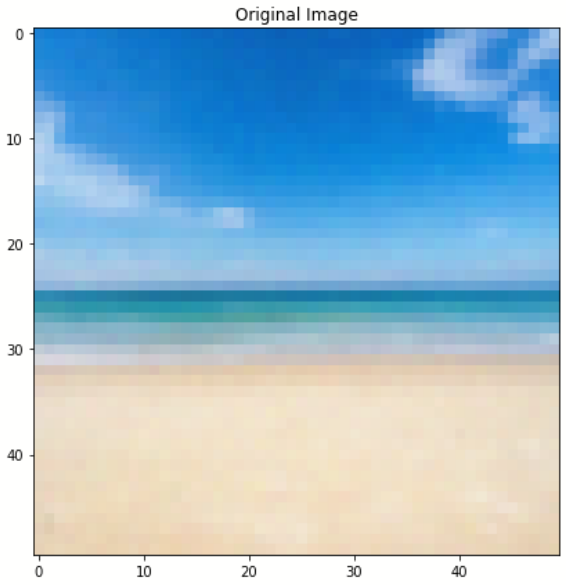


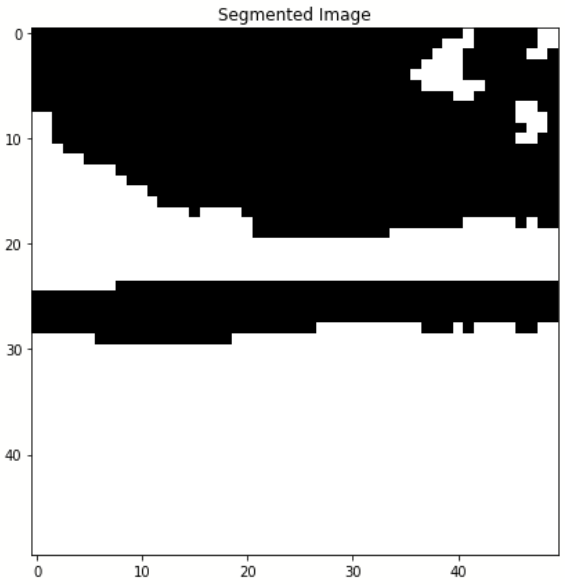
test2.jpg (downscaled - (50*50))



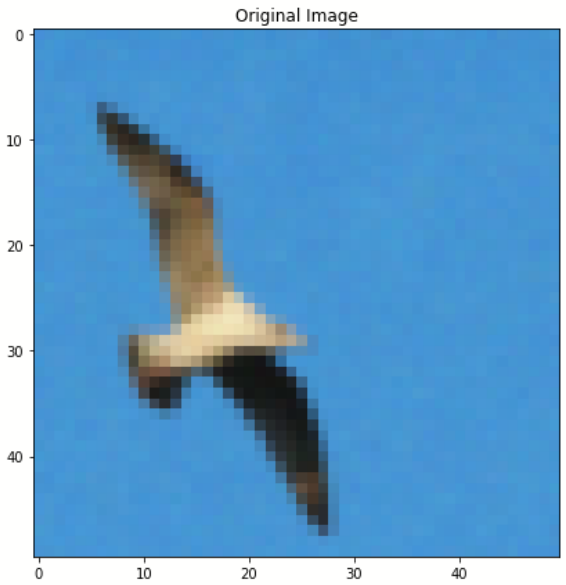


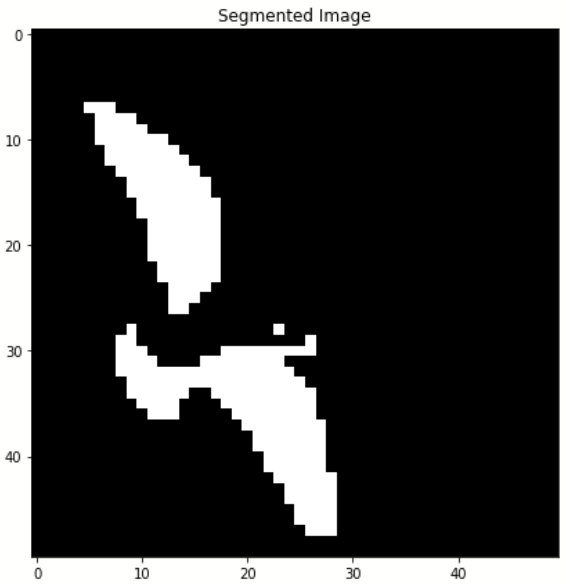
test3.jpg (downscaled - (50*50))



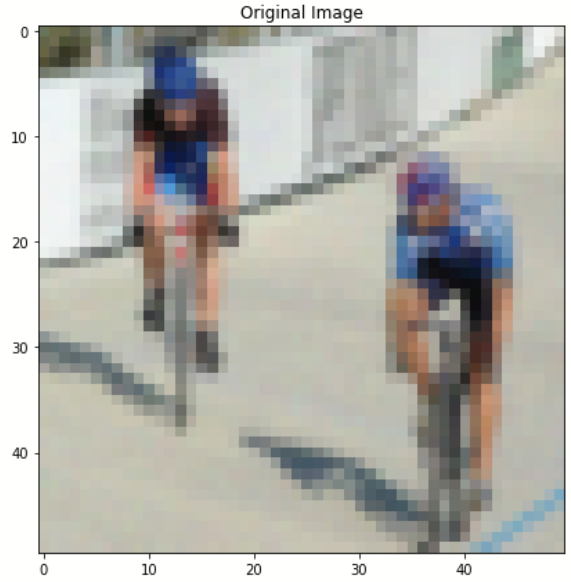


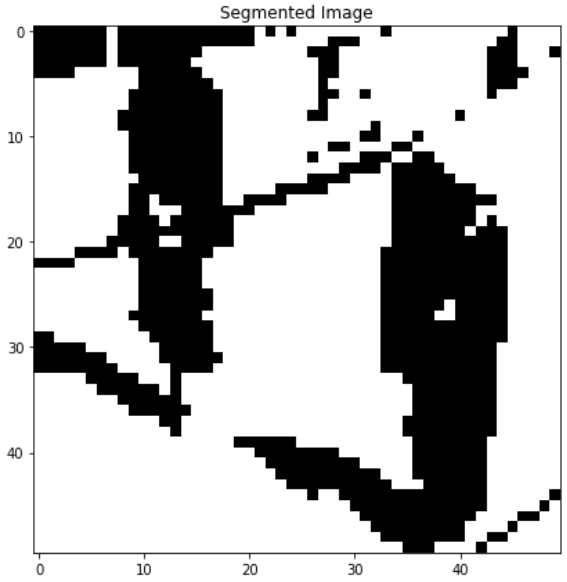
test4.jpg (downscaled - (50*50))





test5.jpg (downscaled - (50*50))





- ## N-Cut for segmenting color Images

- Feature vector of a pixel used for calculating weight
  F(i) = [v, v*s*sin(h),v*s*cos(h)]
  h,s,v are hsv values of the pixel
- Each element (W[i][j]) in the weight matrix is calculated using feature proximity and spatial proximity terms.
  Spatial_proximity [i][j] = `exp(-(distance(i,j)**2)/sigma_x)`
  Feature_proximity [i][j] = `exp(-(norm(F(i) - F(j))**2)/sigma_i)`

  if **`distance(i,j) < r`**
  `W[i][j] =` **Feature_proximity [i][j] * (Spatial_proximity [i][j])**

  else
  W[i][j] = 0

  r-> cutoff distance
- Calculated degree matrix (D) as follows
  `D = np.diag(W.sum(axis = 1))`
- `Calculated eigen vector corresponding to the second smallest eigenvalue for the following generalised eigen system-`
  `(D-W)*y = lambda*D*y`
- `Used threshold = 0 for eigen vector`

- Parameters used-

  R = 50

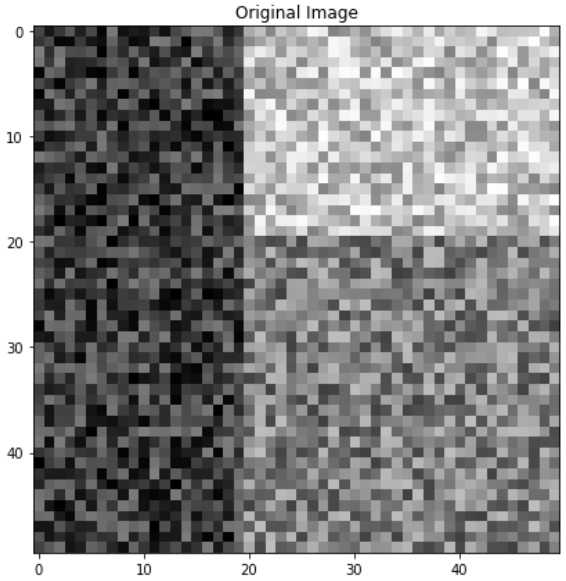
  Sigma_i = 10

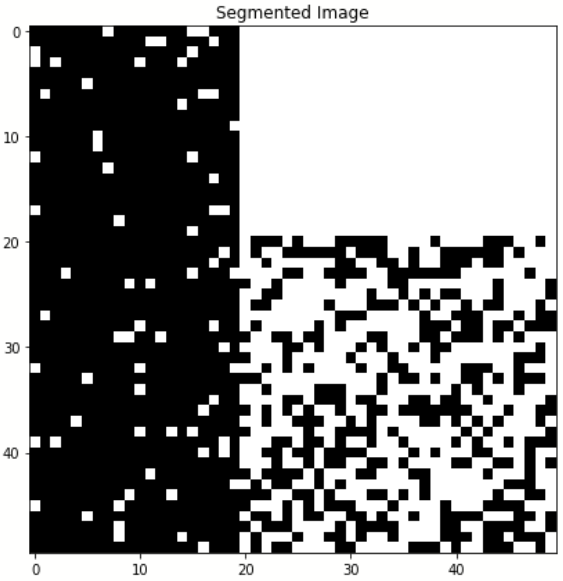
  Sigma_x = 1e7
- Downscaled the images to 50*50 for faster computation

Results:

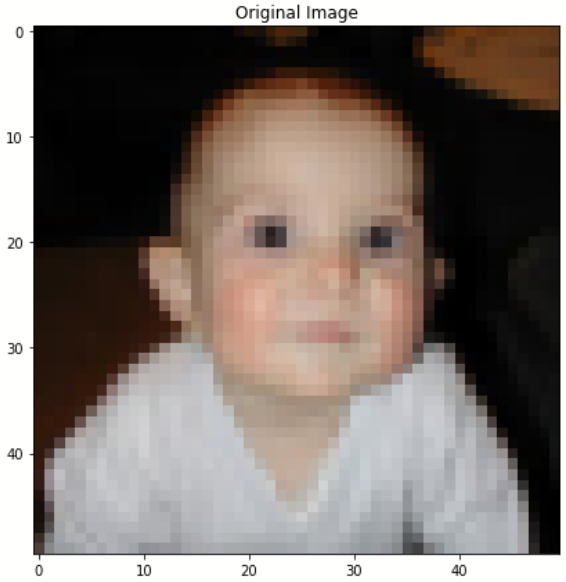
i) **N-Cut on Original Image (Downscaled)**

test1.png (downscaled - (50*50))

