NORTHWEST NAZARENE UIVERSITY

# Posture Analysis from Images

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

BACHELOR OF ARTS

Jayden Weaver

2018

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

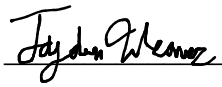
for the degree of

BACHELOR OF SCIENCE ARTS

Jayden Weaver

2018

Posture Analysis from Images

Author:  _____

Jayden Weaver

Approved:  _____

Barry L. Myers, Ph.D., Professor of Computer Science, Department of Mathematics and Computer Science, Faculty Advisor

Approved:  _____

John Davlin

Second Reader

Approved:  _____

Barry L. Myers, Ph.D., Chair,

Department of Mathematics & Computer Science

# ABSTRACT

Posture Analysis (Sublux).

WEAVER, JAYDEN (Department of Mathematics & Computer Science), MYERS, DR. BARRY (Department of Mathematics & Computer Science).

According to the American Chiropractic Association, 31 million Americans experience low-back pain at any given time. Sublux, an Android application written in Java, aims to reduce this number by putting a valuable tool in the average person's hand. This tool analyzes photos of a user and returns the results so that adjustments to the user's posture can be improved as needed through chiropractic therapy. The end result of the project is a fully functional Android application that successfully analyzes a user's posture through photos. The photo is then displayed alongside the results of the analysis for the user to review. While Sublux is fully functional, there may be issues analyzing some photos due to lighting, backdrops and background items, or changes in textures. Sublux recommends good lighting and a solid background for best results, but also contains code to combat any photos that are subpar. The code to combat this is not perfect, however, and will need to be improved in future versions. The source code will be available at github.com/jayden2013/sublux, and an APK will be available for download at jayden2013.github.io/sublux.

# Table of Contents



# Table of Figures

## Overview

According to the American Chiropractic Association, 31 million Americans experience low-back pain at any given time. Sublux is an Android application that aims to reduce this number by analyzing photos of a user for signs of poor posture and returning the results to the user so that an improvement on their posture can be made.

The target audience for this project is limited to Android users, because the application has not been developed for any other platform. However, there is no particular subset of Android users that would benefit more from the application than another because good posture is important to all living beings.

## Data Structures

In order to analyze an image, the image selected by the user is decoded into a Bitmap object using the decodeFile method in Android's BitmapFactory class. Once this is done, information about the image such as height and width are used in the analysis process to avoid null pointers when iterating through each pixel.

During this iteration, each pixel was analyzed and information about the pixel, such as the RGB values was stored in a PixelObject. The PixelObject class was written to store information about a pixel and change information about the pixel. The contents of the PixelObject class can be seen in Figure 1 below. While a package with an existing pixel object could have been used, the PixelObject class was written instead so that information that needed to be stored could be added into the object class if needed.

```
1.  package com.sublux.jayden.sublux;
2.  /**
3.  * A Pixel Object to store information about individual pixels.
4.  * @author Jayden Weaver
5.  */
6.  public class PixelObject {
7.      int red = 0, green = 0, blue = 0, color = 0; //Initialize variables.
8.      public PixelObject(int r, int g, int b, int c) { //Set RGB values.
9.              this.red = r;
10.             this.green = g;
11.             this.blue = b;
12.         } //Returns the color value
13.     public int getColor() {
14.             return this.color;
15.         } //Returns Red Value
16.     public int getRed() {
17.             return this.red;
18.         } //Returns Green Value
19.     public int getGreen() {
20.             return this.green;
21.         } //Returns Blue Value
22.     public int getBlue() {
23.             return this.blue;
24.         } //Sets the red value, returns if value was successfully set.
25.     public boolean setRed(int r) {
26.             this.red = r;
27.             return this.red == r;
28.         } //Sets the green value, returns if value was successfully set.
29.     public boolean setGreen(int g) {
30.             this.green = g;
31.             return this.green == g;
32.         } //Sets the blue value, returns if value was successfully set.
33.     public boolean setBlue(int b) {
34.             this.blue = b;
35.             return this.blue == b;
36.         } //Sets the color value, returns if value was successfully set.
37.     public boolean setColor(int c) {
38.         this.color = c;
39.         return this.color == c;
40.     }
41. }
```

*Figure 1. PixelObject class.*

In order to display the results of the analysis as text, a single string is used. A StringBuilder is used to combine all of the strings throughout the analysis process

opposed to string concatenation. This is done to improve performance, as there could be many strings of varying lengths being appended to one another.

**Analysis**

In order to meet the user's needs and accomplish posture analysis, Sublux evaluates each pixel in the image and uses its RGB values to make a decision on whether or not the pixel is part of the human body, or if it is part of the image background. This is done by checking if a pixel's RGB values are close enough to the RGB values of the first pixel, located at (x:0, y:0) in the bitmap. If the RGB values differ enough from the first pixel's RGB values enough to be outside of the threshold, they are considered to be a different structure inside of the image, not the background, and are colored black. If they were within the threshold of acceptable values, they are considered to be the background of the image and are colored blue. These colors were arbitrarily chosen so that the focus and background pixels were uniform with one another. The pixel at (x:0, y:0) was chosen to be the sample of the background color because it is highly unlikely that the pixel in the top left corner would be the focus of the photo. It is more likely that the pixel is part of the background instead. However, this method of separating the background from the focus of the image is not perfect because of the possibility of a background with varying colors. Because of this, it is recommended that images to be analyzed are taken against a solid background. The code for this is shown in Figure 2 below.

```
1.  //Analyze pixel by pixel
2.  while (y < height) {
3.     while (x < width) {
4.        pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x
    , y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using
     values.
5.        if (pixel.getRed() > bgPixel.getRed() - OFFSET && pixel.getRed() < bgPixel.getRe
    d() + OFFSET) {
6.           result.setPixel(x, y, Color.BLACK);
7.        } else if (pixel.getGreen() > bgPixel.getGreen() - OFFSET && pixel.getGreen() < b
    gPixel.getGreen() + OFFSET) {
8.           result.setPixel(x, y, Color.BLACK);
9.        } else if (pixel.getBlue() > bgPixel.getBlue() - OFFSET && pixel.getBlue() < bgPix
    el.getBlue() + OFFSET) {
10.          result.setPixel(x, y, Color.BLACK);
11.       } else {
12.          result.setPixel(x, y, Color.BLUE);
13.       }
14.       x++; //Increment column position.
15.    }
16.    y++; //Increment row position.
17.    x = 0; //New row, reset column position.
18. }
```

*Figure 2. Initial Analysis Algorithm.*

After this process is complete, it is done once more to check for any pixels that may have been inaccurately determined to be part of the human body. This is done by comparing the pixel color, either black or blue, to the pixel adjacent to it and changing the pixel color. While this process is useful for removing any stray black pixels in the background, it also thins out lines which can be detrimental to the analysis. Because of this, the process is only performed once. The Java code for this is shown below in figure 3.

```
1.  /**
2.  * Cleans up a bitmap, thinning out lines and shadows.
3.  * @param bmp
4.  * @return
5.  */
6.  public Bitmap cleanUp(Bitmap bmp) {
7.      Bitmap cleaned = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
8.      PixelObject pixel, nextPixel;
9.      boolean firstFound = false;
10.     int x = 0, y = 0;
11.     while (y < height) {
12.         while (x < width) {
13.             pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel
    (x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object usi
    ng values.
14.             if (x < width - 1) {
15.                 nextPixel = new PixelObject(Color.red(bmp.getPixel(x + 1, y)), Color.green(b
    mp.getPixel(x + 1, y)), Color.blue(bmp.getPixel(x + 1, y)), bmp.getPixel(x + 1, y)); //Cre
    ate new pixel object using values.
16.             } else {
17.                 nextPixel = pixel;
18.             }
19.             if (pixel.getBlue() != 255 && !firstFound) { //If the pixel is black and it's the first
    one found...
20.                 cleaned.setPixel(x, y, Color.BLACK);
21.                 firstFound = true;
22.             } else if (pixel.getBlue() != 255 && nextPixel.getBlue() == 255) { //If the pixel is
     black and the next pixel is blue...
23.                 cleaned.setPixel(x, y, Color.BLACK);
24.             } else {
25.                 cleaned.setPixel(x, y, Color.BLUE);
26.             }
27.             x++;
28.         }
29.         y++;
30.         x = 0;
31.         firstFound = false;
32.     }
33.     return cleaned;
34. }
```

