Avalanche Occurrence Detection using Sentinel 1 C-band SAR Data

CE-716: Data Processing in Remote Sensing - Mini Project

Submitted by: Sudhir Dhamija (22D0312), Manav Kumar Kawath (22M0593)

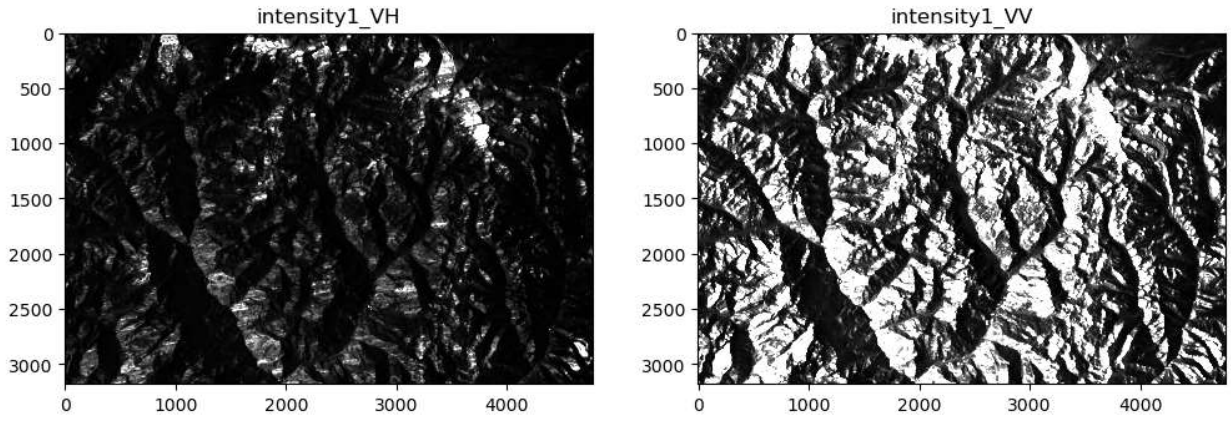
In [1]:
```
import numpy as np;
import sys;
import matplotlib.pyplot as plt;
from osgeo import gdal;
from osgeo.gdalconst import *;
import os;
import rasterio as rio
import folium
from folium import plugins
import rioxarray as rxr
import earthpy as et
import earthpy.spatial as es
import scipy.ndimage
```

In [2]:
```
path="D:\Data\Final Tiff"
pre_image=gdal.Open(path+"\subset_1_of_S1A_IW_SLC__1SDV_20200115T120535_20200115T120602_030811_0388D1_27C3_s
post_image=gdal.Open(path+"\subset_6_of_S1A_IW_SLC__1SDV_20200127T120535_20200127T120602_030986_038EFB_9379_
#post_image=gdal.Open(path+"\subset_0_of_S1A_IW_SLC__1SDV_20230205T124823_20230205T124850_047101_05A693_FDA8
```

Read & Display Image Previous to Avalanche Occurence (15 Jan 2020)

In [3]:
```
#Read Pre Image
i1_VH = pre_image.GetRasterBand(1).ReadAsArray()
q1_VH = pre_image.GetRasterBand(2).ReadAsArray()
i1_VV = pre_image.GetRasterBand(3).ReadAsArray()
q1_VV = pre_image.GetRasterBand(4).ReadAsArray()
intensity1_VH = pre_image.GetRasterBand(5).ReadAsArray()
intensity1_VV = pre_image.GetRasterBand(6).ReadAsArray()
shadow1 = pre_image.GetRasterBand(7).ReadAsArray()

i1_VH = scipy.ndimage.convolve(i1_VH, np.full((3, 3), 1.0/9))
q1_VH = scipy.ndimage.convolve(q1_VH, np.full((3, 3), 1.0/9))
i1_VV = scipy.ndimage.convolve(i1_VV, np.full((3, 3), 1.0/9))
q1_VV = scipy.ndimage.convolve(q1_VV, np.full((3, 3), 1.0/9))


figure, ax = plt.subplots(1,2, figsize=(12,4))

im1 = ax[0].imshow(intensity1_VH, vmin=0, vmax=0.4, cmap='gray')
ax[0].set_title("intensity1_VH")

im2 = ax[1].imshow(intensity1_VV, vmin=0, vmax=0.4, cmap='gray')
ax[1].set_title("intensity1_VV")
```

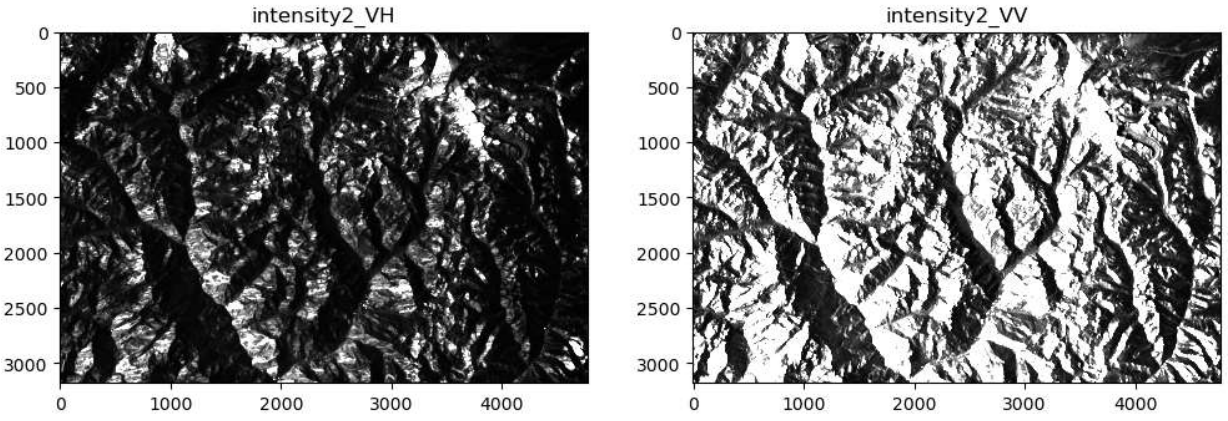
Out[3]:
```
Text(0.5, 1.0, 'intensity1_VV')
```



Read & Display Image Post Avalanche Occurence (27 Jan 2020)

