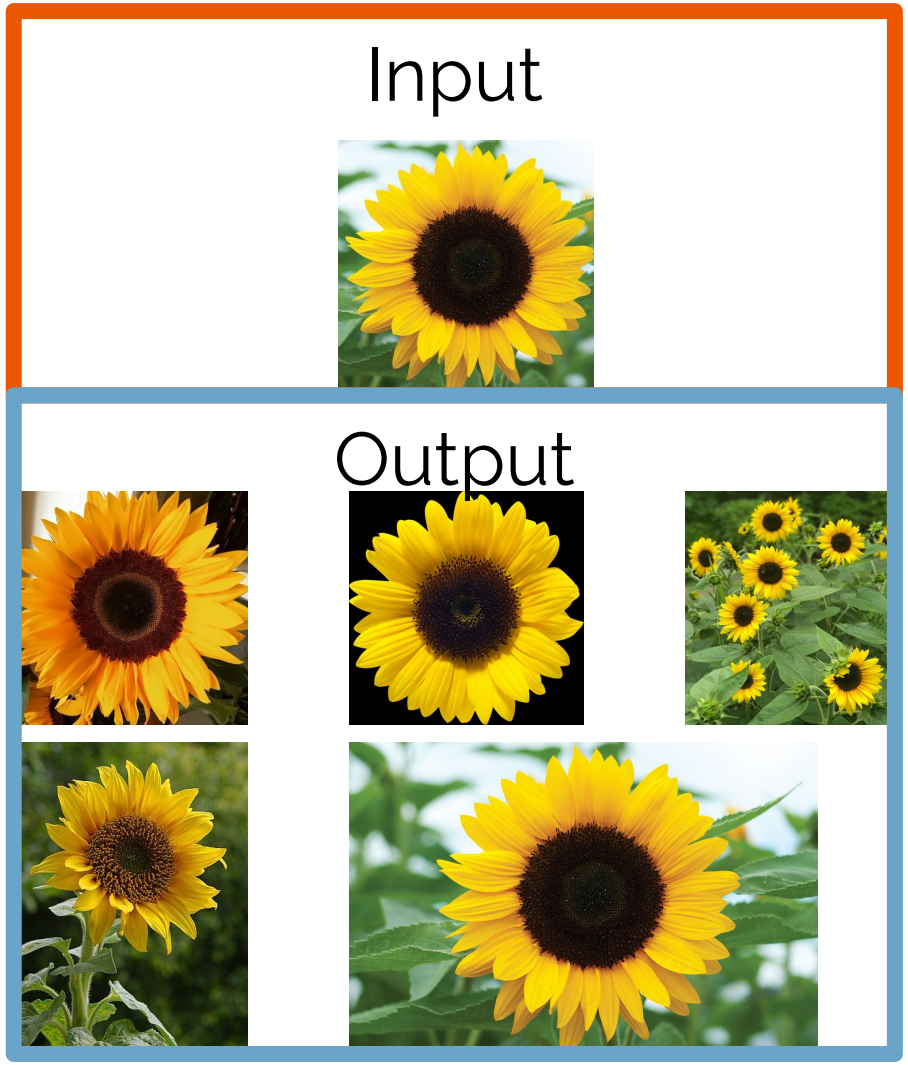
Image Retrieval

Jesus Cantu & Nicholas Synovic

What is Image Retrieval?

- **Search** for similar images based on input
  - Input can be a a full image or a subset of pixels
- **Search** is typically confined to a specific heuristic
  - Similarity, color space, frequency of pixel values

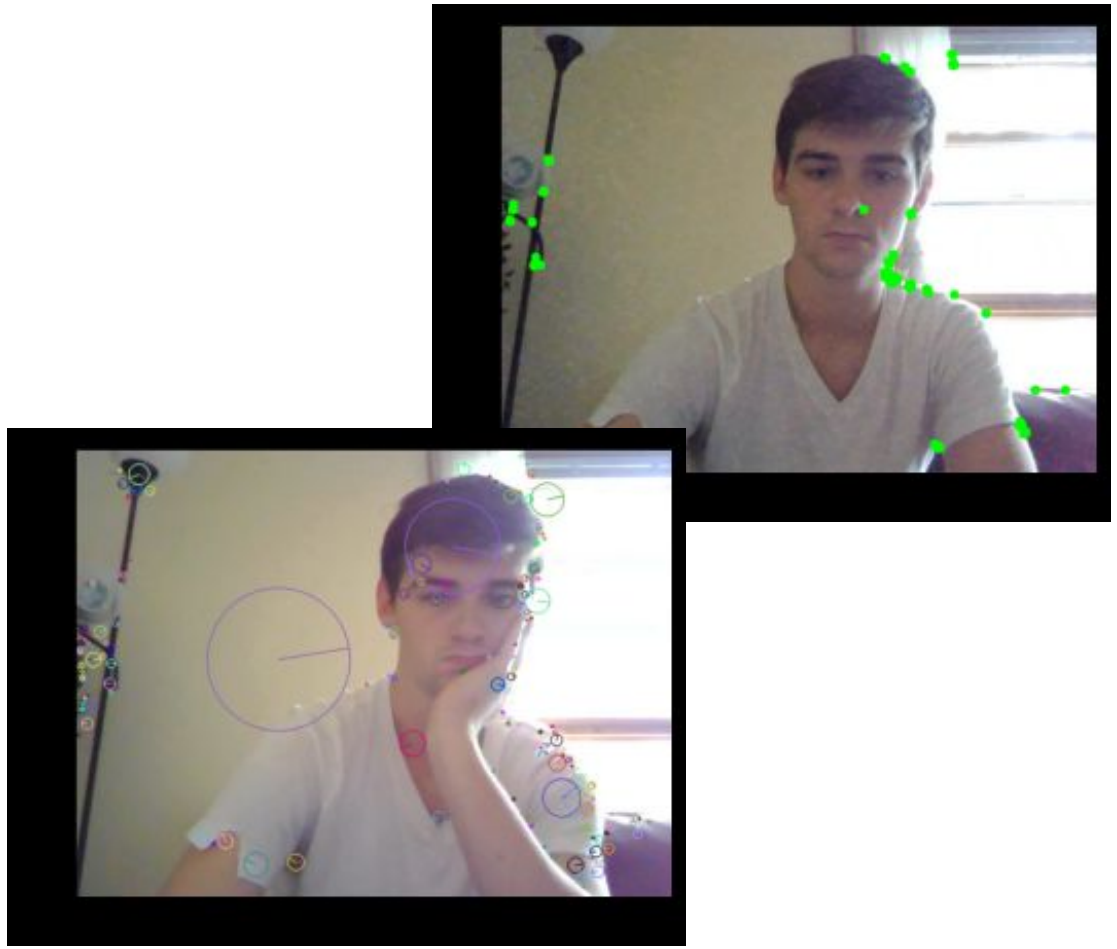


What is “search”?

- Merriam-Webster definition:
  - To look into or over carefully or thoroughly in an effort to find or discover something: such as
    - **to examine in seeking something**
    - To look through or explore by inspecting possible places of concealment or investigating suspicious circumstances
    - **To read thoroughly : check**
    - To examine a public record or register for information about land titles
    - To examine for articles concealed on the person
    - To look at as if to discover or penetrate intention or nature
- Image Retrieval involves *both examining and checking* data for heuristics

What does Image Retrieval *examine*?

- Examine an input image for a heuristic
  - Color space, pixel values, pixel subsets, local and global features, etc.
    - SIFT, SURF, DELF, Harris Corner Detection
- Examine collection of images for the same heuristic



What does Image Retrieval *check*?

- Check heuristic(s) from an input image against the heuristic(s) from the collection of images
  - A comparison with a threshold