*Figure 3. Bitmap Cleanup.*

Once this is complete, the application goes through the pixels once more to determine where the user's head, feet, chin, shoulders, hips, and ears are and compares

the x and y coordinates of each body part to determine if they are within an acceptable range of values. The code for determining the locations of these body parts are shown in figures 4 through 9, respectively.

```
1.  //Determine the coordinates of the top of the head. Also the center mass X coordinate.
2.  while (y < height) {
3.     while (x < width) {
4.        pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x
    , y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using
     values.
5.        if (pixel.getBlue() != 255 && topHeadFound == false && y > 20 && x > width / 4)
     { //If the color is black, the top of the head hasn't been found yet, y is greater than 40 pix
    els, and x is greater than a quarter of the width...
6.           topHeadFound = true;
7.           topHeadX = x; //Get topHeadX value for analysis.
8.           topHeadY = y; //Get topHeadY value for analysis.
9.           while (x < width) { //This is all for testing.
10.             analyzedImage.setPixel(x, y, Color.RED);
11.             x++;
12.          }
13.       } else if (pixel.getBlue() == 255) {
14.          analyzedImage.setPixel(x, y, Color.BLUE);
15.       } else {
16.          analyzedImage.setPixel(x, y, Color.BLACK);
17.       }
18.       x++;
19.    }
20.    y++;
21.    x = 0;
22. }
23. centerMassX = topHeadX; //The center mass X coordinate and the top head X coordinate
    are the same. Using different variables for easier understanding.
```

*Figure 4. Head Analysis.*

In the figure above, the image is analyzed from left to right, starting at the top of the image toward the bottom of the image. Each pixel color is checked, and the first black pixel is considered to be the very top of the head because the analysis is being done from top to bottom. This pixel's x-coordinate is also the x-coordinate of the center mass of the subject being analyzed, which is important to the analysis of other portions of the image.

```
1.  //Determine foot coordinate.
2.  y = height - 1;
3.  x = 0;
4.  boolean bottomYFound = false;
5.  int bottomY = 0;
6.  while (y > 0) {
7.      while (x < width) {
8.          pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x
    , y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using
     values.
9.          if (pixel.getBlue() != 255 && bottomYFound == false) {
10.             bottomYFound = true;
11.             bottomY = y;
12.             while (x < width) { //This is all for testing.
13.                 analyzedImage.setPixel(x, y, Color.RED);
14.                 x++;
15.             }
16.         }
17.         x++;
18.     }
19.     y--;
20.     x = 0;
21. }
```

*Figure 5. Foot Analysis.*

The Java code for determining the foot coordinate is in the figure above. While it may seem like an unnecessary step at first glance, because the height of the image could be used, it is completely necessary for the next step of the image analysis, which is accurately determining the center mass Y-coordinate. If the height of the image is used, the center mass Y-coordinate for the subject of the image would not be accurate, which would result in an inaccurate posture analysis. The process for determining the foot coordinate is very similar to finding the head coordinate. However, it is reversed.

After the coordinates for the top of the head and feet are determined, the Y value of the top of the head is subtracted from the Y value of the feet, which gives the overall height in pixels of the subject. Dividing this value by two results in the Y-Coordinate of the center mass of the subject in the image. This value combined with the X-Coordinate

from the top of the head determined earlier, gives the exact location of the center mass of the subject. This location will then be used later as a start point for hip analysis.

The next step in the image analysis is determining where the chin is. Determining where the chin is will give the relative location of where the shoulders are, so that the shoulders can be located and compared. The Java code for doing this is shown below. Starting at center mass, pixel colors are checked moving upward by decrementing the y value, until a black pixel is found or until the Y-Coordinate is the same as the top of the head. If the Y-Coordinate is the same as the top of the head, then the chin could not be found, and the value is kept at the default 0. This default value will give inaccurate results in the results activity.

```
1.  //Find chin
2.  y = centerMassY;
3.  int bottomHeadY = 0;
4.  while (y > topHeadY) {
5.      pixel = new PixelObject(Color.red(bmp.getPixel(centerMassX, y)), Color.green(bmp.g
    etPixel(centerMassX, y)), Color.blue(bmp.getPixel(centerMassX, y)), bmp.getPixel(cente
    rMassX, y)); //Create new pixel object using values.
6.      if (pixel.getBlue() != 255) {
7.          bottomHeadY = y; //For Testing
8.          x = 0;
9.          while (x < width) {
10.             analyzedImage.setPixel(x, bottomHeadY, Color.RED);
11.             x++;
12.         }
13.         break;
14.     }
15.     y--;
16. }
```

*Figure 6. Chin Analysis.*

```
1.  //Analyze Left Shoulder
2.  int shoulderLine = 0;
3.  shoulderLine = centerMassY - bottomHeadY;
4.  x = 0;
5.  while (x < width) {
6.      analyzedImage.setPixel(x, shoulderLine, Color.WHITE);
7.      x++;
8.  }
9.  y = bottomHeadY; //Set x to x/4 of centerMassX. Can't do it in one operation for some re
    ason.
10. x = centerMassX / 4;
11. x = centerMassX - x;
12. Point leftShoulderPoint = new Point();
13. while (y < shoulderLine) {
14.     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
15.     if (pixel.getBlue() != 255) { //If pixel is black
16.         break;
17.     }
18.     y++;
19.     if (y == shoulderLine) { //If we made it to the shoulder line without finding a suspected
     shoulder, restart with x - 1;
20.         y = bottomHeadY;
21.         x -= 1;
22.     }
23. }
24. leftShoulderPoint.set(x, y);
25. analyzedImage.setPixel(leftShoulderPoint.x, leftShoulderPoint.y, Color.GREEN); //Find
    the right shoulder
26. x = centerMassX / 4;
27. x = centerMassX + x;
28. Point rightShoulderPoint = new Point();
29. while (y < shoulderLine) {
30.     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
31.     if (pixel.getBlue() != 255) { //If pixel is black
32.         break;
33.     }
34.     y++;
35.     if (y == shoulderLine) {
36.         y = bottomHeadY;
37.         x += 1;
38.     }
39. }
```

```
40. rightShoulderPoint.set(x, y);
41. analyzedImage.setPixel(rightShoulderPoint.x, rightShoulderPoint.y, Color.GREEN);
```

