HACETTEPE UNIVERSITY

Department of Computer Engineering

BBM415 Fundamentals of Image Processing Lab

Course Project

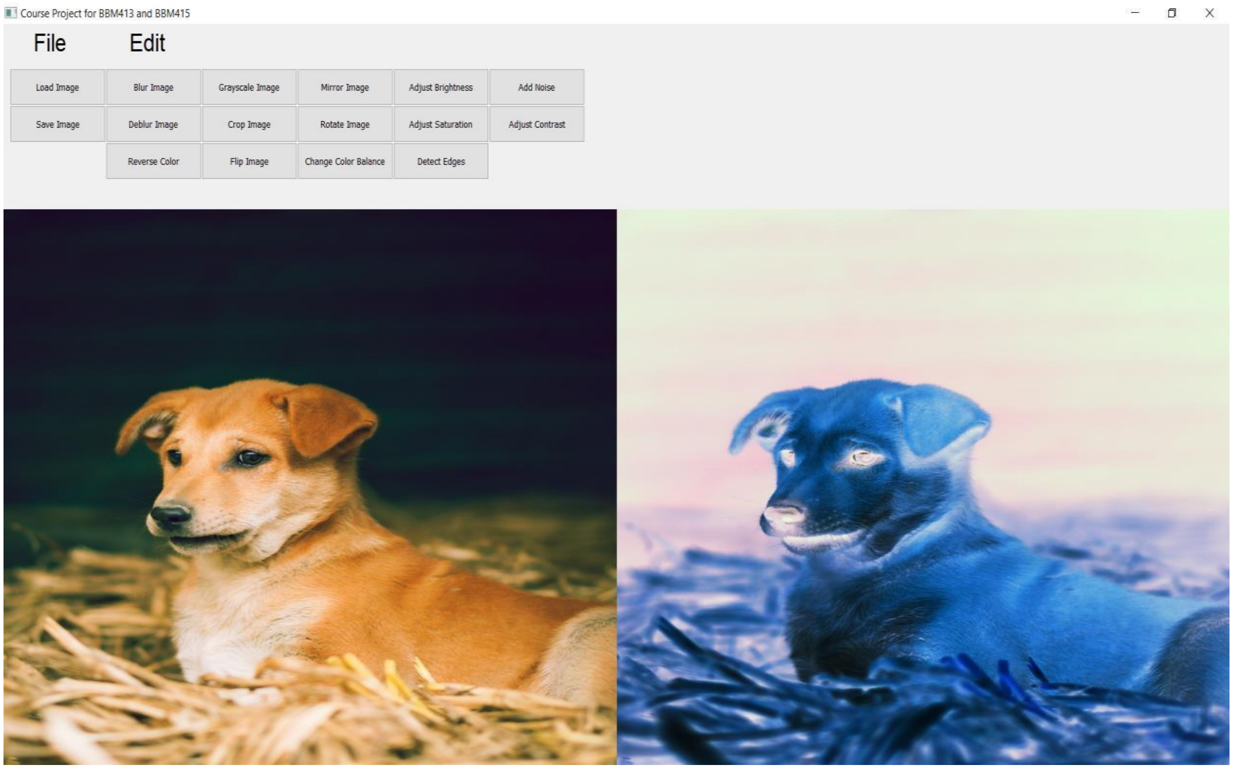
Fall 2021-2022

Subject: Building a simple image editor

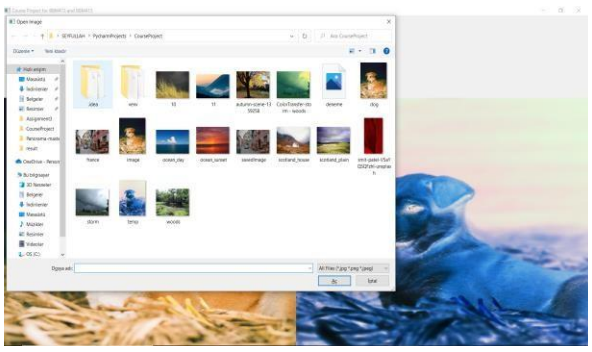
In this assignment, we have built a simple image editor by using pyqt5 for user interface and relevant image processing libraries. We will now explain gui and how we implemented the functions with example input and output files.

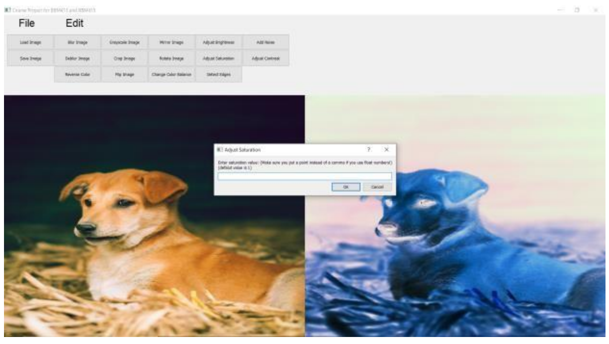
We have simply divided main screen of our gui into 4 parts. We place buttons to top left part. Top right part is about showing warning messages. For example, if you enter invalid inputs you can see warning message there. You can see original image in bottom left part and manipulated image in bottom right part.

Also, we try to fit the images in screen according to different device dimensions for responsive design.

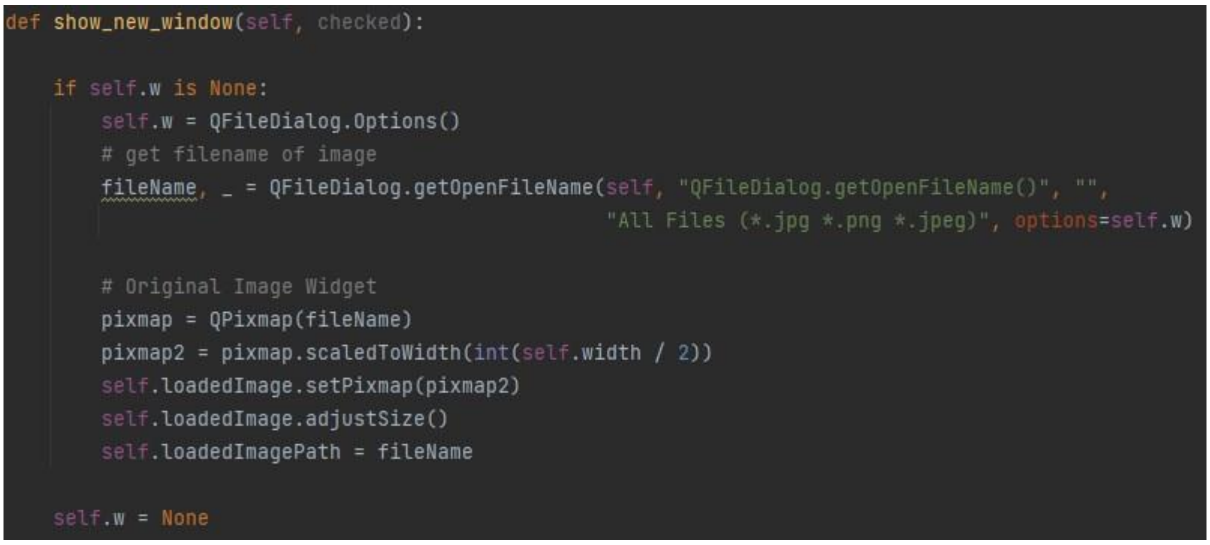


For the other inputs like selecting image or another inputs of image processing functions, we created another window. As seen as below





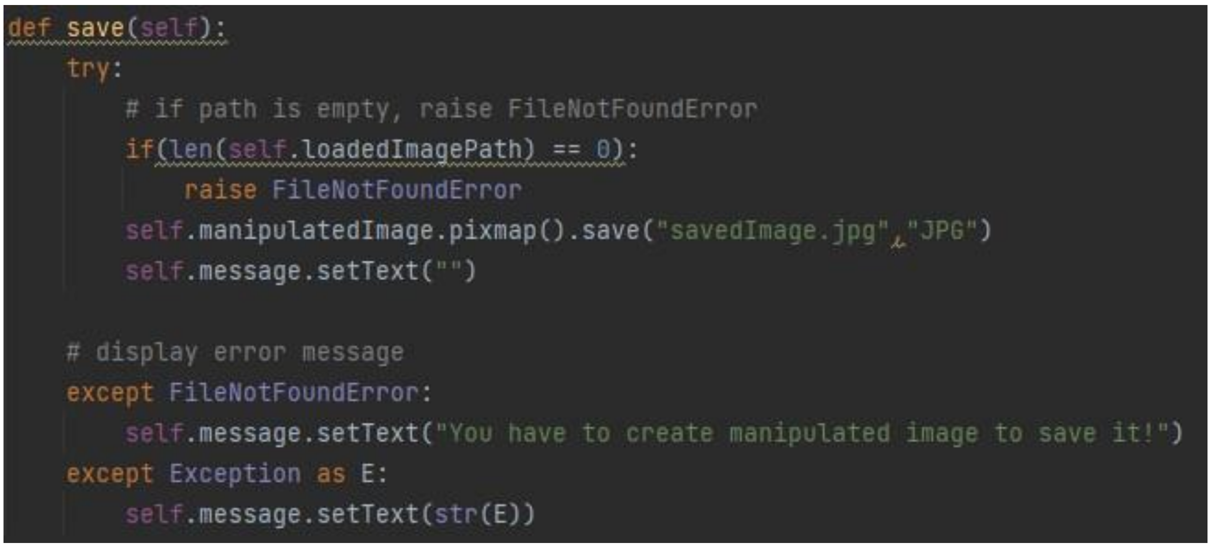
Load and Show Method



```python
def show_new_window(self, checked):

    if self.w is None:
        self.w = QFileDialog.Options()
        # get filename of image
        fileName, _ = QFileDialog.getOpenFileName(self, "QFileDialog.getOpenFileName()", "",
                                                  "All Files (*.jpg *.png *.jpeg)", options=self.w)

        # Original Image Widget
        pixmap = QPixmap(fileName)
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))
        self.loadedImage.setPixmap(pixmap2)
        self.loadedImage.adjustSize()
        self.loadedImagePath = fileName

    self.w = None
```

As you can see we get input image files only with jpg, png and jpeg extensions. Then we show on interface with adjusted size.

