BLG 453E Computer Vision (Fall 2018)
Homework-1

Alperen Kantarcı - 150140140
October 13, 2018

1 Application Usage

In order to use the app you need "Mainapp.py ui.py popup.py Histogram.py HistogramMatcher.py"" and if you have these python files and the requirements that written on the Github site then you can use this command to run the application: python Mainapp.py
Github page of this application is here.
Application has 2 menu buttons for functionality. First one is File menu which has 3 buttons. You can see the initial state of the application in Figure 1. Clicking to the Open Input (Ctl+I) button opens a new file dialog for you to choose image (png or jpg) as input image. After selecting the input image you will see your selected image and it's histogram at the left most part of the application. Also histogram image will be saved the same directory with the "Mainapp.py" file. Open Target (Ctrl+T) button behaves same with the Open Input button and your image selection and corresponding histogram will be placed center of the application as you can see in the Figure 2. Histogram image of the target will be saved the same directory right after importing the target image. Clicking Exit (Ctrl+Q) button closes the application as expected. Equalize Histogram button uses input and target histograms to match the histogram of the input with the target. If you haven't added input or target image then clicking to this button will give error popup like Figure 3. If you have added two images then application will calculate and will show you the final result on the rightmost part of the application like Figure 4. Moreover, you will see the result image histogram as well. Also your result image and it's histogram will be saved in the same directory for you to use or inspect the results. In the below i will give the default saving names but these names can be changed from the code.

Input image histogram = "*input_histogram.png*"
Target image histogram = "*target_histogram.png*"
Result image = "*result.png*"
Result image histogram = "*result_histogram.png*"

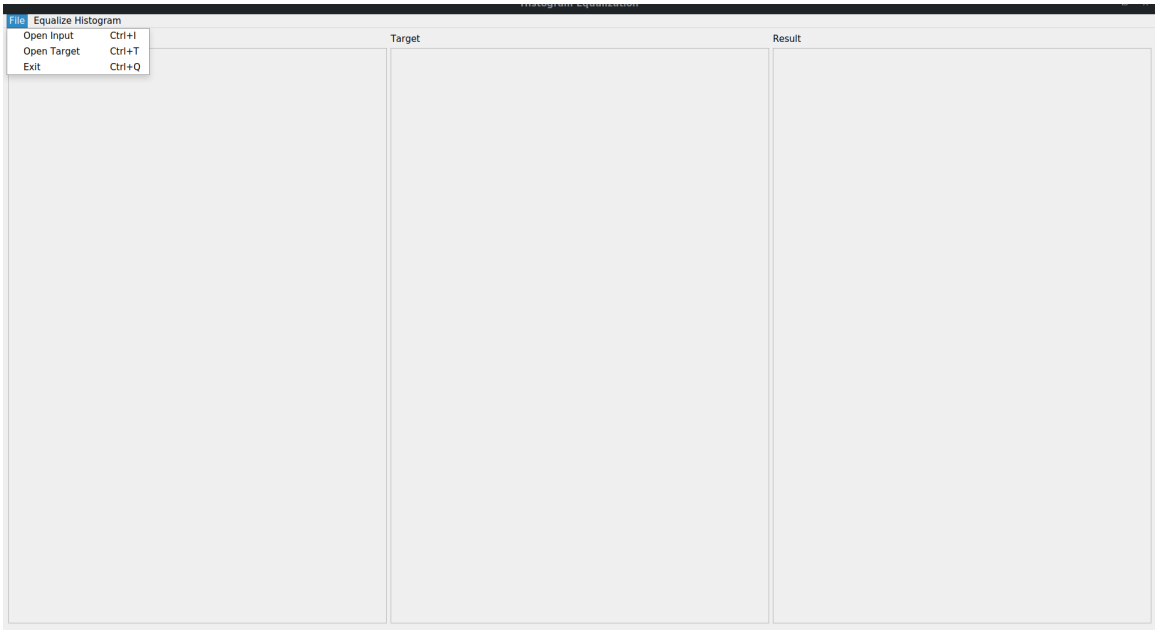


Figure 1: Application general view.

