```
(*                Computer Graphics with Applications of Dr. Makhanov.
                                    Lab 4:
                          Basic Image Processing with Mathematica *)
```

```
SetOptions[{ListPlot3D, Histogram, Plot}, ImageSize → Small];
```

```
SetOptions[EvaluationNotebook[], ShowCellLabel → False];
```

```
(* Problem (1) Import the image val_2023.
 png and output the dimensions and the channels   *)
```

```
(* Set up your directory where you uploaded the images*)
(* C:/ ... for windows style path *)
(* //Users/ictlab/Desktop/... if you run on Mac OS*)
```

```
SetDirectory["/Users/ictlab/Desktop/CSS221 Tuesday/Lab4"]
```

```
/Users/ictlab/Desktop/CSS221 Tuesday/Lab4
```

```
T0 = Import["coo.jpeg"]
```

```
ImageDimensions[T0]
```

```
{1079, 1031}
```

```
(* Usually the color image has R,
G and B channels. Some hardware automatically generates
 AlphaChannel. In this case apply  RemoveAlphaChannel  *)
```

```
ImageChannels[T0]
```

```
3
```

```
(* Problem (2) Split T1 the RGB components, AlphaChannel. *)
```

```
(* In case of 4 channels apply RemoveAlphaChannel *)
```

```
T00 = Import["girl.png"]
```

```
ImageChannels[T00]
```

4

```
(* 4 channels! Apply RemoveAlphaChannel *)
```

```
T00New = RemoveAlphaChannel[T00]
```

```
ImageChannels[T00New]
```

3

```
(* Convert into the gray level image *)
```

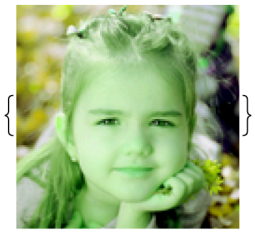
```
{TG = ColorConvert[T00New, "Grayscale"]}
```

```
(* Problem (3) Evaluate each color~matrix and combine
 them back into the color image in a different order *)
```

```
{CRed, CGreen, CBlue} = ColorSeparate[T00New]
```

```
(* Let us combine them into the image in a different way  *)
```

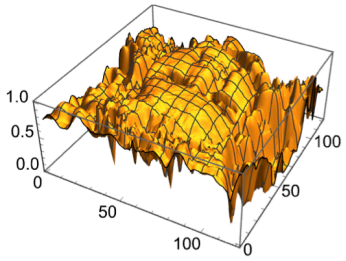
```
{T3 = ColorCombine[{CGreen, CRed, CBlue}]}
```



```
(* Obtain gray levels of the image Cred as a matrix
   and show them  using ListPlot3D. Can take some time....  *)
```

```
M3 = ImageData[CRed];
```

```
ListPlot3D[M3, ImageSize → Small] (* this may take some time *)
```




```
(* Convert M3 back into the image *)
```

```
{M3Image = Image[M3]}
```

```
(* Problem (4) Import cat.jpeg. Swap the red and the green matrices *)
```

```

```

```

```

```
(* Red *)
```

```

```

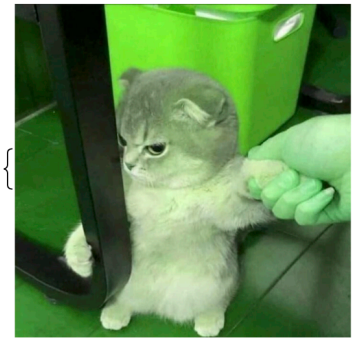
```
(* Green *)
```

```

```

```
(* Blue *)
```

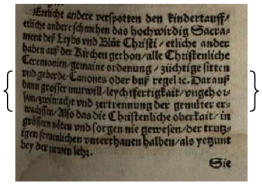
```
(* Swap and combine. Compare  *)
```




```
(* Binarization splits the gray levels into
 black and white regions depending on a given threshold *)
```

```
(* Problem (5) Read the image textold1.jpg
 and binarize it with an appropriate threshold*)
```

```
{T5 = Import["textold.jpg"]}
```

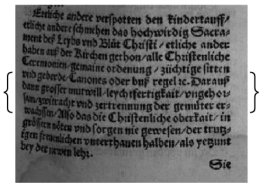


```
ImageDimensions[T5]
```

```
{332, 254}
```

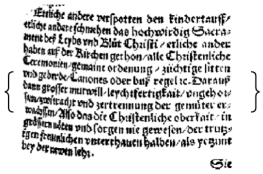
```
T5GL := ColorConvert[T5, "Grayscale"]
```

```
{T5GL}
```