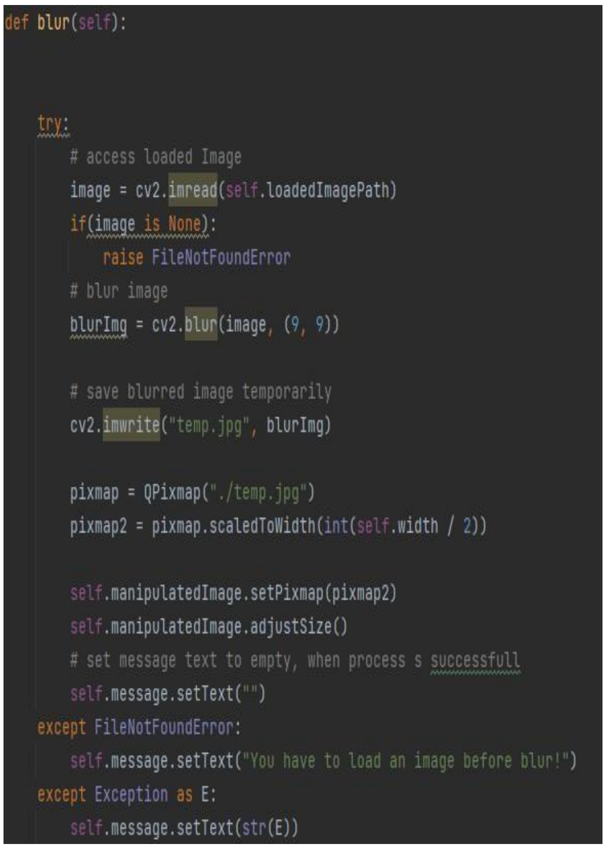
Save Method



```python
def save(self):
    try:
        # if path is empty, raise FileNotFoundError
        if(len(self.loadedImagePath) == 0):
            raise FileNotFoundError
        self.manipulatedImage.pixmap().save("savedImage.jpg","JPG")
        self.message.setText("")

    # display error message
    except FileNotFoundError:
        self.message.setText("You have to create manipulated image to save it!")
    except Exception as E:
        self.message.setText(str(E))
```

After we load the input image file, we can save after some changes like blur, grayscale etc. with this method.

Blur Image Method



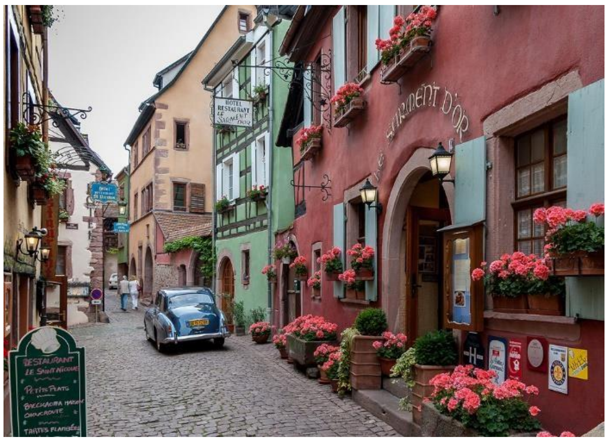
```
def blur(self):


    try:
        # access loaded Image
        image = cv2.imread(self.loadedImagePath)
        if(image is None):
            raise FileNotFoundError
        # blur image
        blurImg = cv2.blur(image, (9, 9))

        # save blurred image temporarily
        cv2.imwrite("temp.jpg", blurImg)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        # set message text to empty, when process s successfull
        self.message.setText("")
    except FileNotFoundError:
        self.message.setText("You have to load an image before blur!")
    except Exception as E:
        self.message.setText(str(E))
```

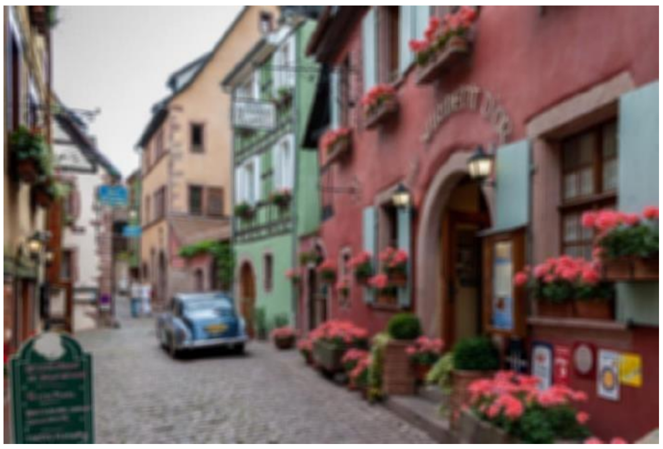
As you can see we read input file with opencv, if there is no file, it raises an error. Then we blur the image with our kernel which has size 9,9 with opencv. After that, we output the blurred image which adjusted size for interface.

And here is the example input and output below.

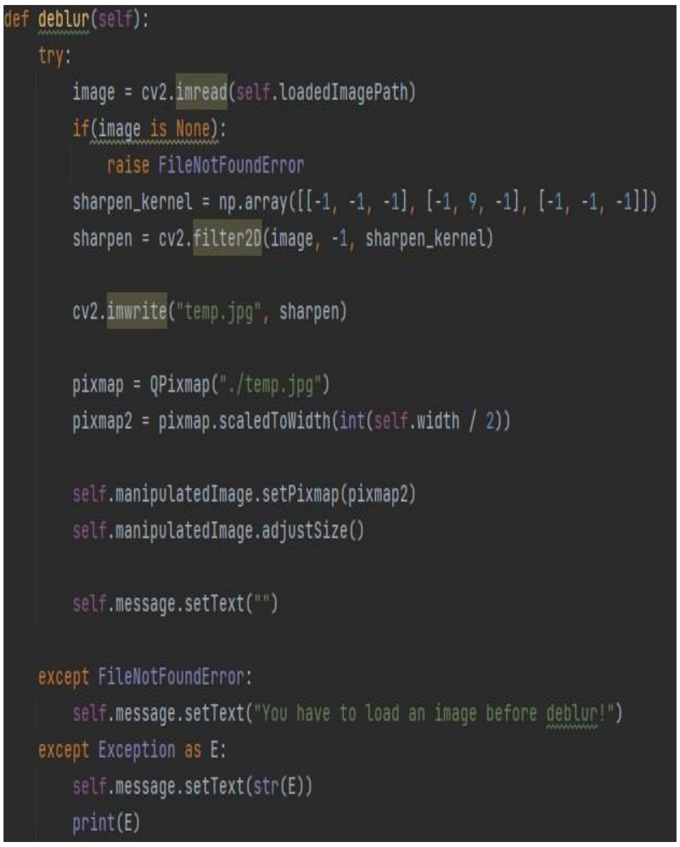
Input Image

Output Image





Deblur Image Method



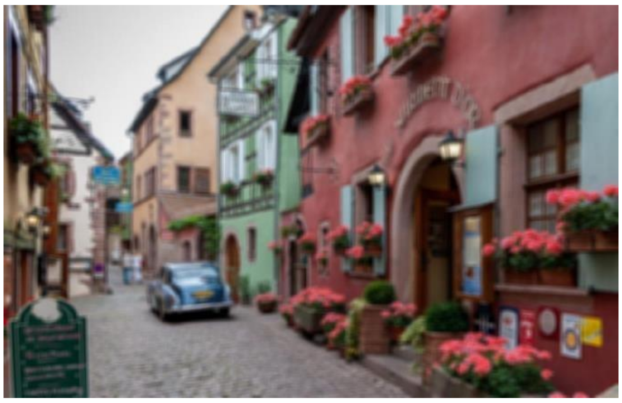
```
def deblur(self):
    try:
        image = cv2.imread(self.loadedImagePath)
        if(image is None):
            raise FileNotFoundError
        sharpen_kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
        sharpen = cv2.filter2D(image, -1, sharpen_kernel)

        cv2.imwrite("temp.jpg", sharpen)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()

        self.message.setText("")

    except FileNotFoundError:
        self.message.setText("You have to load an image before deblur!")
    except Exception as E:
        self.message.setText(str(E))
        print(E)
```

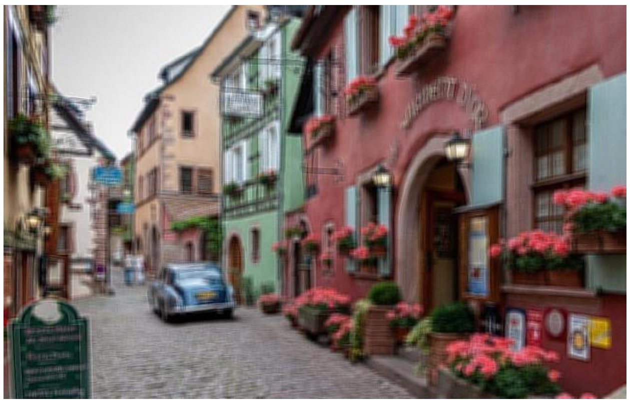
As you can see we read input file with opencv, if there is no such file, it gives an error. Then we convolve our blurred image with our specific sharpen kernel. After that, we output the sharpen image which adjusted size for interface.

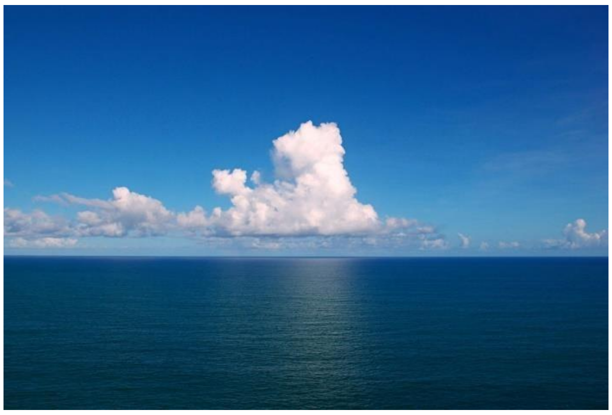
And here is the example input and output below.

