# PICCA - An image processing application for mobile devices

**By**

**Cole Pandya**

**1506492**

**Cpan040@aucklanduni.ac.nz**

# Abstract

On some occasions, a user using a camera to take a picture does not have ideal conditions to ensure optimal quality of the picture and providing a way to enhance the images would be time efficient and cost effective. Even though there is a vast amount of “image-processing” applications available to download, majority of them only consist of artistic effects such as sepia or black and white etc but the essence of image restoration and enhancement is lost. To solve this problem, I have developed Picca, which is a free image processing application for all Android devices with a screen size of 5” or more and consists of more than 20 filters that are not widely available in image processing application. If the users do not wish to capture an image from the camera, they can also load an image from the device library which are displayed as thumbnails in a grid layout and then apply filters. A bitmap is used to the store an image in the application and the bitmap configuration used throughout the application is called ARGB. Each pixel value of the image contains colour information and in the ARGB bitmap configuration, the channel sample is defined by 8 bits, and are arranged in memory in such manner that a single 32-bit unsigned integer has the Alpha sample in the highest 8 bits, followed by the Red sample, Green sample and the Blue sample in the lowest 8 bits. Some of the filters are designed using the convolution matrix to promote reusability and flexibility of the code. Filters like “Sharpen” must be used after an image is being smoothened by “Gaussian blur” so that the noise does not get highlighted. The application promotes multiple filter application on one image and this allows users to have many permutations of the filters which result in many restoration and enhancement effects. To support decision making for selection of filters, the application consists of a RGB histogram feature that displays separate histograms of each colour channel to show the spread of intensities. The application solves the issue of image restoration and image enhancement for Android devices.

# Table of Contents

# Introduction

Mobile applications were initially developed for general productivity and retrieval of information such as email, weather report, stock market etc. However, due to the rapid growth of mobile technology the public demand and developer tools led to expansion of mobile games and social applications. The popularity of mobile applications has continued to rise, as their usage has become increasingly common across mobile phone users.

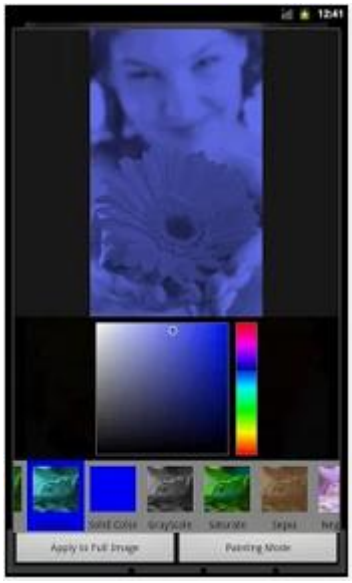
My Btech project is to develop an image processing application for Android Platform by developing enhancement and restoration filters for the images captured or supplied by the user. This can be achieved by mathematical algorithms that are quick and efficient but also useful for this purpose. Providing users with popular functionalities of desktop image processing applications would definitely be an advantage and will encourage users to use my application. The application will be extremely usefully to users who have a basic understanding of the RGB values and how they can analyze them to make filter choices.

The filters range from basic ones such as brightness and contrast adjustment to more sophisticated algorithms such as Image Equalization and Convolution filter.

Image restorations filters will be used to minimize the effect of degradation. This process is heavily dependent on the type of degradation process I use as well the quality of an image. Image enhancement is different to this process because enhancement of image involves more extraction of image features. This report is a guide that provides step by step process for the development of my application and also provides detailed reasoning for filter use and its features.

# Related Work

Image processing is common for smart devices and there are probably thousands of applications available on Google Play for Android users to download but I still wish to develop a free-to-download image processing application. There are many reasons for this; the main one is that free applications like “Photo Art – Colour Effects” or “Mytubo” and many others provide no image restoration capabilities or proper image enhancement facilities. These types of applications only provide ‘arty’ effects which barely even enhance the detail of the image. “Vignette Demo” is an application which provides in-depth camera features and about 80+ filters such as Sepia, Monochrome, and Vintage etc are widely available in majority of applications. Having said that, they still do not enhance the image, they are basically applying ‘arty’ effects on images so that they could be shared on social media. The reason why they use artistic effects is because they are extremely quick to apply whereas, the image restoration and heavy enhancement takes slightly longer (Barloso, 2012).



“Image Processing Camera” is an Android application for $1.56NZD https://play.google.com/store/apps/details?id=kerokawa.jp.imageprocessingcamera but only provides around 6 basic features and is not a fully fledged image processing application for that price. Second reason for me to develop an image processing application is that, my application will be free of cost but will still consist of over 20 filters that enhance or restore images. The only downside is that the complexity of the algorithm will determine the time to apply the filters. The performance aspect is discussed more in the Evaluation section of this report (Barloso, 2012).

“Photo Editor – Fotolr” is a better example of an image processing application because of the usefulness of the filters and it consists of features such as album function and photo sharing. The important feature of this application is the Makeover section which basically allows the user to remove inconsistencies of the person in the image, not the image itself. For example, acne removal is a filter offered that basically serves as a “beauty filter” and usually these types of filters are available only on PC applications that are not always free.

All the applications that I mentioned above have great UI and perhaps my application may not offer that as of now due to the time constraint but my main priority is to provide image processing functionalities first and then worry about designing a sophisticated UI interface. I have many mock-ups for the UI (later in the report) and the final selection is quite user-friendly but there is always room for improvement (Barloso, 2012).

The difference between my application and the rest is the selection of filter offered. My filters are implemented using algorithms that actually restore images or enhance them and I have decided to implement some popular filters such as Hue, Saturation and Value/Brightness from PC applications. I decided to leave the basic functions such as cropping etc as they are already offered by almost all the smart devices equipped with a camera. Also, major difference will be the RGB display feature which will help users with the decision making and also to see the difference in distribution of intensities after applying the filters. Also I have discussed uses of these filters later in the report.

There are thousands of image processing applications available today but most of them lack the essence of image restoration and mostly include the artistic effects which do not serve any purpose except sharing on social media or save to the media folder. This is where my application will be different and also free to download to help users restore, enhance and save images (Barloso, 2012).

# Research on filters and their use:

## Image Equalization (Histogram Equalization)

This is one of the filters that I will be implementing for Picca and this is a non-linear mapping of pixel-wise intensities in order to flatten the distribution of pixel intensities of the image histogram. This advantages of this filter is that it will cause the dynamic range of the image to increase which would lead to an increase in image contrast and also this filter can be applied on images with either dark or bright backgrounds and foregrounds and can also reveal details that are usually not shown. However, disadvantage is that this filter sometimes outputs unrealistic images and this is dependent on the input image and therefore a user who understands when the filter is to be used will be able to use the application efficiently (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

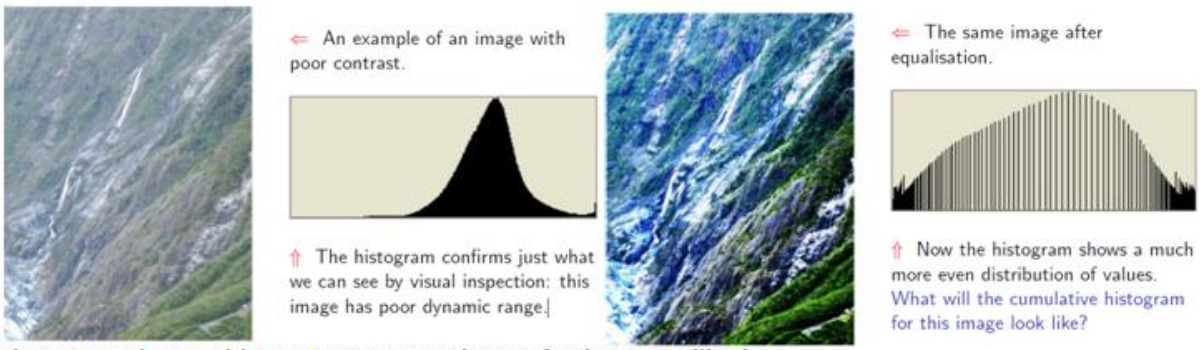



**Figure 11: An image with poor contrast VS an image after image equilisation**

As shown below, Image equalization involves mapping the initial distribution of pixel intensities to a wider and more uniform distribution so the intensity values are spread over the whole range. To accomplish the equalization effect, the remapping should be the cumulative distribution function (cdf) (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

Given an image $f$ and its histogram $H = (H(q) : q = 0, 1, \ldots, Q)$:

**1: Compute the cumulative histogram $C$**

```
C[0] = H[0]
for q = 1,...,Q do C[q] = C[q-1 ]+ H[q]
```

**2: Convert $C$ into the LUT (lookup table) $T$**

```
for q = 0,...,Q do
   T[q] = Q * ( C[q] - C[0] ) / ( C[Q] - C[0] )
```

**3. Using the LUT $T$, transform $f$ into the equalised image $g$**

```
for all pixels (x,y) do g[x,y] = T[ f[x,y] ]
```

**Figure 12: Image equalisation algorithm**

## Median Filter

Median filtering is a nonlinear method used to remove noise from images. The advantage of this filter is that it is efficient and very effective at removing ‘salt and pepper’ noise without hurting the edges. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. The pattern of neighbours is called the "window", which slides, pixel by pixel over the entire image (Gimel'farb & Delmas, Part 3: Image Processing - 3.2. Image Filtering and Segmentation).

The main idea of the median filter is to use current pixel intensity, replacing each entry with the median of neighbouring entries. The ‘window’ is the pattern of neighbours and it slides through the image, over the entire image. For 1D signal (left image below), window is the first few preceding and following entries. In my case, the signal is 2D (right image below) as it’s an image; more complex window patterns are possible such as "box" or "cross" patterns (Gimel'farb & Delmas, Part 3: Image Processing - 3.2. Image Filtering and Segmentation).

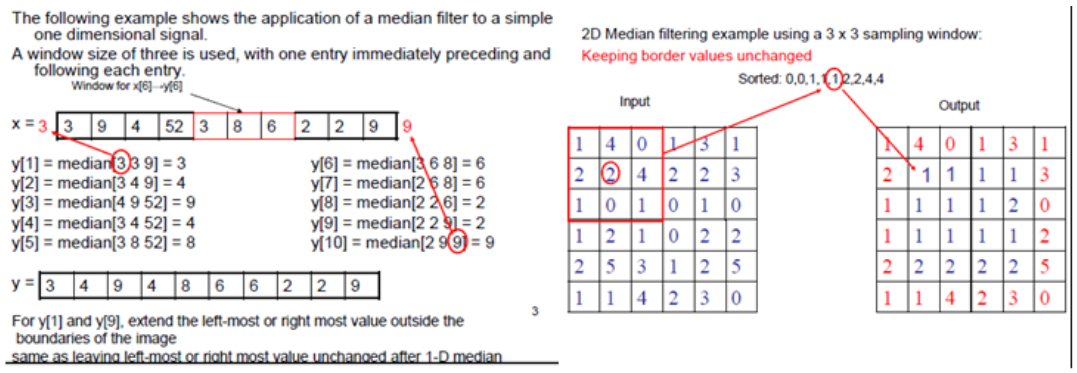



**Figure 13: 1D Median filter process VS 2D Median Filter process**

### 2D signal algorithm:

The 2D implementation will be used to design my median filter but I plan on not to pad the edges with specific values, strictly due to the slight loss of responsiveness. The image is quite large and the user will not be able to notice the difference in the edges because the median filter effect is subtle.

## Mean or Average filter

Mean filtering is another method used for ‘smoothing’ images by reducing the amount of intensity variation between neighbouring pixels. This is achieved by moving through all the pixels, replacing each value with a mean value of neighbouring pixels. When the filter neighbourhood straddles an edge, the filter will interpolate new values for pixels on the edge and so will blur that edge. Disadvantage to this filter is that it could be a problem if sharp edges are required in the output (Gimel'farb & Delmas, Part 3: Image Processing - 3.2. Image Filtering and Segmentation).

Mean filtering is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown below but can use higher square kernel for more severe smoothing. A small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel (Gimel'farb & Delmas, Part 3: Image Processing - 3.2. Image Filtering and Segmentation).

The mean filter (along with many other filters for my application) can be designed using a Convolution matrix and all I would be changing is the kernel matrix and the size (8.2 Convolution Matrix).

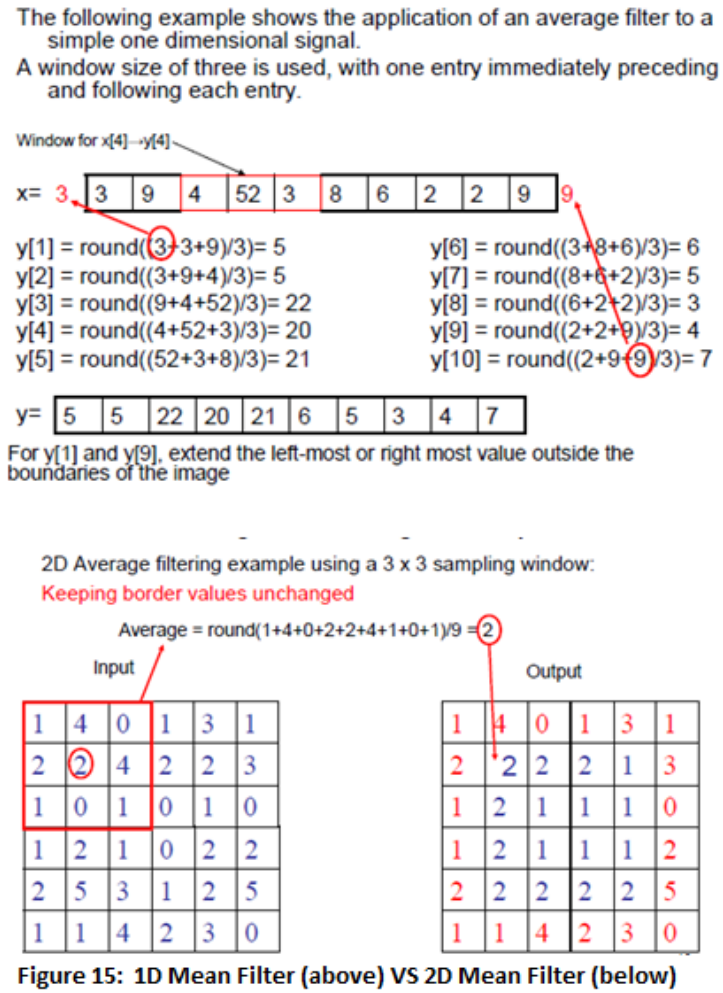


**Figure 15: 1D Mean Filter (above) VS 2D Mean Filter (below)**

## Contrast Adjustment

This is a basic but a quick method to alter the contrast of an image. This process is also known as linear mapping and it involves adjusting the image by applying a constant ***gain a*** **and** ***offset*****, or** ***bias b*** to pixel values of an *image g* to form the new image ***f: f(x; y) = ag(x; y) + b*** (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms)***.***

Contrast adjustment is a process 'stretching' pixel-wise grey values (intensities) to span a larger range of values. This process controls the amount of contrast applied to an image and is shown the effects of different values of ***'a' and 'b'***.

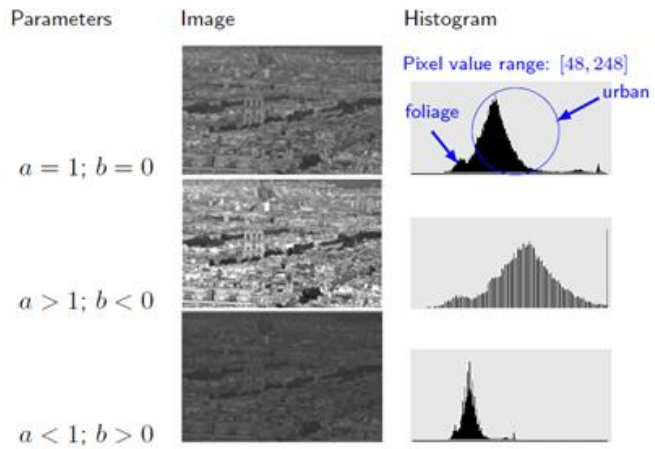




**Figure 16: Shows differences in contrast with respect to different values of 'a' and 'b'**

Again, the above example is shown for a grayscale image but in my project I am dealing with colour and grayscale images. To ensure that I am able to use this process on coloured images, I need to process each RGB colour channel separately with this formula for the algorithm to work and this can be seen in the programming section of this report.

## Gaussian Filter

Another filter I am going to implement in my application is called a Gaussian blur and it is also known as Gaussian smoothing (Fisher, Perkins, Ashley, & Wolfart, Gaussian Smoothing, 2003). As the name suggests, it blurs an image by a Gaussian function to reduce the image noise and reduce some detail. The effect of this filter will give an effect of viewing the image through a translucent screen or create a softly blurred version of the original image. This algorithm then prepares the images to be used by other fuzzy effects or can be used just to remove noise (Gimel'farb & Delmas, Part 3: Image Processing - 3.4. Moving Window Transform) (Efford, 2000).

Mathematically the Gaussian blur is the same as convolving the image with a Gaussian function. Also, applying a Gaussian blur has the effect of reducing the image's high-frequency component; hence a Gaussian blur is thus a low pass filter. This is similar to mean filter as it removes noise and blurs the image but uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. Then the Gaussian kernel is formed (Figure20) and the convolution is performed by convolving into the x direction first and then y direction after. There are many different kernels available for Gaussian blur but I decided to go with the one from compsci373 lecture notes (Gimel'farb & Delmas, Part 3: Image Processing - 3.4. Moving Window Transform) (Cheng, Huang, & Kumimoto, 2006).

Standard deviation of the Gaussian probability density function guides its behaviour:

- 68% of the x-values are the range $[\text{mean} - \sigma, \text{mean} + \sigma]$.
- 95% of the x-values are the range $[\text{mean} - 2\sigma, \text{mean} + 2\sigma]$.
- 99.7% of the x-values are the range $[\text{mean} - 3\sigma, \text{mean} + 3\sigma]$.

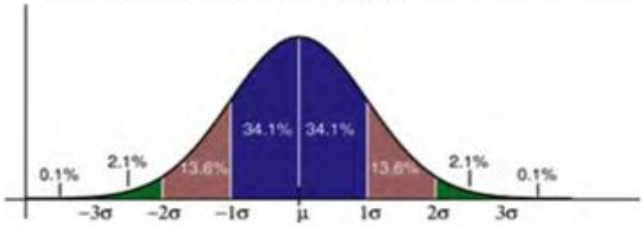
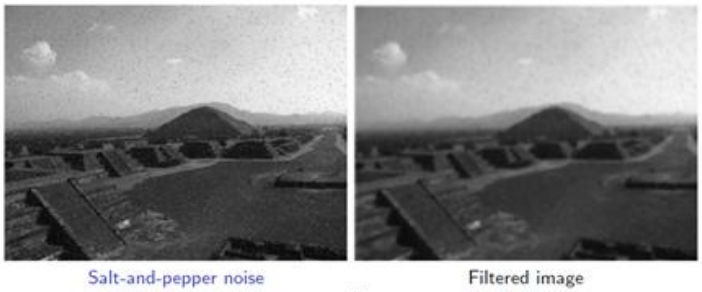



**Figure 19: Standard deviation of the Gaussian probability density function**

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

- An example: $5 \times 5$ window; $\sigma = 1$.
- The larger the value of $\sigma$, the wider the peak of the Gaussian and the larger the blurring.

Gaussian filter: blurred edges; residual noise.



Salt-and-pepper noise Filtered image

**Figure 20: Gaussian filter process and image differences**

## Edge detection

I am also planning to use “Edge detection” filter for my project because it is efficient for image smoothing to more accurate approximation of derivatives in edge detection. However, this filter has a few downsides such as:

- It is not known for removing salt-and-pepper noise
- It is robust when it comes to averaging outliers which leads to large deviations
- In that case, median is more robust when dealing with outliers

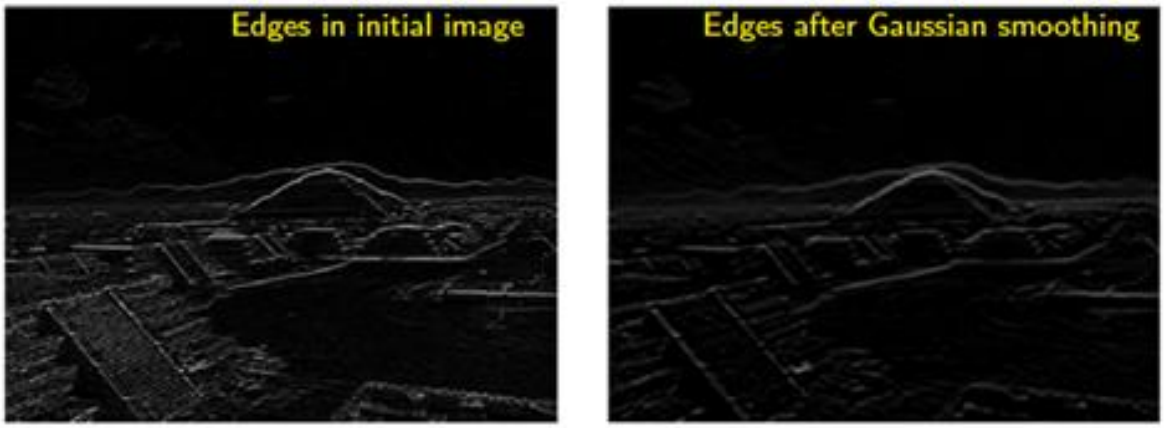



**Figure 21: Edge detection on an image and then Gaussian smoothing after**

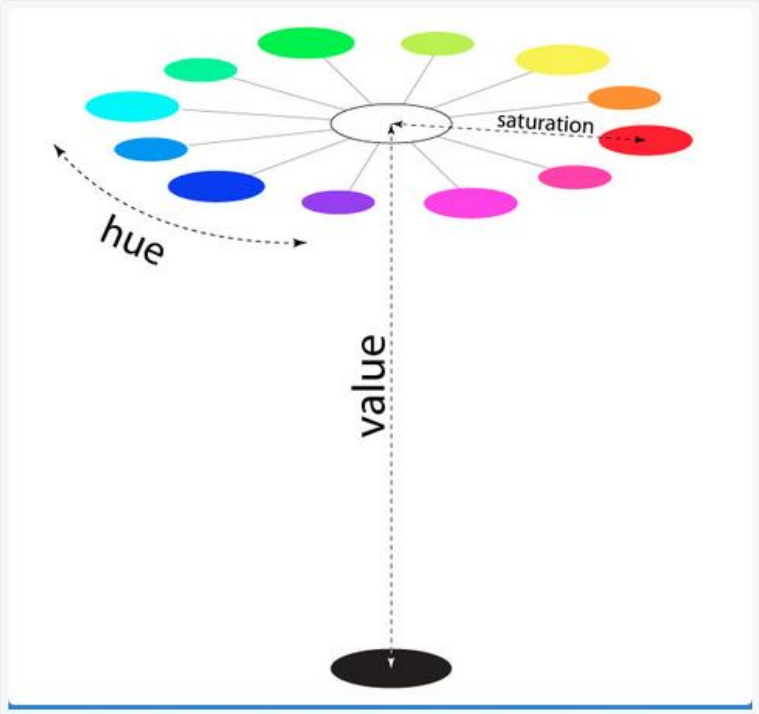
(Gimel'farb & Delmas, Part 3: Image Processing - 3.4. Moving Window Transform) (Jain, 2002).

