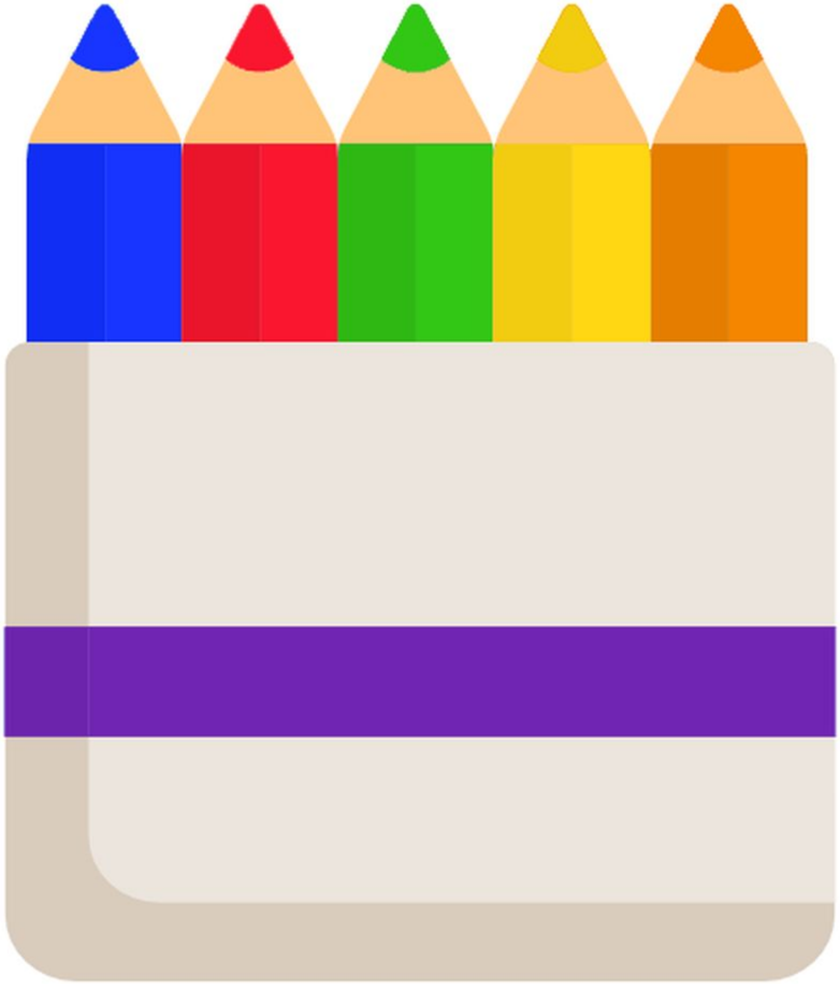


ImageWorks

By: Eric Johns, Eli Siron, and Elijah Thomas

Table of Contents

ImageWorks

Project ID: 2165

Spring-Ford High School
Royersford, PA

Eric Johns
eric275j@gmail.com

Elijah Thomas
elijahthomas774@gmail.com

Eli Siron
elisiron42@gmail.com

1. Overview

1.1 Objectives

The objective of this app is to bring novel image processing capabilities to your phone so a computer is not necessary when you want to manipulate images or create art.

1.2 Vision

The app was created to make image manipulation more convenient to everyone with a cell phone and to help perform minor image modifications without needing computer-based software such as photoshop or paint. We chose to create this app because we have a fascination with image processing and how we can use algorithmic processes to create cool results.

1.3 Purpose

The ImageWorks app will allow people to use image manipulation processes to create image art. The app will also let people perform image modifications in one app such as adjusting color channels, flipping, rotating, or mirroring images where you would typically need a computer or multiple apps to do otherwise.

