2021 SPRING CS6170 PROJECT 2 REPORT
DUE: APR 13TH, 2021

SANGHOON KWAK

1. Submitted Files

1. `report.pdf` – Project 2 report file.
2. Source codes
    - `part1_barcodes.py` – Source code for Part 1 – generates the persistence diagrams of 80 images
    - `part1_distances.py` – Source code for Part 1 – computes Bottleneck, Wasserstein, and raw-image distances and generate the plots of MDS/TSNE projections.
    - `part2(PSSK).py` – Source code for Part 2 – classifies 80 images into 4 and 8 classes with kernel-SVM, and Persistence Scale Space Kernel.
    - `part2(Persim).py` – Source code for Part 2 – classifies 80 images into 4 and 8 classes with SVM, and Persistence Image.
3. Data
    - Input data
        - 80(= 10×8) `MPEG-7` image samples: `data\MPEG\camel-(1∼10).gif, chicken-(1∼10).gif, frog(1∼10).gif, horseshoe(1∼10).gif, key(1∼10).gif, octopus(1∼10).gif, pencil(1∼10).gif, sea_snake(1∼10).gif`
    - Output
        - **Barcodes**–*written from the longest to shortest.*: `barcodes\dim[0,1] \XYZ.gif.txt`
        - **Plots**–*colored with respect to their classes*: `plots\METRIC_DR_DIM.png`

Date: Apr 10th, 2021.

2. Part 1: Compute and Compare barcodes of 80 Images

2.1. **Choice of 80 images.** We chose the first ten images from the following eight classes:

`camel, chicken, frog, horseshoe,`
`key, octopus, pencil, sea_snake.`

2.2. **Images to Barcodes.** We used the same method as last Project1 to find the persistence diagrams of given images. Namely, we took the boundary points of images by comparing each pixel to adjacent eight pixels. Then fed the boundary points to Ripser find the persistence diagram of the image. Different from the last time, we computed the dimension-1 diagrams as well as dimension-0 ones.

2.3. **Distances to Projections.** Before taking a closer look on each of distances, we describe here how did we get the projections/plots out of those. We defined a simple tool function `dgm2dist_matrix` where the inputs are the objects to measure distances(e.g. diagrams for Step 1,2 or images for Step 3) and the metric, and the output is the symmetric distance matrix. With this function, we formed the distance matrices for all of three metrics for Part 1. Then got the projection with the module `sklearn.manifold.MDS` and `sklearn.manifold.TSNE`. Finally, we used `matplotlib.pyplot` to color-plot the projection.

2.4. **Step 1 and 2: Bottleneck and Wasserstein Distances.** We used the modules `gudhi.bottleneck_distance` and `persim.wasserstein` to compute the Bottleneck distances and Wasserstein distances between persistence diagrams respectively. Refer to Figure 1 and Figure 2. Note that Wasserstein distances had more spread than Bottleneck distance, especially in dimension 0 projections. This is because Bottleneck distances only take the *maximum* of the distances in the optimal transport of points, whereas Wasserstein distances use the *average* of them, making the distances having more variety between different images.

Dimension-wise, we observe that dimension 1 has better clustering result than dimension 0. We guess this is because all of the images are essentially having one component, so dimension-0 diagrams are not making any interesting features for the images.

2.5. **Step 3: Computing Distances between Raw Images.** To directly measure the distance between two images, we define the metric `imgDist` as following: First, match the size of the two images by taking the bigger number of widths and heights of two. Then comparing each pixel of both images at the same position, count the number of pixels of different color. Finally, normalize this number by dividing by the area of the images(width $\times$ height).

Now we would like to compare the projections by our metric on images with those by the other distances in Step 1 and 2. Refer to Figure 3. They

