Manual for Product Quantizer

Version 1.10

Wan-Lei Zhao
wlzhao@xmu.edu.cn

August 26, 2022

Contents

1 Introduction

"product quantization" (PQ) once was a popular approximate nearest neighbor search method in both the literature and industry. This method is proposed by Herve Jegou et. al in their famous paper [1]. Until now, PQ is still a classic method for NN search on high-dimensional data. It is particularly useful when the size of your raw reference data is bigger than the size of memory. PQ normal is able to reduce more than one magnitude of memory use while guaranteeing *40-60%* NN search quality.

This document presents a full guidance to run the implementation of PQ from Wan-Lei Zhao. Basically, the order of sections arranged in this manual is also the order that a beginner gets to familiar with this tool. While a thorough understanding about PQ (via reading the original paper) is required before the user could modify the codes and work with the codes.

2 Train Product Quantizer

There are basically two ways to practice PQ. The first way is to encode the original vectors with PQ directly. The second way is to encode the original vector by a coarse quantizer (namely vector quantizer) in the first step. Then the vector residue (the substraction of corresponding vector quantizer from the original vector) is encoded with PQ. The second way brings better search accuracy and efficiency. Compared to the first way, it is not required to compared to all the PQ codes of the reference set. The encoded residues are indexed by an inverted file structure, which makes the search more efficient[1]. Higher search quality is achieved as inverted file indexing also helps to filter out noises. In the following, we assume the user will follow the second way to try PQ. It is possible to choose the first way as well if the user is willing to modify the codes a little bit.

2.1 Train Coarse Quantizer

The coarse quantizer is nothing more than a visual vocabulary which coarsely quantizes high-dimensional data. It restricts the nearest neighbor search (NN) to be undertaken within a sub-region of the whole vector space (spanned by reference vectors). In this implementation, k-means++ is adopted to cluster the training matrix into specified number of clusters. The cluster centers are treated as the visual vocabulary. In our the implementation, user is required to specify the training matrix, the number of clusters (vocabulary size) and the destine file name to save the vocabulary.

With the trained coarse quantizer, it becomes possible to perform NN search on PQ with the support of inverted file structure. The advantage of introducing a coarse quantizer is to avoid a brute-force comparison between query and the whole reference set.

[1]This is not always true. Bigger size of vector quantizer also leads to high quantization cost for the query side.

2.2 Command

In our implementation, the command for producing coarse quantizer is shown as follows. User is asked to provide a training matrix (**-vc** matrix) (please refer to **Section 4.A** for the format), specify the size of vocabulary by '**-k** num'. The output is a $num \times d$ matrix, where d is the vector dimension.

pq **-vc** matrix **-k** num **-d** dstfn

One can find the implementation of this function in "**pqtrainer.h**" and "**pqtrainer.cpp**".

2.3 Train Product Quantizer

Although it is possible to build product quantizer without coarse quantizer, it is not favored since otherwise linear-scanning over whole reference set during NNS has to be undertaken. As a result, user is strongly recommended to train a coarse quantizer (see Section 2.1) in advance.

2.4 Command

Following comand shows the typical option for product quantizer training.

pq **-tc** pq-tc.conf **-s** srcfn **-k** sz **-m** seg **-o** ivfpq **-d** dstfn

Options given in the comnandline are all required. They are explained in detail as follows.

- ◇ **-tc pq-tc.conf** This option specifies the configuration file for the routine, in which the coarse quantizer is specified by 'vocab' (as shown in Table 1).

- ◇ **-s srcfn** This option specifies the training vectors, which are organized into a matrix in row. One has to make sure sufficient number of training vectors are collected. For instance, if one wants to have k numbers of sub-quantizer in each segment, the number of training vectors should be larger than $10 \times k$.

- ◇ **-k sz** This option allows user to specify the vocabulary size of each sub-quantizer.

- ◇ **-m seg** This option allows user to specify how many segments the user would like to partition the original vector into.

- ◇ **-d dstfn** This option specifies the generated product quantizer. Basically, the resulting file is comprised by two parts. The first part is the coarse quantizer which is an exact

Table 1: Configuration for Product quantizer training

vocab=/home/wlzhao/pq/vocab/sift_8k_kpp.txt

copy of coarse quantizer from specified 'vocab'. The second part are *seg* numbers of sub-quantizers.

