Lab 3: Digital imagery and image processing

**Purpose**: The purpose of this lab is to demonstrate concepts of digital image processing. You will be introduced to methods for image smoothing, sharpening, edge detection, morphological processing, texture analysis, resampling and reprojection. At the completion of the lab, you will be able to identify image processing operators that may be useful in extracting information of interest for your image analyses.

**Prerequisites**: Lab 2

## What is a digital image?

A digital image, sometimes called a [raster](https://en.wikipedia.org/wiki/Raster_graphics), is a block of [pixels](https://en.wikipedia.org/wiki/Pixel) ([Besag 1986](http://www.jstor.org/stable/2345426)). Each pixel has a position, measured with respect to the axes of some coordinate reference system (CRS), a [geographic coordinate system](https://en.wikipedia.org/wiki/Geographic_coordinate_system) for example. A CRS in Earth Engine is often referred to as a projection, since it combines a shape of the Earth with a [datum](https://en.wikipedia.org/wiki/Geodetic_datum) and a transformation from that spherical shape to a flat map, called a [projection](https://en.wikipedia.org/wiki/Map_projection).

For example:

1. Make a Point geometry using the [geometry drawing tools](https://developers.google.com/earth-engine/playground#geometry-tools) to define your area of interest. Name the import point.
2. Import NAIP imagery by searching for 'naip' and choosing the 'NAIP: National Agriculture Imagery Program' raster dataset. Name the import naip.
3. Get a single, recent NAIP image over your study area and inspect it:

// Get a single NAIP image over the area of interest.

var image = ee.Image(naip

.filterBounds(point)

.sort('system:time\_start', false)

.first());

// Print the image to the console.

print('Inspect the image object:', image);

// Display the image with the default visualization.

Map.centerObject(point, 18);

Map.addLayer(image, {}, 'Original image');

Map.addLayer(image, {bands: ['N', 'R', 'G']}, 'false color NAIP');

1. Expand the image object that is printed to the console. Expand the bands property and one of the bands (0, for example). Note that the CRS transform is stored in the crs\_transform property and the CRS is stored in the crs property, which references an EPSG code. The CRS of our image is [EPSG:26910](http://spatialreference.org/ref/epsg/nad83-utm-zone-10n/). Often, you can learn more about those [EPSG codes](http://www.epsg-registry.org/) from [the spatialreference.org site](http://spatialreference.org/). The CRS transform is a list: [m00, m01, m02, m10 m11 m12] in the notation of [this reference](http://docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html).
2. Access these data programmatically with the .projection() method:

// Display the projection of band 0.

print('Inspect the projection of band 0:', image.select(0).projection());

1. Note that the projection can differ by band, which is why it's good practice to inspect the projection of individual image bands. (If you call .projection() on an image for which the projection differs by band, you'll get an error.) Explore the ee.Projection docs to learn about useful methods offered by the Projection object. (To play with projections offline, try [this tool](http://www.giss.nasa.gov/tools/gprojector/).)

## Digital image visualization

You've learned about how an image stores pixel data in each band as DNs and how the pixels are organized spatially. When you add an image to the map, Earth Engine handles the spatial display for you by recognizing the projection and putting all the pixels in the right place. However, you must specify how to stretch the DNs to make an 8-bit display image (e.g. the min and max visualization parameters). Specifying min and max applies (where DN' is the displayed value):

DN' = (DN - min) \* 255 / (max - min)

1. To apply a [gamma correction](https://en.wikipedia.org/wiki/Gamma_correction) (DN' = DN𝛾), use:

// Display gamma stretches of the input image.

Map.addLayer(image.visualize({gamma: 0.5}), {}, 'gamma = 0.5');

Map.addLayer(image.visualize({gamma: 1.5}), {}, 'gamma = 1.5');

* 1. Note that gamma is supplied as an argument to image.visualize() so that you can click on the map to see the difference in pixel values (try it!). It's possible to specify gamma, min and max to achieve other unique visualizations.

1. (*Optional*) To apply a [histogram equalization](https://en.wikipedia.org/wiki/Histogram_equalization) stretch, use the [sldStyle() method](https://devsite.googleplex.com/earth-engine/image_visualization#styled-layer-descriptors):

// Define a RasterSymbolizer element with '\_enhance\_' for a placeholder.

var histogram\_sld =

'<RasterSymbolizer>' +

'<ContrastEnhancement><Histogram/></ContrastEnhancement>' +

'<ChannelSelection>' +

'<RedChannel>' +

'<SourceChannelName>R</SourceChannelName>' +

'</RedChannel>' +

'<GreenChannel>' +

'<SourceChannelName>G</SourceChannelName>' +

'</GreenChannel>' +

'<BlueChannel>' +

'<SourceChannelName>B</SourceChannelName>' +

'</BlueChannel>' +

'</ChannelSelection>' +

'</RasterSymbolizer>';

// Display the image with a histogram equalization stretch.

