第 11 节 GEE 的参数类型（Reducer，Kernel，Algorithm）

Reducer 的中文含义是“缩减器，减压器”。与筛选 Filter 相比，Reducer 虽然也有“减少”的意思，但其更多的含义在于“通过分析后获得统计信息”上。比如有 100 个苹果，Filter 处理后只剩下 80 个，而 Reducer 处理后得则可以到“平均重量”。总之，Filter 着重于数量上的减少，而 Reducer 强调数学抽象的汇总。

Kernel 的中文含义是“核，果仁”。在 GEE 中，Kernel 指的是由若干像素构成的矩形平面空间，通常以矩阵形式表示。Algorithm 的中文含义是“算法”。在 GEE 中，Algorithm 指的是一种“样例操作”，其通常与.map 命令一起达到对数据集中的每个数据都进行“样例操作”的目的。

11.1 Reducer

下边介绍数量统计和返回首值的 Reducer，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Reducer_Count = ee.Reducer.count()
var Reducer_CountEvery = ee.Reducer.countEvery()
var Reducer_First = ee.Reducer.first()
var Provinces_Number_1 = China_Provinces.reduceColumns(Reducer_Count,['NAME'])
var Provinces_Number_2 = China_Provinces.reduceColumns(Reducer_CountEvery,[ ])
var Provinces_First = China_Provinces.reduceColumns(Reducer_First,['NAME'])

print('Reducer_Count',Provinces_Number_1)
print('Reducer_CountEvery',Provinces_Number_2)
print('Reducer_First',Provinces_First)
```

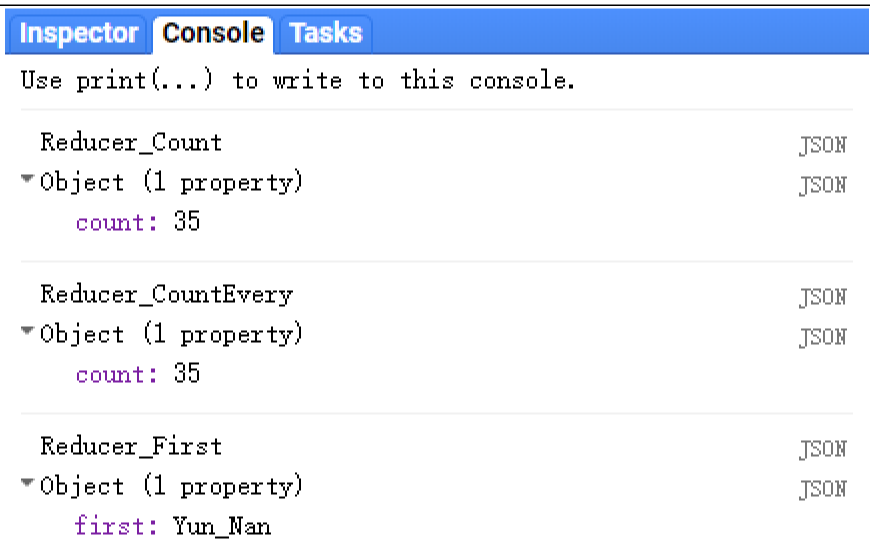


图 11.1 数量统计和返回首值

这里需要强调 ee.Reducer()命令创建的是一个“名词”，其发挥作用只能在于其他“动词”配合的前提下才能完成，本例中的动词是.reducerColumns()，可以理解为“列统计”。此外要注意.count 和 .countEvery 的区别。.count 是计算指定列的数据，如果数据缺失则不进行数量统计(例如某数据不存在“NAME”属性)，而.countEvery 则是统计所有数据。

下边介绍频率直方图的 Reducer，本例中加载是数据是中国城市边界，目的是统计中国各省个有多少个市级行政单位，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
var FrequencyHiso_Reducer = ee.Reducer.frequencyHistogram()
var City_Frequency = China_Cities.reduceColumns(FrequencyHiso_Reducer,['NAME_1'])
var Fig_Histo = ui.Chart.feature.histogram(China_Cities,'NAME_1')

print(China_Cities.limit(10))
print(City_Frequency,Fig_Histo)
```

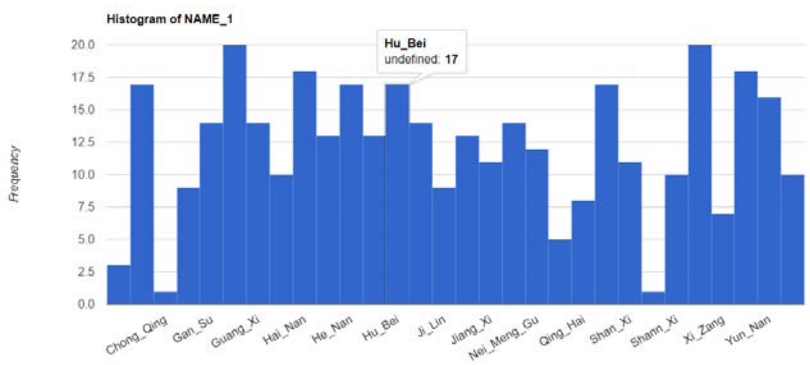


图 11.2 频率直方图

下边介绍 0 值判断的 Reducer，代码及执行效果如下：

```
var No_Zero_Reducer = ee.Reducer.allNonZero( );
var Any_Non_Zero_Reducer = ee.Reducer.anyNonZero( );
var List_Test_1      = ee.List( [1,2,3,4,5,6,7,8,9] );
var List_Test_2      = ee.List( [1,2,3,4,5,6,7,8,9,0] );

var Result_1      = List_Test_1.reduce( No_Zero_Reducer);
var Result_2      = List_Test_2.reduce( No_Zero_Reducer);
var Result_3      = List_Test_1.reduce( Any_Non_Zero_Reducer);
var Result_4      = List_Test_2.reduce( Any_Non_Zero_Reducer);

print( Result_1 );
print(Result_2 );
print(Result_3 );
print(Result_4 );
```

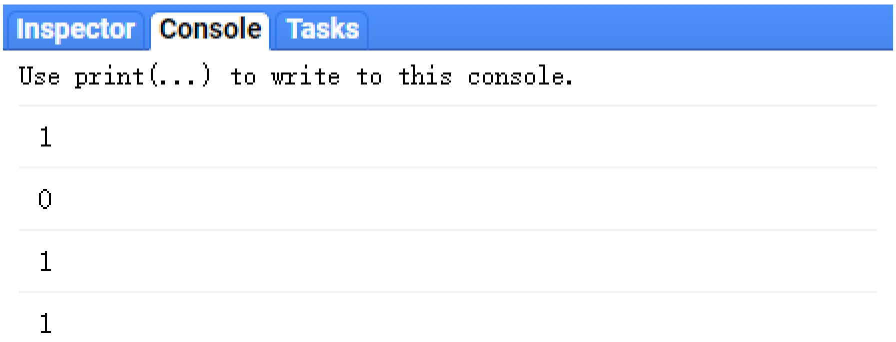


图 11.3 零值判断的 Reducer

零值判断的返回值中 0 代表“不成立”，1 代表“成立”。.allNonZero( )的含义是“是否全部都是非零值”，.anyNonZero( )的含义是“是否存在非零值”。