*Figure 7. Shoulder Analysis.*

After the chin is analyzed, the shoulders are analyzed using the chin's Y-Coordinate. The Java code for this is shown above. The relative location of the shoulders is determined by subtracting the Y-Coordinate of the chin and the Y-Coordinate of center mass. This value is then stored in a variable called shoulderLine. This is done to determine an end point to the analysis, so that the shoulders are not erroneously being determined to be lower than they are. Once this value has been calculated, each pixel has its color checked. Just as previous analyses, the x and y values are incremented and decremented until a pixel is found.

```
1.  //Left Hip
2.  y = centerMassY; //Set x to x/4 of centerMassX. Can't do it in one operation for some rea
    son.
3.  x = centerMassX / 4;
4.  x = centerMassX - x;
5.  while (x < centerMassX) {
6.      pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
7.      if (pixel.getBlue() != 255) { //If the pixel is black.
8.          break;
9.      }
10.     x++;
11.     if (x == centerMassX) { //Set x to x/4 of centerMassX. Can't do it in one operation for
    some reason.
12.         x = centerMassX / 4;
13.         x = centerMassX - x;
14.         y++; //Try again at new Y coordinate.
15.     }
16.     if (y == height) {
17.         break;
18.     }
19. }
20. Point leftHipPoint = new Point();
21. leftHipPoint.set(x, y); //Testing.
22. analyzedImage.setPixel(leftHipPoint.x, leftHipPoint.y, Color.YELLOW); //Right hip
```

```
23. y = centerMassY;
24. x = centerMassX / 4;
25. x = centerMassX + x;
26. while (x > centerMassX) {
27.     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
28.     if (pixel.getBlue() != 255 && x > centerMassX) { //If pixel is black.
29.         break;
30.     }
31.     if (x == 0) {
32.         x = centerMassX / 4;
33.         x = centerMassX + x;
34.         y++;
35.     }
36.     if (y == height) {
37.         break;
38.     }
39.     x--;
40. }
41. Point rightHipPoint = new Point();
42. rightHipPoint.set(x, y); //Testing
43. analyzedImage.setPixel(rightHipPoint.x, rightHipPoint.y, Color.YELLOW);
```

*Figure 8. Hip Analysis.*

The hip analysis is shown above. For each pixel, beginning at the center mass of the subject, and moving left or right based on which hip is being located, the pixel color is checked. If the pixel color is black, the hip has been located and the current pixel's coordinates are saved to be compared after both hips are located. If no black pixel is located, the Y-coordinate is incremented and the analysis is done again at the new Y-coordinate in an attempt to find the hip. This is a remedy to the thinning line problem caused by the initial, but essential, cleanup method. If no black pixels are found, and the Y-coordinate is the height of the image, the attempt to locate the hip is halted, and the center mass X value and the height value are used for the hip location X and Y values respectively. This would then result in values outside of the hip threshold during the comparison of the hips, notifying the user in the app results activity.

Another useful piece of information for analyzing posture is knowing where the ears are. The code below determines where the ears are so that they can be used later in determining if the subject's head is tilted. This is done using the same method as determining where other locations of the body are. The pixel colors are checked, starting at the center Y-coordinate of the head, and if they are black, they are a location of an ear. If an ear cannot be located, the starting Y-Coordinate is decremented, and the search is restarted.

```
1.  //Get ear Y
2.  int midHeadY = bottomHeadY - topHeadY; //Get Right Ear
3.  x = centerMassX;
4.  y = midHeadY;
5.  while (x < width) {
6.      pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
7.      if (pixel.getBlue() != 255) { //If pixel color is black.
8.          break;
9.      }
10.     x++;
11.     if (x == width) {
12.         x = 0;
13.         y -= 1;
14.     }
15.     if (y == 0) {
16.         x = 0;
17.         y = 0;
18.         break;
19.     }
20. }
21. Point rightEarPoint = new Point();
22. rightEarPoint.set(x, y); //Get Left Ear
23. x = leftShoulderPoint.x;
24. y = midHeadY;
25. while (x < centerMassX) {
26.     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y
    )), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using v
    alues.
27.     if (pixel.getBlue() != 255 && x < centerMassX - 25) { //If pixel color is black. //fixfixf
    ix
```

```
            break;
        }
        x++;
        if (x == centerMassX) {
            x = leftShoulderPoint.x;
            y -= 1;
        }
        if (y == 0) {
            x = 0;
            y = 0;
            break;
        }
    }
    Point leftEarPoint = new Point();
    leftEarPoint.set(x, y);
    analyzedImage.setPixel(rightEarPoint.x, rightEarPoint.y, Color.GREEN);
    analyzedImage.setPixel(leftEarPoint.x, leftEarPoint.y, Color.GREEN);
```

*Figure 9. Ear Analysis.*

If the difference between coordinate values of different body parts are within the range of acceptable values, the user is notified that their posture is considered acceptable. If the difference is outside the range of acceptable values, the user is notified that their posture is not considered acceptable and is told how their posture differs from acceptable posture. The threshold of acceptable values has been added to compensate for small variations in pixel coordinates, which would result in an inaccurate posture reading if the values were not exactly equal. Coordinate variations could be caused by image lighting, clothing, variations in body size, hair, or other bodily features. Because of this, a threshold is necessary for accurate readings. The code for this is shown below in figures 10 through 12.

```
1. //Check for tilted head.
2. int leftShoulderToEar = leftEarPoint.x - leftShoulderPoint.x;
3. int rightShoulderToEar = rightShoulderPoint.x - rightEarPoint.x;
4. boolean head_tilted_left = false;
5. String headTiltText = "No Posture Information Available.";
6. if (Math.abs(leftShoulderToEar - rightShoulderToEar) > SHOULDER_TO_EAR_THRE
   SHOLD) {
```

```
    head_tilted_left = leftShoulderToEar < rightShoulderToEar;
    headTiltText = "Head tilt is outside the range of acceptable values. ";
    if (head_tilted_left) {
        headTiltText += "Your head is tilted to the left.\n";
    } else {
        headTiltText += "Your head is tilted to the right.\n";
    }
} else {
    headTiltText = "Your head and neck are looking excellent!\n";
}
resultsString.append(headTiltText);
```

*Figure 10. Head Tilt Check.*

```
String shoulderResultText = "No posture information available."; //Get shoulder posture value.
shoulder_posture_value = Math.abs(leftShoulderPoint.y - rightShoulderPoint.y); //Check shoulder posture value against shoulder threshold.
if (shoulder_posture_value > SHOULDER_THRESHOLD) {
    leftShoulderHigher = leftShoulderPoint.y > rightShoulderPoint.y;
    shoulderResultText = "Shoulder posture is outside the range of acceptable values. ";
    if (leftShoulderHigher) {
        shoulderResultText += "Your left shoulder is higher than your right shoulder.\n";
    } else {
        shoulderResultText += "Your left shoulder is lower than your right shoulder.\n";
    }
} else {
    shoulderResultText = "Your shoulder posture is looking great!\n";
}
resultsString.append(shoulderResultText);
```

*Figure 11. Shoulder Check.*

```
        String hipText; //Check Hip Values
if (Math.abs((rightHipPoint.x - centerMassX) - (centerMassX - leftHipPoint.x)) > HIP_WIDTH_THRESHOLD) {
    hipText = "Hip values are outside of threshold.\n";
} else {
    hipText = "Your hip posture is looking excellent!\n";
}
resultsString.append(hipText);
```

*Figure 12. Hip Check.*

The results are displayed alongside the analyzed image, where the various body parts the application detected are highlighted for the user to review. Displaying the

analyzed image gives the user a look at what the application sees, so that they can verify that the results are correct. This is done because in some cases, an image may be subpar and may be detrimental to the analysis of the image, resulting in inaccurate readings. If a user can see what the application is seeing, the user can adjust their photos accordingly. A screenshot of the results activity is shown below in Figure 13.

