CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

This project use java programming, and use 2 method for image enlargement, that is Bilinear Interpolation and Bicubic Interpolation with different enlargement scale (2x, 4x, 6x). and as a parameter to see the quality of ouput image use PSNR. The data are used 30 images, and downsized based on the enlargement scale using the tool (paint), after that the downsize image is enlarged with a scale 2x, 4x, 6x. After getting result image, the PSNR value is calculated to find out which image quality is better than 2 interpolation method performed.

Below is the code of Bilinear Interpolation:

```
1. private static double lerp(double a, double b, double t)
2.  { return (1 - t) * a + t * b;}
3.
4. private static BufferedImage scale(BufferedImage img, double
   scale) {
5.    long start = System.currentTimeMillis();
6.    int newWidth = (int)(img.getWidth() * scale);
7.    int newHeight = (int)(img.getHeight() * scale);
8.
9.    BufferedImage newImage = new BufferedImage(newWidth,
         newHeight, img.getType());
10.
11.      for (int y = 0; y < newHeight; y++) {
12.       for (int x = 0; x < newWidth; x++) {
13.            double yImgReal = (double)y / newHeight *
                 (img.getHeight() - 1);
14.            double xImgReal = (double)x / newWidth *
                 (img.getWidth() - 1);
15.
16.              int yImgInt = (int)yImgReal;
17.              int xImgInt = (int)xImgReal;
18.
19.              double xt = xImgReal - xImgInt;
20.              double yt = yImgReal - yImgInt;
21.
22.       Pixel q12 = new Pixel(img.getRGB(xImgInt, yImgInt));
23.       Pixel q22 = new Pixel(img.getRGB(xImgInt + 1, yImgInt));
```

```
24.       Pixel q11 = new Pixel(img.getRGB(xImgInt, yImgInt + 1));
      Pixel q21 = new Pixel(img.getRGB(xImgInt + 1, yImgInt + 1));
25.
26.                     Pixel p = new Pixel(0);
27.
28.                     for (int i = 0; i < 4; i++) {
29.                         int i12 = q12.ARGB[i];
30.                         int i22 = q22.ARGB[i];
31.                         int i11 = q11.ARGB[i];
32.                         int i21 = q21.ARGB[i];
33.
34.                         double r1 = lerp(i11, i21, xt);
35.                         double r2 = lerp(i12, i22, xt);
36.
37.                         p.ARGB[i] = (int)lerp(r2, r1, yt);
38.
39.                     }
40.
41.                     newImage.setRGB(x, y, p.getRGB());
42.                 }
43.             }
```

This is the enlargement code with Bilinear interpolation. In line 1 is the linear interpolation equation. In line 6 and 7 to calculate the enlargement scale to be performed, multiplying the height and width of the image with the enlargement scale. Lines 11 to 25 are used to determine the operating point of the pixel calculation. After getting the calculation operation point. We count the interpolation in lines 34 to 37

Below is the code of Bicubic Interpolation :

```
44.     BufferedImage source = ImageIO.read(inputFile);
45.
46.     boolean hasAlpha = source.getColorModel().hasAlpha();
47.
48.           int sourceW = source.getWidth();
49.           int sourceH = source.getHeight();
50.
51.           int w = (int) (sourceW * scaleFactor + 0.5);
52.           int h = (int) (sourceH * scaleFactor + 0.5);
53.
54.     int format = hasAlpha ? BufferedImage.TYPE_INT_ARGB :
      BufferedImage.TYPE_INT_RGB;
55.
56.     BufferedImage output = new BufferedImage(w, h, format);
57.     Graphics2D g = output.createGraphics();
58.
```

```
59.      g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
         RenderingHints.VALUE_INTERPOLATION_BICUBIC);
60.      g.drawImage(source, 0, 0, w, h, null);
61.      writeImage(outputFile, output);
```

This is the enlargement code with Bicubic interpolation. In line 44 we get the input image. Line 48 to 49 used to get the height and width of the image, In line 51 and 52 to calculate the enlargement scale to be performed, multiplying the height and width of the image with the enlargement scale. Lines 56 to 60 are used for interpolation bicubic calculation.

This code is use for count the time process :

```
1. long start = System.currentTimeMillis();
2.
3.
4.
5. long end = System.currentTimeMillis();
6. NumberFormat formatter = new DecimalFormat("#0.00000");
7. System.out.println("Execution time is " + formatter.format((end
      - start) / 1000d) + " seconds");
```

Placed between the function code, to calculate the processing time needed.