In [4]:
```
#Read Post Image
i2_VH = post_image.GetRasterBand(1).ReadAsArray()
q2_VH = post_image.GetRasterBand(2).ReadAsArray()
i2_VV = post_image.GetRasterBand(4).ReadAsArray()
q2_VV = post_image.GetRasterBand(5).ReadAsArray()
intensity2_VH = post_image.GetRasterBand(3).ReadAsArray()
intensity2_VV = post_image.GetRasterBand(6).ReadAsArray()
shadow2 = post_image.GetRasterBand(7).ReadAsArray()

i2_VH = scipy.ndimage.convolve(i2_VH, np.full((3, 3), 1.0/9))
q2_VH = scipy.ndimage.convolve(q2_VH, np.full((3, 3), 1.0/9))
i2_VV = scipy.ndimage.convolve(i2_VV, np.full((3, 3), 1.0/9))
q2_VV = scipy.ndimage.convolve(q2_VV, np.full((3, 3), 1.0/9))

figure, ax = plt.subplots(1,2, figsize=(12,4))

im1 = ax[0].imshow(intensity2_VH, vmin=0, vmax=0.4, cmap='gray')
ax[0].set_title("intensity2_VH")

im2 = ax[1].imshow(intensity2_VV, vmin=0, vmax=0.4, cmap='gray')
ax[1].set_title("intensity2_VV")
```

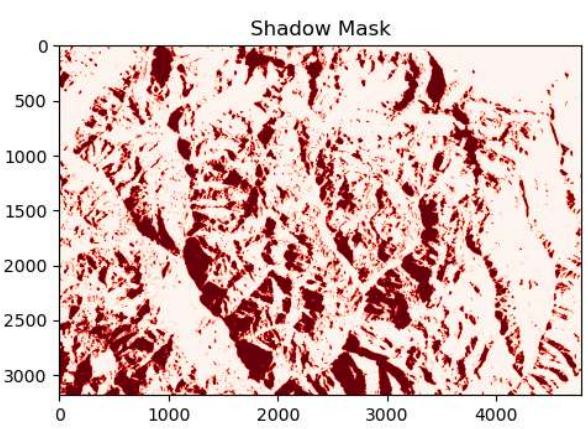
Out[4]:
```
Text(0.5, 1.0, 'intensity2_VV')
```

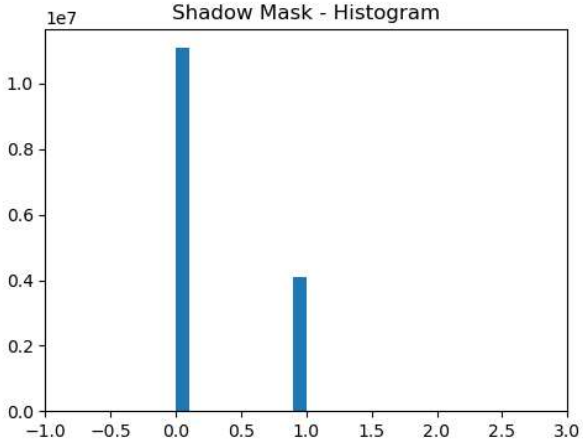


Generate Shadow Mask - Any result in these areas will be neglected

In [5]:
```
# # Mask Shadows

shadow1[np.isnan(shadow1)] = 0
shadow1[shadow1 > 0] = 1

shadow2[np.isnan(shadow2)] = 0
shadow2[shadow2 > 0] = 1


shadow_mask = shadow1+shadow2
shadow_mask[shadow_mask > 1] = 1

figure, ax = plt.subplots(1,2, figsize=(12,4))
ax[0].imshow(shadow_mask, vmin=0, vmax=1, cmap='Reds')
ax[0].set_title("Shadow Mask")
ax[1].hist(shadow_mask.ravel(),  bins=10)
ax[1].set_title("Shadow Mask - Histogram")
ax[1].set_xlim(-1,3)
```

Out[5]:
```
(-1.0, 3.0)
```

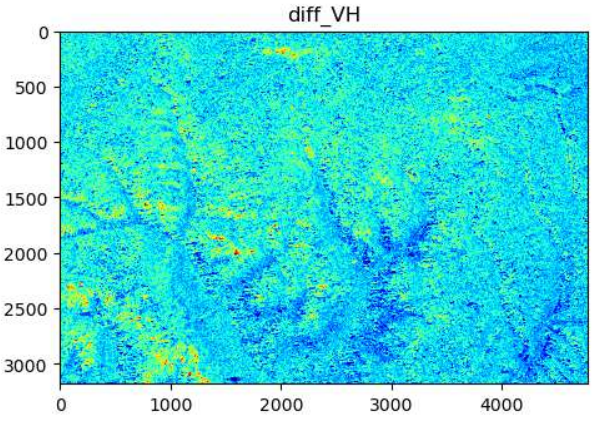


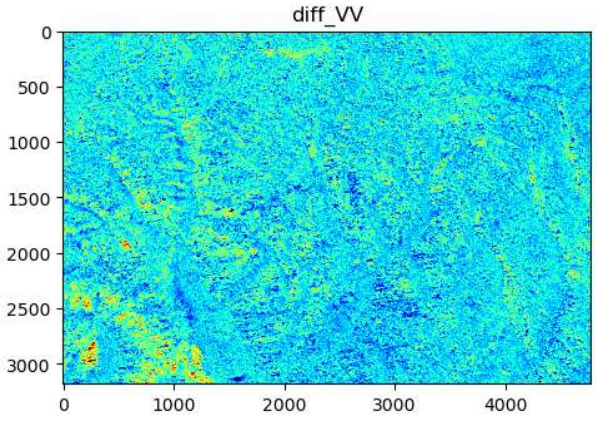


Generate Difference Image (Post_image - Pre_Image)

In [6]:
```
intensity2_VV = np.where(intensity2_VV == 0,0.000000001, intensity2_VV)
intensity2_VH = np.where(intensity2_VH == 0,0.000000001, intensity2_VH)
intensity1_VV = np.where(intensity1_VV == 0,0.000000001, intensity1_VV)
intensity1_VH = np.where(intensity1_VH == 0,0.000000001, intensity1_VH)

diff_VV = 10*np.log10(intensity2_VV) - 10*np.log10(intensity1_VV)
diff_VH = 10*np.log10(intensity2_VH) - 10*np.log10(intensity1_VH)

figure, ax = plt.subplots(1,2, figsize=(12,4))

im1 = ax[0].imshow(diff_VH, vmin=0, vmax=10, cmap='jet')
ax[0].set_title("diff_VH")

im2 = ax[1].imshow(diff_VV, vmin=0, vmax=10, cmap='jet')
ax[1].set_title("diff_VV")
```

