[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

PROCESSING **IMAGE** [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

An image is is an EE variable object that represents a cartographic depiction of geographic space in a format much like that of a digital photograph. Images can be processed by using operations of the types listed below, which vary according to the nature of that processing. Each operation name is linked to a separate page describing that operation.

**UPLOADING** IMAGES[Google Maps Engine](#Upload)

**ACCESSING** IMAGES [ee.Image](#Image) [ee.Image.load](#load)

**CREATING** IMAGES [ee.Image(value)](#Image_constant) [ee.Image.constant](#Image_constant)

[ee.Image.pixelLonLat](#pixelLonLat) [ee.Image.pixelArea](#pixelArea)

**EDITING** IMAGES

BY **MASKING** REGIONS [image.mask](#mask)

BY **CLIPPING** REGIONS [image.clip](#clip)

BY **SELECTING** BANDS [image.select](#select) [image.slice](#slice)

BY **COMBINING** BANDS [image.addBands](#addBands)

BY **REPROJECTING** [image.reproject](#reproject)

BY **RECODING COLORS** [image.rgbtohsv](#rgbtohsv) [image.hsvtorgb](#hsvtorgb)

BY **RECASTING** DATA TYPES [image.uint8](#uint8_) [image.Uint8](#uint8_)

[image.uint16](#uint8_) [image.Uint16](#uint8_)

[image.uint32](#uint8_) [image.Uint32](#uint8_)

[image.int8](#uint8_) [image.toInt8](uint8_#uint8_)  [image.byte](#uint8_) [image.toByte](#uint8_)

[image.int16](#uint8_) [image.toInt16](#uint8_) [image.short](#uint8_)  [image.toShort](#uint8_)

[image.int32](#uint8_) [image.toInt32](#uint8_) [image.int](#uint8_) [image.toInt](#uint8_)

[image.int64](#uint8_) [image.toInt64](#uint8_) [image.long](#uint8_) [image.toLong](#uint8_)

[image.float](#uint8_) [image.toFloat](#uint8_)

[image.double](#uint8_) [image.toDouble](#uint8_)

[image.cast](#cast)

BY **RESETTING** VALUES [image.set](#set) [image.setMulti](#setMulti) [image.remap](#remap)  [image.where](#where)

[image.metadata](#metadata) [image.clamp](#clamp) [image.unitScale](#unitScale) [image.interpolate](#interpolate)

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

PROCESSING **IMAGE** [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

**TRANSFORMING** IMAGES

WITH **LOGICAL** OPERATIONS [image.eq](#eq_) [image.gt](#eq_) [image.lt](#eq_) [image.and](#and_or)

[image.neq](#eq_) [image.gte](#eq_) [array.lte](#eq_) [image.or](#and_or)

[image.not](#not)

WITH **MATHEMATICAL** OPERATIONS [image.abs](#abs_) [image.ceil](#abs_) [image.log](#abs_) [image.floor](#abs_)

[image.round](#abs_) [image.exp](#abs_) [image.sqrt](#abs_) [image.log10](#abs_)

[image.add](#add_) [image.subtract](#add_) [image.multiply](#add_) [image.divide](#add_)

[image.max](#add_) [image.min](#add_) [image.mod](#add_) [image.pow](#add_)

[image.hypot](#add_) [image.first](#add_) [image.first\_nonzero](#add_)

[image.polynomial](#polynomial) [image.expression](#expression)

WITH **TRIGONOMETRIC** OPERATIONS [image.sin](#sin_) [image.cos](#sin_) [image.tan](#sin_)

[image.sinh](#sin_) [image.cosh](#sin_) [image.tanh](#sin_)

[image.acos](#sin_) [image.asin](#sin_) [image.atan](#sin_) [image.atan2](#atan2)

WITH **BITWISE** OPERATIONS [image.bitwiseAnd](#bitwise_) [image.bitwiseOr](#bitwise_) [image.bitwise\_xor](#bitwise_) [image.bitwiseNot](#bitwise_)

[image.bitwise\_and](#bitwise_) [image.bitwise\_or](#bitwise_) [image.bitwiseXor](#bitwise_) [image.bitwise\_not](#bitwise_)

[image.leftShift](#leftShift_) [image.left\_shift](#leftShift_) [image.rightShift](#leftShift_) [image.right\_shift](#leftShift_)

WITH PIXEL **REDUCERS** [image.reduce](#reduce)

WITH **ARRAY** OPERATIONS image.arrayAccum image.arrayFlatten image.arrayGet image.arrayLengths

image.arrayMask image.arrayProject image.arrayReduce. image.arrayRepeat

image.arraySlice image.arraySort image.arrayTranspose

WITH **INSULARITY** OPERATIONS [image.connectedCom…](#connectedComponents) [image.connectedPixelCount](#connectedPixelCount)

WITH **TERRAIN** OPERATIONS [image.derivative](#derivative) [ee.Terrain.products](#Terrain) [ee.Algorithm.Terrain](#Terrain)

[ee.Terrain.slope](#slope) [ee.Terrain.aspect](#aspect) [ee.Terrain.fillMinima](#fillMinima)

[ee.Terrain.hillshade](#hillshade) [ee.Terrain.hillshadow](#hillshadow_) [ee.Algorithm.Hillshadow](#hillshadow_)

WITH **TEXTURE** OPERATIONS [image.entropy](#entropy) [image.glcmTexture](#glcmTexture)

WITH **EDGE** OPERATIONS [image.zeroCrossing](#zeroCrossing) [ee.Algorithms.Canny…](#Canny) [ee.Algorithms.HoughTransform](#Hough)

WITH **DISTANCE** OPERATIONS [image.distance](#distance) [featureCollection.distance](#featureCollectionDistance)

WITH **NEIGHBORHOOD** OPERATIONS [image.focal\_max](#focal_max_) [image.focal\_min](#focal_max_) [image.focal\_median](#focal_max_) [image.focal\_mode](#focal_max_) [image.convolve](#convolve) [image.reduceNeighborhood](#reduceNeighborhood)

WITH **ALIGNMENT** OPERATIONS [ee.Algorithms.CrossCorr…](#CrossCorrelation)

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

PROCESSING **IMAGE** [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

**REPRODUCING** IMAGES

AS **FEATURE COLLECTIONS** [reduceToVectors](#reduceToVectors)

AS **IMAGE COLLECTIONS** [ee.ImageCollection](#ImageCollection) [ee.ImageCollection.fromImages](#fromImages)

AS GOOGLEMAP **OVERLAYS** [image.getMap](#getMap)

AS **IMAGES OF BANDS FOR NEIGHBORS** [image.neighborhoodToBands](#neighborhoodToBands)

AS **IMAGES OF ARRAYS FOR BANDS** [image.toArray](#toArray)

AS **IMAGES OF BANDS FOR ARRAYS** [image.arrayFlatten](#arrayFlatten)

**SUMMARIZING** IMAGES

BY **REGION** [image.reduceRegion](#reduceRegion) image.sampleRegion THIS MUST BE ADDED FOR IMAGE CLASSIFICATION

**COMPARING** IMAGES [ee.Algorithms.IsEqual(image)](#IsEqual)

**DOCUMENTING** IMAGES [ee.Algorithms.Describe](#Describe_getInfo) [image.getInfo](#Describe_getInfo)

[image.toString](#toString_serialize) [image.serialize](#toString_serialize)

**PRESENTING** IMAGES

IN **PRINT** [print(image)](#print_conole) [console.log(image)](#print_conole)

[alert(image)](#alert_confirm) [confirm(image)](#alert_confirm)

IN **MAPS** [Map.addLayer(image)](#addLayer) [image.Visualize](#Visualize) [image.sldStyle](#sldStyle)

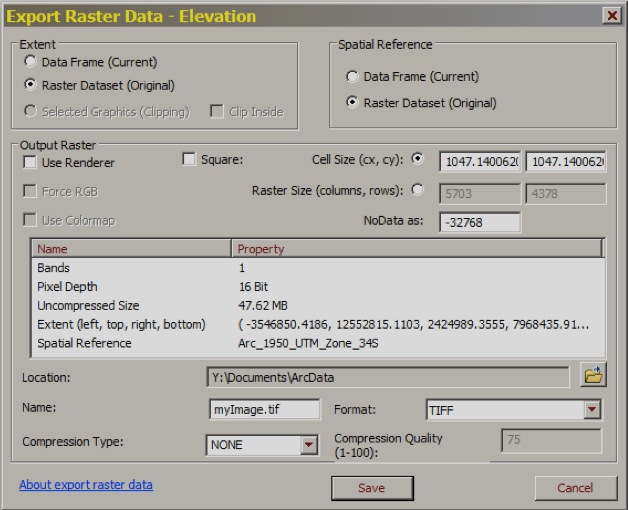
IN **CHARTS** [Chart.image.histogram](#histogram) [Chart.image.byRegion](#byRegion) [Chart.image.regions](#regions)

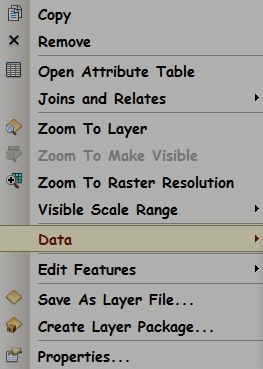
**EXPORTING** IMAGES [Export.image](#Export) [image.getDownloadURL](#getDownloadURL)

[ee.data.getDownloadId](#getDownloadId) [ee.data.getDownloadURL](#getDownloadId)

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**UPLOADING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)



EE images can be generated and stored online by

uploading raster data in any of the following formats:

**.JPEG, .JPG**, **.JPG2**, **.TIF**, **.TIFF**, **.SID**, or **.PNG**.

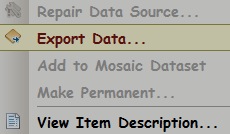
To upload an ArcGIS grid, it must first be exported to

**.TIF** format by right-clicking on its name and using

**Data > Export Data** access a dialog box like this.

The nice thing about that **.TIF**

format is that it will

retain the image’s

geographic position

when uploaded.

To begin the upload

procedure, use Google

Chrome and your

authorized Google Earth

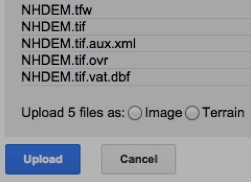
Engine account to access

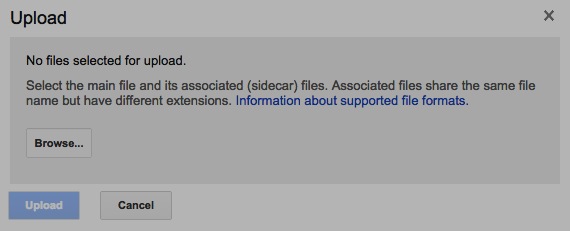
<http://mapsengine.google.com>. Here, you can use

this **Upload** button to open a menu on which

**Files** should be clicked. That will open a dialog box

whose **Browse** button can be used to navigate

 to the folder from which all files associated

 with the image to be

uploaded should be selected

here before clicking on an

**Open** button.

Next, select either **Image** or **Terrain**

(depending on categorical or surficial

nature of the data), and click on this

**Upload** button. That will open a

version of the **Maps Engine** window

that will include this **Access links** button.

A click on that button will yield an Asset ID number that can then be inserted into

a JavaScript statement like this in order to access the now-uploaded image.

var Elevation = ee.Image('GME/images/08039105425737821391-15228977093358892965')

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**ACCESSING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Image creates a new image from an image asset identified by a specified asset ID.

newImage = ee.Image ( assetID)

The new image

The specified asset ID , given as a string

var NewIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

Map.setCenter( -98, 39, 4 );

Map.addLayer( NewIMAGE, {min:0, max: 2500} );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**ACCESSING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Image.load creates a new image from an image asset identified by a specified asset ID and (optional) version.

newImage = ee.Image.load ( assetID, *version* )

The version number. Default: -1 (calling fro the current version

The specified asset ID, given as a string

The new image

var NewIMAGE = ee.Image.load( 'CGIAR/SRTM90\_V4',-1 );

Map.setCenter( -98, 39, 4 );

Map.addLayer(NewIMAGE, {min:0, max: 2500} );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**CREATING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Image and ee.Image.constant both create a new image in which all pixels are set to the same specified value.

newImage = ee.Image( value ) or ee.Image.constant( value )

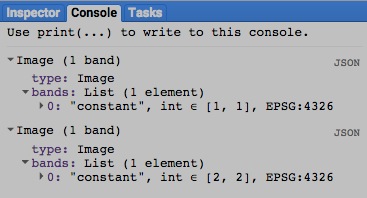
The new image

The specified value

var FirstIMAGE = ee.Image( 1 );

var SecondIMAGE = ee.Image.constant( 2 );

print( FirstIMAGE.getInfo( ), SecondIMAGE.getInfo( ) );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**CREATING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Image.pixelLonLat creates a new two-band image in which each pixel is set to the values of its longitude and latitude in degrees.

newImage = ee.Image.pixelLonLat( )

The new image

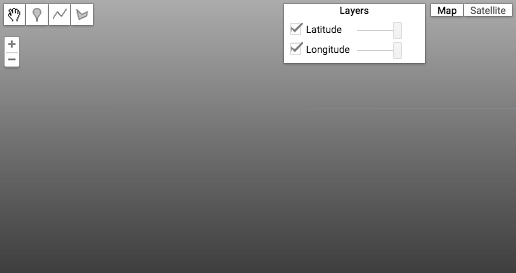
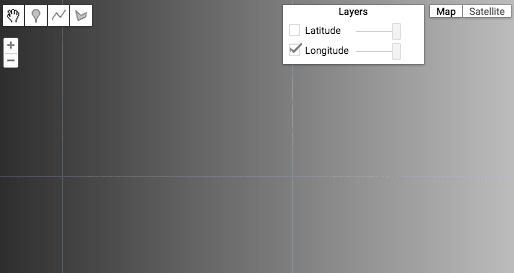
var NewIMAGE = ee.Image.pixelLonLat( );

print( NewIMAGE.getInfo( ) );

Map.addLayer( NewIMAGE, {bands:['longitude'],min:-180,max:180},'Longitude' );

Map.addLayer( NewIMAGE, {bands:['latitude'], min:- 60,max: 60},'Latitude' );

Map.setCenter( 0, 0, 2 );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**CREATING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Image.pixelArea creates a new image in which each pixel is set to a value indicating its area in meters.

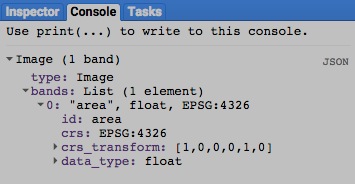
newImage = ee.Image.pixelArea( )

The new image

var NewIMAGE = ee.Image.pixelArea( );

print( NewIMAGE.getInfo( ) );

Map.addLayer( NewIMAGE, {min: 2e8, max: 4e8, opacity: 0.85});



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **MASKING** REGIONS

image.mask creates a new image by applying a specified mask to a specified image. A mask is an image with values ranging from 0 through 1.

When applied to another image, these indicate the degree to which that other images pixels should appear as transparent (0) or opaque (1). . Pixels with mask values of 0 are also eliminated from subsequent processing.

newImage = oldImage.mask( maskImage )

The specified mask

The specified image

The new image

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var MaskIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select(['Land\_Cover\_Type\_1']).neq(9);

var MaskedIMAGE = OldIMAGE.mask( MaskIMAGE );

print( 'Original Image', OldIMAGE );

print( 'Mask Image', MaskIMAGE );

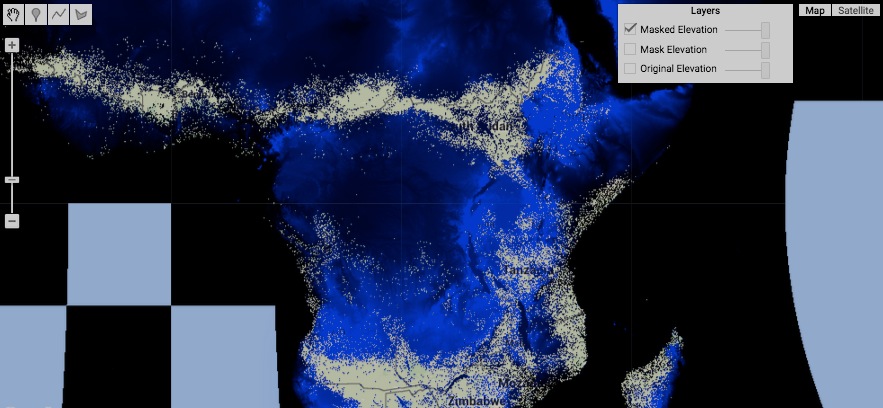
print( 'Masked Image', MaskedIMAGE );

Map.setCenter(26.499, -1, 4);

Map.addLayer( OldIMAGE, { min:0, max:1400, palette:['000000','ff00ff']}, 'Original Elevation' );

Map.addLayer( MaskIMAGE, { min:0, max:1, palette:['ff0000','00ff00']}, 'Mask Elevation' );

Map.addLayer( MaskedIMAGE, { min:0, max:1400, palette:['000000','0000ff']}, 'Masked Elevation' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **CLIPPING** REGIONS

image.clip creates a new image by masking those portions of a specified image lie outside of a region.

Masked pixels will appear as transparent in subsequent displays and eliminated from subsequent processing.

newImage = oldImage.clip( regionToBeKept )

The specified region, given as a geometry, feature, or feature collection

The specified image

The new image

var SatelliteIMAGE = ee.Image( 'LC8\_L1T/LC80140332013216LGN00' ); // Delmarva Penninsula

var StateFEATURES = ee.FeatureCollection( 'ft:1fRY18cjsHzDgGiJiS2nnpUU3v9JPDc2HNaR7Xk8' );

var ClipFEATURE = StateFEATURES.filter( ee.Filter.eq('Name','Delaware') );

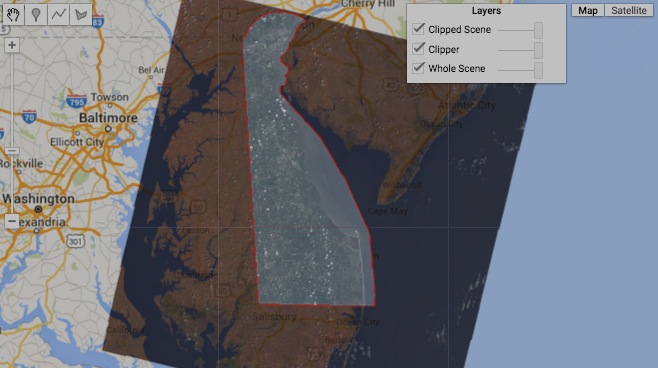
var ClippedIMAGE = SatelliteIMAGE.clip( ClipFEATURE );

Map.centerObject( ClipFEATURE, 8);

Map.addLayer( SatelliteIMAGE, {bands: ['B5','B6','B2'], opacity:0.8}, 'Whole Scene' );

Map.addLayer( ClipFEATURE, {color: 'ff0000'}, 'Clipper' );

Map.addLayer( ClippedIMAGE, {bands: ['B4','B3','B2'], gain:0.015}, 'Clipped Scene' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **SELECTING** BANDS

image.select creates a new image containing only those bands of a specified image that have a specified name, index, or RE2-compatible regex.

Since selections cannot be made from an image collection, such a collection would have to be reduced to an image in order to select bands.

newImage = oldImage.select( bandSelectors, *bandOrder* )

An array of new names to be ascribed to (all of) the

bands of the new image, given as an array of strings.

The specified

image

The specified names, indices, or regexes, given as an array of strings

The new

image

var RedBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B4'] ); // Los Angeles

var GreenBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B3'] );

var BlueBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B2'] );

var MultibandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B4','B3','B2'] );

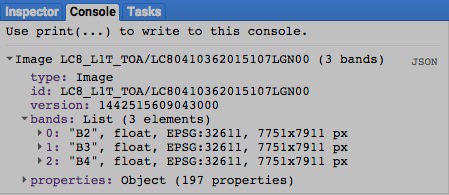
Map.setCenter( -118.2733, 34.0942, 12 );

Map.addLayer( RedBandIMAGE, {min:0, max:0.17, palette:'000000,ff5555'}, 'RednessImage' );

Map.addLayer( GreenBandIMAGE, {min:0, max:0.17, palette:'000000,77ff77'}, 'Greenness Image');

Map.addLayer( BlueBandIMAGE, {min:0, max:0.17, palette:'000000,7777ff'}, 'Blueness Image' );

Map.addLayer( MultibandIMAGE, {min:0, max:0.17, gamma:0.5, bands:'B4,B3,B2'}, 'Multiband Image');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **SELECTING** BANDS

image.slice creates a new image containing only those bands of a specified image that are stored at or after a specified starting position

and before (but not including) a specified stopping position among all such bands.

.

newImage = oldImage.slice( startingPosition, *stoppingPosition* )

The specified stopping position, given as an integer with negative integers counted backwards from the end. Default: the final band

The specified starting position, given as an integer with negative integers counted backwards from the end.