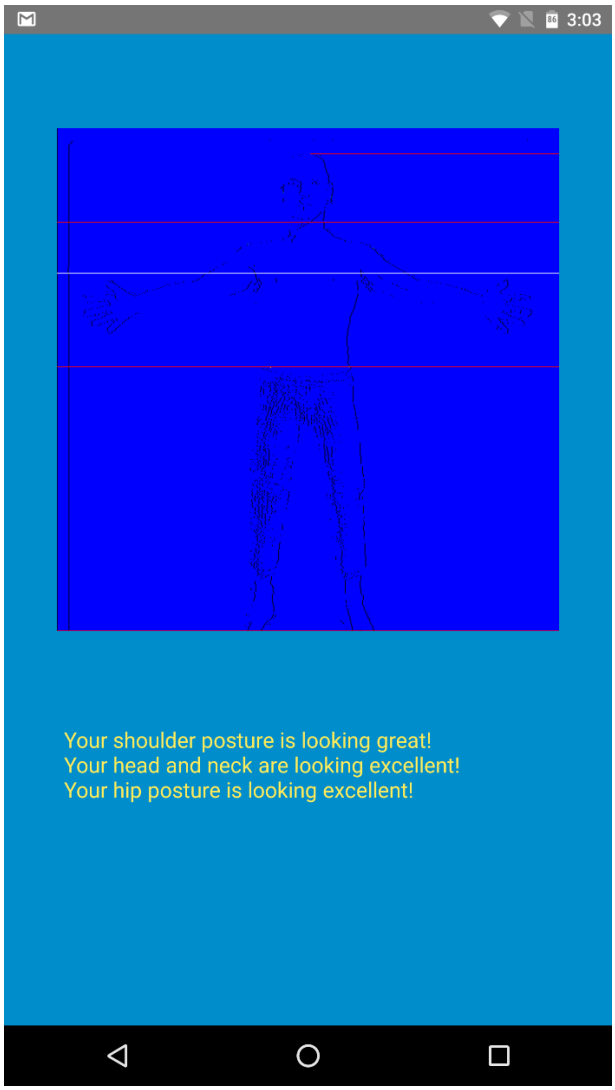



*Figure 13. Results Activity Screenshot.*

While Sublux attempts to combat subpar images, the code for this is not perfect and could be improved in future versions of this application. For best results using Sublux, images with solid backgrounds, adequate lighting, consistent textures, and an absence of shadows is recommended.

**Aesthetics**

Sublux was built with usability, simplicity, and beauty in mind. For this reason, the application maintains a very simple layout, with colors that are visually appealing. This color scheme can be seen in figure 14 below.

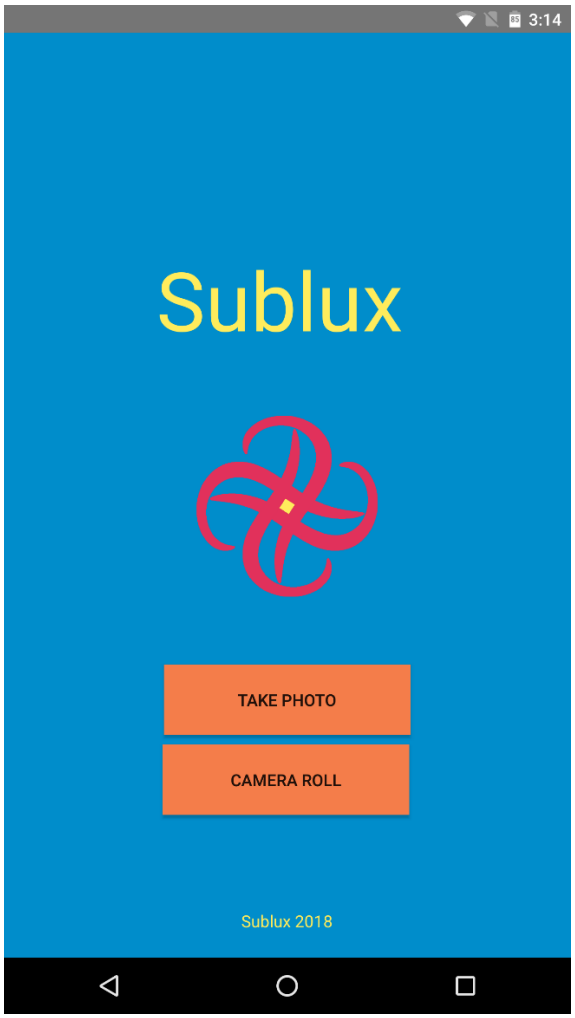



*Figure 14. Application Home Screen Screenshot.*

In addition to an appealing color scheme, Sublux also employs a splash screen with a faux progress bar. The reasoning behind this is to make the app feel more professional, and because the app would immediately open and it felt off to the users who were testing the application. Additionally, a quick progress bar gives the user a good feeling, which in turn gives the user a better experience. A screenshot of the splash screen is shown in figure 15 below.

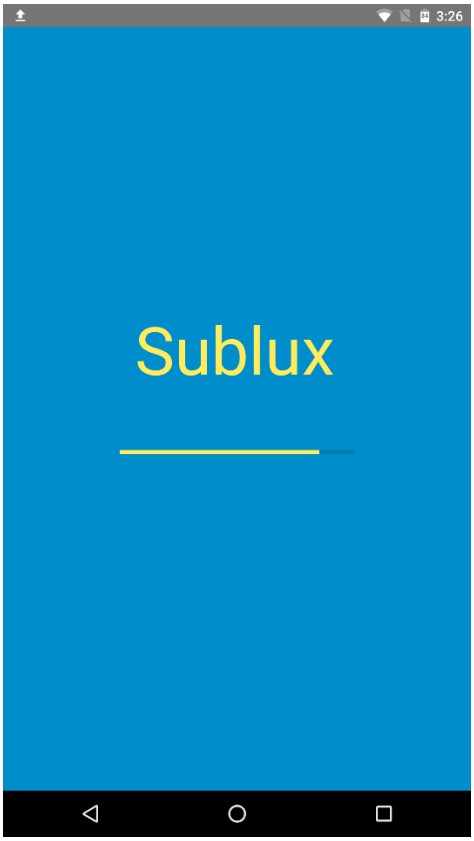



*Figure 15. Splash Screen Screenshot.*

**Language and Platform**

Sublux was built for and tested on Android 7.0 “Nougat”, which uses API level 24. It was developed in Android Studio using the Java programming language. The Java programming language was chosen due to familiarity, because it was taught and used heavily throughout the Computer Science program. In addition to this, Java was the

official programming language of Android development for years. Only recently was it replaced with the alternative Kotlin programming language, which runs on the Java Virtual Machine and is interoperable with Java. Though Kotlin would have had its advantages and is very similar to Java, familiarity has its advantages as well, and if needed, some Kotlin code could have been used if necessary. Because of the familiarity, popularity, and interoperability, choosing Java over any other language made sense.

Android 7.0 “Nougat” was chosen as the development platform because it is a modern Android operating system, providing a higher API level of 24, while still being old enough to encompass a variety of phones on the market. In addition to this, Android development is cheaper than its iOS counterpart, and one of the goals was to keep costs to a minimum. iOS has a $99 yearly fee, while Android has a one-time fee of $25. Additionally, without some sort of hack, iOS development cannot be done outside of Apple’s IDE, XCode, on Macintosh OS X.