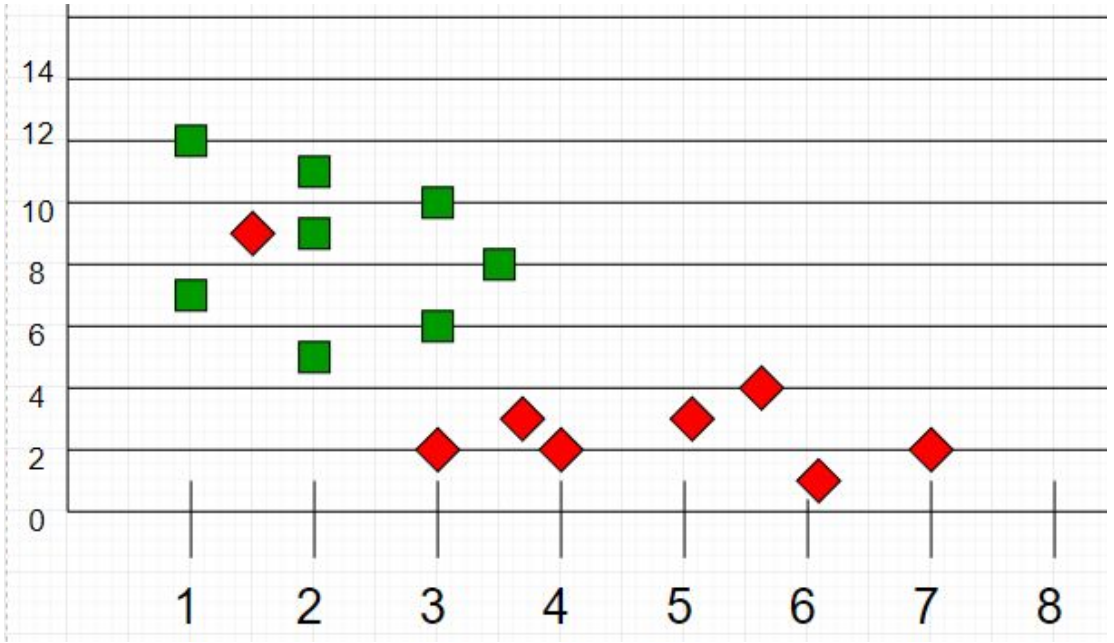
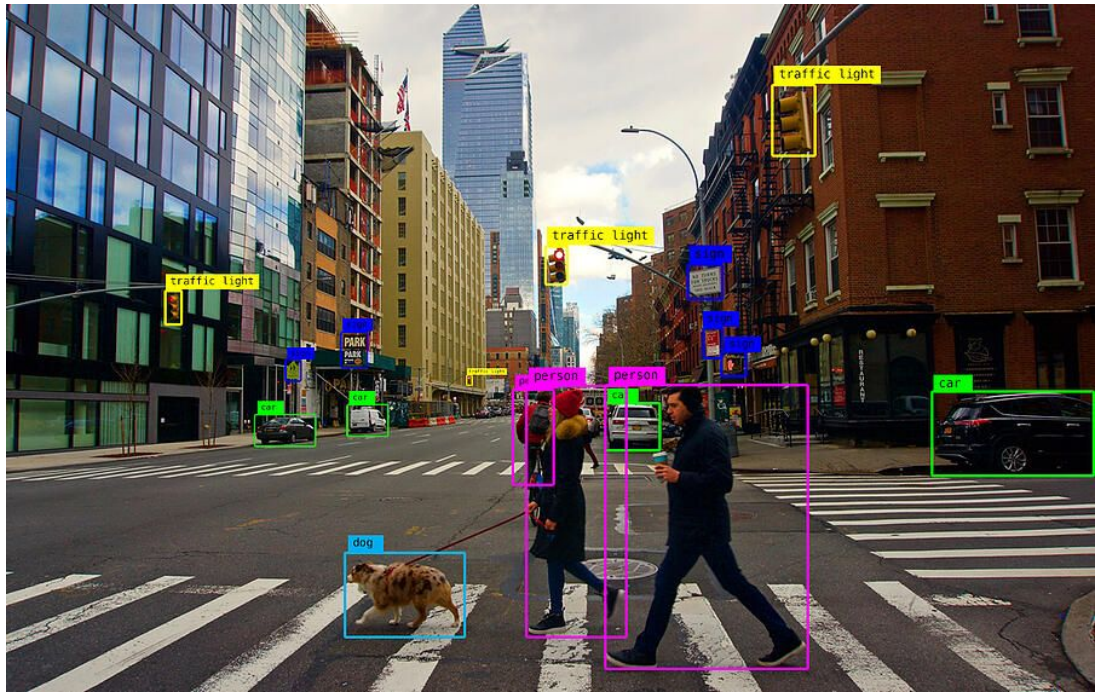


Image Retrieval is a Task

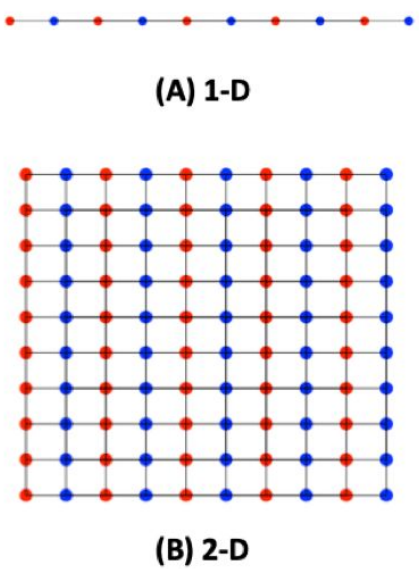
Image Retrieval != Object Classification && Image Retrieval != Object Detection

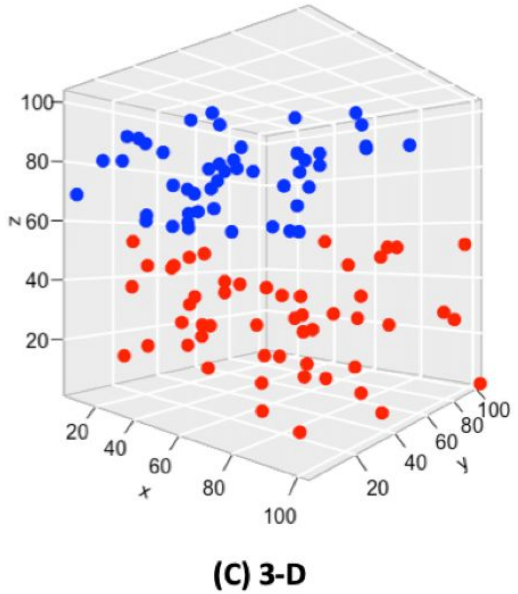
- Image Retrieval searches for images based off of heuristics
  - The subject of the image (object) isn't a concern
  - Heuristics about the objects in the image are of importance
- Does not utilize Image Classification
  - … But can be bolted on as an additional heuristic to search on
- The Image Retrieval process is different than the Object Identification and Classification processes



Curse of Dimensionality with Image Retrieval

- Curse of Dimensionality
  - More heuristics may not mean better results
  - Computational complexity increases with more heuristics
- Different approaches to choosing the best heuristics for analysis





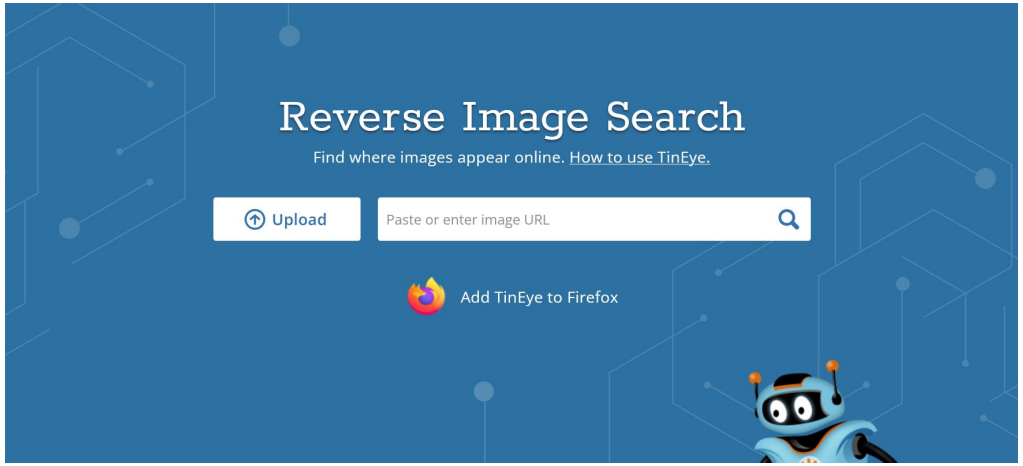
Recall vs Precision for Search

- Recall
  - What proportion of actual positives was identified correctly?
  - In other words: What percentage of results are actually relevant to the search query?
  - Focussed on returning as many relevant images
- Recall is used to benchmark search functions

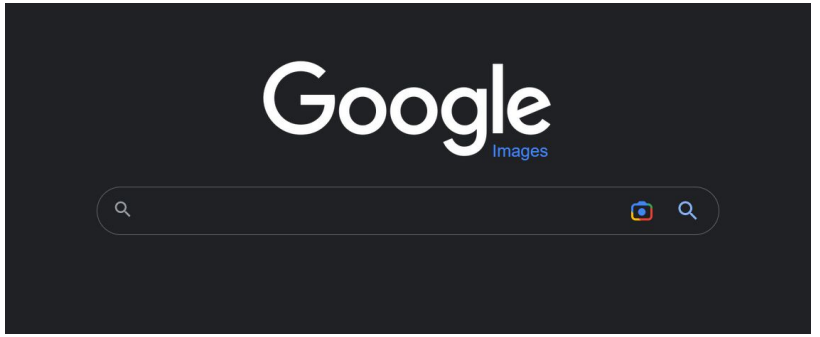
- Precision
  - What proportion of positive identifications was actually correct?
  - In other words: What percentage of relevant results **are the most** important?
  - Focussed on finding the most relevant image

Implementations

- TinEye



- Google Reverse Image Search



- Academic Research (Papers with Code's measurements)
  - 445 papers
  - 30 benchmarks
  - 51 datasets

Product quantization for nearest neighbor search

Overview

- By quantizing vectors into a different space, nearest neighbors can be quickly **approximated**
  - Quantizing example:
    - float32 -> float16 -> int8 ->int4 -> bool
    - Is destructive
- Product Quantization is used to generate a large number of centroids from a small input
  - Centroids used to cluster data
- Search is calculated by either the Euclidean Distance or Squared Distance between vectors in a dataset
  - Symmetric distance computation compares the centroids of two vectors to determine distance
  - Asymmetric distance computation looks only at one centroid

Example of Quantization

- https://qr.page/g/1sh0DjxLtAF

Background

- Data can be clustered by its Euclidean Distance to other data points (nearest neighbor)
  - Pythagoras Theorem
    - $c = \sqrt{(a2 + b2)}$, solve for c
  - Expensive due to the curse of dimensionality
- Clusters can be created by finding the approximate nearest neighbor
- Quantization is applied to reduce the memory usage of nearest neighbor algorithms
  - Nearest neighbor algorithms scale poorly with data

- Product quantization allows for the reduction of 128 dimensions from SIFT into groups of components that are then quantized
  - Instead of individual values being quantized or weighted over one another, vector components are grouped together and quantized from there
  - Is memory efficient