The specified image

The new image

var RedBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B4'] ); // Los Angeles

var GreenBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B3'] );

var BlueBandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B2'] );

var MultibandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).slice( 1,4 );

print( MultibandIMAGE );

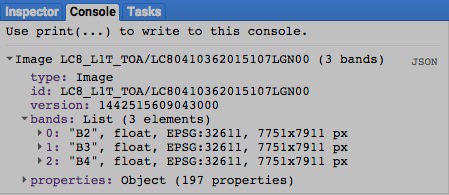
Map.setCenter( -118.2733, 34.0942, 12 );

Map.addLayer( RedBandIMAGE, {min:0, max:0.17, palette:'000000,ff5555'}, 'RednessImage' );

Map.addLayer( GreenBandIMAGE, {min:0, max:0.17, palette:'000000,77ff77'}, 'Greenness Image');

Map.addLayer( BlueBandIMAGE, {min:0, max:0.17, palette:'000000,7777ff'}, 'Blueness Image' );

Map.addLayer( MultibandIMAGE, {min:0, max:0.17, gamma:0.5, bands:'B4,B3,B2'}, 'Multiband Image');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **COMBINING** BANDS

image.addBands creates a new image by replicating the first of two specified images and adding to it

whatever bands of a second specified image have any of a specified set of names.

.

newImage = oldImage1.addBands( oldImage2, *bandNames, overwrite?* )

A Boolean set to True (only) if second image’s bands are to overwrite first image’s bands of the same name. Otherwise, any such band will be renamed by adding the suffix '\_1'

(or '\_2', '\_3', etc.) to its name. Default: False

A list of the names of bands to be added. Default: all bands

The first

specified image

The second

specified image

The new image

var ElevationIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var LandCoverIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' ).select(['Land\_Cover\_Type\_1']).multiply(100) ;

var TwoBandIMAGE = ElevationIMAGE.addBands( LandCoverIMAGE );

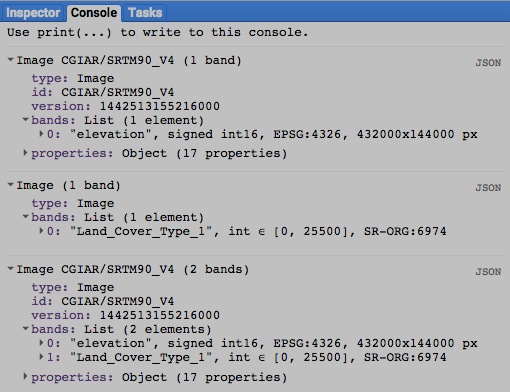
print( ElevationIMAGE.getInfo() ); print( LandCoverIMAGE.getInfo() ); print( TwoBandIMAGE.getInfo() );

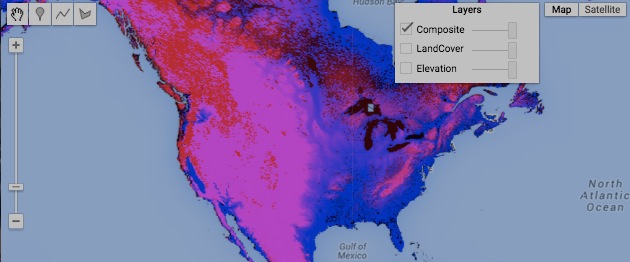
Map.setCenter( -97.12, 39.23, 3 );

Map.addLayer( ElevationIMAGE, {min:0, max:1234, opacity:0.5 , gamma:1}, 'Elevation' );

Map.addLayer( LandCoverIMAGE, {min:0, max:1500, opacity:0.3}, 'LandCover' );

Map.addLayer( TwoBandIMAGE, { bands:['elevation','Land\_Cover\_Type\_1'], gain:0.4, bias:[1,9] }, 'Composite');





[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **REPROJECTING**

image.reproject creates a new image by reprojecting a specified image to a specified coordinate system, geographical placement, and pixel size.

newImage = oldImage.reproject ( *coordinateSystem, placementCoefficients, pixelSize* )

The specified coordinate system, given as an EPSG code ( as described [here](http://spatialreference.org) ) or as a WKT string ( as described [here](http://en.wikibooks.org/wiki/Geospatial_Data_in_SQL_Server/WKT) ). Default: WGS84

The new image

A sequence of six numbers (of type Double) that indicate how the image is to be resized, reoriented, and repositioned with respect to the specified coordinate system. If **pixelSize** is specified, this should not.

A number indicating the width of the new image’s pixels in meters. If **placementCoefficients** is specified, this should not.

The specified image

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = OldIMAGE.reproject( 'EPSG:3191', null, 50000 );

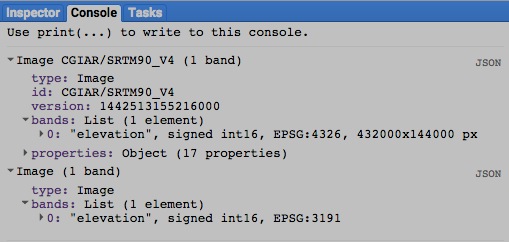
print( OldIMAGE, NewIMAGE );

Map.setCenter ( 10, 10, 3 );

Map.addLayer( OldIMAGE, { min:0, max:2000, palette:['000000','ffffff'] }, 'Original' );

Map.addLayer( NewIMAGE, { min:0, max:1200, palette:['ff0000','00ff00','0000ff'] }, 'Reprojected' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RECODING** COLORS

image.rgbtohsv creates a new image by converting a specified image from RGB to HSV color format, one in which each of three bands

(respectively representing hue, saturation, and value) is represented by a floating point value ranging from 0 to 1.

newImage = oldImage.rgbtohsv( )

The specified image

The new image

var OldIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ).select([3,2,1]).unitScale(0,32767);

var NewIMAGE = OldIMAGE.rgbtohsv( );

print( OldIMAGE.getInfo() , NewIMAGE.getInfo() );

Map.setCenter( -75.8369, 39.4526, 12 );

Map.addLayer( OldIMAGE, { bands:'B4,B3,B2', min:0, max:0.5}, 'Original' );

Map.addLayer( OldIMAGE.select(0), {min:0.1, max:0.8, palette:'ffffff,990000'}, 'Redness' );

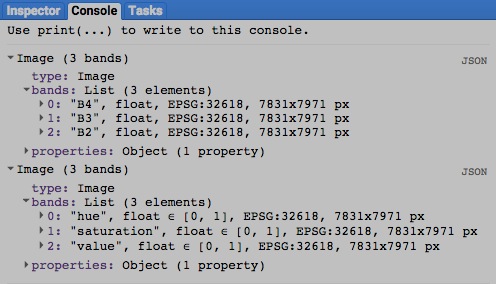
Map.addLayer( OldIMAGE.select(1), {min:0.1, max:0.8, palette:'ffffff,009900'}, 'Greenness' );

Map.addLayer( OldIMAGE.select(2), {min:0.1, max:0.8, palette:'ffffff,000099'}, 'Blueness' );

Map.addLayer( NewIMAGE.select('hue'), {palette:'0000ff,ff0000,00ff00,ff0000,0000ff'}, 'Hue' );

Map.addLayer( NewIMAGE.select('saturation'), {min:0.0,max:0.3, palette:'000000,ff00ff'}, 'Saturation' );

Map.addLayer( NewIMAGE.select('value'), {min:0.2,max:0.4, palette:'000000,ffff00'}, 'Brightness' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RECODING** COLORS

image.hsvtorgb creates a new image by converting a specified image from HSV to RGB color format, one in which each of three bands

(respectively representing red, green, and blue color components) is represented by a floating point value ranging from 0 to 1.

newImage = oldImage.hsvtorgb( )

The specified image

The new image

var OldIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ).select([3,2,1]).unitScale(0,32767);

var HsvIMAGE = OldIMAGE.rgbtohsv( );

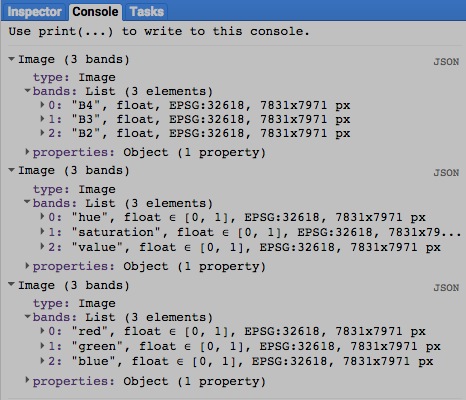
var RgbIMAGE = OldIMAGE.hsvtorgb( );

print( OldIMAGE.getInfo(), HsvIMAGE.getInfo(), RgbIMAGE.getInfo() );

Map.setCenter(-75.8369, 39.4526, 12 );

Map.addLayer( OldIMAGE, { bands:'B4,B3,B2', min:0, max:0.5}, 'Original Image' );

Map.addLayer( RgbIMAGE, { bands:['red','green','blue'], min:0.1, max:0.55}, 'RGB Image' );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RECASTING** VALUES

image.uint8 , .toUint8 , .byte , and .toByte unsigned 8-bit integers

.uint16 and .toUint16 unsigned 16-bit integers

.uint32 and .toUint32 unsigned 32-bit integers

Each of these operations replicates a specified image (of any pixel type) to create a new one whose pixel type is as indicated in red.

.int8 and .toInt8 signed 8-bit integers

.int16 , .toInt16 , .short , and .toShort signed 16-bit integers

.int32 , .toInt32 , .int , and .toInt signed 32-bit integers

.int64 , .toInt64 , .long , and .toLong signed 64-bit integers

.float and .toFloat 32-bit floating-point numbers

.double and .toDouble 64-bit floating-point numbers

newImage = oldImage.uint8( ) or .toUint8( ) or .byte( ) or .toByte( )

or .uint16( ) or .toUint16( )

or .int16( ) or .toInt8( ) or .short( ) or .toShort( )

The new image

or .uint32( ) or .toUint32( )

or .int8( ) or .toInt8( )

The specified image

or .int32( ) or .toInt32( ) or .int( ) or .toInt( )

or .int64( ) or .toInt64( ) or .long( ) or .toLong( )

or .float( ) or .toFloat( )

or .double( ) or .toDouble( )

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RECASTING** VALUES

image.cast creates a new image of specified band-by-band types by replicating a specified image of any type.

newImage = oldImage.cast( bandTypes, *bandOrder* )

A list of (all) band names indicating the order in which they are to be stored. Default: alphabetical order

The specified image

The new image

The specified types, given as a dictionary of band names and corresponding types. Types can be specified as PixelType objects or

as any of the following strings: 'int8', 'int16', 'int32', 'int64', 'uint8', 'uint16', 'uint32', 'byte', 'short', 'int', 'long', 'float', or 'double.'

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.set creates new image by replicating a specified image after setting or resetting one or more specified properties to specified values.

newImage = oldImage.set ( pairsOfPropertiesAndValues )

The specified image

The new image

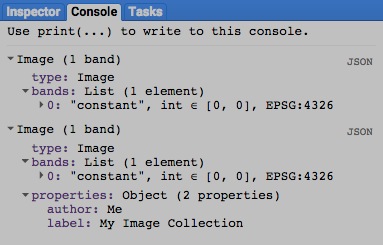
The specified properties and new values, given as a comma-separated sequence

(or a dictionary) of property name strings, each immediately followed by its new value

var OldIMAGE = ee.Image( 0 );

var NewIMAGE = OldIMAGE.set( 'author','Me','label','My Image Collection' );

print( OldIMAGE.getInfo( ), NewIMAGE.getInfo( ) );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.setMulti creates new image by replicating a specified image after setting or resetting one or more specified properties to specified values.

newImage = oldImage.set ( dictionaryOfPropertiesAndValues )

The specified properties and new values, given as a

dictionary of property name strings and new values.

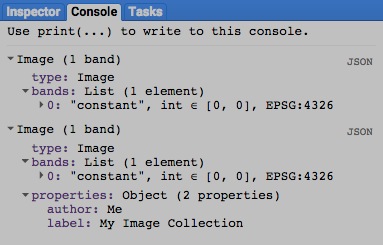
The new image

The specified image

var OldIMAGE = ee.Image( 0 );

var NewIMAGE = OldIMAGE.setMulti( {'author':'Me', 'label':'My Image Collection'} );

print( OldIMAGE.getInfo( ), NewIMAGE.getInfo( ) );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.remap creates a new image on which each pixel’s value is explicitly assigned according to its value on a specified band on a specified image.

newImage = oldImage.remap( listOfOldValues, listOfNewValues, defaultNewValue, bandName )

The new value to be

assumed when none

has been specified.

Default: the mask value

The specified image

The new image

The name of the

specified band,

given as a string.

A list of new values to be

assigned according to those

atcorresponding ordinal

positions in **listOfOldValues**

A list of values from

the specified image

for which new values

are to be assigned

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var NewIMAGE = OldIMAGE.remap( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17],

[0,1,1,1,1,1,2,2,2,3, 3, 4, 5, 6, 5, 7, 8, 9], 0, 'Land\_Cover\_Type\_1' );

print( 'Original Image', OldIMAGE );

print( 'Remapped Image', NewIMAGE );

var ColorsForMODIS = ['aec3d4', // 00 = Water \*/ 'd9903d', // 09 = Savanna

'152106', // 01 = Evergreen Needleleaf Forest \*/ '91af40', // 10 = Grassland

'225129', // 02 = Evergreen Broadleaf Forest \*/ '111149', // 11 = Permanent Wetland

'369b47', // 03 = Deciduous Needleleaf Forest \*/ 'cdb33b', // 12 = Cropland

'30eb5b', // 04 = Deciduous Broadleaf Forest \*/ 'cc0013', // 13 = Urban

'387242', // 05 = Mixed Deciduous Forest \*/ '33280d', // 14 = Crops & Natural Vegetation

'6a2325', // 06 = Closed Shrubland \*/ 'd7cdcc', // 15 = Permanent Snow & Ice

'c3aa69', // 07 = Open Shrubland \*/ 'f7e084', // 16 = Barren / Desert

'b76031', // 08 = Woody Savanna \*/ '6f6f6f' // 17 = Unclassified

].join(',');

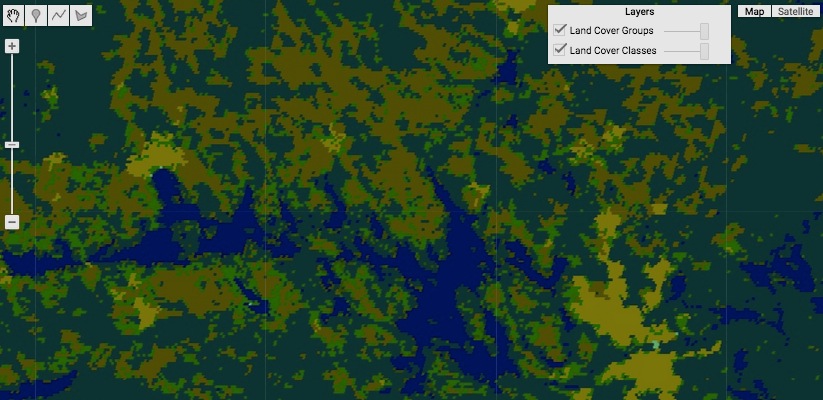
var DisplaySETTINGS = { min:0, max:17, opacity:0.7, palette:ColorsForMODIS };

Map.setCenter(17.3172, 59.5275, 9 );

Map.addLayer( OldIMAGE, DisplaySETTINGS, 'Land Cover Classes' );

Map.addLayer( NewIMAGE, { min:0, max:9, palette:['000066', // 00 = Water \*/ '00ff00', // 05 = Cropland

'008800', // 01 = Forest \*/ 'ff0000', // 06 = Urban

**** '444400', // 02 = Shrubland \*/ 'ffffff', // 07 = Snow & Ice

'aaaa00', // 03 = Savanna \*/ 'ffff00', // 08 = Desert

'00dddd', // 04 = Wetland \*/ '000000' // 09 = Unclassified

] },'Land Cover Groups' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.where creates a new image on which each pixel is set to the value of that pixel on one of two specified images. The choice depends on whether the pixel’s

value on a specified “test”image is - equal to 0, in which case its new value is drawn from the first specified image; or

- other than 0, in which case its new value is drawn from the second specified image.

newImage = nonZeroImage.where( testImage, zeroImage )

The specified image whose values are to be assigned where **testImage** values are other

than 0. This can also be specified as a number to represent an image of constant value.

The specified image whose values

are to be assigned wherever

**testImage** values are equal to 0.

The specified

“test” image

The new

image

var LandcoverPALETTE = ['aec3d4','152106','225129','369b47','30eb5b','387242','6a2325','c3aa69','b76031','d9903d',

'91af40','111149','cdb33b','cc0013','33280d','d7cdcc','f7e084','6f6f6f','000000'].join(',');

var LandcoverIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var ElevationIMAGE = ee.Image('srtm90\_v4');

var UplandIMAGE = ElevationIMAGE.gt( 100 );

var LowlandcoverIMAGE = LandcoverIMAGE.where( UplandIMAGE, 18 );

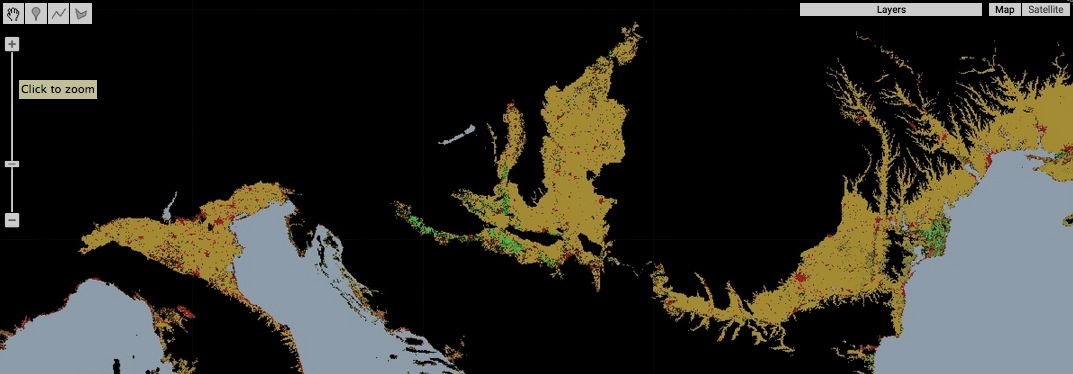
Map.setCenter( 19.644, 45.784, 6 );

Map.addLayer( LandcoverIMAGE, {palette:LandcoverPALETTE,opacity:0.3, min:0, max:18 }, 'Landcover' );

Map.addLayer( ElevationIMAGE, {palette:'000000,ffffff', opacity:0.3, min:0, max:1000}, 'Elevation' );

Map.addLayer( UplandIMAGE, {palette:'ffffff,000000', opacity:0.3, min:0, max:1, }, 'Uplands' );

Map.addLayer( LowlandcoverIMAGE, {palette:LandcoverPALETTE,opacity:1.0, min:0, max:18 }, 'Lowland Landcover' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.metadata creates a new image in which each pixel is set to the (type double) value of a specified numerical property of a specified image.

newImage = oldImage.metadata( property, *newBandName* )

The specified numerical property,

given as a string indicating to its name

The specified

image

A name for the output band, given as a string.