下边介绍转 List 的 Reducer，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
print(China_Cities.first())
var Tolist_Reducer = ee.Reducer.toList()
var City_List = China_Cities.reduceColumns(Tolist_Reducer,['Name_City'])
print(City_List)
```

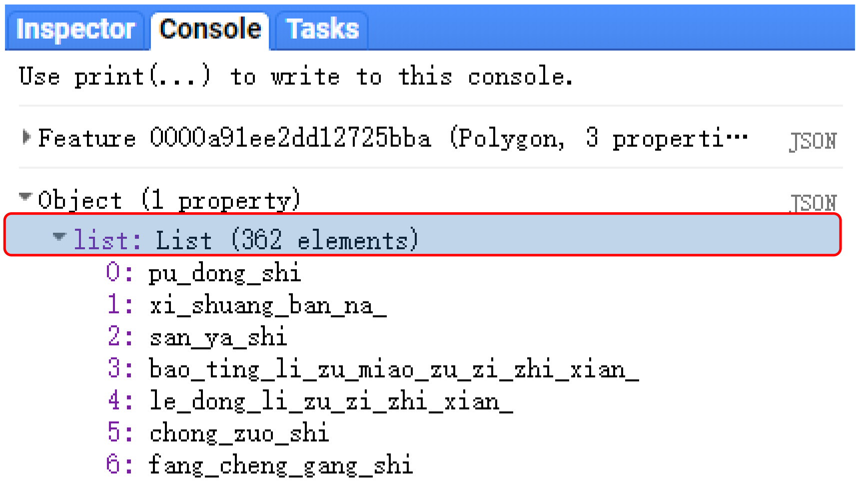


图 11.4 转 List 的 Reducer

下边介绍转 Collection 的 Reducer，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
print(China_Cities.first())
var Reducer_To_Collection = ee.Reducer.toCollection(['City','No'])
var City_Collection = China_Cities
                    .reduceColumns(Reducer_To_Collection,['Name_City','OBJECTID'])
print(City_Collection)
```

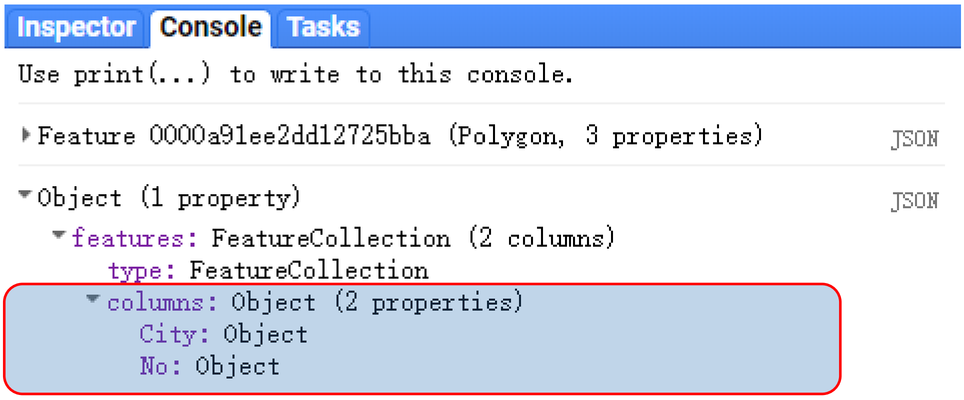


图 11.5 转 Collection 的 Reducer

下边介绍栅格的 Reducer，代码及执行效果如下：

```
var image   = ee.Image('LANDSAT/LC08/C01/T1/LC08_127040_20190407')

var maxValue = image.reduce(ee.Reducer.max());
var median = image.reduce(ee.Reducer.median());
var mean   = image.reduce(ee.Reducer.mean());

Map.centerObject(image, 10);

Map.addLayer(maxValue, {max: 30000}, 'Maximum value image');
Map.addLayer(median, {max: 10000}, 'Median value image');
Map.addLayer(mean, {max: 12000}, 'Mean value image');
```

