에지의 개념

- 한쪽 방향으로 픽셀값이 급격하게 바뀌는 부분
- 객체 판별을 위한 전처리로 에지 검출을 사용

미분

- 에지를 찾기 위한 방법이 픽셀값을 *미분* 해서 변화율이 큰 픽셀을 찾는것
- 미분하기 위해서는 영상이 2차원 평면위의 함수, 정수 단위 좌표에 나열되어 있는 이산함수라는 점을 기억할것
- 보통 중앙차분으로 미분을 근사하는 방식이 오류가 적다.

그래디언트

- 2차원 공간이기에 x,y축 방향으로 각각 미분하고 이를 백터로 표현한것이 그래디언트.
- 그래디언트 백터의 방향은 변화가 가장 큰 방향, 크기는 변화율의 세기
- maginitude()함수에 x,y각각 미분한 값을 넣어주면 그래디언트의 크기를 구할 수 있다.
- phase()함수에 x,y각각 미분한 값을 넣어주면 그래디언트의 방향을 구할 수 있음

에지를 찾는 기본 방법

- 그래디언트 크기가 특정값(임계값)보다 큰 위치를 찾는것

마스크기반 에지검출

- 가장 간단한 예제. 내 입맛대로 np.array()로 x축,y축 마스크를 만들고. cv.filter2D에 인자로 넣어서 마스크연산 수행

소벨필터

- 소벨마스크로 영상을 미분한다.

샤르필터

- 샤르마스크로 영상을 미분한다. 소벨보다 더 정확함

마스크 예제

1. 각각 x,y축 미분해준다.
2. 그래디언트 크기를 구함

```
dx = cv.Sobel(src,)
dy = cv.Sobel(src,)

fmag = cv.magnitude(dx,dy)
# fmag는 실수이기에 정수로 바꿔야 보여줄 수 있음
mag = np.uint8(np.clip(fmag,0,255))
# 정수로 바꾸는 과정

_, edge = cv.threshold(mag, 150, 255)
# 추출한 에지를 깔끔하게 이진화 하는 과정
```

캐니 에지 검출기

- 그래디언트의 크기와 방향을 모두 고려. 좀더 정확하게 에지 검출가능. 에지는 서로 연결될 가능성이 높다는 점을 고려
- 가우시안필터링(잡음제거), 그래디언트계산, 비최대억제(가장 큰 값만 선택), 이중임계값(2개로 판단)
- dst = cv.Canny(src, 50, 150) 으로 간단하게 사용가능

허프변환 직선 검출

- 직선을 찾기 위한 방법. 직선이 많이 교차되는 좌표를 모두 찾고, 교차가 많이 되는곳을 그린다.
- 먼저 에지를 찾고, 에지를 Hough_lines()에 인자로 넣어줘야 한다.

```
edge = cv.Canny()
linse = cv.HoughLines(edge)
```

확률적 허프 변환

- 확률적으로 직선검출하는 방법. 시작점, 끝점을 반환함.

```
edge = cv.Canny()
linse = cv.HoughLinesP(edge)
```

허프변환 원검출

- 원의 중심을 찾고, 적합한 반지름을 구해서 원을 그린다.

```
src = cv.imread()
blurred = cv.blur(src)
circles = cv.HoughCircles(blurred, cv.HOUGH_GRADIENT, 1, 50, param1= ,
param2=)
```

컬러영상

- OPENCV에서는 BGR순서로 컬러 저장. imread()할때 IMREAD_COLOR로 설정해야 함. 1개 픽셀이 3가지 성분을 0~255범위 값을 가짐.

cvtColor()

- 다양하게 컬러영상으로 변환 가능
- BGR2GRAY : 컬러를 그레이스케일로. 성능에 좋음
- GRAY2BGR : 그레이스케일을 컬러로. 비율은 똑같이 계산

- BGR2HSV : 색상, 채도 명도로 색을 표현
- BGR2YCrCb : Y는 밝기, 휘도, Cr,Cb는 색상

색상채널 나누기

```
src = cv.imread(path, cv.IMREAD_COLOR)
bgr = src.split()
bgr[0],bgr[1],bgr[2] # 각각 B,G,R
```

