R05922096 資工所 碩 1 李哲安

Computer Vision Hw8 Report

- Discription
    - Noise Removal
- Algorithm
    - Box filter (3x3, 5x5)
    - Median filter (3x3, 5x5)
    - Opening then Closing
- Parameters (if any)
    - no

- Principal Code Fragment

  Main (file – /src/hw8/ DemoNoiseRemoval.java)

```
// generate noise
System.out.println("generate noise image ...");
BufferedImage noise_gaussian_10 = NoiseGenerator.generateGaussianNoise(lena, 10);
BufferedImage noise_gaussian_30 = NoiseGenerator.generateGaussianNoise(lena, 30);
BufferedImage noise_pepper_005 = NoiseGenerator.generateSaltAndPepperNoise(lena, 0.05);
BufferedImage noise_pepper_01 = NoiseGenerator.generateSaltAndPepperNoise(lena, 0.1);

System.out.printf("noise_gaussian_10 SNR: %f\n",NoiseGenerator.signalToNoiseRatio(lena, noise_gaussian_10));
System.out.printf("noise_gaussian_30 SNR: %f\n",NoiseGenerator.signalToNoiseRatio(lena, noise_gaussian_30));
System.out.printf("noise_pepper_005 SNR: %f\n",NoiseGenerator.signalToNoiseRatio(lena, noise_pepper_005));
System.out.printf("noise_pepper_01 SNR: %f\n",NoiseGenerator.signalToNoiseRatio(lena, noise_pepper_01));
```

```
// noise remove box filter 3x3
System.out.println("remove noise by 3x3 box filter ...");
BufferedImage rn_b3x3_g10 = NoiseRemover.boxFilter(noise_gaussian_10, NoiseRemover.FILTER_3x3);
BufferedImage rn_b3x3_g30 = NoiseRemover.boxFilter(noise_gaussian_30, NoiseRemover.FILTER_3x3);
BufferedImage rn_b3x3_p005 = NoiseRemover.boxFilter(noise_pepper_005, NoiseRemover.FILTER_3x3);
BufferedImage rn_b3x3_p01 = NoiseRemover.boxFilter(noise_pepper_01, NoiseRemover.FILTER_3x3);

// noise remove box filter 5x5
System.out.println("remove noise by 5x5 box filter ...");
BufferedImage rn_b5x5_g10 = NoiseRemover.boxFilter(noise_gaussian_10, NoiseRemover.FILTER_5x5);
BufferedImage rn_b5x5_g30 = NoiseRemover.boxFilter(noise_gaussian_30, NoiseRemover.FILTER_5x5);
BufferedImage rn_b5x5_p005 = NoiseRemover.boxFilter(noise_pepper_005, NoiseRemover.FILTER_5x5);
BufferedImage rn_b5x5_p01 = NoiseRemover.boxFilter(noise_pepper_01, NoiseRemover.FILTER_5x5);
// noise remove median filter 3x3
System.out.println("remove noise by 3x3 median filter ...");
BufferedImage rn_m3x3_g10 = NoiseRemover.medianFilter(noise_gaussian_10, NoiseRemover.FILTER_3x3);
BufferedImage rn_m3x3_g30 = NoiseRemover.medianFilter(noise_gaussian_30, NoiseRemover.FILTER_3x3);
BufferedImage rn_m3x3_p005 = NoiseRemover.medianFilter(noise_pepper_005, NoiseRemover.FILTER_3x3);
BufferedImage rn_m3x3_p01 = NoiseRemover.medianFilter(noise_pepper_01, NoiseRemover.FILTER_3x3);

// noise remove median filter 5x5
System.out.println("remove noise by 5x5 median filter ...");
BufferedImage rn_m5x5_g10 = NoiseRemover.medianFilter(noise_gaussian_10, NoiseRemover.FILTER_5x5);
BufferedImage rn_m5x5_g30 = NoiseRemover.medianFilter(noise_gaussian_30, NoiseRemover.FILTER_5x5);
BufferedImage rn_m5x5_p005 = NoiseRemover.medianFilter(noise_pepper_005, NoiseRemover.FILTER_5x5);
BufferedImage rn_m5x5_p01 = NoiseRemover.medianFilter(noise_pepper_01, NoiseRemover.FILTER_5x5);
// opening the closing
System.out.println("remove noise by opening then closing ...");
BufferedImage rn_opcls_g10 = GrayLevelMorphology.opening(noise_gaussian_10, "3-5-5-5-3", 0);
rn_opcls_g10 = GrayLevelMorphology.closing(rn_opcls_g10, "3-5-5-5-3", 0);
BufferedImage rn_opcls_g30 = GrayLevelMorphology.opening(noise_gaussian_30, "3-5-5-5-3", 0);
rn_opcls_g30 = GrayLevelMorphology.closing(rn_opcls_g30, "3-5-5-5-3", 0);
BufferedImage rn_opcls_p005 = GrayLevelMorphology.opening(noise_pepper_005, "3-5-5-5-3", 0);
rn_opcls_p005 = GrayLevelMorphology.closing(rn_opcls_p005, "3-5-5-5-3", 0);
BufferedImage rn_opcls_p01 = GrayLevelMorphology.opening(noise_pepper_01, "3-5-5-5-3", 0);
rn_opcls_p01 = GrayLevelMorphology.closing(rn_opcls_p01, "3-5-5-5-3", 0);
```