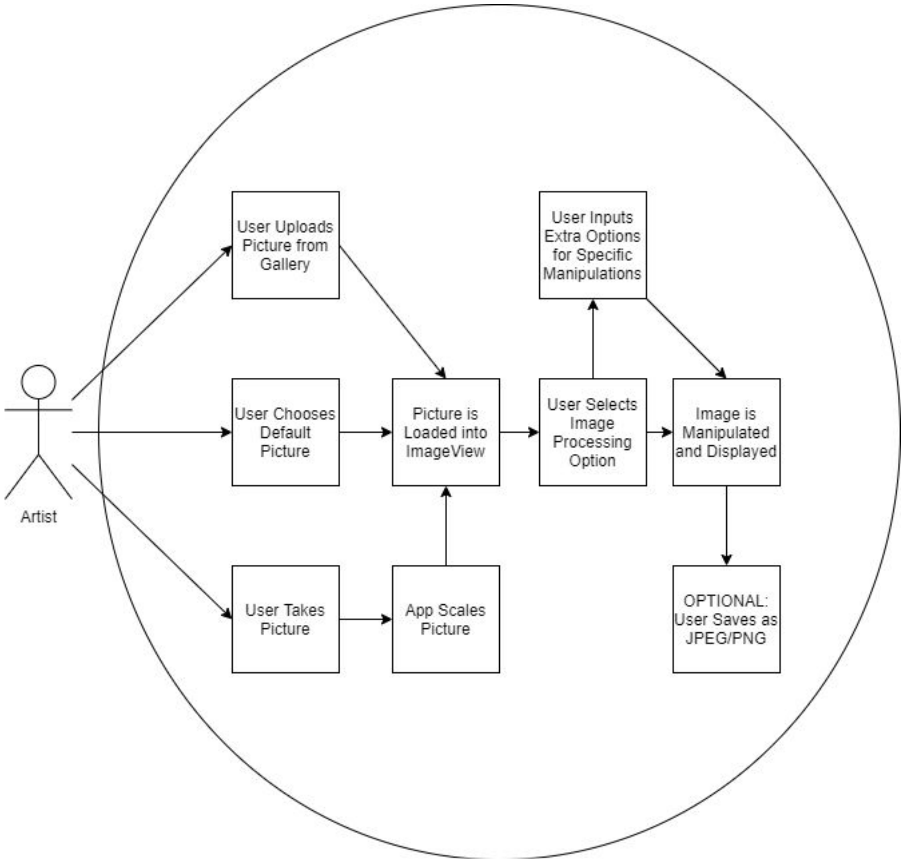
2 Implementation

2.1 Development Tools

This application was developed for iOS using Xcode. Xcode is an application that runs on mac that allows you to develop applications, run them on emulators or personal devices, and deploy straight to the app store. This development tool helped us create the app interface, program the apps functionality, and install the ImageWorks app on the test device. The app uses libraries such as Swift-Image and Toast-Swift to help handle converting integer arrays to pictures and to display output to the user.

2.2 Architecture

This application can be run on any iOS device including iPad and iPhones running iOS 10.0 or later. The use case diagram to the right demonstrates how the user interacts with the app. As you can see, first the user chooses or uploads a picture, which is then loaded into the app's ImageView. Next the user selects which kind of image manipulation process they would like to run, inputs any desired additional arguments (such as degrees for rotation or color channel to manipulate). The user also has the option to save the resulting image as a JPEG or PNG file if desired.



User Uploads Picture from Gallery
User Inputs Extra Options for Specific Manipulations
Artist
User Chooses Default Picture
Picture is Loaded into ImageView
User Selects Image Processing Option
Image is Manipulated and Displayed
User Takes Picture
App Scales Picture
OPTIONAL: User Saves as JPEG/PNG

2.3 Design

This app consists of two different views and numerous classes. The first view is ViewController, which is the main screen of the application and includes the ImageView, Test Features, and Pencil Sketch Button. The ImageView also allows zooming to allow users to see detail in photos and has a "Save Image" button on top. In the top right corner of the screen, there is a Pictures button which brings you to the picture selection view. In the Picture view, you can select from 16 pre-programmed photos, upload your own photos, or take a photo. The following table describes the numerous classes the app has and what each one does.

ViewController	This is the main class of the application that displays the images and includes the buttons for running image processing methods on your pictures.
PictureController	This is the image picker class which allows you to take your picture, choose your picture from gallery, or choose one of the pre-programmed pictures. This class then loads your image into the ImageView on the main screen.
Picture	This is the class that includes all the code for the image manipulations. This stores the image and calls methods to flip, mirror, rotate, or perform other processes on it.
Extensions	The Extensions class stores all extra add-ons needed to the swift language to fix issues with image orientation, create custom keyboards, and scale images down.
EdgeDetection	This class includes all the necessary methods to perform Canny Edge Detection on an image.
Sketch	The Sketch class includes all the methods needed to create a pencil sketched drawing of an image.

Prompt	This class includes the code for creating all the message dialogs that ask what kind of image manipulation you would like to perform.
ArrayUtility	Includes extra array manipulation methods needed to perform Edge Detection and Pencil Sketch without breaking the flow of the code.
Input	Stores all the user input when they are asked for extra arguments when performing specific image processing operations.

The UML diagram in Google Drive (it did not fit on the page) shows all the classes in the applications and what methods they contain.

3.Software Development Process

3.1.1 Functional Requirements

This app needed to be able to take a picture, upload a picture from gallery, or use pictures pre-programmed into the app to be able to manipulate the images. The app also needed to be able to save the results to the gallery as a jpeg (with compression) or png (without compression).

3.1.2 Design Process

The first part of development was finding a library that allowed us to take in pictures and manipulate them as an array. We eventually settled on swift-image because it allowed direct writing to the images pixel by pixel, and included many methods that made manipulating images easier. The next step was to find some test images that we could include in the app to demonstrate functionality. After adding pre-programmed images, we added the ability to take pictures and choose them from the gallery to load into the app. After we handled loading images into the app, it was time to start writing the image processing methods. After working on pixlab in AP Computer Science last year, we were able to reference some of our past code in order to write methods such as flipping, mirroring, converting to gray/binary, and all of edge detection. A powerpoint detailing the development of edge detection is also included in this binder. Pencil Sketch was the next feature we worked on. We referenced a research paper from graduate students in Hong Kong who developed the process. Using this paper, we were able to piece together parts of the process including drawing the sketched lines themselves. After all these functions were developed, we were able to begin refining the app to make it run smoother and have better image results.

