20185124 나영훈 영상처리 프로그래밍 기말프로젝트 보고서

주제 : 영상처리를 이용한 비트코인 자동 투자 도우미 앱.

프로젝트 개요 - API를 사용하여 가격을 불러올 경우 다양한 투자지표를 사용하기 위해 직접 구현해야 하며 시간이 많이 든다. 영상처리 프로그래밍 과목에서 수강한 내용을 바탕으로 여러가지 투자 지표를 코딩을 통해 구현하는 것이 아닌 그래프에 나타나는 정보를 영상처리를 통해 효과적으로 해결할 수 있다고 생각하였다. 따라서 그래프 모양을 보고 투자를 도와줄 수 있는 투자 도우미 프로그램을 만들어 보았다. 빗썸에서는 다양한 영상처리 기법을 사용하여 처리를 하였으며, 업비트에서는 간단한 영상처리만 사용하여 둘의 차이를 비교해 보았다.

사용한 대표 기술 요약

- 대비 증가 후 역방향 사상을 통한 알파벳 검출
- 격자 제거 후 엣지 검출을 위한 블러링 및 Canny Edge 검출 사용
- Canny Edge 검출 후 모폴로지 팽창 연산 사용
- 간단한 매수 알고리즘을 통한 자동 매수 기능

사용 방법

예측하고자 하는 거래소를 킨 후 Main.py를 실행시킨다. 이 때 내가 예측하고자 하는 사이트, 발급받은 API 키, 모드를 입력한다.

--stock_exchange : 거래소를 지정해주며 upbit와 bihumb 두 개가 있으며 default는 upbit이다.

--Access_key : 해당 거래소에서 발급받은 API키 중 Access key를 넣어준다.

--Secret_key : 해당 거래소에서 발급받은 API키 중 Secret Key를 넣어준다.

위의 키를 넣어주지 않는다면 거래 창에서 거래가 진행되지 않는다.

--Mode : bright Mode와 dark 모드가 있으며 업비트에서는 dark모드가 지원된다.

--monitorsize : 모니터의 크기를 넣어주면 해당 모니터의 비율에 따라 출력 이미지를 자동으로 지정한다. Default는 (1920,1080)이다.

해당 그래프의 예측 결과가 상승이면 매수를 진행하며, 하락이면 그대로 프로그램을 종료한다.

핵심 코드 설명

1. 역방향 사상 및 필터 비교

```
while col_num < image.shape[1]:
    col_min, col_max = min_li(image, col_num, image.shape[1])

    if col_max == -1 and col_max == -1: break
    al = (255 - image[:, col_min + col_num:col_max + col_num]) / 255

    resize = np.zeros((10, 6), 'float32')
    ratioX = al.shape[0] / resize.shape[0]
    ratioY = al.shape[1] / resize.shape[1]

    for row in range(resize.shape[0]):
        for col in range(resize.shape[1]):
            temp_x = int(row * ratioX)
            temp_y = int(col * ratioY)
            if row * ratioX >= 10:
                continue
            if temp_y >= 5:
                temp_y = al.shape[1] - 1

            resize[row, col] = al[temp_x, temp_y]

    for alpha, beta in alphabets:
        exit = False
        if np.array_equal(alpha, resize):
            # print("=========================", beta)
            if beta == "/":
                exit = True
                break

            elif beta == "restart":
                image = sub_img
                use_sub = True
                col_num = 0
                col_max = 0

            else:
                event_name += beta
                break
```

제목을 불러온 후 공백 단위로 잘라 while 문에 넣은 다음 (10,6) 사이즈로 역방향 사상을 통해 줄인다. 그 다음 0과 1로 만들어서 np.array_equal()을 통해 미리 저장해둔 알파벳 필터와 비교하여 같은 글자를 찾아낸다.

10,6으로 줄이는 과정에서 C와 O가 동일하게 압축되어 인식에 오류가 생겼다. 따라서 col*ratioY가 인덱스의 마지막인 5일 경우 강제로 알파벳의 마지막 column을 마지막 인덱스에 집어넣었다.

글자마다 기본적으로 들어가는 부분만 비교하여 해당 부분이 모두 True일 경우 해당 글자를 반환하는 방법을 사용하여 알파벳 필터 수를 줄이고자 하으나 반복문 하나를 더 만든 후 시간 복잡도가 너무 커져 기다리는 시간이 생기게 되었습니다. 따라서 반복문을 사용하지 않고 알파벳 필터를 늘렸습니다.

