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:

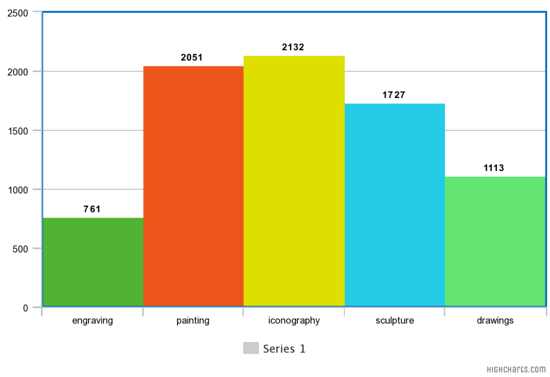
- Drawings
- Engraving
- Iconography
- Painting
- Sculpture

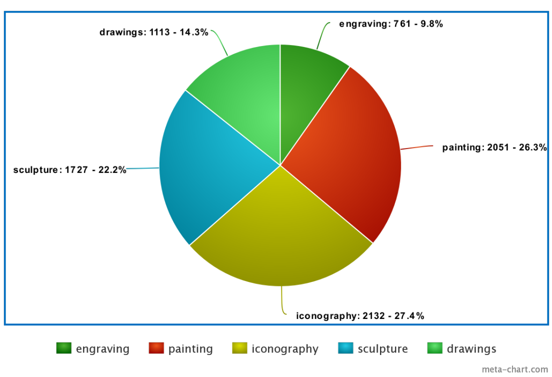
2.1 Data cleaning

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.

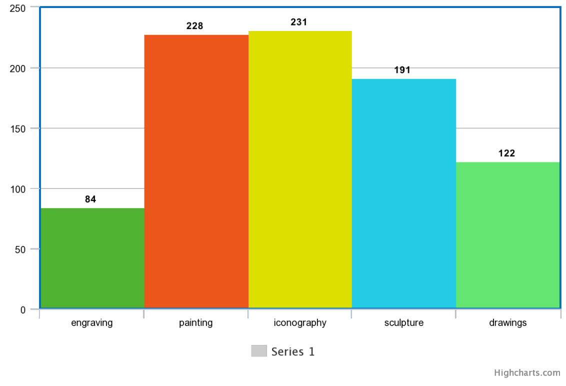
```
from PIL import Image
import glob

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)
```

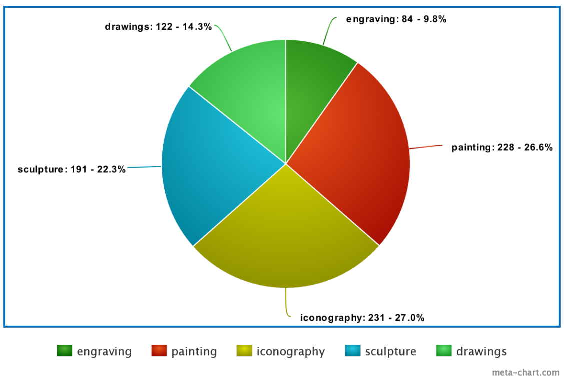
The following figures show the distribution of the classes of the trainig set, after the execution of the snippet of code shown above :





The following figures show the distribution of the classes of the validationset, after the execution of the snippet of code shown above :





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 distance:

- Euclidian distance
- Cosine similarity

We performed the evaluation of the sequential search function, based on the validation set, so that we could have some starting data to compare with respect to 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:

- Trainingset and Distractor are combined as data on which to perform the search algorithm
- Validationset images are used as queries to perform the search

The metric used to measure the goodness of the research is **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

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

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 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 Resnet50

5.1.1 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, obtaining the following results:

k = 10	mAP euclidian	avg dist	mAP cosine	avg sim	#Items
Resnet50	0.306	1.083	0.309	0.999	32791

5.1.2 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 euclidian	avg dist	mAP cosine	avg sim	#Items
g=5 h=2	**0.309**	**1.216**	**0.318**	**0.999**	**5701**
g=6 h=2	0.303	1.227	0.308	0.999	4836
g=7 h=2	0.297	1.206	0.303	0.999	4872
g=7 h=6	0.281	1.518	0.284	0.998	1647
g=3 h=4	0.294	1.376	0.293	0.998	2686

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.