## HSV Colour space filters – Hue, Saturation and Brightness

If I was to include some of the popular PC application filters such as Hue, Saturation and Brightness, I would need to understand the HSV colour space first.

To understand the HSV colour space I would need to understand what 'colour' means first. Colour is considered a visual by-product of the spectrum of light as it is either transmitted through transparent medium, or as it is absorbed and reflected off a surface. So, it is the light wavelengths that the eye receives and processes from reflected objects. The colour is made up of 3 main integral parts: Hue, Saturation (chroma) and Value (lightness or darkness) (Hue, Value, Saturation).

Hue is described as the dominant wavelength and is the first item we refer to when adding in three components of a colour. This is an essential choice for a filter because it is the dimension of colour we readily experience. In my project, the bitmaps are in ARGB configuration and a pure hue equivalent to full saturation is determined by ratio of the dominant wavelength to other wavelengths in colour. Saturation is also known as 'chroma' and it defines the intensity of a colour. When Hue pigment is toned, white and black/grey intensities are mixed with the colour to reduce the effects of saturation. Value refers to the lightness and the darkness of a colour. Dark values with black added are called "shades" of the given hue name and the light values with white pigment added are called "tints" of the hue name (Hue, Value, Saturation). The final project will contain these three filters because they are extremely popular in PC image processing applications.

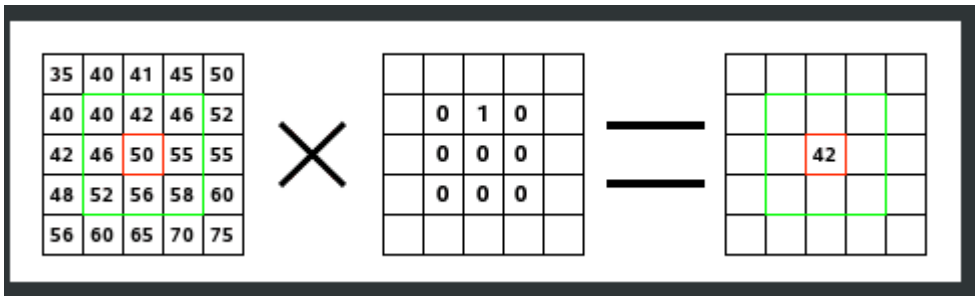


HSV Model with Hue, Saturation, and Value Explained

# Convolution Filter

I will need to reuse some of the code to design some of filters like sharpen, different types of blurs, mean and so on. To achieve this, I will use a Convolution matrix filter which is the treatment of a matrix by another one which is called “kernel”. The first matrix is the image pixels and the second filter is the kernel. The kernel determines the type of filter this will be, for example edge detection will have a different kernel to Gaussian Blur. This filter can handle different sizes of kernels as long as they are odd (3x3 etc) (8.2 Convolution Matrix) (Vandevenne, 2004).

The convolution filter multiplies the values of pixels by the kernel’s corresponding value. Then it adds the results, and the initial pixel is set to this final result value (8.2 Convolution Matrix).




The example above has the image matrix on the left, kernel matrix in the middle and the output matrix on the right. The image matrix has each pixel value marked and the initial pixel has a red border. The filter reads from left to right and top to bottom.

Here is what happened: the filter read successively, from left to right and from top to bottom, all the pixels of the kernel action area.

Process: (40*0)+(42*1)+(46*0) + (46*0)+(50*0)+(55*0) + (52*0)+(56*0)+(58*0) = 42. In the resulting image, the initial pixel moved a pixel downwards (8.2 Convolution Matrix).

## Engrave

To get the engraving effect on images, I am using the below kernel in conjunction with the Convolution Matrix. This gives an image an old style print look of a metal engraving. I decided to use it with conjunction with Edge detection algorithms to highlight the effect (Houston, 2011).

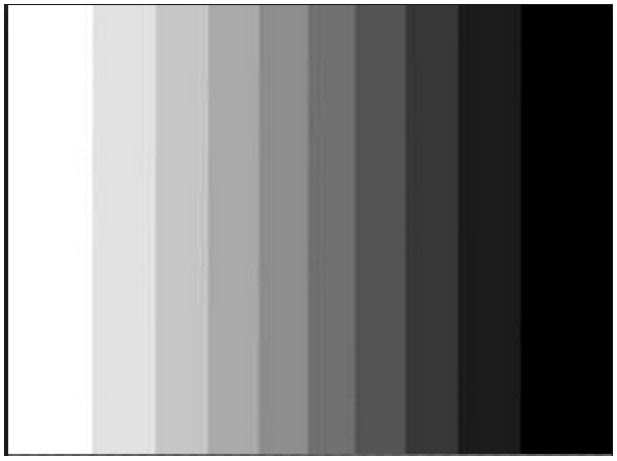
| | | |
|---|---|---|
| -2 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

## Flip Image

In the camera package, some of the devices will automatically flip the image through the camera but if the device is not compatible then there is a need to flip the image manually. This is quite a basic filter and shown how it is implemented in the programming part of the report.

## Grayscale

In image processing, this is a useful and common filter used in image processing applications. This is one of the filters which I would like to include in the application even though many applications already implement it. In digital images, these types of images known as black and white but are made up of shades of gray. Grayscale images are often formed as the result of measuring intensity of light at each pixel in a single band of electromagnetic spectrum and in these cases they are monochromatic (Chanda & Dutta, 2005).



## Horizontal and vertical edge detection

Imagine a case where the user wants to select just the horizontal or vertical edges. The problem with a fully fledged edge detection algorithm is that you cannot separate the vertical edges from horizontal edges. Therefore, implementing different types of edge detection filters will be the solution. There are many edge detection kernels but after trial and error I was able to find a couple that work efficiently which I discuss later in the programming part of the report (Rhody).

Figure 2. Image produced by the horizontal gradient calculation.

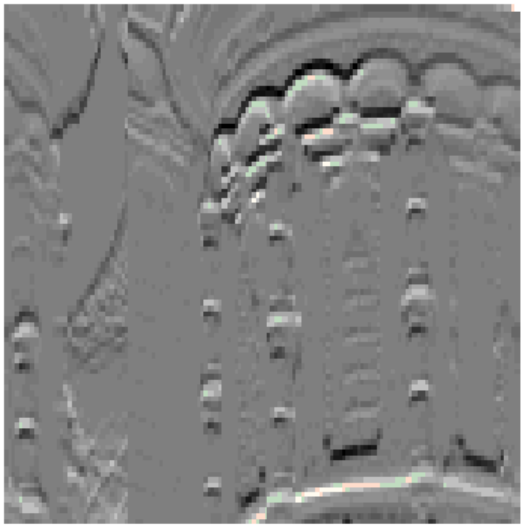


Figure 4 Edge produced by vertical gradient calculation.

## Message on Image

This is a basic filter that should only be applied at the end of image processing steps. This filter is used to place a string at position (X, Y) on the image. Currently I have designed it so that it places the text in the center of the image. The intent was to implement a layered filter similar to Photoshop but that might be too complex for an Android device application.

## ModColour RGB

This filter is used to change selective RGB colour channel. This is done by multiplying values to R/G/B value of each pixel. The input is received from the user and then multiplied with the colour channel values. So for example if the user wishes to increase Blue in an image, the input values should be R = 1, G= 1 and B>1. The objective of this filter to reduce or increase specific colour channels to give a monochrome effect.

## RGB Histograms feature

After some discussion with my supervisor, we came to a conclusion that my project should include a view that will display RGB histograms separately. This is a useful feature for someone who knows how to read and use the histograms to finalize their filter choices.

The colours (or pixel values) you see in an image are derived from varying combinations of red, green and blue. The colour of each pixel in an RGB digital image is determined by the value (0-255) assigned to each colour channel RGB for each pixel. In other words, each pixel contains values for RGB and we need to separate the values to map them on the histogram. The following provides a way to separate ARGB values from a pixel (Hoffmann, 2006):

**A = (pixels [index] >> 24) & 0xFF;**

**R = (pixels [index] >> 16) & 0xFF;**

**G = (pixels [index] >> 8) & 0xFF;**

**B = pixels [index] & 0xFF;**

The RGB colours are expressed as a numbers between 0 and 255 where the 0 represents pure black and 255 represents pure white. Basically, the image histograms will be presented as bar chart with the horizontal axis being the range of values (0-255) and the vertical axis will represent the frequency of those the range of values (Hoffmann, 2006).

Imagine a scenario where the user captures an Image and clicks the RGB Histogram button in the application and sees that there are peaks in the graphs at particular tonal range. This means there is a high frequency of R/G/B values in that range. Depending on the distribution of the frequency, the user can now choose to use a filter like Image Equalization etc to reduce the effects of the peaks and this decision is made by using the histograms shown. To assist the user even more, I include mean and standard deviation of each RGB colour value separately just like the histograms. In Photoshop, the user can click on an area of the histogram graph and read the intensity value from the range, for example: At R = 200, the intensity is 5. This can be an optional addition to Picca if there is any time left at the end (Hoffmann, 2006).

Histograms can vary depending on the RGB content of the image. A histogram of a high key image with a majority of the content being very bright will produce a histogram that has most of the histogram graph located from the center to the right of center. A low key image with

lots of dark and shadow areas will produce a histogram graph that is mostly center and left of center. Filters like increase in brightness can shift the histogram values to the right etc. To take advantage of the RGB histogram feature, the user must apply the filter and then go back and check the change to the histogram and then choose the next filter and so on.

Once the user has a basic understanding of at an image's histogram and figure out which parts of the graph correspond to the different tonal ranges and components in that image.

## Other Filters

In the “App Development” section of this report, I discuss about filters that I added close to the submission date. They are explained in depth and it is easier for the reader to understand because I have provided the source code and explained each part.

# Project Overview

My project goals are to develop an application for mobile devices that provide users with some of the common features of desktop image processing applications and also to provide users with image enhancement and restoration support, provided that the image is in the condition to be restored or enhanced. Secondary goal of this application is to give users an understanding of the RGB histograms display to check how filter algorithms affect the distribution and how it can be used to make decisions on filter selections.

Currently, there are many Android applications that include basic image editing features such as crop, rotate etc but not all of them are useful when the image is degraded in quality or needs to be enhanced to reveal more detail. The user can always use a desktop application to process images; however applications such as Photoshop are expensive to buy and need a PC. So that being said, how can I incorporate some of the major problems of image processing on Android devices and solve it using an Android application?

The application will need to focus on multiple problems such as digital noise, bad exposure, blurry images, and distracting elements in our digital photos etc. The most common issue is digital noise and this is the “grain” that we sometimes notice in film photography. There are many types of digital noise so implementing just one filter for digital noise would not be reliable or usable because the image quality changes because no two images are the same (unless of course we copy) so there needs to be a few options that user can apply to get the best possible result. Considering the scenario of different types of digital noise to use it for other image quality issues, the types of filters needs to be researched and how they can be used together to process the image.

The major reason for motivation was that image processing applications on Android sometimes have limited functionalities such as just smooth filter. I want to incorporate some of PC application features into my application. Also, I am completely new to this area of programming and this will be my first Android application and this project will give me the skills and knowledge to develop a complete application because I want to be a developer of applications for mobile devices in the future.

## Project Requirements

- Picca must be an image-processing application developed for the Android devices with screen size of 5” and over due to the layout of the application.
- Must include at least three different screens. Homescreen, Filter screen and another. The design must be aesthetic and follow usability guidelines.
- The application must provide instructions to the user on how to use it.
- Picca must solve problems such as: digital noise, bad exposure, blurry images, and configure colour and contrast by implementing image enhancement and restoration filters
- Image enhancement filters must improve the quality of an image captured by the camera or stored in the media folder by manipulating the image.
- Image restoration filters must restore corrupted/noisy image and developing a new image from the original. Corruption may come in many forms such as motion blurs noise etc.
- Provide common features such as Hue, Saturation and Brightness etc found in popular desktop applications.
- The application must provide the user with variety of filters who meet the goal of image enhancement and restoration.
- The types of filters must be approved by the supervisor.
- The images can also be loaded from a file location instead of using the camera to capture them.
- The application must be able to restore the image back to original if the user decides not to save the image after applying the filter.
- The application must allow filters to be applied in conjunction with each other.
- The application must provide image-selection functionality directly from the homescreen using buttons.
- When trying to select images from a file location, they must be displayed using GridLayout with thumbnails and sorted according to the date.
- The application must start and close without any errors. Providing an Exit button will help close the application safely.
- The images can be saved in a JPEG due to compression benefits and quality.
- The filtered images must be saved back in the media folder without reduction in quality (kept at 100%).
- The application must provide a display of RGB Histograms of the selected image so that the users can make decisions based on the distribution.

- The application must have reusability of code and must be as responsive (dependent on the filter complexity) as possible.

# Android App Development

Android is an open source OS designed for mobile devices and has the largest share of mobile device market. The reason of its success is because there are a lot of applications that are available for the Android users. The applications are developed using the Standard Development Kit (SDK) and it is a free development kit. The new version of Eclipse Android SDK is offered as a complete download on the Eclipse website as compared to before where it was available to download as a plug-in to Eclipse. The applications can be shared through Google Play (also known as Android Market) and can be used by anyone. Next, I will provide some insight on the Android Architecture (Android).

## Android Architecture

The Android Architecture has made up of different layers, where each layer is a group of program components (see image below). It includes operating system, middleware and important applications. Each layer in the architecture provides different services to the layer above it (Android Architecture – The Key Concepts of Android OS, 2012) (Market).

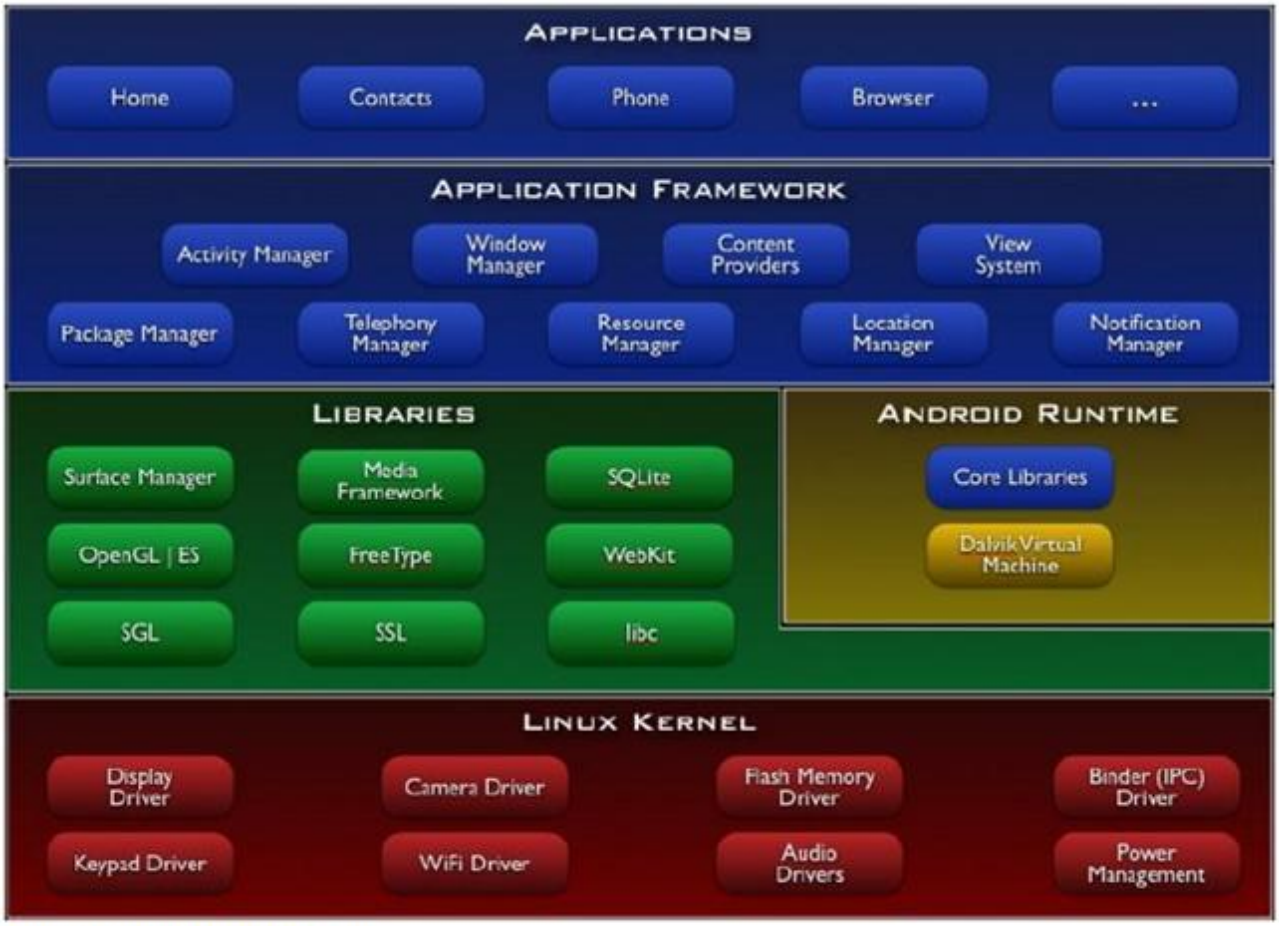



**Figure 22: Android Architecture**

## Linux Kernel

The most basic layer is the Linux kernel and the Android Operating system is built on top of the Linux Kernel. Linux interacts with the hardware and contains all the essential hardware drivers. The Linux kernel also acts as an abstraction layer between the hardware and other software layers. The memory management, process management, networking and security settings are used by Linux as its core functionality (Android Architecture – The Key Concepts of Android OS, 2012).

## Libraries

As seen from the image, the next layer consists of Android libraries. This layer allows the devices to handle different types of data and they are developed in C or C++ language and specific for a particular hardware. **Some of the components are as follows** (Market)**:**

**Surface Manager:** This component is used for combining window manager with off-screen buffering and that means that you cannot directly draw into the screen, instead the drawings go to the off-screen buffer (Android Architecture – The Key Concepts of Android OS, 2012).

**Media framework:** Media framework provides different media codec that allows playback and recording of different media types.

**SQLite:** This component is the database engine used in android for data storage purposes

**WebKit:** This is the browser engine used to display HTML content.

**OpenGL:** Used to render 2D or 3D graphics content to the screen

## Application Framework

The application framework manages the basic functions of phone like voice call management; resource management etc and these are tools with which we can build our applications (Android Architecture – The Key Concepts of Android OS, 2012).

**Some important blocks of Application framework are:**

**Activity Manager:** Manages the activity life cycle of applications

**Content Providers:** Manage the data sharing between applications

**Telephony Manager:** Manages all voice calls. We use telephony manager if we want to access voice calls in our application.

**Location Manager:** Location management, using GPS or cell tower

**Resource Manager:** Manage the various types of resources we use in our Application

## Applications

The top layer in Android architecture is called ‘Applications’ and this where developer applications are going to settle. Several applications are installed from factory such as SMS client app, Dialer, Web browser, Contact manager etc. This layer allows us to write an application which can also replace any existing system application (Android Architecture – The Key Concepts of Android OS, 2012).

# Eclipse Android SDK

## Setting up SDK

To develop Android applications, I need to use the **Android Developer Tools (ADT)** plug-in for Eclipse and it provides an environment for building Android apps. It contains the full java IDE that would allow me to build, test and debug my Android application. This software is free, open-source and runs on the major OS platforms (Get the Android SDK).

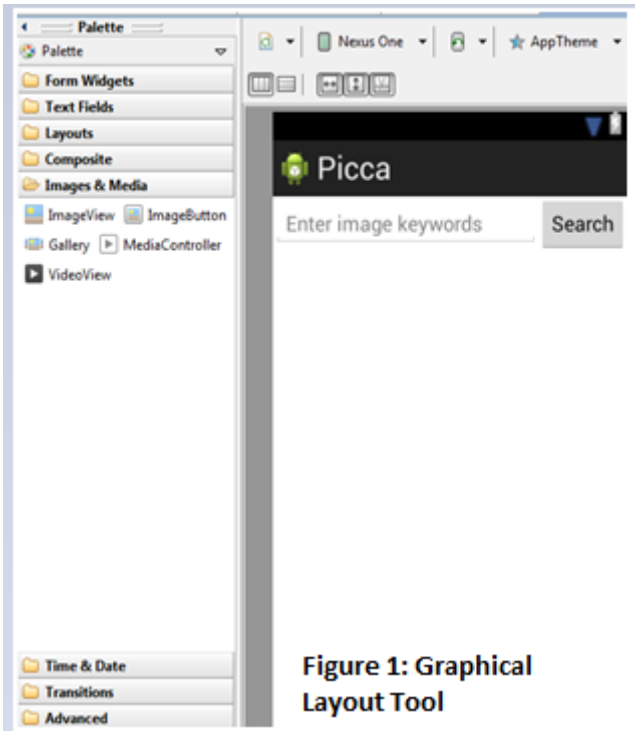



Figure 1: Graphical Layout Tool

## Graphical UI Builders

Android SDK contains a Graphical Layout Tool (Figure 1.) that allows drag and drop of Android UI components. It also allows you to visualize the UI on Android devices and switch themes, even platforms versions without building the code.

## Develop on Hardware Devices

I can run my application on any commercial Android hardware device or multiple devices by deploying my app to connected devices from the IDE. It also allows me to live debug on-device, test and profile my application.