3.1.3 Analysis and Test

This app was tested using Xcode’s built in iOS simulator on multiple virtual devices, our team’s personal phones, and a test iPhone 7+. We ran into numerous problems that we had to work through when we were working on this app. The first problem we had was when pictures were taken, the UIImageViews would rotate the images to the opposite orientation the picture was taken in. We fixed this issue by using an extension to correct pictures. Another issue we had was when taking pictures, they would be taken in 4k, which would be too high to get fast results with when running image processing. To account for this, when you take a picture it is scaled down to a fourth the original size. It takes longer to input the image after taking the picture, but now when you run image manipulation processes on it it runs significantly faster than it would have. To bypass the scaling, you can upload a picture of whatever size you want. In the future, we plan to add an option to disable image scaling. The biggest issue we ran into was the release of iOS 13. With iOS 13, the runtime of the application greatly increased which is the opposite of what an app like this needs. Also, when debugging to iOS 13 we found that Apple's built-in optimization was not performing as it should be. To fix these speed issues, we package the app for release instead of running it as a debug build on the devices, which in turn makes the app 34x faster than it was before.

3.1.4 Deployment

The app would have been loaded onto an iPhone 7+ running iOS 12 to display the app. This phone was picked because iOS 12 runs the swift-image library faster than iOS 13 does and shows test results faster. For online submission purposes, screenshots and screen recordings were created and placed into the “App Demonstration” folder in the project Google Drive.

3.1.5 Future Plans:

- Port to Android
- Implement encoding and decoding by steganography
- Implement affine transform
- Implement real-time filtering on the camera

3.2 Person hours:

Task	**Hours**	**Person**
Programming	200	Eric Johns
Pencil Sketch Components	60	Elijah Thomas
Poster	8	Eric Johns
Documentation	24	Eric Johns
Commenting/Documentation	5	Eli Siron
Testing/Debugging	12	Eli Siron

4. Templates (Dependencies).

This app has the following dependencies:

iOS SDK	Used to develop and test the app on iOS
swift-image	Used to manipulate images as if they were an array
Toast-Swift	Used to display output to the user without using a full message box

5. References

We got our **copyright free open source images** from Barb Erikkson's original Pixlab project, Google Images, and the authors of the Pencil Sketch Research Paper: Cewu Lu, Li Xu, Jiaya Jia.

Pencil Sketch Website:
http://www.cse.cuhk.edu.hk/~leojia/projects/pencilsketch/pencil_drawing.htm

Pencil Sketch Research Paper:
http://www.cse.cuhk.edu.hk/~leojia/projects/pencilsketch/npar12_pencil.pdf

Swift-Image Library:
https://github.com/koher/swift-image

Toast-Swift Library:
https://github.com/scalessec/Toast-Swift

Stackoverflow.com:

- https://stackoverflow.com/questions/29137488/how-do-i-resize-the-uiimage-to-reduce-upload-image-size
- https://stackoverflow.com/questions/24056205/how-to-use-background-thread-in-swift
- https://stackoverflow.com/questions/27272121/is-there-a-uitextfield-keyboard-type-that-like-a-number-pad-but-allows-negative

Wikipedia.com:

- https://en.wikipedia.org/wiki/Normalization_(image_processing)
- https://en.wikipedia.org/wiki/YUV

6. Operating Instructions

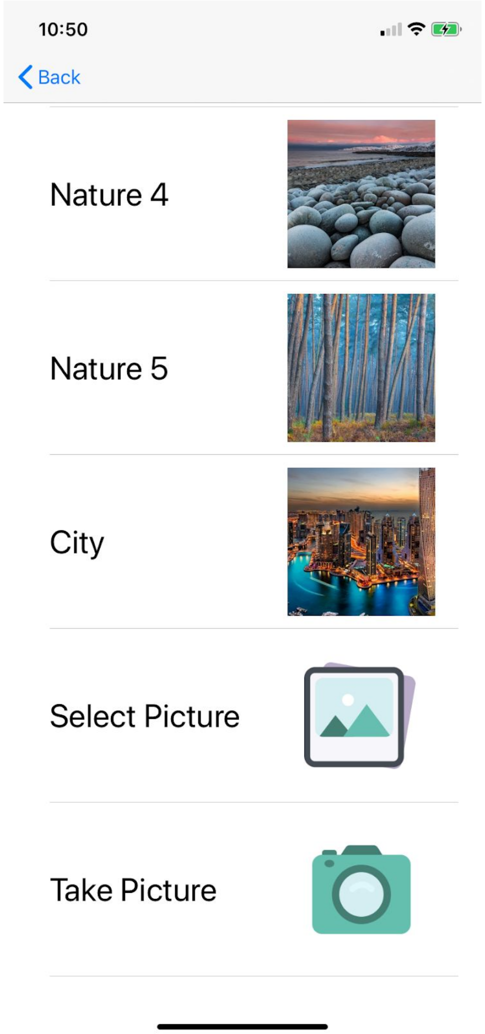
1) Picking Images

   Tap on the “Pictures” button in the top right corner of the main screen of the app.

   a) Tap on one of the pre-programmed images such as Nature 5 or City.
   b) Tap “Select Picture” to be brought to your gallery to upload a picture into the app.
   c) Select “Take Picture” to open your camera and take a picture to upload into the app.

2) Returning to the Main Screen

   a) Tap on the “Back” button in the top left corner of the screen.
   b) Drag your finger from the left edge of the screen to the right.

