EdgeDetection_15February

February 15, 2020

1 Edge Detection in Satellite Imagery

1.1 NOTE : Edit the path in the file for reading and writing image. Do not edit last path of Path as it is part of the loop.

1.2 Abstract

- Agricultural land-use statistics are more informative per-field than per-pixel.
- Land-use classification requires up-to-date field boundary maps potentially covering large areas containing thousands of farms.
- Remote sensing imagery can provide detailed, up-to-date, and spatially explicit information on agricultural land use that would otherwise be difficult to gather.
- A need for combining the accuracy of the remotely sensed imagery available today along with image processing techniques to obtain useful land-use statistics.

1.3 Scope of the Project

Remote sensing is the science of obtaining information about objects or areas from a distance, typically from aircraft or satellites. **Computer science** is the science which studies processes that interact with data and that can be represented as data in the form of programs. It enables the use of algorithms to manipulate, store, and communicate digital information.

1.4 Importing the necessary libraries

1.4.1 Language Used

- Python 3.6.8

1.4.2 Libraries Used

- **OpenCV**
  - Consists of a compact module defining basic data structures and an image processing module that includes linear and non-linear image filtering, geometrical image transformations, color space conversion, histograms, and so on.
- **Numpy** – Fundamental package which contains :
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions

    - tools for integrating C/C++ and Fortran code
    - useful linear algebra, Fourier transform, and random number capabilities

- **PIL**
    - Contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and colour space conversions.
    - Histogram method allows to pull some statistics out of an image, which is used for automatic contrast enhancement, and for global statistical analysis.

- **Matplotlib**
    - 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

- **google.colab.patches**
    - Handling advanced ouput on Google Colab

```
In [0]: import os
        from PIL import Image
        from skimage.color import rgb2gray
        import numpy as np
        import cv2
        import matplotlib.pyplot as plt
        %matplotlib inline
        from scipy import ndimage
        from google.colab.patches import cv2_imshow
```

```
In [2]: from google.colab import drive
        drive.mount('/content/drive',force_remount=True)
        #Variable used for total number of images
        q=1
        z=3
```

```
Mounted at /content/drive
```

1.5 Data/Images Used

The data obtained for testing the implementation of the technique developed have been obtained from Google Earth and the images so obtained have been taken from the WorldView satellite available. The images include a variety of sites, with each showing a different color visually, thus helping us to test the robustness of the algorithm.

Images have been obtained using Google Earth and readily available satellite images. The images belong to different areas namely:

```
- Image 1 : Japan, Asia
- Image 2 : Mexico, North America
- Image 3 : United States of America, North America
```

Image_1
Image_2
Image_3

1.6 Classical Approach

1.6.1 Canny Edge Detection

- Import image
- Perform noise removal using smoothening methods
- Gaussian blurred performed with kernel size of 5 x 5
- Gaussian blurred performed with kernel size of 3 x 3
- Gaussian kernel standard deviation is kept 0 in both steps. Python function automatically calculates the standard deviation x and y .
- A maximum and minimum threshold must be used for Canny Edge Detection. The threshold is calculated using the mean and standard deviation of the image from the previous step.
- Henceforth, the canny edge detection is performed which outputs a binary image with edge boundary (white) and non-edge boundary (black).

```
In [0]: for j in range(q,z):
            i=str(j)
            #PLEASE SET PATH FOR THE TEST IMAGE AS PER SYSTEM .
            #NOTE : PLEASE KEEP IMAGE FORMAT AS 'Final_Image_(Integer).jpg'
            if(j!=2):
              imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpg'
            else:
              imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpeg'
            oimg=cv2.imread(imagebeing_read)
            img=cv2.GaussianBlur(oimg,(5,5),0)


            #THIS IS PATH FOR WRITING THE IMAGE
            # PLEASE EDIT ONLY THE PART BEFORE
            first_GaussianBlur = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Gaussia
            cv2.imwrite(first_GaussianBlur,img)
            nimg=cv2.GaussianBlur(img,(3,3),0)
            second_GaussianBlur = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Gauss
            cv2.imwrite(second_GaussianBlur,nimg)


            v = np.mean(nimg)
            sigma = np.std(nimg)
            lowThresh = int(max(0, (1.0 - sigma) * 2*v))
            high_Thresh = int(min(255, (1.0 + sigma) * 2*v))
            edges_Canny = cv2.Canny(nimg,lowThresh,high_Thresh)

            #Path used for writing the Canny Image Detected.
            WriteImage = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Canny.jpg'
            cv2.imwrite(WriteImage,edges_Canny)
```

