Multimedia Information Retrival and Computer Vision

Alice Nannini
Marco Parola
Stefano Poleggi

A.Y. 2020-2021

Contents

1 Introduction

In this project we implemented a Web Search Engine working on the "ArtImages" dataset, and the "MirFlickr" dataset as distractor. The Web Search Engine is based on the **Locality Sensitive Hashing** (LSH) index and on the pre-trained convolutional neural network ResNet50 with weigths computed on the "Imagenet" dataset.

2 Data Analysis and Preprocessing

The Art dataset is composed of 5 categories of images, corresponding to the 5 classes:

- Drawings
- Engraving
- Iconography
- Painting
- Sculpture

2.1 Data cleaning

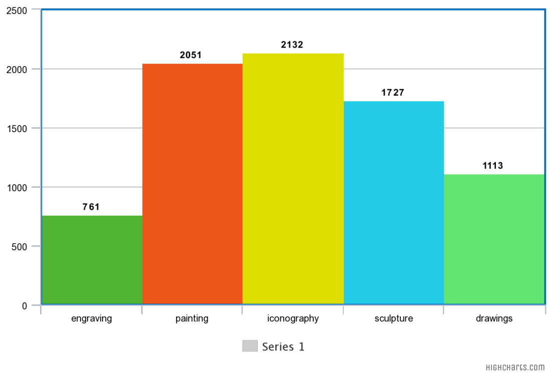
After a preliminary analysis of the dataset, we realized that some images were corrupted, both on the training set and the validation set, so we detected and deleted them using the following script.

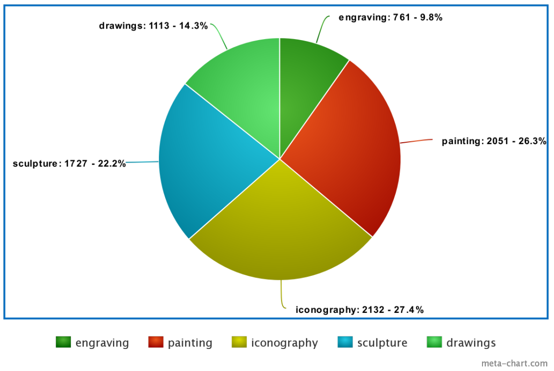
```
from PIL import Image
import glob

for filename in os.listdir(VALIDATION_DATA):
    try:
      image.load_img(VALIDATION_DATA + filename, target_size=(300, 300))

    except :
      print("ERROR" + VALIDATION_DATA + filename)
      os.remove(VALIDATION_DATA + filename)
```

The final training set is composed by a total of **7786** images, while the validation set counts a total of **856** images.

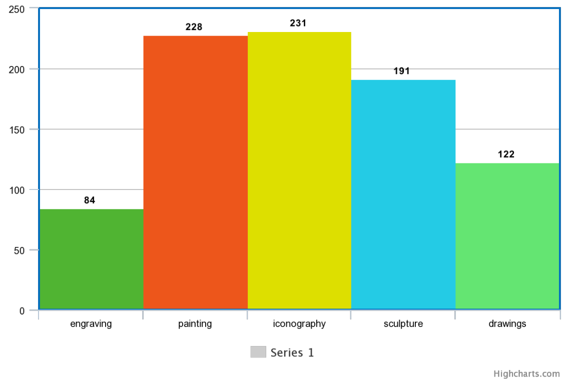
2.2 Class distribution

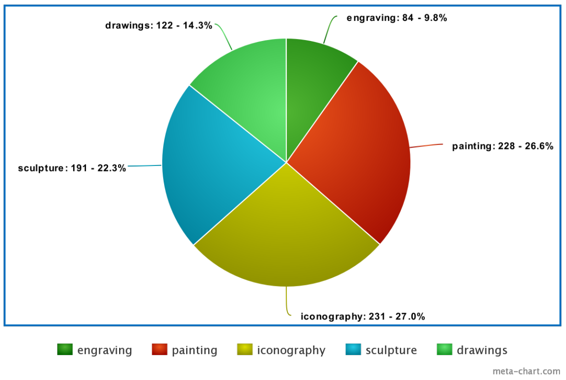
The following figures show the distribution of the classes of the training set, after the execution of the snippet of code shown in the previous section :





The following figures show the distribution of the classes of the validation set, after the execution of the snippet of code shown in the previous section :





3 Sequential search

The first implementation of our image search engine is based on a numpy array as data structure and a sequential scanning as search algorithm, using the following similarity measures:

- Euclidean distance
- Cosine similarity

We performed the evaluation of the sequential search function, using the validation set, so that we could have some starting data to compare with the performance of the LSH index. We compared the queries to the dataset (training set + distractor), first using the cosine similarity and then the Euclidean distance.
We used the images of the dataset as follow:

- Training set and Distractor are combined as data on which to perform the search algorithm
- Validation set images are used as queries to perform the search

The metric used to measure the goodness of the research is the **mean Average Precision** (mAP). Given an image **q** as query we computed the **average precision** (AP)

$$AP(q) = \frac{\sum_{k=1}^{n} Precision(Res_k(q))}{n}$$

Where

$$\text{Res(q) = top k element retrieved using q as query}$$

and

$$Precision(Res_k(q)) = \begin{cases} 1, & \text{if elem}_\text{k} \text{ is relevant.} \\ 0, & \text{otherwise.} \end{cases}$$

4 LSH Index

The final image search engine is based on a **Locality Sensitive Hashing** (LSH) index.
This index stores data in buckets. The intuition on which this data structure is based is that similar images will be stored in the same bucket with high probability.
This could be done building different hash functions **g()**, where each of them is defined exploiting the concept of random projections; more formally as

$$g(p) =< h_1(p),\ ...\ , h_k(p) >$$

Where each element $\mathrm{h_i}()$ is defined as

$$h_{\mathrm{i}}(p) = \lfloor (p * X_{\mathrm{i}} + b_{\mathrm{i}})/w \rfloor \qquad i = 1...k$$

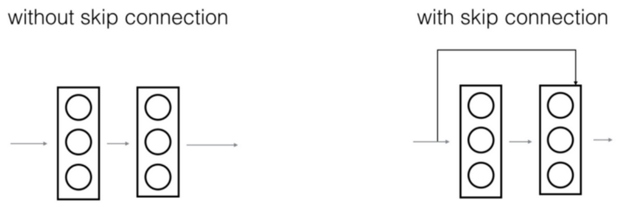
and represents a hyperplane described by X_i and b, while w is the length of the segments in which the hyperplanes are divided.
We defined L hash functions, corresponding to L buckets, then each image is inserted in a bucket for each of the L hash functions. When a query is passed to the search engine, its features are hashed using each g function and the corresponding buckets are identified. The elements belonging to them are retrieved and sorted, in order to select the top-k more similar to the query.