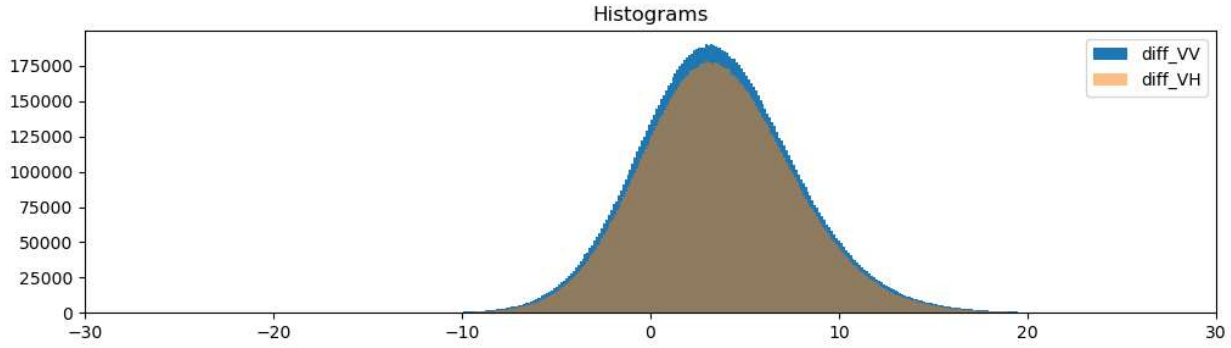
Out[6]:
```
Text(0.5, 1.0, 'diff_VV')
```





In [7]:
```
figure, ax = plt.subplots(1,1, figsize=(12,3))

ax.hist(diff_VV.ravel(),  bins=500, label="diff_VV")
ax.hist(diff_VH.ravel(),  bins=500, label="diff_VH" ,alpha=0.5)
ax.set_title("Histograms")
ax.set_xlim(-30,30)
ax.legend()
```

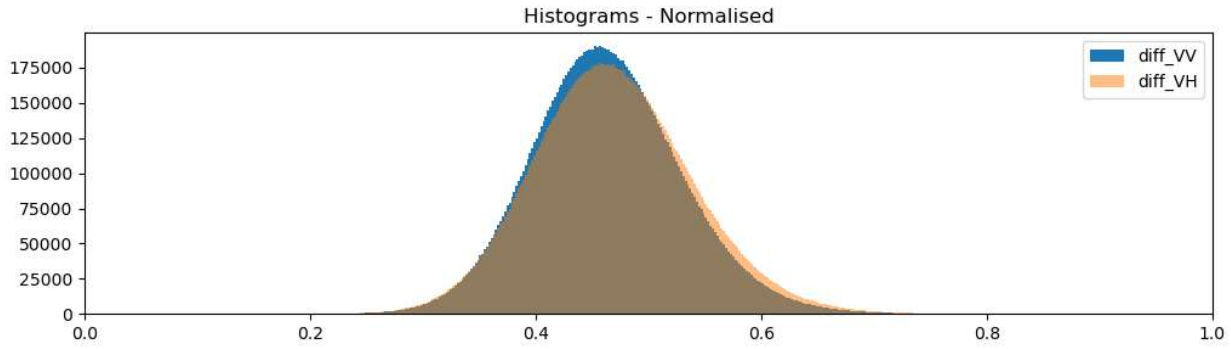
Out[7]:
```
<matplotlib.legend.Legend at 0x1d40c926830>
```



In [8]:
```
def NormalizeData(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data))

diff_VV_norm = NormalizeData(diff_VV)
diff_VH_norm = NormalizeData(diff_VH)
```

In [9]:
```
figure, ax = plt.subplots(1,1, figsize=(12,3))

ax.hist(diff_VV_norm.ravel(),  bins=500, label="diff_VV")
ax.hist(diff_VH_norm.ravel(),  bins=500, label="diff_VH" ,alpha=0.5)
ax.set_title("Histograms - Normalised")
ax.set_xlim(0,1)
ax.legend()
```

Out[9]:
```
<matplotlib.legend.Legend at 0x1d40d01b0d0>
```



In [ ]:
```
#diff = diff_VV_norm + diff_VH_norm
diff = np.multiply(diff_VV_norm, diff_VH_norm)

figure, ax = plt.subplots(1,1, figsize=(12,8))
plt.imshow(diff, vmin=0.1, vmax=0.3, cmap='jet')
plt.title("Difference Image")
```

In [ ]:
```
diff = scipy.ndimage.convolve(diff, np.full((10, 10), 1.0/100))

figure, ax = plt.subplots(1,1, figsize=(12,8))
plt.imshow(diff, vmin=0.1, vmax=0.3, cmap='jet')
plt.title("Difference Image - After 10x10 mean smoothening")
```

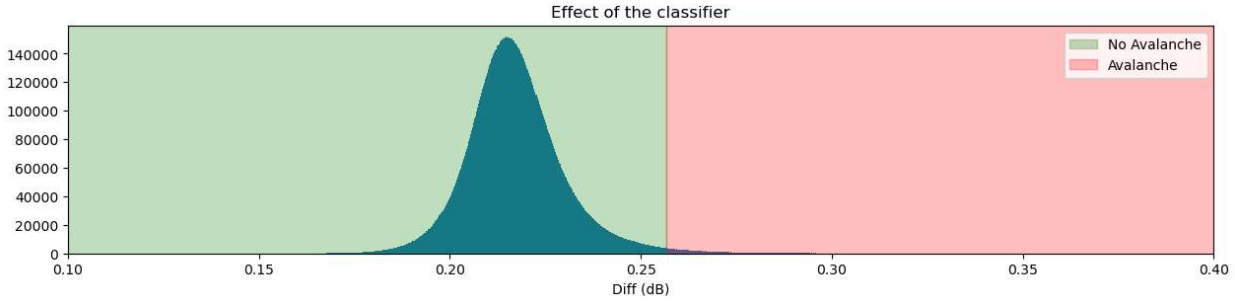
Only keep values where diffrence in backscattering is more than (Mean + 3xStd Deviation)

In [12]:
```
diff_mean = np.mean(diff)
print("Mean: ", diff_mean)

diff_stdev = np.std(diff)
print("std: ", diff_stdev)

threshold = diff_mean + 3 * diff_stdev
print("threshold: ", threshold)
```

```
Mean:  0.21738188
std:  0.013073493
threshold:  0.25660235807299614
```

