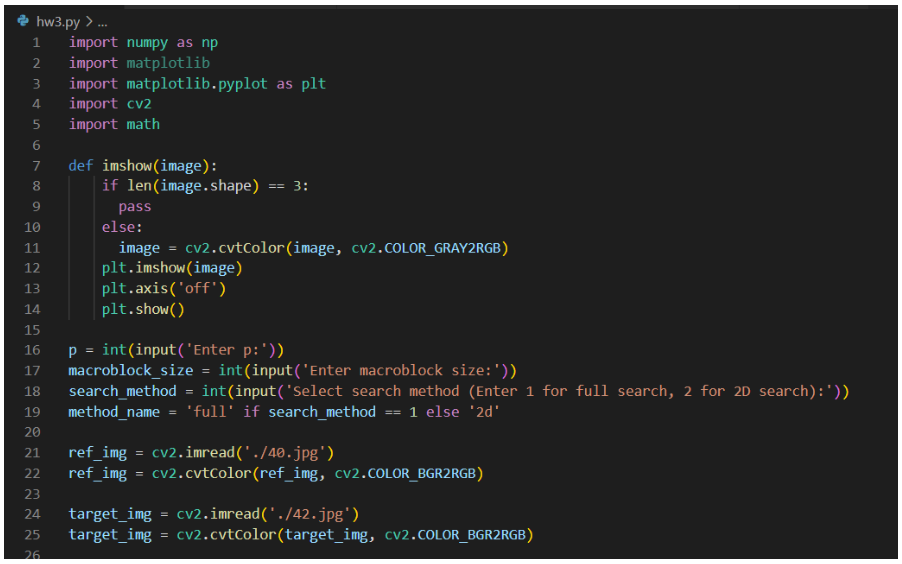
HW3 Report Gideon Levi 109006114

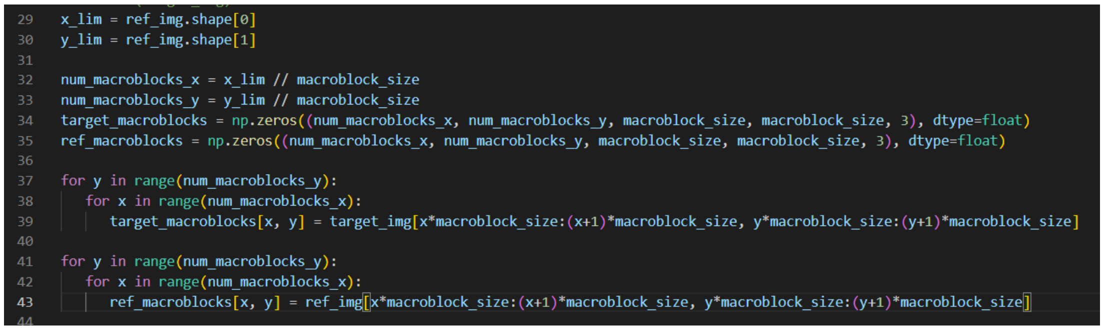
1. Motion Estimation Image Prediction



```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import cv2
import math

def imshow(image):
    if len(image.shape) == 3:
        pass
    else:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    plt.imshow(image)
    plt.axis('off')
    plt.show()

p = int(input('Enter p:'))
macroblock_size = int(input('Enter macroblock size:'))
search_method = int(input('Select search method (Enter 1 for full search, 2 for 2D search):'))
method_name = 'full' if search_method == 1 else '2d'

ref_img = cv2.imread('./40.jpg')
ref_img = cv2.cvtColor(ref_img, cv2.COLOR_BGR2RGB)

target_img = cv2.imread('./42.jpg')
target_img = cv2.cvtColor(target_img, cv2.COLOR_BGR2RGB)
```

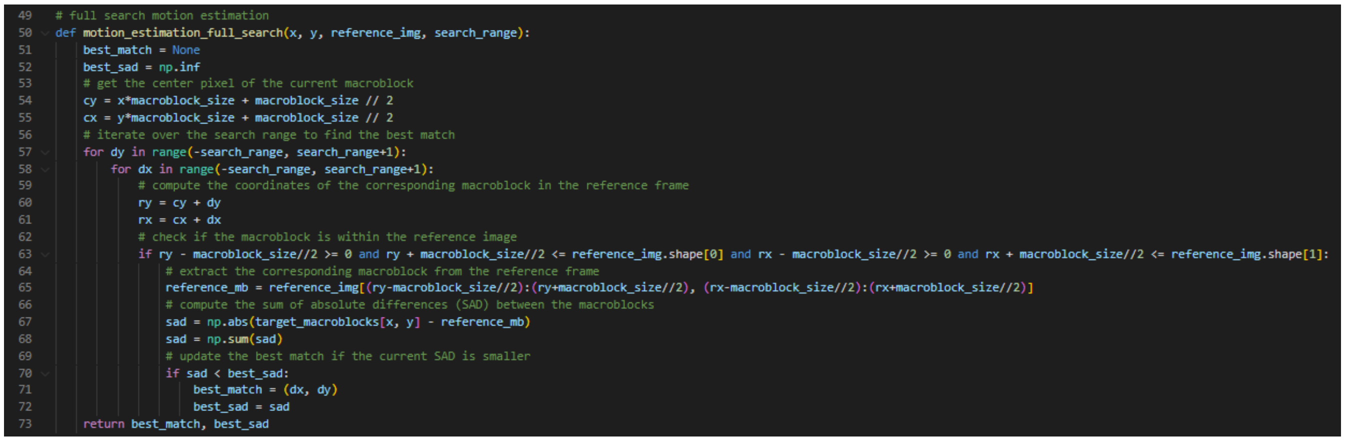
- First I import needed modules such as numpy, matplotlib.pyplot, cv2, math, and scipy. I also initialize an imshow() function to show an image. Then I ask input for value of search range, macroblock size, and to select the picture to be compressed. Then I read both reference and target image with imread() function, and convert them from BGR to RGB with cvtColor() function.