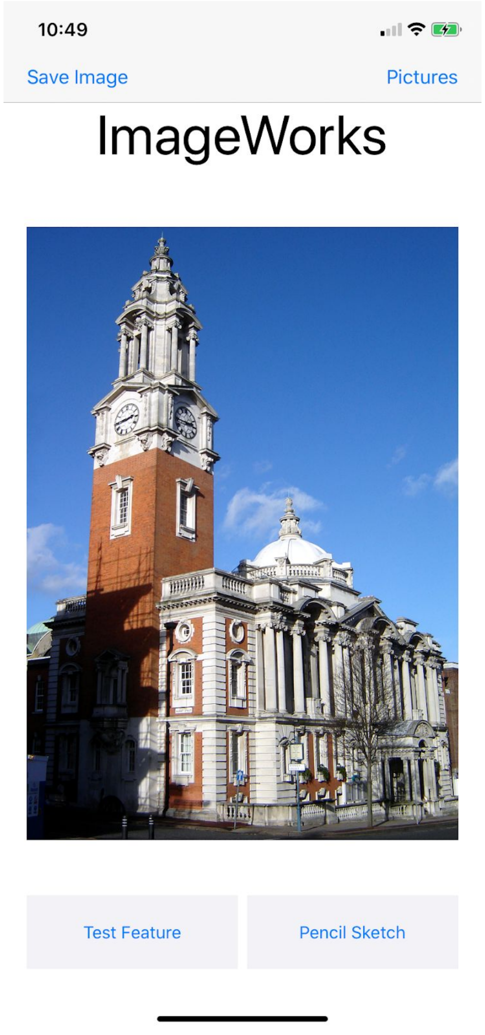


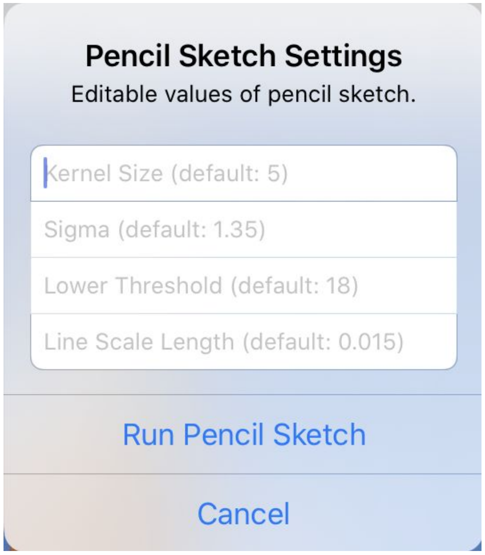
3) Saving an Image
    a) Tap on “Save Image” in the top right corner of the screen.
    b) Tap on “Save as JPG” or “Save as PNG” to save the image in the selected format to the phone’s gallery.

4) Zooming in on an Image
    a) Pinch in or out on the ImageView to zoom in or zoom out of the image.

5) Running Image Manipulation Features
    a) Tap the “Test Feature” button
    b) Tap on which test feature you would like to run.
    c) Some test features require extra user input, input extra arguments into the message boxes if necessary. The table below explains what each text field means.

6) Running Pencil Sketch
    a) Tap on the “Pencil Sketch” button and input the necessary user arguments. If you do not enter any values then it will run the sketch with default values. The table below explains what each text field means.
    b) Tap on “Run Pencil Sketch”




Pencil Sketch Settings

Editable values of pencil sketch.

Kernel Size (default: 5)

Sigma (default: 1.35)

Lower Threshold (default: 18)

Line Scale Length (default: 0.015)

Image Processing Feature	**Argument Explanation**
Channel Manipulation: Channel Values	The red, green, and blue fields represent a value from 0 to 255 that will become the color value for all pixels in that channel.
Channel Manipulation: Add Values	The red, green, and blue fields represent a value from -255 to 255 that will be added to all the pixels in that color channel.
Rotate	The degrees to rotate the image by (clockwise)
Resize	The resize factor is how much you want to scale the image by. 2x would make the image twice as big and ½ would make it half as big. The width and height fields scale the image to an exact size (may include some distortion).
Gradient Magnitude	Kernel Size is the radius of a gaussian filter to apply to the image for smoothing. Sigma is the intensity of smoothing you want to take place.
Edge Detection	Kernel Size is the radius of a gaussian filter to apply to the image for smoothing. Sigma is the intensity of smoothing you want to take place. Lower Threshold changes the bounds for what can be considered an edge. A lower value will include more edges in your final image and a higher value will include less edges.

Pencil Sketch	Kernel Size is the radius of a gaussian filter to apply to the image for smoothing. Sigma is the intensity of smoothing you want to take place. The Lower Threshold changes the bounds for what can be considered an edge. A lower value will include more edges in your final image and a higher value will include less edges. Line Scale Length is the percentage of the image height that will be multiplied to create the length of the sketched lines. A higher value will make the sketch rougher.

7. Copyrights

All of the images in this app are open source and do not have a copyright. The images were provided by the research paper authors Cewu Lu, Li Xu, and Jia, by Barb Erikson through the Pixlab project, and through copyright free sites on Google Images and Bing Images.