1.6.2 Laplacian Edge Detection

- Import image
- Perform noise removal using smoothening methods
- Gaussian blurred performed with kernel size 5 x 5
- Laplacian edge detection is performed which outputs a greyscale image with detected edges.
- Global thresholding performed to obtain a binary image from the grayscale image for displaying boundary pixels and non-boundary pixels.

```
In [0]: for j in range(q,z):
          i=str(j)
          if(j!=2):
            #The image is again being read here. Again set path according to system
            imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpg'
          else:
            imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpeg'
          oimg=cv2.imread(imagebeing_read)
          img=cv2.GaussianBlur(oimg,(5,5),0)
          edges_Laplacian=cv2.Laplacian(img,cv2.CV_64F)
          #Laplacian Image is written here
          WriteImage_Laplacian = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Lapla
          cv2.imwrite(WriteImage_Laplacian,edges_Laplacian)
          ReadImage_Laplacian=WriteImage_Laplacian
          edges_Laplacian=cv2.imread(ReadImage_Laplacian)
          img= cv2.cvtColor(edges_Laplacian, cv2.COLOR_BGR2GRAY)
          mean_value=np.mean(img)
          #Linear Contrast Enhancement
          mean_value=mean_value*3.1
          #Thresholding performed to form definite boundaries
          ret,th1 = cv2.threshold(img,mean_value,255,cv2.THRESH_BINARY)
          th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,
          th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY
          #Creating Array for performing different type of thresholding
          titles = ['Original Image', 'Global Thresholding (v = 127)',
                      'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
          images = [img, th1, th2, th3]

          th1=cv2.medianBlur(th1,3)
          WriteImage = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Laplacian_Gaus
          cv2.imwrite(WriteImage,th1)
```

1.6.3 Sobel Edge Detection

- Import image
- Perform transformation from RGB color triad to YUV color triad.
- On the transformed image, perform linear contrast enhancement.
- Y = aX + b ; X is the input pixel value and a and b are constants chosen for image enhancement and Y is the output pixel value.
- Perform noise removal using smoothening methods

- Gaussian blurred performed with kernel size of 5 x 5
- Sobel filtering is performed on the enhanced image obtained from the previous step.
- The edge detected image is then smoothened again using Gaussian Blur of 3x3.
- The image obtained after smoothening is a grayscale image which is finally binarized/thresholded to produce a binary image with edge boundary (white) and non-edge boundary (black).

YUV Transform and Contrast Enhancement

```
In [0]: for j in range(q,z):
          i=str(j)
          if(j!=2):
            imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpg'
          else:
            imagebeing_read = 'drive/My Drive/Images/Final_Image_'+i+'.jpeg'
          img=cv2.imread(imagebeing_read)
          #YUV TRANSFORM OF IMAGES BEING PERFORMED
          yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
          #YUV IMAGE BEING WRITTEN - Please Edit PATH ACCORDINGLY
          YUV_Write = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'YUV.jpg'
          cv2.imwrite(YUV_Write,yuv)
          img = cv2.imread(YUV_Write)
          image=img
          #CONTRAST ENHANCEMENT
          new_image = np.zeros(image.shape, image.dtype)
          alpha=0.7
          beta=0
          for y in range(image.shape[0]):
              for x in range(image.shape[1]):
                for c in range(image.shape[2]):
                      new_image[y,x,c] = np.clip(alpha*image[y,x,c] + beta, 0, 255)
          WriteImage_YUV = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_yuv_Linear
          cv2.imwrite(WriteImage_YUV,new_image)
```

