**PREPROCESAMIENTO S1 GEE**

Los pasos de preprocesamiento necesarios para los datos de rango terrestre detectado (GRD) del radar de apertura sintética (SAR) Sentinel-1 en Google Earth Engine. Estos pasos incluyen la conversión a decibeles (dB), la eliminación de ruido mediante un filtro de moteado y la corrección del terreno.

1. **Conversión a dB (decibeles):** los datos SAR sin procesar se expresan en unidades lineales de coeficiente de retrodispersión y es habitual convertirlos a unidades logarítmicas de dB para su posterior análisis. La conversión de los datos SAR a decibeles (dB) comprime el rango dinámico, mejora la diferenciación de píxeles, normaliza las secciones transversales del radar para facilitar su comparación y simplifica la interpretación y visualización de los datos.
2. **Eliminación de ruido mediante un filtro de moteado:** el moteado es un ruido granular de tipo "sal y pimienta" presente en las imágenes SAR debido a la interferencia aleatoria constructiva y destructiva de las ondas de retorno desfasadas pero coherentes dispersadas por la dispersión elemental dentro de cada celda de resolución. Un método común para la eliminación del ruido de moteado en las imágenes SAR es el uso de un filtro de moteado. Si bien existen muchos tipos de filtros de moteado, el filtro Lee es una opción popular.
3. **Corrección del terreno:** la corrección del terreno es necesaria para corregir las distorsiones geométricas en las imágenes SAR causadas por la geometría lateral de los datos SAR y la topografía del área observada. Este proceso mueve los píxeles de la imagen a su geolocalización correcta. El proceso de corrección del terreno generalmente implica el uso de un modelo digital de elevación (DEM) y el enfoque de rango-Doppler.

**Convertir entre unidades naturales y unidades de decibeles (dB):**

La conversión entre unidades naturales y decibeles (dB) se utiliza comúnmente cuando se trabaja con datos de rango terrestre detectado (GRD) del radar de apertura sintética (SAR) Sentinel-1 por varias razones:

1. **Reducción del rango dinámico:** la intensidad de la retrodispersión en las imágenes SAR puede variar en varios órdenes de magnitud. La conversión a dB mediante una escala logarítmica reduce este rango, lo que puede mejorar la visualización e interpretación de las imágenes.
2. **Calibración:** La conversión a dB es parte del proceso de calibración, que convierte la intensidad de retrodispersión que recibe el sensor en una sección transversal de radar normalizada (Sigma0). Esta medida calibrada tiene en cuenta el ángulo de incidencia global de la imagen y otras características específicas del sensor.
3. **Reducción de ruido:** la conversión a dB también se utiliza en el proceso de reducción de ruido. Por ejemplo, el conjunto de datos de anotación de vectores de ruido dentro de los productos GRD contiene vectores de ruido térmico para que los usuarios puedan aplicar una corrección de ruido térmico restando el ruido de la imagen de potencia detectada.

Por lo tanto, las funciones toNaturalque toDBusted proporciona se utilizan para convertir entre unidades de reflectividad de radar natural y unidades dB, lo que constituye un paso crucial en el preprocesamiento de los datos GRD del SAR Sentinel-1.

//Función para convertir de   
la función dB a Natural ( img ) {   
 return ee. Image ( 10.0 ). pow ( img. select ( 0 ). divide ( 10.0 ));   
}   
//Función para convertir a   
la función dB a DB ( img ) {   
 return ee. Image ( img ). log10 ( ). multiplicar ( 10.0 );   
}

**Corrección del terreno:**

Se requiere la corrección del terreno en los datos del radar de apertura sintética (SAR) Sentinel 1 **para corregir las distorsiones geométricas** que provocan errores de geolocalización. Estas distorsiones son inducidas por la naturaleza de las imágenes del SAR que miran de lado (en lugar de hacia abajo o hacia el nadir) y se ven agravadas por el terreno accidentado.

En esencia, la corrección del terreno mueve los píxeles de la imagen a la relación espacial adecuada entre sí. Esto es especialmente importante cuando se preparan datos para realizar mosaicos, donde se pueden tener varios productos de datos con diferentes ángulos de incidencia y niveles relativos de brillo.

La corrección del terreno se realizó utilizando SRTM DEM en Google Earth Engine. Implementado originalmente por Andreas Vollrath.

