Homework 0

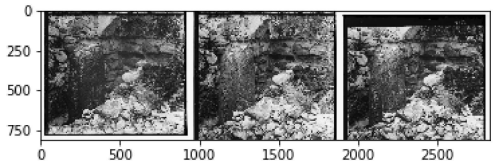
> **Submission Instructions:** Before the deadline, export the completed notebook to PDF and upload it to GradeScope. The PDF should clearly show your code, and the result of running the code. Check the PDF to ensure that it is readable, the font-size is not small, and no information is cut-off. There will be no make-ups or extensions for corrupted/damaged/unreadable PDFs.

Sergei Mikhailovich Prokudin-Gorskii was a color photographer far ahead of his time. He undertook a photographic survey of the Russian Empire for Tsar Nicholas II and was able to capture color pictures before color cameras were invented. His idea was to simply take three pictures of each scene, each with a red, green and blue color filter which could then be used to obtain a color image. There was no way of printing these back in the day, so he envisioned complex display devices to show these. However these were never made, but his pictures survived. In this homework, you will write a program to combine individual photos into a gray-scale photo.

Visualizing the Data

You have been given the red, green, and blue channels of an image that were taken separately using this technique. These files are named red.npy, green.npy, and blue.npy respectively. Let's load these images, and visualize them.

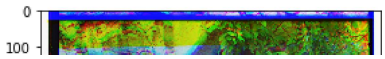
```
# Load libraries and convenience functions

from IPython import display
import matplotlib.pyplot as plt
import numpy as np

def load_image(filename):
    img = np.load(filename)
    img = img.astype("float32") / 255.
    return img

def gray2rgb(image):
    return np.repeat(np.expand_dims(image, 2), 3, axis=2)

def show_image(img):
    plt.imshow(img, interpolation='nearest')


!wget -qN https://www.cs.columbia.edu/~vondrick/class/coms4732/hw0/blue.npy
!wget -qN https://www.cs.columbia.edu/~vondrick/class/coms4732/hw0/green.npy
!wget -qN https://www.cs.columbia.edu/~vondrick/class/coms4732/hw0/red.npy


images = [load_image('red.npy'),
          load_image('green.npy'),
          load_image('blue.npy')]

show_image(gray2rgb(np.concatenate(images, axis=1)))
```



The Problem

Because these images were taken separately, just combining them in a 3-channel matrix may not work. The code below shows what happens if you simply combine the images without shifting any of the channels. Run the code to visualize what happens if you just combine them without correctly aligning them.

```
show_image(np.stack(images, axis=-1))
```



Your Task

Your job is to write a function that takes these three images, and correctly aligns them. Since you have to process many of these images, you do not want to manually align them. Instead, your task is to write a program that automatically finds the alignment, then combines them together to produce the final image.

The easiest way to do this is to find the alignment between two pairs of channels at a time. For example, you can figure out how to align the red channel with the green channel, and the red channel with the blue channel. Then, you can combine them together.

We have given you code to get you started. You should fill in three areas:

1. **score_function(im1, im2)** should take in two images, and return a floating point score to indicate how well the two images are aligned. The lower the score, the better they are aligned. There are many scoring functions you can experiment with. The simplest is Euclidean distance betwen the two images.
2. **align_channels(chan1, chan2)** should take in two images, and return a tuple (dy, dx) indicating how to shift one image into the other. This funnction should call **score_function()** to perform this task. For simplicity, you can assume that the shift is at most -30 pixels to 30 pixels.
3. **combine_images()** should use the found alignments to correctly combine the images into a color image.

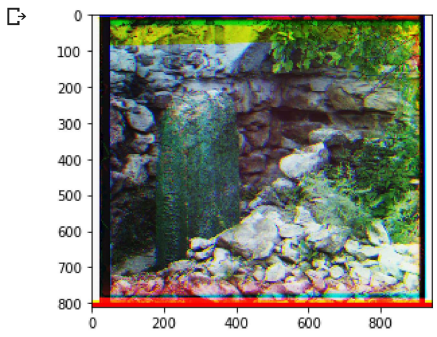
There is no time limit on how fast your algorithm needs to run, however our solution is very fast. For full credit, make sure the center of the image is aligned correctly (it's OK if the edges are ragged). Be careful to check that all of the colors channels are aligned and not just some.