Default: the name of the specified property

The new

image

var OldIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ); // Chesapeake Image

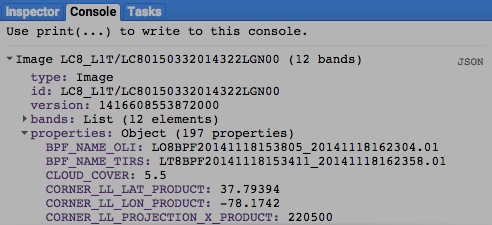
var NewIMAGE = OldIMAGE.metadata('CLOUD\_COVER', 'Cloudiness');

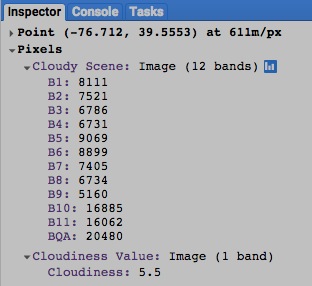
print( OldIMAGE, NewIMAGE );

Map.centerObject( OldIMAGE, 8 );

Map.addLayer( OldIMAGE, { bands:['B4','B3','B2'], gain:0.025}, 'Cloudy Scene' );

Map.addLayer( NewIMAGE, {palette:'ff0000', opacity:0.5}, 'Cloudiness Value' );

****

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.clamp creates a new image on which each pixel is set to its value on a specified image unless that value is greater than a specified maximum

or lower than a specified minimum, in which case the pixel is set to the closest of those specified maximum and minimum values.

newImage = oldImage.clamp( minimumValue, maximumValue )

The specified maximum value

The specified minimum value

The specified image

The new image

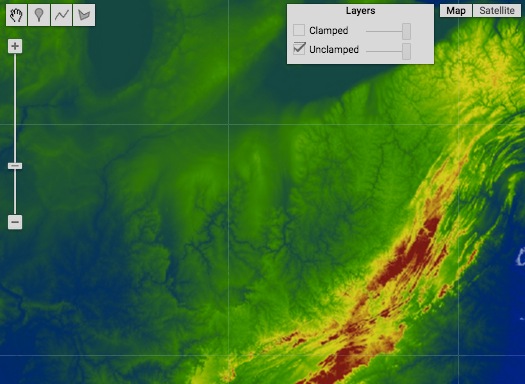
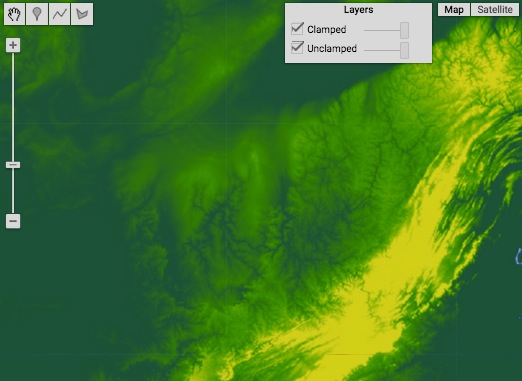
var OldIMAGE = ee.Image('srtm90\_v4');

var NewIMAGE = OldIMAGE.clamp(200, 650);

Map.setCenter( -83.518, 39.504, 6 );

Map.addLayer(OldIMAGE, { min:0, max:1000, palette:['0000aa,00aa00,ffff00,990000'] }, 'Unclamped');

Map.addLayer(NewIMAGE, { min:0, max:1000, palette:['0000aa,00aa00,ffff00,990000'] }, 'Clamped');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.unitScale creates a new image of floating-point values on which each pixel at or below a specified minimum value is set to 0.0,

each pixel at or above a specified maximum value is set to 1.0, and all other pixels are set to values between 0.0 and 1.0

in proportion to their position between those extremes.

newImage = oldImage.unitScale( minimumValue, maximumValue )

The specified maximum

value, given as a Double

The specified minimum

value, given as a Double

The specified image

The new image

var ElevationIMAGE = ee.Image('srtm90\_v4').unitScale( 0,1000 );

var AspectIMAGE = ee.Terrain.aspect( ElevationIMAGE ).unitScale( 0,360 );

var NewIMAGE = ElevationIMAGE.add(AspectIMAGE).divide(2);

Map.setCenter( -83.518, 39.504, 6 );

Map.addLayer(ElevationIMAGE, { min:0, max:1, palette:['0000aa,00aa00,ffff00,990000'] }, 'Elevation');

Map.addLayer(AspectIMAGE, { min:0, max:1, palette:['0000aa,00aa00,ffff00,990000'] }, 'Aspect');

Map.addLayer(NewIMAGE, { min:0, max:1, palette:['0000aa,00aa00,ffff00,990000'] }, 'Elevation-Aspect Mean');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EDITING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **RESETTING** VALUES

image.interpolate creates a new image on which each pixel’s value is calculated as a specified function of its value in the first band of a specified image.

This function is specified by explicitly identifying the output value it associates with each of a selected sample set of input values.

Other output values are then inferred by interpolating or extrapolating from these samples.

newImage = oldImage.interpolate( listOfTypicalInputs, listOfTypicalOutputs, extrapolitionMethod )

The specified output value

samples, given as an array

of Doubles in an order

corresponding to that of

**listOfTypicalOutputs**

The method by which outputs are to be inferred for inputs

above or below the range specified in **listOfTypicalInputs**,

given as one of the following four strings.

- "extrapolate" means extrapolate from the two nearest inputs.

- "clamp" means use the nearest input.

- "input" means use the pixel’s original value.

- "mask" means mask the pixel.

The specified input value

samples, given as an array

of Doubles in strictly

increasing order.

The new image

The specified image

var AllFEATURES = ee.FeatureCollection( 'ft:1G3RZbWoTiCiYv\_LEwc7xKZq8aYoPZlL5\_KuVhyDM' ); // U.S. Cities

var SomeFEATURES = AllFEATURES.filterMetadata('city\_name', 'contains', 'Houston' );

var DistanceIMAGE = SomeFEATURES.distance( 800000 );

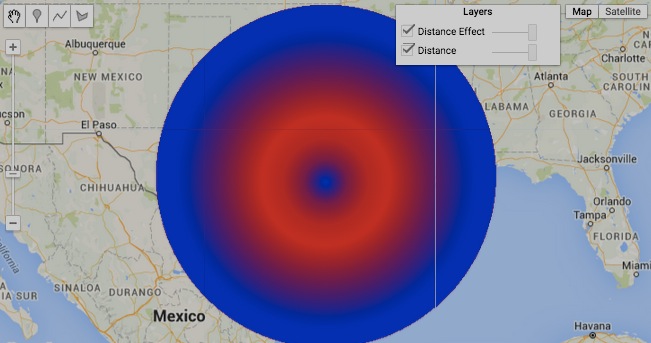
var DistanceEffectIMAGE = DistanceIMAGE.interpolate( [ 0.0, 100000.0, 200000.0, 300000.0, 400000.0 ],

[ 0, 60, 90, 100, 75 ], 'extrapolate' );

Map.centerObject( SomeFEATURES, 5 );

Map.addLayer( DistanceIMAGE, {min:0, max:800000, palette:'0000dd,ff0000'}, 'Distance' );

Map.addLayer( DistanceEffectIMAGE, {min:0, max:100, palette:'0000dd,ff0000'}, 'Distance Effect' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **LOGICAL** OPERATIONS

image.eq , .neq , .gt , .gte, .lt , and .lte create a new grid on which each pixel is set to either 0 or 1 according to whether a specified

relationship between its values on two specified images is either true (1) or false (0).

newImage = 1stImage.eq( 2ndImage ) or .neq( ) or .gt( ) or .gte( ) or .lt( ) or .lte( )

The second specified image

The first specified image

The specified relationship, questioning whether the first specified number is equal to (**eg**), not equal to (**neq**),

greater than (**gt**), greater than or equal to (**gte**), less than (**lt**), or less than or equal to (**lte**) the second.

The new image

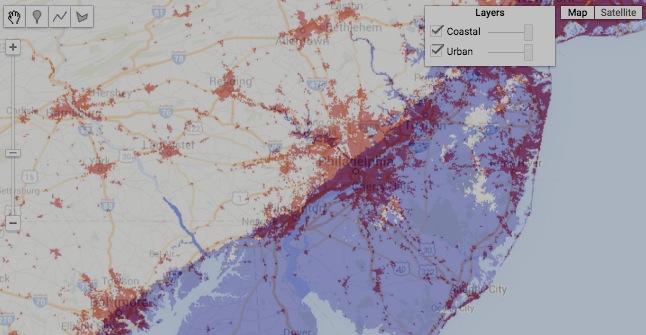
var UrbanIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' ).select(['Land\_Cover\_Type\_1']).eq(13) ;

var CoastalIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' ).lt(50);

Map.setCenter(-75.3662, 39.8802, 8);

Map.addLayer( UrbanIMAGE, {palette:['ffffff,dd0000'],opacity:0.6}, 'Urban' );

Map.addLayer( CoastalIMAGE, {palette:['ffffff,0000dd'],opacity:0.3}, 'Coastal' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **LOGICAL** OPERATIONS

image.and and .or create a new image on which each pixel is set to 0 or 1 according to whether it is true (1) or false (0) that

either (in the case of **or**) or both (in the case of **and**) of its values on two specified images are non-zero (true).

newImage = 1stImage.and( 2ndImage ) or .or( )

The second specified image

The specified relationship, questioning whether *either* of a pixel’s values is non-zero.

The first specified image

The specified relationship, questioning whether *both* of a pixel’s values are non-zero.

The new image

var UrbanIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' ).select(['Land\_Cover\_Type\_1']).eq(13) ;

var CoastalIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' ).lt(50);

var UrbanAndCoastalIMAGE = UrbanIMAGE.and(CoastalIMAGE);

var UrbanOrCoastalIMAGE = UrbanIMAGE.or(CoastalIMAGE);

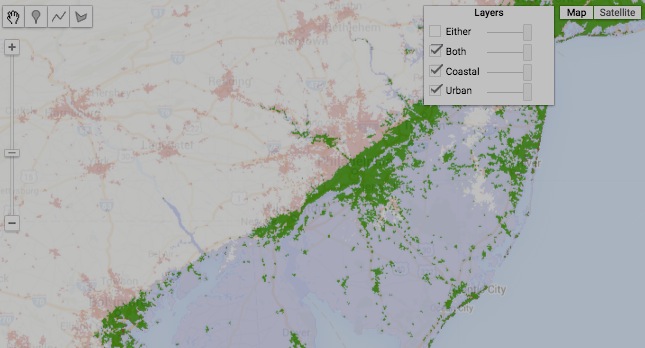
Map.setCenter(-75.3662, 39.8802, 8);

Map.addLayer( UrbanIMAGE, {palette:['ffffff,dd0000'],opacity:0.6}, 'Urban' );

Map.addLayer( CoastalIMAGE, {palette:['ffffff,0000dd'],opacity:0.3}, 'Coastal' );

Map.addLayer( UrbanAndCoastalIMAGE, {palette:['ffffff,00ff00'],opacity:0.6}, 'Both' );

Map.addLayer( UrbanOrCoastalIMAGE, {palette:['ffffff,000000'],opacity:0.3}, 'Either' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **LOGICAL** OPERATIONS

image.not creates a new image on which each pixel is set to 1 if its value on a specified image is 0, or to 0 if its value on that specified image is other than 0.

newImage = oldImage.not( )

The specified image

The new image

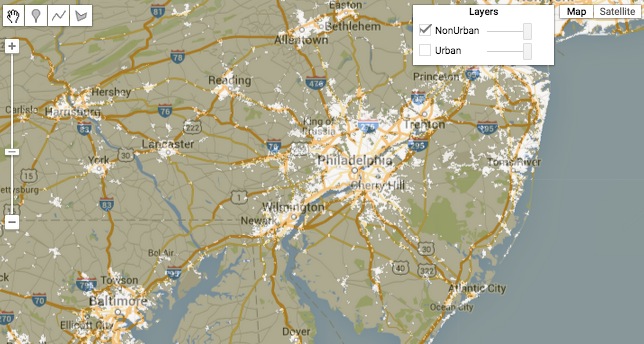
var UrbanIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' ).select(['Land\_Cover\_Type\_1']).eq(13) ;

var NonUrbanIMAGE = UrbanIMAGE.not();

Map.setCenter(-75.3662, 39.8802, 8);

Map.addLayer( UrbanIMAGE, {palette:['ffffff,660000'],opacity:0.6}, 'Urban' );

Map.addLayer( NonUrbanIMAGE, {palette:['ffffff,444400'],opacity:0.4}, 'NonUrban' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **MATHEMATICAL** OPERATIONS

image.abs , .round , .floor , .ceil , .sqrt , .exp , .log, and .log10 all create a new image by applying a specified

mathematical function to the value of each

pixel on each band of a specified image.

newImage = oldImage.abs( ) or .round( ) or .floor( ) or .ceil( ) or .sqrt( ) or .exp( ) or .log( ) or .log10( )

The specified image

The new image

The specified mathematical function

var AllFEATURES = ee.FeatureCollection( 'ft:1G3RZbWoTiCiYv\_LEwc7xKZq8aYoPZlL5\_KuVhyDM' ); // U.S. Cities

var FairbanksFEATURES = AllFEATURES.filterMetadata('city\_name', 'contains', 'Fairbanks' );

var DistanceIMAGE = FairbanksFEATURES.distance( 450000 );

var TransformedIMAGE = DistanceIMAGE.mod(100000);

Map.setCenter( -147.39, 65.15, 5 );

Map.addLayer( DistanceIMAGE, {max:450000, palette:'ff0000,0000ff', opacity:0.8}, 'Distance' );

Map.addLayer( TransformedIMAGE, {max:100000, palette:'ff0000,0000ff' }, 'Modular Distance' );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **MATHEMATICAL** OPERATIONS

image.add , .subtract , .multiply , .divide , .max , .min ,

all create a new image by applying a specified mathematical function the values of corresponding pixels on corresponding bands\* of two specified images.

.mod , .pow , .hypot , .first , and .first\_nonzero

newImage = 1stImage.add( 2ndImage ) or .subtract( ) or .multiply( ) or .divide( ) or .max( ) or .min( )

or .mod( ) or .pow( ) or .hypot( ) or .first( ) or .first\_nonzero( )

The first specified image

The second specified image

The specified mathematical function

The new image, whose

pixel type is defined by

the union of the pixel types

of the two specified images

\* If either of two specified images has only one band, then that band is used against all the bands in the other image.

If the images have the same number of bands, but not the same names, they are used in pairwise order.

The output bands are named for the longer of the two specified images, or the first image if they're equal in length.

var AllFEATURES = ee.FeatureCollection( 'ft:1G3RZbWoTiCiYv\_LEwc7xKZq8aYoPZlL5\_KuVhyDM' ); // U.S. Cities

var FairbanksFEATURES = AllFEATURES.filterMetadata('city\_name', 'contains', 'Fairbanks' );

var AnchorageFEATURES = AllFEATURES.filterMetadata('city\_name', 'contains', 'Anchorage' );

var FairbanksIMAGE = FairbanksFEATURES.distance( 450000 );

var AnchorageIMAGE = AnchorageFEATURES.distance( 450000 );

var MinimumDistanceIMAGE = FairbanksIMAGE.min(AnchorageIMAGE);

var TotalDistanceIMAGE = FairbanksIMAGE.add(AnchorageIMAGE);

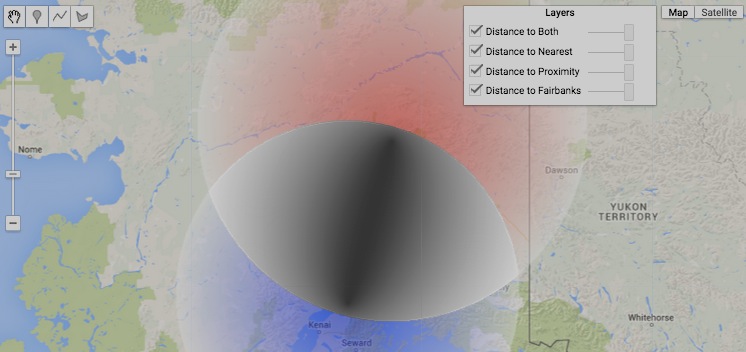
Map.setCenter( -148.623, 63.47, 5 );

Map.addLayer( FairbanksIMAGE, {max:450000, palette:'ff0000,ffffff', opacity:0.5}, 'Distance to Fairbanks' );

Map.addLayer( AnchorageIMAGE, {max:450000, palette:'0000ff,ffffff', opacity:0.5}, 'Distance to Proximity' );

Map.addLayer( MinimumDistanceIMAGE, {max:450000, palette:'444444,ffffff'}, 'Distance to Nearest' );

Map.addLayer( TotalDistanceIMAGE, {min:400000, max:900000, gamma:2.5, }, 'Distance to Both' );

**

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **MATHEMATICAL** OPERATIONS

image.expression creates a new image on which each pixel’s value is calculated by applying a specified arithmetic expression

to variables that correspond to specified bands drawn from one or more specified images.

newImage = oldImage.expression( computationalExpression, *imageVariables* )

A dictionary of ***variableName : image*** pairs, where - ***variableName*** is a user-assigned name, and

- ***image*** is an image

The specified image

The new image

The specified expression, given as a string that may include algebraic operators ( +, -, \*, / ), numerical constants, and references to particular bands of **oldImage**

given as **b( X,X, … )** where each **X** is - a band number such as **b( 0 )**,

- a quoted band name such as **b( 'B4' )**, or

- blank such that **b( )** refers to all available bands**.**

This expression may also refer to the bands of additional images as - **variableName.X** where - **variableName** is as defined by **imageVariables** and

- **X** is an unquoted band name; or as

- **variableName[X,X, … ]** where - **variableName** is as defined by **imageVariables** and

- **X** is a band number or left blank (to reference to all bands).

var FourBandIMAGE = ee.Image( 'NOAA/DMSP-OLS/NIGHTTIME\_LIGHTS/F182012' ); // Night Lighting in 2013

var BandZeroIMAGE = FourBandIMAGE.expression( 'b(0)' ); // Band 0: All Lighting

var BandOneIMAGE = FourBandIMAGE.expression( 'b("stable\_lights")' ); // Band 1: Persistent Lighting

var DifferenceIMAGE = FourBandIMAGE.expression( 'b(0) - b(1)' );

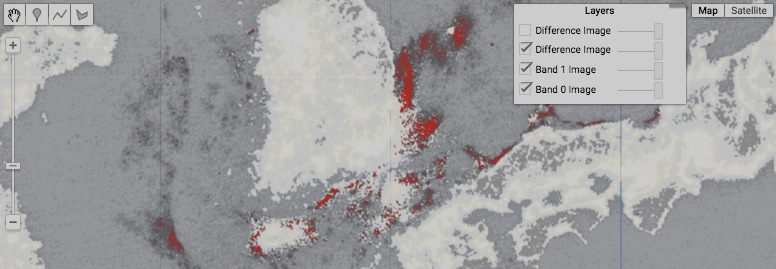
print( FourBandIMAGE, BandZeroIMAGE, BandOneIMAGE, DifferenceIMAGE );

Map.setCenter(129.331, 34.38, 6);

Map.addLayer( BandZeroIMAGE, {min: 0, max: 63, palette: ['000044','ffff00','ffffff'], opacity:0.5}, 'Band 0 Image' );

Map.addLayer( BandOneIMAGE, {min: 0, max: 63, palette: ['000044','ffff00','ffffff'], opacity:0.5}, 'Band 1 Image' );

Map.addLayer( DifferenceIMAGE, {min: 0, max: 15, palette: ['ffffff','777777','ff0000'], opacity:0.9}, 'Difference Image');

var DummyIMAGE = ee.Image(0)

var DifferenceIMAGE = DummyIMAGE.expression(

'A - B',

{ A: FourBandIMAGE.select(0),

B: FourBandIMAGE.select(1)

} );

Map.addLayer( DifferenceIMAGE,

{min: 0, max: 15,

palette: ['ffffff','777777','ff0000'],

opacity:0.9}, 'Difference Image');

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **MATHEMATICAL** OPERATIONS

image.polynomial creates a new image on which each pixel is set to a value calculated by subjecting its value on a specified image to a polynomial equation

of the form ( **a** \* **V0** ) + ( **b** \* **V1** ) + ( **c** \* **V2** ) + ( **d** \* **V3** ) + … where **V** is its value and **a**, **b**, **c**, **d**, … are specified coefficients.

newImage = oldImage.polynomial( listOfCoefficients )

The specified image

The specified coefficients, given as an array of Doubles starting with **a**, then **b**, then **c**, and so on

The new image

var OriginalIMAGE = ee.Image('MOD09GA/MOD09GA\_005\_2012\_03\_09')

.select(['sur\_refl\_b01', 'sur\_refl\_b04', 'sur\_refl\_b03'])

.multiply(0.0001);

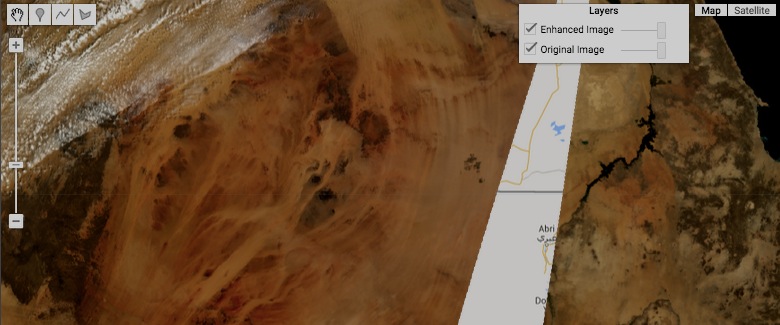
var EnhancedIMAGE = OriginalIMAGE.polynomial([-0.2, 2.4, -1.2]);

// Each pixel of value V is set to -0.2 + (2.4 V) - (1.2 V^2)

Map.setCenter(26.54, 22.35, 6);

Map.addLayer(OriginalIMAGE, {min: 0, max: 1}, 'Original Image');

Map.addLayer(EnhancedIMAGE, {min: 0, max: 1}, 'Enhanced Image');

**

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TRIGONOMETRIC** OPERATIONS

image.sin , .cos , .tan , .sinh , .cosh , .tanh , .acos , .asin , and .atan all create a new image by applying a

specified trigonometric function to the value of each pixel on each band of a specified image.

newImage = oldImage.sin( ) or .cos( ) or .tan( )

or .sinh( ) or .cosh( ) or .tanh( ) or .acos( ) or .asin( ) or .atan( )

The new image

The specified trigonometric function

The specified image, with

values assumed to be in radians

var YshiftIMAGE = ee.Image.pixelLonLat( ).select( 'latitude' );

var XshiftIMAGE = ee.Image.pixelLonLat( ).select( 'longitude' );

var AngleIMAGE = XshiftIMAGE.atan2( YshiftIMAGE );

var SinIMAGE = AngleIMAGE.sin( );

var CosIMAGE = AngleIMAGE.cos( );

Map.setCenter( 0, 0, 14 );

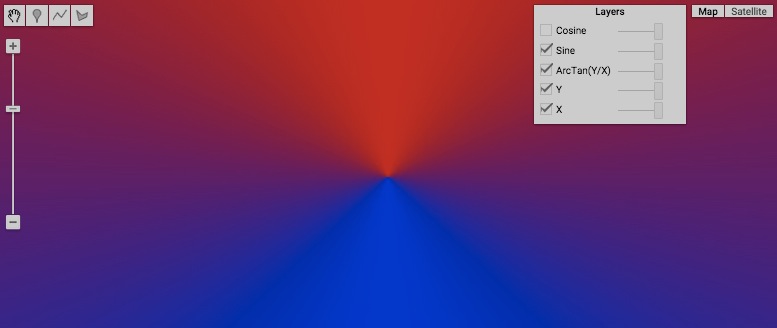
Map.addLayer( XshiftIMAGE, {min:-0.02,max:0.02, palette:'0000ff, ff0000'}, 'X' );

Map.addLayer( YshiftIMAGE, {min:-0.01,max:0.01, palette:'0000ff, ff0000'}, 'Y' );

Map.addLayer( AngleIMAGE, {min:-3.14,max:3.14, palette:'0000ff, ff0000'}, 'ArcTan(Y/X)');

Map.addLayer( SinIMAGE, {min:-1, max:1 , palette:'0000ff, ff0000'}, 'Sine' );

Map.addLayer( CosIMAGE, {min:-1, max:1 , palette:'0000ff, ff0000'}, 'Cosine' );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TRIGONOMETRIC** OPERATIONS

image.atan2 creates a new image on which each pixel’s value is computed as the arctangent (angle whose tangent matches the ratio) of two specified numbers:

- a numerator drawn from the value of the corresponding pixel of the corresponding band\* of one specified image, and

- a denominator drawn from the value of the corresponding pixel of the corresponding band of another specified image.

newImage = 1stImage.atan2( 2ndImage )

\* If either of two specified images has only one band, then

that band is used against all the bands in the other image.