// Implementación de Andreas Vollrath (ESA), inspirada por Johannes Reiche (Wageningen)   
function terrainCorrection ( image ) {   
 var imgGeom = image.geometry ()   
 var srtm = ee.image ( ' USGS /SRTMGL1\_003' ). clip (imgGeom) // 30m srtm   
 var sigma0Pow = ee.image.constant ( 10 ) .pow ( image.divide ( 10.0 ) ) // Artículo (los números se relacionan con los capítulos) // 2.1.1 Geometría del radar var theta\_i = image.select ( 'angle' ) var phi\_i = ee.terrain.aspect (theta\_i) .reduceRegion ( ee.Reducer.mean ( ), theta\_i.get ( ' system :footprint' ), 1000 ) . obtener ( 'aspecto' ) // 2.1.2 Geometría del terreno var alpha\_s = ee. Terrain . slope (srtm). select ( 'slope' ) var phi\_s = ee. Terrain . aspect (srtm). select ( 'aspecto' ) // 2.1.3 Geometría del modelo // reducir a 3 ángulos var phi\_r = ee. Image . constant (phi\_i). subtract (phi\_s) // convertir todo a radianes var phi\_rRad = phi\_r. multiplicar ( Math . PI / 180 ) var alpha\_sRad = alpha\_s. multiplicar ( Math . PI / 180 ) var theta\_iRad = theta\_i. multiplicar ( Math . PI / 180 ) var ninetiesRad = ee. Image . constant ( 90 ). multiplicar ( Math . PI / 180 ) // inclinación de la pendiente en el rango (ecuación 2) var alpha\_r = (alpha\_sRad. tan (). multiplicar (phi\_rRad. cos ())). atan ()   
  
 // inclinación de la pendiente en acimut (ecuación 3)   
 var alpha\_az = (alpha\_sRad. tan (). multiplicar (phi\_rRad. sin ())). atan ()   
  
 // ángulo de incidencia local (ecuación 4)   
 var theta\_lia = (alpha\_az. cos (). multiplicar ((theta\_iRad. restar (alpha\_r)). cos ())). acos ()   
 var theta\_liaDeg = theta\_lia. multiplicar ( 180 / Math . PI )   
 // 2.2   
 // Gamma\_nought\_flat   
 var gamma0 = sigma0Pow. divide (theta\_iRad. cos ())   
 var gamma0dB = ee. Image . constant ( 10 ). multiplicar (gamma0. log10 ())   
 var ratio\_1 = gamma0dB. select ( 'VV' ). subtract (gamma0dB. select ( 'VH' ))   
  
 // Modelo volumétrico   
 var nominator = (ninetyRad. subtract (theta\_iRad). add (alpha\_r)). tan ()   
 var denominador = (ninetyRad. subtract (theta\_iRad)). tan ()   
 var volModel = (nominator. divide (denominator)). abs ()   
  
 // aplicar modelo   
 var gamma0\_Volume = gamma0. divide (volModel)   
 var gamma0\_VolumeDB = ee. Image . constant ( 10 ). multiplicar (gamma0\_Volume. log10 ())   
  
 // agregamos una máscara de sombra/capa a la implementación original   
 // capa, donde pendiente > ángulo de visión del radar   
 var alpha\_rDeg = alpha\_r. multiplicar ( 180 / Math . PI )   
 var layover = alpha\_rDeg. lt (theta\_i);   
  
 // sombra donde LIA > 90   
 var shadow = theta\_liaDeg. lt ( 85 )   
  
 // calcular la relación para RGB vis   
 var ratio = gamma0\_VolumeDB. select ( 'VV' ). subtract (gamma0\_VolumeDB. select( 'VH' ))   
  
 var salida = gamma0\_VolumeDB.addBands (ratio) .addBands ( alpha\_r) .addBands (phi\_s) .addBands (theta\_iRad) .addBands (layover) .addBands (shadow) .addBands (gamma0dB) .addBands ( ratio\_1) return image.addBands ( salida.select ([ 'VV' , ' VH' ], [ 'VV' , 'VH' ] ), null , true ) }

**Reducción de ruido de moteado (filtro Lee refinado)**

El filtro Lee refinado se utiliza habitualmente en el preprocesamiento de los datos de detección de alcance terrestre (GRD) del radar de apertura sintética (SAR) Sentinel-1 para reducir el ruido de moteado. El ruido de moteado aparece como patrones aleatorios y granulares en las imágenes SAR, a menudo causados ​​por la interferencia de ondas electromagnéticas. Este ruido puede degradar la calidad de las imágenes y afectar su interpretación.

El filtro Lee refinado es particularmente eficaz porque conserva los bordes, las características lineales, la información de textura y los puntos de interés. Esto lo hace superior para la interpretación visual en comparación con otros filtros de moteado de un solo producto.

//Apllying a Refined Lee Speckle filter as coded in the SNAP 3.0 S1TBX:  
//https://github.com/senbox-org/s1tbx/blob/master/s1tbx-op-sar-processing/src/main/java/org/esa/s1tbx/sar/gpf/filtering/SpeckleFilters/RefinedLee.java  
//Adapted by Guido Lemoine  
  
function RefinedLee(img) {  
 // img must be in natural units, i.e. not in dB!  
 // Set up 3x3 kernels   
  
 // convert to natural.. do not apply function on dB!  
 var myimg = toNatural(img);  
  
 var weights3 = ee.List.repeat(ee.List.repeat(1,3),3);  
 var kernel3 = ee.Kernel.fixed(3,3, weights3, 1, 1, false);  
  
 var mean3 = myimg.reduceNeighborhood(ee.Reducer.mean(), kernel3);  
 var variance3 = myimg.reduceNeighborhood(ee.Reducer.variance(), kernel3);  
  