(a) Show all predicted images by using the block matching with all the above combinations.



```python
x_lim = ref_img.shape[0]
y_lim = ref_img.shape[1]

num_macroblocks_x = x_lim // macroblock_size
num_macroblocks_y = y_lim // macroblock_size
target_macroblocks = np.zeros((num_macroblocks_x, num_macroblocks_y, macroblock_size, macroblock_size, 3), dtype=float)
ref_macroblocks = np.zeros((num_macroblocks_x, num_macroblocks_y, macroblock_size, macroblock_size, 3), dtype=float)

for y in range(num_macroblocks_y):
    for x in range(num_macroblocks_x):
        target_macroblocks[x, y] = target_img[x*macroblock_size:(x+1)*macroblock_size, y*macroblock_size:(y+1)*macroblock_size]

for y in range(num_macroblocks_y):
    for x in range(num_macroblocks_x):
        ref_macroblocks[x, y] = ref_img[x*macroblock_size:(x+1)*macroblock_size, y*macroblock_size:(y+1)*macroblock_size]
```

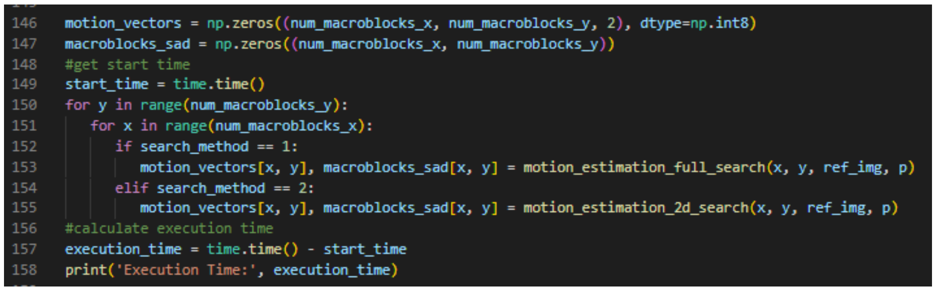
- To do motion estimation, first I divide input images to macroblocks with user's wanted size. I implement this by first calculating the number of macroblocks there will be on x-axis and y-axis, then I create an numpy.array with these numbers as its size. Then using nested for loops, I assign each macroblock_size*macroblock_size pixels as an element of macroblocks array.
- Then I define 2 functions called motion_estimation_full_search() and motion_estimation_2d_search(). These functions will take x-axis and y-axis indexes of a target image's macroblock, the reference image, and search range as arguments, and it will compare the argument's target macroblock to macroblocks on reference image in the search_range, and find the displacement of the image with minimum SAD value. Both of these functions will return the displacement (motion vector) of the image with minimum SAD value and the minimum SAD value.



```python
# full search motion estimation
def motion_estimation_full_search(x, y, reference_img, search_range):
    best_match = None
    best_sad = np.inf
    # get the center pixel of the current macroblock
    cy = x*macroblock_size + macroblock_size // 2
    cx = y*macroblock_size + macroblock_size // 2
    # iterate over the search range to find the best match
    for dy in range(-search_range, search_range+1):
        for dx in range(-search_range, search_range+1):
            # compute the coordinates of the corresponding macroblock in the reference frame
            ry = cy + dy
            rx = cx + dx
            # check if the macroblock is within the reference image
            if ry - macroblock_size//2 >= 0 and ry + macroblock_size//2 <= reference_img.shape[0] and rx - macroblock_size//2 >= 0 and rx + macroblock_size//2 <= reference_img.shape[1]:
                # extract the corresponding macroblock from the reference frame
                reference_mb = reference_img[(ry-macroblock_size//2):(ry+macroblock_size//2), (rx-macroblock_size//2):(rx+macroblock_size//2)]
                # compute the sum of absolute differences (SAD) between the macroblocks
                sad = np.abs(target_macroblocks[x, y] - reference_mb)
                sad = np.sum(sad)
                # update the best match if the current SAD is smaller
                if sad < best_sad:
                    best_match = (dx, dy)
                    best_sad = sad
    return best_match, best_sad
```

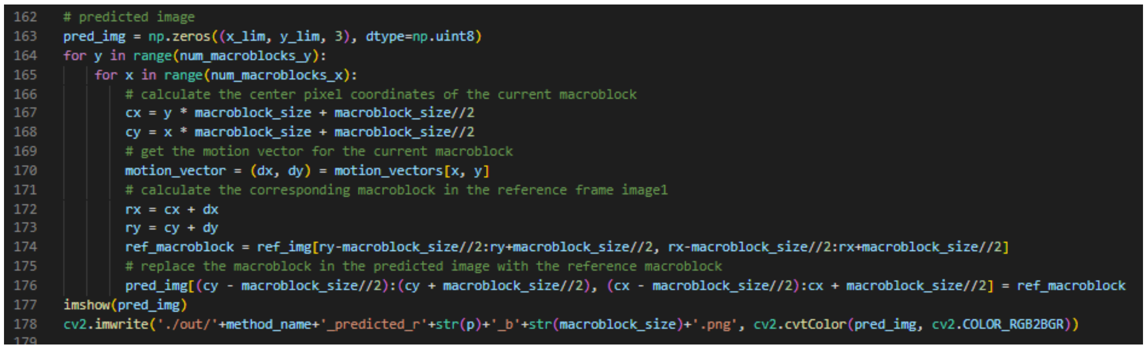
- I implement motion_estimation_full_search() function by first initialize the best match and best SAD (sum of absolute differences) to None and infinity, respectively. Then I compute the center pixel coordinates of the current macroblock in the target image.
- Then I iterate over a search range (specified by the input argument) to find the best match for the target macroblock in the reference image. For each candidate macroblock in the reference image, the function computes its SAD with respect to the target macroblock. If the SAD of the current candidate macroblock is smaller than the current best SAD, the function updates the best match and best SAD.
- Finally, the function returns the best match (in terms of the x and y displacements from the current macroblock) and the best SAD.

