Project 3: Gradient-Domain Fusion

Points Estimate:
I am expecting a total of 100 Points for project 3.
Part 1: Toy Problem 20 Points
- Successfully reconstructed the image with give 2 constraints

Part 2: Poisson Blending 50 Points
- Successfully presented favorite results, not-so good results and a good result. Added explanation wherever is needed

Part 3: Mix Gradient Blending 20 Points
- Successfully presented results along with poisson and mix-gradient results. Added explanation wherever is needed

Finally expecting 10 Points from the quality of the results.

Bells and Whistles:
I am expecting a total of 40 Points from Bells and Whistles

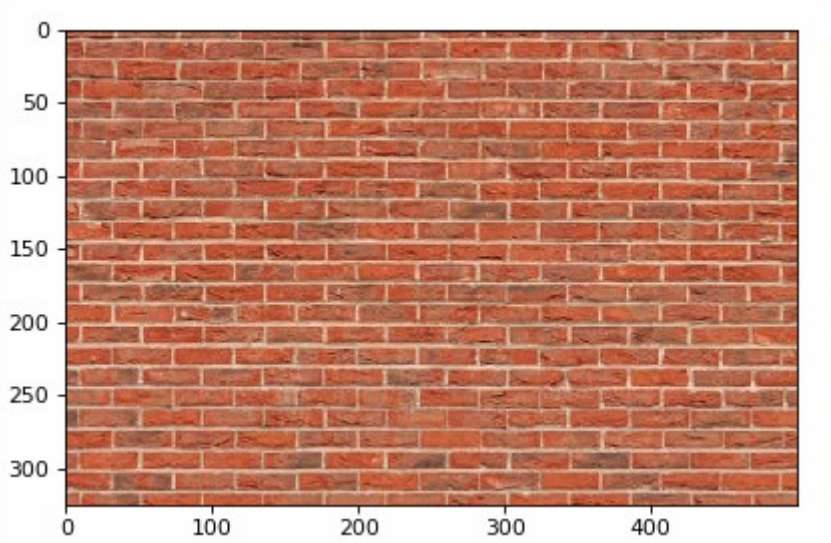
20 Points for Color2Gray
- Presented the Images and explanation behind them.

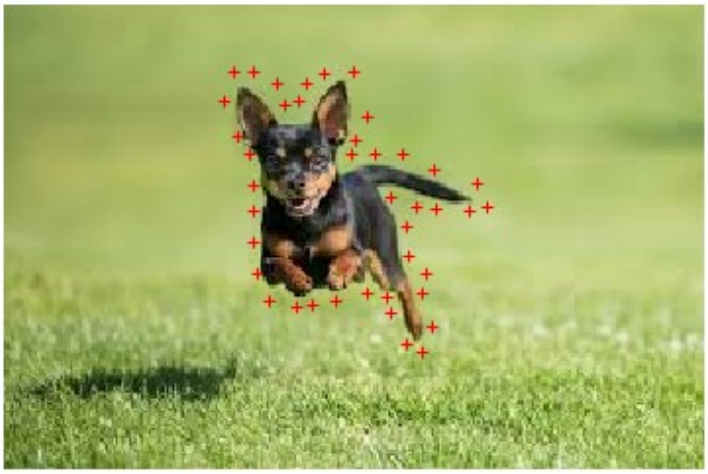
20 Points for Laplacian Blending
- Presented Images for laplacian blending along with poisson and mixed-gradient blends. Added explanation behind the implementation

1. (Favorite)