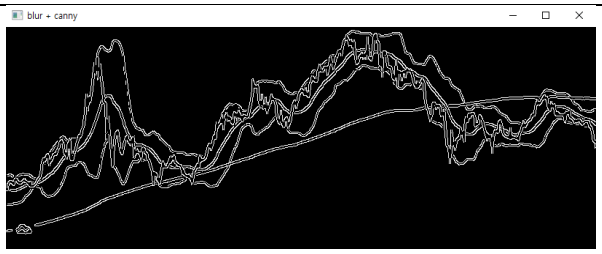
2. 다양한 영상처리 기법을 사용해 격자 및 배경색 제거하기

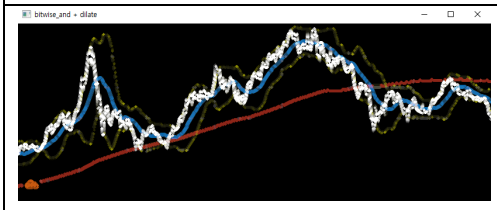
```
def bithumb_coin_pred_algorithm(image, Mode):
    image = 255 - image

    # Blur + Canny Edge를 통한 잡음 제거 후 엣지 검출
    Big_img = cv2.GaussianBlur(image, ksize=(3, 3), sigmaY=0, sigmaX=0)
    Big_img = cv2.Canny(Big_img, 220, 255)

    image = cv2.bitwise_and(image, image, mask=Big_img)     # bitwise_and 해서 다시 칼라 입히기

    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # 모폴로지 팽창 연산을 통해 선 굵게 만들기
    mask = np.array([0, 0, 1, 0, 0,
                     0, 1, 1, 1, 0,
                     1, 1, 1, 1, 1,
                     0, 1, 1, 1, 0,
                     0, 0, 1, 0, 0]).astype('uint8')
    mask = mask.reshape(5, 5).astype('uint8')
    dst = cv2.dilate(image, mask)

    # 대비 증가 시킴
    dst = np.array(dst, 'float32')
    dst = (dst - 50) * 255
    dst = (dst - 51) * 255
    dst = np.clip(dst, 0, 255)
    image = np.array(dst, 'uint8')

    zero_img = np.zeros(image.shape[:2]).astype('uint8')
    # print(image)
    white_mask = cv2.inRange(image, (250, 250, 250), (255, 255, 255))
    green_mask = cv2.inRange(image, (30, 30, 0), (255, 255, 20))
    yellow_mask = cv2.inRange(image, (0, 150, 150), (10, 255, 255))
    # black_mask = cv2.inRange(image, (0, 0,0), (30, 30, 30))
    red_mask = cv2.inRange(image, (0, 0, 120), (110, 110, 255))
    img_mask = cv2.merge([white_mask, green_mask, red_mask])
    bolinger_mask = cv2.merge([zero_img, yellow_mask, yellow_mask])
```

위의 코드는 다양한 영상처리 기법을 활용하여 격자 및 배경색을 제거하는 부분입니다.

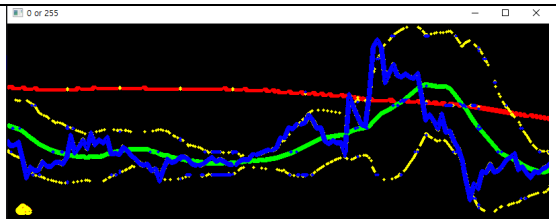
배경을 검정색으로 만든 후 처리하는 방법이 더욱 깔끔하여 255를 뺀 후 진행하였습니다.

Gaussian Blur를 사용하여 이미지를 흐릿하게 만든 후 Canny Edge 검출 방법을 이용해 선을 검출합니다. 그 다음 Gaussian Blur와 Canny Edge 검출 방법을 이용해 나온 그림을 마스크로 사용하여 bitwise_and() 연산을 이용해 컬러를 입힙니다. 이때 선들 사이에 구멍이 뚫려있는 듯한 그림이 나오게 되어 모폴로지 팽창 연산을 통해 선을 이어주었습니다. 그 다음 밝기를 0 혹은 255로 만들어 색상 검출을 쉽게 만들었습니다. 그 다음 각각의 색상에 해당하는 값을 opencv의 inRange함수를 사용하여 해당하는 부분으로 바꿔주었습니다

