Import Libraries

In [29]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
import pyclustering
import pandas as pd
import random
```

Dataset Paths

In [30]:
```
folder_path = "C:\\Users\\Rajat\\Desktop\\SEM_3\\CV\\Project\\Semantic dataset50"
test_path = "C:\\Users\\Rajat\\Desktop\\SEM_3\\CV\\Project\\Semantic dataset50\\image"
ground_truth_path = "C:\\Users\\Rajat\\Desktop\\SEM_3\\CV\\Project\\Semantic dataset50\\ground-tru
th"
```

Function to load images from folder sorted by name

In [31]:
```
def load_images_from_folder(folder):
    images = []
    ls = os.listdir(folder)
    ls.sort()
    ID = []
    #print(ls)
    for filename in ls:
        img = cv2.imread(os.path.join(folder,filename))
        if img is not None:
            images.append(img)
            ID.append(filename)
    return images, ID
```

Function to convert grayscale image to binary

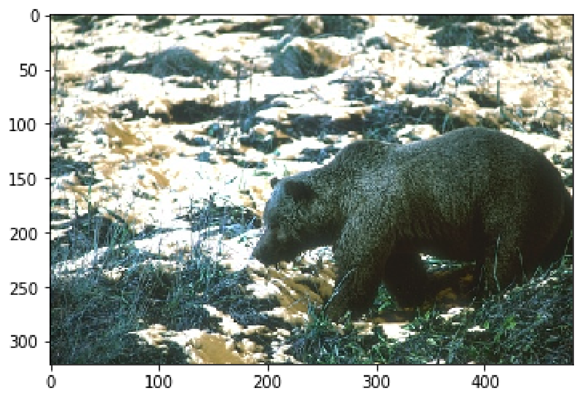
In [32]:
```
# Binary Conversion so to get precison/recall # 256/2 = 128 # 0 For <=127 , 1 else
# 0-> Black 255-> White
def convert_gray_2_binary_data(img):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if(img[i,j] <= 127):
                img[i,j] = 0
            else:
                img[i,j] = 1
    plt.imshow(img, cmap = 'gray')
    return img
def convert_gray_2_binary_truth(img):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if(img[i,j] <= 127):
                img[i,j] = 0
            else:
                img[i,j] = 1
    plt.imshow(img, cmap = 'gray')
    return img
```

In [33]:
```
images, ID_data = load_images_from_folder(test_path)
images_ground_truth, ID_truth = load_images_from_folder(ground_truth_path)
```

Original Image

```
In [34]: plt.imshow( images[0])
```

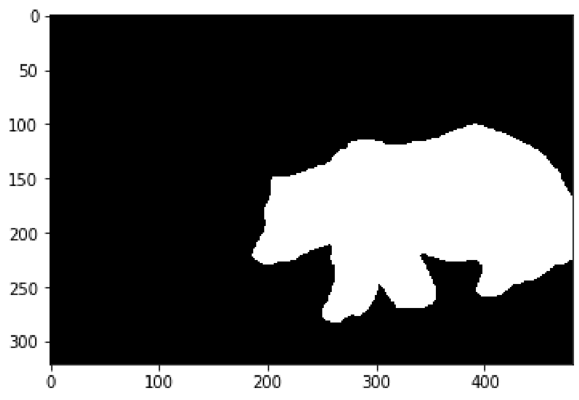
```
Out[34]: <matplotlib.image.AxesImage at 0x77230e62b0>
```



Ground Truth Image

```
In [35]: plt.imshow(images_ground_truth[0])
```

```
Out[35]: <matplotlib.image.AxesImage at 0x771a287b38>
```

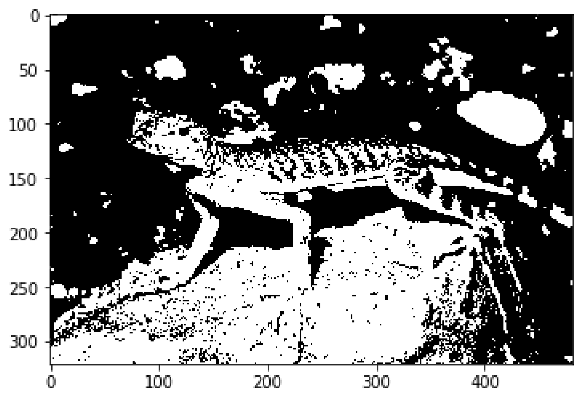


KMEANS from scratch for image segmentation

In [36]:
```
def distance(a,b):
    return np.abs(a-b)
def closest(lst, K):
    return lst[min(range(len(lst)), key = lambda i: abs(lst[i]-K))]

def mean_calc(img, labels, means):
    mean_1 = 0
    mean_2 = 0
    for i in range(len(img)):
        if labels[i] == means[0]:
            mean_1 = mean_1 + img[i]
        else:
            mean_1 = mean_1 + img[i]
    mean_1 = mean_1/len(img)
    mean_2 = mean_2/len(img)

    return mean_1, mean_2


results = []
for p in range(50):
    print('Working on image :', p)
    img = images[p]
    img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    img = img.reshape((-1,))
    print(img.shape)
    K = 2
    means = []
    m1 = random.choice(img)
    m2 = random.choice(img)
    medoids =   [m1,  m2]
    #plt.imshow(img, cmap = 'gray')
    iters = 10
    for i in range(iters):
        labels = np.zeros(len(img))

        # Label assignment
        for j in range(len(img)):
                if distance(img[j], means[0]) < distance(img[j], means[1]):
                    labels[j] = means[0]
                else:
                    labels[j] = means[1]

         # Mean calculation

        mean_1, mean_2 = mean_calc(img, labels, means)
        means[0] = mean_1
        means[1] = mean_2

    print(means)
    for i in range(len(img)):
        if labels[i] == means[0]:
            img[i] = 0
        else:
            img[i] = 1
    img = img.reshape(images[p].shape[:2])
    plt.figure()
    plt.imshow(img, cmap = 'gray')
    results.append(img)
```

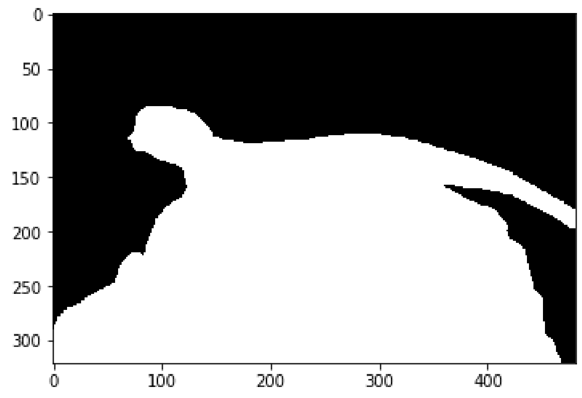
```
Processing: 100098.jpg Number: 0
Processing: 101027.jpg Number: 1
Processing: 103006.jpg Number: 2
Processing: 103029.jpg Number: 3
Processing: 104010.jpg Number: 4
Processing: 105027.jpg Number: 5
Processing: 106005.jpg Number: 6
Processing: 106024.jpg Number: 7
Processing: 106025.jpg Number: 8
Processing: 106047.jpg Number: 9
Processing: 107072.jpg Number: 10
Processing: 12003.jpg Number: 11
Processing: 15062.jpg Number: 12
Processing: 23084.jpg Number: 13
Processing: 26031.jpg Number: 14
Processing: 29030.jpg Number: 15
Processing: 3063.jpg Number: 16
Processing: 3096.jpg Number: 17
Processing: 35010.jpg Number: 18
Processing: 35058.jpg Number: 19
Processing: 35070.jpg Number: 20
Processing: 41006.jpg Number: 21
Processing: 41025.jpg Number: 22
Processing: 41029.jpg Number: 23
Processing: 41033.jpg Number: 24
Processing: 41069.jpg Number: 25
Processing: 41085.jpg Number: 26
Processing: 41096.jpg Number: 27
Processing: 42044.jpg Number: 28
Processing: 42049.jpg Number: 29
Processing: 42078.jpg Number: 30
Processing: 43033.jpg Number: 31
Processing: 43051.jpg Number: 32
Processing: 43074.jpg Number: 33
Processing: 46076.jpg Number: 34
Processing: 48017.jpg Number: 35
Processing: 51084.jpg Number: 36
Processing: 6046.jpg Number: 37
Processing: 61060.jpg Number: 38
Processing: 64061.jpg Number: 39
Processing: 69020.jpg Number: 40
Processing: 69022.jpg Number: 41
Processing: 69040.jpg Number: 42
Processing: 70011.jpg Number: 43
Processing: 80090.jpg Number: 44
Processing: 80099.jpg Number: 45
Processing: 8068.jpg Number: 46
Processing: 81095.jpg Number: 47
Processing: 87015.jpg Number: 48
Processing: 87046.jpg Number: 49
```

```
Out[36]: 50
```



In [37]:
```
ground_truths = []
for img in images_ground_truth:
    img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    img = convert_gray_2_binary_truth(img)
    ground_truths.append(img)
len(ground_truths)
```

Out[37]:
```
50
```



Function to calculate Precision, Recall, F1 Score, Accuracy and IoU Scores

In [45]:
```
# Precision, Recall, F1 Score, IUC
# TP = 11 (ground_truth, result_data)
# TN = 00 (actual, predicted)
# FP = 01
# FN = 10
def calc_precision_recall(results, ground_truths):
    precisions = []
    recalls = []
    F1_scores = []
    IOUs = []
    accuracies =[]

    for k in range(len(results)):
        print('for image',k, '\n')
        TP = 0
        FP = 0
        FN = 0
        TN = 0
        for i in range(results[k].shape[0]):
            for j in range(results[k].shape[1]):
                if results[k][i,j] == 0 and ground_truths[k][i,j] == 0:
                    TN = TN + 1
                elif results[k][i,j] == 0 and ground_truths[k][i,j] == 1:
                    FP = FP + 1
                elif results[k][i,j] == 1 and ground_truths[k][i,j] == 0:
                    FN = FN + 1
                else :
                    TP = TP + 1
        precision = TP / (TP + FP)
        recall = TP / (TP + FN)
        iou = TP /(TP + FN + FP )
        accuracy = (TP + TN) / (TP + TN + FP + FN)
        if (precision + recall) != 0:
            f1_score = (2 * precision * recall)/ (precision + recall)
        else:
            f1_score = 0
        precisions.append(precision)
        recalls.append(recall)
        accuracies.append(accuracy*100)
        IOUs.append(iou)
        F1_scores.append(f1_score)
    return precisions, recalls, accuracies, IOUs, F1_scores
```

In [46]:
```
precisions, recalls, accuracies, IOUs, F1_scores = calc_precision_recall(results, ground_truths)
```