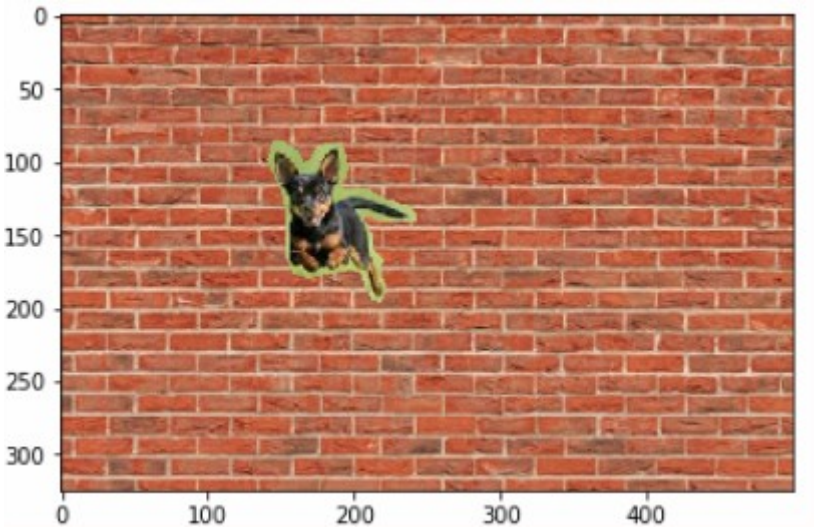
In this results, we can clearly see the the impact of poisson, mixed-gradient and laplacian pyramid blending results. Hence it is my favorite result.
In the mixed-gradient results the wall lines are clearly visible giving us an impression of a painting drawn on the wall.

Source Image and Target Image

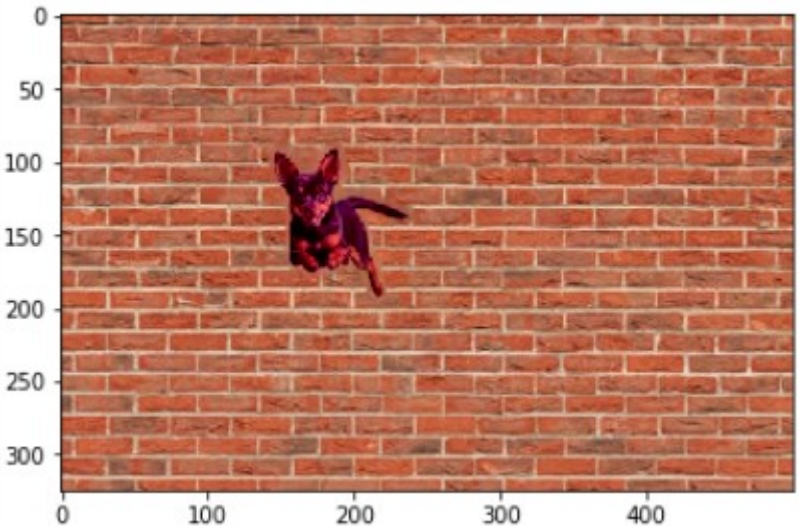




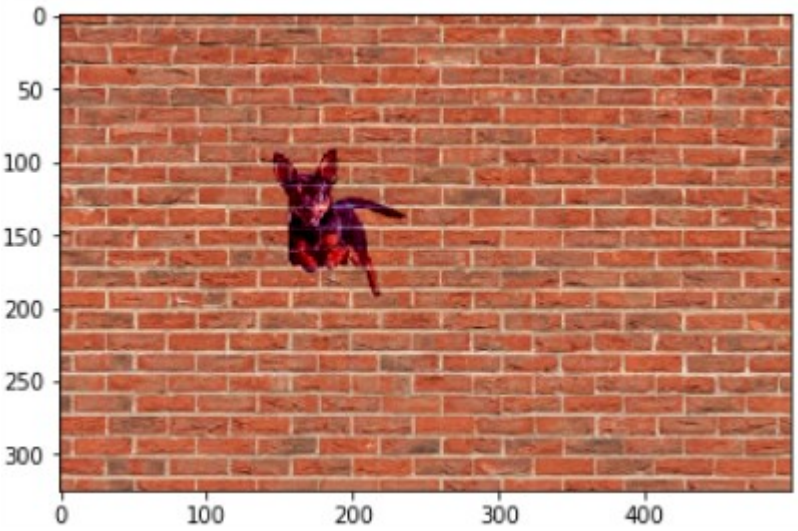
Blended Image with source pixels directly copied in to the target region