**Problems**

While the final product is a fully functional posture analysis application, it was not built without encountering problems. One of the problems that occurred was being unable to successfully access the files on the filesystem. This happened multiple times, and the cause was different each time. The first time this happened was due to lack of permissions. It was not hard to determine why the files could not be accessed in this case, however inexperience in Android development made solving the problem more difficult. While the permissions were specified in the AndroidManifest.xml file, the permissions were never requested by the application itself. To fix this, a permission check was created before image analysis was attempted.

The same problem occurred, but with a different cause, when attempting to run the application on a physical device. Up until that point, an Android emulator was used for testing. In this case, the file system could not be accessed because the code to get the physical file path of a selected image would not return a valid file path and null pointers ensued. It is believed that the file structure of the emulator differs from the file structure of a physical device, which caused varying results on the two. Namely, the fact that physical devices can have physical removable storage.

**Unfinished Portions**

One thing that is unfinished in the application is the option to take a photo to use for image analysis directly from the application. This is currently the only thing in the application that is not working properly. The ability to take a photo is there, however, the application cannot access the photo that was just taken. The cause of this is the inability to pass the photo that was taken from Android's camera intent to the home intent. This problem will need to be addressed, and is likely a simple fix, but it does not affect the main functionality of the application and was not a priority during development.

The next obvious addition to the application would be posture analysis from a side view of the user. This addition would not be very difficult to implement now that frontal analysis has been accomplished. However, to implement this a separate button for side analysis would need to be built along with a results page. It is possible that code could be written to detect if the photo is a side view or a front view, though the code may inaccurately determine some images to be side or front views. The code to do this would also generate additional overhead, because the image would need to be analyzed another time.

## References

InnocentKiller, April 19, 2018, answer on amburnside, “Get filepath and filename of selected gallery image in Android”, Stack Overflow, December 2, 2013, https://stackoverflow.com/a/20324422

## Appendix

```java
package com.sublux.jayden.sublux;
/** * A Pixel Object to store information about individual pixels. * @author Jayden Weaver */
public class PixelObject {
    int red = 0, green = 0, blue = 0, color = 0; //Initialize variables.
    public PixelObject(int r, int g, int b, int c) { //Set RGB values.
            this.red = r;
            this.green = g;
            this.blue = b;
        } //Returns the color value
    public int getColor() {
            return this.color;
        } //Returns Red Value
    public int getRed() {
            return this.red;
        } //Returns Green Value
    public int getGreen() {
            return this.green;
        } //Returns Blue Value
    public int getBlue() {
            return this.blue;
        } //Sets the red value, returns if value was successfully set.
    public boolean setRed(int r) {
            this.red = r;
            return this.red == r;
        } //Sets the green value, returns if value was successfully set.
    public boolean setGreen(int g) {
            this.green = g;
            return this.green == g;
        } //Sets the blue value, returns if value was successfully set.
    public boolean setBlue(int b) {
            this.blue = b;
            return this.blue == b;
        } //Sets the color value, returns if value was successfully set.
    public boolean setColor(int c) {
        this.color = c;
        return this.color == c;
    }
}
```

PixelObject.java

```
1.  package com.sublux.jayden.sublux;
2.  import android.content.Intent;
3.  import android.os.Bundle;
4.  import android.support.v7.app.AppCompatActivity;
5.  import android.widget.ProgressBar;
6.  public class SplashScreen extends AppCompatActivity {@
7.      Override protected void onCreate(Bundle savedInstanceState) {
8.          super.onCreate(savedInstanceState);
9.          setContentView(R.layout.activity_splash_screen);
10.         final int TIME_OUT = 10; //MILLISECONDS
11.         new Thread(new Runnable() {@
12.             Override public void run() {
13.                 increaseProgressBar(TIME_OUT);
14.                 Intent homeIntent = new Intent(SplashScreen.this, home.class);
15.                 startActivity(homeIntent);
16.                 finish();
17.             }
18.         }).start();
19.     }
20.     public void increaseProgressBar(int timeout) {
21.         ProgressBar pb = findViewById(R.id.progressBar);
22.         for (int i = 0; i <= 100; i++) {
23.             try {
24.                 Thread.sleep(timeout);
25.                 pb.setProgress(i);
26.             } catch (InterruptedException e) {
27.                 e.printStackTrace();
28.             }
29.         }
30.     }
31. }
```

SplashScreen.java

```
1.  package com.sublux.jayden.sublux;
2.  import android.app.Activity;
3.  import android.content.Intent;
4.  import android.graphics.Bitmap;
5.  import android.graphics.Color;
6.  import android.icu.text.DateFormat;
7.  import android.os.Bundle;
8.  import android.support.v7.app.AppCompatActivity;
9.  import android.view.View;
10. import android.widget.Button;
11. import android.widget.ImageView;
12. import android.widget.TextView;
13. import java.util.Date;
14. public class home extends AppCompatActivity {
15.     final int CAMERA_REQUEST = 1888;
16.     ImageView cameraView;@
17.     Override protected void onCreate(Bundle savedInstanceState) {
18.         super.onCreate(savedInstanceState);
19.         setContentView(R.layout.activity_home);
20.         Button cameraRollButton = (Button) findViewById(R.id.cameraRollButton);
21.         Button takePhotoButton = (Button) findViewById(R.id.takePhotoButton);
22.         final TextView subluxTitle = (TextView) findViewById(R.id.title); //For cool UI
     effects.
23.         subluxTitle.setOnClickListener(new View.OnClickListener() {@
24.             Override public void onClick(View view) {
25.                 if (subluxTitle.getCurrentTextColor() == Color.parseColor("#FFEC5C")) {

26.                     subluxTitle.setTextColor(Color.parseColor("#E1315B"));
27.                 } else {
28.                     subluxTitle.setTextColor(Color.parseColor("#FFEC5C"));
29.                 }
30.             }
31.         });
32.         cameraRollButton.setOnClickListener(new View.OnClickListener() {@
33.             Override public void onClick(View view) {
34.                 Intent i = new Intent(home.this, selectFromCameraRoll.class);
35.                 startActivity(i);
36.             }
37.         }); //THIS CODE FOR CAMERA SELECTION DOES NOT WORK.
38.         takePhotoButton.setOnClickListener(new View.OnClickListener() {
39.             ImageView cameraView = (ImageView) findViewById(R.id.cameraView);@
40.             Override public void onClick(View view) {
41.                 Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMA
    GE_CAPTURE);
42.                 startActivityForResult(cameraIntent, CAMERA_REQUEST);
43.             }
44.             protected void onActivityResult(int requestCode, int resultCode, Intent dat
    a) {
45.                 home.super.onActivityResult(requestCode, resultCode, data);
46.                 if (requestCode == CAMERA_REQUEST && resultCode == Activity.RESULT_OK)
    {
47.                     Bitmap photo = (Bitmap) data.getExtras().get("data");
48.                     cameraView.setImageBitmap(photo);
49.                 }
50.             }
51.         }); //Checking the date for fun. Maybe to change the app theme based on the dat
    e/time.
52.         String date = DateFormat.getDateInstance(DateFormat.SHORT).format(new Date()).s
    ubstring(0, 5);
53.         final TextView currentYearText = (TextView) findViewById(R.id.theCurrentYear);
```

```
54.         if (date.equals("1/1/1")) { //Check is weird, due to the way the string is subs
    tring'd. We'll still cash it tho.
55.             currentYearText.setText("Happy New Year");
56.         } else if (date.equals("7/4/1")) {
57.             currentYearText.setText("Happy Independence Day");
58.         } else if (date.equals("10/31")) {
59.             currentYearText.setText("Happy Halloween");
60.         }
61.     }
62. }
```

