MBA em Ciência de Dados

Técnicas Avançadas de Captura e Tratamento de Dados

Módulo VII - Dados não estruturados: sinais e imagens

Exercícios - com soluções

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

In [1]:
```
# carregando as bibliotecas necessárias
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Exercício 1)

Quando comparamos imagens e sinais e suas características, o que devemos considerar a priori?

(a) Sinais possuem valores independente e identicamente distribuídos, enquanto Imagens possuem pixels organizados de forma espacial
(b) Sinais possuem valores codificados em 16 bits, enquanto imagens possuem valores codificados em 8 bits
(c) Sinais possuem valores com dependência sequencial, enquanto imagens não possuem padrão de dependência
(d) Sinais possuem valores com dependência sequencial, enquanto Imagens possuem dependência espacial de seus valores

Resposta: *a priori, sempre devemos considerar que sinais e imagens possuem valores com dependência respectivamente sequencial e espacial. Ainda que sinais e imagens podem ser observados com dados i.i.d., quando isso acontece perdemos a coerencia espacial (no caso de imagens) e sequencial (no caso de sinais) que os caraterizam.*

Exercício 2)

Carregue os dados do arquivo `sinais2.csv` utilizando o comando:

```
signals = np.genfromtxt(arquivo, delimiter=',')
```

Esse array possui um sinal por linha `sinal[i]` . Calcule a autocorrelação (usada para identificar padrões de repetição) de cada sinal (utilizando a função vista em aula). Plote os sinals e o valor absoluto de suas respectivas autocorrelações considerando deslocamentos (*lags*) de 1 até 50.

Calcule também o desvio padrão `np.std()` da autocorrelação de 1 até 50 de cada sinal.

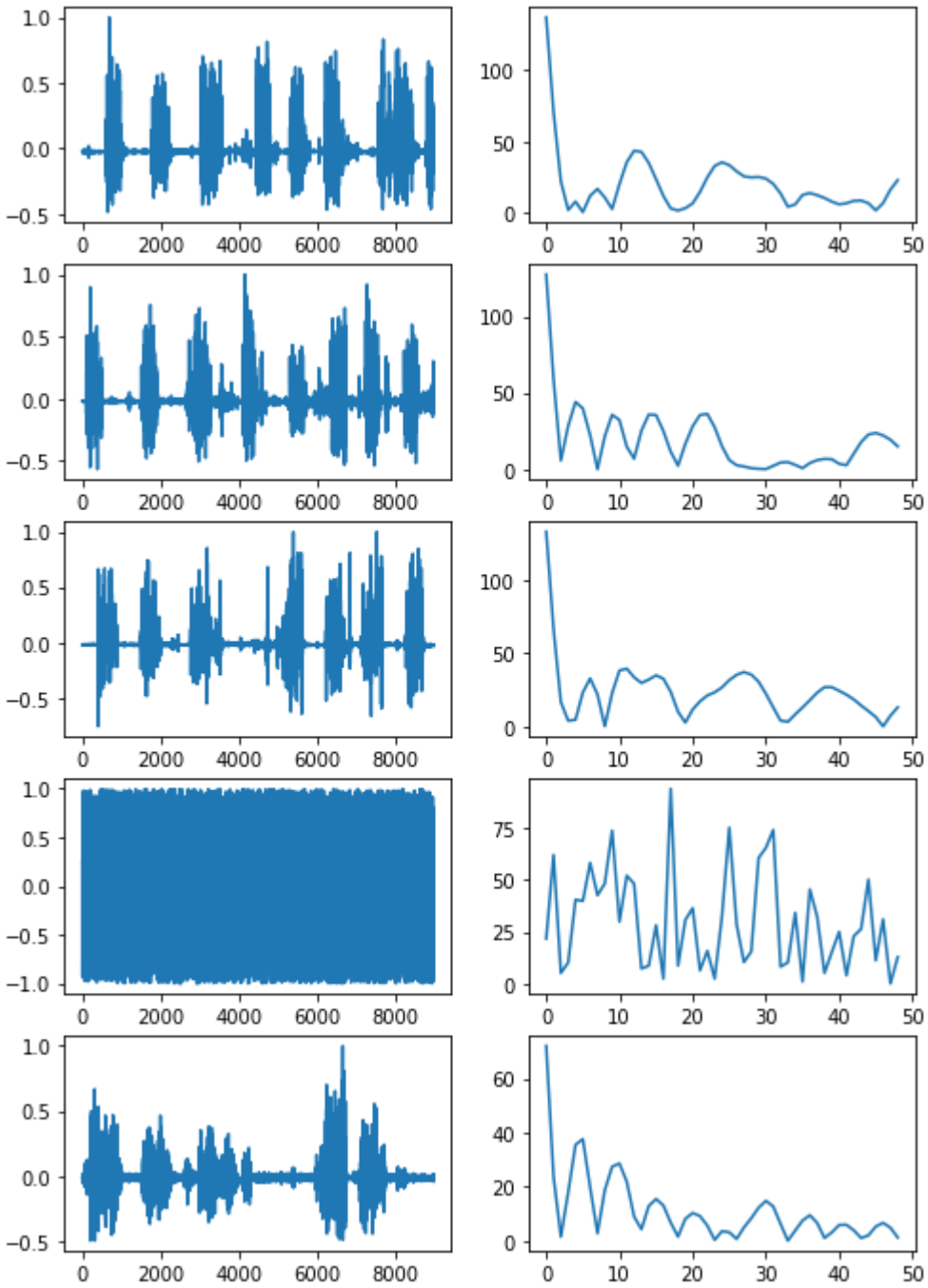
Observe que um dos sinais possui desvio padrão maior relativo à autocorrelação e analise o padrão da autocorrelação exibida no gráfico. Considerando a posição do sinal no array (de 0 a 4), escolha a alternativa verdadeira:

(a) A análise de autocorrelação mostra que os sinais 0 e 1 são sinais similares, sendo os demais bastante diferentes como evidenciado pelo desvio padrão da autocorrelação.
(b) A análise de autocorrelação mostra que o sinal 3 tem desvio padrão superior e gráfico mais instável, indicando que esse possa ser um sinal com menos padrões de repetição, e portanto menor dependência temporal do que os demais.
(c) A análise de autocorrelação indica que os sinais 0, 1, 2 e 3 possuem menor dependência temporal do que o sinal na posição 4, o qual é muito diferente dos demais sinais apresentando valores inferiores na análise.
(d) O desvio padrão da autocorrelação mostra que o sinal da posição 4 tem valor inferior aos demais, indicando que esse possa ser um sinal mais ruidoso.

Resposta: *ver código e resultados abaixo - o sinal na posiçao 3 possui maior desvio padrão de autocorrelação. Quando exibimos o gráfico, notamos que esse sinal provavelmente é apenas ruído, enquanto os demais (0, 1, 2 e 4) possuem algum padrão de repetição indicando dependências temporais mais evidentes.*

In [2]:
```
signals = np.genfromtxt('./dados/sinais2.csv', delimiter=
',').astype(np.float32)

def autocorrelation(f):
    ac = np.correlate(f, f, mode='full')
    return ac[ac.size // 2:]

plt.figure(figsize=(8,12))
for i in range(signals.shape[0]):
    ACi = autocorrelation(signals[i])
    plt.subplot(5,2,i*2+1); plt.plot(signals[i]);
    plt.subplot(5,2,i*2+2); plt.plot(np.abs(ACi[1:50]))
    print("sinal %d - std da AC: %.4f" % (i, np.std(ACi[1:5
0])))
```

