```
import numpy as np
import cv2
import scipy.io
import os
from numpy.linalg import norm
from matplotlib import pyplot as plt
from numpy.linalg import det
from numpy.linalg import inv
from scipy.linalg import rq
from numpy.linalg import svd
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import sys
from scipy import ndimage, spatial
from tqdm.notebook import tqdm, trange

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader, ConcatDataset
from skimage import io, transform,data
from torchvision import transforms, utils
import numpy as np
import math
import glob
import matplotlib.pyplot as plt
import time
import os
import copy
import sklearn.svm
import cv2
from matplotlib import pyplot as plt
import numpy as np
from os.path import exists
import pandas as pd
import PIL
import random
from google.colab import drive
from sklearn.metrics.cluster import completeness_score
from sklearn.cluster import KMeans
from tqdm import tqdm, tqdm_notebook
from functools import partial
from torchsummary import summary
from torchvision.datasets import ImageFolder
from torch.utils.data.sampler import SubsetRandomSampler

#cuda_output = !ldconfig -p|grep cudart.so|sed -e 's/.*\.\([0-9]*\)\.\([0-9]*\)$/cu\1\2/'
#accelerator = cuda_output[0] if exists('/dev/nvidia0') else 'cpu'

#print("Accelerator type = ",accelerator)
#print("Pytorch verision: ", torch.__version__)
```

Importing Drive (Dataset-University)

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

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

```
plt.figure(figsize=(20,10))
```

```
<Figure size 1440x720 with 0 Axes>
<Figure size 1440x720 with 0 Axes>
```

```


class Image:
    def __init__(self, img, position):

        self.img = img
        self.position = position

inlier_matchset = []
def features_matching(a,keypointlength,threshold):
  #threshold=0.2
  bestmatch=np.empty((keypointlength),dtype= np.int16)
  img1index=np.empty((keypointlength),dtype=np.int16)
  distance=np.empty((keypointlength))
  index=0
  for j in range(0,keypointlength):
    #For a descriptor fa in Ia, take the two closest descriptors fb1 and fb2 in Ib
    x=a[j]
    listx=x.tolist()
    x.sort()
    minval1=x[0]                                # min
    minval2=x[1]                                # 2nd min
    itemindex1 = listx.index(minval1)           #index of min val
    itemindex2 = listx.index(minval2)           #index of second min value
    ratio=minval1/minval2                       #Ratio Test

    if ratio<threshold:
      #Low distance ratio: fb1 can be a good match
      bestmatch[index]=itemindex1
      distance[index]=minval1
      img1index[index]=j
      index=index+1
  return  [cv2.DMatch(img1index[i],bestmatch[i].astype(int),distance[i]) for i in range(0,index)]



def compute_Homography(im1_pts,im2_pts):
  """
  im1_pts and im2_pts are 2×n matrices with
  4 point correspondences from the two images
  """
  num_matches=len(im1_pts)
  num_rows = 2 * num_matches
  num_cols = 9
  A_matrix_shape = (num_rows,num_cols)
  A = np.zeros(A_matrix_shape)
  a_index = 0
  for i in range(0,num_matches):
    (a_x, a_y) = im1_pts[i]
    (b_x, b_y) = im2_pts[i]
    row1 = [a_x, a_y, 1, 0, 0, 0, -b_x*a_x, -b_x*a_y, -b_x] # First row
    row2 = [0, 0, 0, a_x, a_y, 1, -b_y*a_x, -b_y*a_y, -b_y] # Second row

    # place the rows in the matrix
    A[a_index] = row1
    A[a_index+1] = row2

    a_index += 2

  U, s, Vt = np.linalg.svd(A)
  #s is a 1-D array of singular values sorted in descending order
  #U, Vt are unitary matrices
  #Rows of Vt are the eigenvectors of A^TA.
  #Columns of U are the eigenvectors of AA^T.
  H = np.eye(3)
  H = Vt[-1].reshape(3,3) # take the last row of the Vt matrix
  return H


def displayplot(img,title):

  plt.figure(figsize=(15,15))
  plt.title(title)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.show()
```

```
def RANSAC_alg(f1, f2, matches, nRANSAC, RANSACthresh):


    minMatches = 4
    nBest = 0
    best_inliers = []
    H_estimate = np.eye(3,3)
    global inlier_matchset
    inlier_matchset=[]
    for iteration in range(nRANSAC):

        #Choose a minimal set of feature matches.
        matchSample = random.sample(matches, minMatches)

        #Estimate the Homography implied by these matches
        im1_pts=np.empty((minMatches,2))
        im2_pts=np.empty((minMatches,2))
        for i in range(0,minMatches):
          m = matchSample[i]
          im1_pts[i] = f1[m.queryIdx].pt
          im2_pts[i] = f2[m.trainIdx].pt
          #im1_pts[i] = f1[m[0]].pt
          #im2_pts[i] = f2[m[1]].pt

        H_estimate=compute_Homography(im1_pts,im2_pts)


        # Calculate the inliers for the H
        inliers = get_inliers(f1, f2, matches, H_estimate, RANSACthresh)

        # if the number of inliers is higher than previous iterations, update the best estimates
        if len(inliers) > nBest:
            nBest= len(inliers)
            best_inliers = inliers

    print("Number of best inliers",len(best_inliers))
    for i in range(len(best_inliers)):
      inlier_matchset.append(matches[best_inliers[i]])

    # compute a homography given this set of matches
    im1_pts=np.empty((len(best_inliers),2))
    im2_pts=np.empty((len(best_inliers),2))
    for i in range(0,len(best_inliers)):
      m = inlier_matchset[i]
      im1_pts[i] = f1[m.queryIdx].pt
      im2_pts[i] = f2[m.trainIdx].pt
      #im1_pts[i] = f1[m[0]].pt
      #im2_pts[i] = f2[m[1]].pt

    M=compute_Homography(im1_pts,im2_pts)
    M=compute_Homography(im1_pts,im2_pts)
    return M
```

```
def get_inliers(f1, f2, matches, H, RANSACthresh):

  inlier_indices = []
  for i in range(len(matches)):
    queryInd = matches[i].queryIdx
    trainInd = matches[i].trainIdx

    #queryInd = matches[i][0]
    #trainInd = matches[i][1]

    queryPoint = np.array([f1[queryInd].pt[0],  f1[queryInd].pt[1], 1]).T
    trans_query = H.dot(queryPoint)


    comp1 = [trans_query[0]/trans_query[2], trans_query[1]/trans_query[2]] # normalize with respect to z
    comp2 = np.array(f2[trainInd].pt)[:2]


    if(np.linalg.norm(comp1-comp2) <= RANSACthresh): # check against threshold
      inlier_indices.append(i)
  return inlier_indices


def ImageBounds(img, H):

    h, w= img.shape[0], img.shape[1]
    p1 = np.dot(H, np.array([0, 0, 1]))
    p2 = np.dot(H, np.array([0, h - 1, 1]))
    p3 = np.dot(H, np.array([w - 1, 0, 1]))
    p4 = np.dot(H, np.array([w - 1, h - 1, 1]))
    x1 = p1[0] / p1[2]
    y1 = p1[1] / p1[2]
    x2 = p2[0] / p2[2]
    y2 = p2[1] / p2[2]
    x3 = p3[0] / p3[2]
    y3 = p3[1] / p3[2]
    x4 = p4[0] / p4[2]
    y4 = p4[1] / p4[2]
    minX = math.ceil(min(x1, x2, x3, x4))
    minY = math.ceil(min(y1, y2, y3, y4))
    maxX = math.ceil(max(x1, x2, x3, x4))
    maxY = math.ceil(max(y1, y2, y3, y4))

    return int(minX), int(minY), int(maxX), int(maxY)


def Populate_Images(img, accumulator, H, bw):



    h, w = img.shape[0], img.shape[1]
    minX, minY, maxX, maxY = ImageBounds(img, H)

    for i in range(minX, maxX + 1):
        for j in range(minY, maxY + 1):
            p = np.dot(np.linalg.inv(H), np.array([i, j, 1]))

            x = p[0]
            y = p[1]
            z = p[2]

            _x = int(x / z)
            _y = int(y / z)
            _y = int(y / z)

            if _x < 0 or _x >= w - 1 or _y < 0 or _y >= h - 1:
                continue

            if img[_y, _x, 0] == 0 and img[_y, _x, 1] == 0 and img[_y, _x, 2] == 0:
                continue

            wt = 1.0

            if _x >= minX and _x < minX + bw:
                wt = float(_x - minX) /bw
            if _x <= maxX and _x > maxX -bw:
                wt = float(maxX - _x) /bw

            accumulator[j, i, 3] += wt

            for c in range(3):
                accumulator[j, i, c] += img[_y, _x, c] *wt
```

```
def Image_Stitch(Imagesall, blendWidth, accWidth, accHeight, translation):
    channels=3
    #width=720

    acc = np.zeros((accHeight, accWidth, channels + 1))
    M = np.identity(3)
    for count, i in enumerate(Imagesall):
        M = i.position
        img = i.img
        M_trans = translation.dot(M)
        Populate_Images(img, acc, M_trans, blendWidth)

    height, width = acc.shape[0], acc.shape[1]

    img = np.zeros((height, width, 3))
    for i in range(height):
        for j in range(width):
            weights = acc[i, j, 3]
            if weights > 0:
                for c in range(3):
                    img[i, j, c] = int(acc[i, j, c] / weights)


    Imagefull = np.uint8(img)
    M = np.identity(3)
    for count, i in enumerate(Imagesall):
        if count != 0 and count != (len(Imagesall) - 1):
            continue

        M = i.position

        M_trans = translation.dot(M)

        p = np.array([0.5 * width, 0, 1])
        p = M_trans.dot(p)


        if count == 0:
            x_init, y_init = p[:2] / p[2]

        if count == (len(Imagesall) - 1):
            x_final, y_final = p[:2] / p[2]


    A = np.identity(3)
    croppedImage = cv2.warpPerspective(
        Imagefull, A, (accWidth, accHeight), flags=cv2.INTER_LINEAR
    )
    displayplot(croppedImage, 'Final Stitched Image')
```

```
#!pip uninstall opencv-python
#!pip install opencv-contrib-python===4.4.0.44
#!pip install opencv-python==4.4.0.44
#!pip install opencv-contrib-python==4.4.0.44
```

```
import cv2
print(cv2.__version__)
```

```
4.1.2
```

