EE610 - Assignment 1
Basic Image Editor

Kruthagnya Myana - 18D070015

Aug 29, 2021

Contents

Abstract

Basic Image Editor is a very basic Image processing interface created based on python to perform some basic image processing operations like Histogram Equalisation, Gamma Correction, Blurring etc. The Interface for Basic Image Editor is made using tkinter library from python. It uses PIL library to load, save and display Images on the Interface. Libraries math,numpy and cv2(openCv) are used for implementing the image processing techniques.

1 Introduction

The main objective of the assignment is to implement a GUI for Image processing techniques both Grayscale or Color images.For GUI development Tkinter library is used, and to load, save and display PIL Image library is used. Color images are transfomed into HSV domain and all the transformations are done on channel V. Apart from the RGB to HSV transformation (and vice-versa) which is done using the OpenCV library, all the image processing operations and transformations are defined from scratch.

2 GUI Design

Initially after loading the Interface, and loading an image using the open button Fig.1 is what we see The Interface contains the following features:

- Open Button: To access and load any image files on the disk
- Quit Button: To Quit and close the whole interface
- Original Button: After performing few Image transformations, we can access the original image by clicking this button
- Save Button with Input Box: Saves the current displaying image with the name given into the input entry box in 'jpg' format
- Histogram Equalization Button: To apply the Histogram Equalization Transform on the current displaying image
- Logarithm Button: To apply Logarithmic Transform on the current displaying image
- Gamma Transform Button with Input Box: To apply Gamma correction with the gamma value, that can be given into the input box right above the "Gamma Transform" Button
- Negative Button: To apply Nehative Intensity Transform on the current displaying image
- Blur Image Button with Slider: To blur an image, with the amount of blurness that can be controlled by the slider right above the "Gamma Transform" Button
- Sharpen Image Button with Slider: To sharpen an image, with the amount of sharpness that can be controlled by the slider right above the "Gamma Transform" Button

Figure 1: Interface

Important Code used in defining the above buttons and placing them on the interface as a proper grid:

```
#Buttons
button_open = Button(root,text = "Open", command = Open)
button_save = Button(root, text = "Save", command = Save)
button_undo = Button(root, text = "Undo", command = Undo)
button_original = Button(root, text = "Original", command = Original)
button_equalize = Button(root, text = "Histogram \n Equalization", command = Hist)
button_log = Button(root, text = "Logarithm", command = Log10)
button_gamma = Button(root, text = "Gamma Transform", command = G_Transform)
button_negative = Button(root, text = "Negative", command = N_Transform)
button_blur = Button(root, text = "Blur Image", command = Blurness)
button_sharp = Button(root, text = "Sharpen Image", command = Sharpness)
```

```
#Gamma Input
g = Entry(root) #for entering gamma value
g.grid(row = 4, column = 0)
#Save File Name Input
s = Entry(root)
s.grid(row = 4, column = 5)
#Sliders
blur = Scale(root, from_ = 3, to = 25, resolution = 2, orient = HORIZONTAL)
blur.grid(row = 4, column = 1)
sharp = Scale(root, from_ = 1, to = 25, orient = HORIZONTAL)
sharp.grid(row = 4, column = 3)
```

```
#Placing of Buttons
button_open.grid(row = 0, column = 0)
button_undo.grid(row = 0, column = 3)
button_original.grid(row = 0, column = 4)

button_equalize.grid(row = 1, column = 5)
button_log.grid(row = 2, column = 5)
button_negative.grid(row = 3, column = 5)

button_gamma.grid(row = 5, column = 0)
button_blur.grid(row = 5, column = 1)
button_sharp.grid(row = 5, column = 3)
button_save.grid(row = 5, column = 5)
```

Some important blocks of code:

Code Block used for 2D Convolution of an image with a kernel

```
def Convolution(image, kernel, padding = 1):
    #Flipping the kernel (so we can take sum products directly)
    kernel = np.flipud(np.fliplr(kernel))
    x_k = kernel.shape[0]
    y_k = kernel.shape[1]
    p = int(padding)
    length = len(image.shape)

    if length == 2:
        typ = "gray"
        img_gray = image.copy()

        #Padding the Image
        padded_img = np.zeros((image.shape[0] + 2*p, image.shape[1] + 2*p))
        padded_img[1*p:-1*p,1*p:-1*p] = img_gray

        #Convolved image
        conv_img = np.zeros_like(image)

        #Computing convolution in a loop
        for y in range(image.shape[1]):
            for x in range(image.shape[0]):
                conv_img[x,y] = (kernel * padded_img[x: x+x_k, y: y+y_k]).sum()
                #Sum Products
```