5.2 Resnet50v2

5.2.1 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, obtaining the following results:

k = 10	mAP euclidian	avg dist	mAP cosine	avg sim	#Items
Resnet50v2	0.351	17.977	0.343	0.747	32791

5.2.2 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 euclidian	avg dist	mAP cosine	avg sim	#Items
g=3 h=2	0.369	22.772	0.375	0.619	477
g=3 h=3	**0.415**	**27.532**	**0.412**	**0.490**	**79**
g=3 h=4	0.334	29.709	0.349	0.409	25
g=4 h=2	0.360	22.285	0.361	0.634	430
g=4 h=3	0.353	27.865	0.354	0.474	71
g=4 h=4	0.287	29.658	0.297	0.411	27
g=5 h=2	0.311	22.456	0.309	0.627	464
g=5 h=3	0.368	26.175	0.373	0.527	121
g=5 h=4	0.263	28.903	0.278	0.432	23
g=6 h=2	0.357	21.995	0.367	0.639	568
g=6 h=3	0.350	26.748	0.347	0.513	88
g=6 h=4	0.282	27.908	0.289	0.457	23
g=7 h=2	0.357	22.128	0.362	0.639	685
g=7 h=3	**0.414**	**28.075**	**0.430**	**0.470**	**61**
g=7 h=4	0.314	29.794	0.331	0.407	22

These values are calculated with w=8, from other tests we noted a small decrease on the value, but not so relevant.

5.3 Conclusion of 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 Resnet50 are greater than that obtained using Resnet50v2.
Indeed passing an image as a query, the average euclidian 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.

It is interesting to note that given a certain value of g and varying the value of h, the best results are given for h=3, because for greater value the mAP always decrease, while for smaller ones (h=2) in some cases the mAP decrease, in others increase, but we also observe a very significant increase in the number of the items to sort.

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

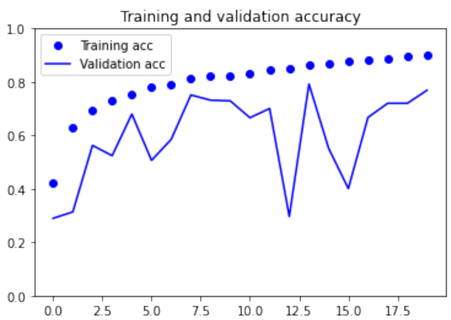
6.1.1 Model 1

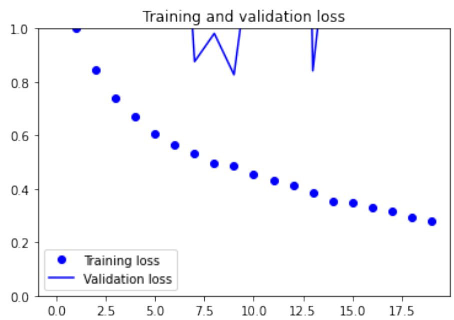
In this trial we trained the pretrained network Resnet50, we appended a fully connected part defined by the following hyperparameter:

- 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

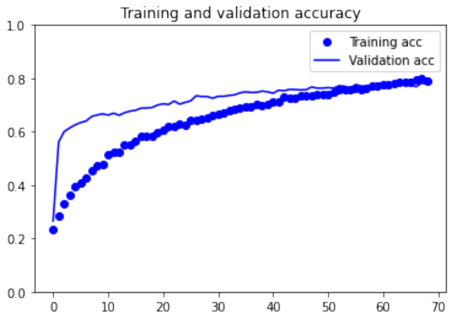
In the fourth trial we trained the pretrained network Resnet50, in order to better generalize the dataset, so we appended a fully connected part defined by the following hyperparameters:

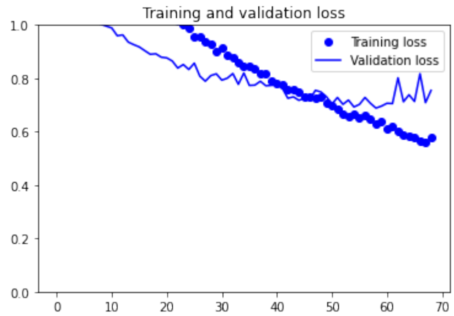
- Unfreezed layers from conv5_block1_1_conv
- Dropout layer, rate: 0.5
- 1 dense layer composed by 256 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 128 neurons
- Dropout layer, rate: 0.5
- 1 dense layer composed by 64 neurons
- Activation function: softmax
- Learning rate: 1e-6

```
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





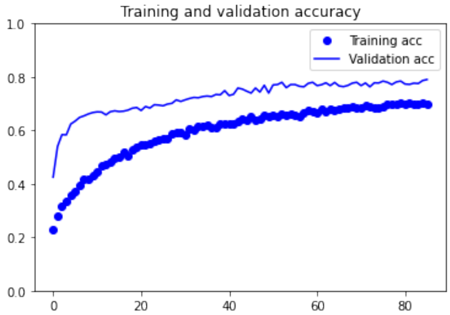
We can see from the results that this is the best model so far, with an overall accuracy of 81%: in addition to the adjustment of the dropout and dense levels, this is due to the further lowering of the learning rate. Also the graphs show a decisive improvement of the overfitting.

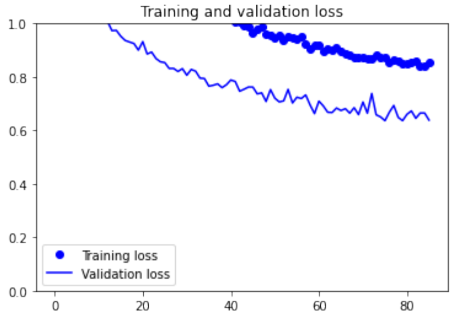
6.1.5 Model 5

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** euclidian	avg dist	**mAP** cosine	avg sim	Items
Model 1	0.7617	0.7307	0.312	0.493	0.309	0.987	32791
Model 2	0.5935	0.9764	0.329	0.885	0.328	0.975	32791
Model 3	0.6051	0.9577	0.324	1.220	0.323	0.963	32791
Model 4	0.8072	0.6162	0.326	4.413	0.326	0.943	32791
Model 5	0.7675	0.6571	0.313	3.679	0.320	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 euclidian	avg dist	**mAP** cosine	avg sim	Items
Model 1, g=5 h=3	0.314	0.574	0.318	0.982	4377
Model 2, g=4 h=2	0.328	1.048	0.330	0.967	4532
Model 3, g=4 h=2	0.326	1.375	0.333	0.952	4507
Model 4, g=6 h=3	0.316	5.647	0.328	0.912	485
Model 5, g=4 h=3	0.311	4.474	0.318	0.931	570

Comparing the performance obtained using different models, we can conclude that computing a sequential scan or using the index, the performance change 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.

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

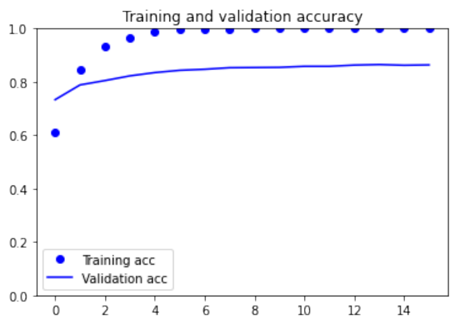
In this first trial with Resnet50v2, we appended a fully connected part defined by the following hyperparameter:

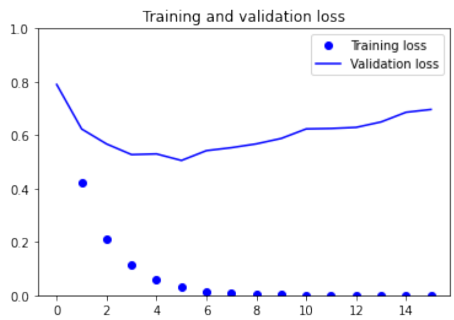
- Unfreezed layers from conv5_block1_preact_bn

- 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

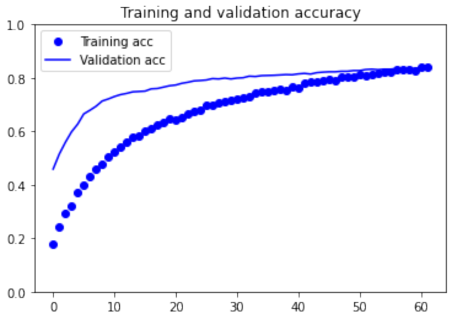
```
precision    recall  f1-score   support

         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
```

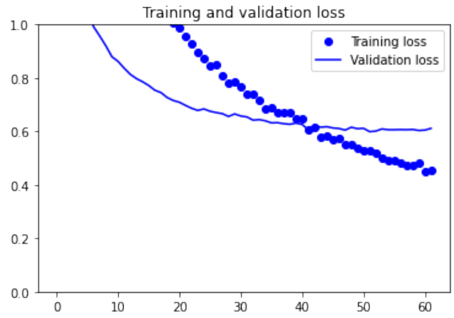
6.2.3 Model 3

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
```

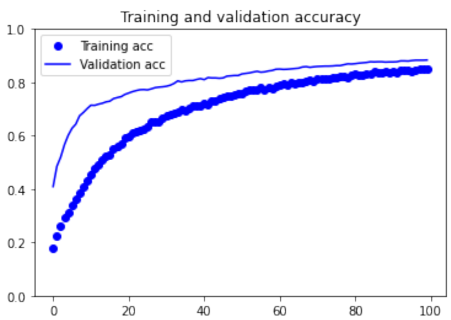


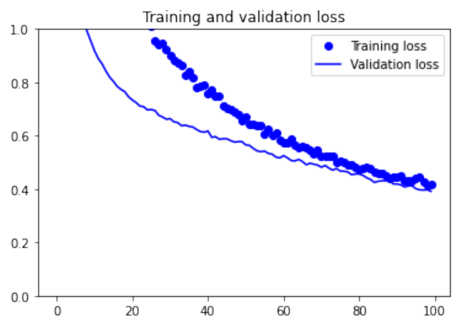


6.2.4 Model 4

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** euclidian	avg dist	**mAP** cosine	avg sim	Items
Model 3	0.8493	0.4819	0.337	16.684	0.335	0.661	32791
Model 4	0.8808	0.3772	0.352	8.997	0.345	0.729	32791

6.2.6 LSH Index after Fine Tuning Resnet50v2

6.3 Conclusion of Fine Tuning approach