ADVANCED IMAGE PROCESSING

Name: Avnish Kumar

S.No.: 20964

ASSIGNMENT-2

Q1. Details of Implementation of N-Cut:

- Normalized Cut is a classical way of Image Segmentation. Which uses Similarity Matrix which can be based on several factors used for segmentation like Intensity, Distance, Texture, Colour. here I have done two kind of implementation first implementation is based on Intensity only and second is based on Intensity and Distance both. also effect of rotation and addition of gaussian noise on segmentation is visualized using Matplotlib.
- The very first step of N-Cut is creating weight matrix based on different arguments (i.e., Intensity weight matrix and Distance weight matrix).

```
def intensity_weight_Matrix(img):

  weight = np.abs(np.float32(img.flatten()[:, np.newaxis]) - np.float32(img.flatten()[np.newaxis, :]))
  W = np.exp(-weight/10)*255
  return W
```

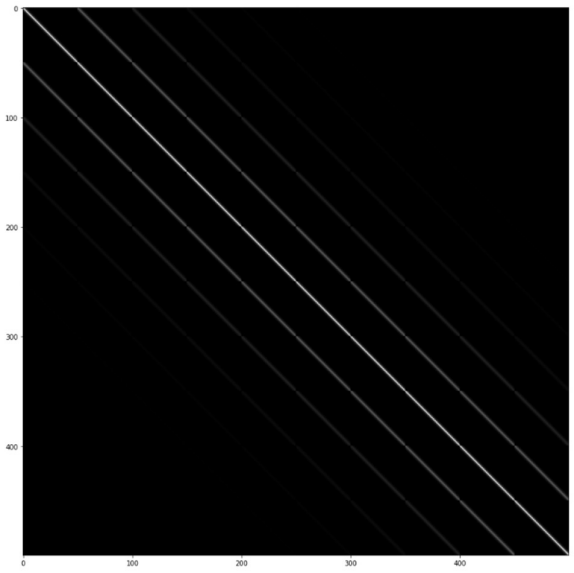
```
def positional_weight_Matrix(img):
  m,n = img.shape
  X, y = np.meshgrid(np.arange(m), np.arange(n))
  X = X.flatten()
  Y = y.flatten()

  distance = np.sqrt((X[:, np.newaxis] - X[np.newaxis, :])**2 + (Y[:, np.newaxis] - Y[np.newaxis, :])**2)
  W = np.exp(-distance/5)
  W =W*(W>0.58)
  return W
```

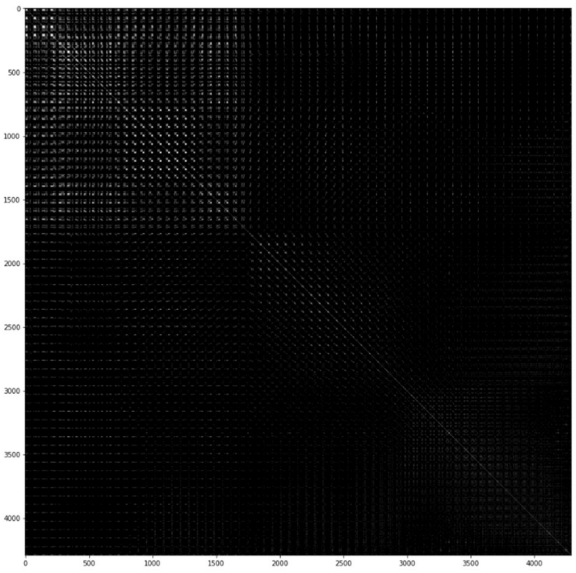
Here, newaxis attribute of numpy is used to create a 1-dimensional matrix vertically and horizontally such that their subtraction gives weight matrix with difference between each pixel *"similarity"*.

- Created Diagonal Matrix D and computed D – W. which will be used for eigen calculations.
- Also, weight matrix accounting for both intensity and distance is used for two similarity-based matrix.

