Mining Multiple Queries for Image Retrieval: On-the-fly learning of an Object-specific Mid-level Representation - Supplementary material

1. Supplementary material

In supplementary material section we provide some additional information about proposed method. In section 1.1 we give a simple data mining example to understand terms such as patterns, transactions and bag-of-patterns. In section 1.2, we give details of the offline transaction indexing algorithm. In section 1.3 we give detailed algorithm for online image retrieval using pattern matching. In section 1.4, we show some visual patterns that we detected. In section 1.5, we perform an analyzis of execution time and memory requirement of our retrieval algorithm.

1.1. Data mining example

In this section we provide a simple data mining example to understand terms such as transaction, pattern, model and pattern matching. Consider we are given two images I_1 consist of three transactions $I_1 = \{t_1, t_2, t_3\}$ and the second image consist of three transactions $I_2 = \{t_4, t_5, t_6\}$ as shown in Table 1. Lets say that we are given a query image consist of four transactions as given in Table 2. By pattern mining we discover two patterns x_1, x_2; where $x_1 = \{1, 2, 3\}$ and $x_2 = \{3, 5, 6\}$. Now we can perform pattern matching to build bag-of-patterns for image I_1 and I_2. See Table 3. Note that because pattern $x_1 \subset t_1$; we say pattern x_1 is matched/mapped to transaction t_1. This way we count how many times each pattern occured in database images and build bag of patterns. To do this efficiently we use inverted file systems.

Transaction	Visual words			
t_1	1	2	3	7
t_2	3	5	6	8
t_3	1	2	3	8
t_4	3	5	6	7
t_5	1	2	3	4
t_6	3	5	6	

Table 1. Transactions from database images.

Transaction	Visual words			
tq_1	1	2	3	
tq_2	3	5	6	1
tq_3	1	2	3	5
tq_4	3	5	6	8

Table 2. Transactions from query images.

$Pattern$	Matched transactions		
$x_1 = \{1, 2, 3\}$	t_1	t_3	t_5
$x_2 = \{3, 5, 6\}$	t_2	t_4	t_6

Table 3. Transactions from query images.

1.2. Indexing algorithm

Here we give implementation details of transaction indexing. By indexing transactions from database images we construct two inverted file systems called IFS_1 and IFS_2. The key of the IFS_1 is a visual word and the corresponsing entries are the transaction IDs (TIDs) that contains the visual word. We need to initialize three things, 1. IFS_1 which has D number of keys (D is the dictionary size) 2. IFS_2 is initialized to an empty inverted file system at the begining and 3. the list of unique transactions ($TRANS$) which contains unique transactions. We do not need $TRANS$ variable at the end of the algorithm. At the end of the algorithm we return only IFS_1 and IFS_2. The highlevel transaction indexing algorithm is given in Algorithm 1 and the detailed indexing algorithm in Algorithm 2.

1.3. Retrieving list of images that contains a pattern

In this section we give implementation details of the retrieval algorithm. We show how to obtain a list of images that contains a given pattern. We use IFS_1 and IFS_2 to retrive images containing pattern $x = \{w_1 \dots w_p\}$. Note that a pattern is a set of visual visual words that occured in a specific feature configuration. The retrival algorithm is given in Algorithm 3.

1.4. Detected patterns

The local patterns are considerably large (compared to individual key points) and possess some local spatial and

Data: Images and Transactions from images
Result: IFS_1 and IFS_2
Initialization;
$IFS_1 \leftarrow IFS(1 \dots D)(\phi)$;
$IFS_2 \leftarrow IFS(\phi)(\phi)$;
$TRANS \leftarrow \phi$;
foreach *Image* I **do**
  foreach *Transaction* $T \in I$ **do**
    if $T \in IFS_1$ **then**
      $TID \leftarrow Find(IFS_1, T)$;
    else
      $TID \leftarrow Insert(IFS_1, T)$;
    end
  end
  $IFS_2(TID) \leftarrow IFS_2(TID) \cup I$;
end

Algorithm 1: Highlevel explanation of the indexing algorithm.

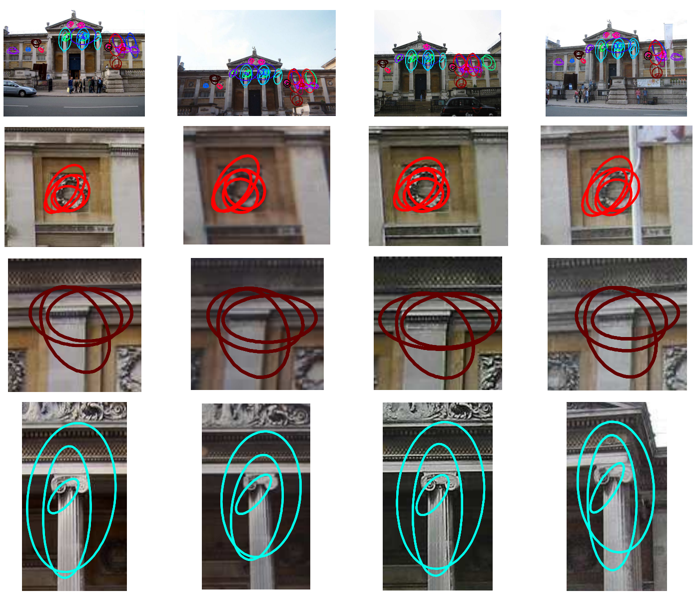

Figure 1. First row: original image with some of the detected patterns in different colors, next closed up view of some of the patterns. (see supplementary material for more examples)

structural information. Figure 8 shows some example patterns. Note how each pattern describes part of the object of interest.

