REPORT FOR COMP9517 ASSIGNMENT 1

zID: z5140893
Name: Chengze Du

In the report, I use 'light_rail.jpg' as the input image. The original picture is as ***Figure 1***:

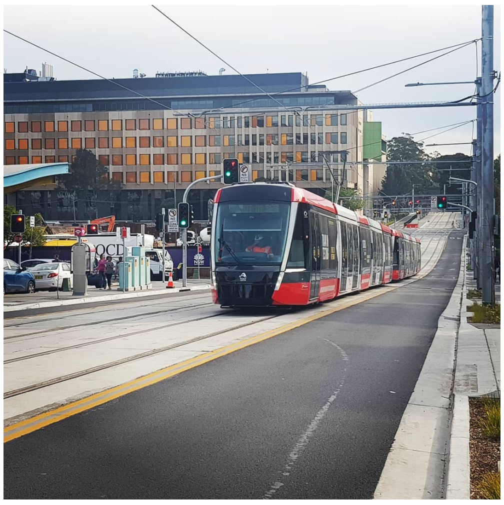
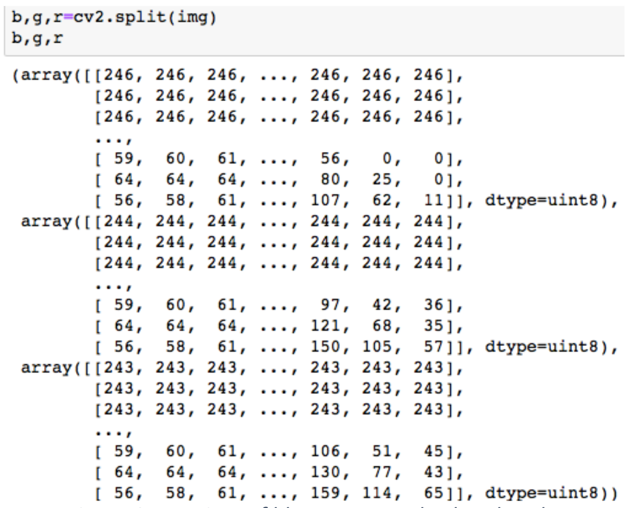


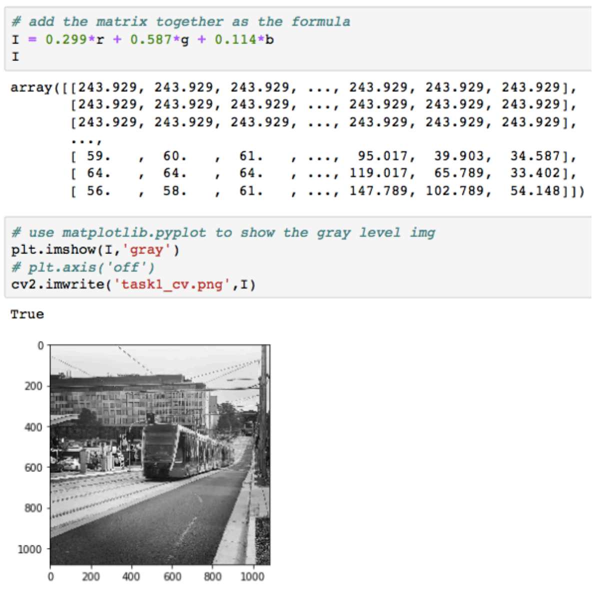
Figure 1: original picture



```
b,g,r=cv2.split(img)
b,g,r
```

```
(array([[246, 246, 246, ..., 246, 246, 246],
        [246, 246, 246, ..., 246, 246, 246],
        [246, 246, 246, ..., 246, 246, 246],
        ...,
        [ 59,  60,  61, ...,  56,   0,   0],
        [ 64,  64,  64, ...,  80,  25,   0],
        [ 56,  58,  61, ..., 107,  62,  11]], dtype=uint8),
 array([[244, 244, 244, ..., 244, 244, 244],
        [244, 244, 244, ..., 244, 244, 244],
        [244, 244, 244, ..., 244, 244, 244],
        ...,
        [ 59,  60,  61, ...,  97,  42,  36],
        [ 64,  64,  64, ..., 121,  68,  35],
        [ 56,  58,  61, ..., 150, 105,  57]], dtype=uint8),
 array([[243, 243, 243, ..., 243, 243, 243],
        [243, 243, 243, ..., 243, 243, 243],
        [243, 243, 243, ..., 243, 243, 243],
        ...,
        [ 59,  60,  61, ..., 106,  51,  45],
        [ 64,  64,  64, ..., 130,  77,  43],
        [ 56,  58,  61, ..., 159, 114,  65]], dtype=uint8))
```