- For motion_estimation_2d_search() function, I implement it using the algorithm in PPT Unit 5 page 53.



```python
motion_vectors = np.zeros((num_macroblocks_x, num_macroblocks_y, 2), dtype=np.int8)
macroblocks_sad = np.zeros((num_macroblocks_x, num_macroblocks_y))
#get start time
start_time = time.time()
for y in range(num_macroblocks_y):
    for x in range(num_macroblocks_x):
        if search_method == 1:
            motion_vectors[x, y], macroblocks_sad[x, y] = motion_estimation_full_search(x, y, ref_img, p)
        elif search_method == 2:
            motion_vectors[x, y], macroblocks_sad[x, y] = motion_estimation_2d_search(x, y, ref_img, p)
#calculate execution time
execution_time = time.time() - start_time
print('Execution Time:', execution_time)
```

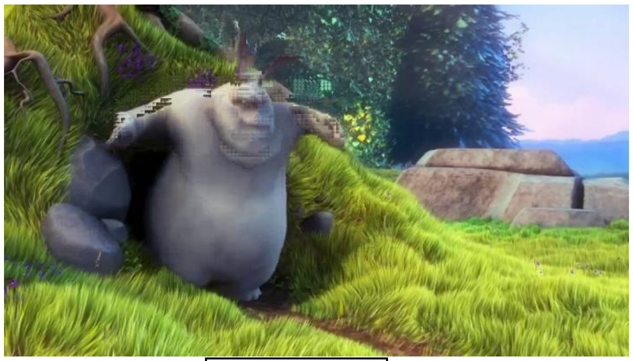
- Then I initialize 2 arrays with the same size as the number of macroblocks, one to store motion vectors and the other one to store the SAD values. For the array to store the motion vectors, for each macroblock it has 2 elements, motion on x-axis and motion on y-axis.
- Then I use nested for loops to find the motion vectors of image with minimum SAD value and its SAD value, and store them in motion_vectors array and macroblocks_sad array. I find them using the functions I defined earlier depending on the selected search method. If the selected search method is full search, I find them using motion_estimation_full_search(), otherwise if the selected search method is 2d logarithmic search, I find them using motion_estimation_2d_search() function.
- Right before the nested for loops, I invoke time.time() function to record the start time, and after the nested for loops is over I print the current time - the start time which is the total execution time of motion estimation for each macroblocks.

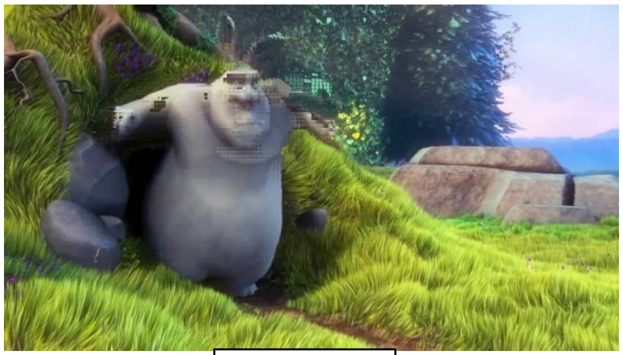


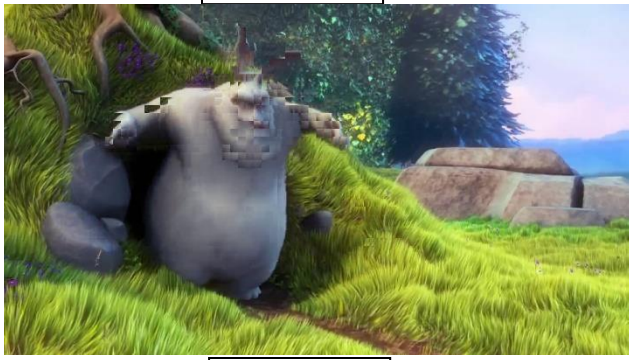
```python
# predicted image
pred_img = np.zeros((x_lim, y_lim, 3), dtype=np.uint8)
for y in range(num_macroblocks_y):
    for x in range(num_macroblocks_x):
        # calculate the center pixel coordinates of the current macroblock
        cx = y * macroblock_size + macroblock_size//2
        cy = x * macroblock_size + macroblock_size//2
        # get the motion vector for the current macroblock
        motion_vector = (dx, dy) = motion_vectors[x, y]
        # calculate the corresponding macroblock in the reference frame image1
        rx = cx + dx
        ry = cy + dy
        ref_macroblock = ref_img[ry-macroblock_size//2:ry+macroblock_size//2, rx-macroblock_size//2:rx+macroblock_size//2]
        # replace the macroblock in the predicted image with the reference macroblock
        pred_img[(cy - macroblock_size//2):(cy + macroblock_size//2), (cx - macroblock_size//2):cx + macroblock_size//2] = ref_macroblock
imshow(pred_img)
cv2.imwrite('./out/'+method_name+'_predicted_r'+str(p)+'_b'+str(macroblock_size)+'.png', cv2.cvtColor(pred_img, cv2.COLOR_RGB2BGR))
```

