BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI

INFORMATION RETRIEVAL
CS F469

2018-19(1st Semester)

REPORT

Content Based Image Retrieval System based on Elasticsearch

Submitted by :-

Arnav Sailesh (2016A7PS0054P)

Nalin Sharma (2016A7PS0309P)

Nishant Arora (2014B4A40795P)

Problem Statement

In this project we try to develop an end-to-end content-based image retrieval system built upon Elasticsearch, a well-known and popular textual search engine.
The core idea is to encode image feature vectors into a collection of string tokens in a way such that closer vectors will share more string tokens in common. By doing that, we can utilize Elasticsearch to efficiently retrieve similar images based on similarities within encoded string tokens.

Problem Background

A) Application Domain

Application Domain of our work is Content-based Image Retrieval (related to Multimedia Information Retrieval).

What is Multimedia IR ?
Multimedia Information Retrieval is a research discipline that aims at extracting semantic information from multimedia data sources. Data sources include directly perceivable media such as audio , image and video, indirectly perceivable sources such as text , semantic descriptions, biosignals as well as not perceivable sources such as bioinformation, stock prices, etc.

What is Content-based Image Retrieval ?
Content-based Image Retrieval is the application of computer vision techniques to the image retrieval problem, that is , the problem of searching for digital images in large databases.
“Content-based” means that the search analyses the contents of the image rather than the metadata such as keywords, tags or descriptions

associated with the image. The term “Content” refers to colors, shapes, textures, or any other information that can be derived from the image itself. CBIR is desirable because searches that rely purely on metadata are dependent on annotation quality and completeness.

B) Motivation of the problem

With the rapid growth of large collections of digital images the problem of efficient organization and retrieval of digital images has become a very popular research topic in recent years. Also, as one image is worth a thousand words, you might want to query something which is better described by an image thus at some places it might be better to do image retrieval based on image as query rather than textual search.

Why Elasticsearch ?
It is extremely easy to be deployed, distributed, scaled and monitored in a cost-friendly manner. Moreover , it is capable of supporting multimodal searches, where visual and textual information can be jointly leveraged in retrieval.

C)Technical issues

We were having some issues using elasticsearch. When we have already extracted the features out of the images and then we were trying to index them , sometimes the documents (list of strings) that were retrieved by the elasticsearch were incorrect while sometimes it returned no results.We tried to google it and found that it is typically related to text being “analysed” - a type of transformation - into index *terms* that are different from what you search for. For example, if you have the text `"Text-analyzer"`, the *standard* analyzer will produce the terms `"text"` and `"analyzer"` for indexing. That's what can be searched for. The term `"Text-analyzer"` is never added to the index. Anything that tries to look for that exact term in the index will not find anything.

Consequently, it's important that the way in which the text is analyzed when searching is compatible with how the text was transformed during indexing. In the previous simple example, this means that terms must be correctly tokenized and lowercased. Some query types perform text analysis automatically, while most do not. None of the filters perform any text processing.
Specifically, the match and query_string family of queries process text. That is why a match query for "Text-analyzer" will return results, while a term query or filter for exactly the same will not. The match query uses the same analyzer and is looking up "text" and "analyzer" in the index, while a term query for "Text-analyzer" looks for exactly that, which does not exist.
Many users try to fix this by adding wildcards around their search input, akin to the SQL WHERE column ILIKE '%query%' which slows SQL databases to a crawl. This only guarantees that searches will be slow, not that you find what you want! For example, a wildcard search for *Text-analyzer* would have to look through all the terms in the index, most likely not finding it.
Problems are also frequently encountered due to the standard analyzer performing stop word removal, i.e. removing words like "it", "no", "be" and so on. This can be very annoying when you index e.g. country codes.
standard is currently the default analyzer. While this will change in Elasticsearch 1.0, relying on the default analyzer is rarely what you want. While Elasticsearch can do (and does) a good job of guessing types for non-string values, it can't possibly know how to properly treat your text. Is it a name, a tag or prose?
On the flip side, sometimes, when the mappings are wrong, you'll find that documents that should not match, in fact do. For example, if the date "2014-01-02" is indexed as a string with the standard analyzer, a match query for 2014-02-01 (a completely different date) will match, because the query is for the terms "2014", "02" and "01" as an OR.

Related work: Literature Survey

Design and Implementation of a Distributed System for Content-Based Image Retrieval.