In [13]:
```
fig, ax = plt.subplots(figsize=(15, 3))
ax.hist(diff.ravel(),  bins=1000, density=False)
plt.xlim(0.1,0.4)
```

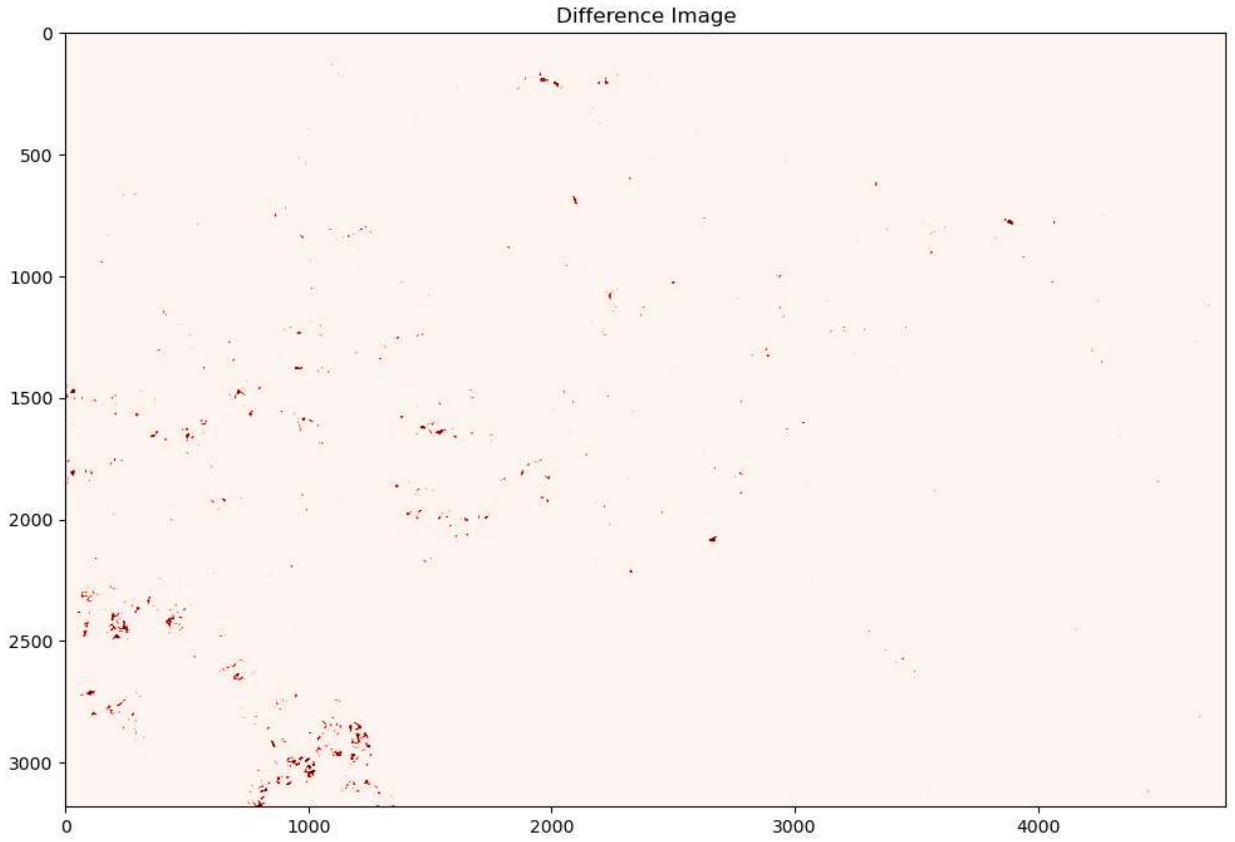
```
ax.axvspan(xmin=0.1, xmax=threshold, alpha=0.25, color="green", label="No Avalanche")
ax.axvspan(xmin=threshold,
           xmax=0.4,
           alpha=0.25,
           color="red",
           label="Avalanche")
plt.legend()
plt.xlabel("Diff (dB)")
plt.title("Effect of the classifier")
plt.show()
```



In [14]:
```
diff_noshadow = diff - np.multiply(diff,shadow_mask)
diff_noshadow[diff_noshadow <= threshold] = 0
```

In [15]:
```
figure, ax = plt.subplots(1,1, figsize=(12,8))
plt.imshow(diff_noshadow, vmin=0.1, vmax=0.28, cmap='Reds')
plt.title("Difference Image")
```

Out[15]:
```
Text(0.5, 1.0, 'Difference Image')
```

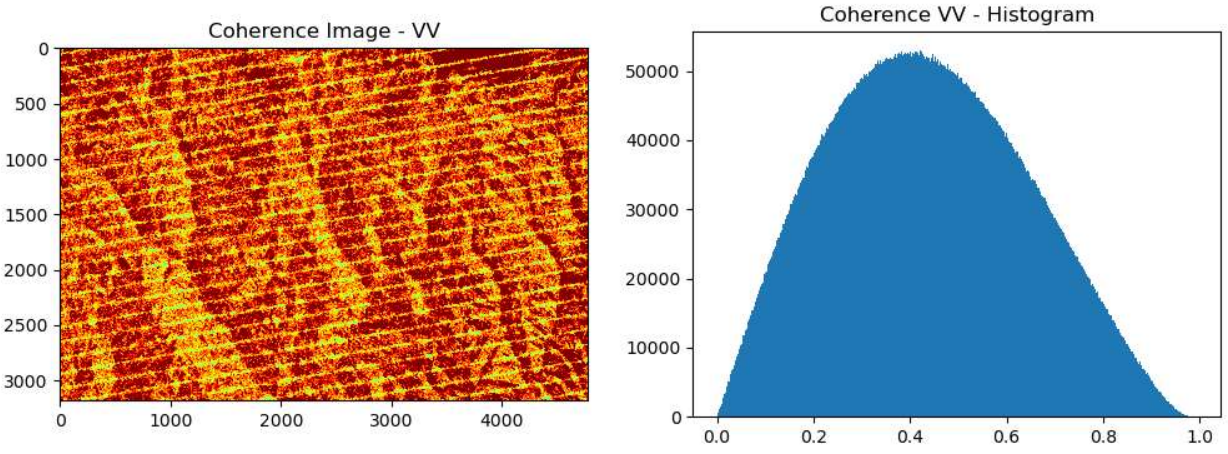


Do same analysis using Coherence between Pre & Post Images

In [16]:
```
#Coherence Map VV

num1_VV = np.multiply(i1_VV,i2_VV) + np.multiply(q1_VV,q2_VV)
num2_VV = np.multiply(q1_VV,i2_VV) - np.multiply(i1_VV,q2_VV)

den1_VV = (i1_VV**2 + q1_VV**2)
den2_VV = (i2_VV**2 + q2_VV**2)

# den_VV = np.sqrt(np.sqrt(np.multiply(intensity1_VV, intensity2_VV)))
```

