CS4442b/9542b: Artificial Intelligence II, Winter 2017
Assignment 2: Computer Vision
Due: March 15

Instructions:

- This assignment is to be implemented in matlab.
- Include all code you write and a report with answers to questions. The report has to be in PDF format. You do not need to include code/data that I give you.
- You can submit one zipped folder, but make sure it is under 10MB in size. We will not grade zipped folders larger than 10MG.
- For most questions, I ask you to save an output image to a file with some particular name. In addition to saving, also insert the output image into your pdf report.
- Use the function names, input/output parameters exactly as I specify.I provide all function stubs in folder *FunctionStubs*, please use them. If you write helper functions, you can give them any names you wish.
- Extra credit counts only towards the assignments.
- Submit your assignment through OWL by 11:55pm on the due date.

Image I/O

Typing **help images** lists matlab functions for handling images. For this assignment you are allowed to use only the image handling functions described in this section. You are not allowed to use any other matlab functions specialized for images, unless you get my explicit permission.

To load an image into matrix **A**, use **A = imread('filename.jpg')**. If the image is grayscale, then **A** has 2 dimensions. To get image width and height, use **[r,c] = size(A)** . Image height (the number of rows) is **r**. Image width (the number of columns) is **c**. The top left pixel of an image is **A(1,1)**, the bottom right pixel is **A(r,c)**. In the statement **A(i,j)**, the valid range for (i, j) is $1 \leq i \leq r$, and is $1 \leq j \leq c$.

For colored images, **A** is a 3 dimensional matrix. The third dimension corresponds to the red, green, and blue color channels. **A(i,j,1)** is the red channel of pixel **(i,j)**, **A(i,j,2)** is the green channel, and **A(i,j,3)** is the blue channel. If you want to separate, say, the red channel of the image image into a two dimensional array, use command **R= A(:,:,1)**. Now **R(i,j)** is the red channel of pixel **(i,j)**. Similarly you can get the green and blue components. If you have separated three channels of a color image into two dimensional arrays **R, G, B**, you can put them back together into a three dimensional color image using **A=cat(3,R,G,B)**.

When you load image into array **A**, it will be of type *uint*8, unsigned integer[1]. In matlab, you cannot do much with array **A** if it is of *unit*8 type. Convert it to type *double* by typing **A = double(A)**. Now you perform the usual arithmetic with **A**. To store *double* array **A** as an image, convert it to type *uint*8 using **A = uint8(A)**. You can convert and save with

[1]Command **whos A** will display the size and type of variable **A**.

imwrite(uint8(A),'somename.jpg')[2]. Matlab handles images in most common formats. For example, to save in the *PNG* format use **imwrite(uint8(A),'somename.png')**. To visualize image inside matlab, use **imagesc** or **image**.

Throughout the assignment I specify input and out parameters as 'gray scale image' or 'color image'. It is always assumed that both gray scale and color images are in *double* format, **not** *uint8* format.

Useful matlab matrix manipulation functions for this assignment: **reshape, colon, exp, size, sum, power, zeros**. Allowed matlab image processing functions are: **imread, imwrite, imresize, imagesc, image, padarray**. If you want to use a function not on this list, ask me.

Intro

In this assignment, you will implement seam carving (in Problem 4) and figure-ground segmentation with interactive graph cuts (in Problem 5). In Problems 1, 2 and 3 you will implement some functions useful for seam carving. Problems 6-7 are extra credit. You can use the interactive tool you develop in Problem 5 to segment object masks from images and use those masks in the seam carving algorithm.

Problem 1 (10%)

(a) Write function

outIm= applyFilter(im,F)

that takes as an input gray scale image **im**, a filter **F** and outputs the result of correlating image **im** with filter **F**, i.e. **outIm** = **F** $\otimes$ **im**. The output image **outIm**should have the same size as image **im**. To handle boundary issues, use the clip filter approach, i.e., pad image with 0. Assume that **F** has an odd number of rows and columns. You cannot use any of the matlab built-in methods for linear filtering, correlation, convolution, etc. You have to write the correlation code from scratch.