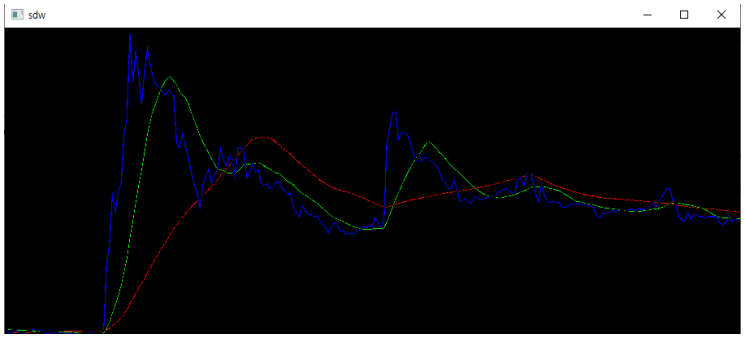




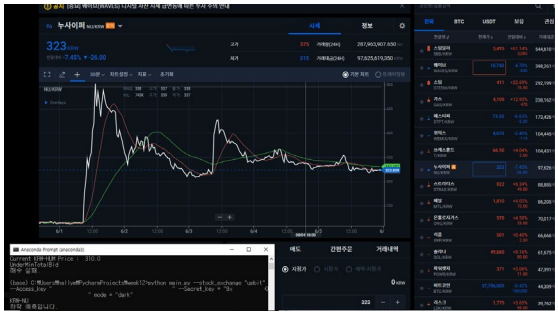
배경, 격자가 사라진 것을 볼 수 있음



bithumb에서 진행한 이미지
변환과정을 거치며 이미지가 흐릿해졌습니다.

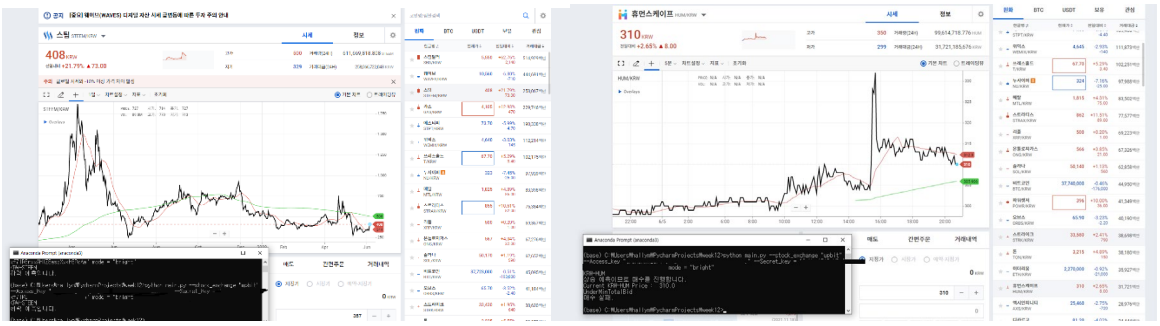


업비트에서 진행한 이미지
구현 사진



다크모드에서도 예측 가능.

구현 사진



다음과 같이 아나콘다 프롬프트 창을 띄워 main을 실행킬 수 있으며, 알고리즘에 맞는 경우 매수를 진행합니다. 위의 창에서는 UnderMinTotalBid Error가 발생하였는데 현재 재 계좌에 돈을 넣어두지 않았으므로 에러가 발생하며 매수가 진행되지는 않았습니다. 돈을 넣게 된다면 매수가 바로 진행될 것입니다.

결론.

- 프로그램은 인간과 다르게 감정을 가지고 있지 않다. 정해진 명령대로 처리를 하여 좀 더 정확하고 냉철한 판단을 내릴 수 있다. 내부에서 격자를 제거하거나 볼린저 밴드 내의 배경색을 거래소 자체에서 없애준다면 간단한 영상처리 기법을 적용만으로도 예측이 가능하다.
- 본 프로그램에서는 영상처리를 하기 쉽게 색을 바꾸기, 선의 두께를 늘리기, 격자나 배경색을 제거하는 등의 이미지 처리를 거래소가 아닌 프로그램에서 진행함에 따라 사용자가 기존의 사용하던 투자 방식을 이어서 할 수 있다는 장점이 있다. 알파벳을 따오는 과정에서 시간 소요가 많이 되었지만 그만큼 다양한 상황에서도 적용할 수 있을 것이다.
- 빗썸에서 잡음 제거 과정에서 모폴로지 팽창 연산이후 색을 지정해주는 과정에서 색이 섞이게 된다는 단점이 생긴다. 이는 다양한 알고리즘을 짜는데 제약이 되며, 심할 경우 오탐지를 할 가능성도 있다. 위의 코드 상세 설명 그림에서 이런 부분을 볼 수 있다. 가장 중요한 현재 가격을 나타내는 파란색이 현재 가격 부분에서 잘 나오고 있으며, 중간이 뚫려있더라도 알고리즘 구현이 가능하기에 약간의 제약이 있지만 투자는 가능할 것으로 보인다.
- 업비트에서는 간단한 영상처리 만으로도 선을 쉽게 가져올 수 있다. 하지만 다양한 투자 지표가 추가될 경우 사용하기 힘들어 질 것이다.
- 이 프로그램을 통해 사용자는 기존에 사용하던 거래소의 투자 그래프를 변경하지 않고 투자하는 것이 가능하다는 것을 영상처리를 통해 보여주고 싶었다. 보완해야할 점이 있지만 그 점을 감안하고도 프로그램을 사용할 수 있다고 생각한다.