Project 3 - README

ΜΥΣΤΑΚΙΔΗΣ ΙΩΑΝΝΗΣ - 1115201600113
ΧΟΥΣΙΑΔΑ ΕΥΑΓΓΕΛΙΑ - 1115201600200

Github: https://github.com/myioannis/Project-3 (Public Repository).

General Notes:

- For the c/c++ executables the datasets have to be **unzipped** and for the python executables they have to be **zipped**. So both the zipped and unzipped versions of the files are going to be needed. Make sure to look at the *How to run* subsection in each question.
- **When you unzip the datasets please make sure the names have contain dashes only and no dots**. For example:

  t10k-images-idx3-ubyte.gz

  will become

  t10k-images-idx3-ubyte

  and not

  t10k-images.idx3-ubyte

  If it contains dots, please change the dots to dashes in all 4 unzipped files.
- Please keep all of the files (source code, executables, datasets etc.) in the same directory in order to avoid unwanted errors with the paths.

Question A:

How to run:

- The datasets have to be **zipped**.
- You can run *reduce* as a python executable (.py) or as a jupyter notebook (.ipynb).

- .py:

  You can run the python executable exactly as specified in the assignment. Any command line argument you ommit will be explicitly asked during execution.

- .ipynb:

  You should run all the jupyter notebook cells. Any command line argument you ommit will be explicitly asked during execution. In case the notebook is run on google colab, the first cell temporarily clones our github repository to your local session so that you don't have to upload all the files and the datasets on colab.

- You can either upload a pretrained encoder model and simply convert the datasets into their latent representations or train your own model(s) and then convert the datasets into their latent representations. We have included 4 pretrained models in our directory. In case you want to use a pretrained model, the path for the model is asked during execution.

- If you do not provide paths/names for the output files (latent representations of the datasets) through the command line arguments, the output files get the default names *output_data* and *output_queries*.

Implementation:

The code for *reduce* is similar to the autoencoder of the second assignment, with some changes such as:

- We're now using conv2DTranspose and the *strides* parameter for upsampling and downsampling. We've created some helper functions in order to make the upsampling and the downsampling symmetrical.

- We avoided overfitting with the use of some dropout layers.

- The values of the output vectors are normalized in the range [1,2,...25500] and are saved in two consecutive bytes, and are read as of type *short*.

- All metadata and pixels are saved in high endianess. The appropriate conversion between different endianess takes place, when needed, in the executables of Question B.

Results/Experiments:

The observations we made about the models in the previous assignment still hold here. The only difference is that when including the layers of the latent representation the training process becomes a little bit more smooth. So there isn't a huge difference between the models.
What became obvious though is that **the larger the dimensions of the**

latent representation, the better the results (both during training and in Question B).
We carried out various experiments with different hyperparameters and different dimensions for the latent representations and we concluded that a model with:

> 3 convolutional layers, kernel size = (3,3), 64-128-256 nodes for the three convolutional layers respectively, 30 epochs and a batch size of 64

did very well for all of the dimensions we tried out.
So we trained this model with 10, 30, 50 and 100 dimensions for the latent representations in order to showcase how the size of the dimensions itself affect the results of Question B. These models are included in our zipped directory and in order to produce the output files for the latent representations you have to pass them through the *reduce* executable.

Question B:

How to run:

- The datasets have to be **unzipped**.
- You can run the *search* executable exactly as specified in the assignment.
- There are default values for the k and L flags, so you can omit them if you want.

Implementation:

- The code for this question was taken from the first assignment. We only made some minor adjustments.
- We also introduced templates to the files of the first assignment in order to support the type change in Question A.

Results/Experiments:

In general, the reduced representations provided better results than LSH. The more the dimensions of the latent representations the closer the reduced approximation was getting to 1. With our pretrained model with 100 dimensions the reduced approximation factor was very close to 1.

Question C:

How to run:

- For search, the datasets and the label files have to be **unzipped**. For EMD.py you need the **zipped** files in the same directory. You don't need

to provide them anywhere as arguments, this is done automatically in search (it concatenates the names of the unzipped files with .gz).

- In order to run EMD you have to install the PuLP library and the GLPK solver. We installed them using **conda**. We had to activate the python environment we were working on with:

  conda activate "name-of-environment"

  and then install the packages with:

  conda install -c conda-forge pulp
  conda install -c conda-forge glpk

  You can omit the envirnment activation and install the packages in base. You might also be able to install the packages with pip if you want.

- You can run the *search* executable exactly as specified in the assignment. First you will be asked how many images you would like to use from the dataset and the queryset, and then you'll see the results of each method.

Implementation:

- We implemented EMD in python and for the Manhattan method we used the files from Question B. When you call the *search* executable, the *system* function is called which runs the python executable to produce the EMD results and then the *search* executable continues with the Manhattan results.

Results/Experiments:

For the EMD, a larger number of pixel clusters provided better results than smaller ones, which is expected since more information is taken into consideration. In most cases, when using 49 (7x7) clusters of 4x4 pixels each or more clusters, EMD provided better results than Manhattan. For example, with 1000 and 10 queries, the EMD accuracy was 8.2 and Manhattan's was 7 (this was also observed with different numbers of data and queries).

Question D:

How to run:

- There is the *cluster* executable that does the clustering with the three different methods and the *classification* that creates the clusters from the neural network.
  - classification (python):

    - The datasets have to be **zipped**.
    - You can run this executable as following:

      python classification.py -d trainSet -dl trainLabels -t testSet -tl testLabels -model encoder_model_name.h5
    - When you classify the training data and make the clusters, the output file has the default name "clustersNN.txt".
    - We have included 4 cluster files in our directory (one for each pretrained model from Question 4).
  - cluster (c++):
    - The datasets have to be **unzipped**.
    - You can run this executable exactly as mentioned in the assignment.
    - In order to run this executable you must have the clusters file from the neural network already created.
    - In order to run the silhouette in reasonable time you have to use fewer data.

Implementation:

- classification:
  - We based our implementation on the classification executable from the second assignment.
  - We removed the flatten layer which is now part of the encoder.
- cluster:
  - We based our implementation on the clustering files from the first assignment.
  - We introduced some new functions to support the mapping from the new (latent) space to the original one.

Results/Experiments:

- classification:
  - Again, the observations we made about the effects of the different hyperparameters in the second asssignment still hold here. For this reason, we made clusters file for each of the four encoder models of Question A, using the following classification model:

    64 dense nodes, 30 epochs, 512 batch size

    which did fairly good in all cases. We named each output (cluster) file as "clustersNN" followed by the number of dimensions in each case.

- cluster:
  - Since we use kmeans++ there is some randomness in the clusters created, therefore to some extend the results depend on this.
  - In general, the clusters made with the neural networks provide a slightly better silhouette than the other methods.
  - The comparison between the other two methods depends on the number of dimensions for the reduced space. Generally, a larger number of dimensions means better cluster and in some cases the results are better than the original space.