◇ **-o ivfpq** In order to train PQ with coarse quantizer support, user is required to specify option '**-o**' to ivfpq.

The implementation of training product quantizer can be found in "**pqtrainer.h**" and "**pqtrainer.cpp**".

3 Encode Vector with Product Quantizer

3.1 Encode Reference Set

Once the product quantizer is trained, user is ready to encode the reference set by product quantizer, which will support the NNS in the next step. Here we assume user will take Asymmetric NNS later, which means there is no need to encode query side vector. The command for encoding is shown and explained in the command subsection.

3.2 Command

Following command shows the options for vector encoding. Notice that all the options in the command are required.

pq **-ec** pq-ec.conf **-i** itmtab -o ivfpq

In this command, user is required to specify a configuration file, where the product quantizer generated in Section 2 is supplied by 'vocab=path'. The second option '-i itmtab' specifies the path for source file and destine file for the output. As shown in Table 2, 'vectab' specifies the source file, while 'pqctab' specifies the destine file for the encoded vectors. One row for one vector. It is NOT suggested for the user to modify the file.

The implementation of encoding with product quantizer can be found in "**pqencoder.h**" and "**pqencoder.cpp**".

Table 2: Format of 'itmtab' for product quantizer encoding

```
<item>
vectab=/home/wlzhao/src/pq/etc/sift_base.txt
pqctab=/home/wlzhao/src/pq/etc/sift_base_pqc.txt
</item>
```

Table 3: Configuration for NNS with product quantizer

```
refer=/home/wlzhao/src/pq/etc/itm_pq_sift_base.txt
vocab=/home/wlzhao/src/pq/etc/sift_8k_kpp_pq256.txt
dim=8192
```

4 Nearest Neighbor Search with Product Quantizer

Once the product quantizer is trained and the reference set have been encoded, we are ready to perform NNS with product quantizer. To conduct the search task, user is required to supply 1. encoded reference set; 2. product quantizers in the configuration file. Table 3 gives an example that how the configuration file is organized. As shown in the table, 'refer' specifies the 'itmtab' of encoded reference set, which is the same as Table 2. While 'vocab' specifies the path for trained product quantizers. In addition, the size of coarse quantizer has to be specified by 'dim'. When the configuration file is set, we are ready to conduct the NNS with product quantizer, given queries are organized into a text file. Each query is put as one row in the file.

4.1 Command

Query with product quantizer is shown in following command.

pq **-nc** pq-nc.conf **-o** *pqs*|*pqa* **-q** qryfn **-d** dstfn

Option '-nc' allows user to specify the configuration file. Here two NNS search options are provided by '-o'. User will conduct asymmetric NNS by specifying '-o' with 'pqa'. In this case, user only needs to provide raw query vectors which are organized as a matrix (refer to Appendix I and Appendix II). As one could see from the appendix, the matrix could be in text file format ('.txt') or binary file format ('.fvecs'). User is also allowed to perform symmetric search with '-o pqs', given the queries are already encoded in advance. In this case, the input queries should be in the same format as the encoded reference set. The query file is specified by '-q qryfn'. The NNS result will be saved into 'dstfn' specified by '-d'.

The implementation of NNS with product quantizer can be found in "**pqnnsearch.h**" and

Table 4: Configuration for similar image search with product quantizer

```
refer=/home/wlzhao/src/pq/etc/itm_vlad_holidays.txt
vocab=/home/wlzhao/src/pq/etc/vlad_imgnet_hesaff_kpp_8k_pq256.txt
dim=8192
```

"**pqnnsearch.cpp**".

5 Similar Image Search with Product Quantizer

Once the product quantizer is trained and the reference set have been encoded, we are ready to perform image search with product quantizer. To conduct the retrieval, user is required to supply 1. encoded reference set; 2. product quantizers in the configuration file. Table 4 gives an example that how the configuration file is organized. As shown in the table, 'refer' specifies the 'itmtab' of encoded reference set, which is the same as Table 4. While 'vocab' specifies the path for trained product quantizers (including the coarse quantizers). In addition, the size of coarse quantizer has to be specified by 'dim'. As one can see, image search is quite similiar as NNS. The only difference is that each query and reference item is associated with a label, namely the image name.

When the configuration file is set, we are ready to conduct the image search with product quantizer, given queries are organized into a text file. Each query is put as one one row in the file.

