Image Search

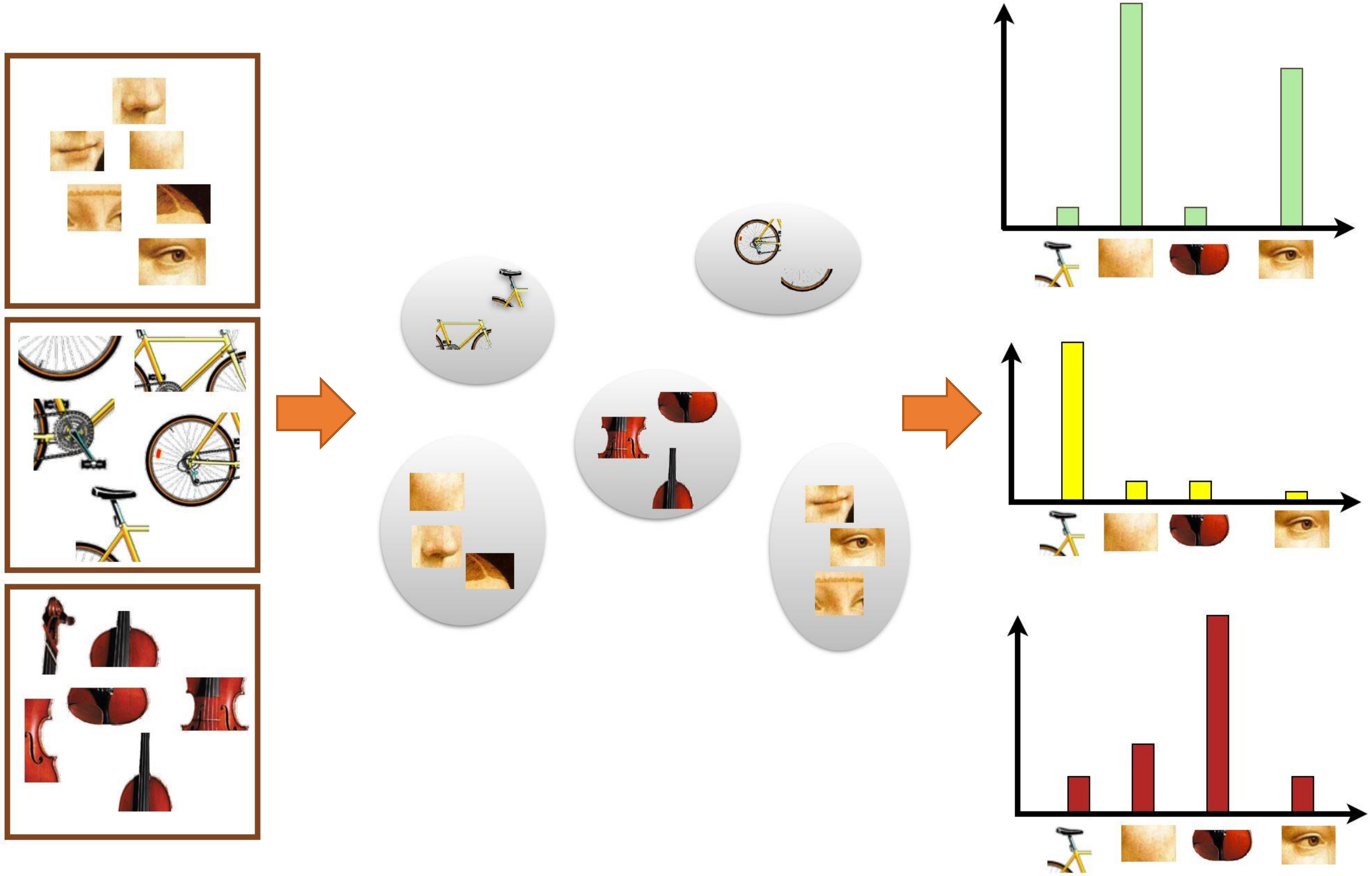
Peiyi Zheng

Goal:

- **Accuracy**
- **Time Complexity**
- **Space Complexity**

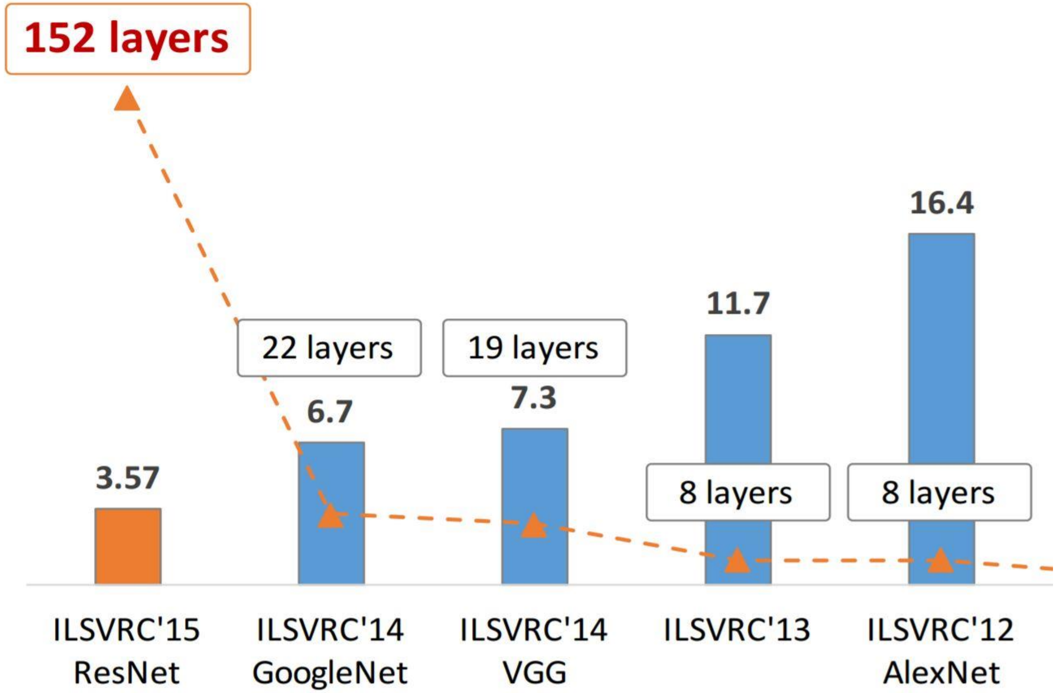
Previous Approach: Not Good Enough

- SIFT
- Bag of Words
- TF-IDF



Deep Learning: Extract Better Features

- **Use ResNet for image features extraction**



ResNet, 152 layers
(ILSVRC 2015)

Deep Learning: Problem

- Features have too many dimensions
- 3.6 GB for 150,000 images → bad space complexity
- Solution: find a better representation

Similarity Preserving Binary Codes

- **Idea from *Iterative Quantization: A Procrustean Approach to Learning Binary Codes***
- ***"We propose a simple and efficient alternating minimization scheme for finding a rotation of zero centered data so as to minimize the quantization error"***

Iterative Quantization

- **Preprocessing: dimensionality reduction via PCA** $\Rightarrow$ **new matrix** $\boldsymbol{V}$

- **Goal: minimize the quantization error** $\boldsymbol{Q(B,R) = \| B - \tilde{V} \|_F^2}$

$$\boldsymbol{\tilde{V} = VR, B = sgn(\tilde{V})} \Rightarrow \boldsymbol{B_{ij} \in \{-1, 1\}}$$

- **When** $\boldsymbol{Q(B,R)}$ **is smaller,** $\boldsymbol{B}$ **preserves more information in** $\boldsymbol{\tilde{V}}$

Iterative Quantization

- $\boldsymbol{Q(B,R) = \| B - \tilde{V} \|_F^2 = nc + \| V \|_F^2 - 2tr(BR^TV^T)}$

- **Minimize $\boldsymbol{Q(B,R)}$ is equivalent to maximize:**

$$\boldsymbol{tr\left(BR^TV^T\right) = \sum_{i=1}^{n} \sum_{j=1}^{c} B_{ij}\tilde{V}_{ij}}$$