Below is the code of PSNR :

```
62.      for(int i=0;i<height;i++)
63.         {
64.          for(int j=0;j<width;j++)
65.            {
66.            Color c = new Color(GambarAsli.getRGB(j,i));
67.            Color s = new Color(InterpolasiGambar.getRGB(j,i));
68.            double redc = (double)(c.getRed());
69.            double reds = (double)(s.getRed());
70.            double greenc = (double)(c.getGreen());
71.            double greens = (double)(s.getGreen());
72.            double bluec = (double)(c.getBlue());
73.            double blues = (double)(s.getBlue());
74.             if(redc>max_rc)
75.             {
76.               max_rc=redc;
77.             }
78.             if(greenc>max_gc)
79.             {
80.               max_gc=greenc;
81.             }
82.             if(bluec>max_bc)
83.             {
```

```
84.               max_bc=bluec;
85.             }
86.             if(reds>max_rs)
87.             {
88.               max_rs=reds;
89.             }
90.             if(greens>max_gs)
91.             {
92.               max_rs=reds;
93.             }
94.             if(blues>max_bs)
95.             {
96.               max_rs=reds;
97.             }
98.             mse_r=mse_r+Math.pow((reds-redc),2);
99.             mse_g=mse_g+Math.pow((greens-greenc),2);
100.            mse_b=mse_b+Math.pow((blues-bluec),2);
101.
102.
103.            mse_r = mse_r/(width*height);
104.            mse_g = mse_g/(width*height);
105.            mse_b = mse_b/(width*height);
106.            mse = (mse_r+mse_g+mse_b)/3;
107.
108.
109.    psnr_r = 10.0 * logbase10(Math.pow(Math.max
110.    (max_rc,max_rs), 2) / mse_r);
111.    psnr_g = 10.0 * logbase10(Math.pow(Math.max
112.    (max_gc,max_gs), 2) / mse_g);
113.    psnr_b = 10.0 * logbase10(Math.pow(Math.max
114.    (max_bc,max_bs), 2) / mse_b);
115.    psnr = (psnr_r+psnr_g+psnr_b)/3;
```

This is the PSNR code, to calculate the image quality In line 66 to 97 used to find pixe value (red,green,blue). After getting the RGB value, we first calculate the MSE in each color (red,green,blue) after that MSE result are divided by 3 in order to get the final MSE value. Likewise with the PSNR calculation, after getting the PSNR value in each color, the PSNR value resultsare divided by 3.

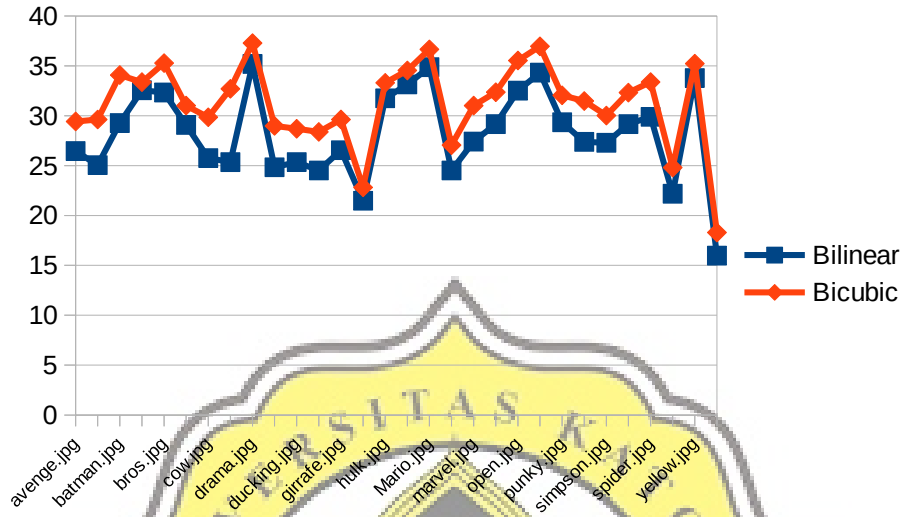
5.2 Testing

The Result of the images that have been downsized and the result of PSNR calculation sort by enlargement scale 2x, 4x, 6x. Displayed in Table 5.1 and graphic.