Map.addLayer(image.sldStyle(histogram\_sld), {}, 'Equalized');

* 1. The [sldStyle() method](https://devsite.googleplex.com/earth-engine/image_visualization#styled-layer-descriptors) requires image statistics to be computed in a region (to determine the histogram).

## Linear filtering

In the present context, linear *filtering* (or [convolution](http://www.dspguide.com/ch24/1.htm)) refers to a linear combination of pixel values in a neighborhood. The neighborhood is specified by a [kernel](https://en.wikipedia.org/wiki/Kernel_(image_processing)), where the weights of the kernel determine the coefficients in the linear combination. (For this lab, the terms *kernel* and *filter* are interchangeable.) Filtering an image can be useful for extracting image information at different [spatial frequencies](http://www.dspguide.com/ch24/5.htm). For this reason, smoothing filters are called *low-pass* filters (they let *low*-frequency data *pass* through) and edge detection filters are called *high-pass* filters. To implement filtering in Earth Engine use [image.convolve()](https://devsite.googleplex.com/earth-engine/image_convolutions) with an ee.Kernel for the argument.

1. **Smoothing**. Smoothing means to convolve an image with a smoothing kernel.
   1. A simple smoothing filter is a square kernel with uniform weights that sum to one. Convolving with this kernel sets each pixel to the mean of its neighborhood. Print a square kernel with uniform weights (this is sometimes called a "pillbox" or "boxcar" filter):

// Print a uniform kernel to see its weights.

print('A uniform kernel:', ee.Kernel.square(2));

Expand the kernel object in the console to see the weights. This kernel is defined by how many pixels it covers (i.e. radius is in units of 'pixels'). A kernel with radius defined in 'meters' adjusts its size in pixels, so you can't visualize its weights, but it's more flexible in terms of adapting to inputs of different scale. In the following, use kernels with radius defined in meters except to visualize the weights.

* 1. Define a kernel in with 2-meter radius (Which corresponds to how many pixels in the NAIP image? Hint: try projection.nominalScale()), convolve the image with the kernel and compare the input image with the smoothed image:

// Define a square, uniform kernel.

var uniformKernel = ee.Kernel.square({

radius: 2,

units: 'meters',

});

// Filter the image by convolving with the smoothing filter.

var smoothed = image.convolve(uniformKernel);

Map.addLayer(smoothed, {min: 0, max: 255}, 'smoothed image');

* 1. To make the image even more smooth, try increasing the size of the neighborhood by increasing the pixel radius.
  2. A Gaussian kernel can also be used for smoothing. Think of filtering with a Gaussian kernel as computing the weighted average in each pixel's neighborhood. For example:

// Print a Gaussian kernel to see its weights.

print('A Gaussian kernel:', ee.Kernel.gaussian(2));

// Define a square Gaussian kernel:

var gaussianKernel = ee.Kernel.gaussian({

radius: 2,

units: 'meters',

});

// Filter the image by convolving with the Gaussian filter.

var gaussian = image.convolve(gaussianKernel);

Map.addLayer(gaussian, {min: 0, max: 255}, 'Gaussian smoothed image');

1. **Edge detection**. Convolving with an edge-detection kernel is used to find rapid changes in DNs that usually signify edges of objects represented in the image data.
   1. A classic edge detection kernel is the [Laplacian](https://en.wikipedia.org/wiki/Discrete_Laplace_operator) kernel. Investigate the kernel weights and the image that results from convolving with the Laplacian:

// Define a Laplacian filter.

var laplacianKernel = ee.Kernel.laplacian8();

// Print the kernel to see its weights.

print(laplacianKernel);

// Filter the image by convolving with the Laplacian filter.

var edges = image.convolve(laplacianKernel)

Map.addLayer(edges, {min: 0, max: 255}, 'Laplacian filtered image');

* 1. (Ignore the reproject() call for now. It is explained in section 6.)
  2. Other edge detection kernels include the [Sobel](https://en.wikipedia.org/wiki/Sobel_operator), [Prewitt](https://en.wikipedia.org/wiki/Prewitt_operator) and [Roberts](https://en.wikipedia.org/wiki/Roberts_cross) kernels. [Learn more about additional edge detection methods in Earth Engine](https://developers.google.com/earth-engine/image_edges).