8. Pseudocode

```
ViewController:
    create variables

    viewDidLoad()
        load default images

        pull default input values from file

        set up user gestures for showing logs and resetting images

    changePicture()
        if picture is pre-programmed:
            load picture from file

        else if picture is loaded from gallery:
            choosePicture()

        else:
            takePicture()

    resetImage()
        ViewController picture is set to the original image before manipulation
        clear log

    showLog()
        displays log of what manipulations were performed on an image

    saveImage()
        create dialog to save as a jpg or png
        display to user
        save image in format based on what the user picked

    pencilSketch(sender: Any)
        present the pencil sketch argument dialog
        manipulateImage()
        testClick(sender: Any)
                present the test features dialog for the user to pick an option from
                prompt for extra input if necessary
                manipulateImage()

        manipulateImage(manipulation: Manipulation)
                start timer

                open background thread:
                        manipulate the image in background so the app does not freeze

                        return to main thread:
                                display the manipulated image
                                add the manipulation to log

        choosePicture()
                open gallery so user can pull a picture from it

        takePicture()
                open the camera to take a picture

ArrayUtility:
        flipHorizontal(array: [[Int]])
                create a copy of input array

                for y in 0 to height
                        for x in 0 to width / 2
                                swap array[y][x] with array[y][width - x- 1]
                return copy of input array

        flipVertical(array: [[Int]])
                create a copy of input array

                for y in 0 to height / 2
                        for x in 0 to width
                                swap array[y][x] with array[height - y - 1][x]
                return copy of input array
```

```
binary(image: Image<RGBA<UInt8>>)
    instantiate array with same dimensions as image

    loop through each pixel
        red = red value * 0.21              // weighted average
        green = green value * 0.72
        blue = blue value * 0.07

        if red + green + blue > 127
            pixel = white
        else
            pixel = black
    return array

toIntArray(array: [[Double]])
    instantiate int array of same dimensions as input

    loop through all pixels
        convert double value to an integer
    return array

toDoubleArray(array: [[Int]])
    instantiate double array of same dimensions as input

    loop through all pixels
        convert int value to a double
    return array

negate(array: [[Int]])
    instantiate array of same size as input

    loop through all pixels
        set pixel value to 255 - current value
    return new array

normalize(array: [[Int]])
    instantiate new array that is the same size as input
    max and min = value at (0, 0)
    loop through each pixel
        if max < pixel
            max = pixel
        else if min > pixel
            min = pixel

    loop through each pixel again
        normalized pixel = 255 / (max - min) * (value - min)
    return normalized array


normalize(array: [[[Int]]])
    instantiate new array that is the same size as input
    max and min = value at (0, 0)

    loop through each color channel   // must normalize each channel separately
        loop through each pixel
            if max < pixel
                max = pixel
            else if min > pixel
                min = pixel

        loop through each pixel again
            normalized pixel = 255 / (max - min) * (value - min)
    return normalized array

copyRange(image1: [[Int]], copyTo: [[Int]], startX: Int, endX: Int, startY: Int, endY: Int, pasteX: Int, pasteY: Int)
    instantiate copy array of size endX - startX by endY - startY

    for y in startY to endY
        for x in startX to end X
            copyPortion at (y - startY, x - startX) = pixel at x, y

    for y in pasteY to minimum of copyPortion + pasteY and image end
        for x in pasteX to minimum of copyPortion + pasteX and image end
            copyTo = pixel at copy portion in relation to the image
    inBounds(i: Int, j: Int, y: Int, x: Int)
        if i, j is in bounds of 0 to y and 0 to x
            return true
        else
            return false

    minimum(array: [[Int]])
        min = array[0][0]

        loop through every pixel
            if array at y, x is less than min
                min = array at y, x
        return min

    maximum(array: [[Int]])
        max = array[0][0]

        loop through every pixel
            if array at y, x is greater than max
                max = array at y, x
        return max


PictureCell:
    declare a pictureName variable to hold a name
    declare a picture variable to hold the image icon


PictureController:
    declare lists of image names to display in tableView

    viewDidLoad()
        immediately select row of current image

    tableView(tableView: UITableView, numberOfRowsInSection section: Int)
        set size of tableView to the length of the images lists

    tableView(tableView: UITableView, cellForRowAt indexPath: IndexPath)
        create a tableview cell with an image and label
        set the image to be an icon of the image
        set the label to be the name of the image
        return the cell

    tableView(tableView: UITableView, didSelectRowAt indexPath: indexPath)
        change picture on the main screen to be what the user selected

    tableView(tableView: UITableView, heightForRowAt indexPath: IndexPath)
        set cell height to be 150


EdgeDetection:
    createBorder(kernel: Int, image: [[Int]])
        instantiate array of size image height + 2kernel, width + 2kernel

        flipVertical = ArrayUtility.flipVertical image

        copy original image into border reflected array
        copy top of flipVertical to the top of border reflect
        copy bottom of flipVertical to bottom of border reflect

        flipHorizontal = ArrayUtility.flipHorizontal image

        copy left of flipHorizontal to border reflect
        copy right of flipHorizontal to border reflect