1.5. Execution times and memory analyzis

In this section we provide an analyzis on execution time and memory requirements of our transaction indexing and image retrival algorithms. We perform all experiments using a single CPU (2.6GHz). We retrive images using Algorithm 3 in 0.3 ± 0.01 miliseconds for a dataset of 105K

Data: Images and Transactions from images
Result: IFS_1 and IFS_2
Initialization;
$IFS_1 \leftarrow IFS(1 \dots D)(\phi)$;
$IFS_2 \leftarrow IFS(\phi)(\phi)$;
$TRANS \leftarrow \phi$;
foreach *Image* I **do**
  foreach *Transaction* $T \in I$ **do**
    $T \leftarrow sort(T)$;
    $isnew \leftarrow false$;
    $list \leftarrow \phi$;
    foreach $word \in T$ **do**
      if *empty*$(IFS_1(word))$ **then**
        $isnew \leftarrow true$;
        break;
      else
        if *word is the first word in* T **then**
          $list \leftarrow IFS_1(word)$;
        else
          $list \leftarrow$
          $intersect(list, IFS_1(word))$;
          if *empty(list)* **then**
            break ;
          end
        end
      end
    end
    $foundit \leftarrow false$;
    $temptr \leftarrow \phi$;
    if $!empty(list)$ **then**
      foreach *tid* $\in$ *list* **do**
        if $|T| == |tid|$ **then**
          $foundit \leftarrow true$;
          $temptr \leftarrow tid$;
        end
      end
    end
    if $isnew$ *or* $empty(list)$ *or* $!foundit$ **then**
      foreach $word \in T$ **do**
        $IFS_1(word) = IFS_1(word) \cup TID$
      end
      $IFS_2(TID) = IFS_2(TID) \cup I$;
      $TRANS(TID) = T$;
      $TID = TID + 1$;
    else
      $IFS_2(temptr) = IFS_2(temptr) \cup I$;
    end
  end
end

Algorithm 2: Indexing algorithm

Data: Inverted file systems IFS_1, IFS_2 and the pattern x
Result: List of images $imlist$ containing pattern x
Initialization;
$imlist \leftarrow \phi$;
$x \leftarrow sort(x)$;
foreach *word* $\in$ *x* **do**
  if *word is the first element in x* **then**
    $list \leftarrow IFS_1(word)$;
  else
    $list \leftarrow intersect(list, IFS_1(word))$;
  end
  if *empty(list)* **then**
    break ;
  end
end
foreach *TID* $\in$ *list* **do**
  $imlist \leftarrow imlist \cup IFS_2(TID)$
end

Algorithm 3: Retrieving a list of images containing a pattern x

images for a single pattern. The retrival time for 300 patterns using a single CPU is 0.085 seconds. We also indexed data using a dataset of 250K images. In this case we can retrive images using 300 patterns in roughly the same time. In Figure 2, we show how retrival time varies with number of images. IFS_1 and IFS_2 are kept in memory for efficiency. 105K images are indexed using 1500Mb of ram. 250K images are indexed using roughly 3000Mb of ram. Note that memory requirement of our approach is sublinear in number of images in the dataset as shown in Figure 3.

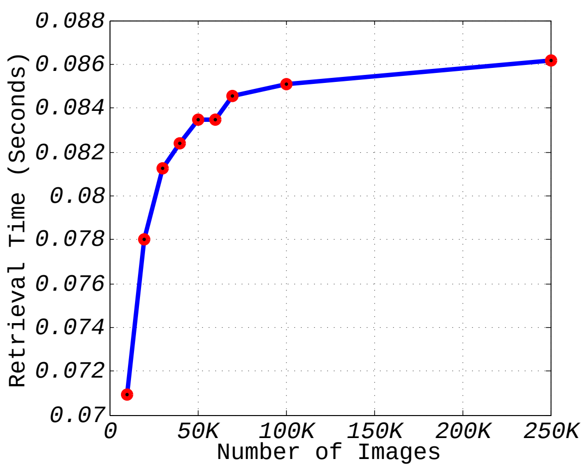

Figure 2. Execution time for 300 patterns by varying the number of images.

