1.

```python
def SAD(ref, target):
    return np.sum(np.abs(ref - target))
```

I compute SAD value by summing absolute value of all elements and all channels between reference block and target block.

```python
def full_search(reference, target, block_size, search_range):
    h, w, c = reference.shape
    predicted_img = np.zeros_like(target)
    motion_vectors = np.zeros((h//block_size, w//block_size, 2), dtype=int)
    residuals_img = np.zeros_like(target)

    for row in range(0, h, block_size):
        for col in range(0, w, block_size):
            center = (row, col)
            block = reference[row:row+block_size, col:col+block_size, :]
            mv, sad = full_search_helper(block, center, target, search_range)
            motion_vectors[row//block_size, col//block_size] = mv
            predicted_img[row:row+block_size,col:col+block_size, :] = target[row+mv[0]:row+mv[0]+block_size,col+mv[1]:col+mv[1]+block_size, :]
    residuals_img = cv2.absdiff(predicted_img, target)

    return predicted_img, motion_vectors, residuals_img
```

```python
def full_search_helper(block, center, target, search_range):
    x, y = center
    best_mv = [0, 0]
    min_sad = SAD(block, target[x:x+block.shape[0], y:y+block.shape[1], :])
    for dx in range(-search_range, search_range+1):
        for dy in range(-search_range, search_range+1):
            i, j = x+dx, y+dy
            if i<0 or i+block.shape[0]>=target.shape[0] or j<0 or j+block.shape[1]>=target.shape[1]:
                continue
            target_block = target[i:i+block.shape[0], j:j+block.shape[1], :]
            sad = SAD(block, target_block)
            if sad < min_sad:
                min_sad = sad
                best_mv = [dx, dy]
    return best_mv, min_sad
```

In this assignment, I implement full search method and 2d logarithmic search method. I pass the reference (40.jpg) and target (42.jpg and 51.jpg) image into the function. In full_search function, it will collect every block_size(8x8 or 16x16) pixels as a block, and put it into full_search_helper. In full_search_helper, it will compute SAD value for each mini block according to search_range(8 or 16). It will find the minimum one and return the related vector. Then, motion vectors will record all minimum SAD vector to each block, and implement predicted image by compute original position and best motion vector. Finally, it will work out residuals image with cv2.absdiff.

```python
def log_search(reference, target, block_size, search_range):
    h, w, c = reference.shape
    predicted_img = np.zeros_like(target)
    motion_vectors = np.zeros((h//block_size, w//block_size, 2), dtype=int)
    residuals_img = np.zeros_like(target)

    for row in range(0, h, block_size):
        for col in range(0, w, block_size):
            block = reference[row:row+block_size, col:col+block_size, :]
            center = (row, col)
            mv, sad = log_search_helper(block, center, target, search_range)
            motion_vectors[row//block_size, col//block_size] = mv
            predicted_img[row:row+block_size,col:col+block_size, :] = target[row+mv[0]:row+mv[0]+block_size,col+mv[1]:col+mv[1]+block_size, :]
    residuals_img = cv2.absdiff(predicted_img, target)
    return predicted_img, motion_vectors, residuals_img
```

```python
def log_search_helper(block, center, target, search_range):
    x, y = center
    dx, dy = 0, 0
    steps = int(math.ceil(math.log2(search_range)))
    for step in range(steps, 0, -1):
        best_mv = [0, 0]
        min_sad = SAD(block, target[x:x+block.shape[0], y:y+block.shape[1], :])
        for dxs in range(-1, 2):
            for dys in range(-1, 2):
                i = x+dx+(dxs<<(step-1))
                j = y+dy+(dys<<(step-1))
                if i<0 or i+block.shape[0]>=target.shape[0] or j<0 or j+block.shape[1]>=target.shape[1]:
                    continue
                target_block = target[i:i+block.shape[0], j:j+block.shape[1], :]
                sad = SAD(block, target_block)
                if sad < min_sad:
                    min_sad = sad
                    best_mv = [i-x, j-y]
        dx, dy = best_mv[0], best_mv[1]
    return best_mv, min_sad
```

Similar to full_search method, I pass reference and target image into 2d_log_search function, and it will also collect pixels to blocks and use log_search_helper to find the best motion vectors. In log_search_helper, it will calculate steps at first by log2(search_range), then in each round, step will decreasing 1 to narrow the search range. After finding the best motion vectors, return it and do the same thing in the full_search to generate predicted image, motion vectors and residual image.

a. Predicted Image

(full_predicted_r8_b16)

(full_predicted_r8_b16)

(full_predicted_r16_b8)

(full_predicted_r16_b16)

(2d_predicted_r8_b8)