Weight Matrix Based on Distance:



Weight Matrix Based on Intensity:

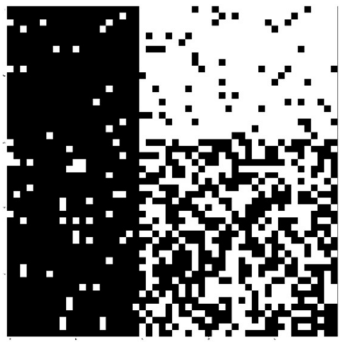


Effect of Rotation and Gaussian Noise addition:

Original Image

Rotated Image

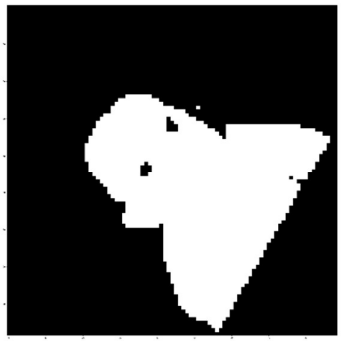
Noisy Image



Test Image: 1

Original Image

Rotated Image



Noisy Image

Test Image: 2

Original Image

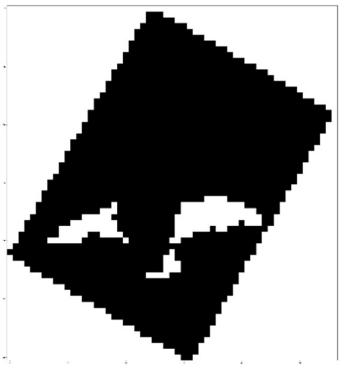
Rotated Image

Noisy Image

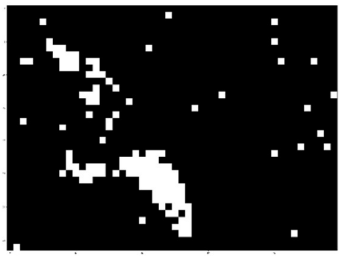
Test Image: 3

Original Image

Rotated Image



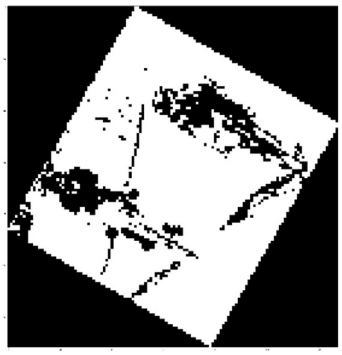
Noisy Image



Test Image: 4

Original Image

Rotated Image

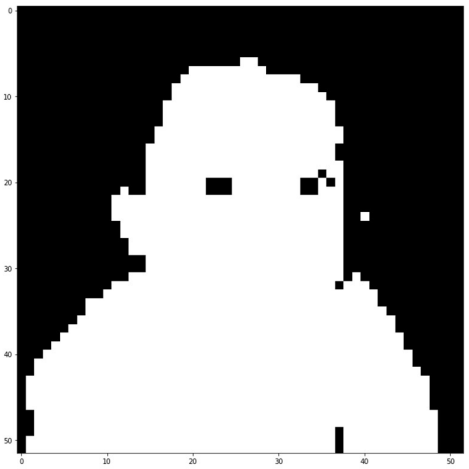


Noisy Image

Test Image: 5

EFFECT OF ENCORPORATING DISTANCE MEASURE:

Segmentation based on Intensity only:



Segmentation based on Intensity and Distance:

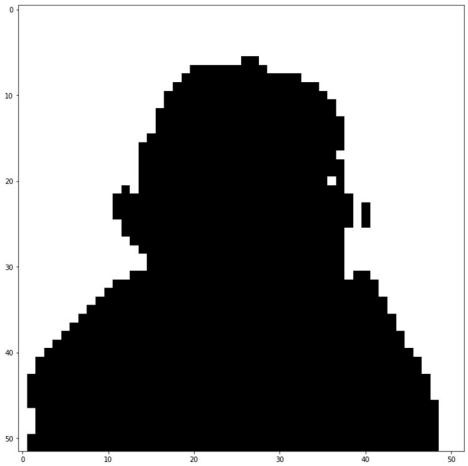


Figure: N-Cut using Intensity and colour similarity and distance measure

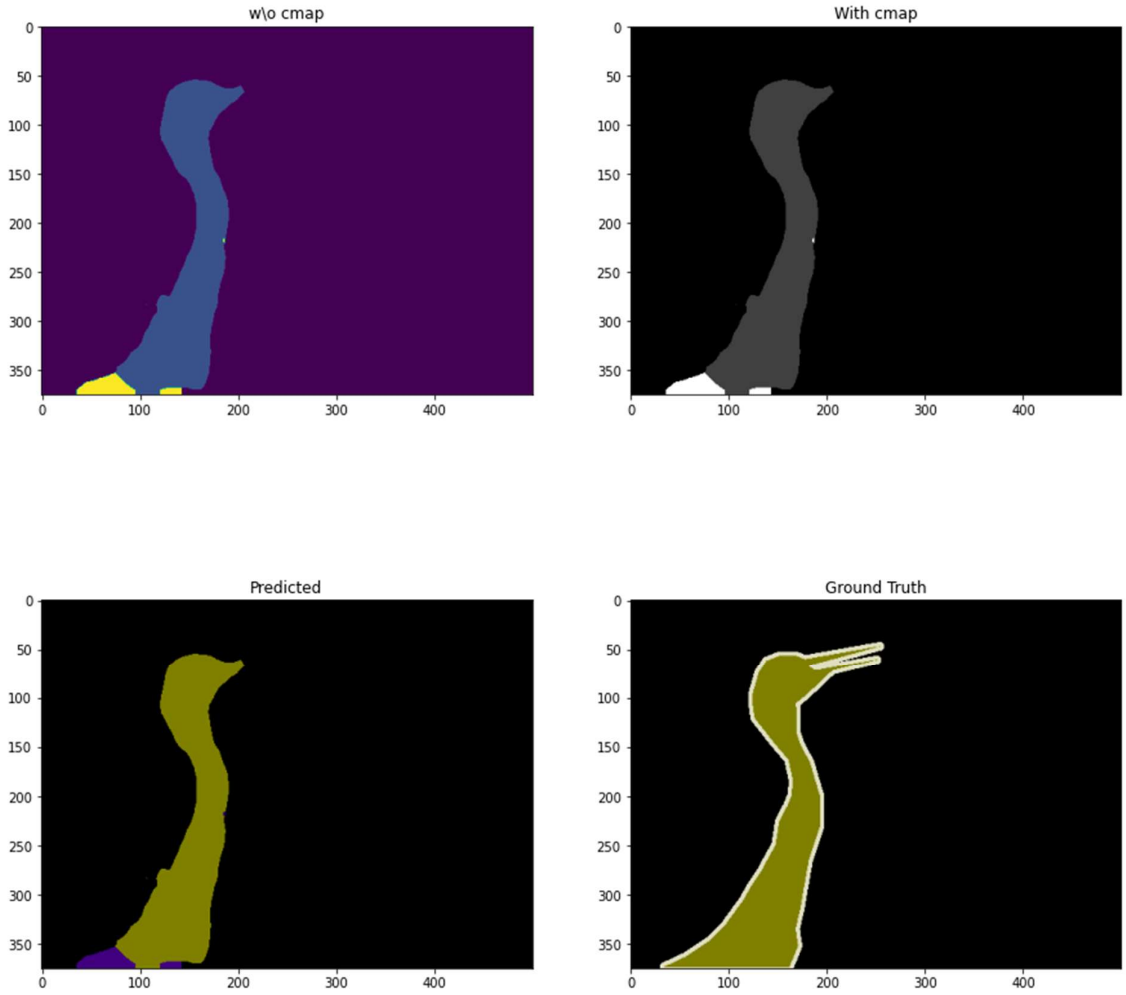
Incorporating distance along with Intensity difference of the pixels helps in segmentation of individual object as whole. in the example shown, we observe that eyes are separated as another segment when performing N-Cut using intensity difference only. but when used along with distance gives better segmentation.

Incorporating colour similarity along with intensity similarity and distance measure gives much better segmentation as the eyes and baby is segmented as a single entity which is desirable.

Also **Rotation** does not have much effect on the segmentation and **noise** addition causes slight distortions in the segmented image.