컬러영상 히스토그램

- 컬러영상을 히스토그램 평활화 할때 각 색 성분이 연계되어 있어서 독립적으로 수행후 합치면 문제가 생김.
- YCrCb에서 Y성분만 히스토그램 평활화를 하고 합쳐야 함.

```
src = cv.imread() # 읽기
ycrcb = cv.cvtColor(src, cv.COLOR_BGR2YCRCB) # ycrcb로 변환
ycrcb = cv.split() # Y, Cr, Cb로 분리
ycrcb[0] = cv.equalizeHist(ycrcb[0]) # Y를 히스토그램 평활화
dst = cv.merge(ycrcb) # 다시 합치기
```

색상범위지정 분할

- inrage() 함수로 특정 범위안에 있는지 확인 가능
- inrage(src, low, high) -> 있으면 true 리턴

히스토그램 역투영

- HSV로 색을 골라내는것으로 한계가 있음
- 미리 기준에서 히스토그램영상을 구해두고, 해당 컬러히스토그램으로 기준에서 영역을 찾는 방법

```
ref = imread()
mask = imread()
ref_ycrcb = imread()
chanels[1,2]
cr_bins = 128
cb_bins = 128

histSize = [cr_bins, cb_bins]
cr_range = [0,256]
cb_range = [0,256]
ranges = cr_range + cb_range
hist = cv.calcHist([ref_ycrcb], chanel, mask, histSize, ranges)
# 기존 영상에서 마스크 부분에 해당하는 컬러 히스토그램 구하기
src = imread()
srcycrcb = cvtColor(src, cv.IMREAD_BGR2YCRCB)
```

```
backproj = cv.calcBackProject([srcycrcb], chanel, hist, ranges, 1)
# 역투영하기
```

이진화

- 영상의 각 픽셀을 두개로 분류. 객체-배경, 관심-비관심. 기준이 되는 값이 임계값

threshold()

- 임계값을 통해 다양한 연산 지원

```
cv.threshold(src, low, high, cv.THERSH_BINARY)
```

적응형 이진화

- 영역별로 이진화를 한다. 그림자진 지역은 임계값을 다르게 해야 이진화가 잘 되기 때문이다.
- adaptiveThreshold()

```
dst = adaptiveThreshold(src, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, blocksize, 5)
```

모폴로지

- 객체를 단순화 시키고, 잡음제거하는 용도로 사용.
- 구조요소 : 마스크라 보면 됨. 해당 구조요소를 통해 모폴로지를 구현. getStructuringElement() 함수로 만듬

침식

- 객체영역의 외곽을 골고루 깎아냄. 구조요소에 완전히 포함되야 고정점을 채운다.
- erode()

팽창

- 객채 외곽을 확대. 구조요소에 하나라도 포함되면 고정점을 채움
- dilate()

```
_,src_bin = cv.threshold(src, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
dst1 = cv.erode(src_bin)
dst2 = cv.dilate(src_bin)
```

열기와 닫기

- 열기 : 침식 -> 팽창, 작은크기의 객체 효과적 제거. 객체 영역을 확실히 구분

- 닫기 : 팽창 -> 침식, 내부의 작은 구멍을 매꿈. 객체간 이어짐을 확인하고 싶을때
- morphologyEx()함수에 인자로 가능

```
_,src_bin = cv.threshold(src, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
dst1 = cv.morphologyEx(src_bin, cv.MORPH_OPEN, None)
dst2 = cv.morphologyEx(src_bin, cv.MORPH_CLOSE, None)
```

레이블링

- 객체,배경을 구분했으면 다시 각각의 객체를 구분해야함. 이게 레이블링!
- 4방향 : 4방향끼리 연결되어 있으면, 8방향 : 8방향이 연결되어 있어야

connectedComponents()

```
_,src_bin = cv.threshold(src, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
cnt, label,stats,centroids = cv.connectedComponents(src_bin) # src는 2진화가 수행된 영상
```

connectedComponentsWithStats()

- 추가적인 정보를 제공

```
cnt, label = cv.connectedComponentsWithStats(src) # src는 2진화가 수행된 영상
```