Reading all Files from Folder

```
files_all=[]
for file in os.listdir("/content/drive/My Drive/Uni_img"):
    if file.endswith(".JPG"):
      files_all.append(file)


files_all.sort()
folder_path = '/content/drive/My Drive/Uni_img/'

all_files_path = []

for file1 in tqdm(files_all):
  all_files_path.append(folder_path+file1)

'''
centre_file = folder_path + files_all[50]
left_files_path_rev = []
right_files_path = []

for file in files_all[:51]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[50:101]:
  right_files_path.append(folder_path + file)
'''
```

```
100%|██████████| 443/443 [00:00<00:00, 933706.87it/s]
'\ncentre_file = folder_path + files_all[50]\nleft_files_path_rev = []\nright_files_path = []\n\nfor file in files_all[:51]:\n  left_files_path_rev.append(folder_path + file)\n\nleft_files_path = left_files_path_rev[::-1]\n\nfor file in files_all[50:101]:\n  right_files_path.append(folder_path + file)\n'
```

Reading GPS and Metdata information

```
from PIL import Image, ExifTags
img = Image.open(f"{all_files_path[0]}")
exif = { ExifTags.TAGS[k]: v for k, v in img._getexif().items() if k in ExifTags.TAGS }
```

```
from PIL.ExifTags import TAGS

def get_exif(filename):
    image = Image.open(filename)
    image.verify()
    return image._getexif()

def get_labeled_exif(exif):
    labeled = {}
    for (key, val) in exif.items():
        labeled[TAGS.get(key)] = val

    return labeled

#exif = get_exif(f"{all_files_path[0]}")
#labeled = get_labeled_exif(exif)
#print(labeled)
```

```
#print(TAGS)
```

```
from PIL.ExifTags import GPSTAGS

def get_geotagging(exif):
    if not exif:
        raise ValueError("No EXIF metadata found")

    geotagging = {}
    for (idx, tag) in TAGS.items():
        if tag == 'GPSInfo':
            if idx not in exif:
                raise ValueError("No EXIF geotagging found")

            for (key, val) in GPSTAGS.items():
                if key in exif[idx]:
                    geotagging[val] = exif[idx][key]
    return geotagging
```

```
#all_files_path = left_files_path[::-1] + right_files_path[1:]
#for file1 in all_files_path:
#  exif = get_exif(f"{file1}")
#  geotags = get_geotagging(exif)
#  print(geotags)
#  print(ok)
```

```
def get_decimal_from_dms(dms, ref):

    degrees = dms[0][0] / dms[0][1]
    minutes = dms[1][0] / dms[1][1] / 60.0
    seconds = dms[2][0] / dms[2][1] / 3600.0

    if ref in ['S', 'W']:
        degrees = -degrees
        minutes = -minutes
        seconds = -seconds

    return round(degrees + minutes + seconds, 5)

def get_coordinates(geotags):
    lat = get_decimal_from_dms(geotags['GPSLatitude'], geotags['GPSLatitudeRef'])

    lon = get_decimal_from_dms(geotags['GPSLongitude'], geotags['GPSLongitudeRef'])

    return (lat,lon)
```

Getting and Storing all Geolocations

```
all_geocoords = []
plt.figure(figsize = (20,10))
```

```
for file1 in tqdm(all_files_path):
  exif = get_exif(f"{file1}")
  geotags = get_geotagging(exif)
  #print(get_coordinates(geotags))
  geocoord = get_coordinates(geotags)
  all_geocoords.append(geocoord)
  #plt.scatter(x=geocoord[0], y=geocoord[1])
```

```
100%                                    443/443 [09:54<00:00, 1.34s/it]

<Figure size 1440x720 with 0 Axes>
```

```
!pip install pyproj
```

```
Collecting pyproj
  Downloading https://files.pythonhosted.org/packages/11/1d/1c54c672c2faf08d28fe78e15d664c048f786225bef95ad87b6c435cf69e/pyproj-3.1.0-cp37-cp37m-manylinux2010_x86_64.whl (6.6MB)
     |████████████████████████████████| 6.6MB 3.2MB/s
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from pyproj) (2020.12.5)
Installing collected packages: pyproj
Successfully installed pyproj-3.1.0
```

```
!pip install gmplot
```

```
Collecting gmplot
  Downloading https://files.pythonhosted.org/packages/2f/2f/45399c0a3b75d22a6ece1a1732a1670836cf284de7c1f91379a8d9b666a1/gmplot-1.4.1-py3-none-any.whl (164kB)
     |████████████████████████████████| 174kB 14.9MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from gmplot) (2.23.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (1.24.3)
Installing collected packages: gmplot
Successfully installed gmplot-1.4.1
```

```
print(np.min(np.array(all_geocoords)[:len1,0]),np.max(np.array(all_geocoords)[:len1,0]))
```

```
14.06462 14.077
```

```
print(np.min(np.array(all_geocoords)[:len1,1]),np.max(np.array(all_geocoords)[:len1,1]))
```

```
100.61506 100.61808
```

```
print(all_geocoords[int(len1/2)][0],all_geocoords[int(len1/2)][1])
```

```
14.06782 100.61706
```