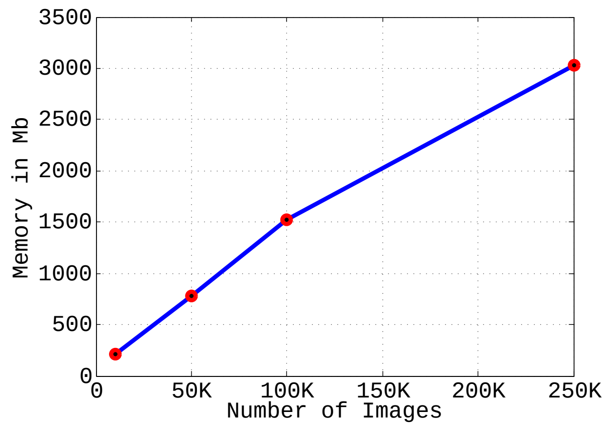

Figure 3. Memory requirement of inverted file systems (IFS_1 and IFS_2) by varying the number of images in the dataset.

Figure 4. Sample query from the new dataset and obtained results using our method.

Figure 5. Sample query from the new dataset and obtained results using our method.

Figure 6. Sample query from the new dataset and obtained results using our method.

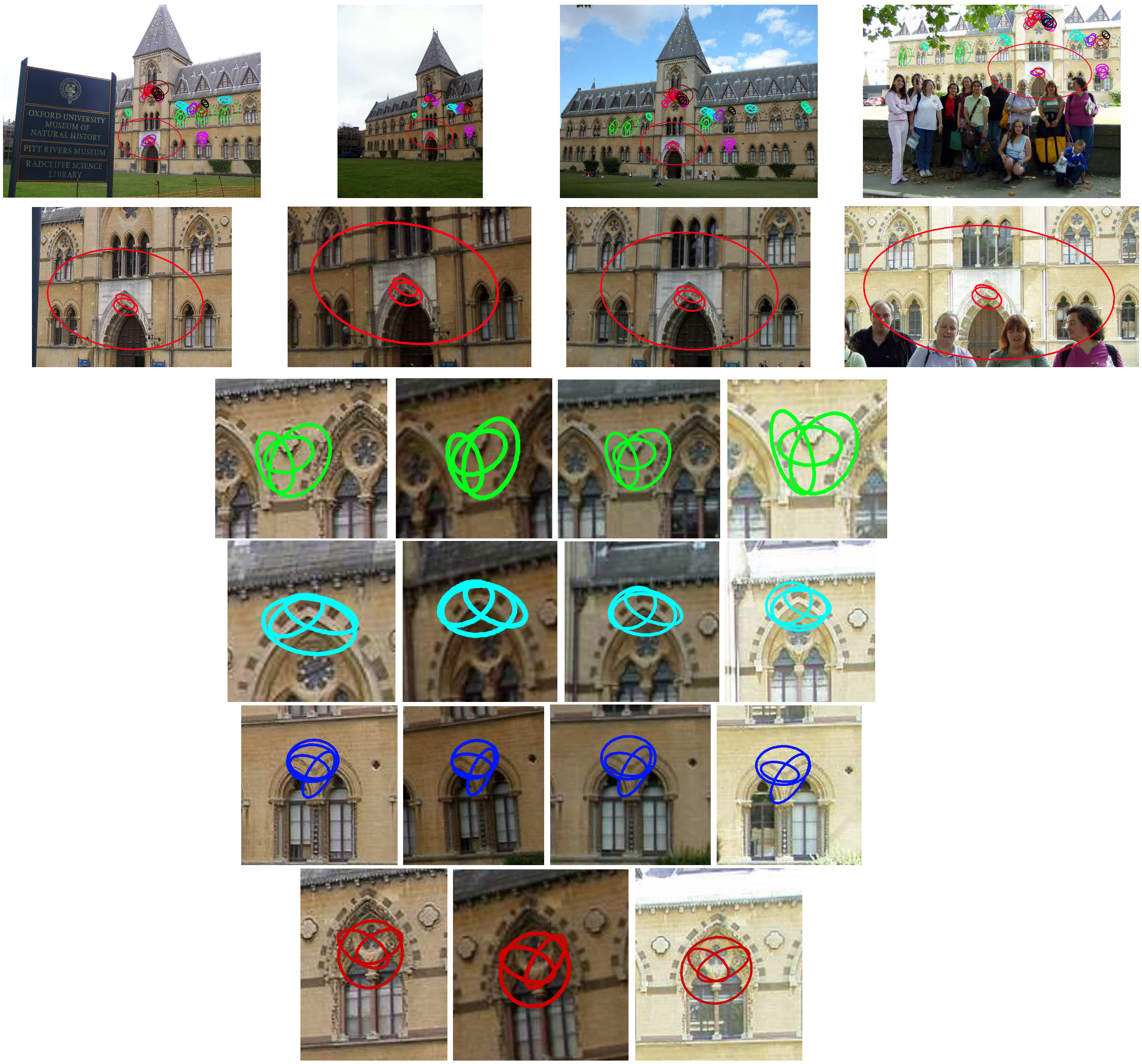

Figure 7. Example patterns detected from Pitt Rivers Oxford.

Figure 8. Some of the patterns we detected using our approach in Oxford buildings dataset.

Figure 9. Some of the patterns we detected using our approach in Oxford buildings dataset.