# CS419 - PROJECT REPORT
# Project 2: Image Retrieval


18125125 Pham Bang Dang
18125014 Le Hoang Nhan
18125040 Nguyen Le Minh
18125063 Bui Xuan Bao


## 1 Introduction

Image Retrieval is a problem in Information Retrieval that involves domain knowledge in computer vision and machine learning. A user of an image retrieval system specifies an image or a part of an image as the input. The system is responsible for looking up its database and returning those images that are most relevant to the inputted one, based on the content of the image. The problem is called Content-based Image Retrieval. There are existing image retrieval system: Google Image Search [4], Bing Image Search [1], SauceNAO [7], etc.

In this report, we report about our image retrieval system and evaluation of our methods on the Oxford5k dataset [2]. We propose and implement many retrieval models, evaluate them and compare their results for academic purposes.

The structure of this report is as follows. Section 2 describes our choice of technology in our system. We overview our system in section 3, discuss about our retrieval methods in section 4, about used similarity measurements in section 5 and about our experiment as well as the results in section 6.

## 2 Technologies used

We think that deploying an image retrieval system on the web is most appropriate, therefore the user interface is implemented by a web front-end, and the query is handled in the back-end server. The images will be stored in the google drive that is shared public to get the urls for displaying on the web front-end.

Furthermore, we emphasize on the ability to quickly implement the pipeline. Therefore, we choose the programming language with their popular frameworks, libraries that we are able to learn them fast (in a week or so):

- React JS framework [6] for web front-end: there are many libraries in ReactJS.
- FastAPI framework [3] for web back-end using Python: support web GUI to test the API.
- Cropper.js, the library for crop function in user interface.
- Netlify to deploy front-end website.
- Google Cloud SDK to retrieve image urls from image dataset stored in the google drive.
- Torch module to create the Deep Learning model.

## 3 The retrieval system

Our retrieval system consists of a front-end user interface (web application) that allows users to choose an image by either browsing or drag-and-dropping the files then crop it arbitrary number of times, and the back-end server that is responsible for initializing the retrieval model, handling requests from front-end. When the user clicks the Search button in the graphical user interface (front-end), the cropped image is sent to the back-end server (which could be locally hosted). The server then calls appropriate functions

of the retrieval model and returns the result list of images' names to the front-end. The front-end then uses the image name to display them.

The web (front-end) is available at https://project-cs419-feir.netlify.app/. Unfortunately, the back-end server uses too many python modules that there is no free cloud server can deploy. Please run the back-end server locally to test the system. The instruction of hosting the local server would be found in the attached README.md file.

Our implementation of the retrieval system, together with execution instruction can be found at https://github.com/zero1778/CS419-Project. If there is any problem with the local submitted file, please go to our github page for latest version.

# 4 Methods

## 4.1 Image Retrieval with 1000-feat on full dataset

For each image in the dataset, we feed it into a pre-trained ResNet50 model [10] to extract the feature vector and save the feature vectors to .pickle file for faster loading next time run. Because there are more than 5 thousand images, our limited memory cannot do the extraction and save at once. We split the dataset into 6 parts that our memory can afford without crashing.

The input query will be also fed into the same pretrained ResNet50 model [10] to extract the feature vector then we calculate the similarity between it with our saved vectors with cosine distance, Euclidean distance and Complement of Normalized Dot Product (CoNDP). We sort the value and then return top K results.

ResNet50 [10] is a deep learning model, pre-trained on ImageNet dataset with 1000 outputs. Since the model is capable of extracting image generic features into a vector, we can make this capability a starting point of our retrieval task.

## 4.2 Image Retrieval with 2048-feat on 55 queries

In this second method, instead of using the last softmax layer to get a 1000-feature vector, we get the second last layer to get its 2048-feature vector. Besides, we did not store 5k feature vectors, we extracted and saved the features of 55 images that had been already labeled. Using labeled 55 images in evaluation seems to be bias when doing the evaluation, however, this could make the retrieval system runs well on other queries due to its shared similarities to the human-expert-labeled ones. In retrieval problem, we cannot know exactly what is the most optimal solution, the right answers. Human annotation helps to minimize the answers, if the image query contains some similarities to a class, some of that class's labeled images should meet the information need of that query.