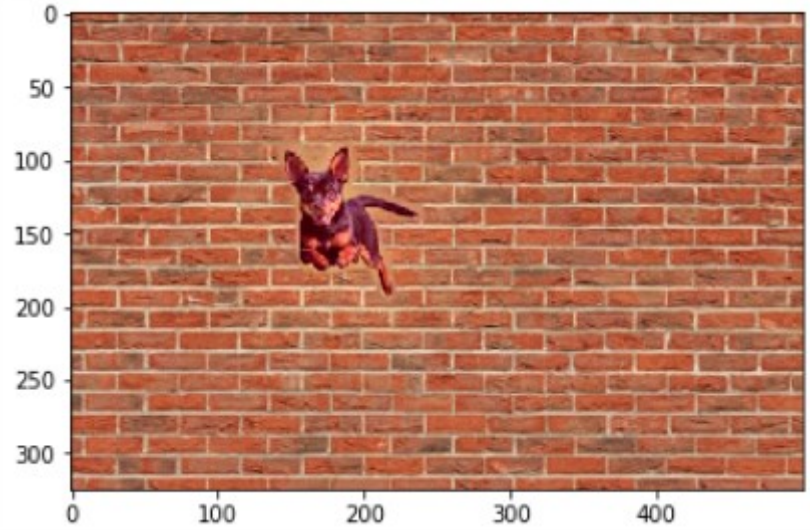
Poisson Blended Result



Mixed Blended Result



Laplacian Blended Result

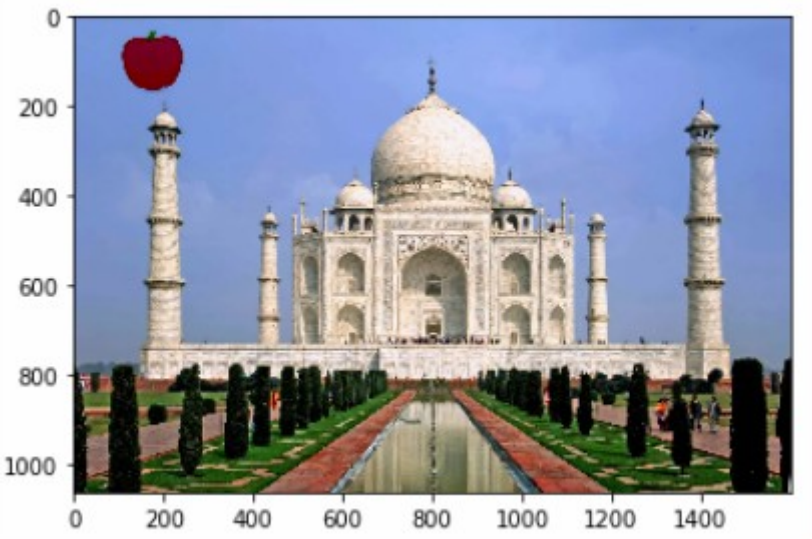


Poisson Blending (50 Points)