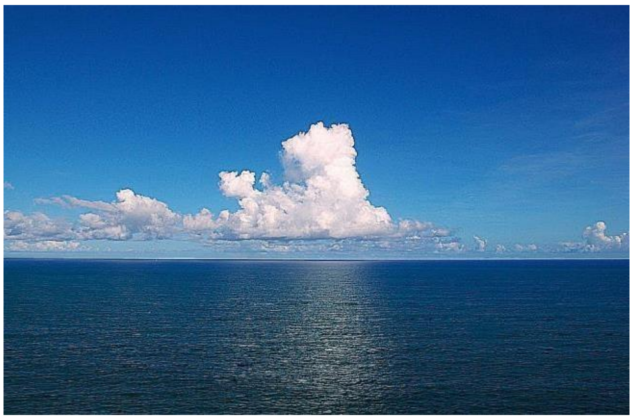
Input Image

Output Image

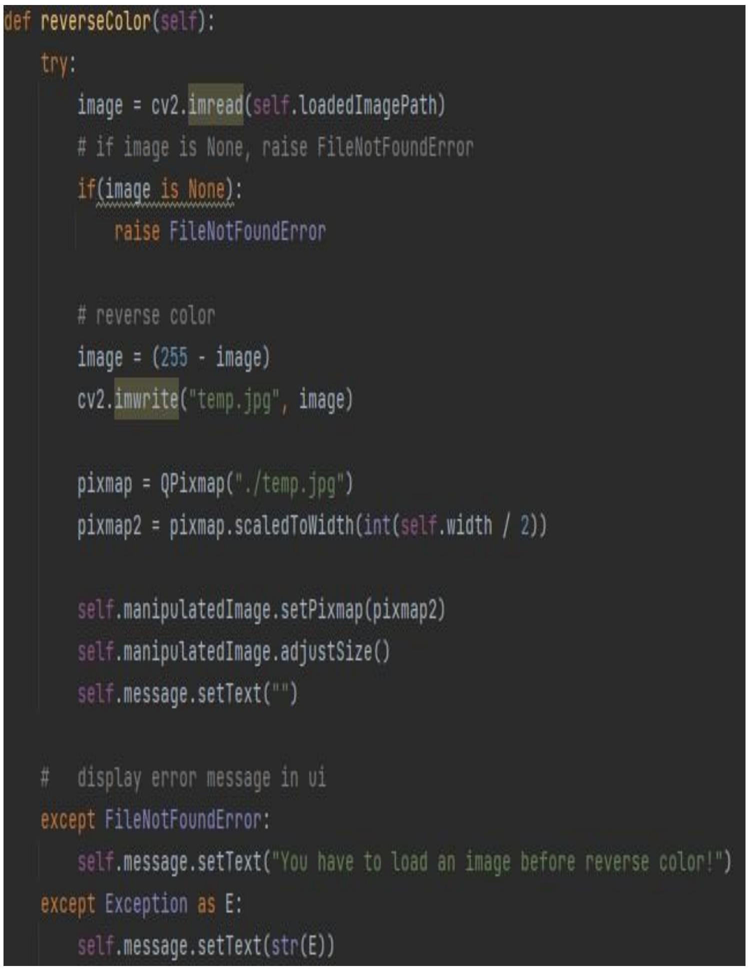








Reverse Image Method

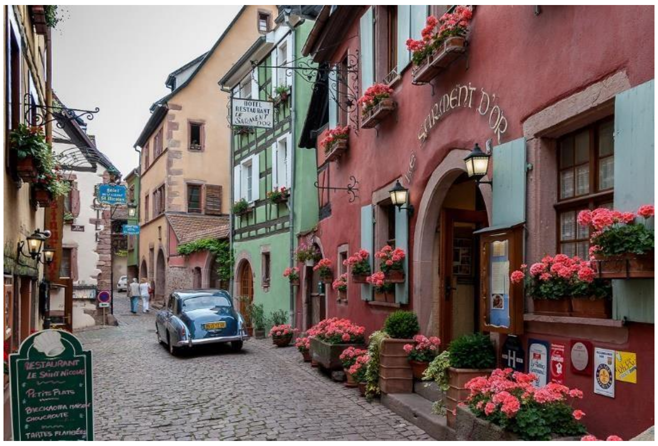


```python
def reverseColor(self):
    try:
        image = cv2.imread(self.loadedImagePath)
        # if image is None, raise FileNotFoundError
        if(image is None):
            raise FileNotFoundError

        # reverse color
        image = (255 - image)
        cv2.imwrite("temp.jpg", image)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        self.message.setText("")

    #   display error message in ui
    except FileNotFoundError:
        self.message.setText("You have to load an image before reverse color!")
    except Exception as E:
        self.message.setText(str(E))
```

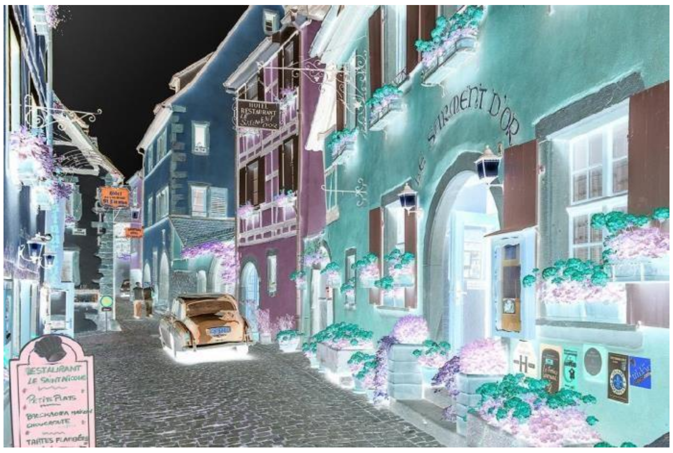
As you can see we read input file with opencv, if there is no such file, it raises an error. Then, we subtract the pixel values of the input image from 255.Then we get the reverse image which adjusted size for interface.

And here is the example input and output

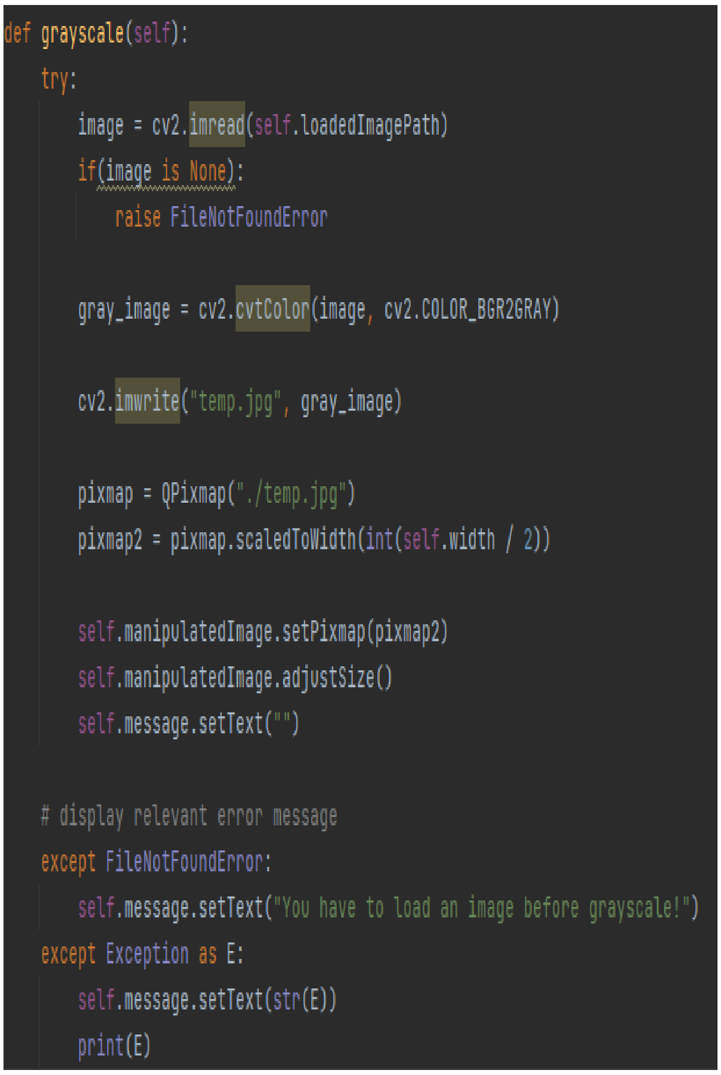
Input Image



Output Image



Grayscale Image Method

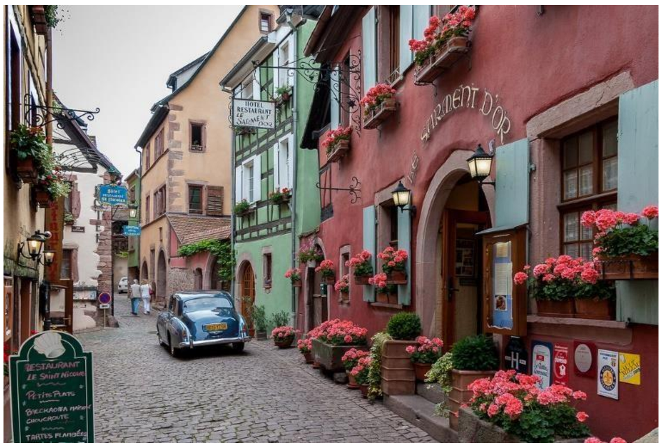


```python
def grayscale(self):
    try:
        image = cv2.imread(self.loadedImagePath)
        if(image is None):
            raise FileNotFoundError

        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        cv2.imwrite("temp.jpg", gray_image)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        self.message.setText("")

    # display relevant error message
    except FileNotFoundError:
        self.message.setText("You have to load an image before grayscale!")
    except Exception as E:
        self.message.setText(str(E))
        print(E)
```

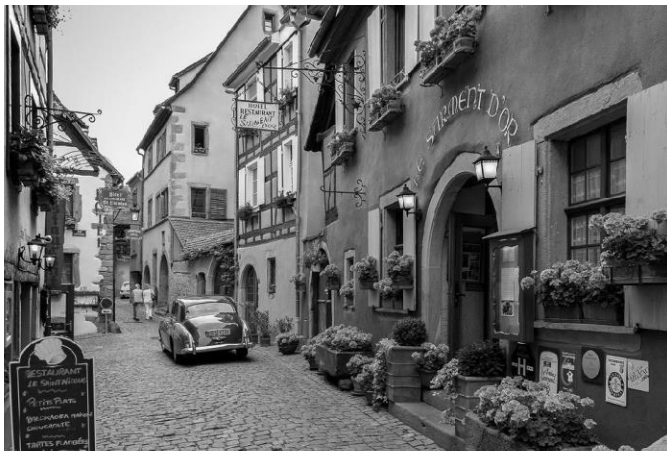
As you can see we read input file with opencv, if there is no such file, it raises an error. Then, we use opencv method which is COLOR_BGR2GRAY to grayscale image.Then we get the grayscale image which adjusted size for interface.