(b) Apply filter

$$F = \begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}$$

to image **'swan.png'** and report the sum of absolute values in the output image. What is this filter useful for?

Visualize the result using function **stretch** that I provide. Stretch takes a two dimensional array of type double and linearly stretches its values into the range $\{0, 1, ..., 255\}$. Save your filtered image (after it goes through my stretching function) as **'swanFiltered.png'**. When saving, do not forget to convert to *uint*8 type.

[2]You can actually save an array of type *double* as image directly with **imwrite** command. However, in this case, matlab assumes that your image values are in the range from 0 to 1 and clips all the values outside this range. So if your double image has values from 0 to 255 and you save it without converting to *uint*8, the saved image might be all black or white because of this clipping. If you first divide the double image by 255 and then save it, the results will look ok.

Problem 2 (5%)

(a) Write function

$$\textbf{eng = computeEngGrad(im,F)}$$

that takes as an input a color image **im** and outputs a gray scale image **eng**, which is equal to the gradient magnitude of the input image. Compute the horizontal gradient component based on filter F and the vertical gradient component based on the transpose of F.

Since the input is a color image, first convert it to grayscale by adding the three color channels and dividing the sum by 3. Do not use matlab function **rgb2gray** as it rescales double images. Let the resulting image be called **imG**. The energy image (i.e. the gradient magnitude image) is computed as:

$$eng = sqrt(|F \otimes imG|^2 + |F^T \otimes imG|^2)$$

where raising to power of two and the squared root function is applied to each matrix element individually.

It is convenient to use **applyFilter** function from Problem 1.

(b) Apply the function you develop in (a) to image **'face.jpg'** with

$$F = \begin{bmatrix} 1 & 3 & 0 & -3 & -1 \end{bmatrix}.$$

and report the sum of all values in the **eng** image. Also save your output image as to **'faceEngG.jpg'** after putting through the function **stretch** that I provide.

Problem 3 (5%)

(a) Write function

$$\textbf{eng = computeEngColor(im,W)}$$

that takes as an input a color image **im**, a vector **W** of size 3 for color based energy, and and outputs color-based energy image **eng**. Vector **W** gives the weights for the color components. Specifically, if im_R, im_G, im_B denote the color components of image im, the output of your function should be

$$eng = W(1) \cdot im_R + W(2) \cdot im_G + W(3) \cdot im_B.$$

(b) Apply the function you develop in (a) to image **'cat.png'** with $\mathbf{W} = [-3 \quad 1 \quad -3]$. Report the sum of values in the energy image. Save the output as **'catEngC.png'** after putting it through the function **stretch** that I provide.

Problem 4 (55%) Seam Carving

In this problem you are to implement the seam carving algorithm. The energy is based on gradient, color, and image mask. When resizing an image, a mask should be resized as well. To implement this efficiently, most of the subroutines you are to develop will work with an image that has four channels: the first three are the usual color channels, and the last channel is the mask. For example, say the three channels R, G, B of the color image are:

11	12	13
14	15	16
17	1 8	19

21	22	23
24	25	26
27	28	29

31	32	33
34	35	36
37	38	39

and the mask M is:

1	0	-1
0	0	1
-1	1	1

To create an image with 4 channels, concatinate them along the third dimension with matlab command **cat**$(3, R, G, B, M)$. This will create the following matrix with 4 channels:

11	12	13
14	15	16
17	1 8	19

21	22	23
24	25	26
27	28	29

31	32	33
34	35	36
37	38	39

1	0	-1
0	0	1
-1	1	1

Suppose the above matrix with 4 channels is called $im4$. To access the color of pixel (r, c), use $im4(r, c, 1:3)$. To access the mask of pixel (r, c), use $im4(r, c, 4)$.

To separate the color image from the 4-channel image, use $im = im4(:, :, 1:3)$. Matrix im is now the original color image. To separate the mask from the 4-channel image, use $m = im4(:, :, 4)$. Matrix m is now the original mask. In this assignment, whenever I refer to the 4-channel image, I mean this combination of color and mask matrix.