5 Features extraction

During this phase we extracted the features from the images and we stored them in order to reload them as numpy-array when it is necessary.
The pretrained convolutional neural networks we used are:

- ResNet50
- ResNet50v2

In this phase we extracted the features from the images using the pretrained convolutional neural networks **Resnet50** and **Resnet50v2** trained on *Imagenet* dataset, adopting a fetures extraction approach.
The strength of this convolutional neural network is the skip connection: the figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. This is called skip connection.



The following snippet of code shows the extraction of the training set using Resnet50, the same script is been used also with Resnet50v2 and for the validation set and the distractor.

```
def extract_features(extractor, generator, sample_count, dim=2048):
  features = extractor.predict_generator(generator, sample_count)
return features
```

```
datagen = image.ImageDataGenerator( rescale = 1./255 )

train_generator_scaled = datagen.flow_from_directory(
        TRAINING_DATA,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='sparse',
        shuffle=False)

conv_base = ResNet50(weights='imagenet',
include_top=False,
input_shape=(300, 300, 3),
pooling='avg')


training_features = extract_features(conv_base,
 train_generator,
TRAINING_SIZE )
np.save(FILE_TRAINING_FEATURES, training_features)
```

5.1 Distance distribution

At this point, we performed a quick analysis on the distribution of distances among the objects in our training-set (or rather among the features extracted from the objects). This was done so that we could better interpret the data that we will obtain in the following sections by applying the k-NN search algorithm.
First, we calculated the average distance between the features extracted from Resnet50 and Resnet50v2, both with the Euclidean distance and with the Cosine simiarity metrics.

	mean Euclidian dist.	mean Cosine sim.
Resnet50	3.015	0.994
Resnet50v2	31.467	0.3614

To complete the analysis, we also calculated the average distance between features divided by class, in this case using only the Euclidean distance for brevity. The results are shown below.

mean Euclidian dist.	0 - drawings	1 - engraving	2 - iconography	3 - painting	4 - sculpture
Resnet50	3.2472	4.3399	3.0041	2.5269	2.8741
Resnet50v2	31.199	31.7557	30.0471	31.078	33.727

Looking at both the tables, we can conclude that the features extracted with Resnet50 are much closer and similar than the features extracted with Resnet50v2. In addition, the distinct values by class are close to the mean value. This means that also in the search results we will get higher distance values for the second network with respect to the first one.

5.2 Resnet50

5.2.1 Sequential search after Features Extraction, using Resnet50

During this section we extracted the features from the dataset using the pretrained network Resnet50, adopting a features extraction technique and we computed a sequential scan *both with and without the distractor*, in order to compare the performance when we introduced some images as noise.

We obtained the following results:

Test without distractor

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
Resnet50	0.715	1.085	0.718	0.999	7784

Class	0 - drawings	1 - engraving	2 - iconography	3 - painting	4 - sculpture
Euclidean mAP	0.561	0.562	0.701	0.917	0.658
Cosine mAP	0.550	0.580	0.707	0.913	0.669

Test introducing distractor

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
Resnet50	0.704	1.083	0.707	0.999	32791

Class	0 - drawings	1 - engraving	2 - iconography	3 - painting	4 - sculpture
Euclidean mAP	0.549	0.554	0.690	0.914	0.639
Cosine mAP	0.536	0.572	0.697	0.909	0.648

From the previous tables we can observe a very small decrease in the mAP values introducing the distractor both using the euclidean distance and the cosine similarity.
Note that the last column of the tables (*#Items*) gives the number of objects visited by each query.

5.2.2 LSH Index after Features Extraction, using Resnet50

During this section we extracted the features from the dataset using the pretrained network Resnet50 adopting a fine tuning technique and we computed a kNN search setting w=4, obtaining the following results:

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
g=5 h=2	0,683	1.216	0,681	0.999	5701
g=6 h=2	0,674	1.227	0,681	0.999	4836
g=7 h=2	0,674	1.206	0,674	0.999	4872
g=8 h=2	**0,681**	**1.241**	**0,687**	**0.998**	**5304**
g=7 h=6	0.583	1.518	0.589	0.998	1647
g=4 h=5	0.556	1.491	0.630	0.998	922

We tested the LSH index also with different values of *w*, but we conclude that w=4 is the best values with Resnet50, because using lower values the buckets are sparse, while increasing *w* the mAP remains constant.

5.3 Resnet50v2

5.3.1 Sequential search after Features Extraction, using Resnet50v2

During this section we extracted the features from the dataset using the pretrained network Resnet50v2, adopting a features extraction technique and we computed a sequential scan *both with and without the distractor*, in order to compare the performance when we introduced some images as noise.
We obtained the following results:

Test without distractor

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
Resnet50v2	0.893	17.986	0.902	0.747	7784

Class	0 - drawings	1 - engraving	2 - iconography	3 - painting	4 - sculpture
Euclidean mAP	0.686	0.698	0.984	0.953	0.929
Cosine mAP	0.638	0.779	0.988	0.950	0.962

Test introducing distractor

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
Resnet50v2	0.888	17.977	0.897	0.747	32791

Class	0 - drawings	1 - engraving	2 - iconography	3 - painting	4 - sculpture
Euclidean mAP	0.682	0.696	0.984	0.947	0.918
Cosine mAP	0.636	0.772	0.988	0.946	0.953

From the previous tables we can observe a very small decrease in the mAP values introducing the distractor both using the euclidean distance and the cosine similarity.
Moreover it is interesting to notice that using Resnet50v2 to extract the features from the images, the average distance computed using the euclidean distance between objects is higher than the one computed using Resnet50 (about 17 times more), while the overall performance of the sequential scan search (measured by the mAP) is better, by about 18%.

5.3.2 LSH Index after Features Extraction, using Resnet50v2

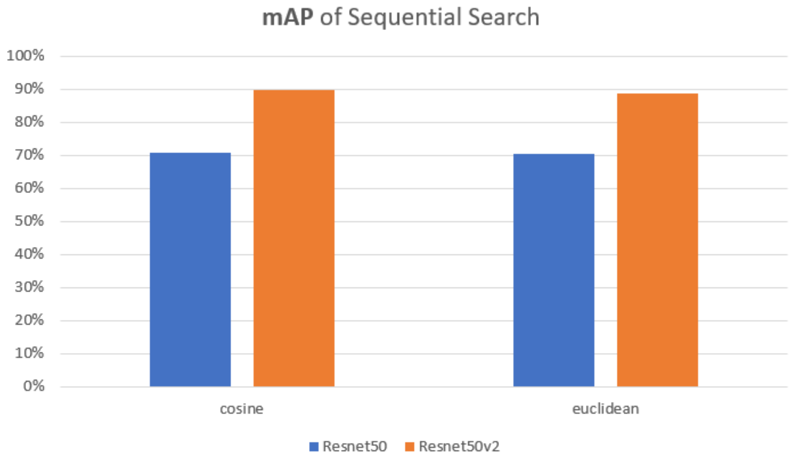
During this section we extracted the features from the dataset using the pretrained network Resnet50v2, adopting a fine tuning technique and we computed a kNN search, obtaining the following results:

k = 10	mAP euclidean	avg dist	mAP cosine	avg sim	#Items
g=3 h=2	**0.772**	**22.772**	**0.801**	**0.619**	**477**
g=3 h=3	0.631	27.532	0.663	0.490	79
g=3 h=4	0.357	29.709	0.404	0.409	25
g=4 h=2	**0.793**	**22.285**	**0.814**	**0.634**	**430**
g=4 h=3	0.608	27.865	0.651	0.474	71
g=4 h=4	0.414	29.658	0.446	0.411	27
g=5 h=2	**0.780**	**22.456**	**0.807**	**0.627**	**464**
g=5 h=3	0.691	26.175	0.728	0.527	121
g=5 h=4	0.486	28.903	0.509	0.432	23
g=6 h=2	**0.794**	**21.995**	**0.822**	**0.639**	**568**
g=6 h=3	0.677	26.748	0.705	0.513	88
g=6 h=4	0.524	27.908	0.545	0.457	23
g=7 h=2	**0.795**	**22.128**	**0.820**	**0.639**	**685**
g=7 h=3	0.584	28.075	0.629	0.470	61
g=7 h=4	0.304	29.794	0.323	0.407	22
g=8 h=2	**0.789**	**22.137**	**0.818**	**0.635**	**450**
g=8 h=3	0.566	27.591	0.615	0.483	74
g=8 h=4	0.380	29.613	0.405	0.414	27

We performed different tests varying the value of w; in the previous table the values shown are calculated with **w=8**, indeed from other tests we noted a small decrease on the value of mAP, but not so relevant, so we reported only the best ones.

5.4 Conclusion of the Features Extraction approach

Observing the results obtained from the experiments based on features extraction approach, we can say that the distances between the features of the objects extracted with Resnet50v2 are greater than that obtained using Resnet50.
Indeed passing an image as a query, the average euclidean distance between it and its k-nearest neighbor using Resnet50 is about 1.2 while using Resnet50v2 is about 25. Moreover the cosine similarity using Resnet50 is about 0.99 while using Resnet50v2 is about 0.5.



We can see from the figure above that the difference between the Euclidean distance and the Cosine similarity in the search is minimal. In addition, while having features with a larger scale, the Resnet50v2 network has better performace than the Resnet50.

In order to better compare the results obtained so far, we reported some other histograms (next page).
The data included are both from the sequential search and the LSH index search. For the latter, the hyper-parameters taken for the comparison are:

- Resnet50: $g = 8, h = 2, w = 4$
- Resnet50v2: $g = 7, h = 2, w = 8$

If we look at the sequential search performance, Resnet50v2 gets worse by 8% while Resnet50 only worsens by 2%. However, we can see that Resnet50v2 has a better mAP even with the index, with 82% compared to 69% of Resnet50. We also have to point out that the average number of objects visited per query by Resnet50v2 is 8 times lower than Resnet50, and this might be the reason why there is so much difference in the mAP from sequential to LSH of the first mentioned network.
In the next approach, fine tuning, we will see that this phenomenon will be buffered.

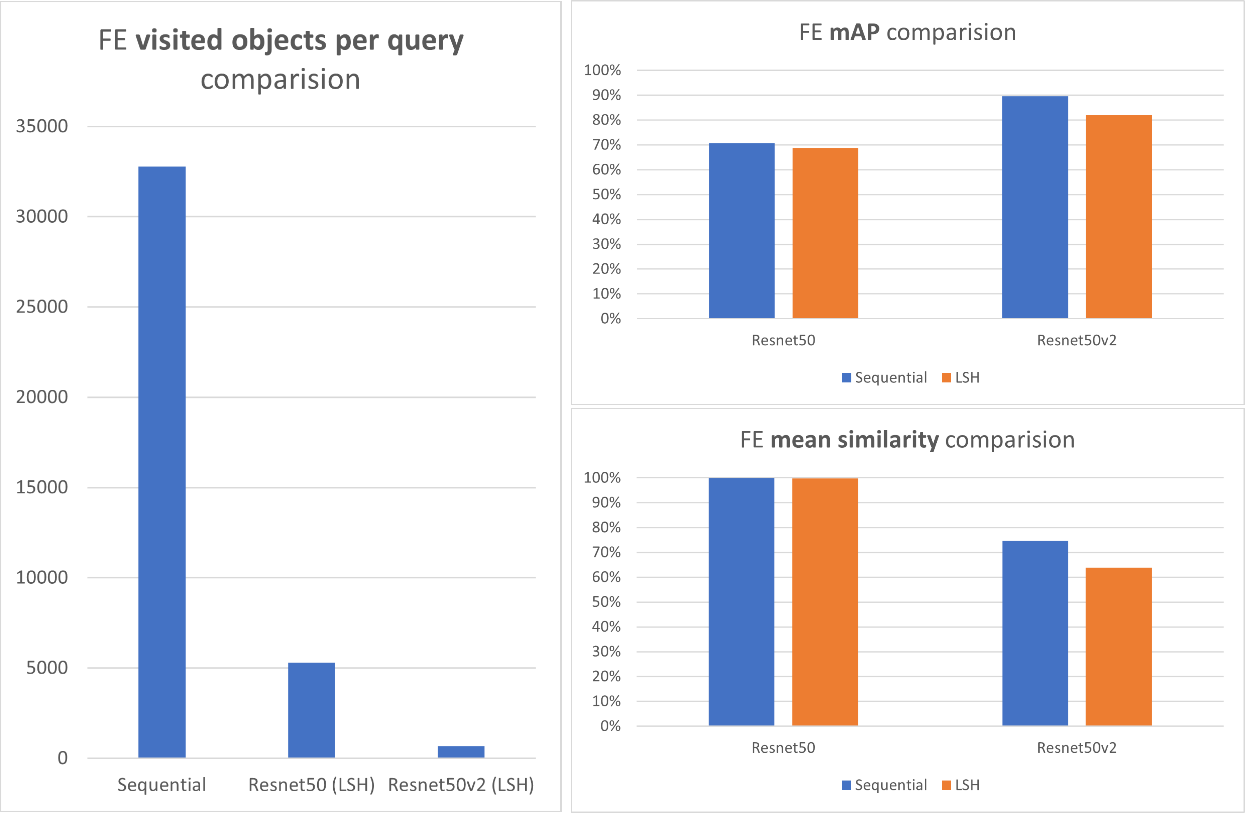


Figure 1: Data comparision for the Feature Extraction approach

6 Fine Tuning approach

During this phase we carried out several tests using different hyperparameters for the fully connected part added at the end both of the networks Resnet50 and Resnet50v2.
We compute this phase adopting a trial and error approach, trying different hyperparameter values for the fully connected neural network appended to the pretrained convolutional one and we compared the performance.
We stopped the training exploiting an *early stopping condition*, monitoring the loss function and setting the patience value equal to 10 and restoring the weights to 10 epochs before.
The following subparagraphs show only the most relevant experiments, we didn't report all of them in order to avoid some redundancies.