NoiseGenerator

(file – /src/cv1.util.cv.noise /NoiseGenerator.java)

```java
public static BufferedImage generateGaussianNoise(BufferedImage bi, double amplitude){
    Random random = new Random();
    BufferedImage result = new BufferedImage(bi.getWidth(), bi.getHeight(), bi.getType());
    for (int y = 0; y < bi.getHeight(); y++) {
        for (int x = 0; x < bi.getWidth() ; x++) {
            /* old rgb */
            int rgb = bi.getRGB(x, y);
            int gray = rgb&0xff;

            /* noise */
            int noise = (int)(amplitude*random.nextGaussian());

            /* new RGB */
            int newGray = gray + noise;
            newGray = newGray > 255 ? 255 : newGray;
            newGray = newGray < 0 ? 0 : newGray;
            int newRGB = 0xff000000 + (newGray<<16)+ (newGray<<8)+ (newGray);
            result.setRGB(x , y , newRGB);
        }
    }
    return result;
}
public static BufferedImage generateSaltAndPepperNoise(BufferedImage bi, double threshold){
    Random random = new Random();
    BufferedImage result = new BufferedImage(bi.getWidth(), bi.getHeight(), bi.getType());
    for (int y = 0; y < bi.getHeight(); y++) {
        for (int x = 0; x < bi.getWidth() ; x++) {
            /* old rgb */
            int rgb = bi.getRGB(x, y);
            int gray = rgb&0xff;

            /* noise */
            double noise = random.nextDouble();

            /* new RGB */
            int newGray = gray;
            newGray = noise > 1 - threshold/2 ? 255 : newGray;
            newGray = noise < threshold/2 ? 0 : newGray;
            int newRGB = 0xff000000 + (newGray<<16)+ (newGray<<8)+ (newGray);
            result.setRGB(x , y , newRGB);
        }
    }
    return result;
}
```

NoiseRemover

(file – /src/cv1.util.cv.noise /NoiseRemover.java)

```
public static BufferedImage boxFilter(BufferedImage bi, int filterType){
    BufferedImage result = new BufferedImage(bi.getWidth(), bi.getHeight(), bi.getType());
    Filter filter = new Filter(filterType);
    for (int y = 0; y < bi.getHeight(); y++) {
        for (int x = 0; x < bi.getWidth() ; x++) {
            int totalWeight = 0;
            int newGray = 0;
            for (Logic logic : filter.logics) {
                try{
                    /* old rgb */
                    int rgb = bi.getRGB(x + logic.x, y + logic.y);
                    int gray = rgb&0xff;

                    /* filter */
                    totalWeight += logic.w;
                    newGray += gray*logic.w;
                }catch (Exception e) {
                    // TODO: handle exception
                }
            }
            newGray /= totalWeight;
            int newRGB = 0xff000000 + (newGray<<16) + (newGray<<8) + (newGray);
            result.setRGB(x, y, newRGB);
        }
    }
    return result;
}
```

```
public static BufferedImage medianFilter(BufferedImage bi, int filterType){
    BufferedImage result = new BufferedImage(bi.getWidth(), bi.getHeight(), bi.getType());
    Filter filter = new Filter(filterType);
    for (int y = 0; y < bi.getHeight(); y++) {
        for (int x = 0; x < bi.getWidth() ; x++) {
            ArrayList<Integer> neighborhoodGray = new ArrayList<Integer>();
            for (Logic logic : filter.logics) {
                try{
                    /* old rgb */
                    int rgb = bi.getRGB(x + logic.x, y + logic.y);
                    int gray = rgb&0xff;
                    /* filter */
                    neighborhoodGray.add(gray);
                }catch (Exception e) {/* Ignore */}
            }
            neighborhoodGray.sort(new Comparator<Integer>() {
                @Override public int compare(Integer o1, Integer o2) {return o1.compareTo(o2);}
            });
            int medianIdx = neighborhoodGray.size()/2;
            int medianGray = neighborhoodGray.get(medianIdx);
            if (neighborhoodGray.size()%2 == 0)medianGray = (medianGray + neighborhoodGray.get(medianIdx+1))/2;
            int newRGB = 0xff000000 + (medianGray<<16) + (medianGray<<8) + (medianGray);
            result.setRGB(x, y, newRGB);
        }
    }
    return result;
```

- Result Image

Noise (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
noise_gaussian_10 SNR: 13.935227
noise_gaussian_30 SNR: 4.272406
noise_pepper_005 SNR: 3.948180
noise_pepper_01 SNR: 0.901626
```

3x3 Box (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
rn_b3x3_g10 SNR: 17.833155
rn_b3x3_g30 SNR: 12.688300
rn_b3x3_p005 SNR: 12.192854
rn_b3x3_p01 SNR: 9.447392
```

5x5 Box (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
rn_b5x5_g10 SNR: 14.876078
rn_b5x5_g30 SNR: 13.325202
rn_b5x5_p005 SNR: 12.875964
rn_b5x5_p01 SNR: 11.156421
```

3x3 Median (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
rn_m3x3_g10 SNR: 17.898592
rn_m3x3_g30 SNR: 11.275927
rn_m3x3_p005 SNR: 20.216775
rn_m3x3_p01 SNR: 18.992429
```

5x5 Median (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
rn_m5x5_g10 SNR: 16.076362
rn_m5x5_g30 SNR: 13.067978
rn_m5x5_p005 SNR: 16.630589
rn_m5x5_p01 SNR: 16.407275
```

Open-Closing (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

SNR (Gaussian-10, Gaussian-30, S&P-0.05, S&P-0.1)

```
rn_opcls_g10 SNR: 13.256950
rn_opcls_g30 SNR: 11.210615
rn_opcls_p005 SNR: 11.051723
rn_opcls_p01 SNR: 5.236337
```