Assigment 6

Upload your code (.ipynb) on Learn dropbox and submit pdfs of the code and the mathematical questions to Crowdmark.

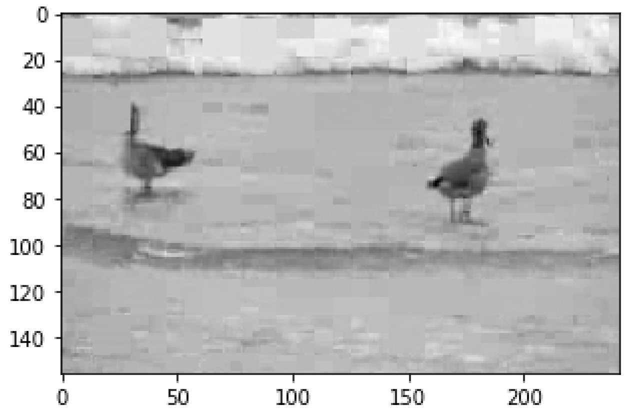
--

In [17]:
```
import matplotlib.pyplot as plt

# Numpy is useful for handling arrays and matrices.
import numpy as np
from scipy.sparse import coo_matrix
import time
from random import randrange as rnd
import sklearn
import sklearn.metrics
import random
import pandas as pd
from pathlib import Path
from imageio import imread
import time
```

In [2]:
```
initParams = {
    'lambda_': 0.001,
    'max_iter': 1000,
    'sigma': 1e-6,
    'delta': 1e-6,
    'k': 11000,
    'p': 0.09,
    'gamma': 0.001
}
```

In [3]:
```
#load data
shape = (156, 242)
A = np.zeros((71, 156 * 242))
for i in range (1, 72):
    filename = 'JPEGS/birds/frame_' + str(i) + '.jpg'
    img = np.array(imread(filename)).flatten()
    A[i - 1] = img.T
A = A.T
plt.figure(1, figsize = (5, 5))
plt.imshow(np.reshape(A[:, 0], shape), cmap = 'gray')
plt.show()
```



Rank-Sparsity

Question 1

Implement ADMM for the problem of separating a background image from foreground interference. Download the datasets at: http://www.svcl.ucsd.edu/projects/background_subtraction/JPEGS.tar.gz (http://www.svcl.ucsd.edu/projects/background_subtraction/JPEGS.tar.gz)

You will have to solve this problem:

$$\begin{aligned} \text{minimize } & \|L\|_* + \gamma\|M\|_1 \\ \text{subj. to: } & L + M = A \end{aligned}$$

where $\gamma > 0$ is a parameter that you will have to tune.

Use only the first dataset, birds, which contains 71 jpeg images each with 37752 gray-scale pixels. Form a matrix A of size 37752 x 71 with these images.

When you are done, print the background image (that is, columns of the component L) for frames 1, 11, ..., 71.

Note that this algorithm requires SVD. Computing the SVD in the usual way is very expensive for this dataset because the U matrix in SVD has size 37752 × 37752. You will have to utilize the "economy" SVD in https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.svd.html (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.svd.html) or https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html) by setting the option full_matrices = False.

Marks: 33.

In [10]:
```
class ADMM():
    def __init__(self):
        self.p = initParams['p']
        self.gamma = initParams['gamma']

    def __str__(self):
        return 'Alternating Direction Method of Multipliers'

    def objFunc(self, L, M, Y, A):
        LNorm = np.linalg.norm(L, ord = 'fro')
        MNorm = np.linalg.norm(M, 1)
        return LNorm + self.gamma * MNorm

    def LProx(self, M, A, Y):
        p = self.p
        z = -M + A - (1 / p) * Y
        u, s, vT = np.linalg.svd(z, full_matrices = False)

        s = np.maximum(s - 1/p, 0)
        sDense = np.zeros((u.shape[1], vT.shape[0]))
        sDense[:len(s), :len(s)] = np.diag(s)

        LNext = u @ sDense @ vT
        return LNext

    def MProx(self, L, A, Y):
        p = self.p
        gamma = self.gamma
        u = -L + A - (1 / p) * Y
        MNext = np.zeros_like(A)

        MNext[u >= gamma/p] = u[u >= gamma/p] - gamma/p
        MNext[np.abs(u) <= gamma/p] = 0
        MNext[u <= -gamma/p] = u[u <= -gamma/p] + gamma/p

        return MNext

    def admm(self, M, A, Y):
        max_iters = initParams['max_iter']
        for i in range(max_iters):
            if (i % 100 == 0): print('Iteration', i + 1, 'of', max_it
ers)
            L = self.LProx(M, A, Y)
            M = self.MProx(L, A, Y)
            Y = Y + self.p * (L + M - A)
        return L, M
```

In [18]:
```
M0 = np.random.rand(A.shape[0], A.shape[1])
Y0 = np.random.rand(A.shape[0], A.shape[1])
admm = ADMM()
start = time.time()
L, M = admm.admm(M0, A, Y0)
print('Done', initParams['max_iter'], 'iterations of', admm.__str__
(), 'in', time.time() - start, 's')
```

```
Iteration 1 of 1000
Iteration 101 of 1000
Iteration 201 of 1000
Iteration 301 of 1000
Iteration 401 of 1000
Iteration 501 of 1000
Iteration 601 of 1000
Iteration 701 of 1000
Iteration 801 of 1000
Iteration 901 of 1000
Done 1000 iterations of Alternating Direction Method of Multipliers i
n 207.82634019851685 s
```

In [12]:
```
print(admm.__str__())

f, axarr = plt.subplots(2, 4, figsize = (20, 10))
axarr[0,0].imshow(np.reshape(L[:, 0], shape), cmap = 'gray', vmin = 0
, vmax = 255)
axarr[0,0].title.set_text('Background of Image 1')

axarr[0,1].imshow(np.reshape(L[:, 10], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[0,1].title.set_text('Background of Image 11')

axarr[0,2].imshow(np.reshape(L[:, 20], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[0,2].title.set_text('Background of Image 21')

axarr[0,3].imshow(np.reshape(L[:, 30], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[0,3].title.set_text('Background of Image 31')

axarr[1,0].imshow(np.reshape(L[:, 40], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[1,0].title.set_text('Background of Image 41')

axarr[1,1].imshow(np.reshape(L[:, 50], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[1,1].title.set_text('Background of Image 51')

axarr[1,2].imshow(np.reshape(L[:, 60], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[1,2].title.set_text('Background of Image 61')

axarr[1,3].imshow(np.reshape(L[:, 70], shape), cmap = 'gray', vmin =
0, vmax = 255)
axarr[1,3].title.set_text('Background of Image 71')

plt.show()
```

```
Alternating Direction Method of Multipliers
```