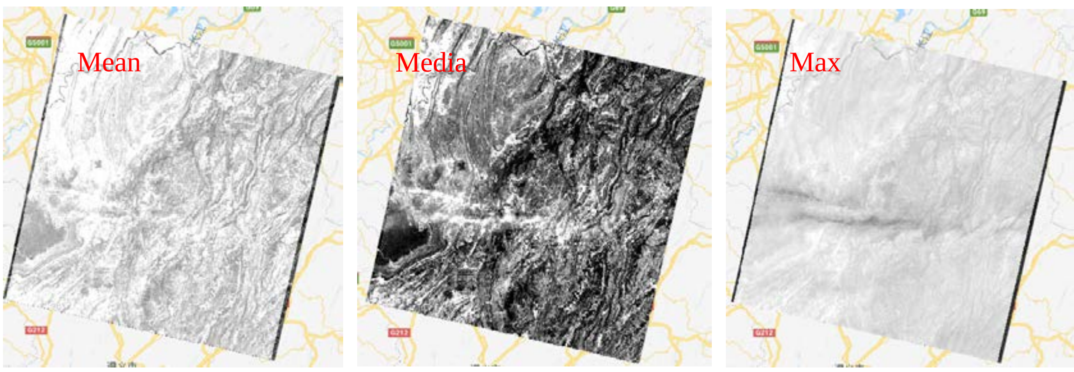


图 11.6 栅格的 Reducer

下边介绍数理统计的 Reducer，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
//利用.map 命令对每个 Feature 增加面积（km2）
function Add_Area (feature){
  var The_Area = ee.Number(feature.area())
  return feature.set('Area_km2',The_Area.divide(1000*1000))
}
var City_WithArea = China_Cities.map(Add_Area)
//设置数理统计的 Reducer
var Reducer_Product              = ee.Reducer.product()
var Reducer_Sum                  = ee.Reducer.sum()
var Reducer_Mean                 = ee.Reducer.mean()
var Reducer_Variance             = ee.Reducer.variance()
var Reducer_SampleVariance       = ee.Reducer.sampleVariance()
var Reducer_Std_dev              = ee.Reducer.stdDev()
var Reducer_SampleStdDev         = ee.Reducer.sampleStdDev()
var Reducer_Max                  = ee.Reducer.max()
var Reducer_Min                  = ee.Reducer.min()
var Reducer_Min_Max              = ee.Reducer.minMax()
var Reducer_Median               = ee.Reducer.median()
var Reducer_Mode                 = ee.Reducer.mode()
//进行 reduce 统计
var Area_Product         = City_WithArea.reduceColumns(Reducer_Product,['Area_km2'])
var Area_Sum             = City_WithArea.reduceColumns(Reducer_Sum,['Area_km2'])
var Area_Mean            = City_WithArea.reduceColumns(Reducer_Mean,['Area_km2'])
var Area_Variance        = City_WithArea.reduceColumns(Reducer_Variance,['Area_km2'])
var Area_Std_dev         = City_WithArea.reduceColumns(Reducer_Std_dev,['Area_km2'])
var Area_Max             = City_WithArea.reduceColumns(Reducer_Max,['Area_km2'])
var Area_Min             = City_WithArea.reduceColumns(Reducer_Min,['Area_km2'])
var Area_Range           = City_WithArea.reduceColumns(Reducer_Min_Max,['Area_km2'])
var Area_Median          = City_WithArea.reduceColumns(Reducer_Median,['Area_km2'])
var Area_Mode            = City_WithArea.reduceColumns(Reducer_Mode,['Area_km2'])
var Area_SampleStdDev  = City_WithArea.reduceColumns(Reducer_SampleStdDev,['Area_km2'])
var Area_SampleVariance= City_WithArea.reduceColumns(Reducer_SampleVariance,['Area_km2'])
//打印结果
print('Area_Product',Area_Product)
print('Area_Sum',Area_Sum)
print('Area_Mean',Area_Mean)
print('Area_Variance',Area_Variance)
print('Area_SampleVariance',Area_SampleVariance)
print('Area_Std_dev',Area_Std_dev)
print('Area_SampleStdDev',Area_SampleStdDev)
print('Area_Max',Area_Max)
print('Area_Min',Area_Min)
print('Area_Range',Area_Range)
print('Area_Median',Area_Median)
print('Area_Mode',Area_Mode)
```

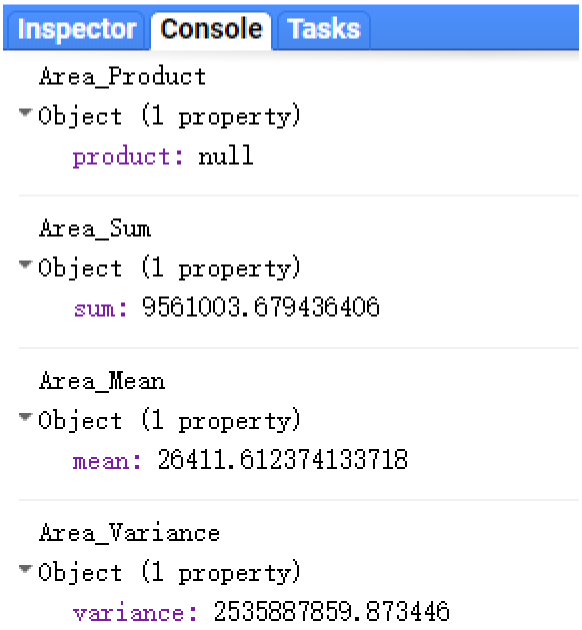


图 11.7 数理统计的 Reducer

下边介绍栅格的 Region Reducer，代码及执行效果如下：

```
// Load input imagery: Landsat 7 5-year composite.
var image = ee.Image('LANDSAT/LE7_TOA_5YEAR/2008_2012');

// Load an input region: Sierra Nevada mixed conifer forest.
var region = ee.Feature(ee.FeatureCollection(
    'ft:1Ec8IWsP8asxN-ywSqgXWMuBaxI6pPaeh6hC64lA')
    .filter(ee.Filter.eq('G200_REGIO', 'Sierra Nevada Coniferous Forests'))
    .first());

// Reduce the region. The region parameter is the Feature geometry.
var meanDictionary = image.reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: region.geometry(),
    scale: 30,
    maxPixels: 1e9
});

// The result is a Dictionary.   Print it.
print(meanDictionary);
```

代码来源：https://developers.google.com/earth-engine/reducers_reduce_region

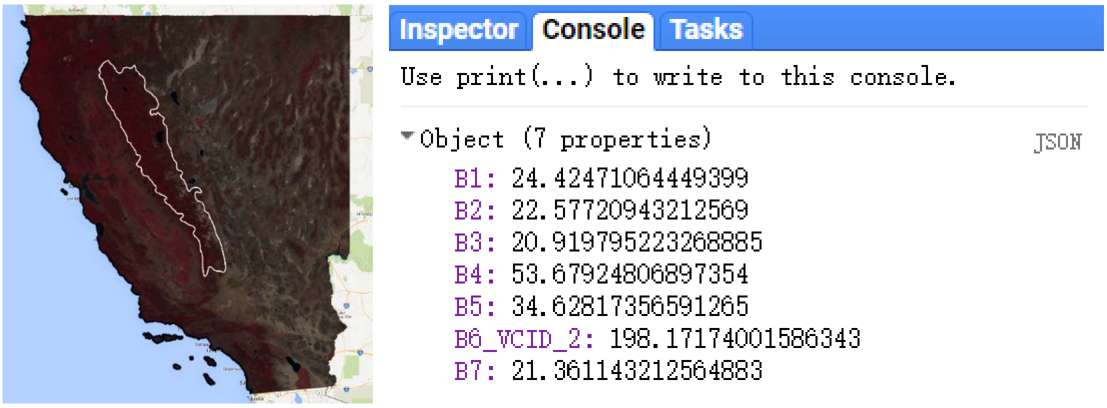


图 11.8 栅格的 Region Reducer

下边介绍栅格的 Neighborhood Reducer，代码及执行效果如下：

```
// Define a region in the redwood forest.
var redwoods = ee.Geometry.Rectangle(-124.0665, 41.0739, -123.934, 41.2029);
// Load input NAIP imagery and build a mosaic.
var naipCollection = ee.ImageCollection('USDA/NAIP/DOQQ')
  .filterBounds(redwoods)
  .filterDate('2012-01-01', '2012-12-31');
var naip = naipCollection.mosaic();
// Compute NDVI from the NAIP imagery.
var naipNDVI = naip.normalizedDifference(['N', 'R']);
// Compute standard deviation (SD) as texture of the NDVI.
var texture = naipNDVI.reduceNeighborhood({
  reducer: ee.Reducer.stdDev(),
  kernel: ee.Kernel.circle(7),
});
// Display the results.
Map.centerObject(redwoods, 12);
Map.addLayer(naip, {}, 'NAIP input imagery');
Map.addLayer(naipNDVI, {min: -1, max: 1, palette: ['FF0000', '00FF00']}, 'NDVI');
Map.addLayer(texture, {min: 0, max: 0.3}, 'SD of NDVI');
```

代码来源：https://developers.google.com/earth-engine/reducers_reduce_neighborhood

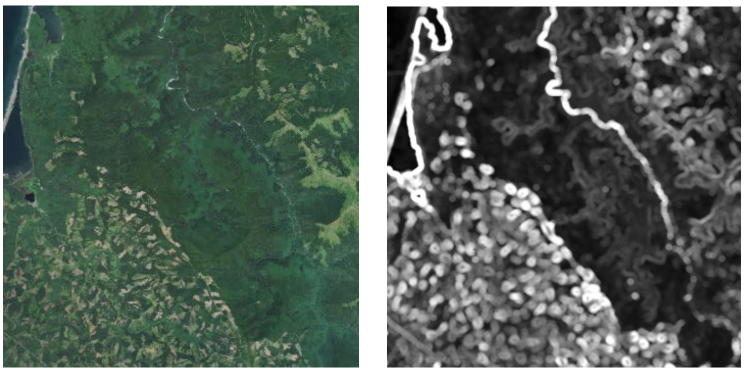


图 11.9 栅格的 Neighborhood Reducer

下边介绍栅格的区间统计 Reducer，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
function Add_Area (feature){
   var The_Area = ee.Number(feature.area())
   return feature.set('Area_km2',The_Area.divide(1000*1000))
}
var City_WithArea        = China_Cities.map(Add_Area)
var Reducer_Interval    = ee.Reducer.intervalMean( 0, 50 );
var Reducer_Percent     = ee.Reducer.percentile([30,50,70])

var Area_IntervalMean_Reduced = City_WithArea.reduceColumns(Reducer_Interval,['Area_km2'])
var Area_Percent_Reduced = City_WithArea.reduceColumns(Reducer_Percent,['Area_km2'])

print('Area_IntervalMean_Reduced',Area_IntervalMean_Reduced)
print('Area_Percent_Reduced',Area_Percent_Reduced)
```

