```
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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
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)
```

```
        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)
    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)

            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, 442820.94it/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 fil
es_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, 100.61708, 100
```

```
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

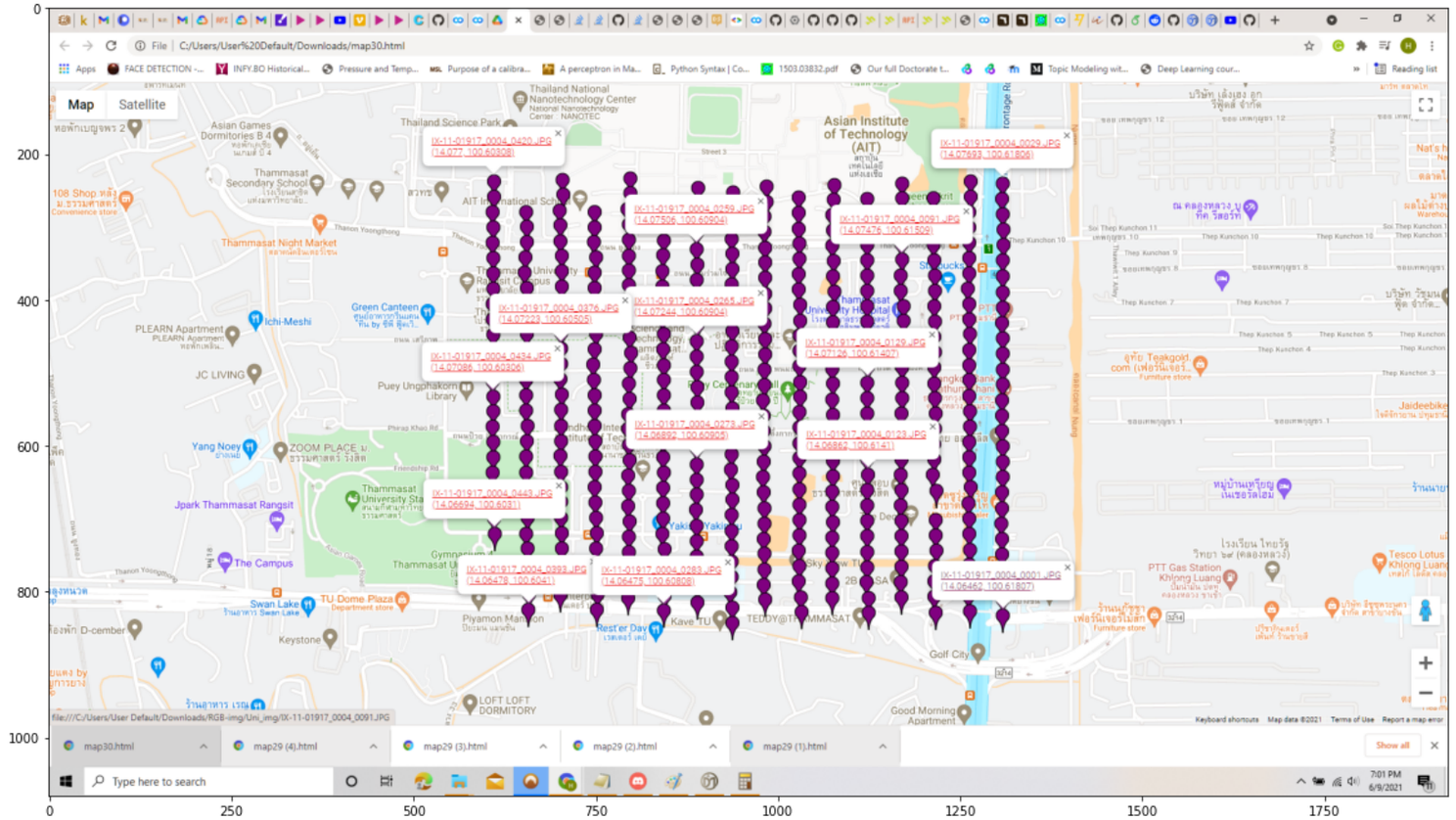
https://www.loom.com/share/f7534dbe837541e7b2ea9611580c6ce6

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)

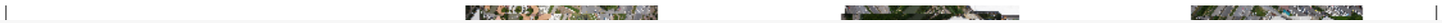
```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