If the images have the same number of bands, but not the

same names, they are used in pairwise order.

The output bands are named for the longer of the two

specified images, or the first image if they're equal in length.

The specified denominator image

The specified numerator image

The new image, whose values are given in radians as floating-point numbers

var YshiftIMAGE = ee.Image.pixelLonLat().select('latitude');

var XshiftIMAGE = ee.Image.pixelLonLat().select('longitude');

var AngleIMAGE = XshiftIMAGE.atan2( YshiftIMAGE );

Map.setCenter( 0, 0, 14 );

Map.addLayer( XshiftIMAGE, {min:-0.02,max:0.02, palette:'0000ff, ff0000'}, 'X' );

Map.addLayer( YshiftIMAGE, {min:-0.01,max:0.01, palette:'0000ff, ff0000'}, 'Y' );

Map.addLayer( AngleIMAGE, {min:-3.14,max:3.14, palette:'0000ff, ff0000'}, 'ArcTan(Y/X)');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **BITWISE** OPERATIONS

image.bitwiseAnd , .bitwiseOr , .bitwiseXor , .bitwiseNot

all create a new image by applying a

specified bitwise function the values

of corresponding pixels on corresponding bands\* of two specified images.

.bitwise\_and , .bitwise\_or , .bitwise\_xor , and .bitwise\_not

newImage = 1stImage.bitwiseAnd( 2ndImage ) or .bitwiseOr( ) or .bitwiseXOr( ) or .bitwiseNot( )

or .bitwise\_and( ) or .bitwise\_or( ) or .bitwise\_xor( ) or .bitwise\_not( )

The second specified image

The first specified image

The specified bitwise function

The new image

\* If either of two specified images has only one band, then that band is used against all the bands in the other image.

If the images have the same number of bands, but not the same names, they are used in pairwise order.

The output bands are named for the longer of the two specified images, or the first image if they're equal in length.

var FirstIMAGE = ee.Image( 1 );

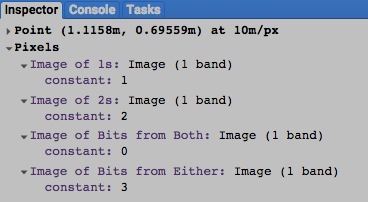
var SecondIMAGE = ee.Image( 2 );

var BothBitIMAGE = FirstIMAGE.bitwiseAnd( SecondIMAGE );

var EitherBitIMAGE = FirstIMAGE.bitwise\_or( SecondIMAGE );

print ( 'Image of bitwise 1s and 2s =', BothBitIMAGE );

print ( 'Image of bitwise 1s or 2s =', EitherBitIMAGE );

Map.addLayer( FirstIMAGE, null, 'Image of 1s' );

Map.addLayer( SecondIMAGE, null, 'Image of 2s' );

Map.addLayer( BothBitIMAGE, null, 'Image of Bits from Both' );

Map.addLayer( EitherBitIMAGE, null, 'Image of Bits from Either' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **BITWISE** OPERATIONS

image.leftShift , .rightShift , left\_shift , and .right\_shift create a new image by shifting the bits of the values of all

pixels of a specified image left or right by a specified amount.

newImage = oldImage.leftShift( numberOfPositions ) or .rightShift( )

The specified amount, given as an integer referring to bit positions

The specified image

The specified function, indicating whether bits are to be shifted left or right

The new image

var OriginalIMAGE = ee.Image( 32 ).int();

var LeftShiftedIMAGE = OriginalIMAGE.leftShift( 1 );

var RightShiftedIMAGE = OriginalIMAGE.rightShift( 1 );

print( 'Original Image', OriginalIMAGE );

print( 'Left-shifted Image', LeftShiftedIMAGE );

print( 'Right-shifted Image', RightShiftedIMAGE );

Map.addLayer( OriginalIMAGE, null, 'Original Image' );

Map.addLayer( LeftShiftedIMAGE, null, 'Left-shifted Image' );

Map.addLayer( RightShiftedIMAGE, null, 'Right-shifted Image' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **PIXEL** REDUCERS

image.reduce creates a new image on which each pixel’s value is calculated by applying a specified reducer to its values on all bands of a specified image.

newImage = oldImage.reduce( reducer )

The specified reducer

The specified image

The new image

var TheREDUCER = ee.Reducer.std\_dev( );

var MultibandIMAGE = ee.Image( 'LC8\_L1T\_TOA/LC80410362015107LGN00' ).select( ['B4','B3','B2'] ); // Los Angeles

var ReducedIMAGE = MultibandIMAGE.reduce( TheREDUCER );

Map.setCenter( -118.5054, 34.2016, 10 );

Map.addLayer( MultibandIMAGE, {min:0, max:0.17, gamma:0.5, bands:'B4,B3,B2'}, 'Multiband Image' );

Map.addLayer( ReducedIMAGE, {min:0, max:0.015, palette:'000000,ffffff'}, 'Deviation Image' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage,**arrayAccum(axis, *reducer*)**

Accumulates elements of each array pixel along the given axis, by setting each element of the result array pixel to the reduction of elements in that pixel along the given axis, up to and including the current position on the axis. May be used to make a cumulative sum, a monotonically increasing sequence, etc.

oldImage Input image.

axis (Integer): Axis along which to perform the cumulative sum.

*reducer (Reducer, default: null):* Reducer to accumulate values. Default is SUM, to produce the cumulative sum of each vector along the given axis.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage.**arrayFlatten(coordinateLabels, *separator*)**

Converts a single band image of equal-shape multidimensional pixels to an image of scalar pixels, with one band for each element of the array.

oldImage Image of multidimensional pixels to flatten.

coordinateLabels (List): Name of each position along each axis. For example, 2x2 arrays with axes meaning 'day' and 'color' could have labels like [['monday', 'tuesday'], ['red', 'green']], resulting in band names'monday\_red', 'monday\_green', 'tuesday\_red', and 'tuesday\_green'.

*separator (String, default: "\_"):* Separator between array labels in each band name.

var MosaickedIMAGE = ee.ImageCollection( 'LT5\_L1T' ).filterDate( '2007-06-01','2007-09-30' ).median();

var ArrayedIMAGE = MosaickedIMAGE.toArray(); //Set each pixel to a 1D array of its 7 band values

var ArrayedIMAGE = ArrayedIMAGE.toArray(1); //Set each pixel to a 2D array of its 7x7 band values

var UrbanARRAY = [88,42,48,38,86,115,59]; //Urban endmember generated by sampling MosaickedIMAGE

var VegetARRAY = [50,21,20,35,50,110,23]; //Veg endmember generated by sampling MosaickedIMAGE

var WaterARRAY = [51,20,14, 9, 7,116, 4]; //Water endmember generated by sampling MosaickedIMAGE

var EndmemberARRAY = ee.Array( [UrbanARRAY, VegetARRAY, WaterARRAY] );

var InverseArrayIMAGE = ee.Image( EndmemberARRAY.matrixPseudoInverse().transpose() );

// Set each pixel to 3x1 array of endmembers fractions ranging from 0 to 1

var UnmixedArrayIMAGE = InverseArrayIMAGE.matrixMultiply( ArrayedIMAGE );

var Unmixed2dIMAGE = UnmixedArrayIMAGE.arrayProject([0]); //Project to axis 0 since axis 1 unnecessary

var Unmixed1dIMAGE = Unmixed2dIMAGE.arrayFlatten([['urban', 'veg', 'water']]); //Flatten to scalar image

Map.setCenter(-98.4, 19, 11);

Map.addLayer( MosaickedIMAGE, null, 'Raw Imagery' );

Map.addLayer( Unmixed1dIMAGE, {min: 0, max: 1}, 'Unmixed (red=urban, green=veg, blue=water)' );

Map.addLayer( ee.Algorithms.Terrain(ee.Image('srtm90\_v4')).select('hillshade'), {opacity:0.5} );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage.**arrayGet(position)**

For each band, an output band of the same name is created with the value at the given position extracted from the input multidimensional pixel in that band.

oldImage Array to get an element from.

position (Image): The coordinates of the element to get.

There must be as many scalar bands as there are dimensions in the input image.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage.**arrayLengths()**

Returns a 1D array image with the length of each array axis.

oldImage Input image.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage.**arrayMask(mask)**

Creates an array image where each array element is masked to those where the corresonding mask position is non-zero. If the mask image has one band it will be applied to all the bands of 'input', otherwise they must have the same number of bands.

oldImage Array image to mask.

mask (Image): Array image to mask with.

swap

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = oldImage.**arrayProject(axes)**

Projects the array in each pixel to a lower dimensional space by specifying the axes to retain. Dropped axes must be at most length 1.

oldImage Input image.

axes (List): The axes to retain. Other axes will be discarded and must be at most length 1.

var MosaickedIMAGE = ee.ImageCollection( 'LT5\_L1T' ).filterDate( '2007-06-01','2007-09-30' ).median();

var ArrayedIMAGE = MosaickedIMAGE.toArray(); //Set each pixel to a 1D array of its 7 band values

var ArrayedIMAGE = ArrayedIMAGE.toArray(1); //Set each pixel to a 2D array of its 7x7 band values

var UrbanARRAY = [88,42,48,38,86,115,59]; //Urban endmember generated by sampling MosaickedIMAGE

var VegetARRAY = [50,21,20,35,50,110,23]; //Veg endmember generated by sampling MosaickedIMAGE

var WaterARRAY = [51,20,14, 9, 7,116, 4]; //Water endmember generated by sampling MosaickedIMAGE

var EndmemberARRAY = ee.Array( [UrbanARRAY, VegetARRAY, WaterARRAY] );

var InverseArrayIMAGE = ee.Image( EndmemberARRAY.matrixPseudoInverse().transpose() );

// Set each pixel to 3x1 array of endmembers fractions ranging from 0 to 1

var UnmixedArrayIMAGE = InverseArrayIMAGE.matrixMultiply( ArrayedIMAGE );

var Unmixed2dIMAGE = UnmixedArrayIMAGE.arrayProject([0]); //Project to axis 0 since axis 1 unnecessary

var Unmixed1dIMAGE = Unmixed2dIMAGE.arrayFlatten([['urban', 'veg', 'water']]); //Flatten to scalar image

Map.setCenter(-98.4, 19, 11);

Map.addLayer( MosaickedIMAGE, null, 'Raw Imagery' );

Map.addLayer( Unmixed1dIMAGE, {min: 0, max: 1}, 'Unmixed (red=urban, green=veg, blue=water)' );

Map.addLayer( ee.Algorithms.Terrain(ee.Image('srtm90\_v4')).select('hillshade'), {opacity:0.5} );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = **arrayReduce(reducer, axes, *fieldAxis*)**

oldImage.Reduces elements of each array pixel.

oldImage Input image.

reducer (Reducer): The reducer to apply

axes (List): The list of array axes to reduce in each pixel.

The output will have a length of 1 in all these axes.

*fieldAxis (Integer, default: null):* The axis for the reducer's input and output fields.

Only required if the reducer has multiple inputs or outputs.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = **oldImage.arrayRepeat(axis, copies)**

Repeats each array pixel along the given axis. Each output pixel will have the shape of the input pixel, except length along the repeated axis, which will be multiplied by the number of copies.

oldImage Image of array pixels to be repeated.

axis (Integer): Axis along which to repeat each pixel's array.

copies (Image): Number of copies of each pixel.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = **oldImage.arraySlice(*axis*, *start*, *end*, *step*)**

Creates a subarray by slicing out each position along the given axis from the 'start' (inclusive) to 'end' (exclusive) by increments of 'step'. The result will have as many dimensions as the input, and the same length in all directions except the slicing axis, where the length will be the number of positions from 'start' to 'end' by 'step' that are in range of the input array's length along 'axis'. This means the result can be length 0 along the given axis if start=end, or if the start or end values are entirely out of range.

oldImage Input array image.

*axis (Integer, default: 0):* Axis to subset.

*start (Image, default: null):* The coordinate of the first slice (inclusive) along 'axis'. Negative numbers are used to position the start of slicing relative to the end of the array, where -1 starts at the last position on the axis, -2 starts at the next to last position, etc. There must one band for start indices, or one band per 'input' band. If this argument is not set or masked at some pixel, then the slice at that pixel will start at index 0.

*end (Image, default: null):* The coordinate (exclusive) at which to stop taking slices. By default this will be the length of the given axis. Negative numbers are used to position the end of slicing relative to the end of the array, where -1 will exclude the last position, -2 will exclude the last two positions, etc. There must be one band for end indices, or one band per 'input' band. If this argument is not set or masked at some pixel, then the slice at that pixel will end just after the last index.

*step (Integer, default: 1):* The separation between slices along 'axis'; a slice will be taken at each whole multiple of 'step' from 'start' (inclusive) to 'end' (exclusive). Must be positive.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = **oldImage.arraySort(*keys*)**

Sorts elements of each array pixel along one axis.

oldImage Array image to sort.

*keys (Image, default: null):* Optional keys to sort by. If not provided, the values are used as the keys.

The keys can only have multiple elements along one axis, which determines the direction to sort in.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ARRAY** OPERATIONS

newImage = **oldImage.arrayTranspose(*axis1*, *axis2*)**

Transposes two dimensions of each array pixel.

oldImage Input image.

*axis1 (Integer, default: 0):* First axis to swap.

*axis2 (Integer, default: 1):*

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **INSULARITY** OPERATIONS

image.connectedComponents creates a new image on which each pixel is set to a unique (but otherwise arbitrary) value identifying it

as part of a particular “connected component” (i.e. island, fragment, conterminous cluster, …) of pixels.

The pixels of each component are those that share a common value on a specified image and are connected to one another (either directly of through a series of connections) by lying within a common neighborhood as defined by a specified kernel. Pixels of components larger than a specified maximum size are masked.

newImage = oldImage.connectedComponents( kernel, maximumSize )

The new image

The specified kernel

The specified maximum component size,

given as a number of pixels no greater than 256

The specified image

Note how changes in pixel size when zooming make this a disquietingly fluid operation

var TheKERNEL = ee.Kernel.square( 3, 'pixels' );

var ElevationIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var MountainIMAGE = ElevationIMAGE.gt( 6000 );

var MountainAreaIMAGE = MountainIMAGE.connectedPixelCount( 1024, true );

var ComponentIMAGE = MountainIMAGE.connectedComponents( TheKERNEL, 256 );

print( ComponentIMAGE );

Map.setCenter( 86.2234, 28.1846, 10 );

Map.addLayer( ElevationIMAGE, {min:0, max:6000, opacity:0.5}, 'Elevation' );

Map.addLayer( MountainAreaIMAGE, {min:0, max:100, palette:'ff0000,0000ff', opacity:0.5}, 'Mountain Area' );

Map.addLayer( ComponentIMAGE.select('labels'),{palette:'ff0000,00ff00'}, 'Components' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **INSULARITY** OPERATIONS

image.connectedPixelCount creates a new image on which each pixel is set to a value indicating the number of pixels within its

“connected component” (i.e. island, fragment, conterminous cluster, …) on a specified image. The pixels

of each component are those that share a common value on a specified image and are connected to one another through a series of lateral or (optionally) diagonal adjacencies. Pixels of components larger than a specified maximum size are masked.

newImage = oldImage.connectedPixelCount( *maximumSize*, *diagonalConnections?* )

The new image

The specified maximum component size,

given as a number of pixels. Default: 100

A Boolean set to true (only) if diagonally adjacent pixels

are to be regarded as potentially connected. Default: true

The specified image

var TheKERNEL = ee.Kernel.square( 3, 'pixels' );

var ElevationIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var MountainIMAGE = ElevationIMAGE.gt( 6000 );

var MountainAreaIMAGE = MountainIMAGE.connectedPixelCount( 1024, true );

var ComponentIMAGE = MountainIMAGE.connectedComponents( TheKERNEL, 256 );

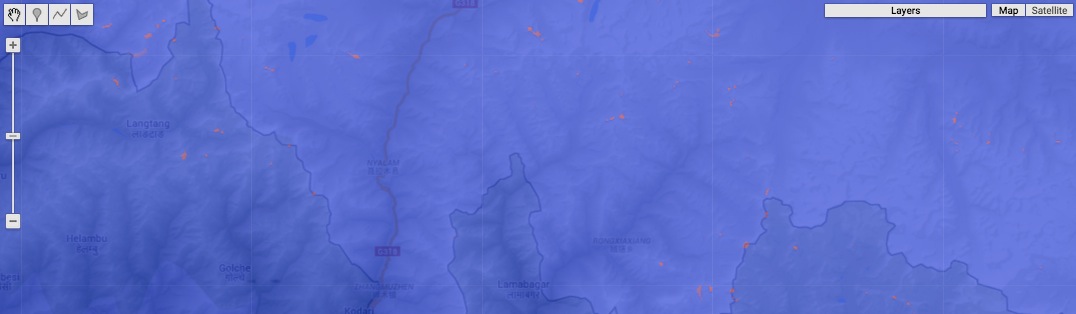
print( ComponentIMAGE );

Map.setCenter( 86.2234, 28.1846, 10 );

Map.addLayer( ElevationIMAGE, {min:0, max:6000, opacity:0.5}, 'Elevation' );

Map.addLayer( MountainAreaIMAGE, {min:0, max:100, palette:'ff0000,0000ff', opacity:0.5}, 'Mountain Area' );

Map.addLayer( ComponentIMAGE.select('labels'),{palette:'ff0000,00ff00'}, 'Components' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

image.derivative creates a new image on which each pixel is set to two values on two separate bands. The first of these two values indicates the rate at which

the values of a specified image change over the distance from that pixel’s left neighbor to its right neighbor. The second does then same

between its upper and lower neighbors.

newImage = oldImage.derivative( )

The specified image

The new image

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = OldIMAGE.derivative( );

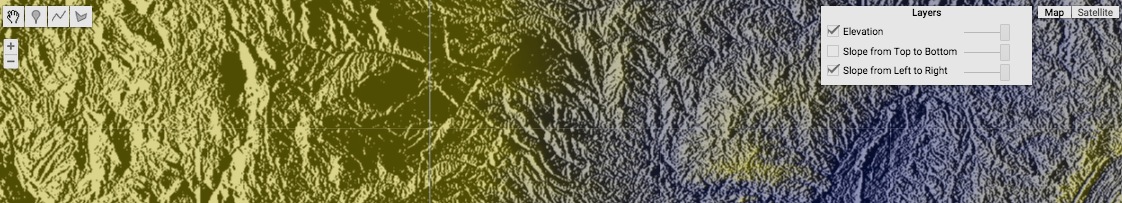
Map.setCenter( -98.042, 36.862, 6 );

Map.addLayer( NewIMAGE, { bands:['elevation\_x'] }, 'Slope from Left to Right' );

Map.addLayer( NewIMAGE, { bands:['elevation\_y'] }, 'Slope from Top to Bottom' );

Map.addLayer( OldIMAGE, {min:0, max:700, opacity:0.4, palette:'000099,dddd00'}, 'Elevation' );





[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.products and ee.Algorithms.Terrain

both create a new image on which each pixel is set to three values characterizing its position on a three-dimensional surface. The third-dimensional position of each pixel on that surface is defined by its value on a specified image usually (though not necessarily) representing a digital elevation model (DEM) of topographic heights above sea level.

ee.Terrain.products or

newImage = ee.Algorithms.Terrain( oldImage)

The specified image, given either as a single-band image or a multi-band image with a band named ‘elevation’ (whose values are then processed like those of a single-band image), and its values are assumed to be in meters.