Code Block used for Resizing an image maintaining aspect ratio

```
def Resize(image):
    np_img = np.array(image)
    #Resizing the image by maintaining the aspect ratio r
    r = 600/np_img.shape[0]
    M = int(np_img.shape[1]*r)
    if M >= 1200:
        M = 1200
    else:
        M = M
    dim = (M,600)
    stretch_near = cv2.resize(np_img, dim,interpolation = cv2.INTER_AREA)
    pil_img = Image.fromarray(stretch_near)
```

Code Block defining the commands for the Open, Undo and Save buttons("Original" Button has code similar to Undo)

```
#Definitions for buttons
#temp - current displaying image
#temp1 - temporary variable used to store prev image for Undo
#original - Original image accessed when clicked Original button
def Open():
    global org_img
    global org_img_label
    global original
    global temp
    org_filename = filedialog.askopenfilename(initialdir = " ", title = "Open", filetypes = (("jpg files","*.jpg"),("all files","*.*")))
    org_img = Image.open(org_filename)
    org_img = Resize(org_img)
    original = org_img.copy()
    temp = org_img.copy()
    temp1 = org_img.copy()
    org_img = ImageTk.PhotoImage(org_img)
    org_img_label = Label(root, image = org_img)
    org_img_label.grid(row = 1, column =0, columnspan = 5, rowspan = 3)

def Save():
    global temp
    #Saves files in jpeg format
    file_name = s.get()
    if temp.mode != "RGB":
        temp = temp.convert("RGB")
    temp.save(file_name+ '.jpeg')

def Undo():
    global temp2
    global temp
    global temp2_label
    temp = temp1
    temp2 = ImageTk.PhotoImage(temp)
    temp2_label = Label(root, image = temp2)
    temp2_label.grid(row = 1, column =0, columnspan = 5, rowspan = 3)
```

Code Block defining the command for Histogram Equalization Button (Logarithm and Negative Buttons have the similar code)

```
def Hist():
    global h_temp_label
    global h_temp
    global temp1
    global temp
    temp1 = temp #Used to acces while doing Undo
    temp = Histogram(temp1)
    h_temp = ImageTk.PhotoImage(temp)
    h_temp_label = Label(root, image = h_temp)
    h_temp_label.grid(row = 1, column =0, columnspan = 5, rowspan = 3)
```

Code Block defining the command for Gamma Transform Button, by taking gamma value from input entry box(Blur and Sharp Buttons have the similar code)

```
def G_Transform():
    global d
    global g_temp_label
    global g_temp
    global temp1
    global temp
    d = float(g.get()) #takes gamma value from the entry box above gamma button
    temp1 = temp
    temp = Gamma(temp1,d)
    g_temp = ImageTk.PhotoImage(temp)
    g_temp_label = Label(root, image = g_temp)
    g_temp_label.grid(row = 1, column =0, columnspan = 5, rowspan = 3)
```

3 Image Processing Operations

List of Image Processing Operations available on this Basic Image Editor Interface are

- Histogram Equalization

- Gamma Correction
- Logarithm Transform
- Blur Filter
- Sharpening Filter
- Negative Transform

3.1 Histogram Equalization

Histogram Equalization is Intensity Transformation technique that can be used to improve contrast in images.To enhance the image's contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image.
Consider a discrete grayscale image f(x,y) and let n_k be the number of pixels for the intensity level r_k. The probability of an occurrence of a pixel of level r_k in the image with total (MxN) pixels is

$$p_r(r_k) = \frac{n_k}{M.N}$$

L being the total number of gray levels in the image (typically 256), M.N being the total number of pixels in the image, and $p_r(r_k)$ being in fact the image's histogram for pixel value r_k, normalized to [0,1].The histogram equalized image s will be defined by

