ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Optimising similarity search between images

Semester Project

Author:
Guillaume RAILLE

Supervisors:
Professor Karl ABERER
Rémi LEBRET
Hamza HARKOUS

Winter 2018

Introduction

Purpose of this project

This project takes an experimental approach to implement, assess and compare different solutions to efficiently detect similarities between images at large scale. It is based on several state-of-the-art techniques and methodologies that are still actively growing and being developed today such as Convolutional Neural Network and transfer learning. The way to address the issue of detecting similarities between images at large scale was largely influenced by this paper [1]. This paper can be considered as a solid starting point for this project as it helped us define an efficient pipeline and evaluation methodology.

Some definitions

Convolutional Neural Network (CNN)

As we introduced just above, Convolutional Neural Network is at the essence of this project and understanding the main concept behind it was also part of the work. In our case we consider a Convolutional Neural Network that takes as input an image, perform multiple operations and convolutions on it and gives as output a fully connected layer (FC) that is just a one dimension vector with each value being a weight that represents the vote of a given image to be part of a given class. The different values of the masks (in the convolutional layers) and the voting weights are learned through back propagation (gradient descent). There are nowadays many implementations of CNN. In this project, we focused on a CNN called ResNet which was developed at Microsoft for the COCO 2015 competition [2]. ResNet stands for Residual Networks, they were the first to introduce shortcut connections between layers to improve the performance of their CNN. More specifically we used throughout the project their 18-layer ResNet implementation.

Transfer learning

A problem with CNN is that before it is trained and its voting weights and values are learned from the targeted data, it is absolutely unable to predict anything. Unfortunately, before becoming useful, CNN usually needs to be trained on a lot of data and it takes a very long time. Transfer learning solves this problem by using a pre-trained CNN (on an unrelated dataset) to solve our specific classification problem. There is basically two ways to perform transfer learning: Fine-tuning or Fixed Feature extractor. The first method consists in using pre-trained weights instead of random initialisation and then train the CNN again on the specific targeted data. The second one consists in fixing the weights for all the network except the Fully Connected layer and perform only a forward operation on the targeted data. The latter is much faster and our project pipeline is based on this concept.

Hardware and Setup

The aim of best efficiency to perform detection of large scale similarity between image is obviously highly dependent on the hardware used to perform the search

in the first place. In our case, we had the opportunity to compute and compare the different results obtained on a server with both GPU and CPU capabilities. During the project we used a Nvidia Titan X to realise the computation on GPU, and an Intel(R) Xeon(R) CPU E5-2620 v3 with 24 cores for computation on CPU. To easily handle GPU computations and switch between CPU/GPU memory, we used the open-source Pytorch[1] library.

1 Pipeline

To evaluate different methods to detect similarities, we followed a specific pipeline similar for each method tested. As previously mentioned, this pipeline is highly inspired from this paper [1]. It is composed of three steps before assessment and evaluation of the results. First we extracted features on each image using a pre-trained CNN, then we reduced the dimensionality of the data through a PCA dimensionality reduction to improve the speed and performance of the computations, finally we applied the k-NN (nearest neighbour) method we wanted to test on the extracted features.

1.1 Feature Extraction

Feature Extraction is very similar to what we described in transfer learning (Fixed Feature extractor). We used a pre-trained CNN (ResNet-18) trained either on ImageNet or Places365 (see more in Dataset's section) and we fixed their weights. Then we extracted for each image the "features" corresponding to the output of the last convolution layer (just before the fully connected layer). In this specific case we ended up with a one vector with 512D for each image in our dataset. The output of last Layer of our CNN represents the high-level features from our convolutional neural network and describe each corresponding image with 512 values. We could use this "description" of the images as is and proceed directly to the third step of our pipeline to find the most similar images inside our dataset but to optimise the process both in computation time and quality of the results we decided to also perform a PCA reduction step.

1.2 Dimensionality Reduction (PCA)

We obtained from the previous step an aggregated matrix of extracted features from each image (a (# of images x 512) dimension matrix). Reducing the dimension of this matrix using PCA from 512 to 128 helped us to get a great increase in computation time while also improving the accuracy (see results). To perform PCA on GPU we used the SVD Pytorch function which performs the SVD decomposition of a given input matrix. From the SVD decomposition we could compute the dimension reduced matrix (with $\hat{U}$*S) equivalent to the PCA reduction. This dimensionality reduction led to the use of two datasets throughout the project:

- raw features (corresponding to the raw 512D vector extracted for each image)

[1] `http://pytorch.org/`

- PCA features (corresponding to the 128D PCA reduced vector for each image).

1.3 k-NN methods

The third and last step was to actually compute similarities between images using the features previously extracted. To do so we used different k-NN methods (Nearest Neighbor) and we evaluated their performance. We tried some on GPU other on CPU and we even compared the results with our own implementation of k-NN.

1.3.1 Brute-force

What we simply call in this project "Brute-force" was our own implementation of a k-NN Brute-force method. It is a very simple method that check for each input vector all possible neighbors and return the closest one. We made it run on GPU using two already optimised functions from PyTorch: "topk" to find the k closest neighbors and "norm" to compute the distance between the different feature vectors. We mostly used this approach to produce a "k-NN ground truth" that could be used as comparison with other approximate methods to assess their performance. It was also interesting to check their computation time performance against our simple brute-force method.

1.3.2 Annoy

Annoy[2] is an approximate Nearest Neighbors library developed in C++ and optimised for memory usage and loading/saving to disk. It was built by Erik Bernhardsson at Spotify during Hack Week and is still currently being used. It supports a full Python API that we used in this project. The main advantage of this library (beside the fact that it is fast and it is optimised to reduce footprint memory), is that it has the ability to use static files as indexes to compute k-NN. In particular, it makes it possible to pass around indexes as files and map them into memory quickly. Annoy was only developed for CPU and as such we used it only on CPU. It is also worth noting that Annoy is an approximate method and we will need to compare it with an exact method (brute-force) to assess its performance.