        return border reflected array

    createKernel(kernel: Int, sigma: Double)
        create array of size 2kernel + 1 by 2kernel + 1
        create a coefficient variable of value $1 / (2\pi\text{^}(sigma^2))$
        sum = 0

        loop from -kernel to kernel + 1 on both sides
                filter = $coefficient \text{ ^ } ( -(u^2 + v^2) / (2sigma^2)$
                sum = sum + filter at u + kernel, v + kernel
        loop from 0 to filter size in both x and y
            filter at y, x = filter at y, x divided by sum
        return filter
```

```
filter(kernel: [[Double]], image: [[Int]])
    k = kernel length / 2
    filteredImage = array of image size - 2k on each side

    loop through image from k to image size - k in both dimensions
        loop through the filter
            average surrounding pixels using the filter
    return filtered image

calcGradientX(image: [[Double]])
    gradX = new array 1 pixel shorter than image

    loop through image - 1 in both dimensions
        gradX at y,x = image (y, x + 1) - image (y, x)
    return gradX

calcGradientY(image: [[Double]])
    gradY = new array 1 pixel shorter than image

    loop through image - 1 in both dimensions
        gradY at y,x = image (y + 1, x) - image (y, x)
    return gradY

calcGradMag(gradX: [[Double]], gradY: [[Double]])
    instantiate new array of similar size to gradX and gradY

    loop through entire array
        gradMag at y, x = $\sqrt{(x^2 + y^2)}$
    return gradMag

calcGradAngle(gradX: [[Double]], gradY: [[Double]])
    instantiate new array of similar size to gradX and gradY

    loop through entire array
        gradAng at y, x = atan2(gradY, gradX)
    return gradAng

adjustGradAngle(gradAngle: [[Double]])
```

```
        instantiate new array that is the same size as gradAngle

        loop through entire gradAngle array
            if gradAngle is negative:
                add pi to angle

            if gradAngle is between $3\pi/4 + \pi/8$ and $3\pi/4 - \pi/8$:
                bin angle to group 3
            else if gradAngle is between $\pi/2 + \pi/8$ and $\pi/2 - \pi/8$:
                bin angle to group 2
            else if gradAngle is between $\pi/4 + \pi/8$ and $\pi/4 - \pi/8$:
                bin angle to group 1
            else if gradAngle is between $\pi/8$ and $-\pi/8$:
                bin angle to group 0
            else:
                bin angle to group 0
        return adjust gradient angles

    calcNonMaxSupp(gradMag: [[Double]], gradAngleAdj: [[Int]])
        create new array that is the same size as gradMag

        loop through entire array:
            switch gradAngleAdj:
                case 0:
                    if gradMag is greater than magnitude of the left and right pixels:
                        calcNonMaxSupp = gradMag at y, x
                case 1:
                    if gradMag is greater than magnitude top right and bottom left pixels:
                        calcNonMaxSupp = gradMag at y, x
                case 2:
                    if gradMag is greater than magnitude of the top and bottom pixels:
                        calcNonMaxSupp = gradMag at y, x
                case 3:
                    if gradMag is greater than magnitude of the bottom right and top left pixels:
                        calcNonMaxSupp = gradMag at y, x
        return calcNonMaxSupp

    doubleThresh(nonMaxSupp: [[Double]], min: Int, max: Int)
        create new array that is the same size as calcNonMaxSupp

        loop through entire nonMaxSupp array:
            if nonMaxSupp > max
                doubleThresh = 255
            else if nonMaxSupp <= max and >= min
                doubleThresh = 127
        return doubleThresh

    calcEdge(doubleThresh: [[Int]])
        create new array that is the same size as doubleThresh

        loop through 1 to array length - 1 in both dimensions:
            if doubleThresh == 127:
                loop through surrounding pixels:
                    if any are 255, set pixel at y, x to 255
            else:
                pixel at y, x = 255

    removeFinalBorder(calcEdge: [[Int]])
        create new array that is the same size as calcEdge - 2

        loop through 1 to array length - 1 in both dimensions:
            copy pixels over to new array
        return final image array
```

```
Prompt:
    controller: ViewController
```

```
init(controller: ViewController)
    self.controller = controller

promptFeatures()
    create message dialog

    add actions for all the test features

    return feature dialog

promptGeneral()      // a lot of the prompt methods are repetitive, here is a general one without
    create dialog for specific test category

    add buttons for each specific test feature

    add cancel buttons

    return dialog

promptWithInput() // some prompts require extra input, here is a general dialog with text fields
    create base dialog

    create resize factor alert action
        pull text field from dialog

        if text is empty
            use default value
        else
            pull in user input

        save user input in Input class

        run test manipulation

    add textfield with a placeholder and number/decimal pad to dialog

    add alert action to dialog
```

```
        return base dialog

    prompt(title: String, message: String)
        create dialog with title "title" and message "message"
        add default close buttons
        return dialog

    func testFeature(alert: UIAlertAction!)
        run manipulation of image

    func dialog(alert: UIAlertAction!)
        open dialog for a manipulation sub category


Input:
    declare a field variables to hold input collected in the prompt class
    these variables will be referenced when image manipulation processes are ran


Picture:
  declare image field variable

  init()
      image = default townhall image

  init(name: String)
      instantiate image with resource named "name"

  init(image: Image<RGBA<UInt8>>)
      self.image = image

  init(width: Int, height: Int)
      instantiate image with inputted width and height

  init(array: [[Double]])
      instantiate image with pixels from double array

  init(array: [[Int]])
```

```
    instantiate image with pixels from int array

getHeight()
    return image height

getWidth()
    return image width

getImage()
    return image object

getUIImage()
    return image as UIImage

setPicture(name: String)
    set image to be picture resource with name "name"