```
for image 0
for image 1
for image 2
for image 3
for image 4
for image 5
for image 6
for image 7
for image 8
for image 9
for image 10
for image 11
for image 12
for image 13
for image 14
for image 15
for image 16
for image 17
for image 18
for image 19
for image 20
for image 21
for image 22
for image 23
for image 24
for image 25
for image 26
for image 27
for image 28
for image 29
for image 30
for image 31
for image 32
for image 33
for image 34
for image 35
for image 36
for image 37
for image 38

for image 39

for image 40

for image 41

for image 42

for image 43

for image 44

for image 45

for image 46

for image 47

for image 48

for image 49
```

Saving results in a pandas dataframe

In [48]:
```python
df = pd.DataFrame(ID_data, columns = ['Image_ID'])
df.insert(1, "Precision", precisions, True)
df.insert(2, "Recall", recalls, True)
df.insert(3, "F1_score", F1_scores, True)
df.insert(4, "Accuracy", accuracies, True)
df.insert(5, "IOU", IOUs, True)
df.head()
```

Out[48]:

	Image_ID	Precision	Recall	F1_score	Accuracy	IOU
0	100098.jpg	0.984780	0.344416	0.510345	57.863615	0.342593
1	101027.jpg	0.598070	0.973649	0.740985	87.201508	0.588544
2	103006.jpg	0.574153	0.282198	0.378407	48.631162	0.233355
3	103029.jpg	0.236820	0.558087	0.332532	84.711887	0.199423
4	104010.jpg	0.439088	0.375433	0.404773	65.336364	0.253740

In [49]:
```python
df.to_csv('Kmeans.csv', index=False)
```

Function to invert the image

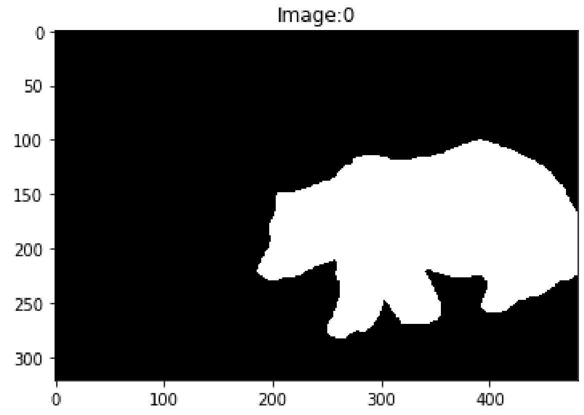
def convert(img): for i in range(img.shape[0]): for j in range(img.shape[1]): if img[i,j] == 0: img[i,j] = 1 else: img[i,j] = 0 return img

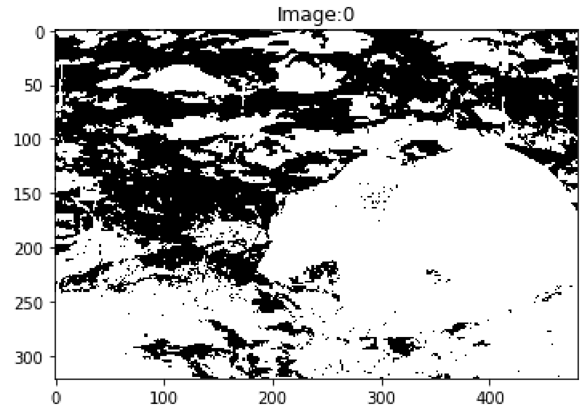
In [50]:
```python
ls = [5]
for i in ls:
    results[i] = convert(results[i])
```

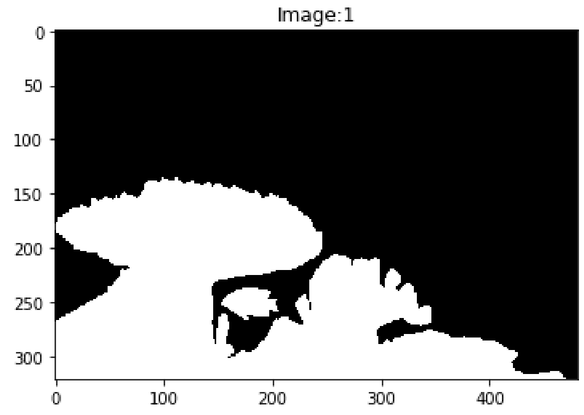
Ground Truths and correcponding results as shown below

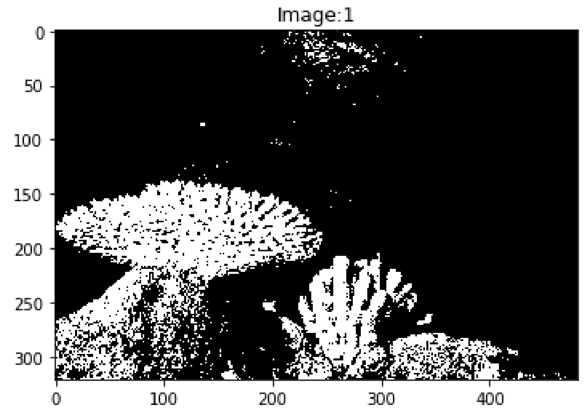
```
In [51]: for i in range(50):
             plt.figure()
             plt.title('Image:' + str(i))
             plt.imshow(ground_truths[i], cmap = 'gray')
             plt.figure()
             plt.title('Image:' + str(i))
             plt.imshow(results[i], cmap = 'gray')
```

```
C:\Users\Rajat\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning: More than 20 f
igures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`)
are retained until explicitly closed and may consume too much memory. (To control this warning, se
e the rcParam `figure.max_open_warning`).

