Ex. 1 - Brute-force Matching

Implement the brute-force matching procedure between a query and a database image.

```
# Distance matrix (rows correspond to query descriptors while columns to database descriptors)
dist_matrix = np.zeros((len(query_descriptors), len(db_descriptors)))
for i, desQ in enumerate(query_descriptors):
  for j, desD in enumerate(db_descriptors):
    dist_matrix[i, j] = np.linalg.norm(desQ-desD)
# More efficient implementation to compute pairwise distances
# import sklearn.metrics
# dist_matrix = sklearn.metrics.pairwise_distances(query_descriptors, db_descriptors)

# Find the closest database descriptor for each query descriptor
# and the corresponding distance
min_distances = dist_matrix.min(axis = 1)
db_descriptors_idx = np.argmin(dist_matrix, axis=1)

# Discard matches above the threshold
thrs = 80
query_descriptors_idx = np.where(min_distances <= thrs)[0]

# Select keypoints, descriptors and indices of retrieved matches

# Query kps and des
query_keypoints = [query_keypoints[idx] for idx in query_descriptors_idx]
query_descriptors = [query_descriptors[idx] for idx in query_descriptors_idx]

# DB kps and des
db_descriptors_idx = [db_descriptors_idx[idx] for idx in query_descriptors_idx]
db_keypoints = [db_keypoints[idx] for idx in db_descriptors_idx]
db_descriptors = [db_descriptors[idx] for idx in db_descriptors_idx]

min_distances = [min_distances[idx] for idx in query_descriptors_idx]

print(f"Number of matches: {len(min_distances)}")
# Print Euclidean distance, query descriptor index and database descriptor index of retrieved matches
for k in range(len(min_distances)):
     print(f"Query Descriptor n. {query_descriptors_idx[k]}, Database Descriptor n. {db_descriptors_idx[k]}, Euclidean distance: {min_distances[k]:.2f}.")
```

Ex. 2 - Ratio test

implement a K-NN matcher to get the K best matches (rather than the closest one). We set K=2 in order to apply the ratio test as reported by D. Lowe.

```
# Detect and compute keypoints and descriptors
query_keypoints, query_descriptors = sift.detectAndCompute(query_image, None)
db_keypoints, db_descriptors = sift.detectAndCompute(db_image, None)

# Pairwise distances
dist_matrix = sklearn.metrics.pairwise_distances(query_descriptors, db_descriptors)
```

```python
## K-NN matcher (K=2)
# For each query descriptor find the indices and distances
# of the 2 closest database descriptors.
k = 2
distances = np.partition(dist_matrix, kth=k, axis=1)[:, :k]
db_idx = np.argpartition(dist_matrix, kth=k, axis=1)[:, :k]

# Apply ratio test
min_dist = []
min_query_idx = []
min_db_idx = []
for k in range(distances.shape[0]):
  if distances[k][0] < 0.70 * distances[k][1]:
    min_dist.append(distances[k][0])
    min_query_idx.append(k)
    min_db_idx.append(db_idx[k][0])

print(f"Number of query keypoints: {len(query_keypoints)}")
print(f"Number of database keypoints: {len(db_keypoints)}")
print(f"Number of matches: {len(min_dist)}")

# Print first 10 matches
nMatches = 10
for k in range(nMatches):
  print(f"Euclidean distance: {min_dist[k]:.2f}, Query Descriptor: n. {min_query_idx[k]},
Database Descriptor: n.{min_db_idx[k]}")
```

Ex. 3 - Brute-force Matcher

Use the brute-force matcher and, rather that using the ratio test, set a threshold value (e.g., 100) and discard all the matches above that threshold.

```python
# Brute-force matcher
bf = cv2.BFMatcher_create(cv2.NORM_L2)

# Finds the best match for each descriptor from a query set.
matches = bf.match(query_descriptors, db_descriptors)

# Order matches
ordered_matches = sorted(matches, key = lambda m:m.distance)

# Discard matches above the threshold
thrs = 100

matched_des = [ordered_matches[idx] for idx in range(len(ordered_matches)) \
               if ordered_matches[idx].distance <= thrs]


print(f"Number of matches: {len(matched_des)}")
print(f"Min distance: {matched_des[0].distance}, Max distance: {matched_des[-
1].distance}.")

fig = plt.figure(figsize=(15,10))
draw_params = dict(matchColor=(0, 255, 0), singlePointColor=(255, 0, 0), flags=0)
out_matches = cv2.drawMatches(query_image, query_keypoints, database_image, db_keypoints,
matched_des[:50], None, **draw_params)
plt.imshow(out_matches)
plt.show()
```