Problems

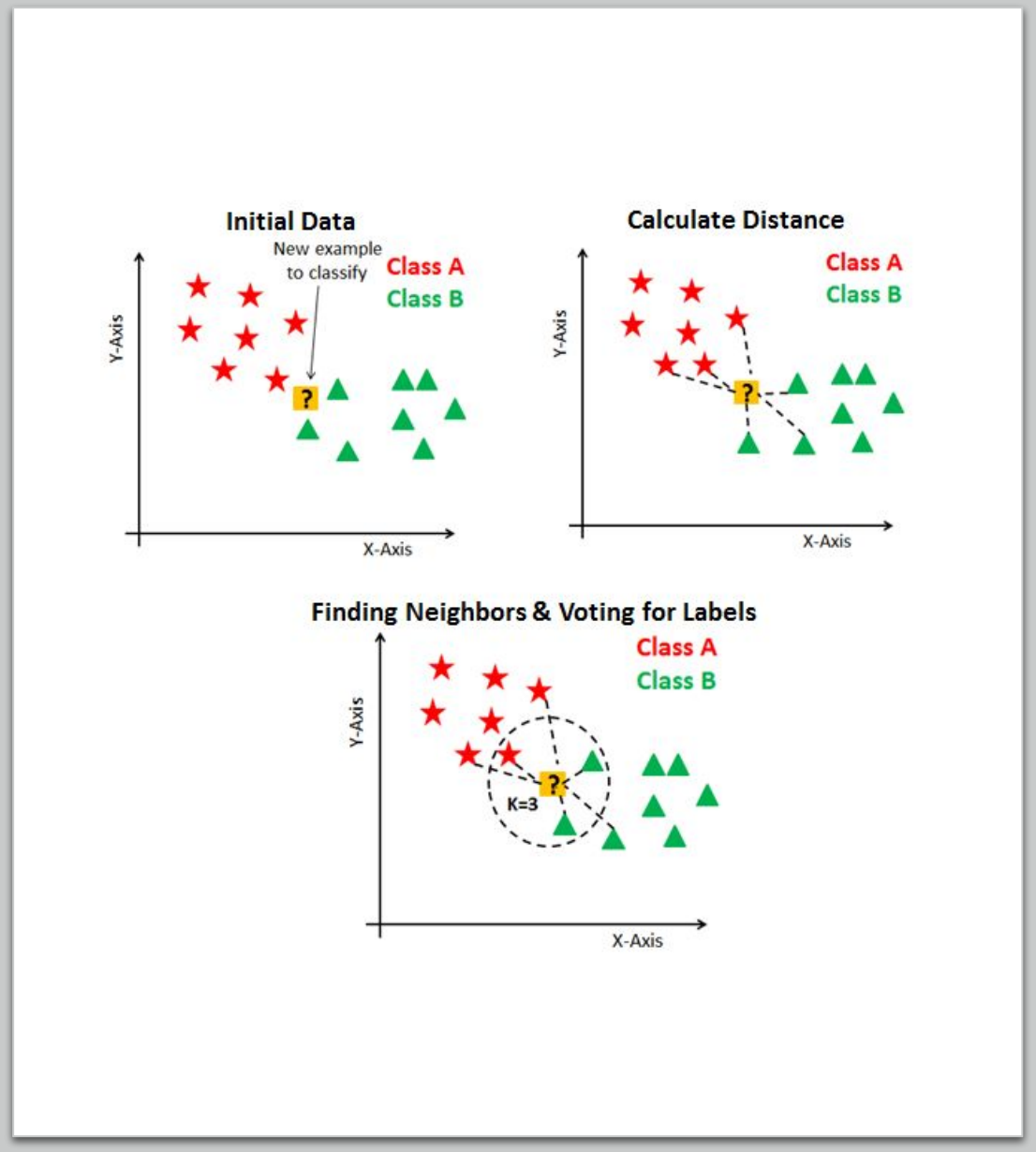
- **Defining Accurate Similarity Measures**
- Saying that a database object is the "nearest neighbor" of the query implies that we have a way to measure distances between the query and database objects.
- The way we choose to measure distances can drastically affect the accuracy of the system. At the same time, defining a good distance measure can be a challenging task.
- For example, what is the right way to measure similarity between two Web pages? A research problem that we are very interested in is designing methods for automatically learning a distance measure given many examples of pairs of similar objects and pairs of dissimilar objects.

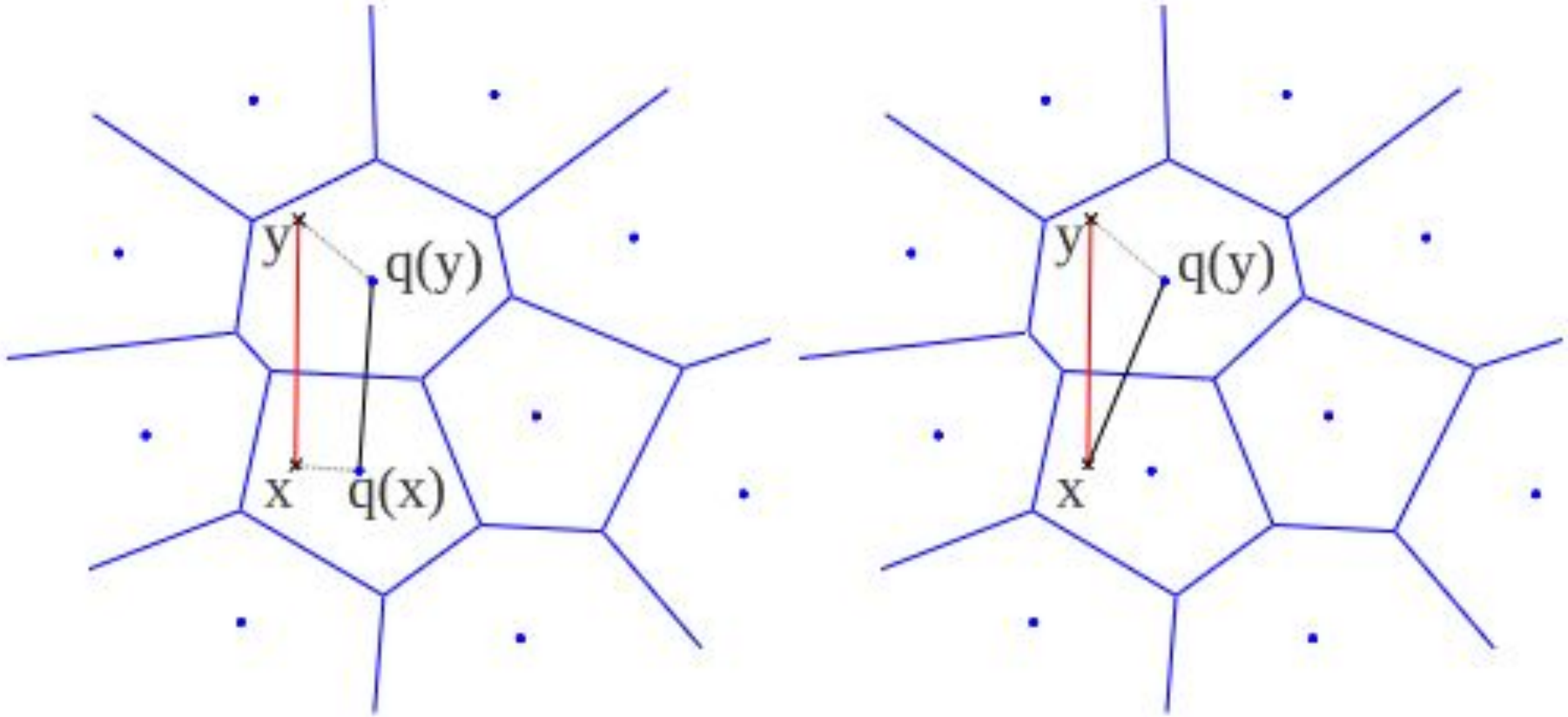
Problems contd.

- **Efficient Retrieval**
- Finding the nearest neighbors of the query can be time-consuming, especially when we have a large database. The problem can be even worse when the distance measure we use is computationally expensive.
- At the same time, computationally expensive distance measures are often used in computer vision and pattern recognition in general. As knowledge expands in many different domains, and ever larger databases are used to store that knowledge, achieving efficient retrieval becomes increasingly important, and at the same time increasingly challenging.

K-Nearest Neighbors

- The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these.
  - K-nearest neighbors is a naive approach
- Description of K-Nearest Neighbor models
  - https://www.youtube.com/watch?v=4HKqjENq9OU&t=402s





symmetric case asymmetric case

Fig. 2. Illustration of the symmetric and asymmetric distance computation. The distance $d(x, y)$ is estimated with either the distance $d(q(x), q(y))$ (*left*) or the distance $d(x, q(y))$ (*right*). The mean squared error on the distance is on average bounded by the quantization error.

Results

- Got state of the art performance on a 2 billion dataset by utilizing product quantization
- Tested symmetric (SDC) and asymmetric distance computation (ADC)
    - Found that ADC is computationally faster than SDC
- Implemented a method to perform non-exhaustive search efficiently
    - Use a coarse grained quantizer to estimate distances from *chunks* of an image. Then fine grained with product quantization
    - Relies on an Inverted File Asymmetric Distance Computation (IVFADC)
- On the SIFT and GIST datasets, SDC, ADC, and IVFADC outperform existing solutions
    - SDC, ADC, and IVFADC beat out Hamming embedding codes
    - IVFADC beats out FLANN

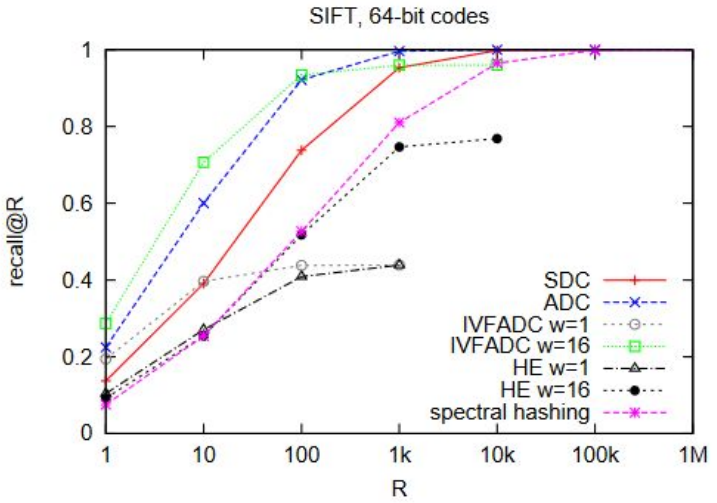


Fig. 8. SIFT dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC, spectral hashing [19] and HE [20]. We have used m=8, k^*=256 for SDC/ADC and k'=1024 for HE [20] and IVFADC.