2 a.) Plotting Images on respective geo-locations

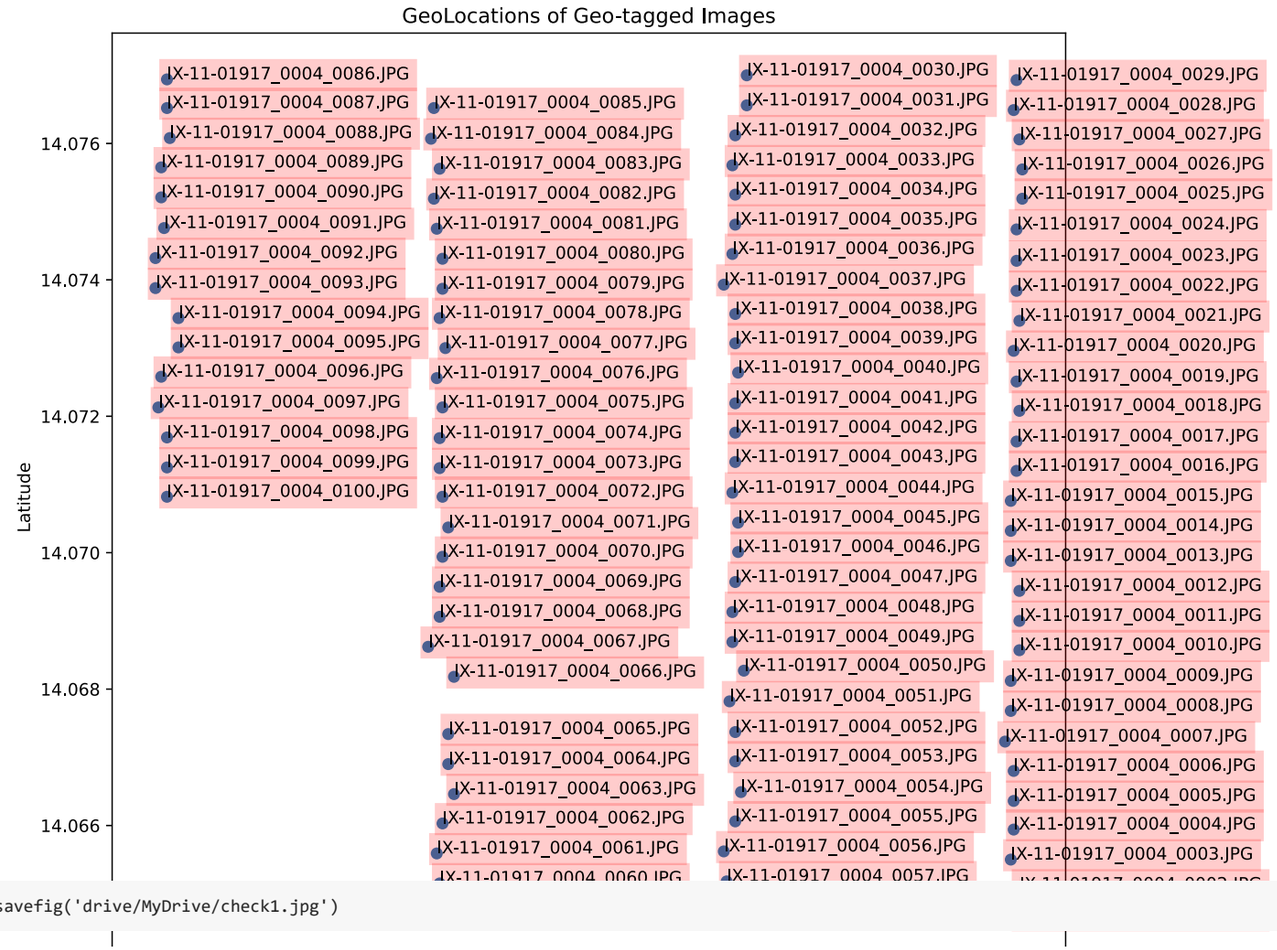
(Obscures images, different to decipher if images are missing/blurred,etc.)

```
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="{}">'''.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

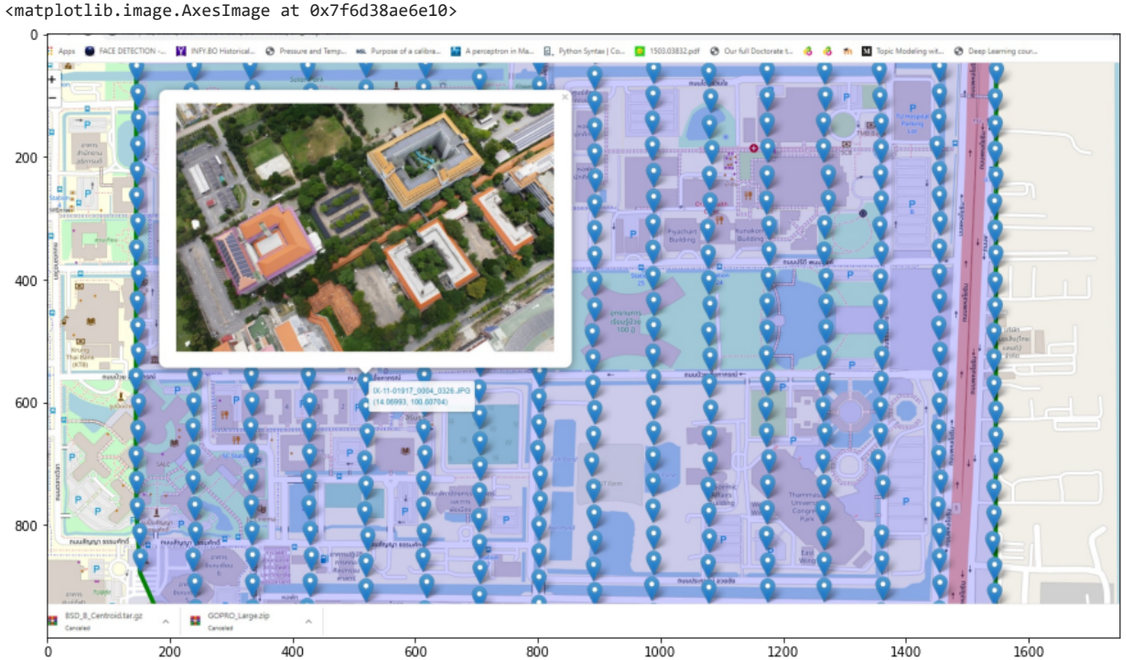
Where Image is embedded as a link (Old): https://www.loom.com/share/2e632e81f49e4578afd7a7512d8b865f

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>
```