setPicture(image: Image<RGBA<UInt8>>)
    set image to image input

setPicture(image: UIImage)
    set image to be converted UIImage

setArray(array: [[Int]])
    load grayscale pixel array into image

setArray(array: [[[Int]]])
    load color pixel array into image

rotate(degrees: Int)
    image = image.rotated(degrees)

resize(factor: Double)
    newWidth = width * factor
    newHeight = height * factor

    image = image.resizedTo(width, height)

resizeExact(width: Int, height: Int)
```

```
        image = image.resizedTo(width, height)

convertToGray()
        loop through all pixels
                red = current red * 0.21    // weighted average
                green = current green * 0.72
                blue = current blue * 0.07

                gray = red + green + blue

                set all pixel color channel values to gray

squareImage()
        if image width != image height
                if image width > image height
                        cropPortion = (width - height) / 21
                        imageSlice = cropPortion to width - cropPortion, 0 to height
                else
                        cropPortion = (height - width) / 21
                        imageSlice = 0 to width, cropPortion to height - cropPortion

                image = imageSlice

flipDiagonal()
        if image width != image height
                squareImage()

        loop through all y coordinates
                loop through all x coordinates to current y
                        swap pixel at (x, y) with pixel at (y, x)

flipVertical()
        image = image.yReversed()

flipHorizontal()
        image = image.xReversed()

channelValue(channel: Int, value: Int)
        loop through all pixels
```

```
            set all values in channel "channel" to value

addValues(channel: Int, value: Int)
    loop through all pixels
            newValue = current channel value + value

            if newValue > 255
                    newValue = 255
            if newValue < 0
                    newValue = 0

            new channel value = newValue

addAll(red: Int, green: Int, blue: Int)
    loop through all pixels
            redValue = current red + red
            greenValue = current green + green
            blueValue = current blue + blue

            check if all values are between 0-255
                    adjust to closest bounds if not

            red pixel = redValue
            green pixel = greenValue
            blue pixel = blueValue

swapChannel(channel1: Int, channel2: Int)
    loop through all pixels
            switch value in channel1 with value in channel2

keep(channel: Int)
    loop through all pixels
            switch channel
                    case 0:
                            zero green and blue channels
                    case 1:
                            zero red and blue channels
                    case 2:
                            zero red and green channels
```

```
mirrorVertical(top: Bool)
    loop through all x values
        loop through all y values / 2
            if top
                set pixel at (x, y) to value of pixel at (x, height - y - 1)
            else
                set pixel at (x, height - y - 1) to value of pixel at (x, y)

mirrorHorizontal(left: Bool)
    loop through all x values / 2
        loop through all y values
            if left
                set pixel at (x, y) to value of pixel at (width - x - 1)
            else
                set pixel at (width - x - 1, y) to value of pixel at (x, y)

mirrorDiagonal(leftRight: Bool)
    if image width != image height
        squareImage()

    loop through all x values
        loop through y values from 0 to height - current x
            if leftRight
                set pixel at (width - x - 1, height - y - 1) to value of pixel at (y, x)
            else
                set pixel at (y, x) to value of pixel at (width - x - 1, height - y - 1)

negate()
    loop through all pixels
        new red pixel value = 255 - current red
        new green pixel value = 255 - current green
        new blue pixel value = 255 - current blue

convertToBinary()
    loop through all pixels
        red = current red * 0.21    // weighted average
        green = current green * 0.72
        blue = current blue * 0.07

        gray = red + green + blue

        if gray > 127
            pixel = 255
        else
            pixel = 0

convertArray()
    instantiate 2 dimensional array with same dimensions as image

    loop through all y and x values
        array at (x, y) = gray pixel at image (x, y)

convertArrayColor()
    instantiate 3 dimension array with same dimensions as image

    loop through y and x values
        loop through all color channels
            color channel value at (x, y) = image channel value at (x, y)

manipulateImage(manipulation: Manipulation)
    switch (manipulation)
        case gradMag
            array = convertArray()
            borderReflect = createBorder(k, array)
            kernel = createKernel(k, sigma)
            filter = filter(kernel, borderReflect)
            gradX = calcGradientX(filter)
            gradY = calcGradientY(filter)
            gradMag = calcGradMag(gradX, gradY)
            setArray(normalize(toIntArray(gradMag))
        case edgeDetection
            array = convertArray()
            borderReflect = createBorder(k, array)
            kernel = createKernel(k, sigma)
            filter = filter(kernel, borderReflect)
            gradX = calcGradientX(filter)
            gradY = calcGradientY(filter)
            gradMag = calcGradMag(gradX, gradY)
            gradAngle = calcGradAngle(gradX, gradY)
            gradAngleAdj = adjustGradAngle(gradAngle)
            calcNonMaxSupp = calcNonMaxSupp(gradmag, gradAngleAdj)
            doubleThresh = doubleThresh(normalize(calcNonMaxSupp), min, min * 3)
            calcEdge = calcEdge(doubleThresh)
            setArray(calcEdge)
        case sketch
            array = convertArray()
            borderReflect = createBorder(k, array)
            kernel = createKernel(k, sigma)
            filter = filter(kernel, borderReflect)
            gradX = calcGradientX(filter)
            gradY = calcGradientY(filter)
            gradMag = calcGradMag(gradX, gradY)
            gradAngle = calcGradAngle(gradX, gradY)
            gradAngleAdj = adjustGradAngle(gradAngle)
            calcNonMaxSupp = calcNonMaxSupp(gradmag, gradAngleAdj)