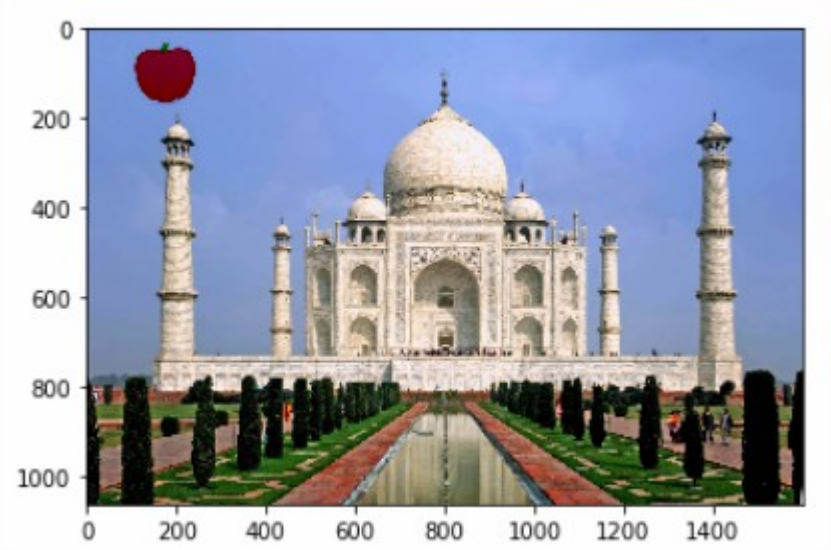
Source Image and Target Image

Blended Image with source pixels directly copied in to the target region

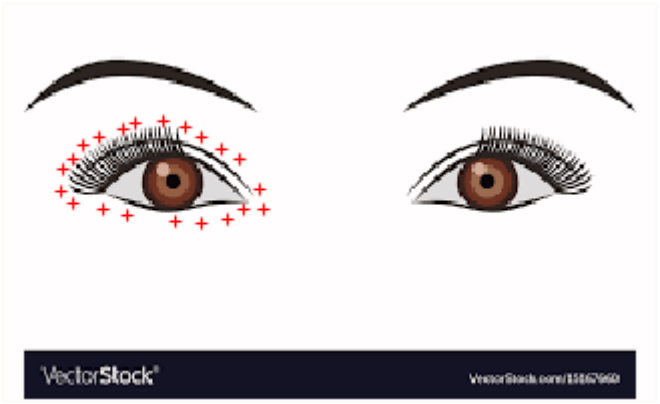
Poisson Blended Result



Mixed Gradients Results



<u>Not so good results:</u>
Source and Target Image



Blended Image with source pixels directly copied in to the target region

Poisson Blending

Mixed Gradients

Laplacian Blending

Explanation on the Implementation:

Toy Problem

Even though no images of Toy problem are added in this report, the project report on the jupyter notebook has the images. Based on the constraints mentioned in the project description. I used lil_matrix and csr_matrix for speed in the processing of the images.
The reconstruction error I got is as below

```
Error is:  0.0001901762385198699
```

Poisson Blending

Incase of Poisson blending I have used 4 constraints along the lines of (y,x+1), (y+1, x), (y,x-1) and (y-1,x). I use the same code as in toy problem but on 3 channels. Reconstructed the final image along the 3 channels using cv2.add(). I used the following logic while solving for the constraints.

$$\mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - (s_i - s_j))^2 + \sum_{i \in S, j \in N_i \cap \neg S} ((v_i - t_j) - (s_i - s_j))^2$$

Mixed Gradient Blending

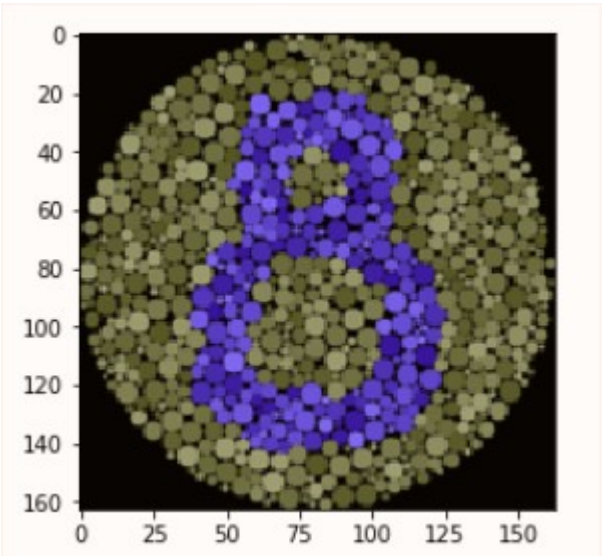
In case of Mixed gradient blending, I pretty much used the same code from toy problem and poisson blending. Since the equation to be follow is shown as below

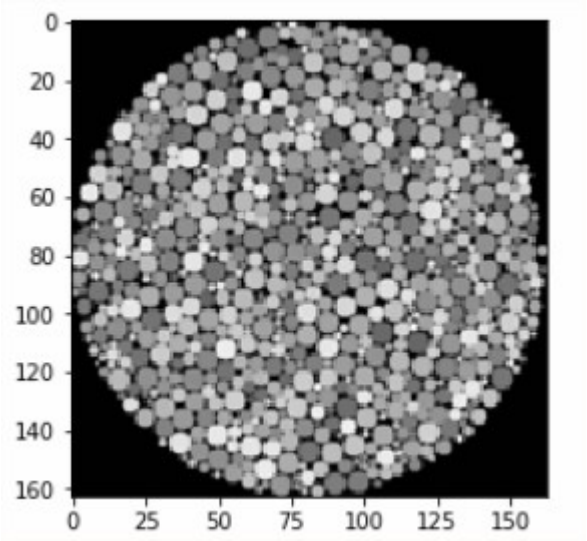
$$\mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - d_{ij})^2 + \sum_{i \in S, j \in N_i \cap \neg S} ((v_i - t_j) - d_{ij})^2$$

I introduced an api to calculate the dij value.