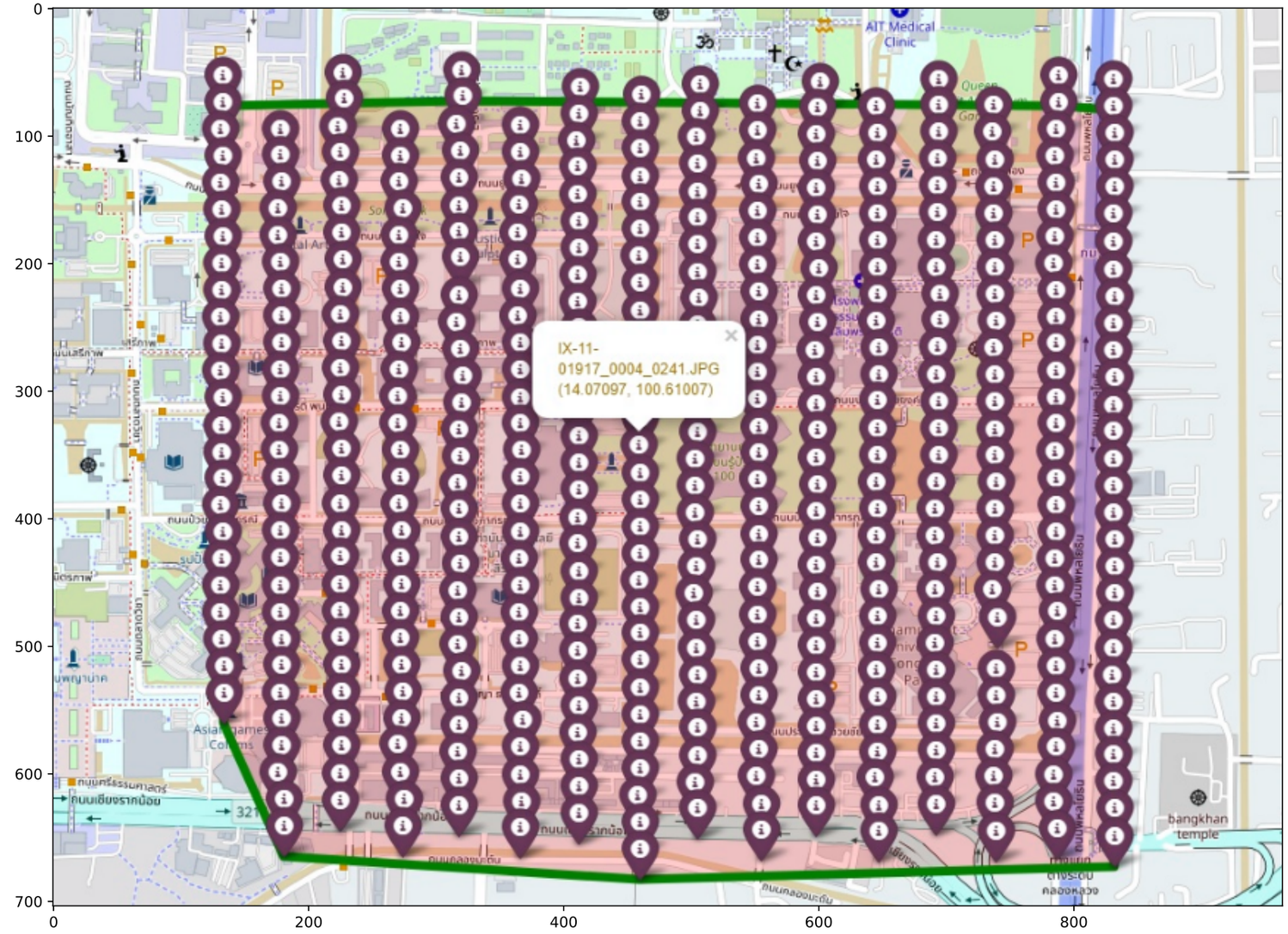


Screenshot of the Output (Where Image is Embedded as a link)

```
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()
```

(Modified) Synthetically Generating Blurred Photos of University Dataset using Gaussian and Motion Blur

```
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.pandas(position=0, leave=True)
```

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

Increased size to full dataset

```
thresh=100
pred_labels = []
actual_labels = []
fm_all = []
for count,file1 in enumerate(tqdm(all_files_path)):
  blur = 1
  image = cv2.imread(file1)
  if count>=int(len(all_files_path)/2):
    #out_image = cv2.resize(image,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = image
    actual_labels.append(0)

  elif count >=int(len(all_files_path)/4) and count <int(len(all_files_path)/2):
    blurred = gauss_blur(image)
    #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = blurred
    actual_labels.append(1)

  elif count<int(len(all_files_path)/4):
    blurred = motion_blur(image)
    #out_image = cv2.resize(blurred,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
    out_image = blurred
    actual_labels.append(1)

  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)
```

```
100%|██████████| 443/443 [09:03<00:00,  1.23s/it]
```

```
print(np.where(np.array(pred_labels)!=np.array(actual_labels))[0])
```

```
[ 92 104 105]
```

```
print([fm_all[i] for i in np.where(np.array(pred_labels)!=np.array(actual_labels))[0] ])
```

```
[102.12924836695245, 87.4009272494633, 97.06603004122752]
```

```
print([actual_labels[i] for i in np.where(np.array(pred_labels)!=np.array(actual_labels))[0] ])
```

```
[1, 0, 0]
```

```
print([pred_labels[i] for i in np.where(np.array(pred_labels)!=np.array(actual_labels))[0] ])
```

```
[0, 1, 1]
```

```
print(len(np.where(np.array(fm_all)<100)[0]))
```

```
print(len(np.where(np.array(fm_all)<100)[0]))
```

```
101
```

```
print(fm_all)
```

```
.24739043, 95.62888135977899, 102.12924836695245, 67.73565237493526, 77.9278639108527, 82.50093028966043, 59.12963420499348, 96.73604820808379, 99.09528732806525, 86.1627887865365, 337.1683222574333, 271.6831068088014, 307.38890868103925, 125
```

```
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)
    print('Confusion matrix')

    print(cm)

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


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

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

    fmt = '.2f'
    thresh = cm.max() / 2.
    print(thresh)
    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
```