The input query will be conducted the same as method 4.1. However, when we get the first top result, we will use its labeled "Good" and "OK" image sets that we already held to return.

## 4.3 Image Retrieval with Classified Feature

There are 17 classes total in the dataset. We trained the data with EfficientNet-B0 [14], ResNet18 [10], ResNet50 [10] to classify the images into 17 classes in hope that if a cropped image that shares similarity to a class, we can return all its class's images. Why we navigate the information retrieval to a classification problem? Because we see that, the feature vector could have a good representation for each image is the feature that could show the class which query image belongs to. That is why we would choose the logit feature vector of the classification problem as a representing feature vector for each image.

The input query will be classified into a category. The category name is used as the file name of labeled query files that we will use their "Good" and "OK" image sets to return the results.

## 4.4 SIFT descriptor with kNN matching

Scale Invariant Feature Transform (SIFT) is a famous algorithm in computer vision to detect the local features in an image. The detected features are invariant to translation, rotation, and scaling, which are robust to be image keypoints. Aside from converting the input image to grayscale as preprocessing, the algorithm consists of 4 main steps: scalespace extrema detection, keypoint localization, orientation

assignment, keypoint descriptor forming, we refer to the paper [11] for the detail. The output of the algorithm for one input image is a matrix $A \in \mathbb{R}^{n \times 128}$, where $n$ is the number of keypoints (variable from image to image) detected, each keypoint is represented by 128 float numbers. The matrix $A$ is also called the SIFT descriptor of the input image.

We use the SIFT descriptor as the concept vector in our vector space model, where we define that two images are similar implies that they have some common features (e.g. they look similar). Hence, we can measure the distance between two images by first compute their SIFT descriptors $A \in \mathbb{R}^{n \times 128}$ and $B \in \mathbb{R}^{m \times 128}$. Subsequently, we perform nearest neighbor matching between $n$ vectors in $A$ and $m$ vectors in $B$. Here the L2 distance is used to calculate distance between two vectors. To reduce noise, the original author of [11] proposes the ratio test, which finds both the nearest neighbor and the second nearest neighbors, if the ratio between them is less than a threshold value (often 0.75 or 0.8 is chosen), then the match is accepted, otherwise rejected. In our method, we use the `crossCheck` option as an alternative to the ratio test provided by OpenCV library [9]. With the `crossCheck` option, only those matches $(i, j)$ such that $i$th vector in $A$ has the $j$th vector in $B$ as the best match and vice versa. After matching, we have $s$ matches which are $s$ pairs of points, the pair $s_k$ consists of two indices $i_k$ from 0 to $n-1$ and $j_k$ from 0 to $m-1$, respectively, together with their distance. Subsequently, We take the average distance of the top 20 nearest pairs as the distance between two images.

In the preprocessing step, we calculate SIFT descriptors of all images in the database and store them into a binary file to be ready to use. However, each original images have about 5000 keypoints in average, each image is averagely about 5000 keypoints $\times$ 128 float per keypoints $\times$ 4 bytes per float $\approx$ 2.56MB in size. Since our dataset (see section 6.1) consists of more than 5000 images, storing all descriptors in the memory is infeasible. Our approach to this problem is that we downscale the image by a factor of 4 before calculating, since the local features of SIFT are scale-invariant, we expect the important keypoints are preserved. Empirically, the size decreases to about 970 MB, which can be stored whole in memory.

In the processing step, for a query image, we convert it into grayscale then calculate its SIFT descriptor. Next, we use a k-Nearest Neighbor (kNN) matcher to iterate each stored descriptor of all images and find the distance between it and the descriptor of the query image, using the method mentioned above. Finally, top $k$ shortest images, where $k$ is a model parameter, are returned as the ranked list. The pipeline of the method is illustrated in figure 1.