```
sinal 0 - std da AC: 28.7686
sinal 1 - std da AC: 27.5278
sinal 2 - std da AC: 30.9749
sinal 3 - std da AC: 37.8208
sinal 4 - std da AC: 16.6157
```



Exercício 3)

Utilizando ainda os sinais carregados na questão anterior `sinais2.csv` , utilize a `np.fft.fft()` para obter a Transformada de Fourier dos sinais. Depois, considerando apenas frequências até 50, calcule quais são as 4 frequências de maior valor de magnitude (obtido pelo `np.abs()`). Aqui não queremos os valores da magnitude, mas a quais frequências (índices) elas se referem. Para complementar a análise, plote as magnitudes das transformadas até a frequência 50.

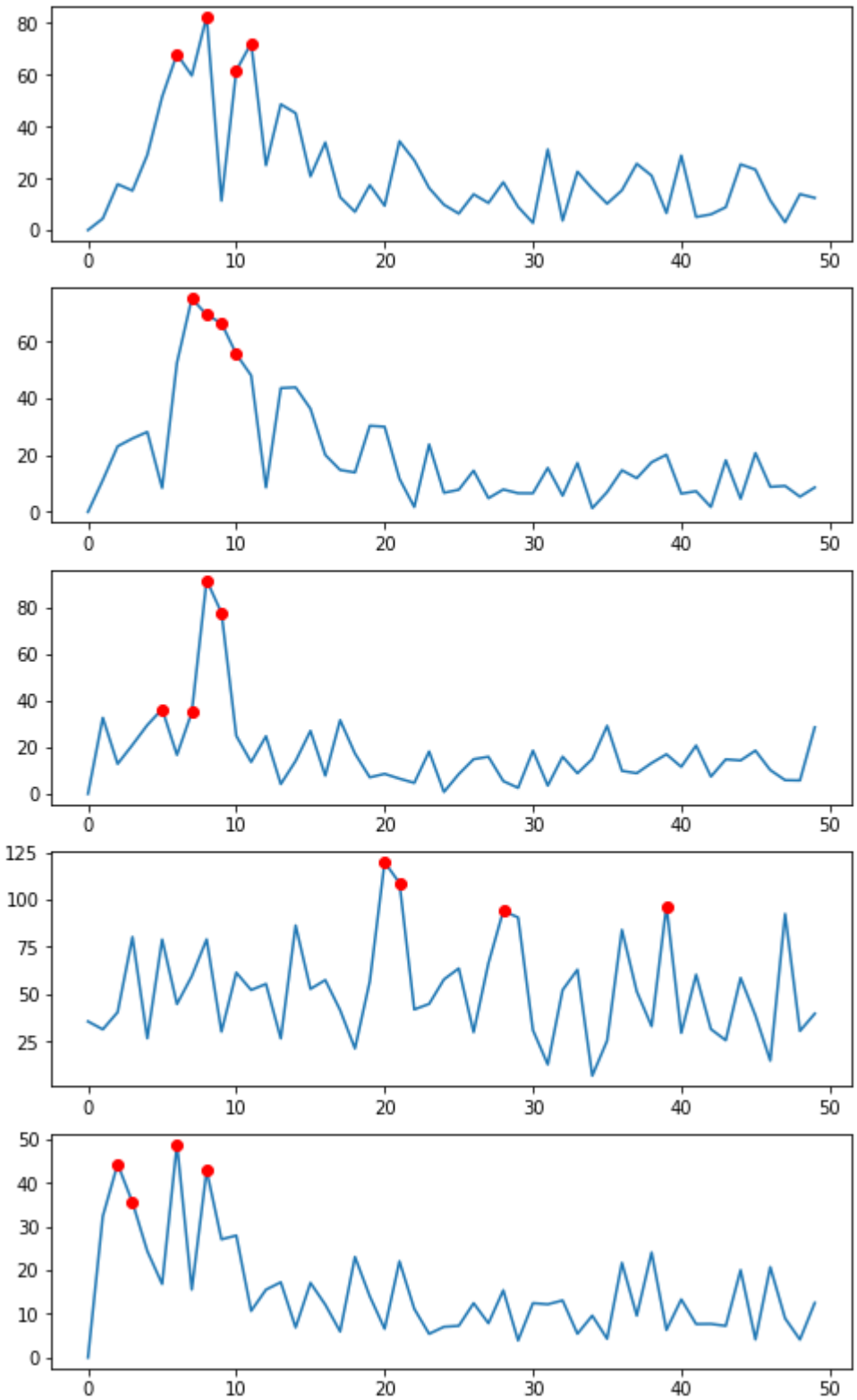
Analisando as frequências de maior magnitude temos as frequências que mais caracterizam o sinal. Considerando as 4 frequências computadas anteriormente, podemos dividir os sinais em categorias distintas. Nesse sentido, qual análise abaixo está correta?

(a) O sinal 4 possui frequências inferiores quando comparado com os demais, indicando que o sinal 4 é provavalmente dependente sequencialmente, enquanto os demais são i.i.d.; assim podemos dividí-los em duas categorias: sinal 4 e sinais 0, 1, 2 e 3.
(b) O sinal 3 possui frequências mais significativas 20 Hz ou superior, indicando que é um sinal com maior qualidade de aquisição, e assim podemos categorizar em: sinal 3, e sinais 0, 1, 2 e 4.
(c) Todas as frequências estão abaixo de 50 Hz, sendo assim podemos dizer que os sinais são todos similares, sendo impossível dividí-los em categorias.
(d) O sinal 3 possui frequências mais significativas 20 Hz ou superior, possuindo transições mais rápidas de valores do que os outros com frequências caracerísticas menores do que 12Hz; e assim podemos categorizar em: sinal 3, e sinais 0, 1, 2 e 4.

Resposta: *o sinal 3 se destaca por possuir frequências características na média 2 vezes superiores quando comparado aos demais, e acima de 20Hz. Assim como na análise de autocorrelação, esse sinal é distinto dos demais, cujas frequências são mais baixas. Mas frequências superiores não indicam melhor qualidade do sinal, descartando a alternativa (b), assim a opção (d) é a mais adequada.*

In [3]:
```
plt.figure(figsize=(8,14))
for i in range(signals.shape[0]):
    Fi = np.abs(np.fft.fft(signals[i]))
    k = 4
    Fi_50 = Fi[:50]
    ind = np.argpartition(Fi_50, -k)[-k:]
    print(i, ' frequências: ', np.sort(ind))
    plt.subplot(5,1,i+1); plt.plot(Fi_50);
    plt.plot(ind, Fi_50[ind], 'ro')
```