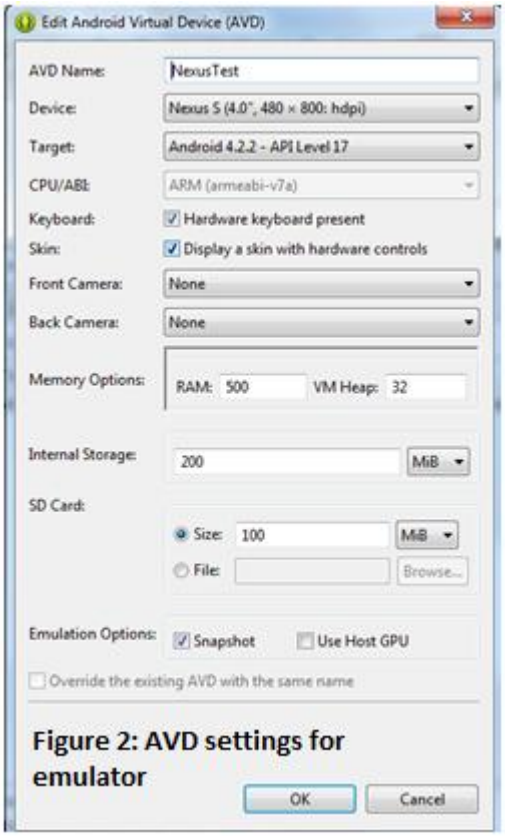



Figure 2: AVD settings for emulator

## Develop on Virtual Devices

I am using the emulator to run my application for the starting stages such as tutorials and buttons. The Android Virtual Device lets me define the parameters such as the device, memory, SDK version and then save the options to use later without redoing the process again. It includes advanced hardware emulation and that means it includes the camera, sensors and multi-touch functionalities. Once the complexity of the project grew, I bought myself a couple of Android Tablets so that it would allow me to debug application quick and easy (Android Emulator).

# AVD and the Emulator

An interesting feature in the Android SDK is that I can configure an Android Virtual Device (**Figure 8**) to create an Emulator (**figure 9**) to parts of my application and in this section I will provide an in-depth explanation of how it works. The emulator feature would basically let me prototype, develop and test my application without using a physical device.

The Android emulator (**Figure 9**) mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls. It still provides variety of navigation and control keys which I am able to click (or use the keyboard for input) for interaction with my application (Android Emulator).

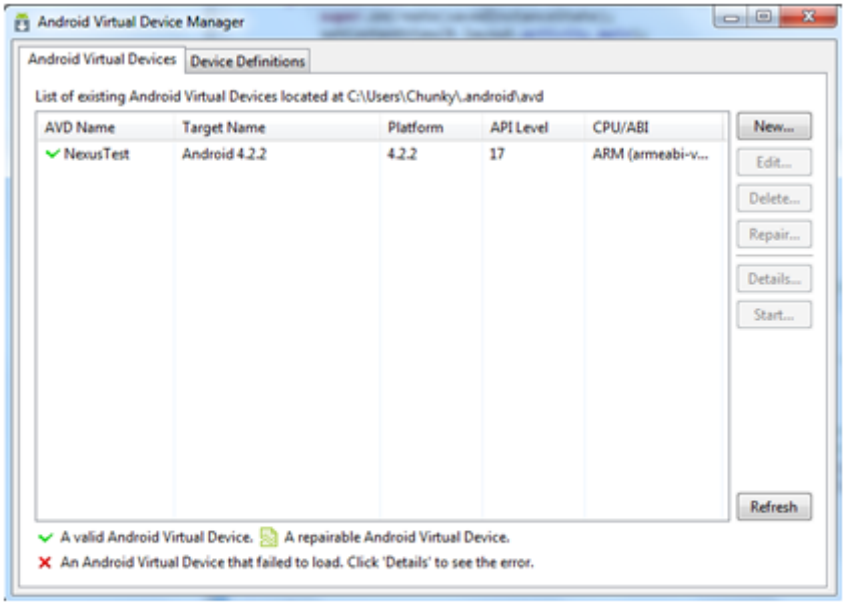



**Figure 8: AVD manager which allows to create/modify the properties of the virtual device**

**Figure 9: A picture of the emulator that I am using to run my application**

It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application. It also provides a screen in which your application is displayed, together with any other active Android applications.

I have to use Android Virtual Device configurations to create a new device to specifications I want such as hardware aspects such as front/back camera etc. The AVD can be used to test applications on many different device configurations. To debug my application, I can use the console from which I log the kernel output, simulate application interrupts and simulate latency effects.

## A test user input program using Emulator

Once I had setup the AVD properties (Note: this sample program is in portrait mode but Picca has landscape orientation), I decided to run a simple user input tutorial just to check if the emulator was able to execute without any errors. At first, the emulator takes around 3 full minutes to load and then I later learnt that it can be run from 'snapshot' which significantly cuts down the time. In this demo, I had added the 'TextField and a 'Search' button as you can see on the left. However, the time to load is way too long and therefore from now on I will be using the Debug mode on my Android tablet to execute the application (Android Emulator).

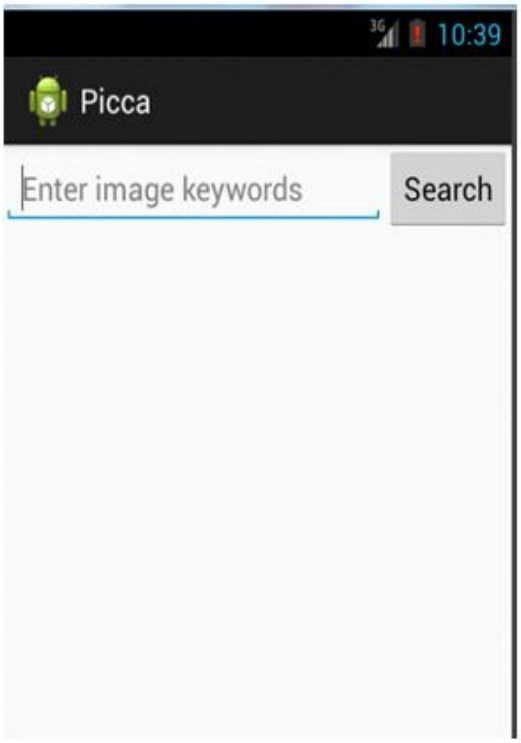



Figure 10: Homescreen tested on emulator

# Balsamiq Mock-up Tool

After some research on mock-up tools, I have able to come across Balsamiq mock-up tool that I can use to develop my prototype for Picca. The reason for using this software is because it feels like the mock-up is hand-drawn, however it is a digital mock-up which allows me to tweak and rearrange easily whenever I want (Balsamiq Mock up Tool).

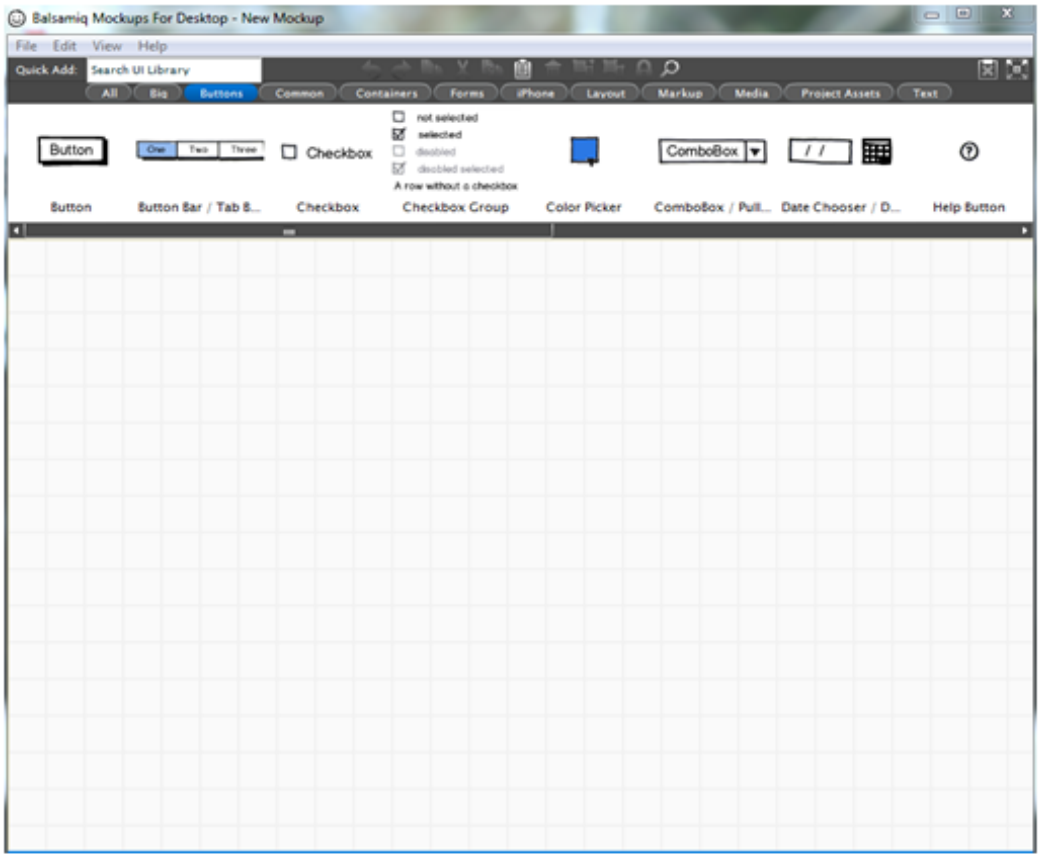



**Figure 3: Balsamiq Mockups Tool displaying a blank canvas with Functionality panel**

## Features

**Low-Fi Sketch Wireframes**

The mock diagrams look as if they have been sketched and appear low-fidelity wireframes and this allows the viewer to focus on the functionality of the components.

**UI Components & Icons**

The version I am using has over 40 built-in UI components and over 100 icons that I can drag and drop on the canvas.

**Export to PNG or PDF**

I can also share or present mock-ups with embedded links using PDF export, or use a 3rd party tool to export to code (Balsamiq Mock up Tool).

## Mock-ups using Balsamiq for Picca:

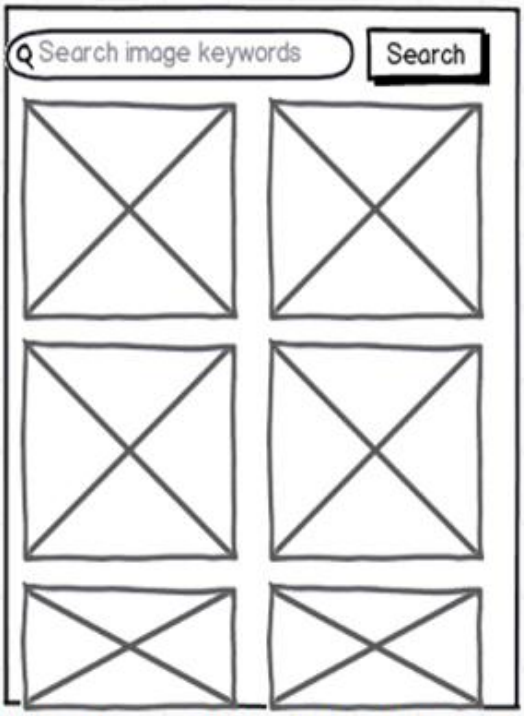



Figure 4: Homescreen mockup created using the Balsamiq Mockup Tool

**Figure 4** displays the homescreen of Picca and this gives the user a basic idea of the layout and function of the application. The boxes with 'crosses' represent images that will be directly loaded from the user's media album which allows selection without going to the media folder every time he/she wants to edit them. Also I decided to include a search bar at the top of the screen so the user can retrieve an image via the name it is saved under. Once the user clicks on an image he/she wants to enhance or restore, they will be directed to the screen below.

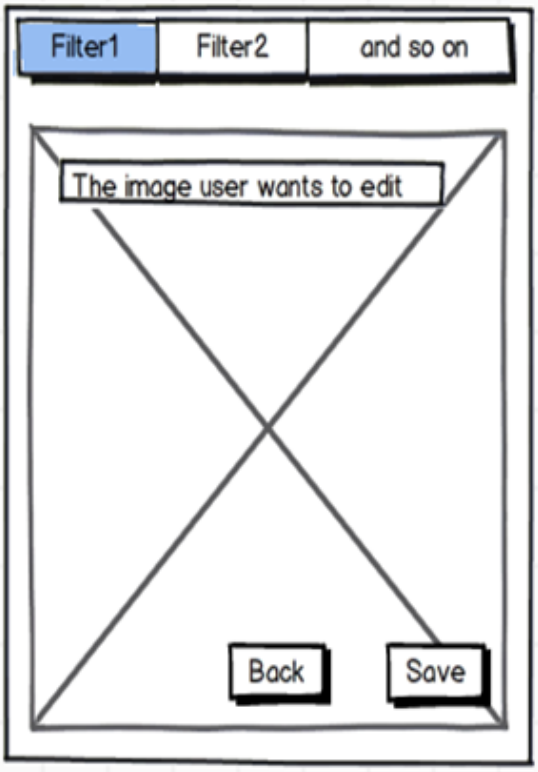



Figure 5: Filter screen mockup created using the Balsamiq Mockup Tool

As seen by **Figure 5**, this is the main screen which allows the users to apply filters on the image of their choice after they select it from the screen above. On the top panel are filters that I will implement and once selected it will calculate and display the new image. The user can then save the image if they wish or they can click 'Back' to apply more filters.

## Revised Picca Mock-ups

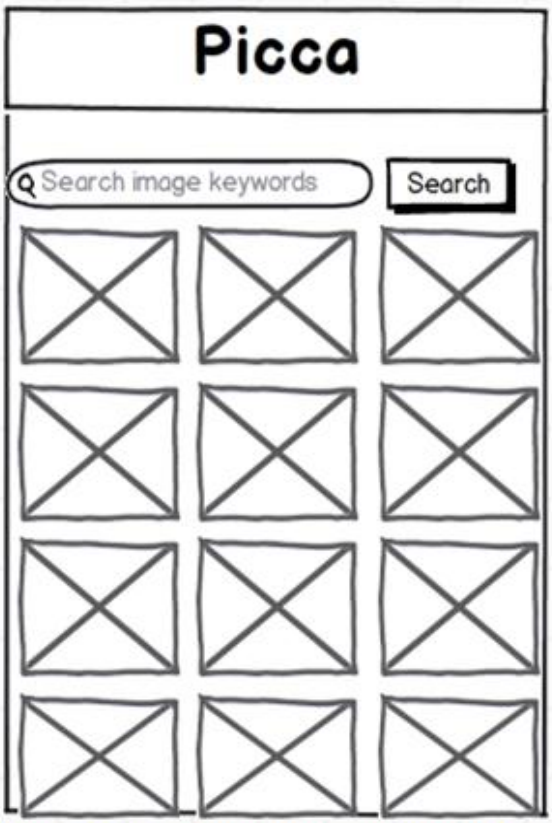



Figure 6: Revised homescreen mockup using the Balsamiq Mockup Tool

As seen by Figure 6, I decided to revise the mock-ups and add the name of the application at the top of the screen. I also ran the application and found that 2 images in a row appear way too big and I decide to add another column of images so that it would look proportional to the screen-size.

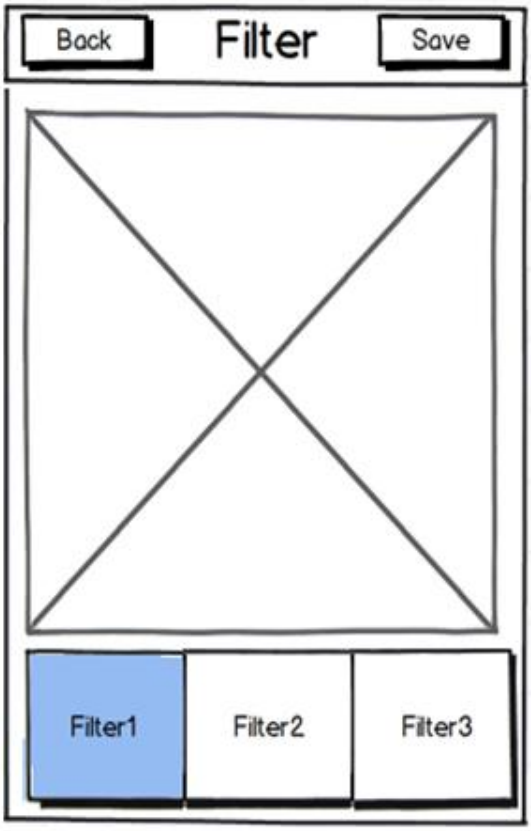
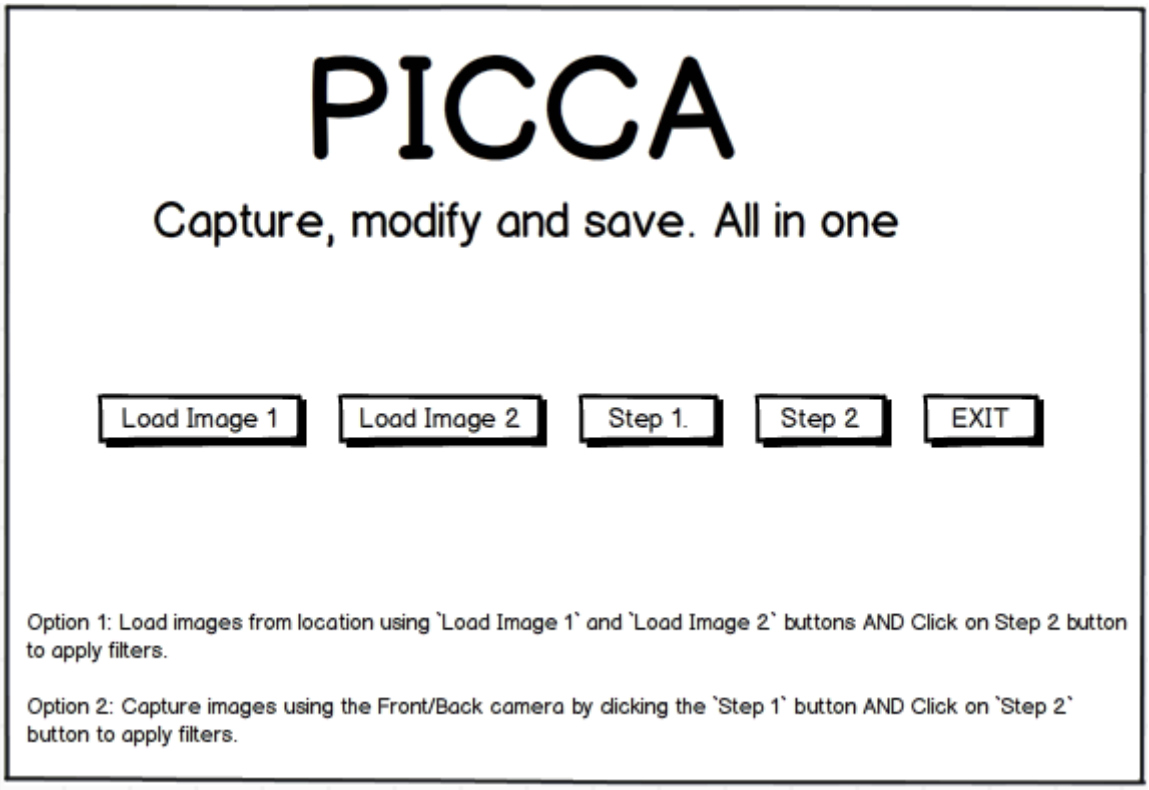


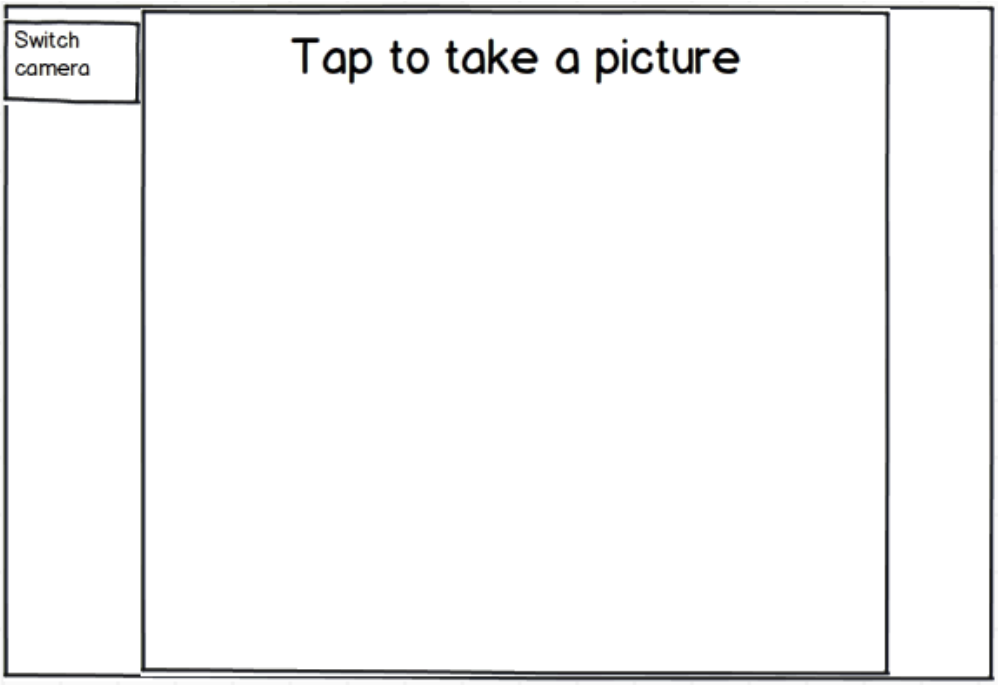

Figure 7: Revised filter screen mockup using the Balsamiq Mockup Tool

Main changes were done to this screen (shown by **Figure 7**) to increase the ease of use and overall layout of the application. The filters will now be placed at the bottom of the screen where the user can easily access it without reaching all the way to the top to change filters. The placement at the bottom also allows a “preview” of the filter on the image and will give more of a ‘thumbnail’ effect.

