Digital Image Processing Project Report

Team: Mob_Psycho

Project Title: Fast Weighted Median Filtering

Project ID: 12 [Link]

Github Link: https://github.com/Kanav-7/DIP-Project

Team Members

Kanav Gupta (20161151)

Anurag Mehta (20161016)

Introduction

In this project we have implemented a faster way to apply weighted median filter on any given image in a much more efficient way than a naive method. The time complexity of original weighted median filtering (WMF) is O(r^2*logr), where r is size of kernel. In our project we have reduced this complexity to O(r) making it 100+ times faster than general method and that too with negligible error. Various Data Structure techniques and mappings were used to get such an efficient output.

Problem Statement/Motivation

The inexpensiveness and simplicity of point and shoot cameras, combined with the speed at which budding photographers can send their photos over the Internet to be viewed by the world, makes digital photography a popular hobby these days.

With each snap of a digital photograph, a signal is transmitted from a photon sensor to a memory chip embedded inside a camera.

Transmission technology is prone to a degree of error,and noise is added to each photograph.

Significant work has been done in both hardware and software to improve the signal-to-noise ratio in digital photograph. The technique of weighted median filtering is mostly used to encounter these errors.

Weighted median, in the form of either solver or filter, has been employed in a wide range of computer vision applications for its beneficial properties in sparsity representation.

As a local operator, weighted median can effectively filter images while not strongly blurring edges. More importantly, it mathematically corresponds to global optimization, which produces results with fully explainable connections to global energy minimization. In many computer vision problems, including stereo matching, optical flow estimation, and image denoising. Making it efficient will have direct impact on time complexity numerous applications using it.

There are many methods which speed up Unweighted median filtering but it is impossible to apply those to weighted median filter because of the spatially varying weights for each local window. These varying weights hamper simplifying computation in a straightforward way.

Also there are other methods that make weighted average filtering efficient (like Bilateral Filtering) etc. but unfortunately methods deployed in these cannot be used here because of following reasons:

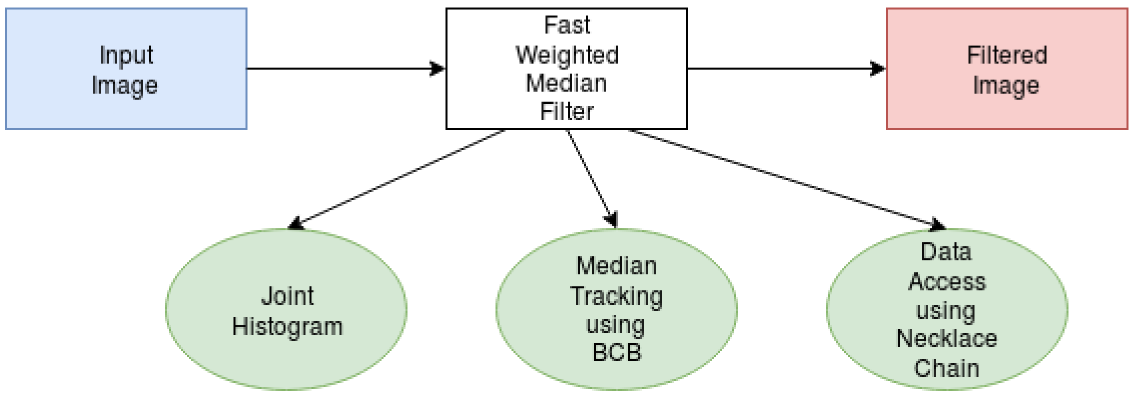
- The filtering kernel is not separable.
- It cannot be approximated by interpolation or down-sampling.
- There is no iterative solution

The above mentioned constraints indeed make the task of making the weighted median filter efficient challenging.

Overview

The project includes three major techniques to achieve the target. which includes data-weight distribution by a new joint histogram to dynamically allocate weights and pixels, a median tracking algorithm to reduce time of seeking median by leveraging color coherency of images, and data structure sparsity to reduce the data access cost. Note: These will be discussed later on in detail.