Figure 2: matrixes of blue, green, red colour bands



```
# add the matrix together as the formula
I = 0.299*r + 0.587*g + 0.114*b
I
```

```
array([[243.929, 243.929, 243.929, ..., 243.929, 243.929, 243.929],
       [243.929, 243.929, 243.929, ..., 243.929, 243.929, 243.929],
       [243.929, 243.929, 243.929, ..., 243.929, 243.929, 243.929],
       ...,
       [ 59.   ,  60.   ,  61.   , ...,  95.017,  39.903,  34.587],
       [ 64.   ,  64.   ,  64.   , ..., 119.017,  65.789,  33.402],
       [ 56.   ,  58.   ,  61.   , ..., 147.789, 102.789,  54.148]])
```

```
# use matplotlib.pyplot to show the gray level img
plt.imshow(I,'gray')
# plt.axis('off')
cv2.imwrite('task1_cv.png',I)
```

```
True
```

Figure 3: get the picture 'I'

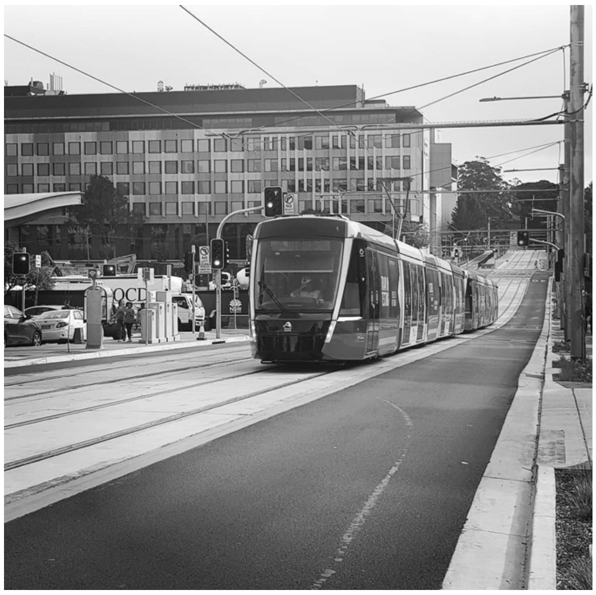


Figure 4: picture 'I'

Task 1:

Firstly, I use 'cv2.split' to split the original picture to 3 colour bands which are blue, green and red. They are all 2-dimension matrixes shown as ***Figure 2***.

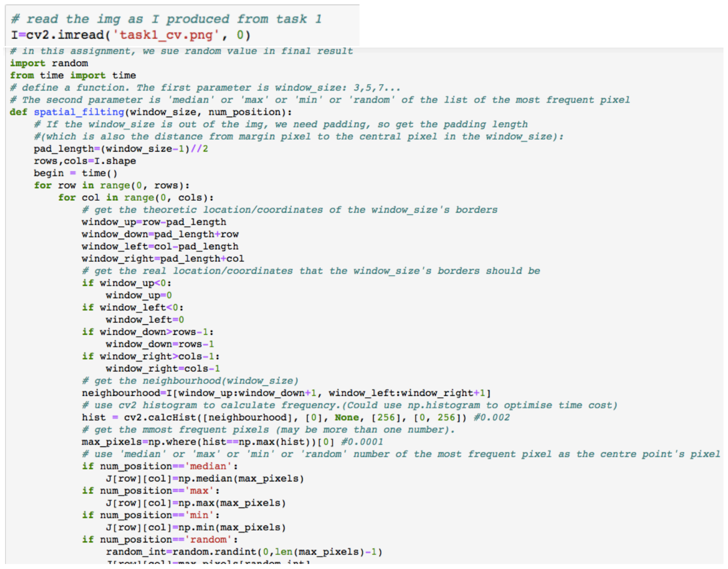
Then, directly use the format ' I = 0.299*r + 0.587*g + 0.114*b ' to get the grey-level image 'I', and save it as 'task1_cv.png' (Shown as ***Figure 3*** and ***Figure 4***).