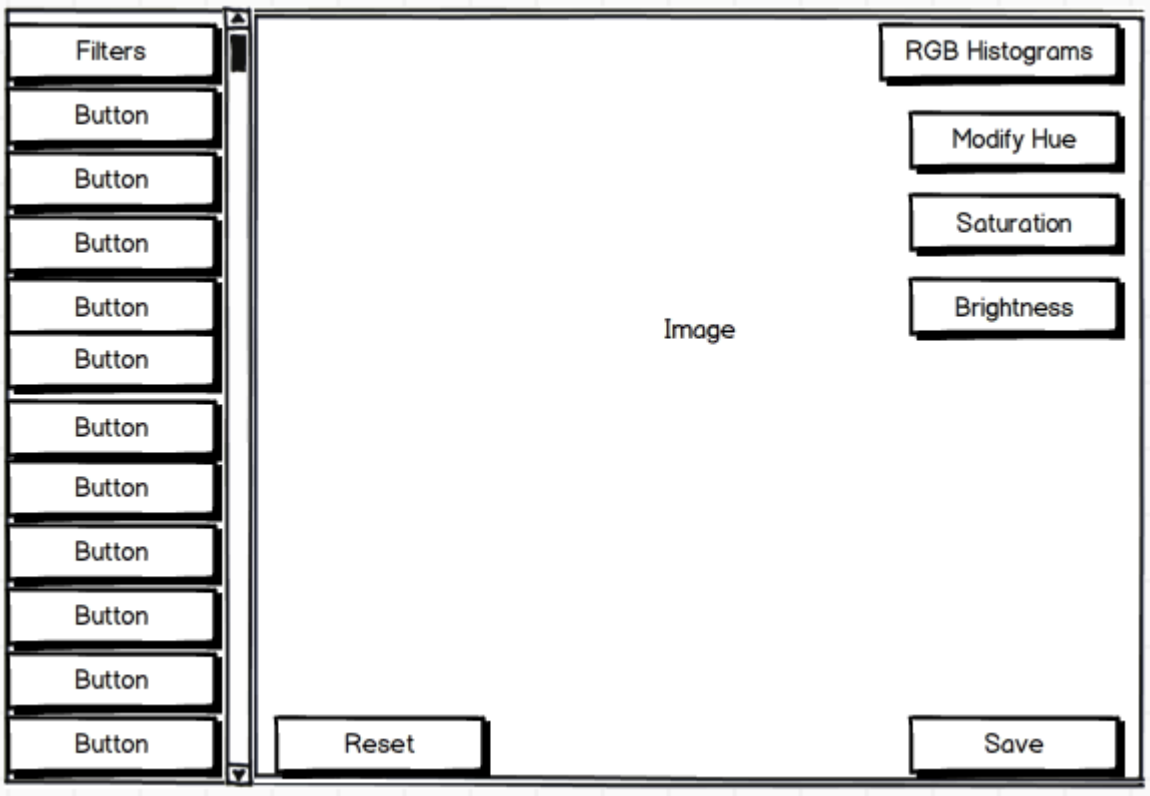
## Final Mock-ups for Picca




The amount of information I wanted to display for this application means that I need a larger screen size and this is why I bought 2 android tablets of different sizes, 5.4” screen and 9.7” screen. I decided to use the Landscape mode for my application and the homescreen in the revised mock-ups included a Grid View image display but to implement it in Android would use up a lot of resources if the user wanted to capture images via the camera. So then it would be efficient to include buttons that would load the images from a file location and then jump straight to Step 2 to apply filters.

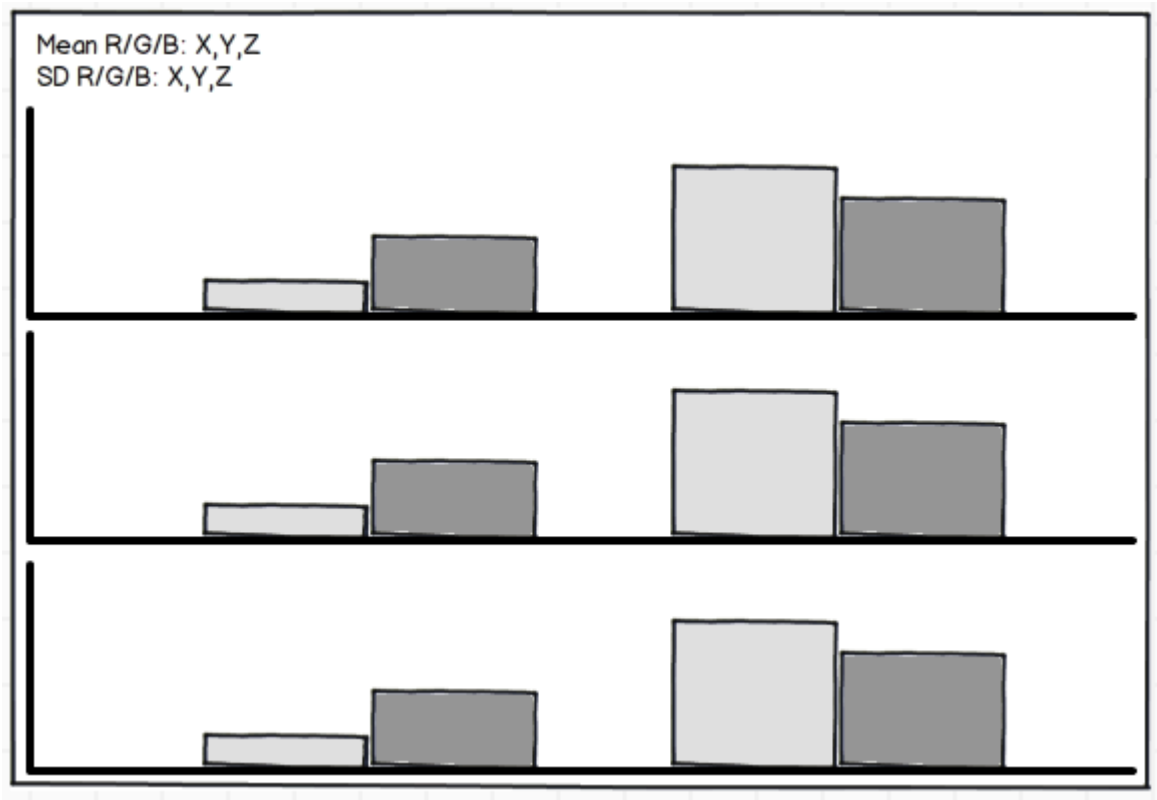


The above screen shows the camera interface and it is a basic design as the user should not be distracted with other insignificant features. If the device has a front camera, the user is able to switch by the clicking the icon on the top left of the screen.




The mock-up above is for the screen where filters will be applied. On the left hand side, the buttons represents different types of filters and the most common operations will be on the ImageView such as Save, Reset, RGB Histogram etc. There has to be a vertical scroll option for the filter buttons because I will be implementing quite a few. The above mock-ups consist of simple design and it is kept consistent all throughout the application.

After consulting Patrice, he wanted me to have a RGB histogram feature that would display the RGB intensities separately for the user to see. So I decided to create an initial mock-up and can be seen below. Just providing the RGB histograms may not be enough as the user would still want more information regarding the image and which is why I decided to include the mean and standard deviation of RGB colour channels separately.

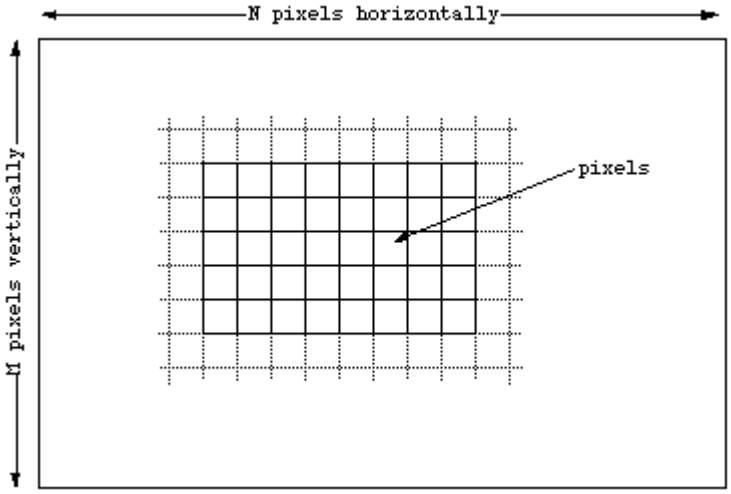



Next, I need another screen to display the RGB histograms for the image selected in the ImageView. The graphs will span the whole screen because the range is quite wide 0-255.

# Introduction to Bitmaps

## What is a Bitmap?

A Bitmap is an array, or a matrix of square pixels (picture elements) arranged in columns and rows with each pixel containing a colour value (Bourke, 1993) (Gonzalez & Woods, 2007).

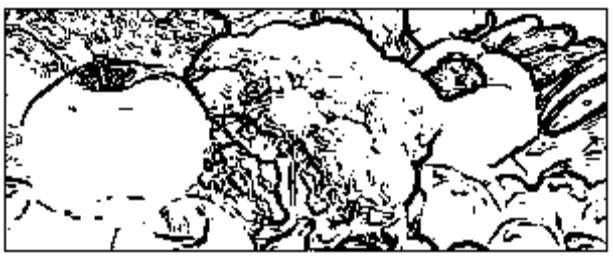



The number of pixels you need to get a realistic looking image depends on the way the image will be used. Most common uses for bitmaps are that they are used to represent images (Introduction to Image Processing).

## Some of the common types of bitmaps:

### 1 bit (black and white)

This type of bitmap contains the smallest possible information per pixel and this type of bitmap is also known as monochrome or black and white. Since it only contains 1 bit, the pixel value of 0 represents pure black and pixel value of 1 represents pure white. Below is an example image of 1 bit bitmap (Bourke, 1993).

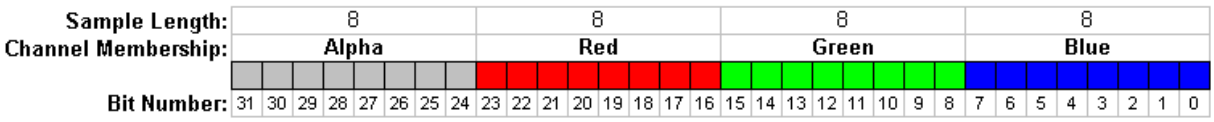


## Grayscale – 8 bit

In a grayscale image each pixel has an intensity that ranges from 0 to 255. By convention 0 is black and 255 white. The gray levels are the numbers in between, for example, pixel value of 127 would be a 50% grey level. A normal greyscale image has 8 bit colour depth = 256 possible grayscale values (Bourke, 1993) (Castleman, 1995).


## 32 bit ARGB

In this type of bitmap, the intensity of each ARGB channel is defined by 8 bit and is arranged in the following manner.




The Alpha colour depth is found in the highest 8 bits, followed by the Red, Green and Blue. This implementation is used in my project to modifying each colour component separately. Similar to this, a bitmap which contains 24 bits is known as RGB bitmap and doesn't contain the alpha value. I can create a grayscale image using this type of bitmap by setting the RGB intensity the same per pixel (Chanda & Dutta, 2005).

## 16 bit RGB

This type of bitmap involves the RGB components to use 5 bits each and 1 bit for alpha (Bourke, 1993).

## Bitmap Resolution

Pixels have no explicit dimensions and therefore resolution is an attribute which is used when viewing or printing bitmaps. Resolution can be specified in a few ways with pixels per inch being the most common but could also be represented in any other unit of measure.

The quality of the bitmap when displayed or printed depends on the resolution. Since the resolution determines the size of a pixel it can also be used to modify the size of the overall image. Whenever a bitmap is displayed, the device screen resolution also needs to be considered (Bourke, 1993).

# Picca programming work

## 1. AndroidManifest.xml

### Description

Basically the manifest file describes the fundamental characteristics of the app and defines each of its components. Firstly, I included the **<uses-sdk>** element and this it to declares Picca's compatibility with other Android versions using the **android:minSdkVersion** and **android:targetSdkVersion** attributes. For my application, I decided to set the **android:targetSdkVersion** high as possible and test your app on the corresponding platform version.

**The full AndroidManifest.xml file is shown below:**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="nz.ac.auckland.Picca.app"
    android:versionCode="12"
    android:versionName="6.2" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.CAMERA" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-feature android:name="android.hardware.camera.any" />
    <uses-feature android:name="android.hardware.screen.landscape" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

```
        <activity
            android:name="picca.main.packages.MainActivity"
            android:configChanges="orientation"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="camera.packages.CameraActivity"
            android:configChanges="orientation"
            android:screenOrientation="landscape" />
        <activity
            android:name="edit.pictures.packages.EditPicturesActivity"
            android:configChanges="orientation"
            android:screenOrientation="landscape" />
    </application>

</manifest>
```

I have different intent-filters in the manifest and an intent filter basically lets the system know which service request an activity, services or broadcast can handle.

For example, if I was coding for an image viewer, then I would add an intent filter to the manifest describing the images I would be handling.

# Introduction to Linear Layout, TextFields and Buttons

## LinearLayout

For my application, I began with creating a LinearLayout and this is a view group (a subclass of ViewGroup) and which allows the setting of orientation.

```
android:orientation="horizontal" >
```

The other two attributes that are needed to create the **LinearLayout** are **android:layout_width** and **android:layout_height**, because they are required for all views in order to specify their size.

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

The reason why the above attributes are set to **match_parent** is because the **LinearLayout** should fill the entire screen area that's available to my application.

## TextField

Next I decided to add a **Text Field** object by doing the following:

```
<EditText android:id="@+id/edit_message"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

The weight value is a number that specifies the amount of remaining space each view should consume, relative to the amount consumed by sibling views. The default weight for all views is 0, so if you specify any weight value greater than 0 to only one view, then that view fills whatever space remains after all views are given the space they require. So, I filled the remaining space in my layout with the EditText element by giving it a weight of 1 and leaving the button with no weight.

**android:id:** provides a unique identifier for the view, which I can use for reference when I want to read and manipulate the object.

The ‘at’ sign (@) is required when you need to refer to any resource object from XML. It is followed by the resource type (id in this case), a slash and then the resource name (**edit_message**).

**android:layout_width** and **android:layout_height:** The **wrap_content** value specifies that the view should be only as big as needed to fit the contents of the view.

### Button

To add a simple button, I use the below code just to see how it works. However, in the main project, I will be implementing custom buttons using the Drawables options to define custom buttons.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_search"
    android:onClick="sendMessage" />
```

During the mid-year break I was able to discuss more about my project with Patrice and we believed that GridLayout should be implemented after all the other aspects of the application are finished.

I am provided by a basic application template by Patrice and then I begin developing it to the specifications of my application.

# Custom buttons for my application

## Background_menu.xml

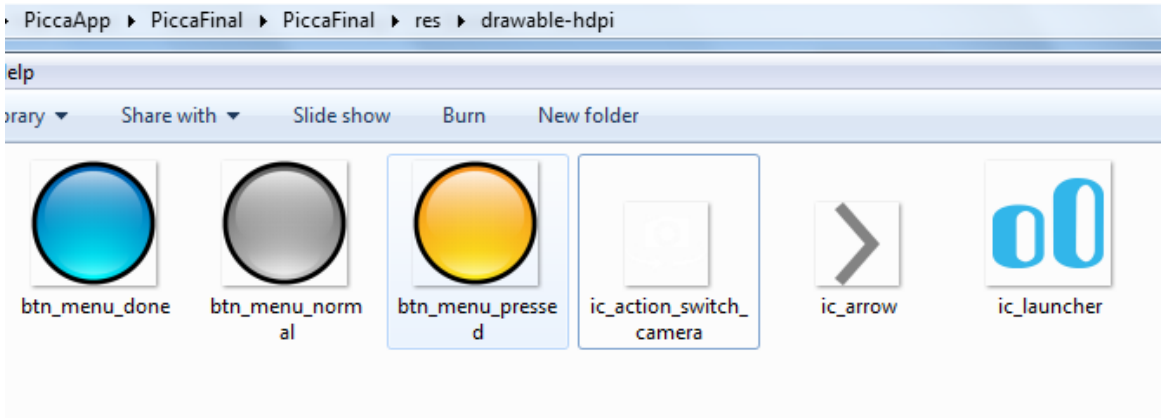
First I design a gradient background for the Homescreen of my application and this is achieved by doing the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >

    <gradient
        android:endColor="#cfe1edFF"
        android:gradientRadius="326"
        android:startColor="#7faacaFF"
        android:type="radial" />

</shape>
```

Basically it has a start colour and an end colour and the gradient is used as the change between the two. To use this effect I need to use the **android:background** attribute and point it to the location of the file and this is shown below.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background_menu"
    tools:context=".MainActivity" >
```

## Items in my Drawables-hdpi/mdpi/xhdpi folders




The above images will be used for the buttons during their different states of use. The names are self explanatory as to where they are going to be used.

## button_blue_red_transluscent.xml

Next, if I wanted to place any buttons on top of the ImageView then I would need to ensure that they are translucent. When the button is pressed it is assigned Red but when the button

ContrastStretch.java | PiccaFinal Manifest | activity_main.xml | button_blue_red_transluscent.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:state_pressed="true"><shape>
            <solid android:color="@color/red_translucent" />

            <stroke android:width="1dp" android:color="#9E9E9E" />

            <padding android:bottom="15dp" android:left="20dp" android:right="20dp" android:top="15dp" />

            <corners android:radius="5dp" />
        </shape></item>
    <item><shape>
            <solid android:color="@color/blue_translucent" />

            <stroke android:width="1dp" android:color="#9E9E9E" />

            <padding android:bottom="15dp" android:left="20dp" android:right="20dp" android:top="15dp" />

            <corners android:radius="5dp" />
        </shape></item>

</selector>
```

is normal it is assigned Blue. Note: for both states, the colours are kept translucent.

## button_white_red_transluscent.xml

Similarly to above, I have designed another .xml file that when pressed the colour is Red and in the normal state it is kept White. Again, both the states use translucent colours.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:state_pressed="true"><shape>
            <solid android:color="@color/red_translucent" />

            <stroke android:width="1dp" android:color="#9E9E9E" />

            <padding android:bottom="15dp" android:left="20dp" android:right="20dp" android:top="15dp" />

            <corners android:radius="5dp" />
        </shape></item>
    <item><shape>
            <solid android:color="@color/white_translucent" />

            <stroke android:width="1dp" android:color="#9E9E9E" />

            <padding android:bottom="15dp" android:left="20dp" android:right="20dp" android:top="15dp" />

            <corners android:radius="5dp" />
        </shape></item>

</selector>
```

### button_menu.xml

To select the images and assign them to the states of the buttons, I use a separate .xml file. First to assign the state where the button is pressed, I assign the corresponding item from the Drawables folder and the set the **state_pressed** as **true** so that it is recognised in the above **.xml** files.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:drawable="@drawable/btn_menu_pressed" android:state_pressed="true"/>
    <item android:drawable="@drawable/btn_menu_normal"/>


</selector>
```

### button_menu_done.xml

After the pressed state, I apply the done button which is blue. It is similar to the above .xml.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:drawable="@drawable/btn_menu_pressed" android:state_pressed="true"/>
    <item android:drawable="@drawable/btn_menu_done"/>

</selector>
```

## Homescreen Layout

### activity_main.xml:

This xml file is used to produce the homescreen of the application and all the fields are encapsulated by a RelativeLayout and this is a view group that displays child views in relative positions of each view's position can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left of center) (Tamada, 2011).

RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID). In the template, we have used this feature for the different buttons which would allow alignment of two elements by right border, or make one below another,

centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout (Layout).

Some of the many layout properties available to views in a RelativeLayout include:

**android:layout_alignParentTop**

If "true", makes the top edge of this view match the top edge of the parent.

**android:layout_centerVertical**

If "true", centers this child vertically within its parent.

**android:layout_below**

Positions the top edge of this view below the view specified with a resource ID.

**android:layout_toRightOf**

Positions the left edge of this view to the right of the view specified with a resource ID.

The homescreen layout is developed by the **activity_main.xml** file shown below:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background_menu"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="5dp"
        android:drawablePadding="10dp"
        android:gravity="bottom|center_horizontal"
        android:text="@string/app_name"
        android:textSize="100sp"
        android:textStyle="bold" />


    <TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/title"
android:layout_centerHorizontal="true"
android:layout_marginTop="5dp"
android:gravity="top|center_horizontal"
android:text="@string/app_subtitle"
android:textSize="25sp"
android:textStyle="bold" />
```

So I begin by adding a few **TextViews** and this displays text to the user and optionally allows them to edit it, however I have disabled editing for this application. For example, one of the **TextView** contains the text from **Strings.xml** where the field is called **app_name** and in this case the value for that is **Picca** and we can refer to **String.xml** for the rest of text allocations.

```
<Button
    android:id="@+id/step1_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:layout_toLeftOf="@+id/arrow1"
    android:background="@android:colour/transparent"
    android:drawableBottom="@drawable/button_menu"
    android:text="@string/step1"
    android:textColour="@android:colour/secondary_text_light" />
```

Now I add a Button which represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action. The button includes an attribute called **android:id="@+id/step1_button"** and this allows me to refer to it while I want to do something when the button is clicked etc. The background of the button is set at **Transparent** and the image of the button is loaded from **Drawables** folder and the images are supplied by Patrice's template. The rest of the attributes are self-explanatory.

```
<ImageView
    android:id="@+id/arrow1"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBottom="@id/step1_button"
android:layout_centerVertical="true"
android:layout_toLeftOf="@+id/step2_button"
android:paddingBottom="5dp"
android:src="@drawable/ic_arrow" />
```

Since I want to have a “process-flow” appearance of the buttons, I decide to add an arrow that points from **Step 1 button to Step 2 buttons** and this is done by using **ImageView. ImageView** allows us to display an arbitrary image, such as the **ic_arrow** icon. The ImageView class can load images from various sources as well and I used this for the camera view as well. It can also be used in any layout manager and provides various display options such as scaling and tinting.