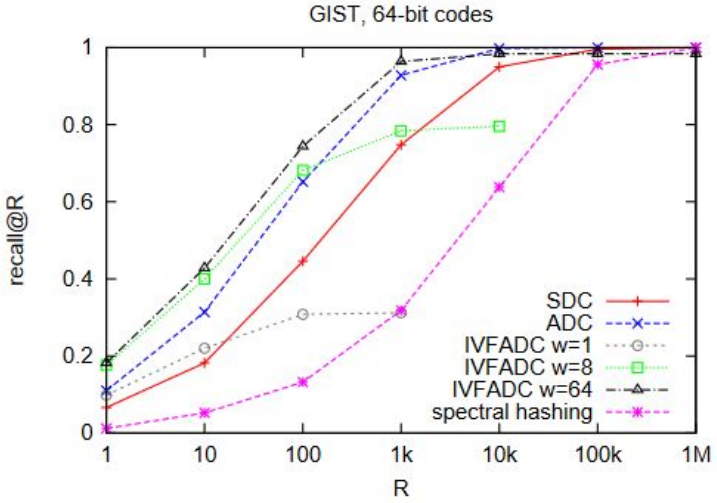


Fig. 9. GIST dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC and spectral hashing [19]. We have used m=8, k^*=256 for SDC/ADC and $k' = 1\,024$ for IVFADC.

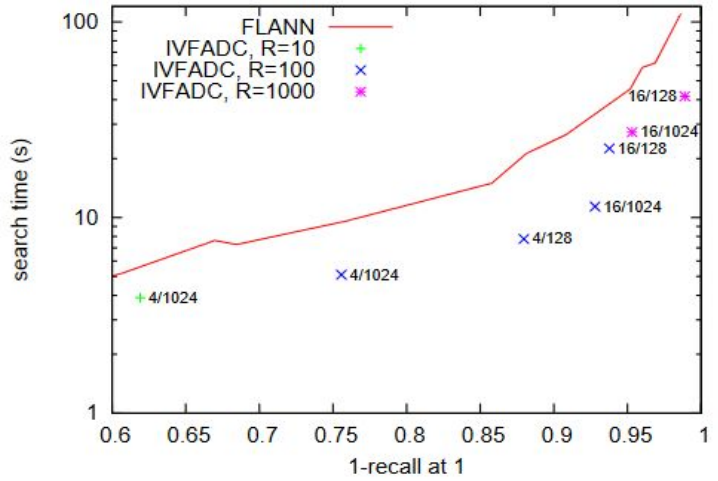


Fig. 10. IVFADC *vs* FLANN: trade-offs between search quality (1-recall@1) and search time. The IVFADC method is parametrized by the shortlist size R used for re-ranking the vector with the L2 distance, and the two parameters w and k' of the inverted file, which correspond to the number of assignments and to the number of coarse centroids.

method	parameters	search time (ms)	average number of code comparisons	recall@100
SDC		16.8	1 000 991	0.446
ADC		17.2	1 000 991	0.652
IVFADC	k'= 1 024, w=1	1.5	1 947	0.308
	k'= 1 024, w=8	8.8	27 818	0.682
	k'= 1 024, w=64	65.9	101 158	0.744
	k'= 8 192, w=1	3.8	361	0.240
	k'= 8 192, w=8	10.2	2 709	0.516
	k'= 8 192, w=64	65.3	19 101	0.610
SH		22.7	1 000 991	0.132

TABLE V

GIST DATASET (500 QUERIES): SEARCH TIMINGS FOR 64-BIT CODES AND DIFFERENT METHODS. WE HAVE USED m=8 AND k^*=256 FOR SDC, ADC AND IVFADC.

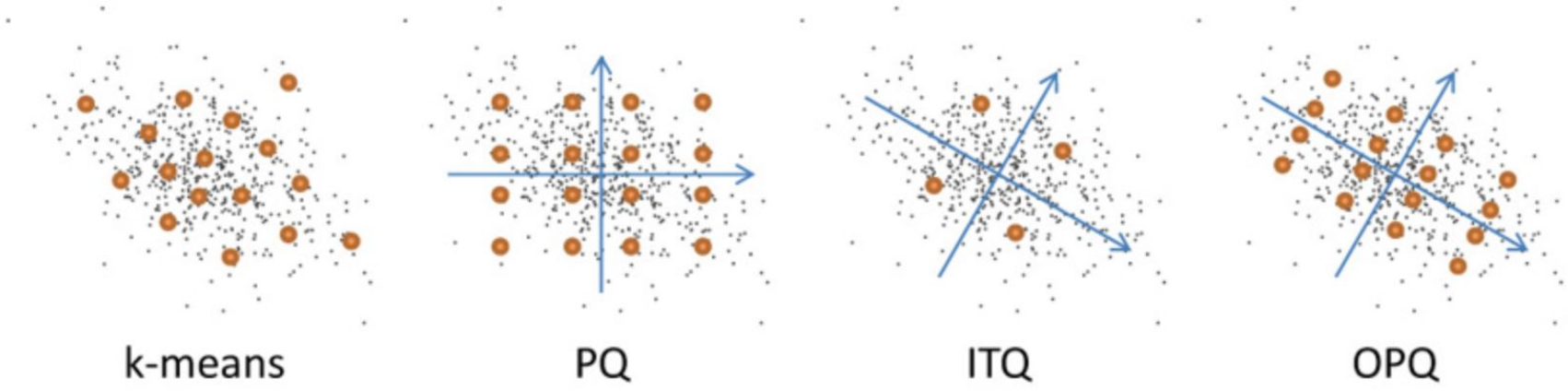
Discussion Questions

- When have you quantized data too far?
- With respect to images, what data can reliably be quantized without degrading recall performance?
- What other examples of quantization are regularly implemented?
  - In general, not just in CV

Optimized Product Quantization (OPQ)

WHAT IS PQ?

- Product quantization (PQ) is an effective vector quantization method. A product quantizer can generate an exponentially large codebook at very low memory/time cost.
- The essence of PQ is to decompose the high-dimensional vector space into the Cartesian product of subspaces and then quantize these subspaces separately. The optimal space decomposition is important for the PQ performance, but still remains an unaddressed issue.



WHAT IS OPQ?

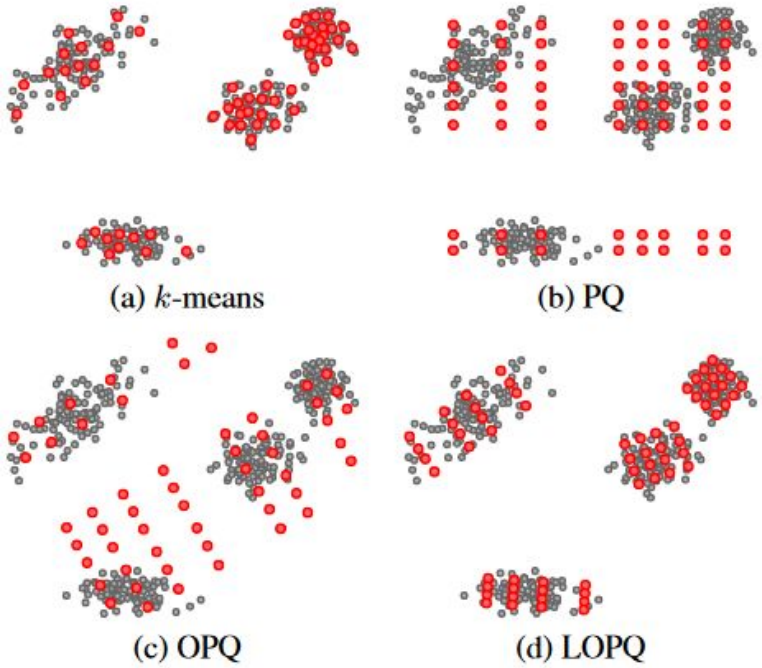
- In their paper Tiezheng Ge et al. 2014, optimize PQ by minimizing quantization distortions w.r.t the space decomposition and the quantization codebooks. We present two novel solutions to this challenging optimization problem.
- The first solution iteratively solves two simpler sub-problems. The second solution is based on a Gaussian assumption and provides theoretical analysis of the optimality. We evaluate our optimized product quantizers in three applications: (1) compact encoding for exhaustive ranking, (2) building inverted multi-indexing for non-exhaustive search, and (3) compacting image representations for image retrieval

TO DO: OPQ IMPLEMENTATION (FB)

Locally Optimized Product Quantization for Approximate Nearest Neighbor Search

Overview

- Authors proposed a new method for clustering data
  - Locally Optimized Product Quantization (LOPQ)
- Achieves state of the art performance on a variety of datasets
  - Centroids focus on reducing *distortion* rather than *increasing coverage*



