# ECE549 Computer Vision: Assignment 0

Yutong Xie
NetID: yutongx6
Instructor: Prof. Saurabh Gupta

February 4, 2020

## 1 Question 3

### 1.1 3a: Combine

In this part, I divide each image into three equal parts and simply stacks them across the **RED** channel(the third color channel). The generated images are shown below.

Figure 1: Stack image 00125v

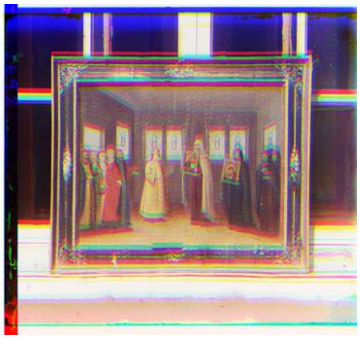

Figure 2: Stack image 00149v

Figure 3: Stack image 00153v

Figure 4: Stack image 00351v

Figure 5: Stack image 00398v

Figure 6: Stack image 01112v

### 1.2 3b: Align

In the last part, I found that images weren't aligned well, so in this section, I tried to align these images using both SSD and ZNCC approach. The general idea is to fix one channel and find the offset between fixed channel and the rest two other channels. I chose to fix blue channel. By using **np.roll()** function, I roll the image over boundaries in the range of [-15,15] on both height and width. And

I use SSD and ZNCC separately to calculate the similarity metric and find the maximum value and corresponding offset. For SSD approach, the calculated formula is

$$D(i,j) = \sum_{s=1}^{M}\sum_{t=1}^{N}[S(i+s-1, j+t-1) - T(s,t)]^2$$

Because SSD is used to find the sum of squared differences, we need to get the minimum value rather than maximum value. For ZNCC approach, the calculated formula is

$$\text{ZNCC}(a,b) = \sum_{i=1}^{H}\sum_{j=1}^{W}\left(\frac{a_{ij}-\text{mean}(a)}{\|a-\text{mean}(a)\|_2}\right)\left(\frac{b_{ij}-\text{mean}(b)}{\|b-\text{mean}(b)\|_2}\right)$$

After did alignment for each image using two different approach, I got the offset shown below.

| Image | Channel | SSD | NCC |
|---|---|---|---|
| 00125v | Green to blue | [6, 2] | [14, -1] |
| | Red to blue | [10, 1] | [9, -3] |
| 00149v | Green to blue | [4, 2] | [4, 2] |
| | Red to blue | [9, 2] | [7, 0] |
| 00153v | Green to blue | [14, 2] | [-15, 1] |
| | Red to blue | [11, 3] | [12, 3] |
| 00351v | Green to blue | [4, 0] | [4, 0] |
| | Red to blue | [13, 1] | [13, 0] |
| 00398v | Green to blue | [6, 2] | [9, -1] |
| | Red to blue | [8, -2] | [14, -2] |
| 01112v | Green to blue | [0, 0] | [0, 0] |
| | Red to blue | [7, 1] | [5, 1] |

Figure 7: Offset of different images using SSD and ZNCC.

According to these offsets, I align these images and the newly aligned images are shown below.

Figure 8: 00125v

Figure 9: 00149v

Figure 10: 00153v

Figure 11: 00351v

Figure 12: 00398v

Figure 13: 01112v

Figure 14: Aligned image based on SSD approach

Figure 15: 00125v

Figure 16: 00149v

Figure 17: 00153v

Figure 18: 00351v

Figure 19: 00398v

Figure 20: 01112v

Figure 21: Aligned image based on ZNCC approach

Through test, I found that ZNCC took longer computation time than SSD, which is easily to explain based on their formula. From the image set, we can see that most of images are aligned well, but some of them are still misaligned such as 00125v and 00153v using SSD approach. From my point of view, the white border in each image has great influence on the result of alignment. I tried to solve it in part 3d.