            doubleThresh = doubleThresh(normalize(calcNonMaxSupp), min, min * 3)
            sketch = sketch(doubleThresh, gradAngle, lineScale)
            setArray(negate(normalize(sketch))
        case sketch2
            array = convertArray()
            borderReflect = createBorder(k, array)
            kernel = createKernel(k, sigma)
            filter = filter(kernel, borderReflect)
            sketch = sketch(filter, lineScale, min, min * 3)
            blendedSketch = addSketch(filter, normalize(sketch))
            setArray(normalize(blendedSketch))
        case sketchColor
            array = convertArray()
            borderReflect = createBorder(k, array)
            kernel = createKernel(k, sigma)
            filter = filter(kernel, borderReflect)
            sketch = sketch(filter, lineScale, min, min * 3)
```

```
                blendedSketchColor = addSketch(brightnessChange(convertArrayColor, 1.35, normalize(sketch))
                setArray(normalize(blendedSketchColor))
            other cases without input:
                run manipulation on image
            other cases with input:
                run manipulation on image using values called from the Input class

Manipulation:
    create enums of Manipulation names assigned to integers
    // helps with readability in the code

Sketch:

sketch(dThresh: [[Int]], gradAng: [[Double]], lineScale: Double)
        create new Int array the same size at the dThresh array
        create the length of the sketch lines as the sqrt(number of pixels in image) * lineScale

        loop through y and x of new array
            if dThresh at (x,y) is 127 or 255
                addLine(length of lines, new array, gradAng at (x,y) + pi/2, y, x)
            if dThresh at (x,y) is 255
                addLine(length of lines, new array, gradAng at (x,y) + pi/2, y, x)
        return the new array with lines added

sketch2(filtered: [[Double]], lineScale: Double, dThreshMin: Double, dThreshMax: Double)
        declare necesary variables for edge detection
        x is width of array and y is height

        create a new arrays: dthresh for holding dThresh values and Sketch for final array

        loop through y and x of sketch array represented as i and j
            angle is tanget of ((i+1,j)-(i,j)) / ((i, j+1) - (i,j))      with values from filtered at the respective points
            bin angle into 4 sections utiliting truncation ((pi/4)*truncated(4.5 + (4 * angle / pi))
            set sine and cosine to the functions of the angle and round (should be -1, 0, or 1)
            make other reference points for non-maximum supression
            r is i + sine, c is j + cosine
            u is i - sine, v is j - cosine

            num1 is the magnitude of the angle at (i,j)
            if r and c are in domain of original image array
                num2 is made the magnitude of the angle at (r,c)
            else
                r is made i
                c is made j
                num2 is made as a copy of num1
            repeat the same things for u and v as r and c, but for num3

            set maximum as the max of num1, num2, and num3

            if num1 is the maximum
                dtresh at (i,j) is made the maximum value
            if maxNum is greater than maximum
                maxNum is made the same value of maximum


        create the length of the sketch lines as the sqrt(number of pixels in image) * lineScale

        loop through array with i and j again
            thresh at point is made value of dthresh at (i,j) * truncated value of 255/maxNum
            slope is made the tangent + pi/2 of filtered ((i+1,j) - (i,j))/((i,j+1)-(i,j))

            if thresh at is 127 or 255
                addLine(length of lines, sketch array, slope, i, j)
            if thresh is 255
                addLine(length of lines, sketch array, slope, i, j)

        return the sketch array with lines added
```

```
addLine(length: Int, array: inout [[Int]], ang: Double, r: Int, c: Int)
    if length is even
        add one to length to make it odd

    slope is tangent of ang

    if absolute value of slope is greater than 1
        loop  as i through length centered, but at -half of length to +next half of length
            y is r + i
            x is slope * c

            if x and y are in bounds of picture
                array at (y,x) gets 1 added to it
                if x - 1 is also in bounds
                    array(y,x-1) gets 1 added to it
    else
        loop  as i through length centered, but at -half of length to +next half of length
            y is slope * r
            x is c + i

            if x and y are in bounds of picture
                array at (y,x) gets 1 added to it
                if y - 1 is also in bounds
                    array(y-1,x) gets 1 added to it
brightnessChange(rgb: [[[Int]]], multiplier: Double)
    make new array same size as rgb
    loop through y and x of new array
        r is the red value/255
        g is the green value/255
        b is the blue value/255
        y is the scaling values for each r g and b, then multiplied by the multiplier
        u is the scaling values for each r g and b
        v is the scaling values for each r g and b
        the red component in the new array will be the conversion back to red from yuv
        the green component in the new array will be the conversion back to green from yuv
        the blue component in the new array will be the conversion back to blue from yuv
    return the new array with the adjusted rgb values

addSketch(image: [[[Int]]], sketch: [[Int]])
    make new array same size as image
    loop through the array and color channels as y x and k
        the new image at (x,y,k) is the original image at the same point - sketch at the point
        if the new image at the point is negative
            make the point a value of 0
    return the new image

addSketch(gray: [[Double]], sketch: [[Int]])
    make new array same size as image
    advNorm(negate(gray), 8, 255)
    loop through the array as y and x
        the new image at (x,y) is the gray image at the same point - sketch at the point
        if the new image at the point is negative
            make the point a value of 0
    return the new image

advNorm(arr: [[Int]], maxNew: Int, minNew: Int)
    min is the minimum of the arr
    max is the maximum of the arr

    loop through y and x of arr
        arr at (x,y) is the normalized formula from wikipedia
    return normalized array
```