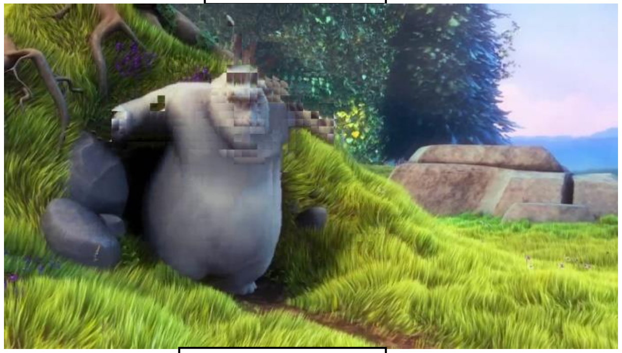
- Then using the reference image and the motion vectors I calculated earlier, I create the predicted image. I implement this by first initializing the numpy.array to store the predicted image. Then using nested for loops, for each macroblock in the predicted image, I get the motion vector for the corresponding macroblock, and the center position of the macroblock. Then I set the macroblock value to be the macroblock in reference image pointed by the motion vector. For example, predicted macroblock in position (1, 2) have (2,3) as its motion vector, so the value of that predicted macroblock is set to be macroblock with position (3, 5) in reference image.
- Show the predicted images

FULL SEARCH METHOD

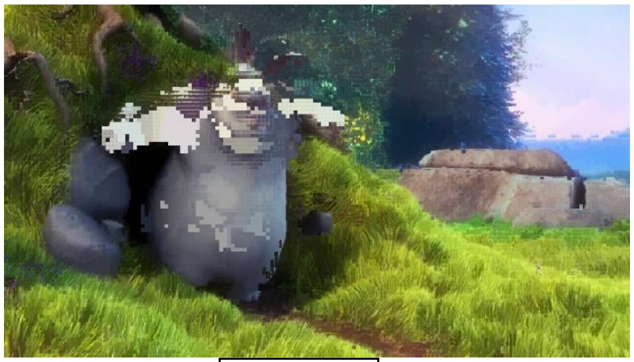

r=8, b=8

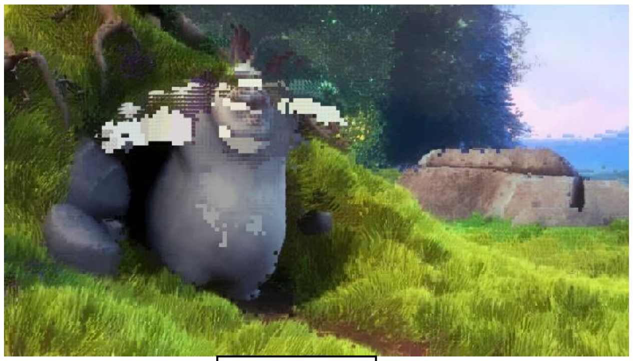

r=16, b=8

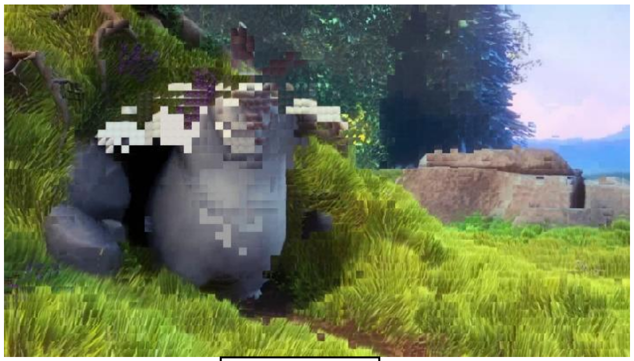

r=8, b=16

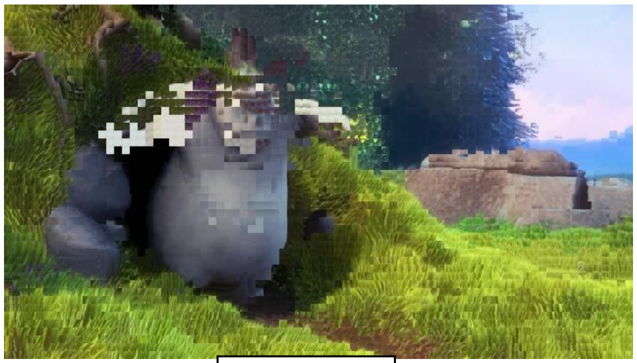

r=16, b=16

2D LOGARITHM SEARCH METHOD

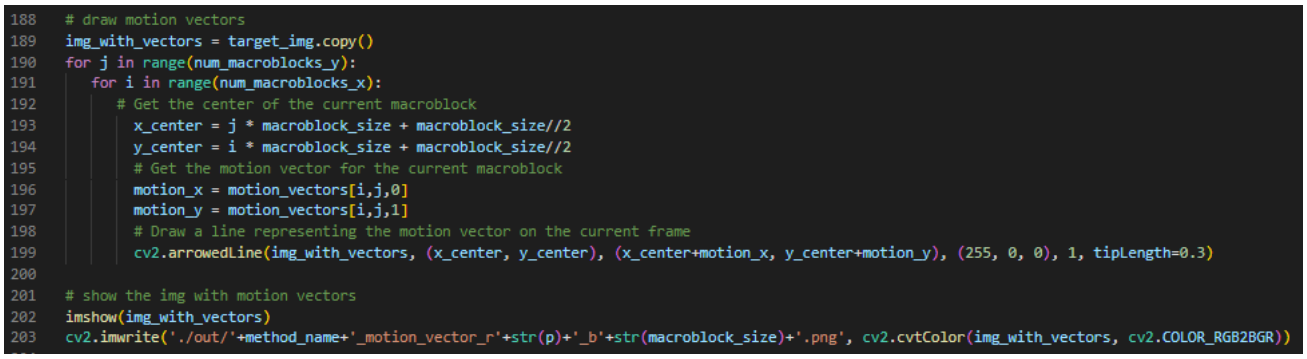

r=8, b=8


r=16, b=8


r=8, b=16


r=16, b=16

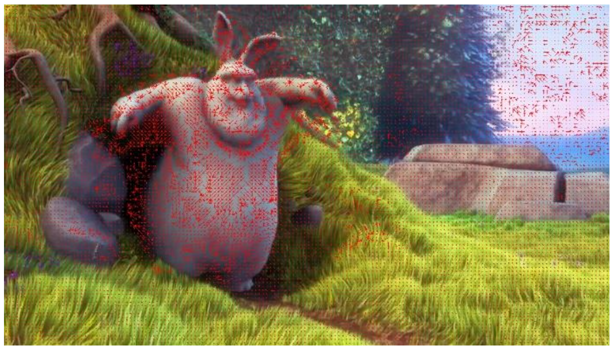
(b) Show the motion vectors images for all the above combinations.