This paper presents a distributed system for content based image retrieval on very large datasets . It relies on the distributed search engine software Elasticsearch, which allows us to split an index into several parts and to spread them among different servers, so that search operations can be parallelized reducing as consequence the query execution time.
In order to enable the full-text search engine to perform visual similarity search, they have used Deep Convolutional Neural Network Features extracted from the images of the dataset and encoded as standard text.
From this experiment, they have proved its scalabilty, i.e., CBIR System is able to keep more or less constant the query execution time when the data volume grows and a new node is added.
Moreover, the response time can be reduced by exploiting the query reduction mechanism at the cost of a lower quality of approximation.

Large scale image retrieval system with Elasticsearch.

This paper presents us with an approach for indexing CNN features as permutations to build a CBIR system (represent feature objects as permutations built using a set of reference object identifiers as permutants). It also relies upon the full-text retrieval engine Elasticsearch, which works in secondary memory and provides horizontal scalability and reliability.The basic idea is to use the same activation values of the neural network as a means to associate CNN feature vectors with permutations.They have explored the impact of introducing a (CReLU) preprocessing phase on R-MAC dense descriptors, which allowed us to regain the informative contribution of the negative elements of the vectors.They also observed that approach exhibits interesting performance in terms of efficiency and effectiveness compared to state-of-the-art approaches, which operate in main memory and hardly scale to large-scale applications.

Content Based Image Retrieval System based on Semantic Information Using Color, Texture and Shape Features .

In this paper, an efficient image retrieval system is represented by constructing the image features, namely Color auto-Correlogram Feature, Gabor Texture Feature and Wavelet Transform Feature. Color is represented by color autocorrelogram where as the texture feature is represented by Gabor wavelet. Another efficient feature like Wavelet transform for edge extraction i.e., shape is also used in conjunction with color and texture feature in order to provide better results and can be used to obtain high precision of image retrieval. From the analysis made in the experimented result, it illustrate that the proposed method achieves the better precision rate as compared to other existing methods. Thus the feature extracted from the proposed method achieves an average accuracy rate of 83% for corel database, whereas 88% for Li database and 70% for Caltech-101 database in Content Based Image Retrieval system.

Content-Based Image Retrieval Using Features Extracted From Halftoning-Based Block Truncation Coding.

In this experiment, an image retrieval system is presented by exploiting the ODBTC encoded data stream to construct the image features, namely Color Co-occurrence and Bit Pattern features. As documented in the experimental results, the proposed scheme can provide the best average precision rate compared to various former schemes in the literature. As a result, the proposed scheme can be considered as a very competitive candidate in color image retrieval application.

Content based image retrieval system using clustered scale invariant feature transforms.

This paper proposes two new approaches for content-based image retrieval using Scale invariant feature transform method. The main goals of these two approaches are tackling two important drawbacks of SIFT method namely, its memory usage and its matching time, that prevents it to be used as a reliable method for CBIR problem. In the first approach using k-means clustering, the feature matrix extracted by SIFT method is clustered to 16 clusters so that instead of having a huge matrix with dimension of $k \times 128$, which k is the number of keypoints, we will have a reduced matrix of size 8×16. In the second approach using the fact that each keypoint is described by eight dominant directions, the feature matrix is reduced to a vector of size 1×18. In both methods there is no loss of discriminative power of SIFT while the size of feature matrix is highly reduced. The proposed approach shows high performance when facing with object images namely the images with an object in them.This fact is because of the concept of SIFT. However, for more complex images like scene images or images with cluttered and occluded background the performance of SIFT needs more improvement.

Lucene4IR: Developing Information Retrieval Evaluation Resources using Lucene.

This report provides an overview of the workshop on developing teaching and training resources for Information Retrieval evaluation. While, the event was informative and enjoyable, it was clear that there is lot of scope for developing greater links between industry and academia - and that tools like Lucene, Solr and ElasticSearch - provide an obvious way to foster collaboration. Furthermore, such toolkits, being supported by a large community, and widely used, mean that researchers working with such tools can effect more efficient knowledge transfer and their research can have greater impact.

Optimizing Top Precision Performance Measure of Content-Based Image Retrieval by Learning Similarity Function.

In this paper they study the problem of similarity learning for content-based image retrieval. The critical novelty of this work is it learns the similarity to maximize the top precision measure over the training set. They have modeled the learning problem as a support top irrelevant database image weighting problem to obtain the optimal similarity function. The experiments over some benchmark databases show its advantages over other similarity learning methods.

Image moment invariants as local features for content based image image retrieval using the Bag-of_Visual_Words model.

