그래디언트 히스토그램 차이를 계산하여

각 점별로 가장 유사한 결과를 제시

소프트웨어학부 20192455 이주영, 기계공학부 20175806 김현우

실험 결과 설명

두 영상 입력 및 마우스 클릭으로 4개의 동일점의 patch(9x9) 저장

dsize=(500,500)로 두 영상의 크기를 조절하고 마우스의 왼쪽 버튼으로 클릭한 각 점의 좌표를 c 배열에 추가한다.

[그림1] 영상 입력 받고 마우스 클릭 대기

[그림2] patch(9x9) 저장

```
[226, 92, 24, 237, 286, 447, 489, 300, 422, 139, 131, 137, 110, 439, 423, 450]
1st 이미지에서 클릭한 좌표 4개
0: ( 226 , 92 )
1: ( 24 , 237 )
2: ( 286 , 447 )
3: ( 489 , 300 )
2nd 이미지에서 클릭한 좌표 4개
0: ( 422 , 139 )
1: ( 131 , 137 )
2: ( 110 , 439 )
3: ( 423 , 450 )
```

[그림3] 클릭 좌표

```
for i in range(8):
    if i in range(4):
        patch_image.append(simage[c[2 * i] - 4:c[2 * i] + 5, c[2 * i + 1] - 4:c[2 * i + 1] + 5])
    else:
        patch_image.append(simage2[c[2 * i] - 4:c[2 * i] + 5, c[2 * i + 1] - 4:c[2 * i + 1] + 5])

    patch_image[i] = np.float32(patch_image[i]) / 255.0
```

[그림4] [클릭 x좌표-4:클릭 x좌표+5, 클릭 y좌표-4:클릭 y좌표+5]

실험 방법1(이주영)

```
    gx.append(cv2.Sobel(patch_image[i], cv2.CV_32F, 1, 0, ksize=1))
    gy.append(cv2.Sobel(patch_image[i], cv2.CV_32F, 0, 1, ksize=1))

    mag.append((cv2.cartToPolar(gx[i], gy[i], angleInDegrees=True))[0])
    angle.append((cv2.cartToPolar(gx[i], gy[i], angleInDegrees=True))[1])

    angle[i][:, :, 0] = angle[i][:, :, 0] % 180
    angle[i][:, :, 1] = angle[i][:, :, 1] % 180
    angle[i][:, :, 2] = angle[i][:, :, 2] % 180

bin = [0, 20, 40, 60, 80, 100, 120, 140, 160]
```

[그림5] 그래디언트 방향과 크기 계산

Sobel 연산자를 사용하여 수평 및 수직 그래디언트를 계산한다. cartToPolar 함수를 사용하여 그래디언트의 크기와 방향을 계산한다. 3개의 채널의 그래디언트 방향을 0~180도가 되도록 변환하고 히스토그램은 0, 20, ..., 160도에 해당하는 9개의 빈을 포함한다.

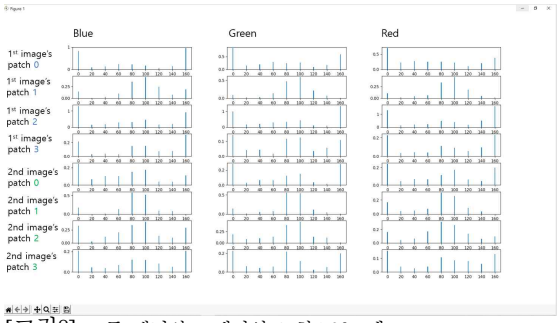
```
countr = np.zeros((8,9))

for m in range(8):
    for i in range(9):
        for j in range(9):
            for k in range(8):
                if angle[m][i, j, 0] == bin[k]:
                    countb[m,k] = countb[m,k] + mag[m][i, j, 0]
                elif bin[k] < angle[m][i, j, 0] < bin[k + 1]:
                    countb[m,k] = countb[m,k] + ((bin[k + 1] - angle[m][i, j, 0]) / 20) * mag[m][i, j, 0]
                    countb[m,k + 1] = countb[m,k + 1] + ((angle[m][i, j, 0] - bin[k]) / 20) * mag[m][i, j, 0]
                elif 160 < angle[m][i, j, 0]:
                    countb[m,0] = countb[m,0] + ((angle[m][i, j, 0] - bin[8]) / 20) * mag[m][i, j, 0]
                    countb[m,8] = countb[m,8] + ((180 - angle[m][i, j, 0]) / 20) * mag[m][i, j, 0]
```