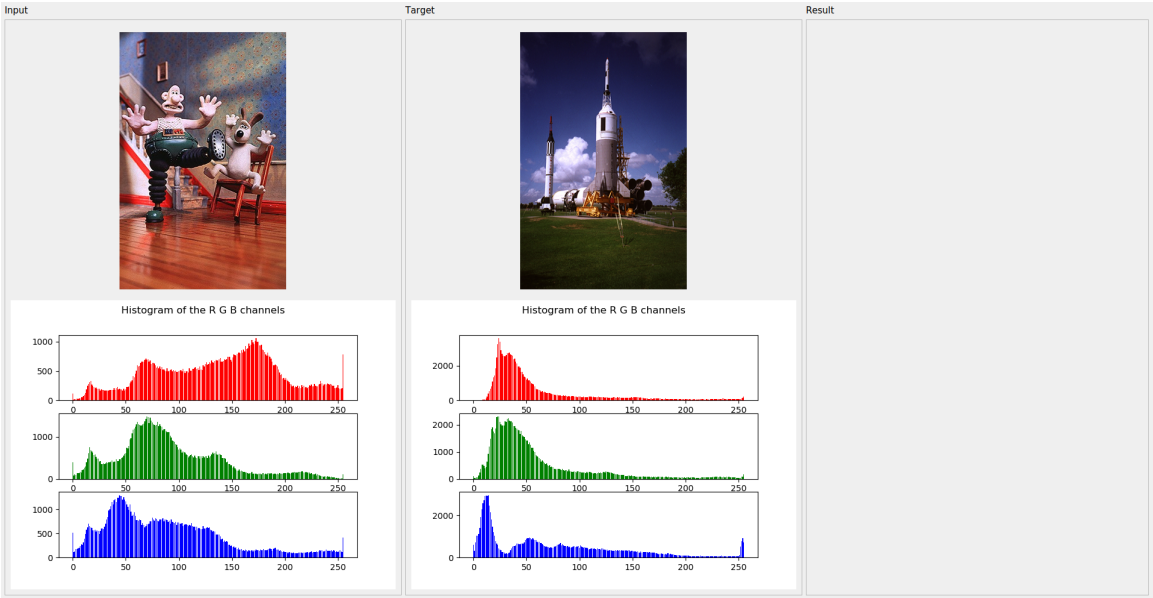


Figure 2: Adding input and target images.