home.java

```
1.  package com.sublux.jayden.sublux;
2.  import android.Manifest;
3.  import android.content.Intent;
4.  import android.graphics.Bitmap;
5.  import android.graphics.BitmapFactory;
6.  import android.graphics.Color;
7.  import android.graphics.Point;
8.  import android.os.Bundle;
9.  import android.support.v4.app.ActivityCompat;
10. import android.support.v4.content.ContextCompat;
11. import android.support.v7.app.AppCompatActivity;
12. import android.view.View;
13. import android.widget.ImageView;
14. import android.widget.TextView;
15. import java.util.ArrayList;
16. public class results extends AppCompatActivity {
17.     String imagePath = "";
18.     int width = 0, height = 0;
19.     Bitmap result;
20.     StringBuilder resultsString = new StringBuilder();@
21.     Override protected void onCreate(Bundle savedInstanceState) {
22.             super.onCreate(savedInstanceState);
23.             setContentView(R.layout.activity_results);
24.             Intent i = getIntent();
25.             this.imagePath = i.getStringExtra("imagePath"); //Set Image Path
26.             try {
27.                 analyzeImage();
28.                 displayResults();
29.             } catch (Exception e) { //Instead of crashing, tell the user the analysis f
    ailed.
30.                 final ImageView modifiedImageView = (ImageView) findViewById(R.id.resul
    tView);
31.                 modifiedImageView.setVisibility(View.INVISIBLE); //Make image view invi
    sible.
32.                 TextView resultsView = (TextView) findViewById(R.id.resultsView);
33.                 resultsView.setText("Failed to analyze image. For best results use adeq
    uate lighting and a solid background.");
34.             }
35.         }
36.         /**     * Method to Analyze an image, probably pixel by pixel. We shall see.
      */
37.     public void analyzeImage() {
38.             int permissionCheck = ContextCompat.checkSelfPermission(this, Manifest.perm
    ission.READ_EXTERNAL_STORAGE);
39.             if (permissionCheck != 0) { //If we don't have permissions, request permiss
    ions
40.                 ActivityCompat.requestPermissions(this, new String[] {
41.                     Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRIT
    E_EXTERNAL_STORAGE
42.                 }, 1);
43.             } //Access file
44.             Bitmap bmp = BitmapFactory.decodeFile(imagePath); //This is the image to be
     analyzed, now dynamic.
45.             final int OFFSET = 35; //Offset for RGB Threshold. 50 doesn't seem to work
    for ears, but 45 does. 40 crashes.
46.             width = bmp.getWidth(); //Set Width
47.             height = bmp.getWidth(); //Set Height //  height = 300; //For Testing
48.             int x = 0, y = 0; //Initialize X + Y
49.             int totalPixels = width * height; //Calculate Total Pixel
50.             PixelObject pixel; //Pixel object for storing values.
```

```
51.             ArrayList < PixelObject > pixelArray = new ArrayList < PixelObject > (); //
    ArrayList for storing Pixels
52.             int backgroundColor = bmp.getPixel(0, 0); //Set the background color to be
    the first pixel for analysis.
53.             result = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
54.             PixelObject bgPixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.
    green(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Analy
    ze pixel by pixel
55.             while (y < height) { //Change height value when testing to something more r
    easonable, so that we don't run out of memory.
56.                 while (x < width) {
57.                     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(
    bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new
    pixel object using values.
58.                     if (pixel.getRed() > bgPixel.getRed() - OFFSET && pixel.getRed() <
    bgPixel.getRed() + OFFSET) {
59.                         result.setPixel(x, y, Color.BLACK);
60.                     } else if (pixel.getGreen() > bgPixel.getGreen() - OFFSET && pixel.
    getGreen() < bgPixel.getGreen() + OFFSET) {
61.                         result.setPixel(x, y, Color.BLACK);
62.                     } else if (pixel.getBlue() > bgPixel.getBlue() - OFFSET && pixel.ge
    tBlue() < bgPixel.getBlue() + OFFSET) {
63.                         result.setPixel(x, y, Color.BLACK);
64.                     } else {
65.                         result.setPixel(x, y, Color.BLUE);
66.                     }
67.                     x++; //Increment column position.
68.                 }
69.                 y++; //Increment row position.
70.                 x = 0; //New row, reset column position.
71.             }
72.             result = cleanUp(result);
73.             result = analyze(result);
74.         }
75.         /**     * Cleans up a bitmap, thinning out lines and shadows.     * @param bmp
        * @return     */
76.     public Bitmap cleanUp(Bitmap bmp) {
77.             Bitmap cleaned = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888
    );
78.             PixelObject pixel, nextPixel;
79.             boolean firstFound = false;
80.             int x = 0, y = 0;
81.             while (y < height) {
82.                 while (x < width) {
83.                     pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(
    bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new
    pixel object using values.
84.                     if (x < width - 1) {
85.                         nextPixel = new PixelObject(Color.red(bmp.getPixel(x + 1, y)),
    Color.green(bmp.getPixel(x + 1, y)), Color.blue(bmp.getPixel(x + 1, y)), bmp.getPixel(x
     + 1, y)); //Create new pixel object using values.
86.                     } else {
87.                         nextPixel = pixel;
88.                     }
89.                     if (pixel.getBlue() != 255 && !firstFound) { //If the pixel is blac
    k and it's the first one found...
90.                         cleaned.setPixel(x, y, Color.BLACK);
91.                         firstFound = true;
92.                     } else if (pixel.getBlue() != 255 && nextPixel.getBlue() == 255) {
    //If the pixel is black and the next pixel is blue...
93.                         cleaned.setPixel(x, y, Color.BLACK);
```

```
                    } else { //could remove this portion for more speed maybe.
                        cleaned.setPixel(x, y, Color.BLUE);
                    }
                    x++;
                }
                y++;
                    x = 0;
                    firstFound = false;
                }
                return cleaned;
            }
            /**     * Analyze posture.     * @return     */
        public Bitmap analyze(Bitmap bmp) {
                Bitmap analyzedImage = Bitmap.createBitmap(width, height, Bitmap.Con
fig.ARGB_8888);
                PixelObject pixel, nextPixel;
                boolean topHeadFound = false;
                int x = 0, y = 0;
                pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(b
mp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new p
ixel object using values.
                int topHeadX = 0, topHeadY = 0, centerMassX = 0;
                final int SHOULDER_THRESHOLD = height / 60; //The acceptable thresho
ld for shoulder comparisons. Anything outside of this will be considered bad posture.
                int shoulder_posture_value = 0;
                boolean leftShoulderHigher = false;
                int SHOULDER_TO_EAR_THRESHOLD = width / 50; //The acceptable thresho
ld for shoulder to ear comparisons. Anything outside of this will be considered bad pos
ture. Checks for tilted heads.
                int HIP_WIDTH_THRESHOLD = width / 45; //The acceptable threshold for
 centered hip comparisons. //Determine the coordinates of the top of the head. Also the
 center mass X coordinate.
                while (y < height) {
                    while (x < width) {
                        pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color
.green(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Crea
te new pixel object using values.
                        if (pixel.getBlue() != 255 && topHeadFound == false && y > 2
0 && x > width / 4) { //If the color is black, the top of the head hasn't been found ye
t, y is greater than 40 pixels, and x is greater than a quarter of the width...
                            topHeadFound = true;
                            topHeadX = x; //Get topHeadX value for analysis.
                            topHeadY = y; //Get topHeadY value for analysis.
                            while (x < width) { //This is all for testing.
                                analyzedImage.setPixel(x, y, Color.RED);
                                x++;
                            }
                        } else if (pixel.getBlue() == 255) {
                            analyzedImage.setPixel(x, y, Color.BLUE);
                        } else {
                            analyzedImage.setPixel(x, y, Color.BLACK);
                        }
                        x++;
                    }
                    y++;
                    x = 0;
                }
                centerMassX = topHeadX; //The center mass X coordinate and the top h
ead X coordinate are the same. Using different variables for easier understanding. //De
termine foot coordinate.
                y = height - 1;
```

```
                    x = 0;
                    boolean bottomYFound = false;
                    int bottomY = 0;
                    while (y > 0) {
                        while (x < width) {
                            pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using values.
                            if (pixel.getBlue() != 255 && bottomYFound == false) { //If the color is black, the top of the head hasn't been found yet, y is greater than 40 pixels, and x is greater than a quarter of the width...
                                bottomYFound = true;
                                bottomY = y;
                                while (x < width) { //This is all for testing.
                                    analyzedImage.setPixel(x, y, Color.RED);
                                    x++;
                                }
                            }
                            x++;
                        }
                        y--;
                        x = 0;
                    } //Determine center mass Y
                    int centerMassY = (bottomY - topHeadY) / 2; //This is all for testing:
                    x = 0;
                    while (x < width) {
                        analyzedImage.setPixel(x, centerMassY, Color.RED);
                        x++;
                    } //Find chin
                    y = centerMassY;
                    int bottomHeadY = 0;
                    while (y > topHeadY) {
                        pixel = new PixelObject(Color.red(bmp.getPixel(centerMassX, y)), Color.green(bmp.getPixel(centerMassX, y)), Color.blue(bmp.getPixel(centerMassX, y)), bmp.getPixel(centerMassX, y)); //Create new pixel object using values.
                        if (pixel.getBlue() != 255) {
                            bottomHeadY = y; //For Testing
                            x = 0;
                            while (x < width) {
