Yushuo Ruan
CSCI 5722 Computer Vision, Spring 2019
Instructor: Fleming
Homework 3, Due Saturday, March 2nd, by 11:00 pm

Problem Set

1. True. Any list of vectors containing the zero vector is linearly dependent. According to definition. A collection or list of n-vectors $a1, \dots, ak$ is called linearly dependent if
$$\beta_1 a_1 + \cdots + \beta_k a_k = 0$$
holds for some $\beta_1, \dots, \beta_k$that are not all zero. Suppose we have a zero vector a_i, then its coefficient β_i can be any number and make no effect to the equation. Therefore, the set of vectors is dependent.

2. True. Suppose a set of vectors $a1, \dots, ak$ is dependent, then at least one vector a_i can be expressed as a linear combination of the other vectors. Using the equation from Q1, a_i can be expressed as
$$a_i = \frac{-(\beta_1 a_1 + \cdots + \beta_{i-1} a_{i-1} + \beta_{i+1} a_{i+1} + \cdots + \beta_k a_k)}{\beta_i}$$
for any larger set of vector $a1, \dots, aj$ where $j > k$, a_i can also be expressed as a linear combination
$$a_i = \frac{-(\beta_1 a_1 + \cdots + \beta_{i-1} a_{i-1} + \beta_{i+1} a_{i+1} + \cdots + \beta_j a_j)}{\beta_i}$$
where $\beta_k, \dots, \beta_j = 0$. Therefore, this larger set of vectors is linear dependent.

3. Separability is matrix's ability to express itself in a lower dimensional vector. A two-dimensional filter kernel is separable of it can be expressed as the outer product of two vectors. A separable filter kernel can speedup computational complexity. Suppose we have a $n \times n$ filter, then the time required to multiply each pixel with the filter is n^2, while, after separation, the time required to multiply each pixel with two vectors is $2n$, which is faster than n^2.

4. Suppose we have a separable kernel
$$\begin{bmatrix} ad & ae & af \\ bd & be & bf \\ cd & ce & cf \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} * \begin{bmatrix} d & e & f \end{bmatrix}$$

And we have a cluster of pixels

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$$

when we perform the traditional 2D convolution, we get

$$new\_E = adA + aeB + afC + bdD + beE + bfF + cdG + ceH + cfI$$

when we perform the two 1D convolution, we get

step 1:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} convolve\ with \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} aA & aB & aC \\ bD & bE & bF \\ cG & cH & cI \end{bmatrix}$$

step 2:

$$\begin{bmatrix} aA & aB & aC \\ bD & bE & bF \\ cG & cH & cI \end{bmatrix} convolve\ with \begin{bmatrix} d & e & f \end{bmatrix}$$
$$= adA + aeB + afC + bdD + beE + bfF + cdG + ceH + cfI$$

which gives us the same result as the 2D convolution.

2D convolution of whole image takes $MN(2k+1)^2$, while the two 1D convolution of whole image takes $MN2(2k+1)$. So the number of operation saved is

$$\begin{aligned} & MN(2k+1)^2 - MN2(2k+1) \\ & = 4MNk^2 + 4MNk + MN - 4MNk - 2MN \\ & = 4MNk^2 - MN \\ & = MN(4k^2 - 1) \end{aligned}$$

5. Convolving a Gaussian with another Gaussian is equivalent to convolving a bigger Gaussian with larger radius, which is the square root of the sum of the squares of the two smaller Gaussians.

6. Dimensionality Reduction is the process of selecting or extracting set of variables or from a larger set to reduce the size of the data to be processed later. Dimensionality reduction is important in image processing, because image consist of large amount of data. A regular image can have millions of pixels, and each of then is considered a feature, so it usually needs large amount of effort to process. Dimensionality Reduction can reduce the size of image and save time and storage required during processing. However, if dimensionality reduction is overused or

used inappropriately, it can lead to data loss during image processing. If the dimensionality reduction applied on a image is highly biased, some important feature on the image could be lost; applying filter is not going to fix the image well.

7.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad A - \lambda I = \begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix}$$

$$\det(A - \lambda I) = \begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} = (a-\lambda)(d-\lambda) - bc = 0$$

$$ad - d\lambda - a\lambda + \lambda^2 - bc = 0$$

$$\lambda^2 - (a+d)\lambda + (ad - bc) = 0$$

$$\lambda_1 = \frac{a + d + \sqrt{(a+d)^2 - 4(ad-bc)}}{2}$$

$$\lambda_1 = \frac{a + d - \sqrt{(a+d)^2 - 4(ad-bc)}}{2}$$

$$(\boldsymbol{a}) \quad \boldsymbol{\lambda_1 * \lambda_2} = \frac{(a+d)^2 - (a+d)^2 + 4(ad-bc)}{4} = ad - bc = \mathbf{det}(\boldsymbol{A})$$

$$(\boldsymbol{b}) \quad \boldsymbol{\lambda_1 + \lambda_2} = \frac{a+d+a+d}{2} = a + d = \boldsymbol{trace}(\boldsymbol{A})$$

8.

a. Median filter works the best.
b. Gaussian filter with sigma = 1 pixel.

9.

a.
b. Rotate and translated version of the image works fine on harris algorithms. The scaled version of the image doesn't work so well.

c. 1. Adding constant: the over exposed regions lose the corners.
2. Subtracting constant: the dark regions lose the corners.
3. Multiplying constant: the over exposed regions lose the corners, but in some region, more corners are detected.
4. Dividing constant: the dark regions lose the corners.