Overview Diagram of Project:


Input Image
Fast Weighted Median Filter
Filtered Image
Joint Histogram
Median Tracking using BCB
Data Access using Necklace Chain

Approach considered

As stated in the overview above the approach used consists of following techniques:

- Joint Histogram
- Balance Counting Box (For median tracking)
- Necklace Table (For efficient data access)

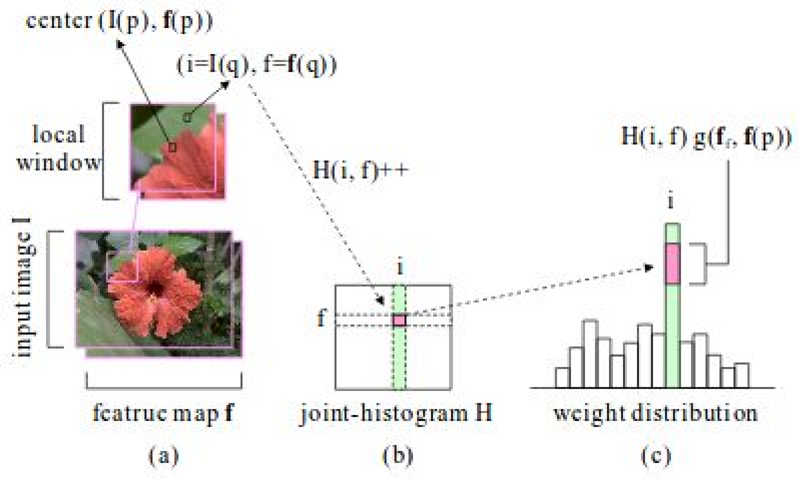
How these approaches work and correlate is discussed below

Joint Histogram

Joint histogram is a two dimensional which stores the pixel count in its bins according to the corresponding features (This is somewhat similar to one done for unweighted median filtering in our course assignment). Each row of histogram corresponds to feature index and each column corresponds to intensity value of the pixel.

What is feature index (f)?

Feature index is a index feature map of image which can possibly be intensity, color or any other high level function.



$$H(i, f) = \#\{q \in \mathcal{R}(p) | I(q) = I_i, \mathbf{f}(q) = \mathbf{f}_f\},$$

Below diagram explains them wellWeights for particular intensity can be obtained using

$$w_i = \sum_{f=0}^{N_f - 1} H(i, f)\, g(\mathbf{f}_f, \mathbf{f}(p)).$$

Now Joint histograms consists of two algorithms:

Median Finding: In this we traverse the joint-histogram to compute weights for each intensity as mentioned by above formula and then calculate the total weight in the first pass and the weight upto mid-point in second pass to obtain the required pixel.

Shift & Update: We shift the filter to the next location if it is not the end of this image. We update the joint-histogram by reducing the count in the corresponding cell by one for each pixel leaving the filter and increase it by one for each pixel moving in.

So we are able to find weighted median by using just joint histogram but this not provide us with the required amount of efficiency as we have to traverse complete joint histogram every time. For handling this issue of traversing histogram median tracking is introduced

Median Tracking

Median tracking basically exploits the property that almost all images have similar features in the adjacent median finding windows to iterate over joint-histogram in a much efficient way. Median tracking work on concept of Cut point and balance.

What are cut point and balance?
Cut point is the intensity point where cumulative sum reaches half of total sum in the current filter frame. The intensity at cut point is the basically the weighted median of that frame.

Balance is absolute min difference between left side weight and right side weight. Refer below diagram for better understanding

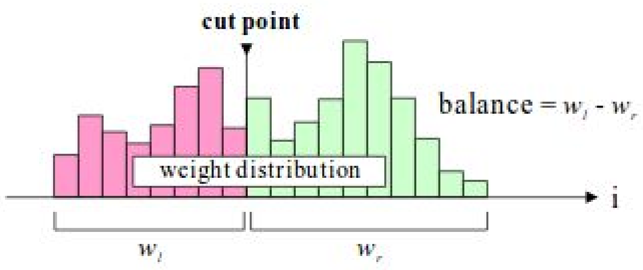


Figure 3. Illustration of the balance and cut point.

How is median tracked using cut point and balances?

Given the cut point c(p) for the frame centered at p, we can estimate the cut point for frame centered at p+1 using balances. If balance is positive we move cut point to left i.e c(p) - 1 and then check the balance. This is repeated until balance become negative, and point previous to this is set as cut point for c(p+1). Same can be done if balance obtained is negative.

How to efficiently calculate balance?