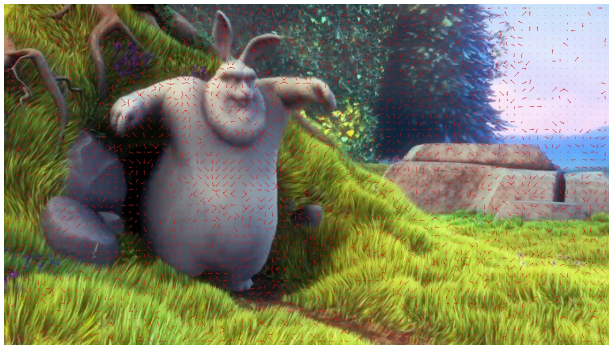
(2d_predicted_r8_b16)

(2d_predicted_r16_b8)

(2d_predicted_r16_b16)

b. Motion Vectors Image

(full_motion_vector_r8_b8)

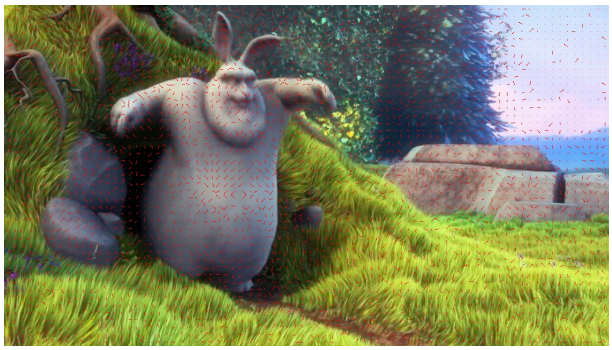


(full_motion_vector_r8_b16)

(full_motion_vector_r16_b8)

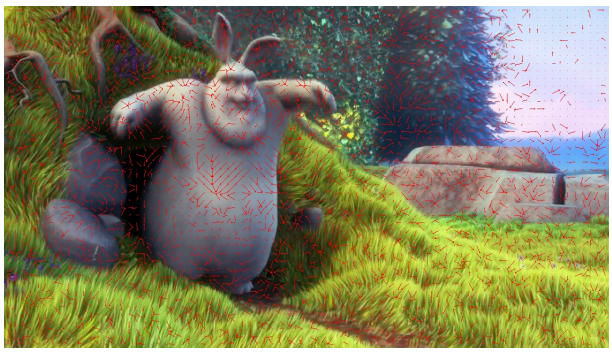
(full_motion_vector_r16_b16)

(2d_motion_vector_r8_b8)



(2d_motion_vector_r8_b16)

(2d_motion_vector_r16_b8)



(2d_motion_vector_r16_b16)

c. Residual Image

(full_residual_r8_b8)

(full_residual_r8_b16)

(full_residual_r16_b8)

(full_residual_r16_b16)

(2d_residual_r8_b8)

(2d_residual_r8_b16)

(2d_residual_r16_b8)

(2d_residual_r16_b16)

d. SAD and PSNR value

```
full_sad_value_r_8_b_8: 199025888
full_psnr_r_8_b_8: 29.699048519692802
full_sad_value_r_16_b_8: 176050038
full_psnr_r_16_b_8: 29.33847169187123
full_sad_value_r_8_b_16: 229264656
full_psnr_r_8_b_16: 29.71577369365624
full_sad_value_r_16_b_16: 197786207
full_psnr_r_16_b_16: 29.25889731784247
2d_sad_value_r_8_b_8: 138341028
2d_psnr_r_8_b_8: 29.889047635019175
2d_sad_value_r_16_b_8: 111408797
2d_psnr_r_16_b_8: 29.443626367526985
2d_sad_value_r_8_b_16: 195377334
2d_psnr_r_8_b_16: 29.85102983448687
2d_sad_value_r_16_b_16: 162994362
2d_psnr_r_16_b_16: 29.30431827791768
```

Compare different search range and block size, I find higher search range and lower block size would get better psnr and sad value. This phenomenon fit the theorem because larger search range could find more blocks, and smaller block can find the most suitable predicted block. In theory, full search should have better result than 2d logarithmic search, however, in my implementation, I don't find this appearance. I consider that I implement the different way in finding vectors, one is searching for all pixels, and the other is searching for 5 blocks only. This may the reason that my performance not fit the aspect

2.

```
Q2_full_sad_value_r_8_b_16: 283410467
Q2_full_psnr_r_8_b_16: 28.98158015080595
```

In Q2, the target image is more different with reference image than image in Q1, so it get higher SAD value and lower PSNR value.

3.

a. Executive Time

```
full_time_r_8_b_8: 58.50278997421265 seconds
2d_time_r_8_b_8: 6.719102144241333 seconds
full_time_r_16_b_8: 223.5230028629303 seconds
2d_time_r_16_b_8: 8.605195999145508 seconds
full_time_r_8_b_16: 15.921366930007935 seconds
2d_time_r_8_b_16: 1.7478749752044678 seconds
full_time_r_16_b_16: 60.04533410072327 seconds
2d_time_r_16_b_16: 2.4170098304748535 seconds
```

b. Theoretical Time Complexity

In theoretical time estimation, full search need O(((2*(search_range+1))^2) * (width/blocksize) * (height/blocksize)), and 2d logarithmicsearch need O(log2(search_range) * (width/blocksize) * (height/blocksize))