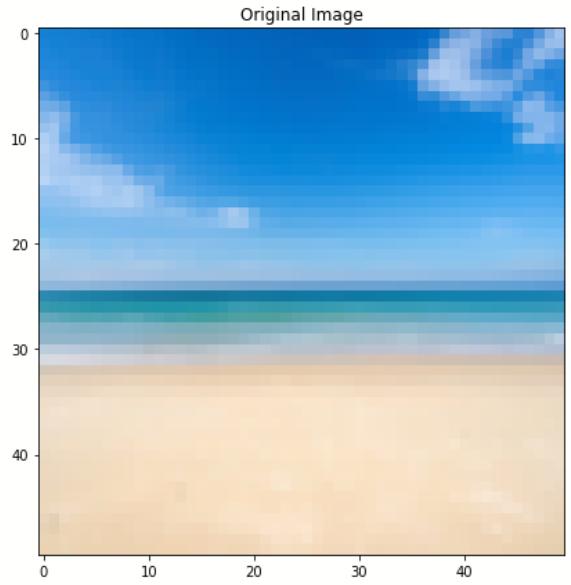


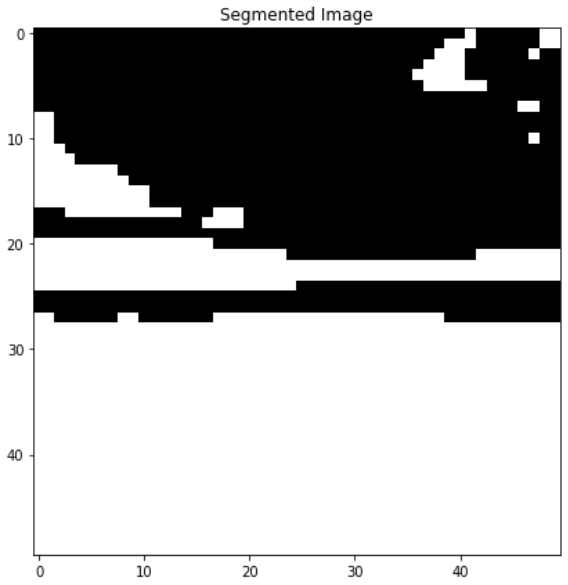


test2.jpg (downscaled - (50*50))

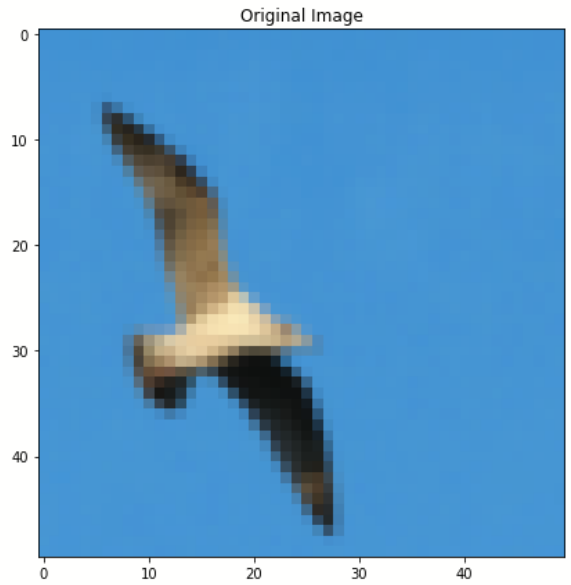


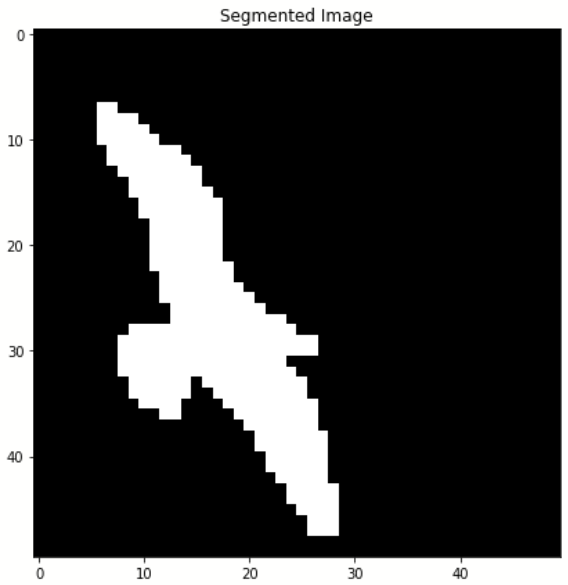
test3.jpg (downscaled - (50*50))



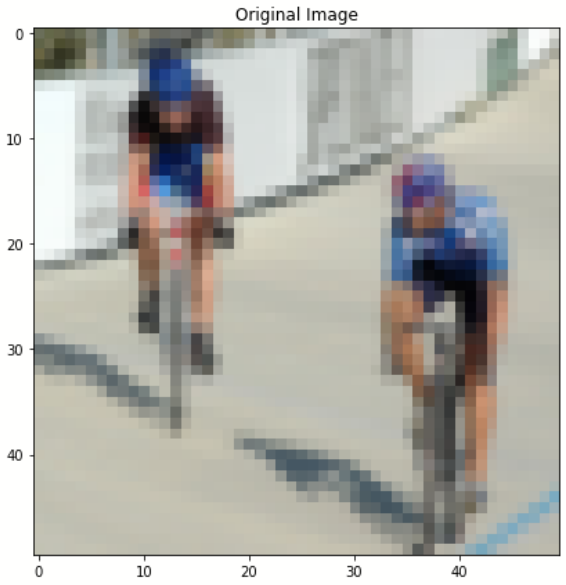


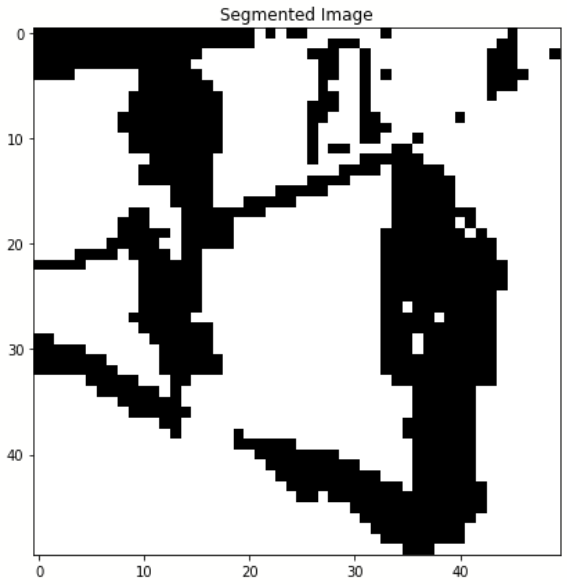
test4.jpg (downscaled - (50*50))





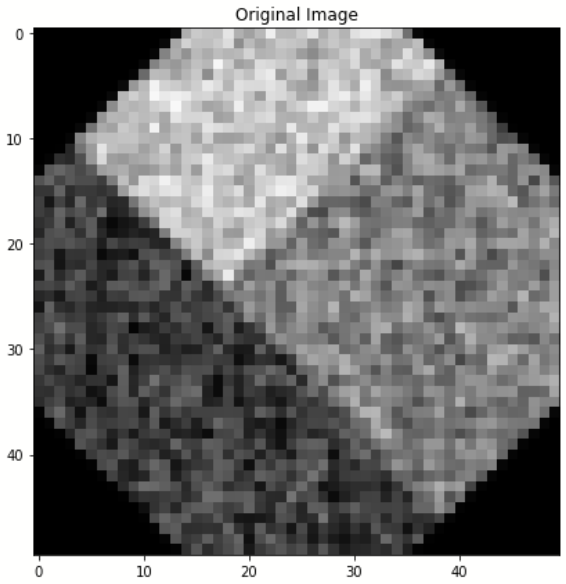
test5.jpg (downscaled - (50*50))

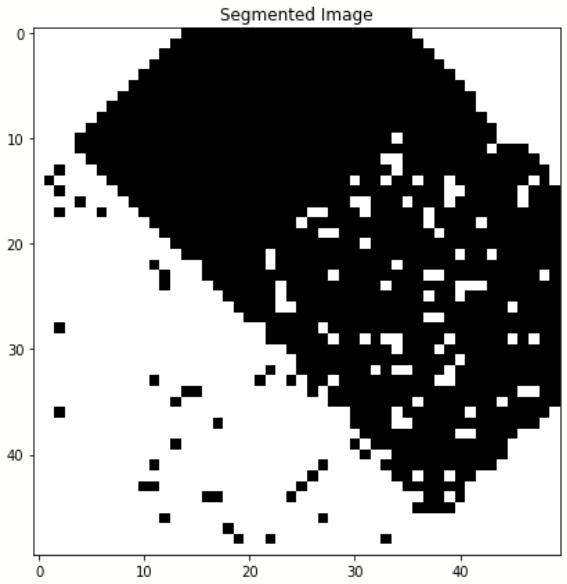




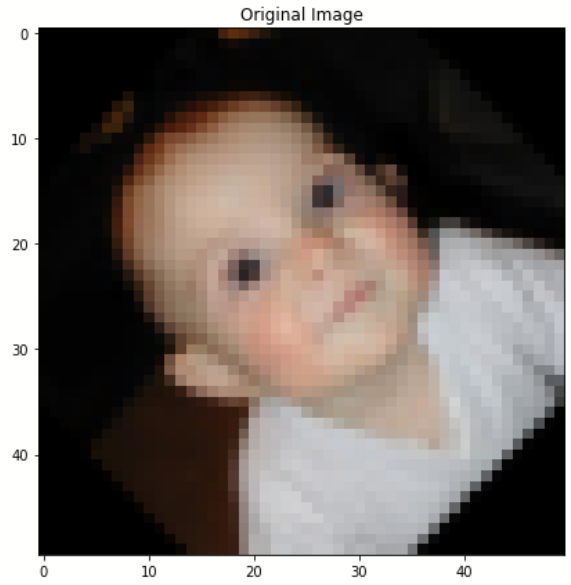
ii) **N-Cut on Rotated Image (Downscaled)**