In this work a novel methodology for image retrieval is proposed. Its main novelty lies in the usage of moment invariants as descriptors of local image areas extracted using the SURF detector. Three different setup designs are considered in the current experimental study. In the first, one color affine moment invariants are calculated. In the second, the used invariants are calculated over the chromaticities of the original image, while in the third one, a normalization scheme takes place in order to widen the invariants' range of values. The setup two improves the performance of the one, while the setup three improves the performance of two.

Semantic content-based image retrieval: A comprehensive study.

This paper has presented a comprehensive survey on different techniques and recent research works in the CBIR domain. Firstly, this study has discussed the general retrieval framework which most of CBIR systems have adopted over the last 15 years. Secondly, the proposed approaches in each block of CBIR framework have been presented along with the impact of research shift from low-level to high-level processing. Thirdly, this study has identified the most recent advances that contribute in reducing the semantic gap. Finally, it has identified some important issues that directly affect CBIR systems as follows: (1) image representation with focus on local descriptors; (2) automatic image annotation which opens the research gate for action-based image retrieval; (3) dimensionality reduction and image indexing that directly affect the CBIR performance in terms of accuracy, speed, and memory usage; (4) deep learning as a promising approach for reducing the semantic gap, especially the use of deep CNNs for classification, distance metric learning, relevance feedback, and annotation; (5) a description of ideal image datasets used for training and testing in the CBIR context; (6) re-ranking approaches associated with relevance feedback as post-processing to minimize users intervention and satisfy their needs; and (7) visualization aspects and the importance of modeling CBIR systems that semantically handle and return the most relevant images with maximum user satisfaction.

Content-Based Image Retrieval Using Error Diffusion Block Truncation Coding Features.

A new method is proposed in this paper for color image indexing by exploiting the simplicity of the EDBTC method. A feature descriptor obtained from a color image is constructed from the EDBTC encoded data (two representative quantizers and its bitmap image) by incorporating the VQ. The CHF effectively represents the color distribution within an image, while the BHF characterizes the image edge and texture. The experimental results demonstrate that the proposed method is not only superior to the former BTC-based image indexing schemes but also to the former existing methods in the literature related to the CBIR. To achieve a higher retrieval accuracy, another feature can be added into the EDBTC indexing scheme with the other color spaces such as YCbCr, hue–saturation–intensity, and lab. An extension of the EDBTC image retrieval system can be brought to index video by considering the video as a sequence of images. This strategy shall consider the temporal information of the video sequence to meet the user requirement in the CBIR context.

Research Gap

While it is certainly possible in theory, producing a consistent and enjoyable search experience for digital shoppers through Elasticsearch requires making decisions on non-trivial issues, which in turn requires experience and specific knowledge. For example (in no particular order of importance/salience):
What is the difference between query context and filter context?
What is the default analyzer behavior on the query ‘I don’t like New York’? By the way, what is an analyzer again?
Which stemming process should I use? And why stemming and not lemmatization?
It is worth pointing out that this is not just a generic tech gap (‘you have to be Albert Einstein to figure it out’): even good engineers, without previous exposure to Information Retrieval, will have to go through the painful learning process. In other words, Elasticsearch is more a “tool” than a “product”, in the same way that, say, Postgres is. For Postgres to become an effective CRM for your sales team, some non-trivial work and non-obvious choices need to be made; similarly, for Elasticsearch to power a satisfactory shopping experience, some non-trivial setup and fine-tuning need to happen. Postgres and Elasticsearch are exceptional tools to build great customer-facing product, but they are not per se ready-made products.
Now that we know that Elasticsearch cannot be in itself the answer to the retailers’ problem, can we come up with a list of purely (or mostly) technical desiderata for a perfect user facing product?

Service oriented: the ideal solution should not involve any maintenance on the retailer side, nor hosting responsibilities, and should ideally be seamless to integrate (API-based integration comes to mind as the default choice). In the subscription economy, a SaaS model looks like the natural offer, but there could be exceptions for enterprise level services.

Search algorithm: matching keywords, tf-idf and the vector space model are all fine concepts — but this is not the Seventies anymore (unfortunately, one may say). If I don’t have more than 100 bucks for my new basketball shoes, I should be able to just type ‘nike shoes 100 dollars’ and get what I want.

No search fits all: search bars don’t care whether you are in Italy or the US, a new customer or a faithful client, a mobile-addicted millennial or a geek with her laptop. A smart search bar should be somewhat personal, intelligently adapting to user behavior and changing according to context: if I navigate to the women’s section of a website and then search for ‘shirts’, women’s shirts should be more relevant than men’s. Why can’t today’s search bars even learn that?