Table 5.1: Downsized Image Result

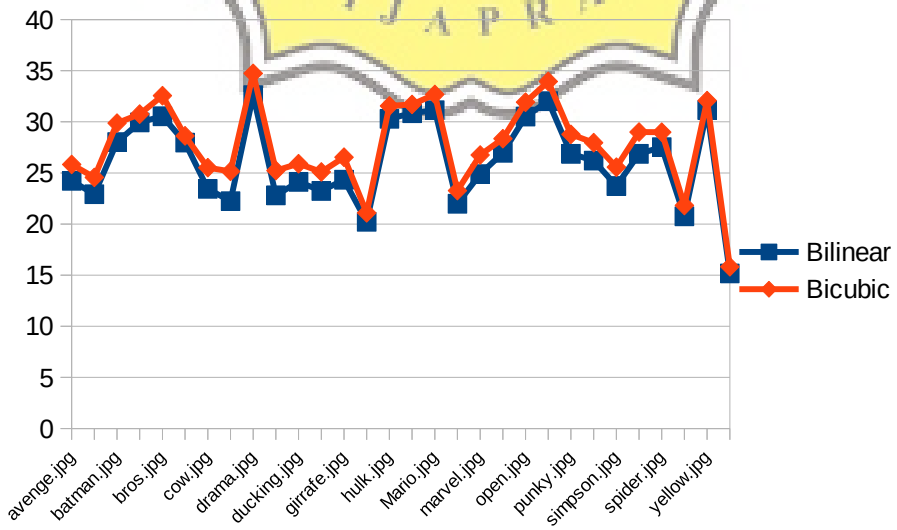
Image	Size / Result	Downsize		
		2x	4x	6x
Avenge.jpg	1920x1080	960x540	480x270	320x180
Babycow.jpg	1920x1080	960x540	480x270	320x180
Batman.jpg	1920x1080	960x540	480x270	320x180
Bowser.jpg	1920x1080	960x540	480x270	320x180
Bros.jpg	1920x1080	960x540	480x270	320x180
Bull.jpg	1200x900	600x450	300x225	200x150
Cat.jpg	1920x1080	960x540	480x270	320x180
Cow.jpg	900x600	450x300	225x150	150x100
Donald.jpg	1920x1080	960x540	480x270	320x180
Drama.jpg	1920x1080	960x540	480x270	320x180
Duck.jpg	900x600	450x300	225x150	150x100
Ducking.jpg	900x600	450x300	225x150	150x100
Flyduck.jpg	900x600	450x300	225x150	150x100
Girrafe.jpg	1920x1440	960x720	480x360	320x240
Gold.jpg	900x900	450x450	225x225	150x150
Hulk.jpg	1920x1200	960x600	480x300	320x200
Logo.jpg	1920x1080	960x540	480x270	320x180
Mario.jpg	1920x1080	960x540	480x270	320x180
Marv.jpg	1920x1080	960x540	480x270	320x180
Marvel.jpg	1920x1080	960x540	480x270	320x180
Peter.jpg	1920x1080	960x540	480x270	320x180
Open.jpg	1920x1080	960x540	480x270	320x180
Pony.jpg	1800x1200	900x600	450x300	300x200
Punky.jpg	1800x1200	900x600	450x300	300x200
Red.jpg	1080x600	540x300	270x150	180x100
Simpson.jpg	1920x1080	960x540	480x270	320x180
Smash.jpg	1920x1080	960x540	480x270	320x180
Spider.jpg	1920x1080	960x540	480x270	320x180
Sunset.jpg	480x360	240x180	120x90	80x60
Yellow.jpg	1980x1080	990x540	495x270	330x180

PSNR value at 2x enlargement scale :



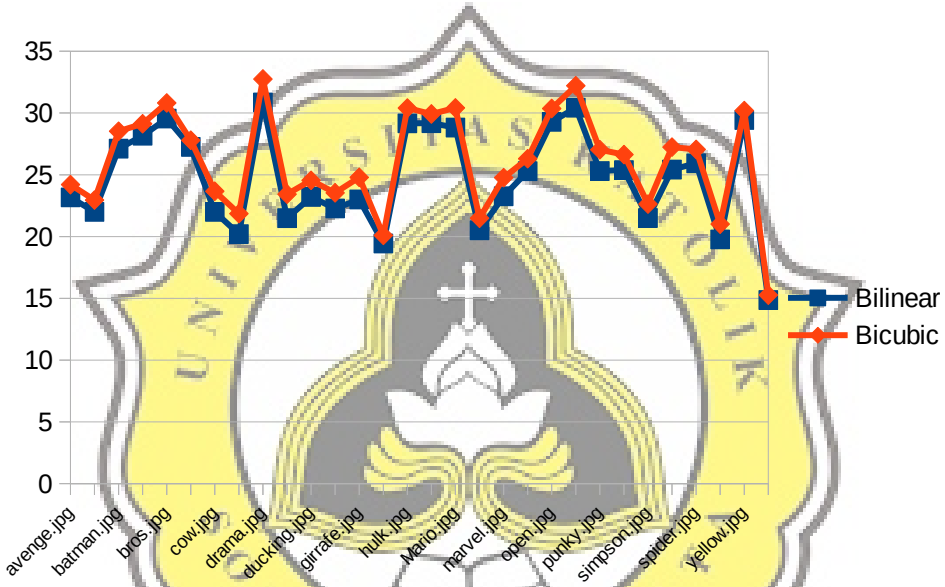
At 2x enlargement, Bilinear has a minimum value of PSNR 15,9 dB and maximum value of PSNR 35,6 dB while Bicubic has a minimum value of PSNR 18,29 and has maximum value of PSNR 37,2 dB. the average PSNR value in Bilinear was 27,7 dB while in interpolasi Bicubic was 30,76 dB.

