Computer Vision Techniques

Oil Paint Report

Shashank Reddy Boosi

Abstract

Image filtering is one of the most popular Image processing techniques which helps us in blurring, sharpening, embossing, edge-detection, and more. It is one of the common ways to pre-process the images before moving on to the feature representation and modelling. This report discusses about oil-paint filtering on a dog and light_rail image in 3 stages. The first stage does the conversion of the BGR image to gray scale image. The second stage calculates the most frequent pixels in the neighborhood pixels of given window size for each (x, y) and replace (x, y) with frequent pixel. The final stage does the intensity calculation of the gray scale pixels on each (x, y) and does the average of the pixel value in the corresponding original image. The stages will be explained in detailed later in the report and the tasks prove that oil painting is prominent, when the window size that is chosen is proportional as given in equation 1, and the filtering is done on 3 different window sizes namely $((5, 5), (15, 15), (25, 25))$ to prove the proportionality.

$$Oil - Paint\ Prominence \quad \alpha \quad Neighborhood\ Window\ Size \tag{1}$$

Implementation

Task 1

In task 1, our goal is to convert the original color image (A) to a grey scale image i.e, to convert the image with 24 bits to 8 bits image. All the tasks will be done mainly using openCV and Numpy packages.

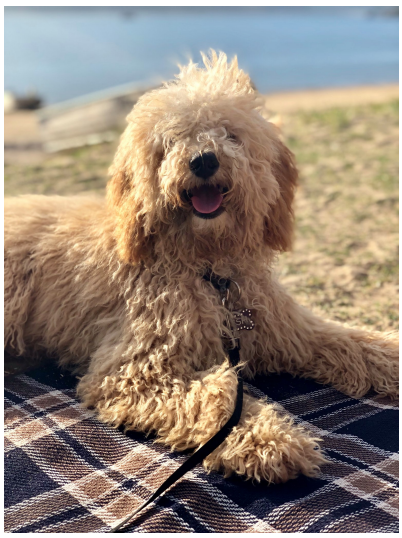
I will first read the image and the colors of the image are in the form of BGR (default). I created an empty array with the size equal to the size of the input image (A). Then the input image is split into B, G, R channels respectively and applied to the equation 2 as given below:

$$I(x,y) = 0.299 * R(x,y) + 0.587 * B(x,y) + 0.114 * G(x,y) \tag{2}$$

where R, G, and B are the channels of the image.

Iteration is done over the equation with the channels obtained from the input image and the newly formed image $I(x,y)$ is saved using *cv2.imwrite*. Figure 1, shows the original image A and grey image I for the images of dog and light_rail which are our given sample images.

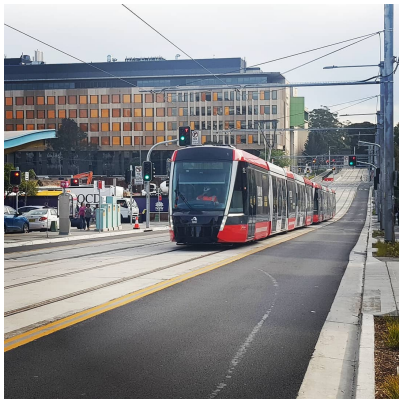
Dog Image



a) Original Image

b) Grey Scale Image

Rail Image



c) Original Image

d) Gray Scale Image

Figure 1: Original to Grey Scale Results I obtained as a result of Task 1

The use of this task is to use a single channel to do the filtering operations in the next tasks instead of using 3 channels which will consume significant amount of time.

Task 2

In Task 2, we need to find the most frequent pixels from the neighborhood pixels of a given window size and replace $I(x, y)$ with the most frequent pixel. Repeat the operation for every (x, y). The input image will the output grey image from Task 1 and the output image will be the oil-painted grey image $J(x, y)$.