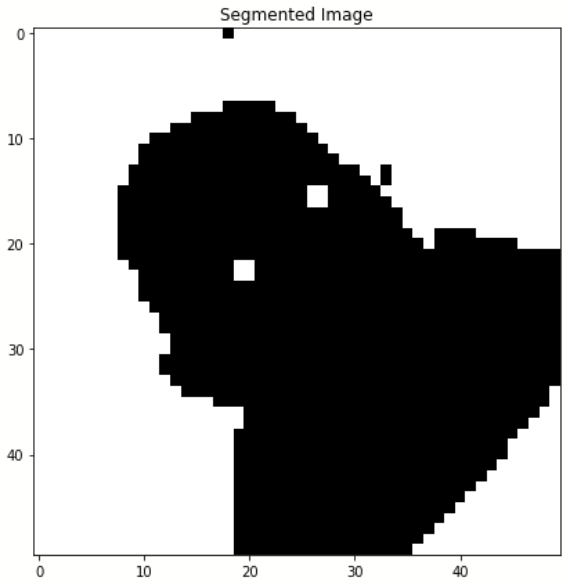
test1.png (downscaled - (50*50))



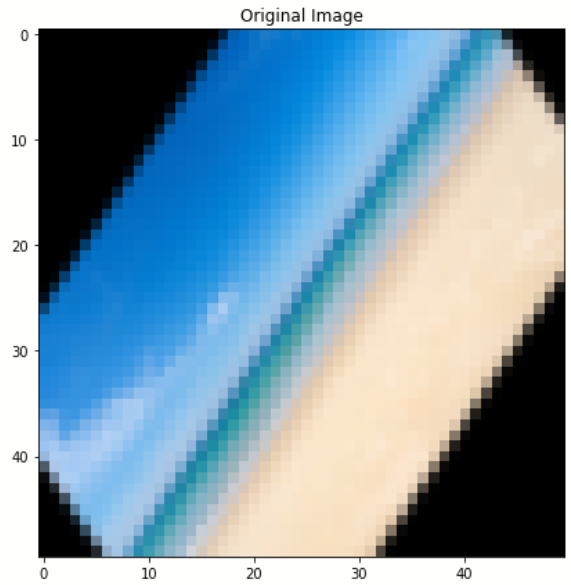


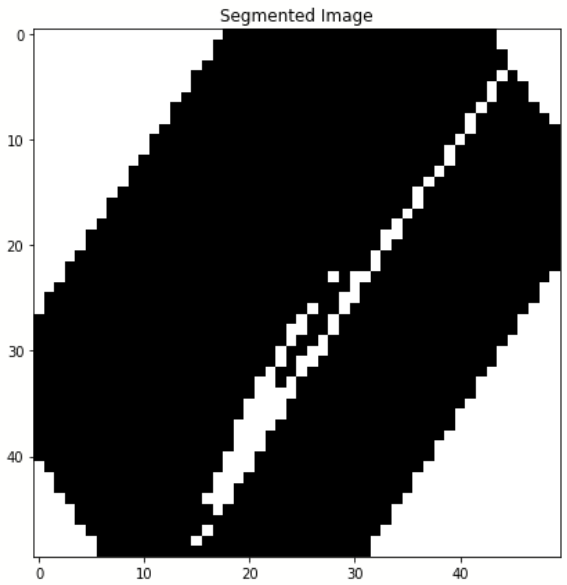
test2.jpg (downscaled - (50*50))



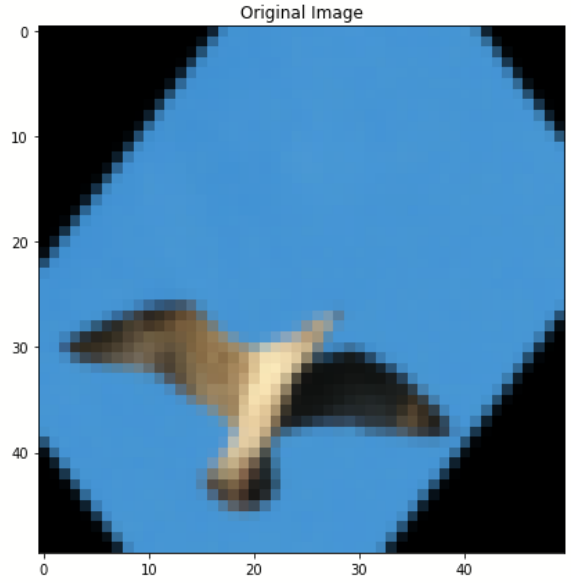


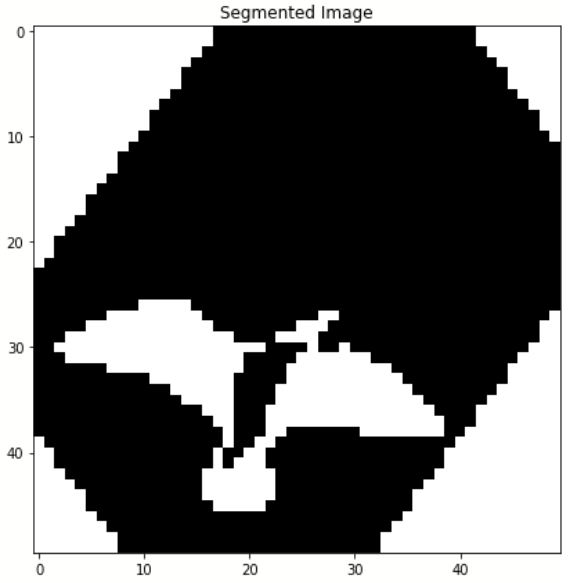
test3.jpg (downscaled - (50*50))



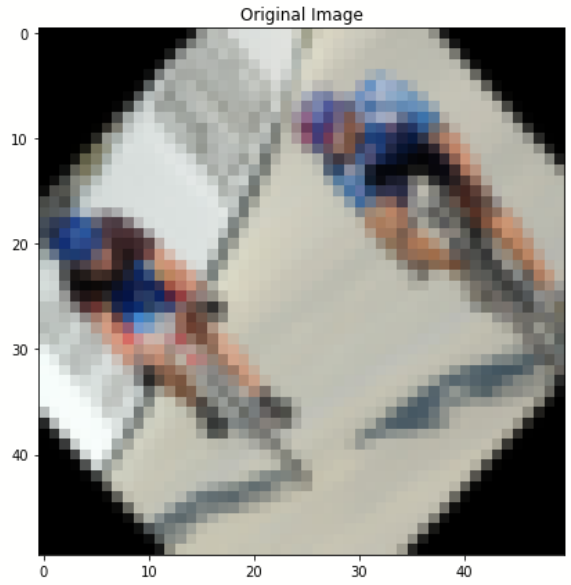


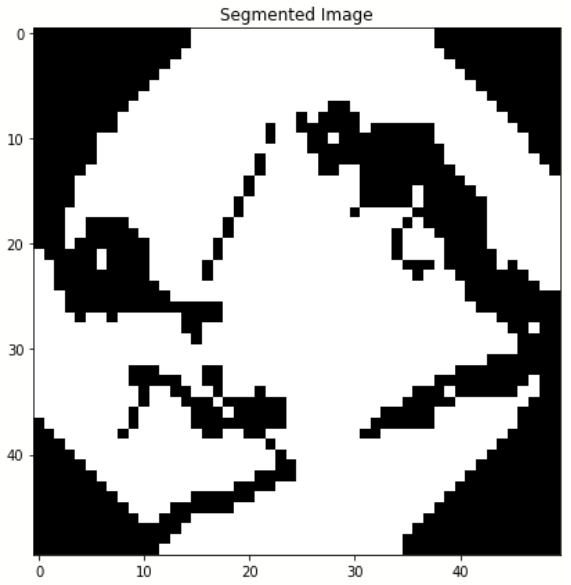
test4.jpg (downscaled - (50*50))





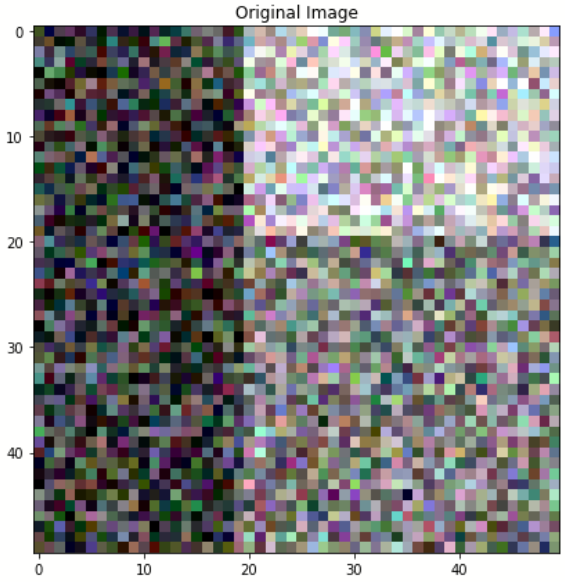
test5.jpg (downscaled - (50*50))

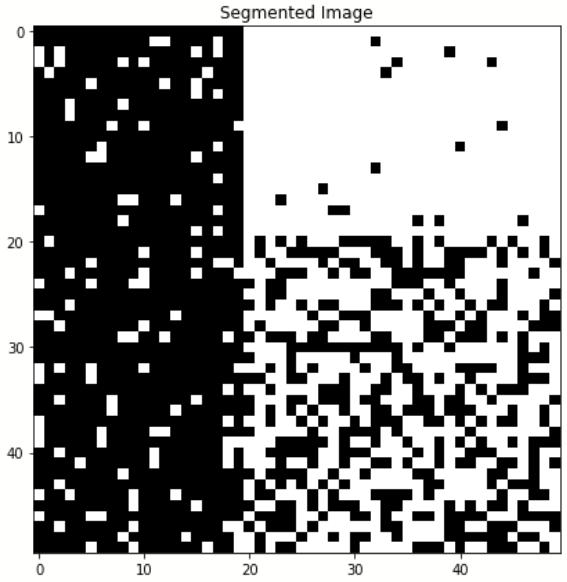




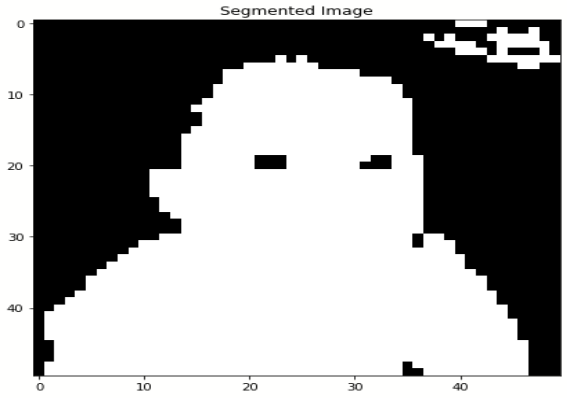
iii) **N-Cut on Gaussian Blurred Image (Downscaled)**

test1.png (downscaled - (50*50))