The new image, including three bands that respectively record surficial slope in degrees, aspect (in clockwise degrees from north), and relief shading (from 0 to 255 – dark to light). This new image will also include all other bands and metadata from the specified image. Because new values are computed from each pixel’s four lateral neighbors, pixels at the edges of new images are masked.

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = ee.Terrain.products( OldIMAGE );

Map.setCenter( 11.096, 46.845, 7 );

Map.addLayer( OldIMAGE,{ min:0,max:2000,opacity:0.5,palette:'000099,dddd00'}, 'Elevation');

Map.addLayer( NewIMAGE,{bands:['slope'], min:0,max:90, opacity:0.8,palette:'000000,ff0000' }, 'Slope');

Map.addLayer( NewIMAGE,{bands:['aspect'], opacity:0.5,palette:'00ff00,0000ff,00ff00' }, 'Aspect');

Map.addLayer( NewIMAGE,{bands:['hillshade'],min:0,max:255,opacity:0.5 }, 'Hillshading');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.slope creates a new image on which each pixel’s value indicates its steepness on a three-dimensional terrain

defined by third-dimensional elevation values drawn from a specified image.

newImage = ee.Terrain.slope( oldImage )

The specified image, whose values are assumed to be in meters

The new image, whose values are in degrees

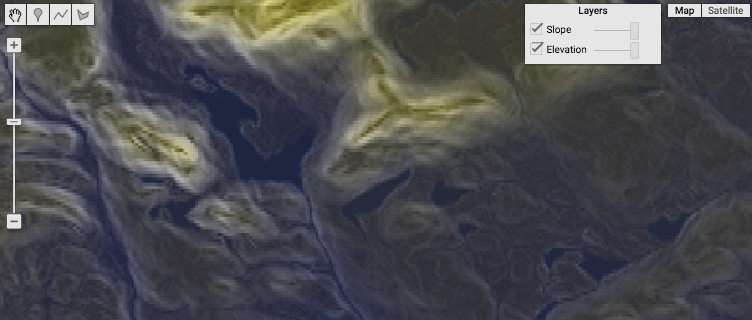
var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = ee.Terrain.slope( OldIMAGE );

Map.setCenter( -69.3299, 45.3907, 12 );

Map.addLayer( OldIMAGE, {min:0, max:700, opacity:0.8, palette:'000099,dddd00'}, 'Elevation' );

Map.addLayer( NewIMAGE, {min:0, max: 30, opacity:0.5}, 'Slope' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.aspect creates a new image on which each pixel’s value indicates its aspect on a three-dimensional terrain

defined by third-dimensional elevation value drawn from a specified image.

newImage = ee.Terrain.aspect( oldImage )

The specified image, whose values are assumed to be in meters

The new image, whose values are

in clockwise degrees from north

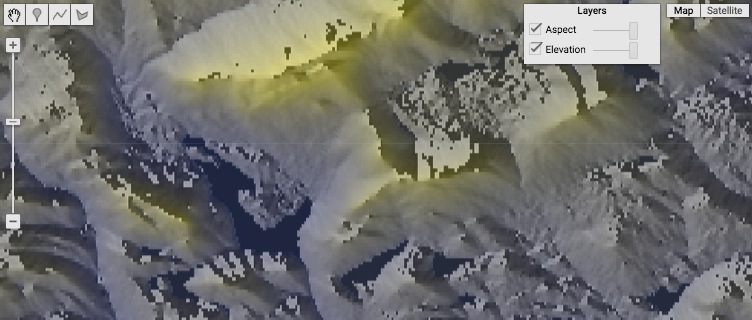
var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = ee.Terrain.aspect( OldIMAGE );

Map.setCenter( -69.3299, 45.3907, 12 );

Map.addLayer( OldIMAGE, {min:0, max:700, opacity:0.8, palette:'000099,dddd00'}, 'Elevation' );

Map.addLayer( NewIMAGE, {opacity:0.5}, 'Aspect' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.hillshade creates a new image on which each pixel’s value indicates its relief shading on a three-dimensional terrain

defined by third-dimensional elevation values drawn from a specified image.

newImage = ee.Terrain.hillshade( oldImage*, sunAngleClockwiseFromNorth, sunAngleAboveHorizon* )

The altitudinal sun angle, given

in degrees above the horizon.

Default: 45

The azimuthal sun angle, given

in clockwise degrees from north.

Default: 270

The specified image, whose values

are assumed to be in meters

The new image, whose values range

from 0 (shaded) to 255 (unshaded)

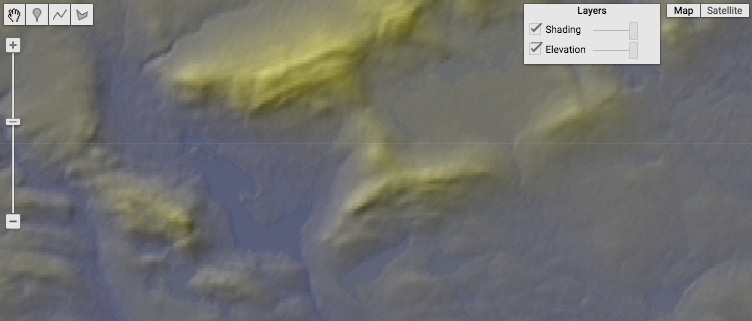
var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var NewIMAGE = ee.Terrain.hillshade( OldIMAGE, 315, 30 );

Map.setCenter( -69.3299, 45.3907, 12 );

Map.addLayer( OldIMAGE, {min:0, max:700, opacity:0.8, palette:'000099,dddd00'}, 'Elevation' );

Map.addLayer( NewIMAGE, {min:0, max:256, opacity:0.5}, 'Shading' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.hillshadow and ee.Algorithms.Hillshadow

both create a new image on which each pixel’s value indicates

whether it lies in sun or shade on a three-dimensional terrain

defined by third-dimensional elevation values drawn from

a specified image, given specified sun angles.

Terrain or

newImage = ee.Algorithms.hillshadow( oldImage*,* azimuthAngle, zenithAngle, neighborhood, hysteresis )

The parameter that I don’t understand but which results in shorter shadows as it gets larger, given as an integer, Default: 0

The new image, whose values

are either 0 (shadow) or 1 (sun)

The specified image, whose values are assumed to be

in meters and which must employ a Mercator CRS.

A Boolean set to true

(only) if computation

is to incorporate past experience in ways that may reduce accuracy but improve appearance

The azimuth sun angle, given as a floating-point number indicating clockwise degrees from north

The zenith sun angle, given as a floating-point number indicating degrees above the horizon

var OriginalIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var ProjectedIMAGE = OriginalIMAGE.reproject( 'EPSG:3857', null, 30 );

var ShadowIMAGE1 = ee.Terrain.hillShadow( ProjectedIMAGE, 315.0, 75.0, 0, true );

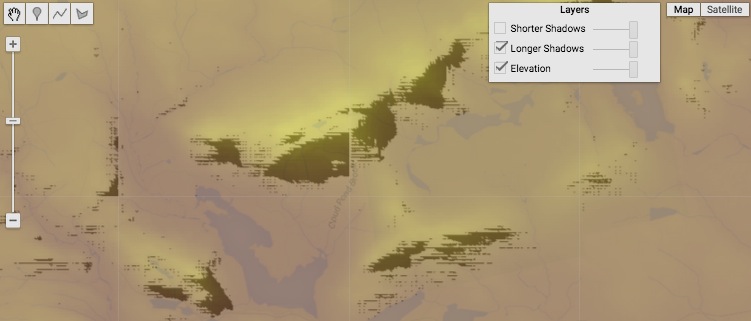
var ShadowIMAGE2 = ee.Terrain.hillShadow( ProjectedIMAGE, 315.0, 70.0, 300, true );

Map.setCenter( -69.3354, 45.4052, 12 );

Map.addLayer( ProjectedIMAGE, {min:0, max:800, opacity:0.7, palette:'550000,dddd00'}, 'Elevation' );

Map.addLayer( ShadowIMAGE1, {min:0, max:1, opacity:0.3}, 'Longer Shadows' );

Map.addLayer( ShadowIMAGE2, {min:0, max:1, opacity:0.3}, 'Shorter Shadows' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TERRAIN** OPERATIONS

ee.Terrain.fillMinima creates a new image by copying an image on which each pixel’s value indicates its third-dimension elevation on a three-dimensional

terrain, then filling each of that terrain’s “sinks” (i.e. basins with no outflow) such that it drains toward its point of lowest elevation.

newImage = ee.Terrain.fillMinima( oldImage*, borberValue, neighborhood* )

The radius of the circular neighborhood,

given as an integral number of pixels widths.

The new image, whose

values must be integers

The specified image

The specified border

value. Default: 50

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var ElevationIMAGE = OldIMAGE.int16();

var FilledIMAGE = ee.Terrain.fillMinima( ElevationIMAGE, 0, 50 );

Map.setCenter(-69.3299, 45.3907, 12 );

Map.addLayer( ElevationIMAGE, {min:0, max:700, opacity:0.8, palette:'000099,dddd00'}, 'Unfilled' );

Map.addLayer( FilledIMAGE, {min:0, max:700, opacity:0.8, palette:'000099,dddd00'}, 'Filled' );

// Some sort of tiling error seems to cause new values to vary only on a tile by tile basis in most areas

append to ee.Terrain.fillMinima

newImage = ee.Algorithms.FMask.fillMinima( oldImage, borderValue, neighborhood )

Fills local minima. Only works on INT types.

oldImage (Image) The specified image.

borderValue (Integer) The border value. Default: 0.

neighborhood (Integer) The size of the neighborhood to compute over. Default: 50.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TEXTURE** OPERATIONS

image.entropy creates a new image on which each pixel is set to a value calculated as the entropy of pixels lying within a specified kernel

on the corresponding band of a specified image. Entropy in this context is a negated sum of neighbor-by-neighbor contributions,

each computed as the product of that neighbor’s proportion of total neighborhood value

times the base-2 logarithm of that proportion. If all neighbors have the same value, the

resulting entropy is 0. As more neighbors have different values, entropy approaches 1.

newImage = oldImage.entropy( kernel )

The specified kernel

The specified image

The new image

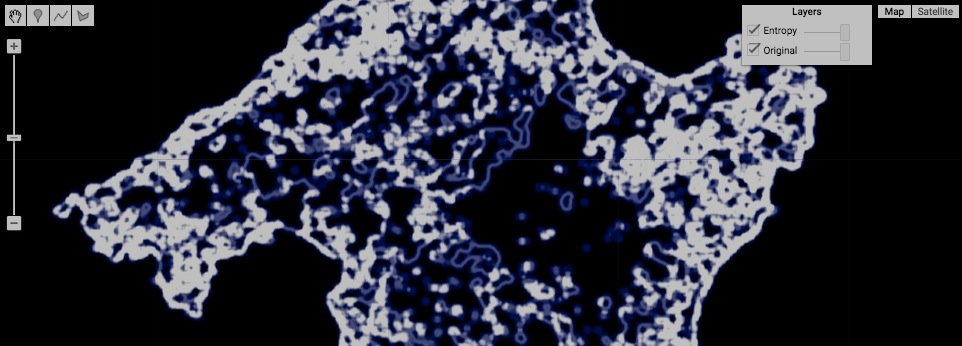
var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var NewIMAGE = OldIMAGE.entropy( ee.Kernel.circle( 5, 'pixels', true ) );

Map.setCenter( 2.9553, 39.6099, 10 );

Map.addLayer( OldIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Original' );

Map.addLayer( NewIMAGE, {min:0, max:1 , palette:'000000,000066,ffffff'}, 'Entropy' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **TEXTURE** OPERATIONS

image.glcmTexture creates a new image on which each pixel is set to 18 different values (on 18 different bands) reflecting the texture of a specified

image within a specified kernel-based neighborhood of that pixel by using a Gray Level Co-occurrence Matrix (GLCM) technique.

.

newImage = oldImage.glcmTexture( size, kernel, average? )

The specified kernel

The specified image

The new image

A Boolean set to true (only) if the directional bands

for each metric are to be averaged. Default: true.

The number of pixel widths across each square neighborhood to be considered

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var NewIMAGE = OldIMAGE.glcmTexture(5, ee.Kernel.circle( 5, 'pixels', true ) );

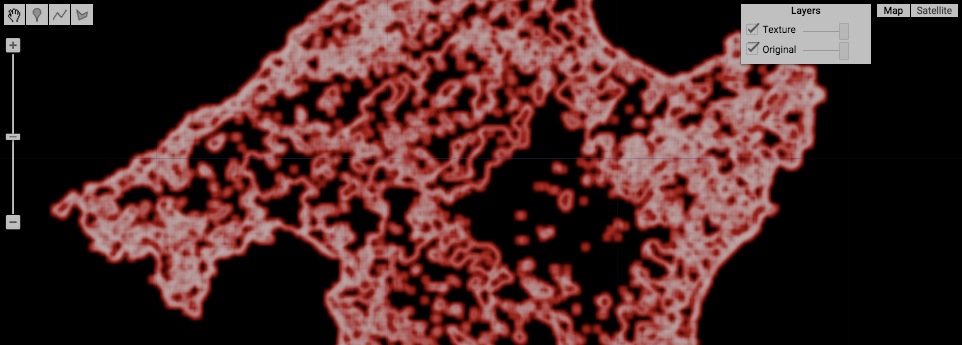
var TextureIMAGE = NewIMAGE.select( 0 );

print( NewIMAGE.getInfo(), TextureIMAGE.getInfo() );

Map.setCenter( 2.9553, 39.6099, 10 );

Map.addLayer( OldIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Original' );

Map.addLayer( TextureIMAGE, {min:0, max:1 , palette:'ffffff,dd0000,000000'}, 'Texture' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **EDGE** OPERATIONS

image.zeroCrossing creates a new image on which each pixel on each band is set to a value of 1 if it contains a zero-crossing point on a

a three-dimensional surface and 0 if it does not. A zero-crossing point is one at which the third-dimensional position

of a three-dimensional surface is 0. The third-dimensional position of each pixel on that surface is defined by its value on a specified image.

newImage = oldImage.zeroCrossing( )

The specified image

The new image

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' ).subtract(300);

var NewIMAGE = OldIMAGE.zeroCrossing( );

Map.setCenter( -76.959, 41.665, 7 );

Map.addLayer( OldIMAGE, { min:-500, max:500, opacity:0.6, palette:'000000,ffffff'}, '3D Surface' );

Map.addLayer( NewIMAGE, { opacity:0.3, palette:'ffffff,ff0000'}, 'Zero Crossing' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **EDGE** OPERATIONS

ee.Algorithms.CannyEdgeDetector creates a new image on which each pixel on each band is set to a value indicating the degree to which

that pixel is likely to be part of a boundary between two adjacent but different regions as depicted by the

corresponding band

of a specified image.

newImage = ee.Algorithms.CannyEdgeDetector( oldImage, minimumValue, *sigma* )

The new image on which a value of 0 indicates

that no edge is apparent, while higher values

indicate higher likelihoods.

The specified image, on which only those

pixels that exceed a specified minimum are

regarded as boundary candidates.

The specified

minimum

value

The sigma value for a Gaussian filter to be

applied to the specified image before edge

detection. If set to 0, no filter is applied.

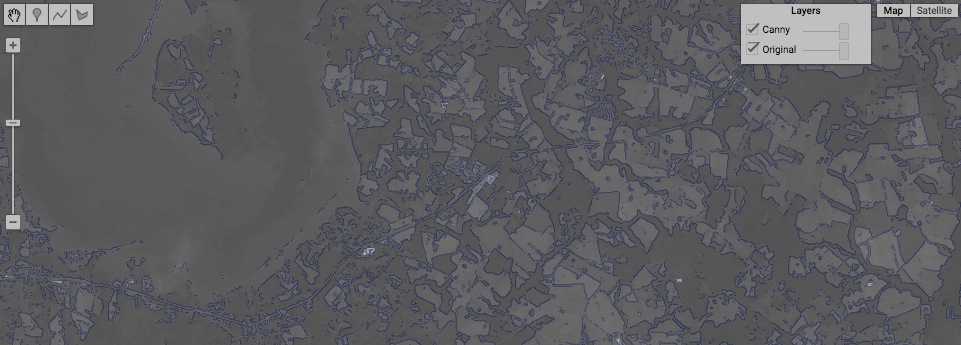
var OneBandIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ).select([2]);

var CannyIMAGE = ee.Algorithms.CannyEdgeDetector( OneBandIMAGE, 2000, 0 );

Map.setCenter( -76.1164, 39.0029, 12 );

Map.addLayer( OneBandIMAGE, { min:0, max: 15000, gamma:0.7 }, 'Original' );

Map.addLayer( CannyIMAGE, { opacity:0.2, palette:'ffffff,0000ff' }, 'Canny' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **EDGE** OPERATIONS

ee.Algorithms.HoughTransform creates a new image on which each pixel on each band is set to a value indicating whether that pixel

is part of a linear condition as depicted by the corresponding band of a specified image.

newImage = ee.Algorithms.CannyEdgeDetector( oldImage*, gridSize, inputMin, lineMin, presmooth? )*

A Boolean set

to true (only) if

input is to be

smoothed before

line detection.

Default: true

The specified

minimum

input value.

Default: 64

The minimum

value required

for designation

as part of a

linear condition.

Default: 72

The new image on which each new band is identified by the name of its corresponding band with the suffix “\_lines.”

The specified image, on which only those

pixels that exceed a specified minimum are

regarded as linear condition candidates

The grid size, given as … Default: 256

var OneBandIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ).select([2]);

var CannyIMAGE = ee.Algorithms.CannyEdgeDetector( OneBandIMAGE, 2000, 0 );

var HoughIMAGE = ee.Algorithms.HoughTransform( CannyIMAGE, 256, 50, 150, true );

Map.setCenter( -76.1164, 39.0029, 12 );

Map.addLayer( OneBandIMAGE, { min:0, max: 15000, gamma:0.7 }, 'Original' );

Map.addLayer( CannyIMAGE, { opacity:0.2, palette:'ffffff,0000ff' }, 'Canny' );

Map.addLayer( HoughIMAGE, { opacity:0.5, palette:'ffffff,ff0000' }, 'Hough' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **DISTANCE** OPERATIONS

oldImage.distance creates a new image on which each pixel is set to a value indicating its distance (in meters) to the nearest pixel

within a specified kernel-based neighborhood that has a non-zero value in each band of a specified image.

newImage = oldImage.distance( kernel )

The specified kernel

The specified image

The new image

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var ZeroIMAGE = ee.Image(0);

var TargetIMAGE = ZeroIMAGE.where( OldIMAGE.eq(0), 1 );

var DistanceIMAGE = TargetIMAGE.distance( ee.Kernel.euclidean( 4000, 'meters' ) );

print( OldIMAGE );

Map.setCenter( 3.3179, 39.6461, 9 );

Map.addLayer( TargetIMAGE, null, 'Target Image' );

Map.addLayer( DistanceIMAGE, {min: 1, max: 5000, opacity:0.7, palette:'ffffff,003300'}, 'Distance Image' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **DISTANCE** OPERATIONS

featureCollection.distance creates a new image on which each pixel is set to a Double indicating the spherical distance in meters from that pixel’s center

to the nearest edge of any item in a specified feature collection that lies within a specified maximum distance of the pixel..

newImage = oldFeatureCollection.distance( *maximumDistance, maximumReprojectionError* )

The maximum distance to be calculated,

given in meters. More distant pixel are

masked. Default: 100000

The maximum reprojection error to be tolerated when reprojection of the

specified feature collection is required, given in meters. Default: 100000

The specified

feature collection

The new

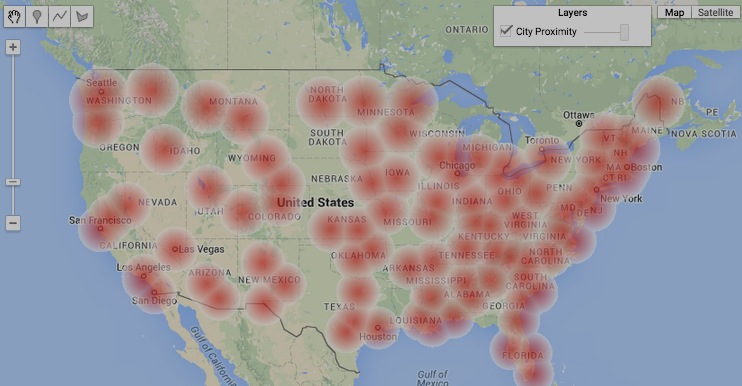
image

var TheFEATURES = ee.FeatureCollection( 'ft:1G3RZbWoTiCiYv\_LEwc7xKZq8aYoPZlL5\_KuVhyDM' ); // U.S. Cities

var TheIMAGE = TheFEATURES.distance( 200000 );

Map.setCenter( -95.9766, 39.3683, 4 );

Map.addLayer( TheIMAGE, {max:200000, palette:'ff0000,ffffff', opacity:0.6}, 'City Proximity' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **NEIGHBORHOODS** OPERATIONS

image.focal\_max , focal\_min , focal\_median , and focal\_mode

create a new image on which each pixel is set to a the maximum, minimum, median, or mode of all values

from a specified image that are associated with pixels lying in a specified kernel-based vicinity.

newImage = oldImage.focal\_max( *radius, kernelShape, units, iterations, kernel* )

or focal\_min

or focal\_median

A kernel radius,

given in pixel

widths.