On a more practical point of view, Annoy uses Random Projection to build up a tree. At every intermediate node in the tree, it divides the space into two subspaces using a random hyperplane (hyperplane chosen by sampling two points from the subset and taking the hyperplane equidistant from them). Doing this operation multiple times will generate of forest of trees which represent the index previously mentioned. As such Annoy let us play with two parameters:

- *n_trees*: the number of tree we are willing to build. More *n_trees* implies more build time and more memory usage but also more precision.
- *search_k*: the number of node to inspect when searching for the nearest neighbor in the trees. Higher *search_k* leads to higher precision but it increase the search computation time.

[2] `https://github.com/spotify/annoy`

1.3.3 NMSLIB

NMSLIB[3] or Non-Metric Space Library (NMSLIB) is an efficient cross-platform similarity search library and a toolkit for evaluation of similarity search methods. It allows searching in generic metric and non-metric spaces. However in this project we only focused on the Euclidean space, and always used a "l2" norm to find similarity between extracted features. Fortunately, NMSLIB claims to have very good performance in the Euclidian domain as well. NMLSIB has been developed and optimised in C++ and also provide a Python API that we used in this project. The library proposes many different similarity search methods with different performance. In this project we chose to use only one of the most promising method they provide: Hierarchical Navigable Small World graphs (also known as HNSW [3]). This approximate approach is fully graph-based, without any need for additional search structures and is able to strongly outperform many previous state-of-the-art vector-only approaches. NMSLIB has been written for CPU and as such we will make or computation on the CPU we have at hand.

1.3.4 Faiss

- What is Faiss?

  Faiss[4] is a library that was originally developed at Facebook AI research. The library was at the beginning a project that led to a published paper [4]. At its essence, the project focused on similarity search at "billion scale" by optimizing GPU usage. The library that came out of it (Faiss) is by nature optimized for GPU and propose an efficient design for brute-force, approximate and compressed-domain search based on product quantization (Voronoi cells).

- How did we use Faiss?

  The Faiss library is very efficient to perform similarity search on CPU and GPU. However, in this project, we focused on the GPU performance of it as it is where the library shine. We applied the usual pipeline on Faiss, the only singularity here was that the library offered much more possibilities compared to all the previous one. In order to properly assess the performance of Faiss approximate method, we made several benchmarks playing around with the parameters (mostly nlist representing the number of Voronoi cells to split our data on, and nprobe the number out of nlist to check when performing the search). We first did our experiment on a k-NN search with k=2 to get the vector itself and its nearest neighbor then we repeated our analysis with k=6 to get the five closest neighbors.

[3] `https://github.com/searchivarius/nmslib`
[4] `https://github.com/facebookresearch/faiss`

2 Experimental Setup

In the previous part, we described the pipeline we used to study and evaluate different approach to image similarity detection. In this part, we will explain what data and metrics we used in our experimental approach to solve this problem.

2.1 Datasets

All in all, in this project, we used three different datasets for two main purposes: two datasets for feature extraction and one dataset to evaluate the performance of our methods.

2.1.1 ImageNet and Places365 for feature extraction

As described in introduction (cf transfer learning) and in the feature extraction part of the Pipeline(cf 1.1), it is wise to extract the features directly from a pre-trained network. After choosing the CNN that we would be using (ResNet-18) we had several choices regarding the data it was going to be pre-trained on. We chose to study and see the resulting difference between two training datasets:

- **ImageNet**[5]: ImageNet is a very wide library of labeled images (almost 14.2 million images) organized according to the WordNet hierarchy (tree of words) in which each node represents an average of 500 words. ImageNet is very famous for the challenge they propose (LSVRC) from which many image classification solution and in particular CNN were developed such as the ResNet used during this project. It's important to note that most images from this library are precisely zoomed at one or a few particular objects corresponding to the word they are stored at. (see Figure 1a for an example)

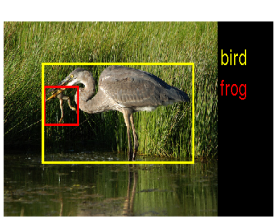
- **Places365**[6]: Places365 is a dataset composed of 1.6 million train images from 365 scene categories. The images are labeled and represented by Meta attributes as well as a scene category. In this dataset, the pictures represent each time a whole scene (see Figure 1b) and not a precise object as in ImageNet. We will see the influence of it in the last part of this project where we compare the performance of our Pipeline on different subgroups of images.

Using transfer learning and pre-trained CNN has some advantages and inconvenient that we will mostly see in the last part of this project (Training set analysis). Most of the results obtained in this project and unless specified otherwise will result from features extracted from the ResNet-18 pre-trained on the Places365 dataset.

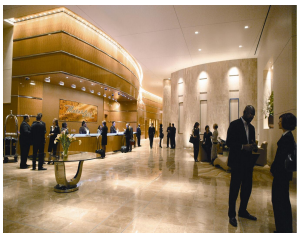
2.1.2 MSCoCo for evaluation

In order to test the different methods and get meaningful results we also needed a third independent dataset. This dataset had to be wide as we were willing to

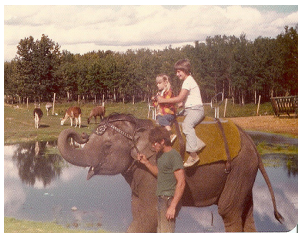
[5] `http://www.image-net.org`
[6] `http://places2.csail.mit.edu`



(a) ImageNet image



(b) Places365 image



(c) COCO image

perform our evaluation at a large scale. These are the reasons why we used the COCO[7] dataset for evaluation. COCO is a large dataset of more than 330'000 images humanly labeled with caption. The images present in this dataset are again more representing a scene rather than a specific object as we can see on Figure 1c. All the results we obtained are coming from the training set 2017 of the COCO challenge which is composed of roughly 118'000 images with 5 humanly made caption each. Another good aspect of using COCO is that it provides an integration with Pytorch and makes it easy to load the images and their corresponding captions through the COCO API.

