Import library

In [66]:

```python
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz
import os
import cv2
import numpy as np
from time import time
import pandas as pd

%matplotlib inline
```

Dataset Paths

In [4]:

```python
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-t
```

Function to load images from folder sorted by name

In [7]:

```python
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 [8]:

```python
# 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 [9]:

```python
images, ID_data = load_images_from_folder(test_path)
images_ground_truth, ID_truth = load_images_from_folder(ground_truth_path)
```

In [10]:

```python
list_img = images

n_data = len(list_img)
```

Fuzzy C Means from scratch for image segmentation

In [ ]:

```
# Fuzzy C means
def compute_clusters(P, flatten_image, m, N ):
    c = [0, 0]
    m = 2
    for i in range(2):
        num = 0
        denom = 0
        for j in range(N):
            num = num + pow(flatten_image[j], m) * flatten_image[j] * P[i][j]
            denom = denom + pow(flatten_image[j], m) * flatten_image[j]
        c[i] = round(num/denom, 2)
    return c

def compute_weights(P, flatten_image, i, j, clusters): # ith point, # jth cluster
    denom = 0
    m = 2
    for k in range(2):
        temp = (abs(flatten_image[i] - clusters[j]))/(abs(flatten_image[i]-clusters[k]))
        denom = denom + pow(temp, 2/(m-1))
    return round((1/denom),2)

def update_partition_matrix(P, flatten_image, N, clusters):
    for i in range(2):
        for j in range(N):
            P[i,j] = round(compute_weights(P, flatten_image, j, i, clusters),2)
    return P

def compute_cost(P, flatten_image, clusters, N):
    cost = 0
    m = 2
    for i in range(N):
        for j in range(2):
            temp = P[j,i]
            cost = cost + pow(temp, m) * pow((flatten_image[i] - clusters[j]),2)
    return round(cost,2)

results  = []
for num in range(1):
    img = gray_images[num]
    # plt.imshow(img, cmap = 'gray')

    # Flatten image
    flatten_image = img.reshape(-1)
    K = 2
    N =len(flatten_image)

    # Select random clusters
    clusters = [random.randrange(0,255,1), random.randrange(0,255,1)]

    # Make partition Matrix - (2 X Num of data points)

    P = np.zeros(shape = (2, N ))

    # Randomly fill P with values between [0,1]

    for i in range(2):
        for j in range(N):
            P[i,j] = round(random.uniform(0,1), 2)


    for iters in range(20):
        # Compute new clusters

        print ('Iteration:', iters)
        clusters = compute_clusters(P, flatten_image, 2 , N)

        # Update Partition Matrix or weights

        P = update_partition_matrix(P, flatten_image, N, clusters)


        # Compute Cost

        # cost = compute_cost(P, flatten_image, clusters, N)
        #print(cost)

```

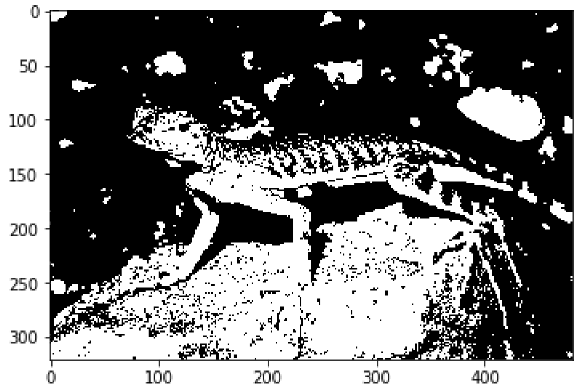
```python
    # Fill Results using P matrix

    new_image = np.zeros(shape = (N,1))
    for r in range(N):
        if P[0,r]  > P[1, r] :
            new_image[r] = 0
        else:
            new_image[r] = 1

    new_image = new_image.reshape(gray_images[num].shape)
    plt.imshow(new_image, cmap = 'gray')



