Homework 1

In this homework, we will go through what we learned during the first three weeks, including basic linear algebra, least squares method, feature descriptor and matching, and bag of visual words.

You should finish this homework in this `hw1.ipynb` file using our provided templates. You can add other functions to solve the problems if necessary, but please only add them in this file.

The due for this homework is 12:00pm, Oct. 18th, Friday. We decided to prolong the time for you to finish this homework. Please submit this `hw1.ipynb` to Gradescope.

In the assignment folder, you will see:

- `features.py` , which contains some auxilary functions that help you extract and visualize keypoints;
- `*.png` and `*.jpg` files, which are some images you will work on in the feature descriptor question;
- `database` and `query` folders, which contain images of 5 classes selected from the famous ImageNet dataset, to be used to build bag of visual words for the last question;
- `hw1.ipynb` , which is the file you will work on and submit. Please only write your answers here.

In [1]:

```
# Setup

import random  # pseudo random number generator library
import numpy as np
from skimage import filters  # if skimage not installed, try "python3 -m pip ins
tall scikit-image"
from skimage.feature import corner_peaks
from skimage.io import imread
import matplotlib.pyplot as plt
from time import time
import math


# This code is to make matplotlib figures appear inline in the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0)  # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipytho
n
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

Question 1: Numpy Practice (10 points)

This question asks basic Numpy operations. Most questions already give you the right answer to check your implementation.

Question 1.1 (2 points)

Define the following using numpy:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

$$a = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$b = \begin{bmatrix} -1 \\ 5 \\ 2 \end{bmatrix}$$

Hint:

1. in Numpy, a 1-dimensional vector represents a column vector
2. if you want to review Numpy programming, check discussion 1 material on the course website (under the Schedule & Assignment tab).

In [2]:

```
### YOUR CODE HERE
M = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
a = np.array([1,3,1])
b = np.array([-1,5,2])
### END OF CODE
print("M = \n", M)
print("a = \n", a)
print("b = \n", b)
```

```
M =
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
a =
 [1 3 1]
b =
 [-1  5  2]
```

Question 1.2 (2 points)

Compute the dot product between a and b using numpy: $c = a^T b$.

In [3]:

```
# YOUR CODE HERE
c = np.dot(a,b)
# END OF CODE

assert c == 16
```

Question 1.3 (2 points)

Compute $(a^T b)Ma$ using Numpy

In [4]:

```
# YOUR CODE HERE
ans = (a @ b) * M @ a
# END OF CODE

assert ans[0] == 160 and ans[1] == 400 and ans[2] == 640 and ans[3] == 880
```

Question 1.4 (2 points)

Perform Singular Value Decomposition with Numpy:

$$M = U\Sigma V^T$$

In the equation above, Σ is a matrix with only diagonal elements, but in the question below, we only need the numbers on the diagonal.

In [5]:

```
# YOUR CODE HERE
# Decompose M (4x3 matrix) into U,S,V. where U is a 4x4 matrix, VT is a 3x3 matr
ix, and S is the singular values (a vector of size 3, the diagnoal of Sigma)
U,S,VT = np.linalg.svd(M)
# END OF CODE

print(S)
```

```
[2.54624074e+01 1.29066168e+00 2.31173375e-15]
```

Question 1.5 (2 points)

Perform eigen decomposition on the following matrix.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Find the largest eigenvalue and the corresponding eigenvector.

In [6]:

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
# YOUR CODE HERE
eigen_values, eigen_vectors = np.linalg.eig(M)
largest_eigen_value = max(eigen_values)
eigen_vector_for_largest_eigen_value = eigen_vectors[np.where(eigen_values == la
rgest_eigen_value)]
# END OF CODE
print(eigen_values)
print(eigen_vectors)
print(largest_eigen_value)
print(eigen_vector_for_largest_eigen_value)
```

```
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
16.116843969807043
[[-0.23197069 -0.78583024  0.40824829]]
```

Question 2 Least Squares Method (20 points)

Question 2.1 Over-determined System and Under-determined System (6 points)

1. consider a system of 4 equations and 3 unknowns (X, Y, Z), which is overdetermined (more equations than unknowns). (3 points)

$$\begin{bmatrix} 2 & 1 & 1 \\ -3 & 4 & 1 \\ -1 & 10 & 3 \\ -4 & 2 & 2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 2 \end{bmatrix}$$

1. consider a system of 2 equations and 3 unknowns (X, Y, Z), which is underdetermined (fewer equations than unknowns). (3 points)

$$\begin{bmatrix} -3 & 4 & 1 \\ -1 & 10 & 3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Find solutions for 1 and 2. If there is no solution, write down your proof.

Write down your calculation or proof of Question 2.1 here:
1.

1) 2X+Y+Z = 1

2) -3X+4Y+Z = 5

3) -X+10Y+3Z = 2

4) -4X+2Y+2Z = 2

From equation 1), 2)and 4) we can get X=0,Y=4/3,Z=-1/3

Plug the values we get into equation 3),0+10*4/3-3*1/3=37/3 != 2, which means this is not soluiton for equation 3).

Therefore, there is no solution for 1.

2.

1) -3X+4Y+Z = 5

2) -X+10Y+3Z = 2

From equation 2), we get -3X+30Y+9Z = 6 (*)

Take the difference of 1) and (*), we get 26Y+8Z = 1

Let Z = a be any variable, then Y = (1-8a)/26. Plug into equation 2), we get X = (-21-a)/13.

Therefore, X = (-21-a)/13, Y = (1-8a)/26, Z = a (a is any variable).