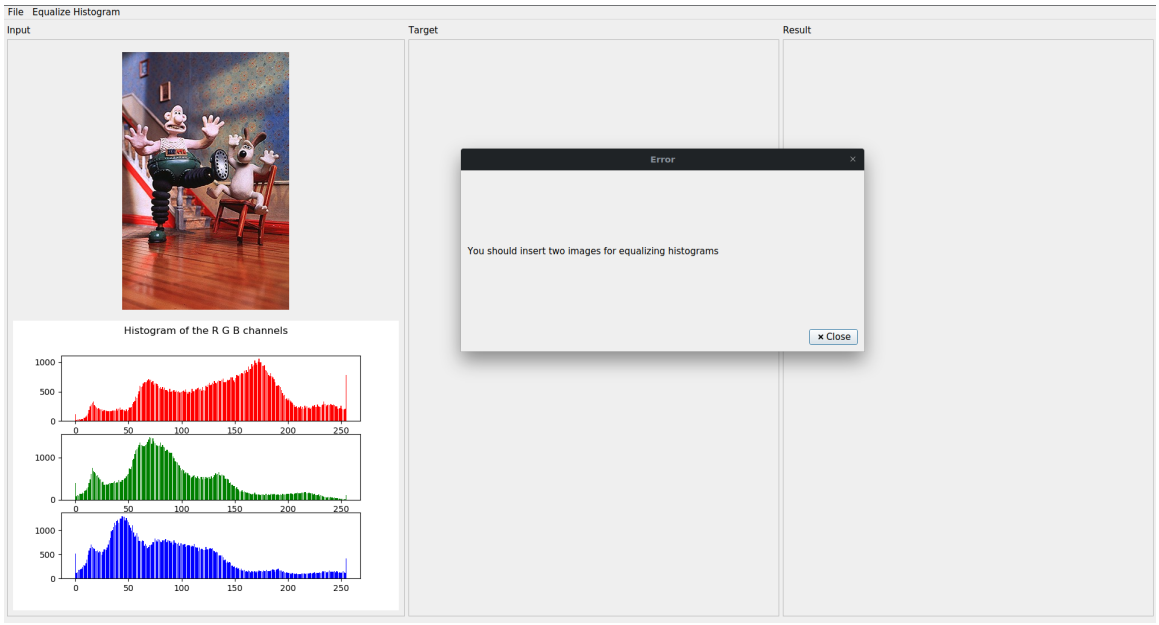


Figure 3: Error popup when input or target not selected.

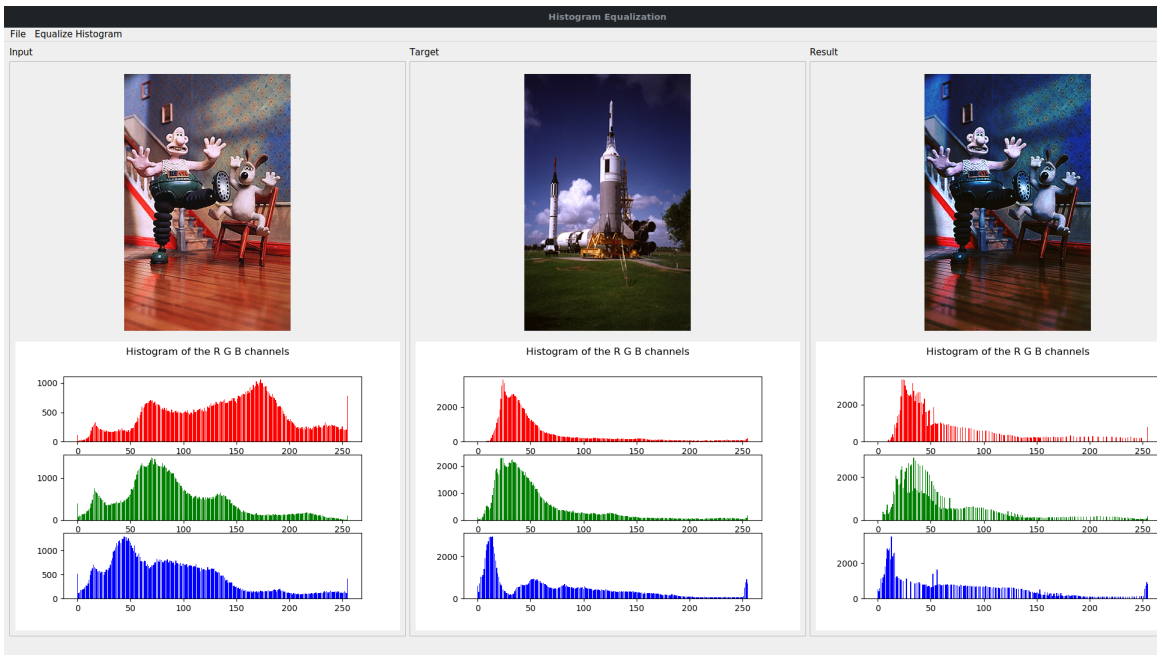
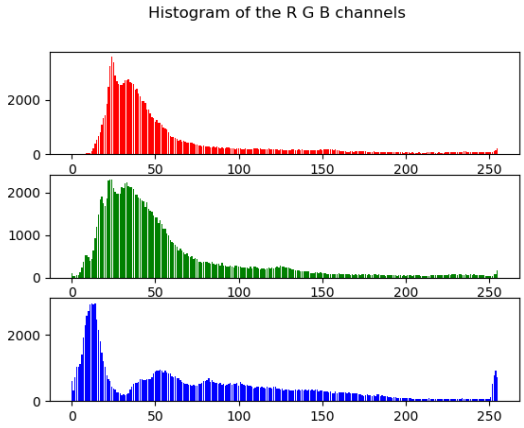


Figure 4: Matched images example.