6.1 Resnet50

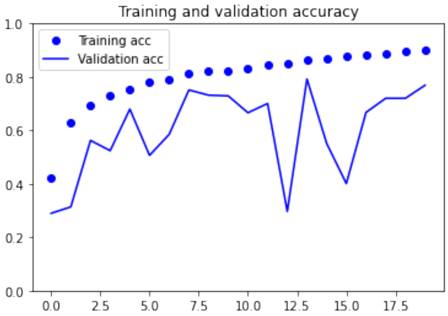
6.1.1 Model 1

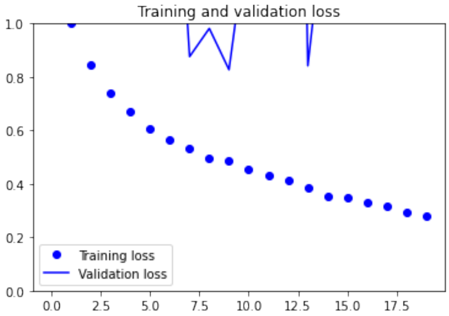
In this trial we trained the pretrained network Resnet50, we appended a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_1_conv
- 1 dense layer composed by 256 neurons
- Activation function: softmax
- Learning rate: 1e-3

These are the obtained results on the validation set we used to test the network:

```
    precision    recall  f1-score   support

         0.0       0.59      0.39      0.47       122
         1.0       0.63      0.77      0.70        84
         2.0       0.90      0.86      0.88       231
         3.0       0.89      0.78      0.83       228
         4.0       0.65      0.86      0.74       191

    accuracy                           0.76       856
   macro avg       0.73      0.73      0.72       856
weighted avg       0.77      0.76      0.76       856

loss_test : 0.7306886315345764
acc_test : 0.7616822719573975
```





6.1.2 Model 2

Looking at the accuacy plot of the first experiment, we can observe the presence of the overfitting phenomenon; in order to reduce this problem, we added some *Dropout Layers* and reduced the learning rate.
So in the second trial we trained the pretrained network Resnet50, we appended a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_1_conv
- Dropout layer, rate: 0.5
- 1 dense layer composed by 256 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 128 neurons
- Dropout layer, rate: 0.5
- Activation function: softmax
- Learning rate: 1e-4

The obtained results are the following:

```
    precision    recall  f1-score   support

         0.0       0.47      0.33      0.39       122
         1.0       0.70      0.56      0.62        84
         2.0       0.60      0.94      0.73       231
         3.0       0.98      0.22      0.36       228
         4.0       0.53      0.81      0.64       191

    accuracy                           0.59       856
   macro avg       0.66      0.57      0.55       856
weighted avg       0.68      0.59      0.55       856

loss_test : 0.9764283299446106
acc_test : 0.5934579372406006
```

The addition of the dropout did not seem to bring any improvement: on the contrary, the performance of the model is lower. So we tried a new test by removing these dropout levels and adding new FC levels.

6.1.3 Model 3

In the third trial, we trained the pretrained network Resnet50, adding a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_1_conv
- 1 dense layer composed by 512 neurons
- 1 dense layer composed by 256 neurons
- 1 dense layer composed by 128 neurons
- 1 dense layer composed by 64 neurons
- Activation function: softmax
- Learning rate: 1e-4

Again, these are the results we obtained from the trial:

```
    precision    recall  f1-score   support

         0.0       0.67      0.02      0.03       122
         1.0       0.69      0.39      0.50        84
         2.0       0.46      0.97      0.62       231
         3.0       0.82      0.82      0.82       228
         4.0       0.80      0.38      0.52       191

    accuracy                           0.61       856
   macro avg       0.69      0.52      0.50       856
weighted avg       0.68      0.61      0.56       856

loss_test : 0.9576621651649475
acc_test : 0.605140209197998
```

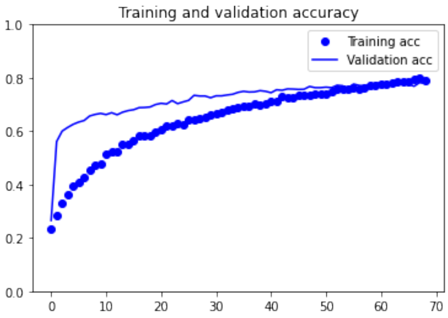
The network performance remained similar to the previous attempt.

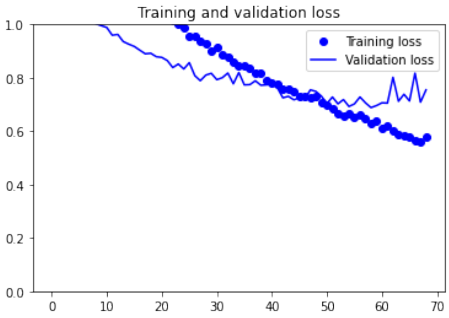
6.1.4 Model 4

In the fourth trial we trained the pretrained network Resnet50, in order to better generalize the dataset, so we appended a fully connected part defined by the following hyperparameters:

- Unfreezed layers from conv5_block1_1_conv
- Dropout layer, rate: 0.5
- 1 dense layer composed by 256 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 128 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 64 neurons
- Activation function: softmax
- Learning rate: 1e-6

```
precision    recall  f1-score   support

         0.0       0.71      0.34      0.46       122
         1.0       0.64      0.68      0.66        84
         2.0       0.80      0.95      0.87       231
         3.0       0.96      0.87      0.91       228
         4.0       0.76      0.92      0.83       191

    accuracy                           0.81       856
   macro avg       0.78      0.75      0.75       856
weighted avg       0.81      0.81      0.79       856

loss_test : 0.6162440776824951
acc_test : 0.8072429895401001
```



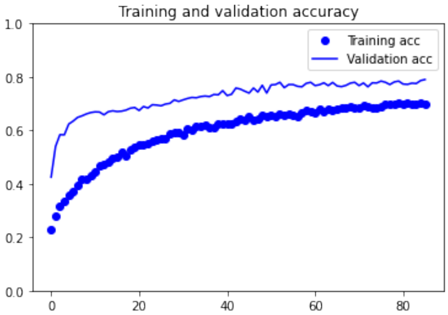


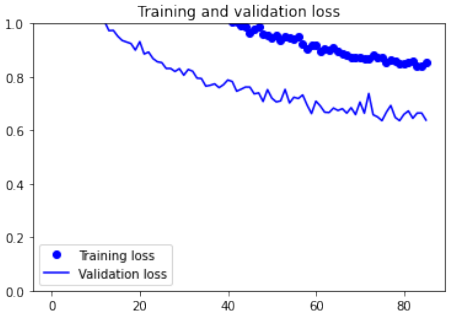
We can see from the results that this is the best model so far, with an overall accuracy of 81%: in addition to the adjustment of the dropout and dense levels, this is due to the further lowering of the learning rate. Also the graphs show a decisive improvement of the overfitting.