2.2 Evaluation Technique and Metrics

After performing the different operations of our pipeline on the datasets, we needed a way to evaluate the performance of the different k-NN method and library tried during the process. In order to do so, a measure we constantly used during the evaluation phase was the computation time in second. If the method used some kind of indexing, we also computed separately the time of building the index, the time of searching the index, and the total time. As we are using a several approximate methods, it was also interesting to look at the performance of their results. For that, we used two strategies described below: similarity with the ground truth (brute-force search), and the BLEU score.

2.2.1 Compare similarity with brute-force

To compare the similarity between two results from different libraries we computed their similarity with our basic brute-force results. It is a good estimator at how the approximate method performed compared to a much longer but "perfect" result. We used in this project two kinds of similarity score:

- Strict Similarity: given several neighbors for a given image this score represent the number of identical value compared to the ground truth (respecting the order of the value) divided by the number of neighbors. Then we average this score on every image of the dataset.
- Permissive Similarity: this is the same as the strict similarity except that this method doesn't look at the position (the order) of the neighbor it just takes the intersection between the brute-force result and the currently being evaluated result for a given image and divide it by the number of neighbors.

[7] `http://cocodataset.org`

Note: Since the brute-force method also use an approximation to compute their neighbors (floating point approximation) some very small difference might be noticed even between two different brute-force implementation. For example, the first and second neighbor of a given image might be swapped if the difference in distance between the two is less than epsilon. That being said the comparison with brute-force results is still a good estimator at how the approximate method performed.

These two metrics allowed us to grasp a good idea about the performance of the method. However the brute-force results aren't absolute ground truth as well and neighbors that are not considered by a brute-force method might be still valid when looking at the two pictures. This is the reason why we introduced the BLEU Score.

2.2.2 Bleu Score

BLEU or Bilingual Evaluation Understudy (paper [5]) is an algorithm to compare a given candidate text with a certain number of reference text. In order to so it uses a combination of n-gram. The best possible value for this score is 1 if all the sentences are identical. It goes down to 0 if there is nothing in common between the candidate sentence and its references. It mainly allowed us to have a metric that was independent from any k-NN search but more related to the captions and human label linked with the images. As we had five captions for each images we first used it by taking one by one each five captions of the original image as candidate and compared it with the five captions of the neighbors image as references. This approach provided very similar results between each method so we decided to use the five first neighbors captions as reference and the original caption as candidate to compute the BLEU score which led to slightly improved results.

Note: Computing the BLEU score for such a large dataset was a relatively intensive computing task so we developed a shell script to parallelise the process on the different CPU core we had at hand.

3 Results

The results gathered here represent all the different metrics described just above computed after applying the pipeline on our 118'000 images COCO dataset. In this section, we described and comapred the results for each k-NN method individually. Once again we considered two datasets here: RAW features and PCA features. The first one is a ˜118'000 rows and 512 column matrix representing the feature extracted for each image (see pipeline) and the latter is the PCA reduction of this matrix (˜118'000 rows and 128 columns).

3.1 Brute-force

As a reference to compute other similarity metrics, the brute-force method was the first method we implemented and computed. As explained in the k-NN method section, we created this basic brute-force method ourselves using some

Pytorch optimised function that runs on the GPU. The computing time results are the one we can observe on Table 1.

	Dataset	Total Time
k=2	RAW	5min 33s
	PCA	3min 31s
k=6	RAW	6min 2s
	PCA	3min 38s

Table 1: Computation time of our own brute-force method on the whole dataset.

Table 1 shows well the effect of the PCA dimensionality reduction. Indeed the computation time of our brute-force method was reduced by more than 35% in both cases (k=2 and k=6) while we reduced the dimension from 512 to 128 (75% decrease). We also notice that increasing k (the number of neighbours wanted) doesn't affect much the search time. Indeed as it is a brute-force search all the distances are computed and classified for each point in any case. After assessing the computation time, we could compare the performance of our brute-force search between RAW and PCA with our different similarity metrics (Table 2).

Strict Similarity k = [1]	Strict Similarity k = [1:]	Permissive Similarity k = [0:]
69.14 %	15.74 %	74.65 %

Table 2: Similarity between RAW and PCA dataset using Brute-force. k = [1] represents the first neighbor, k=[1:] represents the five first neighbors, k=[0:] represents the six first neighbors including self.

On Table 2, we looked at the similarity scores using the usual similarity functions defined previously. The first column represents the similarity between each second neighbor (first neighbor if we exclude self), the second represents the first five neighbors and the last column represent all the first six neighbors including self. We can observe a quiet low strict similarity result on the first five neighbors, however, we clearly see through the Permissive Similarity score that most of the top five neighbors are the same (almost 75 % in this case) so we can be confident that our PCA results are still close enough to our RAW results while being much faster to compute. To confirm this assumption we compute the BLEU Score on the caption of the neighbors and compare the result with and without PCA (see the description of the process in the Evaluation Metrics section)(Table 3).

BLEU on RAW dataset	BLEU on PCA dataset
0.5418	0.5420

Table 3: BLEU score on RAW and PCA dataset after using Brute-force.