2 Example outputs

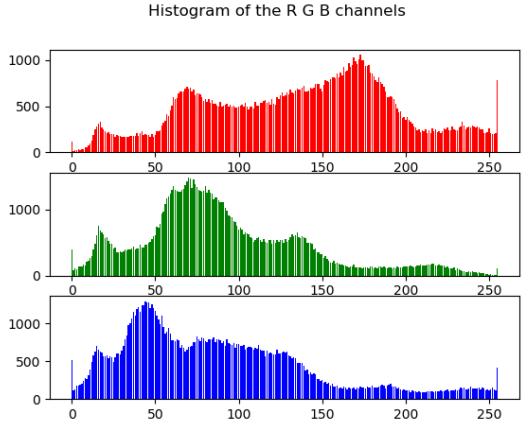
Here we have two images and their corresponding histograms.

(a) Input image



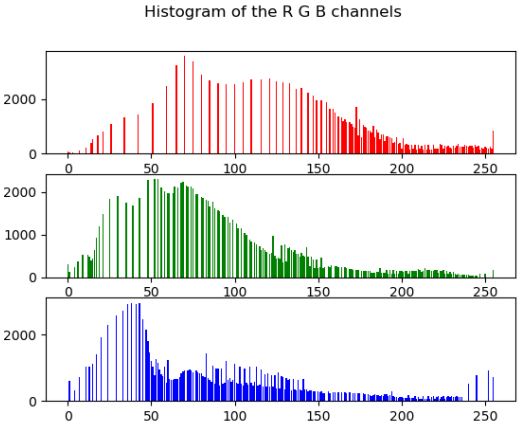
(b) Input image histogram

(c) Target image



(d) Target image histogram

(e) Final result image



(f) Final result image histogram

Using these two histograms we get this result image and it's histogram. If you inspect the histogram it was expected result because image became more reddish because target image has more red than other colors.

3 Python code

We have basically 3 main part as code. First part is UI controller, second is Histogram class that responsible for creating histogram and third one is the HistogramMatcher class which is responsible for matching histograms using two image histograms.
In ui.py there are lot of code that establish the user interface and create the functionalities like opening image. There is three important function inside this python file. Two of them work same; they create file picker dialog get image and create histogram of the image. As you can see below it create histogram then get the histogram and show it to the user. Then it saves this histogram object for further matching process.

```
def openTargetFileDialog(self):
    options = QtWidgets.QFileDialog.Options()
    filename, _ = QtWidgets.QFileDialog.getOpenFileName(None,"
        QFileDialog.getOpenFileName()","","PNG Files (*.png);;JPG Files
        (*.jpg)",options=options)
    if filename:
        pixMap = QtGui.QPixmap(filename)
        self.target_image_label.setPixmap(pixMap)
        hist = Histogram(filename)
        output = "target_histogram.png"
        hist.createHistogramPlotImage(output)
        pixMapHist = QtGui.QPixmap(output)
        self.target_hist_label.setPixmap(pixMapHist)
        # Save the object for matching
        self.target_hist = hist
```

Another important function is the one which matches two histograms. It takes the two saved histogram objects and sends it to the matcher class. If either input or target is empty then it creates error popup to warn the user for adding images.

```
def equalizeHistogram(self):
    if(self.input_image_label.pixmap() is not None and self.
        target_image_label.pixmap() is not None):
        filename = "result.png"
        matcher = HistogramMatcher(self.input_hist,self.target_hist,
            filename)
        matcher.createLookupTable()
        matcher.constructImage()
        # Show new image
        pixMap = QtGui.QPixmap(filename)
        self.result_image_label.setPixmap(pixMap)
        # Create histogram of the result
        hist = Histogram(filename)
        hist_filename = "result_histogram.png"
        hist.createHistogramPlotImage(hist_filename)
```

```
        # Show histogram
        pixMapHist = QtGui.QPixmap(hist_filename)
        self.result_hist_label.setPixmap(pixMapHist)
        self.result_hist = hist
    else:
        self.Ui_Dialog = Ui_Dialog()
        self.Ui_Dialog.setupUi(self.Ui_Dialog)
        self.Ui_Dialog.show()
```