6.1.5 Model 5

In the fifth trial, we trained the Resnet50 convolutional base, where we appended the same fully connected part of the previous experiment applying a data augmentation technique, in order to better generalize the dataset.

```
precision    recall  f1-score   support

         0.0       0.62      0.28      0.38       122
         1.0       0.53      0.74      0.62        84
         2.0       0.85      0.87      0.86       231
         3.0       0.91      0.85      0.88       228
         4.0       0.71      0.87      0.79       191

    accuracy                           0.77       856
   macro avg       0.72      0.72      0.70       856
weighted avg       0.77      0.77      0.76       856

loss_test : 0.6571111083030701
acc_test : 0.7675233483314514
```





The performance did not improve, in fact a slight underfitting appeared.

6.1.6 Sequential search after Fine Tuning, using Resnet50

During this section we extracted the features from the dataset using the pretrained network Resnet50, adopting a fine tuning technique and we computed a sequential scan, obtaining the following results:

	Accuracy	**Loss**	**mAP** euclidean	avg dist	**mAP** cosine	avg sim	Items
Model 1	0.7617	0.7307	0.761	0.493	0.765	0.987	32791
Model 2	0.5935	0.9764	0.749	0.885	0.755	0.975	32791
Model 3	0.6051	0.9577	0.742	1.220	0.749	0.963	32791
Model 4	**0.8072**	**0.6162**	**0.767**	**4.413**	**0.780**	**0.943**	**32791**
Model 5	0.7675	0.6571	0.756	3.679	0.768	0.953	32791

6.1.7 LSH Index after Fine Tuning, using Resnet50

During this section we extracted the features from the dataset using the pretrained network Resnet50, adopting a fine tuning technique and we computed a kNN search, obtaining the following results (w=4):

	mAP euclidean	avg dist	**mAP** cosine	avg sim	Items
Model 1, g=5 h=2	0.765	0.510	0.767	0.986	7864
Model 2, g=5 h=2	0.745	0.965	0.749	0.972	6697
Model 2, g=6 h=2	0.743	0.961	0.748	0.965	7475
Model 3, g=4 h=2	0.721	1.375	0.725	0.952	4507
Model 3, g=6 h=2	0.719	1.344	0.725	0.956	5323
Model 4, g=5 h=2	**0.711**	**4.908**	**0.733**	**0.931**	**1452**
Model 5, g=5 h=2	0.707	4.011	0.728	0.944	2397

Comparing the performance obtained using different models, we can conclude that computing a sequential scan or using the index, the performance change basically only in terms of items retrieved. Moreover we reported only the best results of the indexes: selected a model and varying the hyper-parameters g and h the indexes that retrieve less objects.

Finally we chose the Model 4 with the following hyperparameters g=5, h=2, w=4 because it visits less objects per query with respect to the others, while maintaining a good value of mAP.

Below there are some histograms summarizing the results obtained with the Resnet50 network. The index data are associated to the hyper-parameters that gave the best performance in both the Feature Extraction ($g = 8, h = 2, w = 4$) and Fine Tuning ($g = 5, h = 2, w = 4$) approaches. Moreover, for the latter approach we refer to the Model 4 results.

After the re-training of the last blocks of the network on our training set, we can see that the mAP has increased to 78% in the sequential search, and to 73% in the indexed search. Not only do we have an increase in the mAP, but there is also a large decrease in the average number of objects visited per query, from 5304 to 1452.

Both of these considerations lead us to conclude that this approach achieved the best index search with the Resnet50 network.

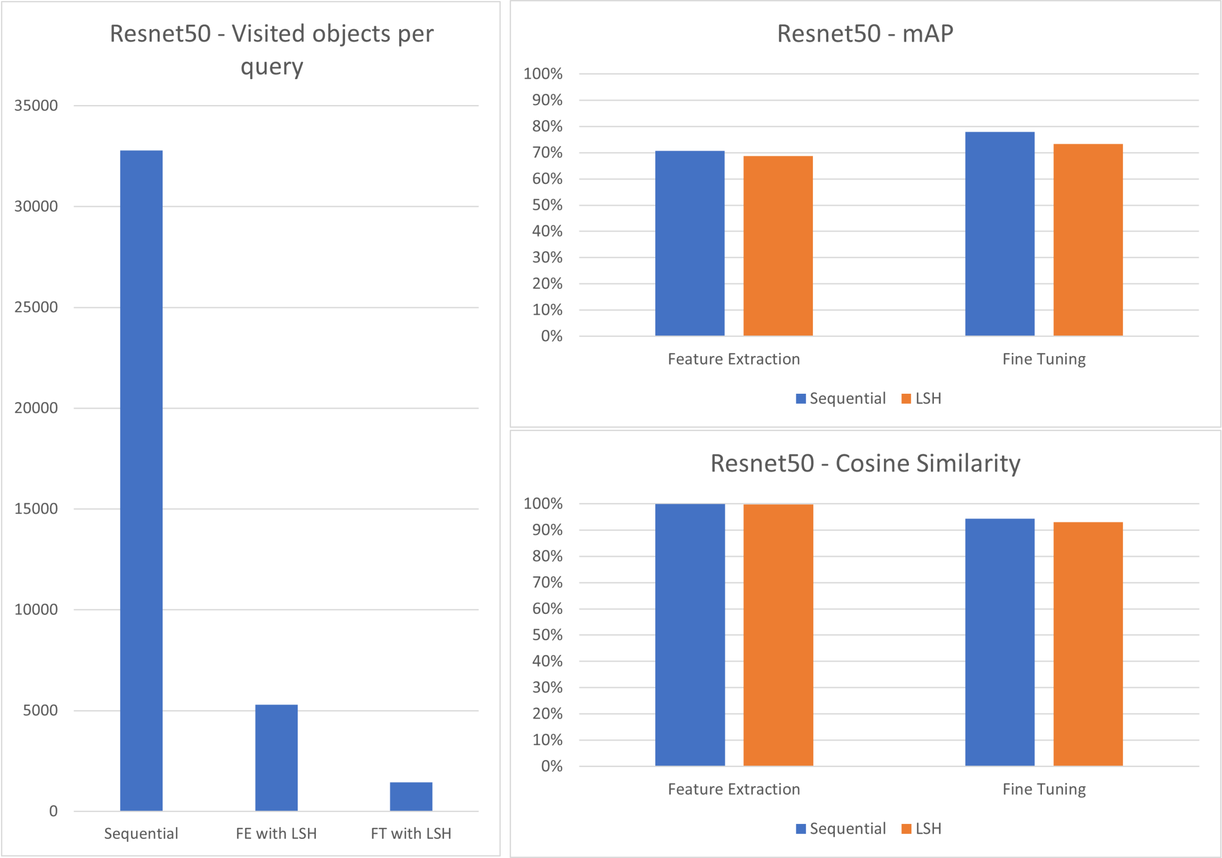


Figure 2: Final data comparision for the Resnet50 network

6.2 Resnet50v2

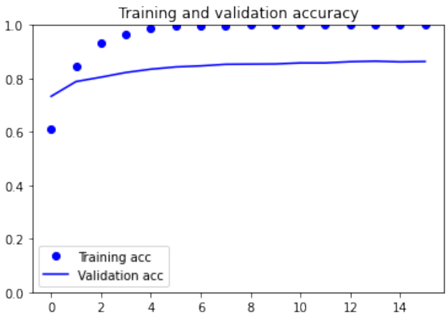
For this convolutional base, we performed the tests by resuming the fully connected blocks used in the previous step. We just report the obtained results, since all the hyperparameters can be consulted in the previous section.