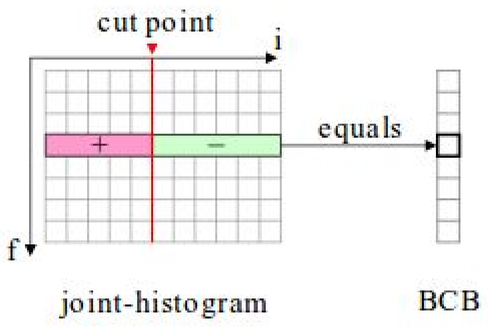
For efficiently calculating balance we make use of Balance counting box (BCB) which stores the difference of number of pixels on either sides of cut points. In mathematical form it can be represented as

$$\begin{aligned} B(f) = \#\{q \in \mathcal{R}(p) | I(q) <= c, \mathbf{f}(q) = \mathbf{f}_f\} \\ -\#\{r \in \mathcal{R}(p) | I(r) > c, \mathbf{f}(r) = \mathbf{f}_f\}, \end{aligned}$$

Now as balance is difference between the weights on both the sides of cut point we can easily calculate balance using BCB using the equation below

$$b = \sum_{f=0}^{N_f-1} B(f)\, g(\mathbf{f}_f, \mathbf{f}(p)).$$

BCB Diagram



BCB updation takes place almost like joint histogram updation where whenever a cut point shifts corresponding BCB boxes are changed depending on number of pixels on each side of that cut point. Similarly when a window shifts leaving points counts are reduced from each bin of BCB and entering ones are added.

Necklace Table

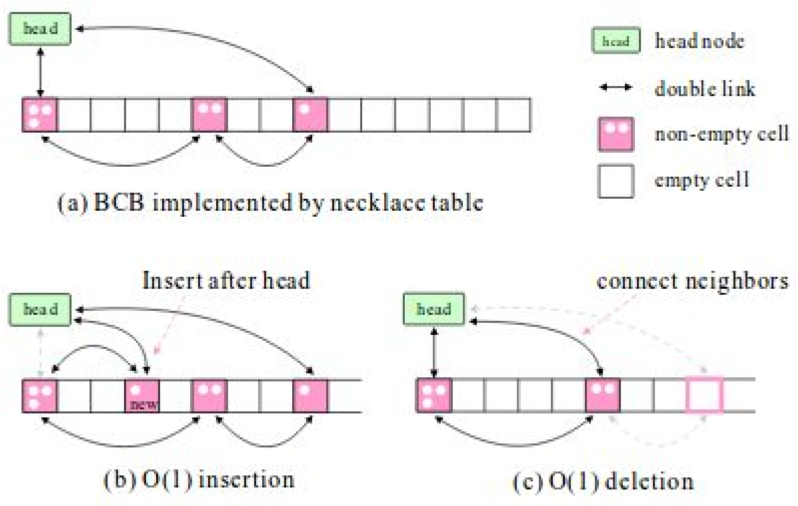
This data structure is made to exploit the data sparsity in joint histogram and BCB and reduces the required space as well as time required to access a particular value. This data structure provides following features:

- O(1) data access
- O(1) element insertion
- O(1) element deletion
- O(N) traversal where N in non empty cells number

Necklace Table consists of two components

- Counting array: Can be accessed by index
- Necklace: which is unordered doubly linked list built on counting array for non empty cells

Insertion and deletions into the necklace is self explanatory from the below diagram



Unlike traditional linked-list that keeps an order for data access, our necklace is unordered thanks to the available array. It allows for very fast traversal by skipping all empty cells. This data structure is used to implement our BCB and all columns of the joint-histogram.

Results (success)

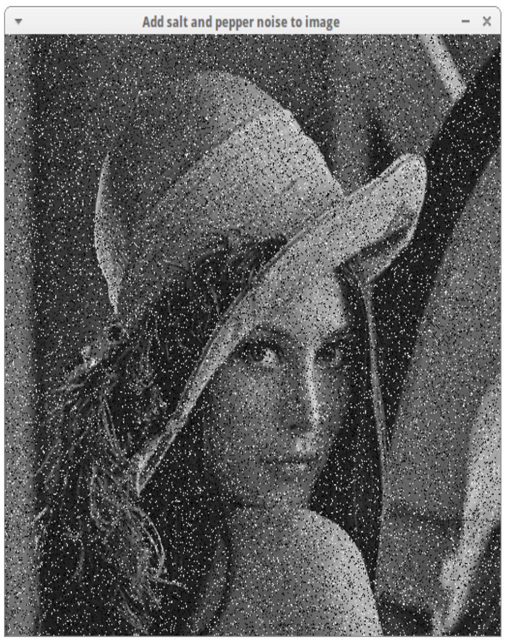
For the results we mainly focused on two types median filters