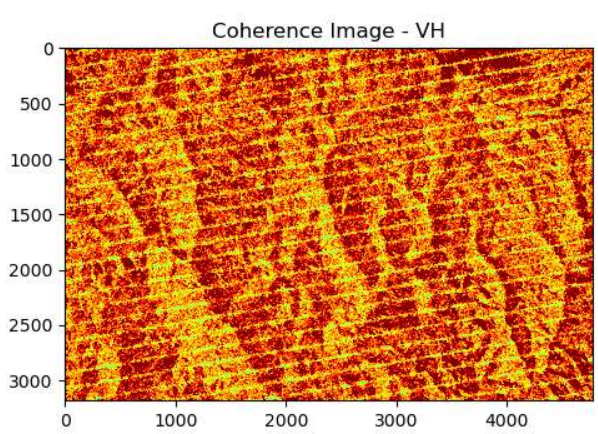
```
num1_VV_con = scipy.ndimage.convolve(num1_VV, np.full((5, 5), 1.0/25))
num2_VV_con = scipy.ndimage.convolve(num2_VV, np.full((5, 5), 1.0/25))
den1_VV_con = scipy.ndimage.convolve(den1_VV, np.full((5, 5), 1.0/25))
den2_VV_con = scipy.ndimage.convolve(den2_VV, np.full((5, 5), 1.0/25))

num_VV_con = np.sqrt((num1_VV_con**2) + (num2_VV_con**2))
den_VV_con = np.sqrt(np.multiply(den1_VV_con, den2_VV_con))

C_VV = np.divide(num_VV_con,den_VV_con)


# PLOT

figure, ax = plt.subplots(1,2, figsize=(12,4))

im1 = ax[0].imshow(C_VV, vmin=0, vmax=0.5, cmap='jet')
ax[0].set_title("Coherence Image - VV")
# figure.colorbar(im1)
ax[1].hist(C_VV.ravel(),  bins=500, density=False)
ax[1].set_title("Coherence VV - Histogram")
```

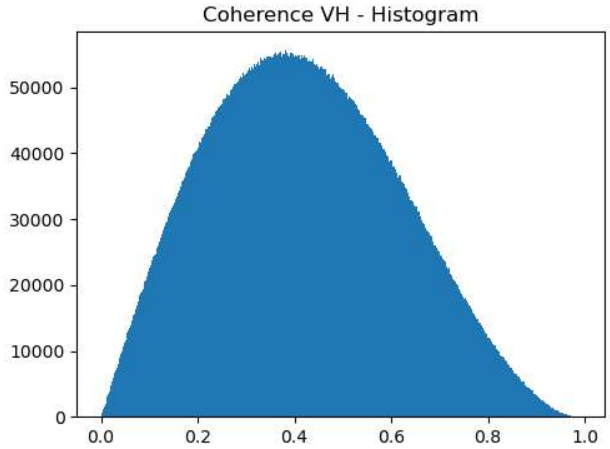
Out[16]:
```
Text(0.5, 1.0, 'Coherence VV - Histogram')
```



In [17]:
```
#Coherence Map VH

num1_VH = np.multiply(i1_VH,i2_VH) + np.multiply(q1_VH,q2_VH)
num2_VH = np.multiply(q1_VH,i2_VH) - np.multiply(i1_VH,q2_VH)

den1_VH = (i1_VH**2 + q1_VH**2)
den2_VH = (i2_VH**2 + q2_VH**2)

# den_VV = np.sqrt(np.sqrt(np.multiply(intensity1_VV, intensity2_VV)))

num1_VH_con = scipy.ndimage.convolve(num1_VH, np.full((5, 5), 1.0/25))
num2_VH_con = scipy.ndimage.convolve(num2_VH, np.full((5, 5), 1.0/25))
den1_VH_con = scipy.ndimage.convolve(den1_VH, np.full((5, 5), 1.0/25))
den2_VH_con = scipy.ndimage.convolve(den2_VH, np.full((5, 5), 1.0/25))

num_VH_con = np.sqrt((num1_VH_con**2) + (num2_VH_con**2))
den_VH_con = np.sqrt(np.multiply(den1_VH_con, den2_VH_con))

C_VH = np.divide(num_VH_con,den_VH_con)


# PLOT

figure, ax = plt.subplots(1,2, figsize=(12,4))

im1 = ax[0].imshow(C_VH, vmin=0, vmax=0.5, cmap='jet')
ax[0].set_title("Coherence Image - VH")
# figure.colorbar(im1)
ax[1].hist(C_VH.ravel(),  bins=500, density=False)
ax[1].set_title("Coherence VH - Histogram")
```