Q2.a

Result of Semantic segmentation on using Resnet50 FCN:



- At first we took pretrained fcn_resnet50 model.
- Defined transforms to be applied on image the took the output of the model and converted into color image using color palette in a very vectorised way:

```
# Make the prediction
model.eval()
with torch.no_grad():
    output = model(img_tensor)['out']

output = output.argmax(1).squeeze().detach().cpu().numpy()

# Create a color map for the segmentation labels
colors = np.array([ (0, 0, 0),  # background
                    (128, 0, 0), # aeroplane
                    (0, 128, 0), # bicycle
                    (128, 128, 0), # bird
                    (0, 0, 128), # boat
                    (128, 0, 128), # bottle
                    (0, 128, 128), # bus
                    (128, 128, 128), # car
                    (64, 0, 0), # cat
                    (192, 0, 0), # chair
                    (64, 128, 0), # cow
                    (192, 128, 0), # dining table
                    (64, 0, 128), # dog
                    (192, 0, 128), # horse
                    (64, 128, 128), # motorbike
                    (192, 128, 128), # person
                    (0, 64, 0), # potted plant
                    (128, 64, 0), # sheep
                    (0, 192, 0), # sofa
                    (128, 192, 0), # train
                    (0, 64, 128)]) #

colored_output = colors[output]
```

> Then defined mIoU metric and pixel accuracy metric for accuracy prediction of the model.

```
# Calculate pixel accuracy
pixel_acc = (output == label).astype(int).sum() / label.size
# print(f"Pixel Accuracy: {pixel_acc:.4f}")

# Calculate mIoU
iou = []
for class_id in colors:
    true_positives = ((output == class_id) & (label == class_id)).sum()
    false_positives = ((output == class_id) & (label != class_id)).sum()
    false_negatives = ((output != class_id) & (label == class_id)).sum()

    if true_positives + false_positives + false_negatives == 0:
        iou.append(0)
    else:
        iou.append(true_positives / (true_positives + false_positives + false_negatives))

miou = np.mean(iou)

# print("mIoU: {:.4f}".format(miou))
return pixel_acc, miou
# print("\nMaximum Intensity in Output of fcn_resnet50 after argmax: ",np.max(output))
```

Accuracy on Test Data:

- Average Pixel Accuracy over Test set is 90.6%
- Average mIoU over Test set is 84.3%

```
pixel_acc_cumulative = 0
mIoU_cumulative = 0
for image_name in tqdm(anno):
  a, b = prediction_accuracy(f"/content/PascalVOC/test/Images/{image_name}.jpg",f"/content/PascalVOC/test/Annotations/{image_name}.png", model)
  pixel_acc_cumulative = pixel_acc_cumulative + a
  mIoU_cumulative = mIoU_cumulative + b
print(f"Average Pixel Accuracy over Test set: {pixel_acc_cumulative/len(anno):.5f}")
print(f"Average mIoU over Test set: {mIoU_cumulative/len(anno):.5f}")
```

```
Average Pixel Accuracy over Test set: 0.90612
Average mIoU over Test set: 0.84320
```

Q2.b

- Took MobileNetv2 backbone pretrained then modified the last layer to have decoder structure because Mobile Net is an encoder and down-sample image.
- Used ConvTranspose2d to up-sample image.
- Then merged both layer using nn.sequential.

```
model = models.mobilenet_v2(pretrained=True)

# Define a new classifier
num_classes = 21

# Encoder
features = nn.Sequential(*list(model.children())[:-1])               # Extract layers up to and excluding the classifier


# Decoder
decoder = nn.Sequential(
    nn.ConvTranspose2d(1280, 320, kernel_size=3, stride=2, padding=1, output_padding=1, bias=False),
    nn.BatchNorm2d(320),
    nn.ReLU(),
    nn.ConvTranspose2d(320, 96, kernel_size=3, stride=2, padding=1, output_padding=1, bias=False),
    nn.BatchNorm2d(96),
    nn.ReLU(),
    nn.ConvTranspose2d(96, 32, kernel_size=3, stride=2, padding=1, output_padding=1, bias=False),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1, output_padding=1, bias=False),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.ConvTranspose2d(16, 21, kernel_size=3, stride=2, padding=1, output_padding=1, bias=False),
    nn.Sigmoid()
)

# Combining both:
########################################

model = nn.Sequential(features, decoder)


########################################
```

