Computational Photography

Assignment #6: Blending

Ram Subramanian

Fall 2016

Part 0: reduce_layer() & expand_layer()

Reduce: `cv2.filter2D(image, -1, kernel, borderType=cv2.BORDER_CONSTANT)[::2, ::2].astype(np.float64)`

Expand:

```
out = np.zeros((image.shape[0]*2, image.shape[1]*2))
out[::2, ::2] = image
return cv2.filter2D(out, -1, kernel, borderType=cv2.BORDER_CONSTANT) * 4
```

Reduce: Quite straightforward – I apply the kernel using cv2.filter2D() after padding the image with zeros, and then use numpy splicing to extract every second row and column.

Expand: I first create an np array whose dimensions are twice those of the image and then I set every second row and column in this new array to be the original array – this is equivalent to adding a row and column of 0's alternatively in the original image. Finally, I run the new array through a filter (zero padding) and I scale the intensities up by a factor of 4.

QUESTION: What is the significance of using a = 0.4 for the generating kernel?

QUESTION: Why does the output of expand_layer have to be multiplied by 4?

Part 1: gaussPyramid()

```
output = [image]
for i in range(levels):
        bimg = cv2.GaussianBlur(output[-1], (5, 5), 0.05)
        output.append(reduce_layer(bimg))
return output
```

To construct a gaussian pyramid, I first place the original image in the pyramid list. Every successive element of the list will simply be a blurred and reduced version (using the reduce algorithm described in the previous slide) of the previous element. So every element past the first will be a ¼ of the size of the previous element, thereby resembling a (linear) pyramid.

I stop when the required depth/height of the pyramid is reached.

Part 1: laplPyramid()

```
gaussPyr = gaussPyr[::-1]
output = [gaussPyr[0]]
for i, img in enumerate(gaussPyr[:-1]):
    img = expand_layer(img)
    img = crop(img, gaussPyr[i+1].shape)
    output.append(gaussPyr[i+1] - img)
return output[::-1]
```

I had to define a function here, crop(), that given an image and a 2d dimension, would crop the image to those dimensions.

To construct a laplacian pyramid, I first reverse the order of the gaussian pyramid, for convenience sake and add the first element (now the smallest entry in the gaussian pyramid) to my output list. For every successive element of the output list, I expand the previous element (using algos above), crop it (only necessary when any of the dimensions of the next larger gaussian is odd), and subtract it from the next largest gaussian. Since the next largest gaussian will be 4 times the size of the current, and that we expanded the current, we can be sure that during subtraction, both sides will have the same dimensions.

I stop when the necessary depth/height of the pyramid is achieved. Since we reversed the order of the gaussian pyramid initially, we need to flip the laplacian pyramid around as well.

Part 2: blend()

```
blendedPyr = np.array(laplPyrWhite) * np.array(gaussPyrMask) + np.array(laplPyrBlack) * (1 -
np.array(gaussPyrMask))

# For readability's sake:
# blendedPyr = laplPyrWhite * gaussPyrMask + laplPyrBlack * (1 - gaussPyrMask)
```

Since python lists don't support broadcasting and since the arguments to blend() are all lists, I first convert them to np arrays.

From there, since the elements that need to be blended together are conveniently in the same indices in each array, I simply do an arithmetic to get the final result.

We want areas that equal 1.0 in the mask to be retained from the white image and areas that equal 0.0 to be retrained from the black image (mask is binary, i.e. all pixels either equal 0.0 or 1.0), the arithmetic then simply goes:

```
        white * mask + black * (1 - mask)
```

Part 2: collapse()

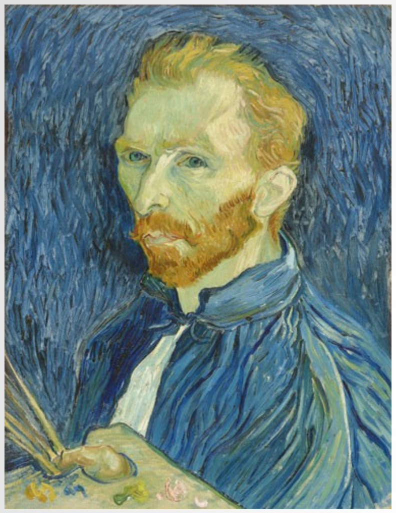
```
pyramid = pyramid[::-1]
out = pyramid[0]
for i in range(1, len(pyramid)):
        out = crop(expand_layer(out), pyramid[i].shape)
        out = out + pyramid[i]
return out
```

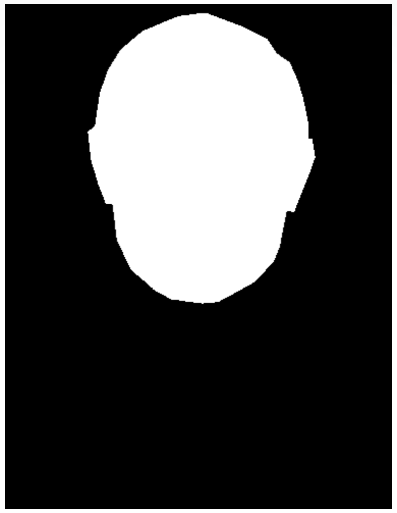
I make use of my earlier crop() function yet again here.