PSNR value at 4x enlargement scale :



At 4x enlargement, Bilinear has a minimum value of PSNR 15,1 dB and maximum value of PSNR 32,5 dB while Bicubic has a minimum value of PSNR 15,8 and has maximum value of PSNR 34,7 dB. the average PSNR value in Bilinear was 26,4 dB while in interpolasi Bicubic was 28,1 dB.

PSNR value at 6x enlargement scale :



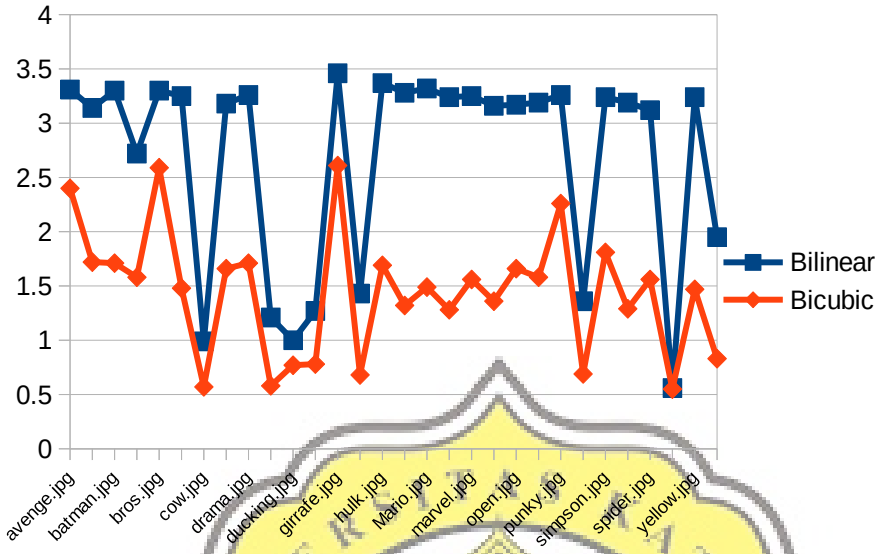
At 6x enlargement, Bilinear has a minimum value of PSNR 14,8 dB and maximum value of PSNR 31,5 dB while Bicubic has a minimum value of PSNR 15,26 dB and has maximum value of PSNR 33,2 dB. the average PSNR value in Bilinear was 25,1 dB while in interpolasi Bicubic was 26,3 dB.

The Result of the images that have been enlarged and the result of time process sort by enlargement scale 2x, 4x, 6x. Displayed in Table 5.2 and graphic.