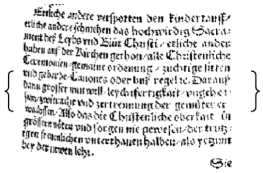
```
T5B := Binarize[T5GL, 0.2];
```

```
{T5B}
```



```
(* Problem (6) Use slider for solving Problem (5) *)
```

```
{Dynamic[Binarize[T5GL, Tre]]}
```



```
(* Design the slider *)
```

```
, 0.122}
```

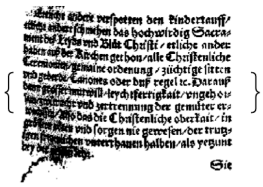
```
(* Finding the Threshold automatically  *)
```

```
Tre6 = FindThreshold[T5GL]
```

```
0.301961
```

```
T6 := Binarize[T5GL, Tre6]
```

{T6}

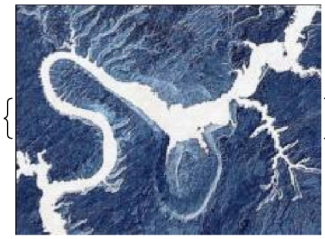


```
(* Problem (7) Find an appropriate threshold to binarize the
 satellite image river1.jpg  to remove details and visualize the
 river flow. Use ColorNegate[Image] to obtain the negative image *)
```

```
(* Import the image *)
```

```
R1 := Import["river1.JPG"]
```

{R1}

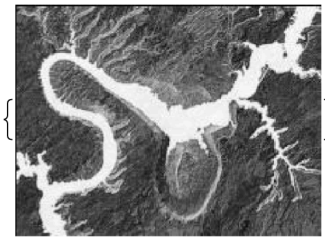


```
(* Convert into the gray level  *)
```

```

```

{R2}

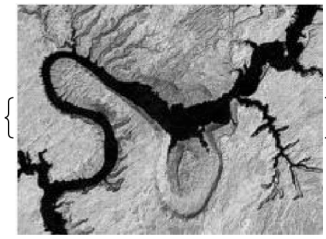


```
(* Use ColorNegate to obtain negative of the image R2
  Use M-13 help to find this function  *)
```

```

```

{R3}



{Dynamic[Binarize[R3, Tr3]]}

```
(* Design the slider for Tr3*)
```

[redacted]

{ , 0.166}

```
(* Binarize with a fixed threshold selected by slider and save in R4*)
```

[redacted]

{Null}

{R4}

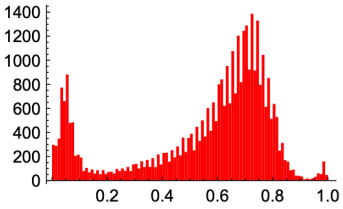
```
(* Gray level histogram shows for each gray level the
 number of pixels in the image that have this gray level The
 one~dimensional histogram functions return a vector N elements
 long containing the number of intensity levels in each bin *)
```

```
(* Problem (7') Binarize image river1.jpg using a slider and a histogram *)
```

```
(* Convert into the image data and flatten the image *)
```

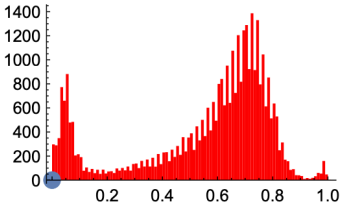
```
RF3 = Flatten[ImageData[R3]];
```

```
HP = Histogram[RF3, {0.01}, ChartStyle → Red]
```




```
{Slider[Dynamic[TreP], Background → LightBlue], Dynamic[TreP]}
```

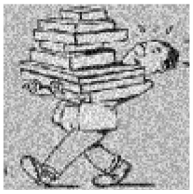

```
Dynamic[Show[HP, ListPlot[{{TreP, 0}}, PlotStyle → PointSize[0.06]]]]
```




```
{RH4 = Dynamic[Binarize[R3, TreP]]}
```

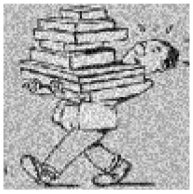
```
(* Problem (8) Remove the background noise from image in
 Heavy_load_noisy.jpg using the method applied in Problem (7') *)
```

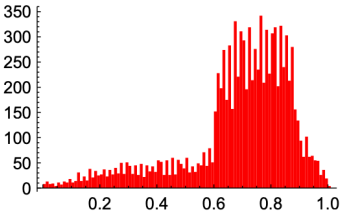
```
HL1 = Import["Heavy_load_noisy.jpg"]
```