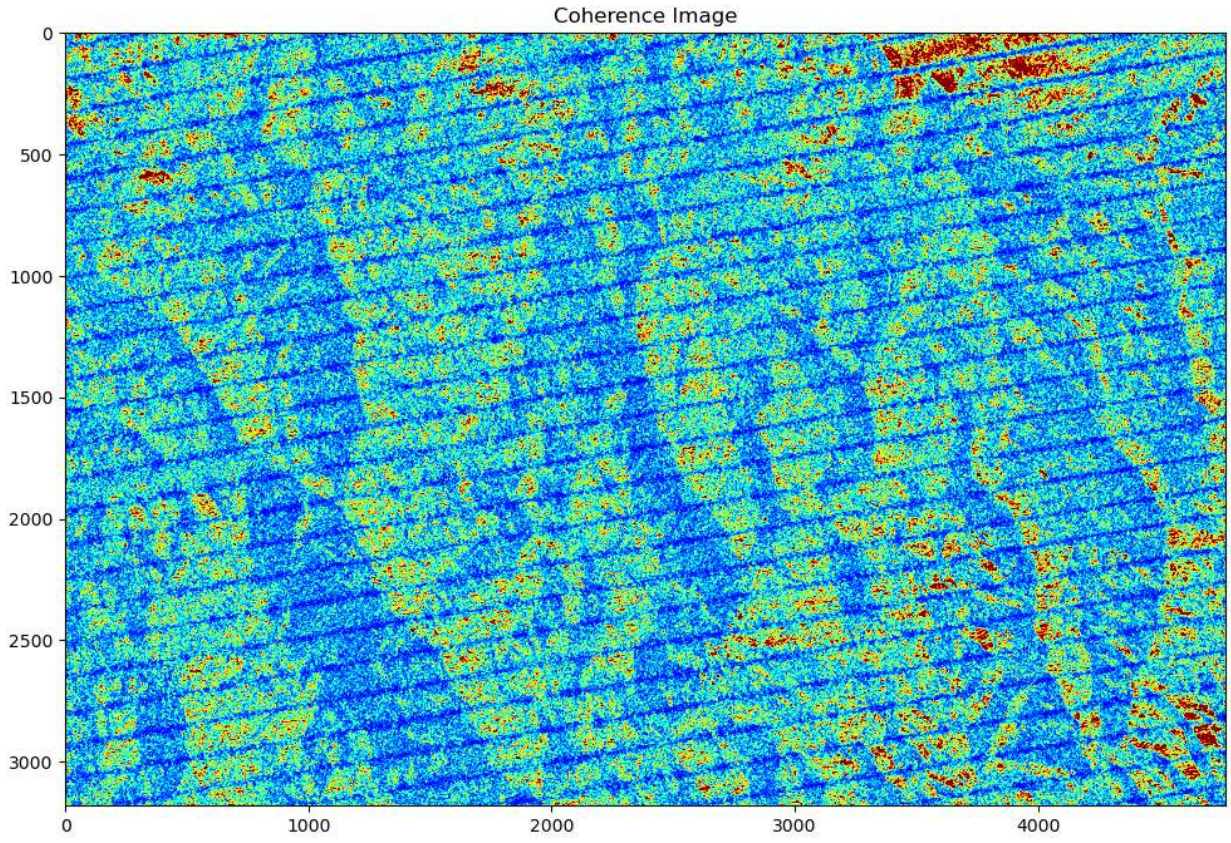
Out[17]:
```
Text(0.5, 1.0, 'Coherence VH - Histogram')
```





In [18]:
```
#diff = diff_VV_norm + diff_VH_norm
Coherence = np.multiply(C_VV, C_VH)

figure, ax = plt.subplots(1,1, figsize=(12,8))
plt.imshow(Coherence, vmin=0, vmax=0.5, cmap='jet')
plt.title("Coherence Image")
```

Out[18]:
```
Text(0.5, 1.0, 'Coherence Image')
```



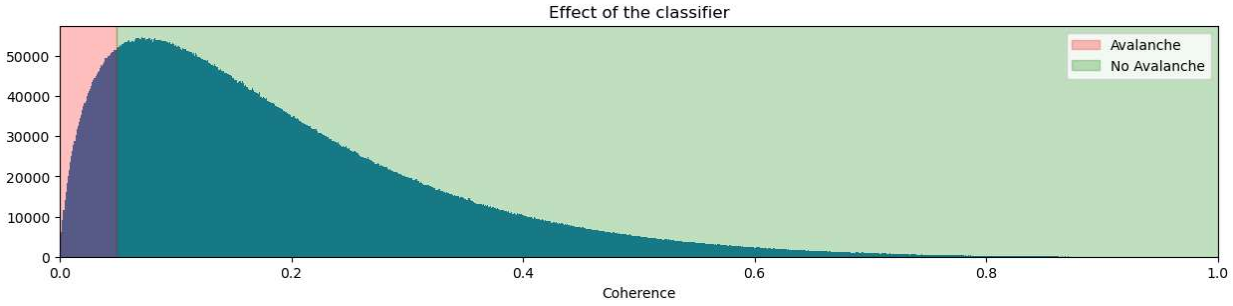
Keep only pixels which have coherence less than (Mean - Std Deviation)

In [19]:
```
Coherence_mean = np.mean(Coherence)
print("Mean: ", Coherence_mean)

Coherence_stdev = np.std(Coherence)
print("std: ", Coherence_stdev)

threshold_C = Coherence_mean - 1*Coherence_stdev
print("threshold: ", threshold_C)
```

```
Mean:  0.18976702
std:  0.14092965
threshold:  0.048837363719940186
```

In [20]:
```
fig, ax = plt.subplots(figsize=(15, 3))
ax.hist(Coherence.ravel(),  bins=1000, density=False)
plt.xlim(0,1)
```

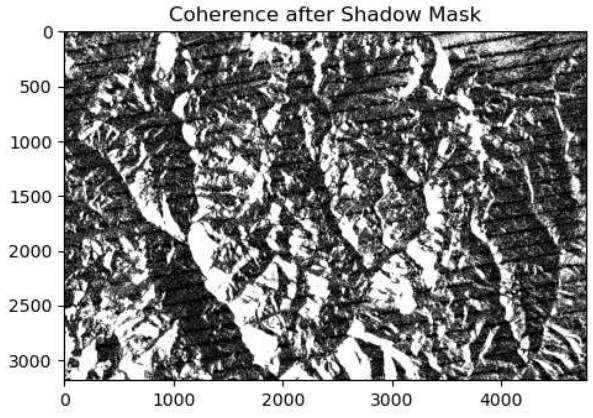
```
ax.axvspan(xmin=0.0, xmax=threshold_C, alpha=0.25, color="red", label="Avalanche")
ax.axvspan(xmin=threshold_C,
           xmax=1,
           alpha=0.25,
           color="green",
           label="No Avalanche")
plt.legend()
plt.xlabel("Coherence")
plt.title("Effect of the classifier")
plt.show()
```

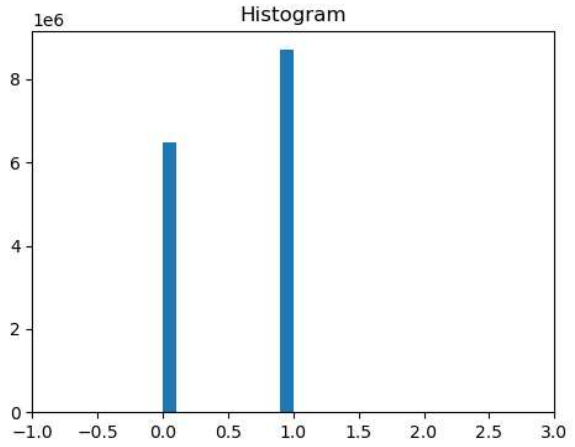


In [21]:
```
# # Mask Shadows
Coherence_noshadow = Coherence - 2 * shadow_mask

Coherence_noshadow[Coherence_noshadow <= 0] = 0
Coherence_noshadow[(Coherence_noshadow > 0) & (Coherence_noshadow < threshold)] = 1
Coherence_noshadow[(Coherence_noshadow >= threshold) & (Coherence_noshadow < 1)] = 0
```