Task 2:

Firstly, I read the gray-level image ‘I’ which is produced in task 1. Then get the number of its rows and cols.

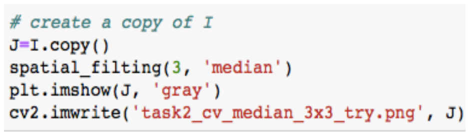
Then, I write a function ‘spatial_filtering’ which will take 2 parameters. One is the ‘window _size’. Because there may be more than one most frequent pixel values in the neighbourhood, the second parameter ‘num_position’ is for choosing the max, min, median or random value of these most frequent pixel values.

The basic idea to achieving this spatial filtering is: for every pixel in picture ‘I’, we ‘cut out’ the neighbor area(window size) around it and then count the most frequent pixel values. At the beginning, I did a padding for the image so that I set a mask to do this, but the time cost is relatively high. Then, I simplified the method to achieve this: I just directly find the neighborhood that we need. For every pixel, I initially find the theoretic window borders which is calculated by plus or minus the padding length. The ‘theoretic’ means those numbers may be less than 0 or more than 255 as they may be out of the image border. Then for the window border which are out of the image border, we just use the corresponding image border to substitute it. Thus we make a new neighborhood which is within the image. The neighborhood is what we exactly need to count. To count the most frequent pixel values in this neighbourhood. I put it into the ‘cv2.calcHist’ function(actually, we can use ‘np.histogram’ to narrow the counting range to optimise it). And find the most frequent pixel values. Then, choose the ‘max’, ‘min’, ‘median’ or ‘random’ value of these most frequent pixel values to replace the central pixel in the corresponding window in J (the copy of the picture ‘I’) with it . (***Figure 5***)



```python
# read the img as I produced from task 1
I=cv2.imread('task1_cv.png', 0)
# in this assignment, we sue random value in final result
import random
from time import time
# define a function. The first parameter is window_size: 3,5,7...
# The second parameter is 'median' or 'max' or 'min' or 'random' of the list of the most frequent pixel
def spatial_filting(window_size, num_position):
    # If the window_size is out of the img, we need padding, so get the padding length
    #(which is also the distance from margin pixel to the central pixel in the window_size):
    pad_length=(window_size-1)//2
    rows,cols=I.shape
    begin = time()
    for row in range(0, rows):
        for col in range(0, cols):
            # get the theoretic location/coordinates of the window_size's borders
            window_up=row-pad_length
            window_down=pad_length+row
            window_left=col-pad_length
            window_right=pad_length+col
            # get the real location/coordinates that the window_size's borders should be
            if window_up<0:
                window_up=0
            if window_left<0:
                window_left=0
            if window_down>rows-1:
                window_down=rows-1
            if window_right>cols-1:
                window_right=cols-1
            # get the neighbourhood(window_size)
            neighbourhood=I[window_up:window_down+1, window_left:window_right+1]
            # use cv2 histogram to calculate frequency.(Could use np.histogram to optimise time cost)
            hist = cv2.calcHist([neighbourhood], [0], None, [256], [0, 256]) #0.002
            # get the mmost frequent pixels (may be more than one number).
            max_pixels=np.where(hist==np.max(hist))[0] #0.0001
            # use 'median' or 'max' or 'min' or 'random' number of the most frequent pixel as the centre point's pixel
            if num_position=='median':
                J[row][col]=np.median(max_pixels)
            if num_position=='max':
                J[row][col]=np.max(max_pixels)
            if num_position=='min':
                J[row][col]=np.min(max_pixels)
            if num_position=='random':
                random_int=random.randint(0,len(max_pixels)-1)
```