```
<Button
  android:id="@id/step2_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerVertical="true"
  android:layout_toLeftOf="@+id/arrow2"
  android:background="@android:colour/transparent"
  android:drawableBottom="@drawable/button_menu"
  android:text="@string/step2"
  android:textColour="@android:colour/secondary_text_light" />
```

Similar to **Step1** button, I create a **Step2** button which has the same properties as **Step1** except the alignment is to the left of **arrow2.** Also for the rest of the fields, I have basically applied the same principle to create “duplicates” with different placements and different Ids. Also notice the android:drawableBottom="@drawable/button_menu" as this ensures the colouring of the button.

**...Similarly, the rest of the xml file is implemented. See the project folder for the full class.**

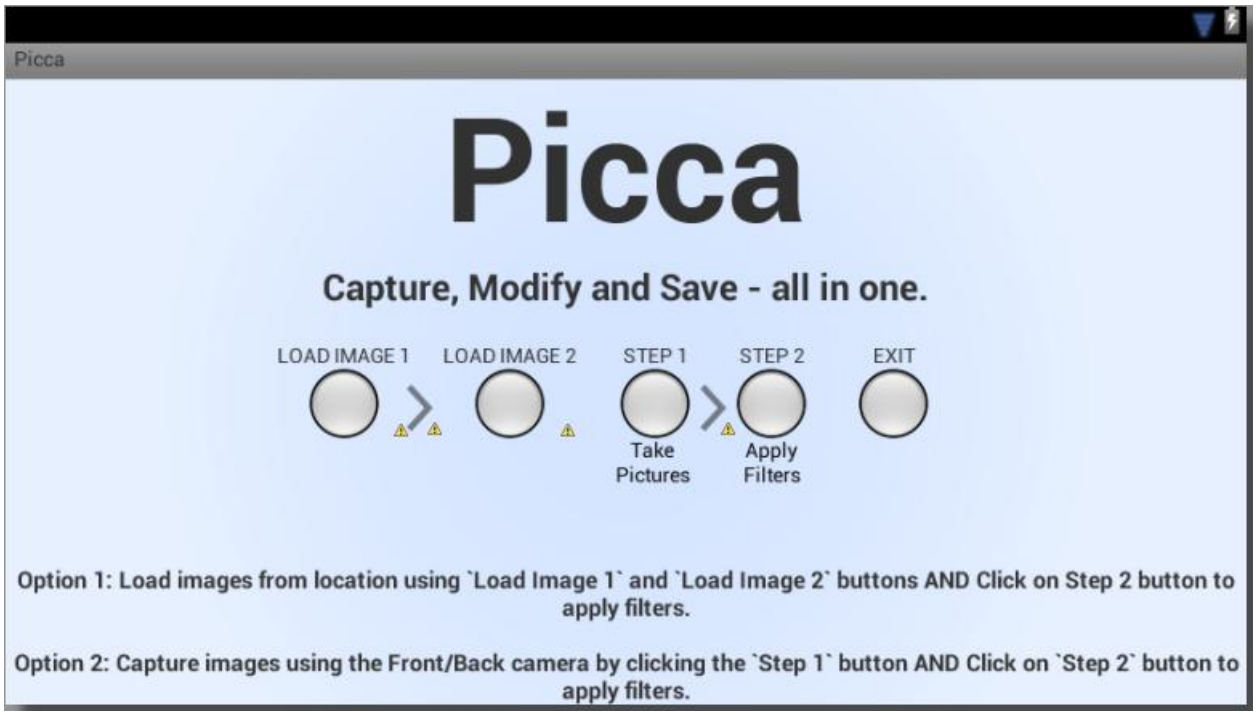
```
</RelativeLayout>
```

In the above xml file, the images of the button is loaded from the Drawables folder which includes all the images the template uses and are easily assigned to fields by using the following:

**android:src="@drawable/ic_arrow"**

The **ic_arrow** is the name of the image from the **Drawables** folder and we are able to assign any images to a field as long as it is present in the **Drawables** folder. The **android:text** attribute refers to a part of the **String.xml** file that consists of all strings in the project.

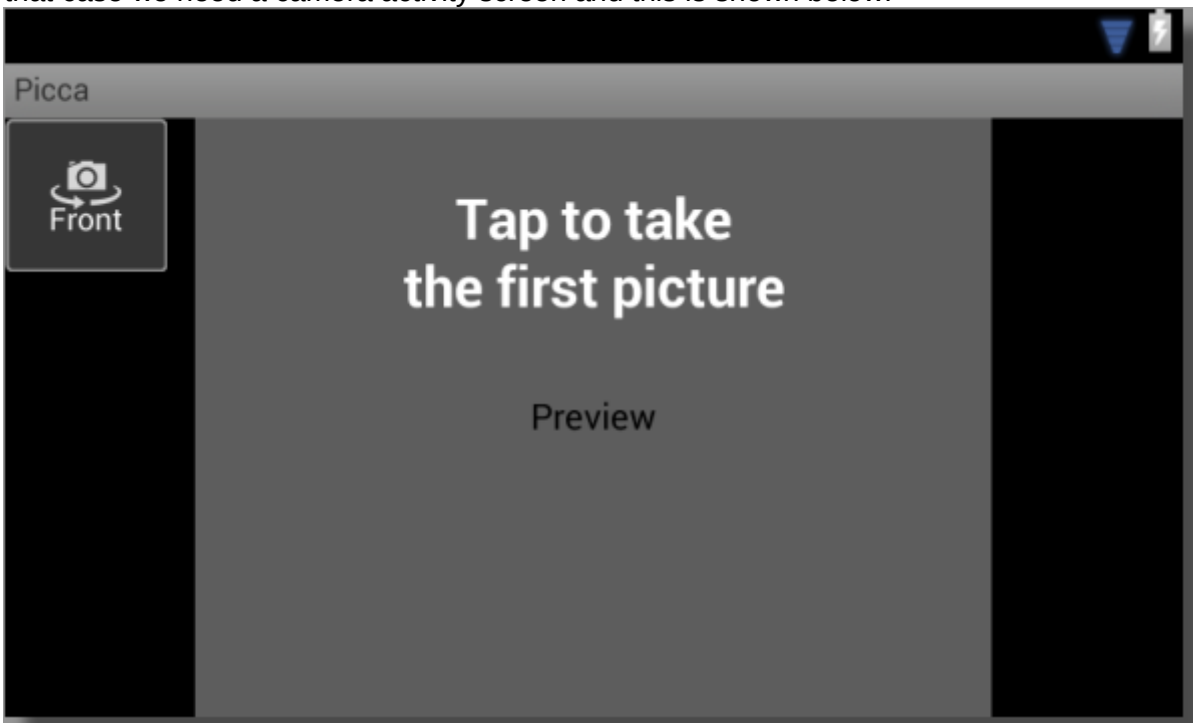
The graphical output of the **activity_main.xml** in a 5.4" device is as follows:



# Camera Screen

## Activity_camera.xml

After the main screen, we need to have a camera interface where the user is able to use the camera (Front/Back) and also see the ImageView once the user clicks the **Step1** button. In that case we need a camera activity screen and this is shown below:




The **Preview** represents the image that will be seen by the camera and the user is able to capture is either using the Camera Button on the Android device or tap the screen (the programming part is covered later). The above screen is developed using the **activity_camera.xml**:

Every field in this screen is encapsulated by a **FrameLayout** and this layout is designed to block out an area on the screen to display a single item such as the black areas around the ImageView shown in the image above. Similarly to activity_main.xml, this screen consists of ImageView, TextViews and Buttons and the images are loaded from **Drawables** folder.

# Apply Filter Screen

## Activity_edit_pictures.xml

After capturing the pictures, the user is then directed to the screen that processes the images and this is the major focus of the application.

This xml file is used to create multiple layouts encapsulated with many components. In this layout, I used a **LinearLayout** which arranges its children in a single column or a single row. The direction of the row can be set by calling **setOrientation()** and in my project, I have used **vertical**. We can also specify gravity, which specifies the alignment of all the child elements by calling **setGravity()** or specify that specific children grow to fill up any remaining space in the layout by setting the weight member of **LinearLayout.LayoutParams**. The default orientation is **horizontal**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:colour/black"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/filter_linear_layout"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <ScrollView
            android:layout_width="wrap_content"
            android:layout_height="match_parent" >
```

After implementing a few buttons, the screen space had run out I had to use a scroll feature by implementing **ScrollView**. It is a layout container for a view hierarchy that allows scrolling by the user, allowing it display more information than screen space. A **ScrollView** is a **FrameLayout**, meaning that I can only place one child in it. Therefore, to display more than

one **Button**, I implemented another **LinearLayout** inside the **ScrollView** that presenting a vertical array of top-level items such as **Buttons** that the user can scroll through.

I had to make sure that I did not use **ScrollView** with a **ListView** because **ListView** takes care of its own vertical scrolling. Most importantly, doing this defeats all of the important optimizations in **ListView** for dealing with large lists, since it effectively forces the **ListView** to display its entire list of items to fill up the infinite container supplied by **ScrollView** (Views).

There are other options for scrolling, however I used **ScrollView** because it only supports **vertical** scrolling and that's what I needed for the buttons.

```
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <Button
                android:id="@+id/filter1_button"
                android:layout_width="fill_parent"
                android:layout_height="0dip"
                android:layout_weight="1"
                android:background="@drawable/button_white_red_transluscent"
                android:text="@string/filter1_text"
                android:textColour="@android:colour/white" />
```

**...and the remaining buttons added similarly. See project files for more details**

```
        </LinearLayout>
    </ScrollView>
</LinearLayout>


<RelativeLayout
```

```
android:id="@+id/rel_layout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_toRightOf="@id/filter_linear_layout" >

<ImageView
   android:id="@+id/filter_image_view"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:layout_toRightOf="@id/filter_linear_layout" />
```

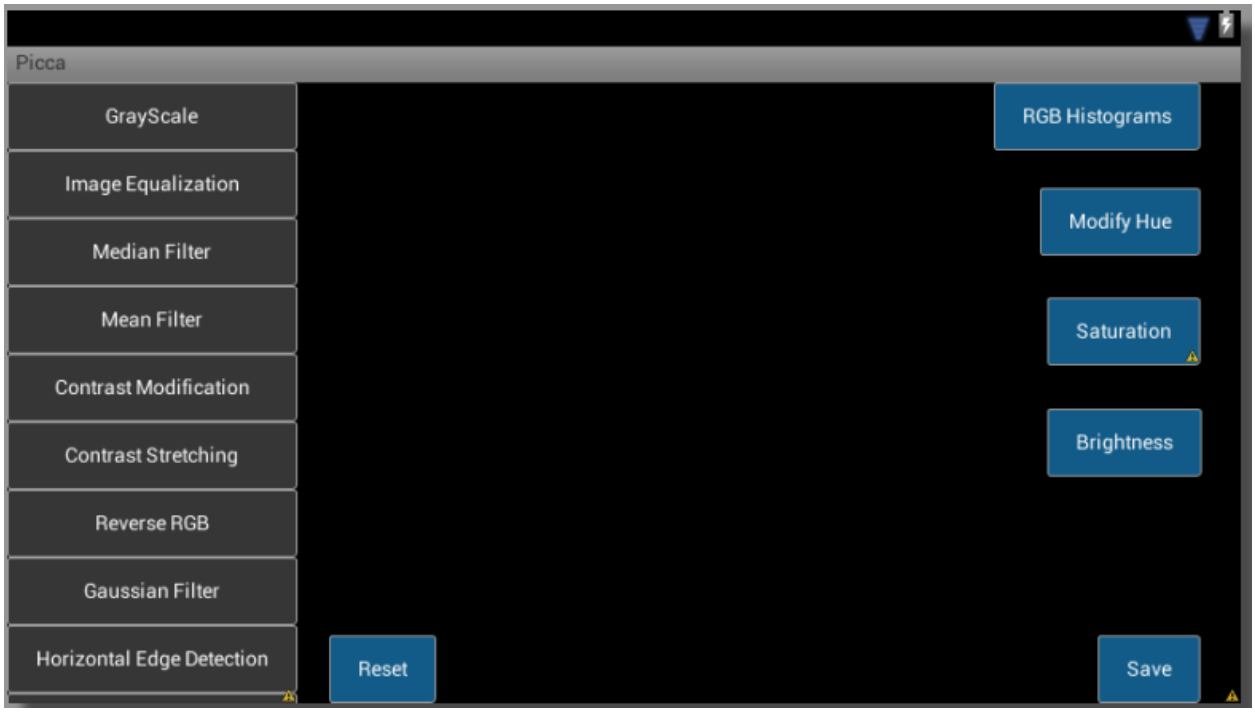
The **ImageView** is where the image to be filtered is shown on and I wanted to show the histograms of RGB values on the **ImageView** by click of a button and therefore I added a button that would display RGB, save the filtered image and reset back to the original image.

```
<Button
   android:id="@+id/reset_button"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentBottom="true"
   android:layout_alignParentLeft="true"
   android:layout_marginLeft="22dp"
   android:background="@drawable/button_blue_red_transluscent"
   android:text="@string/Reset_text"
   android:textColour="@android:colour/white" />
```

**...Similarly, the rest of the class is implemented. See the project folder for more information.**

```
   </RelativeLayout>
</RelativeLayout>
```

The graphical layout for the **activity_edit_pictures.xml** on a 5.4” screen looks as follows:




## Strings.xml

This class holds all the strings I have used in the project. To keep it organized, I have ensured that it is divided into four parts: Main, Camera and Edit Pictures and Edit Pictures ImageView. It just helps me keep track of strings and it is easy to go to back to when I want to edit or add more strings for that section.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- Main -->
    <string name="app_name">Picca</string>
    <string name="app_subtitle">Capture, Modify and Save - all in one.</string>
    <string name="action_settings">Settings</string>
    <string name="app_description">Option 1: Load images from location using `Load Image 1` and `Load Image 2` buttons AND Click on Step 2 button to apply filters.

Option 2: Capture images using the Front/Back camera by clicking the `Step 1` button AND Click on `Step 2` button to apply filters.</string>
    <string name="step1"><b>STEP 1</b></string>
    <string name="step1_text">Take\nPictures\n</string>
```

```
    <string name="step2"><b>STEP 2</b></string>
    <string name="step2_text">Apply Filters\n</string>
    <string name="step3"><b>EXIT</b></string>
    <string name="step3_text">Finish\n</string>
    <string name="step4">LOAD\nIMAGE 1</string>
    <string name="step5">LOAD\nIMAGE 2</string>

    <!-- Camera -->
    <string name="back">Back</string>
    <string name="front">Front</string>
    <string name="tap_to_take_the_left_picture">\nTap to take\nthe first picture</string>
    <string name="tap_to_take_the_right_picture">\nTap to take\nthe second picture</string>
    <string name="info_text_right_picture">\nAlign to take\nthe second picture</string>
    <string name="picture_without_camera">A picture cannot be captured without a
camera</string>
    <string name="overlay_image_description"></string>
    <string name="no_camera_access">Can\'t access the camera.</string>
    <string name="reticle">Reticle</string>
    <string name="are_you_satisfied_pictures_text">Are you satisfied with these
pictures?</string>
    <string name="continue_text"><b>Yes</b>\nContinue</string>
    <string name="take_new_pictures_text"><b>No</b>\nTake new pictures</string>

    <!-- Edit Pictures -->
    <string name="grayscaleText">GrayScale</string>
    <string name="imageEqualizationText">Image Equalization</string>
    <string name="medianText">Median Filter</string>
    <string name="meanText">Mean Filter</string>
    <string name="contrastText">Contrast Modification</string>
    <string name="brightnessText">Brightness</string>
    <string name="reverseRGBText">Reverse RGB</string>
    <string name="gaussianText">Gaussian Filter</string>
    <string name="hEdgeDetectText">Horizontal Edge Detection</string>
    <string name="engraveText">Engrave</string>
    <string name="vEdgeDetectText">Vertical Edge Detection</string>
    <string name="smoothText">Smoothing</string>
    <string name="edgeDetectText">Edge Detection</string>
```

```
    <string name="blackText">BLACK bitmap</string>
    <string name="colorchannelsText">Modify Colour Channels</string>
    <string name="colorIncreaseText">Increase Colour</string>
    <string name="flipImageText">Flip Image</string>
    <string name="hueText">Modify Hue</string>
    <string name="messageOnImageText">Text on Image</string>
    <string name="rotateText">Rotate 90 degrees</string>
    <string name="roundRectBorderText">Round Rectangle Border</string>
    <string name="saturationText">Saturation</string>
    <string name="shadingImageText">Shading Image</string>
    <string name="sharpenText">Sharpen</string>
    <string name="tintingImageText">Tint Image</string>
    <string name="contrastStretchText">Contrast Stretching</string>

    <!-- Edit Pictures ImageView -->
    <string name="Save_text">Save</string>
    <string name="display_RGB">RGB Histograms</string>
    <string name="Reset_text">Reset</string>
    <string name="seek_bar_text">SeekBar</string>

</resources>
```

## EditPicturesActivity.java

As you can see from the previous pages that the **activity_edit_pictures.xml** file allowed me to add views, buttons and text fields, however those items do not have any functionalities.

```
public class EditPicturesActivity extends Activity {
    /************************
     * PROPERTIES
     ************************/
    /* Tag for Log */
    public static final String TAG = "EditPicturesActivity";
    /* Images Path */
    private String[] _imagesPath;
    /* Original Images */
    protected static Bitmap[] _originalBitmaps = new Bitmap[2];
    /* Final Filtered Images */
    protected static Bitmap[] _filteredBitmaps = new Bitmap[2];

    /* Image View */
    private ImageView _filterImageView;
    private String leftImagePath;
    private String rightImagePath;
```

First I begin with a **TAG** which shows up in **LogCat**, which basically helps when debugging the application. Next you can see **_imagespath**, which is used to decode the bitmaps and load it in the **_filterImageView**. In Android, to pass strings or string arrays from one activity to another (in our case from **MainActivity** to this) we need to use **putExtras** which I will talk about in the **OnCreate()**. Also I have declared 2 arrays of bitmaps; **original** and **filtered**. The original bitmaps represents the un-modified image decoded at the start and the filtered bitmaps represent the images modified by the filters available in the application. The size of the Bitmap array is 2 because the camera is used to capture two images and is kept consistent all throughout the classes. After some discussion with Patrice, we believed that we should let the camera capture two images just in case the user does not click it properly the first time and it only takes a second extra to capture the second image.

Next, we need to declare the types of filters to be used for my application. This is a simple task and can be seen below. I have 20+ filters implemented and a filter called **_blackFilter** is to used to test the **RGB histogram** feature. Once applied, the image goes pure black (RGB = 0) and when histograms are displayed all values are at 0. Each filter implemented in the project is explained in detail later in the report.

```
/* Filters */

Filter _GrayScaleFilter = new GrayScaleFilter();
Filter _EqualizationFilter = new HistogramEqualizationFilter();
Filter _MedianFilter = new MedianFilter();
Filter _MeanFilter = new MeanFilter();
Filter _ContrastAdjust = new ContrastAdjust();
Filter _BrightnessIncrease = new Brightness();
Filter _ReverseRGB = new ReverseRGB();
Filter _GaussianBlur = new GaussianBlur();
Filter _HorizontalEdgeDetection = new HorizontalEdgeDetection();
Filter _Engrave = new Engrave();
Filter _VerticalEdgeDetection = new VerticalEdgeDetection();
Filter _Smooth = new Smooth();
Filter _EdgeDetect = new EdgeDetect();
Filter _blackFilter = new BlackBitmap();
Filter _colorChannels = new ColorChannels();
Filter _flipImage = new FlipImage();
Filter _hue = new Hue();
Filter _messageOnImage = new MessageOnImage();
Filter _rotate = new Rotate();
Filter _saturation = new Saturation();
Filter _sharpen = new Sharpen();
Filter _tintingImage = new TintingImage();
//Filter _contrastStretch = new ContrastStretch();
```

Once the declarations are over, the **OnCreate()** needs to be overridden in which I need to initialize the activity. Most importantly, here I will call **setContentView(int)** with a layout resource defining my UI for this class, and using **findViewById(int)** to retrieve the widgets in that UI. The first few lines sort out the window and layout for this class and the intent is used to receive the **_imagesPath** array from the **MainActivity** using **getStringArrayExtra().**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getWindow().setFormat(PixelFormat.TRANSLUCENT);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.activity_edit_pictures);

    Intent intent = getIntent();
    _imagesPath = intent.getStringArrayExtra(MainActivity.IMAGES_PATH);
    leftImagePath = intent.getExtras().getString("leftPath");
    rightImagePath = intent.getExtras().getString("rightPath");
```

Once we get the **_imagesPath** from MainActivity we are ready to decode the bitmap which is located at the **_imagesPath**. I added an If condition to check whether the path is null and if that is the case the application will call the **finish()** (Note: below screenshot is not the updated version). If the **_imagesPath** is not null then we can decode the bitmaps located at the path and load them into the **_originalBitmaps**. The **decodeFile** method in my project uses a file path to locate a bitmap. If the specified file name is null, or cannot be decoded

```
if (_imagesPath == null) {
    // to use with manual locations for images
    _originalBitmaps[MainActivity.LEFT_IMAGE] = BitmapFactory
            .decodeFile(leftImagePath);
    _originalBitmaps[MainActivity.RIGHT_IMAGE] = BitmapFactory
            .decodeFile(rightImagePath);

} else {

    for (int i = MainActivity.LEFT_IMAGE; i <= MainActivity.RIGHT_IMAGE; i++) {
        _originalBitmaps[i] = BitmapFactory.decodeFile(_imagesPath[i]);
    }
}

_filterImageView = (ImageView) findViewById(R.id.filter_image_view);
_filterImageView
        .setImageBitmap(_originalBitmaps[MainActivity.LEFT_IMAGE]);

for (int i = MainActivity.LEFT_IMAGE; i <= MainActivity.RIGHT_IMAGE; i++) {
    // copying the original image in _filteredBitmaps so that a filter
    // can
    // be applied over and over again
    _filteredBitmaps[i] = _originalBitmaps[i].copy(
            _originalBitmaps[i].getConfig(), true);
}
```

into a bitmap, the function returns null. The reason why I am using this method to load my bitmaps is because it loads a '**mutable**' bitmap which basically means I can modify the contents of the bitmap.

Once the bitmap is decoded, we need to add it to the **_filterImageView. MainActivity.LEFT_IMAGE** is a type integer and consists of value 1 and **MainActivity.RIGHT_IMAGE** consists of integer value 2. The loop below, copies the original bitmaps into the bitmaps that are going to be modified using filters and this **copy()** returns a mutable bitmap.

After the image is added to the Image View, we can finally start adding functionalities to the buttons. Majority of the implementations of the buttons are similar because all the modification of the bitmap is done by extending the **Filter** class (discussed later in the report).