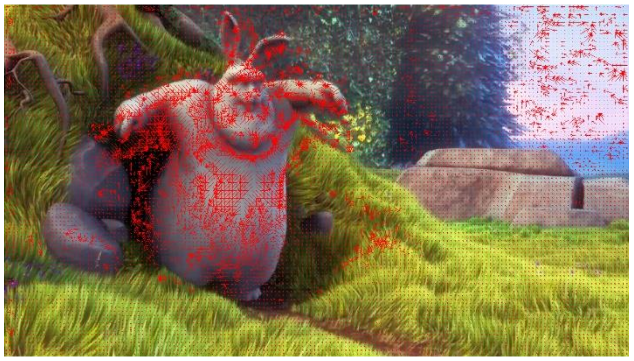


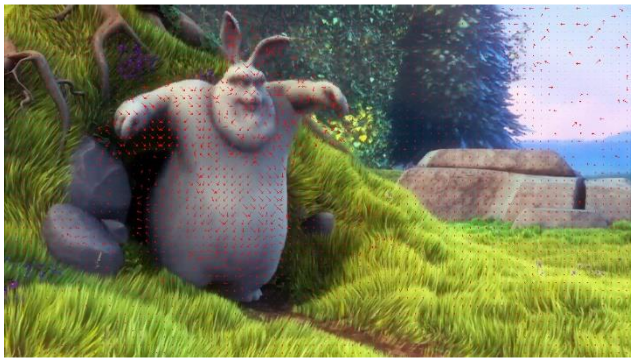
```python
# draw motion vectors
img_with_vectors = target_img.copy()
for j in range(num_macroblocks_y):
    for i in range(num_macroblocks_x):
        # Get the center of the current macroblock
        x_center = j * macroblock_size + macroblock_size//2
        y_center = i * macroblock_size + macroblock_size//2
        # Get the motion vector for the current macroblock
        motion_x = motion_vectors[i,j,0]
        motion_y = motion_vectors[i,j,1]
        # Draw a line representing the motion vector on the current frame
        cv2.arrowedLine(img_with_vectors, (x_center, y_center), (x_center+motion_x, y_center+motion_y), (255, 0, 0), 1, tipLength=0.3)

# show the img with motion vectors
imshow(img_with_vectors)
cv2.imwrite('./out/'+method_name+'_motion_vector_r'+str(p)+'_b'+str(macroblock_size)+'.png', cv2.cvtColor(img_with_vectors, cv2.COLOR_RGB2BGR))
```

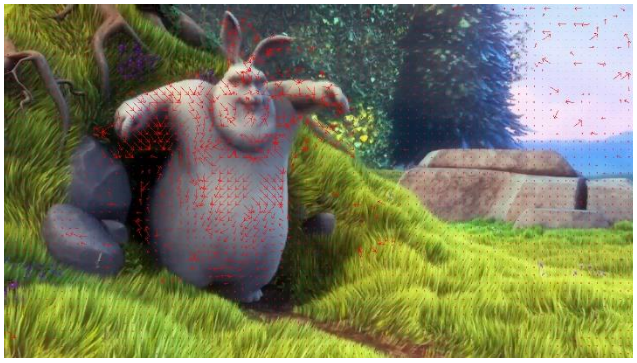
- To create an image with motion vectors, first I copy the target image and save it as “img_with_vectors”. Then for each macroblock, I get the motion vector of the corresponding macroblock from motion vectors array, and get the center position of the macroblock in the target image. Then I draw the motion vector from the center of the macroblock to center added by its displacement (motion vector). I draw the line with cv2.arrowedLine() function.
- I then show the image with motion vectors and save it with cv2.imwrite() function
- Show images