[그림6] Blue 채널의 그래디언트 히스토그램 계산

그래디언트 크기는 해당 방향과 인접한 두 빈의 차이와 반비례하게 나눈다. 160~180도의 인접한 두 빈은 160도와 0도이다.

```
그래디언트 히스토그램의 가로축은 각도 0도, 20도, 40도, 60도, 80도, 100도, 120도, 160도를 의미
Blue 채널에 대한 1st 이미지의 0,1,2,3 패치와 2nd 이미지의 0,1,2,3 패치의 그래디언트 히스토그램 8x9
[[8.22241125e-01 8.19720451e-02 1.23697947e-01 2.28508769e-01
  2.23236896e-01 1.72152689e-01 6.17446645e-02 1.39589180e-01
  9.53972260e-01]
```

[그림7] 1st 이미지의 0 패치의 Blue 채널의 그래디언트 히스토그램



[그림8] 모든 채널의 그래디언트 히스토그램

```
d = np.zeros((4,4))

for i in range(4):
    for j in range(4):
        for k in range(9):
            d[i, j] += (countb[i, k] - countb[j + 4, k]) * (countb[i, k] - countb[j + 4, k])
            d[i, j] += (countg[i, k] - countg[j + 4, k]) * (countg[i, k] - countg[j + 4, k])
            d[i, j] += (countr[i, k] - countr[j + 4, k]) * (countr[i, k] - countr[j + 4, k])
```

[그림9] 패치끼리의 모든 채널의 그래디언트 히스토그램 거리 합 계산

1st 이미지의 패치 0, 1, 2, 3과 2nd 이미지의 패치 0, 1, 2, 3 사이의 모든 채널의 그래디언트 히스토그램 거리 합을 계산한다.

실험 결과1(이주영)

[그림10] 각 점별로 가장 유사한 결과를 제시

```
행: 1st 이미지의 0,1,2,3 패치, 열: 2nd 이미지의 0,1,2,3 패치
각 채널(B,G,R)에 대한 그래디언트 히스토그램 거리의 합 4x4 행렬
[[2.01855429 3.28254717 2.10446034 2.1954606 ]
 [0.85117012 0.19971925 0.18825441 0.80949536]
 [7.46343165 9.22001627 7.75281829 8.12774412]
 [0.12080405 0.93748036 0.50423905 0.06122282]]
```

[그림11] 모든 채널의 그래디언트 히스토그램 거리 합

0.06122282로 1st 이미지의 패치 3과 2nd 이미지의 패치 3 사이의 거리가 가장 짧아 첫 번째로 매칭된다. 0.18825441로 1st 이미지의 패치 1과 2nd 이미지의 패치 2 사이의 거리가 두 번째로 짧아 매칭되고 2.01855429로 1st 이미지의 패치 0과 2nd 이미지의 패치 0 사이의 거리가 세 번째로 짧아 매칭되며 9.22001627로 1st 이미지의 패치 2와 2nd 이미지의 패치 1이 마지막으로 매칭된다.

1st 이미지의 패치 0과 2nd 이미지의 패치 0, 1st 이미지의 패치 3과 2nd 이미지의 패치 3은 제대로 매칭되었지만 1st 이미지의 패치 1과 2nd 이미지의 패치 2, 1st 이미지의 패치 2과 2nd 이미지의 패치 1은 서로 엇갈려 매칭되었다.

0.06122282로 1st 이미지의 패치 3과 2nd 이미지의 패치 3 사이의 거리가 가장 짧아 첫 번째로 매칭된 후 0.18825441과 0.19971925가 0.01146484가 차이가 나는데 0.18825441이 선택되면서 발생한 문제이다.