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

Main Code Block used for Histogram Equalization

```
M = img_hsv_array.shape[0]
N = img_hsv_array.shape[1] #MxN number of pixels
p_r = n_k/(M*N) #probability of an occurrence of a pixel of level r_k
cump_r = np.cumsum(p_r) #Cumulative sum
s_k = 255*cump_r #transformation for equalization of intensities
s_k = np.rint(s_k) #Rounding off to nearesr integers

Iter = dict(zip(r_k,s_k)) #mapping r_k and s_k
def replace(r):
    return Iter[r] #replacing original r_k with s_k
replace_v = np.vectorize(replace) #Replacing via vectorisation
```

3.2 Gamma Correction

Gamma correction is simply a power law transform.a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems. Gamma correction is, defined by the following power-law expression:

$$s = T(r)$$

$$s = c.r^{\gamma}$$

where,
r = input pixel value,
c = scaling constant,
s = output pixel value
The value of 'c' is chosen such that we get the maximum output value corresponding to the bit size used. Input intensity levels(r_k) vary from (0-255) and for output intensity levels(s_k) to be

in the same range we set $c = 255/(255)^{\gamma}$
A gamma value $\gamma < 1$ is called gamma compression and maps a narrow range of dark input values into a wider range of output values, with the opposite being true for higher input values and conversely a gamma value $\gamma > 1$ is called gamma expansion/
The human perception of brightness follows an approximate power function, but our camera does not work like this. Unlike human perception, camera follows a linear relationship. This means that if light falling on the camera is increased by 2 times, the output will also increase 2 folds. When we display the image, the results in images that are darker than intended.To correct this, we apply gamma correction to the input signal

Main Code Block used for Gamma Transform on an RGB image,(we convert RGB to HSV and make changes in the V channel and convert back it to RGB)

```
    typ = "RGB"
    img_hsv = cv2.cvtColor(img_array, cv2.COLOR_RGB2HSV)#convering RGB image to HSV format
    img_hsv_array = np.array(img_hsv)
    for i in range(0,img_hsv_array.shape[0]):
        for j in range(0,img_hsv_array.shape[1]):
            p_ij = img_hsv_array[i,j,2] #V value of the pixel (i,j)
            s_ij = c*(p_ij**g) #gamma transform s = c*(r^g)
            img_hsv_array[i,j,2] = s_ij
    final_image_array = cv2.cvtColor(img_hsv_array,cv2.COLOR_HSV2RGB)#Converting hsv image back to RGB

pil_img = Image.fromarray(final_image_array) #Converting cv2 image to PIL image to display in the gui
```

3.3 Logarithm Transform

Log transformation of an image means replacing all pixel values, present in the image, with its logarithmic values. Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values.The formula for applying log transformation in an image is,

$$s = c.log(1 + r)$$

where,
c = scaling constant,
r = input pixel value,
s = output pixel value.
The value of 'c' is chosen such that we get the maximum output value corresponding to the bit size used. Input intensity levels(r_k) vary from (0-255) and for output intensity levels(s_k) to be in the same range we set $c = 255/(log(1 + 255))$
We are adding '1' to each pixel value at the time of log transformation so that if any pixel value is '0', it will become '1' and its log value will be '0'.
Main Code Block used for Logarithmic Transform on an RGB image,(we convert RGB to HSV and make changes in the V channel and convert back it to RGB)

```
    typ = "RGB"
    img_hsv = cv2.cvtColor(img_array, cv2.COLOR_RGB2HSV)
    img_hsv_array = np.array(img_hsv)

    def log_r(i,j):
        r = img_hsv_array[i,j,2]
        l = math.log10(1+r)
        return c*l
    for i in range(0,img_hsv_array.shape[0]):
        for j in range(0,img_hsv_array.shape[1]):
            img_hsv_array[i,j,2] = log_r(i,j)
    img_hsv_array = cv2.cvtColor(img_hsv_array,cv2.COLOR_HSV2RGB)

final_image_array = img_hsv_array
pil_img = Image.fromarray(final_image_array)

return pil_img
```

3.4 Blur Filter

In blurring, we simple blur an image using the blur filter.The blur, or smoothing, of an image removes "outlier" pixels that may be noise in the image. Blurring is an example of applying a low-pass filter to an image.Blurring can be achieved by many ways. The common type of filters that are used to perform blurring are

- Mean Filter
- Weighted Average Filter
- Gaussian Filter

Out of these, for the interface I have used the mean filter, let us take a look at the mathematical formulae involved

Mean Filter

Mean filter is also known as Box filter and average filter. A mean filter has the following properties.