```
0  frequências:  [ 6  8 10 11]
1  frequências:  [ 7  8  9 10]
2  frequências:  [5 7 8 9]
3  frequências:  [20 21 28 39]
4  frequências:  [2 3 6 8]
```



Exercício 4)

Considerando os mesmos sinais carregados, compute as características: entropia da energia (com 10 blocos), taxa de cruzamentos por zero, entropia espectral (com 10 blocos), formando um vetor com 3 características para cada sinal.

Após isso, compute a matriz de distâncias entre os 4 sinais considerando a distância L1, i.e., a soma dos valores absolutos das diferenças entre dois vetores A e B:

$$\sum_i |A_i - B_i|$$

Da matriz, que indica a dissimilaridade entre pares de sinais, aplique uma soma na direção do eixo 0 (axis=0) e depois arredonde para inteiro `np.round(,0)` . Quais valores foram obtidos para cada sinal?

(a) Sinais 0, 1, 2 e 4, soma 2; Sinal 3, soma 7.
(b) Sinais 0 e 4, soma 3; Sinais 1 e 2, soma 2; Sinal 3, soma 7.
(c) Sinais 0, 1, e 2, soma 2; Sinal 3, soma 7; Sinal 4, soma 3.
(d) Sinais 0, 1, e 2, soma 1; Sinal 3, soma 5; Sinal 4, soma 2.

Resposta: *ver código abaixo.*

In [4]:
```
def entropia_energia(sinal, n_blocos=10):
    '''Entropia da energia do sinal'''
    # energia total
    energia_sinal = np.sum(sinal ** 2)
    M = len(sinal)

    # calcula janelas dentro do sinal
    M_janelas = int(np.floor(M / n_blocos))
    # verifica se tamanho dos blocos
    # é multiplo do tamanho do sinal
    if M != M_janelas * n_blocos:
        sinal = sinal[0:M_janelas * n_blocos]

    # monta matriz [M_janelas x n_blocos]
    janelas = sinal.reshape(M_janelas, n_blocos, order='F
').copy()

    # Computa energias de cada janela (normalizada pela do
sinal)
    e_janelas = np.sum(janelas ** 2, axis=0) / (energia_sin
al + 0.0001)
    #print(e_janelas)

    # Computa entropia entre energias das janelas
    entropia = -np.sum(e_janelas * np.log2(e_janelas + 0.00
01))
    return entropia

def taxa_cruzamentos_por_zero(sinal):
    '''Cruzamentos por zero em um intervalo de tempo '''
    M = len(sinal)
    cont_zero = np.sum(np.abs(np.diff(np.sign(sinal)))) / 2
    return np.float64(cont_zero) / np.float64(M - 1.0)

def entropia_espectral(sinal, n_blocos=16):
    """Computes the spectral entropy"""

    fft_abs = np.abs(np.fft.fft(sinal))

    entropia_esp = entropia_energia(fft_abs, n_blocos=n_blo
cos)

    return entropia_esp
```

In [5]:
```
features = []
for i in range(signals.shape[0]):
    f1 = entropia_energia(signals[i])
    f2 = taxa_cruzamentos_por_zero(signals[i])
    f3 = entropia_espectral(signals[i])
    features.append([f1, f2, f3])

features= np.array(features)
dmat = np.zeros([signals.shape[0],signals.shape[0]])
for i in range(signals.shape[0]):
    for j in range(signals.shape[0]):
        dmat[i,j] = np.sum(np.abs(features[i]-features[j]))
        #dmat[i,j] = np.sqrt(np.sum((features[i]-features
[j])**2))

print(np.round(np.sum(dmat, axis=1),0))
```

```
[2. 2. 2. 7. 3.]
```

Exercício 5)

Carregue as seguintes imagens da base de dados flickr_map_training:

```
img1 = imageio.imread("dados/flickr_map_training/107.jpg")
img2 = imageio.imread("dados/flickr_map_training/101.jpg")
img3 = imageio.imread("dados/flickr_map_training/112.jpg")
img4 = imageio.imread("dados/flickr_map_training/303.jpg")
img5 = imageio.imread("dados/flickr_map_training/400.jpg")
```

Implemente um descritor de cor que computa um histograma utilizando a composição dos canais RGB em um único canal utilizando a seguinte operação, sendo R, G e B as matrizes relativas a cada canal de cor:

$$I = R \cdot 0.3 + G \cdot 0.59 + B \cdot 0.11$$

Permita definir o número de bins do histograma por meio da sua função e, antes de retornar, normalize o histograma dividindo pela soma.

Depois, calcule a distância entre img1 carregada e as outras imagens (2, 3, 4, 5) utilizando: 16 bins e 4 bins. Qual foram as duas imagens mais similares, da mais próxima para a mais distante, nos dois casos?

(a) 16 bins: img2, img4 ; 4 bins: img2, img3
(a) 16 bins: img2, img3 ; 4 bins: img4, img3
(b) 16 bins: img2, img3 ; 4 bins: img2, img4
(d) 16 bins: img4, img2 ; 4 bins: img4, img3

Resposta: *ver código abaixo.*

In [6]:
```
import imageio

def histograma_global_intensity(img, n_colors):
    img_int = img[:,:,0].astype(float)*0.3 + img[:,:,1].ast
ype(float)*0.59 + img[:,:,2].astype(float)*0.11
    hist,_ = np.histogram(img_int, bins=n_colors)
    # normaliza o vetor resultante pela soma dos valores
    hist = hist.astype("float")
    hist /= (hist.sum() + 0.0001)
    return hist
```

In [7]:
```
img1 = imageio.imread("dados/flickr_map_training/107.jpg")
img2 = imageio.imread("dados/flickr_map_training/101.jpg")
img3 = imageio.imread("dados/flickr_map_training/112.jpg")
img4 = imageio.imread("dados/flickr_map_training/303.jpg")
img5 = imageio.imread("dados/flickr_map_training/400.jpg")

ncolors = 16
f1 = histograma_global_intensity(img1, ncolors)
f2 = histograma_global_intensity(img2, ncolors)
f3 = histograma_global_intensity(img3, ncolors)
f4 = histograma_global_intensity(img4, ncolors)
f5 = histograma_global_intensity(img5, ncolors)
features = np.vstack([f1, f2, f3, f4, f5])

dist = np.zeros(5)
for i in range(5):
    dist[i] = np.sum((np.abs(features[0]-features[i])))

plt.subplot(151); plt.imshow(img1); plt.title("%.4f"%  dist
[0]); plt.axis('off')
plt.subplot(152); plt.imshow(img2); plt.title("%.4f"% dist
[1]); plt.axis('off')
plt.subplot(153); plt.imshow(img3); plt.title("%.4f"% dist
[2]); plt.axis('off')
plt.subplot(154); plt.imshow(img4); plt.title("%.4f"% dist
[3]); plt.axis('off')
plt.subplot(155); plt.imshow(img5); plt.title("%.4f"% dist
[4]); plt.axis('off')
```

Out[7]:
```
(-0.5, 499.5, 343.5, -0.5)
```

In [8]:
```
ncolors = 4
f1 = histograma_global_intensity(img1, ncolors)
f2 = histograma_global_intensity(img2, ncolors)
f3 = histograma_global_intensity(img3, ncolors)
f4 = histograma_global_intensity(img4, ncolors)
f5 = histograma_global_intensity(img5, ncolors)
features = np.vstack([f1, f2, f3, f4, f5])