While there are well-established, successful companies out there already offering an out-of-the-box experience that is way better than Elasticsearch (two that we like a lot are Algolia and Fact Finder), it is hard to ignore that a paradigm shift is happening and that Artificial Intelligence is (and increasingly will be) the key for the future of search.

System Description

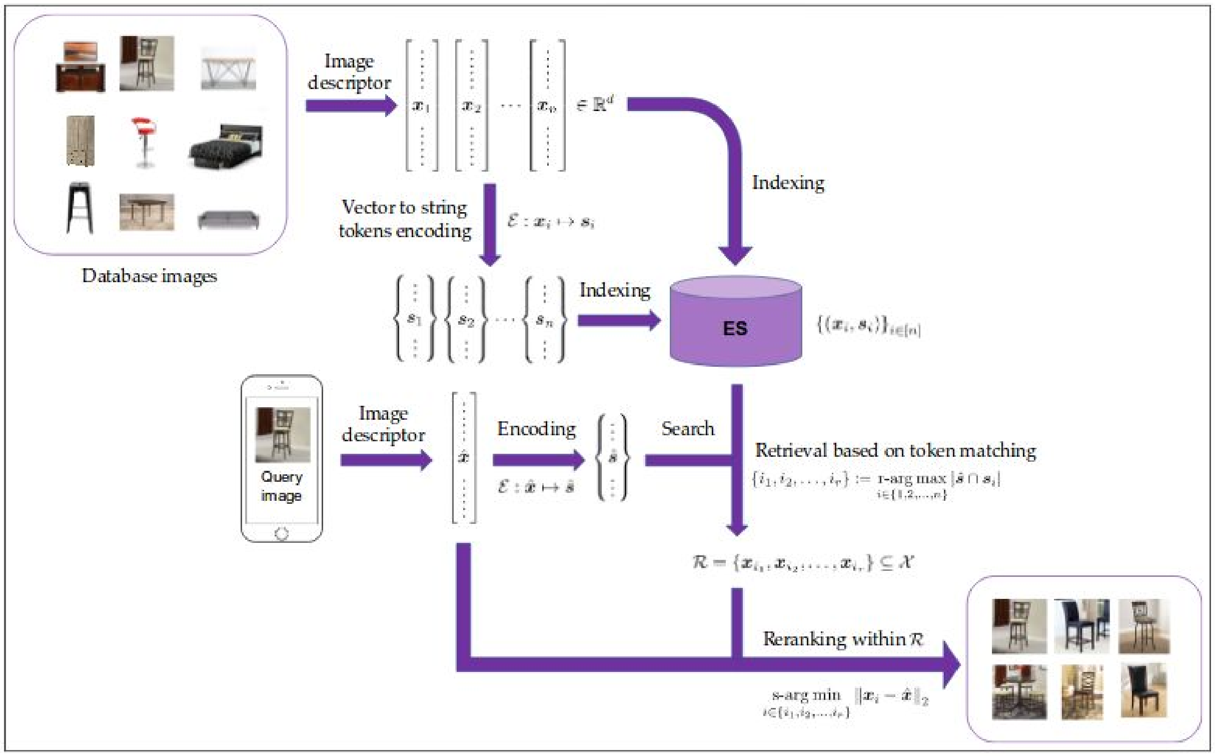


Image descriptor
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\cdots$ $\boldsymbol{x}_n$ $\in \mathbb{R}^d$
Indexing
Vector to string tokens encoding
$\mathcal{E}: \boldsymbol{x}_i \mapsto \boldsymbol{s}_i$
Database images
$\boldsymbol{s}_1$ $\boldsymbol{s}_2$ $\cdots$ $\boldsymbol{s}_n$
Indexing
ES
$\{(\boldsymbol{x}_i, \boldsymbol{s}_i)\}_{i\in[n]}$
Query image
Image descriptor
$\hat{\boldsymbol{x}}$
Encoding
$\mathcal{E}: \hat{\boldsymbol{x}} \mapsto \hat{\boldsymbol{s}}$
$\hat{\boldsymbol{s}}$
Search
Retrieval based on token matching
$\{i_1, i_2, \ldots, i_r\} := \text{r-arg max}_{i\in\{1,2,\ldots,n\}} |\hat{\boldsymbol{s}} \cap \boldsymbol{s}_i|$
$\mathcal{R} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_r}\} \subseteq \mathcal{X}$
Reranking within $\mathcal{R}$
$\text{s-arg min}_{i\in\{i_1,i_2,\ldots,i_r\}} \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}\|_2$

