ECE549 / CS543 Computer Vision: Assignment 0

Instructions

1. Assignment is due at **11:59:59 PM on Monday Feb 3 2020**.

2. Course policies: http://saurabhg.web.illinois.edu/teaching/ece549/sp2020/policies.html.

3. Submission instructions (tentative, will be finalized at most by early next week):

   (a) A single `.pdf` report that contains your work for Q1, Q2 and Q3. For Q1 and Q2 you can type out your responses in the space provided on pages 2 and 3. Alternatively, you can also print pages 2 and 3, respond by handwriting your responses in the space provided, and scan the pages into a PDF. For Q3 your response should be electronic (no handwritten responses allowed). You should respond to the questions 3a, 3b, 3c and 3d individually and include images as necessary. Your response to Q3 in the PDF report should be self-contained. It should include all the output you want us to look at. You will not receive credit for any results you have obtained, but failed to include directly in the PDF report file. PDF file will need to be submitted to Gradescope, and you will need to tag your PDF with where your response to each of the question is.

   (b) You also need to submit code for Q3 in the form of a single `.zip` file that includes all your code, all in the same directory. You can submit Python code in either `.py` or `.ipynb` format. Code will need to be submitted to compass2g.

1. **Calculus Review [10 pts].**

   The $\mathrm{softmax}$ function is a commonly-used operator which turns a vector into a valid probability distribution, i.e. non-negative and sums to 1.

   For vector $\mathbf{z} = (z_1, z_2, \ldots, z_k) \in \mathbb{R}^k$, the output $\mathbf{y} = \mathrm{softmax}(\mathbf{z}) \in \mathbb{R}^k$, and its i-th element is defined as

$$y_i = \mathrm{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}. \tag{1}$$

   (a) **[3 pts]** Verify that $\mathrm{softmax}(\mathbf{z})$ is invariant to constant shifting on $\mathbf{z}$, i.e. $\mathrm{softmax}(\mathbf{z}) = \mathrm{softmax}(\mathbf{z} + C)$ where $C \in \mathbb{R}$. The $\mathrm{softmax}(\mathbf{z} - \max_j z_j)$ trick is used in deep learning packages to ensure numerical stability.

   (b) **[3 pts]** Let $y_i = \mathrm{softmax}(\mathbf{z})_i, 1 \le i \le k$. Compute the derivative $\frac{\partial y_i}{\partial z_j}$ for any i, j. Your result should be as simple as possible, and may contain elements of $\mathbf{y}$ and/or $\mathbf{z}$.

   (c) **[4 pts]** Consider $\mathbf{z}$ to be the output of a linear transformation $\mathbf{z} = W^\top \mathbf{x}$, where vector $\mathbf{x} \in \mathbb{R}^d$ and matrix $W \in \mathbb{R}^{d \times k}$. Denote $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k\}$ as the columns of W. Let $\mathbf{y} = \mathrm{softmax}(\mathbf{z})$. Compute $\frac{\partial y_i}{\partial \mathbf{x}}$ and $\frac{\partial y_i}{\partial \mathbf{w}_j}$. (Hint: You may reuse (b) and apply the chain rule. Vector derivatives: $\frac{\partial(\mathbf{a} \cdot \mathbf{b})}{\partial \mathbf{a}} = \mathbf{b}$, $\frac{\partial(\mathbf{a} \cdot \mathbf{b})}{\partial \mathbf{b}} = \mathbf{a}$.)

2. **Linear Algebra Review [10 pts].**

(a) **[2 pts]** Let $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. Compute $V \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $V \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. What does matrix multiplication Vx do to x?

(b) **[2 pts]** Verify that $V^{-1} = V^\top$. What does $V^\top x$ do?

(c) **[2 pts]** Let $\Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \end{bmatrix}$. Compute $\Sigma V^\top x$ where $x = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}, \begin{bmatrix} -\sqrt{2} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -\sqrt{2} \end{bmatrix}$ respectively. These are 4 corners of a square. What is the shape of the result points?

(d) **[2 pts]** Let $U = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$. What does Ux do? (Rotation matrix wiki)

(e) **[2 pts]** Compute $A = U\Sigma V^T$. From the above questions, we can see a geometric interpretation of Ax: (1) V^T first rotates point x, (2) Σ rescales it along the coordinate axes, (3) then U rotates it again. Now consider a general squared matrix $B \in \mathbb{R}^{n \times n}$. How would you obtain a similar geometric interpretation for Bx?