And here is the example input and output

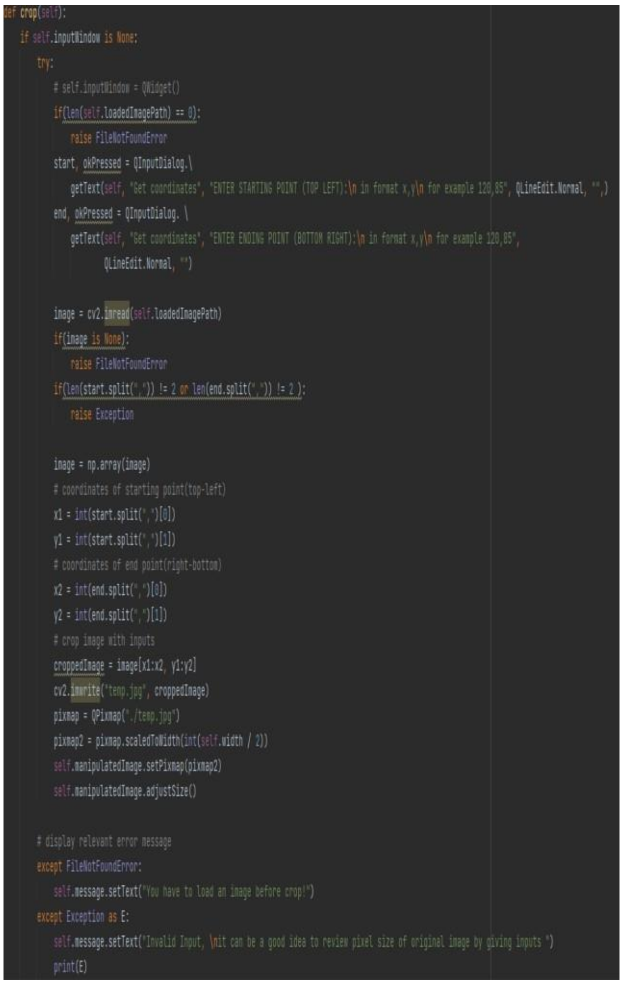
Input Image



Output Image



Crop Image Method



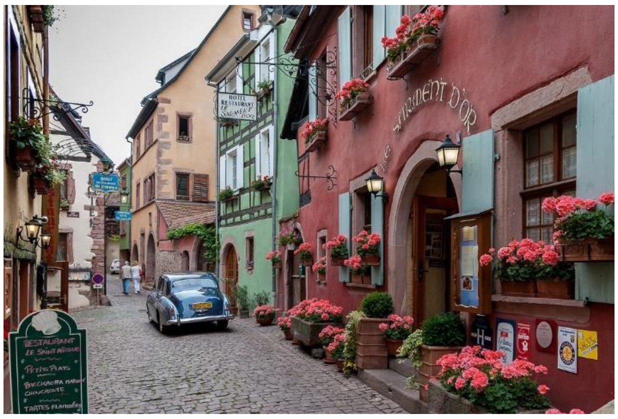
```python
def crop(self):
    if self.inputWindow is None:
        try:
            # self.inputWindow = QWidget()
            if(len(self.loadedImagePath) == 0):
                raise FileNotFoundError
            start, okPressed = QInputDialog.\
                getText(self, "Get coordinates", "ENTER STARTING POINT (TOP LEFT):\n in format x,y\n for example 120,85", QLineEdit.Normal, "",)
            end, okPressed = QInputDialog. \
                getText(self, "Get coordinates", "ENTER ENDING POINT (BOTTOM RIGHT):\n in format x,y\n for example 120,85",
                        QLineEdit.Normal, "")

            image = cv2.imread(self.loadedImagePath)
            if(image is None):
                raise FileNotFoundError
            if(len(start.split(",")) != 2 or len(end.split(",")) != 2 ):
                raise Exception

            image = np.array(image)
            # coordinates of starting point(top-left)
            x1 = int(start.split(",")[0])
            y1 = int(start.split(",")[1])
            # coordinates of end point(right-bottom)
            x2 = int(end.split(",")[0])
            y2 = int(end.split(",")[1])
            # crop image with inputs
            croppedImage = image[x1:x2, y1:y2]
            cv2.imwrite("temp.jpg", croppedImage)
            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))
            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()

        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before crop!")
        except Exception as E:
            self.message.setText("Invalid Input, \nit can be a good idea to review pixel size of original image by giving inputs ")
            print(E)
```

As you can see we read input file with opencv, if there is no such file, it raises an error. We convert image to image array with numpy. Then we take 2 points as top, left and right, bottom to cut it as input. After that, we output our new image using the inputs received.

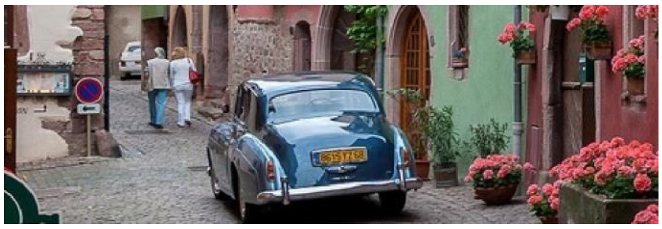
And here is the example input and output

We first load the image then enter the input size (400, 100) and (550,550) as top-left and right-bottom pixels.

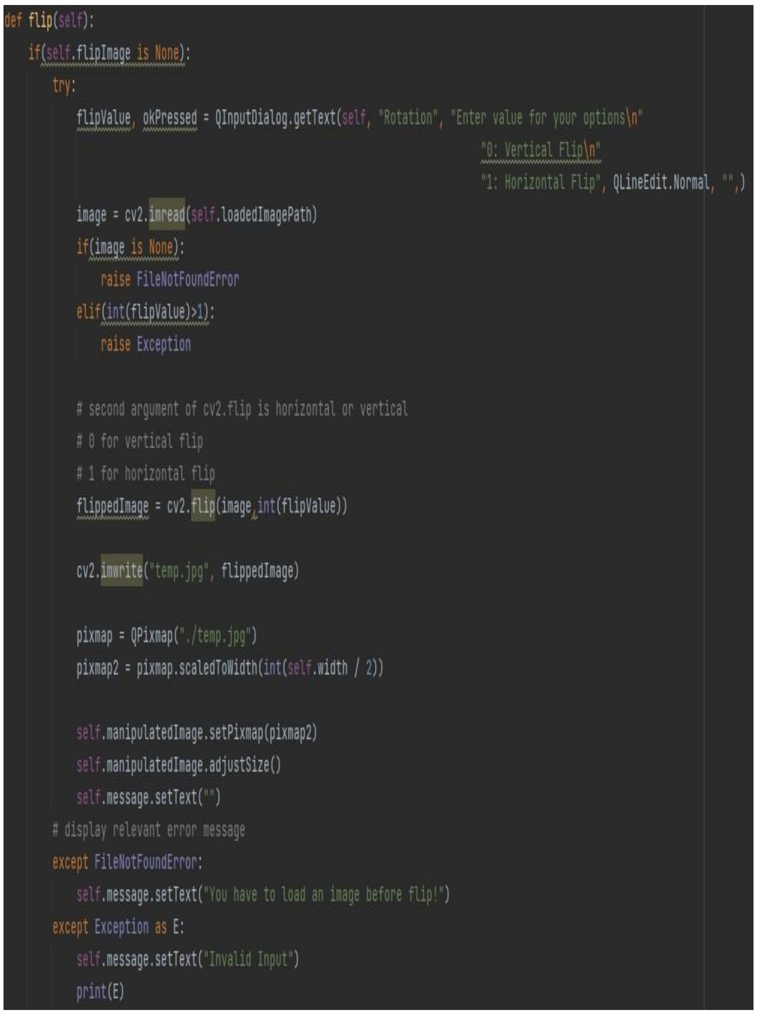
Input Image



Output Image



Flip Image Method

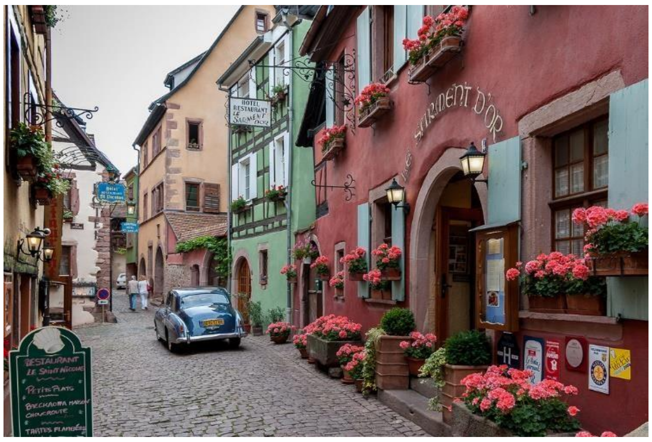


```
def flip(self):
    if(self.flipImage is None):
        try:
            flipValue, okPressed = QInputDialog.getText(self, "Rotation", "Enter value for your options\n"
                                                                          "0: Vertical Flip\n"
                                                                          "1: Horizontal Flip", QLineEdit.Normal, "",)

            image = cv2.imread(self.loadedImagePath)
            if(image is None):
                raise FileNotFoundError
            elif(int(flipValue)>1):
                raise Exception

            # second argument of cv2.flip is horizontal or vertical
            # 0 for vertical flip
            # 1 for horizontal flip
            flippedImage = cv2.flip(image,int(flipValue))

            cv2.imwrite("temp.jpg", flippedImage)

            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()
            self.message.setText("")
        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before flip!")
        except Exception as E:
            self.message.setText("Invalid Input")
            print(E)
```

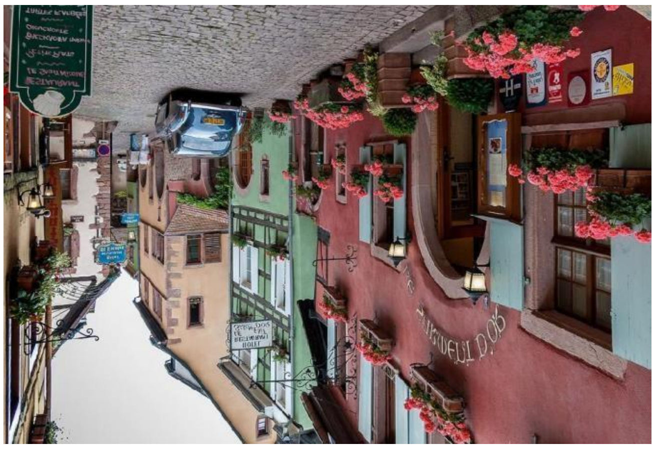
As you can see we read input file with opencv, if there is no such file, it raises an error. We get one of the values 0 and 1 in the input. If we take 0 as input value, we make vertical flip, otherwise we take 1 as input value and we do horizontal flip (mirror). Then we get the flipped image which adjusted size for interface.