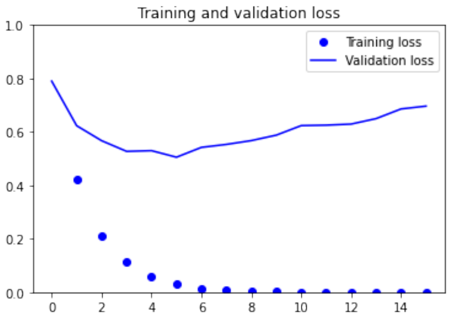
6.2.1 Model 1

In this first trial with Resnet50v2, we appended a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_preact_bn
- 1 dense layer composed by 256 neurons
- Activation function: softmax
- Learning rate: 1e-6

```
precision    recall  f1-score   support

         0.0       0.68      0.65      0.66       122
         1.0       0.67      0.58      0.62        84
         2.0       0.96      0.97      0.97       231
         3.0       0.92      0.96      0.94       228
         4.0       0.92      0.95      0.94       191

    accuracy                           0.88       856
   macro avg       0.83      0.82      0.83       856
weighted avg       0.87      0.88      0.88       856

loss_test : 0.3915505111217499
acc_test : 0.8785046935081482
```





6.2.2 Model 2

In this second trial with Resnet50v2, we appended a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_preact_bn
- Dropout layer, rate: 0.5
- 1 dense layer composed by 256 neurons

- Dropout layer, rate: 0.5
- 1 dense layer composed by 128 neurons
- Dropout layer, rate: 0.5
- Activation function: softmax
- Learning rate: 1e-5

```
precision    recall  f1-score   support

         0.0       0.70      0.58      0.63       122
         1.0       0.76      0.60      0.67        84
         2.0       0.96      0.99      0.97       231
         3.0       0.90      0.96      0.93       228
         4.0       0.90      0.98      0.94       191

    accuracy                           0.88       856
   macro avg       0.84      0.82      0.83       856
weighted avg       0.87      0.88      0.88       856

loss_test : 0.35103410482406616
acc_test : 0.8820093274116516
```

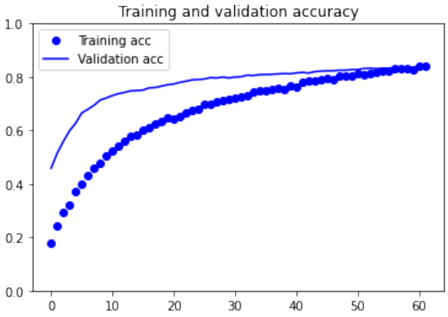
6.2.3 Model 3

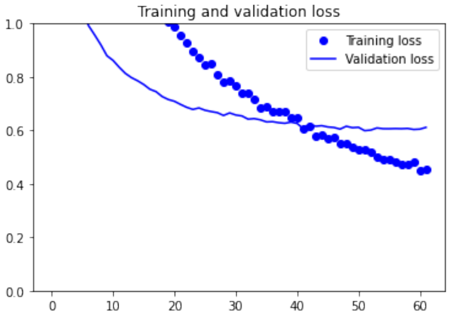
In the fourth trial we trained the pretrained neural network Resnet50v2, we appended a fully connected part defined by the following hyperparameter:

- Unfreezed layers from conv5_block1_preact_bn
- Dropout layer, rate: 0.5
- 1 dense layer composed by 256 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 128 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 64 neurons
- Activation function: softmax
- Learning rate: 1e-6

```
precision    recall  f1-score   support

         0.0       0.90      0.21      0.34       122
         1.0       0.61      0.83      0.70        84
         2.0       0.97      0.98      0.98       231
         3.0       0.83      0.96      0.89       228
         4.0       0.87      0.97      0.92       191
```

```
    accuracy                           0.85       856
   macro avg       0.83      0.79      0.77       856
weighted avg       0.86      0.85      0.82       856

loss_test : 0.48188671469688416
acc_test : 0.8492990732192993
```

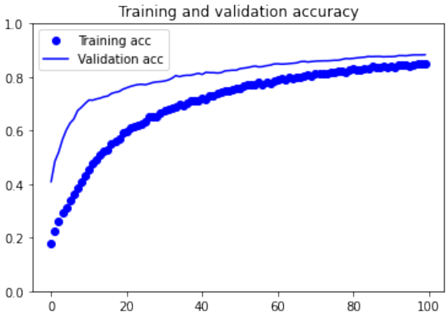


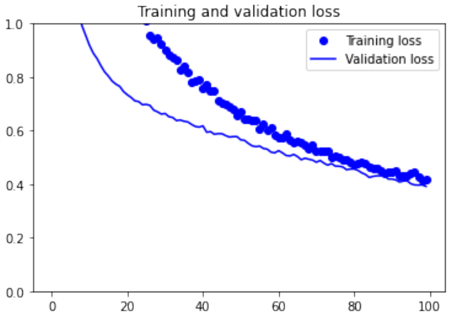


6.2.4 Model 4

In the last trial, we trained the Resnet50 convolutional base, where we appended the same fully connected part of the previous experiment applying a data augmentation technique, in order to better generalize the dataset.

```
precision    recall  f1-score   support

         0.0       0.78      0.48      0.60       122
         1.0       0.70      0.73      0.71        84
         2.0       0.96      0.99      0.98       231
         3.0       0.90      0.95      0.93       228
         4.0       0.88      0.99      0.93       191

    accuracy                           0.88       856
   macro avg       0.84      0.83      0.83       856
weighted avg       0.88      0.88      0.87       856

loss_test : 0.37721073627471924
acc_test : 0.8808411359786987
```





6.2.5 Sequential search after Fine Tuning Resnet50v2

During this section we extracted the features from the dataset using the pretrained network Resnet50v2 after having fine tuned it and we computed a sequential scan, obtaining the following results:

	Accuracy	**Loss**	**mAP** euclidean	avg dist	**mAP** cosine	avg sim	Items
Model 4	0.8493	0.4819	0.875	16.684	0.886	0.661	32791
Model 5	**0.8808**	**0.3772**	**0.893**	**8.997**	**0.908**	**0.729**	**32791**

6.2.6 LSH Index after Fine Tuning Resnet50v2

During this section we extracted the features from the dataset using the pretrained network Resnet50v2, adopting a fine tuning technique and we computed a kNN search, obtaining the results in the table below.

We finally chose the Model 5 with the hyperparameters $g = 7, h = 2, w = 6$ because it has the higher value of mAP and it still visits a low number of objects.