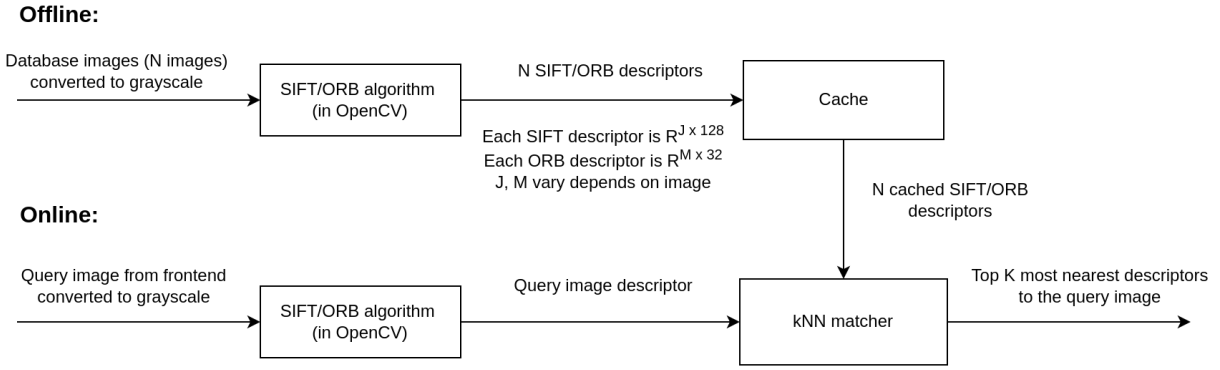


Figure 1: Pipeline for SIFT/ORB-kNN retrieval model

## 4.5 (Human annotated) Fine-tune database

In our discussion, we discuss an interesting method, that instead of relying on computer capabilities for the retrieval task, we can rely on human experts to annotate the ground truth of the corpora. Let's first assume the context of text corpus, we can have human experts annotate common terms used in query for more expected results. In other words, we treat some common queries as a special class to be fine-tuned by more expensive human-labor operations. When we get a query that is similar to a common query, we return our annotated result which we believe to be more desirable than the result of a machine learning model.

Since we are working with image corpus, we can coarsely classify the database into classes, and do

the human annotation on a few images from some classes to get a fine-tune database. Instead of finding the most similar images to the query in the corpus, we may **find the most similar queries in our fine-tune database** and return their (cached) fine-tuned results instead.

This method may make the retrieval results biased and unstable, getting high precision on some queries that is close to queries used for evaluation but low on others. However, in terms of performance, the cached results make retrieval faster, since the number of common queries is often much lower than the image corpus size. The larger the size of the fine-tuned database, the higher the chance the model returns the highly desirable results, since there is a higher chance of getting a similar cached query. We did implement this approach using 55 human queries for evaluation and cache them in our database, described in section 4.2. However, we cannot use these 55 queries for this method since we need them for evaluation, also, more queries are necessary. In our experiment, since we don't have enough human resources, we decide to drop this approach to the task.

Another discussed approach is that we can apply the above approach for any model/method that is computationally expensive but has high potential precision (such as our expensive SIFT-kNN method if its evaluated mAP is high). We treat our (computationally) expensive model the same role as "human expert" in the previous approach by having it compute the ranked lists for common queries then cache them into a database for future look-ups. On our task, since our query term is images, not text, the concept of "common queries" seems not intuitive. Therefore, again, we decide to drop this approach to the task.

### 4.6 ORB descriptor with kNN matching

SIFT is notable for its robustness in in computer vision tasks. However, when implementing with k-Nearest Neighbor (kNN) matcher in the Oxford5k dataset, the retrieval time of about 2 to 10 minutes per query (see section 6) is so high that it is not acceptable. We replace the SIFT features with the ORB features, which are rotation invariant and resistant to noise [13]. Although the ORB features are only partial scale-invariant, the size of each descriptor is smaller, 32 instead of 128 float numbers in SIFT. This results in faster computation in the nearest matching step, around 4x to 8x in our implementation, although the asymptotic complexity is the same. This leads to a tradeoff between precision and performance.