Locally Optimized Product Quantization (LOPQ)

- Is a hierarchical quantization algorithm that produces codes of configurable length for data points.
- These codes are efficient representations of the original vector and can be used in a variety of ways depending on application, including as hashes that preserve locality, as a compressed vector from which an approximate vector in the data space can be reconstructed, and as a representation from which to compute an approximation of the Euclidean distance between points.
- Conceptually, the LOPQ quantization process can be broken into 4 phases. The training process also fits these phases to the data in the same order.

https://youtu.be/RgxCaiQ-kig?t=2059

TO DO: CONTINUE EXPLANATION

1. The raw data vector is PCA'd to D dimensions (possibly the original dimensionality). This allows subsequent quantization to more efficiently represent the variation present in the data.
2. The PCA'd data is then product quantized [2] by two k-means quantizers. This means that each vector is split into two subvectors each of dimension D / 2, and each of the two subspaces is quantized independently with a vocabulary of size V. Since the two quantizations occur independently, the dimensions of the vectors are permuted such that the total variance in each of the two subspaces is approximately equal, which allows the two vocabularies to be equally important in terms of capturing the total variance of the data. This results in a pair of cluster ids that we refer to as "coarse codes".
3. The residuals of the data after coarse quantization are computed. The residuals are then locally projected independently for each coarse cluster. This projection is another application of PCA and dimension permutation on the residuals, and it is "local" in the sense that there is a different projection for each cluster in each of the two coarse vocabularies. These local rotations make the next and final step, another application of product quantization, very efficient in capturing the variance of the residuals.
4. The locally projected data is then product quantized a final time by M subquantizers, resulting in M "fine codes". Usually the vocabulary for each of these subquantizers will be a power of 2 for effective storage in a search index. With vocabularies of size 256, the fine codes for each indexed vector will require M bytes to store in the index.

Background

- Product Quantization (PQ) is fast, but centroids don't have supported data
- Optimized Product Quantization (OPQ) allows for centroids to be rotated to better fit the data model
- LOCP extends upon PQ and OPQ by using their:
  - a *coarse quantizer*
  - a rotation matrix (from OPQ)
  - *product quantizers* but with **local optimizations**

- Quantization speeds up search by implementing lossy compression algorithms on data

Results

- LOPQ requires more space and time overhead in comparison to PQ, but is constant in data size
- Achieves state of the art performance on a variety of large datasets
- “Embarrassingly simple to apply”

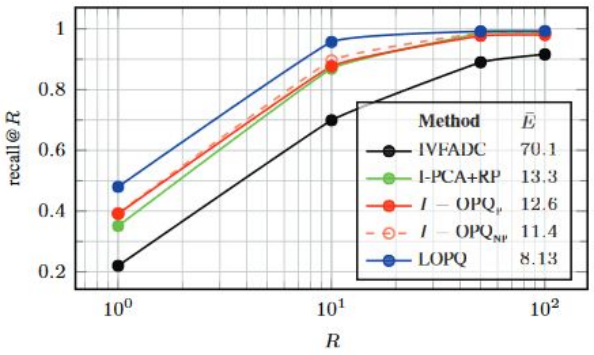


Figure 3. Recall@R on MNIST with $K = 64$, found to be optimal, and $w = 8$. $\bar{E} = E/n$: average distortion per point.

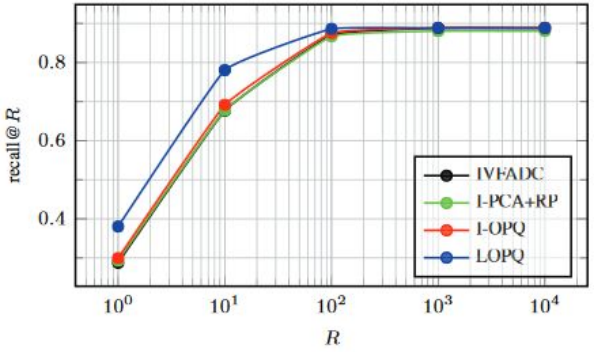


Figure 4. Recall@R on SIFT1M with $K = 1024$, $w = 8$.

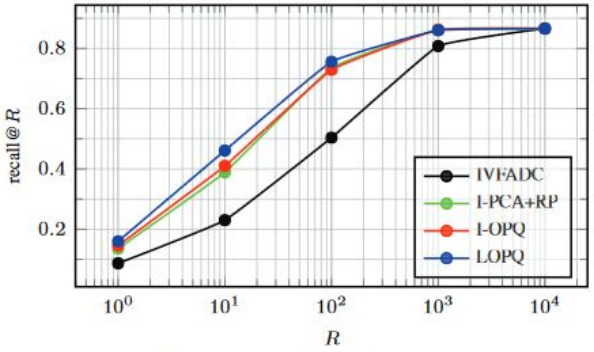


Figure 5. Recall@R on GIST1M with $K = 1024$, $w = 16$.

T	Method	$R = 1$	10	100
	Multi-I-Hashing [16]	–	–	0.420
	KLSH-ADC [17]	–	–	0.894
	Joint-ADC [19]	–	–	0.938
20K	IVFADC+R [13]	0.262	0.701	0.962
	LOPQ+R	**0.350**	**0.820**	**0.978**
10K	Multi-D-ADC [3]	0.304	0.665	0.740
	OMulti-D-OADC [8]	0.345	0.725	**0.794**
	Multi-LOPQ	**0.430**	**0.761**	0.782
30K	Multi-D-ADC [3]	0.328	0.757	0.885
	OMulti-D-OADC [8]	0.366	0.807	**0.913**
	Multi-LOPQ	**0.463**	**0.865**	0.905
100K	Multi-D-ADC [3]	0.334	0.793	0.959
	OMulti-D-OADC [8]	0.373	0.841	**0.973**
	Multi-LOPQ	**0.476**	**0.919**	**0.973**

Table 2. Recall@$\{1, 10, 100\}$ on SIFT1B with 128-bit codes and $K = 2^{13} = 8192$ (resp. $K = 2^{14}$) for single index (resp. multi-index). For IVFADC+R and LOPQ+R, $m' = 8$, $w = 64$. Results for Joint-ADC and KLSH-ADC are taken from [19]. Rows including citations reproduce authors' results.

Discussion Questions

- Why can't Product Quantization regenerate its original inputs?
  - Example: .zip files utilize lossy compression algorithms, why can't PQ?
  - Should they be concerned about this at all?

Efficient Large-scale Approximate Nearest Neighbor Search on the GPU

Overview + Results

- Creates a new data structure off of PQ to allow for GPU optimizations
- Performs well, but is not state of the art in recall
- Is significantly faster than CPU approximate nearest neighbor algorithms

method	ms	R@1	R@10	R@100	su
FLANN [16]	5.32	0.97	-	-	×9.6
LOPQ [11]	51.1	0.51	0.93	0.97	×1
IVFADC*	11.2	0.28	0.70	0.93	×4.5
PQT_1 (CPU)	4.89	0.45	0.86	0.98	×10.4
PQT_2 (CPU)	5.74	0.98	(exact re-ranking)		×8.9
PQT (GPU)	0.02	0.51	0.83	0.86	×2555
GPU brutef.	23.7	1	1	1	×2

Table 1: Performance on the SIFT1M dataset using different methods. Reported query times include query + re-ranking times. The GPU implementation uses the first 2^{12} vectors from the proposed bins and $(64 \cdot 8)^4$ bins. The reported CPU performance is base on $(8 \cdot 4)^2$ bins. Speedup (su) is reported relative to the slowest method. PQT_2 is PQT_1 but with additional exact re-ranking. (*) indicates that the timing was reported by the authors. R@n means, the correct vector is within the first n returned vectors from the algorithm.

Background

- Approximate nearest neighbor algorithms utilize CPUs and KD-Tree data structures
- Memory restrictions with GPUs make GPU optimizations difficult for this task
- Extends the work on Product Quantization by introducing a new data structure called a PQ Tree (PQT)
    - Built upon a combination of an inverted multi-index and hierarchical PQ

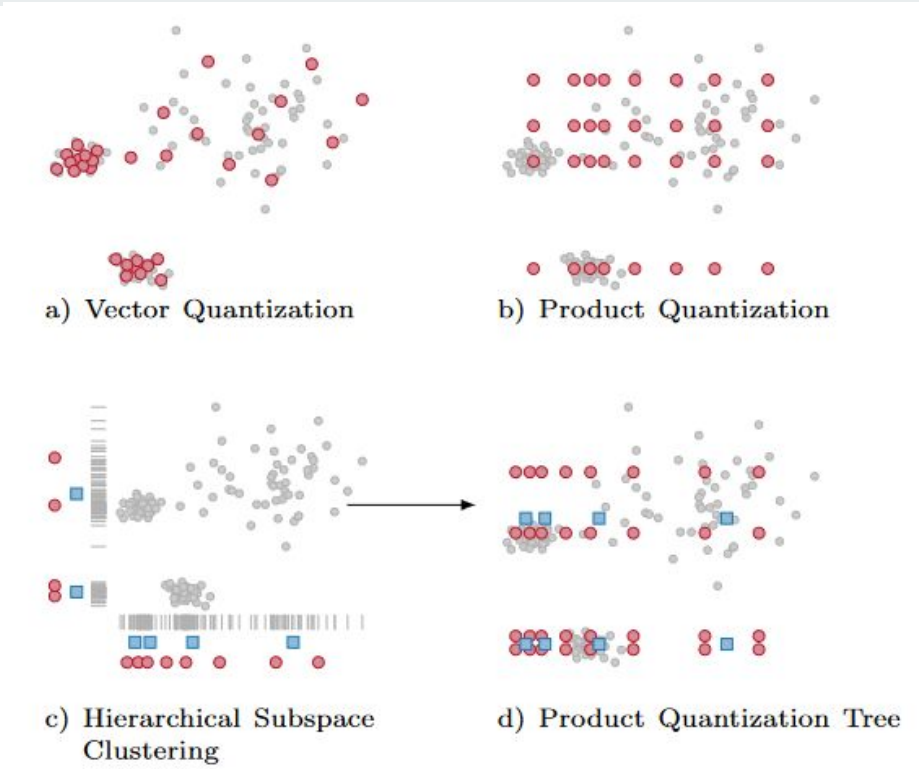


Figure 1: Three different quantization schemes with $k = 32$ clusters. Vector Quantization (a) represents vectors by their closest centroids. Product Quantization performs the clustering in subspaces (here axes) (b). A tree structure can be used to build a hierarchy of clusters on each axis (c). Our method use the hierarchy of two quantization levels, first using PQ with a low number of centroids, and then a second-layer of PQ within these bins (d). Points drawn as ■ are PQ centroids, and each corresponding cluster is split again into finer 4 clusters (2 on each axis) with centroids illustrated as ● .