Getting Bounds for plotting Polygon

This is still under-progress (almost completed) due to partial plotting of polygon by gmplot, so this will not be seen in the current plot, will be working on finishing this.

```
def get_geoloc_bounds(l, n):
    index_list = [None] + [i for i in range(1, len(l)) if abs(l[i] - l[i - 1]) > n] + [None]
    return [l[index_list[j - 1]:index_list[j]] for j in range(1, len(index_list))]
```

```
example =list(np.array(all_geocoords)[:,1])
```

```
print(list(np.array(all_geocoords)[:40,1]))
```

```
.61807, 100.61807, 100.61804, 100.61804, 100.61804, 100.61806, 100.61806, 100.61807, 100.61806, 100.61805, 100.61807, 100.61806, 100.61806, 100.61806, 100.61808, 100.61808, 100.61807, 100.61805, 100.61806, 100.61712, 100.61712, 100.617
```

```
print(len(example))
```

```
    101
```

```
for i in range(90,91):
  num = 1*i*1e-5
  split =get_geoloc_bounds(example, num)
```

```
print(len(split))
```

```
    16
```

Get upper and lower bound indices of each section

```
len_tot_split = 0
indx_lst = []
for num,each in enumerate(split):
  len_each_split = len(each)
  first_index = len_tot_split
  len_tot_split += len_each_split
  last_index = len_tot_split-1
  print(first_index,last_index)
  if num==0:
    continue
  indx_lst.append(first_index)
  indx_lst.append(last_index)
  #indx_lst_all.append(indx_lst)
```

```
    0 28
    29 57
    58 84
    85 112
    113 140
    141 168
    169 196
    197 224
    225 253
    254 281
    282 308
    309 336
    337 363
    364 391
    392 418
    419 442
```

```
lon_bounds = [list(np.array(all_geocoords)[:,1])[i] for i in indx_lst]
```

```
lat_bounds = [list(np.array(all_geocoords)[:,0])[i] for i in indx_lst]
```

Ideas for Image Registration of Geo-tagged Images

1) Online Method using Google Maps API through GmPlot

(Not useful when internet connection is weak/remote locations)

Creating Google Map Object using API Key and Gmplot

```
import gmplot
len1 = len(all_files_path)

# Create the map plotter:
apikey = '' # (It's hidden because it's a private key)
```

```
mid_lat = all_geocoords[int(len1/2)][0]
mid_lon = all_geocoords[int(len1/2)][1]

latMax = np.max(np.array(all_geocoords)[:len1,0])
latMin = np.min(np.array(all_geocoords)[:len1,0])



lngMax = np.max(np.array(all_geocoords)[:len1,1])
lngMin = np.min(np.array(all_geocoords)[:len1,1])


bounds = {'north':latMax, 'south':latMin, 'east':lngMax, 'west':lngMin}

gmap = gmplot.GoogleMapPlotter(mid_lat, mid_lon, 19, apikey=apikey,fit_bounds = bounds,tilt=45)


# Mark a hidden gem:
#gmap.marker(all_geocoords[0][0], all_geocoords[0][1], color='cornflowerblue')
```

Creating Marker object as well as embedding link of each image on your desktop as each marker

```
for count,file1 in enumerate(all_files_path[:len1]):
  fname = file1.split('/')[-1]
  img_tag = f"C:/Users/User%20Default/Downloads/RGB-img/Uni_img/{fname}"
  gmap.marker(all_geocoords[count][0], all_geocoords[count][1], color='purple',info_window =f'<a href={img_tag}>{fname} <br/> {(all_geocoords[count][0],all_geocoords[count][1])} </a>')
```

```
#gmap.polygon(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1], face_color='green', edge_color='cornflowerblue', edge_width=5,face_alpha=0.6)
```

```
gmap.polygon(lat_bounds, lon_bounds, face_color='blue', edge_color='cornflowerblue', edge_width=5,face_alpha=0.6)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-886adf5ce43a> in <module>()
----> 1 gmap.polygon(lat_bounds, lon_bounds, face_color='blue', edge_color='cornflowerblue', edge_width=5,face_alpha=0.6)

NameError: name 'lat_bounds' is not defined
```

SEARCH STACK OVERFLOW

Saving the GMap plot

[ ] ↳ *1 cell hidden*

Video Link of Output

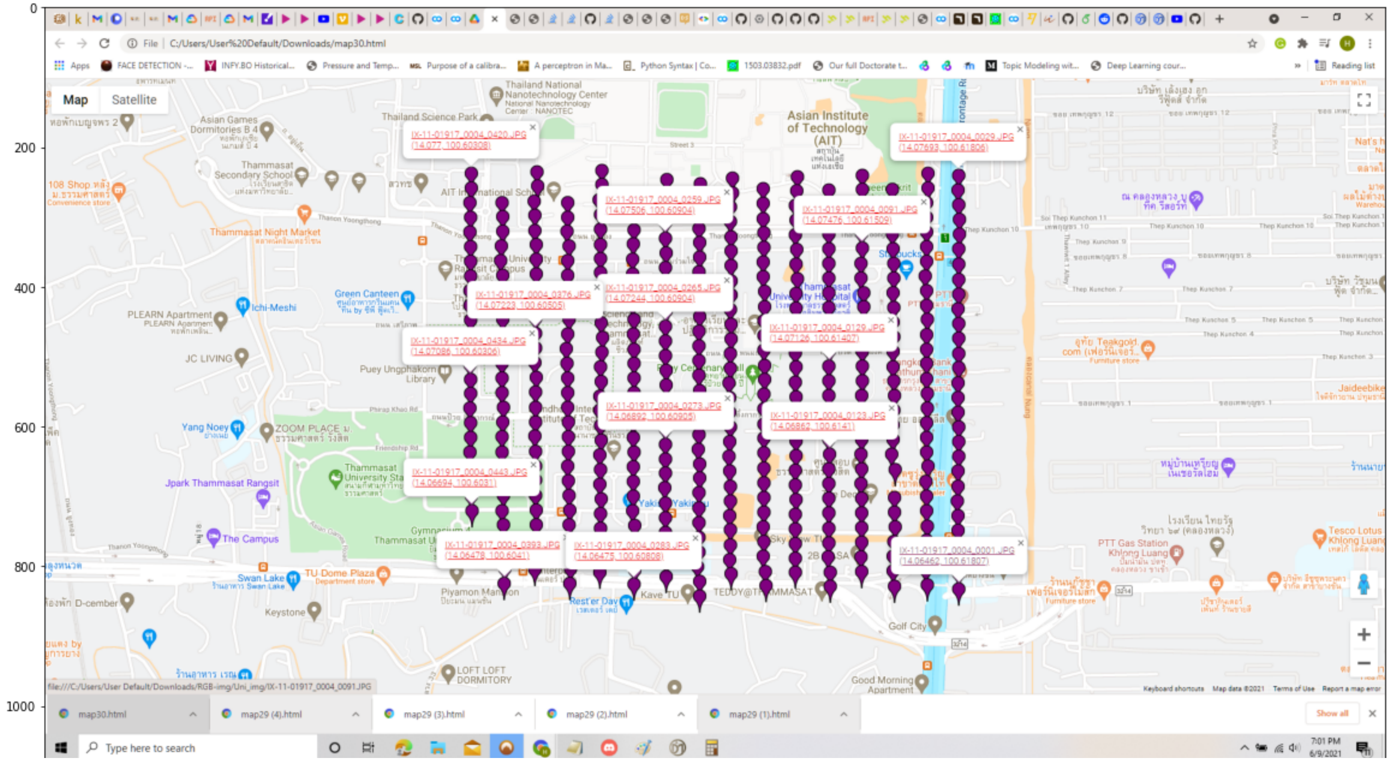
https://www.loom.com/share/f7534dbe837541e7b2ea9611580c6ce6