dist = np.zeros(5)
for i in range(5):
    dist[i] = np.sum((np.abs(features[0]-features[i])))

plt.subplot(151); plt.imshow(img1); plt.title("%.4f"%  dist
[0]); plt.axis('off')
plt.subplot(152); plt.imshow(img2); plt.title("%.4f"% dist
[1]); plt.axis('off')
plt.subplot(153); plt.imshow(img3); plt.title("%.4f"% dist
[2]); plt.axis('off')
plt.subplot(154); plt.imshow(img4); plt.title("%.4f"% dist
[3]); plt.axis('off')
plt.subplot(155); plt.imshow(img5); plt.title("%.4f"% dist
[4]); plt.axis('off')
```

Out[8]: (-0.5, 499.5, 343.5, -0.5)

Exercício 6)

Vamos repetir o procedimento da questão anterior, agora utilizando o descritor de texturas LBP visto em aula. Utilizaremos uma função que também realiza uma normalização dos valores máximos das imagens, bem como permite definir o raio, número de pontos e quantidade de bins para esse descritor, conforme abaixo.

Calcule a distância L1 entre img1 carregada e as outras imagens utilizando o descritor LBP com os seguintes parâmetros:

- número de pontos = 14
- raio = 2
- bins = 16

Quais foram as três imagens mais similares, da mais próxima para a mais distante?

(a) img3, img2, img5
(b) img2, img3, img4
(c) img3, img5, img2
(d) img5, img3, img2

Resposta: *ver código abaixo.*

In [9]:
```
from skimage import feature

def lbp_features(img, points=8, radius=1, n_bins=10):
    # LBP opera em imagens de um só canal, aqui vamos converter
    # RGB para escala de cinza usando o método Luminance
    img = np.array(img, dtype=np.float64, copy=False)
    if (len(img.shape) > 2):
        img = img[:,:,0]*0.3 + img[:,:,1]*0.59 + img[:,:,2]*0.11

    # normaliza a imagem para ter máximo = 255
    if (np.max(img) > 0):
        img = ((img/np.max(img))*255).astype(np.uint8)

    # aqui definimos o numero de pontos e o raio, padrao = 8, 1
    lbp = feature.local_binary_pattern(img.astype(np.uint8), points, radius, method="uniform")

    # lbp retorna um matriz com os códigos, então devemos extraír o histograma
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, n_bins+1), range=(0, n_bins))

    # normaliza o histograma
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)
    # return the histogram of Local Binary Patterns

    return hist
```

In [10]:
```
pts = 14
rad = 2
nbi = 16
f1 = lbp_features(img1, pts, rad, nbi)
f2 = lbp_features(img2, pts, rad, nbi)
f3 = lbp_features(img3, pts, rad, nbi)
f4 = lbp_features(img4, pts, rad, nbi)
f5 = lbp_features(img5, pts, rad, nbi)
features = np.vstack([f1, f2, f3, f4, f5])

dist = np.zeros(5)
for i in range(5):
    dist[i] = np.sum(np.abs(features[0]-features[i]))


plt.subplot(151); plt.imshow(img1); plt.title("%.4f"%  dist
[0]); plt.axis('off')
plt.subplot(152); plt.imshow(img2); plt.title("%.4f"% dist
[1]); plt.axis('off')
plt.subplot(153); plt.imshow(img3); plt.title("%.4f"% dist
[2]); plt.axis('off')
plt.subplot(154); plt.imshow(img4); plt.title("%.4f"% dist
[3]); plt.axis('off')
plt.subplot(155); plt.imshow(img5); plt.title("%.4f"% dist
[4]); plt.axis('off')
```

Out[10]: (-0.5, 499.5, 343.5, -0.5)

Exercício 7)

No método Bag-of-Features quais dos parâmetros pertencem ao *framework* influenciam mais drasticamente a performance do método no caso de uso em imagens?

(a) O tamanho do dicionário, a quantidade de cores nas imagens, a quantidade de classes do problema
(b) O tamanho do dicionário, o descritor base, o método utilizado para aprender o dicionário
(c) O descritor base e o número de componentes principais utilizados
(d) O tamanho do patch extraído da imagem, que deve ser compatível com a resolução das imagens

Resposta: *Ainda que todos os itens acima possam influenciar de alguma maneira, os parâmetros que podemos definir no framework Bag-of-Features são: o descritor base, o tamanho do dicionário, o método utilizado para aprender o dicionário, o tamanho do patch e o número de patches extraídos, sendo fundamentais para a performance do método: o tamanho do dicionário, o método utilizado para aprendê-lo e o descritor base.*

Exercício 8)

Execute o método Bag-of-Features estudado em aula, agora com os seguintes parâmetros:

- tamanho do patch = (13, 13)
- número de patches = 1000
- principais componentes = 10
- tamanho do dicionário = 50

Utilize imagens de consulta `flower.jpg` e `football.jpg` e recupere as 12 imagens mais similares utilizando o modelo BoF aprendido. Qual a proporção de imagens da mesma categoria da consulta?

(a) flower = 3/12, football = 3/12
(b) flower = 5/12, football = 2/12
(c) flower = 6/12, football = 0/12
(d) flower = 9/12, football = 1/12

Resposta: *ver codigo abaixo*

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt

from os import listdir
from imageio import imread
from sklearn.feature_extraction.image import extract_patche
s_2d
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from joblib import Parallel, delayed


def get_patches(img_file, random_state, tam_patch=(11, 11),
n_patches=250):
    '''Extração de subimagens a partir de uma imagem
       Parametros
           img_file: caminho para a imagem
           random_state: semente aleatoria
           tam_patches: tamanho de cada subimagem
           n_patches: numero maximo de subimagens a extrair
    '''

    img = imread(img_file)

    # Extrai subimagens
    patch = extract_patches_2d(img,
                               patch_size=tam_patch,
                               max_patches=n_patches,
                               random_state=random_state)

    return patch.reshape((n_patches,
                          np.prod(tam_patch) * len(img.shap
e)))
```

In [12]:
```
# Parametros do BOF
tam_patch = (13, 13) # param
n_patches = 1000 # param
path_imgs = './dados/flickr_map_training/'
random_state = 1
# pega lista de arquivos no caminho
l_imgs = listdir(path_imgs)

# total de imagens
n_imgs = len(l_imgs)

# Extrai patches de cada imagem, de forma paralela para cad
a imagem
# retorna uma lista do mesmo tamanho do número de imagens
patch_arr = Parallel(n_jobs=-1)(delayed(get_patches)(path_i
mgs+arq_img,
                                                     random_
state,
                                                     tam_pat
ch,
                                                     n_patch
es)
                                      for arq_img in l_imgs)

print('Patches extraídos para criação do dicionário de feat
ures')
print('Total de imagens = ', len(patch_arr))
print('Tamanho de cada array de patches = ', patch_arr[0].s
hape)
```

```
Patches extraídos para criação do dicionário de features
Total de imagens =  80
Tamanho de cada array de patches =  (1000, 507)
```