In [22]:
```
figure, ax = plt.subplots(1,2, figsize=(12,4))
ax[0].imshow(Coherence_noshadow, vmin=0, vmax=1, cmap='binary')
ax[0].set_title("Coherence after Shadow Mask")
ax[1].hist(Coherence_noshadow.ravel(),  bins=10)
ax[1].set_title("Histogram")
ax[1].set_xlim(-1,3)
```

Out[22]:
```
(-1.0, 3.0)
```

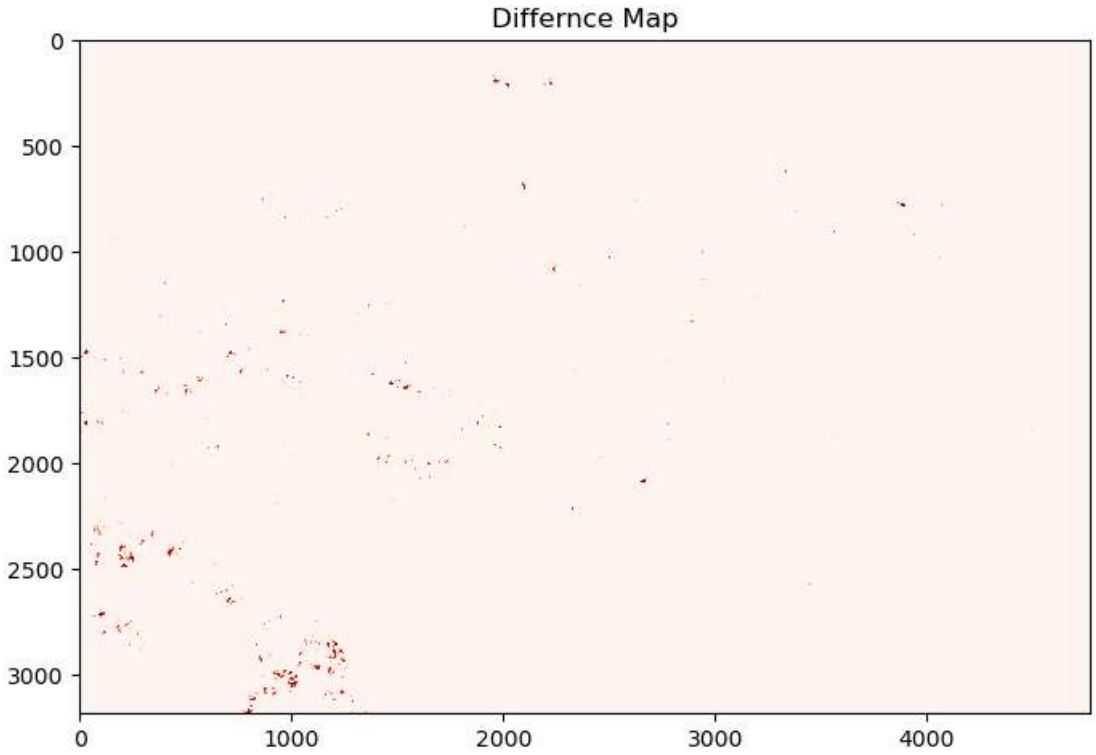


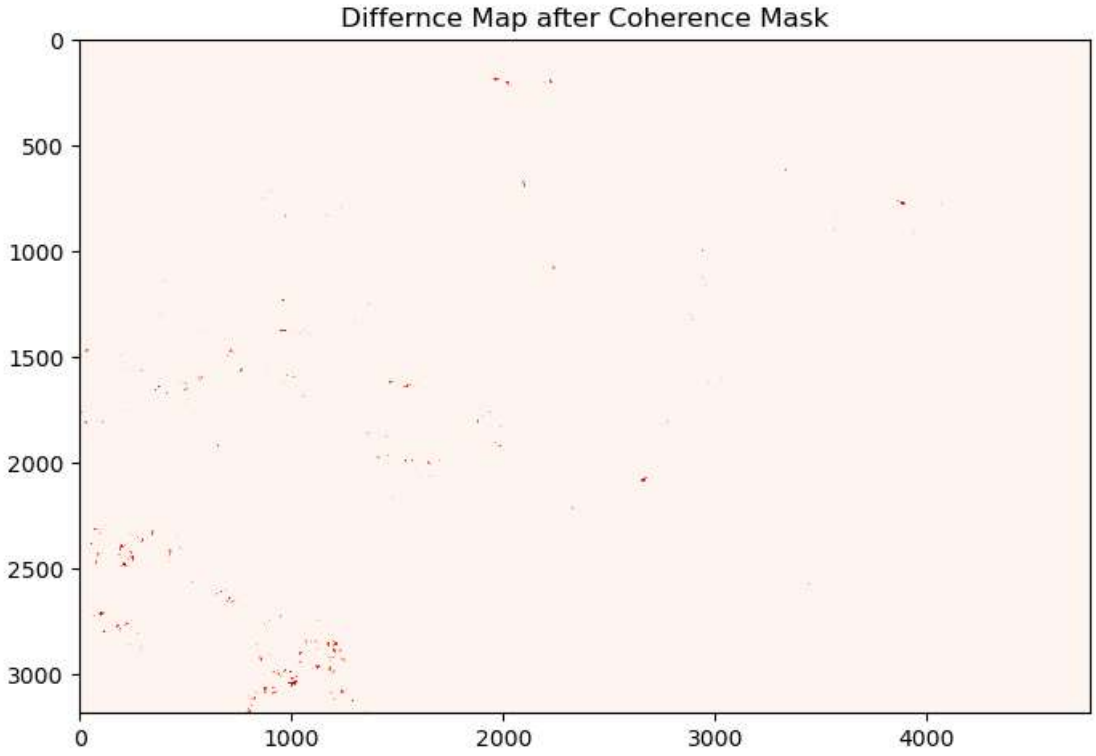


In [23]:
```
diff_final = np.multiply(diff_noshadow,Coherence_noshadow)
```

In [28]:
```
figure, ax = plt.subplots(2,1, figsize=(20,12))
ax[0].imshow(diff_noshadow, vmin=0.1, vmax=0.28, cmap='Reds')
ax[0].set_title("Differnce Map")
ax[1].imshow(diff_final, vmin=0.1, vmax=0.28, cmap='Reds')
ax[1].set_title("Differnce Map after Coherence Mask")
```