Screenshot of the Output

```
img_scrnsht = cv2.imread('drive/MyDrive/Screenshot_gmaps_gelocation_marker_embed_443_images.png')
```

```
plt.figure(figsize = (20,20))

plt.imshow(img_scrnsht)
```

```
<matplotlib.image.AxesImage at 0x7fa5381bfe90>
```



Extra Stuff

[ ] ↳ *30 cells hidden*

Ideas for Image Registration of Geo-tagged Images

2) Offline Method using Matplotlib

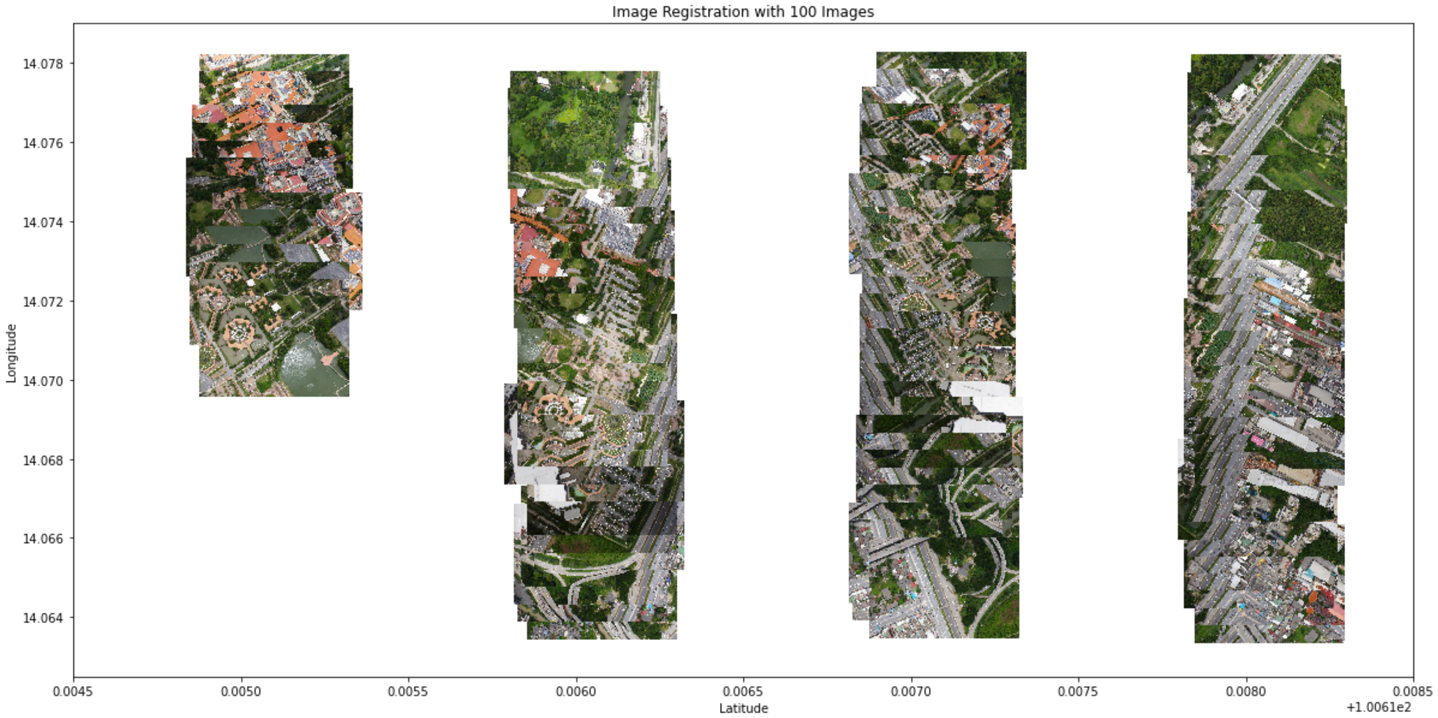
(Works offline but not overlayed on a map)

```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

2 a.) Plotting Images on respective geo-locations

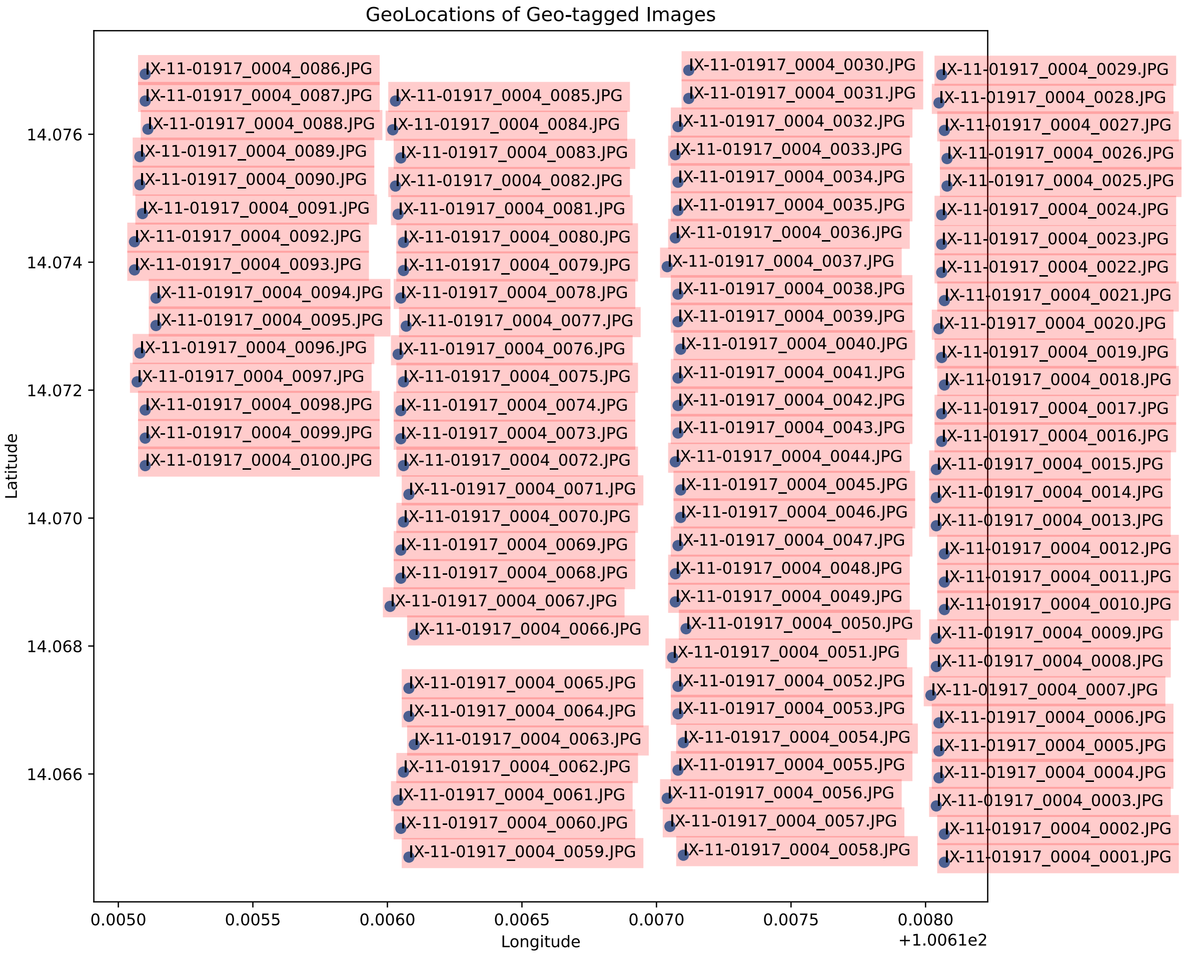
(Obscures images, different to decipher if images are missing/blurred,etc.)