Figure 5: spatial filtering function



```python
# create a copy of I
J=I.copy()
spatial_filting(3, 'median')
plt.imshow(J, 'gray')
cv2.imwrite('task2_cv_median_3x3_try.png', J)
```

```
True
```

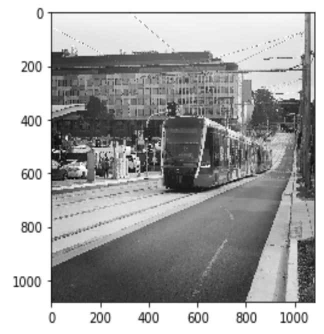


Figure 6: the result of spatial filtering

Finally, do the spatial filtering for the picture ‘I’ and get a new picture ‘J’. (Shown as ***Figure 6***, the window size is 3x3 and use median of the most frequent pixel values in the neighbourhood as the central pixel value).

***Unintentional discovery* : Analyze of choosing most frequent pixel values and choosing window size(erodibility):**

The general process is as above. Also, I have done several experiments with the window size 3x3, 5x5 and 7x7. For each of them, I choose max, median, min and random value of the most frequent pixel values. I use 3x3 as an example (shown as ***Figure 7,8,9,10***). Even though the ‘max’ (***Figure** 7*) and ‘min’ (***Figure 9***) are more blurred which may mean they have better performance in this oil painting task, yet ‘max’ will make the picture brighter and dark areas near the bright area are eroded, such as the wires in the red circle (***Figure** 7*); as for ‘min’, vice visa, ‘max’ will make the picture darker and bright areas near the dark area are eroded. ‘Median’ and ‘Random’(***Figure 10***) are between ‘max’ and ‘min’. Even though ‘median’ and ‘random’ cannot make the picture very blurred compared to ‘max’ and ‘min’, it is relatively stable and mild. However, this phenomenon only appears in choosing small window size, because in small window size, it would have similar probability that the most frequent pixel values may different from the central pixel value or same as the central pixel value which means that the central pixel will mostly influenced by its close neighbors which may have the same pixel value as the central pixel’s. Therefore, the value choice has significant influence on the most frequent value choice. With the increasing of the window size(more than 10x10), the difference of the most frequent value choice is gradually not that obvious, because the most frequent pixel values will be more probably not from the close neighbors of the central pixel.

As the small window sizes have no oil-painting impact on the final picture. So, these intermediate pictures will not be submitted. To achieve the final good oil-painting effect. My final assignment will more focus on window sizes which are more than 10x10. Also, the choice of the most frequent values does not matter either.