(a) (5 %) Implement function

eng = computeEng(im4, F, W, maskW)

that takes as an input 4-channel image **im4**, a filter **F** to use for gradient energy, a weight vector **W** to use for color energy, and the weight of the mask **maskW**. This function should compute and return energy for seam carving that is the sum of gradient, color, and mask energies. For gradient and color energies, you should reuse your previously implemented functions. The mask energy is simply the **maskW** multiplied by the mask channel. In formula,

$$eng = computeEngGrad(im4(:, :, 1:3), F) + computeEngColor(im4(:, :, 1:3), W) + maskW * im4(:, :, 4)$$

(b) (5 %) Implement function

imOut = removeSeamV(im4,seam)

that takes as an input a 4-channel image **im4**, a vertical **seam** and returns the input 4-channel image with the seam removed. Recall that a vertical **seam** is a vector of the same size as image height. That is the length of vector **seam** is equal to the number of rows in **im4**. Also, **seam(i)** specifies the index of the column to remove in row **i** of image **im4**. Specifically, for each **i**, pixel **(i,seam(i))** is to be removed. If **im** has **r** rows and **c** columns, the output image **imOut** has **r** rows and **c-1** columns. Both **im4** and **imOut** are 4-channel images. So when you remove pixel **(i,seam(i))**, you must remove all of its 4 channel values.

(c) (5 %) Implement function

imOut = addSeamV(im4,seam)

This is the same function as in part (a), but now a vertical **seam** is added to **im4**. If **im4** has **r** rows and **c** columns, the output image **imOut** has **r** rows and **c+1** columns. Both **im4** and **imOut** are 4-channel images. So when you add pixel **(i,seam(i))**, you must add all of its 4 channel values.

(d) (5 %) Implement function

[M,P] = seamV_DP(E)

that takes as an input a gray scale energy image **E**, and performs dynamic programming for finding the optimal vertical seam. This function should return arrays **M** and **P** that are constructed during dynamic programming. The sizes of **M,P,E** are equal. If during dynamic programming the smallest cost path is not unique, you should chose the leftmost column for the path[3].

(e) (5 %) Implement function

[seam,c] = bestSeamV(M,P)

that takes as an input arrays **M,P** constructed in part (c) above, and computes and returns the best **seam** and its cost **c**.

(f) (5 %) Implement function

[seam,im4Out,c] = reduceWidth(im4,E)

that takes as an input a 4-channel image **im4**, its corresponding energy image **E** and reduces the width of the input image by one column, using the functions you have implemented above. Namely, you should find the best vertical seam and remove it from **im4**, thus reducing the width. The output should be the removed **seam**, the input 4-channel image reduced in size by one column, and the cost of the removed seam **c**.

[3]That is, following the pseudo-code notation in the lecture notes, if there is no unique smallest number among $option_1, option_2, option_3$ option with the smallest subscript should be preferred

(g) (5 %) Implement function

$$\textbf{[seam,im4Out,c] = reduceHeight(im4,E)}$$

which is exactly the same as the one in (e) except it reduces the width of the image by removing one row. For implementation, I suggest transposing[4] the input 4-channel image and the energy E after which you can reuse **ReduceWidth** function you have implemented above. Note that the image should be transposed twice, before **ReduceWidth** and after **ReduceWidth**.

(h) (5 %) Implement function

$$\textbf{[seam,im4Out,c] = increaseWidth(im4,E)}$$

that is the same as **reduceWidth** in part (e), except the image width is increased.

(i) (5 %) Implement function

$$\textbf{[seam,im4Out,c] = increaseHeight(im4,E)}$$

that is the same as **reduceHeight** in part (f), except the image height is increased.

(j) (5 %) Reusing the functions you have developed in parts (a-i), implement function

$$\textbf{[totalCost,imOut] = intelligentResize(im,v,h,W, mask,maskWeight)}$$

that takes input color image **im**, the number of vertical seams to process **v**, the number of horizontal seams to process **h**, the weight vector **W** for the color energy, and the **mask** image with its weight **maskWeight**. The sign of **v** and **h** signal insertion/removal of seams. Positive sign means seam insertion, negative seam removal.