Question 2.2 Compute Pseudo Inverse (2 points)

Implement the get_pinv method, which takes a matrix as input, and outputs its pseudo inverse. Please call get_pinv (written by yourself) to calculate the pseudo-inverse of the two matrices:

$$A_1 = \begin{bmatrix} 2 & 1 & 1 \\ -3 & 4 & 1 \\ -1 & 10 & 3 \\ -4 & 2 & 2 \end{bmatrix}, A_2 = \begin{bmatrix} -3 & 4 & 1 \\ -1 & 10 & 3 \end{bmatrix}$$

(Hint: $A^{\dagger} = V\Sigma^{-1}U^{T}$)
You may use np.linalg.svd, but you may not use np.linalg.pinv. (You may secretly use it to check you answer. We won't know)

In [7]:

```
def get_pinv(matrix):
    """ Implement the pseudo inverse of a matrix
    Args:
        matrix: a numpy array with shape (m,n)

    Returns:
        out: the pseudo inverse of input matrix
    """
    ### YOUR CODE HERE
    U,S,VT = np.linalg.svd(matrix)
    m,n = matrix.shape
    inv_sigma = np.zeros((n,m))
    r = min(m,n)
    for i in range(r):
        inv_sigma[i][i] = 1/S[i]
    out = VT.T @ inv_sigma @ U.T
    ### END YOUR CODE

    return out
```

In [8]:

```
A_1 = np.array([[2, 1, 1], [-3,4,1],[-1, 10, 3], [-4, 2, 2]])
A_2 = np.array([[-3,4,1], [-1, 10, 3]])
pinv_A_1 = get_pinv(A_1)
pinv_A_2 = get_pinv(A_2)
print(pinv_A_1)
print(pinv_A_2)
```

```
[[ 0.15400411 -0.12217659  0.0523614  -0.09445585]
 [-0.10677618  0.09137577  0.1036961  -0.14784394]
 [ 0.46748802 -0.26865161 -0.02772074  0.4421629 ]]
[[-0.38172043  0.15053763]
 [-0.02688172  0.10215054]
 [-0.03763441  0.04301075]]
```

Question 2.3 Least squares solution for an over-determined system (4 points)

For an over-determined linear system $Ax = b$ where $A \in \mathbb{R}^{m \times n}, m > n$, there may not be any solution. However, we can formulate the following least square problem to obtain the best approximation that minimizes the residual of $Ax - b$:

$$\min_x \|Ax - b\|^2$$

1. Show that $A^\dagger = (A^T A)^{-1} A^T$ for a tall A matrix. (Hint: check the hint of Question 2.1)
2. Show that the solution to the above optimization can be written using the pseudo-inverse $A^\dagger$ of matrix A: $x = A^\dagger b$ (Hint: Calculate gradient to x)

Write down your proof of Question 2.2 here:
1.

From Question 2.1 $A^\dagger = V\Sigma^{-1}U^T$ and $A = U\Sigma V^T$, then $A^T = (U\Sigma V^T)^T = V\Sigma^T U^T$

Since U and V have orthonormal columns, $U^T U = I$ and $V^T V = I$, then
$A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$

Thus, $(A^T A)A^\dagger = (V\Sigma^2 V^T)V\Sigma^{-1}U^T = V\Sigma^2\Sigma^{-1}U^T = V\Sigma U^T = A^T$

Since A is tall matrix, and $A^T A = V\Sigma^2 V^T$, then $A^T A(V\Sigma^{-2}V^T) = I$, so $A^T A$ is invertible, $(A^T A)^{-1}$ exists.

Multiply $(A^T A)^{-1}$ to both sides, we get $(A^T A)^{-1}(A^T A)A^\dagger = (A^T A)^{-1}A^T$

Therefore,$A^\dagger = (A^T A)^{-1}A^T$.

2.

Let $J = ||Ax - b||^2 = (b - Ax)^T(b - Ax) = b^T b - 2(Ax)^T b + (Ax)^T Ax$

Then $\frac{dJ}{dx} = 0 - 2A^T b + 2A^T Ax = -2A^T b + 2A^T Ax$.

Set $\frac{dJ}{du} = 0$, we get $2A^T b = 2A^T Ax$. Check $\frac{dJ^2}{dx^2}||Ax - b||^2 = 2A^T A$ which is positive-semidefinite matrix, so the solution minimizes J.

Multiply $(A^T A)^{-1}$ to both sides ($A^T A$ is invertible from part 1), $(A^T A)^{-1}A^T b = (A^T A)^{-1}A^T Ax$, $A^\dagger b = x$.

Therefore, $x = A^\dagger b$ is the solution to the above optimization.

Question 2.4 Minimum norm solution for an under-determined system (6 points)

For an under-determined system $Ax = b$ where $A \in \mathbb{R}^{m\times n}$, $m < n$, there can be infinite solutions (in fact, a linear space), thus we cannot return a unique answer. However, there is one solution that has the minimum ℓ_2-norm. The minimum norm solution is obtained by solving this optimization problem:

$$\begin{aligned} \min_x \quad & \|x\|^2 \\ \text{subject to} \quad & Ax - b = 0 \end{aligned}$$

1. Show that
$$A^\dagger = A^T(AA^T)^{-1}$$
for a fat matrix.
2. Prove that for underdetermined system, $A^\dagger b$ gives the optimal solution to the minimum norm problem. (Hint: Use the Lagrangian multipliers method)

Write down your proof of Question 2.3 here:
1.