Default: 1.5

The kernel radius

units, given either

'pixels' or 'meters.'

Default: 'pixels'

A kernel (which, if specified, will cause **radius** and **kernelShape** are ignored). Default: Null.

The new

image

or focal\_mode

The specified

image

A kernel shape, given as one of the strings 'circle', 'square',

'cross', 'plus', octagon' or 'diamond'. Default: 'circle'

The number of times the specified kernel is to be re-applied

(to the results of a previous application). Default: 1

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var MaxIMAGE = OldIMAGE.focal\_max( 5, 'circle', 'pixels' );

var MinIMAGE = OldIMAGE.focal\_min( 5, 'circle', 'pixels' );

var MedianIMAGE = OldIMAGE.focal\_median( 5, 'circle', 'pixels' );

var ModeIMAGE = OldIMAGE.focal\_mode( 5, 'circle', 'pixels' );

Map.setCenter( 2.9553, 39.6099, 10 );

Map.addLayer( OldIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Original ' );

Map.addLayer( MaxIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Maximum' );

Map.addLayer( MinIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Minimum' );

Map.addLayer( MedianIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Median' );

Map.addLayer( ModeIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Mode' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **NEIGHBORHOODS** OPERATIONS

image.convolve creates a new image on which each pixel is set to a value calculated by summing the kernel-weighted values

of pixels lying within a specified kernel on the corresponding band of a specified image.

newImage = oldImage.convolve( kernel )

The specified kernel

The specified image

The new image

var OldIMAGE = ee.ImageCollection('NOAA/DMSP-OLS/NIGHTTIME\_LIGHTS').select('stable\_lights').mean();

var TheKERNEL = ee.Kernel.fixed( 7,7,[[ 0, 0, 0, 0, 0, 0, 1 ],

[ 0, 0, 0, 0, 0, 0, 0 ],

[ 0, 0, 0, 0, 0, 0, 0 ],

[ 0, 0, 0, 0, 0, 0, 0 ],

[ 0, 0, 0, 0, 0, 0, 0 ],

[ 0, 0, 0, 0, 0, 0, 0 ],

[ 0, 0, 0, 0, 0, 0, 0 ]] );

var NewIMAGE = OldIMAGE.convolve( TheKERNEL );

print( NewIMAGE );

Map.setCenter( -80.222, 40.816, 6 );

Map.addLayer( OldIMAGE, {min:0, max:63, opacity:0.7, palette:'000044,ffff22'}, 'Night Lighting')

Map.addLayer( NewIMAGE, {min:0, max:63, opacity:0.7, palette:'000044,ffff22'}, 'Night Lighting')



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **NEIGHBORHOODS** OPERATIONS

image.reduceNeighborhood creates a new image on which each pixel is set to a value calculated by applying a specified reducer

to the values of neighbors within a specified kernel on the corresponding band of a specified image.

newImage = oldImage.reduceNeighborhood( reducer, kernel )

The specified image

The specified kernel

The specified reducer

The new image, whose band names will identify their corresponding input bands and, when necessary, their reduction types as well with a suffix (e.g. band1\_max).

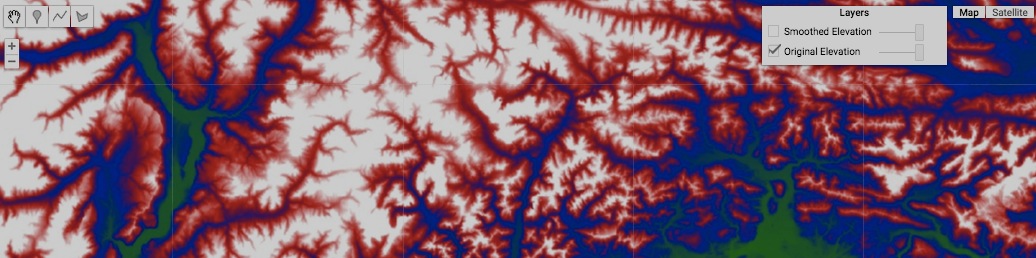
var OldIMAGE = ee.Image('CGIAR/SRTM90\_V4');

var NewIMAGE = OldIMAGE.reduceNeighborhood( ee.Reducer.mean( ), ee.Kernel.circle( 4 ) );

Map.setCenter( 15.469, 46.134, 6 );

Map.addLayer(OldIMAGE, {min: 0, max: 2000, palette:'009900,0000aa,cc0000,ffffff'}, 'Original Elevation');

Map.addLayer(NewIMAGE, {min: 0, max: 2000, palette:'009900,0000aa,cc0000,ffffff'}, 'Smoothed Elevation');





[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**TRANSFOMING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) WITH **ALIGNMENT** OPERATIONS

ee.Algorithms.CrossCorrelation creates a new image on which each pixel is set to four values (on three four) that describe the degree to which

two specified images (with the same number of bands and the same content) are spatially coincident at that pixel’s location. The four new bands respectively record, for each pixel in the first image, its 0) horizontal distance,

1) vertical distance, 2) direct (Euclidean) distance to whichever pixel in the second specified image has the highest correlation coefficient (indicating similarity), and 3) the correlation coefficient for this pixel, ranging from -1 to 1.

newImage = ee.Algorithms.CrossCorrelation( oldImage1*,* oldImage2, maximumGap, windowSize )

The second of the two specified images

The first of the two specified images

The width (and height) of the neighborhood to be considered in seeking coincident pixels, given in pixel widths

The new image

The maximum allowable horizontal or vertical distance between potentially coincident pixels, given in pixel widths.

EXAMPLE NOT YET FINALIZED

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var NewIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2002\_01\_01').select('Land\_Cover\_Type\_1');

var ComparisonIMAGE = ee.Algorithms.CrossCorrelation( OldIMAGE, NewIMAGE, 5, 5 );

print( OldIMAGE.getInfo(), NewIMAGE.getInfo(), ComparisonIMAGE.getInfo() );

Map.setCenter( -1.6013, 49.6009, 11 );

var ColorsForMODIS = ['aec3d4', // 00 = Water

'152106', // 01 = Evergreen Needleleaf Forest

'225129', // 02 = Evergreen Broadleaf Forest

'369b47', // 03 = Deciduous Needleleaf Forest

'30eb5b', // 04 = Deciduous Broadleaf Forest

'387242', // 05 = Mixed Deciduous Forest

'6a2325', // 06 = Closed Shrubland

'c3aa69', // 07 = Open Shrubland

'b76031', // 08 = Woody Savanna

'd9903d', // 09 = Savanna

'91af40', // 10 = Grassland

'111149', // 11 = Permanent Wetland

'cdb33b', // 12 = Cropland

'cc0013', // 13 = Urban

'33280d', // 14 = Crops & Natural Vegetation

'd7cdcc', // 15 = Permanent Snow & Ice

'f7e084', // 16 = Barren / Desert

'6f6f6f' // 17 = Unclassified

].join(',');

var DisplaySETTINGS = { min:0, max:17, opacity:0.7, palette:ColorsForMODIS };

Map.addLayer( OldIMAGE, DisplaySETTINGS, 'Earlier' );

Map.addLayer( NewIMAGE, DisplaySETTINGS, 'Later' );

Map.addLayer( ComparisonIMAGE.select(2), {min:0, max:10, opacity:0.7}, 'Change' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **FEATURE COLLECTIONS**

image.reduceToVectors

creates a new feature collection in which each feature represents an insular grouping of conterminous pixels that share a common value on the first band of a specified image.

newFeatureCollection = oldImage.reduceToVectors ( reducer*, extent, scale, shape, 8-way,* … )

A geometry that delineates the spatial extent to

be processed.

Default: the extent of the specified image's first band.

A Boolean set to True (only) if all eight neighboring pixels (including the four diagonals) are to be considered as potentially connected to a given pixel in forming the insular groups of like-valued pixels that are to be converted into features. Default: True.

The specified image, with a first band of integers

The new feature collection

A reducer that will be used to generate a value for each new feature by summarizing

the values of that feature’s pixels on all bands of the specified except for the first. This reducer must have as many inputs as there are bands (after the first) or a single input to be used for all bands. Default: Reducer.countEvery()

A number which, when divided by the nominal size of a meter in the specified **CRS**, indicates the scale of the specified image. This cannot be used in conjunction with **scale**. Default: Null.

One of three strings indicating how each insular group of pixels (sharing a common value on the first band of the specified image) is to be interpreted as a feature: **'polygon'** (calling for a polygon), **'bb'** (calling for a bonding rectangle), or **'centroid'** (calling for a centroid point). Default : "polygon".

(Additional settings are described on the following page.)

var SingleBandIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select(0); // Algeria

var TheEXTENT = ee.Geometry.Polygon([[-2.818,31.618],[-2.818,31.496],[-2.619,31.496],[-2.619,31.618]]);

var TheREDUCER = ee.Reducer.countEvery();

var TheBOUNDARIES = SingleBandIMAGE.reduceToVectors( TheREDUCER, TheEXTENT, null, 'polygon', false );

var TheBOUNDINGBOXES = SingleBandIMAGE.reduceToVectors( TheREDUCER, TheEXTENT, null, 'bb', false );

var TheCENTROIDS = SingleBandIMAGE.reduceToVectors( TheREDUCER, TheEXTENT, null, 'centroid', false );

var NewCENTROIDS = SingleBandIMAGE.reduceToVectors( TheREDUCER, TheEXTENT, null, 'centroid', true );

print( SingleBandIMAGE.getInfo() ); print( TheCENTROIDS.getInfo() ); print( NewCENTROIDS.getInfo() );

Map.setCenter(-2.7304, 31.5581, 11 );

Map.addLayer( SingleBandIMAGE, {min: 0, max: 17, palette:'99aaee,dd0000,00ff00,ffaaaa'}, 'Band 0' );

Map.addLayer( TheBOUNDARIES, {color:'dddddd'}, 'The Polygons' );

Map.addLayer( TheBOUNDINGBOXES, {color:'888888'}, 'The Bounding Boxes' );

Map.addLayer( TheCENTROIDS, {color:'000000'}, '4-way Centroids' );

Map.addLayer( NewCENTROIDS, {color:'ffffff'}, '8-way Centroids' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **FEATURE COLLECTIONS**

image.reduceToVectors … continued

. . . reduceToVectors ( … property, *CRS, placement, reScale, pixelMax, tilingFactor, pixelCRS* )

The name to be associated with a property that will be used to record, for each new feature, the first-band value from which it was created, given as a string. Default: “label”

A number that may avoid memory overload when increased from 1 to 2 or 3 or more. Default: 1

The maximum number of pixels to be reduced Default: 10,000,000

A list of six numbers that indicate how the specified image is to be situated with respect to the CRS. This cannot be used in conjunction with ***scale***. Default: Null

A Boolean set to True (only) if new features are to be created in the pixel projection rather than WGS84. Default: False

The cartographic reference system (CRS) to be used given as an EPSG code or WKT. Default: the CRS of the specified image’s first band

A Boolean set to True (only) if a larger scale is to be assumed automatically when the current scale would result in too many pixels per polygon. Default: False

var SingleBandIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select(0); // Algeria

var TheEXTENT = ee.Geometry.Polygon([[-2.818,31.618],[-2.818,31.496],[-2.619,31.496],[-2.619,31.618]]);

// reducer, extent, scale, shape, 8way

var TheCENTROIDS = SingleBandIMAGE.reduceToVectors( null, TheEXTENT, null, 'centroid', false,

'pixel value', null, null, true, 1000000, 1, true);

// property, CRS, placement, reScale, pixelMAx, tilingFactor, pixelCRS

print( TheCENTROIDS.getInfo() );

Map.setCenter(-2.7304, 31.5581, 11 );

Map.addLayer( SingleBandIMAGE, {min: 0, max: 17, palette:'99aaee,dd0000,00ff00,ffaaaa'}, 'Band 0' );

Map.addLayer( TheCENTROIDS, {color:'000000'}, 'The Centroids' );





[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **IMAGE COLLECTIONS**

ee.ImageCollection creates new image collection containing one or more specified images.

newImageCollection = ee.ImageCollection ( imageOrListOfImages )

The specified images, given as a list of images or as a single image

The new image collection

var FirstIMAGE = ee.Image( 'LC8\_L1T/LC80130312014276LGN00' ); // Mid-Atlantic Imagery

var SecondIMAGE = ee.Image( 'LC8\_L1T/LC80140322014219LGN00' );

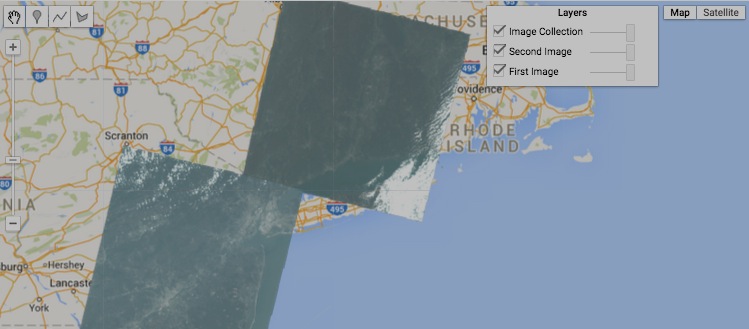
var TwoIMAGES = ee.ImageCollection( [FirstIMAGE, SecondIMAGE] );

Map.setCenter( -73.916, 41.063, 7 );

Map.addLayer( FirstIMAGE, { bands:['B3','B2','B1'], gain:0.003, opacity:0.3 },'First Image' );

Map.addLayer( SecondIMAGE, { bands:['B3','B2','B1'], gain:0.003, opacity:0.3 },'Second Image' );

Map.addLayer( TwoIMAGES, { bands:['B4','B2','B1'], gain:0.015, bias:[5,10,0]},'Image Collection' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **IMAGE COLLECTIONS**

ee.ImageCollection.fromImages creates new image collection containing one or more specified images.

newImageCollection = ee.ImageCollection.fromImages ( listOfImages )

The specified images, given as a list

The new image collection

var FirstIMAGE = ee.Image( 'LC8\_L1T/LC80130312014276LGN00' ); // Mid-Atlantic Imagery

var SecondIMAGE = ee.Image( 'LC8\_L1T/LC80140322014219LGN00' );

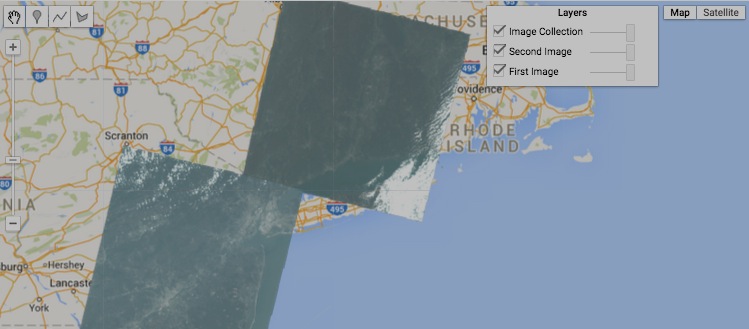
var TwoIMAGES = ee.ImageCollection.fromImages( [FirstIMAGE, SecondIMAGE] );

Map.setCenter( -73.916, 41.063, 7 );

Map.addLayer( FirstIMAGE, { bands:['B3','B2','B1'], gain:0.003, opacity:0.3 },'First Image' );

Map.addLayer( SecondIMAGE, { bands:['B3','B2','B1'], gain:0.003, opacity:0.3 },'Second Image' );

Map.addLayer( TwoIMAGES, { bands:['B4','B2','B1'], gain:0.015, bias:[5,10,0]},'Image Collection' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **GOOGLEMAP OVERLAYS**

image.getMap creates a new object containing a specified image, Map ID, and token from which to create a Google Maps overlay.

newObject = oldImage.getMap ( settings)

Instructions identifying data-display options in the same manner as **Map.addlayer**

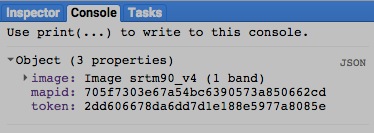
The specified image

The new object

var TheIMAGE = ee.Image( 'srtm90\_v4' );

var ThePACKAGE = TheIMAGE.getMap( { min:0, max:700, palette:'001100,009900'} );

print( ThePACKAGE );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **IMAGES OF BANDS FOR NEIGHBORS**

image.neighborhoodToBands creates a new image on which pixel is represented by multiple bands, each recording

the value of just one of that pixel’s neighbors on one of the bands of a specified image.

newImage = oldImage.neighborhoodToBands, kernel )

The specified kernel, whose weights will have no effect (other than to eliminate pixels of 0 weight).

The specified image

The new image, whose band names will identify input band and kernel positions (e.g. band1\_2\_3).

var OldIMAGE = ee.Image('CGIAR/SRTM90\_V4');

var NewIMAGE = OldIMAGE.neighborhoodToBands( ee.Kernel.plus(1) );

print( NewIMAGE );

Map.setCenter( -64.6731, 18.4001, 11 );

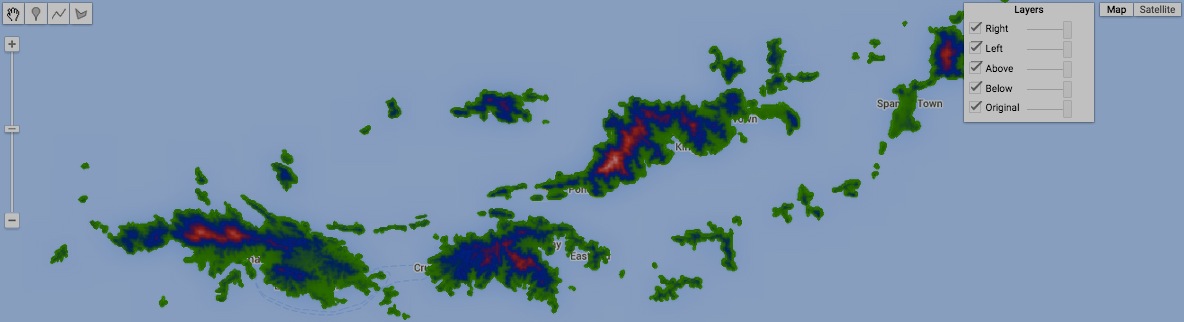
Map.addLayer(NewIMAGE.select('elevation\_0\_0'), {min: 0, max: 600, palette:'009900,0000aa,cc0000,ffffff'}, 'Original');

Map.addLayer(NewIMAGE.select('elevation\_0\_1'), {min: 0, max: 600, palette:'009900,0000aa,cc0000,ffffff'}, 'Below');

Map.addLayer(NewIMAGE.select('elevation\_1\_0'), {min: 0, max: 600, palette:'009900,0000aa,cc0000,ffffff'}, 'Above');

Map.addLayer(NewIMAGE.select('elevation\_-1\_0'), {min: 0, max: 600, palette:'009900,0000aa,cc0000,ffffff'}, 'Left');

Map.addLayer(NewIMAGE.select('elevation\_0\_-1'), {min: 0, max: 600, palette:'009900,0000aa,cc0000,ffffff'}, 'Right');



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **IMAGES OF ARRAYS FOR BANDS**

image.toArray creates a new image in which each pixel is represented by a one-dimensional array of its band-by-band values from a specified image.

newImage = oldImage.toArray ( axis )

The array axis, given as an integer starting at 0

The specified image

The new image

var BandedIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01');

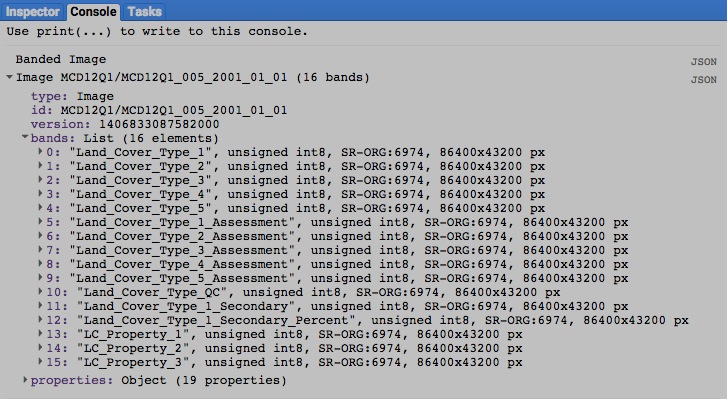
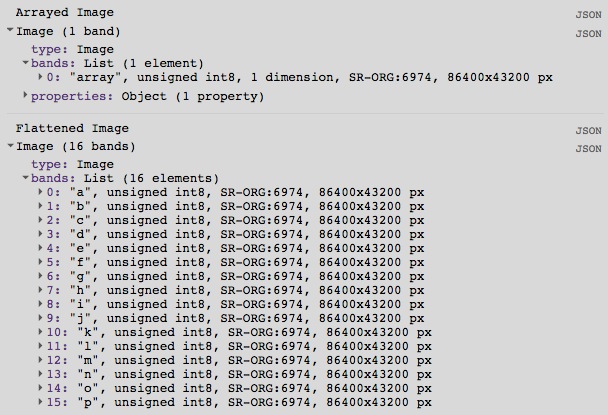
var ArrayedIMAGE = BandedIMAGE.toArray( 0 );

var FlattenedIMAGE = ArrayedIMAGE.arrayFlatten( [['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p']] );

print( 'Banded Image', BandedIMAGE );

print( 'Arrayed Image', ArrayedIMAGE );

print( 'Flattened Image', FlattenedIMAGE );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**REPRODUCING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) AS **IMAGES OF BANDS FOR ARRAYS**

image.arrayFlatten creates a new image by replicating a specified image of arrayed values such that every pixel’s Nth value is now on an Nth band.