```
fig, ax = plt.subplots()
fig.set_size_inches(20,10)
ax.set_xlabel('Latitude')
ax.set_ylabel('Longitude')
ax.set_xlim(100.6145,100.6185)
len1 = 100
ax.set_title(f'Image Registration with {len1} Images')
ax.set_ylim(14.0625,14.079)

ax.plot(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1],linestyle='None')

def aerial_images_register(x, y,ax=None):
    ax = ax or plt.gca()
    for count,points in enumerate(zip(x,y)):
        lat,lon = points
        image = plt.imread(all_files_path[count])
        #print(ax.figure.dpi)
        im = OffsetImage(image, zoom=1.5/ax.figure.dpi)
        im.image.axes = ax
        ab = AnnotationBbox(im, (lat,lon), frameon=False, pad=0.0,)

        ax.add_artist(ab)

aerial_images_register( np.array(all_geocoords)[:len1,1],np.array(all_geocoords)[:len1,0], ax=ax)
```



2 b.) Embed the Images on respective geo-locations markers so as to take care of problem in 2 a)

```
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("svg")

len1 = 100
fig, ax = plt.subplots()
fig.set_size_inches(10,10)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('GeoLocations of Geo-tagged Images')

ax.scatter(np.array(all_geocoords)[:len1,1], np.array(all_geocoords)[:len1,0])
#text = ax.annotate("Link", xy=(2,5), xytext=(2.2,5.5),
#                    url='http://matplotlib.org',
#                    bbox=dict(color='w', alpha=1e-6, url='http://matplotlib.org'))
def hover(event):
  vis = annot.get_visible()
  if event.inaxes == ax:
      cont, ind = sc.contains(event)
      if cont:
          update_annot(ind)
          annot.set_visible(True)
          fig.canvas.draw_idle()
      else:
          if vis:
              annot.set_visible(False)
              fig.canvas.draw_idle()
for count,file1 in enumerate(all_files_path[:len1]):
  fname = file1.split('/')[-1]
  img_tag = f"C:/Users/User%20Default/Downloads/RGB-img/Uni_img/{fname}"
  txt = plt.text(all_geocoords[count][1], all_geocoords[count][0],f'{fname}' , url=file1)
  txt.set_bbox(dict(color='r', alpha=0.2, url=txt.get_url()))
  fig.canvas.mpl_connect("motion_notify_event", hover)
```



```
fig.savefig('drive/MyDrive/check1.jpg')
```

Ideas for Image Registration of Geo-tagged Images

3)Offline Method using Folium

(Works offline and and overlayed on map)

Modified today so that each image pops up directly when clicking on marker, instead of a link-The Image name shows up when it hovers on the marker, and when you click it, the image pops up

Hovering images caused few lags, so need to work on that on-the-side, but for now, when you click the marker, image directly appears

```
!pip install folium
```

```
    Requirement already satisfied: folium in /usr/local/lib/python3.7/dist-packages (0.8.3)
    Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from folium) (2.23.0)
    Requirement already satisfied: branca>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from folium) (0.4.2)
    Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from folium) (1.15.0)
    Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from folium) (1.19.5)
    Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from folium) (2.11.3)
    Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (3.0.4)
    Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2.10)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2020.12.5)
    Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (1.24.3)
    Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->folium) (2.0.1)
```

```
import folium
from folium import features
from scipy.spatial import ConvexHull
```

```
#Reference: https://nbviewer.jupyter.org/github/python-visualization/folium/blob/master/examples/Polygons_from_list_of_points.ipynb


def create_convexhull_polygon(
    map_object, list_of_points, layer_name, line_color, fill_color, weight, text
):

    # Since it is pointless to draw a convex hull polygon around less than 3 points check len of input
    if len(list_of_points) < 3:
        return

    # Create the convex hull using scipy.spatial
    form = [list_of_points[i] for i in ConvexHull(list_of_points).vertices]

    # Create feature group, add the polygon and add the feature group to the map
    fg = folium.FeatureGroup(name=layer_name)
    fg.add_child(
        folium.vector_layers.Polygon(
            locations=form,
            color=line_color,
            fill_color=fill_color,
            weight=weight,
            popup=(folium.Popup(text)),
        )
    )
    map_object.add_child(fg)

    return map_object
```

Creating Folium Map Object

```
len1 = len(all_files_path)
len1 = 2
mid_lat = all_geocoords[int(len1/2)][0]
mid_lon = all_geocoords[int(len1/2)][1]
```

```
latMax = np.max(np.array(all_geocoords)[:len1,0])
latMin = np.min(np.array(all_geocoords)[:len1,0])



lngMax = np.max(np.array(all_geocoords)[:len1,1])
lngMin = np.min(np.array(all_geocoords)[:len1,1])

SJER_map = folium.Map([mid_lat,mid_lon],
                  zoom_start=15,min_lat=latMin, max_lat = latMax, min_lon = lngMin, max_lon=lngMax)
```

Creating Marker object and embedding the local image of your local desktop folder on each marker and adding to the Map Object

```
import PIL
import base64

target_resized_folder = 'drive/MyDrive/Uni_img_resized/'

for count,file1 in enumerate(all_files_path[:len1]):
  fname = file1.split('/')[-1]
  #img_tag = f"C:/Users/User%20Default/Downloads/RGB-img/Uni_img/{fname}"
  #image = PIL.Image.open(file1)
  width = int(632)
  height = int(420)
  #image = image.resize((width, height), PIL.Image.ANTIALIAS)
  #image.save(target_resized_folder + fname , quality=100)
  encoded = base64.b64encode(open(target_resized_folder + fname, 'rb').read()).decode("UTF-8")
  html = '''<img style="width:100%; height:100%;" src="data:image/jpg;base64,{}">'''.format

  iframe = folium.IFrame(html(encoded),width=width, height=height)
  popup = folium.Popup(iframe,max_width=width)
  #mk = features.Marker([all_geocoords[count][0], all_geocoords[count][1]],popup=f'<a href={img_tag}>{fname} <br/> {(all_geocoords[count][0],all_geocoords[count][1])} </a>',icon=folium.Icon(color="darkpurple"))
  #mk = features.Marker([all_geocoords[count][0], all_geocoords[count][1]],tooltip=f'<a href={img_tag}>{fname} <br/> {(all_geocoords[count][0],all_geocoords[count][1])} </a>',popup=popup)
  mk = features.Marker([all_geocoords[count][0], all_geocoords[count][1]],tooltip=iframe,popup=popup)