Dataset of images used:[**Dataset.**](#)

Steps:

1. Image Feature Extraction:

This is the first step of our process. In this step we basically convert each image into a set of feature vectors through the use of a pre-trained Artificial Neural Network Model (Resnet-18) .(an Image vector can be visualized as an list of floating point values) and then store all these vectors in a file. The length of the feature vector obtained is 512.
To extract these 512 features from the image, we leverage the Resnet18's last layer's outputs i.e. from the last fully connected layer of the neural

network which are stored in a list to be sent to be encoded in the form of string tokens by the encoder we have implemented in the next stage. In order to implement these steps we had used the deep learning library of PyTorch - to load the resnet model and apply it on our image.

```
[0.9052236676216125, 1.5989054441452026, 1.4928816556930542, 1.1170400381088257, 1.258564829826355, 3.025547504425049, 0.7741681933403015, 0.585510432
7201843, 1.2538460493087769, 1.4626930952072144, 0.49749475717544556, 1.4467765092849731, 0.6675364971160889, 0.40336138010025024, 0.1199135109782219,
 0.5142383575439453, 2.6689581871032715, 0.5518437027931213, 0.1576499044895172, 0.22838743031024933, 0.080537810921669, 0.08967635780572891, 0.082715
19094705582, 0.02607782743871212, 0.579940140247345, 0.6634822487831116, 1.0670286417007446, 2.0525877475738525, 0.3489430546760559, 0.031708110123872
76, 0.04725312814116478, 1.0358115434646606, 3.8930776119232178, 0.09400314092636108, 0.241555854678154, 1.1954625844955444, 0.707098126411438, 0.7419
471740722656, 0.7636706233024597, 0.08106584846973419, 0.4924722909927368, 1.5449178218841553, 0.055439334362745285, 0.497448593378067, 0.390547096729
27856, 0.5148037672042847, 0.6706290245056152, 0.4963374733924866, 0.4934752285480499, 0.9609358906745911, 0.05108841508626938, 1.2253389358520508, 0.
1779221147298813, 0.02945125661790371, 1.4510279893875122, 0.26881101727485657, 0.22174382209777832, 0.46156859397888184, 0.4459463357925415, 0.932009
5777511597, 0.549396812915802, 0.18617424368858337, 0.06735911965370178, 0.5370017886161804, 1.5352602005004883, 0.0, 1.396571397781372, 0.35605075955
39093, 0.5514580607414246, 0.04599640890955925, 0.1679290235042572, 0.029113585129380226, 0.661171555519104, 0.19920384883880615, 0.4909878969192505,
0.39613452553749084, 0.8702415227890015, 1.8368922472000122, 2.1497409343719482, 0.3301388621330261, 1.5928276777267456, 1.3660151958465576, 0.4443849
9212265015, 1.0949647426605225, 0.23253649473190308, 1.2999153137207031, 0.7211756706237793, 0.5465493202209473, 0.4033564627170563, 0.046460624784231
186, 0.7544358968734741, 0.4016079306602478, 1.0064092874526978, 2.3742306232452393, 0.1488771289587021, 0.10881271958351135, 0.508345365524292, 1.121
073842048645, 0.9697357416152954, 1.2484723329544067, 0.0740552619099617, 2.0109033584594727, 0.32175716757774353, 2.177133798599243, 0.30634364485740
66, 0.48806053400039673, 0.5360159277915955, 0.4663271903991699, 0.08857949823141098, 0.3109065294265747, 0.4109047055244446, 1.1916090250015259, 0.48
86322021484375, 1.0126672983169556, 0.025090061128139496, 1.5673786401748657, 2.3829312324523926, 1.3468387126922607, 0.0243082195520401, 2.1102743148
80371, 0.3309817910194397, 0.006817698013037443, 4.14601469039917, 0.8812322616577148, 0.032072119414806366, 0.9051386713981628, 0.013377987779676914,
 2.020986318588257, 3.187628984451294, 0.6417372822761536, 0.7006428837776184, 1.5842115879058838, 0.3574197292327881, 0.9765315055847168, 1.280908823
0133057, 1.502949595451355, 0.8370804786682129, 0.0901520699262619, 0.42836999893188477, 0.8537828922271729, 1.406243920326233, 0.2715785801410675, 0.