- cnt는 개수, label은 각 객체 영역이 담김, stats (x,y,w,h,area), centroids(center_x, center_y)

외곽선검출

- 객체의 외곽선은 객체 영역 픽셀중 배경과 인접한 일련의 픽셀

findContours()

- 객체의 외곽선을 검출하는 함수
- 검출모드에 따라 제공하는 기능이 다름.
- RETR_EXTERNAL, RETR_LIST, RETR_CCOMP, RETR_TREE

```
contours , _ cv.findContours(src, cv.RETR_LIST, cv.CHAIN_APPROX_NONE)
# 2번째 인자는 계층구조를 받는 자리임
for i in range(len(contours)):
    cv.drawContours(dst, contors)
```

외곽선 처리함수

- boundingRect(), minAreaRect, minEnclosingCircle(), arcLength(), contourArea()

외곽선 근사화

- approxPolyDP()함수로 근사화 가능
- 가장 멀리 떨어진 두 점을 찾고, 잇는다. 해당 선에서 가장 먼 점을 찾고 입실론 인자보다 크다면 다시 잇고 작으면 끝.

```
_,src_bin = cv.threshold(src, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
contours , _ cv.findContours(src, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

for pts in contours:
    if cv.contours(pts) < 400:
    approx = cv.approxPolyDP(pts, cv.arcLengh(pts,True) * 0.02)
    # 근사화 하고 점의 개수를 가져온다.
    vtc = len(approx) #점이 몇개 인지 판별 가능
```

템플릿 매칭

- 찾고자 하는 부분영상이 있을때 입력 영상에서 해당 부분을 찾는다.

```
res = cv.matchTemplate(img, temp1, cv.TM_CCOEFF_NORMED)
# 매칭을 수행하고
res_norm = cv.matchTemplate(res, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
# 0~255사이 값으로 정규화를 해준다.
```

유사 하르 필터

- 흑백 사각형이 서로 붙어있는 형태로 구성된 필터. 흰색은 더하고 검정은 빼줌

비올라 존스 알고리즘

- 유사하르 필터에서 얼굴검출에 효과적인것을 선별
- 단계별로 필터의 개수를 늘려가며 판별한다.

```
classfier = cv.CascadeClassifier('xml')
faces = classifier.detectMultiScale(img)
# faces에는 x,y,w,h값이 담긴다.
```

HOG알고리즘 보행자 검출

- 그래디언트의 방향 히스토그램을 통해 보행자를 검출. 방향에 대한 값을 구하고, 몇도인지 히스토그램을 구함.

```
hog = cv.HOGDescriptor()
hog.setSVMDetector(cv.HOGDescriptor_getDefaultPeopleDetector())
detected, _ = hog.detectMultiScale(frame)
```

QR코드

- 코드내의 정사각형 3개를 찾아 투시변환

```
decetor = cv.QRCodeDector()
info, points, _ = detector.detectAndDecode(frame)
```

특징, 코너, 특징점

- 특징 : 영상으로 부터 추출할 수 있는 유용한 정보. 코너 : 에지가 급격히 바뀌는 부분. 특징점 : 한점의 형태로 표현할 수 있는 특징

해리스 코너 검출

- 특정위치에서 주변픽셀의 합을 구해서 중심과의 차이가 얼마나 큰지 확인 -> 크면 코너가 된다.
- cornerHarris()함수를 사용. 리턴값이 임계값보다 크다면 코너로 판별 가능. 그러나 중복으로 비최대억제 수행이 요구됨

FAST 코너 검출

- 빠르게 픽셀 값만 비교해서 코너 검출. 9개를 기준으로 삼는다.
- FAST(src,dst,임계값,비최대억제수행여부 = True)

크기불변 특징점 검출

- 영상의 크기가 변경되면 코너는 더이상 검출되지 않을 가능성이 높다.

SIFT 크기불변특징점알고리즘

- 옥타브내에서 DOG를 구하며 특징점을 찾는다.