### 1.3 3c: Fast Align

In this part, I implemented a recursive version of image alignment based on ZNCC approach. First, I estimated an image's alignment on a low resolution version. Second, I used this alignment as a starting place to run the alignment. I down-sampled the image by a factor of 2. Hence, when I applied the first level alignment to the second level image, I multiplied the offset by 2. The intermediate offset, the offset at the full resolution and the overall total offset are shown below.

| Image | Channel | Intermediate offset | offset at full resolution | overall total offset |
|---|---|---|---|---|
| seoul | Green to blue | [1, 2] | [0, 1] | [2, 5] |
| | Red to blue | [1, -1] | [0, 1] | [2, -1] |
| vancouver | Green to blue | [-5, 5] | [1, 0] | [-9, 10 ] |
| | Red to blue | [-5, -6] | [2, 0] | [-8, -12] |

Figure 22: Offset of different images using fast align.

And also, the aligned images in color are shown.

Figure 23: Aligned image at the coarse resolution

Figure 24: Aligned image at the full resolution

Figure 25: Aligned image at the coarse resolution

Figure 26: Aligned image at the full resolution

## 1.4 Extra Credit

In part b, we found that three channels cannot be align exactly due to the white borders. Hence, I tried to eliminate the borders by cutting the raw image. I found that almost all the images have white borders and also black thin borders inside the white one. Hence, I tried to find the boundary that makes an image transfer between white and black.
First, I set two threshold to make the boundary be easier to check using the following code.

```
binary = np.copy(img)
    for i in range(x):
        for j in range(y):
            if int(img[j][i]) <= 80:
                binary[j][i] = 0
            elif int(img[j][i]) >= 220:
                binary[j][i] = 255
            else:
                binary[j][i] = img[j][i]
```

Then, I used two for loop to search the whole image and find all boundary point coordinates. I also added a restriction that the point should be located in the thin edge of the image, because there still existed point satisfied the boundary requirement inside the image. The relevant code is shown below.

```
for i in range(x-1):
        for j in range(y-1):
            if int(binary[j][i]) == 255 and binary[j][i+1] == 0 and i<0.05*x:
                left_x.append(i)
            elif int(binary[j][i]) == 0 and binary[j][i+1] == 255 and i>0.95*x:
                right_x.append(i)
            elif int(binary[j][i]) == 255 and binary[j+1][i] == 0 and j<0.01*y:
                top_y.append(j)
            elif int(binary[j][i]) == 0 and binary[j+1][i] == 255 and j>0.97*y:
                bottom_y.append(j)
```

After getting all the points, I select the maximum value for left boundary and top boundary and minimum value for bottom boundary and right boundary, because the white border may not perpendicular or parallel. We need to make sure that we can remove as many as white pixels. Finally, I cut the image based on the **left, right, top, and bottom** value.

```
left = max(left_x)
right = min(right_x)
top = max(top_y)
bottom = min(bottom_y)
output = img[top:bottom,left:right]
```

After deleting the white borders, the image matched exactly. I tested the improvement on ZNCC approach. In part b, image 00153v and image 00398v cannot be aligned perfectly, but the problem was solved as shown below.

Figure 27: ZNCC of image 153

Figure 28: Improved ZNCC of image 153

Figure 29: ZNCC of image 398

Figure 30: Improved ZNCC of image 398

Besides improving the image quality, I still tried to improve the speed. In part b, I applied SSD and ZNCC to the whole channel image which took a lot of time. However, I found that I could only select part of the image and calculate the similarity metrics, because the local offset should equal to the offset of whole image. In my program, I selected 50x50 section and the speed improvement can be visualized according to calculation time shown below. The left image indicates the time for whole image. The right one shows the time for 50x50 section.

```
MacBook-Pro:mp0 yutong$ python ncc.py
green offset, red offset:  [6, 2] [10, 1]
time cost 2.7198190689086914 s
green offset, red offset:  [4, 2] [9, 2]
time cost 2.9078171253204346 s
green offset, red offset:  [14, 2] [11, 3]
time cost 2.7991549968719482 s
green offset, red offset:  [4, 0] [13, 1]
time cost 2.651045322418213 s
green offset, red offset:  [6, 2] [8, -2]
time cost 2.6820333003997803 s
green offset, red offset:  [0, 0] [7, 1]
time cost 2.6360158920288086 s
```