C:\Users\Rajat\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: RuntimeWarning: More than 20 f
igures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`)
are retained until explicitly closed and may consume too much memory. (To control this warning, se
e the rcParam `figure.max_open_warning`).
  """
```









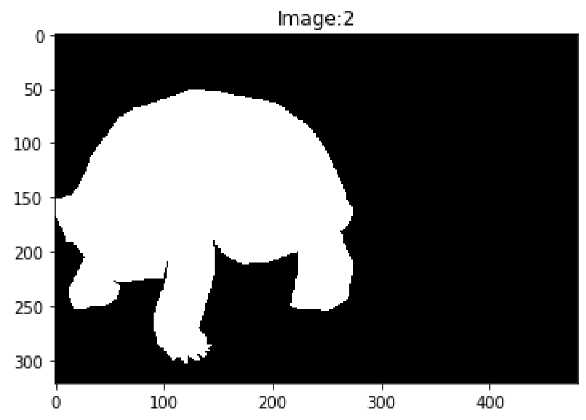


Image:2
0
50
100
150
200
250
300
0
100
200
300
400

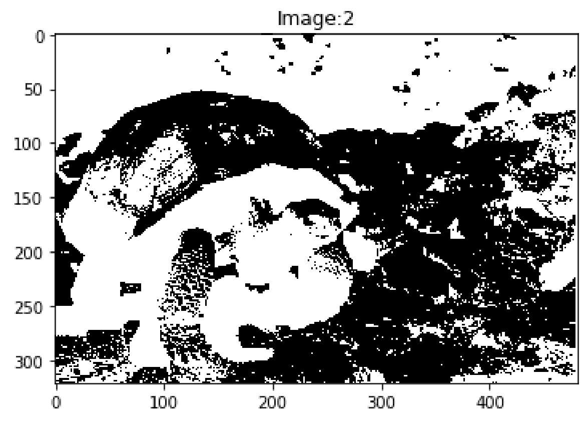


Image:2
0
50
100
150
200
250
300
0
100
200
300
400

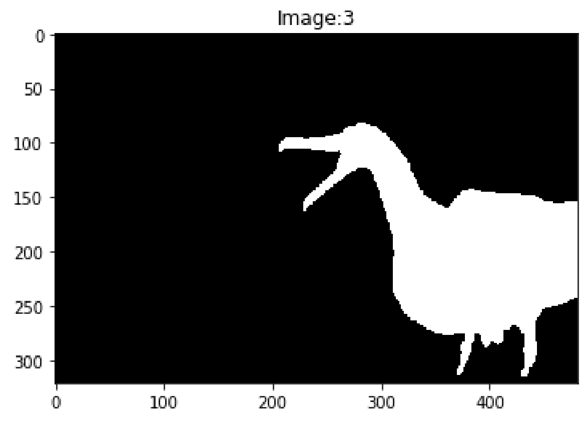


Image:3
0
50
100
150
200
250
300
0
100
200
300
400

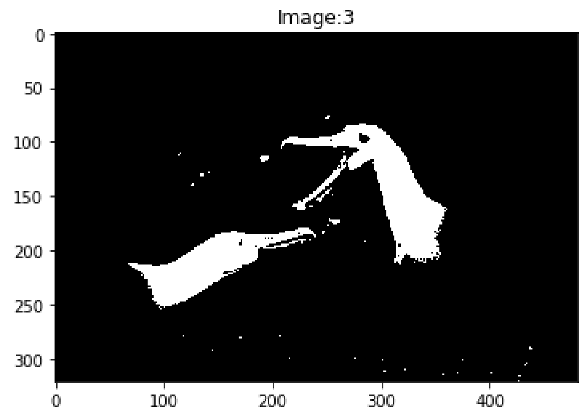


Image:3
0
50
100
150
200
250
300
0
100
200
300
400

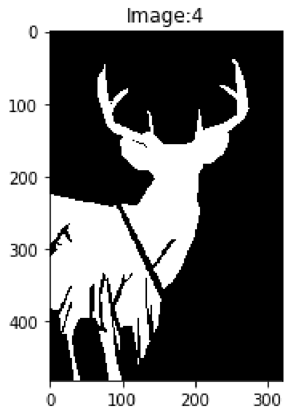


Image:4
0
100
200
300
400
0
100
200
300

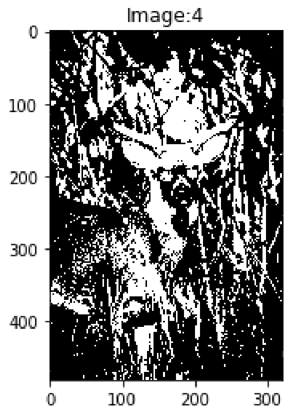


Image:4
0
100
200
300
400
0
100
200
300

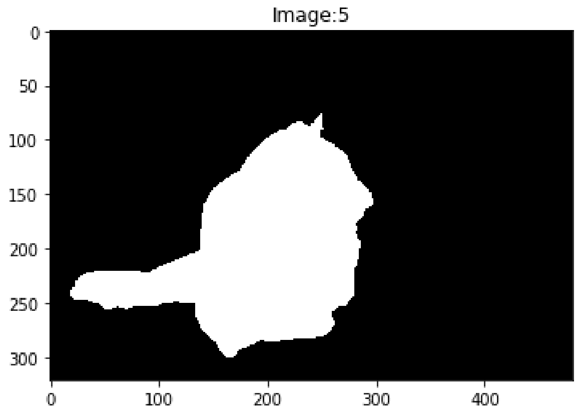


Image:5
0
50
100
150
200
250
300
0
100
200
300
400

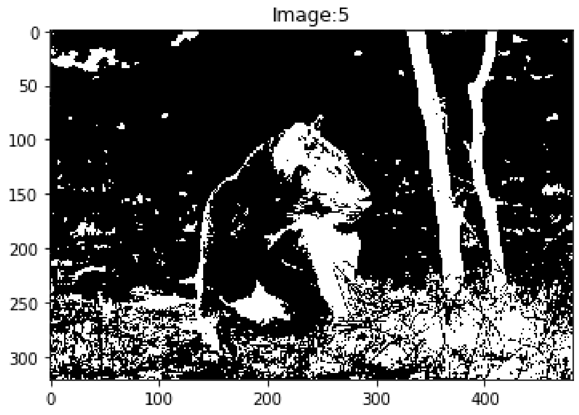


Image:5
0
50
100
150
200
250
300
0
100
200
300
400

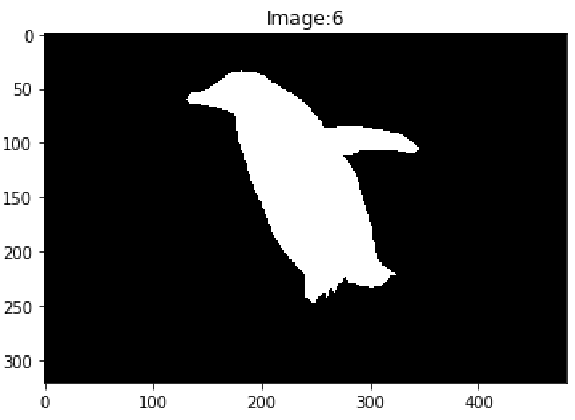


Image:6
0
50
100
150
200
250
300
0
100
200
300
400

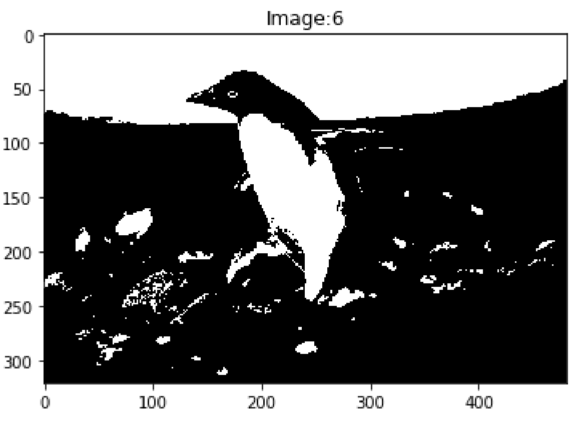


Image:6
0
50
100
150
200
250
300
0
100
200
300