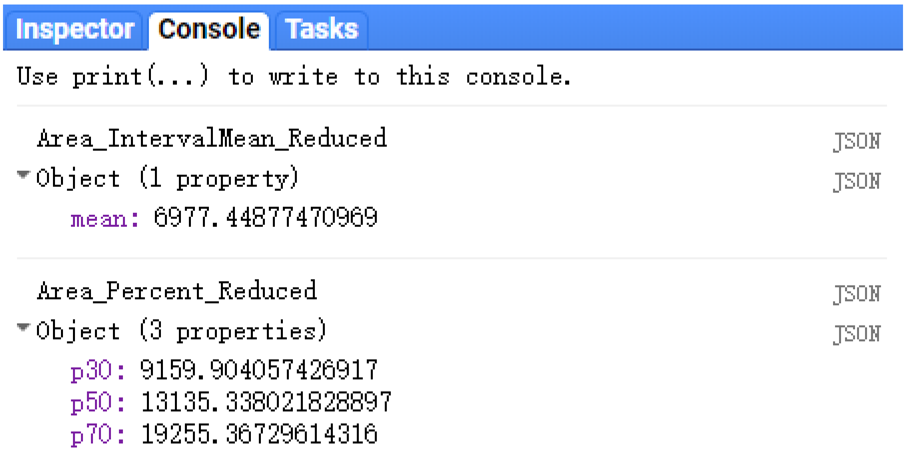


图 11.9 栅格的区间统计 Reducer

下边介绍线性拟合的 Reducer，代码及执行效果如下：

```
var Data_X = ee.Array([13,15,16,21,22,23,25,29,30,31,36,40,42,55,60,62,64,70,72,100,130])
var Data_Y = ee.List([11,10,11,12,12,13,13,12,14,16,17,13,14,22,14,21,21,24,17,23,34])
var Fig = ui.Chart.array.values(Data_X,0,Data_Y)
print(Fig)

var Data_x = ee.List([13,15,16,21,22,23,25,29,30,31,36,40,42,55,60,62,64,70,72,100,130])
var Data_y = ee.List([11,10,11,12,12,13,13,12,14,16,17,13,14,22,14,21,21,24,17,23,34])
var Linear_Reducer = ee.Reducer.linearFit()
var Fited = ee.List([Data_x,Data_y]).reduce(Linear_Reducer)
print(Fited)
```

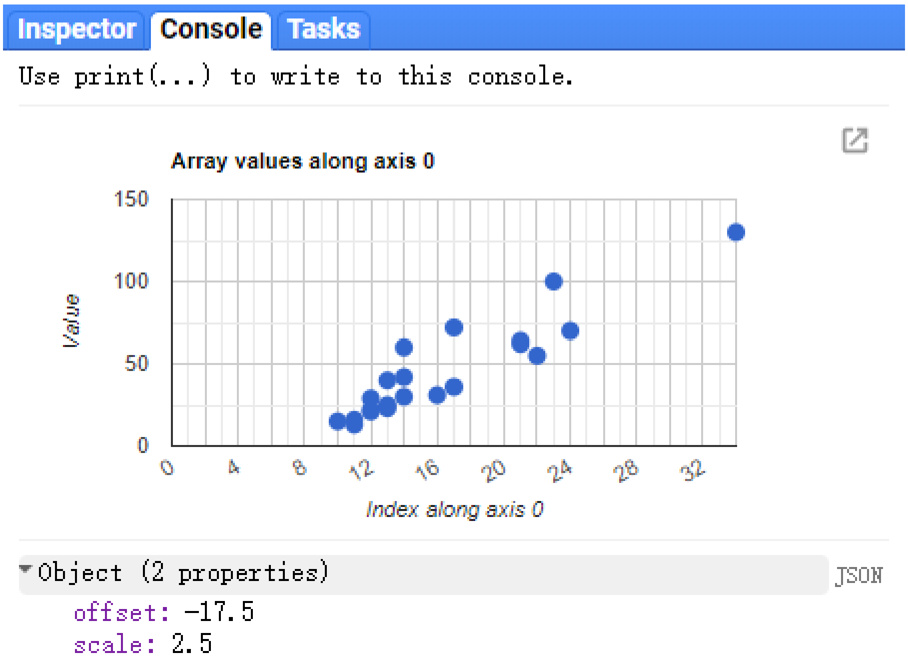


图 11.10 线性拟合的 Reducer

下边介绍栅格线性拟合，代码及执行效果如下：

```
// This function adds a time band to the image.
var createTimeBand = function(image) {
  // Scale milliseconds by a large constant to avoid very small slopes
  // in the linear regression output.
  return image.addBands(image.metadata('system:time_start').divide(1e18));
};

// Load the input image collection: projected climate data.
var collection = ee.ImageCollection('NASA/NEX-DCP30_ENSEMBLE_STATS')
  .filter(ee.Filter.eq('scenario', 'rcp85'))
  .filterDate(ee.Date('2006-01-01'), ee.Date('2050-01-01'))
  // Map the time band function over the collection.
  .map(createTimeBand);

// Reduce the collection with the linear fit reducer.
// Independent variable are followed by dependent variables.
var linearFit = collection.select(['system:time_start', 'pr_mean'])
  .reduce(ee.Reducer.linearFit());

// Display the results.
Map.setCenter(-100.11, 40.38, 5);
Map.addLayer(linearFit,
  {min: 0, max: [-0.9, 8e-5, 1], bands: ['scale', 'offset', 'scale']}, 'fit');
```