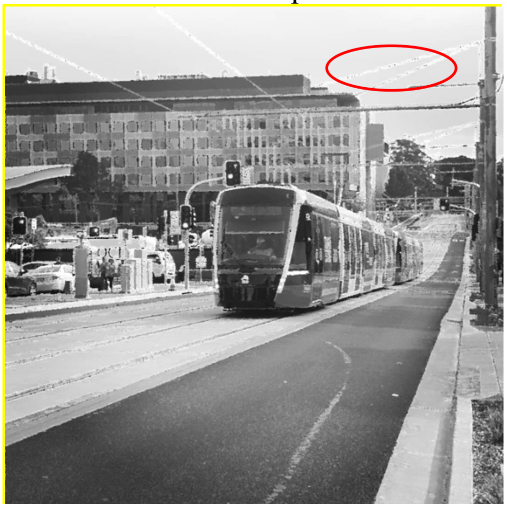


Figure 7: 3x3 ‘max’ of most frequent pixel values

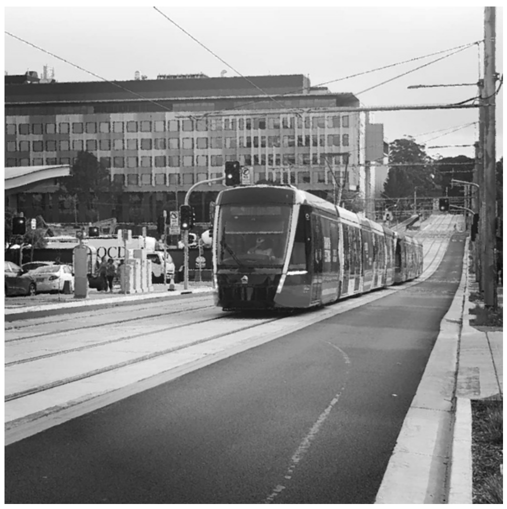


Figure 8: 3x3 ‘median’ of most frequent pixel values

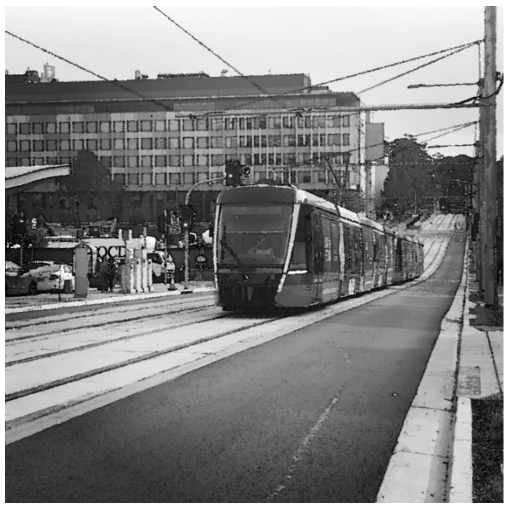


Figure 9: 3x3 ‘min’ of most frequent pixel values

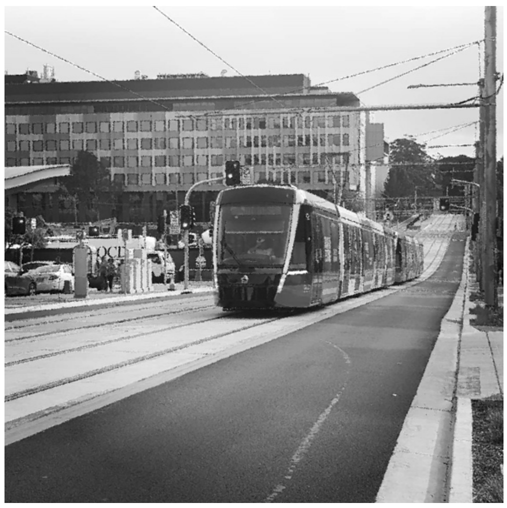


Figure 10: 3x3 ‘random’ of most frequent pixel values

Task 3:

Followed by the research of task 2, since the choice of the most frequent values does not matter. We will all choose 'random' mode to do the choice and we will compare the result from window size 11x11, 17x17, 23x23, 29x29, 35x35, 41x41, 47x47, 53x53, 59x59, 65x65, 71x71, 77x77 (shown as ***Figure 12 – 14*** at the end of the report). This is because the window size less than 10x10 cannot achieve the oil painting effect.

The task 3 needs us to find the points which have the same pixel values as the central pixel in the neighborhood in the gray level image, and then find the corresponding positions of these values in the original image and calculate the average intensities of those pixels in each band to replace the central pixel value in the original image.

Basically, we do not use the padding and mask method either. We just directly find the neighborhood area like what we did in task 2. We do not need to repeat how to get the neighborhood here. The key difficulty in task 3 is that after we find the location of the needed pixels, we need to find the corresponding location in the original picture. Let's assume that 'B' is the copy of the original image and 'neighborhood' is the window size area(e.g., 10x10). The key formula is:

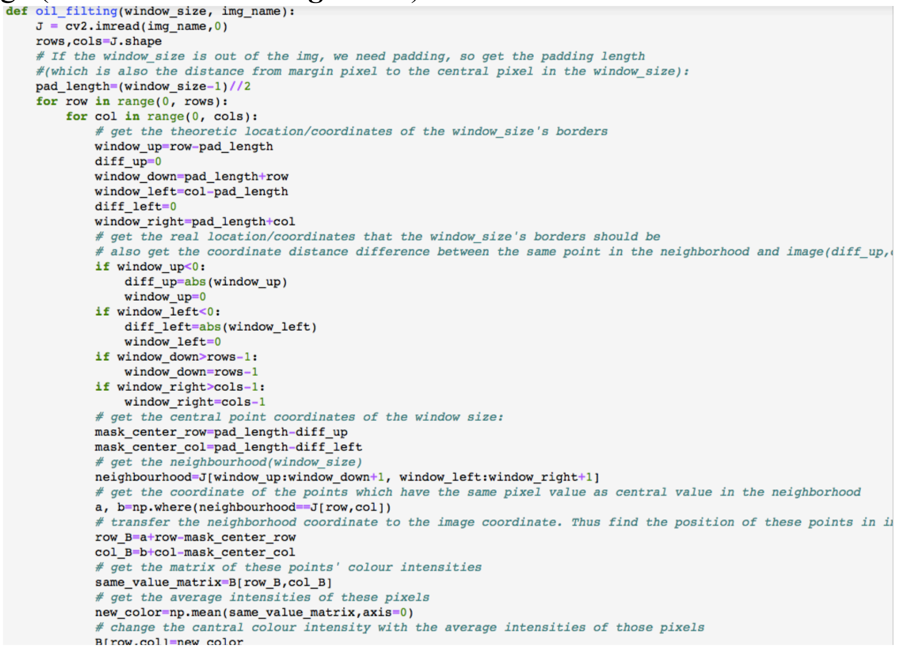
$$\boldsymbol{central\_point\_location\_in\_neighborhood} - \boldsymbol{central\_point\_location\_in\_B} = \boldsymbol{point\_location\_in\_neighborhood} - \boldsymbol{point\_location\_in\_B}$$

$$\boldsymbol{central\_point\_location\_in\_B} = (\boldsymbol{row}, \boldsymbol{col})$$

$$\boldsymbol{central\_point\_location\_in\_neighborhood} = (\boldsymbol{pad\_length} - \boldsymbol{diff\_up}, \boldsymbol{pad\_length} - \boldsymbol{diff\_left})$$

'diff_up' is the distance difference between theoretic upper neighborhood border and image border(absolute value): $\boldsymbol{abs}(\boldsymbol{row} - \boldsymbol{pad\_length})$
'diff_left' is the distance difference between theoretic left neighborhood border and image border(absolute value): $\boldsymbol{abs}(\boldsymbol{col} - \boldsymbol{pad\_length})$
'point_location_in_neighborhood' can be find by 'np.where'.

Combine the above formula, we can get the 'point_ location_in_B' easily. Then just use the average intensities of those pixels in each band to replace the central pixel value in the original image.(Code is shown as ***Figure 11***)



```
def oil_filting(window_size, img_name):
    J = cv2.imread(img_name,0)
    rows,cols=J.shape
    # If the window_size is out of the img, we need padding, so get the padding length
    #(which is also the distance from margin pixel to the central pixel in the window_size):
    pad_length=(window_size-1)//2
    for row in range(0, rows):
        for col in range(0, cols):
            # get the theoretic location/coordinates of the window_size's borders
            window_up=row-pad_length
            diff_up=0
            window_down=pad_length+row
            window_left=col-pad_length
            diff_left=0
            window_right=pad_length+col
            # get the real location/coordinates that the window_size's borders should be
            # also get the coordinate distance difference between the same point in the neighborhood and image(diff_up,
            if window_up<0:
                diff_up=abs(window_up)
                window_up=0
            if window_left<0:
                diff_left=abs(window_left)
                window_left=0
            if window_down>rows-1:
                window_down=rows-1
            if window_right>cols-1:
                window_right=cols-1
            # get the central point coordinates of the window size:
            mask_center_row=pad_length-diff_up
            mask_center_col=pad_length-diff_left
            # get the neighbourhood(window_size)
            neighbourhood=J[window_up:window_down+1, window_left:window_right+1]
            # get the coordinate of the points which have the same pixel value as central value in the neighborhood
            a, b=np.where(neighbourhood==J[row,col])
            # transfer the neighborhood coordinate to the image coordinate. Thus find the position of these points in i
            row_B=a+row-mask_center_row
            col_B=b+col-mask_center_col
            # get the matrix of these points' colour intensities
            same_value_matrix=B[row_B,col_B]
            # get the average intensities of these pixels
            new_color=np.mean(same_value_matrix,axis=0)
            # change the cantral colour intensity with the average intensities of those pixels
            B[row,col]=new_color
```