In [13]:
```
pca_components = 10 # param

# Criando matriz com todos os patches para aplicar PCA
patch_arr2 = np.array(patch_arr, copy=True)
patch_arr2 = patch_arr2.reshape((patch_arr2.shape[0] * patc
h_arr2.shape[1],
                                 patch_arr2.shape[2]))

print('Total de instancias = ', len(patch_arr2), ' de taman
ho = ', patch_arr2[0].shape[0])

# Construindo modelo de componentes principais
modelo_PCA = PCA(n_components=pca_components, random_state=
random_state)
modelo_PCA.fit(patch_arr2)
patch_pca = modelo_PCA.transform(patch_arr2)

print('Espaço de características PCA criado')

print('\tpatches = ', len(patch_pca), ' de tamanho = ', pat
ch_pca[0].shape[0])
```

```
Total de instancias =  80000  de tamanho =  507
Espaço de características PCA criado
        patches =  80000  de tamanho =  10
```

In [14]:
```
n_dic = 50 # parametro
random_state = 1

# Construindo o dicionário
kmeans_model = KMeans(n_clusters=n_dic,
                      verbose=False,
                      init='random',
                      random_state=random_state,
                      n_init=3)
kmeans_model.fit(patch_pca)

print('Dicionário aprendido')
```

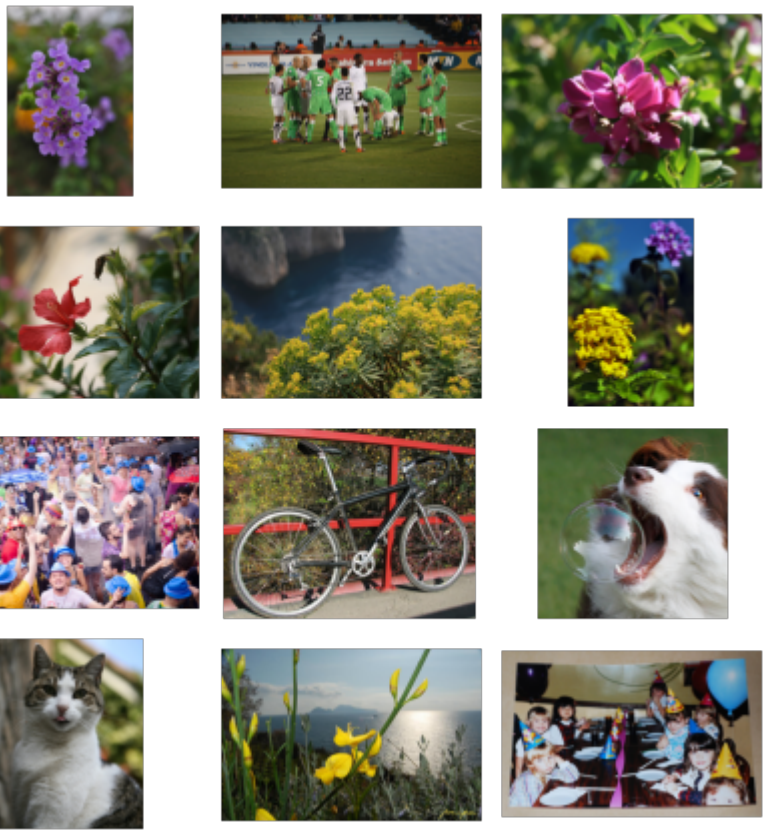
```
Dicionário aprendido
```

In [15]:
```
img_feats = []

# para cada imagem
for i in range(n_imgs):
    # predicao para os n_patches de uma imagem
    y = kmeans_model.predict(patch_pca[i*n_patches: (i*n_pa
tches)+n_patches])

    # computa histograma e armazena no array de features
    hist_bof,_ = np.histogram(y, bins=range(n_dic+1), densi
ty=True)
    img_feats.append(hist_bof)

img_feats = np.array(img_feats, copy=False)
print('Número de imagens e features = ', img_feats.shape)
```

```
Número de imagens e features =  (80, 50)
```

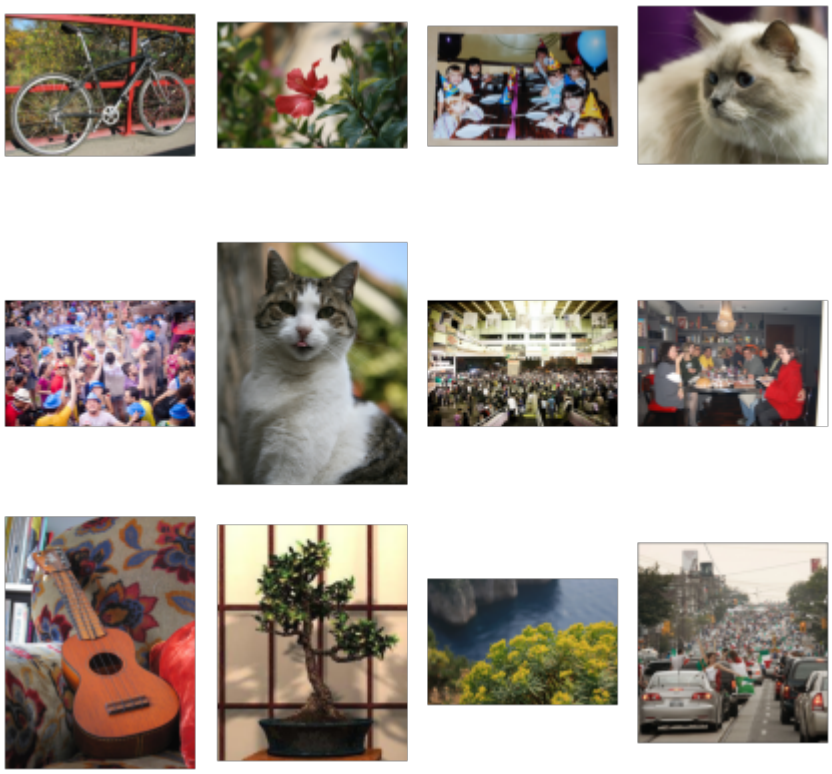
In [16]:
```
path_query = './dados/flickr_map_test/flower.jpg'
# pegando patches
query_patches = get_patches(path_query, random_state, tam_p
atch, n_patches)
query_patches = np.array(query_patches, copy=False)
print('Patches extraídos')
print(query_patches.shape)

# redimensionando e aplicando pca
query_pca = modelo_PCA.transform(query_patches)
print('PCA executado')
print(query_pca.shape)

# obtem palavras visuais
y = kmeans_model.predict(query_pca)
# computa histograma como feature
query_feats,_ = np.histogram(y, bins=range(n_dic+1), densit
y=True)

print('Features do BOF obtidas')

dists = []
for i in range(n_imgs):
    diq = np.sqrt(np.sum((img_feats[i]-query_feats)**2))
    dists.append(diq)

k = 12

# pega imagens mais proximas
k_cbir = np.argsort(dists)[:k]

print('Distancias calculadas')
print('imagem mais similar =', k_cbir[0], ' distancia =', d
ists[k_cbir[0]])

import imageio
imgq = imageio.imread(path_query)

fig, axes = plt.subplots(4, 3, figsize=(6, 6))
ax = axes.ravel()
imgs = []
cats = np.zeros(k)
for i in range(k):
    imgs.append(imageio.imread(path_imgs+l_imgs[k_cbir
[i]]))
    ax[i].imshow(imgs[i])
    ax[i].axis('off')
    cats[i] = int(l_imgs[k_cbir[i]][0])
fig.tight_layout()

tot_cat = 10
imgs_cat = np.sum(cats==6)
recall = imgs_cat/tot_cat
print("Recall = %.4f" % (recall))
print("Total na categoria = %d" % (imgs_cat))
```

```
Patches extraídos
(1000, 507)
PCA executado
(1000, 10)
Features do BOF obtidas
Distancias calculadas
imagem mais similar = 48  distancia = 0.2244415291339818
Recall = 0.6000
Total na categoria = 6
```