代码来源：https://developers.google.com/earth-engine/reducers_regression

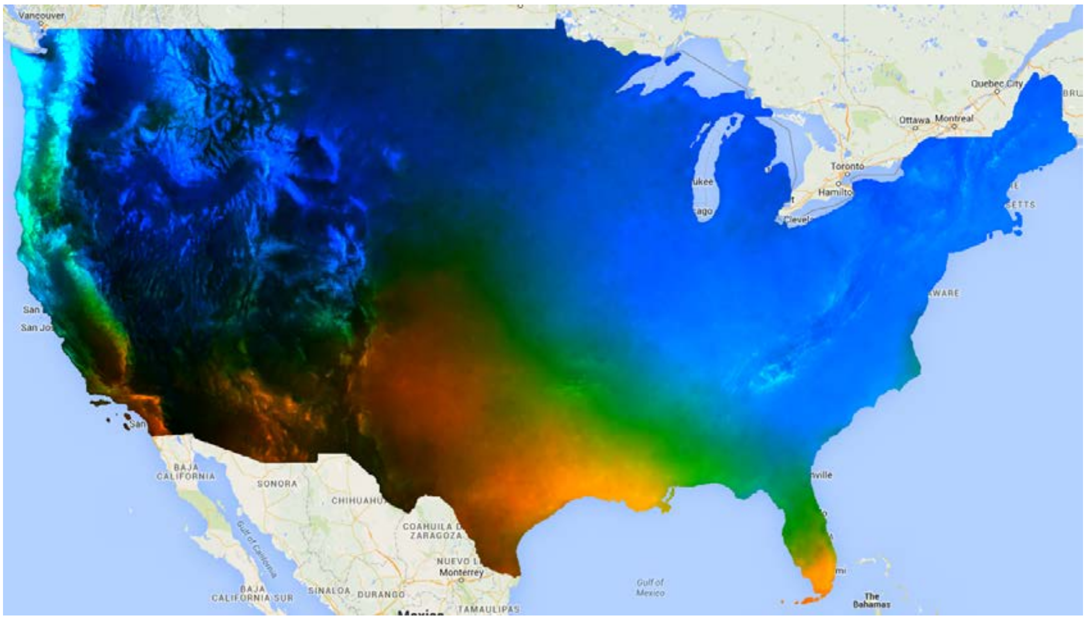


图 11.11 栅格的线性拟合

下边介绍 Reducer 的联合，代码及执行效果如下：

```
var Reducer_Max      = ee.Reducer.max( );
var Reducer_Min      = ee.Reducer.min( );
var Reducer_Combine    = Reducer_Max.combine( Reducer_Min);
var Array_Example = ee.Array( [ [1,2],
                                [3,4] ]);

var Combine_Reduced_1 = Array_Example.reduce( Reducer_Combine,[0],1 );
var Combine_Reduced_2 = Array_Example.reduce( Reducer_Combine,[1],0 );

print( 'Array_Example Array',    Array_Example );
print( 'Max of [1, 3] and Min of [2, 4]', Combine_Reduced_1 );
print( 'Max of [1, 2] and Min of [3, 4]', Combine_Reduced_2 );
```

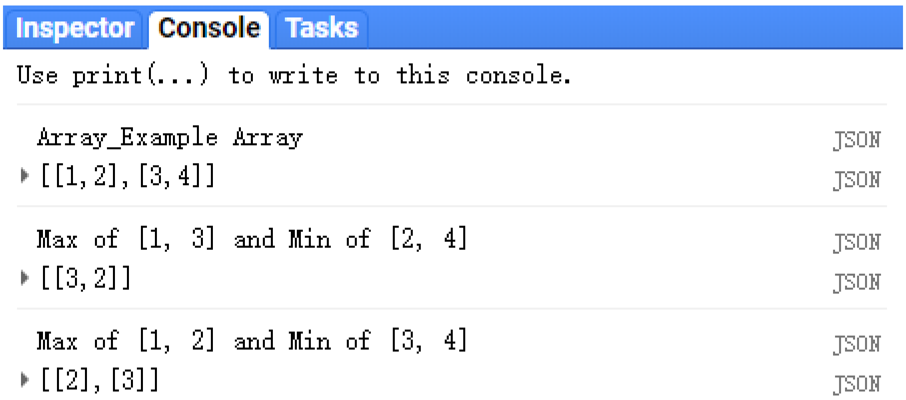


图 11.12 Reducer 的联合

需要注意的是，本例中，.reduce()有三个参数，第一个表示“缩减器”，第二个表示“缩减方向”，第三个表示“结果输出方向”。

下边介绍 Reducer 的重复，代码及执行效果如下：

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");
var Reducer_Repeat = ee.Reducer.frequencyHistogram().repeat(2)
var Province_City_Frequency = China_Cities.reduceColumns(Reducer_Repeat,['NAME_1','Name_City'])
print(Province_City_Frequency)
```

图 11.13 Reducer 的重复

需要注意的是，本例中，Reducer.repeat(2)相当于 Reduc.combine(Reducer)。在后边的命令.reduceColumns()中有两个参数，第一个参数 Reducer_Repeat 相当于要进行两侧 reducer，所以下一个参数用 List 的形式告诉 GEE 这两侧 reduce 分别对应哪两列数据。

下边介绍 Reducer 的结群，代码及执行效果如下：

```
// Load a collection of US counties with census data properties.
var counties = ee.FeatureCollection('ft:1S4EB6319wWW2sWQDPhDvmSBIVrD3iEmCLYB7nMM');

// Compute sums of the specified properties, grouped by state name.
var sums = counties
  .filter(ee.Filter.and(
    ee.Filter.neq('Census 2000 Population', null),
    ee.Filter.neq('Census 2000 Housing Units', null)))
  .reduceColumns({
    selectors: ['Census 2000 Population', 'Census 2000 Housing Units', 'StateName'],
    reducer: ee.Reducer.sum().repeat(2).group({
      groupField: 2,
      groupName: 'state',
    })
});