Figure 11 Task 3 oil-painting spatial filtering function

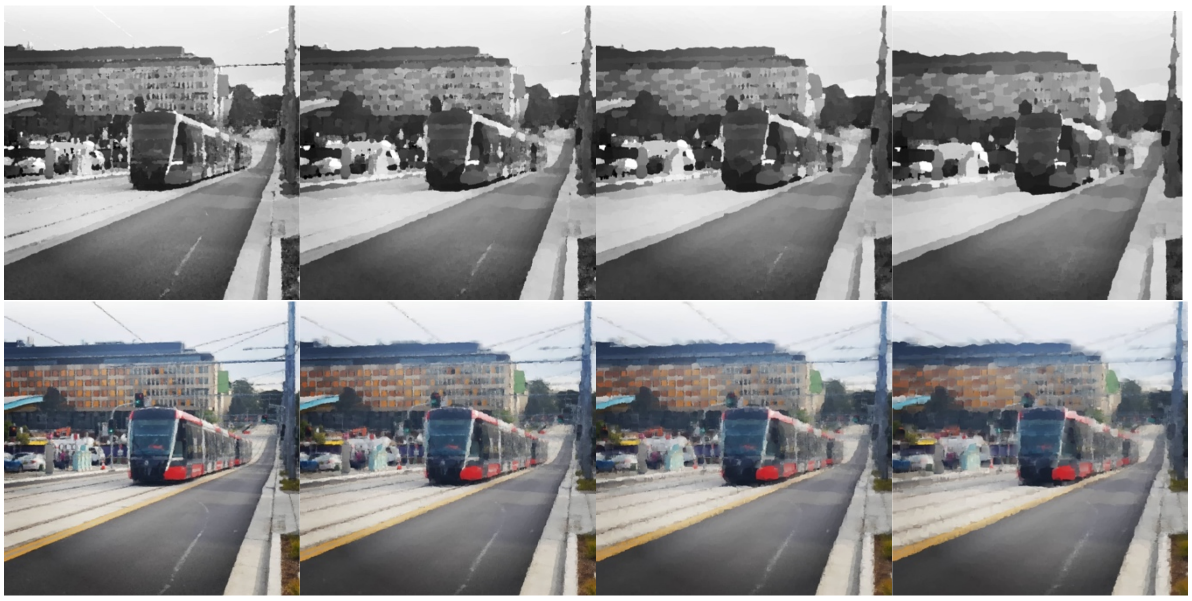


Figure 12 window size of 11x11, 17x17, 23x23, 29x29