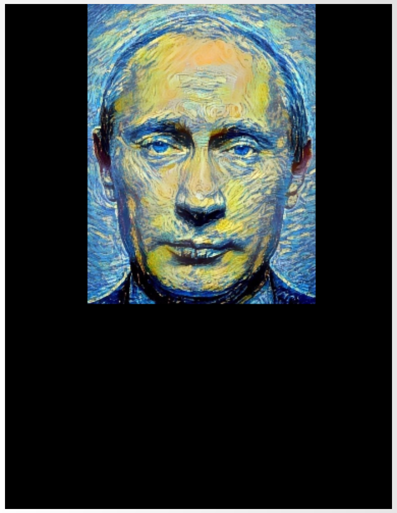
In order to collapse a blended pyramid, we simply need to expand the smallest layer and add it to the next larger layer and repeat until we are left with only one layer.

Again, for convenience sake, I invert the input pyramid and immediately move the first element to the output layer. Then I expand the output layer and add it to the next element in the pyramid (which would be of the same size). Repeat until there are no more layers to add in the pyramid.

Input Images


Black


Mask

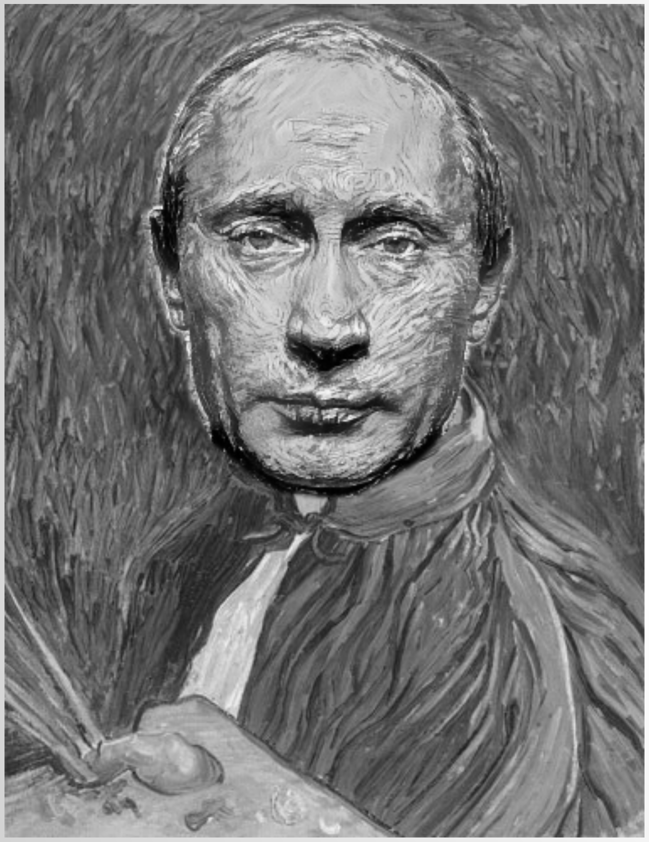

White

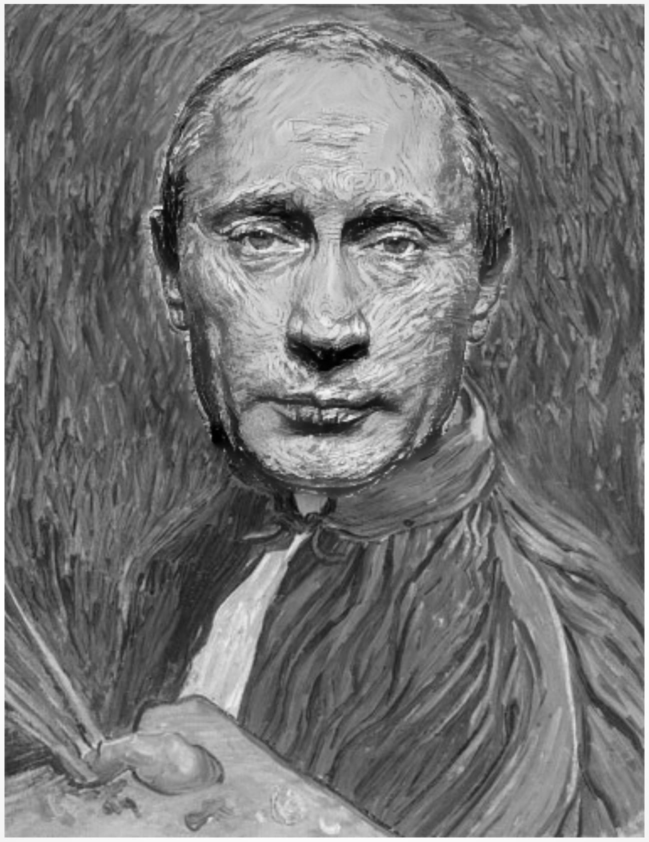
The black image is a self-portrait of the famed painter Vincent van Gogh. And the white image is a close-up portrait of Vladimir Putin – not the subject I would have normally chosen, but it was the only reasonable image I could find to fit this scenario. I had to run the original through a neural network (deepart.io) to convert it so it 'matches' van Gogh's style in his self-portrait.

