EEP 596 Computer Vision

HW #3

The purpose of this homework is to continue to explore basic image processing and computer vision concepts using Python and associated libraries. You are free to work with fellow students but should turn in your own work. Specifically, you should be able to explain anything in your assignment if asked.

You should turn in your assignment as "Assignment3.py" and "Report.pdf" in Gradescope. You should develop code with the skeleton provided. You could use print() and cv2.imsave() to help you to generate outputs for the report, but should be commented while turning in. *Important: Before submission, make sure to comment out any function calls in your methods such as cv2.imshow() and cv2.waitKey(). This will ensure that the autograder does not fail through timeout.*

The questions are as follows.

1. **Basic tensor arithmetic.** Load the `original_image.png` image provided using opencv. Swap BGR to RGB channels first.
    a. Convert the image to a PyTorch tensor. The specifications of expected output are:

    Original OpenCV Image shape: (321, 433, 3)
    Original OpenCV Image dtype: uint8
    Original OpenCV Image dtype: uint8
    Torch Image shape: torch.Size([321, 433, 3])
    Torch Image dtype: torch.float32
    Torch Image Pixel dtype: torch.float32

    b. Add the constant 100.0 to each color channel of the image to brighten it. The specifications of expected output are:

    Torch Image shape: torch.Size([321, 433, 3])
    Torch Image dtype: torch.float32
    Torch Image Pixel dtype: torch.float32

    c. Write a method to perform saturation arithmetic. The method takes as input as an opencv image. Convert that to a uint8 tensor after swapping channels to RGB. Then add a constant integer 100 to each channel. Expected output is a uint8 tensor. (Due to uint8 being constrained to (0,255), any value j over 255 will be saturated to j = 255)

2. **Data augmentation.** One common trick in data augmentation is to add random Gaussian noise to an image. To better appreciate this step, in this question you will add random Gaussian noise to an image. Add noise with mean= 0 and σ=100.0 gray levels. When returning a float32 tensor, remember that the range of 0..1 is expected, so be sure to divide by 255 and clamp.

3. **Image normalization.** It is common practice to normalize an image before sending it to a neural network. To do this you should compute the mean and standard deviation of the image for each of the 3 color channels. (That is, a vector of 3 means, and a vector of 3 standard deviations.) Use float64 and RGB format throughout this question.

   Now suppose we have a network that was trained on a collection of images whose mean and std are given. To prepare our image for this network, we need to subtract the given mean and normalize by the std. (It will help to be familiar with broadcasting semantics.)
   Apply this method in two ways (Do not forget to clamp final values):
   - **a.** First, use the mean and std computed from the image itself, so that the output has zero mean and unit std.
   - **b.** Second, use the ImageNet means = [0.485, 0.456, 0.406] and standard deviations = [0.229, 0.224, 0.225], for red, green, and blue, respectively (See here.). Apply this to the image, and return the results.

4. **Dimension rearranging.** Rearrange the dimensions of the `original_image.png` image so that the tensor is NxCxHxW instead of HxWxC, where C is the number of color channels, and N=1 is the batch size. (see PyTorch's permute method). Use float32 and RGB format.

5. **Stride (10 points).** Apply a stride convolution with stride 2, using a 3x3 Scharr_x filter (as mentioned in lecture 2) to "cat_eye.jpg." Load the image in **grayscale**. Return the convolved image in the torch array (2 dimension) with the type torch.FloatTensor. (*Remember to pad the image with value 0, such that when performing convolve only without stride, the image size after convolution is the same as the original image size*.)