First implementation is the **GrayScale** filter and this is done by creating a button and linking it to the item on the **activity_edit_pictures.xml** file by using

**findViewById(R.id.grayScale_button)**. Once that is completed we can add the **setOnClickListener** which will pass the bitmap to the filter class and this can be seen below. Once the image is modified we need to again load that image to the Image View so the user is able to see the modified image.

```
/* GrayScaleFilter */
Button grayScaleButton = (Button) findViewById(R.id.grayScale_button);
grayScaleButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Log.i(TAG, "GrayScaleFilter");

        for (int i = MainActivity.LEFT_IMAGE; i <= MainActivity.RIGHT_IMAGE; i++) {

            _filteredBitmaps[i] = _GrayScaleFilter
                    .filterImage(_filteredBitmaps[i]);
        }
        _filterImageView
                .setImageBitmap(_filteredBitmaps[MainActivity.LEFT_IMAGE]);
    }
});
```

The filters that do not need user input are implemented similarly to **GrayScaleFilter** except the only difference is that the Filter type is changed. I have implemented many filters that require user to specify value(s) that would determine the degree at which the filter is applied. Below is an example of **ContrastAdjust** filter which needs **Gain** and **Bias** input from the user. To do so I have used an **AlertDialog** to do so and then pass those values to the **Filter** class.

```
/* ContrastAdjust */

Button ContrastButton = (Button) findViewById(R.id.contrast_button);
ContrastButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.i(TAG, "ContrastAdjust");

        popupBiasGain(v);
    }
});
```

The **Alert Dialog** is pretty simple to use and is similar to requesting user input for Java but in case the user just gets prompted in an alert window. The message to be displayed for the user is shown below and I have also given an example of an accepted input. The user is only able to see a keyboard that looks like below. The problem with just **KEYBOARD_12Key** is that it will not let the user type a decimal point and in that case I need to set the input type

as **TEXT**. After the input is entered by the user, I need split the input for **Bias** and **Gain** using **split()** and parse them both.


```
AlertDialog.Builder alert = new AlertDialog.Builder(v.getContext());

alert.setTitle("User Input");
alert.setMessage("Enter an integer value for Bias between 0 and 255 followed by a " +
        "double value for Gain between 0.0 and 2.0 separated by a space. For example: 50 1.1");

// Set an EditText view to get user input
final EditText input = new EditText(getApplicationContext());
alert.setView(input);
input.setRawInputType(Configuration.KEYBOARD_12KEY
        | InputType.TYPE_CLASS_TEXT);

alert.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                // Canceled.
            }
        });
alert.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        String values = input.getText().toString();

        String[] parts = values.split(" ");
        String biasString = parts[0];
        String gainString = parts[1];

        bias = Integer.parseInt(biasString);

        gain = Double.parseDouble(gainString);
        contrastAdjustFilter(bias, gain);

    }
});
```

Once that is completed, the bias and gain values are passed to another method which is used to update the pixel values. After that the user is prompted of their user input as shown in the below picture.

```
private void contrastAdjustFilter(int bias, double gain) {

    for (int i = MainActivity.LEFT_IMAGE; i <= MainActivity.RIGHT_IMAGE; i++) {

        _filteredBitmaps[i] = _ContrastAdjust.filterImage(
                _filteredBitmaps[i], gain, bias);

    }

    _filterImageView
            .setImageBitmap(_filteredBitmaps[MainActivity.LEFT_IMAGE]);
    Toast.makeText(
            getApplicationContext(),
            "The RGB values of the image are altered by multiplying Gain value of "
                    + gain + " and then the Bias value of " + bias
                    + " is added to the RGB values.", Toast.LENGTH_LONG)
            .show();

}
```

## Saving the images

Once the user has finished modifying the images using filters, the "**Save**" allows the user to save the images in a folder called "**Picca**". Once the button is clicked the **applyFilter()** is called which uses **FileOutputStream** to save the images based on the image paths. Next step is to compress the bitmap (in our case the quality is 100%) and save it in the **JPEG** format and then close the stream. A similar method is used for reloading the state of the Activity after the **RGB Histogram** button is clicked by the user. Once the user clicks the **RGB Histogram** button, the method saves the images at a location and reloads them back again the **ImageView**.

```
private void applyFilter() {
    new Thread(new Runnable() {
        public void run() {
            for (int i = MainActivity.LEFT_IMAGE; i <= MainActivity.RIGHT_IMAGE; i++) {
                try {
                    if (_filteredBitmaps[i] != null) {
                        FileOutputStream fos = new FileOutputStream(
                                new File(_imagesPath[i]));
                        _filteredBitmaps[i].compress(CompressFormat.JPEG,
                                MainActivity.PICTURE_QUALITY, fos);
                        fos.close();
                    }
                } catch (FileNotFoundException e) {
                    Log.d(TAG, "File not found: " + e.getMessage());
                } catch (IOException e) {
                    Log.d(TAG, "Error accessing file: " + e.getMessage());
                }
            }
        }
    }).start();
}
```

In image processing, **JPEG** is a commonly used method of lossy compression. The **JPEG** abbreviation stands for Joint Photographic Experts Group and there are many reasons why I am saving images in the **JPEG** format. First, this format is suitable for images that contain many continuous colours such as images captured by the camera. Photos are typically made up of thousands of colours. The **JPEG** format can handle that many colours as well as keep the file size to a minimum. The file size might not be as small as a **GIF**, but the size is still pretty small compared to other formats for similar quality.

The degree of compression can be adjusted, but trying to decrease the file size will result in loss of quality and this is why I have kept the quality at 100%.

# Different types of filters - Filter.java

To filter images with supplied input from the user, I need to pass those values as parameter to the **Filter** type. To do so, I need to overload the **filterImage()** with different parameters and this can be seen in the below picture:

```
    protected Bitmap filterImage(Bitmap image) {
        return image;
    }

    protected Bitmap filterImage(Bitmap image, int value) {
        // TODO Auto-generated method stub

        return image;
    }

    protected Bitmap filterImage(Bitmap image, double value, int value1) {
        return image;
    }

    protected Bitmap filterImage(Bitmap image, int type, float percent) {
        return image;
    }

    protected Bitmap filterImage(Bitmap bitmap, double red, double green,
            double blue) {
        // TODO Auto-generated method stub
        return null;
    }

    protected Bitmap filterImage(Bitmap bitmap, String message) {
        // TODO Auto-generated method stub
        return null;
    }

    protected Bitmap filterImage(Bitmap image, float border) {
        // TODO Auto-generated method stub
        return image;
    }
}
```

This would allow me to create a **Filter** object in the **EditPicturesActivity** class by doing the following:

**Filter _MedianFilter = new MedianFilter();**

And then I can do the following to modify the image:

**_filteredBitmaps[i] = _MedianFilter.filterImage(_filteredBitmaps [i]);**

This would pass the image as a parameter to the **MedianFilter** class which would carry out the image processing and then return an updated image and load it back to the **ImageView.**

## Brightness Filter - BrightnessIncrease.java

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class BrightnessIncrease extends Filter {
    protected Bitmap filterImage(Bitmap image, int bias) {

        int width = image.getWidth();
        int height = image.getHeight();
        int R, G, B, A;
        int[] pixels = new int[width * height];

        image.getPixels(pixels, 0, width, 0, 0, width, height);

        int index = 0;
        Bitmap returnBitmap = Bitmap.createBitmap(width, height,
                Bitmap.Config.ARGB_8888);
```

The start of BrightnessIncrease.java class extends the Filter.java class shows that the bias and the image are the parameters. The image passed will be the image captured by the camera whereas the bias is an integer that will be supplied by the user (using **AlertDialog** and **TextField**). The width and height of the image is necessary to process the current image and create a processed image of the same size. We need a pixel array that would store all the pixel vales of the current image. The **getPixels()** returns all pixels of the image and stores it in the **pixel array** and this is more efficient than the **getPixel()** method because you only need to call it once to get all pixel values.

After I have the width and height, I am able to create a Bitmap; however it will not contain any pixel data. The important aspect of the **createBitmap()** is the configuration of the Bitmap and in this case I have used **Bitmap.Config.ARGB_8888** . The **ARGB_8888** is used because each pixel is stored on 4 bytes. Each channel (RGB and alpha or 'A' for translucency) is stored with 8 bits of precision (256 possible values.) This configuration is very flexible and offers the best quality. It should be used whenever possible.

Next, I would need to span the whole image and change value of each channel on each pixel value and to do that I need to separate the ARGB values and then add the bias to it and then combine it back again. I was able to use the compsci373 tutorial to separate each channel of each pixel. After separating I then added bias to the RGB value and applied a **min()** function so that the value does not exceed 255.

```
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                // using the cs373 tutorial pdf I found a way to separate the
                // values of ARGB
                A = (pixels[index] >> 24) & 0xFF;
                R = (pixels[index] >> 16) & 0xFF;
                G = (pixels[index] >> 8) & 0xFF;
                B = pixels[index] & 0xFF;

                // the algorithm used to apply the brightnessIncrease
                // Adjustment by adding a constant offset, or bias b to pixel
                // values of
                // an image g to form the new image f :
                // f(x; y) = g(x; y) + b

                // the following makes sure that if the value of RGB exeeds 255
                // after
                // the addition of the bias then use 255.
                R = Math.min(R + bias, 255);
                G = Math.min(G + bias, 255);
                B = Math.min(B + bias, 255);

                // updates the pixel values to the new RGB calculations
                pixels[index++] = (A << 24) | (R << 16) | (G << 8) | B;

            }

        }

        returnBitmap.setPixels(pixels, 0, width, 0, 0, width, height);
        // now the returnBitmap contains the updated values of RGB by using the
        // updated pixels[] array.
        return returnBitmap;

    }
}
```

After changing the RGB values, the **returnBitmap** (the modified Bitmap) is updated with the modified values and then returned and can be shown in **ImageView**.

## Modifying Contrast - ContrastAdjust.java

Contrast enhancements improve the perceptibility of objects in the images by enhancing the brightness difference between objects and their backgrounds. The term 'contrast' refers to the separation between the darkest and brightest areas of the image. When the contrast is increased, the separation between the dark and bright areas is also increased which results in making the shadows darker and highlights brighter. Similarly, decreasing the contrast will cause the shadows to increase and highlights to decrease.

For this filter I am using a linear contrast increase formula which involves the use of **gain** and **bias**. This class is similar to **BrightnessIncrease.java** and there is a minor difference in the algorithm and can be seen below:

```
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        A = (pixels[index] >> 24) & 0xFF;
        R = (pixels[index] >> 16) & 0xFF;
        G = (pixels[index] >> 8) & 0xFF;
        B = pixels[index] & 0xFF;

        //Algorithm:

        // constant gain a and offset, or bias b to pixel values of an
        // image g to form the new image f: f(x; y) = ag(x; y) + b
        // Similar to BrightnessIncrease but the RGB values get
        // multiplied by the value of gain and then bias is added to it.
        // Both of these values will be supplied by user

        R = (int) Math.min(gain * R + bias, 255);
        G = (int) Math.min(gain * G + bias, 255);
        B = (int) Math.min(gain * B + bias, 255);
        pixels[index++] = (A << 24) | (R << 16) | (G << 8) | B;
    }
}
```

This algorithm uses '**gain**' and '**bias**' to modify each channel value of each pixel of the image by multiplying the value of gain to the current RGB values and then adding bias to it. The **min()** function is used to make sure the values don't exceed 255 after modification and then the **returnBitmap** is updated and returned.

Increasing contrast and brightness when not needed could also destroy the image. For example, the distribution of RGB values of the image is already quite even but attempting to increase the brightness or contrast may cause the values to exceed 255 (when values exceed 255, I set them to 255) and causing them to increase the frequency of 255

excessively. A user with some basic knowledge of histograms will be able to aid their decision making process using my RGB histogram display feature in my project. There is another way to modify contrast and it is known as linear stretching or contrast stretching and it is explained in detail in the next page.

## Linear Stretching - ContrastStretch.java

Contrast stretch is another process to modify the contrast of an image. A contrast stretch improves the brightness differences uniformly across the dynamic range of the image, whereas tonal enhancements improve the brightness differences in the shadow (dark), midtone (grays), or highlight (bright) regions at the expense of the brightness differences in the other regions. This process re-distributes RGB values of an image over a wider or narrower range of values and in my case I have decided to stretch them from 0 to 255. In my project, I am using two types of stretch methods, linear/contrast stretching and image equalization (Spatial Analyst).

Contrast stretching is achieved by the following formula:

Linear stretching uses the following formula:

OUTVAL = (INVAL - INLO) * ((OUTUP-OUTLO)/(INUP-INLO)) + OUTLO

where:

| | |
|---|---|
| OUTVAL | Value of pixel in output map |
| INVAL | Value of pixel in input map |
| INLO | Lower value of 'stretch from' range |
| INUP | Upper value of 'stretch from' range |
| OUTLO | Lower value of 'stretch to' range |
| OUTUP | Upper value of 'stretch to' range |

When the '**stretch from**' range is specified as values, these are **INLO** and **INUP**. These are the values from the input image and refer to the minimums and maximum of the RGB arrays. The **OUTLO** and the **OUTUP** values are determined by me and they represent the spread of the new values and I have decided to just stretch them from 0 to 255. The **OUTVAL** represent the final value after the stretching. The below image shows part of how I have implemented contrast stretch filter (Spatial Analyst):

```
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        A = (pixels[index] >> 24) & 0xFF;
        R = (r[index] - Rmin) * ((255 - 0) / (Rmax - Rmin)) + 0;
        G = (g[index] - Gmin) * ((255 - 0) / (Gmax - Gmin))+ 0;
        B = (b[index] - Bmin) * ((255 - 0) / (Bmax - Bmin))+ 0;


        if (R > 255)
            R = 255;
        if (G > 255)
            G = 255;
        if (B > 255)
            B = 255;

        if (R < 0)
            R = 0;
        if (G < 0)
            G = 0;
        if (B < 0)
            B = 0;

        // convert it back
        pixels[index] = (A << 24) | (R << 16) | (G << 8) | B;
        index++;

    }
}
```

## Convert to Grayscale - GrayScale.java

I am implementing a grayscale filter which results in an image which has colours that are shades of gray. The reason for differentiating such images from any other sort of colour image is that less information needs to be provided for each pixel. The gray colour is achieved when RGB components all have the same intensity in the RGB colour space and this basically means that we need to specify a single intensity value for each pixel as opposed to three.

Grayscale images are common in world of imaging; however the camera feature on a device captures the image in colour. Therefore, the user has to manually convert the image to Grayscale using desktop applications but Picca has a grayscale filter that sets the saturation to 0 by using the **setSaturation()** and a value of 0 maps the colour to gray-scale and 1 is identity.

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class GrayScaleFilter extends Filter {

    @Override
    protected Bitmap filterImage(Bitmap image) {
        int width, height;
        height = image.getHeight();
        width = image.getWidth();

        Bitmap bmpGrayscale = Bitmap.createBitmap(width, height, Bitmap.Config.RGB_565);
        Canvas c = new Canvas(bmpGrayscale);
        Paint paint = new Paint();
        ColorMatrix cm = new ColorMatrix();
        cm.setSaturation(0);
        ColorMatrixColorFilter f = new ColorMatrixColorFilter(cm);
        paint.setColorFilter(f);
        c.drawBitmap(image, 0, 0, paint);
        return bmpGrayscale;
    }
}
```

## Histogram Equalization Filter - HistogramEqualization.java

Histogram equalization is also referred to as Image Equalization and used on images that looks plain and lacks contrast. This algorithm is also available in the desktop application called Photoshop when the user clicks the “quick fix” button. My goal was to implement it on a mobile platform so it can be used with portable android devices.

A histogram is a graph of a sort that plots the frequency at which each grey level occurs from 0 (black) to 255 (white) (in black and white images). I am familiar with this algorithm and have implemented it for Grayscale images in one of my previous courses however, that same algorithm would not work in my case because the camera captures images with colour. With colour images, I need to plot the frequency of Red, Green and Blue levels of the images. In images that have “poor contrast” (when majority of frequencies lie in the middle of histogram), histogram equalization fixes these (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

The primary goal of this algorithm is to obtain a uniform histogram. So suppose if I capture an image and if it has areas that have peak frequencies, even after the histogram equalization there will still be areas with peaks but they will be shifted. In simple words, the existing values will be mapped to new values but the actual number of intensities in the resulting image will be equal to or less than the original number of intensities.

Histogram equalization can be done in three steps:

- compute histogram,
- calculate the normalized sum of the histogram,
- transform input image to output image

So I created a method called **imageHistogram** that starts off with getting each pixel value of the image. As mentioned before in my report, each pixel value contains three colour channels, RGB and then I increase the values of the colours in the table (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

```
public static ArrayList<int[]> imageHistogram(Bitmap image) {
    int red, green, blue;
    int[] histoR = new int[256];
    int[] histoG = new int[256];
    int[] histoB = new int[256];

    for (int i = 0; i < histoR.length; i++)
        histoR[i] = 0;
    for (int i = 0; i < histoG.length; i++)
        histoG[i] = 0;
    for (int i = 0; i < histoB.length; i++)
        histoB[i] = 0;

    for (int x = 0; x < image.getWidth(); x++) {
        for (int y = 0; y < image.getHeight(); y++) {

            red = Color.red(image.getPixel(x, y));
            green = Color.green(image.getPixel(x, y));
            blue = Color.blue(image.getPixel(x, y));

            // Increase the values of colors
            histoR[red]++;
            histoG[green]++;
            histoB[blue]++;

        }
    }
  ArrayList<int[]> hist = new ArrayList<int[]>();
  hist.add(histoR);
  hist.add(histoG);
  hist.add(histoB);

  return hist;

}
```

Next I had to calculate the normalized sum and this is the part where I had difficulty getting it right. After researching many types of resources I was able to find a similar example to mine. The below method calculates the cumulative distribution function which is also the images accumulated normalized histogram. Then I call the method created before to get the histogram. After that I created a **LUT** which would correspond to each colour in our RGB channel and then I fill the table with 0's to calculate the scale factor. The next step would be to go through a loop in which the value of the current pixel needs to be added to the sum. The new value is calculated by multiplying the sum with the scale factor and similar to other filters, if the sum exceeds 255 then 255 is used in its place (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

```
// Get the histogram equalization lookup table for separate R, G, B channels
private static ArrayList<int[]> BitmapEqualization(Bitmap input) {

    // Get an image histogram - calculated values by R, G, B channels
    ArrayList<int[]> imageHist = imageHistogram(input);

    // Create the lookup table
    ArrayList<int[]> imageLUT = new ArrayList<int[]>();

    // Fill the lookup table
    int[] histoR = new int[256];
    int[] histoG = new int[256];
    int[] histoB = new int[256];

    for (int i = 0; i < histoR.length; i++)
        histoR[i] = 0;
    for (int i = 0; i < histoG.length; i++)
        histoG[i] = 0;
    for (int i = 0; i < histoB.length; i++)
        histoB[i] = 0;
    long sumr = 0;
    long sumg = 0;
    long sumb = 0;

    // Calculate the scale factor
    float scale_factor = (float) (255.0 / (input.getWidth() * input
            .getHeight()));

    for (int i = 0; i < histoR.length; i++) {
        sumr += imageHist.get(0)[i];
        int valr = (int) (sumr * scale_factor);

        // if it doesnt match ImageJ try the following
        // int valr = (int) (Math.sqrt(sumr * scale_factor));
        if (valr > 255) {
            histoR[i] = 255;
        } else
            histoR[i] = valr;

          sumg += imageHist.get(1)[i];
          int valg = (int) (sumg * scale_factor);
          // if it doesnt match ImageJ try the following
          // int valG = (int) (Math.sqrt(sumg * scale_factor));
          if (valg > 255) {
              histoG[i] = 255;
          } else
              histoG[i] = valg;

          sumb += imageHist.get(2)[i];
          // if it doesnt match ImageJ try the following
          // int valb = (int) (Math.sqrt(sumb * scale_factor));
          int valb = (int) (sumb * scale_factor);
          if (valb > 255) {
              histoB[i] = 255;
          } else
              histoB[i] = valb;
      }

      imageLUT.add(histoR);
      imageLUT.add(histoG);
      imageLUT.add(histoB);

      return imageLUT;

  }
```

The last part that I need to do is just call the method for equalization, create a new image and set the new pixel values. This can be achieved by the following method:

```
protected Bitmap filterImage(Bitmap image) {

    int width = image.getWidth();
    int height = image.getHeight();
    int red;
    int green;
    int blue;
    int alpha;
    int newPixel = 0;

    // Get the Lookup table for histogram equalization
    ArrayList<int[]> histLUT = BitmapEqualization(image);

    Bitmap returnBitmap = Bitmap.createBitmap(width, height,
            Bitmap.Config.ARGB_8888);



  for (int x = 0; x < width; x++) {
      for (int y = 0; y < height; y++) {

          // Get pixels by R, G, B
          alpha = Color.alpha(image.getPixel(x, y));
          red = Color.red(image.getPixel(x, y));
          green = Color.green(image.getPixel(x, y));
          blue = Color.blue(image.getPixel(x, y));

          // Set new pixel values using the histogram lookup table
          red = histLUT.get(0)[red];
          green = histLUT.get(1)[green];
          blue = histLUT.get(2)[blue];

          // Return back to original format
          newPixel = colorToRGB(alpha, red, green, blue);

          // Write pixels into image
          returnBitmap.setPixel(x, y, newPixel);

      }
  }
```

## Drawing histogram using View - DrawHistograms.java

Filter like Histogram Equalization modifies the colour values of RGB and maps them evenly. This feature allows user to see the distribution of the RGB values as separate histograms regardless of which filter is applied and this is an important feature because it can give an indication to which filter the user must apply. For example, a user has captured an image that has majority of values distributed on the left hand side of the histogram which means that the image is dark and this could indicate a few options of filters such as Histogram Equalization, Brightness Increase, Contrast Adjust and so on. To draw the histograms, I am extending the View class and using onDraw() of Canvas to draw everything on the screen.

```
public class DrawHistograms extends View {
    Bitmap mBitmap;
    Paint _paintBlack;
    Paint _paintWhite;
    Paint _paintRed;
    Paint _paintGreen;
    Paint _paintBlue;
    byte[] mYUVData;
    int[] mRGBData;
    int mImageWidth, mImageHeight;
    int[] mRedHistogram;
    int[] mGreenHistogram;
    int[] mBlueHistogram;
    double[] mBinSquared;
    Paint paint = new Paint();
```

First, I need to declare the bitmap that will be used to calculate the RGB histograms and also the Paint components which will be the colour of the font and histograms. To keep it simple I have kept the colours of RGB histograms as red, green, and blue respectively. Also to store the data of RGB values I need to create separate integer arrays so that the histograms have data to be mapped. Next, I need to initialize the paint components with the colour, fill and font size that will be drawn on the Canvas (ViewFinderEE368). The below image shows how this is done and is self-explanatory of what is being done. The RGB histogram arrays are set to size 256 because of the range of colour values 0-255.

```
    public DrawHistograms(Context context) {
        super(context);

        _paintRed = new Paint();
        _paintRed.setStyle(Paint.Style.FILL);
        _paintRed.setColor(Color.RED);
        _paintRed.setTextSize(25);

        _paintGreen = new Paint();
        _paintGreen.setStyle(Paint.Style.FILL);
        _paintGreen.setColor(Color.GREEN);
        _paintGreen.setTextSize(25);

        _paintBlue = new Paint();
        _paintBlue.setStyle(Paint.Style.FILL);
        _paintBlue.setColor(Color.BLUE);
        _paintBlue.setTextSize(25);
```

```
        _paintBlack = new Paint();
        _paintBlack.setStyle(Paint.Style.FILL);
        _paintBlack.setColor(Color.BLACK);
        _paintBlack.setTextSize(25);

        _paintWhite = new Paint();
        _paintWhite.setStyle(Paint.Style.FILL);
        _paintWhite.setColor(Color.WHITE);
        _paintWhite.setTextSize(25);

        mBitmap = null;
        mYUVData = null;
        mRGBData = null;
        mRedHistogram = new int[256];
        mGreenHistogram = new int[256];
        mBlueHistogram = new int[256];
        mBinSquared = new double[256];
        for (int bin = 0; bin < 256; bin++) {
            mBinSquared[bin] = ((double) bin) * bin;
        } // bin
    }
```

In this case, we are dealing with two colour spaces, RGB and YUV. The reason I need to use this is because it encodes a colour image or video taking human perception into account. This means that the bandwidth for the chrominance components is reduced resulting in masking errors such as transmission errors or compression artifacts as compared to the RGB format. The conversion from YUV to RGB is called 'decodeYUV420SP' (ViewFinderEE368) and the conversion from RGB to YUV is called 'encodeYUV420SP'. Understanding the conversion is slightly complicated and is out of scope for my project which is why I used pre-defined conversion methods widely available from the internet to avoid errors in calculations and can be found in the project class (Pratt, 2007).

```
    static public void doIntensityHisto(int[] rgb, int[] histogram, int width,
            int height, int histogramType) {
        for (int bin = 0; bin < 256; bin++) {
            histogram[bin] = 0;
        } // bin
        if (histogramType == 0) // red
        {
            for (int pix = 0; pix < width * height; pix++) {
                int pixelValue = (rgb[pix] >> 16) & 0xff;
                histogram[pixelValue]++;
            } // pix
        } else if (histogramType == 1) // green
        {
            for (int pix = 0; pix < width * height; pix++) {
                int pixelValue = (rgb[pix] >> 8) & 0xff;
                histogram[pixelValue]++;
            } // pix
        } else // blue
        {
            for (int pix = 0; pix < width * height; pix++) {
                int pixelValue = rgb[pix] & 0xff;
                histogram[pixelValue]++;
            } // pix
        }
    }
```

Above method calculates the intensities of each colour channel and then increments the histogram.

