Avantari Technologies: Machine Learning Task Solution

Shashank Kumar Singh: shashank9830@gmail.com

Problem statement: You are provided with a dataset of ~5k 512x512 images, your program should accept an 512x512 input image and return N images from the provided dataset similar to the input image.

Approach to solve this task:

This task was divided into two parts:

1. Given an input image (512x512x3), find N similar images from the dataset.
2. Cluster the dataset into K-groups.

[1] Given an input image (512x512x3), find N similar images

Below mentioned approach was used to tackle this problem:

1. Images in the given dataset were resized to 256x256 to reduce computation cost.
2. For resizing, bilinear interpolation was utilized (refer resize_dataset.py).

3. AutoEncoder model was developed (refer create_autoencoder.py) and trained on this dataset.
    a. Input and output to the model were 256x256x3 images.
    b. It was a combination of Encoder + Decoder.
    c. For training code please refer "trainer_notebook.ipynb".

4. After training the AutoEncoder, its encoder part was separated (refer create_autoencoder.py).

5. Encodings for all the images in the dataset were computed using the separated encoder.
6. Each encoding was 32x32x4 in dimension. On flattening, its dimension changed to (4096,).
7. After this all the encodings were saved in NumPy format (refer get_encodings.ipynb).

8. Now with 4738 such encodings, the task was to find similarity between all encodings (refer get_similarity.ipynb).
9. Cosine Similarity was the chosen metric to determine how similar two encodings were.
10. Since there were 4738 images, a similarity matrix of (4738x4738) was created.
    a. Each field in the matrix contained a value between -1 to 0.
    b. -1 shows maximum similarity whereas 0 shows minimum similarity.

	0	1	2	3	4	5	6	…	4737
0	-1	-0.94	-0.36	-0.25	-0.16	-0.13	-0.55	…	-0.67
…	…	…	…	…	…	…	…	…	…
4737	-0.99	-0.96	-0.45	-0.73	-0.51	-0.57	-0.01	…	-1

Table 1: An example of similarity matrix is shown above.

11. After this, rows were converted to lists and sorted in descending order of similarity.
    a. For each row in the similarity matrix (which shows the similarity of 1 image with 4738 different images), sorting was applied.

Row0	-1	-0.94	-0.36	-0.25	-0.16	-0.13	-0.55	…	-0.67

Table 2: Image 0.jpg's row depicting its similarity with other images.

b. Values in the rows were tagged with image no and then sorted in descending order of similarity.

List0	-1, '0'	-0.94, '1'	-0.67, '4737'	-0.55, '6'	-0.36, '2'	…

Table 3: Row is converted to list, values are tagged with image numbers and then sorting is applied.

c. This gave us a list of 4738 such lists.
d. Each internal list was sorted in descending order of similarity.
e. Quite obviously, the first image inside each list was that image itself. For clarity,
    i. Row 0 (or later, list 0) was for 1st image (or image 0.jpg).
    ii. First item in List 0 was also 0.jpg (as an image is most similar to itself).
    iii. Items after that had decreasing similarities.
f. This was done for all the rows (each row depicting image equal to its row number).

12. At last, two modes of execution were created to handle requests (refer final_solution.ipynb).

    a. If an image from dataset is given as input (cached mode).
        i. This image is already encoded and cached with its similarity measure calculated and sorted against every image in the database.
        ii. The only task left is to select first N images from the sorted list and present them as similar images.
        iii. This requires no extra computation.

    b. If an image was given which is NOT in the dataset (default mode).
        i. The image is first preprocessed to 256x256x3 size using bilinear interpolation.
        ii. It is encoded using the encoder.
        iii. Cosine similarity is calculated against all the encodings saved in the cache.
        iv. The similarities are sorted in decreasing order.
        v. First N images are presented as the solution.
        vi. This is slightly more time taking compared to the cached mode but works exactly as expected.

13. This is how first part of the task was solved.

[2] Cluster the dataset into K-groups.

This problem was again divided into two parts:

I. Find the optimal value for K, medoids and clusters.
II. Divide the dataset into K groups.

[2 (I)] Find the optimal value of K

Elbow method was used to find the optimal value of K.

- Minimum value of K was set to 2.

- Maximum value of K was set to 100.
- WCE for different values of K is calculated using Elbow method.