400

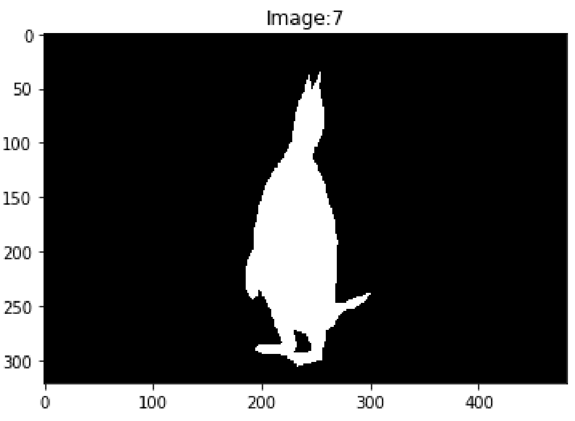


Image:7
0
50
100
150
200
250
300
0
100
200
300
400

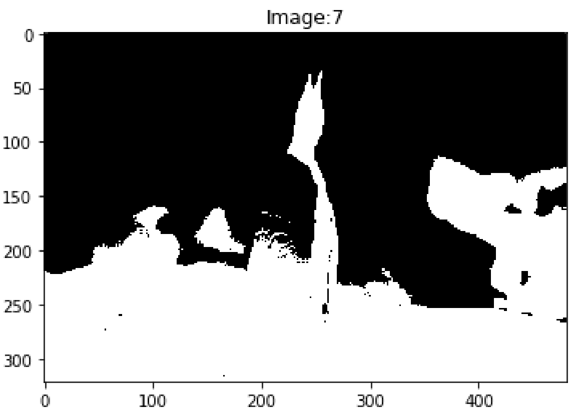


Image:7
0
50
100
150
200
250
300
0
100
200
300
400

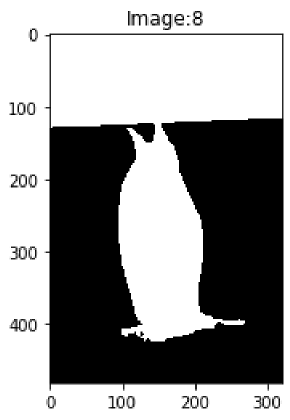


Image:8

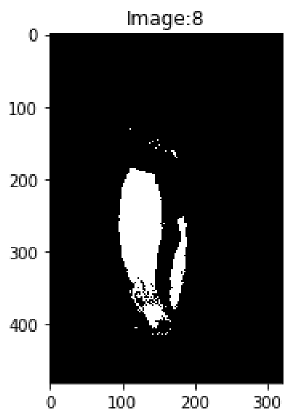


Image:8

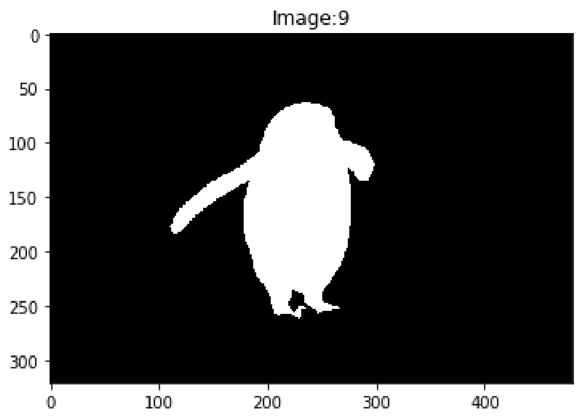


Image:9

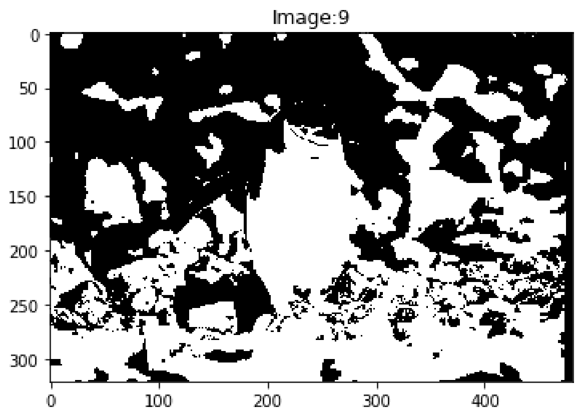


Image:9

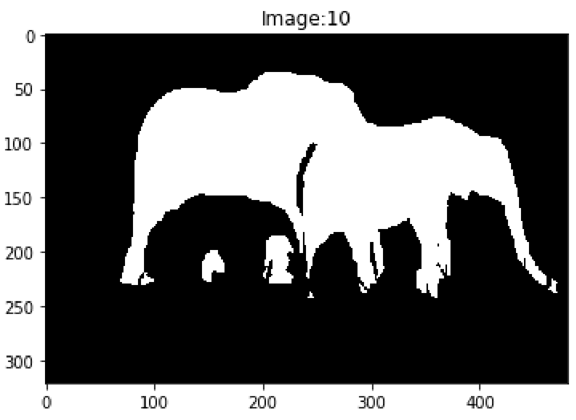


Image:10
0
50
100
150
200
250
300
0
100
200
300
400

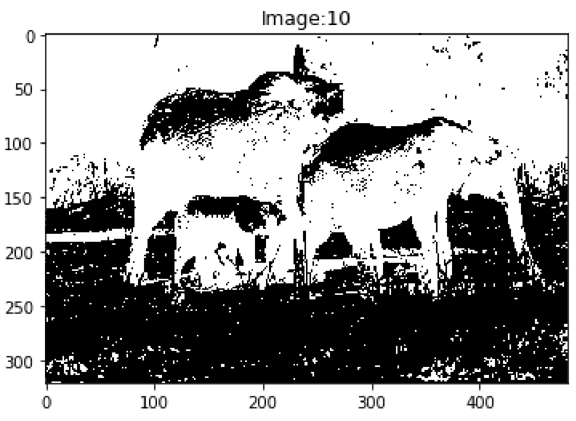


Image:10
0
50
100
150
200
250
300
0
100
200
300
400

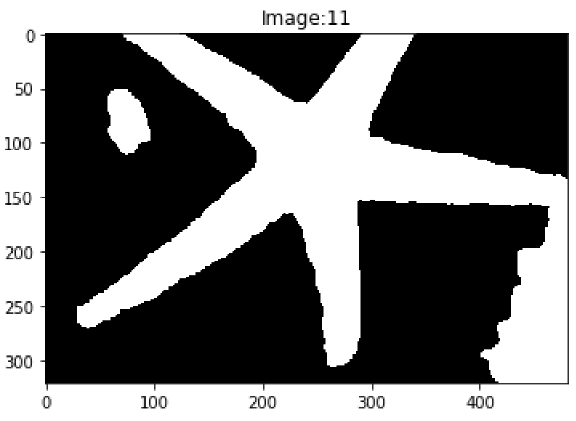


Image:11
0
50
100
150
200
250
300
0
100
200
300
400

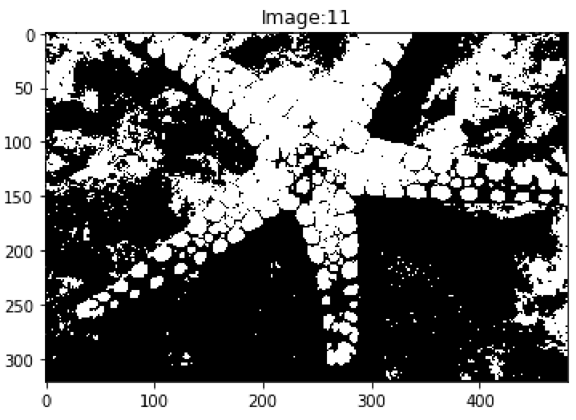


Image:11
0
50
100
150
200
250
300
0
100
200
300
400

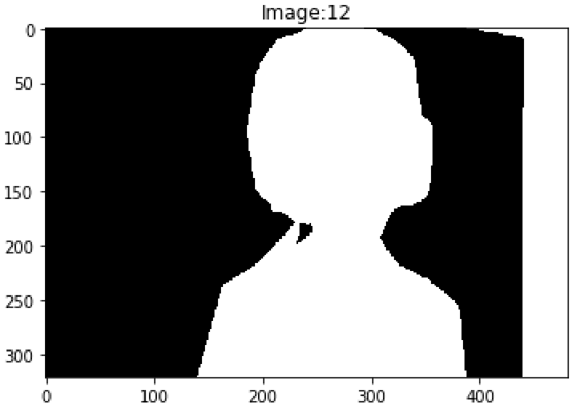


Image:12

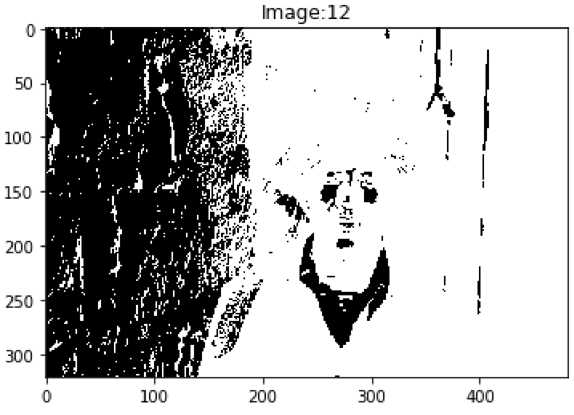


Image:12

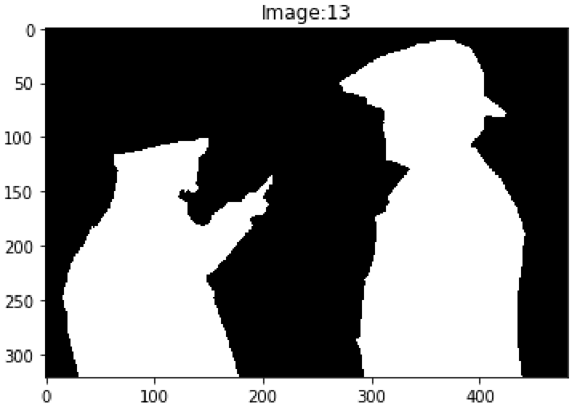


Image:13

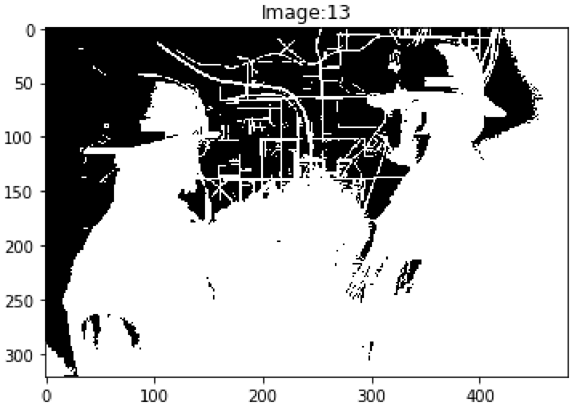


Image:13

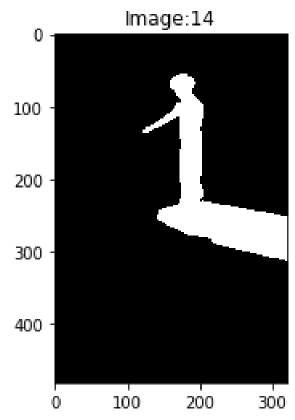


Image:14
0
100
200
300
400
0
100
200
300

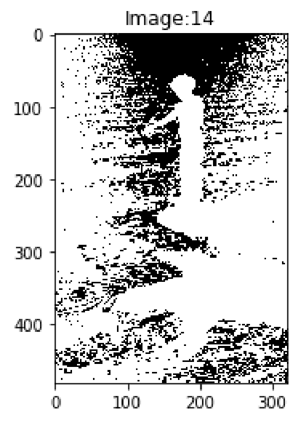


Image:14
0
100
200
300
400
0
100
200
300

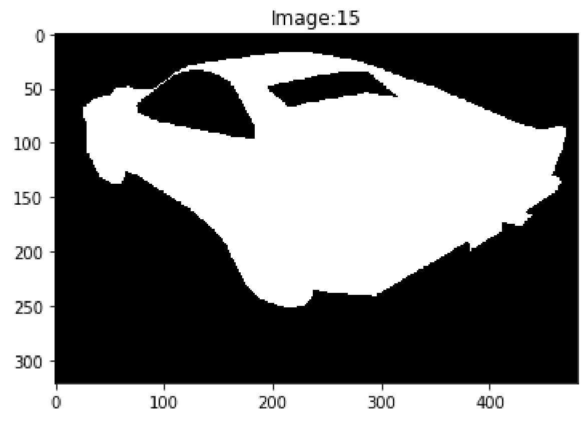


Image:15
0
50
100
150
200
250
300