And here is the example input and output

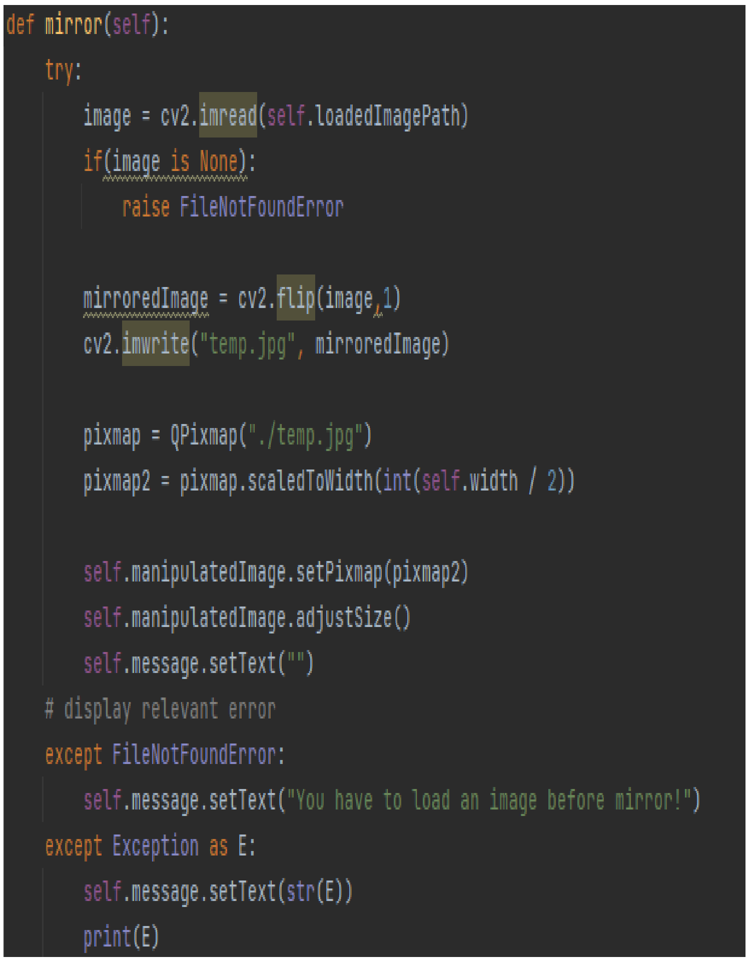
Input Image



Output Image



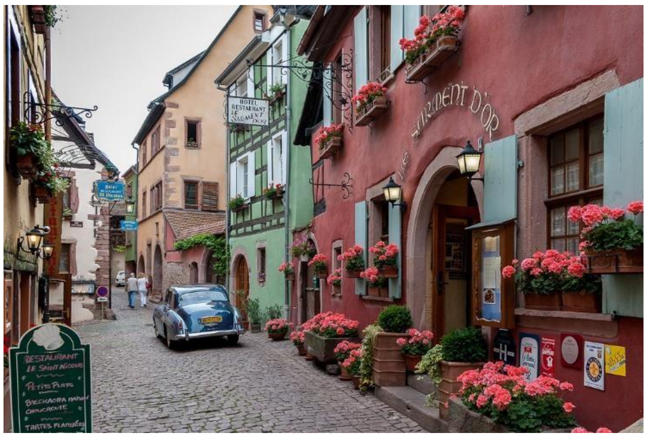
Mirror Image Method


```
def mirror(self):
    try:
        image = cv2.imread(self.loadedImagePath)
        if(image is None):
            raise FileNotFoundError

        mirroredImage = cv2.flip(image,1)
        cv2.imwrite("temp.jpg", mirroredImage)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        self.message.setText("")
    # display relevant error
    except FileNotFoundError:
        self.message.setText("You have to load an image before mirror!")
    except Exception as E:
        self.message.setText(str(E))
        print(E)
```

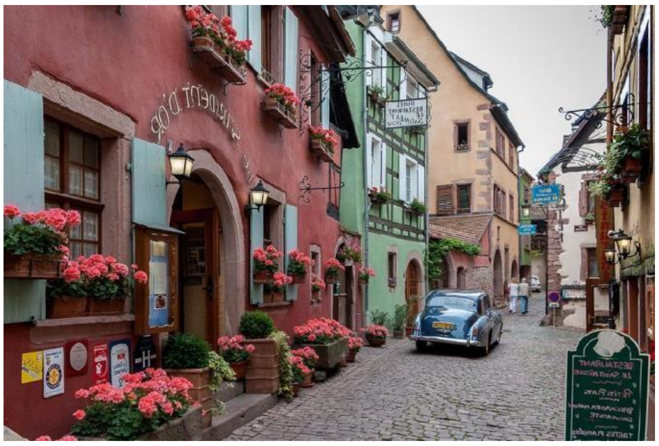
As you can see we read input file with opencv, if there is no such file, it raises an error. Then we use flip method from opencv to get mirror image. After that we get the mirror image which adjusted size for interface.

And here is the example input and output

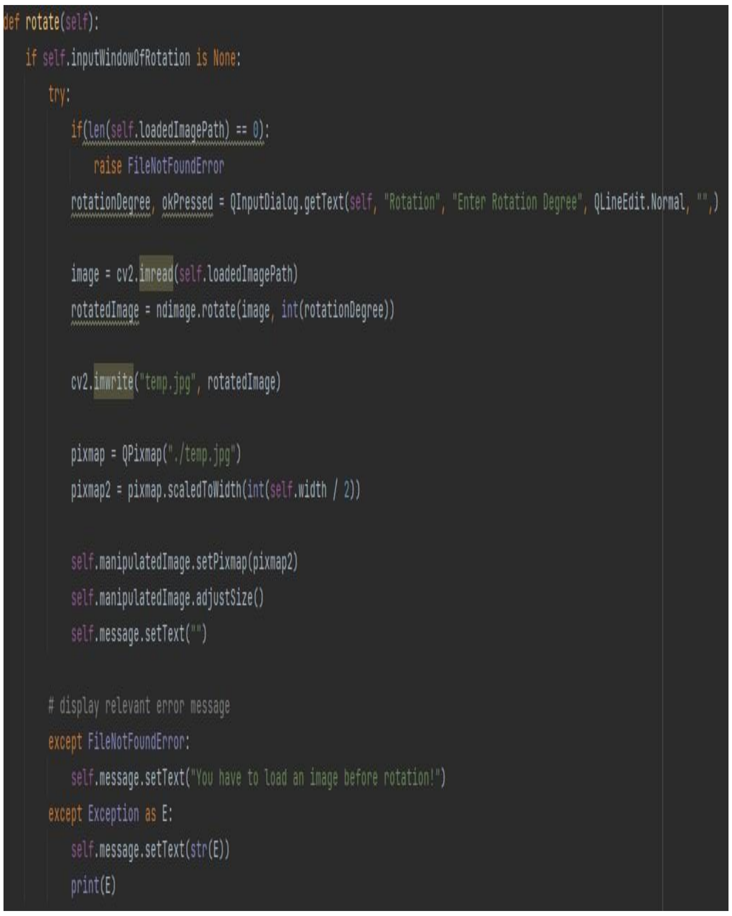
Input Image



Output Image



Rotate Image Method

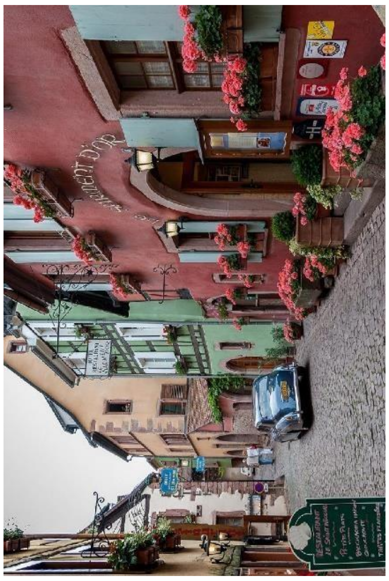


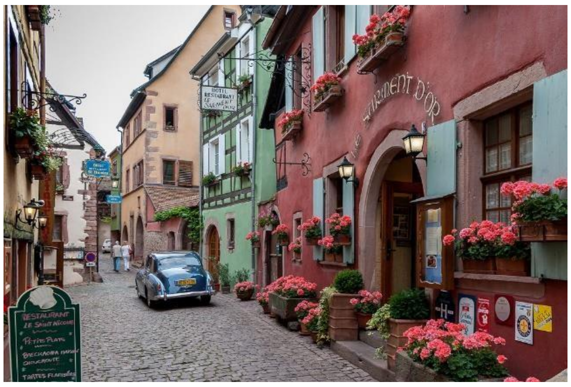
```
def rotate(self):
    if self.inputWindowOfRotation is None:
        try:
            if(len(self.loadedImagePath) == 0):
                raise FileNotFoundError
            rotationDegree, okPressed = QInputDialog.getText(self, "Rotation", "Enter Rotation Degree", QLineEdit.Normal, "",)

            image = cv2.imread(self.loadedImagePath)
            rotatedImage = ndimage.rotate(image, int(rotationDegree))

            cv2.imwrite("temp.jpg", rotatedImage)

            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()
            self.message.setText("")

        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before rotation!")
        except Exception as E:
            self.message.setText(str(E))
            print(E)
```

As you can see we read input file with opencv, if there is no such file, it raises an error. Then we use rotate method from scipy to get rotated image. After that, we get the rotated image which adjusted size for interface. The function make rotate for all degrees like 60, 90, 270, 45 etc.