The pipeline illustrated in figure 1 stays the same as described SIFT-kNN method in section 4.4, there are differences:

- We use ORB algorithm to extract ORB features, instead of SIFT algorithm.
- Each ORB descriptor is in $\mathbb{R}^{N\times 32}$ where $N$ is the number of keypoints in an image. Therefore, a keypoints is described by a vector in $\mathbb{R}^3 2$, instead of $\mathbb{R}^1 28$ as SIFT.
- Although the idea of kNN matching is the same, the distance between two descriptors are defined differently. Since ORB is a binary descriptor [13], it is more sensible to use Hamming distance as the distance between two keypoints, rather than L2 distance. For two ORB descriptors $A \in \mathbb{R}^{n\times 32}$ and $B \in \mathbb{R}^{m\times 32}$, after the nearest neighbor matching, we have $s$ pairs of integer incides $(i_k,\ j_k)$ for $k = 0, ..., s-1$ where $0 \leq i_k \leq n-1$ and $0 \leq j_k \leq m-1$ together with their Hamming distance $d_k$. The distance between $A$ and $B$ is defined as the average Hamming distance of these $s$ pairs.

### 4.7 Ensemble Classified-feat with EfficientNet-B0 and ResNet50

Observing that after fine-tuning ResNet50 and EfficientNet-B0 on our dataset, both gets high mAP that could be ensembled to make our model better. We give 2 models different weights to affect the system decision. The score of the ensemble model is given by the weighted mean of scores of two model:

$$s = s_e \times \alpha + s_r \times (1 - \alpha)$$

where $s_e$ and $s_r$ are the score given by EfficientNet and Resnet model, respectively, and $\alpha$ is a constant. In our implementation, we choose $\alpha = 0.5$, this means that we take the average of two scores. Sequentially, we sort the result increasing by score and return top $k$ results as the ranked list.

### 4.8 Attention with Deep Feature

In this approach, we try another deep learning mechanism which apply to the model that need to focus on same special things of image or object - "Attention Module". In previous approach, we just see that the feature of each image just navigate to the feature of each class. In this dataset, it could has a lot of image which is not labeled in the right way for each class, then you can see in its ground truth. That is why we need another way to learn feature between each class for finding the similar features between them.

As you can see in the Figure 2, at first, we get the feature of each class through the backbone network of CNN (specifically it is EfficientNetB1). And then for learning the weight of each class that affect to each others, we stack L layers of Self-Attention module to the model so that we can learn the feature of each one. For training the parameter of Attention module, the output of this pipeline we navigate to each class correspondingly. That means after processing through the pipeline, each vector would assign to each class for each input already.