Submission

You should export your completed notebook as a PDF and upload to GradeScope. The completed notebook should show your code, as well as the final combined image you created.

```
# Store the height and width of the images
height, width = images[0].shape

# Pad each image with black by 30 pixels. You do not need to use this, but
# padding may make your implementation easier.
pad_size = 30
images_pad = [np.pad(x, pad_size, mode='constant') for x in images]

# Given two matrices, write a function that returns a number of how well they are aligned.
# The lower the number, the better they are aligned. There are a variety of scoring functions
# you can use. The simplest one is the Euclidean distance between the two matrices.
def score_function(im1, im2):
    # TODO
    distance = np.linalg.norm(im1-im2)
    return distance

"""
A score value of image[0] and image[1] gives 7.1666
It measures the average of euclidean distance of each numpy array of the image
"""

# Given two matrices chan1 and chan2, return a tuple of how to shift chan2 into chan1. This
# function should search over many different shifts, and find the best shift that minimizes
# the scoring function defined above.
def align_channels(chan1, chan2):
    best_offset = (0,0)
    best_score = 0
    # TODO
    best_score = 1000000
    for i in range(-30, 30):
      for j in range(-30, 30):
        # horizontal shift the image to right:
        temp = 0
        if i == 0:
          temp = np.copy(chan2)

        if i > 0:
          temp = np.pad(chan2, ((0,0),(i,0)), mode='constant')[:, :-1*i]
        elif i < 0:
          temp = np.pad(chan2, ((0,0), (0,i*-1)), mode='constant')[:, i*-1:]


        if j < 0:
          temp = np.pad(temp, ((j*-1,0),(0,0)), mode='constant')[-1*j:, :]
        elif j > 0:
          temp = np.pad(temp, ((0,j), (0,0)), mode='constant')[j:, :]

        score = score_function(temp, chan1)
```

```
            score = score_function(temp, chan1)
            if score < best_score:
              best_score = score
              best_offset = (i,j)

    return best_offset


# Use the best alignments to now combine the three images. You should use any of the variables
# above to return a tensor that is (Height)x(Width)x3, which is a color image that you can visualize.
def combine_images():
    #TODO
    """
      Get how to shift green and blue images to align with picture red
    """
    rg_shift_y, rg_shift_x = align_channels(images[0], images[1])
    rb_shift_y, rb_shift_x = align_channels(images[0], images[2])

    if rg_shift_y > 0:
      temp = np.pad(images[1], ((0,0),(rg_shift_y,0)), mode='constant')[:, :-1*rg_shift_y]
    elif rg_shift_y < 0:
      temp = np.pad(images[1], ((0,0), (0,rg_shift_y*-1)), mode='constant')[:, rg_shift_y*-1:]
    if rg_shift_x < 0:
      temp = np.pad(images[1], ((rg_shift_x*-1,0),(0,0)), mode='constant')[-1*rg_shift_x:, :]
    elif rg_shift_x > 0:
      temp = np.pad(images[1], ((0,rg_shift_x), (0,0)), mode='constant')[rg_shift_x:, :]


    if rb_shift_y > 0:
      temp2 = np.pad(images[2], ((0,0),(rb_shift_y,0)), mode='constant')[:, :-1*rb_shift_y]
    elif rb_shift_y < 0:
      temp2 = np.pad(images[2], ((0,0), (0,rb_shift_y*-1)), mode='constant')[:, rb_shift_y*-1:]
    if rb_shift_x < 0:
      temp2 = np.pad(images[2], ((rb_shift_x*-1,0),(0,0)), mode='constant')[-1*rb_shift_x:, :]
    elif rb_shift_x > 0:
      temp2 = np.pad(images[2], ((0,rb_shift_x), (0,0)), mode='constant')[rb_shift_x:, :]

    new_images = [images[0], temp, temp2]
    return np.stack(new_images, axis=-1)


final_image = combine_images()
if final_image is not None:
    show_image(final_image)
```



Acknowledgements

This homework is based on assignments from Subhransu Maji at University of Massachusetts, Amherst, Alyosha Efros at University of California, Berkeley, Jia Deng at University of Michigan, and Deva Ramanan at Carnegie Mellon University.

✓ 17s completed at 9:48 PM