 // Use a sample of the 3x3 windows inside a 7x7 windows to determine gradients and directions  
 var sample\_weights = ee.List([[0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0], [0,1,0,1,0,1,0], [0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0]]);  
  
 var sample\_kernel = ee.Kernel.fixed(7,7, sample\_weights, 3,3, false);  
  
 // Calculate mean and variance for the sampled windows and store as 9 bands  
 var sample\_mean = mean3.neighborhoodToBands(sample\_kernel);   
 var sample\_var = variance3.neighborhoodToBands(sample\_kernel);  
  
 // Determine the 4 gradients for the sampled windows  
 var gradients = sample\_mean.select(1).subtract(sample\_mean.select(7)).abs();  
 gradients = gradients.addBands(sample\_mean.select(6).subtract(sample\_mean.select(2)).abs());  
 gradients = gradients.addBands(sample\_mean.select(3).subtract(sample\_mean.select(5)).abs());  
 gradients = gradients.addBands(sample\_mean.select(0).subtract(sample\_mean.select(8)).abs());  
  
 // And find the maximum gradient amongst gradient bands  
 var max\_gradient = gradients.reduce(ee.Reducer.max());  
  
 // Create a mask for band pixels that are the maximum gradient  
 var gradmask = gradients.eq(max\_gradient);  
  
 // duplicate gradmask bands: each gradient represents 2 directions  
 gradmask = gradmask.addBands(gradmask);  
  
 // Determine the 8 directions  
 var directions = sample\_mean.select(1).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(7))).multiply(1);  
 directions = directions.addBands(sample\_mean.select(6).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(2))).multiply(2));  
 directions = directions.addBands(sample\_mean.select(3).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(5))).multiply(3));  
 directions = directions.addBands(sample\_mean.select(0).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(8))).multiply(4));  
 // The next 4 are the not() of the previous 4  
 directions = directions.addBands(directions.select(0).not().multiply(5));  
 directions = directions.addBands(directions.select(1).not().multiply(6));  
 directions = directions.addBands(directions.select(2).not().multiply(7));  
 directions = directions.addBands(directions.select(3).not().multiply(8));  
  
 // Mask all values that are not 1-8  
 directions = directions.updateMask(gradmask);  
  
 // "collapse" the stack into a singe band image (due to masking, each pixel has just one value (1-8) in it's directional band, and is otherwise masked)  
 directions = directions.reduce(ee.Reducer.sum());   
  
 var sample\_stats = sample\_var.divide(sample\_mean.multiply(sample\_mean));  
  
 // Calculate localNoiseVariance  
 var sigmaV = sample\_stats.toArray().arraySort().arraySlice(0,0,5).arrayReduce(ee.Reducer.mean(), [0]);  
  
 // Set up the 7\*7 kernels for directional statistics  
 var rect\_weights = ee.List.repeat(ee.List.repeat(0,7),3).cat(ee.List.repeat(ee.List.repeat(1,7),4));  
  
 var diag\_weights = ee.List([[1,0,0,0,0,0,0], [1,1,0,0,0,0,0], [1,1,1,0,0,0,0],   
 [1,1,1,1,0,0,0], [1,1,1,1,1,0,0], [1,1,1,1,1,1,0], [1,1,1,1,1,1,1]]);  
  
 var rect\_kernel = ee.Kernel.fixed(7,7, rect\_weights, 3, 3, false);  
 var diag\_kernel = ee.Kernel.fixed(7,7, diag\_weights, 3, 3, false);  
  
 // Create stacks for mean and variance using the original kernels. Mask with relevant direction.  
 var dir\_mean = myimg.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel).updateMask(directions.eq(1));  
 var dir\_var = myimg.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel).updateMask(directions.eq(1));  
  
 dir\_mean = dir\_mean.addBands(myimg.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel).updateMask(directions.eq(2)));  
 dir\_var = dir\_var.addBands(myimg.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel).updateMask(directions.eq(2)));  
  
 // and add the bands for rotated kernels  
 for (var i=1; i<4; i++) {  
 dir\_mean = dir\_mean.addBands(myimg.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));  
 dir\_var = dir\_var.addBands(myimg.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));  
 dir\_mean = dir\_mean.addBands(myimg.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));  
 dir\_var = dir\_var.addBands(myimg.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));  
 }  
  
 // "collapse" the stack into a single band image (due to masking, each pixel has just one value in it's directional band, and is otherwise masked)  
 dir\_mean = dir\_mean.reduce(ee.Reducer.sum());  
 dir\_var = dir\_var.reduce(ee.Reducer.sum());  
  
 // A finally generate the filtered value  
 var varX = dir\_var.subtract(dir\_mean.multiply(dir\_mean).multiply(sigmaV)).divide(sigmaV.add(1.0));  
  
 var b = varX.divide(dir\_var);  
  
 var result = dir\_mean.add(b.multiply(myimg.subtract(dir\_mean)));  
 //return(result);  
 return(img.addBands(ee.Image(toDB(result.arrayGet(0))).rename("filter")));  
   
}