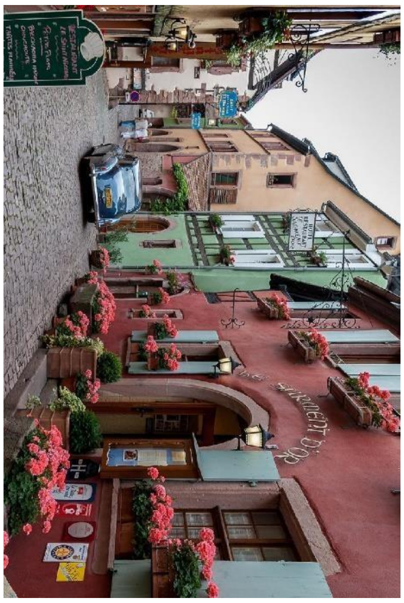
And here is the example input and output

270 degree rotate

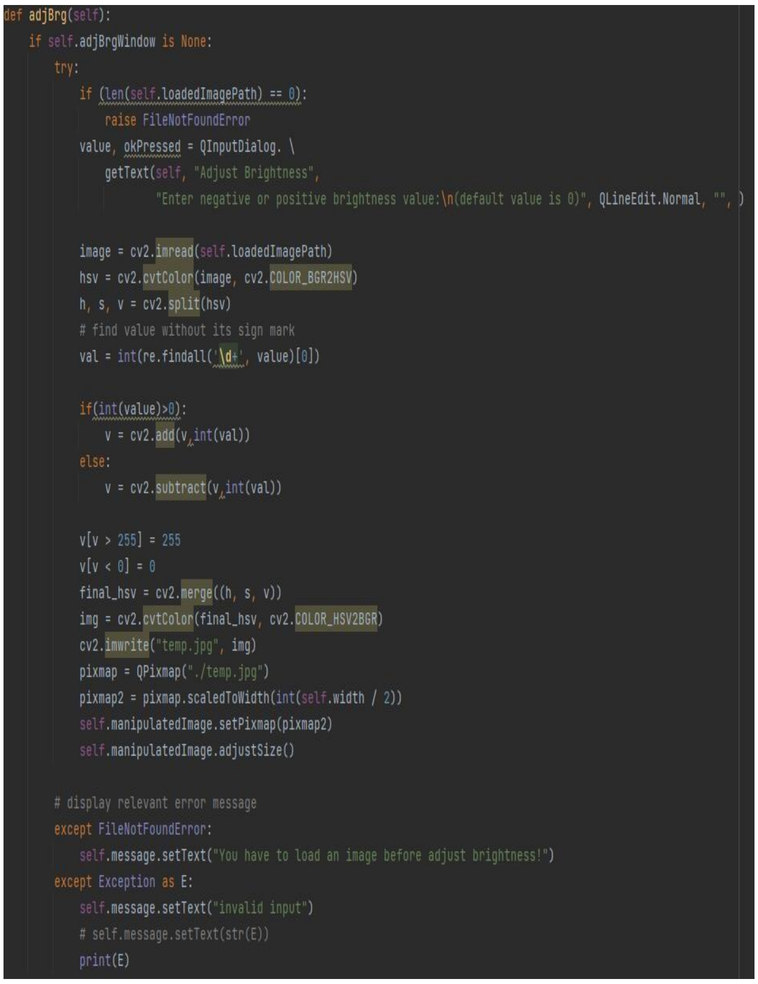




90 degree rotate



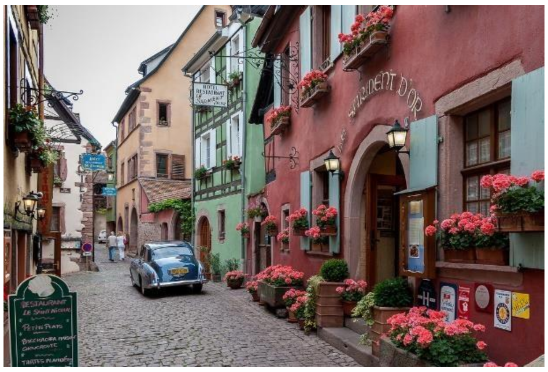
Adjust Brightness Method



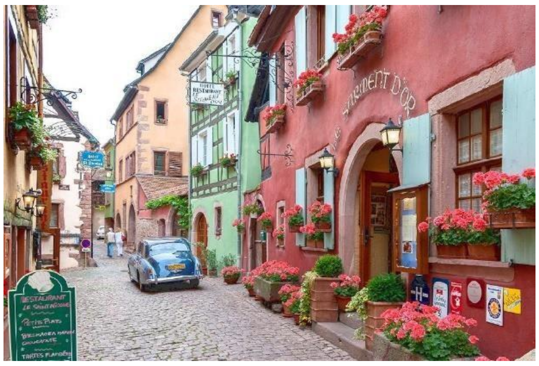
```python
def adjBrg(self):
    if self.adjBrgWindow is None:
        try:
            if (len(self.loadedImagePath) == 0):
                raise FileNotFoundError
            value, okPressed = QInputDialog. \
                getText(self, "Adjust Brightness",
                        "Enter negative or positive brightness value:\n(default value is 0)", QLineEdit.Normal, "", )

            image = cv2.imread(self.loadedImagePath)
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            h, s, v = cv2.split(hsv)
            # find value without its sign mark
            val = int(re.findall('\d+', value)[0])

            if(int(value)>0):
                v = cv2.add(v,int(val))
            else:
                v = cv2.subtract(v,int(val))

            v[v > 255] = 255
            v[v < 0] = 0
            final_hsv = cv2.merge((h, s, v))
            img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
            cv2.imwrite("temp.jpg", img)
            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))
            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()

        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before adjust brightness!")
        except Exception as E:
            self.message.setText("invalid input")
            # self.message.setText(str(E))
            print(E)
```

As you can see we read input file with opencv, if there is no such file, it raises an error. We converted the input image from bgr to hsv, then we adjusted the brightness depends on the v value of hsv if the input is positive we increase the v value otherwise we decrease it. Then we change the image hsv to bgr. Finally, we get the output image which adjusted size for interface.

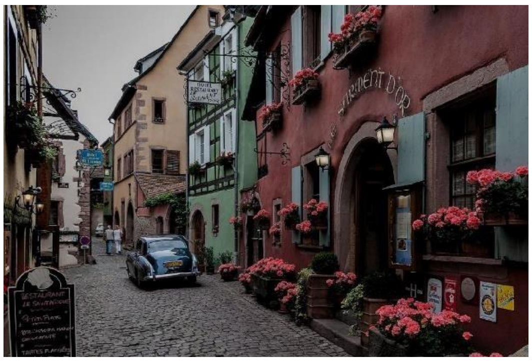
And here is the example input and output



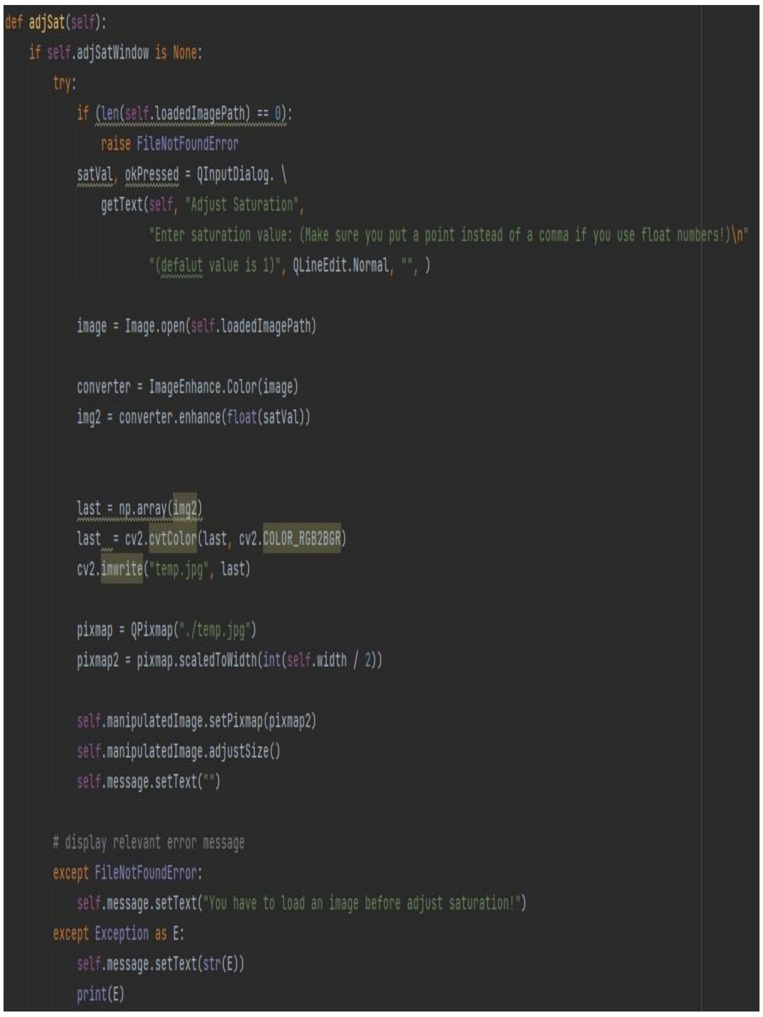
Brightness value is 50



Brightness value is -50



Adjust Saturation Method



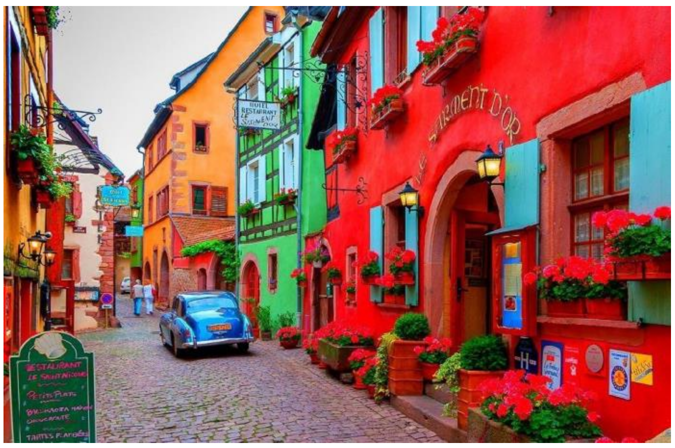
```python
def adjSat(self):
    if self.adjSatWindow is None:
        try:
            if (len(self.loadedImagePath) == 0):
                raise FileNotFoundError
            satVal, okPressed = QInputDialog. \
                getText(self, "Adjust Saturation",
                        "Enter saturation value: (Make sure you put a point instead of a comma if you use float numbers!)\n"
                        "(defalut value is 1)", QLineEdit.Normal, "", )

            image = Image.open(self.loadedImagePath)

            converter = ImageEnhance.Color(image)
            img2 = converter.enhance(float(satVal))


            last = np.array(img2)
            last_ = cv2.cvtColor(last, cv2.COLOR_RGB2BGR)
            cv2.imwrite("temp.jpg", last)

            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()
            self.message.setText("")

        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before adjust saturation!")
        except Exception as E:
            self.message.setText(str(E))
            print(E)
```

As you can see we read the input image. Then we use enhance method from PIL for saturation. After that, we convert the image rgb to bgr and we get the output image which adjusted size for interface.