From Question 2.1, 2.2 $A^\dagger = V\Sigma^{-1}U^T, A = U\Sigma V^T, A^T = V\Sigma^T U^T$. And $U^T U = I$, $V^T V = I$, then $AA^T = U\Sigma^2 U^T$

Thus, $A^\dagger(AA^T) = V\Sigma^{-1}U^T(U\Sigma^2 U^T) = V\Sigma^{-1}\Sigma^2 U^T = V\Sigma U^T = A^T$

Since A is fat matrix, and $AA^T = U\Sigma^2 U^T$, then $AA^T(U\Sigma^{-2}U^T) = I$, so AA^T is invertible, $(AA^T)^{-1}$ exists.

Multiply $(AA^T)^{-1}$ to both sides, we get $A^\dagger(AA^T) = A^T(AA^T)^{-1}(AA^T)$

Therefore,$A^\dagger = A^T(AA^T)^{-1}$.

2.

By Lanrange multipliers method, let $\mathcal{L}(x,\lambda) = \|x\|^2 + \lambda^T(Ax-b) = x^T x + \lambda^T(Ax-b)$.

Set $\frac{\partial\mathcal{L}(x,\lambda)}{\partial x} = 0$ and $\frac{\partial\mathcal{L}(x,\lambda)}{\partial\lambda} = 0$, then $2x + A^T\lambda = 0$ (1), $Ax - b = 0$ (2).

Multiply A to both sides of (1), then $2Ax + AA^T\lambda = 0$, since $Ax = 0$ by (2), then $2b + AA^T\lambda = 0$, $AA^T\lambda = -2b$.

Multiply $(AA^T)^{-1}$ to both sides of (3) (AA^T is invertible from part 1), then $\lambda = -2(AA^T)^{-1}b$.

Plug into (1), then $2x - 2A^T(AA^T)^{-1}b = 0$, $x = A^T(AA^T)^{-1}b$.

Therefore, $x = A^\dagger b$ is the optimal solution to the minimum norm problem.

Question 2.5 Finish the calculation (2 points)

Implement a method called linear_solver, to solve the least squares problem or minimum norm problem for $Ax = b$.

It takes matrix A, b as input, and output a vector, which is the solution. For the linear equations in Question 2.1, use this solver to compute the solution for least squares problem (2.1.1) or minimum norm problem (2.1.2).

In [9]:

```
def linear_solver(matrix, vector):
    """ Implement a linear solver of a least squares problem/minimum norm proble
m
    Args:
        matrix: input matrix on LHS
        vector: target vector on RHS
    Returns:
        out: the solution for least squares problem/minimum norm problem
    """
    p_inv = get_pinv(matrix)
    ### YOUR CODE HERE
    out = p_inv @ vector
    ### END YOUR CODE
    return out
```

In [10]:

```
A_1 = np.array([[2, 1, 1], [-3, 4, 1],[-1, 10, 3], [-4, 2, 2]])
A_2 = np.array([[-3, 4, 1], [-1, 10, 3]])
b_1 = np.array([1, 5, 2, 2])
b_2 = np.array([5, 2])
ls_res = linear_solver(A_1, b_1)
min_norm_res = linear_solver(A_2, b_2)
print('least square result', ls_res)
print('minimum norm result', min_norm_res)
```

```
least square result [-0.54106776  0.26180698 -0.04688569]
minimum norm result [-1.60752688  0.06989247 -0.10215054]
```

Question 3: Local Feature Descriptors and Matching (30 points)

In this part, you will be implementing keypoint detector, feature descriptors, and feature matching. Keypoint detectors find salient points in an image, such as corners. Using information surrounding a keypoint, we can extract a feature descriptor. Once we have descriptors, we can use them to match keypoints between images by measuring feature similarities.

Prerequesite: Image Filtering and Box filter

Load the `chair.png` image and we can smooth it by a 5×5 box filter by the following code:

In [11]:

```
from scipy.signal import convolve
from skimage import filters

I = imread('chair.jpg', as_grey=True)
plt.imshow(I)
plt.axis('off')
plt.title('original image')
plt.show()

window_size = 5
box_filter=np.ones((window_size, window_size))/(window_size*window_size) # defin
e a box filter
from scipy.signal import convolve
I_box = convolve(I, box_filter, mode='same')
plt.imshow(I_box)
plt.axis('off')
plt.title('box smoothed image')
plt.show()
```

```
/usr/local/lib/python3.7/site-packages/skimage/io/_io.py:48: UserWar
ning: `as_grey` has been deprecated in favor of `as_gray`
  warn('`as_grey` has been deprecated in favor of `as_gray`')
```

Note that we do not need to flip the kernel when using the convolve function provided by scipy.signal, because this function really just does the `correlation` (or filtering) operation, instead of the true `convolution` defined in math books. Nonetheless, let us just get used to this engineering convention and call the scipy.signal.convolve to compute the filtering.