FULL SEARCH METHOD

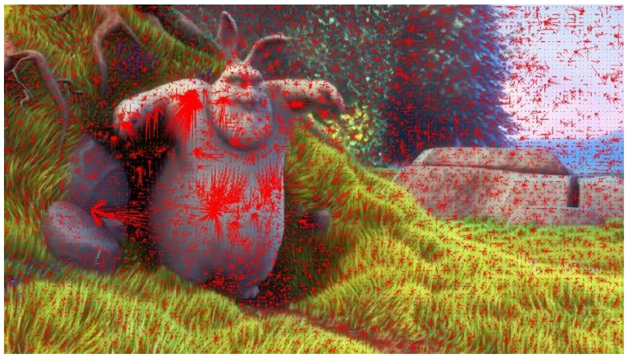

r=8, b=8

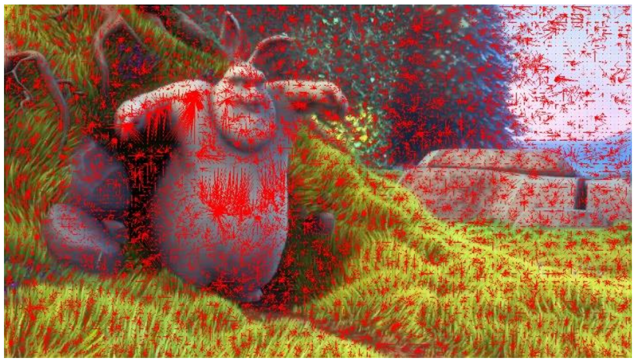

r=16, b=8

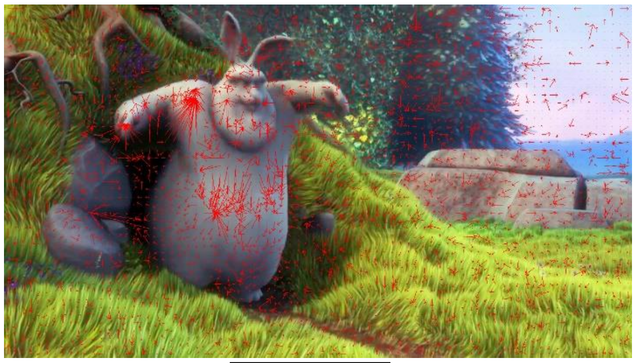

r=8, b=16

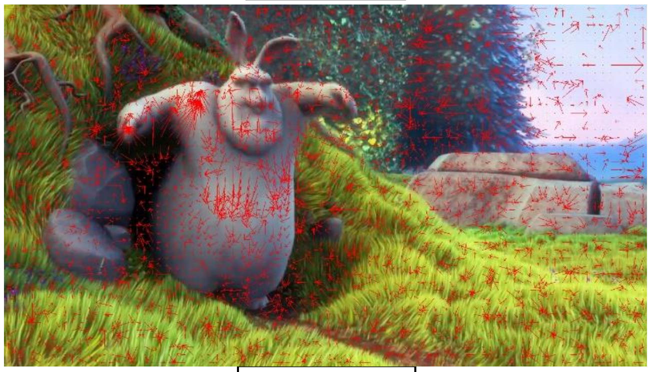

r=16, b=16

2D LOGARITHMIC SEARCH METHOD

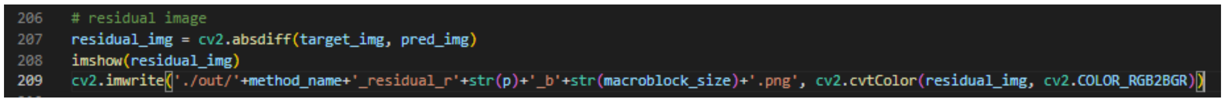

r=8, b=8


r=16, b=8


r=8, b=16


r=16, b=16

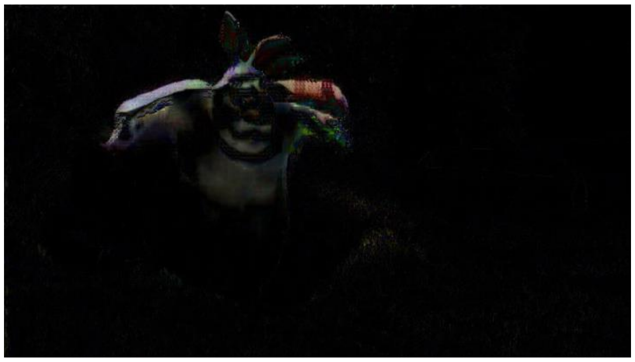
(c) Show the residual images for all the above combinations.



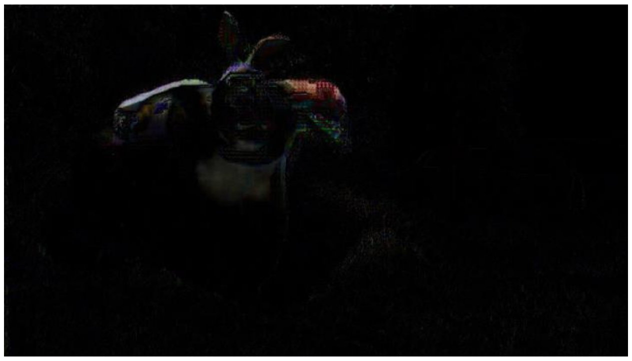
```
# residual image
residual_img = cv2.absdiff(target_img, pred_img)
imshow(residual_img)
cv2.imwrite('./out/'+method_name+'_residual_r'+str(p)+'_b'+str(macroblock_size)+'.png', cv2.cvtColor(residual_img, cv2.COLOR_RGB2BGR))
```

- I get the residual image by taking the absolute value of the difference between target image and predicted image, and store them as residual image.
- Then I show the image and save it with cv2.imwrite() function.