0
100
200
300
400

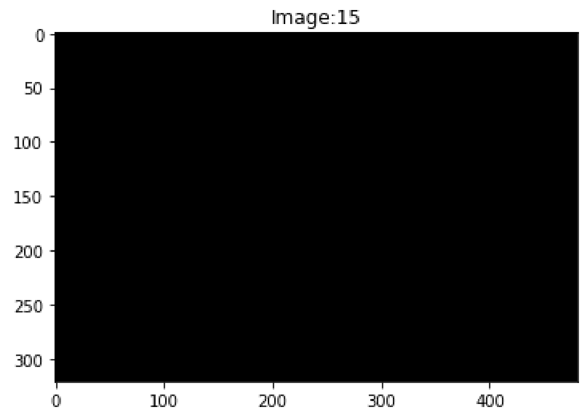


Image:15
0
50
100
150
200
250
300
0
100
200
300
400

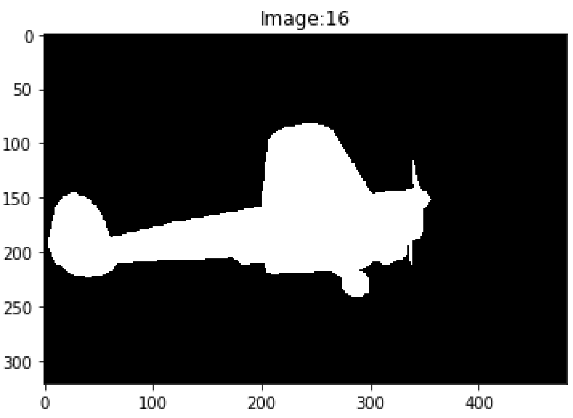


Image:16
0
50
100
150
200
250
300
0
100
200
300
400

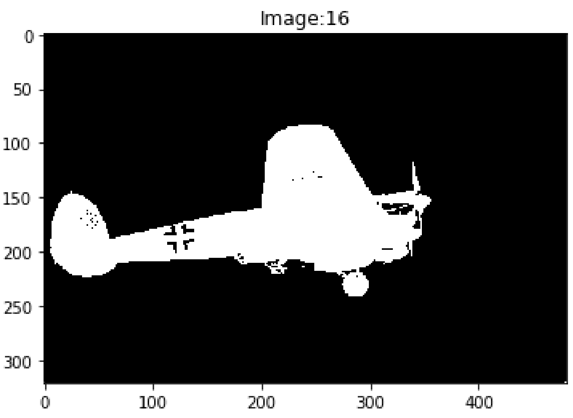


Image:16
0
50
100
150
200
250
300
0
100
200
300
400

Image:17
0
50
100
150
200
250
300
0
100
200
300
400

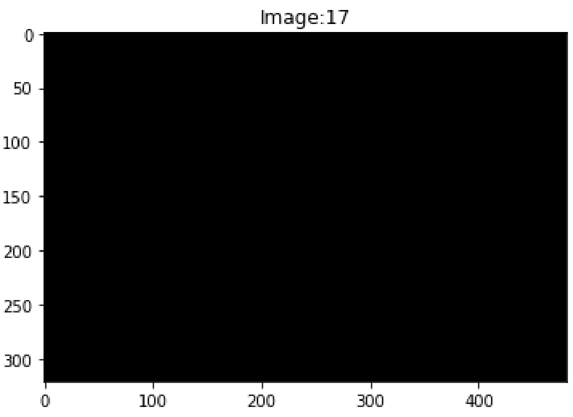


Image:17
0
50
100
150
200
250
300
0
100
200
300
400

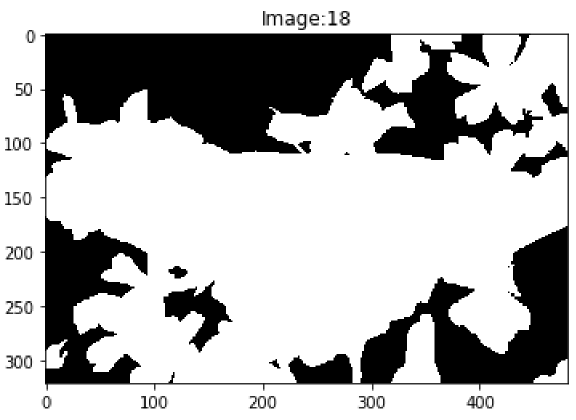


Image:18
0
50
100
150
200
250
300
0
100
200
300
400

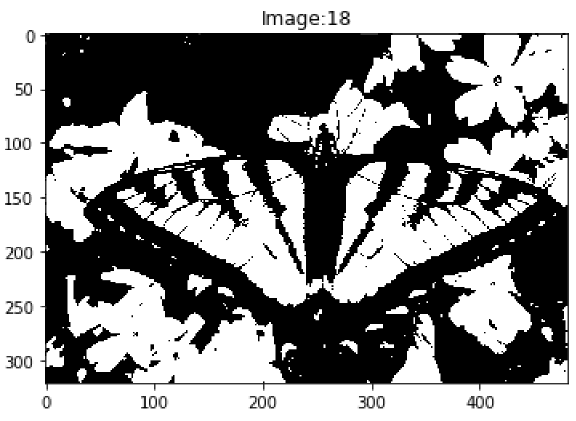


Image:18
0
50
100
150
200
250
300
0
100
200
300
400

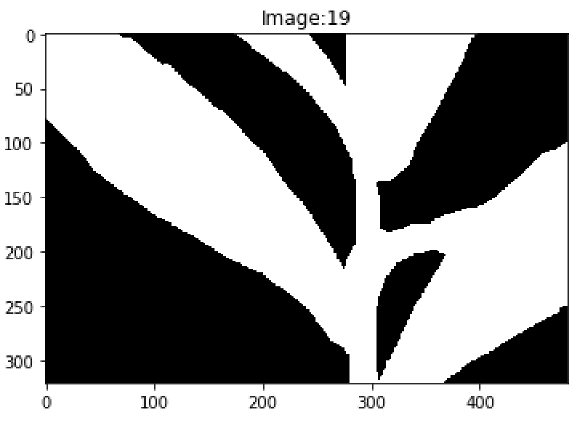


Image:19
0
50
100
150
200
250
300
0
100
200
300
400

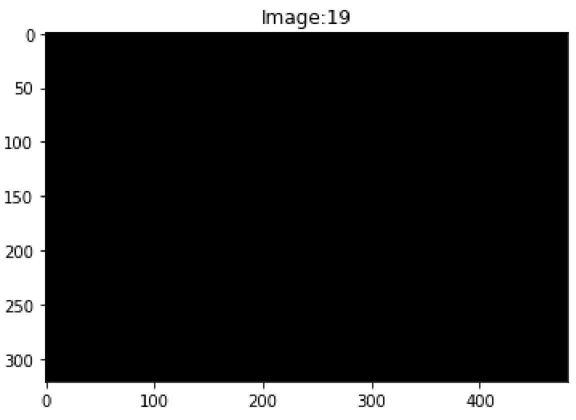


Image:19
0
50
100
150
200
250
300
0
100
200
300
400

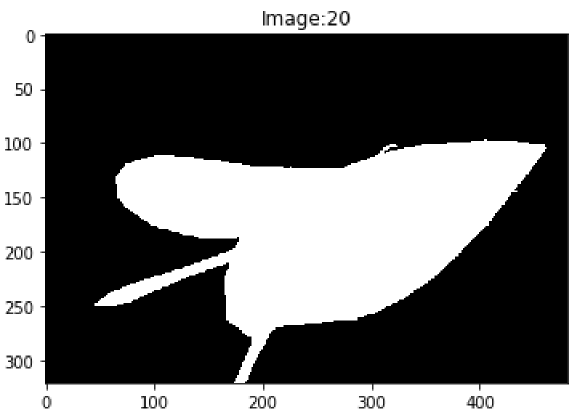


Image:20
0
50
100
150
200
250
300
0
100
200
300
400

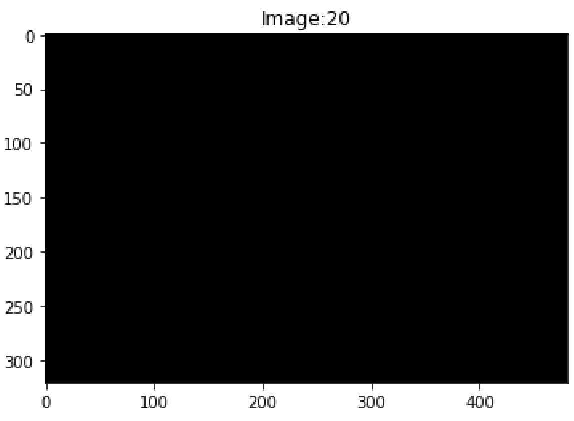


Image:20
0
50
100
150
200
250
300
0
100
200
300
400

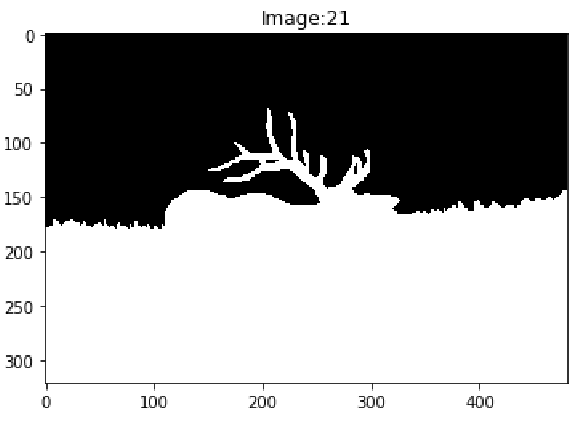


Image:21
0
50
100
150
200
250
300
0
100
200
300
400

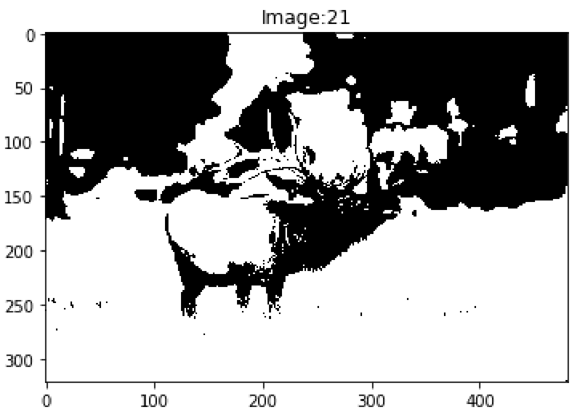


Image:21
0
50
100
150
200
250
300
0
100
200
300
400

Image:22

Image:22