Question 3.1 Vertical Edge filter (5 points)

Use you favorate vertical edge filter to extract vertical edges from the `chair.jpg` image

In [12]:

```
img = imread('chair.jpg', as_grey=True)
I_vert_edge = None

### YOUR CODE HERE
vert_edge_filter=np.zeros((3, 3))
vert_edge_filter[:,0] = [1]*3
vert_edge_filter[:,2] = [-1]*3
I_vert_edge = convolve(img, vert_edge_filter, mode='same')
### END YOUR CODE

plt.imshow(I_vert_edge)
plt.axis('off')
```

Out[12]:

```
(-0.5, 255.5, 255.5, -0.5)
```

Question 3.2 Harris Corner Detector and Feature Matching (15 points)

In this question, you are going to implement Harris corner detector for keypoint localization. Review the lecture slides on Harris corner detector to understand how it works. The Harris detection algorithm can be divide into the following steps:

1. Compute x and y derivatives (I_x, I_y) of an image
2. Compute products of derivatives (I_x^2, I_y^2, I_{xy}) at each pixel
3. Compute matrix M at each pixel, where $$ M(x_0,y*0)* = *sum*{x,y} w(x-x_0,y-y_0) \begin{bmatrix}

   ```
   I_{x}^2 & I_{x}I_{y} \\
   I_{x}I_{y} & I_{y}^2
   ```

   \end{bmatrix} $$ Here, we set weight $w(x,y)$ to be a box filter of size 3×3 (the box is placed centered at (x_0, y_0)).

This looks slightly more complicated than what we taught in class. But if you stare at the equation for a while, you will realize that this is just the "smoothed" version of the M we taught in class, smoothed by a box filter w. Obviously, the smoothing should be done by a filtering operation.

1. Compute corner response $R = Det(M) - k(Trace(M)^2)$ at each pixel
2. Output corner response map $R(x, y)$

Step 1 is already done for you in the function **`harris_corners`** . You need to complete the function implementation and run the code below.

- Hint: You may use the function `scipy.signal.convolve` .
  You may refer to `chair_harris_ground_truth.png` for the expected output.

In [13]:

```
def harris_corners(img, window_size=3, k=0.04):
    """
    Compute Harris corner response map. Follow the math equation
    R=Det(M)-k(Trace(M)^2).

    Hint:
        You may use the function scipy.signal.convolve,
        which is already imported above

    Args:
        img: Grayscale image of shape (H, W)
        window_size: size of the window function
        k: sensitivity parameter

    Returns:
        response: Harris response image of shape (H, W)
    """

    H, W = img.shape
    window = np.ones((window_size, window_size))/(window_size *window_size)

    response = np.zeros((H, W))

    dx = filters.sobel_v(img)
    dy = filters.sobel_h(img)

    ### YOUR CODE HERE
    dx2 = dx * dx
    dy2 = dy * dy
    dxdy = dx * dy
    grad1 = convolve(dx2, window, mode = "same")
    grad2 = convolve(dy2, window, mode = "same")
    grad3 = convolve(dxdy, window, mode = "same")
    for i in range(H):
        for j in range(W):
            M = np.array([grad1[i][j],grad3[i][j],grad3[i][j],grad2[i][j]]).resh
ape(2,2)
            response[i][j] = np.linalg.det(M) - k*(np.trace(M)**2)
    ### END YOUR CODE

    return response
```

In [14]:

```
img = imread('chair.jpg', as_grey=True)

# Compute Harris corner response
response = harris_corners(img)

# Display corner response
plt.imshow(response)
plt.axis('off')
plt.title('Harris Corner Response')
```

Out[14]:

```
Text(0.5,1,'Harris Corner Response')
```

Question 3.3 Histogram of Oriented Gradients (10 points)

In this section, you are going to implement a simplified version of HOG descriptor. Similar to the SIFT feature introduced in class, HoG is also based on image gradients but with fewer engineering tricks. We implement (a simplied) HoG instead of SIFT to get some taste of image feature engineering.

HOG stands for Histogram of Oriented Gradients (Histograms of oriented gradients for human detection, by N. Dalal and B. Triggs). In HOG descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions. The steps of HOG are:

1. compute the gradient image in x and y
   Use the sobel filter provided by `skimage.filters.sobel_h` and `skimage.filters.sobel_v`
2. compute gradient histograms
   Divide image into patches, each of which contains 16×16 pixels.
3. A patch is then evenly partitioned as 2×2 cells. Then calculate the histogram of gradients in each cell, with each cell contains 8×8 pixels. Normalize the gradient vector in each cell to be a unit vector.
4. concatenate the unit vectors of each cell in a patch to be a single vector for encoding the patch

The function `corner_peaks` from `skimage.feature` performs non-maximum suppression to take local maxima of the response map and localize keypoints. You can just regard this function as a keypoint selection module given some manually designed thresholds.