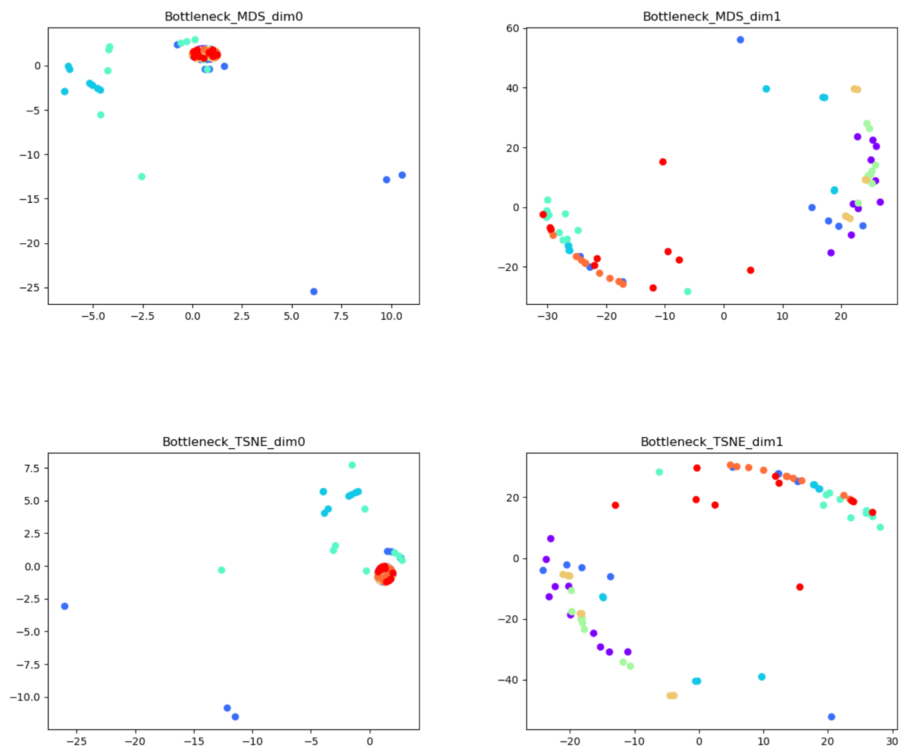

FIGURE 1. Bottleneck distance with MDS and TSNE for dimension 0 and 1 persistence diagrams.

showed much better clustering than Bottleneck and Wasserstein distances. We conjecture that this is because not many of the images needed rotation to have faithful raw distances with other images in the same class. If there were more images perturbed by rotation, the projection result with `imgDist` would be worse.

3. PART 2: CLASSIFY 80 IMAGES INTO 4 AND 8 CLASSES USING SVM/KERNEL-SVM

3.1. **Forming 4 classes, Train/Test data.** We formed the 4 classes of images by combining two classes in pairs out of 8 classes based on their overall appearance as follows:

- `oneleg`: key + pencil
- `fewleg`: chicken + frog
- `manyleg`: camel + octopus
- `ushape`: horseshoe + sea_snake

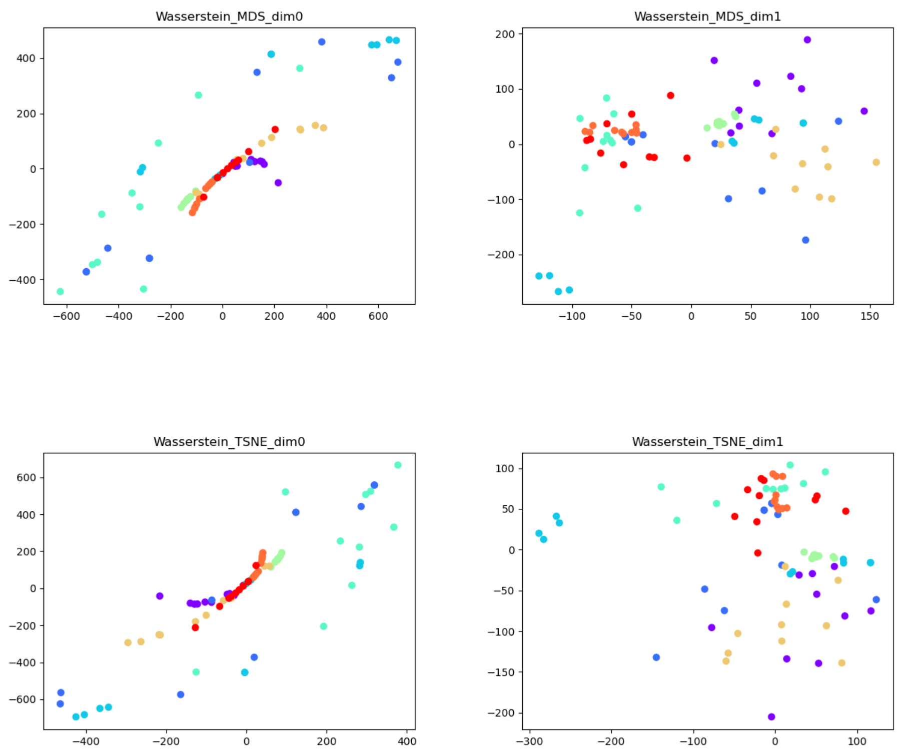

FIGURE 2. Wasserstein distance with MDS and TSNE for dimension 0 and 1 persistence diagrams.

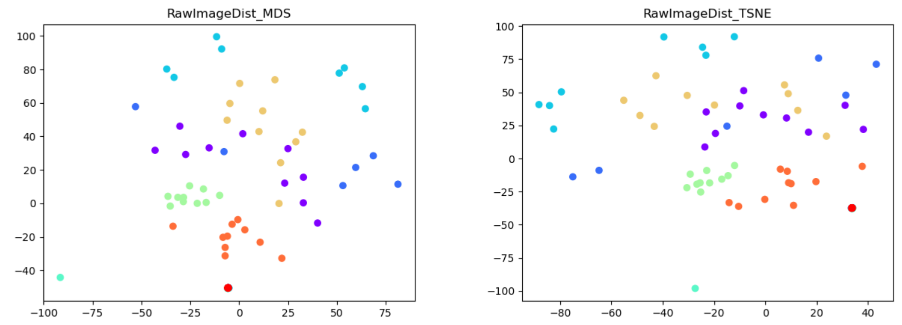

FIGURE 3. Raw image distances with MDS and TSNE.