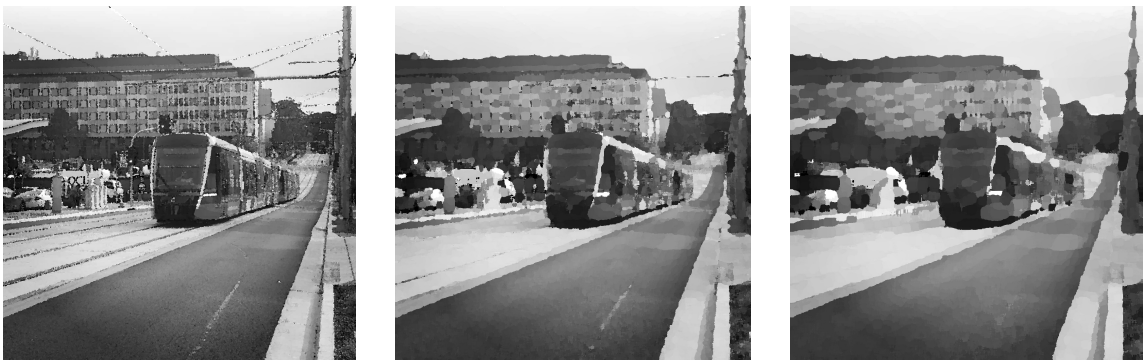
The task is performed on 3 different window sizes namely $((5, 5), (15, 15), (25, 25))$. One of the main operations to be done before starting the Task is to pad the image so that the edges of the image can also take part in the task. After a lot of analysis, I used the reflection of the image to the degree of half the window size on all directions because if we use paddings like zero padding, constant number padding, replicate padding, I noticed that the Task is about frequency and there is a high chance for the paddings listed above to change the outcome of the task which is not the goal of padding for this task. Reflection padding will not effect the operation of the task because it is the mirror image of half the window size used and the pixel count for all the neighbors will just be twice without effecting the max frequency pixel. Padding is applied using the function *cv2.copyMakeBorder* and the reflection is applied using the variable *cv2.BORDER_REFLECT_101*, which is given as the parameter to *cv2.copyMakeBorder*.

Border is assigned based on the window size and a copy of the output image $J(x, y)$ is created from $I(x, y)$. The newly padded image that is formed is iterated over the window size on each pixel of $I(x, y)$. In every iteration, we get the neighborhood pixels of the defined window size and the most frequently occurred pixel is calculated during a histogram. In openCV, the histogram gives us the plot of *No of pixels vs The pixel count* we get the list of pixel counts from the histogram and extract the pixels with maximum count which is going to be the most frequent pixel in that neighborhood. The histogram is called using the function *cv2.calcHist* and we use *np.argmax* to get the maximum pixel. Once the most frequent pixel is extracted, the $J(x, y)$ in that iteration will be replaced with the most frequent pixel value.

Dog Image

a) Image with window Size 5 b) Image with window Size 15 c) Image with window Size 25

Rail Image



d) Image with window Size 5 e) Image with window Size 15 f) Image with window Size 25

Figure 2: Intermediate Results J obtained as a result of Task 2

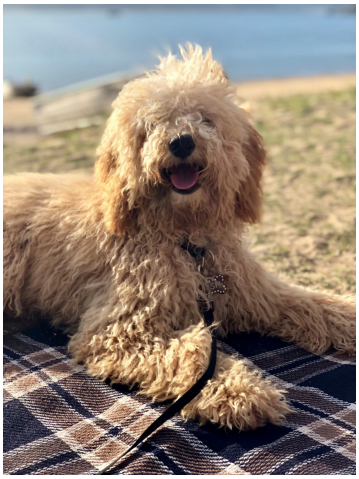
Figure 2, above shows the implementation of the task 2 on dog and light_rail images where for each image, 3 images are extracted for each window size. As we see from the figure, the window size with maximum value is getting more oil-painted which proves the equation 1 shown before.

Task 3

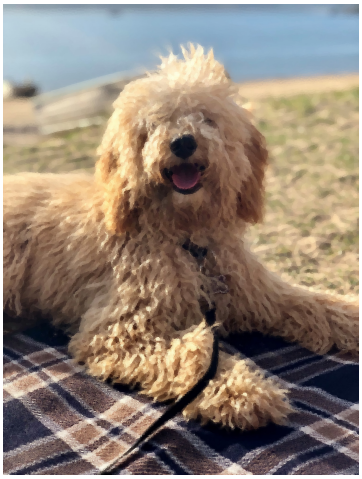
In Task 3, we need to construct the oil-painting effect on the original Image $A(x, y)$ using the oil-painted grey image $J(x, y)$. The input image will be the oil-paint grey image $J(x, y)$ from Task 2 and the output image will be the oil-painted color image $B(x, y)$.