Table 3 presents a very small difference between the two datasets in terms of BLEU score. We notice a slightly higher score for the PCA dataset that confirms the fact that using PCA reduced features, not only reduce the computing time but also keep the same performance in image similarity detection or even

slightly improves it.

We can confirm the results obtained in this part on Figure 2. We observe the original picture followed by its brute-force closest neighbor with RAW feature and PCA feature. We see that in most cases, the raw and PCA images are the same and even when they are not the resulting picture is still close to the original one.

3.2 Annoy

The first approximate method we tried was Annoy. As described in the k-NN method section Annoy can only be run on CPU. During this project we tuned two parameters: *n_trees* and *search_k* (see k-NN method section for more details). *n_trees* influence only the build time while *search_k* influence only the search time. We present on Table 4 and 5 the computing time of Annoy on our datasets as the two parameters vary.

dataset	n_trees				
	1	4	16	32	64
RAW	1.13 s	4.55 s	18.2 s	13.6 s	2 min 21 s
PCA	0.42 s	1.78 s	7.15 s	13.9 s	56.3 s

Table 4: Building time of Annoy index on the RAW and PCA dataset as n_trees vary.

	dataset	search_k				
		1	8	64	512	4096
k=2	RAW	54.7 s	54.0 s	54.2 s	89.5 s	322.5 s
	PCA	6.5 s	7.7 s	6.7 s	22.7 s	107.7 s
k=6	RAW	53.4 s	55.5 s	53.9 s	90.5 s	323.9 s
	PCA	7.0 s	8.0 s	6.9 s	22.9 s	107.9 s

Table 5: Searching time of Annoy on the RAW and PCA dataset as search_k vary.

Table 4 shows that the relation between the building time and the number of trees is strictly linear. It is also important to note that there is a constant to add to this value (before building the index the dataset needs to be added to the library class) this constant depends solely on the size of the dataset it was about 500 ms for the PCA reduced dataset while around 2 seconds for the raw dataset (note the factor 4 between the 2 value is the same as the dimensionality reduction factor).

On Table 5, we can clearly see the computation time improvement provided by the dimensionality reduction on every search. We also notice that changing the number of neighbors we want k doesn't have much impact on the search time. We obviously see and increase in searching time as the number search_k of nodes to search gets larger. The next step was to see the improvement in our results following the increasing computation time. In order to do that we compared the results from Annoy with our brute-force "ground truth" results (Table 6).

			n_trees				
			1	4	16	32	128
search_k	1	S.S.	1.5%	3.2%	5.5%	6.5%	8.8%
		P.S.	23.0%	30.3%	37.3%	40.2%	45.2%
		S.F.	26.9%	36.1%	44.5%	47.8%	53.6%
	8	S.S.	1.5%	3.2%	5.5%	6.5%	8.8%
		P.S.	23.0%	30.3%	37.3%	40.2%	45.2%
		S.F.	26.9%	36.1%	44.5%	47.8%	53.6%
	64	S.S.	1.6%	3.3%	5.6%	6.7%	9.0%
		P.S.	23.5%	30.8%	37.7%	40.7%	45.6%
		S.F.	27.4%	36.5%	45.0%	48.2%	54.0%
	512	S.S.	1.7%	25.9%	35.2%	39.3%	45.1%
		P.S.	56.1%	65.9%	73.3%	76.2%	79.8%
		S.F.	61.6%	71.7%	79.3%	82.3%	85.6%
	4096	S.S.	5.8%	77.1%	84.8%	86.5%	88.2%
		P.S.	86.1%	93.6%	96.0%	96.6%	97.1%
		S.F.	89.4%	95.6%	97.5%	97.9%	98.3%

Table 6: Similarity Score results for Annoy on the PCA dataset compared with PCA brute-force results. (S.S. represent strict similarity between the five nearest neighbors excluding self, P.S. represent the permissive similarity between the five nearest neighbors excluding self, and S.F. represent the strict similarity between the first neighbors.)

On table 6, we observe the increase of performance as *n_trees* and *search_k* increase. Overall we see that a small increase in the amount nodes searched doesn't improve much the performance (it didn't affect the search time significantly either) however a more significant increase definitely improves it. The important thing to note here is that when increasing one parameter while keeping the other one at the same value, we have an improvement in the performance. It means that a huge advantage of using Annoy is that we can choose between long building time or long searching time against performance by moving up and down one of these two parameters.

Regarding the BLEU score we performed the usual analysis on the PCA dataset using *n_trees* = 16 and *search_k* increasing from 1 to 4096. If we look at the corresponding similarity results (Table 6), we noticed that the similarity with brute-force PCA is low at low search_k and very high at high search_k. The results we obtained for the BLEU score are presented in Table 7

search_k	1	64	512	4096
BLEU score	0.54588	0.54587	0.5430	0.5422

Table 7: BLEU score on PCA dataset after performing Annoy with n_trees = 16 and different value of search_k.

Table 7 shows us that the BLEU score is very similar between each iteration. Furthermore the value seems to slightly decrease as we increase the precision of the method. It is also worth noting that every value we had here were higher than the BLEU score obtain on the brute-force method. All these elements tends to show that the BLEU score doesn't represent a good metrics to com-

pare such close results in terms of similarity.

We can confirm all the results obtained in this part on Figure 3 where we present some first neighbors of images resulting in two iteration of Annoy with different parameters.

To conclude about Annoy, after trying the different parameters we can say it is a good option for CPU nearest neighbor search. An advantage of this method is the possibility to control the build time and search time separately to lower or increase the precision. We got better computation time than our own implementation of a brute-force method (even running on GPU) with a small penalty on the precision. In the next part we will compare it with another CPU k-NN function using NMSLIB.