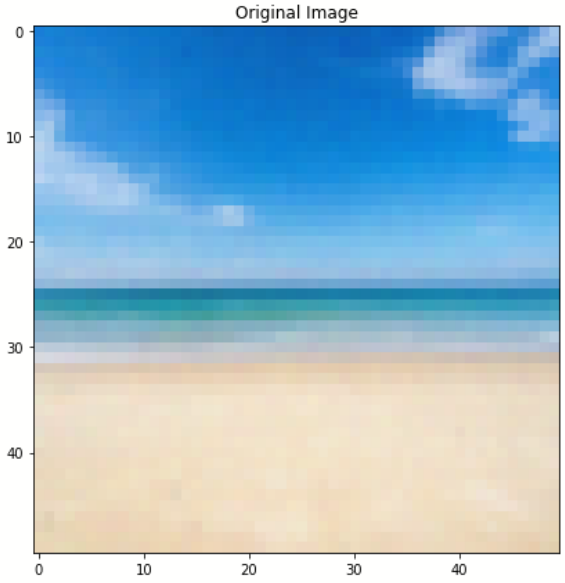


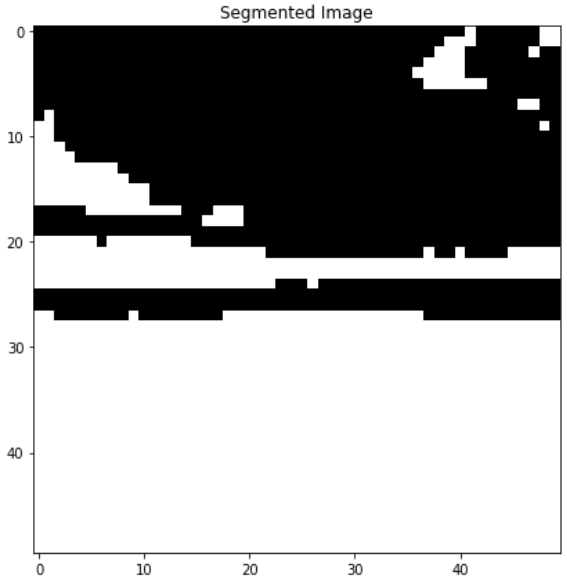


test2.jpg (downscaled - (50*50))

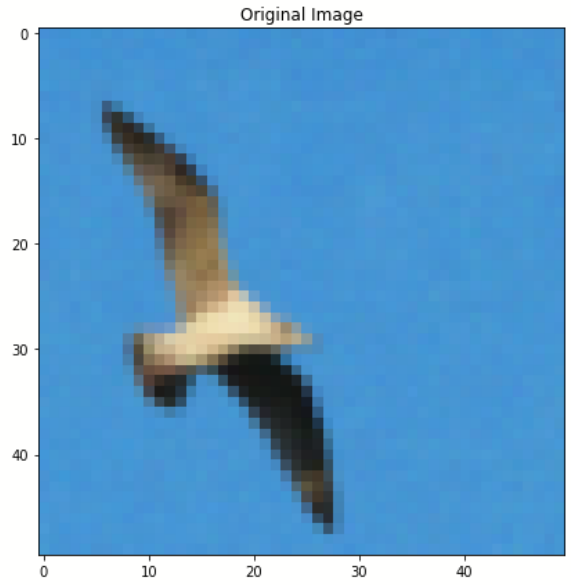


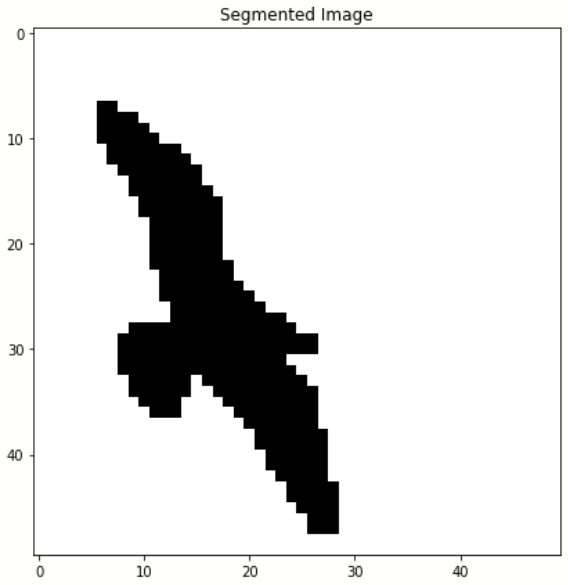
test3.jpg (downscaled - (50*50))



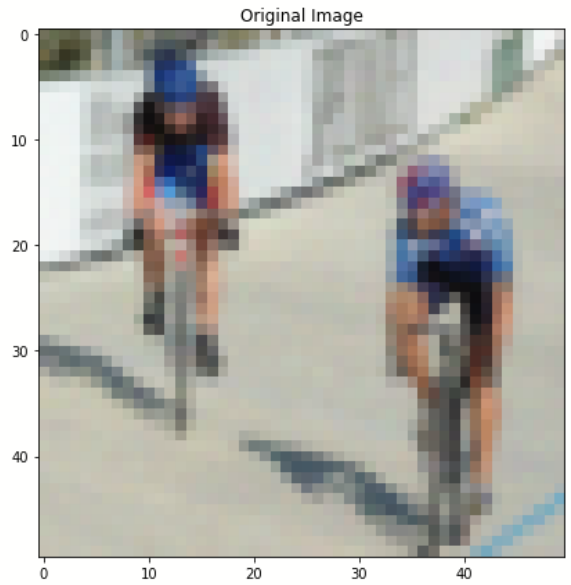


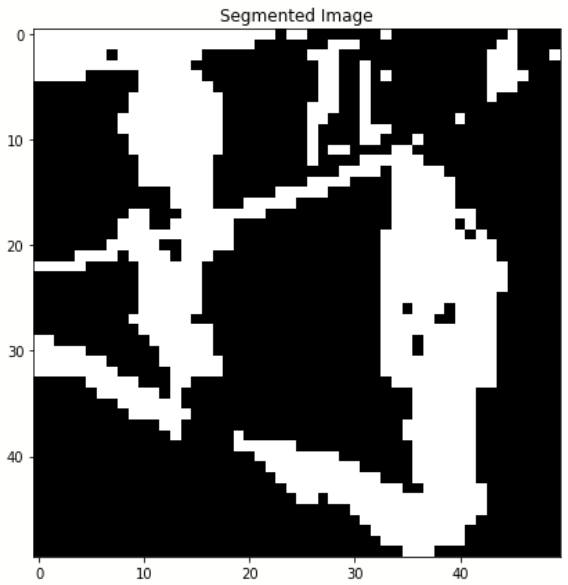
test4.jpg (downscaled - (50*50))





test5.jpg (downscaled - (50*50))





Q2) Fully Convolutional Networks

- *Image segmentation using ResNet50 based FCN*

- Created a custom dataset class (SegmentationDataset)for the given dataset and implemented __init__, len and get_item methods in the class
- Split the trainval dataset into train and val data using the provided train.txt and val.txt entries
- Applied following data transformations on images and masks-

  Image - Transforms

  ```
  transforms.Resize((520,520)),
  transforms.ToTensor(),
  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,0.225])
  ```

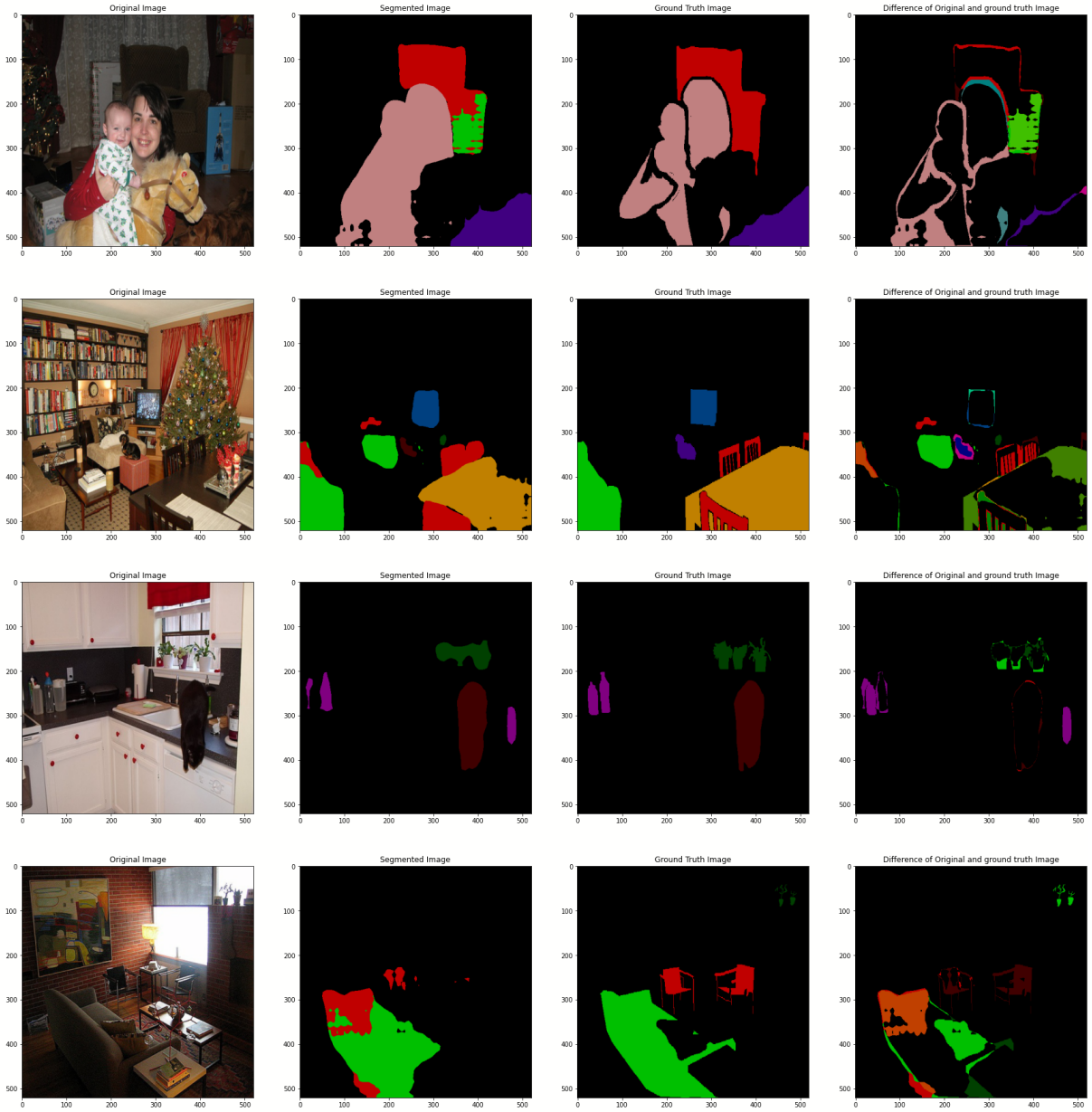
  Mask - Transforms

  ```
  resize_transform =
  transforms.Resize((520,520),interpolation=InterpolationMode.NEAREST)
  convert_tensor = transforms.PILToTensor()
  ```