  '''
  vega = folium.features.VegaLite(
    html(encoded),
    width="50%",
    height="50%",)

  popup = folium.Popup()
  vega.add_to(popup)
  popup.add_to(mk)
  '''
  SJER_map.add_child(mk)
```

```
list_of_points = list(zip(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1]))
```

Creating and Drawing Polygon on the list of (lat,lon) points

```
SJER_map = create_convexhull_polygon(
    SJER_map,
    list_of_points,
    layer_name="Boundary",
    line_color="green",
    fill_color="blue",
    weight=7,
    text="Boundary",
)
```

Output Map

```
SJER_map
```

```
SJER_map.save('drive/MyDrive/off_map12.html')
```

Video Link of Output

Where Image Just pops up (New): https://www.loom.com/share/b3631218aec34478ae3723da2c0c8782

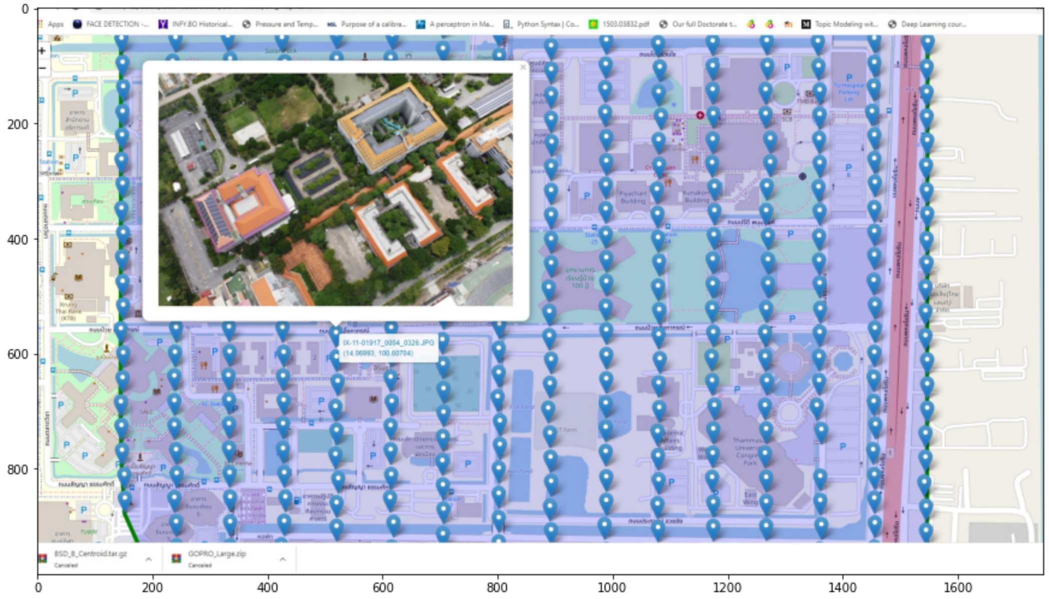
Where Image is embedded as a link (Old): https://www.loom.com/share/2e632e81f49e4578afd7a7512d8b865f

Screenshot of the Output (Where Image Pops Up) (Latest)

```
img_scrnsht = cv2.imread('drive/MyDrive/Screenshot_gmaps_gelocation_marker_embed_443_images_offline_image_pops_up.jpg')
```

```
plt.figure(figsize = (15,15))

plt.imshow(cv2.cvtColor(img_scrnsht,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f6d38ae6e10>
```

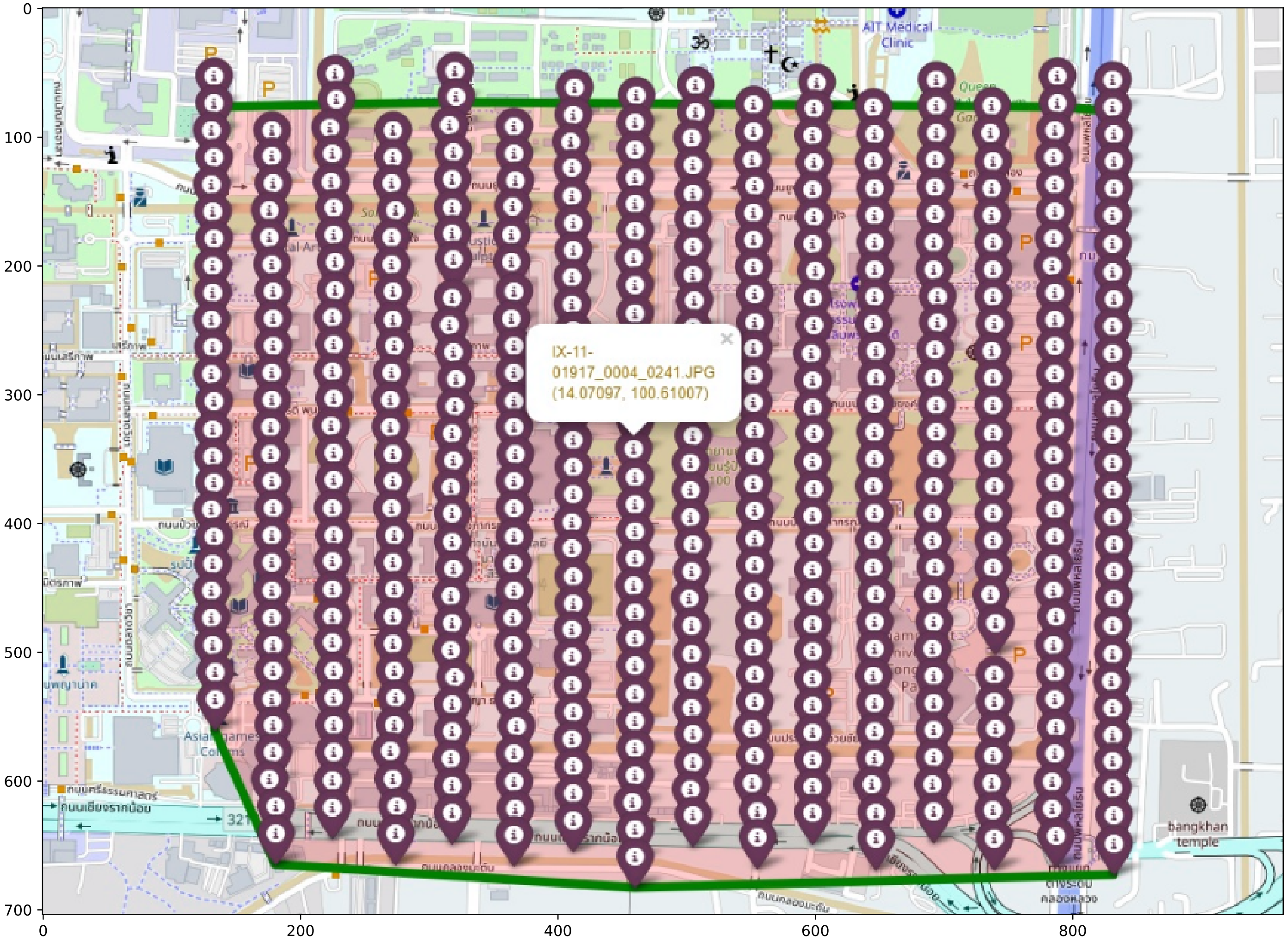


Screenshot of the Output (Where Image is Embedded as a link)

```
img_scrnsht = cv2.imread('drive/MyDrive/Screenshot_gmaps_gelocation_marker_embed_443_images_offline.jpg')
```