In [17]:
```
path_query = './dados/flickr_map_test/football.jpg'
# pegando patches
query_patches = get_patches(path_query, random_state, tam_p
atch, n_patches)
query_patches = np.array(query_patches, copy=False)
print('Patches extraídos')
print(query_patches.shape)

# redimensionando e aplicando pca
query_pca = modelo_PCA.transform(query_patches)
print('PCA executado')
print(query_pca.shape)

# obtem palavras visuais
y = kmeans_model.predict(query_pca)
# computa histograma como feature
query_feats,_ = np.histogram(y, bins=range(n_dic+1), densit
y=True)

print('Features do BOF obtidas')

dists = []
for i in range(n_imgs):
    diq = np.sqrt(np.sum((img_feats[i]-query_feats)**2))
    dists.append(diq)

k = 12
# pega imagens mais proximas
k_cbir = np.argsort(dists)[:k]

print('Distancias calculadas')
print('imagem mais similar =', k_cbir[0], ' distancia =', d
ists[k_cbir[0]])

import imageio
imgq = imageio.imread(path_query)

fig, axes = plt.subplots(3, 4, figsize=(6, 6))
ax = axes.ravel()
imgs = []
cats = np.zeros(k)
for i in range(k):
    imgs.append(imageio.imread(path_imgs+l_imgs[k_cbir
[i]]))
    ax[i].imshow(imgs[i])
    ax[i].axis('off')
    cats[i] = int(l_imgs[k_cbir[i]][0])
fig.tight_layout()

tot_cat = 11
imgs_cat = np.sum(cats==3) # categoria futebol = 3
recall = imgs_cat/tot_cat
print("Recall = %.4f" % (recall))
print("Total na categoria = %d" % (imgs_cat))
```

```
Patches extraídos
(1000, 507)
PCA executado
(1000, 10)
Features do BOF obtidas
Distancias calculadas
imagem mais similar = 57  distancia = 0.14515508947329403
Recall = 0.0000
Total na categoria = 0
```



Exercício 9)

Execute o método Bag-of-Features estudado em aula, agora com os seguintes parâmetros:

- tamanho do patch = (13, 13)
- número de patches = 1000
- tamanho do dicionário = 50
- descritor base = LBP com raio 2, 16 pontos e 10 bins

Vamos usar a versão da função LBP que permite usar como parâmetros o número de pontos e raio.

Utilize imagens de consulta `flower.jpg` e `football.jpg` e recupere as 12 imagens mais similares utilizando o modelo BoF aprendido. Qual a proporção de imagens da mesma categoria da consulta?

(a) flower = 5/12, football = 3/12
(b) flower = 6/12, football = 0/12
(c) flower = 6/12, football = 2/12
(d) flower = 6/12, football = 6/12

In [18]:
```
def lbp_features(img, points=8, radius=1, n_bins=10):
    # LBP opera em imagens de um só canal, aqui vamos conve
rter
    # RGB para escala de cinza usando o método Luminance
    img = np.array(img, dtype=np.float64, copy=False)
    if (len(img.shape) > 2):
        img = img[:,:,0]*0.3 + img[:,:,1]*0.59 + img[:,:,2]
*0.11

    # normaliza a imagem para ter máximo = 255
    if (np.max(img) > 0):
        img = ((img/np.max(img))*255).astype(np.uint8)

    # aqui definimos o numero de pontos e o raio, padrao =
8, 1
    lbp = feature.local_binary_pattern(img.astype(np.uint
8), points, radius, method="uniform")

    # lbp retorna um matriz com os códigos, então devemos e
xtraír o histograma
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0,
n_bins+1), range=(0, n_bins))

    # normaliza o histograma
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)
    # return the histogram of Local Binary Patterns

    return hist
```

In [19]:
```
# Parametros do BOF
tam_patch = (13, 13) # parametro
n_patches = 1000 # parametro
path_imgs = './dados/flickr_map_training/'
random_state = 1
# pega lista de arquivos no caminho
l_imgs = listdir(path_imgs)

# total de imagens
n_imgs = len(l_imgs)

# Extrai patches de cada imagem, de forma paralela para cad
a imagem
# retorna uma lista do mesmo tamanho do número de imagens
patch_arr = Parallel(n_jobs=-1)(delayed(get_patches)(path_i
mgs+arq_img,
                                                     random_
state,
                                                     tam_pat
ch,
                                                     n_patch
es)
                                      for arq_img in l_imgs)

print('Patches extraídos para criação do dicionário de feat
ures')
print('Total de imagens = ', len(patch_arr))
print('Tamanho de cada array de patches = ', patch_arr[0].s
hape)

# Criando matriz com todos os patches para aplicar PCA
patch_arr2 = np.array(patch_arr, copy=True)
patch_arr2 = patch_arr2.reshape((patch_arr2.shape[0] * patc
h_arr2.shape[1],
                                 tam_patch[0],tam_patch[0],
3))

print('Tamanho de cada array de patches = ', patch_arr2.sha
pe)
```

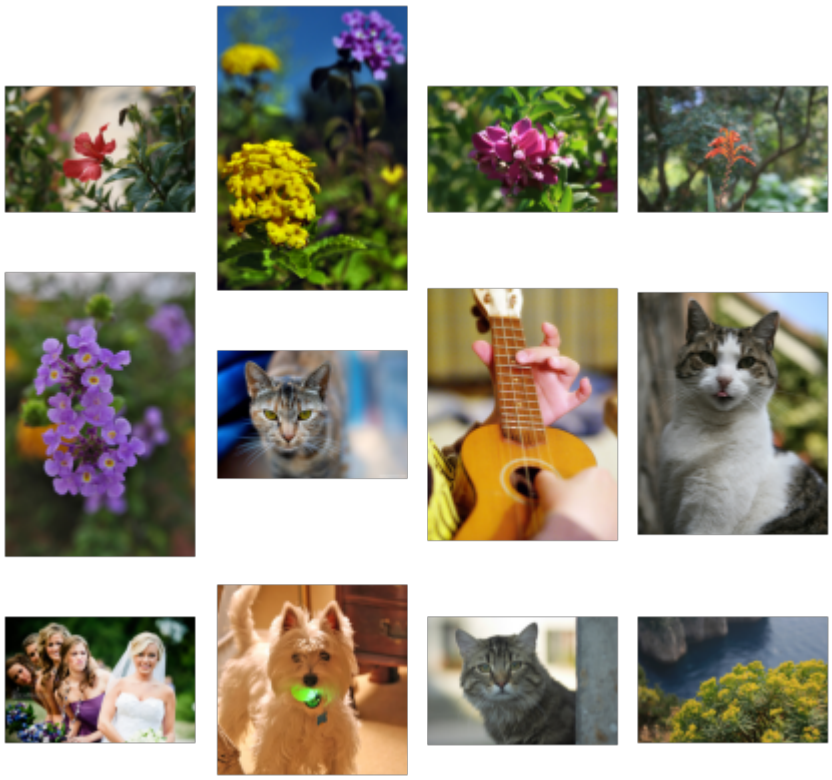
```
Patches extraídos para criação do dicionário de features
Total de imagens =  80
Tamanho de cada array de patches =  (1000, 507)
Tamanho de cada array de patches =  (80000, 13, 13, 3)
```