Figure 31: Calculation time when applied ZNCC on the whole image

```
MacBook-Pro:mp0 yutong$ python improve_speed.py
green offset, red offset:  [5, 0] [9, 1]
time cost 0.2604031562805176 s
green offset, red offset:  [4, 2] [9, 2]
time cost 0.26216769218444824 s
green offset, red offset:  [7, 2] [-7, 0]
time cost 0.3664219379425049 s
green offset, red offset:  [4, 0] [13, 1]
time cost 0.3646237850189209 s
green offset, red offset:  [5, 2] [11, 4]
time cost 0.2814629077911377 s
green offset, red offset:  [0, 0] [5, 1]
time cost 0.2979319095611572 s
```

Figure 32: Calculation time when applied ZNCC on partial image

1. **Calculus Review [10 pts].**

The $\mathrm{softmax}$ function is a commonly-used operator which turns a vector into a valid probability distribution, i.e. non-negative and sums to 1.

For vector $\mathbf{z} = (z_1, z_2, \ldots, z_k) \in \mathbb{R}^k$, the output $\mathbf{y} = \mathrm{softmax}(\mathbf{z}) \in \mathbb{R}^k$, and its $i$-th element is defined as

$$y_i = \mathrm{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}. \quad (1)$$

(a) **[3 pts]** Verify that $\mathrm{softmax}(\mathbf{z})$ is invariant to constant shifting on $\mathbf{z}$, i.e. $\mathrm{softmax}(\mathbf{z}) = \mathrm{softmax}(\mathbf{z} + C)$ where $C \in \mathbb{R}$. The $\mathrm{softmax}(\mathbf{z} - \max_j z_j)$ trick is used in deep learning packages to ensure numerical stability.

$$\mathrm{softmax}(z + C1) = \frac{e^{(z_i + C)}}{\sum_{j=1}^{k} e^{(z_j + C)}} = \frac{e^{z_i} \cdot e^{C}}{e^{C} \cdot \sum_{j=1}^{k} e^{z_j}} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \mathrm{softmax}(z)$$

Hence, $\mathrm{softmax}(z)$ is invariant to constant shifting on $z$.

(b) **[3 pts]** Let $y_i = \mathrm{softmax}(\mathbf{z})_i, 1 \le i \le k$. Compute the derivative $\frac{\partial y_i}{\partial z_j}$ for any $i, j$. Your result should be as simple as possible, and may contain elements of $\mathbf{y}$ and/or $\mathbf{z}$.

$$\frac{\partial y_i}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_i}}{\sum_{a=1}^{k} e^{z_a}}\right) = \frac{(e^{z_i})' \Sigma - \Sigma' e^{z_i}}{\Sigma^2}$$

if $i = j$: $\frac{\partial y_i}{\partial z_j} = \frac{e^{z_i}\Sigma - e^{z_j} e^{z_i}}{\Sigma^2} = \frac{e^{z_i}(\Sigma - e^{z_j})}{\Sigma^2} = \frac{e^{z_i}}{\Sigma} \cdot \frac{\Sigma - e^{z_j}}{\Sigma} = y_i(1 - y_j)$

if $i \neq j$: $\frac{\partial y_i}{\partial z_j} = \frac{-e^{z_j} e^{z_i}}{\Sigma^2} = -y_i \cdot y_j$