	Hyperparameters	**mAP** euclidean	avg dist	**mAP** cosine	avg sim	Items
Model 4	g=4 h=2 w=5	0.725	22.231	0.793	0.478	266
Model 4	g=4 h=2 w=6	0.764	22.038	0.824	0.516	407
Model 4	g=5 h=2 w=5	0.738	21.862	0.800	0.489	284
Model 4	g=5 h=2 w=6	0.760	21.431	0.822	0.504	423
Model 4	g=6 h=2 w=5	0.710	21.759	0.801	0.486	254
Model 4	g=6 h=2 w=6	0.752	20.978	0.819	0.511	369
Model 4	g=7 h=2 w=5	0.721	21.851	0.799	0.488	345
Model 4	**g=7 h=2 w=6**	**0.761**	**20.861**	**0.830**	**0.519**	**532**
Model 5	g=4 h=2 w=5	0.790	11.181	0.840	0.611	724
Model 5	g=4 h=2 w=6	0.793	11.661	0.840	0.596	727
Model 5	g=5 h=2 w=5	0.808	11.147	0.848	0.607	1061
Model 5	g=5 h=2 w=6	0.823	10.958	0.865	0.625	1534
Model 5	g=6 h=2 w=5	0.797	10.887	0.847	0.616	704
Model 5	g=6 h=2 w=6	0.777	10.958	0.838	0.616	1012
Model 5	g=7 h=2 w=5	0.814	11.293	0.863	0.606	1007
Model 5	**g=7 h=2 w=6**	**0.829**	**11.002**	**0.861**	**0.623**	**1440**

Below there are some histograms summarizing the results obtained with the Resnet50v2 network. The index data are associated to the hyper-parameters that gave the best performance in both the Feature Extraction ($g = 7, h = 2, w = 8$) and Fine Tuning ($g = 7, h = 2, w = 6$) approaches. Moreover, for the latter approach we refer to the Model 5 results.

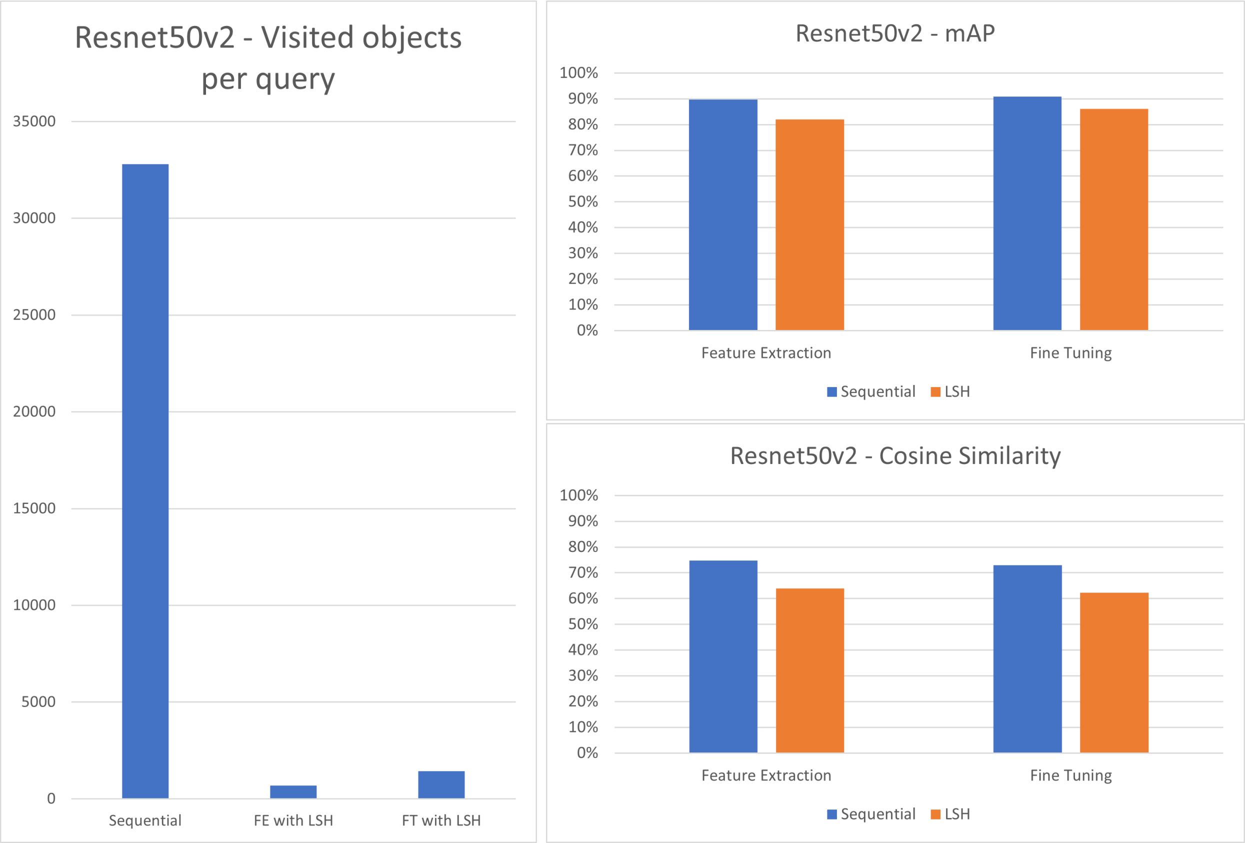


Figure 3: Final data comparision for the Resnet50v2 network

In this case, the partial re-training of the network brought the mAP to 86% using the index, whereas before it was 82%. This value of mAP is the highest obtained so far with LSH index search. The number of visited objects is raised compared to the feature extraction data, from 685 to 1440, still remaining lower than the Resnet50 data. This is given by the fact that in the first approach we selected $w = 8$ in the index creation, while in the second approach $w = 6$ was enough to obtain a high average similarity and a high mAP.

6.3 Conclusion of Fine Tuning approach

In order to better compare the obtained results, we reported some other histograms.
The data included are both from the sequential search and the LSH index search. For the latter, the hyper-parameters taken for the comparison are:

- Resnet50 (model 4): $g = 5, h = 2, w = 4$
- Resnet50v2 (model 5): $g = 7, h = 2, w = 6$

We see that, for the same average objects visited per query, the model built on Resnet50v2 offers much higher performance, with a mAp of 86%, versus Resnet50's 73%.