만약 0.19971925로 1st 이미지의 패치 1과 2nd 이미지의 패치 1가 두 번째로 매칭되면 동일하게 2.01855429로 1st 이미지의 패치 0과 2nd 이미지의 패치 0 사이의 거리가 세 번째로 짧아 매칭된 후 마지막으로 7.75281829로 1st 이미지의 패치 2와 2nd 이미지의 패치 2가 매칭되어 모든 패치가 제대로 매칭되게 된다.

패치의 거리 합을 구하면 (0.18825441 + 9.22001627) - (0.19971925 + 7.75281829) = 1.45573314로 0.18825441보다 0.19971925를 선택하는 것이 차이가 더 적게 난다. 가장 거리가 짧은 쌍을 선택하는 현재의 방식에서 다음 쌍의 거리와 현재 쌍의 거리의 합도 고려하여 더 짧은 것을 선택한다면 매칭을 더 정확히 할 수 있는 것을 확인하였다.

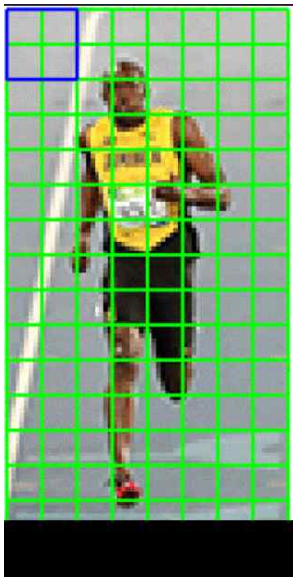
실험 방법2(김현우)

두 번째 실험에서는 그래디언트 히스토그램을 직접 코드로 계산하지 않고 OpenCV 내부함수인 HOGDescriptor를 이용하여 Histogram of Oriented Gredients를 구해낸다.

Histogram of Oriented Gredients (이하 HOG)은 실험 방법1의 Gredients Histogram과 동일하지만 Block 단위의 정규화과정이 포함된다. 이 내부함수를 통해 Patch의 feature에 대한 1차원 벡터를 얻게 되는데 이에 대한 내부 동작 과정 설명이 필요하다.

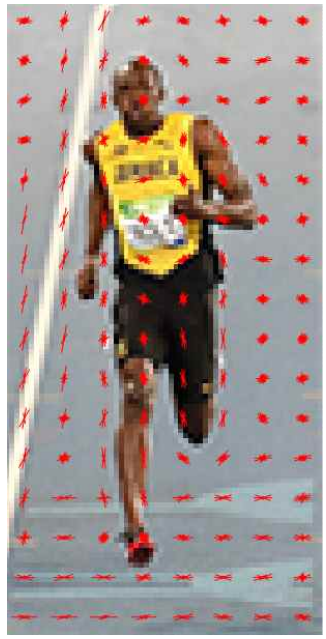
내부 동작 과정 설명에 앞서 이미지 단위에 대한 설명이 선행되어야 한다. 이미지는 가장 작은 Pixel로 이루어져 있으며 Pixel이 모인 Cell로 구성된다. 이러한 Cell이 모여 Block을 형성하게 되는 데 논문

에서는 (2,2)의 Cell이 모인 Block을 추천하고 있다. 이러한 Cell들은 모여 Window를 이루고 이 Window가 우리가 검출하려는 이미지가 되는 것이다. [그림11]에서 예제 이미지는 Window이며 파란 상자는 (2,2) Cell로 이루어진 Block을 나타내며 초록 상자는 (8,8) Pixel로 이루어진 Cell을 나타낸다.



[그림12]

이미지로부터 Gredients Histogram을 구하는 과정은 이전과 동일하다. 근접 픽셀과의 차이로부터 gx , gy를 찾고 이를 이용하여 angle , magnitude를 구한다. unsignedGredient에서 0~180도를 9bins로 나누어 intensity를 표현하면 Gredients Histogram이 생성되게 된다. 이러한 Histogram은 Cell 단위에서 이루어진다. [그림13]에는 예제 그림의 각 Cell에 대한 Gredient가 표현되어 출력되어있다.