1. (*Optional*) **Gradients**. Image gradient is the change in pixel values over space (analogous to computing slope from a DEM).
   1. In Earth Engine, use [image.gradient()](https://devsite.googleplex.com/earth-engine/image_gradients) to compute the gradient in an image band. For example, gradients in the NIR band indicate transitions in vegetation:

// Compute the image gradient in the X and Y directions.

var xyGrad = image.select('N').gradient();

// Compute the magnitude of the gradient.

var gradient = xyGrad.select('x').pow(2)

.add(xyGrad.select('y').pow(2)).sqrt();

// Compute the direction of the gradient.

var direction = xyGrad.select('y').atan2(xyGrad.select('x'));

// Display the results.

Map.addLayer(direction, {min: -3, max: 3, format: 'png'}, 'direction');

Map.addLayer(gradient, {min: -10, max: 50, format: 'png'}, 'gradient');

* 1. (Ignore the reproject() call for now. It is explained in section 6.)
  2. For an in-depth study of gradients in multi-spectral imagery, see [Di Zenzo (1986)](http://www.sciencedirect.com/science/article/pii/0734189X86902239).

1. (*Optional*) **Sharpening**. Image sharpening, or [edge enhancement](http://www.dspguide.com/ch24/2.htm), is related to the idea of the image second derivative. Specifically, mimic the perception of [Mach bands](https://en.wikipedia.org/wiki/Mach_bands) in human optical response by adding the image to its second derivative.
   1. One implementation of this idea is to convolve an image with a Laplacian-of-a-Gaussian or [Difference-of-Gaussians](https://en.wikipedia.org/wiki/Difference_of_Gaussians) filter (see [Schowengerdt 2007](http://www.sciencedirect.com/science/book/9780123694072) for details), then add that to the input image:

// Define a "fat" Gaussian kernel.

var fat = ee.Kernel.gaussian({

radius: 3,

sigma: 3,

magnitude: -1,

units: 'meters'

});

// Define a "skinny" Gaussian kernel.

var skinny = ee.Kernel.gaussian({

radius: 3,

sigma: 0.5,

units: 'meters'

});

// Compute a difference-of-Gaussians (DOG) kernel.

var dog = fat.add(skinny);

// Add the DoG filtered image to the original image.

var sharpened = image.add(image.convolve(dog));

Map.addLayer(sharpened, {min: 0, max: 255}, 'Edges enhanced');

* 1. Related concepts include [*spectral inversion*](http://www.dspguide.com/ch14/5.htm) from digital signal processing and *unsharp masking* ([Burger and Burge 2008](http://imagingbook.com/)).

## Non-linear filtering

The previous convolution examples can all be implemented as linear combinations of pixel values in a neighborhood (gradient needs a couple extra steps, but nevermind that). Non-linear functions applied to a neighborhood are also useful. Implement these functions in Earth Engine with the [reduceNeighborhood()](https://devsite.googleplex.com/earth-engine/reducers_reduce_neighborhood) method on images.

1. **Median**. A median filter can be useful for denoising images. Specifically, suppose that random pixels in your image are saturated by anomalously high or low values that result from some noise process. Filtering the image with a mean filter (as in section 3.a.i) would result in pixel values getting polluted by noisy data. To avoid that, smooth the image with a median filter (reusing the 5x5 uniform kernel from above):

var median = image.reduceNeighborhood({

reducer: ee.Reducer.median(),

kernel: uniformKernel

});

Map.addLayer(median, {min: 0, max: 255}, 'Median');

1. **Mode**. For categorical maps, methods such as median and mean make little sense for aggregating nominal data. In these cases, use neighborhood mode to get the most frequently occurring value.
   1. For demonstration purposes, make a categorical map by thresholding the NIR band. Two classes, with labels 1 and 0 are the result:

// Create and display a simple two-class image.

var veg = image.select('N').gt(150);

// Display the two-class (binary) result.

var binaryVis = {min: 0, max: 1, palette: ['black', 'green']};

Map.addLayer(veg, binaryVis, 'veg');

* 1. Compute the mode in each 5x5 neighborhood:

// Compute the mode in each 5x5 neighborhood and display the result.

var mode = veg.reduceNeighborhood({

reducer: ee.Reducer.mode(),

kernel: uniformKernel

});

Map.addLayer(mode, binaryVis, 'mode');

* 1. Compare the shape of patches in the mode image to the original veg image. Note the smoothing effect that the mode has on the shape of patches.