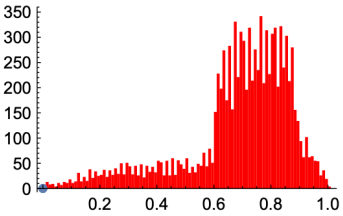
```
ImageChannels[HL1]
```

```
3
```





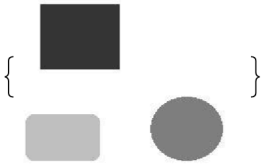

```
{Slider[Dynamic[TreHL], Background → LightBlue], Dynamic[TreHL]}
```


```
(* Binarize[image, {t1,t2}] creates a binary image by replacing all values in
 the range t1 through t2 (two thresholds) with 1 and others with 0. This is
 used to segment an object with the gray levels in a specified interval.
  The image is imported by "cut-and-paste".
 *)
```

```
T12 :=
```

```
{T9 = ColorConvert[T12, "Grayscale"]}
```



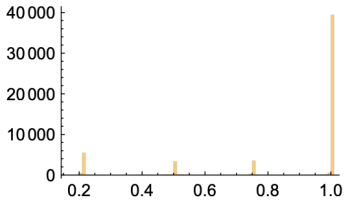
```
(* Problem (9) Binarize image T9 using a histogram and binarization
 with the two thresholds so that only one object appears at the time *)
```

```
(* Flatten[ImageDatap[]] *)
```

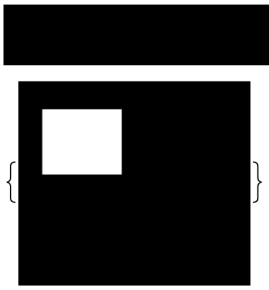
```
T9F := Flatten[ImageData[T9]]
```

```
(* Find the histogram *)
```

```
HP9 = Histogram[T9F, {0.01}]
```



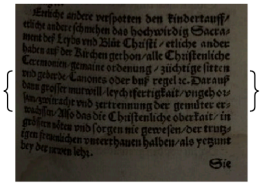

```
(* Use  Binarize[image,{t1,t2}] *)
```



```
(* Turning the gray level up and down *)
```

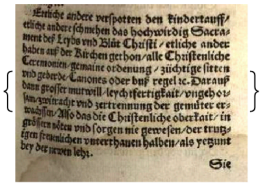
```
(* Gray level down by 0.5 *)
```

```
{Tdown = ImageMultiply[T5, 0.5]}
```



```
(* Gray level up 1.5 with the automatic adjustment to [0,1] *)
```

```
{Tup = ImageMultiply[T5, 1.5]}
```



```
(* Problem (10) Turn the gray level up by 1.5 of the Blue
 component of image mona.jpg  using ImageMultiply[image,a] *)
```

```
(* Read *)
```

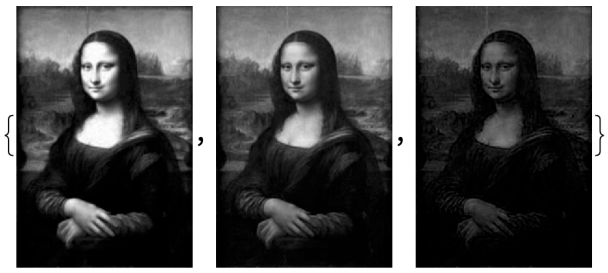
```
Mona := Import["mona.jpg"]
```

```
(* RemoveAplhaChannel in case your hardware generates the forth channel *)
```

```
(* Color separate *)
```

```

```



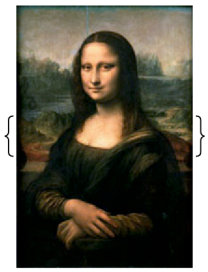
```
(* Turn up the grey level *)
```

```
{MonaBnew = ImageMultiply[MonaB, 1.5]}
```

```
(* Combine into a new image *)
```

```

```



```
(* Compare *)
```

```
(* ImageApply applies a function f to each pixel in the image *)
```

```
(* Problem (11)  Apply a non-linear point operation given by
   f1[x_]:=1/2(1+1/Tan[a*Pi/2]*Tan[a Pi(x-1/2)])
   to mona.jpg  *)
```