[그림13]

이제 실험방법에서 사용된 HOGDescriptor의 특징인 정규화 과정과 1D 거대 벡터 형성과정을 보면 Descriptor 내부에서 정규화는 Block 단위로 이루어진다. 실험에서 사용된 Patch Size는 (9,9)이며 Block Size는 (2,2), Cell Size는 (1,1), Block Stride는 (1,1), Bins number = 9으로 설정하였다.

전체 Window에 대해서 (2,2) Block이 stride 1로 이동한다면 Width = 8 , Height = 8로 움직이게 된다. 즉 8*8 연산을 하게 된다. 그 후 한 Block 정규화 연산 시 [0,1] 범위의 정규화된 (36,1) 행렬이 생성되게 된다. 그 이유는 1 Cell 당 9bins에 대한 Gredient Intensity가 존재하고 1 Block에 4개의 Cell이 존재하므로 (4*9,1) = (36,1) 행렬이 생성되게 된다. 이러한 Block 연산이 (8*8) 일어나게 되므로 HOGDescriptor를 이용한 Patch image의 feature는 (64*36,1) = (2304,1) 이 생성되게 된다.

모든 Patch image의 feature vector를 다시 한 번 histogram을 구해 유사도를 구하는 실험 방식으로 진행였다. 이는 numpy 내부 함수 numpy.histogram을 이용하였다.

8개의 Patch에 feature vector histogram을 얻은 뒤 진행해야하는 것은 histogram간의 유사도를 측정하

는 것이다. OpenCV에서는 두 histogram간의 유사도를 측정하는 compareHist 함수를 제공하고 있다. 이 함수에서 유사도를 측정하는 방식에는 Cross Correlation, Chi-Square, Intersection, Bhattacharyya distance 총 4가지가 존재한다. [그림14]에서는 Bhattacharyya distance의 공식을 확인할 수 있다.

Bhattacharyya distance

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

[그림14]

이번 실험방법에서는 Bhattacharyya distance를 이용하여 유사도를 측정해보았다. Bhattacharyya distance 는 histogram을 전체 크기로 나눠 1로 만든 확률분포를 이용하여 유사도를 측정한다. 이 Bhattacharyya distance는 [0,1] 범위의 값을 가지게 되는 데 0에 가까울수록 두 histogram이 유사하고 1에 가까울수록 두 histogram이 불일치하다고 판단할 수 있다.

Fig1의 4개의 Patch [0,1,2,3] 과 Fig2의 4개의 Patch [0,1,2,3]을 행,열로 설정한 뒤 두 Patch feature vector histogram의 유사도를 Bhattacharyya distance 방식으로 구하여 (4,4)를 구하게 된다.

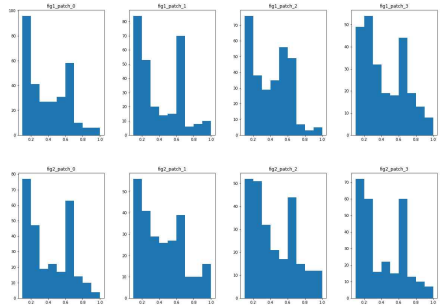
유사도를 구한 뒤의 진행은 유사도가 높은 Patch끼리 일대일 대응시키는 것이다. 앞서 구한 (4,4) 유사도를 Patch 사이의 거리로 생각한 뒤 최근접 이웃 문제로 접근하여 matching 시켰다.

처음 (4,4) 유사도 행렬에 대해 최솟값을 구하게 되면 모든 Patch사이의 거리 중 가장 가까운 2개의 Patch 쌍이 결정되게 된다. 그 뒤 해당 Patch들의 행,열을 제거하여 (3,3) 유사도 행렬을 만들게 된다. 마찬가지로 최솟값을 구하게 되면 남은 Patch들 중 가장 가까운 2개의 쌍이 결정된다. 이러한 방식을 반복하여 (1,1) 행렬이 남게될 때까지 진행하면 모든 근접 Patch 쌍이 일대일로 대응되게 된다.

