Method

1. Rotate

1. 建立一張全黑圖(放旋轉後的結果)，以及**順時針**旋轉 30 度的旋轉矩陣
2. 遍歷全黑圖的所有點，找到他對中心順時針旋轉 30 度後，對應到原圖的座標位置(src_x, src_y)
3. 看位置是否超出界線，如果無就要對那個點做 interpolation，因為旋轉完座標不是整數。

```
for i in range(h):
    for j in range(w):
        vec_i, vec_j = i-center_x, j-center_y
        src_x, src_y = rotate_mat @ np.array([vec_i, vec_j])
        src_x = center_x + src_x
        src_y = center_y + src_y
        if in_bound(src_x, src_y, h, w):
            emp_img[i, j, :] = bicubic_interpolation(img, src_x, src_y) if interpolation == True else img[int(src_x), int(src_y), :]
```

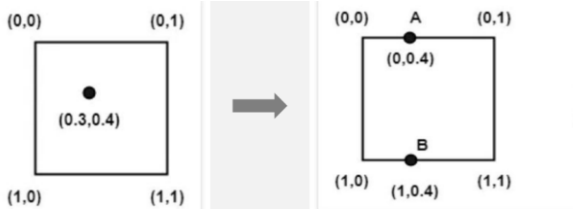
Interpolation:

Way 1(Nearest Neighbor Interpolation):

直接無條件捨去(src_x, src_y)找到小於他們最靠近的整數點

```
def NN_interpolation(img, src_x, src_y):
    x = int(src_x)
    y = int(src_y)
    return img[x, y, :]
```

Way 2(Bilinear Interpolation):




```
def Bilinear_interpolation(img, src_x, src_y):
    x = int(src_x)
    y = int(src_y)
    h, w, c = img.shape

    dx = src_x - x
    dy = src_y - y

    if x == h-1 or y == w-1:
        return img[x, y]

    tmp1 = img[x, y] * (1-dy) + img[x, y+1] * (dy)
    tmp2 = img[x+1, y] * (1-dy) + img[x+1, y+1] * (dy)

    return (tmp1 * (1-dx) + tmp2 * dx)
```

先用 y 到附近左右兩點的距離比例，算出圖中 AB 兩點的值，存在 tmp1 與 tmp2 裡

再用這兩個值用 x 到兩點的距離比例，得到(src_x, src_y)真正的值

Way 3(Bicubic interpolation)

Bicubic interpolation 的方法就是先找周圍最近的 16 個點，對每個水平的四個點算出大約的三次函數曲線，在算出該水平對應點的值，最後拿算出來的四個點再找一個三次函數，算出最後結果，code 有點長就不貼了。算三次曲線的方法，是用以下推出來的公式，最後 clip 在(0~255)

```
def function_val(p0, p1, p2, p3, x):
    return np.clip((-0.5*p0 + 1.5*p1 - 1.5*p2 + 0.5*p3)*(x**3) + (p0 - 2.5*p1 + 2*p2 - 0.5*p3)*(x**2) + (-0.5*p0 + 0.5*p2)*(x)+ p1, 0, 255)
```

2. Image wrapping and homography

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{array}{c} \\ \text{Point 1} \\ \\ \text{Point 2} \\ \\ \text{Point 3} \\ \\ \text{Point 4} \\ \\ \end{array} \overset{\text{2N x 8}}{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}} \overset{\text{8 x 1}}{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}} = \overset{\text{2N x 1}}{\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}}$$

找到四組對應的點，要先找出左圖中的 H 矩陣，找到電視上的座標(x , y)對應原圖(x', y')的 H 矩陣。

用右圖的大矩陣，找出四組點，並使用 NumPy 裡的 linear Algebra function 解出 H(h11~h32)

最後遍歷電視裡的點，找到對應原圖的 src_x 與 src_y，再用前面講的一模一樣的三種 interploation 得出那個 pixel 的值。

2. Result

由左而右分別為 Nearest neighbor、bilinear 與 bicubic interpolation

由左而右分別是 Nearest neighbor、bilinear 與 bicubic interpolation

Feedback

因為空間不夠，把結果圖片縮小，所以可能看不太出來，但是看原圖可以明顯看出差異。

使用 nearest neighbor interpolation 會明顯有馬賽克的感覺，因為我們插值的方法就單純是取靠近的 pixel 的 value 值，每次插值只考慮附近一個點，所以旋轉後的圖可能會有幾個 pixel 對應到原圖同樣的地方，造成資訊變少、畫質變差，有馬賽克的感覺。

使用 bilinear interpolation 有明顯變好的感覺，因為考慮了附近四個 pixels 的值，相較 nearest neighbor interpolation 畫質有變好許多。

使用 bicubic interpolation 又比 bilinear interpolation 更好了，許多物品邊界處理有更好的效果，變得更加的銳利，還有後面山脈的細節處理更好，肉眼看畫質明顯更好許多，因為這個方法足足考慮了周圍 16 個點，用了 5 個三次函數後才得到每個 pixel 應該賦予的值，運算複雜度與執行時間也需要更長。