```
# USAGE
# python myimagestitch.py --images folderToOpen --images2 folderToOpen --images3 folderToOpen --extractor method

import numpy as np
import matplotlib.pyplot as plt
import argparse
import imutils
from imutils import paths
import cv2
#select sift/surf/orb/brisk
feature_extractor = 'orb'


ap = argparse.ArgumentParser()

#path in the images
ap.add_argument("-i", "--images", type=str, required=True,
	help="path to input directory of images to stitch")
ap.add_argument("-e", "--images2", type=str, required=True,
	help="path to input directory of images to stitch")
ap.add_argument("-f", "--images3", type=str, required=True,
	help="path to input directory of images to stitch")

#ipath the method
ap.add_argument("-d", "--extractor", type=str, required=False,
help="method of the feature extractor")

args = vars(ap.parse_args())

if args["extractor"]:
        feature_extractor = args["extractor"]
# path the images
print("[INFO] loading images...")
#read the folder to get the image
imagePaths = sorted(list(paths.list_images(args["images"])))
images = []
images = [cv2.imread(imagePath) for imagePath in imagePaths]

image2Paths = sorted(list(paths.list_images(args["images2"])))
images2 = []
images2 = [cv2.imread(image2Path) for image2Path in image2Paths]


image3Paths = sorted(list(paths.list_images(args["images3"])))
images3 = []
images3 = [cv2.imread(image3Path) for image3Path in image3Paths]


//find the keypoints
def detectAndDescribe(image, method):
        if method == 'sift':
            descriptor = cv2.xfeatures2d.SIFT_create()
        elif method == 'surf':
            descriptor = cv2.xfeatures2d.SURF_create()
        elif method == 'brisk':
            descriptor = cv2.BRISK_create()
        elif method == 'orb':
            descriptor = cv2.ORB_create()
        (kps, features) = descriptor.detectAndCompute(image, None)
        return (kps, features)

def matchKeyPointsKNN(featuresA, featuresB, ratio, method):
        bf = createMatcher(method, crossCheck=False)
        #find match and do matching
        rawMatches = bf.knnMatch(featuresA, featuresB, 2)
        print("Raw matches (knn):", len(rawMatches))
        matches = []


        for m,n in rawMatches:
            if m.distance < n.distance * ratio:
                matches.append(m)
        return matches

def matchKeyPointsBF(featuresA, featuresB, method):
        bf = createMatcher(method, crossCheck=True)

        best_matches = bf.match(featuresA,featuresB)

        #sorting

        rawMatches = sorted(best_matches, key = lambda x:x.distance)

        print("Raw matches (Brute force):", len(rawMatches))

        return rawMatches


def getHomography(kpsA, kpsB, featuresA, featuresB, matches, reprojThresh):

        # change keypoints to array

        kpsA = np.float32([kp.pt for kp in kpsA])

        kpsB = np.float32([kp.pt for kp in kpsB])


        if len(matches) > 4:


            # build RANSAC

            ptsA = np.float32([kpsA[m.queryIdx] for m in matches])

            ptsB = np.float32([kpsB[m.trainIdx] for m in matches])

            (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,

                reprojThresh)


            return (matches, H, status)

        else:

            return None

print("no. of images set1",len(images))

print("no. of images set2",len(images2))

print("no. of images set3",len(images3))
```

```
for i in range(len(images)-1):
        kpsA, featuresA = detectAndDescribe(images[i], feature_extractor)
        kpsB, featuresB = detectAndDescribe(images[i+1], feature_extractor)
        #match with knn
        matches = matchKeyPointsKNN(featuresA, featuresB, ratio = 0.5, method =
feature_extractor)
        #draw matching line
        img = cv2.drawMatches(images[i], kpsA, images[i+1], kpsB, matches[:100],
        None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

        #output process of matching file
        cv2.imwrite('output/matching1/'+feature_extractor+str(i)+'.jpg',img)

for i in range(len(images2)-1):
          #choose extractor
          kpsA, featuresA = detectAndDescribe(images2[i], feature_extractor)
          kpsB, featuresB = detectAndDescribe(images2[i+1], feature_extractor)
          #match the images with K-Nearest Neighbor method of the matching key point
          matches = matchKeyPointsKNN(featuresA, featuresB, ratio = 0.5, method =
feature_extractor)
          #draw matching line
          img = cv2.drawMatches(images2[i], kpsA, images2[i+1], kpsB, matches[:100],
          None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
          #plot and show the image
          #plt.imshow(img)
          #plt.show()
```

```
    #output process of matching file
    cv2.imwrite('output/matching2/'+feature_extractor+str(i)+'.jpg',img)

for i in range(len(images3)-1):
    #choose extractor
    kpsA, featuresA = detectAndDescribe(images3[i], feature_extractor)
    kpsB, featuresB = detectAndDescribe(images3[i+1], feature_extractor)
    #match the images with K-Nearest Neighbor method of the matching key point
    matches = matchKeyPointsKNN(featuresA, featuresB, ratio = 0.5, method = feature_extractor)
    #draw matching line
    img = cv2.drawMatches(images3[i], kpsA, images3[i+1], kpsB, matches[:100],
    None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    #plot and show the image
    #plt.imshow(img)
    #plt.show()
    #output process of matching file
    cv2.imwrite('output/matching3/'+feature_extractor+str(i)+'.jpg',img)



print("[INFO] stitching images...")
stitcher = cv2.createStitcher() if imutils.is_cv3() else cv2.Stitcher_create()
(status, stitched1) = stitcher.stitch(images)
(status2, stitched2) = stitcher.stitch(images2)
(status3, stitched3) = stitcher.stitch(images3)
```

```
# status != 0 means error

if status == 0:
        cv2.imwrite("output/stitched1.jpg", stitched1)
else:
        print("[INFO] Failed ({})".format(status))

if status2 == 0:
        #uncrop output
        cv2.imwrite("output/stitched2.jpg", stitched2)
else:
        print("[INFO] Failed ({})".format(status2))

if status3 == 0:
                # uncrop output
        cv2.imwrite("output/stitched3.jpg", stitched3)
else:
        print("[INFO] Failed ({})".format(status3))
stitches = [stitched1, stitched2, stitched3]

(status4, stitched4) = stitcher.stitch(stitches)
if status4 ==0:
        cv2.imwrite("output/output.jpg", stitched4)
        cv2.imshow("Stitched", stitched4)
else:
        print("[INFO] Failed ({})".format(status4))
```

```
cv2.waitKey(0)
```