Always process seams in alternating order, starting with a horizontal seam. For example, if **v** = 4 and **h** = 2, then seam processing is done in order: horizontal insert, vertical insert, horizontal insert, vertical insert, vertical insert, vertical insert. If **v** = -2 and **h** = 4, then seam processing is done in order: horizontal insert, vertical remove, horizontal insert, vertical remove, horizontal insert, horizontal insert.

For gradient energy, use filter $F = [-1, 0, 1]$. You should first construct the 4-channel image **im4** by concatenating the color image with its mask, as explained in the beginning of the problem description. Then you should compute energy E and carve out the first seam. Recompute E for the new reduced/enlarged **im4** after each seam carving operation[5].

The output of your function should be the sum of all seams carved and the output carved color image.

If you wish, you can record a seam removal/insertion movie, using **videowriter** command. You can paint in red the inserted/removed seam using output parameter **seam** that functions **increaseHeight,decreaseHeight,increaseWidth,decreaseWidth** return.

[4]Transposing an image will change columns into rows and transposing back returns rows into columns.

[5]You you want more efficiency, you can recompe E only around the changed part of the 4-channel image, i.e. around the location of the carved out seam.

(k) (5 %) Apply **intelligentResize** to **'cat.png'** with **v=-20, h= -20**, **W=[1, -2, 1]**, and zero mask (i.e. mask has no effect on the energy). Save the output as **'catResized.png'**. Report the total cost of all seams.

Apply **intelligentResize** to **'face.jpg'** with $v = -20, h = -20$, **W=[1, -2, 1]**, mask in image **'faceMask.png'** and **maskWeight = -100**. Save the output as **'faceResized.png'**. Report the total cost of all seams.

Generate 1-3 examples of seam carving with your program and include them in the report. Show only the final result. Use either your own images or my images with new parameter setting for seam carving. Describe the energy you used and how many vertical/horizontal seams were removed/added.

(l) (0 %) Just for fun. Submit one of your seam carving results for the best result competition.

Problem 5 (25% and 5% optional extra credit) Interactive Figure-Ground Segmentation

You will use interactive segmentation tool **brushStrokes_gui** for this problem. Invoke it from matlab command line by typing **brushStrokes_gui**. Use **load Image** button to load an image[6]. Paint foreground/background strokes using mouse and buttons **FG label**, **BG label** for switching between the stroke types. Brush size can be changed by using the scroll down **Scribble Radius** menu. You can use buttons **Clear Curr Stroke** and **Clear All Strokes** to remove strokes. Use **Segment** button to invoke the **segmentGC** function that performs segmentation. You are to implement this function. After it finishes, a new figure window will pop showing the foreground object against the black background.

In folder **ToStudents\Code\forFG** there are many files needed to support computation of the minimum cut. For the program to work, you need all of these files. However, you only need to know/use the following two:

- **segmentGC.m** This is the function to fill in with the code for figure-ground segmentation. I created this file so that **brushStrokes_gui** GUI does not crash on invocation. Currently **segmentGC** returns whole image as the foreground object. Fill it in with your segmentation code, see part (a) below.
- **[labels,eng_start,eng_finish] = solveMinCut(dataB,dataF,W)** This is function for computing the minimum cut on a graph. You should use it, but do not make any changes to it. I explain how to use this function in Figure 1. Also see comments in the file **solveMinCut.m** itself.

(a) (20 %) Fill in implementation for function

[segm,eng_finish] = segmentGC(im,scribbleMask,lambda,numClusters,inftyCost)

that takes a color input image **im**, foreground/background scribbles **scribbleMask**, the strength of regularization **lambda**, number of clusters for kmeans **numClusters**, and cost to

[6]Please note that only color images can be processed with **brushStrokes_gui**.