3.3 NMSLIB

As described in the pipeline section NMSLIB is a toolkit that offers many k-NN methods and evaluation possibilities. In this project, we tried only one of the most promising methods that NMSLIB put at our disposal: HSNW.

As done for the previous method tested, we first performed a benchmark the computing time offered by the method and then its performance against our basic brute-force method. Similarly to Annoy analysis, we started by looking at the building time of the index generated by HSNW. And then the searching time it required to query the k nearest neighbors.

The building time of HSNW for a given dataset is mainly determined by two parameters: M and efConstruction. M refers to the maximum amounts of connections that an element can have per layer in the graph generated by HSNW. EfConstruction controls the recall of the greedy search used when building the graph. More information about these two parameters can be found in the paper [3]. We see the results building for different values of these parameters on both the raw and PCA datasets on Table 8.

		EfConstruct		
		50	200	400
RAW	M=6	5.40 s	16.15 s	26.60 s
	M=24	9.82 s	32.66 s	56.92 s
	M=48	9.85 s	36.46s	1min 6.55 s
PCA	M=6	1.61 s	4.23 s	7.38 s
	M=24	2.42 s	8.16 s	15.44 s
	M=48	2.52 s	9.18 s	18.42 s

Table 8: Building time of HSNW on both the RAW and PCA dataset.

We can observe on Table 8 that both parameters increase leads to a building time increase. Again it is important to note that during build time a small constant of time depending on the size of the dataset should be added to this value (around 150ms for RAW and 50ms for the PCA dataset). We see that the influence of M on building time increase as EfConstruct increase. The increase in building time in both cases should give us better precision results and that is

what we can see on our PCA features dataset in Table 9 with a constant search parameter.

		EfConstruct		
		50	200	400
M=6	S.S.	72.37%	82.65%	84.40%
	P.S.	91.13%	94.96%	95.95%
	S.F.	91.47%	95.62%	96.33%
M=24	S.S.	93.51%	98.64%	99.05%
	P.S.	98.48%	99.71%	99.80%
	S.F.	99.02%	99.86%	99.92%
M=48	S.S.	94.31%	99.15%	99.57%
	P.S.	98.69%	99.82%	99.91%
	S.F.	99.17%	99.89%	99.94%

Table 9: Similarity of HSNW compared to brute-force on the PCA dataset. Search parameter Ef=50. S.S. represent the strict similarity with the five closest neighbors, P.S. represent the permissive similarity with the five closest neighbors and S.F. represent the strict similarity with the closest neighbor.

Fortunately, Table 9 validate our hypothesis and we see a clear increase in performance (similarity with a brute-force PCA). We also noticed that the results are much higher than what we could observe with Annoy (even with high computation time). To be able to draw a full comparison we looked next at the searching time and performance on Table 10.

	ef		
	10	50	100
Searching Time	0.91 s	2.14 s	3.64 s
Strict Similarity	87.74 %	98.34 %	99.17 %
Permissive Similarity	96.98 %	99.65 %	99.83 %
Strict Similarity (k=1)	99.07 %	99.73 %	99.78 %

Table 10: Similarity and Searching time of HSNW compared to brute-force on the PCA dataset with build parameters fixed at EfConstruct=200 and M=24.

Table 10 present the different similarity metrics as we were tuning the search parameter ef. ef allows us to determine how many connections to check in the graph. Increasing it, also increase the searching time but improve the performance. Overall the performance of HSNW is impressing. If we take M=24, efConstruct=200 and ef=50, we obtained almost 99% on every similarity metrics in a total of 10.2 seconds (8.16 + 2.14 = 10.2). When we obtained little lower results with Annoy in more than two minutes.

For the BLEU score, we obtained again very similar results as for Annoy. Confirming that the BLEU score is not suitable for neigbors as similar as they are in this case. However being very close to the brute-force gives us enough evidence to say that HSNW is a very performant method with impressive results on CPU. HSNW like Annoy allows to tune searching time and building time to any specific needs while HSNW gives us better results faster.

An example on five images is once again presented on Figure 4. We kept for this example efConstruct=200, M=24 and increased ef from 10 to 100. In this example we see only a perfect match with the PCA results indeed the similiraty score are above 99% in each case.

3.4 Faiss

3.4.1 Bruteforce with Faiss

To assess the performance of the brute-force k-NN search with Faiss, we computed the average time used by the method to build the index of the whole 118'000 images dataset as well as the time needed to perform the search (search the whole dataset over the index). We performed this computation on both the RAW features dataset and the PCA reduced one. In the two cases we used a number of neighbours to find k=2 and then k=6. We averaged our results over 10 iterations to make it more consistent. We obtained the results presentend on Table 11.

	Dataset	Building Index	Searching Time	Total
k=2	RAW	0.030 s	1.746 s	1.776 s
	PCA	0.013 s	0.633 s	0.646 s
k=6	RAW	0.067 s	1.763 s	1.830 s
	PCA	0.014 s	0.616 s	0.630 s

Table 11: Computation time of Faiss Bruteforce on the whole dataset.

On table 11, we observe for both datasets that the dominant factor is the search time or, in other words, the time to perform the query on each row of the dataset. Indeed as Faiss doesn't need to build an index in this case the building time is very fast and only the actual computation time is longer. We also observe that using the PCA reduced dataset lowered the time to compute by a factor of almost 3 (when we reduced the dimensionality from 512 to 128). It is interesting to note again that we don't observe significant increase in time when computing two or six nearest neighbours. To confirm the results obtained by the method, we computed the similarity between the dataset with and without PCA reduction and observed the exact same result as in Table 2. Which is very promising knowing that it took only a fraction of the time to compute (1.8 seconds only when it was more than 5 minutes in our brute-force implementation).