At the moment, you can notice that the mBitmap is kept null and the basically everything works only if the mBitmap is kept null. This is because I will initialize the RGB data, YUV data and the Bitmap in another class when the surface is created and as I mentioned before, this class is used as a blueprint for the final product. I want to cover the full screen for this feature because the histograms will be quite large and it would be difficult to get all the data displayed in small screen space. The canvas dimensions are gathered to be used later and also the decoding of YUV to RGB needs to be done at the start. Once that is completed the intensities of the histograms are calculated and stored in their respective histogram arrays and all this can be seen below (Canvas and Drawables).

```
protected void onDraw(Canvas canvas) {
    if (mBitmap != null) {

        int canvasWidth = canvas.getWidth();
        int canvasHeight = canvas.getHeight();
        int newImageWidth = canvasWidth;
        int newImageHeight = canvasHeight;
        int marginWidth = (canvasWidth - newImageWidth) / 2;

        // Convert from YUV to RGB
        decodeYUV420SP(mRGBData, mYUVData, mImageWidth, mImageHeight);

        // Calculate histogram
        doIntensityHisto(mRGBData, mRedHistogram, mImageWidth,
                mImageHeight, 0);
        doIntensityHisto(mRGBData, mGreenHistogram, mImageWidth,
                mImageHeight, 1);
        doIntensityHisto(mRGBData, mBlueHistogram, mImageWidth,
                mImageHeight, 2);
```

The doIntensityHisto must be called thrice because of the 3 colour channels but for this to work all the variables need to be initialized and this is done in another class which I have included later. The next step would be to calculate the mean and standard deviation and this is where it got tricky to implement and after a few days of researching online and it is easy to source algorithms for mean and standard deviations. After the mean and SD calculations, I need to display the mean and SD on the canvas for the user to see. So I use the canvas.drawText() which takes the string, float x, float y and paint components as parameters. This basically means that the text is drawn with the origin at (x, y), using the specified paint and in this case it is the white font colour. Keep in mind that the background is black. Next the intensity histograms are drawn using Rect object and that can be found in my project folder.

## Previewing RGB histograms - Preview.java

In the last couple of pages I discussed about the blueprint of drawing histograms and to draw the histograms I need to initialize the bitmap, RGB and YUV data arrays and so on. I am using a class called Preview.java which extends **SurfaceView** and implements **SurfaceHolder.Callback**. Usually **SurfaceView** is used with developing games but I decided to use it with my project as it was a simple implementation without any complexity (Sharma).

```
public class Preview extends SurfaceView implements SurfaceHolder.Callback {
    SurfaceHolder mHolder;

    DrawHistograms mDrawHistograms;
    boolean mFinished;

    Preview(Context context, DrawHistograms drawOnTop) {
        super(context);

        mDrawHistograms = drawOnTop;
        mFinished = false;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
```

Now you will notice that the Constructor has Context as well as **DrawHistograms** as the parameters and that is because I need to initialize everything here and can also use it as shown below in my EditPicturesActivity.java class.

```
// Create our Preview view and set it as the content of our
// activity.
// Create our DrawOnTop view.
mDrawHistograms = new DrawHistograms(getApplicationContext());
mPreview = new Preview(getApplicationContext(), mDrawHistograms);
setContentView(mPreview);
addContentView(mDrawHistograms, new LayoutParams(
        LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
```

Next, I need to initialize everything when the surface is created and not before. Since I have implemented SurfaceHolder interface, I have implemented a method called **surfaceCreated(SurfaceHolder)** because this is called at least once when I make changes. First I need to gather the image dimensions from the EditPicturesActivity so that I can

declare the size of **mRGBdata** array and this can be seen below. Next I need to get pixel values (and store it in the mRGBdata array) of the image I want the histograms to be drawn for and this can get done by **getPixels()**. Next, I update the **mBitmap** in the DrawHistograms class by the **setPixels()** and updating it by the values using the **mRGBdata** array.

```
public void surfaceCreated(SurfaceHolder holder) {
    mDrawHistograms.mImageWidth = EditPicturesActivity._filteredBitmaps[0]
            .getWidth();
    mDrawHistograms.mImageHeight = EditPicturesActivity._filteredBitmaps[0]
            .getHeight();
    mDrawHistograms.mRGBData = new int[mDrawHistograms.mImageWidth
            * mDrawHistograms.mImageHeight];

    EditPicturesActivity._filteredBitmaps[0].getPixels(mDrawHistograms.mRGBData, 0,
            mDrawHistograms.mImageWidth, 0, 0, mDrawHistograms.mImageWidth,
            mDrawHistograms.mImageHeight);

    mDrawHistograms.mBitmap = Bitmap.createBitmap(mDrawHistograms.mImageWidth,
            mDrawHistograms.mImageHeight, Bitmap.Config.ARGB_8888);

    mDrawHistograms.mBitmap.setPixels(mDrawHistograms.mRGBData, 0,
            mDrawHistograms.mImageWidth, 0, 0, mDrawHistograms.mImageWidth,
            mDrawHistograms.mImageHeight);

    mDrawHistograms.mYUVData = new byte[mDrawHistograms.mImageWidth
            * mDrawHistograms.mImageHeight * 3 / 2];
    DrawHistograms.encodeYUV420SP(mDrawHistograms.mYUVData, mDrawHistograms.mRGBData,
            mDrawHistograms.mImageWidth, mDrawHistograms.mImageHeight);


    invalidate();

}
```

The problem I had during this process was finding the size of the YUV array and after some research I was able to find a solution that works. The **encodeYUV420SP()** is also found at the same resource http://stackoverflow.com/questions/5960247/convert-bitmap-array-to-yuv-ycbcr-nv21. Even though all the values are updated in the Preview class, I still call invalidate and this forces the view to draw.

## Median Filter - MedianFilter.java

I have implemented Median Filter as one of the filters used for noise reduction on the captured image. This type of implementation is non-linear and we can use noise reduction prior to further image processing (pre-processing) so that the end image is free of noise when you use a filter like Edge Detection.

Basically the idea of a median filter is to iterate through the image pixels and replacing the pixel values with the median of the neighbouring entries. This is achieved by the help of a "window" which slides along the whole image and to keep things simple, I have used a box pattern of window (3x3 square box window) (Vandevenne, 2004).

Just like Image Equalization, this process takes a while to filter the image as the each entry of the image must be processed and the window slides and repeats. This brings me to Edge Preservation during the median filtering process.

As mentioned, median filtering is already a slow process and preserving the edges would take more processing time and therefore I decided not to process edges. This would also yield the most accurate results as padding edges with values do tend to give inaccurate results and also there are many padding options. With the image being so large and the window size being only 3, I decided to not preserve edges because they would only slow down the process and would be a feasible option only if the image was small and the

```
// no padding neccessary
for (int x = size / 2; x < width - (size / 2); x++) {
    for (int y = size / 2; y < height - (size / 2); y++) {

        int index = 0;

        for (int filterX = -filterWidth / 2; filterX <= filterWidth / 2; filterX++) {
            for (int filterY = -filterHeight / 2; filterY <= filterHeight / 2; filterY++) {

                A = (pixels[x + filterX + (width) * (y + filterY)] >> 24) & 0xFF;
                R = (pixels[x + filterX + (width) * (y + filterY)] >> 16) & 0xFF;
                G = (pixels[x + filterX + (width) * (y + filterY)] >> 8) & 0xFF;
                B = pixels[x + filterX + (width) * (y + filterY)] & 0xFF;

                RArray[index] = R;
                GArray[index] = G;
                BArray[index] = B;
                ++index;
            }
        }
        Arrays.sort(RArray);
        Arrays.sort(GArray);
        Arrays.sort(BArray);
        medianR = RArray[RArray.length / 2];
        medianG = GArray[GArray.length / 2];
        medianB = BArray[BArray.length / 2];
```

changes were visible after application. Below is the algorithm for median filtering that does

not involve preserving the edges. The first two for loops are for iterating through each pixel value and the other two for loops take care of the sliding window through the image. The last part sorts the arrays of RGB values and then selects the median value (Vandevenne, 2004).

Currently, the size of the window is odd (3x3) but if in the future you change to an odd window size such as 4x4 then there is a problem selecting the median value. In that case I have added the following code to process even window sizes (Vandevenne, 2004).

```
// take the median, if the length of the array is even,
// take the average of both center values
if ((filterWidth * filterHeight) % 2 == 1) {
    RResult[x][y] = medianR;
    GResult[x][y] = medianG;
    BResult[x][y] = medianB;

} else if (filterWidth >= 2) {
    RResult[x][y] = (RArray[RArray.length / 2] + RArray[RArray.length / 2 + 1]) / 2;
    GResult[x][y] = (GArray[GArray.length / 2] + GArray[GArray.length / 2 + 1]) / 2;
    BResult[x][y] = (BArray[BArray.length / 2] + BArray[BArray.length / 2 + 1]) / 2;
}

tempR = RResult[x][y];
tempG = GResult[x][y];
tempB = BResult[x][y];

returnBitmap.setPixel(x, y, Color.argb(A, tempR, tempG, tempB));
```

In my project, I have added a few noise removing filters such Gaussian Blur but each filter is applicable to certain scenarios. For small to moderate level of noise, the median filter is demonstrably better than Gaussian blur at removing noise even with preserving edges.

However, the Gaussian blur performs better for high levels of noise but majority of times, the image only has speckle noise or pepper noise and so median filter is recommended for that particular instance.

## Camera Image flipping - FlipImage.java

In many cases, the camera does not flip image correctly and the image looks off. This filter allows the user to flip the image vertically or horizontally through user input. The camera package in my project makes sure that the image is flipped accordingly to the device but it cannot include all devices which is why it is essential to implement this filter so that the user can manually change the image.

```
if(type == FLIP_VERTICAL) {
    // y = y * -1
    matrix.preScale(1.0f, -1.0f);
}
// if horizonal
else if(type == FLIP_HORIZONTAL) {
    // x = x * -1
    matrix.preScale(-1.0f, 1.0f);
// unknown type
} else {
    return null;
}
```

The prescale() preconcats the matrix with the specified scale. M' = M * S(sx, sy) and in this case the image is flipped vertically using the negative ‘y’ value and the opposite for horizontally using the negative ‘x’ value.

## Tinting Image - TintingImage.java

While researching, I came across an interesting algorithm here: http://www.developer.com/ws/android/programming/Working-with-Images-in-Googles-Android-3748281-2.htm that tints images according to input level. It is difficult to understand the complete algorithm. But basically, each pixel is tinted using an angle in degrees and after the process is finished all the pixels are ensured to stay within the 0-255 range.

## Writing message on the image - MessageOnImage.java

Sometimes, when sending images through email or other means, the only way to keep it from copied and used in any matter, we watermark it. For my project, I have implemented a filter that will place text on top of the captured image. I could use a design of some sort to watermark it but it would not be useful for all applications such as sending the same image from multiple entities as each entity would have a different watermark. Hence, some text on the image would make it unique as well as provide control to the user as to what will appear on the image. In many cases, the user can write a message on the same such as “The water is leaking through this location” instead of explaining it to the party on the other end through descriptive messages.

```
protected Bitmap filterImage(Bitmap image, String message) {
    int w = image.getWidth();
    int h = image.getHeight();
    Bitmap returnBitmap = Bitmap.createBitmap(w, h, image.getConfig());
    int size = 25;
    boolean underline = true;
    int alpha = 200;

    Point position = new Point(image.getWidth()/2 , image.getHeight()/2);

    Canvas canvas = new Canvas(returnBitmap);
    canvas.drawBitmap(image, 0, 0, null);

    Paint paint = new Paint();
    paint.setColor(Color.WHITE);
    paint.setAlpha(alpha);
    paint.setTextSize(size);
    paint.setAntiAlias(true);
    paint.setUnderlineText(underline);
    canvas.drawText(message, position.x, position.y, paint);

    return returnBitmap;
}
```

The approach for this filter is quite different to previous filters because in this filter, we do not need to iterate through each pixel value, instead we use the Canvas feature for Android. First we get the image Width and Height and create the bitmap using those values. Then we define the text size of the watermark. Then we create the bitmap on the Canvas using drawBitmap. Once that is completed, we create a Paint object and this is used to hold the style and colour information about the text to be displayed on the image. The position of the text can be modified but at the moment I have kept it in the center of the image.

## 'Inverting' colours - ReverseRGB.java

Inverting colour is a feature offered by Photoshop and as the name suggests, it inverts the values of each pixel value and its colour channel. There are a couple of uses for this filter, one of them being making an edge mask to apply sharpening and other adjustments to selected areas of an image and the second use is for visually impaired people who cannot see certain detail in the image that they are able to see after applying this filter.

When you invert an image, the brightness value of each pixel in the channels is converted to the inverse value on the 256-step colour-values scale. For example, a pixel in a positive image with a value of 255 is changed to 0, and a pixel with a value of 5 is changed to 250. The algorithm can be seen below (Gimel'farb & Delmas, Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms).

```
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        pixel = image.getPixel(x, y);

        A = (pixel >> 24) & 0xFF;
        R = (pixel >> 16) & 0xFF;
        G = (pixel >> 8) & 0xFF;
        B = pixel & 0xFF;

        tempR = 255 - R;
        tempG = 255 - G;
        tempB = 255 - B;

        returnBitmap.setPixel(x, y, Color.argb(A, tempR, tempG, tempB));

    }

}
```

## Saturation and Hue Filter - Saturation.java and Hue.java

HSV colour scale stands for Hue, Saturation and Value. Saturation is a feature available in many desktop image processing applications and is widely used to modify images (Hue, Value, Saturation). In my project, I will be implementing saturation algorithm using HSV colour space. The HSV[1] will be used for saturation and HSV[0] will be used for Hue. The pixels are converted to HSV colour space after iteration and then multiplied by the saturation level which is basically the intensity of saturation algorithm. The level will be supplied by the user and this is further explained in EditPicturesAcitivty.java. Once the HSV calculation is done, the HSV is converted back to colour to store in the pixel array.

```
public class Saturation extends Filter {

    protected Bitmap filterImage(Bitmap image, int level) {

        int width = image.getWidth();
        int height = image.getHeight();
        int[] pixels = new int[width * height];
        float[] HSV = new float[3];
        // get pixel array from image
        image.getPixels(pixels, 0, width, 0, 0, width, height);

        int index = 0;
        // iteration through pixels
        for (int y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                // get current index in 2D-matrix
                index = y * width + x;
                // convert to HSV
                Color.colorToHSV(pixels[index], HSV);
                // increase Saturation level
                HSV[1] *= level;
                HSV[1] = (float) Math.max(0.0, Math.min(HSV[1], 1.0));
                // take color back
                pixels[index] |= Color.HSVToColor(HSV);
            }
        }
        // output bitmap
        Bitmap returnBitmap = Bitmap.createBitmap(width, height,
                Bitmap.Config.ARGB_8888);
        returnBitmap.setPixels(pixels, 0, width, 0, 0, width, height);
        return returnBitmap;

    }

}
```

Hue is the wavelength within the visible-light spectrum at which the energy output from a source is greatest. In basic words, it refers to a tone of colour but it is not another name for colour as it can have saturation and brightness as well as hue. The code is similar to Saturation.java however, the HSV array value points to location 0 for Hue and 1 for Saturation.

```
import android.graphics.Bitmap;

public class Hue extends Filter {

    protected Bitmap filterImage(Bitmap image, int level) {

        int width = image.getWidth();
        int height = image.getHeight();
        int[] pixels = new int[width * height];
        float[] HSV = new float[3];

        image.getPixels(pixels, 0, width, 0, 0, width, height);

        int index = 0;
        // iteration through pixels
        for (int y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                // get current index in 2D-matrix
                index = y * width + x;
                // convert to HSV
                Color.colorToHSV(pixels[index], HSV);
                // increase Saturation level
                HSV[0] *= level;
                HSV[0] = (float) Math.max(0.0, Math.min(HSV[0], 360.0));
                // take color back
                pixels[index] |= Color.HSVToColor(HSV);
            }
        }
        // output bitmap
        Bitmap returnBitmap = Bitmap.createBitmap(width, height,
                Bitmap.Config.ARGB_8888);
        returnBitmap.setPixels(pixels, 0, width, 0, 0, width, height);
        return returnBitmap;

    }

}
```

## Modifying RGB colour channels – ModColourRGB.java

This class allows each RGB values to be multiplied by a double and is a quick and responsive filter. It has been implemented in a similar way to Brightness.java and ContrastAdjust.java. The heart of the algorithm is shown below and the red, green and blue values are user input through Alert Dialog.

```
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        // using the cs373 tutorial pdf I found a way to separate the
        // values of ARGB
        A = (pixels[index] >> 24) & 0xFF;
        R = (pixels[index] >> 16) & 0xFF;
        G = (pixels[index] >> 8) & 0xFF;
        B = pixels[index] & 0xFF;


        R = (int) (R * red);
        G = (int) (G * green);
        B = (int) (B * blue);


        // updates the pixel values to the new RGB calculations
        pixels[index++] = (A << 24) | (R << 16) | (G << 8) | B;

    }

}
```

# Reusability of code - ConvolutionMat.java

After discussing this project some more with my supervisor, he believed that I should consider some reusability of code to design new filters and after some research I was able to find Convolution example at http://lodev.org/cgtutor/filtering.html.I have used that to create my own Convolution Matrix algorithm that will help me design custom 2D filters. The idea is that for every pixel of the image, take the sum of the products and each product is the colour value of the current pixel or a neighbour of it, with the corresponding value of the filter matrix. The center of the filter matrix has to be multiplied with the current pixel, the other elements of the filter matrix with corresponding neighbour pixels (Vandevenne, 2004).

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class ConvolutionMat extends Filter {

    public static final int SIZE = 3;

    public double[][] Matrix;
    public double Factor = 1;
    public double Offset = 1;

    public ConvolutionMat(int size) {
        Matrix = new double[size][size];
    }

    public void setAll(double value) {
        for (int x = 0; x < SIZE; ++x) {
            for (int y = 0; y < SIZE; ++y) {
                Matrix[x][y] = value;
            }
        }
    }

    public void applyConfig(double[][] config) {
        for(int x = 0; x < SIZE; ++x) {
            for(int y = 0; y < SIZE; ++y) {
                Matrix[x][y] = config[x][y];
            }
        }
    }
```

If I am going to use Convolution Matrix to design many filters then I would need to make it re-usable and this can be achieved by adding the above methods which allows the user to set a value in a kernel matrix, set ALL values in kernel matrix and copy the kernel into the Matrix[][] which would be used to carry out the calculations (shown later). When creating filters, I mostly use the applyConfig() as I define the kernel in another class and I will explain this later with source code of a filter designed using Convolution Matrix (Vandevenne, 2004).

The filters I am using with convolution are quite simple and majority of them use 3x3 kernel but creating complex filters such as Coloured Pencil in Photoshop are also possible but they are not included in this project.