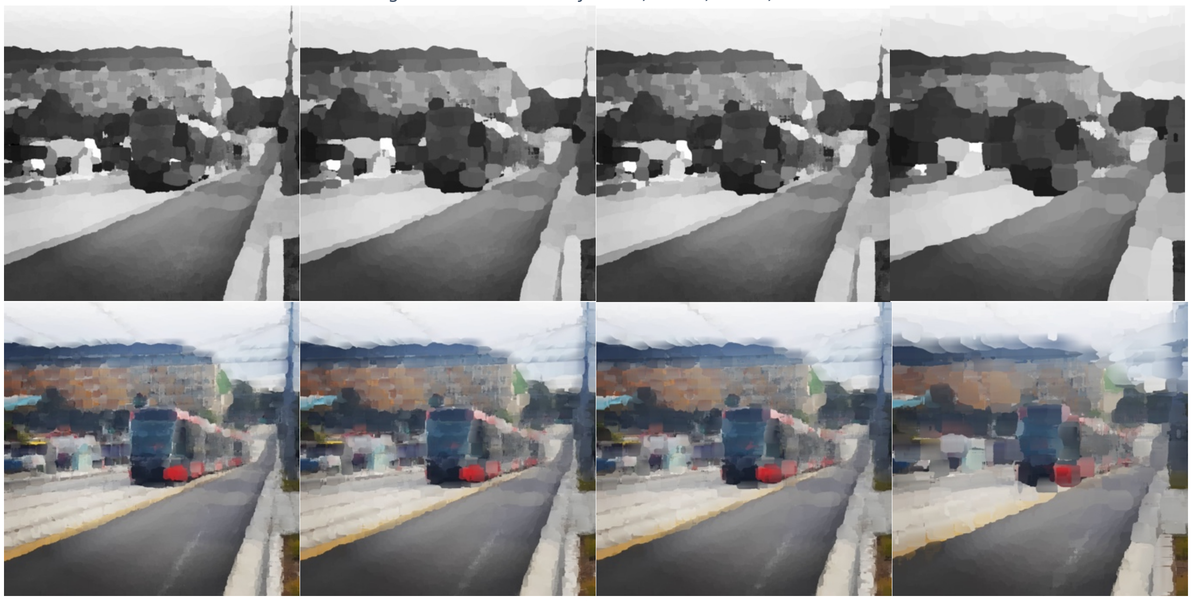


Figure 13 window size of 35x35, 41x41, 47x47, 53x53

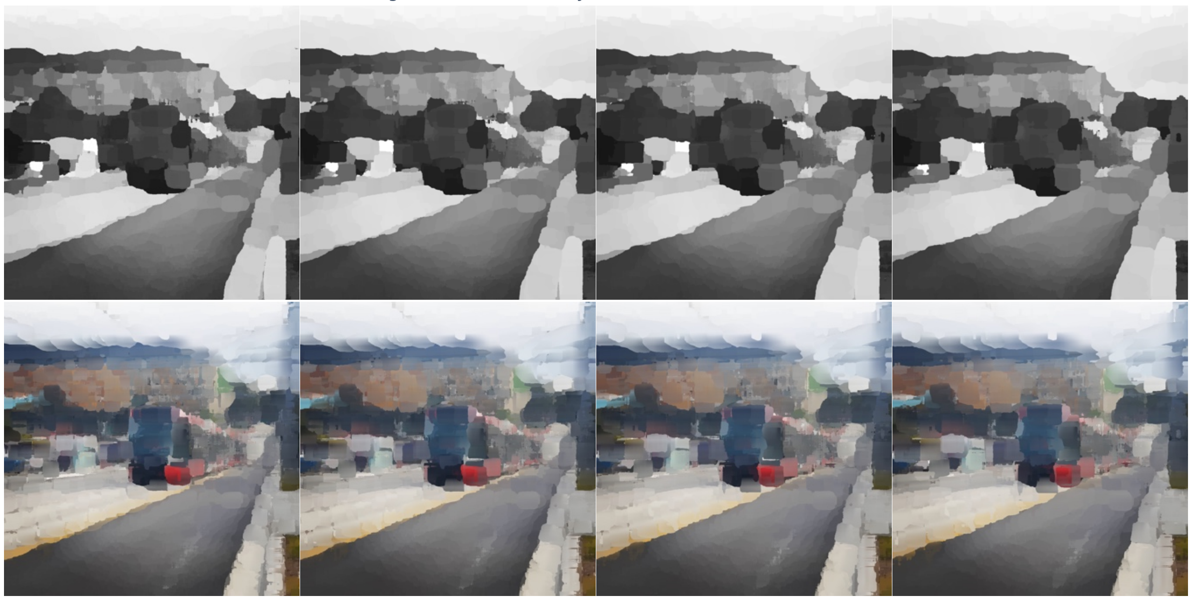


Figure 14 window size of 59x59, 65x65, 71x71, 77x77