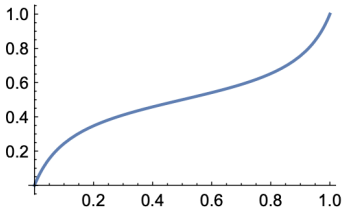
```
(* Define the transformation function *)
```

```
a := 0.8
```

```
f1[x_] := 1 / 2 (1 + 1 / Tan[a * Pi / 2] * Tan[a Pi (x - 1 / 2)])
```

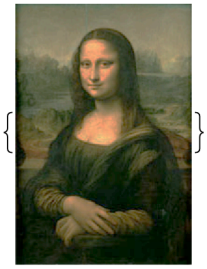
```
(* Show f1[x]  *)
```

```
Plot[f1[x], {x, 0, 1}]
```




```
(* Apply f1 to the image *)
```

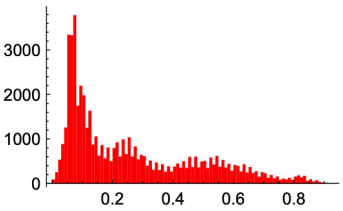
```
{MonaNew1 = ImageApply[f1, Mona]}
```

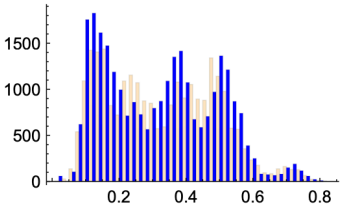


```
(* Compare *)
```

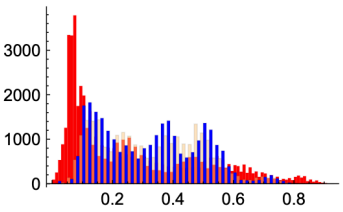
```
{Mona, MonaNew1}
```

```
(* Convert to the gray level and compare histograms of Mona and MonaNew1  *)
```





```
Show[HM, HMnew]
```

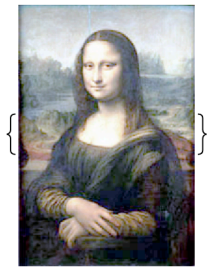



```
(* HistogramTransform performs the histogram equalization. Equalization is
 used a to enhance the image features. The histogram of the equalized
 image is almost flat i.e. every gray is equally represented *)
```

```
(* Problem (12)  Apply HistogramTransform to mona.jpg  *)
```

```
(* Apply HistogramTransform to mona.jpg  *)
```

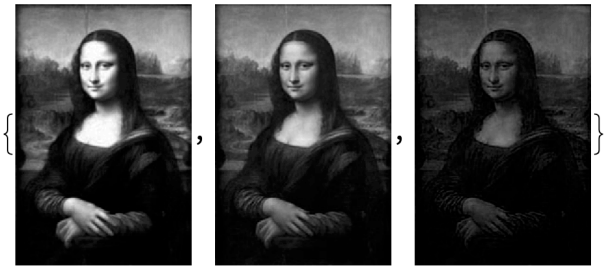
```
{MonaEq = HistogramTransform[Mona]}
```



```
{MonaRE, MonaGE, MonaBE} = ColorSeparate[MonaEq]
```

```
(* Equalize the Green Matrix of Mona Lisa and apply function
f2[x_]:=
 x+x(1-x)*a to the Blue Matrix. a=1. Compose and save the resulting image *)
```

```
(* Separate *)
```



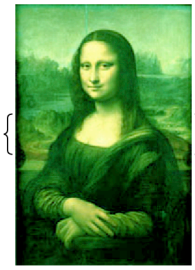
```
(* Equalize the Green Matrix  *)
```

```
(* Apply f2 to the Blue Matrix *)
```

```
1
```

```
{MonaBnew = ImageApply[f2, MonaB]}
```

```
(* Combine the matrices back into the color image  *)
```



```
(* Adding, subtracting and multiplying images *)
```

```
(* Subtracting *)
```

```
(* Problem (12). Find the difference between pill1.jpg and pill2.jpg *)
```

```
P1 = Import["pill1.jpg"]
```

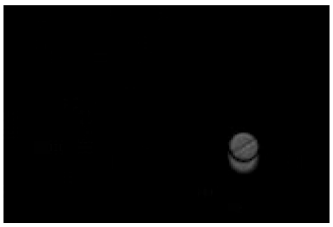
```
P2 = Import["pill2.jpg"]
```

```
(* Convert to the gray level *)
```

```
{PG1 = ColorConvert[P1, "Grayscale"], PG2 = ColorConvert[P2, "Grayscale"]}
```