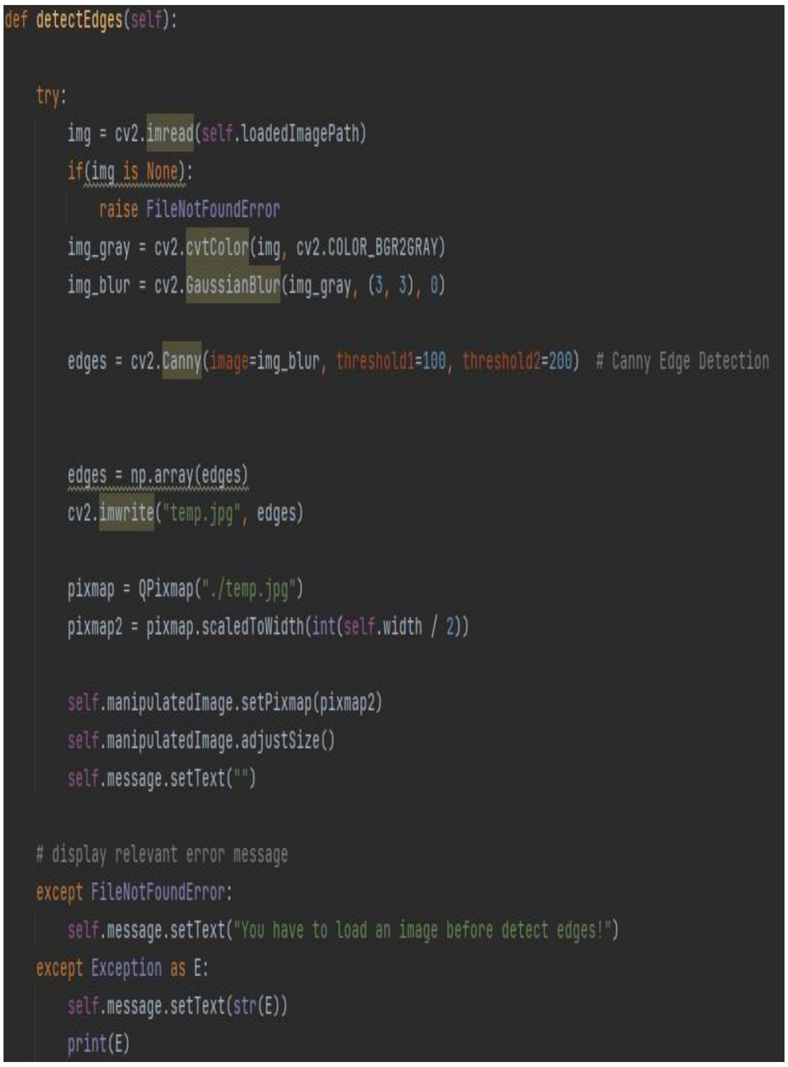
And here is the example input and output

Input Image

Output Image (value = 4)



Detect Edges Method

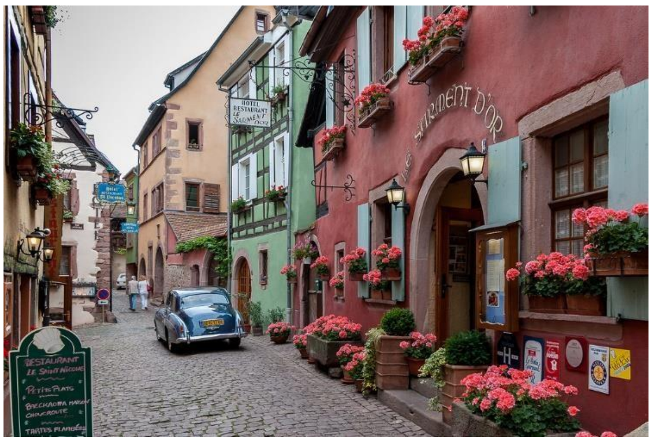


```
def detectEdges(self):

    try:
        img = cv2.imread(self.loadedImagePath)
        if(img is None):
            raise FileNotFoundError
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)

        edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200)  # Canny Edge Detection


        edges = np.array(edges)
        cv2.imwrite("temp.jpg", edges)

        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        self.message.setText("")

    # display relevant error message
    except FileNotFoundError:
        self.message.setText("You have to load an image before detect edges!")
    except Exception as E:
        self.message.setText(str(E))
        print(E)
```

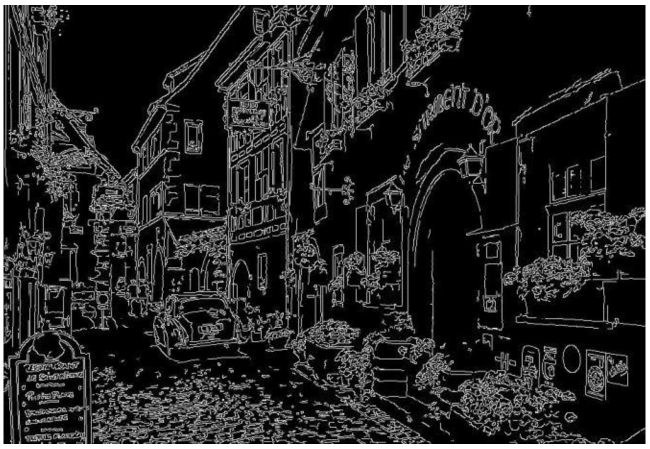
As you can see we read the input image with opencv. Then we change image bgr to gray. After that, we use GaussianBlur and Canny method from opencv to detect edges. Then, we get the output image with edges which adjusted size for interface.

And here is the example input and output

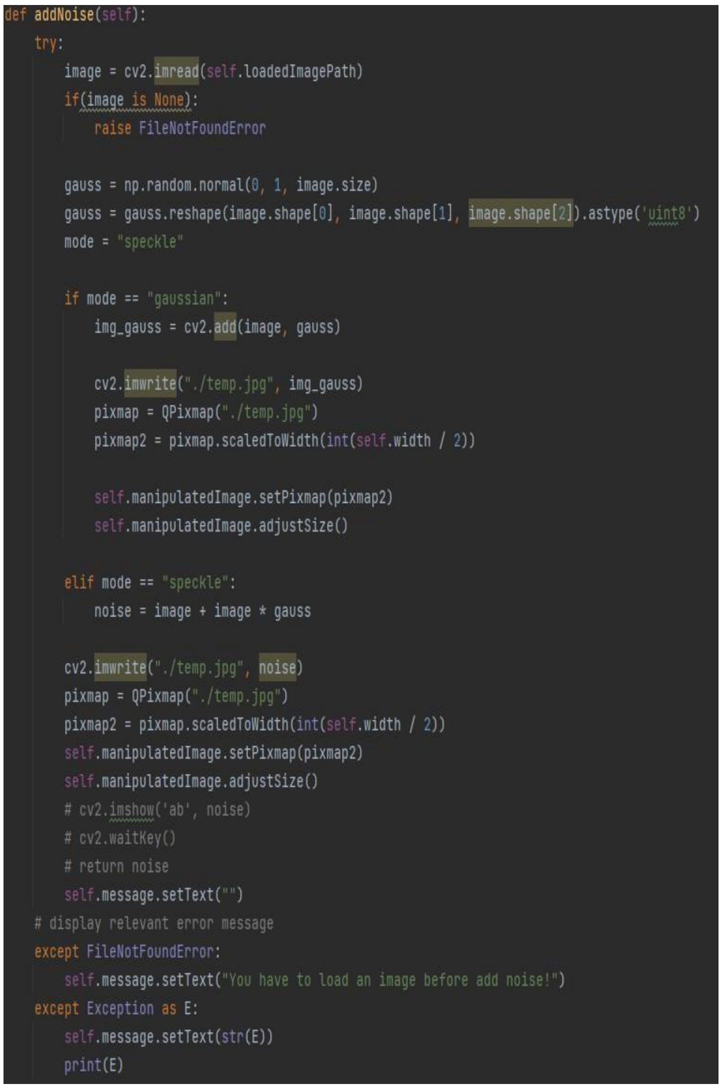
Input Image



Output Image



Adding Noise Method

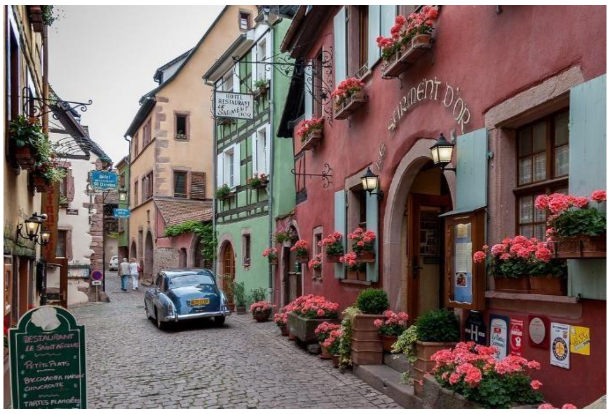


```
def addNoise(self):
    try:
        image = cv2.imread(self.loadedImagePath)
        if(image is None):
            raise FileNotFoundError

        gauss = np.random.normal(0, 1, image.size)
        gauss = gauss.reshape(image.shape[0], image.shape[1], image.shape[2]).astype('uint8')
        mode = "speckle"

        if mode == "gaussian":
            img_gauss = cv2.add(image, gauss)

            cv2.imwrite("./temp.jpg", img_gauss)
            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()

        elif mode == "speckle":
            noise = image + image * gauss

        cv2.imwrite("./temp.jpg", noise)
        pixmap = QPixmap("./temp.jpg")
        pixmap2 = pixmap.scaledToWidth(int(self.width / 2))
        self.manipulatedImage.setPixmap(pixmap2)
        self.manipulatedImage.adjustSize()
        # cv2.imshow('ab', noise)
        # cv2.waitKey()
        # return noise
        self.message.setText("")
    # display relevant error message
    except FileNotFoundError:
        self.message.setText("You have to load an image before add noise!")
    except Exception as E:
        self.message.setText(str(E))
        print(E)
```

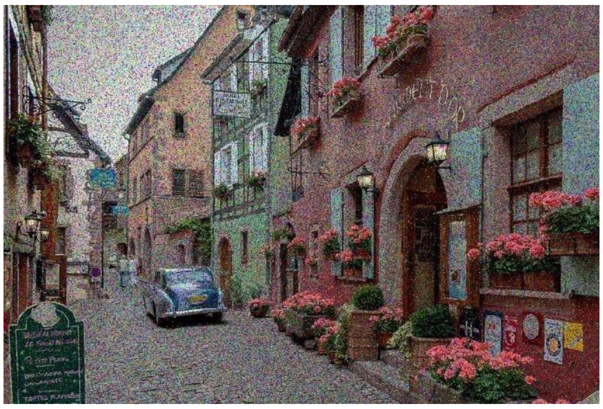
As you can see we read the input image with opencv. Then we describe gauss variable from random number which depends on image size. After that, we reshape from it. Then we describe noise as image + image * gauss which we describe above. Then, we get the output image which adjusted size for interface.