  Note: InterpolationMode.NEAREST is used while resizing the image, so that colour corresponding to a label does not change in the mask.
- Loaded fcn_resnet50 model with pretrained weights FCN_ResNet50_Weights.DEFAULT (weights obtained by training on COCO datasets)
- Tested the model on the provided test dataset and calculated the Pixel_accuracy and MIOU

- **Results:**
- Obtained the following results on the provided test dataset-
  - **Pixel Accuracy** - 87.92256797689491%
  - **MIOU** - 92.14278211168683%

- Visualisation of few outputs of model



- ***Image Segmentation using FCN (MobileNetv2 backbone)***
- Created a custom dataset class (SegmentationDataset)for the given dataset and implemented __init__, len and get_item methods in the class
- Split the trainval dataset into train and val data using the provided train.txt and val.txt entries
- Applied following data transformations on images and masks- Image - Transforms

```
transforms.Resize((520,520)),
transforms.ToTensor(),
transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,0.225])

Mask - Transforms
resize_transform =
transforms.Resize((520,520),interpolation=InterpolationMode.NEARE
ST)
convert_tensor = transforms.PILToTensor()
```

Note: InterpolationMode.NEAREST is used while resizing the image, so that colour corresponding to a label does not change in the mask.

- Created a FCN using MobileNetV2 as backbone ,removing the final fc layer and then adding 5 upsample layers to get a final output of size [21, 224, 224] (21- classes, (224,224) - H*W of input image)
  Note-1: Each Upsample layer upscales the input by a factor of 2 using nn.ConvTranspose2d module.
  Note-2: Final convolution layer of MobileNetV2 output size is 7*7(h*w) when input size is 224*224(h*w), so we upsample it 5 times.
- Weights of the nn.ConvTranspose2d are initialised using bilinear interpolation. (Observed that initialising weights in this way gives much better results compared to random initialization.
- Model Summary-