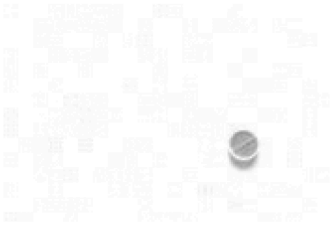
```
(* Let us depict the difference image. The light areas correspond
 to the motion whereas the black color indicates that no changes
 have occurred during the time period between the two images *)
```

```
DP1P2 = ImageDifference[PG1, PG2]
```



```
(* ColorNegate  *)
```

```
Result = ColorNegate[DP1P2]
```



```
(* Problem (13). Use ImagePartition and
 ImageDifference to find the difference  between the pirate
 images in pirat.gif Use the solution of Problem 12 *)
```

```
(* Apply RemoveAplhaChannel in case your hardware generates the forth channel *)
```

```
{Pirat1 = Import["pirat.gif"]}
```

```
{Pirat = RemoveAlphaChannel[Pirat1]}
```

```
DimP = ImageDimensions[Pirat]
```

```
{300, 180}
```

```
xPir = DimP[[1]]
```

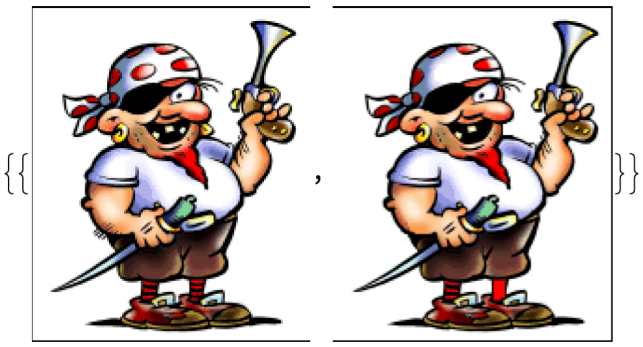
```
300
```

```
yPir = DimP[[2]]
```

```
180
```

```
(* Split vertically at x=Xpir/2 , 300/2=150 *)
```

```
PiratL = ImagePartition[Pirat, {xPir / 2, yPir}]
```



```
(* if your image is too big,
use Magnify. Understand the index [[1]][[1]] and [[1]][[2]] *)
```

```
{Magnify[PiratL[[1]][[1]], 0.5], Magnify[PiratL[[1]][[2]], 0.5]}
```

```
Part1 = PiratL[[1]][[1]]
```

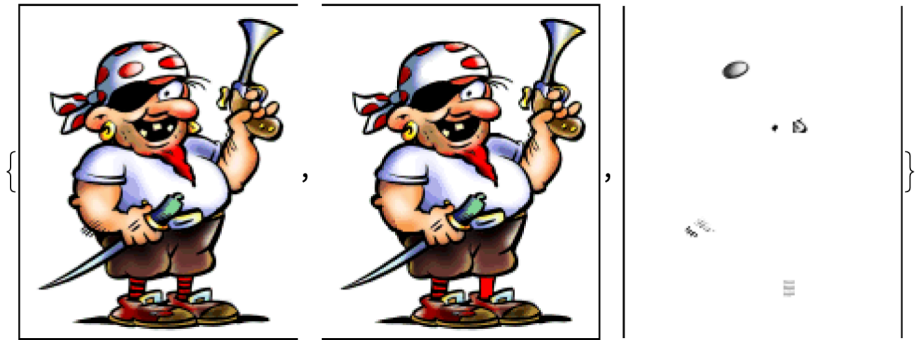
Part2 =

```
(* Find the difference *)
```

ImageDifference

```
(* ColorNegate *)
```

```
(* Compare *)
```



```
(* Answer:
 The differences are (1) Hair on the left arm (2) Red dot on the bandana
  (3) Left earring (4) Two vs three teeth (5) Stripes on the left sock *)
```

```
(* Problem (14) Analyze the code to
 morph two images. Bush.jpg and Arnold.jpg" *)
```

```
{Bu = Import["cool_cow.jpg"]}
```

Import : File cool _cow.jpg not found during Import.

```
{$Failed}
```

```
Ar := Import["arnold.jpg"]
```

```
(* Note ImageResize *)
```

```
Morph[t_] := ImageResize[Blend[{Bu, Ar}, t], 100]
```

```
Mor = Table[Morph[t], {t, 0, 1, 0.05}] ;(* precompute 20 frames *)
```

```
{ListAnimate[Mor, AnimationRunning → False]}
```