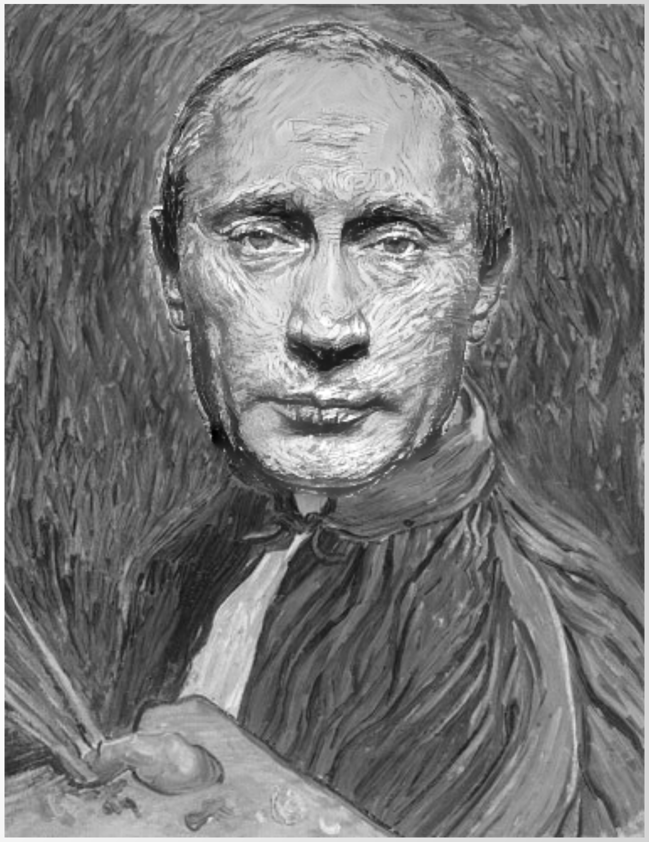
In order to create a mask, I had to resize the white image and position it around where I'd like it blended (while also filling the rest of the space with zeros, so the image sizes match). Then I used GIMP to create the mask (using the lasso tool and fill bucket).

Black image from: nga.gov, http://www.nga.gov/content/ngaweb/education/teachers/lessons-activities/self-portraits/van-gogh/jcr%3Acontent/parmain/textimage/image.img.jpg/1404756844333.jpg
White image from: Imgur, https://i.imgur.com/lnZXTOJ.jpg
Photo convertor: https://deepart.io/

Blended Image


Pyramid depth = 1


Pyramid depth = 5


Pyramid depth = 8

Discussion

1. More or less. I like that the the blending is less and less visible as you increase the depth of the pyramid and around 8 it's starts get quite seamless. Although visually, level 5 looks best, albeit with some blending edges visible.
2. A lot of flaws! To name a few, the unusually big face for the body (but this was necessary to cover van Gogh's head entirely). The unusually bright nature of the 'strokes' that make up Putin's face, compared to the rest of the 'painting'. The visible portion of van Gogh's neck towards the right side. And finally the expression on Putin's face is in stark contrast with the spirit of the painting or the painter xD, but that's not a blending flaw..
3. I could have tried to play around with the hue, saturation of Putin's portrait before the blend, in order to make it more suitable. I'm sure if I dig deep enough I can find a good replacement for Putin, who would ideally be angled the same way as van Gogh and have roughly the same proportions as van Gogh.
4. I had to manually superimpose both images to determine a blend location (using GIMP).
5. I'm not sure if it would quite work for my scenario since the subjects face different directions, but one could use feature matching to match the eyes, the nose and mouth to get a precise location of the blend and the necessary scaling factor so both faces are roughly the same size.
6. In this case, I think blending is the better approach as both images vary widely in many parameters, such as size, background, foregroound, style (though deepart did a good job, it was nowhere close to perfect transform), size, etc. Cut will work better when a lot of the above parameters are the same between the two images.

Resources

Deepart.io – converting one image into the style of another
Nga.gov – van Gogh’s self-portrait
Imgur.com – Putin’s portrait
GIMP – creating the mask
OpenCV, Numpy

Above & Beyond

Split Family Faces (http://www.mymodernmet.com/profiles/blogs/split-family-faces) is a blog of pictures of faces, whose halves come from different but closely related people. Inspired by the blog, I set out to make the same. Since I didn't have good equipment (camera, lighting, etc) or good closely related models on time, I decided to blend two images I found on the internet, together. However, I do plan on recreating the SFF project when the right circumstances come :)