Applying the Sobel Filter on the YUV Transformed Image

```
In [0]: for j in range(q,z):
          i=str(j)
          imageRead = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_yuv_LinearTrans
          img = cv2.imread(imageRead,cv2.IMREAD_GRAYSCALE)
          img = cv2.GaussianBlur(img,(3,3),0)
          h = cv2.Sobel(img, cv2.CV_64F, 0, 1)
          v = cv2.Sobel(img, cv2.CV_64F, 1, 0)
          edges_Sobel = cv2.add(h, v)
          imageWrite = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel.jpg'
          cv2.imwrite(imageWrite,edges_Sobel)
```

```
In [0]: for j in range(q,z):
          i=str(j)
```

```
from PIL import Image
import math
path = "drive/My Drive/Edge_Detection_End/Image"+i+"/Image"+i+"YUV.jpg" # Your image
im = cv2.imread(path)
width=im.shape[1]
height=im.shape[0]
img = Image.open(path)
newimg = Image.new("RGB", (width, height), "white")
for x in range(1, width-1):  # ignore the edge pixels for simplicity (1 to width-1)
    for y in range(1, height-1): # ignore edge pixels for simplicity (1 to height-1)

        # initialise Gx to 0 and Gy to 0 for every pixel
        Gx = 0
        Gy = 0

        # top left pixel
        p = img.getpixel((x-1, y-1))
        r = p[0]
        g = p[1]
        b = p[2]

        # intensity ranges from 0 to 765 (255 * 3)
        intensity = r + g + b

        # accumulate the value into Gx, and Gy
        Gx += -intensity
        Gy += -intensity

        # remaining left column
        p = img.getpixel((x-1, y))
        r = p[0]
        g = p[1]
        b = p[2]

        Gx += -2 * (r + g + b)

        p = img.getpixel((x-1, y+1))
        r = p[0]
        g = p[1]
        b = p[2]

        Gx += -(r + g + b)
        Gy += (r + g + b)

        # middle pixels
        p = img.getpixel((x, y-1))
        r = p[0]
        g = p[1]
        b = p[2]

        Gy += -2 * (r + g + b)

        p = img.getpixel((x, y+1))
        r = p[0]
        g = p[1]
        b = p[2]

        Gy += 2 * (r + g + b)

        # right column
        p = img.getpixel((x+1, y-1))
        r = p[0]
        g = p[1]
        b = p[2]

        Gx += (r + g + b)
        Gy += -(r + g + b)

        p = img.getpixel((x+1, y))
        r = p[0]
        g = p[1]
        b = p[2]

        Gx += 2 * (r + g + b)

        p = img.getpixel((x+1, y+1))
        r = p[0]
        g = p[1]
        b = p[2]

        Gx += (r + g + b)
        Gy += (r + g + b)

        # calculate the length of the gradient (Pythagorean theorem)
        length = math.sqrt((Gx * Gx) + (Gy * Gy))

        # normalise the length of gradient to the range 0 to 255
        length = length / 4328 * 255

        length = int(length)

        # draw the length in the edge image
        #newpixel = img.putpixel((length,length,length))
        newimg.putpixel((x,y),(length,length,length))
writePath = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual.jp
newimg.save(writePath)
```

Thresholding the Sobel Filtered Image

```
In [0]: for j in range(q,z):
            i=str(j)
            imageRead = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual.jpg
            image=cv2.imread(imageRead)
            Gaussian = cv2.GaussianBlur(image,(5,5),0)
            Gaussian = cv2.GaussianBlur(Gaussian,(3,3),0)
            median=cv2.medianBlur(image,5)
            median=cv2.medianBlur(median,3)
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Med
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Gau
            img= cv2.cvtColor(median, cv2.COLOR_BGR2GRAY)
            mean_value=np.mean(img)
            mean_value=mean_value*3.1
            ret,th1 = cv2.threshold(img,mean_value,255,cv2.THRESH_BINARY)
            th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,
            th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY

            titles = ['Original Image', 'Global Thresholding (v = 127)',
                        'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
            images = [img, th1, th2, th3]

            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Med
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Med
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Med
            img= cv2.cvtColor(Gaussian, cv2.COLOR_BGR2GRAY)
            mean_value=np.mean(img)
            mean_value=mean_value*3.1
            ret,th1 = cv2.threshold(img,mean_value,255,cv2.THRESH_BINARY)
            th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,
            th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY

            titles = ['Original Image', 'Global Thresholding (v = 127)',
                        'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
            images = [img, th1, th2, th3]

            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Gau
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Gau
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_Manual_Gau

            imageRead = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel.jpg'
            image=cv2.imread(imageRead)
            Gaussian = cv2.GaussianBlur(image,(5,5),0)
            Gaussian = cv2.GaussianBlur(Gaussian,(3,3),0)
            median=cv2.medianBlur(image,5)
            median=cv2.medianBlur(median,3)
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_MedianFil
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_GaussianFi
            img= cv2.cvtColor(median, cv2.COLOR_BGR2GRAY)
            mean_value=np.mean(img)
            mean_value=mean_value*3.1
            ret,th1 = cv2.threshold(img,mean_value,255,cv2.THRESH_BINARY)
            th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,
            th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY

            titles = ['Original Image', 'Global Thresholding (v = 127)',
                        'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
            images = [img, th1, th2, th3]

            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_MedianFil
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_MedianFil
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_MedianFil
            img= cv2.cvtColor(Gaussian, cv2.COLOR_BGR2GRAY)
            mean_value=np.mean(img)
            mean_value=mean_value*3.1
            ret,th1 = cv2.threshold(img,mean_value,255,cv2.THRESH_BINARY)
            th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,
            th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY

            titles = ['Original Image', 'Global Thresholding (v = 127)',
                        'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
            images = [img, th1, th2, th3]

            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_GaussianFi
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_GaussianFi
            cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel_GaussianFi
```

