In [ ]:

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

In [17]:

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving superpoint_v1.pth to superpoint_v1.pth
```

In [15]:

```
%matplotlib inline
import glob
import numpy as np
import os
import time

import cv2
import torch
# Jet colormap for visualization.
from IPython.display import display, clear_output
import matplotlib.pyplot as plt

myjet = np.array([[0.        , 0.        , 0.5       ],
                  [0.        , 0.        , 0.99910873],
                  [0.        , 0.37843137, 1.        ],
                  [0.        , 0.83333333, 1.        ],
                  [0.30044276, 1.        , 0.66729918],
                  [0.66729918, 1.        , 0.30044276],
                  [1.        , 0.90123457, 0.        ],
                  [1.        , 0.48002905, 0.        ],
                  [0.99910873, 0.07334786, 0.        ],
                  [0.5       , 0.        , 0.        ]])
```

In [20]:

```
class SuperPointNet(torch.nn.Module):
  """ Pytorch definition of SuperPoint Network. """
  def __init__(self):
    super(SuperPointNet, self).__init__()
    self.relu = torch.nn.ReLU(inplace=True)
    self.pool = torch.nn.MaxPool2d(kernel_size=2, stride=2)
    c1, c2, c3, c4, c5, d1 = 64, 64, 128, 128, 256, 256
    # Shared Encoder.
    self.conv1a = torch.nn.Conv2d(1, c1, kernel_size=3, stride=1, padding=1)
    self.conv1b = torch.nn.Conv2d(c1, c1, kernel_size=3, stride=1, padding=1)
    self.conv2a = torch.nn.Conv2d(c1, c2, kernel_size=3, stride=1, padding=1)
    self.conv2b = torch.nn.Conv2d(c2, c2, kernel_size=3, stride=1, padding=1)
    self.conv3a = torch.nn.Conv2d(c2, c3, kernel_size=3, stride=1, padding=1)
    self.conv3b = torch.nn.Conv2d(c3, c3, kernel_size=3, stride=1, padding=1)
    self.conv4a = torch.nn.Conv2d(c3, c4, kernel_size=3, stride=1, padding=1)
    self.conv4b = torch.nn.Conv2d(c4, c4, kernel_size=3, stride=1, padding=1)
    # Detector Head.
    self.convPa = torch.nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
    self.convPb = torch.nn.Conv2d(c5, 65, kernel_size=1, stride=1, padding=0)
    # Descriptor Head.
    self.convDa = torch.nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
    self.convDb = torch.nn.Conv2d(c5, d1, kernel_size=1, stride=1, padding=0)

  def forward(self, x):
    # Shared Encoder.
    x = self.relu(self.conv1a(x))
    x = self.relu(self.conv1b(x))
    x = self.pool(x)
    x = self.relu(self.conv2a(x))
    x = self.relu(self.conv2b(x))
    x = self.pool(x)
    x = self.relu(self.conv3a(x))
    x = self.relu(self.conv3b(x))
    x = self.pool(x)
    x = self.relu(self.conv4a(x))
    x = self.relu(self.conv4b(x))
    # Detector Head.
    cPa = self.relu(self.convPa(x))
    semi = self.convPb(cPa)
    # Descriptor Head.
    cDa = self.relu(self.convDa(x))
    desc = self.convDb(cDa)
    dn = torch.norm(desc, p=2, dim=1) # Compute the norm.
    desc = desc.div(torch.unsqueeze(dn, 1)) # Divide by norm to normalize.
    return semi, desc
```

```
class SuperPointFrontend(object):
  def __init__(self, weights_path, nms_dist, conf_thresh, nn_thresh,
               cuda=False):
    self.name = 'SuperPoint'
    self.cuda = cuda
    print(cuda)
    self.nms_dist = nms_dist
    self.conf_thresh = conf_thresh
    self.nn_thresh = nn_thresh # L2 descriptor distance for good match.
    self.cell = 8 # Size of each output cell. Keep this fixed.
    self.border_remove = 4 # Remove points this close to the border.