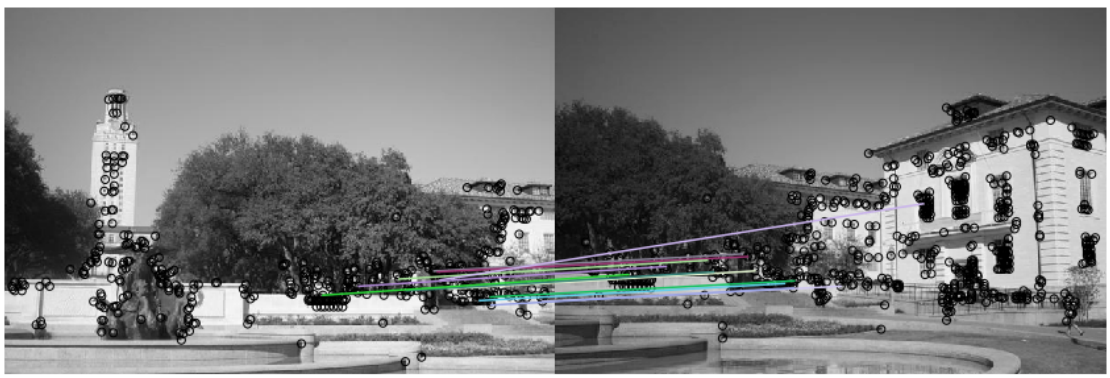
In [15]:

```
def pairwise_euclidean_distance(feature1, feature2):
    """
    Args:
        feature1: numpy array with shape (N, k)
        feature2: numpy array with shape (M, k)
    Returns:
        S: numpy array with shape (N, M) where S[i,j] is the l2 distance between
feature1[i] and feature2[j]
    """
    # YOUR CODE HERE
    N = feature1.shape[0]
    M = feature2.shape[0]
    S = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            S[i][j] = np.linalg.norm(feature1[i] - feature2[j])
    # END OF CODE

    return S


def hog_descriptor(patch, pixels_per_cell=(8,8)):
    """
    Generating hog descriptor by the following steps:

    1. compute the gradient image in x and y (already done for you)
    2. compute gradient histograms
    3. normalize across cells
    4. flattening cells into a feature vector

    Args:
        patch: grayscale image patch of shape (h, w)
        pixels_per_cell: size of a cell with shape (m, n)

    Returns:
        cells: 1D array of shape ((h*w*n_bins)/(m*n))
    """
    assert (patch.shape[0] % pixels_per_cell[0] == 0), \
        'Heights of patch and cell do not match'
    assert (patch.shape[1] % pixels_per_cell[1] == 0), \
        'Widths of patch and cell do not match'

    n_bins = 9
    degrees_per_bin = 180 // n_bins

    Gx = filters.sobel_v(patch)
    Gy = filters.sobel_h(patch)

    # Unsigned gradients
    G = np.sqrt(Gx**2 + Gy**2)
    theta = (np.arctan2(Gy, Gx) * 180 / np.pi) % 180

    G_cells = view_as_blocks(G, block_shape=pixels_per_cell) # view_as_blocks is
a function from skimage, which partitions a patch G into cells of pixels_per_cel
l size
    theta_cells = view_as_blocks(theta, block_shape=pixels_per_cell)
    rows = G_cells.shape[0]
    cols = G_cells.shape[1]
```

```
    cells = np.zeros((rows, cols, n_bins))
    # Compute histogram per cell

    ### YOUR CODE HERE
    for i in range(rows):
        for j in range(cols):
            temp = theta_cells[i][j].reshape(-1)
            for t in temp:
                cells[i][j][int((t%180)/degrees_per_bin)] += 1
            cells[i][j] = cells[i][j]/np.linalg.norm(cells[i][j])
    cells = cells.flatten()
    ### YOUR CODE HERE

    return cells
```

In [16]:

```
from features import describe_keypoints, plot_matches, match_descriptors
from skimage.util.shape import view_as_blocks

img1 = imread('uttower1.jpg', as_grey=True)
img2 = imread('uttower2.jpg', as_grey=True)

# Detect keypoints in both images
keypoints1 = corner_peaks(harris_corners(img1, window_size=3),
                          threshold_rel=0.05,
                          exclude_border=8)
keypoints2 = corner_peaks(harris_corners(img2, window_size=3),
                          threshold_rel=0.05,
                          exclude_border=8)

# Extract features from the corners
desc1 = describe_keypoints(img1, keypoints1,
                           desc_func=hog_descriptor,
                           patch_size=16)

desc2 = describe_keypoints(img2, keypoints2,
                           desc_func=hog_descriptor,
                           patch_size=16)

# Match descriptors in image1 to those in image2.
# You can read into match_descriptors and adjust the last parameter to see what
 will happen, which is a threshold for screening correspondences
matches = match_descriptors(desc1, desc2, pairwise_euclidean_distance, 0.76)

# Plot matches
fig, ax = plt.subplots(1, 1, figsize=(15, 12))
ax.axis('off')
plot_matches(ax, img1, img2, keypoints1, keypoints2, matches)
plt.show()
# compare your result with the provided `correspondence_ref_result.png`
```



Question 4: Visual Words and Image Retrieval (40 points)

We are now able to extract features using a simplified HOG. The next question is, how can we discribe an image by our feature discriptor? Recall what we learned about visual words, we can build a visual vocabulary by clustering the local features.

Question 4.1: 1-D KMeans (5 points)

Consider a very simple one-dimensional dataset. The data points are **2, 5, 12, 20, 23, 26**.

Run the algorithm using two clusters, where the initial guesses for the cluster centers are 3.5 (cluster 1) and 5.0 (cluster 2). For each step of the K-Means algorithm, specify the entries in each cluster and calculate the new mean below. Assume that a step consists of assigning points to each cluster and computing the new mean. If it converges in fewer than five steps, you may leave the remaining steps blank.

You can manually calculate the answers or write an program. You only need to provide a final answer below.

Step 1:

Cluster index of [2, 5, 12, 20, 23, 26]: [1, 2, 2, 2, 2, 2] (an example here)

Cluster 1 new mean: 2

Cluster 2 new mean: 17.2

Step 2:

Cluster index of [2, 5, 12, 20, 23, 26]: [1, 1, 2, 2, 2, 2]

Cluster 1 new mean: 3.5

Cluster 2 new mean: 20.25

Step 3:

Cluster index of [2, 5, 12, 20, 23, 26]: [1, 1, 2, 2, 2, 2]

Cluster 1 new mean: 3.5

Cluster 2 new mean: 20.25

Step 4:

Cluster index of [2, 5, 12, 20, 23, 26]:

Cluster 1 new mean:

Cluster 2 new mean:

Step 5:

Cluster index of [2, 5, 12, 20, 23, 26]:

Cluster 1 new mean:

Cluster 2 new mean:

In [17]:

```
def kmean(list1, a, b):
    for n in range(5):
        new = [0 if abs(i-a)<abs(i-b) else 1 for i in list1]
        a = np.mean([list1[i] for i in range(len(list1)) if new[i] == 0])
        b = np.mean([list1[i] for i in range(len(list1)) if new[i] != 0])
        print(a,b)
        print(new)
kmean([2,5,12,20,23,26],3.5,5)
```

```
2.0 17.2
[0, 1, 1, 1, 1, 1]
3.5 20.25
[0, 0, 1, 1, 1, 1]
3.5 20.25
[0, 0, 1, 1, 1, 1]
3.5 20.25
[0, 0, 1, 1, 1, 1]
3.5 20.25
[0, 0, 1, 1, 1, 1]
```

Question 4.2: Construct Visual Words

There are four steps for you to construct the visual words for an image

1. Divide an image into patches
2. Cluster the patches through K-Means and build the visual vocabulary
3. Index patches by the ID of the nearest cluster center (visual word)
4. Build a histogram of visual words by aggregating cluster IDs of patches in the entire image

We first implement a function to load all images in a folder for you:

In [18]:

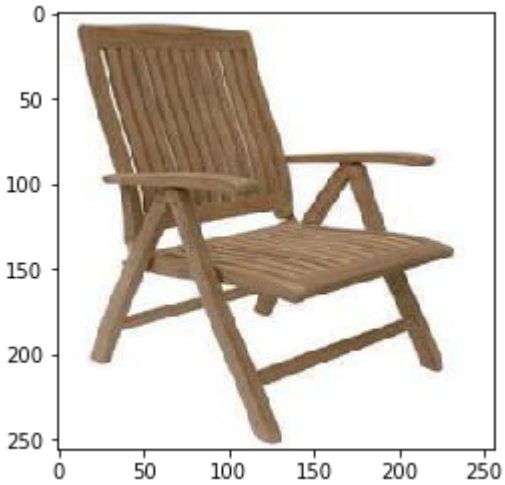
```
import os
def read_image(image_path):
    files = os.listdir(image_path)
    images = []
    for fi in files:
        if fi.endswith('.JPEG') or fi.endswith('.jpg') or fi.endswith('.jpeg'):
            image = imread(os.path.join(image_path, fi), as_grey=False)
            images.append(image)
    return np.array(images)
```

In [19]:

```
all_images = read_image('./database')
plt.imshow(all_images[0])  # have a look at the first image
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x11a1e3390>
```



4.2.1 Images to patches (5 points)

Implement the following function based on the comments. You may use any function from `sklearn.feature_extraction` , but it is not required.

The input images are already resized to $256 \times 256 \times 3$. Decompose each image densely and evenly as patches without overlapping. The patch_size by default is 16, which is both the width and height of the patch. As a result, there should be 16×16 patches.

In [20]:

```
def images_to_patches(images, patch_size):
    """
    Args:
        images: numpy array with shape (N, H, W, 3)  where N is total number of
 images, H is width, W is height
        patch_size: size of a square patch (patch_size, patch_size)

    Returns:
        patches: numpy array with shape (M, 16, 16, 3) where M is total number o
f patches (a lot)
    """

    ### YOUR CODE HERE
    N,H,W = images.shape[:3]
    patches = []
    for n in range(N):
        for x in range(int(H/patch_size)):
            for y in range(int(W/patch_size)):
                single = np.zeros((patch_size,patch_size,3))
                for row in range(patch_size):
                    for col in range(patch_size):
                        single[row][col] = images[n][x*patch_size+row][y*patch_s
ize+col]
                patches += [single]
    patches = np.array(patches)

#     max_patches = int(H*W/patch_size/patch_size)
#     from sklearn.feature_extraction import image
#     patches = image.PatchExtractor((patch_size, patch_size), max_patches ).tra
nsform(images)
    ### END OF CODE

    return patches
```

In [21]:

```
patches = images_to_patches(all_images, 16)
color_patches = patches.reshape([-1, 16*16*3])  # (M, 768), each patch is repres
ented by flattened color channels
```

4.2.2 KMeans (5 points)

You need to implement the `fit_kmeans` function to fit color_patches.

You may use `sklearn.cluster.KMeans` .

In [22]:

```
from sklearn.cluster import KMeans
```

In [23]:

```
def fit_kmeans(data, n_clusters=100):
    """
    Args:
        data: numpy array with shape (N, k), N is the number of data points, k i
s the dimension of feature vector.
        n_clusters: cluster number

    Returns:
        out: sklearn.cluster.KMenas model
    """
    ### YOUR CODE HERE
    kmeans = KMeans(n_clusters).fit(data)
    ### END OF CODE

    return kmeans
```

Run kmeans on patches to build the visual vocabulary. (It will take a while)

In [24]:

```
kmeans_model = fit_kmeans(color_patches)
```

4.2.3 Plot clusters (5 points)

Randomly choose 10 clusters from the Kmeans result. Plot 5 original patches ($16 \times 16 \times 3$) displayed in color for each of the clusters. Hint: use the `predict` function from `sklearn.cluster.KMeans`