// Print the resultant Dictionary.
print(sums);
```

代码来源：https://developers.google.com/earth-engine/reducers_grouping

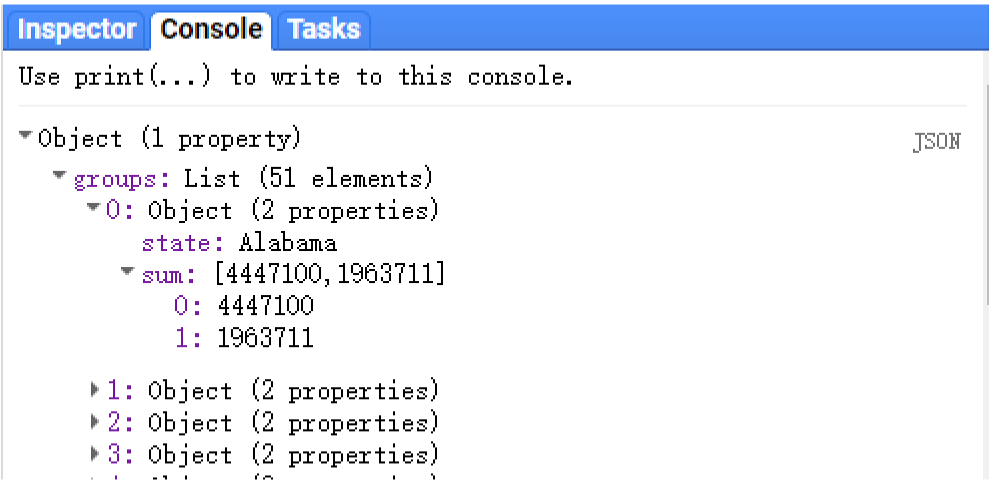


图 11.13 Reducer 的重复

这里需要注意两点，第一，缩减前 filter 的目的是排除空值数据。第二，.group 命令需要指出结群的数据列的位置，比如本列以“StateName”进行结群，则需要给出其位置，考虑到编程环境下数字编码由 0 开始，因此给出 2 来代表“StateName”所处的第 3 个位置。

下边是本节介绍的有关 Reducer 的常见命令，尝试回忆其语法与功能。

ee.Reducer.count()	.countEvery()	.first()	.histogram()	.allNonZero()
.anyNonZero()	.frequencyHistrogram()		.toList()	.toCollection()
sum()	.product()		.mean()	.variance()
.std_dev()	.sampleVariance()		sampleStdDev()	.max()
.min()	.minMax()		.median()	.mode()
.intervalMean()	.percentile()		.linearFit()	.combine()
.repeat()	.repeat()		.group()	

11.2 Kernel

如果一个像素的值由它周围像素的值来确定，那么确定这个像素值的过程叫做“卷积”，“周围”的概念则可以通过核 Kernel 来表现。

下边介绍几种常见的锐化卷积操作，代码及执行效果如下：

```
var CQ = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
              .filterMetadata('NAME','equals','Chong_Qing').first().geometry()

var DEM = ee.Image("USGS/SRTMGL1_003").clip(CQ);
var DEM_Roberts        = DEM.convolve(ee.Kernel.roberts())
var DEM_Prewitt        = DEM.convolve(ee.Kernel.prewitt())
var DEM_Sobel          = DEM.convolve(ee.Kernel.sobel())
var DEM_Compass        = DEM.convolve(ee.Kernel.compass())
var DEM_Kirsch         = DEM.convolve(ee.Kernel.kirsch())

Map.addLayer(DEM,{min:0,max:2000},'DEM')
Map.centerObject(CQ,7)
Map.addLayer(DEM_Roberts     ,{min:-60,max:60},'DEM_Roberts')
Map.addLayer(DEM_Prewitt     ,{min:-270,max:270},'DEM_Prewitt')
Map.addLayer(DEM_Sobel       ,{min:-370,max:370},'DEM_Sobel')
Map.addLayer(DEM_Compass     ,{min:-300,max:300},'DEM_Compass')
Map.addLayer(DEM_Kirsch      ,{min:-1100,max:1100},'DEM_Kirsch')
```

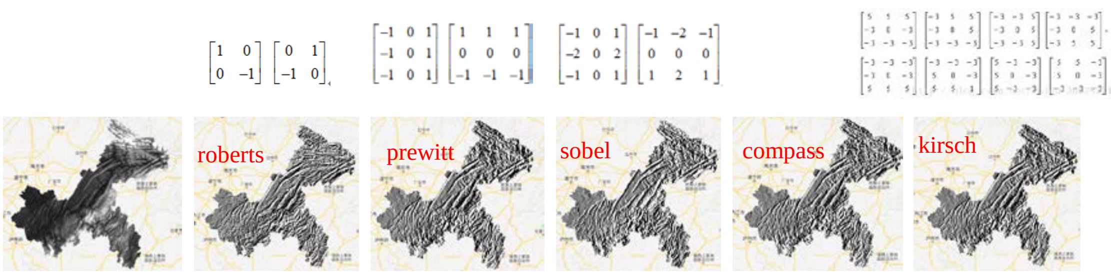
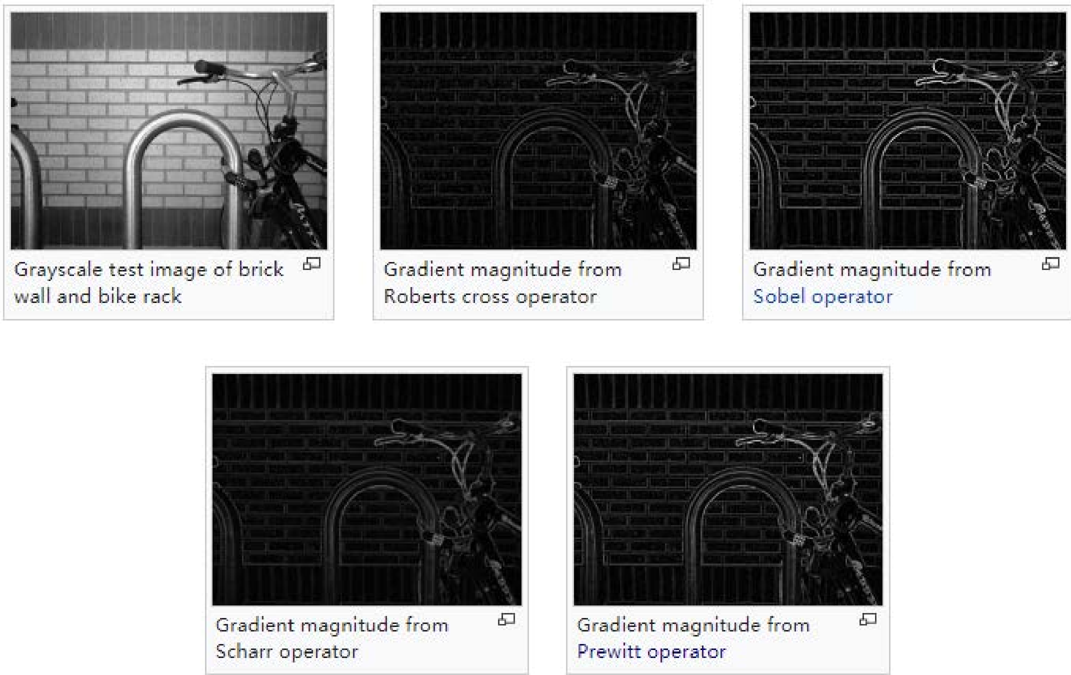


图 11.14 几种常见的卷积操作

可以看出，通过卷积操作之后，图像的纹路更加明显了(或者更加不明显了)。这种处理有助于让计算机识别图像特征，进而促进了图像数据的自动化分析。从下边的例子可以看出，卷积处理能显著的增强图像的可识别性。



图片来源：https://en.wikipedia.org/wiki/Roberts_cross

图 11.15 锐化卷积操作增加图像的可识别性

下边介绍几种常见的钝化卷积操作，代码及执行效果如下：