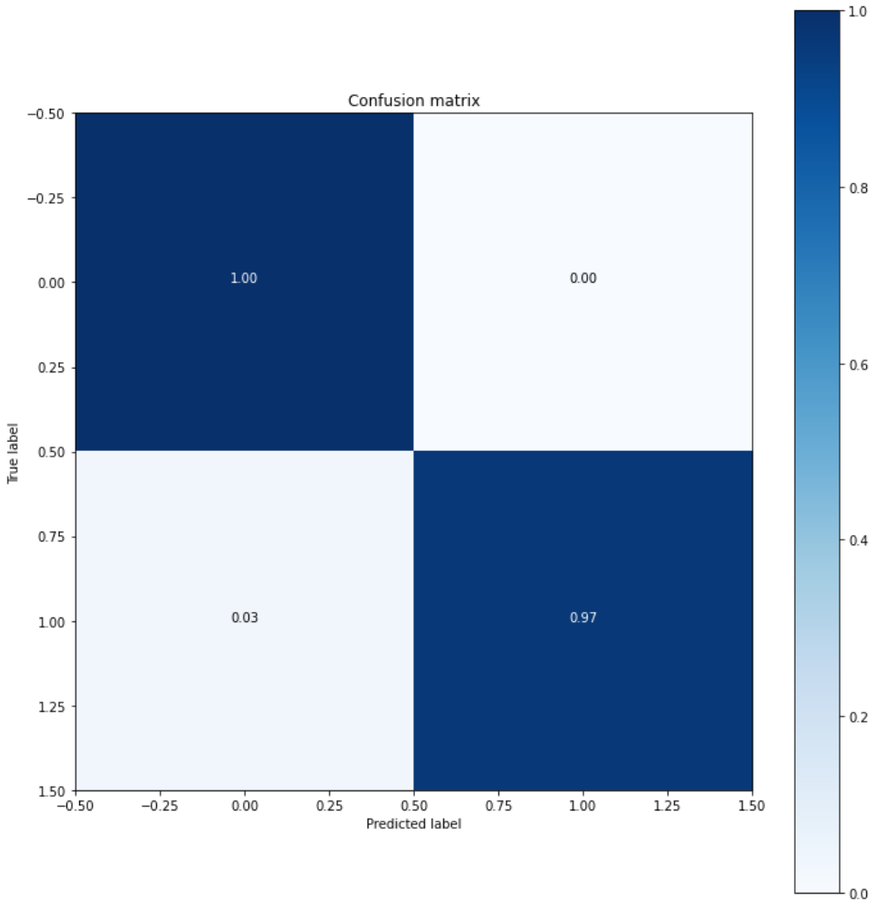
```
    Not Blurred       0.97      1.00      0.98       222
        Blurred       1.00      0.97      0.98       221

       accuracy                           0.98       443
      macro avg       0.98      0.98      0.98       443
   weighted avg       0.98      0.98      0.98       443
```

+ Code + Text

Confusion Matrix

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

```
Confusion matrix
[[222   0]
 [  7 214]]
0.5
```

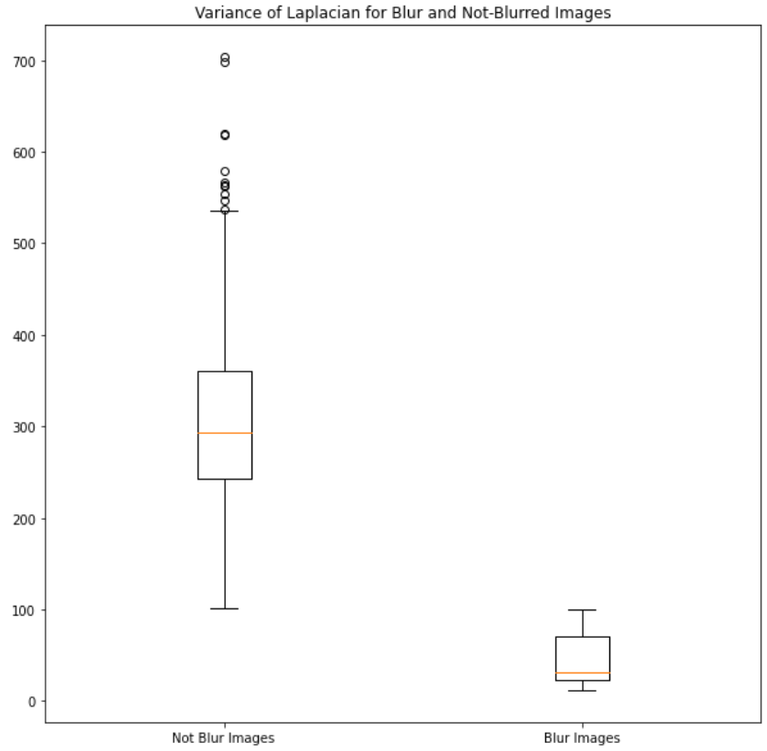



```
data = [[fm_all[i] for i in np.where(np.array(pred_labels)==0)[0]], [fm_all[i] for i in np.where(np.array(pred_labels)==1)[0]]]
```

Boxplot of VoL on Blurred and Non-Blurred Images, We can see there is a very good distinction b/w the clusters

```
fig, ax = plt.subplots(figsize=(10,10))

ax.set_xticklabels(['Not Blur Images','Blur Images'])
ax.set_title('Variance of Laplacian for Blur and Not-Blurred Images')
ax.boxplot(data)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depre
  return array(a, dtype, copy=False, order=order)
```



Let's Now Check with Different Machine Learning Classifiers and see if they do better- The Purpose is that, we could skip setting a threshold

```
# Import required libraries for performance metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_validate

# Define dictionary with performance metrics
scoring = {'accuracy':make_scorer(accuracy_score),
           'precision':make_scorer(precision_score),
           'recall':make_scorer(recall_score),
           'f1_score':make_scorer(f1_score)}

# Import required libraries for machine learning classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier



# Instantiate the machine learning classifiers
log_model = LogisticRegression(max_iter=10000)
svc_model = LinearSVC(dual=False)
rbf_model = SVC(gamma = 2, C=1)

dtr_model = DecisionTreeClassifier()
rfc_model = RandomForestClassifier()
gnb_model = GaussianNB()
mlp_model =  MLPClassifier(alpha=1, max_iter=1000)

adb_model = AdaBoostClassifier()

# Define the models evaluation function
def models_evaluation(X, y, folds):

    '''
    X : data set features
    y : data set target
    folds : number of cross-validation folds

    '''

    # Perform cross-validation to each machine learning classifier
    log = cross_validate(log_model, X, y, cv=folds, scoring=scoring)
    svc = cross_validate(svc_model, X, y, cv=folds, scoring=scoring)
    dtr = cross_validate(dtr_model, X, y, cv=folds, scoring=scoring)
    rfc = cross_validate(rfc_model, X, y, cv=folds, scoring=scoring)
    gnb = cross_validate(gnb_model, X, y, cv=folds, scoring=scoring)
    mlp = cross_validate(mlp_model, X, y, cv=folds, scoring=scoring)
    svc_rbf = cross_validate(rbf_model, X, y, cv=folds, scoring=scoring)
    adb = cross_validate(adb_model, X, y, cv=folds, scoring=scoring)

    # Create a data frame with the models perfoamnce metrics scores
    models_scores_table = pd.DataFrame({'Logistic Regression':[log['test_accuracy'].mean(),
                                                               log['test_precision'].mean(),
                                                               log['test_recall'].mean(),
                                                               log['test_f1_score'].mean()],
```

```
                                    'Support Vector Classifier':[svc['test_accuracy'].mean(),
                                                                 svc['test_precision'].mean(),
                                                                 svc['test_recall'].mean(),
                                                                 svc['test_f1_score'].mean()],

                                    'Support Vector RBF Kernel':[svc_rbf['test_accuracy'].mean(),
                                                                 svc_rbf['test_precision'].mean(),
                                                                 svc_rbf['test_recall'].mean(),
                                                                 svc_rbf['test_f1_score'].mean()],


                                    'Multi-Layer Perceptron':[mlp['test_accuracy'].mean(),
                                                                 mlp['test_precision'].mean(),
                                                                 mlp['test_recall'].mean(),
                                                                 mlp['test_f1_score'].mean()],

                                    'AdaBoost Classifer':[adb['test_accuracy'].mean(),
                                                                 adb['test_precision'].mean(),
                                                                 adb['test_recall'].mean(),
                                                                 adb['test_f1_score'].mean()],

                                    'Decision Tree':[dtr['test_accuracy'].mean(),
                                                     dtr['test_precision'].mean(),
                                                     dtr['test_recall'].mean(),
                                                     dtr['test_f1_score'].mean()],

                                    'Random Forest':[rfc['test_accuracy'].mean(),
                                                     rfc['test_precision'].mean(),
                                                     rfc['test_recall'].mean(),
                                                     rfc['test_f1_score'].mean()],

                                    'Gaussian Naive Bayes':[gnb['test_accuracy'].mean(),
                                                            gnb['test_precision'].mean(),
                                                            gnb['test_recall'].mean(),
                                                            gnb['test_f1_score'].mean()]},

                                    index=['Accuracy', 'Precision', 'Recall', 'F1 Score'])


    # Return models performance metrics scores data frame
    return models_scores_table

# Run models_evaluation function
models_scores = models_evaluation(np.array(fm_all).reshape(-1,1), actual_labels, 5)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from IPython.display import display
```