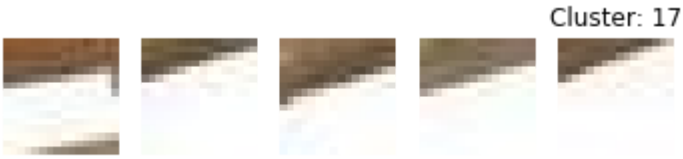
In [25]:

```
### YOUR CODE HERE
indices = np.random.choice(100, 10, replace=False)
pred = kmeans_model.predict(color_patches)

for i in indices:
    fig, ax = plt.subplots(1,5)
    plt.title("Cluster: "+str(i))
    choose = np.where(pred == i)[0]
    for j in range(5):
        ax[j].axis("off")
        ax[j].imshow(patches[choose[j]].astype(np.uint8))
plt.show()
### END OF CODE
```

Cluster: 29

Cluster: 14

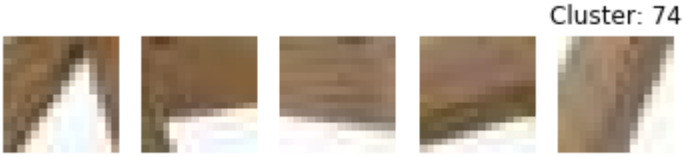
Cluster: 17



Cluster: 39

Cluster: 84

Cluster: 74



Cluster: 95

Cluster: 79

Cluster: 59

4.2.4 Visual word (10 points)

Implement `get_patch_indices` and `get_visual_words` functions below.

In [26]:

```
def get_patch_indices(patches, kmeans_model):
    """
        Args:
            patches: numpy array with shape (N, 16, 16, 3), where N is the numbe
r of batches
            kmeans_model: already fitted k-means model

        Returns:
            ind: the cluster indices that the patches belong to
    """

    ### YOUR CODE HERE
    ind = kmeans_model.predict(patches.reshape([-1, 16*16*3]))
    ### END YOUR CODE

    return ind


def get_visual_words(image, kmeans_model, patch_size=16):
    """
    Args:
        image: numpy array with shape (H, W, 3), which is a single image
        kmeans_model: the fitted kmeans model
        patch_size: size of a square patch

    Returns:
        hist: histogram of visual words that describes the images
    """
    patches = images_to_patches(image[None, :], patch_size=patch_size)
    # image[None, :] means expand the shape (H, W, 3) to (1, H, W, 3), so that i
s will fit your previous implementation of images_to_patches
    visual_words = get_patch_indices(patches, kmeans_model)

    # build histogram
    hist = [0]*len(kmeans_model.labels_)
    ### YOUR CODE HERE
    for vw in visual_words:
        hist[vw] += 1
    ### END YOUR CODE

    return hist
```

Read 5 query images from the `./query` foler

In [27]:

```
test_images = read_image('./query')
for i in range(test_images.shape[0]):
    plt.subplot(1, 5, i+1)
    plt.imshow(test_images[i])
    plt.axis('off')
```

```
/usr/local/lib/python3.7/site-packages/skimage/io/_io.py:48: UserWar
ning: `as_grey` has been deprecated in favor of `as_gray`
  warn('`as_grey` has been deprecated in favor of `as_gray`')
```

In [ ]:

```

```

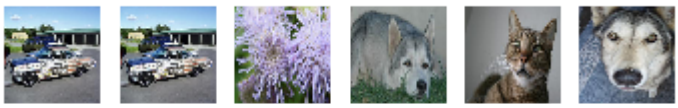
4.2.5 Image retrieval (10 points)

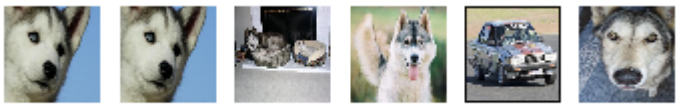
For each image in the `./database` folder, compute its bag of visual words representation. Then for each image in the `./query` folder, compute its bag of visual words representation.

For each test image, find 5 most similar images from the `./database` folder using cosine similarity between their bag of visual words representations, and plot them.

In [28]:

```
all_visual_words = np.array([get_visual_words(img, kmeans_model) for img in all_
images])
all_test_words = np.array([get_visual_words(img, kmeans_model) for img in test_i
mages])

### YOUR CODE HERE
for j in range(5):
    sim = []
    index = 0
    t = all_test_words[j]
    for v in all_visual_words:
        sim += [(t @ v/np.linalg.norm(t)/np.linalg.norm(v),index)]
        index +=1
    sort_sim = sorted(sim, reverse = True)
#     print(sort_sim[:5])
    fig,ax = plt.subplots(1,6)
    ax[0].axis("off")
    ax[0].imshow(test_images[j])
    for i in range(5):
        ax[i+1].axis("off")
        ax[i+1].imshow(all_images[sort_sim[i][1]])
plt.show()
### END OF CODE
```





Do you notice any unexpected retrival results? Explain why.

Yes, there are some unexpected retrival results. For example, in row 3, car is recognized as cat. In row 4, dogs and flowers are recognized as car. Also, in row 5, cars are recognized as dogs, which suggest that cars are hard to be recognized and are always mistakenly recognized as some other objects.

The explanation is that the cars are colorful and have various color patterns which makes it highly possible to share similar visual words with other objects such as dog.

4.2.6 Extra Credit (15 points)

Perform the whole image retrieval pipeline (4.2.1 - 4.2.5) with hog features. What are the differences in the final results compared to the color features? Explain.