The task is performed on 3 different window sizes namely $((5, 5), (15, 15), (25, 25))$. We need to remember that the window size that is used should be the same as the window size used to create $J(x, y)$. One of the main operations to be done before starting the task is to pad the image so that the edges of the image can also take part in the task. After a lot of analysis, I used the zero padding of the image to the degree of half the window size on all the directions because if we use paddings like reflection padding, replicate padding, the task is about finding the similar pixel (x, y) in the neighborhood and zero is suitable because it only effects the outcome of the output if the edges are black and in any other case the padding will not effect the outcome of the operation. If we use the ones not used, there is a high chance for the paddings listed above to change the outcome of the task which is not the goal of padding for this task. Zero padding is applied using the variable *cv2.BORDER_CONSTANT* with *value=0*, which are given as the parameters to *cv2.copyMakeBorder*.

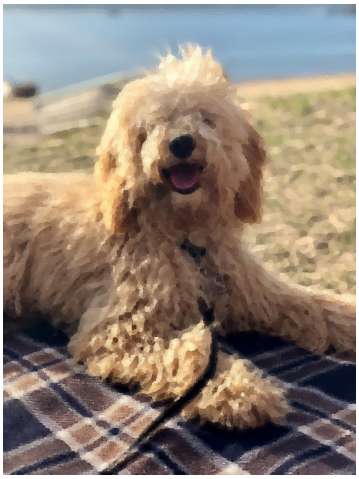
Dog Image



a) Image with window Size 5

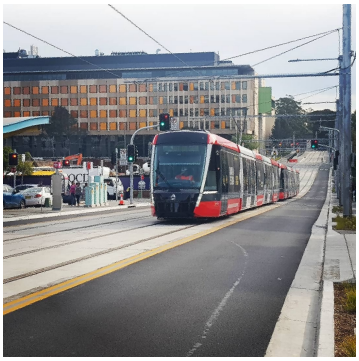


b) Image with window Size 15

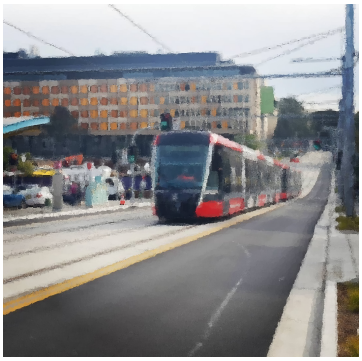


c) Image with window Size 25

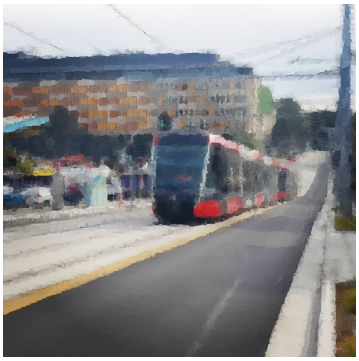
Rail Image



d) Image with window Size 5



e) Image with window Size 15



f) Image with window Size 25

Figure 3: Oil Paint Results B obtained as a result of Task 3

Initially, the original image is converted to RGB from GBR using the operation *cv2.cvtColor(image, cv2.COLOR_BGR2RGB)*. The RGB image is split into R, B, G channels and the border is assigned to all the channels and $J(x, y)$ based on the window size and a copy of the output image $B(x, y)$ is created from $A(x, y)$. The newly padded image from $J(x, y)$ that is formed is iterated over the window size on each pixel of $J(x, y)$.

For every iteration, we get the neighborhood pixels of (x, y) that are similar to (x, y). We then average the intensities of all the pixels in the original image $A(x, y)$ for each padded channel R, G, B at positions that are similar to (x, y) and replace (x, y) at each channel and then after the whole iteration merge the three channel in the order of B, G, R to get the oil painted BGR color image $B(x, y)$. Merging is done using the function *cv2.merge(B, G, R)*.

Figure 3, above shows the implementation of the task 3 on dog and light_rail images where for each image, 3 images are extracted for each window size. As we see from the figure, the window size with maximum value is getting more oil-painted on the original image because the effect on gray image and the color image should be the same. This further confirms equation 1.

Interpretations from the tasks

1. The resolution of the image reduces when the size of the window increases.
2. Choosing the window size is intuitive because after a certain window size the details of the oil painted image are getting lost.
3. Prominence of oil paint is proportional to the size of the image
4. The computational time is more when the size of the window increases.
5. Choosing the padding type is intuitive and depends upon the operations we perform.