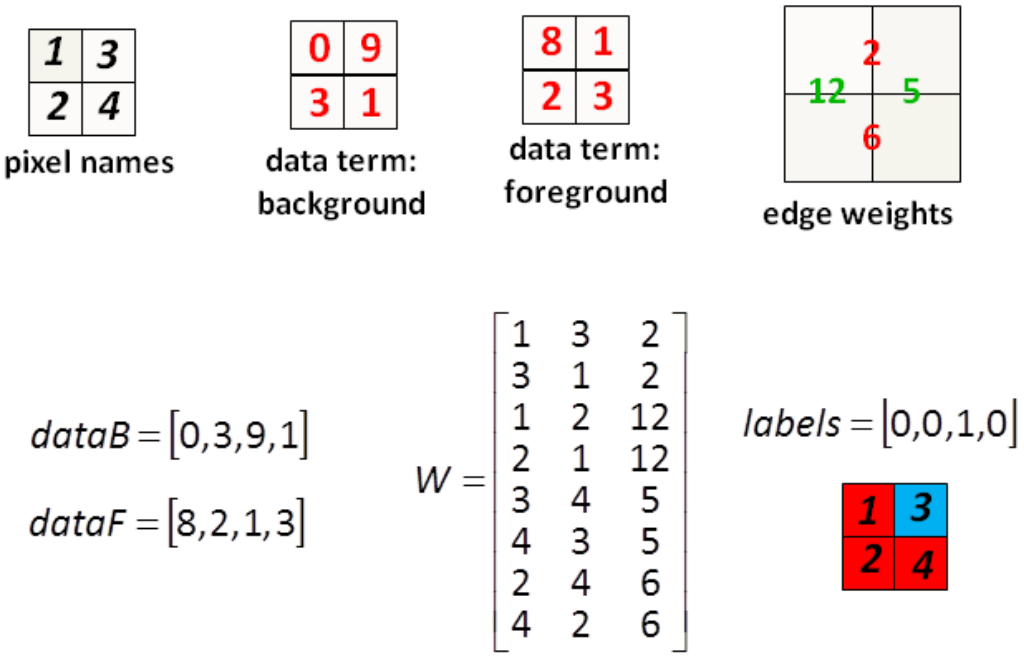

Figure 1: First row: pixel indexes, data terms for background and foreground, and the edge weights between neighboring pixels. Second row: the input to function **solveMinCut(dataB,dataF,W)**. Each row of **W** specifies the graph edges. The rows of **W** can be in any order but column order has to stay the same. That is the order, in each row is: node i, node j, the weight of the edge between nodes i and j. Note that all edge information has to be repeated twice in **W**, once for each direction. For example, the edge between pixels **1** and **3**, which has weight **2**, appears in the first row (from **1** to **3**), and in the second row (from **3** to **1**). Arrays **dataB, dataF, labels** follow the order imposed by pixel names. In this example, the image is indexed in the column-wise manner. But you can index it in row-wise manner, if you wish. As long as pixel names are consecutive integers from 1 to n, where n is the total number of pixels, any indexing will work. The optimal segmentation corresponding to the output **labels** is shown on the bottom right. Blue means the foreground and red background. The output energy values are **eng_start = 13**, **eng_end = 12**.

set the infinity links to **inftyCost**. In **scribbleMask**, foreground seeds have value 2, background seeds have value 1, and the rest of the pixels are 0. This function should compute interactive figure-ground minimum cut segmentation and output the segmentation mask **segm** as well as the final energy **eng_finish**. The size of **segm** is the same as the input image size. It should have two values: 1 for foreground pixels and 0 for background pixels.

If the input parameter **numClusters** is set to a value bigger than 0, you should use kmeans to quantize image into **numClusters** color clusters and implement histogram based color models for the foreground/background. Smooth histograms by adding 1 to every bin count, as explained in class. Otherwise (if **numClusters** = 0), use just the infinity constraints corresponding to the user scribbles for the data terms. Input parameter **inftyCost** specifies the large value you should set your infinity cost links to. I recommend to keep **inftyCost** around 1000, otherwise the graph cut library might have overflow issues for larger images.

You can use the matlab built in function **kmeans** that computes kmeans. Also helpful commands to use are **clusters** and **hist**.