    # Load the network in inference mode.
    self.net = SuperPointNet()
    # if cuda:
    #   # Train on GPU, deploy on GPU.
    #   self.net.load_state_dict(torch.load(weights_path))
    #   self.net = self.net.cuda()
    # else:
      # Train on GPU, deploy on CPU.
    self.net.load_state_dict(torch.load(weights_path,
                           map_location=lambda storage, loc: storage))
    self.net.eval()


  def nms_fast(self, in_corners, H, W, dist_thresh):
    grid = np.zeros((H, W)).astype(int) # Track NMS data.
    inds = np.zeros((H, W)).astype(int) # Store indices of points.
    # Sort by confidence and round to nearest int.
    inds1 = np.argsort(-in_corners[2,:])
    corners = in_corners[:,inds1]
    rcorners = corners[:2,:].round().astype(int) # Rounded corners.
    # Check for edge case of 0 or 1 corners.
    if rcorners.shape[1] == 0:
      return np.zeros((3,0)).astype(int), np.zeros(0).astype(int)
    if rcorners.shape[1] == 1:
      out = np.vstack((rcorners, in_corners[2])).reshape(3,1)
      return out, np.zeros((1)).astype(int)
    # Initialize the grid.
    for i, rc in enumerate(rcorners.T):
      grid[rcorners[1,i], rcorners[0,i]] = 1
      inds[rcorners[1,i], rcorners[0,i]] = i
    # Pad the border of the grid, so that we can NMS points near the border.
    pad = dist_thresh
    grid = np.pad(grid, ((pad,pad), (pad,pad)), mode='constant')
    # Iterate through points, highest to lowest conf, suppress neighborhood.
    count = 0
    for i, rc in enumerate(rcorners.T):
      # Account for top and left padding.
      pt = (rc[0]+pad, rc[1]+pad)
      if grid[pt[1], pt[0]] == 1: # If not yet suppressed.
        grid[pt[1]-pad:pt[1]+pad+1, pt[0]-pad:pt[0]+pad+1] = 0
        grid[pt[1], pt[0]] = -1
        count += 1
    # Get all surviving -1's and return sorted array of remaining corners.
    keepy, keepx = np.where(grid==-1)
    keepy, keepx = keepy - pad, keepx - pad
    inds_keep = inds[keepy, keepx]
    out = corners[:, inds_keep]
    values = out[-1, :]
    inds2 = np.argsort(-values)
    out = out[:, inds2]
    out_inds = inds1[inds_keep[inds2]]
    return out, out_inds

  def run(self, img):
    assert img.ndim == 2, 'Image must be grayscale.'
    assert img.dtype == np.float32, 'Image must be float32.'
    H, W = img.shape[0], img.shape[1]
    inp = img.copy()
    inp = (inp.reshape(1, H, W))
    inp = torch.from_numpy(inp)
    inp = torch.autograd.Variable(inp).view(1, 1, H, W)
    if self.cuda:
      inp = inp.cuda()
    # Forward pass of network.
    outs = self.net.forward(inp)
    semi, coarse_desc = outs[0], outs[1]
    # Convert pytorch -> numpy.
    semi = semi.data.cpu().numpy().squeeze()
    # --- Process points.
    dense = np.exp(semi) # Softmax.
    dense = dense / (np.sum(dense, axis=0)+.00001) # Should sum to 1.
    # Remove dustbin.
```

