CS 415: Computer Vision I

Homework 3

Prof.: Sathya N. Ravi Assigned: 11/05/2021, Due: 12/03/2021

Few reminders:

- Homework is due at the end of day on the designated date on Blackboard.
- No homework or project is accepted in mailbox of instructor.
- You may discuss homework with classmates and work in groups of up to **three**. However, you may not share any code, carry out the assignment together, or copy solutions from any other groups. Discussions between groups should be minimal, verbal during lecture or if appropriate, on Campuswire only. The submitted version must be worked out, written, and submitted by your group alone.
- **Important:** Each question (or subpart) should have a **lead** who has finalized the submitted solutions after discussions within group. The lead should be *explicitly* indicated in the submissions. There can be two leads for any question. The submitted solution is assumed to be verified by the other person, and so the grade is assigned equally.
- All solutions must be typeset (I recommend Latex, Markdown, Word etc., please do not use nonstandard formats) with code attached in MATLAB or Python format whichever is appropriate.
- Your final submission will be a zip folder with a PDF file containing solutions for each question including text, sample figures, calculation, analysis, and general writeup. Code and more experiments including figures should be provided in a separate folders.
- Submitting someone else's work (outside of the group) as your own is academic misconduct. Such cheating and plagiarism will be dealt with in accordance with University procedures (see the page on Academic Misconduct at this link).

Answer the following questions as completely as possible.
Important Note: You are given with 7 videos for this homework as data.

1. **3D Optical Flow.** Derive the gradient constraint equation for optical flow between volumetric images (e.g. CT images) – the image sequence is now $I(x, y, z, t)$.
2. **RGB Optical Flow.** Consider the RGB, that is, $I(x, t) = [R(x, t), G(x, t), B(x, t)]$. Suggest an analogous gradient-constraint equation for optical flow between RGB images. Explain under what conditions the new equation will have: 1) infinitely-many solutions (recall our discussion on aperture problem in lecture); 2) unique solutions; 3) no solution.

3. **Tracking using Optical Flow Estimation in Python.** You will implement, and compare Lucas-Kanade (LK) algorithm in two different settings – Sparse and Dense. Starter code is provided in `lktrack_skeleton.py` (you may use the background subtraction notebook to structure your solution also).

   (a) **Good Features to Track for Spare LK.** Fill in the code to compute good features to track every once in a while including the starting frame. For dense flow, we will do it at every point. For sparse flow, Do not worry about the track length here, it should be filtered using flow estimates since we are working with videos in flow estimation problems.

   (b) **Compute Optical Flow – both forward and backward.** In your sparse LK implementation, you may use cv2.calcOpticalFlowPyrLK. This automatically constructs the image pyramid, so it localizes features both in position and scale space. In your dense implementation, you may construct your own image pyramid or just compute on the highest possible scale. Do not pick the neighborhood to be too small since the Lucas-Kanade Matrix will be singular in these cases.

   (c) **Update features to track using Optical flow estimation in sparse LK algorithm.** You have to make sure that the number of features tracked is less than prespecified length.

   (d) Suppose we are given with two square images I_1 and I_2, explain (not implement) in detail how to do coarse-to-fine (pyramid) optical flow estimation. You may assume that the image sizes from level 1 to 3 are $100, 50, 25$.

   (e) State in your report which of the settings performed the best for each of the sequences given using quantitative comparison. You can compute the error map of your estimate using ground truth flow given in the video.

4. **Background Subtraction in Python with Frame Differencing.** In this problem, we will implement a basic background subtraction algorithm and test its variants in the videos provided to you. Starter code is provided in `bg_sub_starter.ipynb`.

   (a) **Threshold Tracker.** One simple way to determine foreground is to use a predetermined threshold, that is, pixels over a certain threshold are regarded as foreground. That is, the background at time t, B_t is given by, $B_t = \max(I_t, \tau)$ where τ is the threshold. Fill in the code in `my_threshold_detect`.

   (b) **Frame Differencing Trackers.** Notice that the simple thresholding scheme only works if all the objects and regions inside the objects in the foreground satisfy the threshold requirement which is an unrealistic assumption.

   As we saw in the lecture, frame differencing alleviates this problem by using the frame information in an online fashion, that is, $B_t = d\left(I_t, I_{t-1}\right)$ for some appropriate d. In this case, you will choose your threshold based on estimated background image B_t. You can even try using I_{t-k} (and similarly d_k) for some k so you only update B_t every once in a while.

Report your results for three different frame differencing algorithms by choosing d appropriately: (i) Non-adaptive, (ii) adaptive – better suited for changes in illumination, (iii) mean – running averaging as the background. Fill in the code in `my_frame_differencing`.

(c) **Trajectory of motion.** Sometimes, it is easy to view how our background subtraction works using *Motion Energy* plots. Regions with high energy indicate a large movement in the short length span. This is just the accumulation of frame differences for a certain number l of pairs of frames. Fill in the code in `my_motion_energy`.

(d) State in your report which of the algorithms performed the best for each of the sequences given using quantitative comparison. You can compute the error map of your estimate by using high flow pixels in the ground truth flow given in the video as the foreground.

5. **Getting our hands dirty with Abstract/Parameter gradients.** A trivial CNN takes a single, scalar input x, has a first layer with kernel k_1, bias b_1, with the activation function $h(a) = a^2$, and a second layer with kernel k_2, bias b_2, with a softmax activation function. All kernels and biases are scalar, so the output $\hat{y}$ is a scalar as well. The loss function is the cross entropy, i.e., $l(y, \hat{y}) = y \log \hat{y}$. Compute analytically each of the four components of the gradient of the error $E_1(w)$ for training sample (x_1, y_1) relative to the vector $w = [k_1, b_1, k_2, b_2]^T$ of the neural network's parameters. Your answer should only involve variables mentioned in the question.

6. **Helping Munchkins.** Munchkins wants to improve the accuracy of a virus detector using a CNN with l layers and ReLU activation functions. However, Munchkins' boss Munchies wants to use linear activation functions, that is, $\sigma(x) = x$. Show that if the activation functions are linear, then the entire network can be replaced by single layer.