Table 5.2: Image Enlargement Result

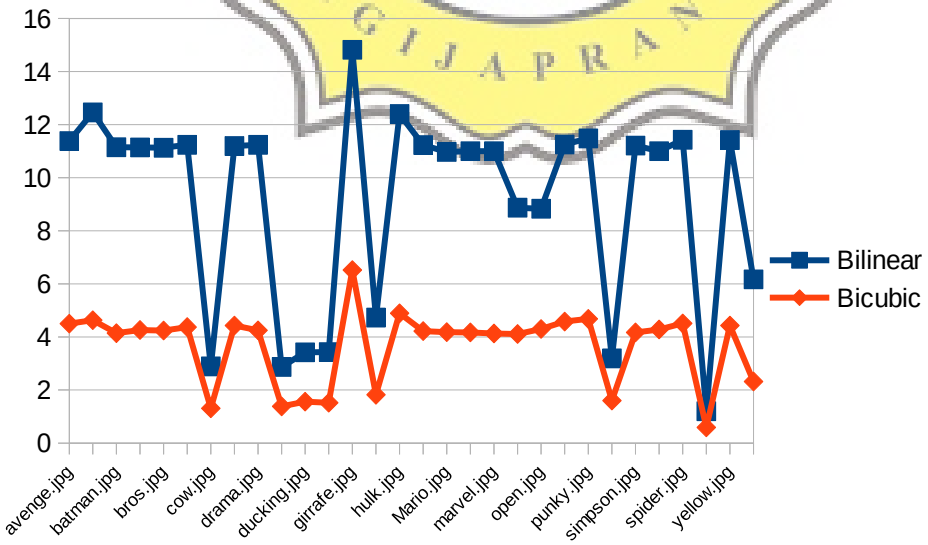
Image	Original Size	Result Size		
		2x	4x	6x
Avenge.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Babycow.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Batman.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Bowser.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Bros.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Bull.jpg	1200x900	2400x1800	4800x3600	7200x5400
Cat.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Cow.jpg	900x600	1800x1200	3600x2400	5400x3600
Donald.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Drama.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Duck.jpg	900x600	1800x1200	3600x2400	5400x3600
Ducking.jpg	900x600	1800x1200	3600x2400	5400x3600
Flyduck.jpg	900x600	1800x1200	3600x2400	5400x3600
Girrafe.jpg	1920x1440	3840x2880	7680x5760	11520x8640
Gold.jpg	900x900	1800x1800	3600x3600	5400x5400
Hulk.jpg	1920x1200	3840x2400	7680x4800	11520x7200
Logo.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Mario.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Marv.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Marvel.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Peter.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Open.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Pony.jpg	1800x1200	3600x2400	7200x4800	10800x7200
Punky.jpg	1800x1200	3600x2400	7200x4800	10800x7200
Red.jpg	1080x600	2160x1200	4320x2400	6480x3600
Simpson.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Smash.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Spider.jpg	1920x1080	3840x2160	7680x4320	11520x6480
Sunset.jpg	480x360	960x720	1920x1440	2880x2160
Yellow.jpg	1980x1080	3960x2160	7920x4320	11880x6480

Time process at 2x enlargement:



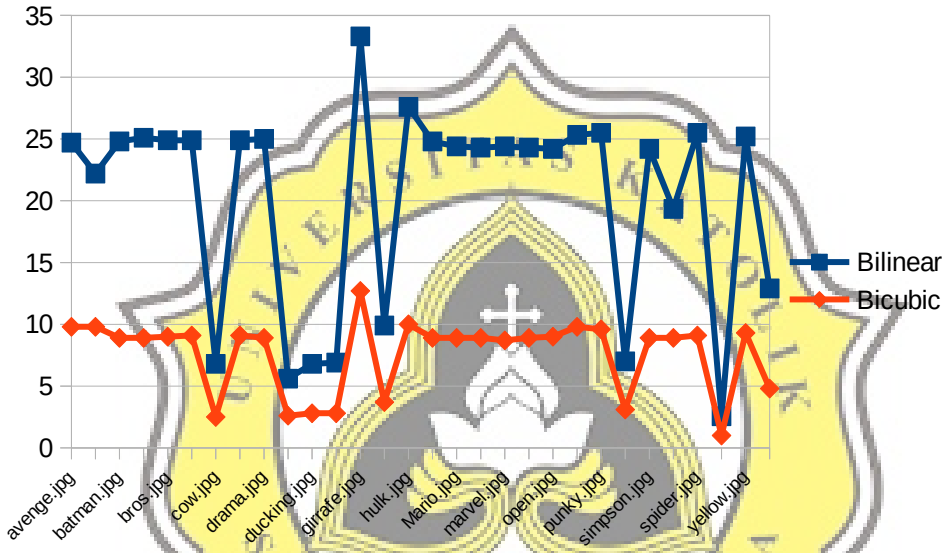
At 2x enlargement, Bilinear has a shortest time process 0,56 second and longest time process 3,5 second while Bicubic has a shortest time process 0,5 second and longest time process 2,7 second . the average time process in Bilinear was 2,64 second while in interpolasi Bicubic was 1,47 second.

Time process at 4x enlargement:



At 4x enlargement, Bilinear has a shortest time process 0,99 second and longest time process 14,8 second while Bicubic has a shortest time process 0,57 second and longest time process 6,5 second . the average time process in Bilinear was 9,6 second while in interpolasi Bicubic was 3,9 second.

Time Process at 6x enlargement:



At 6x enlargement, Bilinear has a shortest time process 2,56 second and longest time process 33,3 second while Bicubic has a shortest time process 1 second and longest time process 12,7 second . the average time process in Bilinear was 18,7 second while in interpolasi Bicubic was 7,3 second.