5576869249343872, 0.0395902834897041, 0.6468229293823242, 0.44416213035583496, 0.25812846422195435, 1.3550182580947876, 0.1794312298297882, 0.0853976
160287857, 0.37867841124534607, 0.8328068852424622, 0.29563331604003906, 0.9307411909103394, 0.718416690826416, 1.3734066486358643, 0.0448962152004241
94, 0.3446524739265442, 0.3362923562526703, 0.12955908477306366, 0.5710390210151672, 0.7167191505432129, 0.18123570084571838, 1.2756636142730713, 0.29
15724217891693, 0.009806386195123196, 0.4635046422481537, 3.1952524185180664, 0.4411658048629761, 0.7246800661087036, 1.7954615354537964, 0.4471054375
1716614, 0.8723710775375366, 0.42392224073410034, 0.765166699886322, 0.15570713579654694, 2.479724884033203, 0.339632511138916, 0.16238898038864136, 2
.522204875946045, 1.2589902877807617, 0.2122936099767685, 1.9455183744430542, 0.467572957277298, 0.625236988067627, 0.45483899116516113, 0.49427580833
43506, 0.6505541801452637, 0.14541207253932953, 0.45986124873161316, 1.3767045736312866, 0.16124257445335388, 1.395393967628479, 0.3905797004699707, 0
.06971696764230728, 0.27656760811805725, 0.21249833703041077, 0.3441259562969208, 0.3770764470100403, 0.2528552711009979, 0.12006359547376633, 0.28988
88885974884, 0.5712056159973145, 0.30389147996902466, 0.9045536518096924, 3.4061341285705566, 0.12867440283298492, 2.223106861114502, 0.47232726216316
223, 0.7769080996513367, 0.586260974407196, 0.3777961730957031, 1.4470545053482056, 0.5205171704292297, 0.9935681819915771, 2.058337926864624, 0.68263
42940330505, 0.0, 0.14099986851215363, 1.3196003437042236, 0.648614227771759, 0.5920573472976685, 0.46030092239379883, 0.2975245416164398, 0.053098198
026418686, 0.0489933155477047, 0.9783178567886353, 0.17435482144355774, 1.0512229204177856, 0.10817209631204605, 1.0582209825515747, 0.500192403793335
, 0.005015663802623749, 2.6272289752960205, 0.9245918393135071, 2.0913443565368652, 0.6172261238098145, 0.897724986076355, 0.31434473395347595, 0.1748
8229274749756, 1.466507077217102, 0.4443501830101013, 0.5811530351638794, 0.43653133511543274, 0.16012215614318848, 0.12751509249210358, 0.11047609150
409698, 1.4117553234100342, 0.8125691413879395, 1.0820896625518799, 0.2926781177520752, 0.09709252417087555, 0.025064455345273018, 1.640019416809082,
0.12529908120632172, 1.2924973964691162, 0.005193499382585287, 1.5277831554412842, 2.2190916538238525, 0.40292850136756897, 1.2334367036819458, 1.8686
553239822388, 0.6097294688224792, 0.36888864636421204, 0.6721209287643433, 2.4443020820617676, 1.3098376989364624, 1.277597188949585, 1.28171467781066
9, 2.200049638748169, 0.08379030227661133, 0.04374231770634651, 0.34104830026626587, 0.04334179311990738, 0.10949111729860306, 0.5665532946586609, 0.0
2998104877769947, 0.25262653827667236, 3.508409261703491, 0.6627151966094971, 0.09119167178869247, 1.2441883087158203, 0.31363368034362793, 0.69913220
40557861, 0.8698427677154541, 0.4784395396709442, 2.0363266468048096, 0.3037959039211273, 0.3142200708389282, 1.1624882221221924, 0.3621387481689453,
0.7663037180900574, 0.25633955001831055, 0.19516372680664062, 0.5697951912879944, 1.5156997442245483, 1.1643575429916382, 0.15047304332256317, 0.45985
162258148193, 1.5646628141403198, 0.017956240102648735, 0.8099767565727234, 0.2335776835680008, 0.712037205696106, 2.552124261856079, 0.77731734514236
45, 0.9875128865242004, 1.1005730628967285, 0.10303216427564621, 0.059074148535728455, 0.6539087295532227, 0.6128044128417969, 3.4068732261657715, 0.8
```

2. Encoding of Feature Vectors

This is the second step which involves two steps:

1. Rounding

In this step we retrieve the image feature vectors which we have previously stored in a file and then using technique known as element-wise rounding , we convert all the image feature vectors into rounded image feature vectors(basically rounding the float values of the image feature vectors).