Discussion Questions

- What did you think of the paper holistically?
- The authors created a new data structure to solve this problem using GPUs. Could this be done on CPUs for similar performance?
- Their results weren't state of the art for recall, thus raising the question that aside from speed, why bother utilizing this method for the purposes of Image Retrieval?

Fast Local Spatial Verification for Feature-Agnostic Large-Scale Image Retrieval

Overview

- Adapt image retrieval to composite images to determine what composited images make up a scene
- Proposed Objects in Scene to Objects in Scene (OS2OS) score
- A method of Content Based Image Recognition

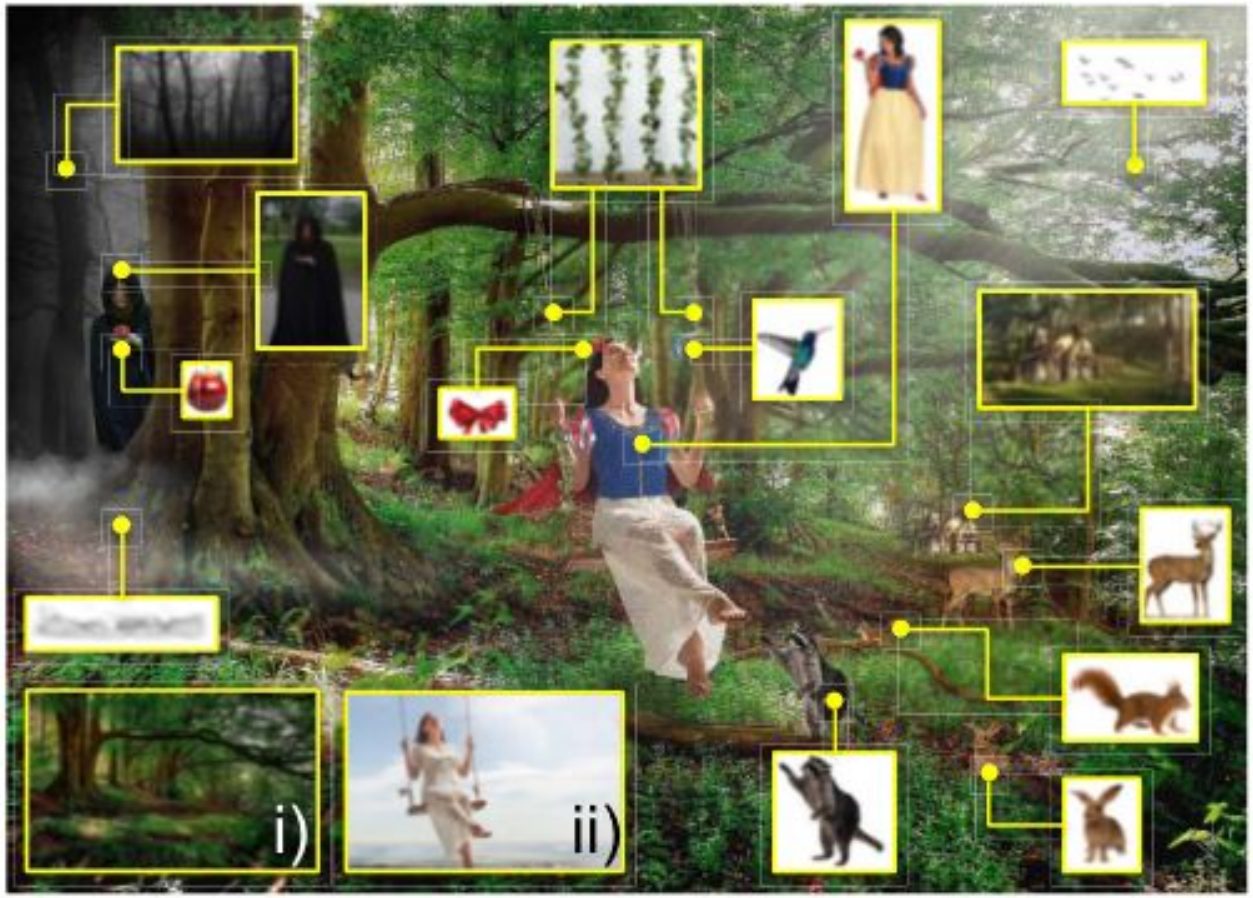


Fig. 1. An example of a meme-style image from the *r/photoshopbattles* subreddit. Internet memes are humorous messages that are spread on social media, often conforming to a set genre with a distinct style. In this paper, we provide spatial verification to find object-level correspondences between images, which assist in retrieving the donor images (highlighted in yellow) that contribute to composites like the one above.

Background

- Current approximate nearest neighbor algorithms utilize local and global features + quantization
  - SIFT, SURF, LIFT, and DELF
  - OPQ, Generalized Product Quantization + Inverted File Indices
- Region Based Image Retrieval
- Object Based Image Retrieval

- Implementation involves:
  - *Spatially Constrained Similarity Measure* to avoid querying areas of interest ahead of analysis.
  - A *coarse Pairwise Geometric Matching* to accumulate Hough votes in bins for analysis
  - A *two stage, O(n) solution* to **remove spurious solutions**
  - A *retrieval score (OS2OS)* to compare the returned image components

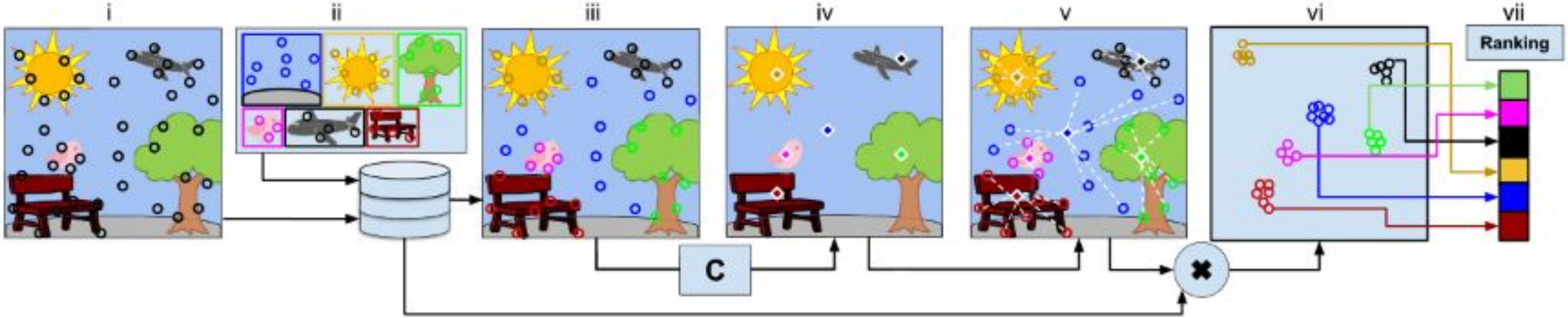


Fig. 2. Steps of the OS2OS method. (i) Local features with associated geometric data (*i.e.*, (x, y) coordinates, scale, and rotation) are extracted from the query. (ii) Local features, along with their associated geometric data, are collected in a database. In this hypothetical example, we associate features for each object with separate images; however, this is not a necessary stipulation for OS2OS. (iii) Query features are assigned to corresponding database matches (represented by feature colors). (iv) For each database image sharing matches with the query, a feature centroid is computed, considering only the matched features. (v) Keypoint geometric transformations are calculated relative to the estimated centroids. (vi) Geometric transformations are applied to the database features clustered in the query (x, y) space. (vii) As each cluster represents a potentially shared object, image ranking scores are calculated on an object-by-object basis.

Results

- Implementing OS2OS in existing CBIR algorithms increases the overall performance of the algorithms
  - OS2OS can be implemented either in CPU based or GPU based algorithms and still provide performance improvements

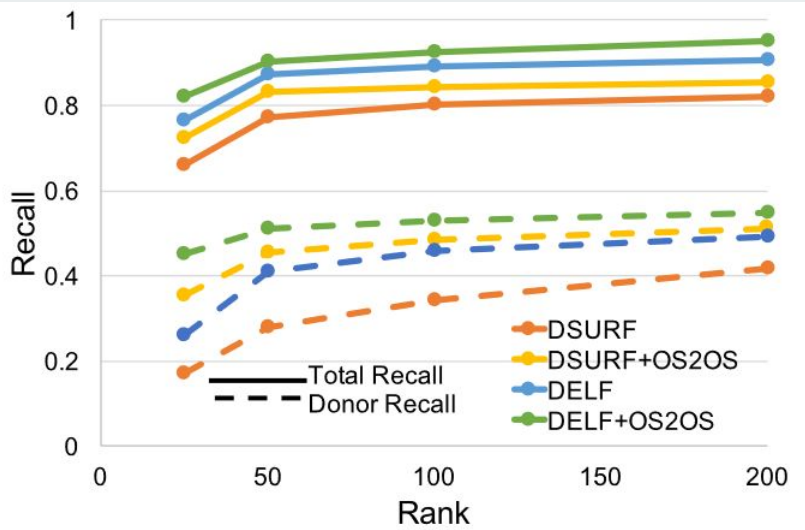


Fig. 6. Recall scores for the *NIST MFC2018* dataset for ranks of 25, 50, 100, and 200 images. Total recall is represented by solid lines, while small-donor-only recall is represented by dashed lines. OS2OS scoring improves retrieval in all scenarios.