In [20]:
```
# obtendo features lbp para cada patch
patch_lbp = []
for pat in patch_arr2:
        f = lbp_features(pat,16,2,10)
        patch_lbp.append(f)

patch_lbp = np.array(patch_lbp, copy=False)

print('Total de instancias = ', len(patch_lbp), ' de tamanh
o = ', patch_lbp[0].shape[0])

print('Espaço de características LBP criado')

print('\tpatches = ', len(patch_lbp), ' de tamanho = ', pat
ch_lbp[0].shape[0])

n_dic = 50 # parametro
random_state = 1

# Construindo o dicionário
kmeans_model = KMeans(n_clusters=n_dic,
                      verbose=False,
                      init='random',
                      random_state=random_state,
                      n_init=3)
kmeans_model.fit(patch_lbp)

print('Dicionário aprendido')

img_feats = []

# para cada imagem
for i in range(n_imgs):
    # predicao para os n_patches de uma imagem
    y = kmeans_model.predict(patch_lbp[i*n_patches: (i*n_pa
tches)+n_patches])

    # computa histograma e armazena no array de features
    hist_bof,_ = np.histogram(y, bins=range(n_dic+1), densi
ty=True)
    img_feats.append(hist_bof)

img_feats = np.array(img_feats, copy=False)
print('Número de imagens e features = ', img_feats.shape)
```

```
Total de instancias =  80000  de tamanho =  10
Espaço de características LBP criado
        patches =  80000  de tamanho =  10
Dicionário aprendido
Número de imagens e features =  (80, 50)
```

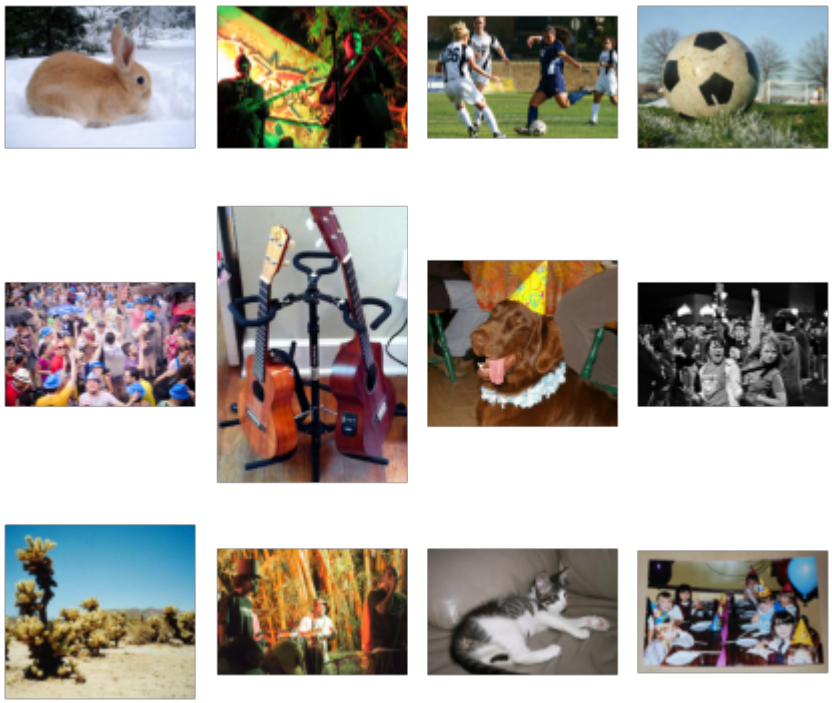
In [21]:
```
path_query = './dados/flickr_map_test/flower.jpg'
# pegando patches
query_patches = get_patches(path_query, random_state, tam_p
atch, n_patches)
query_patches = np.array(query_patches, copy=False)
print(query_patches.shape)

query_patches = query_patches.reshape((query_patches.shape
[0],
                                       tam_patch[0],tam_patch[0],
3))

print('Patches extraídos')
print(query_patches.shape)

# obtendo features LBP
query_lbp = []
for pat in query_patches:
        f = lbp_features(pat,16,2,10)
        query_lbp.append(f)

query_lbp = np.array(query_lbp, copy=False)
print('LBP executado')
print(query_lbp.shape)

# obtem palavras visuais
y = kmeans_model.predict(query_lbp)
# computa histograma como feature
query_feats,_ = np.histogram(y, bins=range(n_dic+1), densit
y=True)

print('Features do BOF obtidas')

dists = []
for i in range(n_imgs):
    diq = np.sqrt(np.sum((img_feats[i]-query_feats)**2))
    dists.append(diq)

k = 12
# pega imagens mais proximas
k_cbir = np.argsort(dists)[:k]

print('Distancias calculadas')
print('imagem mais similar =', k_cbir[0], ' distancia =', d
ists[k_cbir[0]])


import imageio
imgq = imageio.imread(path_query)

fig, axes = plt.subplots(3, 4, figsize=(6, 6))
ax = axes.ravel()
imgs = []
cats = np.zeros(k)
for i in range(k):
```

```
(1000, 507)
Patches extraídos
(1000, 13, 13, 3)
LBP executado
(1000, 10)
Features do BOF obtidas
Distancias calculadas
imagem mais similar = 6  distancia = 0.08070935509592429
Recall = 0.6000
Total na categoria = 6
```



In [22]:
```
path_query = './dados/flickr_map_test/football.jpg'
# pegando patches
query_patches = get_patches(path_query, random_state, tam_p
atch, n_patches)
query_patches = np.array(query_patches, copy=False)
print(query_patches.shape)

query_patches = query_patches.reshape((query_patches.shape
[0],
                                       tam_patch[0],tam_patch[0],
3))

print('Patches extraídos')
print(query_patches.shape)

# redimensionando e aplicando pca
query_lbp = []
for pat in query_patches:
        f = lbp_features(pat,16,2,10)
        query_lbp.append(f)

query_lbp = np.array(query_lbp, copy=False)
print('LBP executado')
print(query_lbp.shape)

# obtem palavras visuais
y = kmeans_model.predict(query_lbp)
# computa histograma como feature
query_feats,_ = np.histogram(y, bins=range(n_dic+1), densit
y=True)

print('Features do BOF obtidas')

dists = []
for i in range(n_imgs):
    diq = np.sqrt(np.sum((img_feats[i]-query_feats)**2))
    dists.append(diq)

k = 12
# pega imagens mais proximas
k_cbir = np.argsort(dists)[:k]

print('Distancias calculadas')
print('imagem mais similar =', k_cbir[0], ' distancia =', d
ists[k_cbir[0]])

import imageio
imgq = imageio.imread(path_query)

fig, axes = plt.subplots(3, 4, figsize=(6, 6))
ax = axes.ravel()
imgs = []
cats = np.zeros(k)
for i in range(k):
    imgs.append(imageio.imread(path_imgs+l_imgs[k_cbir
```

```
(1000, 507)
Patches extraídos
(1000, 13, 13, 3)
LBP executado
(1000, 10)
Features do BOF obtidas
Distancias calculadas
imagem mais similar = 5  distancia = 0.09338094023943003
Recall = 0.1818
Total na categoria = 2
```