- It must be odd ordered
- The sum of all elements should be 1
- All the elements should be same

If we follow this, for a 3x3 filter we get,

$$w = \frac{1}{9}.\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

so for general b, we have bxb sized matrix with each element equal to $\frac{1}{b^2}$. As the number b increases the amont of blurring increases. so the extent of blurring can be controlled by this element, b
Now we get the blurred image output,g(x,y) by convolving the kernel w with the input image f(x,y) for every x,y in the MxN sized image.

$$g(x,y) = w(x,y) \star f(x,y)$$

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t).f(x-s,y-t)$$

where, a,b belongs to the set {-1,0,1}
Main Code Block used for Blurring an Image

```
def Blur(img,b):
    pd = (b-1)/2
    c = b**2
    k = np.ones((b,b),np.float32)/c
    image_array = np.array(img)
    conv_img = Convolution(image_array,k,pd)
    pil_img = Image.fromarray(conv_img)

    return pil_img
```

3.5 Sharpening Filter

Sharpening is used to sharpen and highlight the edges and make the transitioning of features and details more significant.Sharpening is used to find the difference by the neighborhood and enhancing them even more. It is a process of differentiation. There are different types of Sharpening Filters. for the interface I have used the Unsharp Masking. In this we do the following

- Blur the image
- Mask = Original Image - Blurred Image. This output now contains now contains most of the high frequency components that were blocked by the blurring filter.
- Adding the mask to original image will enhance the high frequency components.

$$m(x,y) = f(x,y) - f_b(x,y)$$
$$g(x,y) = f(x,y) + m(x,y)$$

where,
m(x,y) - Mask
f(x,y) - Original Image
f_b(x,y) - Blurred Image
g(x,y) - Sharpened Image
By controlling the amount of blurrness in the first step, the amount of sharpening can be controlled. we have already seen how we control the extent of blurrness in the previous section Main Code Block used for Sharpening an Image

```
    typ = "RGB"
    hsv_img_array = cv2.cvtColor(img_array,cv2.COLOR_RGB2HSV)
    channel_v = hsv_img_array[:,:,2]
    blur_v = Convolution(channel_v,k,padding = (b-1)/2)
    mask = channel_v - blur_v
    sharp = mask + channel_v
    hsv_img_array[:,:,2] = sharp
    hsv_img_array[:,:,2] = np.clip(hsv_img_array[:,:,2],0,255)
    shp_img_array = cv2.cvtColor(hsv_img_array,cv2.COLOR_HSV2RGB)
pil_img = Image.fromarray(shp_img_array)
```

3.6 Negative Transform

Negative Transform is a linear intensity transformation technique.In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image. The following transition has been done.

$$s = (L–1)–r$$

where,
r = input pixel value,
L = Number of intensity levels,
s = output pixel value
We deal with 8bpp images so, L = 256 ;i.e (0-255) levels Main Code Block used for Negative Transform of an Image

```
    typ = "RGB"
    img_array = np.array(img)
    for i in range(0,img_array.shape[0]):
        for j in range(0,img_array.shape[1]):
            for a in range(0,img_array.shape[2]):
                p_ij = img_array[i,j,a]
                s_ij = L-1-p_ij
                img_array[i,j,a] = s_ij
    final_image_array = img_array
pil_img = Image.fromarray(final_image_array)
```

4 Experiments and Results

Demonstration of above stated Image Processing Techniques
(Note: All the Images included in the report after the process are saved using the save button on the interface)

4.1 Histogram Equalization

Let us look at the result when we equalize the histogram of intensities of an image, both grayscale and RGB images

On Gray Scale Image

Original Image

After Histogram Equalization

Reason for choosing these images: Bad spread of intensities, dark background for Color image which got brighter on equalising We can see spreading out of most of the intensity values, i.e. stretching out of the intensity range of the image. we see areas of lower local contrast have gained a higher contrast.

