Lab 3 & 4 Fingerprint Recognition

1. Classical fingerprint recognition

During the exercise we will use the simplest approach - feature based matching.

1. Get the database: http://bias.csr.unibo.it/fvc2002/download.asp (DB1). Take a quick look at the structure of dataset. There are 10 different classes, 8 images for each one, so 80 images in total. Remember the name's convention: X_Y.tif, where X - class name (101-110), Y - image index inside class (1 - 8). In our case we will use images with index 1 (i.e. 101_1, 102_1, etc.) as reference images, saved in the database (like a police database). Other images (70 in total) we will try to match with our database (so they will be "suspects' fingerprints").
2. Note that we need only features to match, not the entire images, so we can divide our script into two phases. Firstly we read all images, extract features from them and save features into two dictionaries ("base" and "suspects"). Secondly we match each "suspects' fingerprint" with these in base to find out who is who.
3. Part 1 - feature extraction:
    a. create two dictionaries: for base and for suspects;
    b. loop over images, read them, convert to grayscale;
    c. run SIFT and ORB feature detection and description. Note that for SIFT you need opencv-contrib (this is not a free algorithm) - it is installed on lab computers (in *venv* inside the Biometrics folder);
    d. create a simple dictionary to save extracted feature, e.g.:

```
feat_dict = {'sift': sift_des, 'orb': orb_des}
```

    e. determine if the current image belongs to the "base" or "suspects" and save *feat_dict* in the proper dictionary.
    NOTE. You should use different keys for base and suspect dictionaries. In first it should be just class id, while in second - class with image index. You can get them by splitting image name, e.g.:

```
img_id = img_name.split('.')[0]
if img_id.split('_')[-1] == '1':
    features_base[img_id.split('_')[0]] = feat_dict
else:
    features_suspects[img_id] = feat_dict
```

4. Part 2 - feature matching:
    a. create two matchers (BFMatcher) - one for SIFT (with *cv2.NORM_L2*) and one for ORB (with *cv2.NORM_HAMMING*);
    b. for each descriptor, create two lists: *y_test* (for reference values) and *y_pred* (for algorithm's predictions);
    c. iterate over features from suspects dictionary and try to match them with each of base features using knnMatch with *k=2* (to get two best matches for each feature);

d. use match distance to perform so-called “ratio test” and select only “good matches”, i.e. matches where the second best match has significantly higher distance. You can do it in the following way (M_COEFF determines how much the best match distance should be smaller than the second match distance - you can use “default” value 0.75):

```
good_matches = list()
for m, n in matches:
    if m.distance < M_COEFF * n.distance:
        good_matches.append([m])
```

e. save the length of *good_matches* list for each base fingerprint (we will use it as a “measure of matching”);

f. find the best (i.e. the one with largest length of *good_matches*) match throughout matches with fingerprint base - append it to *y_pred* list. To *y_test* list append the reference value - you can get it from suspects dictionary key (remember that it should be in the format X_Y), e.g:

```
y_test.append(suspect_feat_key.split('_')[0])
y_pred.append(best_match)
```

g. after getting results for each suspect’s features with use of one matcher (for SIFT or ORB) print the summary using classification_report from scikit-learn package - it is installed on lab computers (in *venv* inside the Biometrics folder). You should see results similar to those presented in Figure 1 below.
NOTE. If you don’t know how to read the results you can read the following article.

	precision	recall	f1-score	support
101	1.00	0.57	0.73	7
102	1.00	0.57	0.73	7
103	0.67	0.29	0.40	7
104	0.29	0.29	0.29	7
105	0.35	1.00	0.52	7
106	1.00	0.43	0.60	7
107	0.50	0.71	0.59	7
108	1.00	1.00	1.00	7
109	0.86	0.86	0.86	7
110	1.00	0.71	0.83	7
accuracy			0.64	70
macro avg	0.77	0.64	0.65	70
weighted avg	0.77	0.64	0.65	70

Figure 1. Example results of fingerprint recognition for SIFT features.

2. AI based fingerprint recognition

FCV2000 (train & val)
https://drive.google.com/file/d/1iAAVOMmLaQcgvjMUC4Bss437GkZAy1mE/view?usp=sharing
FCV2000 (final test)
https://drive.google.com/file/d/1lkP4sCABFGtQ5JdDryFiXNZ-L0H3gBET/view?usp=sharing

Task 1. Following the approach presented in the paper FingerNet: Pushing The Limits of FingerprintRecognition Using Convolutional Neural Network, using PyTorch, ResNet (or other DCNN architecture) and transfer learning implement fingerprint recognition.

Transfer learning (TL) in PyTorch is described in this tutorial: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. We can use it in our experiments.

Some detailed remarks below:

1. In FCV2000 database images are already prepared, so we don't need all transformations from the TL tutorial - just use transformations from 'val' in 'train' (with proposed parameters) and add RandomAffine transformation:
   `transforms.RandomAffine(5,translate=(0.2,0.2),scale=(0.9,1.1))`
   Transformations for 'val' leave unchanged.
2. As the criterion in train_model function we will use function (1) from the article. To calculate the norm part we can use torch.norm function on weights from the fc layer. To get weights from model:
   `weights = model.fc.weight`
3. Change ResNet18 for the proper NN from the article and remember that we have more than 2 classes in our databases.
4. When TL is finished, save the network (torch.save).

Conduct the experiments for FCV2000 on a separate dataset (also provided) - some remarks below:

1. Load saved network (torch.load) and set eval mode.
2. Load test images in loop (as in the previous classes).
3. For each image do the preprocess step with transformations used for 'val' set during TL.
4. After preprocessing, unsqueeze the tensor to form a 4D input batch.
5. With no_grad() get network predictions and choose the proper class (one with the highest outcome) - check if the result is correct.

Task 2 (EXTENSION EXERCISE). Use the same approach as in Task 1 to implement the fingerprint recognition for DB1 dataset:
https://drive.google.com/file/d/1LYIE_TfH4oDK2iQWCXtuSGTWuRbn1h5-/view?usp=sharing

You can use your script from Task 1 (remember to create a copy of it) and add some elements to increase accuracy. You can use data augmentation and/or additional transformations on 'train' set (e.g. RandomRotation, RandomHorizontalFlip).