```python
    nodust = dense[:-1, :, :]
    # Reshape to get full resolution heatmap.
    Hc = int(H / self.cell)
    Wc = int(W / self.cell)
    nodust = nodust.transpose(1, 2, 0)
    heatmap = np.reshape(nodust, [Hc, Wc, self.cell, self.cell])
    heatmap = np.transpose(heatmap, [0, 2, 1, 3])
    heatmap = np.reshape(heatmap, [Hc*self.cell, Wc*self.cell])
    xs, ys = np.where(heatmap >= self.conf_thresh) # Confidence threshold.
    if len(xs) == 0:
      return np.zeros((3, 0)), None, None
    pts = np.zeros((3, len(xs))) # Populate point data sized 3xN.
    pts[0, :] = ys
    pts[1, :] = xs
    pts[2, :] = heatmap[xs, ys]
    pts, _ = self.nms_fast(pts, H, W, dist_thresh=self.nms_dist) # Apply NMS.
    inds = np.argsort(pts[2,:])
    pts = pts[:,inds[::-1]] # Sort by confidence.
    # Remove points along border.
    bord = self.border_remove
    toremoveW = np.logical_or(pts[0, :] < bord, pts[0, :] >= (W-bord))
    toremoveH = np.logical_or(pts[1, :] < bord, pts[1, :] >= (H-bord))
    toremove = np.logical_or(toremoveW, toremoveH)
    pts = pts[:, ~toremove]
    # --- Process descriptor.
    D = coarse_desc.shape[1]
    if pts.shape[1] == 0:
      desc = np.zeros((D, 0))
    else:
      # Interpolate into descriptor map using 2D point locations.
      samp_pts = torch.from_numpy(pts[:2, :].copy())
      samp_pts[0, :] = (samp_pts[0, :] / (float(W)/2.)) - 1.
      samp_pts[1, :] = (samp_pts[1, :] / (float(H)/2.)) - 1.
      samp_pts = samp_pts.transpose(0, 1).contiguous()
      samp_pts = samp_pts.view(1, 1, -1, 2)
      samp_pts = samp_pts.float()
      if self.cuda:
        samp_pts = samp_pts.cuda()
      desc = torch.nn.functional.grid_sample(coarse_desc, samp_pts)
      desc = desc.data.cpu().numpy().reshape(D, -1)
      desc /= np.linalg.norm(desc, axis=0)[np.newaxis, :]
    return pts, desc, heatmap
```

In [24]:

```
import argparse
if __name__ == '__main__':

        # Parse command line arguments.
        parser = argparse.ArgumentParser(description='PyTorch SuperPoint Demo.')
        # parser.add_argument('input', type=str, default='',
        #   help='Image directory or movie file or "camera" (for webcam).')
        parser.add_argument('--weights_path', type=str, default='superpoint_v1.pth',
          help='Path to pretrained weights file (default: superpoint_v1.pth).')
        parser.add_argument('--img_glob', type=str, default='*.png',
          help='Glob match if directory of images is specified (default: \'*.png\').')
        parser.add_argument('--skip', type=int, default=1,
          help='Images to skip if input is movie or directory (default: 1).')
        parser.add_argument('--show_extra', action='store_true',
          help='Show extra debug outputs (default: False).')
        parser.add_argument('--H', type=int, default=120,
          help='Input image height (default: 120).')
        parser.add_argument('--W', type=int, default=160,
          help='Input image width (default:160).')
        parser.add_argument('--display_scale', type=int, default=2,
          help='Factor to scale output visualization (default: 2).')
        parser.add_argument('--min_length', type=int, default=2,
          help='Minimum length of point tracks (default: 2).')
        parser.add_argument('--max_length', type=int, default=5,
          help='Maximum length of point tracks (default: 5).')
        parser.add_argument('--nms_dist', type=int, default=4,
          help='Non Maximum Suppression (NMS) distance (default: 4).')
        parser.add_argument('--conf_thresh', type=float, default=0.015,
          help='Detector confidence threshold (default: 0.015).')
        parser.add_argument('--nn_thresh', type=float, default=0.7,
          help='Descriptor matching threshold (default: 0.7).')
        parser.add_argument('--camid', type=int, default=0,
          help='OpenCV webcam video capture ID, usually 0 or 1 (default: 0).')
        parser.add_argument('--waitkey', type=int, default=1,
          help='OpenCV waitkey time in ms (default: 1).')
        parser.add_argument('--cuda', action='store_true',
          help='Use cuda GPU to speed up network processing speed (default: False)')
        parser.add_argument('--no_display', action='store_true',
          help='Do not display images to screen. Useful if running remotely (default: False).')
        parser.add_argument('--write', action='store_true',
          help='Save output frames to a directory (default: False)')
```