1. **Morphological processing**. The idea of morphology is tied to the concept of objects in images. For example, suppose the patches of 1's in the veg image from the previous section represent patches of vegetation.
   1. **Dilation** (max). If the patches underestimate the actual distribution of vegetation, or contain "holes", a max filter can be applied to [dilate](https://en.wikipedia.org/wiki/Dilation_(morphology)) the areas of vegetation:

// Dilate by takaing the max in each 5x5 neighborhood.

var max = veg.reduceNeighborhood({

reducer: ee.Reducer.max(),

kernel: uniformKernel

});

Map.addLayer(max, binaryVis, 'max');

* 1. Try increasing the amount of dilation by increasing the size of the kernel (i.e. increase the radius) or applying reduceNeighborhood() more than once.
  2. **Erosion** (min). The opposite of dilation is [erosion](https://en.wikipedia.org/wiki/Erosion_(morphology)), for decreasing the size of the patches:

// Erode by takaing the min in each 5x5 neighborhood.

var min = veg.reduceNeighborhood({

reducer: ee.Reducer.min(),

kernel: uniformKernel

});

Map.addLayer(min, binaryVis, 'min');

* 1. Carefully inspect the result compared to the input. Note that the shape of the kernel affects the *shape* of the eroded patches (the same effect occurs in the dialtion). Explore this effect by testing different shape kernels. As with the dilation, note that you can get more erosion by increasing the size of the kernel or applying the operation repetitively.
  2. **Opening**. To "open" possible "holes" in the patches, perform an erosion followed by a dilation. This process is called [opening](https://en.wikipedia.org/wiki/Opening_(morphology)). Try that by performing a dilation of the eroded image:

// Perform an opening by dilating the eroded image.

var opened = min.reduceNeighborhood({

reducer: ee.Reducer.max(),

kernel: uniformKernel

});

Map.addLayer(opened, binaryVis, 'opened');

* 1. **Closing**. The opposite of opening is [closing](https://en.wikipedia.org/wiki/Closing_(morphology)), or dilation followed by a erosion. Use this to "close" possible "holes" in the input patches:

// Perform a closing by eroding the dilated image.

var closed = max.reduceNeighborhood({

reducer: ee.Reducer.min(),

kernel: uniformKernel

});

Map.addLayer(closed, binaryVis, 'closed');

* 1. Closely examine the difference between each morphological operation and the veg input. Tune these morphological operators by adjusting the size and shape of the kernel (also called a [*structuring element*](https://en.wikipedia.org/wiki/Structuring_element) in this context, because of its effect on the shape of the result), or applying the operations repetively.

## Texture

Define texture as some measure of DN distribution within a neighborhood. There are a variety of ways to [compute texture in Earth Engine](https://developers.google.com/earth-engine/image_texture).

1. **Standard Deviation** (SD). The SD measures the spread of the DN distribution in the neighborhood. A textureless neighborhood, in which there is only one DN, has SD=0. Compute neighborhood SD for the NAIP image with:

// Define a big neighborhood with a 7-meter radius kernel.

var bigKernel = ee.Kernel.square({

radius: 7,

units: 'meters'

});

// Compute SD in a neighborhood.

var sd = image.reduceNeighborhood({

reducer: ee.Reducer.stdDev(),

kernel: bigKernel

});

Map.addLayer(sd, {min: 0, max: 70}, 'SD');

1. **Entropy**. For discrete valued inputs, you can compute [entropy](https://en.wikipedia.org/wiki/Diversity_index#Shannon_index) in a neighborhood, where entropy in this context is like an index of DN diversity in the neighborhood:

// Compute entropy in a neighborhood.

var entropy = image.entropy(bigKernel);

Map.addLayer(entropy, {min: 1, max: 5}, 'entropy');

1. (*Optional*) **Gray-level co-occurrence matrices** (GLCM). The GLCM is computed by forming an *M*x*M* matrix for an image with *M* possible DN values, then computing entry *i*,*j* as the frequency at which DN=*i* is adjacent to DN=*j*. A variety of texture metrics can be computed from that matrix, among them is contrast:

// Use the GLCM to compute a large number of texture measures.

var glcmTexture = image.glcmTexture(7);