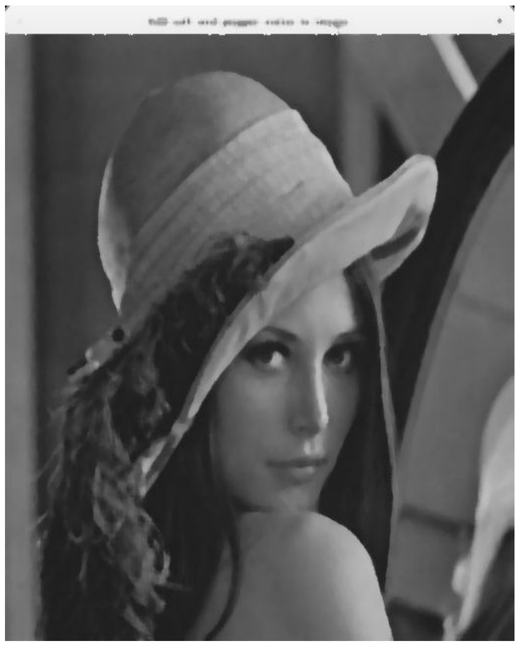
- Unweighted median filter
- Exponential (Gaussian)   weighted median filter

For better analysis of results over the possible domains we hyper parameterised the implementation. The domain/factor taken into account are : time , radius of kernel , size of image , type of weighted kernel used and much obvious, the output accuracy with the original algorithm.

Some basic implementations/comparisons are shown below:

Use Case 1: Removing salt/pepper noise



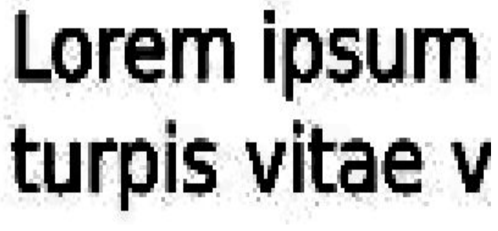


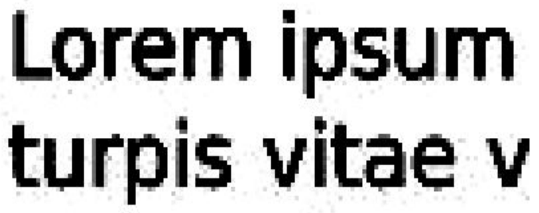
Unweighted median to remove basic salt/pepper noise from our image

Use Case 2: Removing JPEG artifacts

Lorem ipsum
turpis vitae v

Original Image with jpeg artifacts





Removal with unweighted, radius =3

Removal with unweighted, radius =5

A better example

OH NOES!!1!
JPEG
COMPRESSION!

OH NOES!!1!
JPEG
COMPRESSION!