```
plt.figure(figsize = (15,15))
plt.imshow(img_scrnsht)
```

```
<matplotlib.image.AxesImage at 0x7f258f97c890>
```



```
print(len(all_files_path))
```

```
443
```

Blur Detection in Images

Two Methods:

a) Using Variance of Laplacian

b) Deep Learning

```
def laplacian_var(image):
  return cv2.Laplacian(image, cv2.CV_64F).var()
```

Synthetically Generating Blurred Photos of University Dataset using Gaussian and Motion Blur (Will add mroe types later on)

```
import numpy as np
from PIL import Image
from scipy.signal import convolve2d
```

```
def gauss_blur(img,filter_size=3):
  gblurred = cv2.GaussianBlur(img, ksize=(filter_size, filter_size), sigmaX=0, sigmaY=0)

  return gblurred
```

```
def motion_blur(image, degree=12, angle=45):
    image = np.array(image)

    M = cv2.getRotationMatrix2D((degree / 2, degree / 2), angle, 1)
    motion_blur_kernel = np.diag(np.ones(degree))
    motion_blur_kernel = cv2.warpAffine(motion_blur_kernel, M, (degree, degree))

    motion_blur_kernel = motion_blur_kernel / degree
    blurred = cv2.filter2D(image, -1, motion_blur_kernel)

    # convert to uint8
    cv2.normalize(blurred, blurred, 0, 255, cv2.NORM_MINMAX)
    blurred = np.array(blurred, dtype=np.uint8)
    return blurred
```

```
randomlist = list(range(0,20))
```

```
random.shuffle(randomlist)
```

```
actual_labels = [1]*100 + [0]*100
```

```
tqdm = partial(tqdm, position=0, leave=True)
```

```
thresh=100
pred_labels = []
blur = 1
fm_all = []
for count,file1 in tqdm(enumerate(all_files_path[:200])):
  image = cv2.imread(file1)
  if count>=100:
    #out_image = cv2.resize(image,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = image

  elif count >=50 and count <100:
    blurred = gauss_blur(image)
    #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = blurred

  elif count<50:
    blurred = motion_blur(image)
    #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = blurred

  gray = cv2.cvtColor(out_image, cv2.COLOR_BGR2GRAY)
  fm = laplacian_var(gray)
  fm_all.append(fm)
  if fm < thresh:
    pred_labels.append(blur)
  else:
    blur=0
    pred_labels.append(blur)
```

```
200it [06:57,  2.09s/it]
```

```
print(fm_all)
```

```
.15764999328951, 77.43002590493079, 99.51975612425079, 88.28907024995883, 87.85860037163116, 79.3718407484331, 68.46566953535296, 69.13205816666556, 65.45203099972775, 295.46137270636336, 336.44413041657657, 294.4099185221482, 305.70404
```

```
from sklearn.metrics import classification_report
```

```
target_names = ['Not Blurred', 'Blurred']
```

```
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(pred_class, actual_class,
                          title='Confusion matrix'):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    Code from: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
    """
    cm = confusion_matrix(actual_class, pred_class)

    cmap = plt.cm.Blues
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    cm = np.nan_to_num(cm)

    print('Confusion matrix')
    print(cm)

    plt.figure(figsize=(10,10))

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Classification Results

```
print(classification_report(actual_labels, pred_labels, target_names=target_names))
```

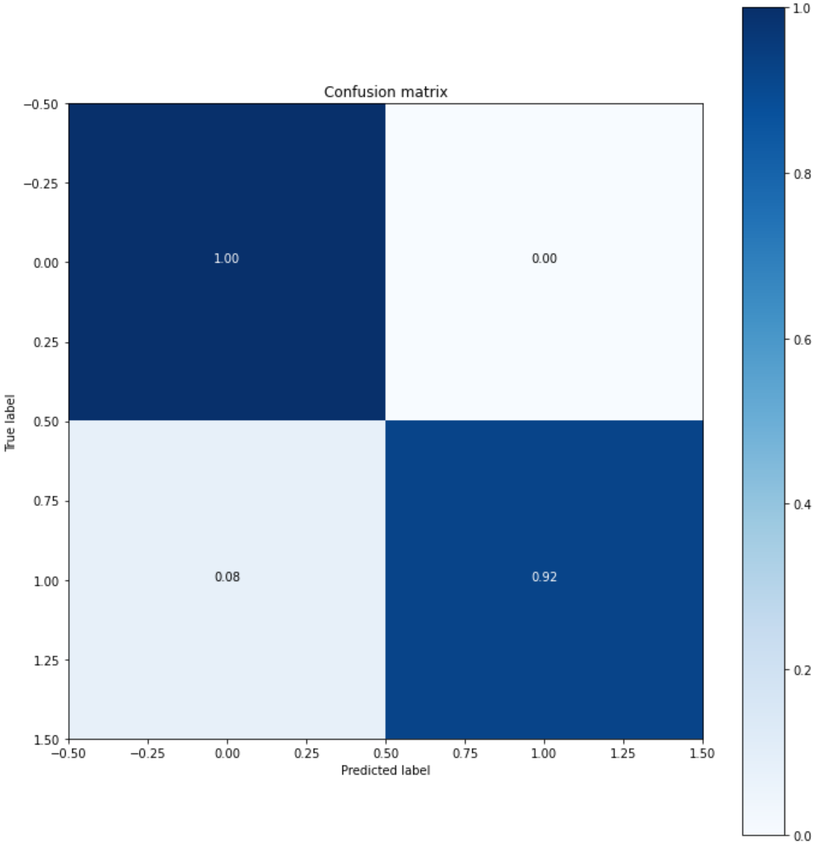
```
              precision    recall  f1-score   support

 Not Blurred       0.93      1.00      0.96       100
     Blurred       1.00      0.92      0.96       100

    accuracy                           0.96       200
   macro avg       0.96      0.96      0.96       200
```

```
    weighted avg       0.96      0.96      0.96       200
```

Confusion Matrix

```
plot_confusion_matrix(pred_labels,actual_labels)
```

```
    Confusion matrix
    [[1.   0.  ]
     [0.08 0.92]]
```



Seems to work pretty well, will use more images and blur conditions and check performance and indivudual cases where the algorithm breaks after that