TABLE III

RECALL SCORES FOR THE *Reddit* DATASET AT THE TOP-50, 100, AND 200 MOST RELATED RETRIEVED IMAGES. OS2OS SPATIAL VERIFICATION IMPROVES THE RESULTS IN ALL SCENARIOS. THE BOTTOM TWO ROWS DENOTE RESULTS USING OUR OS2OS APPROACH

Method	R@50	R@100	R@200
DSURF	0.317	0.432	0.478
DELF	0.402	0.516	0.551
DSURF + HPM	0.351	0.358	0.437
DSURF + PGM	0.327	0.398	0.442
DSURF + VaV	0.310	0.423	0.479
DSURF + OS2OS	0.424	0.509	0.546
DELF + OS2OS	**0.479**	**0.548**	**0.593**

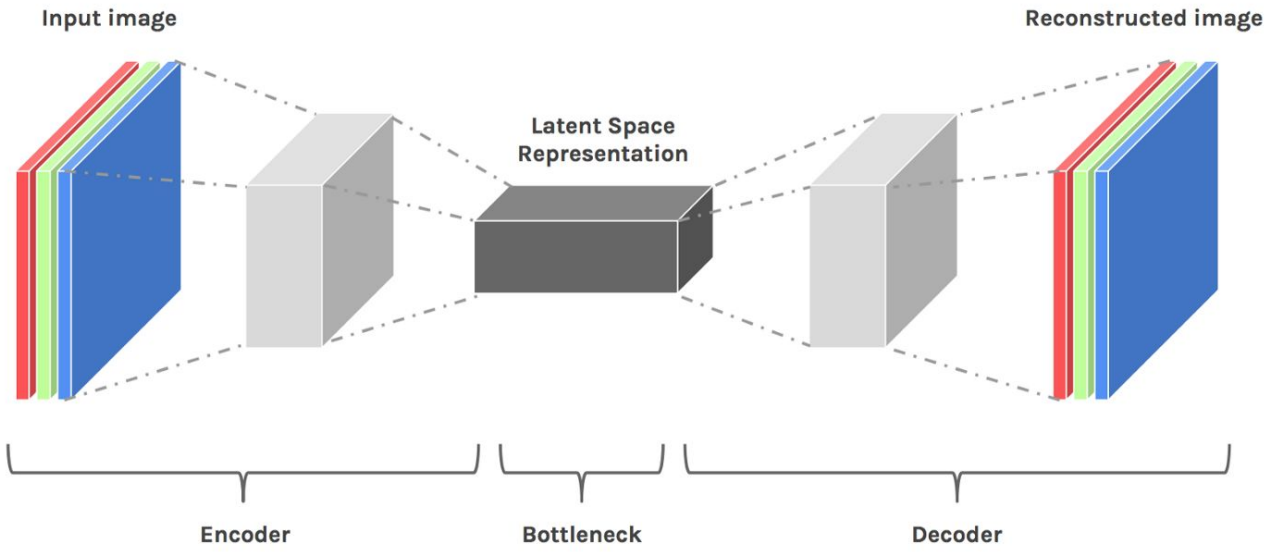
Discussion Questions

- Would OS2OS work well with composite images of composite images?
  - Does this solution fail to provide performance benefits with respect to recursion?
- Is this work limited by the amount of data that was able to be searched through?
  - In other words, would more data = better performance?

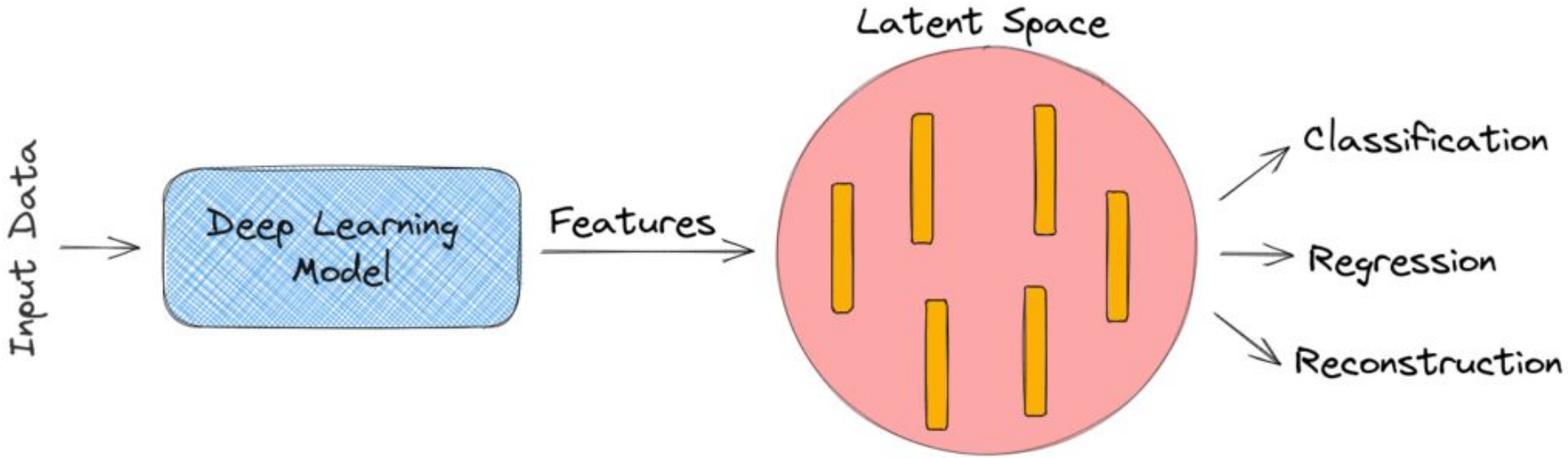
Latent Variables in Computer Vision

Understanding latent space

- What is latent space?
  - If you must describe latent space in one sentence, it simply means a representation of compressed data.
  - The concept of “latent space” is *important* because its utility is at the core of ‘deep learning’ — *learning the features of data and simplifying data representations for the purpose of finding patterns.*



- A latent space or vector is basically a distribution of the latent variables above. It is also commonly referred to as a feature representation.
- Think of a latent vector as a collection of an image's 'features', I.e. variables that describe what is going on in an image, such as the setting (medieval or modern), the time of day, and so on. This is not exactly how it works - it is just an intuition. The idea is that the latent variables represent high level attributes, rather than raw pixels with little meaning.
- Latent variables can be used when connecting computer vision models to models from other domains that do not deal with image data. For example, a common task in natural language processing is image captioning. To generate a caption for an image, an NLP model would require the image's latent variables. That is where they get the understanding necessary to describe what is going on in the image.

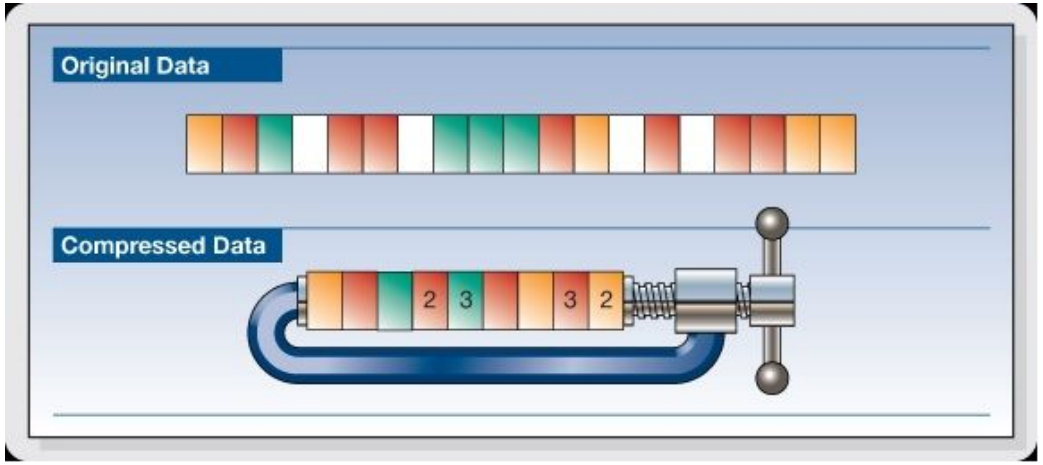


How is data simplified?

- Why do we compress data in ML?
- Data is compressed in machine learning to learn important information about data points
- Say we would like to train a model to classify an image using a fully convolutional neural network (FCN). (i.e., output digit number given image of digit). As the model ‘learns’, it is simply learning features at each layer (edges, angles, etc.) and attributing a combination of features to a specific output.

Data compression

- But each time the model learns through a data point, the *dimensionality* of the image is first *reduced* before it is ultimately increased. When the dimensionality is reduced, we consider this a form of lossy compression.
- Because the model is required to then *reconstruct* the compressed data, it must learn to store *all relevant information* and disregard the noise. This is the value of compression- it allows us to get rid of any extraneous information, and only focus on the most important features.
- This 'compressed state' is the Latent Space Representation of our data.