Experiments/Analytics

Note: The time taken includes only applying filter not displaying

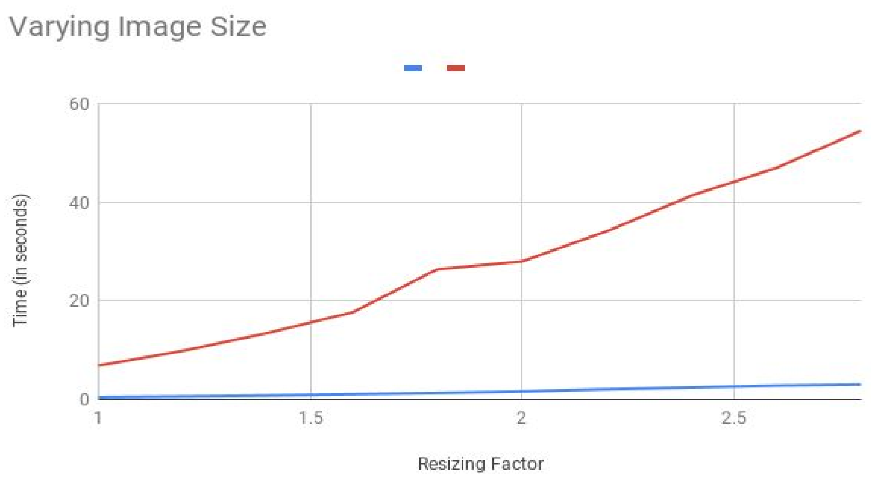
Varying Image Size

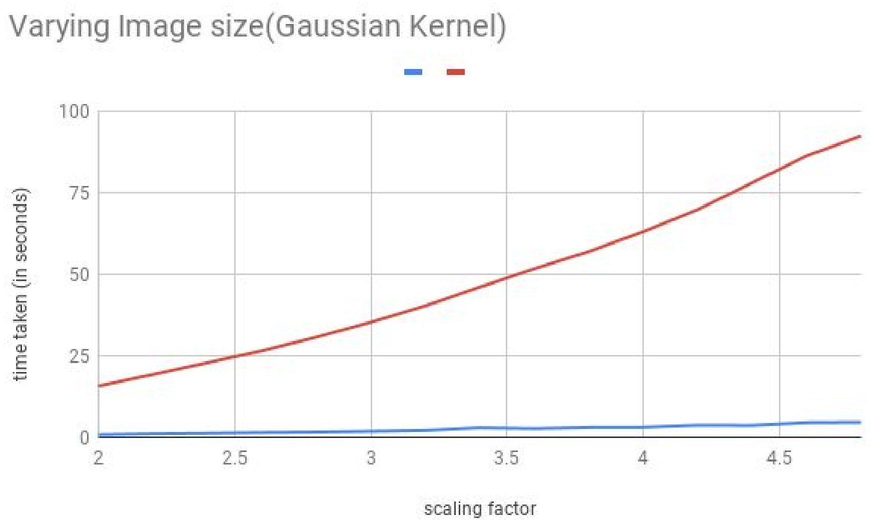
We increased the size of image and plotted time taken graph across various sizes.

Image Taken

Initial Size: *850X850 px*

Plot obtained



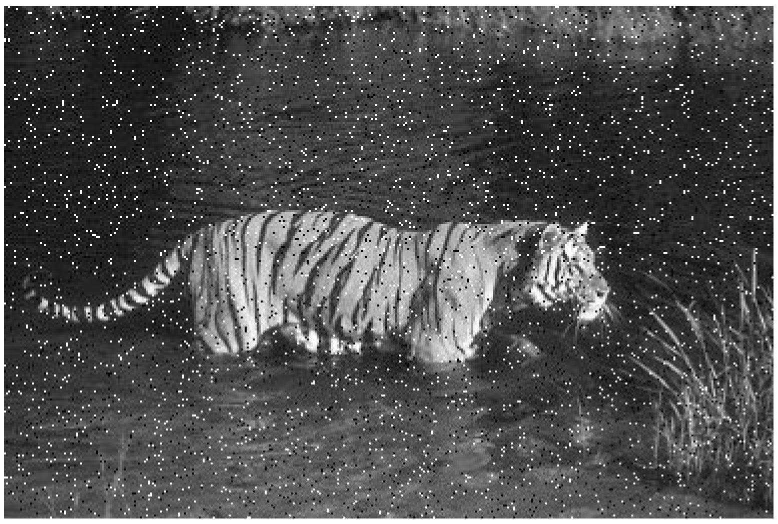


Clearly from the graph above as we vary the scaling factor the scaling factor the naive implementation results in a drastic increase in the time required for unweighted median on a constant window size of 3 radius for the kernel. The variations in 'our' implementation resulted in nearly in a constant time over varying sizes in a range, however if the image size is greatly increased along with window size then the change over time for our implementation is also visible.