Figure 5 shows us again that some brute-force PCA images are not identical to their raw brute-force counterpart. However they still have some similarities with the original picture.

3.4.2 Approximate Search

Using again the Faiss library, we performed again on the full dataset the approximate method they propose using product quantization. We used previously described metrics to precisely assess the performance of this method as well as its computation time. We successively increased the different parameters (namely nlist and nprobe) to identify their impact on computation time and performance as described in the pipeline section. We can observe the results obtained on table 12 and 13.

nprobe	nlist							
	1024		2048		4096		8192	
	Search	Total	Search	Total	Search	Total	Search	Total
1	0.08 s	0.77 s	0.07 s	0.58 s	0.07 s	0.68 s	0.08 s	1.09 s
2	0.09 s	0.77 s	0.08 s	0.59 s	0.08 s	0.70 s	0.09 s	1.10 s
4	0.12 s	0.80 s	0.09 s	0.60 s	0.10 s	0.71 s	0.10 s	1.11 s
8	0.17 s	0.86 s	0.12 s	0.63 s	0.11 s	0.73 s	0.11 s	1.12 s
16	0.29 s	0.97 s	0.18 s	0.69 s	0.14 s	0.76 s	0.13 s	1.14 s
32	0.55 s	1.23 s	0.31 s	0.81 s	0.20 s	0.82 s	0.18 s	1.19 s
64	1.12 s	1.81 s	0.57 s	1.08 s	0.35 s	0.97 s	0.26 s	1.27 s

Table 12: Computation time of Faiss approximate method on the PCA features with k=6. (Search represent the search time (query time) and Total the sum of the search time and index building time.

nprobe	nlist							
	1024		2048		4096		8192	
	S.S.	P.S.	S.S.	P.S.	S.S.	P.S.	S.S.	P.S.
1	8.41%	41.52%	5.37%	36.37%	3.16%	31.71%	1.99%	28.26%
2	20.19%	58.72%	13.46%	52.32%	8.83%	46.74%	5.91%	42.45%
4	38.89%	74.41%	28.97%	68.09%	20.88%	62.19%	14.99%	57.36%
8	59.91%	86.13%	49.03%	81.14%	39.04%	76.06%	30.55%	71.53%
16	77.72%	93.53%	68.64%	90.20%	59.55%	86.68%	50.84%	83.21%
32	90.06%	97.51%	83.65%	95.60%	76.95%	93.49%	70.30%	91.37%
64	96.54%	99.23%	93.22%	98.39%	89.18%	97.31%	84.83%	96.13%

Table 13: Similarity score of Faiss approximate method on the PCA features with k=6. (S.S. represent the Strict Similarity and P.S. represent the Permissive Similarity compared to PCA features brute-force search).

On Table 12 and 13, there are a few interesting things we can observe. First we see that there is no case in which the total time is less than 0.63 seconds (the time necessary to perform the brute-force search). This result might appear as a deception, but it can be explained by the fact that building the index to perform the search takes significantly more time when using the approximate method. By looking further at the results we observe that even though the overall time is larger, performing only a search is several orders of magnitude smaller (depending on the parameters) which could be really interesting for applications that needs to query several times on the same indexed dataset. When changing the different parameters, we noticed quite obviously that increasing the number nprobe (number of veronoi cells to check out of nlist) leads to an increase in search time and an improved similarity score. Similarly increasing the number of cells (nlist) reduced the search time for a given nprobe (the search span is smaller) however it increases the index building time. It is also good to notice that we obtained results above 99% with nlist=1024 and nprobe=64 corresponding to a total (search + build) time of 1.81 seconds where HSNW took 10.81 seconds to achieve similar performance.

All these results can be confirmed visually on Figure 5. An important thing to note is that even though there is only 69% match in exact similarity between

first neighbours with PCA and RAW images, and even less with approximate method, the matched image seems to always be relevant.

Once again we computed the BLEU score and got very similar results. The similarity comparison with brute-force shows enough evidence to evaluate the performance of the method.

Finally, we saw that k-NN methods are always a tradeoff between fast build-time, fast search time and precision. When performing a single search on a whole dataset only once we would recommend using the Faiss brute-force search which is overall faster and more accurate. However when performing several searches on the same dataset it could be interesting to consider a fine-tuned Faiss approximate search.

3.5 Conclusion on the results

To conclude on the results, we had the opportunity to compare here the performance and optimisation of four k-NN methods. Two were computed on GPU (brute-force and Faiss) and two were computed on CPU (Annoy, NMSLIB). Our results showed that with the setup we had at hand, the GPU method: Faiss cameon top. For a one-time usage it is most efficient to use the brute-force method proposed by Faiss. However if we need to query several times an already built index then the approximate method can bring better performance. In a CPU environment only, NMSLIB through HSNW showed some very good results and outperformed Annoy by far on our dataset. Annoy still seems to be a good solution as it still performed better than our unoptimised GPU brute-force implementation. It could also be a good choice for its simplicity

4 Training set analysis

During all this project and through all the report, all the results we presented was obtained after extraction of the features using a resNet-18 CNN pre-trained on Places365.The questions we asked ourselves next were: what if we used a different training dataset for our CNN ? How can the training dataset of our neural network influence our results and if it does how should we choose our training datasets ? This part aim at answering these questions and in order to do so we will first make general observations on the full dataset, then we will focus on a specific subgroup of the dataset to illustrate the results.