1.7 Ensemble the Results

- The ensemble method begins with taking the best output from the previous images processed from the three independent edge detection techniques.
- For every pixel in the image, we deploy a count-check algorithm, to determine if the pixel is an edge according to the different edge detection techniques used earlier.
- If a pixel with coordinates (x,y) in the image is detected as an edge pixel in all the three methods, we can know with maximum confidence that this pixel is an edge pixel. Thus we mark this pixel to be an edge pixel in the ensembled output
- If a pixel with coordinates (x,y) in the image is detected as an edge pixel in two of the three methods, we check the neighbouring pixels to determine if the pixel under consideration should be marked as an edge pixel or not. The check, centered at (x,y), marks the pixel to be an edge if any of the following conditions, for the algorithm in which (x,y) was not an edge pixel, may be true :
  - (x-1,y) and (x+1,y) are edge pixels
  - (x,y-1) and (x,y+1) are edge pixels
  - (x+1,y-1) and (x-1,y+1) are edge pixels

  - (x-1,y-1) and (x+1,y+1) are edge pixels If any of the above condition holds true, the pixel is marked as a edge pixel in the ensembled output

- If the pixel is not marked as an edge pixel in atleast two of the three methods, we consider this to be a noise in terms of the edge detection and thus mark it as a non-edge pixel or leave the pixel values as the original, depending upon whether we need a binarized image or not.
- Lastly, the ensembled output obtained is sharpened to produce crisp edges in the final image obtained.

```
In [0]: for j in range (q,z):
          i=str(j)
          img_Canny = cv2.imread('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Canny
          img_Laplacian = cv2.imread('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_
          img_Sobel = cv2.imread('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Sobel
          if j!=2:
            img_image = cv2.imread('drive/My Drive/Images/Final_Image_'+i+'.jpg')
          else:
            img_image = cv2.imread('drive/My Drive/Images/Final_Image_'+i+'.jpeg')
          width=img_image.shape[1]
          height=img_image.shape[0]
          for y in range(1,width-1):
            for x in range(1,height-1):
              ## Checking if all three images have pixel values as edge
              count=0
              count_Canny=False
              count_Laplacian=False
              count_Sobel=False
              if(img_Canny[x][y]>200):
                count=count+1
                count_Canny=True
              if(img_Laplacian[x][y]>200):
                count=count+1
                count_Laplacian=True
              if(img_Sobel[x][y]>200):
                count=count+1
                count_Sobel=True
              if count==3 :
                img_image[x][y][0]=255
                img_image[x][y][1]=255
                img_image[x][y][2]=255
              #Checking if the neighbouring pixels are marked edge pixels
              elif count==2:
                if(img_Canny[x-1][y]==255 and img_Canny[x+1][y]==255) or (img_Canny[x][y+1]==25
                    img_image[x][y][0]=255
                    img_image[x][y][1]=255
                    img_image[x][y][2]=255
                elif(count_Laplacian==False):
                  if(img_Laplacian[x-1][y]==255 and img_Laplacian[x+1][y]==255) or (img_Laplaci
                    img_image[x][y][0]=255
                    img_image[x][y][1]=255
                    img_image[x][y][2]=255
                elif(count_Sobel==False):
                  if(img_Sobel[x-1][y]==255 and img_Sobel[x+1][y]==255) or (img_Sobel[x][y+1]==
                    img_image[x][y][0]=255
                    img_image[x][y][1]=255
                    img_image[x][y][2]=255
                else:
                  img_image[x][y][0]=0
                  img_image[x][y][1]=0
                  img_image[x][y][2]=0
              else:
                img_image[x][y][0]=0
                img_image[x][y][1]=0
                img_image[x][y][2]=0

          writePath = 'drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_EnsembledImage.
          cv2.imwrite(writePath,img_image)
```

```
In [0]: for j in range(q,z):
          i=str(j)
          image = cv2.imread('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_Ensemble
          kernel_sharpening = np.array([[-1,-1,-1],
                                        [-1, 9,-1],
                                        [-1,-1,-1]])
          sharpened = cv2.filter2D(image, -1, kernel_sharpening)
          ima=cv2.GaussianBlur(sharpened,(3,3),0)
          cv2.imwrite('drive/My Drive/Edge_Detection_End/Image'+i+'/Image'+i+'_EnsembledImage_
```