Classification Report of the Different Classifers

```
display(models_scores)
```

	Logistic Regression	Support Vector Classifier	Support Vector RBF Kernel	Multi-Layer Perceptron	AdaBoost Classifer	Decision Tree	Random Forest	Gaussian Naive Bayes
Accuracy	0.997727	0.997727	0.907125	0.797727	0.997727	0.997727	0.997727	0.993182
Precision	0.995556	0.995556	1.000000	0.595556	0.995556	0.995556	0.995556	1.000000
Recall	1.000000	1.000000	0.813636	0.600000	1.000000	1.000000	1.000000	0.986364

```
best_score_all=[]
best_score_accuracy  = ['Logistic Regression','Support Vector Classifier','AdaBoost Classifer', 'Decision Tree', 'Random Forest']
best_score_preci = ['Support Vector RBF Kernel','Gaussian Naive Bayes']
best_score_recall = ['Logistic Regression', 'Support Vector Classifier', 'AdaBoost Classifer', 'Decision Tree', 'Random Forest']
best_f1score = ['Logistic Regression', 'Support Vector Classifier', 'AdaBoost Classifer', 'Decision Tree', 'Random Forest']

best_score_all.append(best_score_accuracy)
best_score_all.append(best_score_preci)
best_score_all.append(best_score_recall)
best_score_all.append(best_f1score)

models_scores['Best Score'] = best_score_all
```

Best Classifers according to each cateogory

```
pd.options.display.max_columns = None
pd.set_option('display.max_columns', 100)
from IPython.core.display import HTML
display(HTML(models_scores.to_html()))
```

	Logistic Regression	Support Vector Classifier	Support Vector RBF Kernel	Multi-Layer Perceptron	AdaBoost Classifer	Decision Tree	Random Forest	Gaussian Naive Bayes	Best Score
Accuracy	0.997727	0.997727	0.907125	0.797727	0.997727	0.997727	0.997727	0.993182	[Logistic Regression, Support Vector Classifier, AdaBoost Classifer, Decision Tree, Random Forest]
Precision	0.995556	0.995556	1.000000	0.595556	0.995556	0.995556	0.995556	1.000000	[Support Vector RBF Kernel, Gaussian Naive Bayes]
Recall	1.000000	1.000000	0.813636	0.600000	1.000000	1.000000	1.000000	0.986364	[Logistic Regression, Support Vector Classifier, AdaBoost Classifer, Decision Tree, Random Forest]

It looks like ML methods perform better than setting the threshold, which means the feature works very well in disnguishing b/w blurred and non-blurred images

Blur Detection using Deep Learning techniques

Setting up the Uni Dataset Preprocessing and Access

```
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
```

```
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>=int(len(self.root_dir_list)/2):
      #out_image = cv2.resize(image,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
      #out_image = image
      label=0

    elif index >=int(len(self.root_dir_list)/4) and index <int(len(self.root_dir_list)/2):
      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<int(len(self.root_dir_list)/2):
      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
```

Creating Dataloaders for Training and Validation

```
#Pre processing the data
```

```
#Pre processing the data
normalize = transforms.Normalize(mean = [0.485,0.456,0.406],
                                 std = [0.229,0.224,0.225])
resize = transforms.Resize((256,256))



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

aerial_dataset_full = UniData(all_files_path,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']
              y = sample['label']

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

              label_predicted = model(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
```

Initializing the VGG16 Model

```
# 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, 256, 256))
```