```
parser.add_argument('--write_dir', type=str, default='tracker_outputs/',
  help='Directory where to write output frames (default: tracker_outputs/).')
opt = parser.parse_args()
print(opt)


#print('==> Loading pre-trained network.')
# This class runs the SuperPoint network and processes its outputs.
fe = SuperPointFrontend(weights_path=opt.weights_path,
                      nms_dist=opt.nms_dist,
                      conf_thresh=opt.conf_thresh,
                      nn_thresh=opt.nn_thresh,
                      cuda=opt.cuda)
print('==> Successfully loaded pre-trained network.')

font = cv2.FONT_HERSHEY_DUPLEX
font_clr = (255, 255, 255)
font_pt = (4, 12)
font_sc = 0.4

path = 'IX-11-01917_0004_0005.JPG'
path2 = 'IX-11-01917_0004_0006.JPG'
input_image = cv2.imread(path2)
grayimg = cv2.cvtColor(input_image, cv2.COLOR_RGB2GRAY)
size = [600, 400]
input_image = cv2.resize(grayimg, (size[1], size[0]),
                       interpolation=cv2.INTER_AREA)
input_image = input_image.astype('float')/255.0
img = input_image.astype('float32')

start1 = time.time()
pts, desc, heatmap = fe.run(img)
#print(pts[0].astype(np.uint8))
#print(pts[2])
end1 = time.time()

out1 = (np.dstack((img, img, img)) * 255.).astype('uint8')
out2 = (np.dstack((img, img, img)) * 255.).astype('uint8')

for pt in pts.T:
        pt1 = (int(round(pt[0])), int(round(pt[1])))
        cv2.circle(out2, pt1, 1, (0, 255, 0), -1, lineType=16)
cv2.putText(out2, 'Raw Point Detections', font_pt, font, font_sc, font_clr, lineType=16)
cv2.imwrite('out.png', out2)

results = []
keypoints_x = pts[0]
keypoints_y = pts[1]
scores = pts[2]
pt_list = []
scores_list = []
for j in range(len(keypoints_x)):
        pt_list.append(int(keypoints_x[j]))
        pt_list.append(int(keypoints_y[j]))
        pt_list.append(2)
        scores_list.append(round(scores[j], 3))

print(len(keypoints_x))
score_final = float(np.mean(np.array(scores_list)))
result = {"confidence":score_final, "category_id":"pose61", "keypoints":pt_list, "area":0}
results.append(result)

#print(results)
```

```
usage: ipykernel_launcher.py [-h] [--weights_path WEIGHTS_PATH]
                             [--img_glob IMG_GLOB] [--skip SKIP]
                             [--show_extra] [--H H] [--W W]
                             [--display_scale DISPLAY_SCALE]
                             [--min_length MIN_LENGTH]
                             [--max_length MAX_LENGTH] [--nms_dist NMS_DIST]
                             [--conf_thresh CONF_THRESH]
                             [--nn_thresh NN_THRESH] [--camid CAMID]
                             [--waitkey WAITKEY] [--cuda] [--no_display]
                             [--write] [--write_dir WRITE_DIR]
ipykernel_launcher.py: error: unrecognized arguments: -f /root/.local/share/jupyter/runtime/kernel-37b1704c-ea26-4c6b
-990d-9cf690584e76.json
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2890: UserWarning: To exit: use 'exit', 'qui
t', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```