Exercício 10)

Execute o método Bag-of-Features para aprender features nas imagens da pasta `flickr_map_training` conforme código fornecido em aula, com os seguintes parâmetros:

- tamanho do patch = (11, 11)
- número de patches = 300
- descritor base = PCA com 16 componentes
- random_state = 1
- para o KMeans use random_state=1 e n_init=3

Vamos investigar a influência do tamanho do dicionário no modelo gerado com os seguintes valores: 8, 16, 32, 64, 128, 256 e 512. Utilize a imagem de teste `flickr_map_test\flower.jpg` para recuperar as 16 imagens mais similares no conjunto de treinamento (sabendo que há 10 imagens dessa categoria no conjunto de treinamento). Calcule a revocação, ou seja, a razão entre o total de imagens de flores retornadas na busca das 16 mais similares e o número total de imagens de flores que deveriam ter sido retornadas (10).

DICA: as imagens de flores tem nome iniciando com o número '6'.

Quais tamanhos de dicionário resultam em maior revocação?

(a) 256 e 512
(b) 64 e 128
(c) 32, 64, 128 e 256
(d) 64, 128 e 256

Resposta: *Ver código abaixo. Nem sempre um maior dicionário resulta em melhor performance. Em geral um dicionário muito pequeno não representa bem a base de dados, e um dicionário muito grande gera uma espécie de sobreajuste (overfitting) dos dados.*

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

from os import listdir
from imageio import imread
from sklearn.feature_extraction.image import extract_patche
s_2d
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from joblib import Parallel, delayed


def get_patches(img_file, random_state, tam_patch=(11, 11),
n_patches=250):
    '''Extração de subimagens a partir de uma imagem
       Parametros
           img_file: caminho para a imagem
           random_state: semente aleatoria
           tam_patches: tamanho de cada subimagem
           n_patches: numero maximo de subimagens a extrair
    '''

    img = imread(img_file)

    # Extrai subimagens
    patch = extract_patches_2d(img,
                               patch_size=tam_patch,
                               max_patches=n_patches,
                               random_state=random_state)

    return patch.reshape((n_patches,
                          np.prod(tam_patch) * len(img.shap
e)))
```

In [3]:
```
# Parametros do BOF
tam_patch = (11, 11)
n_patches = 350
pca_components = 16
path_imgs = './dados/flickr_map_training/'
random_state = 1
# pega lista de arquivos no caminho
l_imgs = listdir(path_imgs)

# total de imagens
n_imgs = len(l_imgs)

# Extrai patches de cada imagem, de forma paralela para cad
a imagem
# retorna uma lista do mesmo tamanho do número de imagens
patch_arr = Parallel(n_jobs=-1)(delayed(get_patches)(path_i
mgs+arq_img,
                                                     random_
state,
                                                     tam_pat
ch,
                                                     n_patch
es)
                                      for arq_img in l_imgs)

print('Patches extraídos para criação do dicionário de feat
ures')
print('Total de imagens = ', len(patch_arr))
print('Tamanho de cada array de patches = ', patch_arr[0].s
hape)
```

```
Patches extraídos para criação do dicionário de features
Total de imagens =  80
Tamanho de cada array de patches =  (350, 363)
```

In [4]:
```
# Criando matriz com todos os patches para aplicar PCA
patch_arr2 = np.array(patch_arr, copy=True)
patch_arr2 = patch_arr2.reshape((patch_arr2.shape[0] * patc
h_arr2.shape[1],
                                 patch_arr2.shape[2]))

# Construindo modelo de componentes principais
modelo_PCA = PCA(n_components=pca_components, random_state=
random_state)
modelo_PCA.fit(patch_arr2)
patch_pca = modelo_PCA.transform(patch_arr2)

print('Espaço de características PCA criado')
print('\tpatches = ', len(patch_pca), ' de tamanho = ', pat
ch_pca[0].shape[0])
```

```
Espaço de características PCA criado
        patches =  28000  de tamanho =  16
```

In [5]:
```
search_dicts = [8, 16, 32, 64, 128, 256, 512]
random_state = 1

kmeans_d = []
# Construindo os dicionários
for n_dic in search_dicts:
    km = KMeans(n_clusters=n_dic,
                        verbose=False,
                        init='random',
                        random_state=random_state,
                        n_init=3)
    km.fit(patch_pca)
    kmeans_d.append(km)
    print('K =', n_dic, end=' ')

print('> Dicionários aprendidos')
```

```
K = 8 K = 16 K = 32 K = 64 K = 128 K = 256 K = 512 > Dicion
ários aprendidos
```

In [6]:
```
path_query = './dados/flickr_map_test/flower.jpg'
# pegando patches
query_patches = get_patches(path_query, random_state, tam_p
atch, n_patches)
query_patches = np.array(query_patches, copy=False)
print('Patches extraídos')
print(query_patches.shape)

# redimensionando e aplicando pca
query_pca = modelo_PCA.transform(query_patches)
print('PCA executado')
print(query_pca.shape)

d = 0
for n_dic in search_dicts:
    img_feats = []

    # para cada imagem
    for i in range(n_imgs):
        # predicao para os n_patches de uma imagem
        y = kmeans_d[d].predict(patch_pca[i*n_patches: (i*n
_patches)+n_patches])

        # computa histograma e armazena no array de feature
s
        hist_bof,_ = np.histogram(y, bins=range(n_dic+1), d
ensity=True)
        img_feats.append(hist_bof)

    # features dicionário atual
    img_feats = np.array(img_feats, copy=False)

    # obtem palavras visuais
    y = kmeans_d[d].predict(query_pca)
    d = d + 1

    # computa histograma como feature
    query_feats,_ = np.histogram(y, bins=range(n_dic+1), de
nsity=True)
    dists = []
    for i in range(n_imgs):
        diq = np.sqrt(np.sum((img_feats[i]-query_feats)**
2))
        dists.append(diq)

    # pega imagens mais proximas
    k = 16
    k_cbir = np.argsort(dists)[:k]
    cats = np.zeros(k)
    for i in range(k):
        cats[i] = int(l_imgs[k_cbir[i]][0])
    tot_cat = 10
    recall = np.sum(cats==6)/tot_cat
    print("N_dic = %d, Recall = %.4f" % (n_dic, recall))
```

```
Patches extraídos
(350, 363)
PCA executado
(350, 16)
N_dic = 8, Recall = 0.3000
N_dic = 16, Recall = 0.1000
N_dic = 32, Recall = 0.6000
N_dic = 64, Recall = 0.7000
N_dic = 128, Recall = 0.7000
N_dic = 256, Recall = 0.5000
N_dic = 512, Recall = 0.5000
```