Project Group Members: Hal Brynteson and Farah Kamleh

CS 415 Computer Vision Homework 1

Hal Brynteson

1. Image Filtering (Leads: Hal and Farah)

    - a) The 4 Gaussian filters are separable filters, as are the first and second order derivatives that are oriented vertically or horizontally. This would make filters 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 37, 38, 39 all separable. This is proven by the code following the markdown cell that says "Examining Separable Filters". For each filter, I check the rank with the following code:

```
    if(np.linalg.matrix_rank(F[:,:,i]) == 1):
        printout = "filter %d is separable, with a rank of %s" % (i, 
np.linalg.matrix_rank(F[:,:,i]))
        print(printout)
```

    - b) Filters are applied to grayscale images in the cell following the markdown cell that reads "Loop and apply all filters to all images - Grayscale". This cell loops through every filter for every image, calling the `makeResponses()` function once per image.

    - c) The L1 norms are calculated in the same cell where filters are applied. In the loop that runs for each filter, the `computeMagnitude()` function is called, and the computed norm for that filter is stored in a 2D `norms` array.

    - d) Grayscale Thresholding: For each filter, I computed a threshold value that would split the data where everything greater than this value is a mammal, and everything less than is a bird. For this, I took the norm of each filtered image, then split them into bird_norms and mammals_norms. If the largest bird_norm was less than the smallest mammal_norm, I took the average of these two values, and set that as the threshold for that filter. Otherwise, the minimum mammal_norm would be the threshold. With this method, filters 14, 15, and 16, all first order derivative filters, were able to get the best classification. With their associated threshold values, they were able to correctly identify all 8 images. This classification method is based on the assumption that the bird norms are less than the mammal norms which, for filters 0-35, is true. The final list of thresholds is: 1.0086085931834663, 1.0093817984046412, 0.9774139212286351, 0.8598662373049294, 0.8802130294773347, 0.9491701758731731, 0.7028308556925308, 0.7109835025380711, 0.7020621827411168, 0.6443844438939806, 0.6549764676740154, 0.6850371646120377, 0.5396174764321973, 0.5543863306744018, 0.5317370624299558, 0.5117107010156057, 0.5091678721823136, 0.532532632342277, 0.32164612037708484, 0.314537708484409, 0.2664899677978697, 0.27084468664850136, 0.26616299232103047, 0.3038179840464104, 0.09402374909354605, 0.08796135744364628, 0.07597225662620757, 0.0754520683675997, 0.06785236561803319, 0.08478077780530097, 0.029465192168237852, 0.02482596084118927, 0.022681290790427848, 0.028942168237853515, 0.023249938072826357, 0.023445431472081217, 201.0512558830815, 221.17237218999276, 228.6442667875273, 234.59520123277738, 148.798399709038, 77.58124931805783, 51.72440443717039, 38.76909256228405, 51.72440443717039, 23.831234769958176, 7.569214402618658, 0.5092562284051646, 618140.0 * The final threshold is for the summed response.

- e) Repeat b-d for RGB Filters are applied to RGB images in the cell following the markdown cell that reads "Loop and apply all filters to all images - RGB". This cell loops through every filter for every image, calling the `makeResponses_rgb()` function once per image. The L1 norms are calculated in the same cell where filters are applied. In the loop that runs for each filter, the `computeMagnitudeRGB()` function is called, and the computed norm for that filter is stored in a 2D `norms` array.

  RGB Thresholding: The best classifier is no longer the same. Using the same methods, my threshold values were different, and none of them were able to perfectly classify the group of eight images. This pass was, however, a lot more consistent. Filters 0-35 were nearly entirely correct, and classified everything correctly except for the first bird image. The final filters, 36-47, classified everything as mammals. The final list of thresholds is: 3.1209191443074693, 3.1234109862219, 2.9566955660143672, 2.6000941293039386, 2.6411890017339608, 2.842893237552638, 2.158105511240029, 2.176333393763597, 2.1488769035532993, 1.9431904879861284, 1.958563289571464, 2.116092277012328, 1.6447951414068165, 1.6819760696156636, 1.680084300217549, 1.6035868218974485, 1.6069804310131286, 1.6409662799129805, 1.0083272569444444, 0.9454446371067624, 0.8015011146891255, 0.8227198414664355, 0.8020708446866485, 0.9168255208333334, 0.29846718636693254, 0.26310131285608124, 0.22790686153083972, 0.2277235570968541, 0.20333911320287343, 0.2522764429031459, 0.09607777374909354, 0.08144670050761421, 0.07302121102248006, 0.09179024655547498, 0.06983403517463463, 0.07443437273386512, 561.8210420076377, 656.3985360768673, 685.4718772661349, 705.4531236403191, 397.66315148208764, 210.9112965993817, 140.61666848517913, 105.39749408983451, 140.61666848517913, 64.78721767594108, 20.57806146572104, 1.411656664848154, 0, 3832.031067466812 The final threshold is for the summed response.