```
var CQ = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
                .filterMetadata('NAME','equals','Chong_Qing').first().geometry()
var DEM = ee.Image("USGS/SRTMGL1_003").clip(CQ);
var DEM_Euclidean   = DEM.convolve(ee.Kernel.euclidean( 10, 'pixels', true ))
var DEM_Gaussian    = DEM.convolve(ee.Kernel.gaussian( 10.0, 1.0, 'pixels', true ))
var DEM_Manhattan   = DEM.convolve(ee.Kernel.manhattan( 10, 'pixels', true ))
var DEM_Chebyshev   = DEM.convolve(ee.Kernel.chebyshev( 10, 'pixels', true ))

Map.centerObject(CQ,7)
Map.addLayer(DEM,{min:0,max:2000},'DEM')
Map.addLayer(DEM_Euclidean ,{min:281   ,max:681 },'DEM_Euclidean')
Map.addLayer(DEM_Gaussian    ,{min:358   ,max:898 },'DEM_Gaussian')
Map.addLayer(DEM_Manhattan ,{min:287   ,max:630 },'DEM_Manhattan')
Map.addLayer(DEM_Chebyshev ,{min:285   ,max:630 },'DEM_Chebyshev')
```

图 11.16 几种常见的钝化卷积操作

下边介绍几种常见的卷积核，代码及执行效果如下：

```
var CQ = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
              .filterMetadata('NAME','equals','Chong_Qing').first().geometry()
var DEM = ee.Image("USGS/SRTMGL1_003").clip(CQ);

var DEM_Circle      = DEM.convolve(ee.Kernel.circle( 10, 'pixels', true ))
var DEM_Octagon     = DEM.convolve(ee.Kernel.octagon( 10, 'pixels', true ))
var DEM_Square      = DEM.convolve(ee.Kernel.square( 10, 'pixels', true ))
var DEM_Diamond     = DEM.convolve( ee.Kernel.diamond( 10, 'pixels', true ))
var DEM_Cross       = DEM.convolve( ee.Kernel.cross( 10, 'pixels', true ))
var DEM_Plus        = DEM.convolve( ee.Kernel.plus( 10, 'pixels', true ))
var DEM_Fixed       = DEM.convolve( ee.Kernel.fixed( 3,3,[
                                              [ -1,0,0 ],
                                              [   0,0,0 ],
                                              [   0,0,1 ]      ]))

Map.centerObject(CQ,7)
Map.addLayer(DEM,              {min:0,max:2000},'DEM')
Map.addLayer(DEM_Circle ,      {min:328   ,max:746 },'DEM_Circle')
Map.addLayer(DEM_Octagon,      {min:323   ,max:721 },'DEM_Octagon')
Map.addLayer(DEM_Square ,      {min:320   ,max:694 },'DEM_Square')
Map.addLayer(DEM_Diamond ,     {min:334   ,max:786 },'DEM_Diamond')
Map.addLayer(DEM_Cross ,       {min:327   ,max:717 },'DEM_Cross')
Map.addLayer(DEM_Plus    ,     {min:340   ,max:799 },'DEM_Plus')
Map.addLayer(DEM_Fixed ,       {min:-382 ,max:415 },'DEM_Fixed')
```

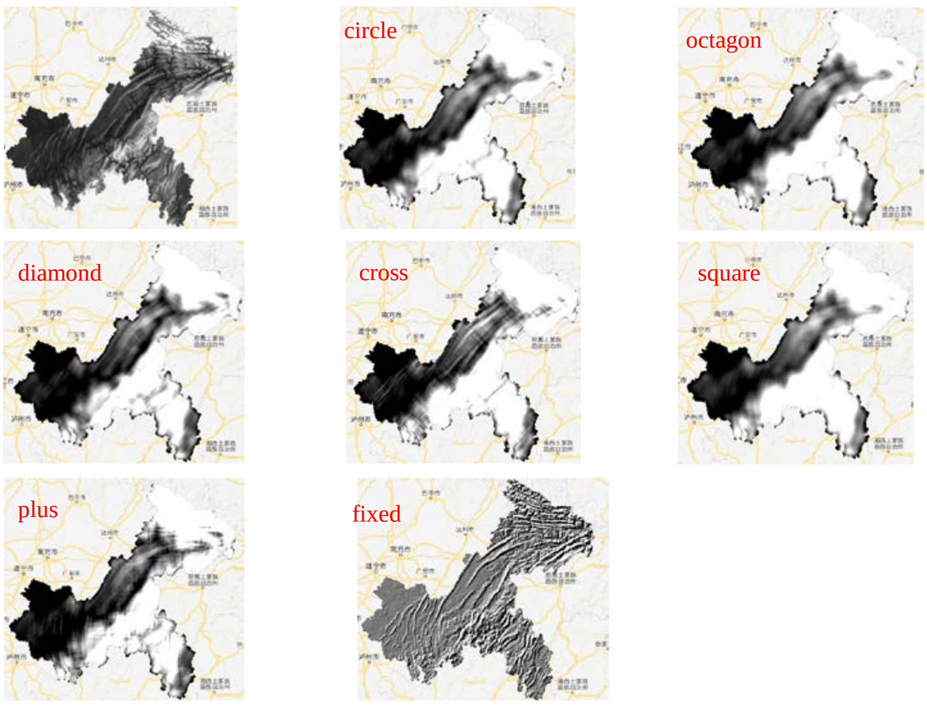


图 11.17 几种常见的卷积核

下边介绍 Kernel 的旋转和添加操作，代码及执行效果如下：

```
var CQ = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
                 .filterMetadata('NAME','equals','Chong_Qing').first().geometry()
var DEM = ee.Image("USGS/SRTMGL1_003").clip(CQ);

var DEM_Kernel              = DEM.convolve(ee.Kernel.roberts( ));
var DEM_Kernel_Rotate = DEM.convolve( ee.Kernel.roberts( ).rotate(1) );

var Add_Kernel = ee.Kernel.fixed( 3,3,[  [ -1,0,0 ],
                                          [   0,0,0 ],
                                          [   0,0,1 ]   ])
var DEM_Added    = DEM.convolve(ee.Kernel.roberts( ).add(Add_Kernel))