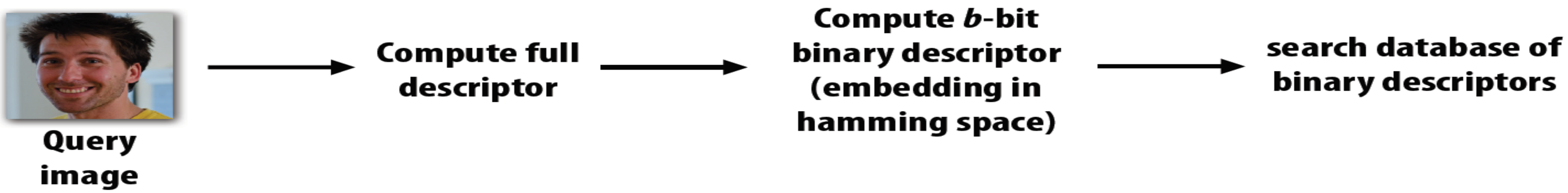
How to maximize: $tr\left(BR^TV^T\right) = \sum_{i=1}^{n}\sum_{j=1}^{c} B_{ij}\widetilde{V}_{ij}$

- **Initializing rotation matrix R to a random orthogonal matrix**
- **Fix R and update B : set $B_{ij} = 1$ when ever $\widetilde{V}_{ij} \geq 0$ and otherwise set $B_{ij} = -1$**
- **Fix B and update R : compute the SVD of B^TV as $S\Omega\widehat{S}^T$ and update R to $\widehat{S}S^T$**
- **Iterate the procedures above for 50 times**

Improvement

- **The size of features database decreases to 36 MB (256 bits binary code)**

- **Problem: it takes about 10 seconds to perform Top-K similarity search(150,000 images) bad time complexity**



Multi-Index Hashing

- **Idea from *Fast Search in Hamming Space with Multi-Index Hashing***
- **Basic intuition:**
  - **Divide query b bits string into m disjoint $\frac{b}{m}$ bits substrings**
  - **Bit strings that are close in one of the substrings might be close overall**

Multi-Index Hashing

- **Goal: find all images within hamming distance $\boldsymbol{r}$ from query**

- **Number of candidates:** $\boldsymbol{N(b,r) = \sum_{k=0}^{r} \binom{b}{k}}$

- $\boldsymbol{b}$ **is the length of binary code**

- $\boldsymbol{N(64,7) \approx}$ **1 billion** ⇨ **too many possibilities**

Multi-Index Hashing: Reduce $\boldsymbol{N(b, r)}$

- **When two binary codes $\boldsymbol{h}$ and $\boldsymbol{g}$ differ by $\boldsymbol{r}$ bits or less(hamming distance), then at least one of their $\boldsymbol{m}$ substrings satisfy:**

$$\| \boldsymbol{h^k - g^k} \| \le \frac{\boldsymbol{r}}{\boldsymbol{m}}$$

- **Proof: if they differed by more than $\boldsymbol{\frac{r}{m}}$ bits in each substring, then overall $\boldsymbol{h}$ and $\boldsymbol{g}$ must differ by more than $\boldsymbol{r}$ bits**

Multi-Index Hashing: Reduce $N(b, r)$

- For each set of length $\frac{b}{m}$ substrings, find substrings of within Hamming radius of $\frac{r}{m}$
- Previously: search needed to examine $N(b, r)$ situations
- Now need to examine only $N\left(\frac{b}{m}, \frac{r}{m}\right)$ situations in m buckets

Multi-Index Hashing: KNN Search

- Given $\boldsymbol{b}$ bits query and initialize $\boldsymbol{r = m}$:

- For each of the $\boldsymbol{m}$ substrings of the query:

  - Find radius $\frac{\boldsymbol{r}}{\boldsymbol{m}}$ neighbors and add them to candidate set

- The candidate set is a superset of the true set of elements within hamming distance $\boldsymbol{r}$, so compute actual set by executing full Hamming distance computation for all elements in candidate set

- Increase $\boldsymbol{r}$ in each iteration until we find more than K candidates and then return K nearest neighbors in them

Improvement

- **In my implementation, b is 256 and m is 16**
- **We can compute the possible hamming distances between each substring offline**
- **Now we only need 1.5 – 2.0 seconds to perform a KNN search on the images database**
- **Further work: We can use m threads to accelerate the search**

Result

Result