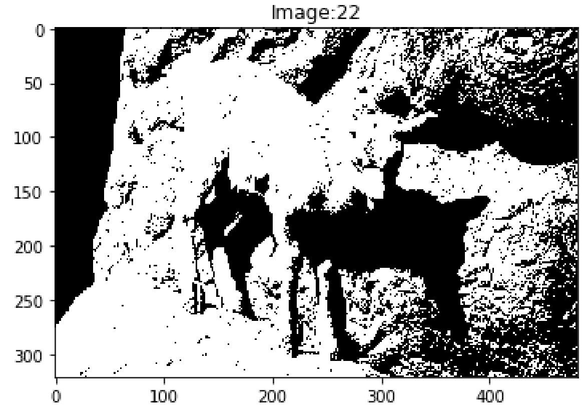


Image:23

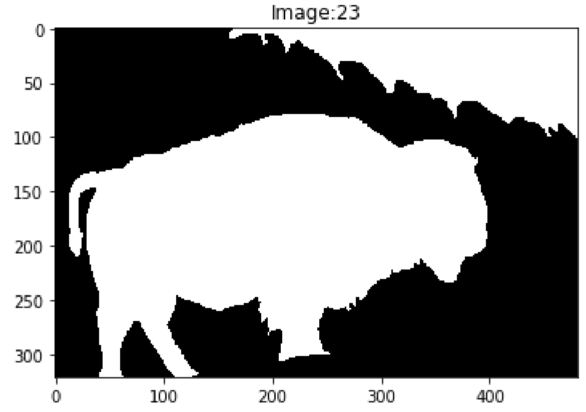
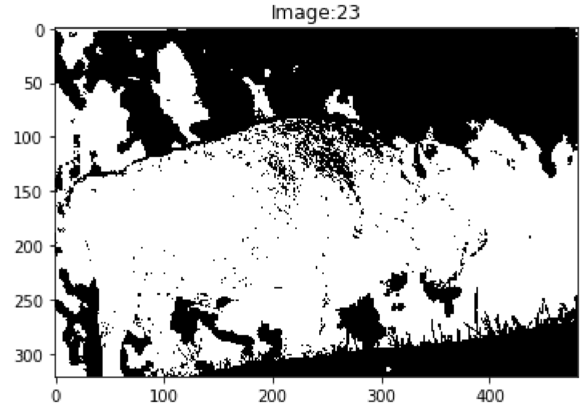


Image:23



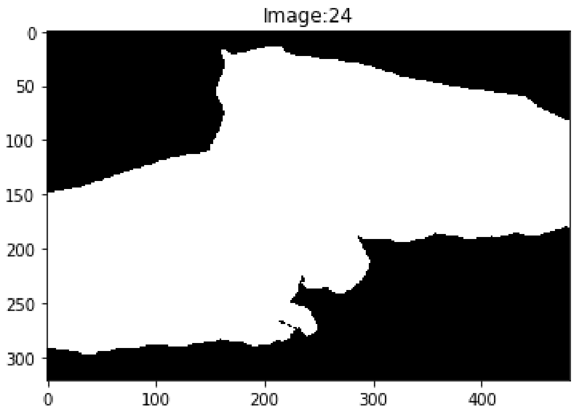


Image:24
0
50
100
150
200
250
300
0
100
200
300
400

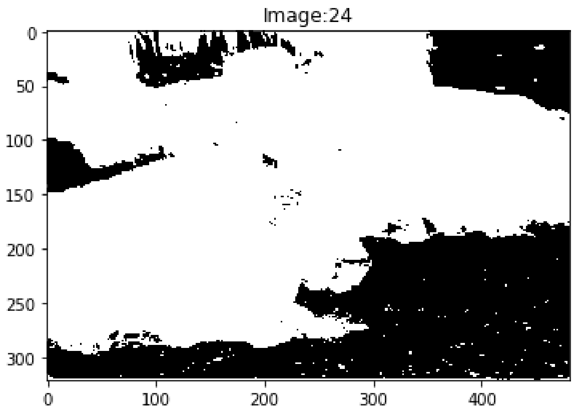


Image:24
0
50
100
150
200
250
300
0
100
200
300
400

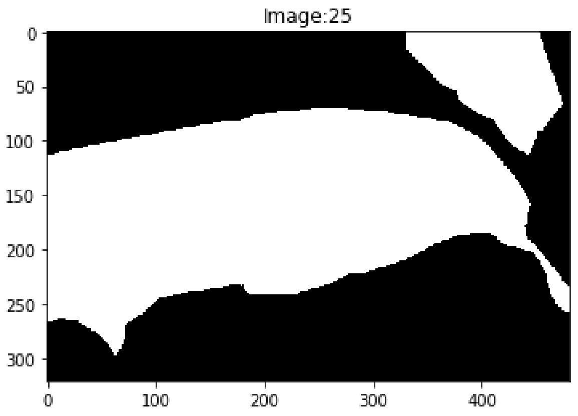


Image:25
0
50
100
150
200
250
300
0
100
200
300
400

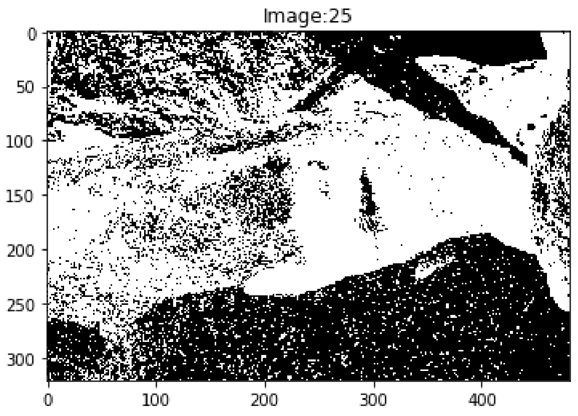


Image:25
0
50
100
150
200
250
300
0
100
200
300
400

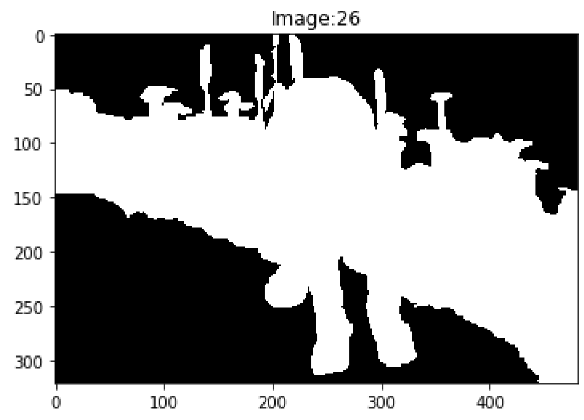


Image:26
0
50
100
150
200
250
300
0
100
200
300
400

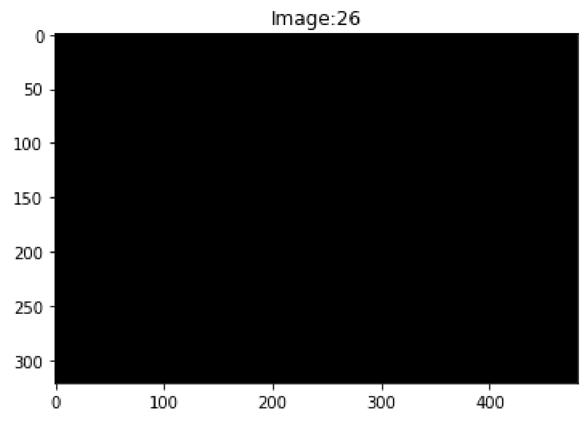


Image:26
0
50
100
150
200
250
300
0
100
200
300
400

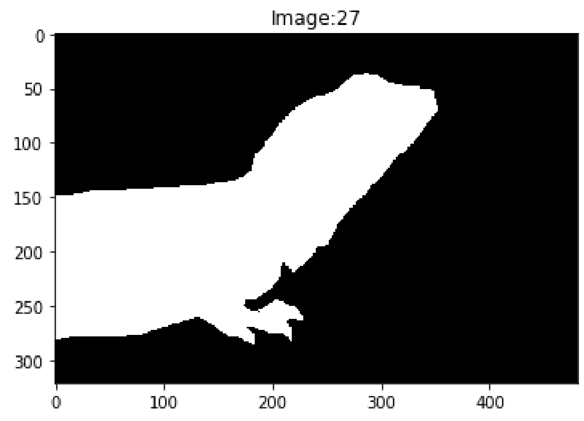


Image:27
0
50
100
150
200
250
300
0
100
200
300
400

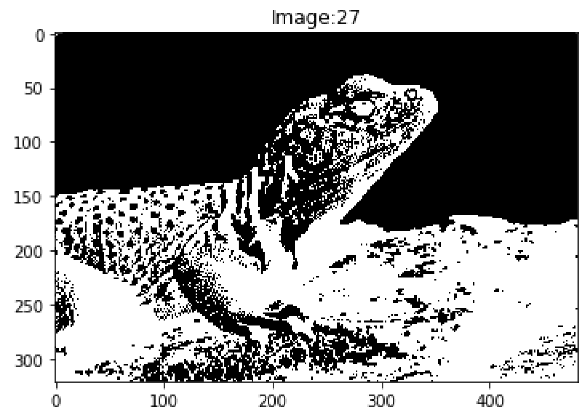


Image:27
0
50
100
150
200
250
300
0
100
200
300
400

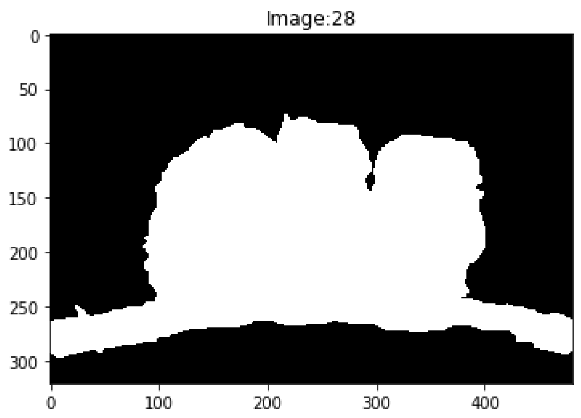


Image:28

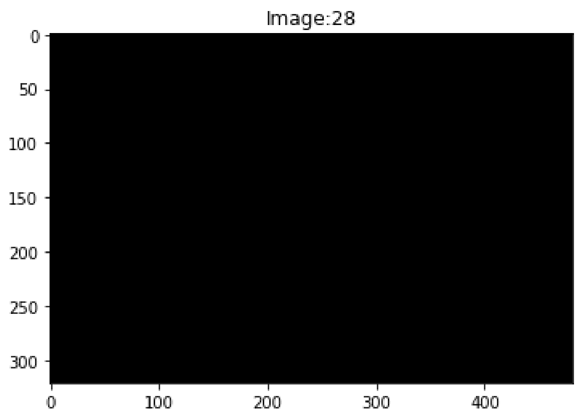


Image:28

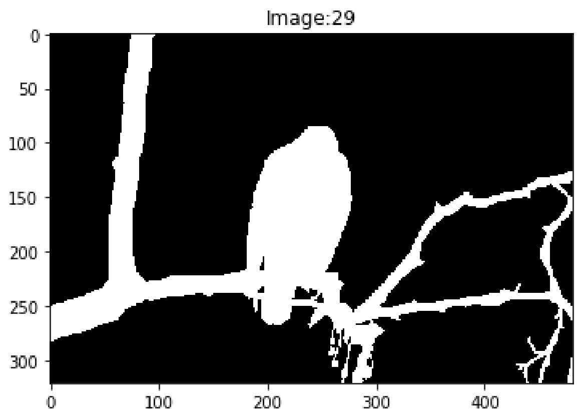


Image:29

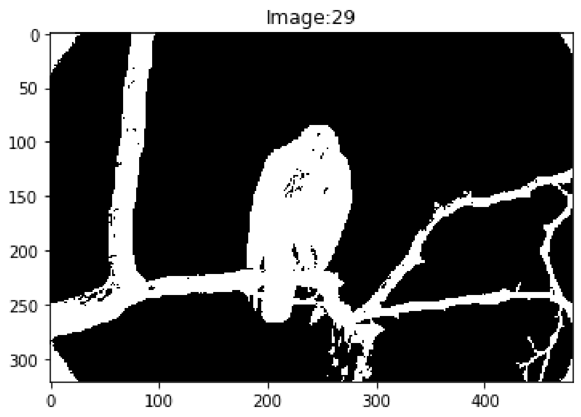


Image:29

Image:30

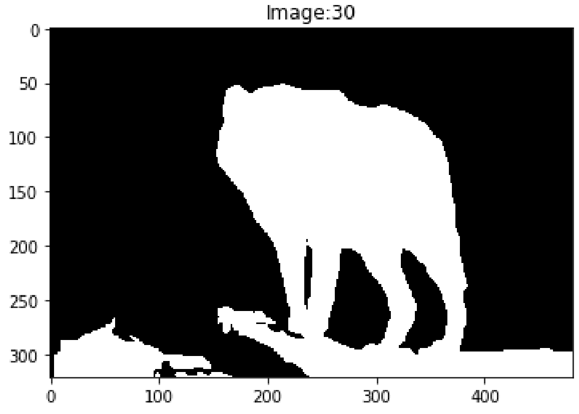


Image:30

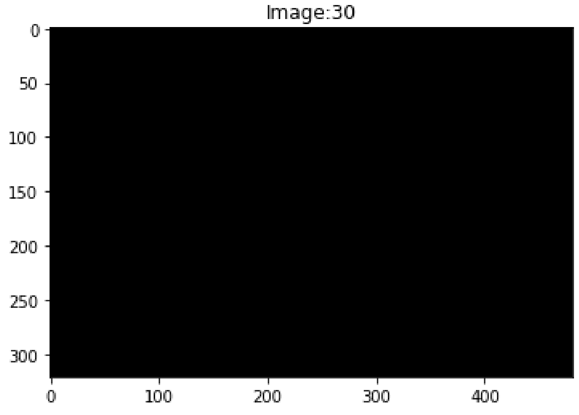


Image:31