Map.addLayer(DEM,                        {min:0,max:2000},'DEM')
Map.centerObject(CQ,7)
Map.addLayer(DEM_Kernel ,             {min:-350,max:232 },'DEM_Kernel')
Map.addLayer(DEM_Kernel_Rotate,    {min:-409,max:320 },'DEM_Kernel_Rotate')
Map.addLayer(DEM_Added ,              {min:-380,max:294 },'DEM_Added')
```

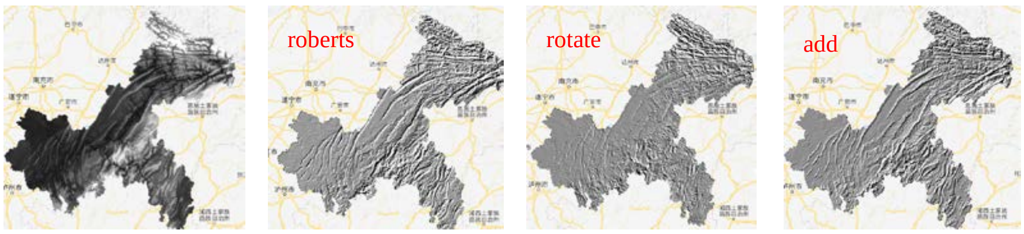


图 11.18 Kernel 的旋转和添加

下边是本节介绍过的常见的 Kernel 和卷积操作命令，尝试回忆其语法和功能。

ee.Kernel.roberts()	ee.Kernel.prewitt ()	ee.Kernel.sobel ()	ee.Kernel.compass ()
ee.Kernel.kirsch ()	ee.Kernel.laplacian4()	ee.Kernel.laplacian8()	ee.Kernel.euclidean()
ee.Kernel.gaussian()	ee.Kernel.manhattan()	ee.Kernel.chebyshev()	ee.Kernel.circle()
ee.Kernel.octagon()	ee.Kernel.square()	ee.Kernel.diamond()	ee.Kernel.cross()
ee.Kernel.plus()	ee.Kernel.fixed()	Kernel.rotate()	kernel.add()

11.3 Algorithm

算法的目的是减少重复运算，我们可将其理解为一个“小程序”，借助这个小程序可以对数据集内的每一个数据都进行同样的操作。

下边是算法的语法格式：

function　函数名（变量）　{　　操作　　}

算法的核心在于操作的编写。编写操作时要注意两点，第一，应该按照目标数据集确定变量名，比如针对栅格数据集的操作变量可以写作 Image 或者 img，这样能够提高操作的可读性。第二，操作必须包含 return 命令以告诉 GEE 算法的目的是什么。

下边是算法的两个例子，其目的分别是给栅格数据集中的每个图像添加 NDVI 数据，以及给 Feature Collection 中的每个 Feature 添加面积字段，具体代码和操作如下：

```
var L8   = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
             .filterBounds(ee.Geometry.Point(107.193, 29.1373))
             .filterDate('2018-01-01','2018-12-31')
             .select('B[4,5]')
             .limit(3);

function add_NDVI (image){
   var NDVI = image.normalizedDifference(['B5','B4'])
   return image.addBands(NDVI)
}

var L8_NDVI = L8.map(add_NDVI)

print(L8.first(), L8_NDVI.first())
```

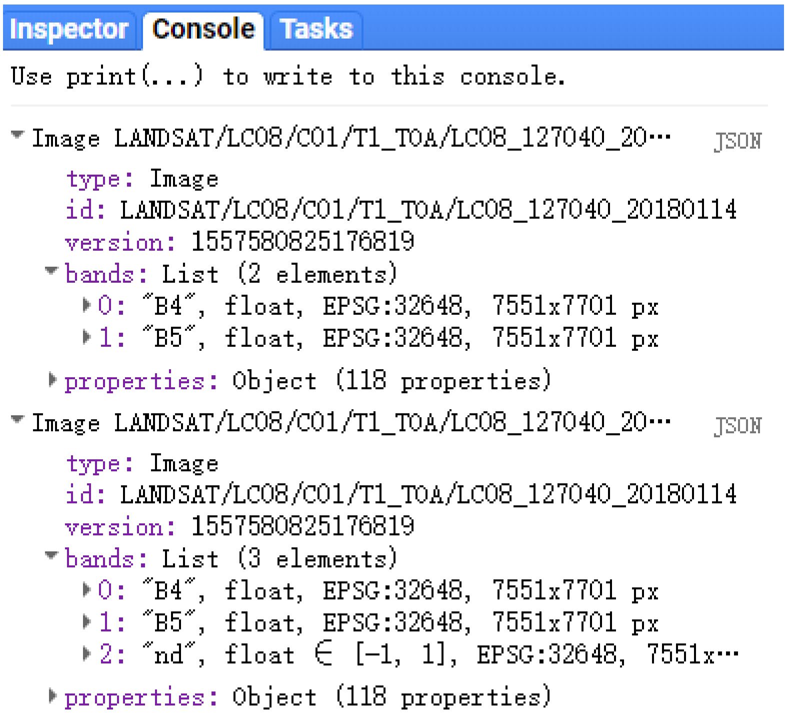


图 11.19 利用算法添加 NDVI

```
var China_Cities = ee.FeatureCollection("users/wangjinzhulala/China_Cities");

function Add_Area (feature){
   var The_Area = ee.Number(feature.area())
   return feature.set('Area_km2',The_Area.divide(1000*1000))
}

var City_With_Area = China_Cities.map(Add_Area)

print(China_Cities.first(),City_With_Area.first())
```

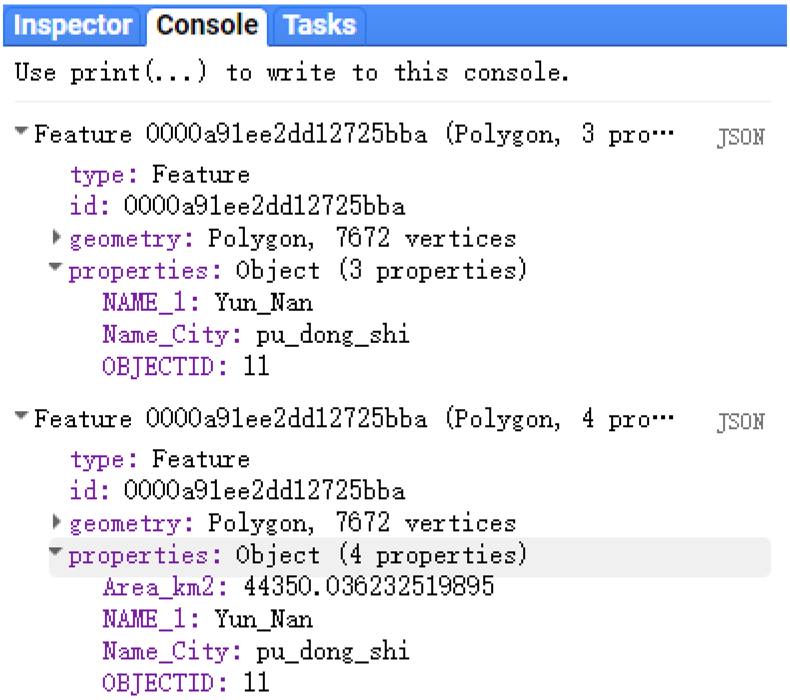


图 11.20 利用算法添加面积

本材料由王金柱（西南大学&迪肯大学）创作。如有需要请与我联系。
This doc contributed by Jinzhu Wang of Southwest University & Deakin University.
Email: wangjinzhulala@gmail.com