Changing 3 Channel to 1 Channel:

```python
def ch3_to_ch1(img):
  unique_color = np.unique(img.reshape(-1, img.shape[2]), axis = 0) #### Change the image to have colors in the columns and can have any numbers of rows
  img_new = np.zeros(img.shape[:2]).astype(np.uint8)

  for col in unique_color:

    locations = np.where(np.all(img==col,axis=2))                    #### Unique color's locations

    try:
      img_new[locations[0], locations[1]] = np.where(np.all(colors == col, axis=1))[0][0]  #### Filling Index of color palette(colors) at location

    except Exception as e:
      pass
  return img_new
```

Creating Custom DataSet for loading Image:

```python
class ImgDataset(Dataset):
  def __init__(self, img_dir, mask_dir, transform=None):
    self.img_dir = img_dir
    self.mask_dir = mask_dir
    self.transform = transform
    self.images = os.listdir(mask_dir)

  def __len__(self):
    return len(self.images)

  def __getitem__(self, index):
    img_path = os.path.join(self.img_dir, self.images[index].replace(".png", ".jpg"))
    mask_path = os.path.join(self.mask_dir, self.images[index].replace(".png", ".png"))
    image = np.array(Image.open(img_path).convert("RGB"))
    mask = np.array(Image.open(mask_path).convert("RGB"))
    mask[mask == 255.0] = 1.0

    if self.transform is not None:

      transform_img = transforms.Compose([transforms.ToTensor(),
                                          transforms.Resize((224, 224)),
                                                    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
      image = transform_img(image)

      mask = ch3_to_ch1(mask)
      transform_mask = transforms.Compose([transforms.ToTensor(),
                                           transforms.Resize((224, 224))])
      mask = transform_mask(mask)

    return image, mask
```

Training:

Training Loss was decreasing, So our model is working

```
training loss: 3.04173
training loss: 3.03322
training loss: 3.02451
training loss: 3.01626
training loss: 3.00852
training loss: 3.00096
training loss: 2.99340
training loss: 2.98600
training loss: 2.97863
training loss: 2.97111
training loss: 2.96344
training loss: 2.95589
training loss: 2.94809
training loss: 2.94025
training loss: 2.93251
```

Calculating Pixel Accuracy and mIoU on Test Images:

- Average Pixel Accuracy over Test set: 76.5%
- Average mIoU over Test set: 61.4%

```
pixel_acc_cumulative = 0
mIoU_cumulative = 0
for image_name in tqdm(anno):
  a, b = metric_calculation(f"/content/PascalVOC/test/Images/{image_name}.jpg",f"/content/PascalVOC/test/Annotations/{image_name}.png", model)
  pixel_acc_cumulative = pixel_acc_cumulative + a
  mIoU_cumulative = mIoU_cumulative + b
print(f"Average Pixel Accuracy over Test set: {pixel_acc_cumulative/len(anno):.5f}")
print(f"Average mIoU over Test set: {mIoU_cumulative/len(anno):.5f}")
```

```
100% 210/210 [01:04<00:00, 3.48it/s]
Average Pixel Accuracy over Test set: 0.76564
Average mIoU over Test set: 0.61378
```

Q2.c

Fcn Resnet model is end to end designed for Image Segmentation but MobileNetV2 is just encoder for image segmentation so MobileNetV2 is to be combined with decoder then trained on very less data. So, we observe The Performance of Fcn Resnet 50 is comparatively better than the modified MobilenetV2.

TEST DATA ACCURACY

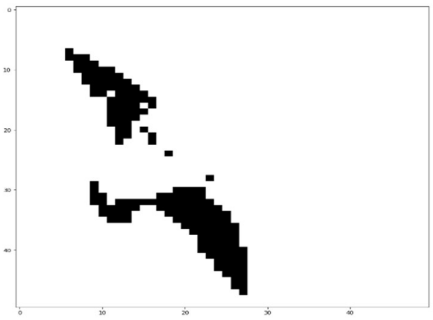
Performance of Fcn Resnet50: Pixel Accuracy: 90.6%