Blur Detection using Deep Learning techniques (Incomplete-Will finish it up tomorrow)

```
import os
import torch
from torch.utils.data import Dataset, DataLoader, ConcatDataset
from skimage import io, transform,data
from torchvision import transforms, utils
import numpy as np
import math
from matplotlib.patches import Ellipse
from skimage.draw import ellipse
import glob
import random
import torchvision.transforms.functional as TF

class UniData(Dataset):
  def __init__(self, root_dir_list,train=True,test=False, transform=None,transform_test=None):
    self.root_dir_list = root_dir_list
    self.train = train
    self.transform = transform


  def __len__(self):
    if self.train==True:
      return len(self.root_dir_list)



  def __getitem__(self,index):
    #index+=1
    if torch.is_tensor(index):
      index = index.tolist()

    image = cv2.imread(self.root_dir_list[index])
    img_name = self.root_dir_list[index].split('/')[-1]
    label = 1
    out_image = image
    if index>=20:
      #out_image = cv2.resize(image,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
      #out_image = image
      label=0

    elif index >=10 and index <20:
      blurred = gauss_blur(image)
      #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
      out_image = blurred

    elif index<10:
      blurred = motion_blur(image)
      #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
      out_image = blurred

    out_image = cv2.cvtColor(out_image,cv2.COLOR_BGR2RGB)

    if self.transform:
      out_image = self.transform(out_image)

    sample = {'image': out_image,'label':label,'name':img_name}

    #print(sample['label'])
    #plt.imshow(sample['image'])
    #plt.show()

    return sample
```

```
#Pre processing the data
normalize = transforms.Normalize(mean = [0.485,0.456,0.406],
                                std = [0.229,0.224,0.225])
resize = transforms.Resize((224,224))



preprocessor = transforms.Compose([ transforms.ToTensor(),resize, normalize
                                   ])

aerial_dataset_full = UniData(all_files_path[:40],transform=preprocessor)

# Creating data indices for training and validation splits:
dataset_size = len(aerial_dataset_full)
indices = list(range(dataset_size))
validation_split = 0.2
```

```
split = int(np.floor(validation_split * dataset_size))
shuffle_dataset = True
random_seed= 101

if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]

# Creating PT data samplers and loaders:
train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)

aerial_train_loader = torch.utils.data.DataLoader(aerial_dataset_full, batch_size=2,
                                           sampler=train_sampler)
aerial_validation_loader = torch.utils.data.DataLoader(aerial_dataset_full, batch_size=2,
                                                sampler=valid_sampler)
```

```
def training_and_validation_loop(epochs,xp_lr_scheduler,model,optmizer,aerial_train_loader,aerial_validation_loader,best_acc,best_model_wts,saved_model_name):

  train_loss = []
  test_loss = []
  accuracy = []

  for e in range(epochs):
        step_lr_scheduler.step()

        #put model in training mode
        model.train()
        avg_loss = 0

        for i, sample in enumerate(aerial_train_loader):
              optimizer.zero_grad()
              x = sample['image']
              print(x.shape)
              y = sample['label']
              print(y.shape)

              if gpu_flag:
                    img_var = Variable().cuda()
                    label_actual = Variable(y).cuda()
              else:
                    img_var = Variable(x)
                    label_actual = Variable(y)

              label_predicted = model.forward(img_var)
              loss = criterion(label_predicted,label_actual)
              loss.backward()

              if(i%10 == 0):
                    print(i, loss.item())
              avg_loss+=loss.item()
              optimizer.step()

        print("Done Training")
        train_loss.append(avg_loss*1.0/(i+1))

        #set model in evaluation mode
        model.eval()
        avg_loss = 0
        correct_pred = 0
        total_pred = 0

        for i, sample in enumerate(aerial_validation_loader):
            x_test = sample['image']
            y_test = sample['label']

            if gpu_flag:
                img_test_var = Variable(x_test).cuda()
                label_test_var = Variable(y_test).cuda()
            else:
                img_test_var = Variable(x_test)
                label_test_var = Variable(y_test)

            label_predicted_test = model.forward(img_test_var)
            loss = criterion(label_predicted_test,label_test_var)
            avg_loss+=loss.item()
            vals, label_predicted = torch.max(label_predicted_test,1)

            correct_pred += (label_predicted.cpu().data.numpy()==label_test_var.cpu().data.numpy()).sum()
            total_pred += len(label_predicted_test.cpu())

        test_loss.append(avg_loss*1.0/i)
        accuracy.append(correct_pred*100.0/total_pred)
        print("Epoch: ", e, "Train Loss: ", train_loss[-1], "Test Loss: ", test_loss[-1], "Accuracy: ", accuracy[-1])
        '''
        #replace model saved
        if accuracy[-1]>best_acc:
            best_acc = accuracy[-1]
            best_model_wts = copy.deepcopy(model.state_dict())
            model.load_state_dict(best_model_wts)
            torch.save(model,f'/content/drive/My Drive/sentinel-2_rgb/{saved_model_name}.pt')
            print("Saved model with accuracy: ", best_acc)
        '''
  return train_loss,test_loss,accuracy
```

```
# Initialize the model
model = models.vgg16(pretrained=True)

# Change the device to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else "cpu")
```

```
# Freeze training for all layers
for param in model.features.parameters():
    param.require_grad = False

num_classes=2

num_features = model.classifier[6].in_features
# Remove last layer
features = list(model.classifier.children())[:-1]

# Add our layer with 2 outputs
features.extend([nn.Linear(num_features, num_classes)])

# Replace the model classifier
model.classifier = nn.Sequential(*features)

# define loss function
criterion = nn.CrossEntropyLoss()

# setup SGD
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

```
gpu_flag = torch.cuda.is_available()
print(gpu_flag)
if gpu_flag:
    model = model.cuda()
    criterion = criterion.cuda()

print(model)
```

```
True
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

```
summary(model, (3, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
```

```
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                    [-1, 2]           8,194
================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.77
Params size (MB): 512.19
Estimated Total Size (MB): 731.54
----------------------------------------------------------------
```

```
tqdm = partial(tqdm, position=0, leave=True)
```

```
epochs=3
#best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0
train_loss_vgg,test_loss_vgg,accuracy_vgg = training_and_validation_loop(epochs,step_lr_scheduler,model,optimizer,aerial_train_loader,aerial_validation_loader,best_acc,best_model_wts,'vgg16')
```

```
torch.Size([2, 3, 224, 224])
torch.Size([2])
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-63-c0573419a60e> in <module>()
      2 #best_model_wts = copy.deepcopy(model.state_dict())
      3 best_acc = 0.0
----> 4 train_loss_vgg,test_loss_vgg,accuracy_vgg = training_and_validation_loop(epochs,step_lr_scheduler,model,optimizer,aerial_train_loader,aerial_validation_loader,best_acc,best_model_wts,'vgg16')
```

6 frames

```
<ipython-input-56-511f5b7deed6> in training_and_validation_loop(epochs, xp_lr_scheduler, model, optmizer, aerial_train_loader, aerial_validation_loader, best_acc, best_model_wts, saved_model_name)
     26                     label_actual = Variable(y)
     27 
---> 28             label_predicted = model.forward(img_var)
     29             loss = criterion(label_predicted,label_actual)
     30             loss.backward()

/usr/local/lib/python3.7/dist-packages/torchvision/models/vgg.py in forward(self, x)
```

To-Do Tasks

- Finish up training with multiple cnn networks and compare performance
- Increase dataset and blur conditions and check where algorithm breaks
- After getting a satisfactory analysis, proceed to Marker Detection

```
--> 889             result = self.forward(*input, **kwargs)
```

Reading images and Extracting SuperPoint Keypoints and Descriptors from each image

↳ 15 cells hidden

```
    118         for module in self:
```

Loading and Initialing the SuperPoint Pretrained Network

[ ] ↳ 1 cell hidden

```
    888         else:
```

Now Extracting Keypoints and Descriptors from all images and storing them

[ ] ↳ 7 cells hidden

```
    397
```

Image Matching (Robust) through RANSAC and Homography Matrix computation

[ ] ↳ 8 cells hidden

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py in _conv_forward(self, input, weight, bias)
```

Auto-Selection/Ordering of Images (Complete)

[ ] ↳ 20 cells hidden

```
RuntimeError: Expected 4-dimensional input for 4-dimensional weight [64, 3, 3, 3], but got 1-dimensional input of size [0] instead
```

Perspective Transformation b/w consecutive pairs through the computed Homography Matrices

[ ] ↳ 6 cells hidden

Final Mosaiced Image (with 22 images)

[ ] ↳ 1 cell hidden

To-Do Tasks

- Seam Removal
- Improve On this Enhancement
- Extend to 50 images

✓ 0s completed at 7:48 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.