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

```
epochs=10
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')
```

```
0 2.8030271530151367
10 0.6220592260360718
20 0.5158349871635437
30 0.7423640489578247
40 0.7370043992996216
50 0.49600934982299805
60 0.7398114204406738
70 0.7146323919296265
80 0.6555542945861816
90 0.6547068357467651
100 0.2574985921382904
110 0.6879836320877075
120 0.692436695098877
130 0.7015418410301208
140 0.7185297012329102
150 0.18911603093147278
160 0.6272095441818237
170 0.6623414754867554
Done Training
Epoch:  0 Train Loss:  0.77527278854271 Test Loss:  0.7077033526675646 Accuracy:  50.0
0 0.6887071132659912
10 0.5575665831565857
20 0.6637589931488037
30 0.6999661922454834
40 0.9511702060699463
50 0.6542500257492065
60 0.6725294589996338
70 0.7514493465423584
80 1.4947562217712402
90 0.7251157164573669
100 0.7318922281265259
110 0.6869311332702637
120 0.6401200294494629
130 0.6772067546844482
140 0.7064130306243896
150 0.6717066168785095
160 0.7590036392211914
170 0.7828446626663208
Done Training
Epoch:  1 Train Loss:  0.735066422167119 Test Loss:  0.7120464513468188 Accuracy:  47.72727272727273
0 0.719799280166626
10 0.7732001543045044
20 1.0099092721939087
30 0.3798700273036957
40 0.7348355054855347
50 0.6353421211242676
60 0.8350538015365601
70 0.8389166593551636
80 0.8621783256530762
90 0.7916948199272156
100 0.7674549221992493
110 0.5307213068008423
120 0.576384425163269
130 0.4711390435695648
140 0.7738372087478638
150 0.7791233062744141
160 0.6631245613098145
170 0.7778736352920532
Done Training
Epoch:  2 Train Loss:  0.6924040792313708 Test Loss:  0.7622970692640127 Accuracy:  59.09090909090909
0 0.6191563606262207
10 0.3838135004043579
20 0.6655979156494141
30 0.34934017062187195
40 0.3069117069244385
50 0.41744881868362427
60 0.5538212060928345
70 0.7209505438804626
```

```
80 0.7837321758270264
90 0.5267115831375122
100 0.6263555288314819
110 0.7447944283485413
120 0.6595875024795532
130 0.738926887512207
140 0.42704057693481445
150 0.6897527575492859
160 0.7621309757232666
170 0.9239047169685364
Done Training
Epoch:  3 Train Loss:  0.7186959479632002 Test Loss:  0.6824356567027957 Accuracy:  54.54545454545455
0 0.6718703508377075
10 0.6767399311065674
20 0.5128225684165955
30 0.7927548885345459
40 0.6610244512557983
50 0.6497194170951843
60 0.6686978936195374
```

VGG Model fails to learn well

```
160 0.5031174504661572
```

Initializing the ResNet-50 Model

```
140 0.5020640194418702
```

```
gpu_flag = torch.cuda.is_available()

#preloading Resnet18
model = models.resnet50(pretrained = True)

#append a new last layer
model.fc = nn.Linear(2048,num_classes)

# define loss function
criterion = nn.CrossEntropyLoss()

# setup SGD
optimizer = torch.optim.SGD(model.parameters(), lr=0.004, momentum=0.9)

step_lr_scheduler = lr_scheduler.StepLR(optimizer,5,.3)

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

print(model)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
100%                                97.8M/97.8M [00:00<00:00, 180MB/s]

True
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (4): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (5): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
```

```
epochs=3
best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0
train_loss_resnet,test_loss_resnet,accuracy_resnet = training_and_validation_loop(epochs,step_lr_scheduler,model,optimizer,aerial_train_loader,aerial_validation_loader,best_acc,best_model_wts,'resnet50')
```

```
    "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
0 0.7055091261863708
10 1.6464200019836426
20 0.04654381051659584
30 1.939384937286377
40 2.4572513103485107
50 17.550615310668945
60 0.016611192375421524
70 1.824987530708313
80 0.5396012663841248
90 3.022589683532715
100 0.022189592942595482
110 3.096186876296997
120 1.2905211448669434
130 0.24214239418506622
140 1.0656715631484985
150 5.514803409576416
160 1.4641908407211304
170 0.7917044758796692
Done Training
Epoch:  0 Train Loss:  1.493820184716998 Test Loss:  9.656762310522366 Accuracy:  57.95454545454545
0 1.0994724035263062
10 1.0280179977416992
20 0.2408321052789688
30 1.0278725624084473
40 0.7191191911697388
50 0.14520974457263947
60 0.3796749413013458
70 2.7448348999023438
80 0.6452761292457581
90 0.5227341055870056
100 0.3614877760410309
110 0.4799572825431824
120 1.2727408409118652
130 1.315704584121704
140 1.4151747226715088
150 1.3554267883300781
160 2.063218593597412
170 0.3278350532054901
Done Training
```

```
Epoch:  1 Train Loss:  0.8950671281121242 Test Loss:  1.3057816848267713 Accuracy:  56.81818181818182
0 0.7777076363563538
10 0.3182208836078644
20 0.8774762749671936
30 0.4227159321308136
40 0.45799556374549866
50 1.2464993000030518
60 1.5389583110809326
70 1.4036967754364014
80 1.2759300470352173
90 0.6806668043136597
100 0.824142336845398
110 0.4665899872779846
120 1.2191548347473145
130 0.9092104434967041
140 0.6405949592590332
150 0.495177686214447
160 0.27613067626953125
```

ResNet seems promising, but doesn't look like it would give as great results like the threshold / ML methods above

In our case, ML and Thresholding seems more promising than deep-learning in Blur-Detection

To-Do Tasks

- Include a module for Low SNR Images with the Blur module
- Include more blurs like out-of-focus, etc.