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 112, 112]             864
       BatchNorm2d-2         [-1, 32, 112, 112]              64
             ReLU6-3         [-1, 32, 112, 112]               0
            Conv2d-4         [-1, 32, 112, 112]             288
       BatchNorm2d-5         [-1, 32, 112, 112]              64
             ReLU6-6         [-1, 32, 112, 112]               0
            Conv2d-7         [-1, 16, 112, 112]             512
       BatchNorm2d-8         [-1, 16, 112, 112]              32
  InvertedResidual-9         [-1, 16, 112, 112]               0
           Conv2d-10         [-1, 96, 112, 112]           1,536
      BatchNorm2d-11         [-1, 96, 112, 112]             192
            ReLU6-12         [-1, 96, 112, 112]               0
           Conv2d-13           [-1, 96, 56, 56]             864
      BatchNorm2d-14           [-1, 96, 56, 56]             192
```

ReLU6-15	[-1, 96, 56, 56]	0
Conv2d-16	[-1, 24, 56, 56]	2,304
BatchNorm2d-17	[-1, 24, 56, 56]	48
InvertedResidual-18	[-1, 24, 56, 56]	0
Conv2d-19	[-1, 144, 56, 56]	3,456
BatchNorm2d-20	[-1, 144, 56, 56]	288
ReLU6-21	[-1, 144, 56, 56]	0
Conv2d-22	[-1, 144, 56, 56]	1,296
BatchNorm2d-23	[-1, 144, 56, 56]	288
ReLU6-24	[-1, 144, 56, 56]	0
Conv2d-25	[-1, 24, 56, 56]	3,456
BatchNorm2d-26	[-1, 24, 56, 56]	48
InvertedResidual-27	[-1, 24, 56, 56]	0
Conv2d-28	[-1, 144, 56, 56]	3,456
BatchNorm2d-29	[-1, 144, 56, 56]	288
ReLU6-30	[-1, 144, 56, 56]	0
Conv2d-31	[-1, 144, 28, 28]	1,296
BatchNorm2d-32	[-1, 144, 28, 28]	288
ReLU6-33	[-1, 144, 28, 28]	0
Conv2d-34	[-1, 32, 28, 28]	4,608
BatchNorm2d-35	[-1, 32, 28, 28]	64
InvertedResidual-36	[-1, 32, 28, 28]	0
Conv2d-37	[-1, 192, 28, 28]	6,144
BatchNorm2d-38	[-1, 192, 28, 28]	384
ReLU6-39	[-1, 192, 28, 28]	0
Conv2d-40	[-1, 192, 28, 28]	1,728
BatchNorm2d-41	[-1, 192, 28, 28]	384
ReLU6-42	[-1, 192, 28, 28]	0
Conv2d-43	[-1, 32, 28, 28]	6,144
BatchNorm2d-44	[-1, 32, 28, 28]	64
InvertedResidual-45	[-1, 32, 28, 28]	0
Conv2d-46	[-1, 192, 28, 28]	6,144
BatchNorm2d-47	[-1, 192, 28, 28]	384
ReLU6-48	[-1, 192, 28, 28]	0
Conv2d-49	[-1, 192, 28, 28]	1,728
BatchNorm2d-50	[-1, 192, 28, 28]	384
ReLU6-51	[-1, 192, 28, 28]	0
Conv2d-52	[-1, 32, 28, 28]	6,144
BatchNorm2d-53	[-1, 32, 28, 28]	64
InvertedResidual-54	[-1, 32, 28, 28]	0
Conv2d-55	[-1, 192, 28, 28]	6,144
BatchNorm2d-56	[-1, 192, 28, 28]	384
ReLU6-57	[-1, 192, 28, 28]	0

Conv2d-58	[-1, 192, 14, 14]	1,728
BatchNorm2d-59	[-1, 192, 14, 14]	384
ReLU6-60	[-1, 192, 14, 14]	0
Conv2d-61	[-1, 64, 14, 14]	12,288
BatchNorm2d-62	[-1, 64, 14, 14]	128
InvertedResidual-63	[-1, 64, 14, 14]	0
Conv2d-64	[-1, 384, 14, 14]	24,576
BatchNorm2d-65	[-1, 384, 14, 14]	768
ReLU6-66	[-1, 384, 14, 14]	0
Conv2d-67	[-1, 384, 14, 14]	3,456
BatchNorm2d-68	[-1, 384, 14, 14]	768
ReLU6-69	[-1, 384, 14, 14]	0
Conv2d-70	[-1, 64, 14, 14]	24,576
BatchNorm2d-71	[-1, 64, 14, 14]	128
InvertedResidual-72	[-1, 64, 14, 14]	0
Conv2d-73	[-1, 384, 14, 14]	24,576
BatchNorm2d-74	[-1, 384, 14, 14]	768
ReLU6-75	[-1, 384, 14, 14]	0
Conv2d-76	[-1, 384, 14, 14]	3,456
BatchNorm2d-77	[-1, 384, 14, 14]	768
ReLU6-78	[-1, 384, 14, 14]	0
Conv2d-79	[-1, 64, 14, 14]	24,576
BatchNorm2d-80	[-1, 64, 14, 14]	128
InvertedResidual-81	[-1, 64, 14, 14]	0
Conv2d-82	[-1, 384, 14, 14]	24,576
BatchNorm2d-83	[-1, 384, 14, 14]	768
ReLU6-84	[-1, 384, 14, 14]	0
Conv2d-85	[-1, 384, 14, 14]	3,456
BatchNorm2d-86	[-1, 384, 14, 14]	768
ReLU6-87	[-1, 384, 14, 14]	0
Conv2d-88	[-1, 64, 14, 14]	24,576