newImage = oldImage.arrayFlatten ( axisLabels, *separatorCharacter* )

A character to appear between **axisLabels** when composing multiple-axis band names. Default: “\_”

The specified image

The new image

Labels to be associated with array axes when composing band namess, given as a list of strings accounting for all axes available

var BandedIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01');

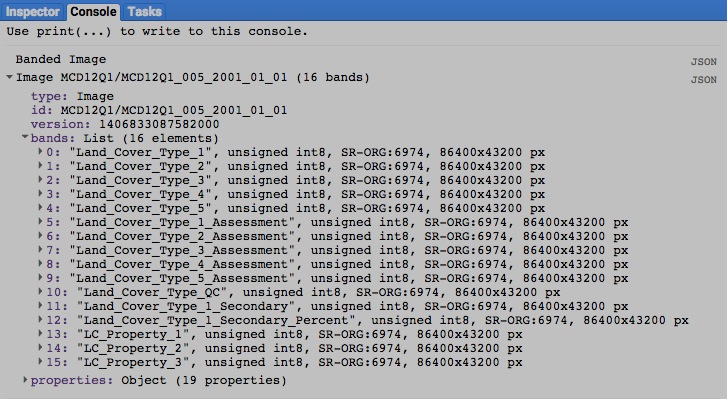
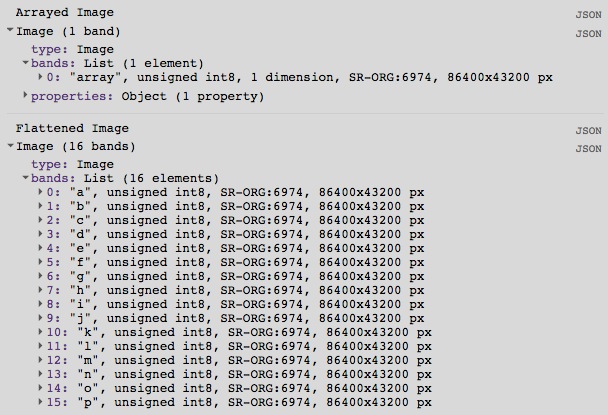
var ArrayedIMAGE = BandedIMAGE.toArray( 0 );

var FlattenedIMAGE = ArrayedIMAGE.arrayFlatten( [['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p']] );

print( 'Banded Image', BandedIMAGE );

print( 'Arrayed Image', ArrayedIMAGE );

print( 'Flattened Image', FlattenedIMAGE );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**SUMMARIZING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) BY **REGION**

image.reduceRegion creates a new dictionary that records the results generated by applying a specified reducer

to all of the pixels in each band of a specified image that lie within a specified region.

newDictionary = oldImage.reduceRegion( reducer*, region, scale, CRS, crsTransform, coarsen?* )

A Boolean set to True (only)

if the specified region should be automatically converted a

coarser scale as necessary

when it would otherwise

contain too many points to

process. Default: False.

CRS transform

values, given as

a 3x2 array of Doubles. The **crsTransform** and

**scale** options are mutually exclusive. Default: Null.

The new dictionary

The specified image

The specified reducer

The specified region, given as a geometry. Default: the specified image’s band 0 region

A nominal scale in meters from which to compute **crsTransform**. If **scale** is specified, then a **crsTransform** is computed by dividing the specified **scale** value by the nominal size of a meter in the specified projection. Default: Null.

The coordinate reference system (CRS) to be assumed for the specified image, given as an EPSG code or WKT. Default: "EPSG:4326"

var LandcoverPALETTE = ['aec3d4','152106','225129','369b47','30eb5b','387242','6a2325','c3aa69','b76031',

'd9903d','91af40','111149','cdb33b','cc0013','33280d','d7cdcc','f7e084','6f6f6f'].join(',');

var LandcoverIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

var WyomGEOMETRY = ee.Geometry.Polygon([[-111.05,41],[-111.05,45],[-104.10,45],[-104.10,41] ] );

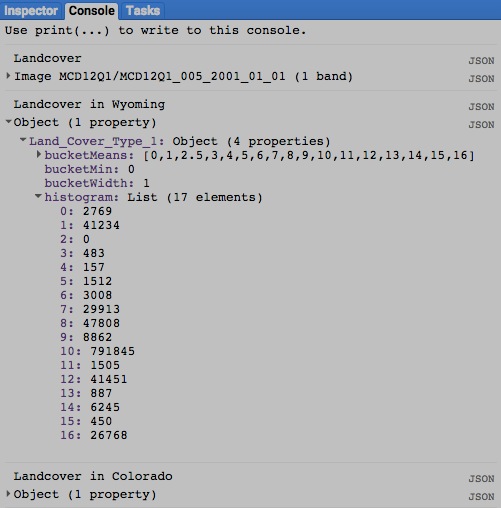
var ColoGEOMETRY = ee.Geometry.Polygon([[-109.05,41],[-109.05,37],[-102.05,37],[-102.05,41] ] );

var TheREDUCER = ee.Reducer.histogram();

var WyomDICTIONARY = LandcoverIMAGE.reduceRegion( TheREDUCER, WyomGEOMETRY, 500, null, null, true );

var ColoDICTIONARY = LandcoverIMAGE.reduceRegion( TheREDUCER, ColoGEOMETRY, 500, null, null, true );

print( 'Landcover', LandcoverIMAGE );

print( 'Landcover in Wyoming', WyomDICTIONARY );

print( 'Landcover in Colorado', ColoDICTIONARY );

Map.setCenter( -109.05, 41, 5 );

Map.addLayer( WyomGEOMETRY, {color:'3333ff'}, 'Wyoming' );

Map.addLayer( ColoGEOMETRY, {color:'ff3333'}, 'Colorado' );

Map.addLayer( LandcoverIMAGE, {palette:LandcoverPALETTE,

opacity:0.7, min:0, max:18 }, 'Landcover' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**COMPARING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Algorithms.IsEqual creates a new Boolean set to True (only) if the first of two specified images is identical to the second in both structure and content.

newBoolean = ee.Algorithms.IsEqual ( 1stImage, 2ndImage )

The first specified image

The second specified image

The new Boolean

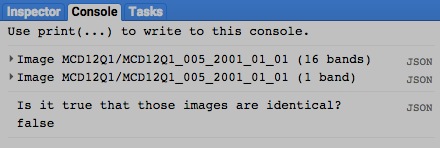
var The1stIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' );

var The2ndIMAGE = ee.Image( 'MCD12Q1/MCD12Q1\_005\_2001\_01\_01' ).select(['Land\_Cover\_Type\_1'] );

var TheBOOLEAN = ee.Algorithms.IsEqual( The1stIMAGE, The2ndIMAGE );

print( The1stIMAGE, The2ndIMAGE );

print( 'Is it true that those images are identical?', TheBOOLEAN );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**DOCUMENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.Algorithms.Describe and image.getInfo

each creates a JSON-compatible text object representing a specified image.

newObject = ee.Algorithms.Describe( oldImage )

and oldImage.getInfo( )

The specified image

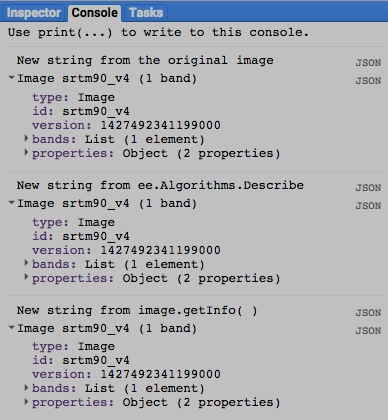
The new object

var OldIMAGE = ee.Image( 'srtm90\_v4' );

print( 'New string from the original image', OldIMAGE );

print( 'New string from ee.Algorithms.Describe', ee.Algorithms.Describe( OldIMAGE ) );

print( 'New string from image.getInfo( )', OldIMAGE.getInfo( ) );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**DOCUMENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

image.toString and .serialize each creates a new string presenting information on a specified image.

newString = oldImage.toString ( )

and oldImage.serialize( )

The specified image

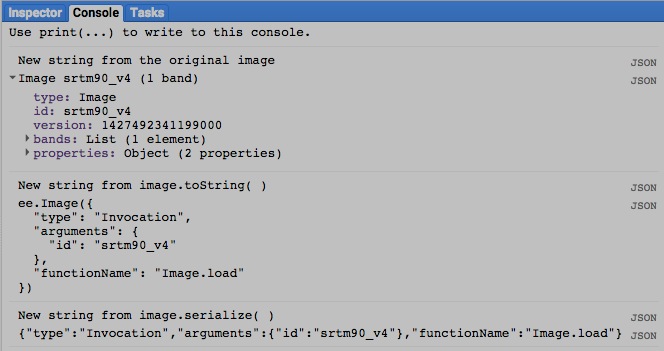
The new string

var OldIMAGE = ee.Image( 'srtm90\_v4' );

print( 'New string from the original image', OldIMAGE );

print( 'New string from image.toString( )', OldIMAGE.toString( ) );

print( 'New string from image.serialize( )', OldIMAGE.serialize( ) );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **PRINT**

print ( image ) and console.log ( image ) present a JSON-formatted text rendition of a specified image in the console.

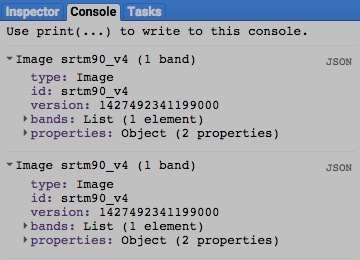
print( oldImage ) or console.log( oldImage )

The specified number

var TheIMAGE = ee.Image( 'srtm90\_v4' );

print( TheIMAGE );

console.log( TheIMAGE );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **PRINT**

alert ( image ) and confirm( image ) presents J SON-formatted text rendition text representation of a specified image

in a pop-up message box.

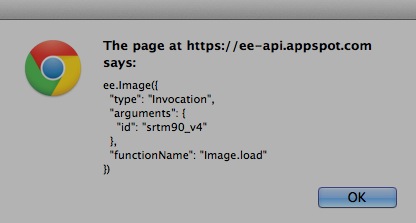
alert( oldImage ) or confirm( oldImage )

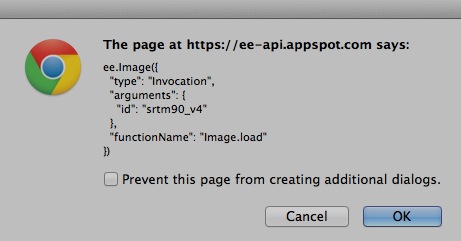
The specified image

var TheIMAGE = ee.Image( 'srtm90\_v4' );

alert( TheIMAGE );

confirm( TheIMAGE );





[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

Map.addLayer ( image ) adds a specified image to the map as a layer with specified graphic settings.

Map.addLayer ( image*,  settings, name, visibility* )

A Boolean set to TRUE (only) if the layer being displayed should initially be toggled on (visible) rather than off (invisible).Visibility can also be adjusted interactively by using the map’s layer listing.

The specified image

The name for the layer to be displayed, given as a string. This name will appear on the the map’s layer list listing.Default: "Layer N," where N is 1, 2, 3, and so on according to the order in which layers are added to the display.

Instructions indicating how to display the specified image, given as a curly bracketed and comma-separated sequence of the following **key: value** pairs.

**format: 'x'** gives **'x'**as one of the strings **'jpg'** or **'png'** to specify the digital format of the layer to be displayed.

**version: #** gives **#** as a number specifying the version of the image to be displayed. For the current version, give **#** as -1.

**bands: rgb** gives **rgb** as a list of the form **['r','g','b']**  (or as a concatenation of the form **'r, g, b'**) containing three strings that respectively

specify the names of the bands whose values are to be represented by the brightness of red (**r**), green (**g**), and blue (**b**) color

components.

**gain: #, bias: ##** gives **#** as a number specifying an amount by which each value is to be multiplied and/or **##** as a number specifying an amount to be

added to each value before being displayed. **#** and **##** may be given either as single numbers referring to all bands or as list s of numbers referring to (all) individual bands in consecutive order. These settingscannot be used in conjunction with **min** and **max** settings.

(Additional settings are described on the following page.)

var FirstIMAGE = ee.Image( 'LC8\_L1T/LC80150332014322LGN00' ); // Mid-Atlantic Imagery

var SecondIMAGE = ee.Image( 'LC8\_L1T/LC80140322014219LGN00' );

var ThirdIMAGE = ee.Image( 'LC8\_L1T/LC80130312014276LGN00' );

Map.setCenter( -74.8718, 40.4169, 7 );

Map.addLayer( FirstIMAGE, null, 'Unseen Scene', false );

Map.addLayer( SecondIMAGE, { format:'png', version:-1 }, 'Dark Scene' );

Map.addLayer( ThirdIMAGE, { bands:['B4','B3','B2'], gain:0.020, bias:[1,1,60] }, 'Bright Blue Scene' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

Map.addLayer ( image ) … continued

Map.addLayer ( image*,  settings, name, visibility* )

Additional instructions indicating how to display the specified image

**min: #, max: ##** gives **#** as a number specifying the lower end of the range of values that are to be represented by whatever range of colors are called for by **palette** settings and/or **##** to specify the upper end of that value range. Values below **#** and above **##** are respectively represented by lowest and highest extremes of the color range. **#** and **##** may be given either as single numbers referring to all bands or as list s of numbers referring

to (all) individual bands in consecutive order. These settingscannot be used in conjunction with **gain** or **bias** settings.

**gamma: #** gives **#** as a number ranging from 0 to 10 that calls for greater discrimination in the display of lower values (when # is higher) or higher values (when # is lower). **gamma** may be given either as a single number referring to all bands or as a list of consecutive numbers referring to (all) consecutive bands.

**opacity: #** gives **#** as a number ranging from 0 to 1 that calls for greater opacity in the display of values (when # is higher) or transparency (when # is

lower). Opacity can also be adjusted interactively by using the the layer list in the upper right corner of the map.

**palette: X** gives **X** as the range of colors to be used to display the range of values on a single-band. Each color within this range is given as a string of three two-character codes that indicate the amount of redness, greenness, and blueness (in that order) comprising the color. The characters in each two-character code are digits that represent quantities ranging from 0 to 15 as follows: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Thus, A (or a) represents ten, B (or b) represents eleven, and so on while A0 represents 160, while A represents 161, and FF represents 255. To the right are some examples; each six-character code appears in the color it represents as described [here](http://www.w3schools.com/tags/ref_colormixer.asp). Ranges of color are given as lists (or concatenations) of these strings such that the order in which colors are listed determines the order in which they appear in the color range, each occupying an equal portion of that range.

**000000 505050**

**FF0000 800000**

**FFAA00 DDCCAA**

**FFFF00 808000**

**00FF00 008000**

**00FFFF 008080**

**0000FF 000080**

**FF00FF 800080**

**FFFFFF 808080**

**000000 505050**

**FF0000 800000**

**FFAA00 DDCCAA**

**FFFF00 808000**

**00FF00 008000**

**00FFFF 008080**

**0000FF 000080**

**FF00FF 800080**

**FFFFFF 808080**

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

var TheIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' ); // NYC Elevation

Map.setCenter( -74.1083, 40.6723, 8 );

Map.addLayer( TheIMAGE, { min:0, max:1234, gamma: 1, opacity:0.8 }, 'Higher Elevations' );

Map.addLayer( TheIMAGE, { min:0, max:1234, gamma:10, opacity:1.0 }, 'Lower Elevations' );

Map.addLayer( TheIMAGE, { min:0, max:400, palette: ['ff0000','00ff00','0000ff'] }, 'Brighter Elevations' );

Map.addLayer( TheIMAGE, { min:0, max:400, palette: '660000, 006600, 000066' }, 'Darker Elevations' );

var mySpectrum = '220000, ff0000, 002200, 00ff00, 000022, 0000ff' ;

Map.addLayer( TheIMAGE, { min:0, max:400, palette:mySpectrum }, 'Tie-dyed Elevations' );

var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1\_005\_2001\_01\_01').select('Land\_Cover\_Type\_1');

print( OldIMAGE.getInfo() );

Map.setCenter( -17.93, 26.75, 3 );

var ColorsForMODIS = ['aec3d4', // 00 = Water

'152106', // 01 = Evergreen Needleleaf Forest

'225129', // 02 = Evergreen Broadleaf Forest

'369b47', // 03 = Deciduous Needleleaf Forest

'30eb5b', // 04 = Deciduous Broadleaf Forest

'387242', // 05 = Mixed Deciduous Forest

'6a2325', // 06 = Closed Shrubland

'c3aa69', // 07 = Open Shrubland

'b76031', // 08 = Woody Savanna

'd9903d', // 09 = Savanna

'91af40', // 10 = Grassland

'111149', // 11 = Permanent Wetland

'cdb33b', // 12 = Cropland

'cc0013', // 13 = Urban

'33280d', // 14 = Crops & Natural Vegetation

'd7cdcc', // 15 = Permanent Snow & Ice

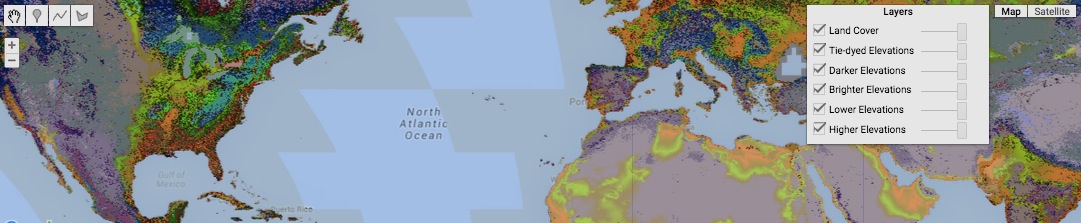
'f7e084', // 16 = Barren / Desert

'6f6f6f' // 17 = Unclassified

].join(',');

var DisplaySETTINGS = { min:0, max:17, opacity:0.7, palette:ColorsForMODIS };

Map.addLayer( OldIMAGE, DisplaySETTINGS, 'Land Cover' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

image.Visualize creates a new image by reproducing a specified image along with specified instructions on how it is to be displayed.

newImage = oldImage.visualize ( *bands,  gain, bias, min, max, gamma, opacity, palette* )

These three arguments are described on the following page.

The specified image

The new image

A number specifying the upper end of the range of values that are to

be represented by whatever range of colors are called for by **palette** settings. Higher values are represented by the color at the high end of the color range and. This may be given either as a single number referring to all bands or as a list of numbers referring to (all) individual bands in consecutive order. This settingcannot be used in conjunction with **gain** or **bias** settings.

A list of three strings that specify the names of the bands whose values are to control the brightness of red, green, and blue color components in that order

A number specifying an amount by which each value is to be multiplied. This may be given either as a single number (referring to all bands) or as a list of numbers (referring to every band in consecutive order), but it cannot be used in conjunction with **min** and **max** settings.

A number specifying the lower end of the range of values that are to be represented by whatever range of colors are called for by **palette** settings. Lower values are represented by the color at the low end of the color range. This may be given either as a single number referring to all bands or as a list of numbers referring to (all) individual bands in consecutive order. This settingcannot be used in conjunction with **gain** or **bias** settings.

A number specifying an amount to be added to each value. This may be given either as a single number (referring to all bands) or as a list of numbers (referring to every band in consecutive order), but it cannot be used in conjunction with **min** and **max** settings.

var OldIMAGE = ee.Image('L5\_L1T/LT50440342011181PAC01'); // Silicon Valley

Map.setCenter( -122.0794, 37.4716, 12 );

// Arguments for image.visualize: bands, gain, bias, min, max,

var RebandedIMAGE = OldIMAGE.visualize( ['30','20','10'] );

var GainBiasedIMAGE = OldIMAGE.visualize( ['30','20','10'], [1,2,1], [0,0,100] );

var MinMaxedIMAGE = OldIMAGE.visualize( ['30','20','10'], null, null, 0, 100 );

Map.addLayer( OldIMAGE, null, 'Original Image' );

Map.addLayer( RebandedIMAGE, null, 'Rebanded Image' );

Map.addLayer( GainBiasedIMAGE, null, 'Gain-Biased Image' );

Map.addLayer( MinMaxedIMAGE, null, 'Min-Maxed Image' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

image.Visualize … continued

newImage = oldImage.visualize ( *bands,  gain, bias, min, max, gamma. opacity, palette* )

A number ranging from 0 to 10 that calls for greater discrimination in the display of lower values (when it is higher) or higher values (when it is lower). This may be given either as a single number referring to all bands or as a list of numbers referring to individual bands in consecutive order (in which case all bands must be accounted for).