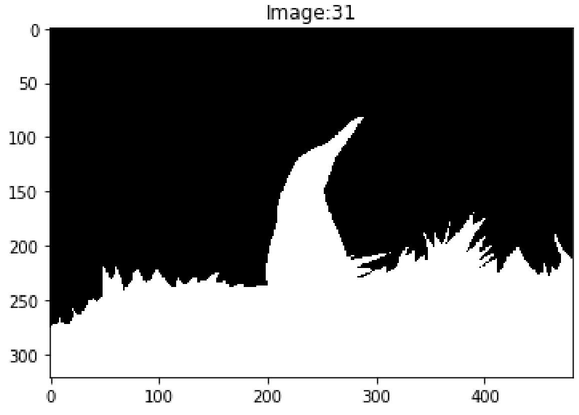
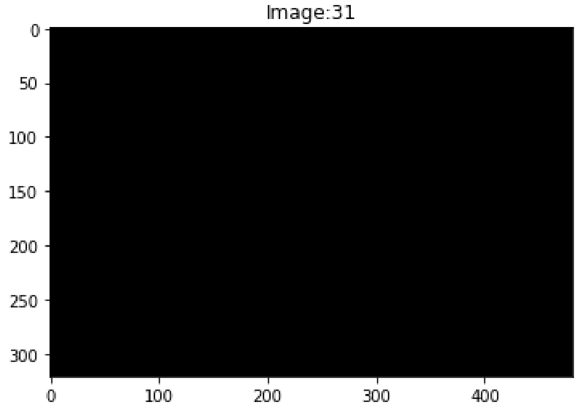


Image:31



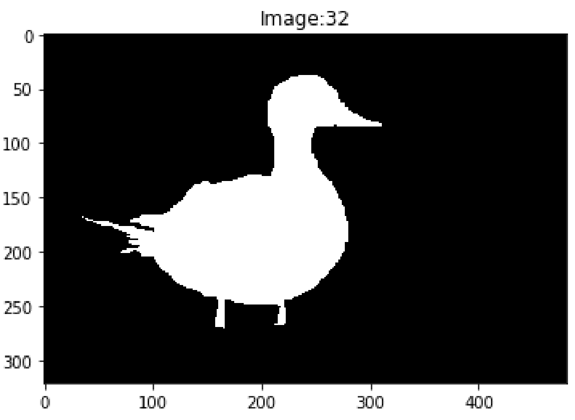


Image:32

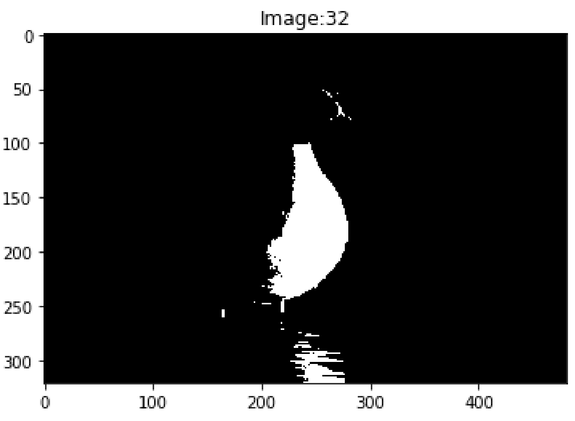


Image:32

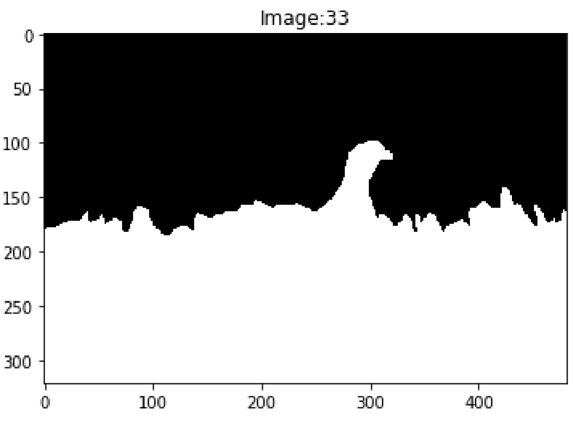


Image:33

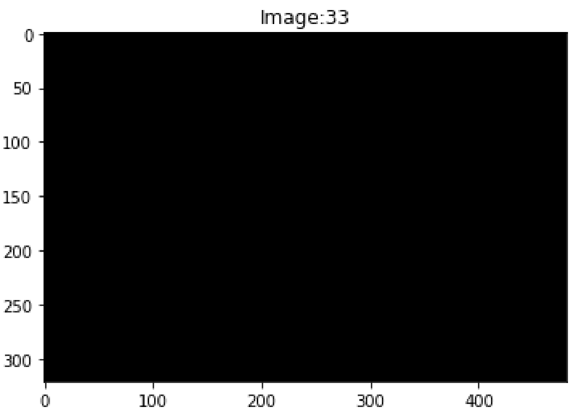


Image:33

Image:34

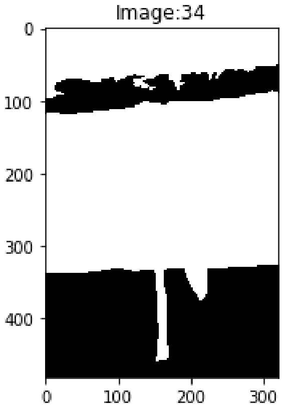


Image:34

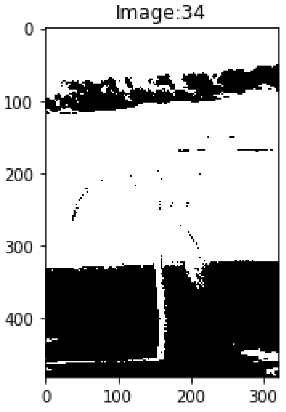


Image:35

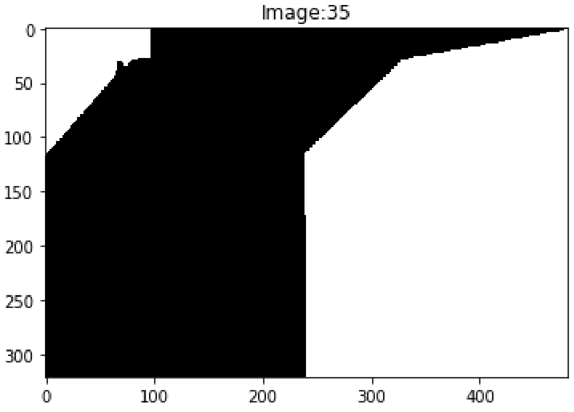
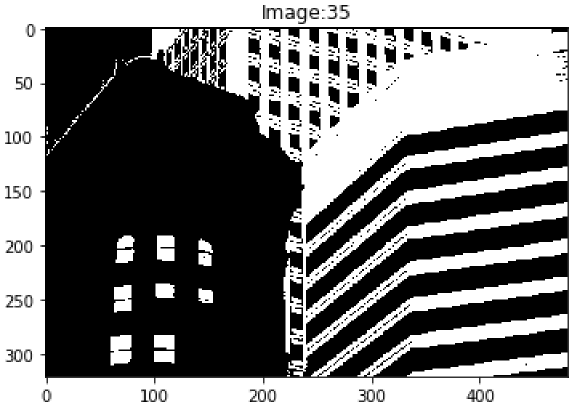


Image:35



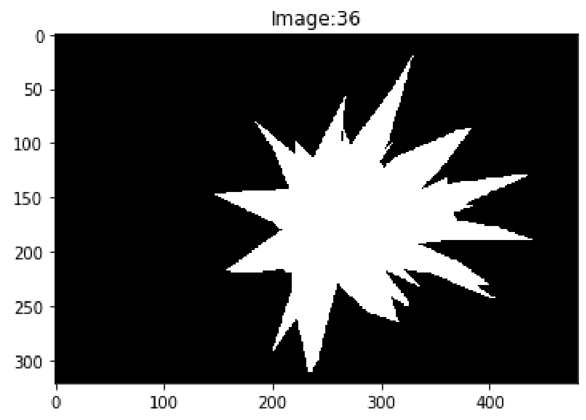


Image:36
0
50
100
150
200
250
300
0
100
200
300
400

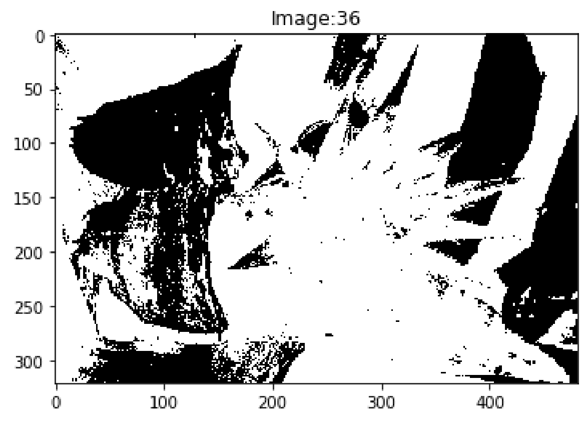


Image:36
0
50
100
150
200
250
300
0
100
200
300
400

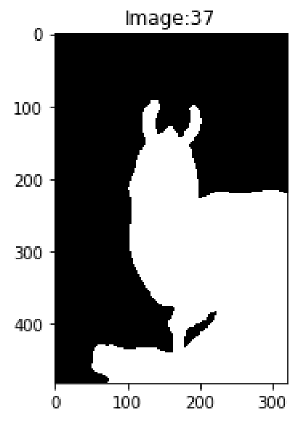


Image:37
0
100
200
300
400
0
100
200
300

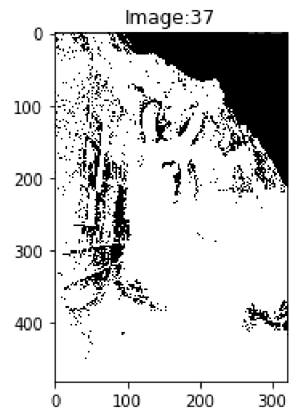


Image:37
0
100
200
300
400
0
100
200
300

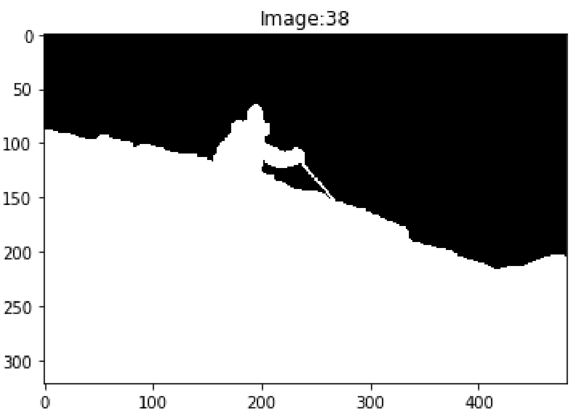


Image:38

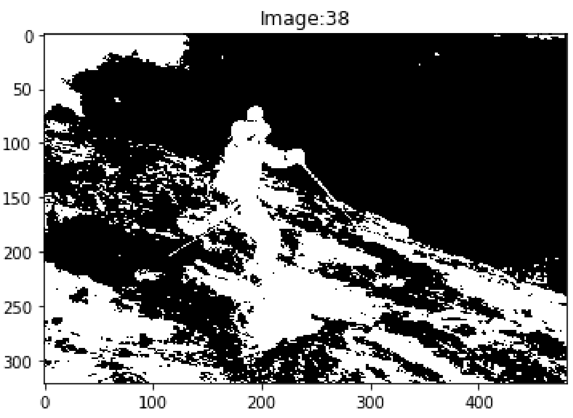


Image:38

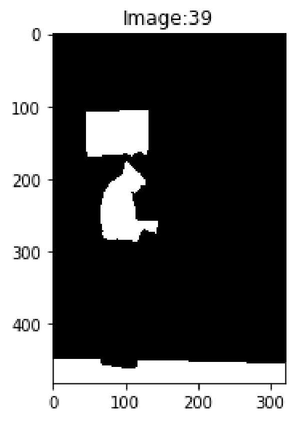


Image:39

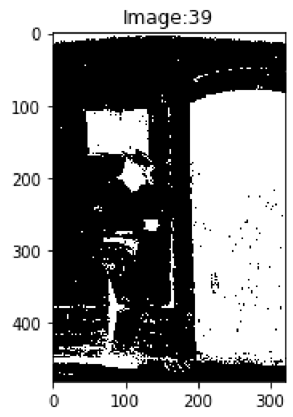


Image:39

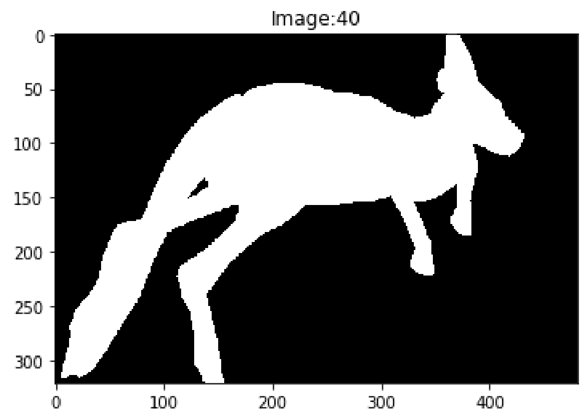


Image:40
0
50
100
150
200
250
300
0
100
200
300
400

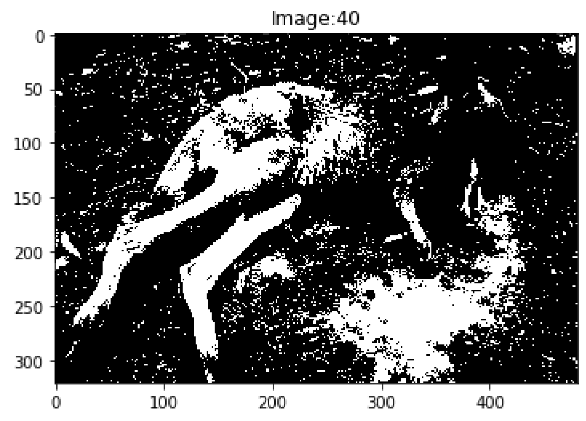


Image:40
0
50
100