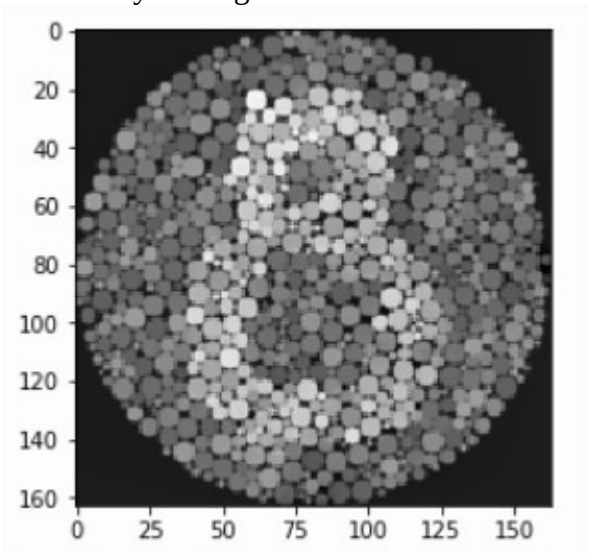
```
def dij(si, sj, ti, tj):
#b[e] = im[y][x][ch] - im[y][x-1][ch] + background_img[y][x-1][ch]
#b[e] = im[y][x][ch] - im[y][x-1][ch]
    if (abs(si - sj) > abs(ti - tj)):
        return (si - sj)
    else:
        return (ti - tj)
```

Bells & Whistles
Color2Gray (20 Points)
Source Image and Target Image





Color2Gray'd Image



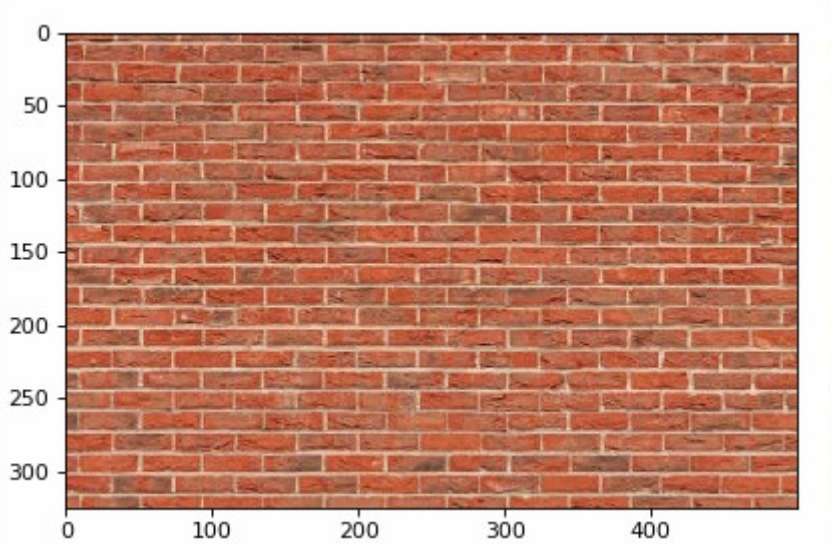
Explanation:
We follow the same procedure done for toy reconstruction problem but instead of the taking the background image directly, we take the gradient_norm value. I configured the gradient_norm value as the maximum value in any of the three channels in the image and assign that value. Hence we can see the gradient differentiation in the final image.
Once notable variation from the toy problem in this color2gray code is as shown below.

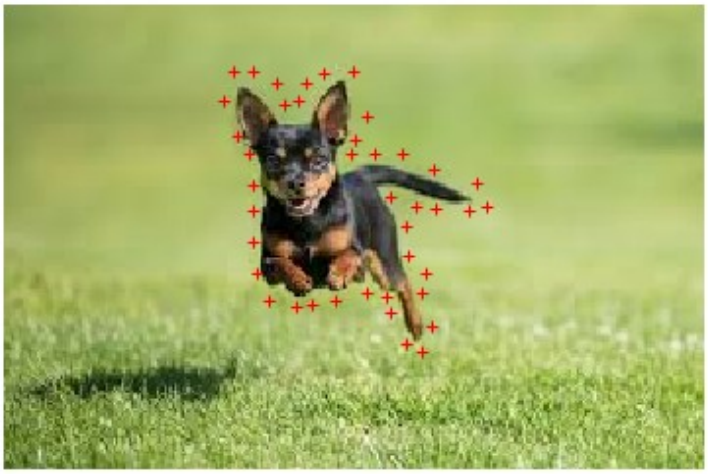
A[e, im2var[y][x+1]] = -1
A[e, im2var[y][x]] = 1

I interchange the -1 and 1 values from the toy problem for better visibility.