BatchNorm2d-89	[-1, 64, 14, 14]	128
InvertedResidual-90	[-1, 64, 14, 14]	0
Conv2d-91	[-1, 384, 14, 14]	24,576
BatchNorm2d-92	[-1, 384, 14, 14]	768
ReLU6-93	[-1, 384, 14, 14]	0
Conv2d-94	[-1, 384, 14, 14]	3,456
BatchNorm2d-95	[-1, 384, 14, 14]	768
ReLU6-96	[-1, 384, 14, 14]	0
Conv2d-97	[-1, 96, 14, 14]	36,864
BatchNorm2d-98	[-1, 96, 14, 14]	192
InvertedResidual-99	[-1, 96, 14, 14]	0
Conv2d-100	[-1, 576, 14, 14]	55,296

BatchNorm2d-101	[-1, 576, 14, 14]	1,152
ReLU6-102	[-1, 576, 14, 14]	0
Conv2d-103	[-1, 576, 14, 14]	5,184
BatchNorm2d-104	[-1, 576, 14, 14]	1,152
ReLU6-105	[-1, 576, 14, 14]	0
Conv2d-106	[-1, 96, 14, 14]	55,296
BatchNorm2d-107	[-1, 96, 14, 14]	192
InvertedResidual-108	[-1, 96, 14, 14]	0
Conv2d-109	[-1, 576, 14, 14]	55,296
BatchNorm2d-110	[-1, 576, 14, 14]	1,152
ReLU6-111	[-1, 576, 14, 14]	0
Conv2d-112	[-1, 576, 14, 14]	5,184
BatchNorm2d-113	[-1, 576, 14, 14]	1,152
ReLU6-114	[-1, 576, 14, 14]	0
Conv2d-115	[-1, 96, 14, 14]	55,296
BatchNorm2d-116	[-1, 96, 14, 14]	192
InvertedResidual-117	[-1, 96, 14, 14]	0
Conv2d-118	[-1, 576, 14, 14]	55,296
BatchNorm2d-119	[-1, 576, 14, 14]	1,152
ReLU6-120	[-1, 576, 14, 14]	0
Conv2d-121	[-1, 576, 7, 7]	5,184
BatchNorm2d-122	[-1, 576, 7, 7]	1,152
ReLU6-123	[-1, 576, 7, 7]	0
Conv2d-124	[-1, 160, 7, 7]	92,160
BatchNorm2d-125	[-1, 160, 7, 7]	320
InvertedResidual-126	[-1, 160, 7, 7]	0
Conv2d-127	[-1, 960, 7, 7]	153,600
BatchNorm2d-128	[-1, 960, 7, 7]	1,920
ReLU6-129	[-1, 960, 7, 7]	0
Conv2d-130	[-1, 960, 7, 7]	8,640
BatchNorm2d-131	[-1, 960, 7, 7]	1,920
ReLU6-132	[-1, 960, 7, 7]	0
Conv2d-133	[-1, 160, 7, 7]	153,600
BatchNorm2d-134	[-1, 160, 7, 7]	320
InvertedResidual-135	[-1, 160, 7, 7]	0
Conv2d-136	[-1, 960, 7, 7]	153,600
BatchNorm2d-137	[-1, 960, 7, 7]	1,920
ReLU6-138	[-1, 960, 7, 7]	0
Conv2d-139	[-1, 960, 7, 7]	8,640
BatchNorm2d-140	[-1, 960, 7, 7]	1,920
ReLU6-141	[-1, 960, 7, 7]	0
Conv2d-142	[-1, 160, 7, 7]	153,600
BatchNorm2d-143	[-1, 160, 7, 7]	320

InvertedResidual-144	[-1, 160, 7, 7]	0
Conv2d-145	[-1, 960, 7, 7]	153,600
BatchNorm2d-146	[-1, 960, 7, 7]	1,920
ReLU6-147	[-1, 960, 7, 7]	0
Conv2d-148	[-1, 960, 7, 7]	8,640
BatchNorm2d-149	[-1, 960, 7, 7]	1,920
ReLU6-150	[-1, 960, 7, 7]	0
Conv2d-151	[-1, 320, 7, 7]	307,200
BatchNorm2d-152	[-1, 320, 7, 7]	640
InvertedResidual-153	[-1, 320, 7, 7]	0
Conv2d-154	[-1, 1280, 7, 7]	409,600
BatchNorm2d-155	[-1, 1280, 7, 7]	2,560
ReLU6-156	[-1, 1280, 7, 7]	0
ConvTranspose2d-157	**[-1, 512, 14, 14]**	**10,486,272**
BatchNorm2d-158	**[-1, 512, 14, 14]**	**1,024**
ReLU-159	**[-1, 512, 14, 14]**	**0**
UpsampleBlock-160	**[-1, 512, 14, 14]**	**0**
ConvTranspose2d-161	**[-1, 256, 28, 28]**	**2,097,408**
BatchNorm2d-162	**[-1, 256, 28, 28]**	**512**
ReLU-163	**[-1, 256, 28, 28]**	**0**
UpsampleBlock-164	**[-1, 256, 28, 28]**	**0**
ConvTranspose2d-165	**[-1, 128, 56, 56]**	**524,416**
BatchNorm2d-166	**[-1, 128, 56, 56]**	**256**
ReLU-167	**[-1, 128, 56, 56]**	**0**
UpsampleBlock-168	**[-1, 128, 56, 56]**	**0**
ConvTranspose2d-169	**[-1, 64, 112, 112]**	**131,136**
BatchNorm2d-170	**[-1, 64, 112, 112]**	**128**
ReLU-171	**[-1, 64, 112, 112]**	**0**
UpsampleBlock-172	**[-1, 64, 112, 112]**	**0**
ConvTranspose2d-173	**[-1, 21, 224, 224]**	**21,525**
BatchNorm2d-174	**[-1, 21, 224, 224]**	**42**
ReLU-175	**[-1, 21, 224, 224]**	**0**
UpsampleBlock-176	**[-1, 21, 224, 224]**	**0**

==========

Total params: 15,486,591

Trainable params: 13,262,719

Non-trainable params: 2,223,872

Input size (MB): 0.57