GCP Marker Detection using Deep-Learning (Incomplete)

```
#!ls 'drive/MyDrive/ML - Dataset #2'
```

```
img1 = cv2.imread('drive/MyDrive/ML - Dataset #2/M1_F1.3_0335.JPG')
```

```
gcp_location = pd.read_csv('drive/MyDrive/ML - Dataset #2/GCP_location.csv')
```

```
print(gcp_location)
```

```
                                FileName                                GCPLocation
0   TUV_Flight_M1_F1.3\M1_F1.3_0402.JPG   [[721.4228548467461, 252.04009341596282]]
1   TUV_Flight_M1_F1.3\M1_F1.3_0403.JPG      [[704.4642079038748, 655.56671594847]]
2   TUV_Flight_M1_F1.3\M1_F1.3_0404.JPG   [[684.9581536643873, 1065.7316003803392]]
3   TUV_Flight_M1_F1.3\M1_F1.3_0405.JPG   [[664.3215416922897, 1486.2287261023178]]
4   TUV_Flight_M1_F1.3\M1_F1.3_0406.JPG   [[648.0009406268432, 1907.9962121234441]]
5   TUV_Flight_M1_F1.3\M1_F1.3_0407.JPG   [[633.8990872070165, 2339.0637155008794]]
6   TUV_Flight_M1_F1.3\M1_F1.3_0408.JPG   [[625.4728358915672, 2786.5173383491406]]
7   TUV_Flight_M1_F1.3\M1_F1.3_0426.JPG   [[2136.9490982246843, 405.0045577681861]]
8   TUV_Flight_M1_F1.3\M1_F1.3_0427.JPG    [[2133.1756375824625, 810.819264192166]]
9   TUV_Flight_M1_F1.3\M1_F1.3_0428.JPG   [[2129.358431894121, 1230.6276037965329]]
10  TUV_Flight_M1_F1.3\M1_F1.3_0429.JPG  [[2119.8668041069864, 1643.2107130087516]]
11  TUV_Flight_M1_F1.3\M1_F1.3_0430.JPG   [[2103.760739295174, 2066.9467854913237]]
12  TUV_Flight_M1_F1.3\M1_F1.3_0431.JPG  [[2089.1120668421922, 2486.7911430365907]]
```

```
13  TUV_Flight_M1_F1.3\M1_F1.3_0432.JPG     [[2067.84553580918, 2927.186941524264]]
14  TUV_Flight_M1_F1.3\M1_F1.3_0458.JPG    [[2971.430534154526, 309.5995296913926]]
15  TUV_Flight_M1_F1.3\M1_F1.3_0459.JPG   [[2962.2968975858043, 720.6453725791202]]
16  TUV_Flight_M1_F1.3\M1_F1.3_0460.JPG    [[2962.49137374448, 1136.2657037875156]]
17  TUV_Flight_M1_F1.3\M1_F1.3_0461.JPG    [[2963.5133709632046, 1551.98818976519]]
18  TUV_Flight_M1_F1.3\M1_F1.3_0462.JPG   [[2970.315316494529, 1980.0574675215194]]
19  TUV_Flight_M1_F1.3\M1_F1.3_0463.JPG     [[2977.676191338294, 2408.82351048085]]
20  TUV_Flight_M1_F1.3\M1_F1.3_0464.JPG  [[2979.5093581110914, 2852.0093340792473]]
```

```
print(img1.shape)
```

```
(3000, 4000, 3)
```

Creating Dataset through Splitting into Samples which contain the marker and those which don't contain it-

Goal is so as to make the model detect the marker feature-point through different resolution-scales surrounded by different high and low-frequency features

(Need to optmize creating the Synthestic datatset some more)

```
!
```

```
data = pd.read_csv('./MLDataset1/gcp_locations.csv')
files = data.FileName
locs = data.GCPLocation

def if_img_exist_in_csv(imgfile):
    index = 0
    for file in files:
        id = file.index('\\')
        filename = file[id + 1:]
        if filename == imgfile:
            loc = locs[index].replace("[", "")
            loc = loc.replace("]", "")
            loc = np.fromstring(loc, dtype=float, sep=',')
            loc = np.reshape(loc, (-1, 2))
            loc = loc.astype(dtype=int)
            return loc
        index+=1
    return None

def ispositive_example(x1,y1,x2,y2,gcp_loc):
    if gcp_loc is None:
        return False
    for gcp_point in gcp_loc:
        if gcp_point[0]>x1 and gcp_point[0]<x2 and gcp_point[1]>y1 and gcp_point[1]<y2 :
            return True



exampleno = 0

debug = 0
for file in glob.glob("drive/MyDrive/ML - Dataset #2/*.JPG"):
    # if file == "M1_F1.3_0460.JPG":
```

```
    # if file == "M1_F1.3_0460.JPG":
    print(file)
    print(ok)
    img = cv2.imread(file)
    gcp_loc= if_img_exist_in_csv(file)

    grayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    graythres = cv2.inRange(grayimg, 242, 255)
    kernel = np.ones((3, 3), np.uint8)

    closing = cv2.morphologyEx(graythres, cv2.MORPH_CLOSE, kernel)
    opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)

    im2, contours, hierarchy = cv2.findContours(closing, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)   #find contours
    cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
```

Will be Using U-Net for Marker Localization

```
import torch
import torch.nn as nn
from torchvision import models

class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)


class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)
```

```
class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels , in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)


    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854e247a9c615f175f76fbb2e3a
        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)


class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

class UNet(nn.Module):
    def __init__(self, n_channels=1, n_classes=2, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
```

```
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        #print(x.shape)
        logits = self.outc(x)
        #print(logits.shape)
        #print(ok)
        return logits
```