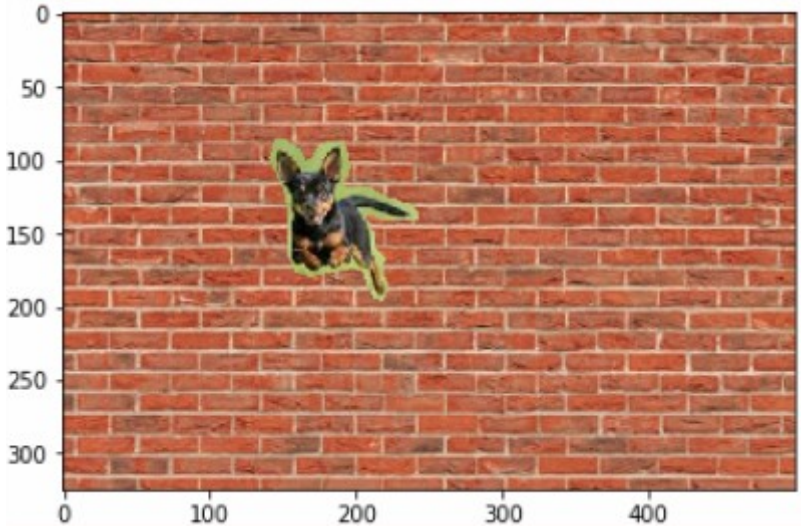
```
#taking the max among the 3 color channels solved it for me.
def gradient_norm(si, sj):
    max = 0
    for ch in range(3):
        if abs(si[ch] - sj[ch]) > abs(max):
            max = si[ch] - sj[ch]
    return max
```

Laplacian Pyramid Blending (20 Points)
Source Image and Target Image

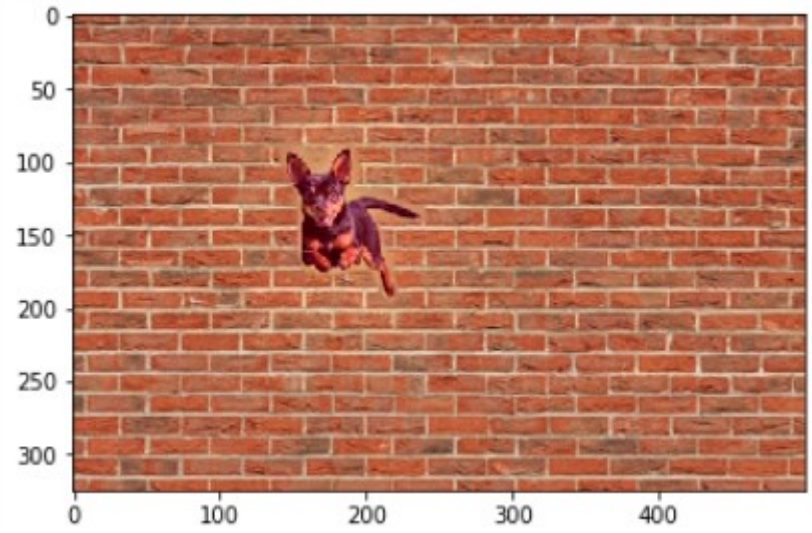




Blended Image with source pixels directly copied in to the target region



Laplacian Blended Image:



Explanation:
Below is the explanation of the code implementation.
1. We load the cropped image(img1) and the background image(img2) and the mask from the earlier blending.
2. We find the gaussian pyramid of img1 and img2
3. We find the laplacian pyramids of the gaussian pyramids
4. We use alpha matte transition on the img1 and img2 and the mask based on below logic
    (img1) *alpha + (img2) * (1-alpha)
5. We reconstruct the image at all levels of the pyramids.

For test purposes I used different values for levels ranging from 3 to 121. I could notice the difference in the final picture.