```
public static Bitmap computeConvolution(Bitmap src, ConvolutionMat matrix) {
    int width = src.getWidth();
    int height = src.getHeight();
    Bitmap returnBitmap = Bitmap.createBitmap(width, height,
            Bitmap.Config.ARGB_8888);

    int A, R, G, B;
    int sumR, sumG, sumB;
    int[][] pixels = new int[SIZE][SIZE];

    for(int y = 0; y < height - 2; ++y) {
        for(int x = 0; x < width - 2; ++x) {
             // get pixel matrix
            for(int i = 0; i < SIZE; ++i) {
                for(int j = 0; j < SIZE; ++j) {
                    pixels[i][j] = src.getPixel(x + i, y + j);
                }
            }

             A = Color.alpha(pixels[1][1]);
            sumR = sumG = sumB = 0;

            // get sum of RGB on matrix
            for(int i = 0; i < SIZE; ++i) {
                for(int j = 0; j < SIZE; ++j) {
                    sumR += (Color.red(pixels[i][j]) * matrix.Matrix[i][j]);
                    sumG += (Color.green(pixels[i][j]) * matrix.Matrix[i][j]);
                    sumB += (Color.blue(pixels[i][j]) * matrix.Matrix[i][j]);
                }
            }

            R = (int)(sumR / matrix.Factor + matrix.Offset);
            if(R < 0) { R = 0; }
            else if(R > 255) { R = 255; }

            G = (int)(sumG / matrix.Factor + matrix.Offset);
            if(G < 0) { G = 0; }
            else if(G > 255) { G = 255; }

            B = (int)(sumB / matrix.Factor + matrix.Offset);
            if(B < 0) { B = 0; }
            else if(B > 255) { B = 255; }

            returnBitmap.setPixel(x + 1, y + 1, Color.argb(A, R, G, B));
        }
    }


    return returnBitmap;
}
```

The 2D convolution operation requires 3 double for-loops, so it isn't extremely fast, unless you use small filters. Here we'll usually be using 3x3 or 5x5 filters (Vandevenne, 2004).

**There are a few rules about the filter:**

- The size of the kernel has to be uneven so that it has a center, for example 3x3, 5x5 and 7x7.
- If you want the brightness to be the same as the original then the sum of all elements of the filter must be 1 but this is not compulsory as some filters affect the brightness.
- If the sum of the elements is larger than 1, the result will be a brighter image, and if it's smaller than 1, a darker image. If the sum is 0, the resulting image isn't necessarily completely black, but it'll be very dark.

The above code has a double field called **Factor** which is used as a denominator for the kernel values.

For example a Matrix:

1/3, 1/3, 1/3
1/3, 1/3, 1/3
1/3, 1/3, 1/3

But using the Factor field, we can define the same Matrix as:

1, 1, 1
1, 1, 1
1, 1, 1

With Factor = 3 and Offset =0;

The above code also consists of a field called **Offset** which is used to add values to the RGB after filter is applied but I have left this field 0.

## Detection of Edges - EdgeDetect.java

Using the Convolution Matrix class, I was able to design a custom edge detection filter. Edges are significant local changes of intensity in an image and they typically occur on the boundary between two different regions in an image. Edge Detection is used in many fields such as image processing, machine vision, feature detection and extraction.

The main goal if edge detection filter I have implemented is to produce a line drawing of an image so that important features can be extracted from the edges of an image (e.g., corners, lines, curves).These features are used by higher-level computer vision algorithms (e.g., recognition) (8.2 Convolution Matrix).

As shown by the example below, the result of applying edge detection to an image sometimes leads to connected curves that indicate boundaries of objects, the boundaries of surface markings as well as curves that correspond to discontinuities in surface orientation.

Thus, applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image. In my application, the filters can be applied on top of each other, which mean that once the user applies Edge Detect filter, they can apply Gaussian Blur on it and then Reverse RGB. The combination of filters such as the ones mentioned just before would return a smoothened image.

If the edge detection step is successful, the subsequent task of interpreting the information contents in the original image may therefore be substantially simplified. However, it is not always possible to obtain such ideal edges from real life images of moderate complexity.

The edgeFind is a 2D double array which basically represents the 3x3 kernel I am using with the values shown above and I create a new instance of ConvolutionMat class and pass 3 as the size of the kernel. After that, I use the applyConfig() and pass the kernel which sets the matrix.Matrix in the Convolution class to the values of the kernel and carries out the calculation. The Factor is set to 1 because there is no denominator for this kernel and offset is set to 0 because I do not want to modify the value apart from the kernel (8.2 Convolution Matrix).

## Horizontal Edge Detection Filter – HorizontalEdgeDetection.java

The edge detection filter above detects both vertical and horizontal edges in an image but what if the user wants to select only the horizontal edges? In that case, I have a kernel below that is able to detect mostly horizontal edges of an image.

```
protected Bitmap filterImage(Bitmap image) {

    //another edge detection kernel
/*  double[][] edgeFind = new double[][]
        {
             {0,  0,  0,  0,  0},
             {0,  0,  0,  0,  0},
             {-1, -1, 2, 0,  0},
             {0,  0,  0,  0,  0},
             {0,  0,  0,  0,  0}
        };*/

    double[][] edgeFind = new double[][]
            {
                 {1, 0, -1},
                 {2, 0, -2},
                 {1, 0, -1}
            };

        ConvolutionMat convMatrix = new ConvolutionMat(3);
        convMatrix.applyConfig(edgeFind);
        convMatrix.Factor = 1;
        convMatrix.Offset = 0;
        return ConvolutionMat.computeConvolution(image, convMatrix);
    }
```

## Vertical Edge Detection – VerticalEdgeDetection.java

Similar to horizontal Edge Detection, I have also implemented a separate vertical edge detection algorithm that uses the following kernel. I implemented vertical and horizontal as optional features for edge detection if the user wishes to choose them separately (Rhody).

```
protected Bitmap filterImage(Bitmap image) {

        double[][] edgeFind = new double[][]
                {
                     {1,  2, 1},
                     {0,  0, 0},
                     {-1,-2, 1}
                };

            ConvolutionMat convMatrix = new ConvolutionMat(3);
            convMatrix.applyConfig(edgeFind);
            convMatrix.Factor = 1;
            convMatrix.Offset = 0;
            return ConvolutionMat.computeConvolution(image, convMatrix);
        }
```

## Emboss effect on images - Emboss.java

Using Convolution Matrix class, I create a similar class called Emboss.java. This effect causes each pixel of an image is replaced either by a highlight or a shadow, depending on light/dark boundaries on the original image. If there are any low contrast areas then they are replaced by a grey background. The resulting image often results in an image resembling a paper or metal embossing of the original image, hence the name (8.2 Convolution Matrix).

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class Emboss extends Filter {

    protected Bitmap filterImage(Bitmap image) {
        //another kernel
         /*-1, -1,  0,
            -1,  0,  1,
             0,  1,  1*/

        //http://docs.gimp.org/en/plug-in-convmatrix.html

    double[][] EmbossConfig = new double[][] {
            { -2 , -1, 0 },
            { -1 , 1, 1 },
            {0 , 1, 2 }

    };

        ConvolutionMat convMatrix = new ConvolutionMat(3);
        convMatrix.applyConfig(EmbossConfig);
        convMatrix.Factor = 1;
        convMatrix.Offset = 0;
        return ConvolutionMat.computeConvolution(image, convMatrix);
    }

}
```

Everything is kept the same as EdgeDetect except the kernel. The emboss kernel is applied and there are many available kernels for emboss but after a few trials I found the current one most accurate.

## Engrave effect on images - Engrave.java

Another filter developed using the Convolution Matrix class. This filter produces an engraving effect: the image is turned black and white and some horizontal lines of varying height are drawn depending on the value of underlying pixels (Houston, 2011).

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class Engrave extends Filter {

    public Bitmap filterImage(Bitmap image) {
        ConvolutionMat convMatrix = new ConvolutionMat(3);
        convMatrix.setAll(0);
        convMatrix.Matrix[0][0] = -2;
        convMatrix.Matrix[1][1] = 2;
        convMatrix.Factor = 1;
        convMatrix.Offset = 95;
        return ConvolutionMat.computeConvolution(image, convMatrix);
    }

}
```

## Noise Reduction and blur - GaussianBlur.java

This filter is slightly different to the previous ones designed using Convolution filter because the kernel size is 5x5 and the Factor value is 273. The main purpose of this filter is to reduce image noise and reduce minimal detail. The result image looks as if the user is viewing the original image through a translucent screen (Gimel'farb & Delmas, Part 3: Image Processing - 3.4. Moving Window Transform).

```
package edit.picture.packages;

import android.graphics.Bitmap;

public class GaussianBlur extends Filter {

    protected Bitmap filterImage(Bitmap image) {

        double[][] blurGauss = new double[][] {

                { 1, 4, 7, 4, 1 },
                { 4, 16, 26, 16, 4 },
                { 7, 26, 41, 26, 7 },
                { 4, 16, 26, 16, 4 },
                { 1, 4, 7, 4, 1 },

        };

        ConvolutionMat convMatrix = new ConvolutionMat(5);
        convMatrix.applyConfig(blurGauss);
        convMatrix.Factor = 273;
        convMatrix.Offset = 0;
        return ConvolutionMat.computeConvolution(image, convMatrix);


    }
}
```

## Sharpening using Laplacian kernel – Sharpen.java

Image sharpening in the field of image processing is categorized as spatial filtering is used to reveal fine detail in an image. I am one of Laplacian kernel which highlights regions of rapid intensity changes. The disadvantage to this filter is that if the image contains noise then it will highlight that too and the image will be worse off than before. So this filter is only to be used on images that are crisp in quality or with images that have been smoothened before applying this to reduce the noise intensity. The two types of kernels that I can use for my application are as follows (Fisher, Perkins, Ashley, & Wolfart, Laplacian/Laplacian of Gaussian, 2003):

| | | |
|---|---|---|
| 0 | −1 | 0 |
| −1 | 4 | −1 |
| 0 | −1 | 0 |

| | | |
|---|---|---|
| −1 | −1 | −1 |
| −1 | 8 | −1 |
| −1 | −1 | −1 |

I am using the kernel on the right for my application but the left one will do the job too. These kernels are approximating a second derivative measure on the image and are extremely sensitive to noise. Usually, it is recommend to use Gaussian blur beforehand as this reduces high frequency noise components and then apply the sharpen effect.

```
package edit.pictures.packages;

import android.graphics.Bitmap;

public class Sharpen extends Filter {

    protected Bitmap filterImage(Bitmap image) {
        double[][] SharpConfig = new double[][] {
                { -1, -1, -1 },
                { -1,  8, -1 },
                { -1, -1, -1 }
            };
            ConvolutionMat convMatrix = new ConvolutionMat(3);
            convMatrix.applyConfig(SharpConfig);
            convMatrix.Factor = 1;
            convMatrix.Offset = 0.0;
            return ConvolutionMat.computeConvolution(image, convMatrix);
        }

}
```

# Evaluation

After the completion of the project, I decided to evaluate the application using Nielson's Heuristics (NIELSEN, 1995) and decided to get my friends and family to be the evaluators of the project. The reason for Nielson's Heuristic evaluation is that it is widely known and practiced where the UI are often designed in a short space of time and I am on a budget so cannot get expert evaluators. The goal of this usability evaluation is to collect feedback in hopes to improve the application in the future. The feedback is summarised below:

## Visibility of system status

Picca keeps the user informed about which screen they are navigating to by the use of TextFields on the homescreen. Also the switch between the screens is made more obvious by the sound of the click of the button and change in button click states. For example, Homescreen has a gradient background whereas when the user clicks the Take Picture button, they are directed to a camera screen with a black background and the camera feature.

## Match between system and the real world

Picca consists of camera feature that has an interface of a real camera object in smart devices and has a reticle and a shutter effect. The user is able to capture an image when clicking the camera screen. The application mostly consisted of buttons and Image Views so it was really difficult to find matches between system and the real world.

## User control and freedom

Picca allows users to jump to higher levels when the user wishes to modify images. For example, the user is able to load images and then straight jump to the filter screen to apply images. The application provides a quick and safe way to exit the application and leave the unwanted state without having to go through an extended dialogue. The application also provides ways to undo and redo things such as load images again, undo modification if bitmaps using reset button and redo image capture and this allows the user has a sense of control and freedom when using this application.

## Consistency and standards

The buttons were quite consistent in design and click state colours. The features were only displayed on the screen if they worked. The layout slightly varied as per screen size of the device but otherwise the application was consistent and followed design standards.

## Error prevention

The application was error-prone was when the user is prompted for input and also when loading images. For example, for brightness increase, only integer value should be accepted or only allow numeric keys to appear on the keyboard. If the user loads the images straight after opening the application, it crashes with a Nullpointer so that needs to be fixed.

## Recognition rather than recall

This is application consists of a few screens only and barely any icons in the application. The only one that is used is the camera switch from back to front icon and that is a common image used for almost all devices so the user does not have to remember what the icon means.

## Flexibility and efficiency of use

Picca did not provide a lot of opportunities to tailor specific actions so that it could be performed faster but depended on the complexity of the algorithms for the filters. However, it did provide a way so that users can make decisions efficiently by having a RGB histogram feature. The user would need to tap the screen to capture an image and once tapped; the shutter effect is shown on the screen and could probably employ a quicker shutter effect to speed up the process. Apart from that the application is flexible and efficient.

## Aesthetic and minimalist design

In majority of cases, the application has only provided information where it is needed and not overwhelmed the user. For example, user input is only shown when a button is clicked and is only relevant to filter. The Grid View is also the same; it is only shown when the user requests it through a button and not taken over the whole homescreen.

## Help users recognize, diagnose, and recover from errors

The application does not help users recover from errors properly. For example, the user selects an image from a location and then displays the path on the screen and it is up to the user to diagnose whether the path is null or is not null.

## Help and documentation

The application does not really need lengthy help documentation on how to operate but still it provides more than enough information on the homescreen to operate this application properly. It is focused on the user's task, lists concrete steps to be carried out and isn’t too large.

The performance of the application was not the main goal for this project but I did still consider making it as efficient as possible while implementing the filters. I have made sure (where possible) to not use getPixel() to get each pixel value and split it into the ARGB format for the filter implementation. I have used the getPixels() and then used the following way to split it to the ARGB format.

```
A = (pixels[index] >> 24) & 0xFF;

R = (pixels[index] >> 16) & 0xFF;

G = (pixels[index] >> 8) & 0xFF;

B = pixels[index] & 0xFF;
```

This way is efficient and faster to use but it is not always possible to use as it all depends on the way the filter is implemented. For example, the Gaussian blur that I am using is from the Compsci 373 lecture notes and consist of 5x5 kernel but there is an efficient way to implement it using 3x3 kernel but the effect is not the same. Obviously, the 5x5 kernel is much slower to apply but the time varies depending on the device because of the hardware specs.

The image is saved in JPEG format after being captured and modified with the image quality of 100%. . This application is for image restoration and enhancement and it would not be appropriate to reduce the quality of the image because it defeats the purpose of restoration and enhancement. I decided to leave the quality as it is because the smart devices these days have quite a lot of internal memory and can also have external memory so the memory space was not a big focus of this project.

The completion of this application was a major achievement because I was also enrolled in other courses and the complexity of the application for me due to being a beginner in Android programming. The hardest part was to get the LoadImage buttons working because the communication between my Android device and my laptop was failing every time I clicked the LoadImage button. I later found out that the application was running fine but the problem was the USB connection and that is why the application was crashing. To make this work, I uploaded the .apk file to my Google drive and then downloaded it onto the device and it worked fine without crashing. There are things I would like to improve on and they are discussed in Conclusion and Future Work section.

# Conclusion and Future Work

Overall, Picca meets the project requirements and provides users with some of the common features of desktop image processing applications with image enhancement and restoration support. The application is able to successfully process images and also give the user an understanding of the RGB histogram display so that it provides decision support for filter selection. The application provides 20+ filters to user which can be applied in conjunction with each other to achieve an optimum result. The application works well with tablets and other Android devices as long as the screen size is over 5” and the API is 8 and over.

For the future, I would like to focus on the performance and optimizing the layout in multiple ways. The performance can be increased by using Android NDK which uses C/C++.The NDK allows users to implement a part of the application using native-code languages like C and C++. This means that I might be able to design my filters in C and this could slightly increase the performance. The reason for this is because programming languages like java have an additional processing overhead which affect the performance whereas C does not but the user would need to allocate and free memory to prevent memory leaks. Optimizing layout for larger screens should be considered for the future because as of now my application is only designed to work for screen size of 5” or more the same way but to give your users the best possible experience on each screen size configuration (especially for tablets) I should optimize layouts and other user interface features for specific screen size configuration. This means that the larger the screen size, the more space I have to add extra features, display new material or enhance the interaction. The easiest way to start is by changing the font size and making it dependable on the size of the screen as in my project the same font size is kept throughout the screen. For the large and xlarge screen, I should provide a custom layout for the application that suits that screen size configuration or I can provide layouts that are loaded based on the screen’s shortest dimension or minimum width or height. Also, the positioning of the user interface components should be easily accessible.

Next, I would direct my focus onto research and implementation of more filters to provide variety of filters because the key to attract more users of the application is to provide them with extended functionalities. To improve the human computer interaction, I think implementing a filter gallery would be an excellent strategy to attract more users of the application. The filter gallery will provide previews of many of the special effects of the filters as a thumbnail instead of just the filter name. The most important aspect of improvements to the application is to keep it unique and not generalize functionalities of other applications too much but still be able to achieve the goals of the application.

# Reference List

(n.d.). Retrieved from Spatial Analyst: http://spatial-analyst.net/ILWIS/htm/ilwisapp/stretch_algorithm.htm

*8.2 Convolution Matrix*. (n.d.). Retrieved from GIMP: http://docs.gimp.org/en/plug-in-convmatrix.html

Android Architecture – The Key Concepts of Android OS. (2012, February 17).

*Android Emulator*. (n.d.). Retrieved from Android Developers: http://developer.android.com/tools/help/emulator.html

Android, D. (n.d.). *Building Your First App*. Retrieved from http://developer.android.com/training/basics/firstapp/index.html

*Balsamiq Mock up Tool*. (n.d.). Retrieved from http://www.balsamiq.com/download

Barloso, K. (2012, February 22). *10 Excellent Photo Editing Android Apps*. Retrieved from http://android.appstorm.net/roundups/photography/excellent-photo-editing-android-apps/

Bourke, P. (1993, November). *A Beginners Guide to Bitmaps*. Retrieved from http://paulbourke.net/dataformats/bitmaps/

*Canvas and Drawables*. (n.d.). Retrieved from Android Developer: http://developer.android.com/guide/topics/graphics/2d-graphics.html

Castleman, K. R. (1995). *Digital Image Processing (2nd Edition).* New Jersey: Prentice Hall.

Chanda, B., & Dutta, D. (2005). *Digital image processing and analysis.* New Delhi: Prentice Hall of India.

Cheng, H., Huang, Z., & Kumimoto, M. (2006). *Final Project Report – Image Processing Techniques.*

Diniy, G., Martinelliz, F., Matteucciz, I., & Petrocchiz, M. *A Multi-Criteria-based Evaluation.* Pisa: Dipartimento di Ingegneria dell' Informazione.

Efford, N. (2000). *Digital Image Processing.* Delhi: Pearson Education Asia.

Fisher, R., Perkins, S., Ashley, W., & Wolfart, E. (2003). *Gaussian Smoothing*. Retrieved from http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

Fisher, R., Perkins, S., Ashley, W., & Wolfart, E. (2003). *Laplacian/Laplacian of Gaussian*. Retrieved from http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

*Get the Android SDK*. (n.d.). Retrieved from Android Developers: http://developer.android.com/sdk/index.html

Gimel'farb, G., & Delmas, P. (n.d.). *Part 3: Image Processing - 3.2. Image Filtering and Segmentation*. Retrieved from http://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-02.pdf

Gimel'farb, G., & Delmas, P. (n.d.). *Part 3: Image Processing - 3.4. Moving Window Transform*.

Gimel'farb, G., & Delmas, P. (n.d.). *Part 3: Image Processing: 3.1. Digital Images and Intensity Histograms*. Retrieved from http://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-01.pdf

Gonzalez, R., & Woods, R. (2007). *Digital Image Processing (3rd Edition).* Prentice Hall.

Gupta, S., & Abhijit, S. *Image Processing Project Report - RGB IMAGE TO PENCIL SKETCH FILTER FOR MONUMENTS.*

Hoffmann, S. (2006, May 2). *A PRACTICAL GUIDE TO INTERPRETING RGB HISTOGRAMS*. Retrieved from http://www.sphoto.com/techinfo/histograms/histograms.htm

Houston, P. (2011, June 22). *Image Processing – Engraving Effect*. Retrieved from http://xjaphx.wordpress.com/?s=engrave

*Hue, Value, Saturation*. (n.d.). Retrieved from http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/

*Introduction to Image Processing*. (n.d.). Retrieved from http://www.spacetelescope.org/static/projects/fits_liberator/image_processing.pdf

Jain, A. (2002). *Fundamentals of digital image processing.* New Delhi: Prentice Hall of India.

*Java Image Filters*. (n.d.). Retrieved from JH Labs: http://www.jhlabs.com/ip/filters/

*Layout*. (n.d.). Retrieved from Android Developers: http://developer.android.com/guide/topics/ui/declaring-layout.html

Lucas, J. *The SalsaJ software.* Paris: Université Pierre et Marie Curie.

Market, A. A. (n.d.). *Android Architecture – The Key Concepts of Android OS*. Retrieved from http://www.android-app-market.com/android-architecture.html

NIELSEN, J. (1995, January 1). 10 Usability Heuristics for User Interface Design. pp. http://www.nngroup.com/articles/ten-usability-heuristics/.

Pratt, W. (2007). *Digital Image Processing: PIKS Scientific Inside.* Wiley-Interscience.

Rhody, H. (n.d.). *Simple Gradient Calculation*. Retrieved from http://www.cis.rit.edu/people/faculty/rhody/EdgeDetection.htm

Sharma, A. (n.d.). *Using Surface View for Android*. Retrieved from http://www.mindfiresolutions.com/Using-Surface-View-for-Android-1659.php

Tamada, R. (2011, July 27). *Android Layouts: Linear Layout, Relative Layout and Table Layout*. Retrieved from Android Hive: http://www.androidhive.info/2011/07/android-layouts-linear-layout-relative-layout-and-table-layout/

Vandevenne, L. (2004). *Lode's Computer Graphics Tutorial - Image Filtering*. Retrieved from http://lodev.org/cgtutor/filtering.html

*ViewFinderEE368*. (n.d.). Retrieved from Stanford Edu: http://www.stanford.edu/class/ee368/Android/ViewfinderEE368/

*Views*. (n.d.). Retrieved from Android Developer: http://developer.android.com/reference/android/view/View.html