Assume $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$

$$\frac{\partial y_i}{\partial z_j} = y_i(\delta_{ij} - y_j)$$

(c) **[4 pts]** Consider $\mathbf{z}$ to be the output of a linear transformation $\mathbf{z} = W^\top \mathbf{x}$, where vector $\mathbf{x} \in \mathbb{R}^d$ and matrix $W \in \mathbb{R}^{d \times k}$. Denote $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k\}$ as the columns of $W$. Let $\mathbf{y} = \mathrm{softmax}(\mathbf{z})$. Compute $\frac{\partial y_i}{\partial \mathbf{x}}$ and $\frac{\partial y_i}{\partial \mathbf{w}_j}$. (Hint: You may reuse (b) and apply the chain rule. Vector derivatives: $\frac{\partial(\mathbf{a}\cdot\mathbf{b})}{\partial \mathbf{a}} = \mathbf{b}$, $\frac{\partial(\mathbf{a}\cdot\mathbf{b})}{\partial \mathbf{b}} = \mathbf{a}$ . )

$(z_i = w_i^\top \cdot x)$

$$\frac{\partial y_i}{\partial x} = \frac{\partial y_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial x} = y_i(1 - y_i) w_i$$

$$\frac{\partial y_i}{\partial w_j} = \frac{\partial y_i}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_j} = y_i(\delta_{ij} - y_j) \cdot x$$

Assume Rotation matrix $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

2. **Linear Algebra Review [10 pts].**

(a) **[2 pts]** Let $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. Compute $V\begin{bmatrix}1\\0\end{bmatrix}, V\begin{bmatrix}0\\1\end{bmatrix}$. What does matrix multiplication $Vx$ do to $x$?

$V\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$ $V\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$

$V = \begin{bmatrix} \cos(-\frac{\pi}{4}) & -\sin(-\frac{\pi}{4}) \\ \sin(-\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix}$ $Vx$ rotates $x$ counterclockwise about $x$-axis by $-45°$.

(b) **[2 pts]** Verify that $V^{-1} = V^\top$. What does $V^\top x$ do?

$V^{-1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} = R(\frac{\pi}{4})$

$V^\top x$ rotates $x$ counterclockwise about x-axis by $45°$.

(c) **[2 pts]** Let $\Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \end{bmatrix}$. Compute $\Sigma V^\top x$ where $x = \begin{bmatrix}\sqrt{2}\\0\end{bmatrix}, \begin{bmatrix}0\\\sqrt{2}\end{bmatrix}, \begin{bmatrix}-\sqrt{2}\\0\end{bmatrix}, \begin{bmatrix}0\\-\sqrt{2}\end{bmatrix}$ respectively. These are 4 corners of a square. What is the shape of the result points?

Assume $x = (x_1 \cdot x_2 \cdot x_3 \cdot x_4)$

$\Sigma V^\top x_1 = \begin{bmatrix}\sqrt{3}\\1\end{bmatrix}$ $\Sigma V^\top x_3 = \begin{bmatrix}-\sqrt{3}\\-1\end{bmatrix}$

$\Sigma V^\top x_2 = \begin{bmatrix}-\sqrt{3}\\1\end{bmatrix}$ $\Sigma V^\top x_4 = \begin{bmatrix}\sqrt{3}\\-1\end{bmatrix}$

This is a rectangle.

(d) **[2 pts]** Let $U = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$. What does $Ux$ do? (Rotation matrix wiki)

$U = \begin{bmatrix} \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) \\ \sin\frac{\pi}{3} & \cos(\frac{\pi}{3}) \end{bmatrix}$ $Ux$ rotates $x$ counterclockwise about $x$-axis by $60°$.

(e) **[2 pts]** Compute $A = U\Sigma V^T$. From the above questions, we can see a geometric interpretation of $Ax$: (1) $V^T$ first rotates point $x$, (2) $\Sigma$ rescales it along the coordinate axes, (3) then $U$ rotates it again. Now consider a general squared matrix $B \in \mathbb{R}^{n\times n}$. How would you obtain a similar geometric interpretation for $Bx$?

According to singular value decomposition, $B$ equals to $U\Sigma V^T$, which means that $Bx$ rotates $x$ first, then rescale it, and rotates it again.