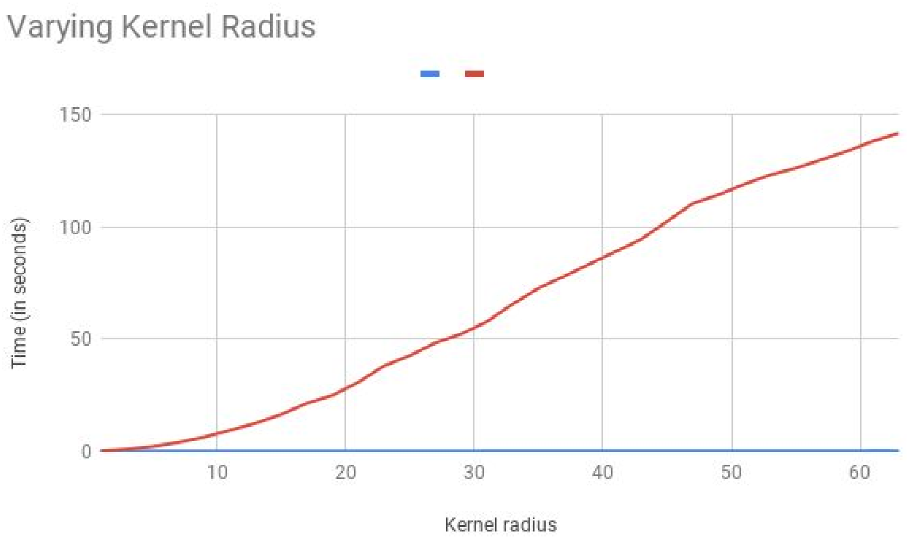
A similar case can be seen here as well, for the weighted kernel implementation the extra time added is corresponding to the calculation of the exponential terms.

Varying Kernel Radius

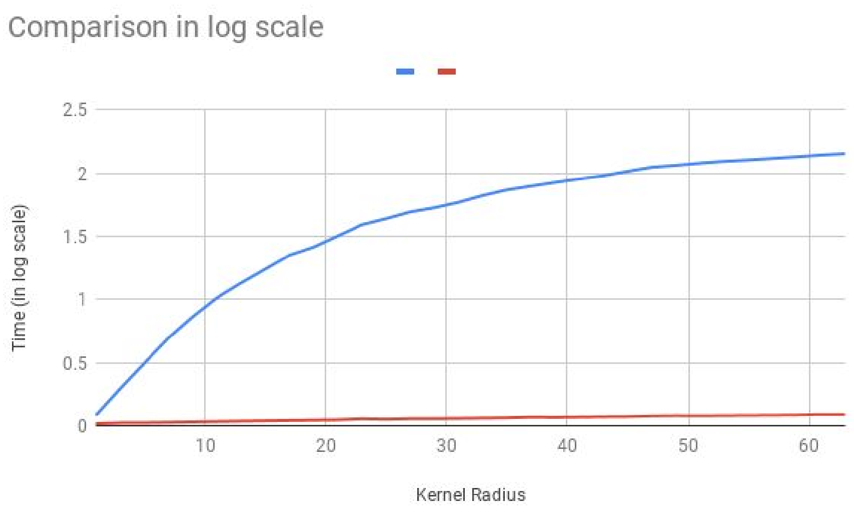
Input Image



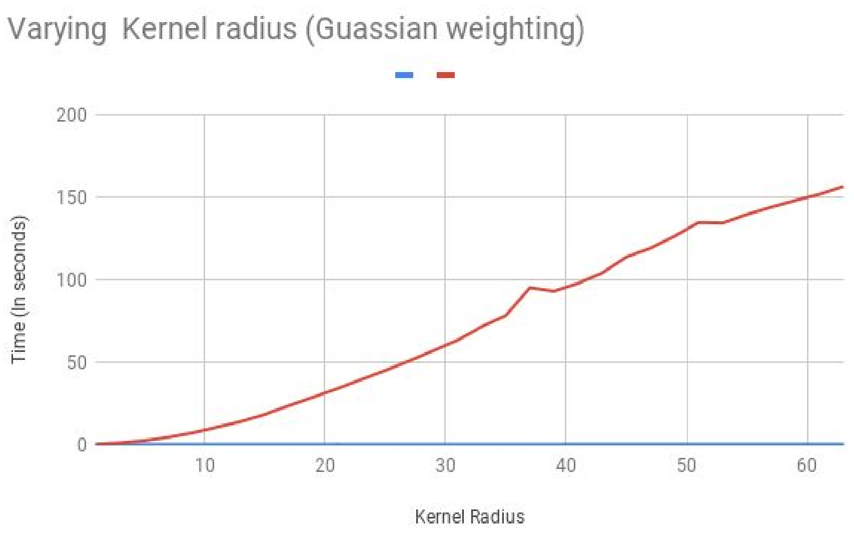
Graphs Obtained



Much similar to the above observations we observed a similar relation between increasing kernel radius and the time taken by the algorithm. On comparison scale our implementation is taking is so less that the blue nearly appears similar to the x-axis, however the approach is not constant time. Below graph shows varying kernel radius and log of time scale for better understanding



A similar case can be seen here as well, for the weighted kernel implementation the extra time added is corresponding to the calculation of the exponential terms.