                                analyzedImage.setPixel(x, bottomHeadY, Color.RED);
                                x++;
                            }
                            break;
                        }
                        y--;
                    } //Analyze Left Shoulder
                    int shoulderLine = 0;
                    shoulderLine = centerMassY - bottomHeadY;
                    x = 0;
                    while (x < width) {
                        analyzedImage.setPixel(x, shoulderLine, Color.WHITE);
                        x++;
                    }
                    y = bottomHeadY; //Set x to x/4 of centerMassX. Can't do it in one operation for some reason.
                    x = centerMassX / 4;
                    x = centerMassX - x;
                    Point leftShoulderPoint = new Point();
                    while (y < shoulderLine) {
```

```
193.                         pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.gre
    en(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create n
    ew pixel object using values.
194.                         if (pixel.getBlue() != 255) { //If pixel is black
195.                             break;
196.                         }
197.                         y++;
198.                         if (y == shoulderLine) { //If we made it to the shoulder line wi
    thout finding a suspected shoulder, restart with x - 1;
199.                             y = bottomHeadY;
200.                             x -= 1;
201.                         }
202.                         if (x == 0) {
203.                             break;
204.                         }
205.                     }
206.                     leftShoulderPoint.set(x, y);
207.                     analyzedImage.setPixel(leftShoulderPoint.x, leftShoulderPoint.y, Col
    or.GREEN); //Find the right shoulder
208.                     x = centerMassX / 4;
209.                     x = centerMassX + x;
210.                     Point rightShoulderPoint = new Point();
211.                     while (y < shoulderLine) {
212.                         pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.gre
    en(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create n
    ew pixel object using values.
213.                         if (pixel.getBlue() != 255) { //If pixel is black
214.                             break;
215.                         }
216.                         y++;
217.                         if (y == shoulderLine) {
218.                             y = bottomHeadY;
219.                             x += 1;
220.                         }
221.                         if (x == width - 1) {
222.                             break;
223.                         }
224.                     }
225.                     rightShoulderPoint.set(x, y);
226.                     analyzedImage.setPixel(rightShoulderPoint.x, rightShoulderPoint.y, C
    olor.GREEN);
227.                     String shoulderResultText = "No posture information available."; //G
    et shoulder posture value.
228.                     shoulder_posture_value = Math.abs(leftShoulderPoint.y - rightShoulde
    rPoint.y); //Check shoulder posture value against shoulder threshold.
229.                     if (shoulder_posture_value > SHOULDER_THRESHOLD) {
230.                         leftShoulderHigher = leftShoulderPoint.y > rightShoulderPoint.y;

231.                         shoulderResultText = "Shoulder posture is outside the range of a
    cceptable values. ";
232.                         if (leftShoulderHigher) {
233.                             shoulderResultText += "Your left shoulder is higher than you
    r right shoulder.\n";
234.                         } else {
235.                             shoulderResultText += "Your left shoulder is lower than your
     right shoulder.\n";
236.                         }
237.                     } else {
238.                         shoulderResultText = "Your shoulder posture is looking great!\n"
    ;
239.                     }
```

```
240.                     resultsString.append(shoulderResultText); //Analyze head for tilt. /
    /Get distance between shoulder points.
241.                     int shoulderDistance = rightShoulderPoint.x - leftShoulderPoint.x; /
    /Get ear Y
242.                     int midHeadY = bottomHeadY - topHeadY; //Get Right Ear
243.                     x = centerMassX;
244.                     y = midHeadY;
245.                     while (x < width) {
246.                         pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.gre
    en(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create n
    ew pixel object using values.
247.                         if (pixel.getBlue() != 255) { //If pixel color is black.
248.                             break;
249.                         }
250.                         x++;
251.                         if (x == width - 1) {
252.                             x = 0;
253.                             y -= 1;
254.                         } //ADDED THIS IN TO TRY TO FIX CRASHES
255.                         if (y == 0) {
256.                             x = 0;
257.                             y = 0;
258.                             break;
259.                         }
260.                     }
261.                     Point rightEarPoint = new Point();
262.                     rightEarPoint.set(x, y); //Get Left Ear
263.                     x = leftShoulderPoint.x;
264.                     y = midHeadY;
265.                     while (x < centerMassX) {
266.                         pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.gre
    en(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create n
    ew pixel object using values.
267.                         if (pixel.getBlue() != 255 && x < centerMassX - 25) { //If pixel
     color is black. //fixfixfix
268.                             break;
269.                         }
270.                         x++;
271.                         if (x == centerMassX) {
272.                             x = leftShoulderPoint.x;
273.                             y -= 1;
274.                         } //ADDED THIS IN TO TRY TO FIX CRASHES
275.                         if (y == 0) {
276.                             x = 0;
277.                             y = 0;
278.                             break;
279.                         }
280.                     }
281.                     Point leftEarPoint = new Point();
282.                     leftEarPoint.set(x, y);
283.                     analyzedImage.setPixel(rightEarPoint.x, rightEarPoint.y, Color.GREEN
    );
284.                     analyzedImage.setPixel(leftEarPoint.x, leftEarPoint.y, Color.GREEN);
     //Check for tilted head.
285.                     int leftShoulderToEar = leftEarPoint.x - leftShoulderPoint.x;
286.                     int rightShoulderToEar = rightShoulderPoint.x - rightEarPoint.x;
287.                     boolean head_tilted_left = false;
288.                     String headTiltText = "No Posture Information Available.";
289.                     if (Math.abs(leftShoulderToEar - rightShoulderToEar) > SHOULDER_TO_E
    AR_THRESHOLD) {
290.                         head_tilted_left = leftShoulderToEar < rightShoulderToEar;
```

```
                        headTiltText = "Head tilt is outside the range of acceptable values. ";
                        if (head_tilted_left) {
                            headTiltText += "Your head is tilted to the left.\n";
                        } else {
                            headTiltText += "Your head is tilted to the right.\n";
                        }
                    } else {
                        headTiltText = "Your head and neck are looking excellent!\n";
                    }
                    resultsString.append(headTiltText); //Analyze Waist. //Left Hip
                    y = centerMassY; //Set x to x/4 of centerMassX. Can't do it in one operation for some reason.
                    x = centerMassX / 4;
                    x = centerMassX - x;
                    while (x < centerMassX) {
                        pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using values.
                        if (pixel.getBlue() != 255) { //If the pixel is black.
                            break;
                        }
                        x++;
                        if (x == centerMassX) { //Set x to x/4 of centerMassX. Can't do it in one operation for some reason.
                            x = centerMassX / 4;
                            x = centerMassX - x;
                            y++; //Try again at new Y coordinate.
                        } //ADDED THIS TO TRY FIXING CRASH.
                        if (y == height) {
                            break;
                        }
                    }
                    Point leftHipPoint = new Point();
                    leftHipPoint.set(x, y); //Testing.
                    analyzedImage.setPixel(leftHipPoint.x, leftHipPoint.y, Color.YELLOW); //Right hip
                    y = centerMassY;
                    x = centerMassX / 4;
                    x = centerMassX + x;
                    while (x > centerMassX) {
                        pixel = new PixelObject(Color.red(bmp.getPixel(x, y)), Color.green(bmp.getPixel(x, y)), Color.blue(bmp.getPixel(x, y)), bmp.getPixel(x, y)); //Create new pixel object using values.
                        if (pixel.getBlue() != 255 && x > centerMassX) { //If pixel is black. fixx
                            break;
                        }
                        if (x == 0) {
                            x = centerMassX / 4;
                            x = centerMassX + x;
                            y++;
                        } //ADDED THIS TO TRY FIXING CRASH.
                        if (y == height) {
                            break;
                        }
                        x--;
                    }
                    Point rightHipPoint = new Point();
                    rightHipPoint.set(x, y); //Testing
```

```
342.                     analyzedImage.setPixel(rightHipPoint.x, rightHipPoint.y, Color.YELLO
    W);
343.                     String hipText; //Check Hip Values
344.                     if (Math.abs((rightHipPoint.x - centerMassX) - (centerMassX - leftHi
    pPoint.x)) > HIP_WIDTH_THRESHOLD) {
345.                         hipText = "Hip values are outside of threshold.\n";
346.                     } else {
347.                         hipText = "Your hip posture is looking excellent!\n";
348.                     }
349.                     resultsString.append(hipText);
350.                     return analyzedImage; //Return an offset maybe of how offset the pos
    ture is?
351.                 }
352.                 /**     * Displays the results of the image analysis.     */
353.             public void displayResults() {
354.                 final ImageView modifiedImageView = (ImageView) findViewById(R.id.result
    View);
355.                 modifiedImageView.setImageBitmap(result);
356.                 TextView resultsView = (TextView) findViewById(R.id.resultsView);
357.                 resultsView.setText(resultsString.toString());
358.             }
359.         }
```