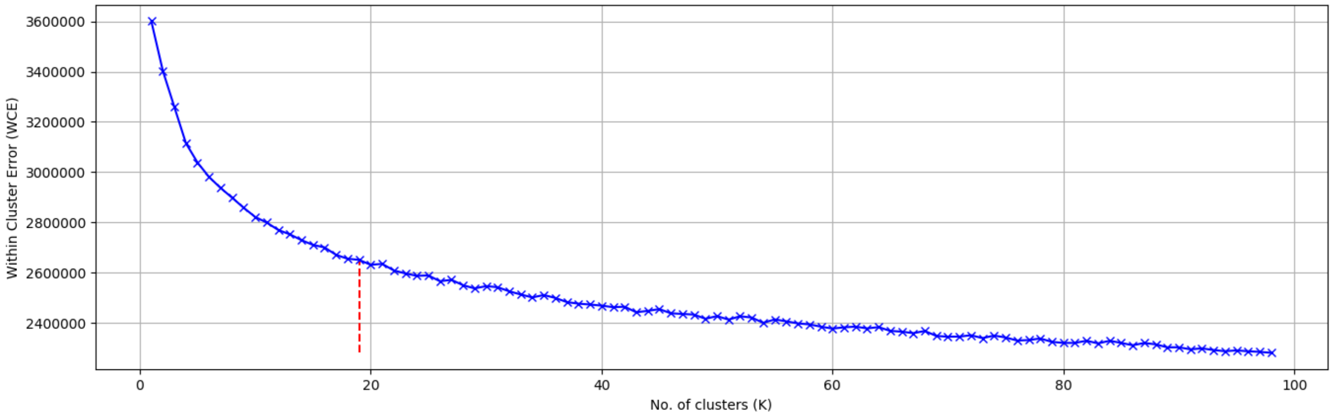

Image: Within Cluster Error (WCE) vs. No. of clusters (K) plot

- According to the Elbow method, the optimal value of K was 19.

[2 (II)] Divide the dataset into K groups

K-Medoids algorithm was used to partition (or form clusters) the dataset into K groups (clusters according to the algorithm).

- Take value of K given by Elbow method.
- Randomly initialize K medoids.
- Perform the K-Medoids algorithm to find actual medoids and clusters.

NOTE: refer k_grouping.py and partition_dataset.py for all the code of (2)(I) and (2)(II).

Pyclustering library was used for all the tasks in section 2.

Model Type: AutoEncoder

Programming language: Python 3

Machine Learning Framework: Tensorflow/Keras

Libraries: tensorflow, numpy, matplotlib, time, json, os, math, pillow, pyclustering and shutil.

Information on files inside the main directory (listed in order of approach)

	Filename	**Type**	**Information**
1	dataset	Directory	Original dataset.
2	resize_dataset.py	Python script	Resizes the dataset images to 256x256.
3	resized_256	Directory	Resized dataset.
4	create_autoencoder.py	Python script	Creates an autoencoder model.
5	autoencoder.h5	H5 file	AutoEncoder model saved in H5 format.
6	trainer_notebook.ipynb	Jupyter Notebook	Model training code.
7	trained_autoencoder.h5	H5 file	Trained autoencoder saved in H5 format.
8	trained_encoder.h5	H5 file	Encoder part of the trained autoencoder.

9	get_encodings.ipynb	Jupyter Notebook	Code to get the encodings of all the images.
10	encodings.npy	NumPy file	Encodings of all 4738 images.
11	get_similarity.ipynb	Jupyter Notebook	Code to find similarity of all the images with each other.
12	cosine_similarity_matrix.npy	NumPy file	Cosine similarity matrix generated in the previous step.
13	sim_mat_sorted.json	JSON file	Images sorted in decreasing order of similarity to each other.
14	final_solution.ipynb	Jupyter Notebook	Main user notebook. Run this for final output.
15	Sample outputs	Directory	Some pre-executed notebook outputs in HTML format. One example of both cached and default mode.
16	k_grouping.py	Python script	Code to implement Elbow and K-medoids algorithm.
17	k_groups.json	JSON file	JSON file containing list of medoids and clusters
18	partition_dataset.py	Python script	Code to partition the dataset as mentioned in the above JSON file
19	K Groups	Directory	Folder containing K-Groups
20	.ipynb_checkpoints	---	---