Compressed Data = Space?

- In this rather simplistic example, let's say our original dataset are images with dimensions 5 x 5 x 1. We will set our latent space dimensions to be 3 x 1, meaning our compressed data point is a vector with 3-dimensions.
- Now, each compressed data point is uniquely defined by only 3 numbers. That means we can graph this data on a 3D Plane (One number is x, the other y, the other z).

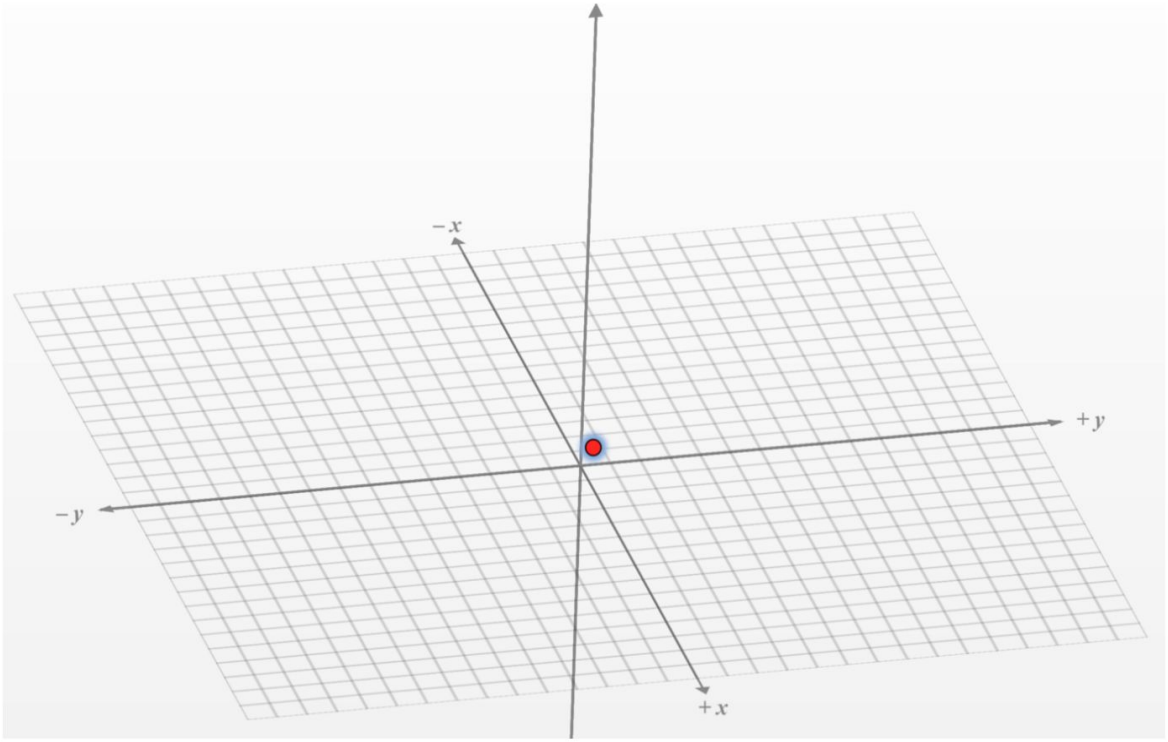
$$\begin{bmatrix} 1 & 0.5 & 0.8 & 0.2 & 0.7 \\ 0.2 & 0.11 & 0.78 & 0.3 & 0.2 \\ 1 & 0.01 & 0 & 0.9 & 0.56 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0.4 & 0.2 & 0.1 & 0.01 \end{bmatrix}$$

Example 5×5×1 data

$$\begin{bmatrix} 0.4 \\ 0.3 \\ 0.8 \end{bmatrix}$$

Example compressed 3×1 data in 'latent space'

- This is the “space” that we are referring to.
- *Whenever we graph points or think of points in latent space, we can imagine them as coordinates in space in which points that are* “similar” *are closer together on the graph.*



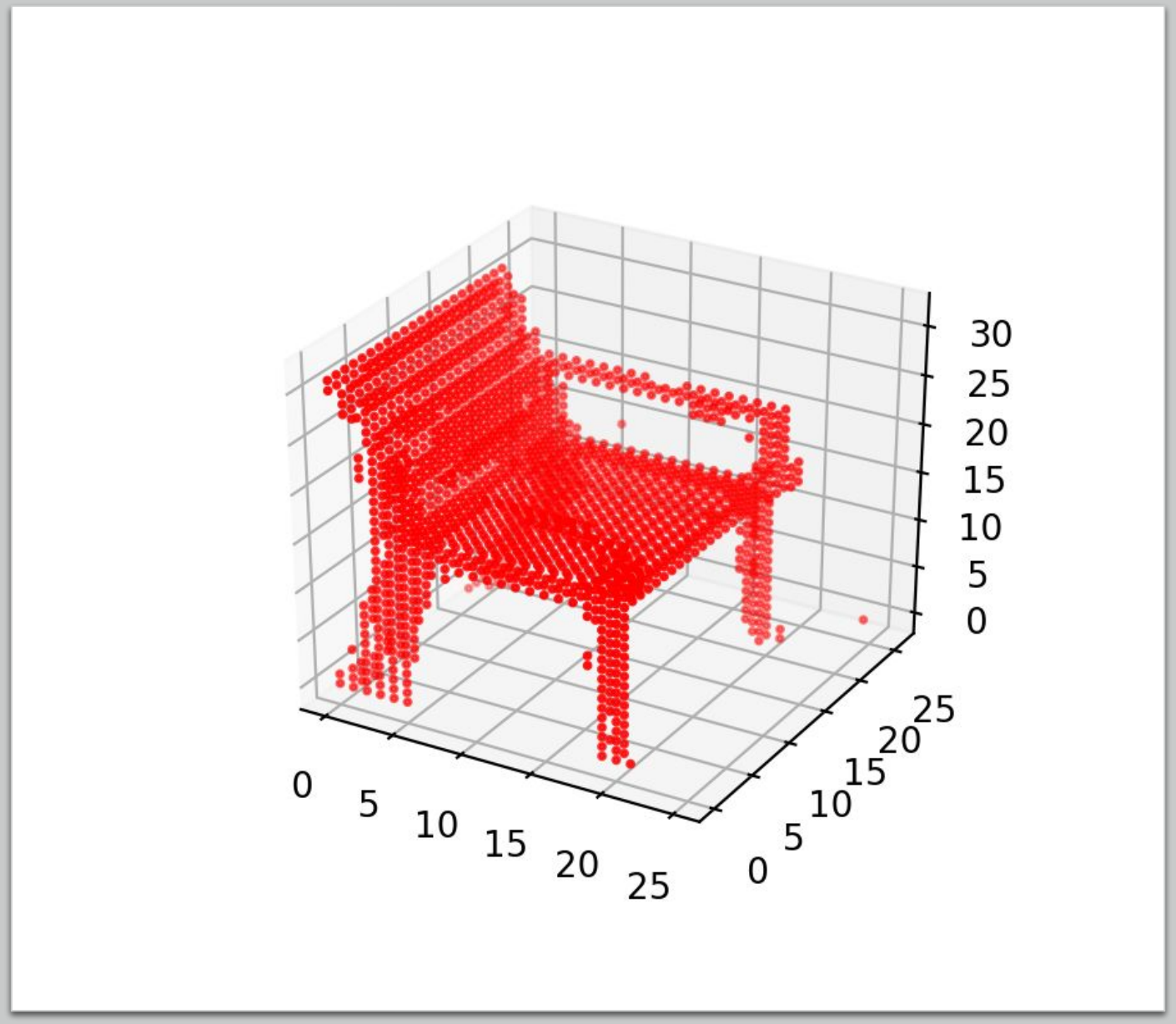
Point (0.4, 0.3, 0.8) graphed in 3D space

What defines ‘similarity’ between points?

Two chairs and a desk.

- If we look at three images, two of a chair and one of a desk, we will easily say that the two chair images are the most similar whereas the desk is the most different from either of the chair images.
- But what makes these two chair images “more similar?” A chair has distinguishable features (i.e., back-rest, no drawer, connections between legs). These can all be ‘understood’ by our models by learning patterns in edges, angles, etc.
- As explained, such features are packaged in the latent space representation of data.

- Thus, as dimensionality is reduced, the ‘extraneous’ information which is distinct to each image (i.e.,. chair color) is ‘removed’ from our latent space representation, since only the most important features of each image are stored in the latent space representations.
- As a result, as we reduce dimensionality, the representations of both chairs become less distinct and more similar. If we were to imagine them in space, they would be ‘closer’ together.



- Generally, any two similar images will lie closer to each other in the latent space whereas dissimilar images will lie far away. This is the basic governing rule with which we will train our model.
- Once we do this, the retrieval part simply scours the latent space to pick up the closest image in the latent space given the representation of the query image. Most of the time, it is done with the help of nearest neighbor search.

References:

- **For Image Retrieval:**
- https://towardsdatascience.com/a-hands-on-introduction-to-image-retrieval-in-deep-learning-with-pytorch-651cd6dba61e
- **For Latent Space:**
- https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d
- https://dev.to/badasstechie/latent-variables-in-computer-vision-14fa
- **For Nearest Neighbor Classification:**
- https://athitsos.utasites.cloud/projects/nearest_neighbors/
- **For LOPQ:**
- https://github.com/yahoo/lopq/blob/master/README.md
- https://lou.dev/talks/2018-08-09-lopq#:~:text=Locally%20Optimized%20Product%20Quantization%20is,and%20high%20dimensionality%20data%20sets.
- **For OPQ:**
- http://kaiminghe.com/cvpr13/index.html