results.java

```
1.  package com.sublux.jayden.sublux;
2.  import android.content.Intent;
3.  import android.database.Cursor;
4.  import android.graphics.Color;
5.  import android.net.Uri;
6.  import android.os.Bundle;
7.  import android.provider.MediaStore;
8.  import android.support.v7.app.AppCompatActivity;
9.  import android.view.View;
10. import android.widget.Button;
```

```
11. import android.widget.ImageView;
12. import android.widget.TextView;
13. public class selectFromCameraRoll extends AppCompatActivity {
14.     private static final int PICK_IMAGE = 100;
15.     Uri imageUri;
16.     ImageView imageView;@
17.     Override protected void onCreate(Bundle savedInstanceState) {
18.         super.onCreate(savedInstanceState);
19.         setContentView(R.layout.activity_select_from_camera_roll); //Image View Button

20.         Button imageSelectButton = (Button) findViewById(R.id.imageSelectButton);
21.         imageView = (ImageView) findViewById(R.id.imageView);
22.         imageSelectButton.setOnClickListener(new View.OnClickListener() {@
23.             Override public void onClick(View view) {
24.                 openGallery();
25.             }
26.         }); //Image Analysis Button
27.         final Button analysisButton = (Button) findViewById(R.id.analysisButton);
28.         analysisButton.setOnClickListener(new View.OnClickListener() {@
29.             Override public void onClick(View view) {
30.                 if (imageUri != null) { //do stuff
31.                     analysisButton.setText("Analyzing...");
32.                     displayResults();
33.                 } //If there isn't an image selected, then prevent the app from crashin
    g and change the text color to alert the user.
34.                 else {
35.                     TextView txt = (TextView) findViewById(R.id.instructionText);
36.                     if (txt.getCurrentTextColor() == Color.parseColor("#E1315B")) {
37.                         txt.setTextColor(Color.parseColor("#FFEC5C"));
38.                     } else {
39.                         txt.setTextColor(Color.parseColor("#E1315B"));
40.                     }
41.                 }
42.             }
43.         });
44.     }
45.     private void displayResults() {
46.             Intent results = new Intent(selectFromCameraRoll.this, results.class);
47.             results.putExtra("imagePath", getImagePath(imageUri)); //Put Extra to acces
    s selected image on following screen.
48.             startActivity(results);
49.         } //Open a Gallery View
50.     private void openGallery() {
51.         Intent gallery = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.INTERNA
    L_CONTENT_URI);
52.         startActivityForResult(gallery, PICK_IMAGE);
53.     }@
54.     Override protected void onActivityResult(int requestCode, int resultCode, Intent da
    ta) {
55.             super.onActivityResult(requestCode, resultCode, data);
56.             if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
57.                 imageUri = data.getData(); //Get imageUri
58.                 imageView.setImageURI(imageUri); //Set imageView to selected image.
59.             }
60.         }
61.         /**     * https://stackoverflow.com/questions/20067508/get-real-path-from-uri-
    android-kitkat-new-storage-access-
    framework     * @param uri     * @return     */ //    public String getImagePath(Uri ur
    i){ //        Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    //        cursor.moveToFirst(); //        String document_id = cursor.getString(0); //
           document_id = document_id.substring(document_id.lastIndexOf(":")+1); //        c
```

```
ursor.close(); // //        cursor = getContentResolver().query( //                android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI, //                null, MediaStore.Images.Media._ID + " = ? ", new String[]{document_id}, null); //        cursor.moveToFirst(); // //        String path = cursor.getString(cursor.getColumnIndex(MediaStore.Images.Media.DATA)); //        cursor.close(); // //        return path; //    }
        /**     * https://stackoverflow.com/questions/20324155/get-filepath-and-filename-of-selected-gallery-image-in-android     * @param uri     * @return     */
    public String getImagePath(Uri uri) {
        String[] projection = {
            MediaStore.Images.Media.DATA
        };@
        SuppressWarnings("deprecation") Cursor cursor = managedQuery(uri, projection, null, null, null);
        int column_index = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);

        cursor.moveToFirst();
        return cursor.getString(column_index);
    }
}
```

selectFromCameraRoll.java