Defining different loss-functions to train with as well as defining the training and validation loop

```
from collections import defaultdict
import torch.nn.functional as F
import copy
from sklearn.metrics import jaccard_similarity_score

def dice_loss(pred, target, smooth = 1.):
    pred = pred.contiguous()
    target = target.contiguous()

    intersection = (pred * target).sum(dim=2).sum(dim=2)

    loss = (1 - ((2. * intersection + smooth) / (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2) + smooth)))

    return loss.mean()

smooth = 1e-12

def jaccard_approx(pred, target, smooth=1e-12 ):
    #intersection = K.sum(y_true * y_pred, axis=[0, -1, -2])
    intersection = (pred * target).sum(dim=2).sum(dim=2)
    #sum_ = K.sum(y_true + y_pred, axis=[0, -1, -2])
    sum_ = (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2))

    jac = (intersection + smooth) / (sum_ - intersection + smooth)

    return jac.mean()

def calc_loss(pred, target, metrics, bce_weight=0.5):
    #print(pred.shape)
    #print(target.shape)
    bce = F.binary_cross_entropy_with_logits(pred, target)
    MSE = torch.nn.MSELoss()
    mse = MSE(pred,target)
    #pred = F.sigmoid(pred)

    #dice = dice_loss(pred, target)
    #loss = bce * bce_weight + dice * (1 - bce_weight)
    loss = 10*bce + 1000*mse
    #loss = torch.log(jaccard_approx(pred,target))
    #loss = jaccard_approx(pred, target )
    #loss = dice

    metrics['loss'] += loss.data.cpu().numpy() * target.size(0)
```

```
    return loss
from sklearn.metrics import mean_squared_error

def update_errors(true_values,predicted_values,size):
    """
    Calculate errors and standard deviation based on current
    true and predicted values.
    """
    err = [true - predicted for true, predicted in
                zip(true_values, predicted_values)]
    abs_err = [abs(error) for error in err]
    mean_err = sum(err) / size
    mean_abs_err = sum(abs_err) / size
    #std = np.array(err).std()



    #rmse = mean_squared_error(true_values, predicted_values, squared=False)

    return mean_abs_err

def print_metrics(metrics, epoch_samples, phase):
    outputs = []
    for k in metrics.keys():
        outputs.append("{}: {:4f}".format(k, metrics[k] / epoch_samples))

    print("{}: {}".format(phase, ", ".join(outputs)))


train_loss = []
test_loss = []
mean_abs_err_train = []
mean_abs_err_test = []

def train_model(model, optimizer, scheduler, num_epochs=10):
    #best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = 1e10

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        since = time.time()

      # Each epoch has a training and validation phase


        for param_group in optimizer.param_groups:
            print("LR", param_group['lr'])
        print('Started training')

        model.train()  # Set model to training mode

        true_values = []
        predicted_values = []


        metrics = defaultdict(float)
        epoch_samples = 0

        for inputs,labels in dataloader['train']:
            #print(dataloaders['train'])
```

```
    #print(dataloaders['train'])
    #print('Entered')
    inputs = inputs.to(device)
    #print(inputs['image'].shape)
    labels = labels.to(device)

    # zero the parameter gradients
    optimizer.zero_grad()

    # forward
    # track history if only in train

    outputs = model(inputs)
    #outputs = torch.max(outputs,torch.tensor([0.]).to(device))
    loss = calc_loss(outputs, labels, metrics)

    # backward + optimize only if in training phase
    loss.backward()
    optimizer.step()

    # statistics
    epoch_samples += inputs.size(0)


    # loop over batch samples
    for true, predicted in zip(labels, outputs):
        # integrate a density map to get no. of objects
        # note: density maps were normalized to 100 * no. of objects
        #       to make network learn better
        true_counts = torch.sum(true).item() / 100
        #predicted_counts = torch.sum(predicted).item() / 100

        p=torch.nn.functional.relu(predicted).cpu().detach().numpy() # contains a prediction result
        # normalize threshold to 0 or 1
        i = p >= 0.5
        p[i] = 1
        i = p < 0.5
        p[i] = 0
        labeled_array, num_features = lab(p, return_num=True)
        #predicted_counts = torch.sum(predicted).item() / 100
        predicted_counts = num_features

        # update current epoch results
        true_values.append(true_counts)
        predicted_values.append(predicted_counts)


print_metrics(metrics, epoch_samples, 'train')
epoch_loss = metrics['loss'] / epoch_samples
train_loss.append(epoch_loss)

mean_abs_err = update_errors(true_values,predicted_values,len(dataset['train']))
mean_abs_err_train.append(mean_abs_err)

#torch.save(model,f'/content/drive/My Drive/cars_latest/resunet50_epoch_{e}_1.pt')

#train_score, trs = calc_jacc_img_msk(model, inputs['image'].cpu().numpy(), inputs['segmented_mask'].cpu().numpy(), 8, 1)
scheduler.step()


#set model in evaluation mode
model.eval()
```

```
avg_loss = 0
true_values = []
predicted_values = []

metrics = defaultdict(float)
epoch_samples=0

for inputs,labels in dataloader['valid']:

    #print(dataloaders['train'])
    inputs = inputs.to(device)
    #print(inputs['image'].shape)
    labels = labels.to(device)

    outputs = model(inputs)
    #outputs = torch.max(outputs,torch.tensor([0.]).to(device))

    loss = calc_loss(outputs, labels, metrics)
    epoch_samples += inputs.size(0)

    # loop over batch samples
    for true, predicted in zip(labels, outputs):
        # integrate a density map to get no. of objects
        # note: density maps were normalized to 100 * no. of objects
        #       to make network learn better
        true_counts = torch.sum(true).item() / 100
        #predicted_counts = torch.sum(predicted).item() / 100

        p=torch.nn.functional.relu(predicted).cpu().detach().numpy() # contains a prediction result
        # normalize threshold to 0 or 1
        i = p >= 0.5
        p[i] = 1
        i = p < 0.5
        p[i] = 0
        labeled_array, num_features = lab(p, return_num=True)
        #predicted_counts = torch.sum(predicted).item() / 100
        predicted_counts = num_features


        # update current epoch results
        true_values.append(true_counts)
        predicted_values.append(predicted_counts)

print_metrics(metrics, epoch_samples, 'val')
epoch_loss = metrics['loss'] / epoch_samples
test_loss.append(epoch_loss)


mean_abs_err = update_errors(true_values,predicted_values,len(dataset['valid']))
mean_abs_err_test.append(mean_abs_err)

print(true_values)
print(predicted_values)


#print("\n")
print("Epoch: ", epoch, "Train Loss: ", train_loss[-1], "Test Loss: ", test_loss[-1], "Train MAE: ", mean_abs_err_train[-1], "Test MAE: ", mean_abs_err_test[-1])

time_elapsed = time.time() - since
print('{:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
print("\n")
```

```
      #print(ok)
      if (epoch%10==0) or (epoch==(num_epochs-1)):

        torch.save({
              'epoch': epoch,
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_loss[-1],
              }, f'/content/drive/My Drive/Input_dataset/epoch_{epoch}_trainmae_{mean_abs_err_train[-1]}_testmae_{mean_abs_err_test[-1]}_unet.pt')



  #print('Best val loss: {:4f}'.format(best_loss))

  # load best model weights
  #model.load_state_dict(best_model_wts)
  return model
```

Reading images and Extracting SuperPoint Keypoints and Descriptors from each image

15 cells hidden

Loading and Initialing the SuperPoint Pretrained Network

[ ] *1 cell hidden*

Now Extracting Keypoints and Descriptors from all images and storing them

[ ] *7 cells hidden*

Image Matching (Robust) through RANSAC and Homography Matrix computation

[ ] *8 cells hidden*

Auto-Selection/Ordering of Images (Complete)

[ ] *20 cells hidden*

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