```

Process

In [109]:

```python
# looping every images
results = []
for i in range(50):
    print('For image:', i)
    rgb_img = list_img[i]
    img = cv2.cvtColor(rgb_img,cv2.COLOR_RGB2GRAY)
    new_img = img.reshape(-1,1)
    N = new_img.shape[0]
    shape = np.shape(img)

    # error = 0.005
    # maximum iteration = 1000
    # cluster = 2

    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(new_img.T, 2, 2, error=0.005, maxiter=1000,
    # u = partition matrix
    new_image = np.zeros(shape = (N,1))
    for r in range(N):
        if u[0,r]  > u[1, r] :
            new_image[r] = 0
        else:
            new_image[r] = 1

    new_image = new_image.reshape(list_img[i].shape[0], list_img[i].shape[1])
    results.append(new_image)
    plt.imshow(new_image, cmap = 'gray')

```

```
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
```



In [116]:

```python
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
```

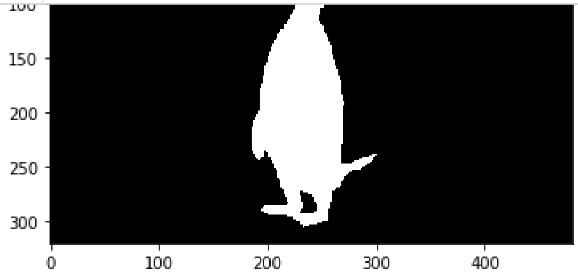
In [118]:

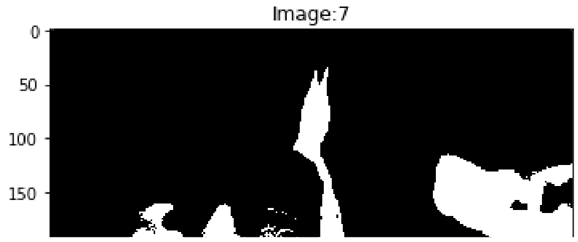
```python
ls = [1, 2, 3, 4, 5, 7, 9, 11, 13, 15, 21, 22, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 39, 4
for i in ls:
    results[i] = convert(results[i])
```

Results vs Ground Truths

In [119]:

```python
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')

```



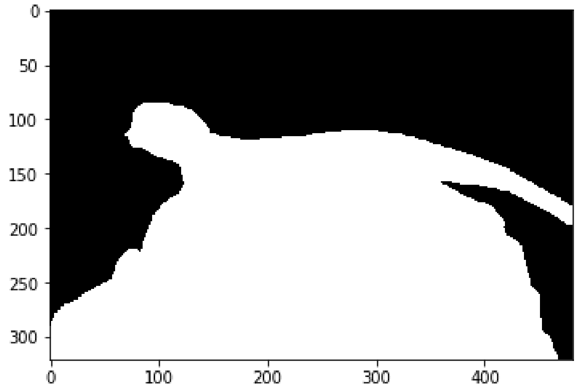


In [72]:

```
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[72]:

```
50
```



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

In [126]:

```
# Precision, Recall, F1 Score, IUC
# TP = 11 (ground_truth, result_data)
# FN = 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(round(precision,2))
        recalls.append(recall)
        accuracies.append(accuracy*100)
        IOUs.append(iou)
        F1_scores.append(f1_score)
    return precisions, recalls, accuracies, IOUs, F1_scores
```

In [127]:

```
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 pandas dataframe

In [128]:

```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[128]:

	Image_ID	Precision	Recall	F1_score	Accuracy	IOU
0	100098.jpg	0.99	0.347130	0.513464	58.340943	0.345410
1	101027.jpg	0.64	0.706929	0.671105	80.836264	0.505010
2	103006.jpg	0.57	0.279452	0.375599	48.153186	0.231223
3	103029.jpg	0.26	0.502324	0.343642	83.958005	0.207468
4	104010.jpg	0.40	0.329133	0.359918	62.091567	0.219451

In [129]:

```python
df.to_csv('FCM.csv', index=False)
```

In [ ]:

```python

```