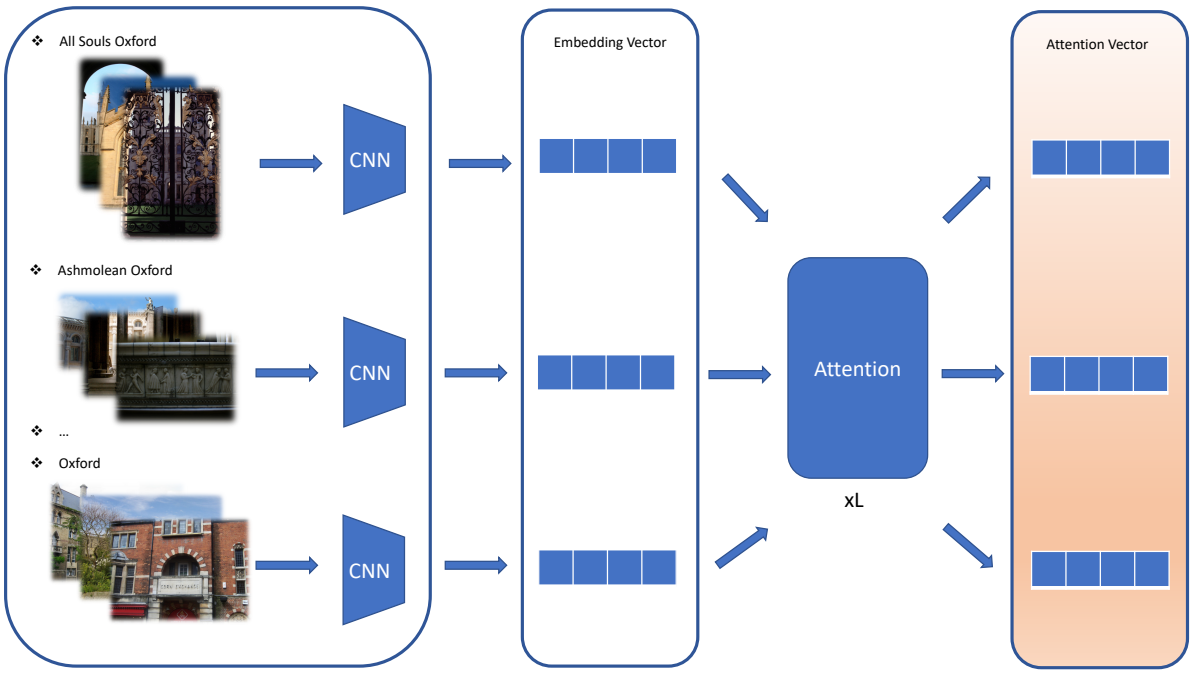


Figure 2: Pipeline for building the attention collection vector representing for each class

Next step, after we have the vector that representing for each class, we have to generate the feature that could leverage all the information from that class-featured vector. In the Figure 3, we utilize the CNN backbone as in the building query step, to find the correlation between in that image with other class. After going through attention, module we would get the final feature which could have the information of the collection image and each class for representation.