Histogram class takes image filename and creates histogram of the image. Also it can create the cumulative histogram too. Note that it is mandatory to create cumulative histogram if you want to use HistogramMatcher. I didn't set cumulative histogram creation mandatory because it takes time to calculate and as object oriented paradigm says this class can be used outside of the matching algorithm so developer should call the cumilative histogram creation before using this object in HistogramMatcher.
Creation of the histogram easy but costly operation. It counts the instensity values at every pixel of each channels and make them array for fast access. Array dimensions are depending on intensity range and channels.

```
def createHistogramArray(self):
  """
  Creates histogram of the intensities for each channel and save them
      into one numpy array
  """
  # Read image
  I = cv2.imread(self.imageFilename)
  self.row , self.column , self.channel = I.shape
  # Create histogram array of the all channels
  hist = np.zeros([self.intensityRange, 1, self.channel])
  # Range through the intensity values and find the frequences
  for g in range(self.intensityRange):
    hist[g, 0,...] = np.sum(np.sum(I == g, 0), 0)
  return hist
```

Creation of the cumulative histogram easier than creating histogram. You just calculate the probability of each intensity level by dividing total pixel number. Then at each level you add the probabilities and write to it's place in the array. So at the end you get 1 probability, but because of the floating point precision you generally end up with 0.99 probability.

```
def createCdfHistogram(self):
  # Create histogram that has all the channels with appropiate shape
  self.cdfArray = np.zeros([self.intensityRange,1,self.channel])
  # Temporary variable array for holding values
  temp = np.zeros([self.channel,1])
  totalPx = self.row * self.column
  for i in range(self.intensityRange):
    for j in range (self.channel):
      # Calculate cumilative probability
      temp[j] += self.histArray[i,0,j] / totalPx
      self.cdfArray[i,0,j] = temp[j]
```

Histogram Matcher class is the class that takes two histogram object and output filename and

matching histograms. If you want it also creating and saving the image from matched histogram. For histogram matching algorithm we need to create lookup table. Then using lookup table each channel is mapped into target histogram that will be matched.

```
def createLookupTable(self):
  # Lookup table will have shape of (intensityRange,channel)
  self.lookupTable = np.zeros([self.inputCDF.intensityRange,self.
      inputCDF.channel])
  # Temp variable for counting intensity level
  gj = np.zeros([self.inputCDF.channel,1],dtype=np.uint8)
  for i in range(self.inputCDF.intensityRange):
    # For every level we need one value for lookup table
    for j in range(self.inputCDF.channel):
      # For all channels
      while self.targetCDF.cdfArray[gj[j,0],0,j] < self.inputCDF.
          cdfArray[i,0,j] and gj[j,0] < 255:
        # If target cdf value is bigger than current(i) instensity then
            save that intensity level to the lookup table
        # actually make mapping from i to gj[j,0] value
        gj[j,0] += 1
      # Save the lookup table
      self.lookupTable[i,j] = gj[j,0]
      # Reset the temp values
      gj[j,0] = 0
```

As you can see below each channel is mapped. This operation can be shortened somehow but i couldn't figure out to map using only 3 dimensional array. So i map 3 times, two dimensional channel arrays and then merge them as one image. Output image is in the PNG format with default compression value of OpenCV which is 3.

```
def constructImage(self):
  # Read the image that will be mapped
  I = cv2.imread(self.inputCDF.imageFilename)
  # Make the mappings for each channel
  red = np.uint8(self.lookupTable[:,0][I])[:,:,0]
  green = np.uint8(self.lookupTable[:,1][I])[:,:,1]
  blue = np.uint8(self.lookupTable[:,2][I])[:,:,2]
  # Construct the rgb image from new mapped arrays
  image = np.dstack((red,green,blue))
  self.imageArray = image
  # Write image to the given path
  cv2.imwrite(self.resultFilename,self.imageArray,[cv2.
      IMWRITE_PNG_COMPRESSION, 3])
```