D2-Net (Deep learning Feature Matching)

It follows detect and describe approach. Thus, Feature Detecttion and description are done at the same time. It has a VGG-16 network to generate detections and descriptors.

In [ ]:

```
%cd /content/drive/My Drive/Aero2Astro/D2-Net/d2-net
```

/content/drive/My Drive/Aero2Astro/D2-Net/d2-net

In [ ]:

```
import cv2

import matplotlib.pyplot as plt

import numpy as np

import os

from PIL import Image

from skimage.feature import match_descriptors
from skimage.measure import ransac
from skimage.transform import ProjectiveTransform
```

In [ ]:

```
!mkdir models
!wget https://dsmn.ml/files/d2-net/d2_ots.pth -O models/d2_ots.pth
!wget https://dsmn.ml/files/d2-net/d2_tf.pth -O models/d2_tf.pth
!wget https://dsmn.ml/files/d2-net/d2_tf_no_phototourism.pth -O models/d2_tf_no_phototo
urism.pth
```

Don't forget to run feature extraction before running this script

python extract_features.py --image_list_file image_list_qualitative.txt

In [ ]:

```
!python extract_features.py --image_list_file testimages.txt
```

Namespace(image_list_file='testimages.txt', max_edge=1600, max_sum_edges=2
800, model_file='models/d2_tf.pth', multiscale=False, output_extension='.d 2-
net', output_type='npz', preprocessing='caffe', use_relu=True)
100% 2/2 [00:12<00:00, 6.10s/it]

Change the pair index here (possible values: 1, 2 or 3)

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

In [ ]:

```
pair_idx = 2
assert(pair_idx in [1, 2, 3])
```

Loading the features

In [ ]:

```
pair_path = "/content/drive/My Drive/Aero2Astro/D2-Net/d2-net/qualitative/images/pair_
4"
```

In [ ]:

```
image1 = np.array(Image.open(os.path.join(pair_path, '1.jpg')))
image2 = np.array(Image.open(os.path.join(pair_path, '2.jpg')))
```

In [ ]:

```
feat1 = np.load(os.path.join(pair_path, '1.jpg.d2-net'))
feat2 = np.load(os.path.join(pair_path, '2.jpg.d2-net'))
```

Mutual nearest neighbors matching

In [ ]:

```
matches = match_descriptors(feat1['descriptors'], feat2['descriptors'], cross_check=Tru
e)
```

In [ ]:

```
print('Number of raw matches: %d.' % matches.shape[0])
```

Number of raw matches: 379.

It generates more matches than SIFT on the same images.

Plotting all raw matches

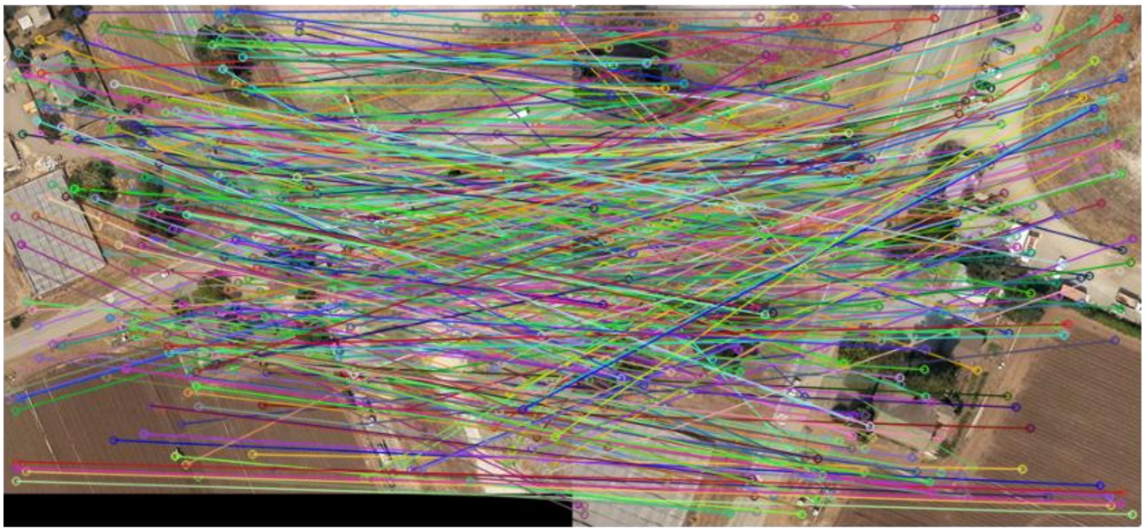
In [ ]:

```
n_matches = matches.shape[0]
raw_keypoints_left = [cv2.KeyPoint(point[0], point[1], 1) for point in keypoints_left]
raw_keypoints_right = [cv2.KeyPoint(point[0], point[1], 1) for point in keypoints_right
]
rawmatches = [cv2.DMatch(idx, idx, 1) for idx in range(n_matches)]
```

In [ ]:

```
image4 = cv2.drawMatches(image1, raw_keypoints_left, image2, raw_keypoints_right, rawma
tches, None)

plt.figure(figsize=(15, 15))
plt.imshow(image4)
plt.axis('off')
plt.show()
```



Homography fitting

In [ ]:

```
keypoints_left = feat1['keypoints'][matches[:, 0], : 2]
keypoints_right = feat2['keypoints'][matches[:, 1], : 2]
np.random.seed(0)
model, inliers = ransac(
    (keypoints_left, keypoints_right),
    ProjectiveTransform, min_samples=4,
    residual_threshold=4, max_trials=10000
)
n_inliers = np.sum(inliers)
print('Number of inliers: %d.' % n_inliers)
```

Number of inliers: 13.

Plotting only Inlier Matches

In [ ]:

```
inlier_keypoints_left = [cv2.KeyPoint(point[0], point[1], 1) for point in keypoints_left[inliers]]
inlier_keypoints_right = [cv2.KeyPoint(point[0], point[1], 1) for point in keypoints_right[inliers]]
placeholder_matches = [cv2.DMatch(idx, idx, 1) for idx in range(n_inliers)]
image3 = cv2.drawMatches(image1, inlier_keypoints_left, image2, inlier_keypoints_right, placeholder_matches, None)

plt.figure(figsize=(15, 15))
plt.imshow(image3)
plt.axis('off')
plt.show()
```