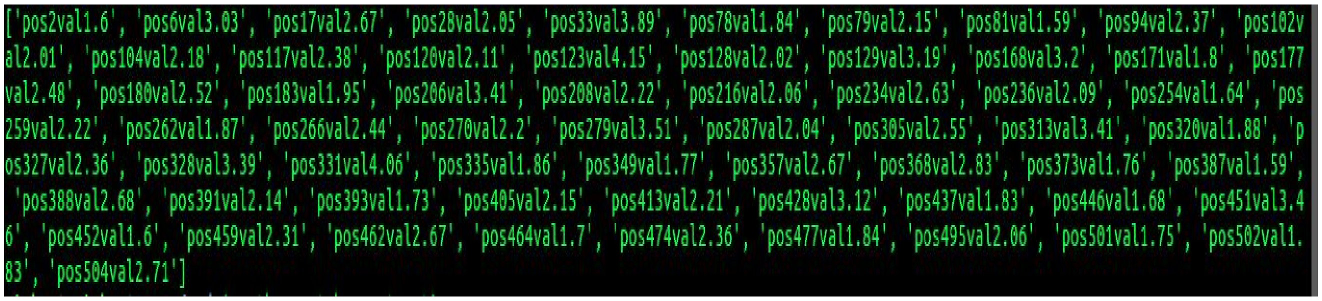
Explanation of Element wise rounding encoder:

The element-wise rounding encoder rounds each value in the numerical vector to p decimal places (where p ≥ 0 is a fixed integer), and then

concatenates its positional information and rounded value as the string tokens.
Example 1.Rounding Step with p=2
Initial image feature vector X=[0.1234,0.1254,0.4556] , now after rounding to p places.
Image feature vector after rounding to X’=[0.12,0.12,0.45]

Note: In this step we have also added strings like pos to every rounded feature of the vector but for explanation simplicity we can excluded those strings for now.
Actual Image feature vector after rounding to
X’=[pos0.12,pos0.12,pos0.45]
Also note that we are here considering that the image vectors before and after rounding do retain their positional order otherwise it would result in a very large permutation of vectors which would be difficult for elasticsearch to index .

2. Filtering
This is the second part of our encoder .It basically selects the top m rounded feature values (m can be varied so that it gives the best accuracy/evaluation feature we want to).
For example :
If m=2 then if X’=[0.12,-0.23,0.54], then we select top 2 highest values image vectors.So after filtering the final encoded image feature vector will be :X’’=[-0.23,0.54].



```
['pos2val1.6', 'pos6val3.03', 'pos17val2.67', 'pos28val2.05', 'pos33val3.89', 'pos78val1.84', 'pos79val2.15', 'pos81val1.59', 'pos94val2.37', 'pos102v
al2.01', 'pos104val2.18', 'pos117val2.38', 'pos120val2.11', 'pos123val4.15', 'pos128val2.02', 'pos129val3.19', 'pos168val3.2', 'pos171val1.8', 'pos177
val2.48', 'pos180val2.52', 'pos183val1.95', 'pos206val3.41', 'pos208val2.22', 'pos216val2.06', 'pos234val2.63', 'pos236val2.09', 'pos254val1.64', 'pos
259val2.22', 'pos262val1.87', 'pos266val2.44', 'pos270val2.2', 'pos279val3.51', 'pos287val2.04', 'pos305val2.55', 'pos313val3.41', 'pos320val1.88', 'p
os327val2.36', 'pos328val3.39', 'pos331val4.06', 'pos335val1.86', 'pos349val1.77', 'pos357val2.67', 'pos368val2.83', 'pos373val1.76', 'pos387val1.59',
 'pos388val2.68', 'pos391val2.14', 'pos393val1.73', 'pos405val2.15', 'pos413val2.21', 'pos428val3.12', 'pos437val1.83', 'pos446val1.68', 'pos451val3.4
6', 'pos452val1.6', 'pos459val2.31', 'pos462val2.67', 'pos464val1.7', 'pos474val2.36', 'pos477val1.84', 'pos495val2.06', 'pos501val1.75', 'pos502val1.
83', 'pos504val2.71']
```

3. Indexing, Searching and Retrieval

Indexing

Given image feature vectors $X := \{x1, x2, . . . , xn \} \subseteq Rd$, we will first encode them into string tokens $S := \{s1, s2, . . . , sn \}$, where $si := E(xi)$ for some encoder $E(\cdot)$ converting a d dimensional vector into a collection of string tokens of cardinality m. The original numerical vectors X and encoded tokens S, together with their textual metadata (e.g, product titles, prices, attributes), will be all indexed into the Elasticsearch database, to wait for being searched.