On RGB/Color Images

Original Image

After Histogram Equalization

4.2 Gamma Correction

On Gray Scale Image

for $\gamma = 0.5$:

Original Image

After Gamma Transform $\gamma = 0.5$

for $\gamma = 2$:

Original Image

After Gamma Transform $\gamma = 2$

Reason for choosing the above grayscale image: we can clearly see the difference and the concept of gamma encoding through this image.We can clearly see some hidden information after gamma transform($\gamma = 0.5$)and, the vice versa i.e encoding image can be done by $\gamma > 1$ as shown in above comparisons.

On RGB/Color Images

for $\gamma = 0.4$:

Original Image

After Gamma Transform $\gamma = 0.4$

for $\gamma = 3$:

Original Image

After Gamma Transform $\gamma = 3$

Gamma correction controls the overall brightness of an image. Images which are not properly corrected can look either bleached out, or too dark.Varying the amount of gamma correction changes not only the brightness, but also the ratios of red to green to blue.

4.3 Logarithmic Transform

Reason for choosing these images: These images have more darker intensities, which transform into higher contrasts after transform. We clearly see that the darker intensities contrast change and also the image brightness increased

On Gray Scale Image

Original Image

After Logarithmic Transformation

On RGB/Color Images

Original Image

After Logarithmic Transformation

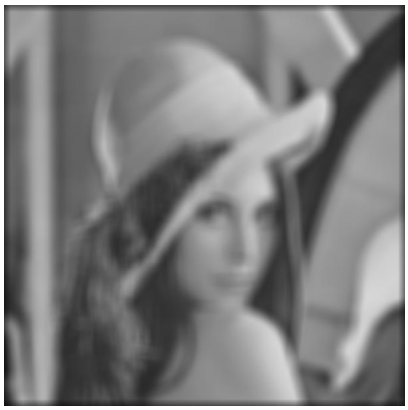
4.4 Blur Filter

Choose these images to observe the blur effect on human faces

On Gray scale Image

Original Image

Little amount of Blurring

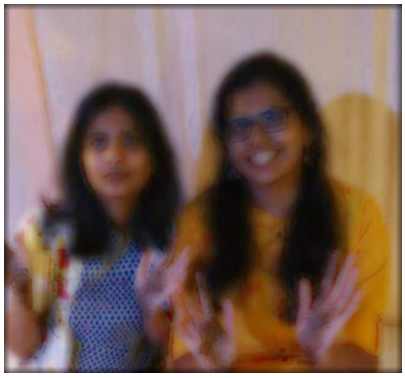


Large amount of Blurring

On RGB/Color Images

Original Image

Little amount of Blurring



Large amount of Blurring

4.5 Sharpening Filter

Lets us try using sharpening filter on a blurred image, taking the little amount of blurred image from previous section

On Gray scale Image

Original Image

Little amount of Sharpening

Large amount of Sharpening

On RGB/Color Images

Original Image

Little amount of Sharpening

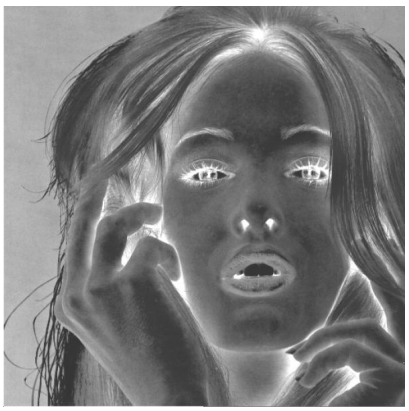
Large amount of Sharpening

We can see a mild removal of noise when sharpened little, but a significant change can be seen after large amount of sharpening