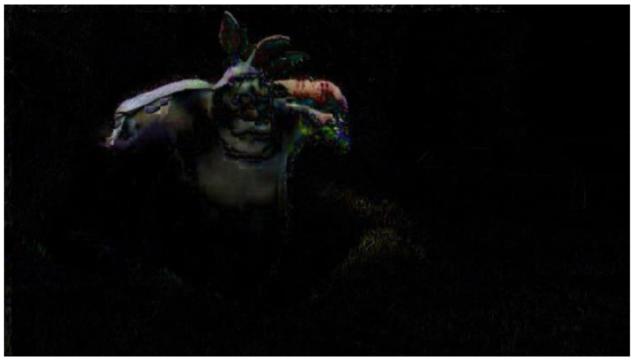
FULL SEARCH METHOD



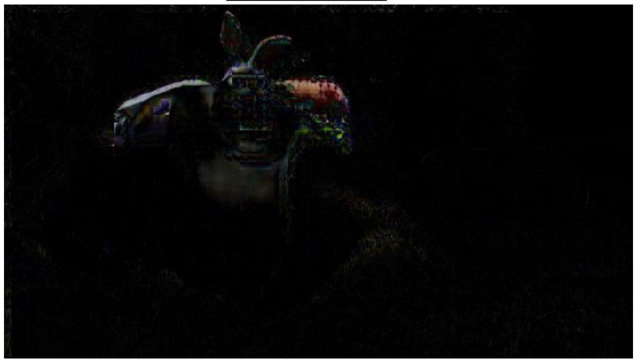
r=8, b=8



r=16, b=8

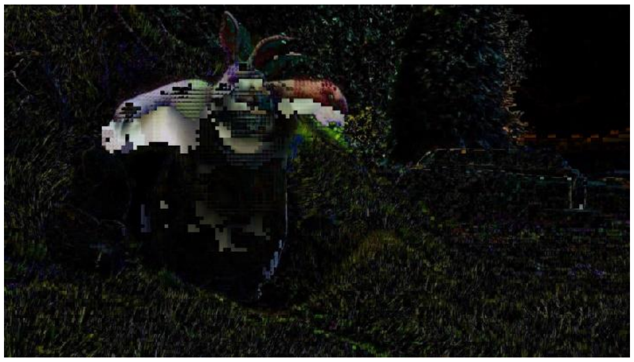


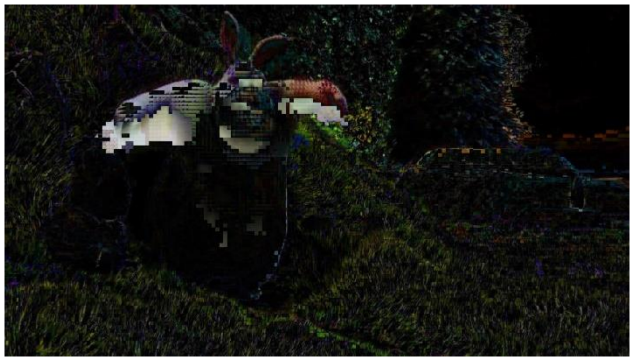
r=8, b=16

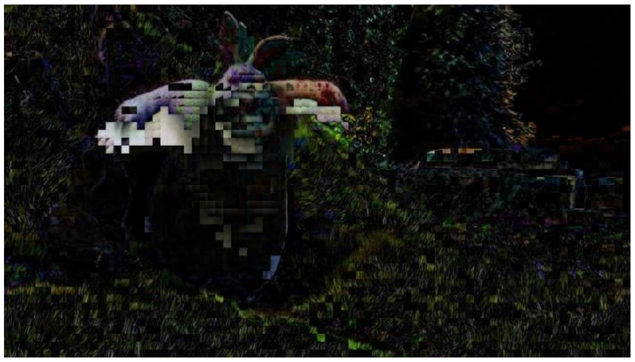


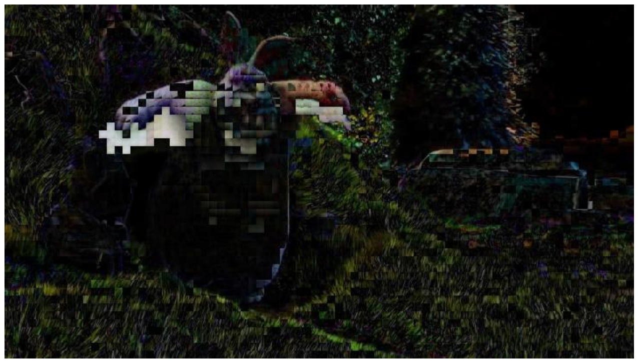
r=16, b=16

2D LOGARITHMIC SEARCH METHOD

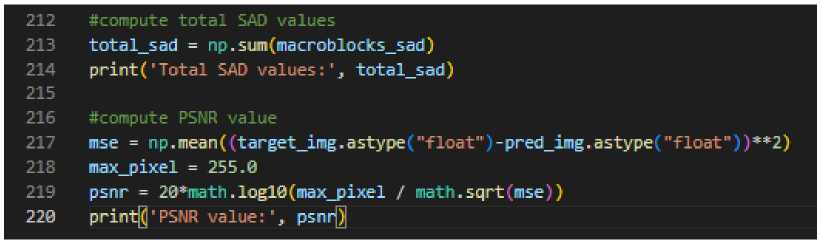

r=8, b=8


r=16, b=8


r=8, b=16


r=16, b=16

(d) Compute the total SAD values and PSNR for all the results. Discuss the motion-based image prediction quality for all the above settings.



```
#compute total SAD values
total_sad = np.sum(macroblocks_sad)
print('Total SAD values:', total_sad)

#compute PSNR value
mse = np.mean((target_img.astype("float")-pred_img.astype("float"))**2)
max_pixel = 255.0
psnr = 20*math.log10(max_pixel / math.sqrt(mse))
print('PSNR value:', psnr)
```

- To calculate the total SAD I sum all the elements in macroblocks_sad array by using numpy.sum() function. Then I print the SAD total value.
- For PSNR values, I calculate it by first calculate the mean squared error value, and since the peak value of a pixel is 255, the PSNR is 20 multiplied by log 10 of (255/mse).

- Discuss motion-based image prediction quality for all the above settings.

FULL SEARCH METHOD

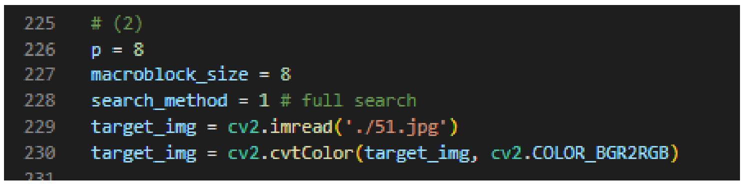
```
Enter p:8
Enter macroblock size:8
Total SAD values: 15262980.0
PSNR value: 24.297260496195744

Enter p:8
Enter macroblock size:16
Total SAD values: 17148940.0
PSNR value: 23.551103922947437

Enter p:16
Enter macroblock size:8
Total SAD values: 11430553.0
PSNR value: 27.478381938014902

Enter p:16
Enter macroblock size:16
Total SAD values: 13372111.0
PSNR value: 26.262348114944913
```

2D LOGARITHMIC SEARCH METHOD

```
Enter p:8
Enter macroblock size:8
Total SAD values: 119387536.0
PSNR value: 18.12382349813217

Enter p:8
Enter macroblock size:16
Total SAD values: 162138615.0
PSNR value: 17.791947942509204

Enter p:16
Enter macroblock size:8
Total SAD values: 103705574.0
PSNR value: 18.07288347040934

Enter p:16
Enter macroblock size:16
Total SAD values: 143282964.0
PSNR value: 17.49313869176525
```

- Here we can see that the PSNR value of predicted image with full search method is higher than the PSNR value of predicted image with 2D logarithmic search method. This shows that predicting with full search method is more accurate than with 2D logarithmic search method.
- Another thing I observe is the bigger the macroblock size is, then the bigger the total SAD values is, this means using macroblock with smaller size is better but in price of more computing time and smaller compression ratio.
- I also find out that for full search method, using bigger search range will increase the PSNR value, which results in better predicted image. For 2D logarithmic search on the other hand, using bigger search range will decrease the PSNR value.