- f) Non-separated filters took 23.1 seconds, while computation with the separated filters took 17 seconds. Computation with separable filters is faster. The vectors for each separable filter are created in the cell that follows the markdown cell that read "Comparing Separable vs. Non-Separable compute times". They are stored in `filter_vectors` and applied in the `makeResponses_sep()` function, which is called in a loop over each image.

2. Histogram Equalization (Lead: Hal)

   - a) In my first pass, a histogram is created by the `l_count_per_pixel()` function. Later, to examine the effect of changing the bin number, I switch to using openCV's histogram function, which I call in my own function, `opencv_hist()`. This is placed in its own function to allow me to optionally normalize.

   - b) Given then I is a probability density function, shifting it would affect the the overall intensity of the image. A shift left would alter the probability that a pixel is darker, so by shifting the majority of pixels to the left, the image, I, would get darker. On the other hand, a shift to the right would increase the probability of high pixel intensity, creating a brighter image. If p is stretched, the contrast of the image will decrease, as the majority of the pixels will fall towards the middle intensity. If p is squashed, the contrast of the image will increase, as the probability shifts so that more pixels will be near maximum and minimum intensity.

- c) Using either my own `create_chistogram()` function, or by simply taking the `.cumsum()` of my histogram, I calculate the CDF. This CDF is then used to create the `transform_map` array, which is used to transform the image. The following code shows the creation of the transform map from the CDF, taking into account smaller bin sizes:

```
#for every possible l value
for i in range(0, 256):
    transform_map[i] = 255 * chistogram_array[chist_index]

    chist_iter = chist_iter + 1
    if(chist_iter >= vals_per_bin):
     chist_iter = 0
     chist_index = chist_index + 1
```

- d) To examine the effect of the number of bins, I implemented openCV's histogram function in place of my own. Then, for all the given images, I equalized the image's histogram for 16, 32, 64, 128, and 256 bins. I noticed that a lower number of bins caused noticable artifacting in the output image, with blocky edges between changes in intensity/value.

- e) The transform done with transformation_map T is monotonic. For every element in the transformation map, starting with element 0, that element is either less than or equal to the element that follows it. Thus, it is monotonic. This is shown in the cell following the markdown cell that reads "Monotonic Proof". In the following code, I loop through the transform_map and, if the current element is ever larger than the element that follows it, I set the a flag, and break from the loop. When the code is run, the flag is never set to false, and the mapping is a monotonic transformation.

```
# If the map element
for i in range (0,len(transform_map)):
    if(transform_map[i] > transform_map[i]):
        monoFlag = False
        print("Map is not monotonic")
        break
```

3. Generalized Patch Extraction (Lead: Hal)

- a) Given a starting pixel `center`, the `axis_aligned_patch(image, center, height, width)` function will take a patch of `image` defined by a bounding box that is `height` tall and `width` wide, centered on the starting pixel `center`.
- b)
  - i) Translating origin to p, where p is the center pixel, and origin is [0,0], simply requires adding px to origin x, and py to origin y. So, the translation vector would be [px, py].
  - ii) The rotation matrix to rotate θ counterclockwise would be as follows:

```
cosθ   -sinθ
sinθ   cosθ
```

- iii) The composite matrix to translate the origin to p and rotate θ would be as follows:

```
cosθ  -sinθ px
sinθ  cosθ  py
0     0     1
```

- c) Rotated image is moved to a larger canvas by the `rotate_bound()` function
- d) The `rotate_bound_scale()` function takes a scaling factor and applies a uniform scale to the scaling matrix. This is done by multiplying the scaling factor to the terms in cells [0,0] and [1,1] in the transformation matirx. This is done because a scaling matrix that scales vector [x,y] by factor s is as follows:

```
sx 0
0  sy
```

4. Perspective Projection (Lead: Hal)

- a) A circular disk parallel to the image plane would appear circular.
- b) The equation to get the image plane coordinates would be pt = [-(xt + ut) / (zt + wt)] and qt = [-(yt + ut) / (zt + wt)]. The velocity can be found by taking the difference between 2D points. So, velocity up = pt = p0, and vq = qt - q0.
- c)
  - i) When the point is at a maximum, the velocity will equal 0. So, we can fill the v(t) = v - 2gt equation where v = 0 and rearrange to get t = v0 / 2g. This will give us the time where the velocity becomes 0. [p,q] could then be found by converting [xt, yt, zt].
  - ii) The call hits the ground when y = 0. To get the y position from velocity, integrate the velocity equation to get yt = y0 + v0t - g(t^2). Solving for t, we get t = (-v0 ± sqrt(v0 - 4gy0)) / 2g. So, given the starting world position and starting velocity, we could find the time where the ball hits the ground, and get the [p,q] coordinates by converting the world coordinates.
- d) Since this set up models the movement of a ball in motion, the current set up is sufficient.