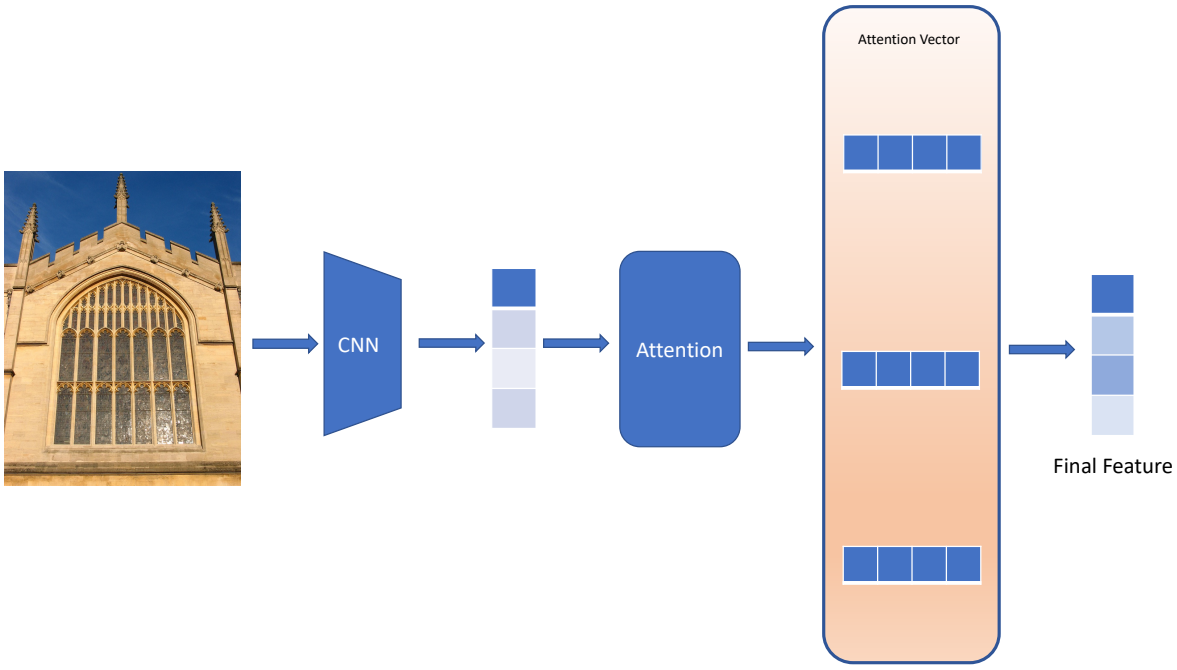


Figure 3: Pipeline for building final feature for each image in collection

# 5 Similarity measurements

In this work, we use multiple measurements such as Cosine distance, Euclidean distance, Complement of Normalized Dot product, and Hamming distance. The reason behind why we choose to use the 'distance' to measure the similarity is for uniform the way the models return top K smallest score.

## 5.1 Cosine distance

The cosine distance indicates two vectors overlapped that makes its distance/difference be 0. The Cosine distance [8] between 2 feature vectors $u$ and $v$, is defined as

$$1 - \frac{u \cdot v}{||u||_2||v||_2}$$

## 5.2 Euclidean distance

The Euclidean distance (or called as L2 distance) between 2 feature vectors $u$ and $v$ shows how far/difference of the two are. The nearer distance means the two images are similar. L2 distance is calculated as

$$\sqrt{\sum_{i}^{m}(u_i - v_i)^2}$$

where m is number of dimension of the vector.

## 5.3 Complement of Normalized Dot product (CoNDP)

The dot product tells what amount of one vector goes in the direction of another. The formula of complement of normalized dot product of two feature vectors $u_i$ and $v_i$ is

$$1 - \frac{u_i \cdot v_i - \min_{j \in n}(u_j \cdot v_j)}{\max_{j \in n}(u_j \cdot v_j) - \min_{j \in n}(u_j \cdot v_j)}$$

where n is total number of images in dataset.

### 5.4 Hamming distance

The Hamming distance between two feature vectors of equal length is the number of positions at which the corresponding values are different. The smaller the Hamming distance value is, the more similar the vectors are.

## 6 Experiment

### 6.1 Dataset

The dataset [2] consists of 55 queries over 11 different landmarks, each presented by 5 queries. For each query, human experts classify the dataset into 4 classes: "Good" (nice, clear picture), "OK" (more than 25% visible), "Junk" (less than 25% visible, or high level of distortions) and "Bad" (the query object is not present).

### 6.2 Evaluation and Result

We evaluate our methods using the mAP (mean Average Precision) metric over all 55 queries. For calculating the Average Precision on each query, we adopt the method of [12], we treat each retrieved "Good" or "OK" image as a match, each retrieved "Bad" image as an unmatch and each retrieved "Junk" image as null (as if the image never appears in the database and the ranked list).

Table 1, 2 show the mAP accuracies of our methods with top 100 and top 200 retrieved images, respectively. Table 3 shows the average response time the back-end server of every model.

Aside from 1000-feat model, we also evaluate the model that haves 2048 features extracted. The resulting mAP using cosine distance is around 3.88% to 4.02%.

A notable observation is, the SIFT method is computationally expensive since, for each query image, we must match its SIFT descriptor with all SIFT descriptors of all images in the database. If there are $k$ images in the database, the query image descriptor has $n$ keypoints, and the database images averagely have $m$ keypoints, then the complexity of brute-force matching is $O(nmk)$, which is the bottleneck of the method. Indeed, one query takes the back-end server around 2 to 10 minutes of process time.

We see that mAP of the kNN method using ORB descriptor drops from 22.71 to 13.52 (40.47%) and 23.27 to 13.83 (40.57%), compared to using SIFT descriptor. However, the time for one query is around 20 seconds to 1 minute, which is 4x to 10x faster. We think that this is a reasonable tradeoff between precision and performance.