ORB

- FAST를 이용하는데 영상 크기 변화에 강하게 대응할 수 있게 추가함.
- BRIEF로 기술자를 생성함.
- 해밍거리 : 두 기술자 사이에 값이 다른 비트의 개수를 세는 방법

keyPoint class

- create() : 기술자 만들기 -> detect() : 키포인트 검출 -> compute() : 키포인트로 기술자 생성.
- FAST는 compute 사용 X, BRIEF는 detect 사용 X

특징점 매칭

- 두 영상에서 추출한 특징점 기술자를 비교해 비슷한 특징점을 찾는것.

- DescriptorMatcher 클래스가 있음. match()- 비슷한 기술자 하나를 찾음. knnMatch() - 비슷한 기술자 쌍 k개를 찾음. radiusMatch() - 지정 거리 안에 있는 기술자 모두를 찾음
- BFMatcher 클래스 : 가장 거리가 작은 기술자를 하나 찾는다.
- FlannBasedMathcer : 근사화된 최근방 이웃 알고리즘

```
orb = cv.ORB_create()
key1, des1 = orb.detectAndCompute(img1,None)
key2, des2 = orb.detectAndCompute(img2,None)
# 각각 특징점을 찾고

matcher = cv.BFMatcher_create(cv.NORM_HAMMING) # 기술자를 만들고
mathes = matcher.match(desc1, desc2) # 매칭시켜봄
```

findHomograpy()

- 특징점을 찾고, 서로 이동정보를 매칭하는 방식. 투시변환과 같은 연산.

영상 이어 붙이기

- 서로 일정 비율 이상으로 겹치는 영역이 있어야 함. 서로 같은 위치를 분간하도록 유효한 특징점이 있어야 함.

```
sticher = cv.Sticher_create() # 붙이기 기술자 부르고
status, dst = sticher.stitch(imgs) # 기술자가 붙여줌
```

머신러닝

- 머신러닝에 영상을 입력으로 줄때 대부분 특징정보를 추출하여 전달.

지도학습 & 비지도학습

- 지도학습 : 정답을 알고 있음. 회귀 : 연속된 수치 예측. 분류 : 이산값을 결과로 출력
- 비지도 학습 : 정답없이 데이터 차제만을 이용해서 분류
- 오버피팅 : 전체를 학습하면 치우쳐서 학습하게 되어 새로운 입력에 대한 예측을 잘 못함

K폴드 교차검증

- 훈련데이터를 k개로 나누어 분할하여 학습과 검증을 반복
- 훈련데이터의 잡음, 이상치를 고려해야 한다.

OpenCV 머신러닝 클래스

- ml모듈에 구현되어있음
- train() : 훈련데이터가 어떻게 저장되어 있는지 명시 , predict() : 순수 가상함수로 상속받은 클래스가 재정의

최근접이웃

- 분류나 회귀에 사용되는 지도학습알고리즘
- 인접(NN) : 새로운 입력에 대해서 가장 가까이에 있는 데이터와 같은 클래스로 지정
- K개 인접(KNN) : 새로운 입력에 대해서 인접지역에 가장 많은 데이터의 클래스로 지정

KNearest

- knn에서 k를 1로 잡으면 nn이랑 같다.
- knearst객체를 생성하고, 속성을 설정한후, train()함수로 학습. 예측수행시 predict()보다 자체 제공인 findNearset()가 더 유용하기에 사용.

```
knn = cv.ml.KNearest_create()
train_array = np.array()
label_array = np.array()

knn.train(train_array, cv.ml.ROW_SAMPLE, label_array) # 학습시키기

# 이후 각 픽셀에 대해서 정보를 가져 올 수 있다.
for j in range(img.shape[0]):
    for i in range(img.shape[1]):
        sample = np.array([[i,j]])

        ret,res,_,_ = knn.findNearset(sample, k_values) # 예측결과 가져오기
```

knn을 통한 필기체 숫자 인식

```


cells = [np.hsplit(row, w//20) for row in np.vsplit(digits, h//20)]
# 20 * 20으로 쪽ㅐ기

train_img = cells.reshape(-1,400)
# 1 * 400늘리기
train_table = np.repeat(np.arrage(10)m len(train_img) / 10)

knn = cv.ml.KNearest_create()
knn.train(train_img, cv.ml.ROW_SAMPLE, train_table)

ret, res, _, _ = knn.findNearest(입력, 3)
print(int(res[0,0]))
```