5.1 Command

Image retrieval with product quantizer is shown in following command.

pq **-rc** pq-rc.conf **-q** qryfn **-d** dstfn

Option '-rc' allows user to specify the configuration file. Here two NNS search options are provided by '-o'. Notice that, in the implementation, image search is done only by asymmetric PQ. As a result, unlike NNS task, options for choosing asymmetric or symmetric search is unneccesary. As we did with asymmetric PQ, user only needs to provide raw query vectors which are organized as a matrix (refer to Appendix I and Appendix II). The query file is specified by '-q qryfn'. The NNS result will be saved into 'dstfn' specified by '-d'. The output format of image retrieval is similar as NNS (see Appendix IV), the only difference is that the index numbers for the query and reference have been replaced by image names. In the item table for both query and reference sets, 'imgtab' is required to supply, which specifies the label/image name for each query or indexed reference image. As a result, the item table for query images should look like

Table 5: Examplar item table on the query side and reference for image retrieval

Query side
imgtab=/home/wlzhao/src/prj/pq/data/vlad_holidays-qry_hesaff_img.txt vectab=/home/wlzhao/src/prj/pq/data/vlad_holidays-qry_hesaff_vct.txt
Reference side
imgtab=/home/wlzhao/src/prj/pq/data/vlad_holidays-qry_hesaff_img.txt pqctab=/home/wlzhao/src/prj/pq/data/vlad_holidays-qry_hesaff_vct.txt

as Table 5(1). While for the reference set, the item table should look like Table 5(2). Before conducting retrieval, user has to be sure the reference set has been encoded with PQ.

The implementation of image search with product quantizer can be found in "**pqnnsearch.h**" and "**pqnnsearch.cpp**".

6 Possible Extensions

In my current implementation, PQ is only designed for L2-distance. It is possible to extend PQ to Lp-distance when one make the k-means++ perform smoothly on the Lp-distance. In the literature, there is no paper to evaluate the performance of PQ other than L2-distance space. Besides the low quality caused by quantization loss, another limitations for PQ from my experience is that it is still inefficient in comparison to graph-based approaches. Quantization with small vocabulary size is still inefficient in comparison to graph-based approaches.

It is possible to support PQ NN search with graph-based approaches. In this way, the original raw vectors are replaced by PQ codes. The distances between original vectors are replaced by the distances between PQ compressed vectors. This is how FAISS [2] works. Such kind of hybrid approach only makes the search faster on very large-scale case, while making no big difference on the search quality.

7 Copyrights Issue and Restrictions

This project is fully supported by **Odd Concepts Inc.** from Seoul, South Korea during 2014∼2015.

8 Appendix

I Format of input text matrix (.txt)

n	d			
val_{11}	val_{12}	val_{13}	...	val_{1d}
val_{21}	val_{22}	val_{23}	...	val_{2d}
			.	
			.	
			.	
val_{n1}	val_{n2}	val_{n3}	...	val_{nd}

It is a standard matrix with **n** rows and **d** columns.

II Format of input binary matrix (.fvecs)

d	val_{11}	val_{12}	...	val_{1d}
d	val_{21}	val_{22}	...	val_{2d}
			.	
			.	
			.	
d	val_{n1}	val_{n2}	...	val_{nd}

A standard matrix with **n** rows and **d** columns is written in n rows. In each row, the leading field is an integer d, which indicates there are d numbers of float values in the following fields. All the fields in one row are kept in four bytes binary format. Please be noted that there is no separator between two fields.

III Format of visual vocabulary

n	d			
V_1^1	V_2^1	V_3^1	...	V_d^1
V_1^2	V_2^2	V_3^2	...	V_d^2
			.	
			.	
			.	
V_1^n	V_2^n	V_3^n	...	V_d^n

It is a standard matrix with **n** rows and **d** columns. Each row writes one d-dimensional visual word.

IV Format of nearest neighbor search output

query1 candidate1	score_a
query1 candidate4	score_r
.	.
.	.
.	.
query4 candidate8	score_d
.	.
.	.
.	.
queryn candidate5	score_f

First column shows the retrieved pair consisting of one query ID and one reference ID separated by 'tab'. Second column shows the squared distance estimated by product quantizer.

References

[1] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2010.

[2] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs,'' *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535-547, 2019.