150
200
250
300
0
100
200
300
400

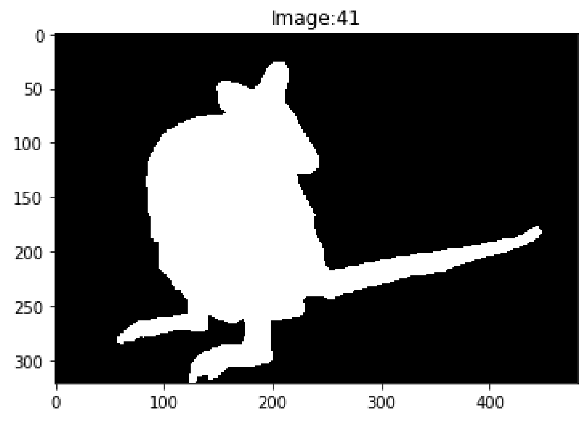


Image:41
0
50
100
150
200
250
300
0
100
200
300
400

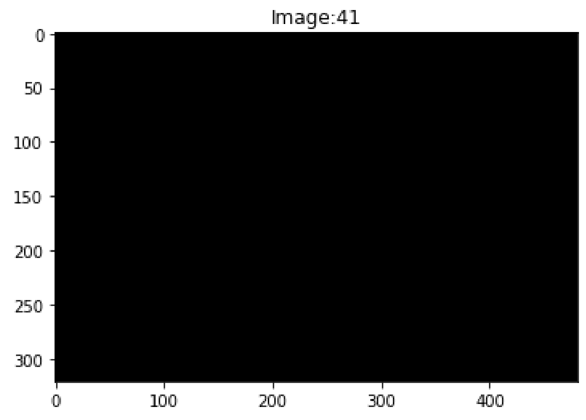


Image:41
0
50
100
150
200
250
300
0
100
200
300
400

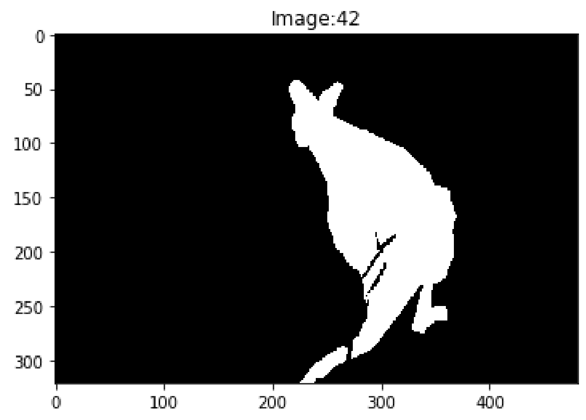


Image:42
0
50
100
150
200
250
300
0
100
200
300
400

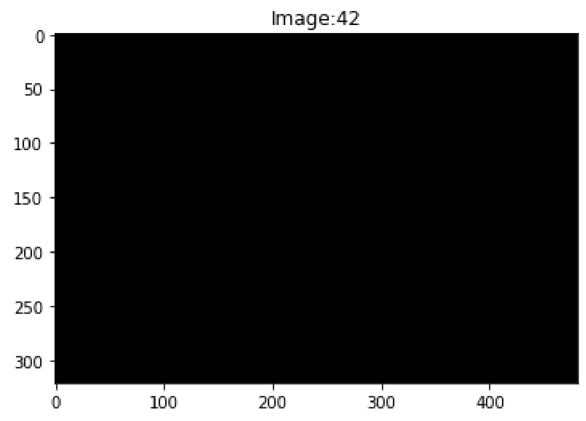


Image:42
0
50
100
150
200
250
300
0
100
200
300
400

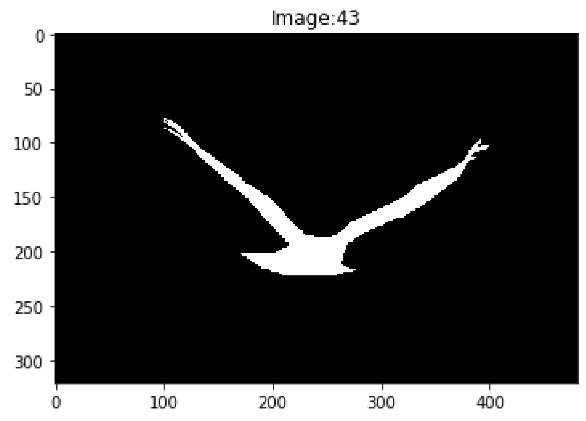


Image:43
0
50
100
150
200
250
300
0
100
200
300
400

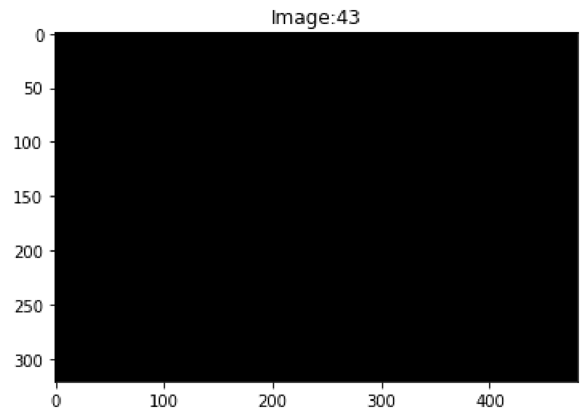


Image:43
0
50
100
150
200
250
300
0
100
200
300
400

Image:44

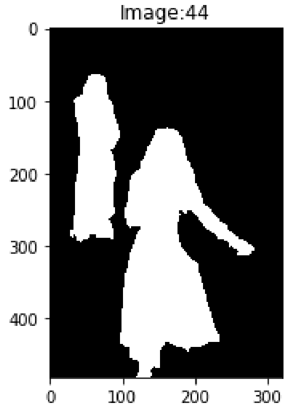


Image:44

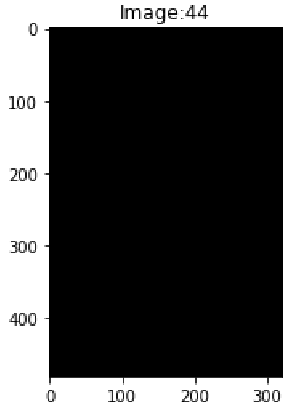


Image:45

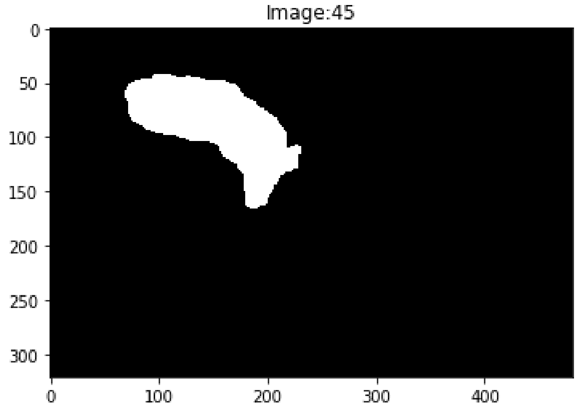


Image:45

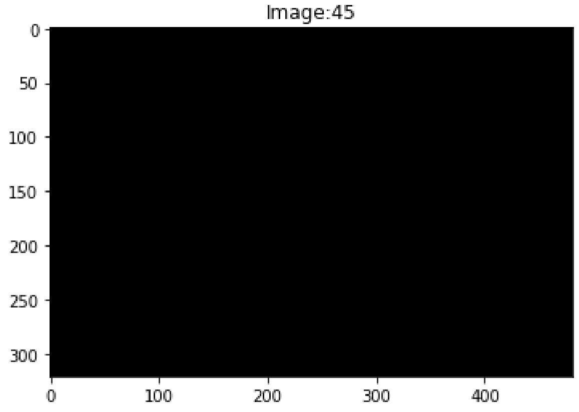


Image:46

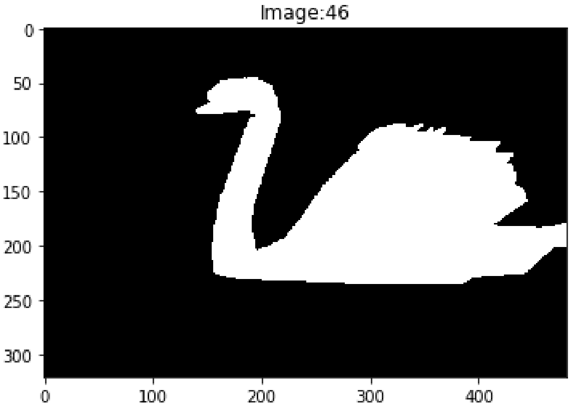


Image:46

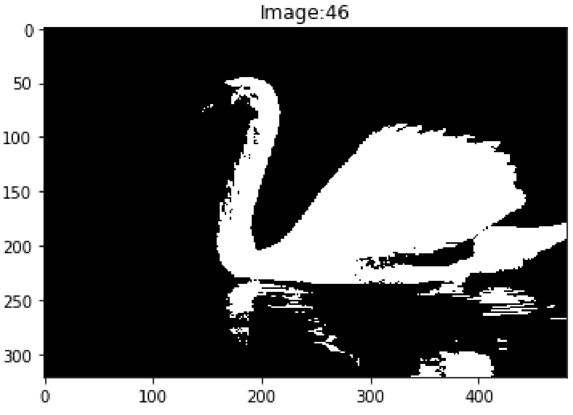


Image:47

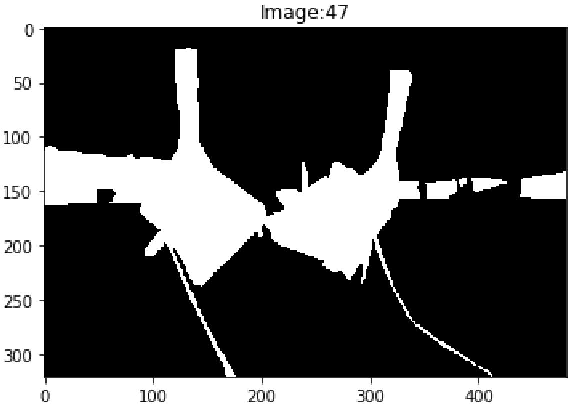
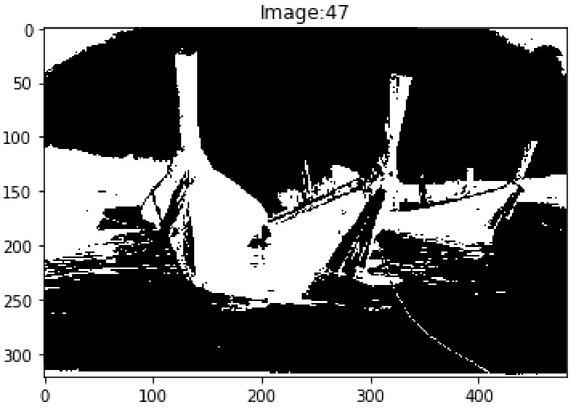


Image:47



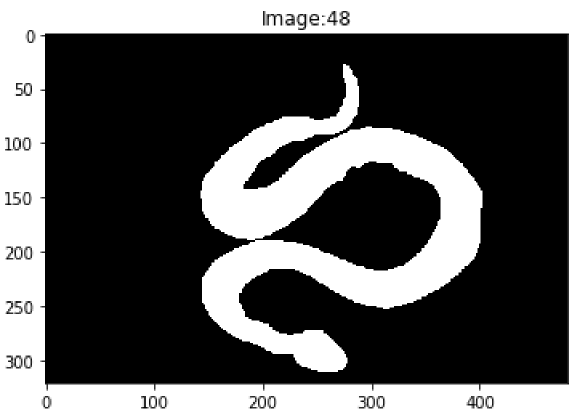


Image:48

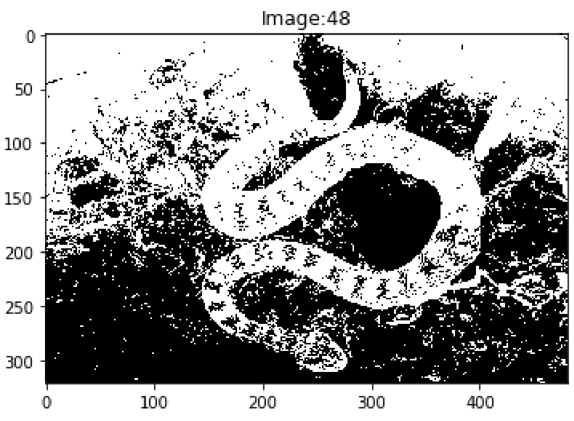


Image:48

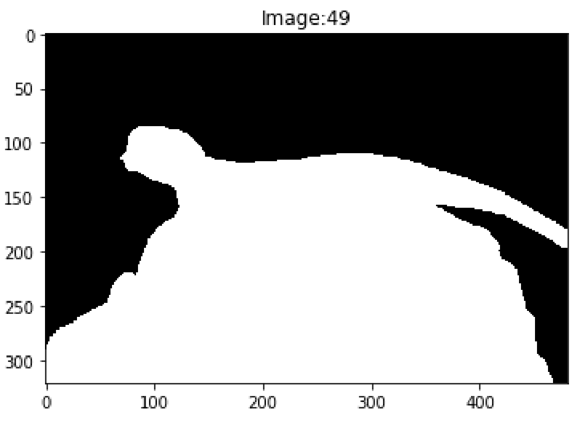


Image:49

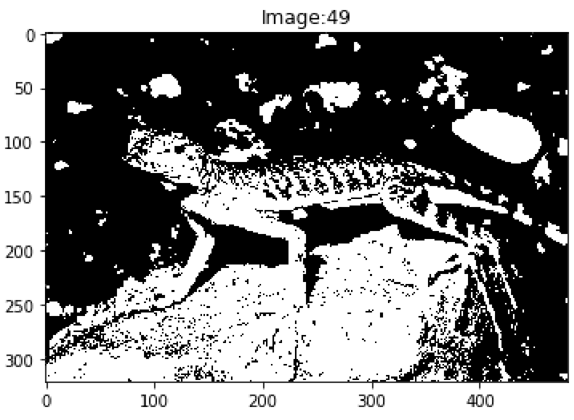


Image:49