4.1 General comparison of the features extracted

A first thing we did to be able to properly compare the results is that we extracted the features from our 118'000 images dataset from both ResNet (one pre-trained with imageNet and one pre-trained with PLcaes365) then we performed our best k-NN method for a one-time use. In this specific case we used Faiss brute-force. After that we compared the similarity and the BLEU SCORE given by faiss for the two different datasets. These are the results we obtained on Table 14 and 15.

	ImageNet vs Places365	
	RAW	PCA
Strict Similarity	1.10%	0.87%
Permissive Similarity	1.14%	0.97%

Table 14: Strict Similarity averaged of the first neighbors and Permissive Similarity on the five first neighbors given by pretraining on ImageNet compared to Places365 (full dataset).

ImageNet		Places365	
RAW	PCA	RAW	PCA
0.5725	0.5729	0.5418	0.5420

Table 15: BlEU Score obtained after different pretraining on the full dataset.

Table 14 shows us that the difference obtained after computing a brute-force search on features extracted from two CNN with different training set is very significant. For the first neighbor, we got around only 1% strict similarity which very small compared to what we could observe before. This sustain our hypothesis according which the pre-training dataset is an important factor in our image similarity pipeline. The BLEU score displayed on Table 15 is in this case more useful than the BLEU score we obtained before in this study. Indeed we notice in both cases a small increase in the score when switching from RAW extracted features to PCA reduced features but also a more significant change when changing the training dataset of our CNN. At first sight, ImageNet seems to perform better than Places365. However we must be careful as ImageNet is a library of zoomed object images with categories and Places365 a library of scenes object with scene attributes. This difference is important as in the COCO dataset (evaluation dataset) people labeled the images with what they saw and it may be easier for them to describe objects than the scene around the object. Hence the BLEU score might also be biased just by some objects appearing in the picture. To see more visually what it means on our dataset we had a closer look at a specific subset of data in the next part.

4.2 Traffic Light subset

In this part, we chose to study in more details the “traffic light” subset for a few reasons. First reason is that we were able to extract a sufficient amount of data from the COCO dataset by filtering every caption containing "traffic light" (1366 images). The second reason is that "traffic light" is one of ImageNet category meaning that ImageNet should perform very well at recognizing traffic light. Lastly, traffic lights are usually not far from a scene such as a road, some buildings, a street mostly outdoors that Places365 was mainly trained for. For all these reasons we expect to see very different results from the different pre-trained dataset with the CNN trained on ImageNet matching mostly the "traffic light" while the one trained on Places365 matching mostly the background from the scene. The result we got can be observed on Figure 6.

On Figure 6, we observe that the background of the scene with the shape of the buildings and the bricks seems to be matched with Places365 while the "traffic light" itself is matched with ImageNet.

We also computed similarity and BLEU score on this subset to confirm that it matched with the general case depicted above. We obtained the results described in Table 16 and 17.

	ImageNet vs Places365	
	RAW	PCA
Strict Similarity	1.32%	1.10%
Permissive Similarity	1.58%	1.54%

Table 16: Strict Similarity averaged of the first neighbors and Permissive Similarity on the five first neighbors given by pretraining on ImageNet compared to Places365 (Traffic Light subset).

ImageNet		Places365	
RAW	PCA	RAW	PCA
0.5845	0.5865	0.5684	0.5706

Table 17: BLEU Score obtained after different pretraining on the Traffic Light subset.

We can observe on Table 16 and 17 very similar results as the one observed in the general case. Arround 1% similarity between the matched neighbors and slightly higher BLEU score for ImageNet with PCA features always slightly above.

4.3 Which training set to choose?

To conclude this part about pre-training set for image similarity through CNN feature extraction, there is two main idea to take away. First we realised that the choice of pre-training set has a huge impact on the quality of our matching hence it is a determinant factor to take into consideration. The second is that the pre-training set chosen should depend on the matching wished to obtain every pre-training set will have its pros and cons in our case Places365 was very good at matching the scene and backgrounds of our dataset while ImageNet seems to be better at matching standalone objects present on the images.

Conclusion

In this project we were able to implement a pipeline to evaluate different state of the art k-NN methods and perform similarity search between images at large scale. We put in place different metrics to provide measure and numbers to our analysis. Through this practical approach we learned a lot about the main concepts and ideas behind the methods used. This study helps determine which methods and parameters are more suitable for which use case given certain conditions such as hardware set up, datasets, and results expected.

References

[1] Matthijs Douze, Hervé Jégou and Jeff Johnson. 2017. *An evaluation of large-scale methods for image instance and class discovery.*

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. 2015. *Deep Residual Learning for Image Recognition.*

[3] Yu. A. Malkov, D. A. Yashunin. 2017. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.*

[4] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. *Billion-scale similarity search with GPUs.*

[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. *BLEU: a Method for Automatic Evaluation of Machine Translation.*

Figure 2: First neighbor of some images using brute-force on raw and PCA features.