Forward/backward pass size (MB): 230.94

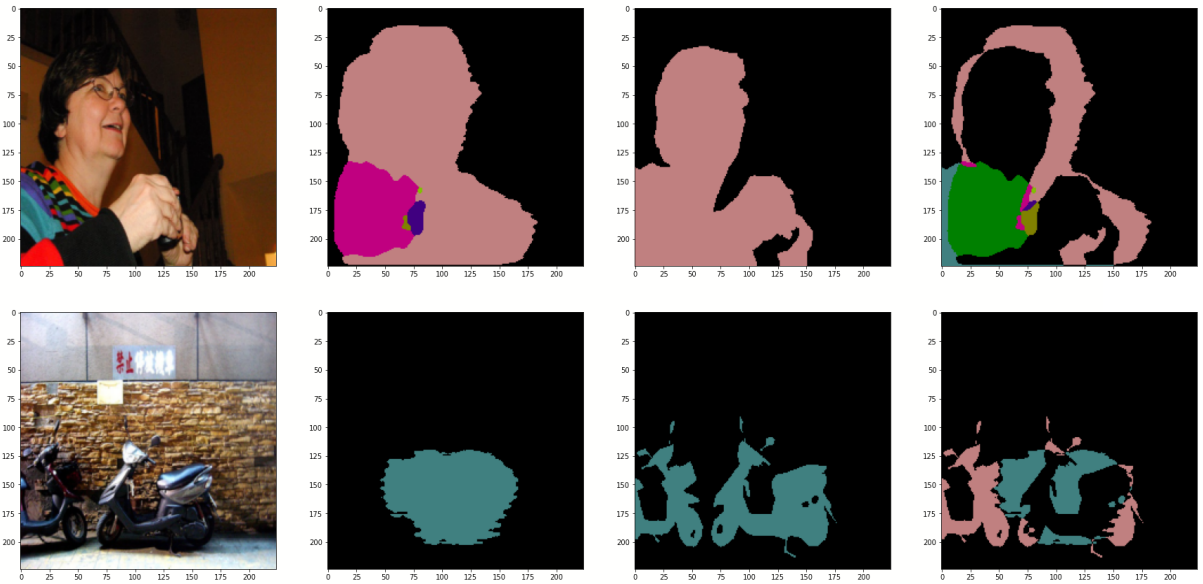
Params size (MB): 59.08

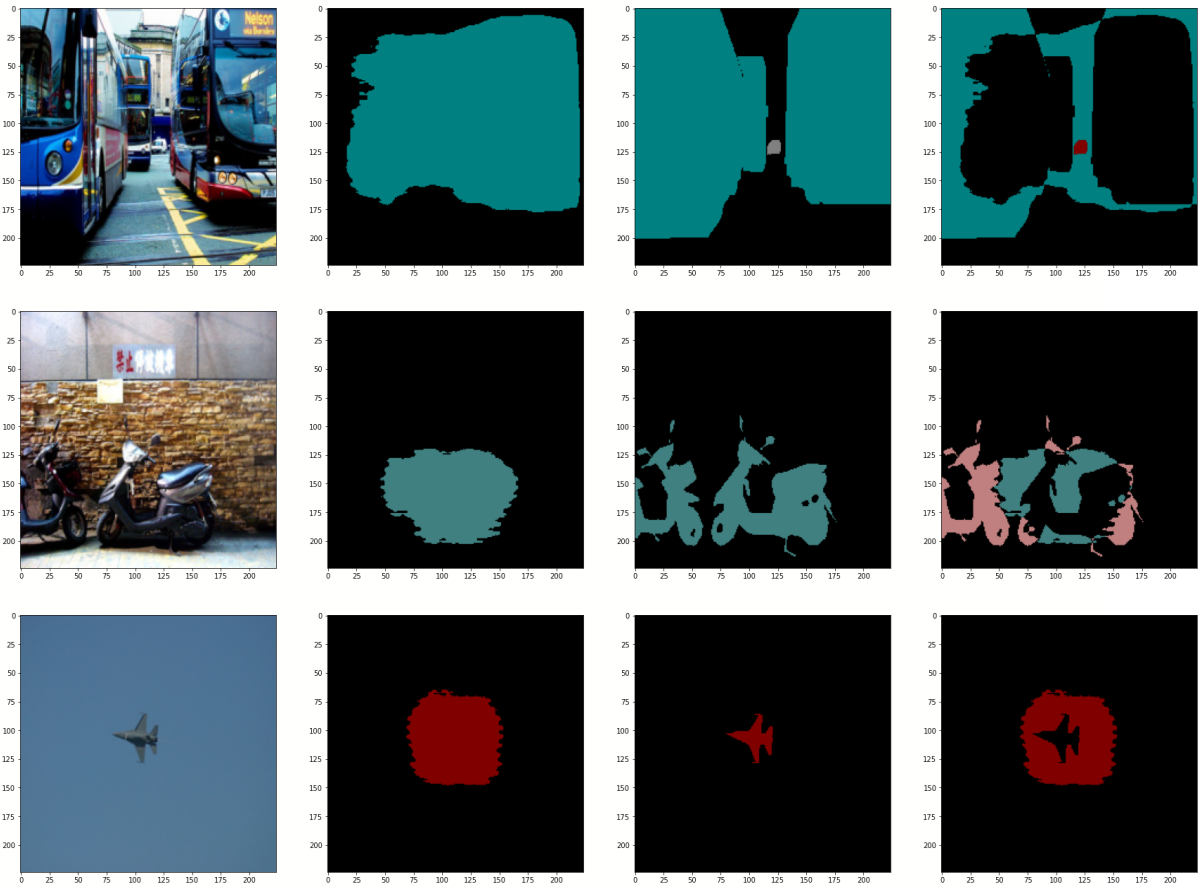
Estimated Total Size (MB): 290.59

Note: Layers added by me are in **bold**

- Trained the model on provided training data using the following hyperparameters
    - optimizer_ft = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
    - criterion = nn.CrossEntropyLoss(ignore_index=255)
    - exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=70, gamma=1)
    - Epochs - 150
- Tested the model on the provided test dataset and calculated the pixel accuracy and MIOU.

- **Results:**
- Obtained the following results on the provided test dataset-
    - **Pixel Accuracy** - 71.20890655369291%
    - **MIOU** - 81.37229567087272%
- Visualisation of few outputs of model-



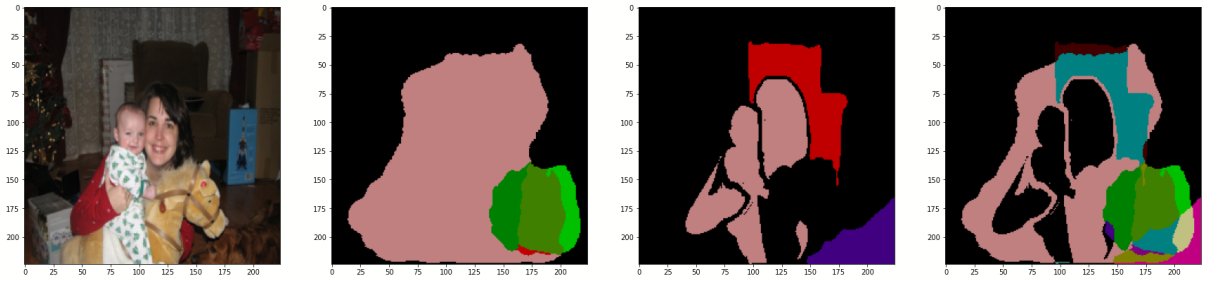


Note: 1st Image- Original Image
2nd Image- Model Output
3rd Image- Ground Truth
4th Image- Difference between Model Output and ground truth

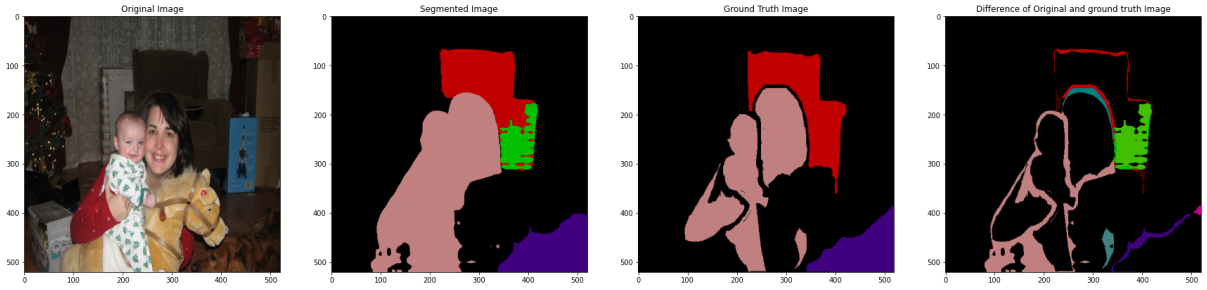
- ***Comparison Between the two Models***

- From the calculated model accuracies and visualised outputs, it is clear that FCN ResNet50 with pretrained weights is performing better than trained MobileNetV2 backbone FCN.
- Lower performance of trained MobileNetV2 backbone FCN can attributed to the following reasons
    - No Skip connections in the model(Adding skip connections will lead to much better performance of our model)
    - Insufficient data- ResNet50 FCN is trained on COCO ` Dataset which is very large compared to PASCAL VOC dataset (a subset of it)

- Comparison of few visualised outputs



Output of MobileNetV2 backbone FCN



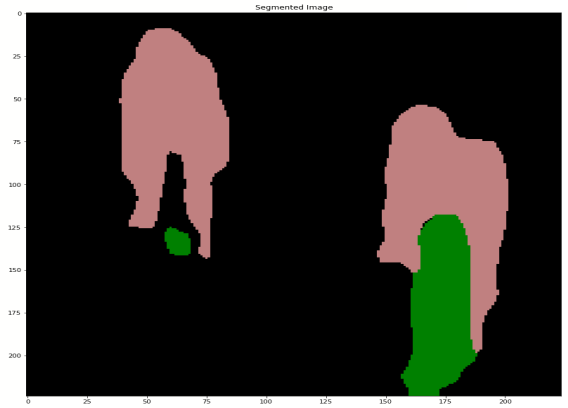
Output of ResNet50 FCN

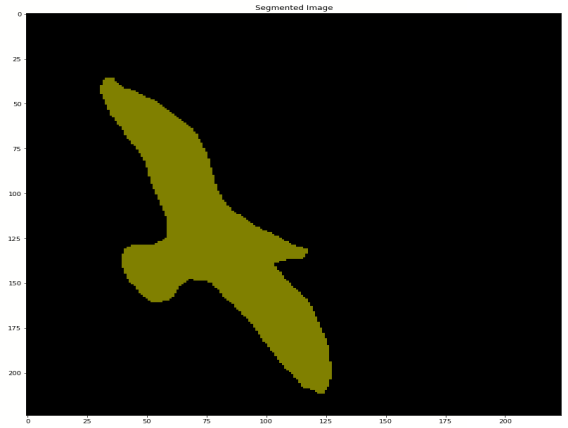
- ***Comparison between N-Cut and FCN'S***
- Compared N-cut and FCN's by segmenting test4.jpg and test5.jpg images using both N-cut and FCN's
- Observed that the segmented image by N-cut and ResNet50 FCN are good, but computation time for N-cut is much higher than ResNet50 FCN.
- MobileNetV2 backbone FCN did not perform well on these images due to the reasons mentioned in the previous section.

- Visualisation of segmented images by FCN's

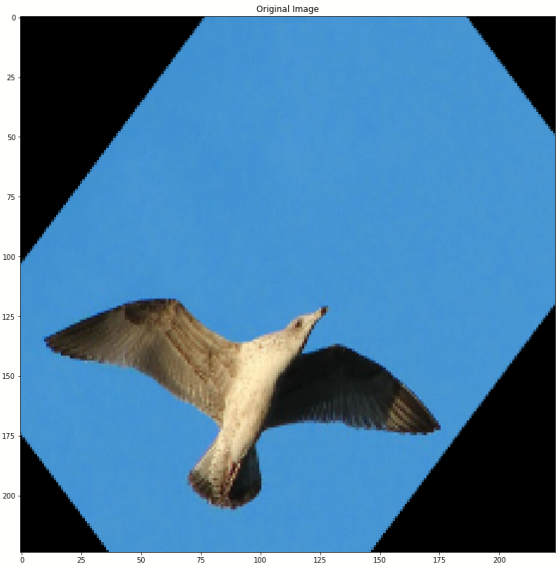
Outputs of ResNet-50 FCN

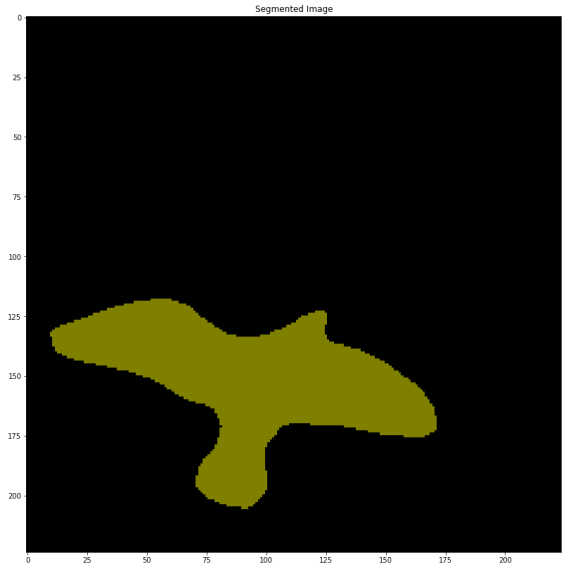
Original Images

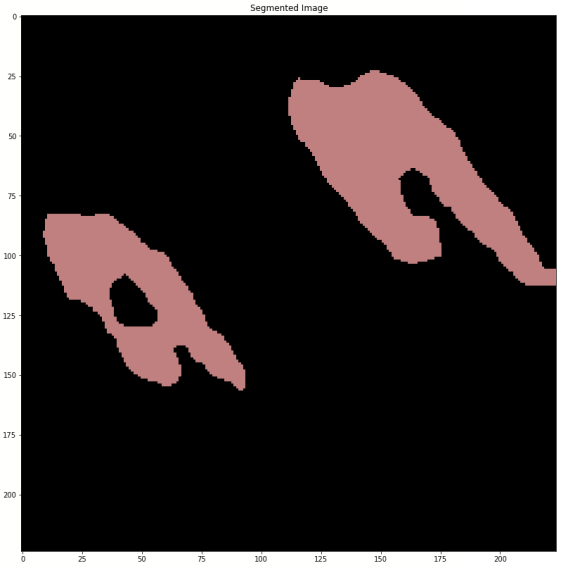




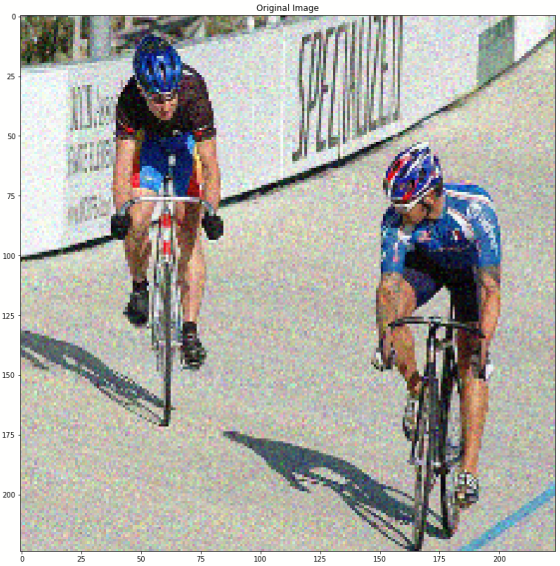
Rotated Images

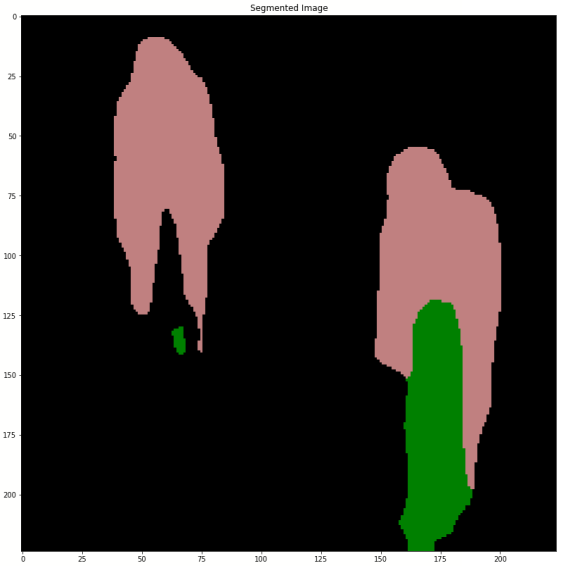


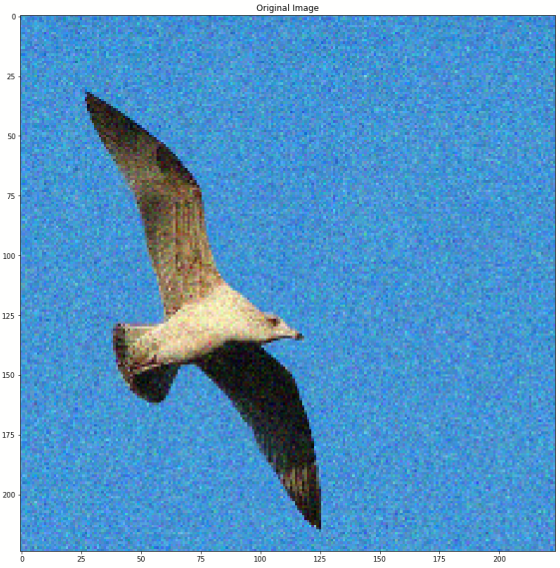


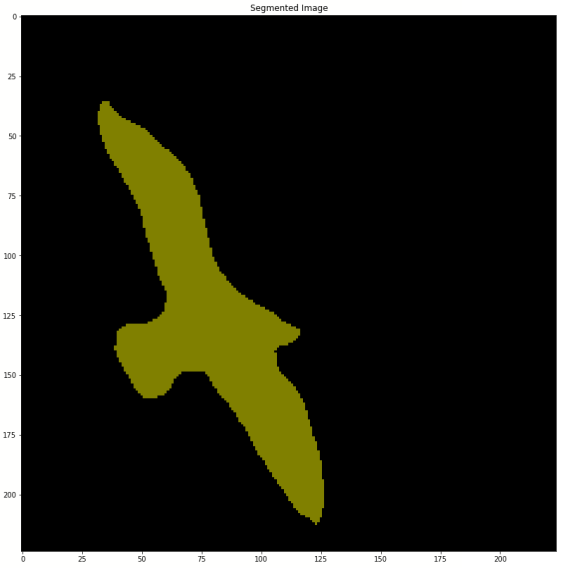


Noisy Images



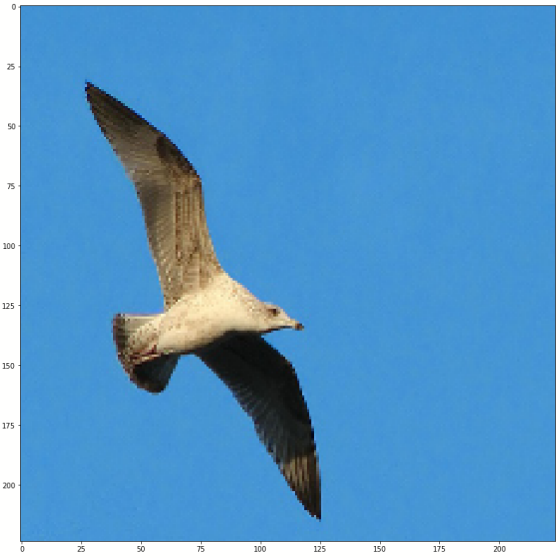
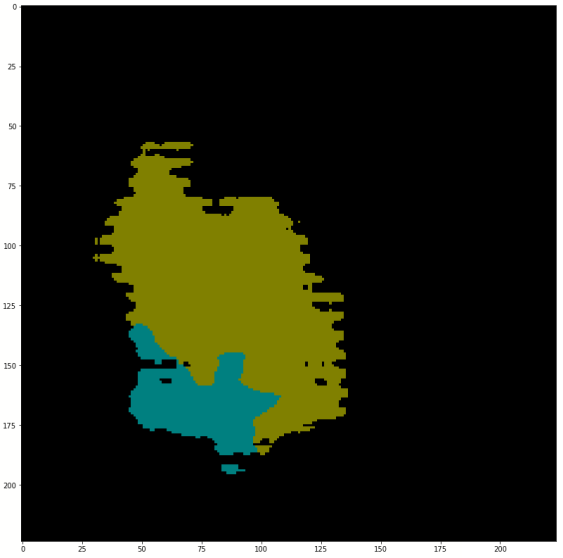


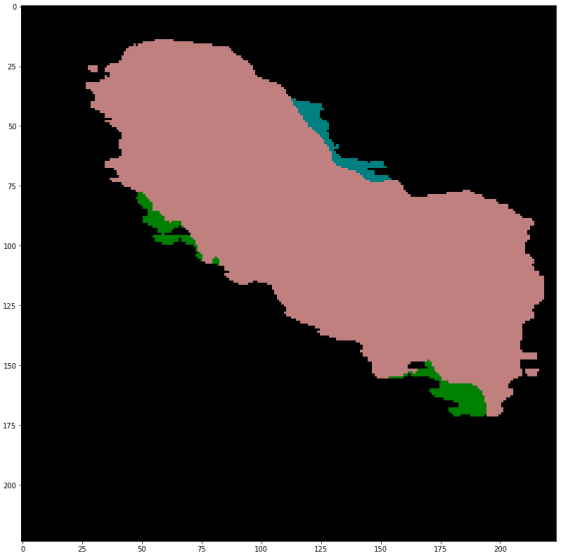




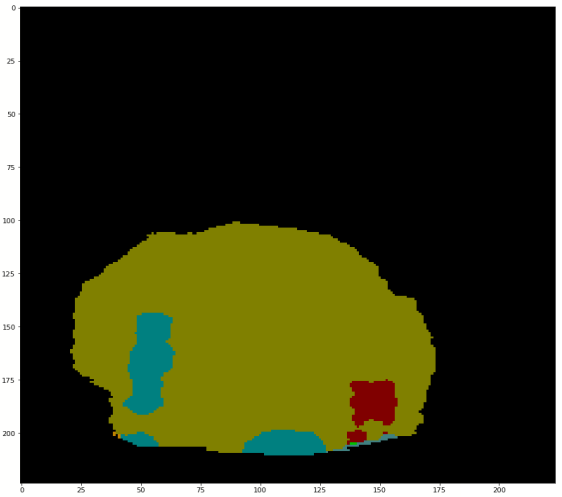
Outputs of MobileNetV2 Backbone FCN

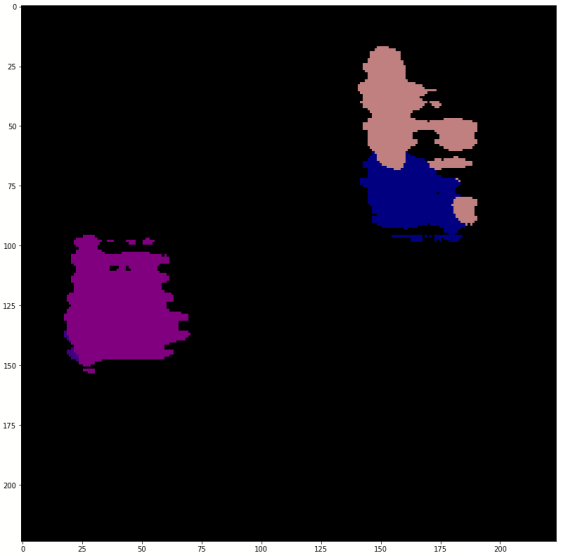
Original Images

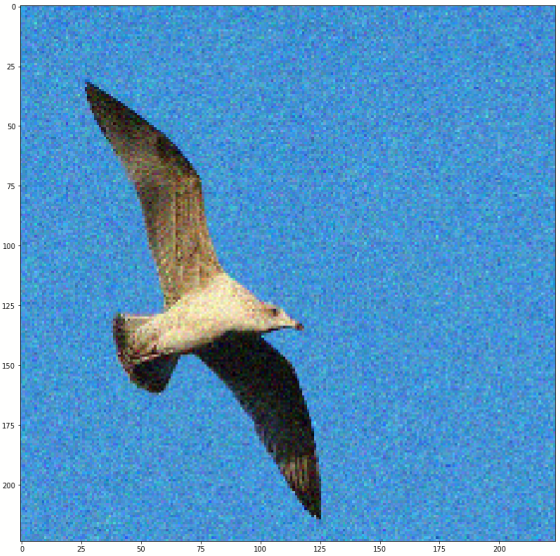


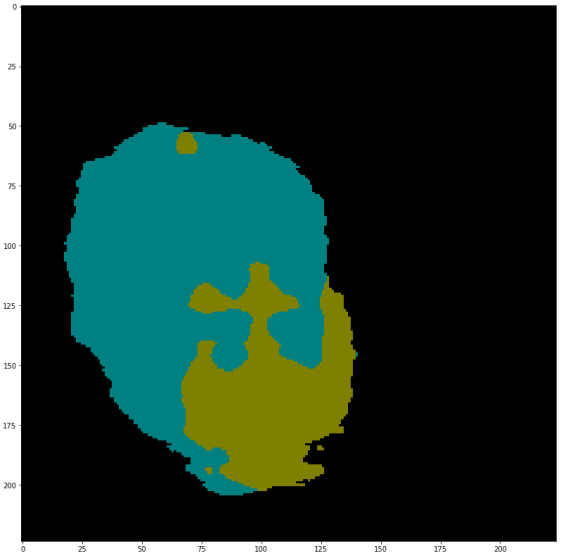
Rotated Images

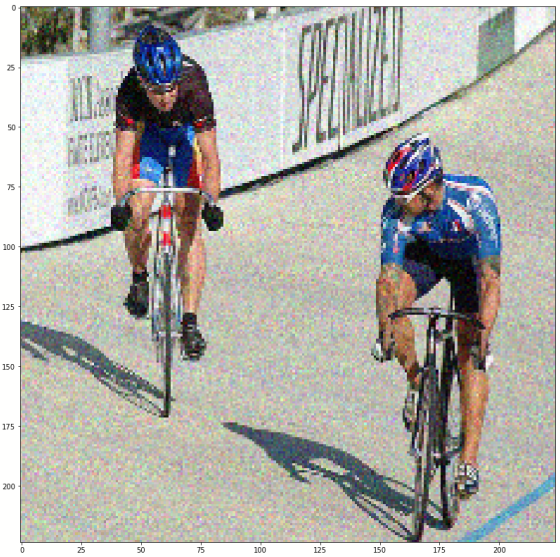



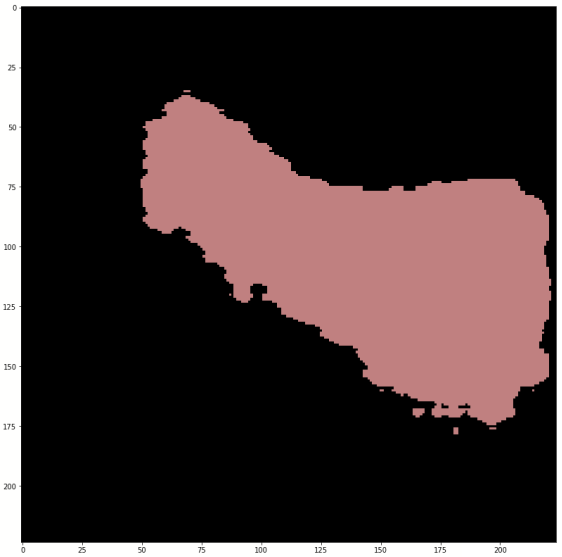


Noisy Images









References:

https://d2l.ai/chapter_computer-vision/fcn.html
https://github.com/sagieppel/Fully-convolutional-neural-network-FCN-for-semantic-segmentation-with-pytorch/blob/master/NET_FCN.py
https://poutyne.org/examples/semantic_segmentation.html
https://pytorch.org/vision/main/_modules/torchvision/datasets/voc.html#VOCSegmentation