실험 결과2(김현우)

OpenCV의 내부함수 HOGDescriptor를 이용해 Patch feature를 구하고 이 feature를 다시 histogram으로 구하게 되면 대부분의 gredient가 0으로 출력되는 것을 확인할 수 있었다. 총 2304개의 feature값들 중 2000개 이상의 값들이 0인 것을 확인할 수 있었다. 이 histogram을 그대로 사용되게 되면 문제가 발생할 수 있다. 다른 bins에서 유의미한 intensity차이가 있다고해도 0값이 다른 bins들에 비해 매우 많으므로 intensty가 매우 크다. 따라서 다른 bins들의 유의미한 차이가 작게 반영이 되게 되는 문제가 발생할 수 있다. 따라서 0값을 제외한 histogram을 이용하는 게 좋다는 결론을 내리게 되었다.

[그림15]를 보게 되면 8개의 Patches에 대한 feature histogram을 확인할 수 있다. 0값을 제외한 histogram을 시각화해보면 다른 intensity 차이가 유의미함을 알 수 있었다.



[그림15]

다음으로 compareHist 함수에서 Bhattacharyya distance method를 사용하여 Patch들간의 유사도 행렬을 구한 결과를 확인할 수 있었다. [그림16]에서 모든 행렬값은 [0,1] 범위의 값을 가지는 것을 확인할 수 있다. 또한 0에 가까울수록 두 Patch가 유사하다고 판단할 수 있었다.

	0	1	2	3
0	0.448312	0.402323	0.436077	0.414315
1	0.409033	0.432267	0.412991	0.397026
2	0.459038	0.40588	0.429242	0.447851
3	0.456335	0.491189	0.46808	0.394772

[그림16]

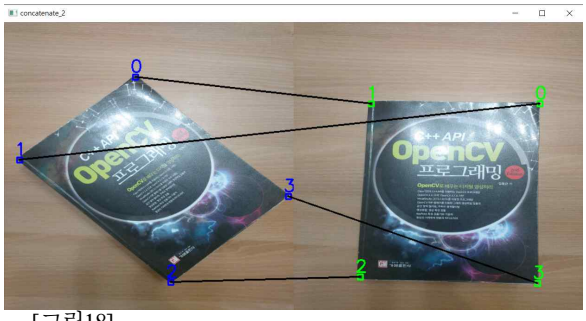
다음은 유사도 행렬을 이용하여 일대일 매칭을 시킨 결과를 확인할 수 있었다. [그림17]에서 각 행은 유사도가 높은 Patch 쌍을 나타내주고 있다.

예를들어 Fig1의 Patch1 과 Fig2의 Patch2 가 유사하다고 판단되었다.

	0	1
0	3	3
1	0	1
2	1	0
3	2	2

[그림17]

이러한 결과를 [그림18]에서 시각화하여 매칭되는 Patch끼리 Line으로 연결하여 확인할 수 있었다.



[그림18]

실험 결과를 통해 확인한 Patch 매칭의 결과가 부정확했는데 오차 원인을 다음과 같이 고찰해 볼 수 있었다.

1. 마우스를 이용해 Patch의 Center를 선택하게 되는데 Fig1과 Fig2에서 Center가 잘못 선택되어 오차가 발생하였다.
2. Patch로 사용된 image의 크기가 너무 작아 feature를 충분히 찾아낼 수 없었다.
3. 원본 이미지는 (3024,4032)로 매우 큰 해상도를 가지는데 이를 (500,500)으로 resize하는 과정에서 데이터가 손실이 일어나고 scale또한 맞추지 않았기 때문에 오차가 발생했다.

부 록

GitHubURL:https://github.com/jjjuurang/2022-1-ComputerVisionProject.git

조원 역할 분담:

- 이주영: 영상 입력 받고 마우스 클릭으로 patch(9X9) 저장, 세 채널의 그래디언트 히스토그램 차이를 계산, 패치 시각화
- 김현우: HOGDescriptor를 이용하여 Block 단위의 정규화과정이 포함된 Histogram of Oriented Gredients 계산, 각 점별로 가장 유사한 결과 시각화