Another notable observation is that the model using classified features extracted by EfficientNet-b0 performs well on cosine distance and dot product distance but not if L2 distance is used. We think this is reasonable since the extracted features of EfficientNet-b0 are of dimension $1280 \times 7 \times 7$, its length is subject to noise if the L2 distance is used.

| Method | Cosine distance | L2 distance | CoNDP | Hamming Distance |
|---|---|---|---|---|
| 1000-feat | 2.09 | 1.4 | 0.3 | - |
| 2048-feat | 3.88 | - | - | - |
| SIFT | - | 22.71 | - | - |
| ORB | - | - | - | **13.52** |
| Classified-feat (ResNet18) | 31.37 | 31.4 | 5.07 | - |
| Classified-feat (ResNet50) | 34.88 | 31.97 | 8.09 | - |
| Classified-feat (EfficientNet-B0) | 37.86 | 13.21 | **36.19** | - |
| Classified-feat with Attention | **49.5** | **32.98** | 33.56 | - |
| Ensemble Classified-feat (EfficientNet-B0 + ResNet50) | 47.8 | 15.76 | 34.8 | - |

Table 1: mAP (%) of methods with top 100 images retrieved in different similarity measurement

| Method | Cosine distance | L2 distance | CoNDP | Hamming Distance |
|---|---|---|---|---|
| 1000-feat | 2.28 | 1.54 | 0.46 | - |
| 2048-feat | 4.02 | - | - | - |
| SIFT | - | 23.27 | - | - |
| ORB | - | - | - | **13.83** |
| Classified-feat (ResNet18) | 35.7 | 35.62 | 6.05 | - |
| Classified-feat (ResNet50) | 40.01 | 35.65 | 11.1 | - |
| Classified-feat (EfficientNet-B0) | 42.8 | 15.24 | **40.02** | - |
| Classified-feat with Attention | **55.92** | **38.27** | 39.2 | - |
| Ensemble Classified-feat (EfficientNetB0 + ResNet50) | 51.34 | 18.24 | 38.07 | - |

Table 2: mAP (%) of methods with top 200 images retrieved in different similarity measurement

| Method | Response time (s) |
|---|---|
| 1000-feat | 0.2 |
| 2048-feat | 0.2 |
| SIFT | 300.63 |
| ORB | 37.54 |
| Classified-feat (ResNet18) | 0.11 |
| Classified-feat (ResNet50) | 0.25 |
| Classified-feat (EfficientNet-B0) | 1.32 |
| Classified-feat with Attention | 1.3 |
| Ensemble Classified-feat | 1.8 |

Table 3: Average back-end response time for each model

# 7 Conclusion

In this report, we report about our image retrieval system and evaluation of our methods on the Oxford5k dataset [2]. We propose and implement many retrieval models, evaluate them and compare their results for academic purposes.

We apply the Computer Vision techniques through SIFT and ORB algorithms to achieve 23.27% and 13.83% of mAP. These techniques are quite old but gave such acceptable results. However, comparing to Deep Learning techiques, these two lost in both mAP and process time. The Attention method reaches the highest mAP score of 55.92% when retrieving top 200. Besides, its rapid speed (1.3s) can please the user when query an image.

Furthermore, we observe that cosine distance is the best similarity measurement. All of our deep learning methods using this measurement returns better mAP.

We discuss about the fine-tune database method and leave this for future work.

# References


[1] See it, search it | Bing Visual Search. URL https://www.bing.com/visualsearch.

[2] The oxford buildings dataset. https://www.robots.ox.ac.uk/~vgg/data/oxbuildings/index.html.

[3] Fastapi. https://fastapi.tiangolo.com/.

[4] Google Images Search. URL https://www.google.com/imghp.

[5] Develop and deploy the best web experiences in record time. https://www.netlify.com/.

[6] React. https://reactjs.org/.

[7] SauceNAO Reverse Image Search. URL https://saucenao.com/.

[8] Scipy.spatial.distance.cosine. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cosine.html.

[9] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60 (2):91–110, Nov. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544.

[14] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.