4.6 Negative Intensity Transform

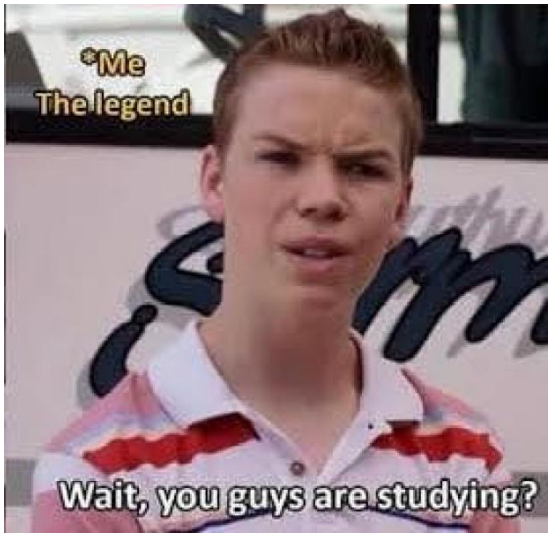
On Gray Scale Image

Original Image



Negative of the Image

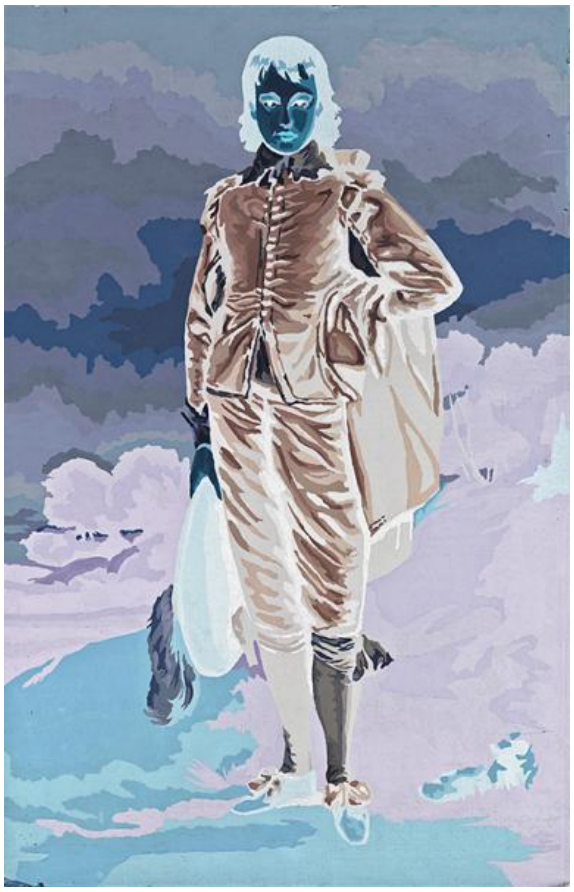
On RGB/Color Image



Original Image

Negative of the Image

Taking a color negative picture(Ref [5]) and applying negative transform to the image we retrieve the original color image



Original Image

Negative of the Image

Taking an image through multiple steps to enhance it

Original Image

Step1: $\gamma = 1.6$

Step2: logarithm

Step3: Sharp (Scale = 3)

Step4: $\gamma = 4$

Step5: $\gamma = 1.6$

Step6: Blur(Scale = 5)
Final Image

Original
Image

5 Conclusion

My Observations and Challanges:
For increasing brightness of an image gamma transform works better compared to logarithmic transform, because of the control over the gamma value and also the RGB contranst also varies in gamma transformation.Same goes with gamma correction vs histogram equalization. gamma correction gives you more control; you would use gamma correction when you want to darken an image or brighten an image. Whereas equalisation doesn't give you much control, it just stretches the histogram to make better use of the colour space.
Challanges I have faced while working on the assignment:

- Performing 2D convolution without using the filter2D inbuilt function. later I found a github repository(Ref [4]) which helped me write the convolution from scratch
- I was using PIL to load, save images and openCv to work on the intensities to transform them. openCv load image in GBR and PIL does it in RGB, so not losing contrast was a challange
- Clipping the intensities that go out of range in every transform, mainly while sharpening the image
- Now I have used vectorization in Histogram Equalization and used for loops in other transforms because of the time constraint. If I had more time i would use vectorization for all the rest transforms
- If I had more time i would try Filtering in Fourier domain

References

[1] Playlist referred to write code using openCv library: **Image Processing using openCv**

[2] Video referred to write code using tkinter library for building gui: **Tkinter Course**

[3] Code references used to check whether an image is color or gray scale: **Gray vs RGB**

[4] Code referred to write function for 2D convolution from scratch: **2D Convolution**

[5] **Blue Boy Negative Image**