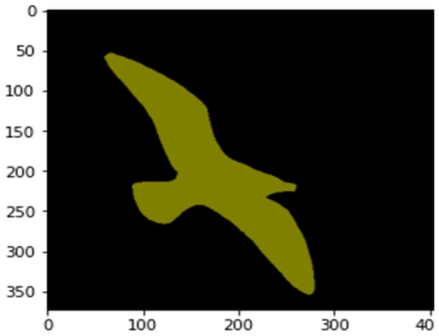
mIoU: 84.3%

Performance of modified MobileNetV2: Pixel Accuracy: 76.5%

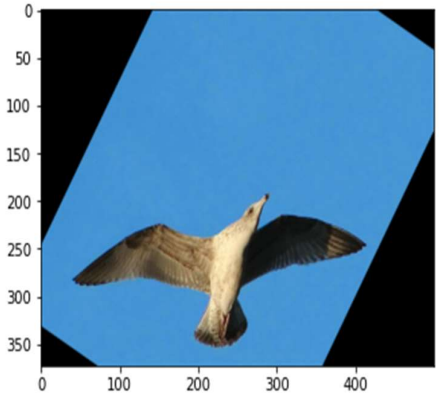
mIoU: 61.4%

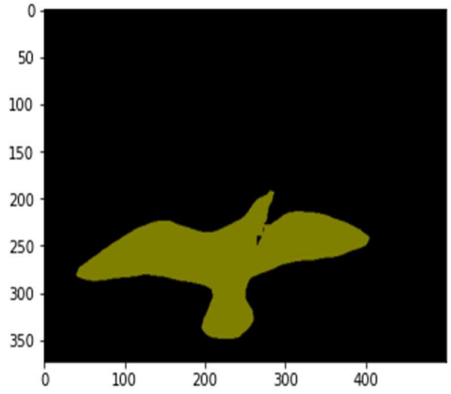
Test Image 4:



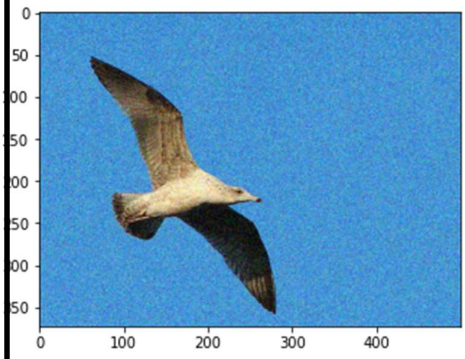


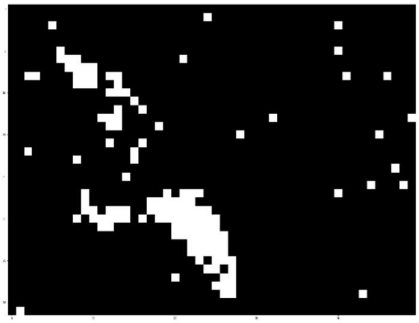
Rotated:

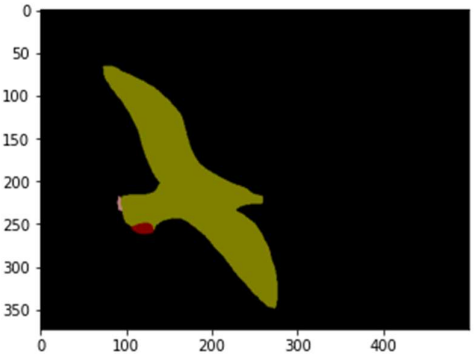




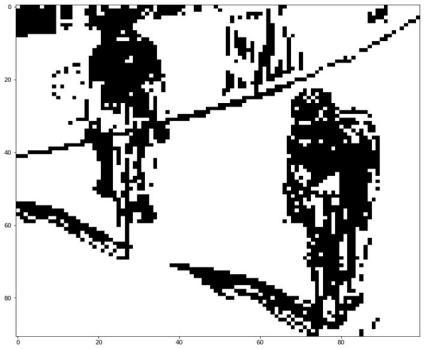
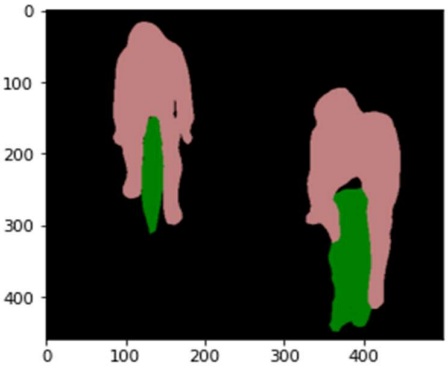
Noisy:



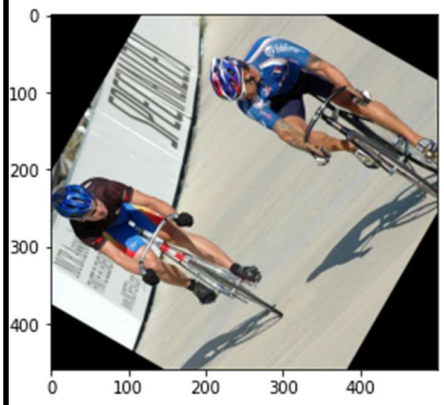


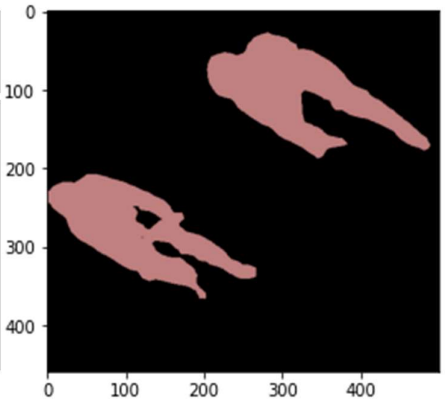


Test Image 5:

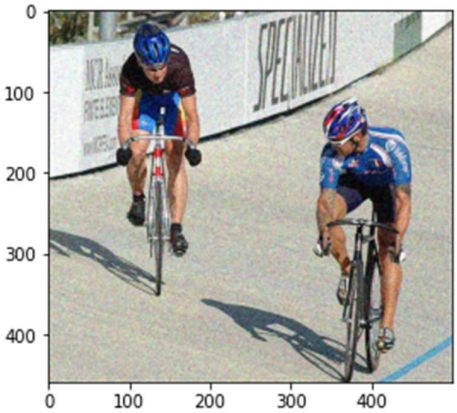




Rotated:



Noisy:



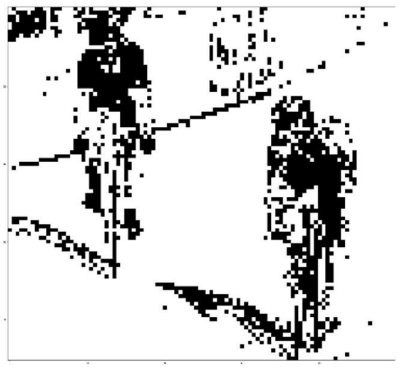
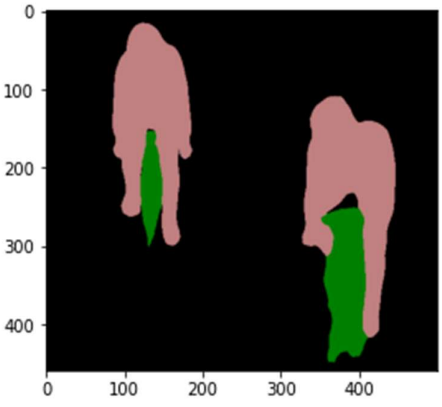



Image segmentation using FCN performs much better when classifying a object as a whole rather than in N-Cut it is difficult to get an object segmented as whole.

Also FCN gives much smoother segmentation is significantly better at distinguishing objects as we observed that in Test Image 5 the N-Cut algorithm tries to segment person along with cycle and other background objects as a single entity but in the case of FCN there a clearer distinguish between person cycle and background.

There is not much observable effect of rotation and gaussian noise addition to both algorithms.

But still gaussian noise have comparatively bad effect on the segmentation in case of N-Cut rather than in case of FCN.

DIFFICULTY FACED:

Most challenging part of Q2 was to train the model because of difference of opinion for the size and channels of the input and annotation to be given to loss function. It was a highly time taking task.