And here is the example input and output

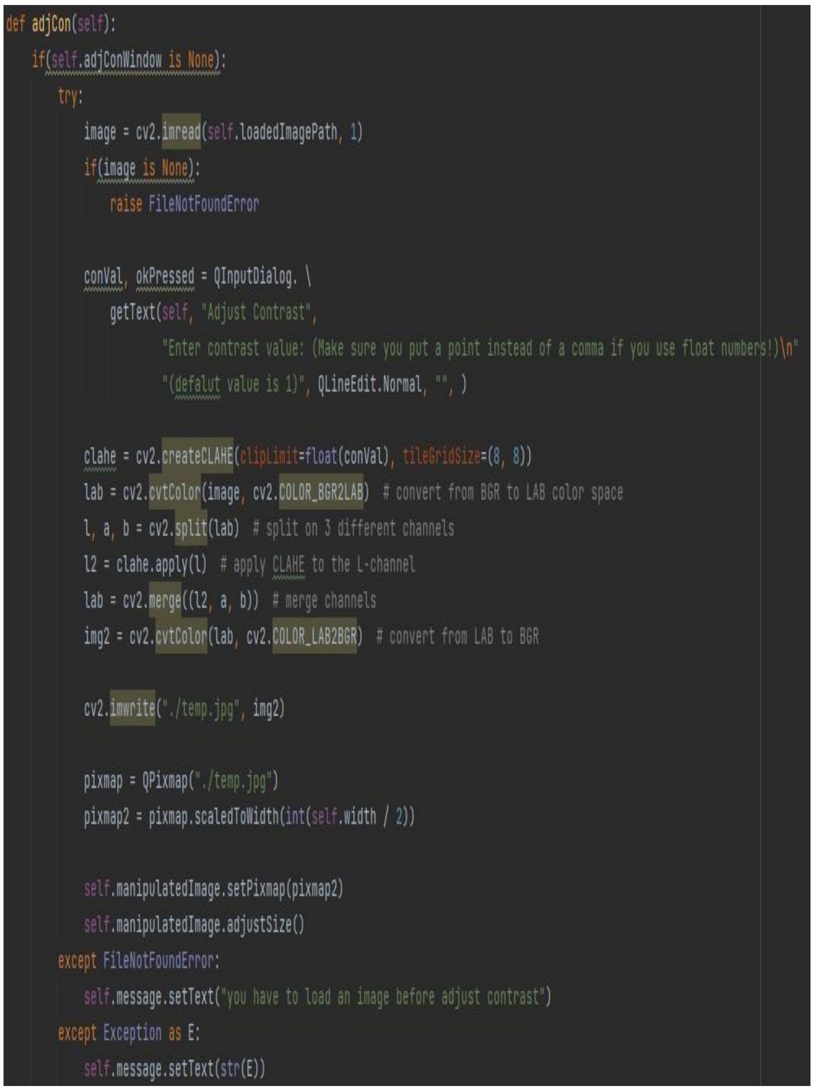
Input Image



Output Image



Adjust Contrast Method

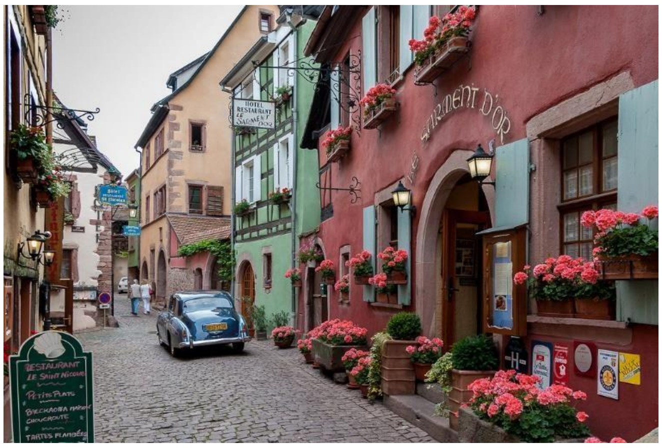


```python
def adjCon(self):
    if(self.adjConWindow is None):
        try:
            image = cv2.imread(self.loadedImagePath, 1)
            if(image is None):
                raise FileNotFoundError

            conVal, okPressed = QInputDialog. \
                getText(self, "Adjust Contrast",
                        "Enter contrast value: (Make sure you put a point instead of a comma if you use float numbers!)\n"
                        "(defalut value is 1)", QLineEdit.Normal, "", )

            clahe = cv2.createCLAHE(clipLimit=float(conVal), tileGridSize=(8, 8))
            lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)  # convert from BGR to LAB color space
            l, a, b = cv2.split(lab)  # split on 3 different channels
            l2 = clahe.apply(l)  # apply CLAHE to the L-channel
            lab = cv2.merge((l2, a, b))  # merge channels
            img2 = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)  # convert from LAB to BGR

            cv2.imwrite("./temp.jpg", img2)

            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))

            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()
        except FileNotFoundError:
            self.message.setText("you have to load an image before adjust contrast")
        except Exception as E:
            self.message.setText(str(E))
```

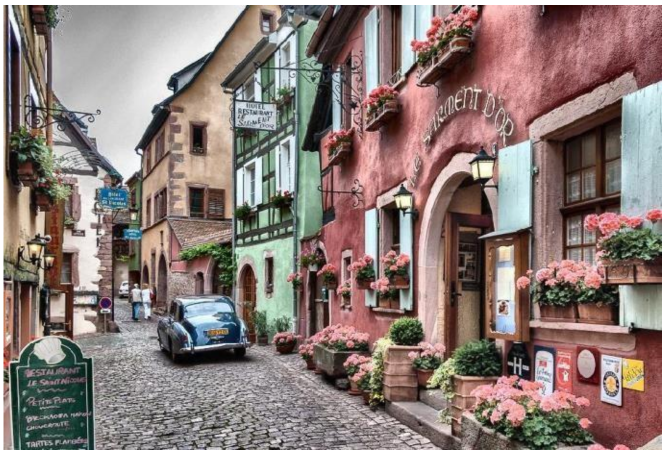
As you can see we read the input image with opencv. Then we create a clahe variable to equalization. After that, we convert the image from BGR to LAB. Then we take L value and apply clahe to L channel. Then, we merge channels and convert the final image from LAB to BGR. Finally, we get the output image which adjusted size for interface.

And here is the example input and output

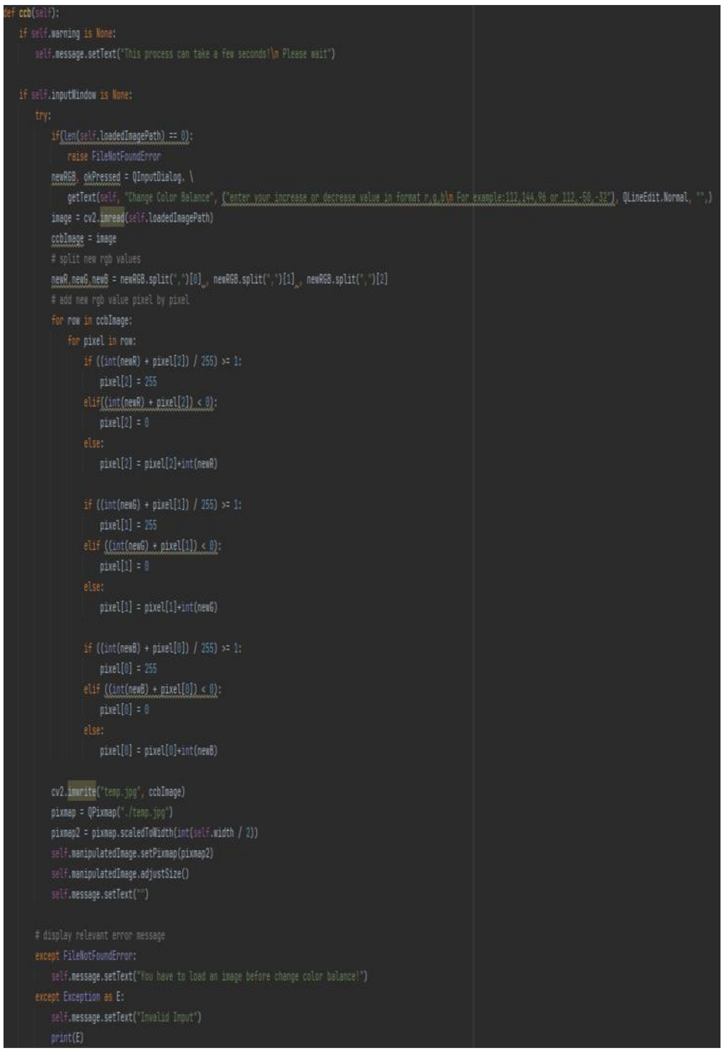
Input Image



Output Image



Change Color Balance Method



```
def ccb(self):
    if self.warning is None:
        self.message.setText("This process can take a few seconds!\n Please wait")

    if self.inputWindow is None:
        try:
            if(len(self.loadedImagePath) == 0):
                raise FileNotFoundError
            newRGB, okPressed = QInputDialog. \
                getText(self, "Change Color Balance", ("enter your increase or decrease value in format r,g,b\n For example:112,144,96 or 112,-50,-32"), QLineEdit.Normal, "",)
            image = cv2.imread(self.loadedImagePath)
            ccbImage = image
            # split new rgb values
            newR,newG,newB = newRGB.split(",")[0], newRGB.split(",")[1], newRGB.split(",")[2]
            # add new rgb value pixel by pixel
            for row in ccbImage:
                for pixel in row:
                    if ((int(newR) + pixel[2]) / 255) >= 1:
                        pixel[2] = 255
                    elif((int(newR) + pixel[2]) < 0):
                        pixel[2] = 0
                    else:
                        pixel[2] = pixel[2]+int(newR)

                    if ((int(newG) + pixel[1]) / 255) >= 1:
                        pixel[1] = 255
                    elif ((int(newG) + pixel[1]) < 0):
                        pixel[1] = 0
                    else:
                        pixel[1] = pixel[1]+int(newG)

                    if ((int(newB) + pixel[0]) / 255) >= 1:
                        pixel[0] = 255
                    elif ((int(newB) + pixel[0]) < 0):
                        pixel[0] = 0
                    else:
                        pixel[0] = pixel[0]+int(newB)

            cv2.imwrite("temp.jpg", ccbImage)
            pixmap = QPixmap("./temp.jpg")
            pixmap2 = pixmap.scaledToWidth(int(self.width / 2))
            self.manipulatedImage.setPixmap(pixmap2)
            self.manipulatedImage.adjustSize()
            self.message.setText("")

        # display relevant error message
        except FileNotFoundError:
            self.message.setText("You have to load an image before change color balance!")
        except Exception as E:
            self.message.setText("Invalid Input")
            print(E)
```

As you can see we read the input image with opencv. Then we take input values as rgb values and split it. After that we put some conditions like if any value is bigger than 255 we equal it to 255. Also, we make same thing for less than 0. Then we merge it again and get the output image which adjusted size for interface.

And here is the example input and outputs