We fixed the first five images($1 \sim 5$) of each class as train data, and the rest($6 \sim 10$) to be test data set.

3.2. **Kernel-SVM with Persistence Scale Space Kernel.** We form two 40×40 Persistence Scale Space Kernel(PSSK) matrices out of train and test data respectively. Then we construct a SVM with kernel being `precomputed` option, and fit it with PSSK matrix associated to train data. Then plugging in the PSSK matrix associated to test data by varying the target values between 4 classes and 8 classes, we could get the prediction for both cases of 4 and 8 classes.

3.3. **SVM with Persistence Image.** We first generate Persistence Image(PI) out of 80 persistence diagrams. Then flatten the images to be ready with SVM. Separate the flattened images(= `img_array`) into train/test image arrays. (again, by taking first five of each class to be train images and the rest to be test ones) Then construct a normal SVM and fit it with train image array. With the input test image array, we get the predictions for both 4 and 8 classes.

3.4. **Classification Results.** In contrast to our expectation, the overall results were better for PSSK than PI. Refer to Table 1. They were both poor at classifying the images into 8 classes with dimension 0 persistence diagram.

	dim0	dim1	dim 0	dim 1
	4 classes	4 classes	8 classes	8 classes
PSSK	0.25	**0.6**	0.125	**0.425**
PI	**0.425**	**0.475**	0.15	0.275

TABLE 1. Comparison of accuracy of classification using PSSK vs PI into 4 and 8 images.

Taking a closer look of PSSK classification results, we found that `horseshoe` and `sea_snake` are both 100%-classified into `ushape` examples. (Table 2) However, they are both classified as `horseshoe` in 8-class ones, yielding the precision 0 for `sea_snake`. (Table 3) This might be due to their resemblance in dimension 1 homology; namely their major(=high persistence) loops forming their 1st homology are relatively young in both cases, explaining the name "`ushape`".

On the other hand, for PI classification, there are two interesting things to note. First, in 4-class classification, 8 out of 10 images from `ushape` class are classified into `oneleg` class. This might be because of the fact that the U-shaped objects can be deformed into I-shaped ones. That is, if we straighten the images of "U-shaped", then we will get "I-shaped" ones, which correspond to `oneleg` class. Second, in 8-class classification, PI did 100%-precision with `frog` classification with 1-dimensional diagrams. We think this is because it is the only class with a lot of noises within the interior of the image, yielding a lot of 1-dimensional homology classes, in contrary to other classes.

	precision	recall	f1-score	support
`fewlegs`	0.43	0.9	0.58	10
`manylegs`	0.64	0.7	0.67	10
`oneleg`	1	0.6	0.75	10
`ushape`	1	0.2	0.33	10
accuracy			0.6	40
macro avg	0.77	0.6	0.58	40
weighted avg	0.77	0.6	0.58	40

Table 2. 4-class classification with PSSK, dimension 1 diagrams

	precision	recall	f1-score	support
`camel`	0	0	0	5
`chicken`	0	0	0	5
`frog`	0.22	1	0.36	5
`horseshoe`	1	0.4	0.57	5
`key`	0.5	0.2	0.29	5
`octopus`	0.62	1	0.77	5
`pencil`	0.8	0.8	0.8	5
`sea_snake`	0	0	0	5
accuracy			0.42	40
macro avg	0.39	0.42	0.35	40
weighted avg	0.39	0.42	0.35	40

Table 3. 8-class classification with PSSK, dimension 1 diagrams

	precision	recall	f1-score	support
`fewlegs`	0.33	0.2	0.25	10
`manylegs`	0.67	0.4	0.5	10
`oneleg`	0.4	1	0.57	10
`ushape`	0.33	0.1	0.15	10
accuracy			0.42	40
macro avg	0.43	0.43	0.37	40
weighted avg	0.43	0.42	0.37	40

Table 4. 4-class classification with PI, dimension 0 diagrams

	precision	recall	f1-score	support
`fewlegs`	1	0.2	0.33	10
`manylegs`	0.67	0.8	0.73	10
`oneleg`	0.36	0.9	0.51	10
`ushape`	0	0	0	10
accuracy			0.48	40
macro avg	0.51	0.47	0.39	40
weighted avg	0.51	0.47	0.39	40

Table 5. 4-class classification with PI, dimension 1 diagrams

	precision	recall	f1-score	support
`camel`	0.15	0.4	0.22	5
`chicken`	0	0	0	5
`frog`	1	0.4	0.57	5
`horseshoe`	0	0	0	5
`key`	0.21	0.8	0.33	5
`octopus`	0.5	0.2	0.29	5
`pencil`	0.67	0.4	0.5	5
`sea_snake`	0	0	0	5
accuracy			0.28	40
macro avg	0.32	0.28	0.24	40
weighted avg	0.32	0.28	0.24	40

Table 6. 8-class classification with PI, dimension 1 diagrams