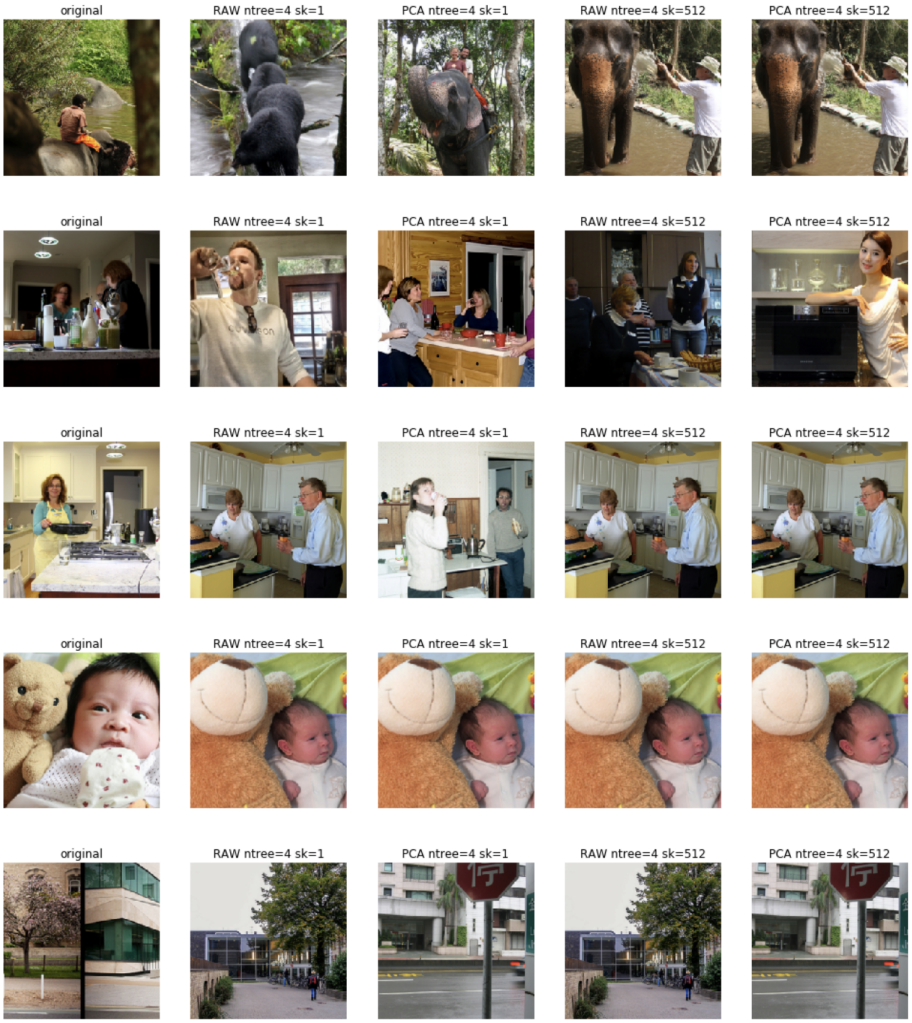


Figure 3: First neighbor of some images using Annoy on raw and PCA features with different parameters.

Figure 4: First neighbor of some images using NMSLIB on the PCA features with different parameters.

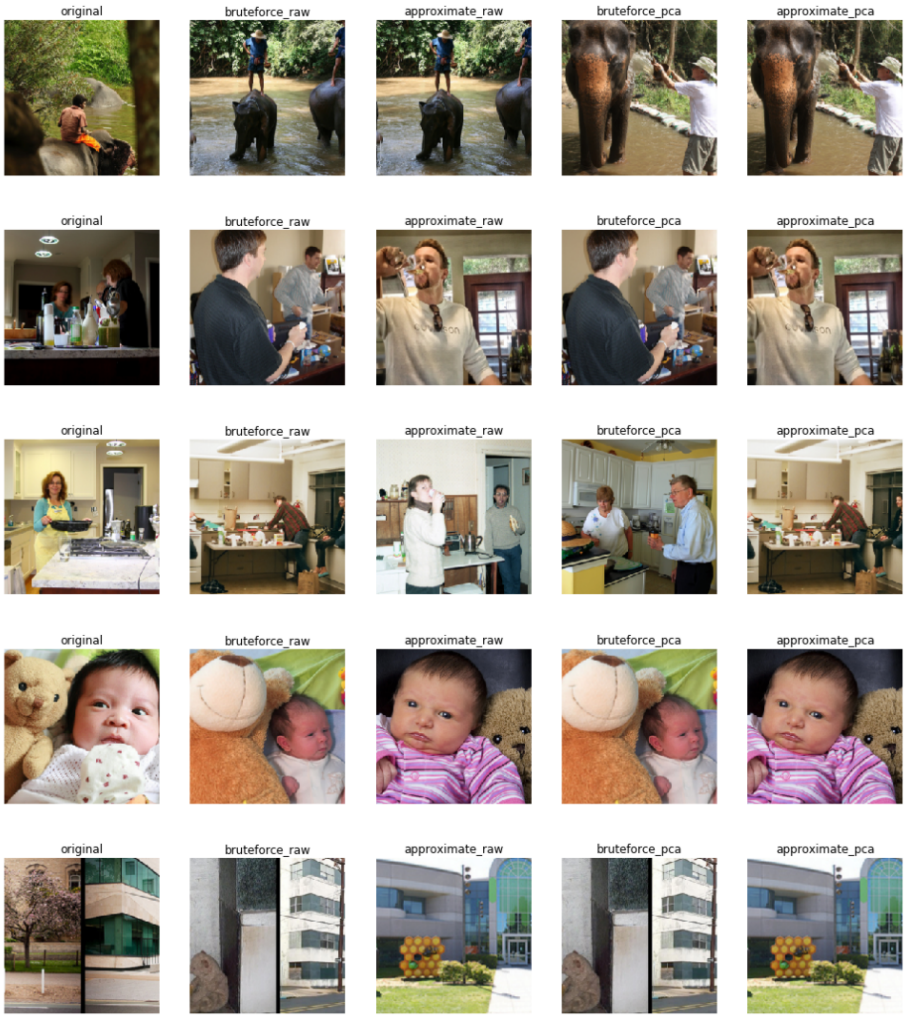


Figure 5: First neighbour of some images using bruteforce and approximate faiss methods on raw and PCA features.

Figure 6: First neighbour of a traffic light image with pretraining on ImageNet and on Places365.