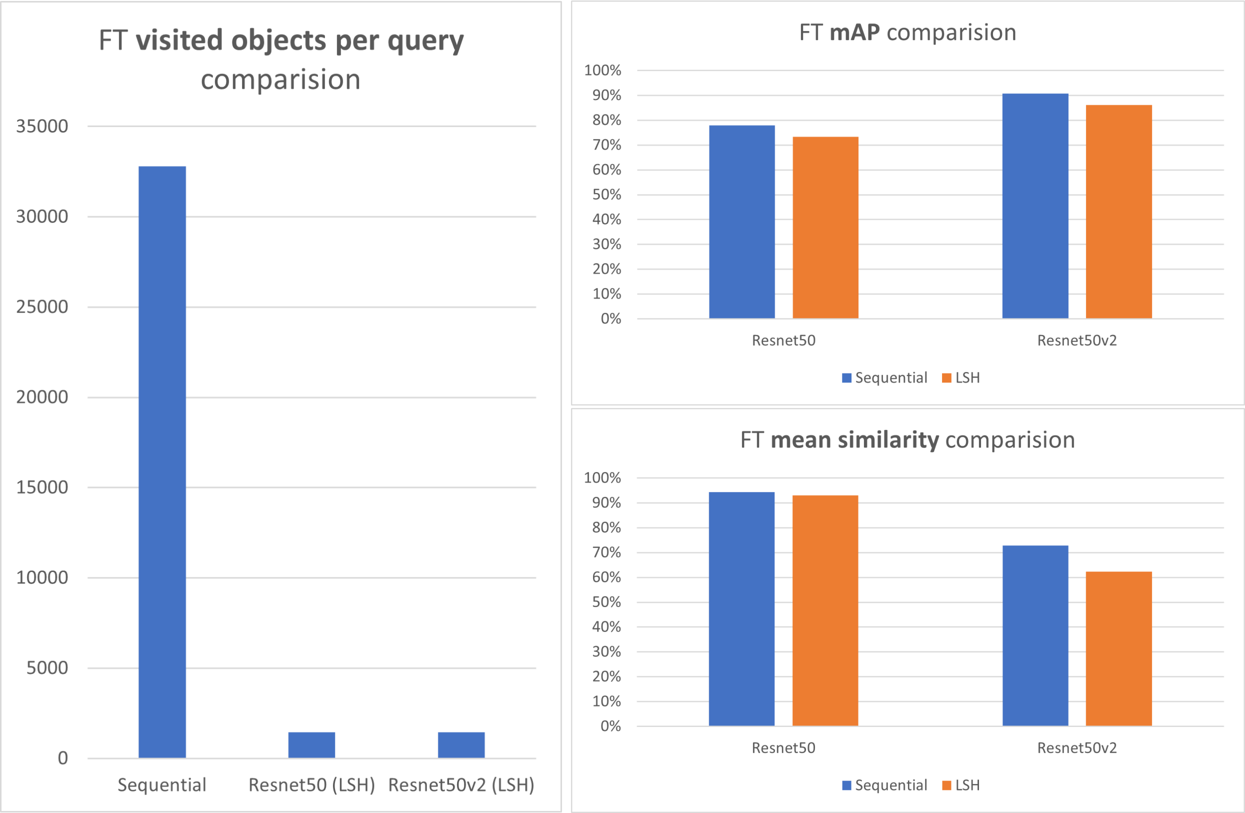


Figure 4: Data comparision for the Fine Tuning approach

7 LSH index considerations

During this part of the project we analyzed the space of the hyperparameter values after running different tests in the previous sections.
All the following figures show some plots where the data are computed using the cosine similarity, but similar results are obtained using the euclidean distance, we didn't report them to avoid redundancy.

The first aspect that we noted is that we obtained the highest values of the mAP setting a low value of the number of hyperplanes h (h=2) fixed the length of the segment w and the number of g-functions g: indeed increasing the value of h we can observe lower curves.
Moreover we can observe that increasing the value of w, also the value of mAP increases.

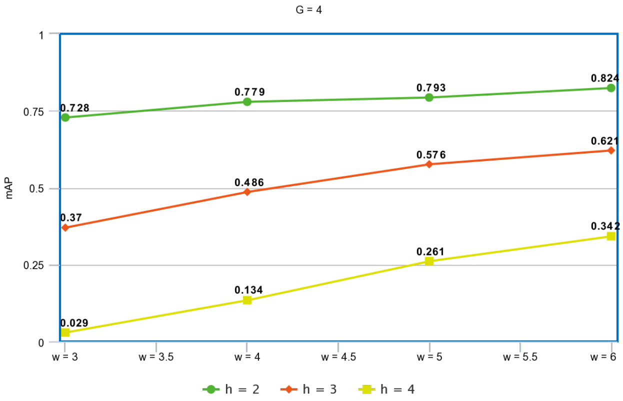
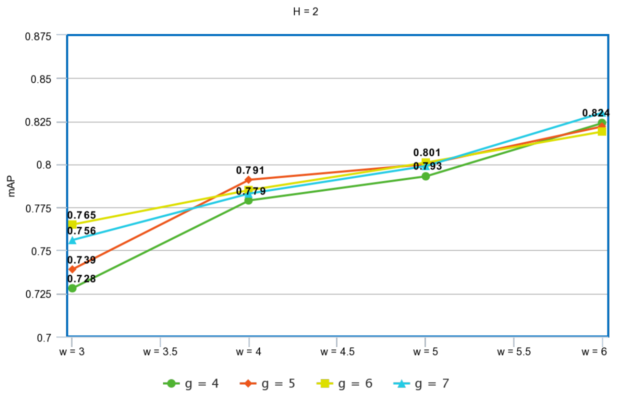


Figure 5: Resnet50v2 fine tuned model 5 - **mAP** trend with fixed $g = 4$

The following two figures show the variation of the mAP value and the number of visited objects when we increase the number of g-functions.



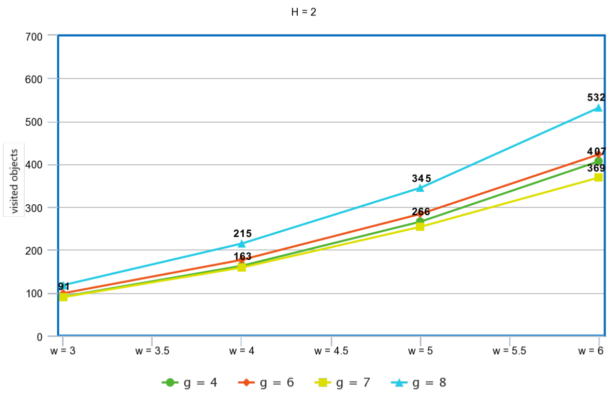


Figure 6: Resnet50v2 fine tuned model 5 - **mAP** trend with fixed $h = 2$

Figure 7: Resnet50v2 fine tuned model 5 - **visited objects** trend with fixed $h = 2$

The following figure shows the decrease of the mAP value when we increase the number of hyperplanes for each g-function. We can observe this trend, not only for the model in figure but also for all the other models we worked with.

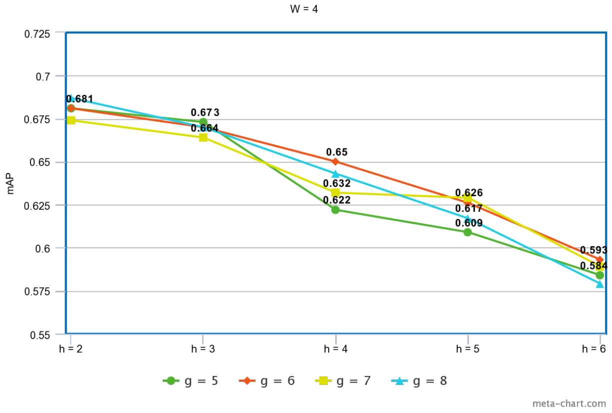


Figure 8: Resnet50 feature extraction model - **mAP** trend with fixed $w = 4$