To implement the indexing step, we call Elasticsearch RESTful API's built over HTTP using elasticsearch-py library, and index the encoded strings we had obtained from the feature vectors extracted in the preceding step. To better focus on the comparison of the efficacy in encoding scheme, only vanilla setting of Elasticsearch (one shard and
zero replica) is used in creating each index.

Searching and Re-Ranking

The search phase consists of two steps: retrieval and reranking.

Retrieval

Given a query vector $xˆ$, we will first encode it into $sˆ := E(xˆ)$ via the same encoder used in indexing, and retrieve r ($r \ll n$) most similar vectors $R := xi1 , xi2 , . . . , xir$ as candidates based on the overlap between the string token set $sˆ$ and the ones in $\{s1, s2, . . . , sn \}$, i.e., $\{i1,i2, . . . ,ir \}$ = r-arg max $i \in\{1,2,...,n\} |sˆ \cap si |$.

Re-ranking

We will then re-rank vectors in the candidate set R according to their exact Euclidean distances with respect to the query vector $xˆ$, and choose the top-s ($s \leq r$) ones as the final visual search result to output, i.e., s-arg min i

$\in \{i_1, i_2, ..., i_r\}$ $\| x_i - \hat{x} \|_2$. As expected, the choice of E(·) is extremely critical to the success of the above approach.

Evaluation criterion and Experimental Results

To measure ad hoc information retrieval effectiveness in the standard way, there is the need for a test collection consisting of three things:

1. a document collection
2. a test suite of information needs, expressible as queries
3. a set of relevance judgments, standardly a binary assessment of either relevant or nonrelevant for each query-document pair.

The standard approach to information retrieval system evaluation revolves around the
notion of relevant and nonrelevant documents. With respect to a user information need,
a document in the test collection is given a binary classification as either relevant or nonrelevant. This decision is referred to as the gold standard or ground truth judgment of relevance.
The most frequent and basic measures for information retrieval effectiveness are reported below.

Precision (P)
It is the fraction of retrieved documents that are relevant:
P =
#(relevant items retrieved)/#(retrieved items)
= P(relevant|retrieved)

Recall (R)
It is the fraction of relevant documents that are retrieved:
R =
#(relevant items retrieved)/#(relevant items)
= P(retrieved|relevant)

Mean Average Precision (mAP)
For a single information need, Average Precision is the average of the precision value

obtained for the set of top k documents existing after each relevant document is
retrieved, and this value is then averaged over all the information needs.

Settings:
Our image datasets consists of 16K images selected from Stanford's car dataset. For each image, we extract its image feature vector using the pre-trained ResNet Model. In specific, each image is embedded into a vector by taking the output from the penultimate layer (i.e., the last average pooling layer) of the neural network model. String tokens are produced respectively with encoding schemes at different configurations. For the element-wise rounding encoder, we select p={1,2,3} and m={32,64,128,256}.
.

Evaluation:
To evaluate the element-wise encoding schemes, we randomly select 100 images to act as our visual queries. For each of the query image, we find the set of its 24 nearest in Euclidean distance, which is treated as gold standard We use Precision@24 , which measures the over-
lap between the 24 images retrieved from Elasticsearch (with r∈{24,48,72 so on} respectively) and the gold standard, to evaluate the retrieval efficacy of different encoding methods under various settings.

Results:

We report the Precision@24 y averaged over the 100 queries randomly selected.
When we require the search latency to be smaller than 0.3 second, the Element-wise encoder is able to achieve an average Precision@24 as 81% which is less than what is achieved by the subvector-wise encoding scheme.

Conclusions and future work

Although our Element-wise rounding encoder is outperformed by a Sub-vector-wise clustering encoder (one which splits image feature vectors into sub-vectors , applies k means clustering to classify them and then does the searching and retrieval step) , our encoder has a certain advantage over this one that it is not restrictive to enforce a vector to be divided into subvectors exclusively which could potentially downgrade the performance of the encoder.

Also , some sort of preprocessing through a linear operation before applying the encoding to the vectors should probably be there , which we believe would make our encoding scheme more robust and adaptive with respect to different image feature vectors extracted from various image descriptors.We can also use some other neural network unlike resnet for image feature extraction which might would have increased the precision.