3. **Colorizing Prokudin-Gorskii images of the Russian Empire.**[1] **[50 pts]**

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available online.

The goal of this assignment is to learn to work with images by taking the digitized Prokudin-Gorskii glass plate images and automatically producing a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three color channel images, place them on top of each other, and align them so that they form a single RGB color image.

Prokudin-Gorskii's black-and-white (grayscale) image composites, and some other test images are available at http://saurabhg.web.illinois.edu/teaching/ece549/sp2020/mp/mp0-data.tgz. Note that the filter order from top to bottom is BGR, not RGB.

(a) **Combine [5 pts].** Choose a single image from the `prokudin-gorskii` folder (your favorite) and write a program in Python that first divides the image into three equal parts (channels), and takes the three grayscale channels and simply stacks them across the third color channel dimension to produce a single, colored image. We expect this stacked photo to look wonky and unaligned – fixing this is what you will do in the next parts. Save this generated image. Make sure to save your image with the correct channel ordering (it varies depending on what library you are using to save images from Python). **Include the generated image into your report.**

(b) **Align [25 pts].** As you will have noticed, the photos are misaligned due to inadvertent jostling of the camera between each shot. Your second task is to fix this. You need to search over possible pixel offsets in the range of [-15, 15] to find the best alignment for the different R, G, and B channels. The simplest way is to keep one channel fixed, say R, and search over horizontal and vertical offsets that align the G and the B channels to the R channel. Pick the offset that maximizes a similarity metric (of your choice) between the channels. You can measure similarity using (negative of) the *sum of squared differences* which is simply the L2 norm of the pixel difference between the two channels. You can also use *zero mean normalized cross correlation*, which is simply the dot product between the two channels after they have been normalized to have zero mean and unit norm. After writing this function, run it on all of the images in the `prokudin-gorskii` folder. Find a similarity metric that aligns all these images. **Save the newly aligned images in your report, along with the offsets for each color channel. Also include a brief description of your implemented solution, focusing especially on the more non-trivial or interesting parts of the solution. What design choices did you make, and how did they affect the quality of the result and the speed of computation? What are some artifacts and/or limitations of your implementation, and what are possible reasons for them? Hint:** To offset the channels while keeping the same dimensions among them, you can use either `np.roll()` to roll over boundaries, or `np.pad()` for padding.

(c) **Fast Align [20 pts].** For very large offsets (and high resolution images), comparing all the alignments for a broad range of displacements (e.g. [-30, 30]) can be computationally intensive. We will have you implement a recursive version of your algorithm that starts by estimating an image's alignment on a low-resolution version of itself, before refining it on higher resolutions. To implement this, you will build a two-level image pyramid. To do this, you must first scale the triple-frame images down by a factor of 2 (both the width and height should end up halved). Starting with your shrunk, coarse images, execute your alignment from part (b) over the [-15, 15] offset range. Choose the best alignment based on your similarity metric and treat it as the new current alignment. Then in the full resolution images, use this new current alignment as a starting place to again run the alignment from part (b) in the small offset range of [-15, 15]. Run this faster alignment on the `seoul_tableau.jpg` and `vancouver_tableau.jpg` images. **Report the intermediate offset (at the coarse resolution), the next offset at the full resolution, and what the overall total offset was that includes both of these. Also include the aligned images in color in your report. Hint:** If you're struggling, use a different color channel as your anchor!

[1]This assignment was originally designed by Alexei Efros. This version is adapted from adaptations by David Fouhey, and Svetlana Lazebnik.

(d) **Extra Credit [Upto 10 pts].** Implement and test any additional ideas you may have for improving the speed or quality of the colorized images. For example, the borders of the photograph will have strange colors since the three channels won't exactly align. See if you can devise an automatic way of cropping the border to get rid of the bad stuff. One possible idea is that the information in the good parts of the image generally agrees across the color channels, whereas at borders it does not. **In your report describe the improvements you made, include images (if you improved the quality) and supporting metrics (if you improved the speed). You should provide enough implementation details about how you made these improvements.**