You are to implement contrast-sensitive weights

$$w_{pq} = lambda \cdot e^{-\frac{|f(p)-f(q)|^2}{2\sigma^2}}.$$

Here *lambda* is set to the input parameter **lambda** and σ^2 is computed as the average of squared intensity differences between neighboring image pixels.

If you implement data costs just based on user interaction (without kmeans based models), you will get 5% off this assignment part, i.e. maximum score is 15 instead of 20. In this case, when input parameter **numClusters** is set to a value bigger than 0, your program should terminate with a message that kmeans clustering is not implemented, to alert the teaching assistant. You will get 5% extra credit if you implement this function without loops (whether you are using kmeans clustering or not).

You can experiment with your program interactively using **brushStrokes_gui**. This program calls **segmentGC_internal** each time the user presses "segment" button. Inside **segmentGC_internal**, parameters **lambda, inftyCost, numClusters** are loaded from file "param.mat" if this file is present, or set to some default values. Then **segmentGC_internal** calls your function **segmentGC**. Therefore, if you want to change parameters **lambda, inftyCost, numClusters** from my default values to your own, you should store them in file "param.mat". Use command **save('param.mat','lambda','numClusters','inftyCost')** to save just the parameters you need in file "param.mat".

(b) (5%) Save labeling results into file "faceL.png" after you run your program on image "face.jpg" with parameters: segmentGC(im,scribblesFace,50,50,1000). Here scribblesFace is provided in file "q5.mat". Also report the final energy. If you do not implement kmeans clustering data terms, provide results for segmentGC(im,scribblesFace,50,0,1000).

Save labeling results into file "liftL.png" after you run your program on image "lift.jpg" with parameters: segmentGC(im,scribblesLift,50,50,1000). Here scribblesLift is provided in file "q5.mat". Also report the final energy. If you do not implement kmeans clustering data terms, provide results for segmentGC(im,scribblesLift,50,0,1000).

If your code runs very slowly on 'face.jpg' and 'lift.png', use the smaller versions of these images, 'faceSmall.jpg' and 'liftSmall.jpg'. Use scribblesFaceSmall and scribblesLiftSmall as

well in this case. Make sure to indicate in your assignment report if you are using smaller images.

Problem 6 Extra Credit (10%)

Develop an interseting modification of the seam carving algorithm. For example, you can think of a new term for the energy, or improve the repetitive-looking results that our current seam insertion procedure tends to produce, etc. Implement your idea and illustrate/discuss the new image generation results of your modified seam carving that were not possible before. Chose a meaningful function name and input/output parameters as you wish. Explain what the input/output parameters are.

Problem 7 Extra Credit (20%)

Write function

disp = stereoCorrespondence(left,right,wSize,maxDisp)

that performs stereo correspondence based on integral images and window matching. The inputs are the **left** and the **right** images of the stereo pair, **wSize** is the window size for the matching, **maxDisp** is the largest possible disparity to consider. In the report, show your results on the four stereo pairs I provide. For visibility of the output, you may want to use the function **stretch** for the disparity map.

Problem 8 Extra Credit, CNN (20%)

Download and install ConvNet matlab package from `http://www.vlfeat.org/matconvnet/`, version `1.0-beta23`. You should follow the installation instructions. If you have trouble with MEX (compiling C++ code in matlab), you can take files in `ToStudents\Code\CNN_CodeToStudents\Mex` directory and copy them into your directory `matconvnet-1.0-beta23 \matlab\mex`. I provided you with the code that trains a basic CNN network on the training data and evaluates the result on test data on MNIST dataset.

The main function to call is `cnn_run`. Before you call it, change directory names (the first two lines of file `setup.m` to the correct that have the data on your computer in file `setup`. During training, the code displays "traintop1err" and "traintop5err". You should be looking at "traintop1err" as classification error.

Change the network structure by adding more stages to get a better classification rate. The network structure is built in file `initializeCNN_basic.m`. This is the file you should change to build a better network structure. In this file, you can also change all the important meta parameters of the network such as momentum, learning rate, etc. You should hand in this file, as well as describe in the report the changes you made and the test error on MNIST dataset. You should also run your new network on the competition data and report the result. I will reopen the digit competition on the Kaggle Web site.