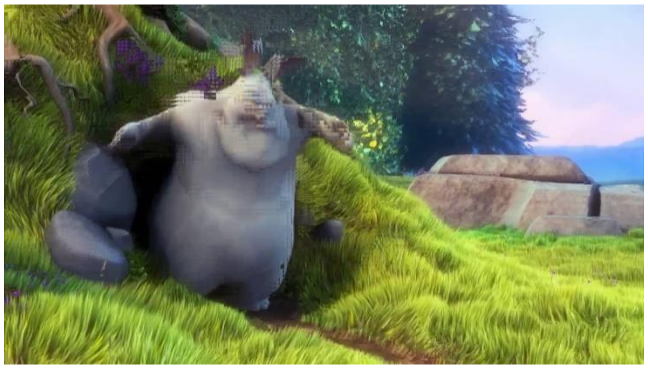
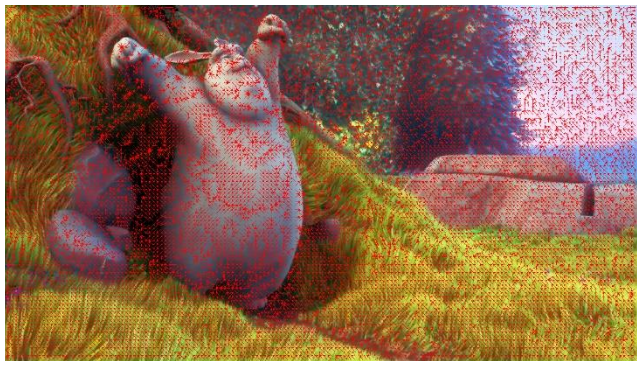
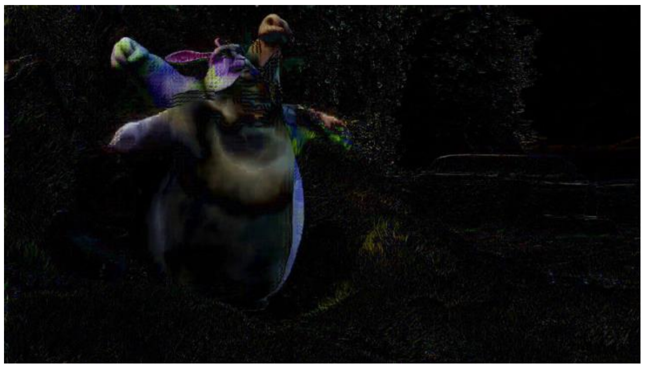
2. **Try the full search method with search range p=8 and macroblock sizes = 8x8. The reference image is 40.jpg, and the target image is 51.jpg. Show the PSNR of the result. Compare and discuss the PSNR with the result of same search range and macroblock in question 1.**



```python
# (2)
p = 8
macroblock_size = 8
search_method = 1 # full search
target_img = cv2.imread('./51.jpg')
target_img = cv2.cvtColor(target_img, cv2.COLOR_BGR2RGB)
```

- For this part, I use the same implementation as number 1, but I change the value of p to 8, macroblock size to 8, and the target image to 51.jpg.

- Compare and discuss the PSNR with the result of same search range and macroblock in question 1.

Total SAD values: 30306992.0

PSNR value: 21.482402867966833

- The total SAD value is the biggest we have seen so far, it is because the motion is a lot bigger than the search range. The PSNR value is also worse if compared to the result of the same search range and macroblock size with full search method, but still better if compared to the result of the same search range and macroblock size with 2D logarithmic search method.

3. Analyze the time complexity

(a) Measure the execution time required for the two search algorithms with the two different search range sizes (p=8 and p=16)

- Show execution time of all combinations.

FULL SEARCH METHOD

```
Enter p:8
Enter macroblock size:8
Execution Time: 49.55596113204956

Enter p:8
Enter macroblock size:16
Execution Time: 14.435173988342285

Enter p:16
Enter macroblock size:8
Execution Time: 198.0193212032318

Enter p:16
Enter macroblock size:16
Execution Time: 58.77709102630615
```

2D ALGORITHMIC SEARCH METHOD

```
Enter p:8
Enter macroblock size:8
Execution Time: 5.9046430587768555

Enter p:8
Enter macroblock size:16
Execution Time: 2.4309868812561035

Enter p:16
Enter macroblock size:8
Execution Time: 6.761638164520264

Enter p:16
Enter macroblock size:16
Execution Time: 2.171905517578125
```

(b) Compare and discuss the execution time with the theoretical time complexity for the two search algorithms.

- Let p be the search range, s be the size of macroblocks, and n be the number of macroblocks. Theoretically, the time complexity for full search method is O((2p+1)^2 * s^2 * n) which equals to O(p^2 * s^2 * n). As for 2D logarithmic search method, the time complexity is O(log_2(p) * s^2 * n).
- Discuss.
    - From the execution times I just shown, we can see that the execution time of full search method is a lot larger than the execution time of 2D algorithmic search method, similar to what we can see from the theoretical time of both search method.
    - We can also see that for full search method, increasing search range and/or decreasing macroblock size will cause a big increase in execution time, as theoretically, increasing search range and/or decreasing macroblock will increase the execution time exponentially.
    - As for 2D algorithmic search method, increasing search range don’t have as much impact as decreasing macroblock size, as it will increase in logarithmically.