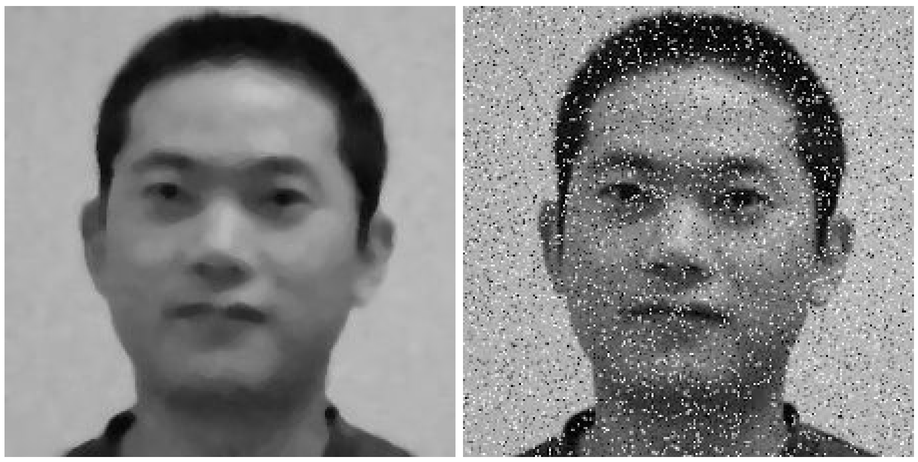


Note: Exact values for these points can be seen from csv files uploaded on github.

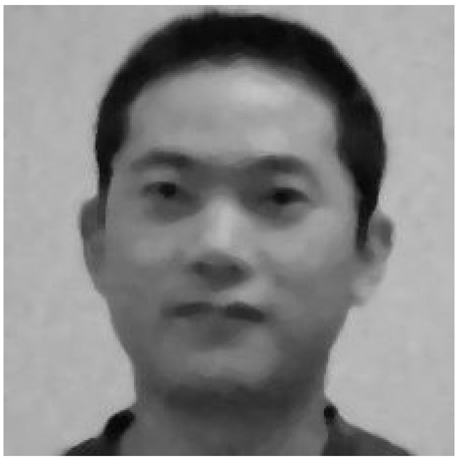
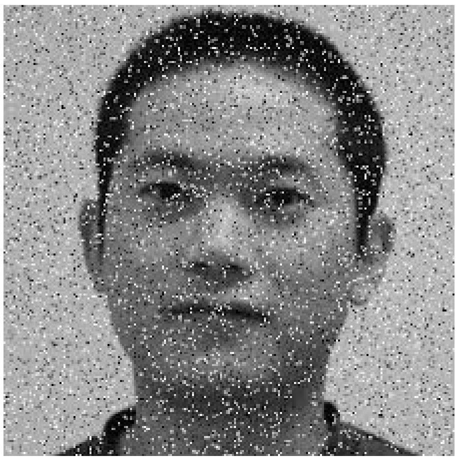
Accuracy

Accounting the accuracy of the code, the output image (most of times except one case we encountered) matches the output of original algorithm upto 3 floating point spaces.