Hint: load images in gray scale and perform the pipeline with gray scale

In [29]:

```
def read_image_grey(image_path):
    files = os.listdir(image_path)
    images = []
    for fi in files:
        if fi.endswith('.JPEG') or fi.endswith('.jpg') or fi.endswith('.jpeg'):
            image = imread(os.path.join(image_path, fi), as_grey=True)
            images.append(image)
    return np.array(images)
```

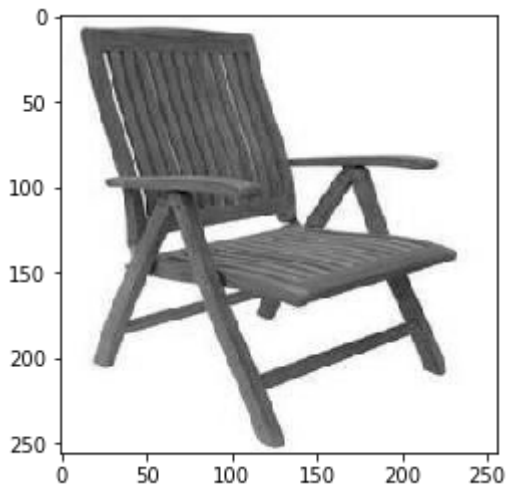
In [30]:

```
all_images_grey = read_image_grey('./database')
plt.imshow(all_images_grey[0])  # have a look at the first image
```

```
/usr/local/lib/python3.7/site-packages/skimage/io/_io.py:48: UserWar
ning: `as_grey` has been deprecated in favor of `as_gray`
  warn('`as_grey` has been deprecated in favor of `as_gray`')
```

Out[30]:

```
<matplotlib.image.AxesImage at 0x11d0e1550>
```



4.2.6.1 Images to patches

In [31]:

```
# Detect keypoints in both images
def images_to_patches_hog(images, patch_size):
    keypoints = []
    describe = []
    for img in images:
        kp = corner_peaks(harris_corners(img, window_size=3),
                          threshold_rel=0.05,
                          exclude_border=8)

        desc = describe_keypoints(img, kp,
                           desc_func=hog_descriptor,
                           patch_size=16)
        for d in desc:
            describe += [d]
    describe = np.array(describe)

    return describe
```

In [32]:

```
patches_hog = images_to_patches_hog(all_images_grey, 16)
```

4.2.6.2 KMeans

In [33]:

```
kmeans_hog = fit_kmeans(patches_hog)
```

4.2.6.3 Plot clusters

In [34]:

```
### YOUR CODE HERE
indices = np.random.choice(100, 10, replace=False)
pred_hog = kmeans_hog.predict(patches_hog)

for i in indices:
    fig, ax = plt.subplots(1,5)
    plt.title("Cluster: "+str(i))
    choose = np.where(pred_hog == i)[0]
    for j in range(5):
        ax[j].axis("off")
#         ax[j].imshow(np.expand_dims(describe[choose[j]], axis=0))
        ax[j].imshow(patches_hog[choose[j]].reshape(6,6))
plt.show()
### END OF CODE
```


Cluster: 69


Cluster: 73


Cluster: 49


Cluster: 83


Cluster: 4


Cluster: 78


Cluster: 65


Cluster: 70

4.2.6.4 Visual word

In [35]:

```
def get_patch_indices_hog(patches, kmeans_hog):
    ind = kmeans_hog.predict(patches)
    return ind

def get_visual_words_hog(image, kmeans_hog, patch_size=16):
    """
    Args:
        image: numpy array with shape (H, W, 3), which is a single image
        kmeans_model: the fitted kmeans model
        patch_size: size of a square patch

    Returns:
        hist: histogram of visual words that describes the images
    """
    patches = images_to_patches_hog(image[None, :], patch_size=patch_size)
    # image[None, :] means expand the shape (H, W, 3) to (1, H, W, 3), so that i
s will fit your previous implementation of images_to_patches
    visual_words = get_patch_indices_hog(patches, kmeans_hog)

    # build histogram
    hist = [0]*len(kmeans_hog.labels_)
    ### YOUR CODE HERE
    for vw in visual_words:
        hist[vw] += 1
    ### END YOUR CODE

    return hist
```

In [36]:

```
test_images_grey = read_image_grey('./query')
for i in range(test_images.shape[0]):
    plt.subplot(1, 5, i+1)
    plt.imshow(test_images_grey[i])
    plt.axis('off')
```

```
/usr/local/lib/python3.7/site-packages/skimage/io/_io.py:48: UserWar
ning: `as_grey` has been deprecated in favor of `as_gray`
  warn('`as_grey` has been deprecated in favor of `as_gray`')
```

4.2.6.5 Image retrieval

In [37]:

```
all_visual_words_hog = np.array([get_visual_words_hog(img, kmeans_hog) for img i
n all_images_grey])
all_test_words_hog = np.array([get_visual_words_hog(img, kmeans_hog) for img in
test_images_grey])

for j in range(5):
    sim = []
    index = 0
    t = all_test_words_hog[j]
    for v in all_visual_words_hog:
        sim += [(t @ v/np.linalg.norm(t)/np.linalg.norm(v),index)]
        index +=1
    sort_sim = sorted(sim, reverse = True)
#     print(sort_sim[:5])
    fig,ax = plt.subplots(1,6)
    ax[0].axis("off")
    ax[0].imshow(test_images_grey[j])
    for i in range(5):
        ax[i+1].axis("off")
        ax[i+1].imshow(all_images_grey[sort_sim[i][1]])
plt.show()
### END OF CODE
```

In general, it seems that there are more unexpected retrival results than that of color feature. However, it does a much better job in car. The explanation is that according to HOG, we care about patches around keypoints in a image. As HOG is not perfect and will make mistakes, it is possible that two irrelavent images have similar HOG features. So they share similar visual words which makes the performance worse than the color feature.