Out[28]:
```
Text(0.5, 1.0, 'Differnce Map after Coherence Mask')
```





Display Results on a Map

In [29]:
```
dst_filename = path+'\diff.tiff'
data0 = pre_image
array=diff_final

x_pixels= array.shape[1]  # number of pixels in x
y_pixels = array.shape[0]  # number of pixels in y
x_pixels, y_pixels
```

Out[29]:
```
(4774, 3182)
```

In [30]:
```
driver = gdal.GetDriverByName('GTiff')
dataset = driver.Create(dst_filename,x_pixels, y_pixels, 1,gdal.GDT_Float32)
dataset.GetRasterBand(1).WriteArray(array)
```

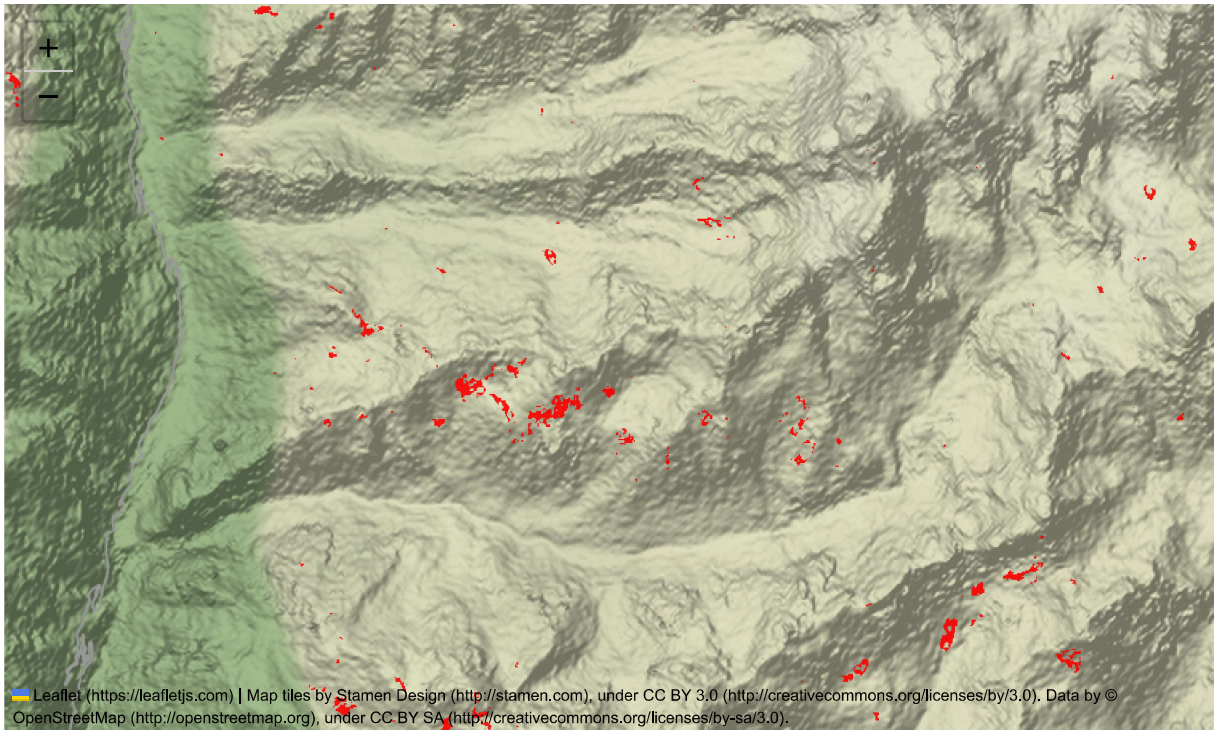
```
# follow code is adding GeoTranform and Projection
geotrans=data0.GetGeoTransform()  #get GeoTranform from existed 'data0'
proj=data0.GetProjection() #you can get from a exsited tif or import
dataset.SetGeoTransform(geotrans)
dataset.SetProjection(proj)
dataset.FlushCache()
dataset=None
```

In [31]:
```
# Create a variable for destination coordinate system
dst_crs = 'EPSG:4326'

# Path to raster
in_path = path+'\diff.tiff'

# Open the raster in rioxarray
img = rxr.open_rasterio(in_path, masked=True)

# Reproject the raster to be the correct crs
img = img.rio.reproject(dst_crs)

# Replace all null values with the minimum value in the array
img_plot = img.where(~img.isnull(), img.min())

# Scale the array from 0 to 255
scaled_img = es.bytescale(img_plot.values[0])
```

In [32]:
```
import branca.colormap as cm
from matplotlib import colors as colors
```

In [33]:
```
# Create a map using Stamen Terrain, centered on study area with set zoom level
m = folium.Map(location=[27.72, 88.72],
               tiles = 'Stamen Terrain', default_zoom_start=18)


map_bounds = [[27.60, 88.40],
              [28.00, 89.00]]

vmin = np.floor(np.nanmin(scaled_img))
vmax = np.ceil(np.nanmax(scaled_img))

colormap = cm.linear.RdBu_11.scale(vmin, vmax)

def mapvalue2color(value, cmap):
    """
    Map a pixel value of image to a color in the rgba format.
    As a special case, nans will be mapped totally transparent.

    Inputs
        -- value - pixel value of image, could be np.nan
        -- cmap - a linear colormap from branca.colormap.linear
    Output
        -- a color value in the rgba format (r, g, b, a)
    """
    if np.equal(0, value):
        return (1, 0, 0, 0)
    else:
        return (1,0,0,0.7)


# Overlay raster called img using add_child() function (opacity and bounding box set)
m.add_child(folium.raster_layers.ImageOverlay(scaled_img,
                                              opacity=.9,
                                              colormap=lambda value: mapvalue2color(value, colormap),
                                              bounds=map_bounds))

# Display map
m
```

Out[33]:


Leaflet (https://leafletjs.com) | Map tiles by Stamen Design (http://stamen.com), under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0). Data by © OpenStreetMap (http://openstreetmap.org), under CC BY SA (http://creativecommons.org/licenses/by-sa/3.0).