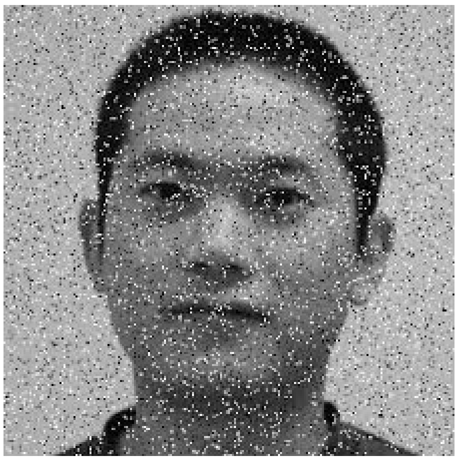
Result Obtained



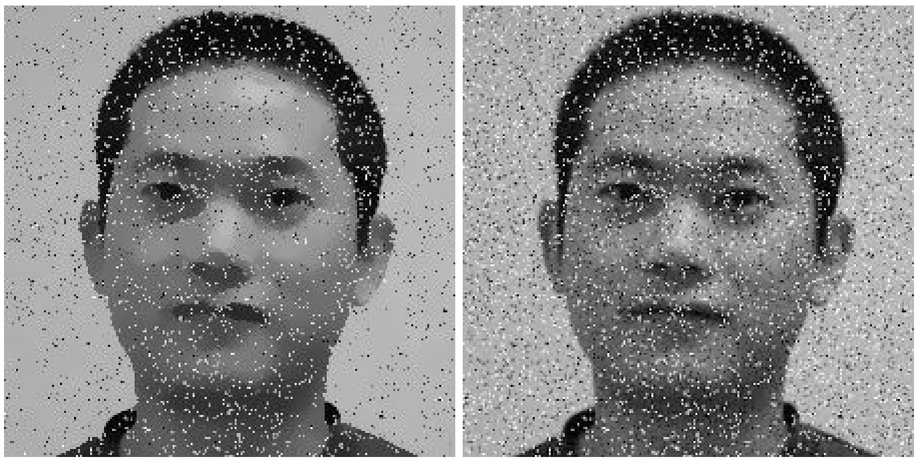
Output from ibuilt implementation(unweighted)

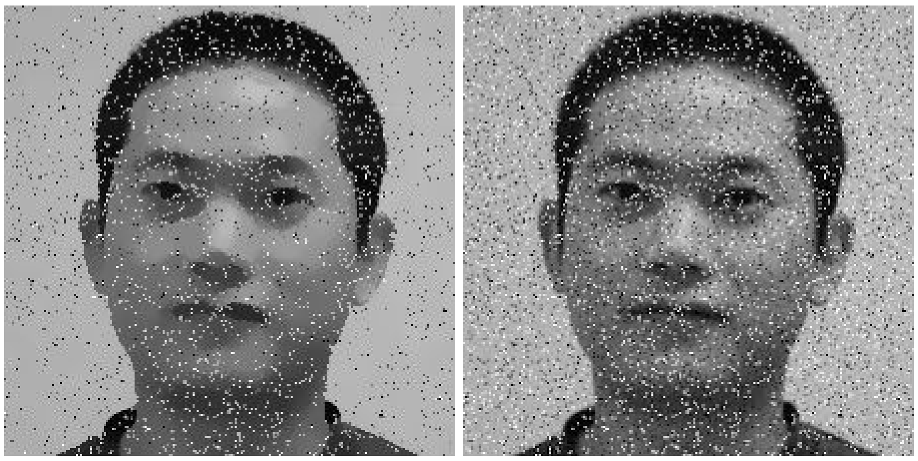
Output from our implementation(unweighted)



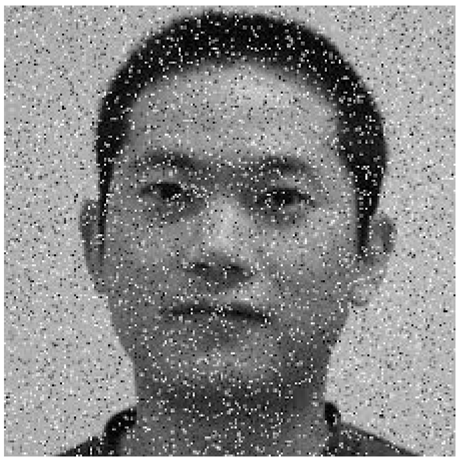
Effects of varying the kernel radius (unweighted)



Output from original filter (weighted gaussian kernel radius 5)



Output from our implementation(weighted gaussian kernel radius 5)



Effects of varying the kernel radius (weighted gaussian)

Task Assessment

Note: Opposed to as mentioned in proposal, we implemented joint histogram using necklace table in first go only as first implementing without it would be a waste of time

There were two major tasks only i.e development of **Joint Histogram** and development of **Balance Counting Box**. (Both using Necklace Tables). As both of these segments were highly related to each other, there was equal contribution of both of us to each of these tasks. Debugging codes was also the major tasks which was also performed by both of us for hours :(

Contribution Percentage (to complete project)

Kanav Gupta - 50%

Anurag Mehta - 50%

Acknowledgements/References

- http://www.cse.cuhk.edu.hk/leojia/projects/fastwmedian/download/fastweightedmedian_cvpr14.pdf
- https://en.wikipedia.org/wiki/Weighted_median
- http://www.cs.tut.fi/~moncef/publications/fast-algorithms-for-analyzing.pdf
- https://ieeexplore.ieee.org/document/6193457?arnumber=6193457