// Display the 'contrast' results for the red, green and blue bands.

var contrastVis = {

bands: ['R\_contrast', 'G\_contrast', 'B\_contrast'],

min: 40,

max: 2000

};

Map.addLayer(glcmTexture, contrastVis, 'contrast');

1. (*Optional*) **Spatial statistics**. Two interesting texture measures from the field of spatial statistics include local [Moran's I](https://en.wikipedia.org/wiki/Moran%27s_I) and local [Geary's C](https://en.wikipedia.org/wiki/Geary%27s_C) ([Anselin 1995](http://onlinelibrary.wiley.com/doi/10.1111/j.1538-4632.1995.tb00338.x/abstract)). To compute a local Geary's C with the NAIP image as input, use:

// Create a list of weights for a 9x9 kernel.

var list = [1, 1, 1, 1, 1, 1, 1, 1, 1];

// The center of the kernel is zero.

var centerList = [1, 1, 1, 1, 0, 1, 1, 1, 1];

// Assemble a list of lists: the 9x9 kernel weights as a 2-D matrix.

var lists = [list, list, list, list, centerList, list, list, list, list];

// Create the kernel from the weights.

// Non-zero weights represent the spatial neighborhood.

var kernel = ee.Kernel.fixed(9, 9, lists, -4, -4, false);

// Use the max among bands as the input.

var maxBands = image.reduce(ee.Reducer.max());

// Convert the neighborhood into multiple bands.

var neighs = maxBands.neighborhoodToBands(kernel);

// Compute local Geary's C, a measure of spatial association.

var gearys = maxBands.subtract(neighs).pow(2).reduce(ee.Reducer.sum())

.divide(Math.pow(9, 2));

Map.addLayer(gearys, {min: 20, max: 2500}, "Geary's C");

## Resampling and Reprojection

Earth Engine makes every effort to handle projection and scale so that you don't have to. However, there are occasions where an understanding of projections is important to get the output you need. As an example, it's time to demystify the reproject() calls in the previous examples. Earth Engine requests inputs to your computations in the projection and scale of the output. The map attached to the playground has a [Maps Mercator projection](http://epsg.io/3857). The scale is determined from the map's zoom level. When you add something to this map, Earth Engine secretly reprojects the input data to Mercator, resampling (with nearest neighbor) to screen resolution pixels based on the map's zoom level, then does all the computations with the reprojected, resampled imagery. In the previous examples, the reproject() calls force the computations to be done at the resolution of the input pixels: 1 meter.

1. Re-run the edge detection code with and without the reprojection (Comment out all previous Map.addLayer() calls except for the original one):

// Zoom all the way in.

Map.centerObject(point, 21);

// Display edges computed on a reprojected image.

Map.addLayer(image.convolve(laplacianKernel), {min: 0, max: 255},

'Edges with little screen pixels');

// Display edges computed on the image at native resolution.

Map.addLayer(edges, {min: 0, max: 255},

'Edges with 1 meter pixels');

* 1. What's happening here is that the projection specified in reproject() propagates backwards to the input, forcing all the computations to be performed in that projection. If you don't specify, the computations are performed in the projection and scale of the map (Mercator) at screen resolution.

1. You can control how Earth Engine resamples the input with resample(). By default, all resampling is done with nearest neighbor. To change that, call resample() on the *inputs*. Compare the input image, resampled to screen resolution with a bilinear and bicubic resampling:

// Resample the image with bilinear instead of nearest neighbor.

var bilinearResampled = image.resample('bilinear');

Map.addLayer(bilinearResampled, {}, 'input image, bilinear resampling');

// Resample the image with bicubic instead of nearest neighbor.

var bicubicResampled = image.resample('bicubic');

Map.addLayer(bicubicResampled, {}, 'input image, bicubic resampling');

* 1. Try zooming in and out, comparing to the input image resampled with nearest neighbor (i.e. without resample() called on it).

1. ***You should rarely, if ever, have to use* reproject() *and* resample()**. Do not use reproject() or resample() resample unless necessary. They are only used here for demonstration purposes.

## Assignment

So many concepts! Your job is to find the textbook definitions (no Wikipedia) for the following terms:

Gamma Correction

Histogram equalization

Image convolution

Image sharpening

Image texture

Moran’s I

Geary’s C

Bilinear Resampling

Image classification

map projection

[](http://creativecommons.org/licenses/by-sa/4.0/)

This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

Copyright 2015-2016, Google Earth Engine Team