**000000 505050**

**FF0000 800000**

**FFAA00 DDCCAA**

**FFFF00 808000**

**00FF00 008000**

**00FFFF 008080**

**0000FF 000080**

**FF00FF 800080**

**FFFFFF 808080**

A number ranging from 0 to 1 that indicates how opaque (non-transparent) the layer should appear.

**000000 505050**

**FF0000 800000**

**FFAA00 DDCCAA**

**FFFF00 808000**

**00FF00 008000**

**00FFFF 008080**

**0000FF 000080**

**FF00FF 800080**

**FFFFFF 808080**

A range of colors to be used to display the range of values on a single-band. Each color within this range is given as a string of three two-character codes that indicate the amount of redness, greenness, and blueness (in that order) comprising the color. The characters in each two-character code are digits that represent quantities ranging from 0 to 15 as follows: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Thus, A (or a) represents ten, B (or b) represents eleven, and so on while A0 represents 160, while A represents 161, and FF represents 255. To the right are some examples; each six-character code appears in the color it represents as described [here](http://www.w3schools.com/tags/ref_colormixer.asp). Ranges of color are given as lists (or concatenations) of these strings such that the order in which colors are listed determines the order in which they appear in the color range, each occupying an equal portion of that range.

var OldIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' ); // NYC Elevation

Map.setCenter( -74.1083, 40.6723, 8 );

// Arguments for image.visualize: bands, gain, bias, min, max, gamma, opacity, pallette

var TranslucentIMAGE = OldIMAGE.visualize( null, null, null, 0, 500, null, 0.6 );

var OpaqueIMAGE = OldIMAGE.visualize( null, null, null, 0, 500, null );

var LowGammaIMAGE = OldIMAGE.visualize( null, null, null, 0, 500, 2 );

var HighGammaIMAGE = OldIMAGE.visualize( null, null, null, 0, 500, 8 );

var BrightIMAGE = OldIMAGE.visualize( null, null, null, 100, 500, null, null, ['FF0000','00FF00','0000FF'] );

var DarkIMAGE = OldIMAGE.visualize( null, null, null, 0, 100, null, null, ['000044','00FF00','440000'] );

Map.addLayer( TranslucentIMAGE, null, 'Translucent' );

Map.addLayer( OpaqueIMAGE, null, 'Opaque' );

Map.addLayer( LowGammaIMAGE, null, 'Low Gamma' );

Map.addLayer( HighGammaIMAGE, null, 'High Gamma' );

Map.addLayer( BrightIMAGE, null, 'Bright Uplands' );

Map.addLayer( DarkIMAGE, null, 'Dark Lowlands' );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **MAPS**

image.sldStyle creates a new image by reproducing a specified image along with specified OCG-SLD instructions on how it is to be displayed.

newImage = oldImage.sldStyle ( *settings* )

A set of image-display settings, given as a string in OCG-SLD (Open

Geospatial Consortium Styled Layer Descriptor) format as described [here](http://www.opengeospatial.org/standards/sld).

The specified image

The new image

var OldIMAGE = ee.Image('srtm90\_v4'); // NYC Elevation

Map.setCenter(-97.6465, 40.5806, 4 );

var NewIMAGE = OldIMAGE.sldStyle( '<RasterSymbolizer> \

<ColorMap type="ramp" extended="false"> \

<ColorMapEntry color="#000099" quantity="1" label=" 1" /> \

<ColorMapEntry color="#00ffff" quantity="10" label=" 10" /> \

<ColorMapEntry color="#009900" quantity="100" label=" 100" /> \

<ColorMapEntry color="#ffff00" quantity="500" label=" 500" /> \

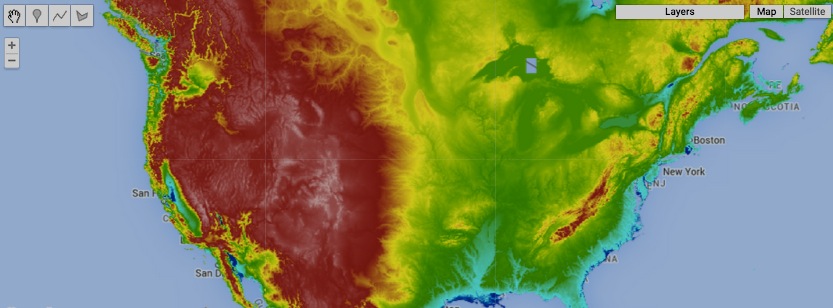
<ColorMapEntry color="#990000" quantity="1000" label="1000" /> \

<ColorMapEntry color="#ffffff" quantity="5000" label="5000" /> \

</ColorMap> \

</RasterSymbolizer>'

);

Map.addLayer( NewIMAGE );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **CHARTS**

Chart.image.histogram presents a chart

- whose horizontal (X) axis identifies the (grouped) values of a specified image;

- whose vertical (Y) axis identifies a number of pixels; and

- whose charted data indicate the number of pixels associated with each (grouping of) image band value.

newChart = Chart.image.histogram( oldImage, region, scale*, groupNumber, groupSize, maxSample* )

A minimum size of property group over which to count

(numerical) feature frequencies.

The region reducer’s

pixel resolution,

given as a number

of meters

The new chart

The specified image

A minimum number of (numerical) features to use before constructing an initial histogram.

The specified region, given as a feature or geometry. Default: the entire image

A maximum number of property groups over which to count (numerical) feature frequencies.

var TheIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var TheFEATURES = ee.FeatureCollection('ft:1fRY18cjsHzDgGiJiS2nnpUU3v9JPDc2HNaR7Xk8'); // U.S. States

var TheFEATURE = TheFEATURES.filterMetadata('Name','equals','Wyoming');

var TheCHART = Chart.image.histogram( TheIMAGE, TheFEATURE, 1000);

var TheCHART = TheCHART.setOptions( { title:'Statewide Elevation Distribution',

hAxis:{ title: 'Elevation in Meters' },

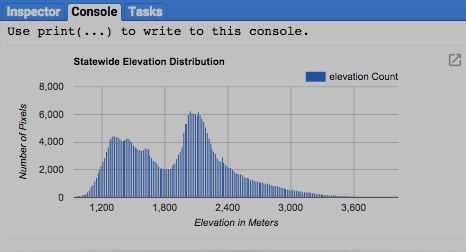
vAxis:{ title: 'Number of Pixels'} } );

print( TheCHART );

Map.centerObject( TheFEATURE,6 );

Map.addLayer( TheFEATURE, { color:'444444', opacity:0.01 }, 'The State' );

Map.addLayer( TheIMAGE, { min:0, max:3000, opacity:0.8, palette: ['000099','00aa00','ff0000'] }, 'Elevation' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **CHARTS**

Chart.image.byRegion presents a chart

- whose X axis identifies each of a specified set of regions;

- whose Y axis identifies the values generated by applying a specified reducer to a specified image; and

- whose charted data indicate the reduced value of each band within each of the specified set of regions.

newChart = Chart.image.byRegion( oldImage*, regions, reducer, scale, xRegionProperty* )

The reducer’s pixel resolution,

given as a number of meters

The region property from

which to draw X-axis labels.

Default: “system:index”

The specified image

The new chart

The specified reducer, which must compute a

single value per band. Default: ee.Reducer( )

The specified region(s), given as a feature, a list of features, a feature

collection, a geometry, or a list of geometries. Default: entire image.

var TheIMAGE = ee.Image( 'CGIAR/SRTM90\_V4' );

var TheFEATURES = ee.FeatureCollection('ft:1fRY18cjsHzDgGiJiS2nnpUU3v9JPDc2HNaR7Xk8'); // U.S. States

var TheFEATURES = TheFEATURES.filterMetadata('Name','contains','New');

var TheCHART = Chart.image.byRegion( TheIMAGE, TheFEATURES, ee.Reducer.mean(), 1000, 'Name');

var TheCHART = TheCHART.setOptions( { title:'Mean Elevation by State',

hAxis:{ title: 'Selected States' },

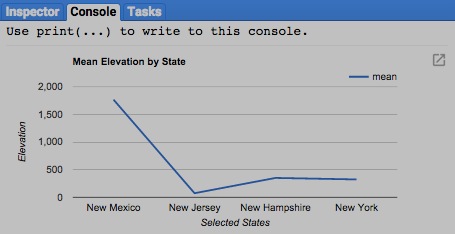
vAxis:{ title: 'Elevation'} } );

print( TheCHART );

Map.centerObject( TheFEATURES,4 );

Map.addLayer( TheFEATURES, { color:'000000', opacity:0.1 }, 'The States' );

Map.addLayer( TheIMAGE, { min:0, max:3000, opacity:0.5, palette: ['000099','00aa00','ff0000'] }, 'Elevation' );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**PRESENTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx) IN **CHARTS**

Chart.image.regions presents a chart

- whose X axis identifies each of the bands in a specified image;

- whose Y axis identifies the values generated by applying a specified reducer to that image; and

- whose charted data indicate the reduced value of each band within each of a specified series of regions.

newChart = Chart.image.regions( oldImage*, regions, reducer, scale, seriesProperty, xLabels* )

The specified image

The specified

reducer, which

must compute

a single value

per band.

Default:

ee.Reducer( )

The reducer’s pixel

resolution, given as a

number of meters

The X-axis labels, given as a list of as many strings as there are bands. If labels are numerical,

the X-axis will be continuous.

Default: band names

The new chart

The specified region(s), given as a feature, a list of features, a feature

collection, a geometry, or a list of geometries. Default: the entire image.

The region property from which to draw labels.

Default: “system:index”

var SatelliteIMAGES = ee.ImageCollection('LANDSAT/LC8\_L1T\_8DAY\_TOA').filterDate('2014-01-01','2014-12-31');

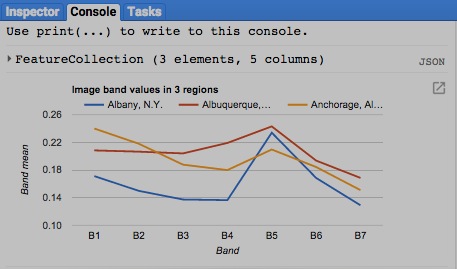
var SatelliteIMAGE = SatelliteIMAGES.median().select( ['B1','B2','B3','B4','B5','B6','B7'] );

var CityFEATURES = ee.FeatureCollection( 'ft:1G3RZbWoTiCiYv\_LEwc7xKZq8aYoPZlL5\_KuVhyDM' ).limit(3); // U.S. Cities

print( CityFEATURES );

var TheCHART = Chart.image.regions( SatelliteIMAGE, CityFEATURES, ee.Reducer.mean(), 30, 'city\_name');

console.log( TheCHART );



[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EXPORTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

image.getDownloadURL creates the link to a web address from which specified image bands can be downloaded in GeoTIFF format.

To export large files (of more than ~1 Gb),

use **Export.image** rather than **getDownloadURL**.

newLink = oldImage.getDownloadURL ( settings )

The new link

The specified image

Instructions indicating how to prepare the specified image, given as a curly bracketed and comma-separated sequence of the following **key: value** pairs.

**name: 'x'** gives **'x'** as a string to be used as the base name for all exported files.

**dimensions: 'x'** gives **'x'** as a string specifying the default width and height (in meters) of the region to be exported when none is otherwise specified.

**crs:'x'** gives **'x'** as a string naming the cartographic reference system (CRS) to be used for the exported image. For example,

‘EPSG:4326’ for WGS 84 or ‘SR-ORG:6974’ for MODIS Sinusoidal. For all possibilities, look [here](http://spatialreference.org).

**crs\_transform: 'x'** gives **'x'** as a string of six numbers (as described in the handout *22.00 PROJECTIONS*) that indicate

how each band is to be situated with respect to its CRS if not otherwise specified.

**scale: #** gives **#** as a number specifying the default pixel size (in meters) for each exported band when none is otherwise specified.

This setting is gnored if **crs** and crs\_**transform** are specified.

**region: X** gives **X** as a list of point coordinates bounding the region to be exported. This setting is Ignored if **crs** and crs\_**transform** are specified.

**bands: [ X ]** gives **[ X ]** as a list of dictionaries with the following keys, only the first of which (**id:'x'**) is required.

**id:'x'** gives **'x'** as a string naming the band to which the following settings apply.

**crs:'x'** gives **'x'** as a string naming the cartographic reference system (CRS) to be used for this band.

**crs\_transform:** **x'** gives **'x'** as a string of six numbers indicating how this band is to be situated with respect to its CRS.

**dimensions: [ X ]** gives **[ X ]** as a list of two integers indicating the width and height (in meters) of the region to be exported.

**scale:** is an optional number specifying the band’s scale in meters; ignored if crs and crs\_transform is specified.

var TheGEOMETRY = ee.Geometry.Polygon( [ [-61.91,17.18], [-61.91,16.99], [-61.65,16.99], [-61.65,17.18] ] ); // Antigua

var TheIMAGE = ee.Image('srtm90\_v4').unitScale(0,300).multiply(255).toUint8(); // Normalize Elevation to 0-255 range Map.centerObject( TheGEOMETRY, 11 );

Map.addLayer( TheGEOMETRY, { color:'ffffff' }, 'The Geometry' );

Map.addLayer( TheIMAGE, { palette:'000000,ffffff', opacity:0.4 }, 'The Image' );

var TheLINK = TheIMAGE.getDownloadURL( { name:'Antigua', crs:'EPSG:4326', scale:30,

region:'[ [-61.91,17.18], [-61.91,16.99], [-61.65,16.99], [-61.65,17.18] ]' } );

print( TheLINK );

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EXPORTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.data.getDownloadId creates a dictionary containing a specified image’s asset ID and token for use with **ee.data.makeDownloadUrl** .

**image.getDownloadUrl**  is a much simpler alternative to **ee.data.getDownloadUrl** and **ee.data.makeDownloadUrl** .

To export large files (of more than ~1 Gb), use **Export.image**

newDictionary = ee.data.getDownloadId ( settings )

The new dictionary

Instructions identifying the image and specifying how to prepare it, given as a curly bracketed and comma-separated sequence of the following **key: value** pairs.

**id: 'x'** gives **'x'** as a string indicating the asset ID of the specified image.

**name: 'x'** gives **'x'** as a string to be used as the base name for all exported files.

**dimensions: 'x'** gives **'x'** as a string specifying the default width and height (in meters) of the region to be exported when none is otherwise specified.

**crs:'x'** gives **'x'** as a string naming the cartographic reference system (CRS) to be used for the exported image. For example,

‘EPSG:4326’ for WGS 84 or ‘SR-ORG:6974’ for MODIS Sinusoidal. For all possibilities, look [here](http://spatialreference.org).

**crs\_transform: 'x'** gives **'x'** as a string of six numbers (as described in the handout *22.00 PROJECTIONS*) that indicate

how each band is to be situated with respect to its CRS if not otherwise specified.

**scale: #** gives **#** as a number specifying the default pixel size (in meters) for each exported band when none is otherwise specified.

This setting is ignored if **crs** and crs\_**transform** are specified.

**region: X** gives **X** as a list of point coordinates bounding the region to be exported. This setting is Ignored if **crs** and crs\_**transform** are specified.

**bands: [ X ]** gives **[ X ]** as a list of dictionaries, each with the following keys for a given band (of which only the first is required)

**id:'x'** gives **'x'** as a string naming the band to which the following settings apply.

**crs:'x'** gives **'x'** as a string naming the cartographic reference system (CRS) to be used for this band.

**crs\_transform:** **x'** gives **'x'** as a string of six numbers indicating how this band is to be situated with respect to its CRS.

**dimensions: [ X ]** gives **[ X ]** as a list of two integers indicating the width and height (in meters) of the region to be exported.

**scale:** an optional number specifying the band’s scale in meters; ignored if crs and crs\_transform is specified.

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EXPORTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

ee.data.getDownloadURL creates the link to a web address from which specified image bands can be downloaded in GeoTIFF format.

This operation is typically used

with **ee.data.getDownloadId**.

newLink = ee.data.makeDownloadURL ( downloadID )

A dictionary containing a specified image’s asset ID and token, as generated by **ee.data.getDownloadId**

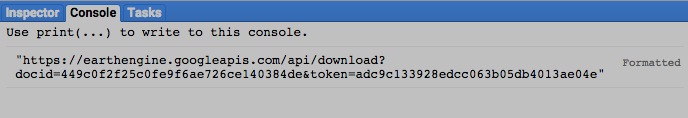
The new link

var TheDownloadID = ee.data.getDownloadId( { id:'srtm90\_v4', name:'Antigua', crs:'EPSG:4326', scale:30,

region:'[ [-61.91,17.18], [-61.91,16.99], [-61.65,16.99], [-61.65,17.18] ]' } );

var TheLINK = ee.data.makeDownloadUrl( TheDownloadID );

print( TheLINK );

****

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EXPORTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

Export.image initiates a task with a specified name that will export a specified rectangular portion of a specified image in GeoTIFF format.

After executing this operation, the Report Panel’s **Tasks** tab will present a **RUN** button that can be used to open a dialog box in which you can (re)specify

* the task name,
* the scale,
* the Google Drive folder name, and
* the selection of a Google Drive folder or Google Maps Engine project

as the export destination.

Once started, a spinning arrow will indicate that the task is running (even after leaving the EE API). For task information, hover over this arrow and click on the **?** button that will appear. To cancel the task, hover over this arrow and click on the **X** button.

Export.image ( image*,* taskName*, settings* )

The specified image

The specified task name, given as a string with no blank spaces

Instructions indicating how to prepare the specified image, given as a curly bracketed and comma-separated sequence of the following **key: value** pairs.

**scale: #** gives **#** as a number specifying the pixel size of the exported image in meters This requires that **region** also be specified.

**region: X** gives **X** as a polygonal geometry, a GeoJSON linear ring, or list of point coordinates bounding the region to be exported.

This requires that **scale** also be specified. Default: the extent of the current map display screen

**maxPixels: #** gives **#** as the maximum number of exported pixels to be allowed. Default: 100,000,000

**driveFolder: 'x'** gives **'x'** as a string naming the (preexisting) Google Drive folder to which the image is to be exported.

Default: the Drive root.

**driveFileNamePrefix: 'x'** gives **'x'** as a string to be used a s a prefix for all exported files. Default: the specified task name

**crs:'x'** gives **'x'** as a string naming the cartographic reference system (CRS) to be used for the exported image. For example,

‘EPSG:4326’ for WGS 84 or ‘SR-ORG:6974’ for MODIS Sinusoidal. For all possibilities, look [here](http://spatialreference.org).

**crs\_transform: 'x'** gives **'x'** as a string of six numbers (as described in the handout *22.00 PROJECTIONS*) that indicate how the exported

image is to be situated with respect to its CRS. This requires that **dimensions** also be specified.

**dimensions: 'x'** gives **'x'** as a string specifying the width and height (in meters) of the exported image.

This requires that **crs\_transform** also be specified.

(Additional settings relating to the use of Google Maps Engine (GME) appear on the following page, but GME is to be discontinued).

[GOOGLE EARTH ENGINE](EE01%20Earth%20Engine%20(EE).docx) [APPLICATION PROGRAMMING INTERFACE](EE05%20%20%20The%20EE%20API.docx) [CAPABILITIES](EE07%20%20%20%20%20%20API%20Capabilities.docx)

**EXPORTING** [IMAGE](#_top) [VARIABLES](EE13%20%20%20%20%20%20%20%20%20Variables.docx)

Export.image … continued

Export.image ( image*,* taskName*, settings* )

**gmeProjectId: 'x'**  gives **'x'** as a string naming a Google Maps Engine project to which the image is to be exported.

This requires that **gmeAttributionName** and **gmeMosaic** also be specified.

**gmeAttributionName: 'x'** gives **'x'** as a string indicating to whom the Google Maps Engine export is to be attributed.

**gmeMosaic: 'x'**  gives **'x'** as a string identifying the Google Maps Engine mosaic to which the image is to be exported.

**gmeTerrain: X**  gives **X** as a Boolean set to true (only) if a Google Maps Engine export is to create a TERRAIN asset.

If so, the exported image must contain a single band of floating-point value

var VertexLIST = [ [-69.46,45.43],[-69.46,45.35],[-69.29,45.35],[-69.29,45.43] ]; // Maine

var TheIMAGE = ee.Image('srtm90\_v4').unitScale(100,850).multiply(255).toUint8(); // Normalize Elevation to 0-255 range

var TheGEOMETRY = ee.Geometry.Polygon( VertexLIST );

Map.centerObject( TheGEOMETRY, 11 );

Map.addLayer( TheGEOMETRY, { color:'ffffff' }, 'The Geometry' );

Map.addLayer( TheIMAGE, { palette:'000044,00bb00,cc0000,ffffff', opacity:0.4 }, 'The Image' );

Export.image( TheIMAGE, 'TopoExport', {scale:100, region:VertexLIST, maxPixels:111111111,

driveFolder:'EEexports', driveFileNamePrefix:'EE'} );

