MULTIMEDIA INFORMATION AND COMPUTER VISION PROJECT

Academic Year 2020/21

Barsellotti Luca, Bongiovanni Marco, Gholami Sina and Paolini Emilio

INDEX

INTRODUCTION

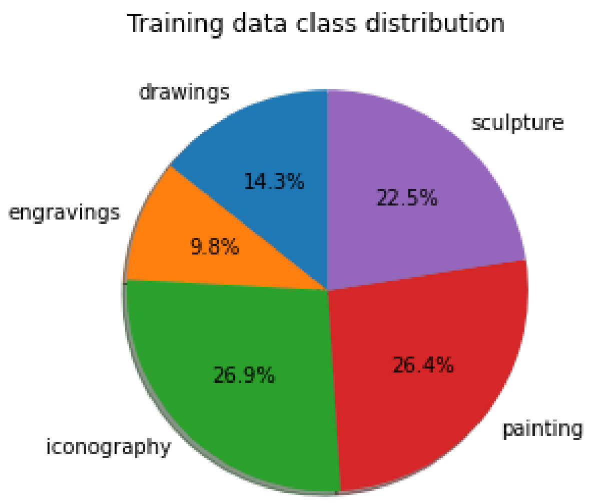
The aim of this project is to build a Web Search Engine based on art images, which works efficiently thanks to a Vantage Point Tree Index approach. The list of images that can be retrieved are contained in the "Art Dataset", a dataset split into five classes ("Drawings", "Engraving", "Iconography", "Paintings" and "Sculpture"), and in a "Distractor Dataset", based on the Mirflickr25k dataset. To compare the images, evaluate their similarity and build the index, we use a Convolutional Neural Network. In particular, we use two models: one is based on a Pre-Trained Network ("InceptionV3") with weights computed on "Imagenet" and the other one is a fine-tuned version of the same Pre-Trained Network through the "Art Dataset".

Dataset Analysis

TRAINING SET

First of all, we have analyzed the datasets. Both "Art Dataset" and "Distractor Dataset" contained some corrupted images and so we have removed them. At the end of the removing process, the training dataset contained in the "Art Dataset" was composed by

Total:	7721
Drawings:	1107
Engraving:	757
Iconography:	2077
Painting:	2042
Sculpture:	1738



TEST SET

The test set contained into the “Art Dataset” is composed by

Total: 856

Drawings: 122

Engraving: 84

Iconography: 231

Painting: 228

Sculpture: 191

The distribution of the classes is the same of the “Training Set”.

VANTAGE POINT TREE IMPLEMENTATION

INTRODUCTION

The Vantage Point Tree is an Exact Similarity Searching Method based on ball regions that recursively divide the given datasets. In particular, at each step a vantage point *p* (also called pivot), belonging to the actual dataset *X*, is chosen. After that, the median m of all the distances from *p* of the points belonging to X is computed

$$m = median(d(x, p)) \quad \forall x \in X$$

and the dataset is split into two subsets according to:

$$S_1 = \{x \in X - \{p\} | d(x, p) \leq m\}$$

$$S_2 = \{x \in X - \{p\} | d(x, p) \geq m\}$$

At the end, the objects are stored into the leaves. The resulting Vantage Point Tree is a balanced binary tree. This approach requires a time cost approximately of $O(n \log \log n)$ during the building phase and $O(\log \log n)$ in a 1-NN search.

BUILD

The Vantage Point Tree has been built using the approach described in the introduction, using as stopping condition the maximum bucket size that can be contained by a leaf.

```python
def build(node, feature_subset, leaf=False):
    if leaf is True:
        node.set_leaf(feature_subset)
    else:
        pivot = random.choice(feature_subset)
        distances = list()
        for feature_schema in feature_subset:
            distances.append(feature_schema.distance_from(pivot))
        distances = np.array(distances)
        median = np.median(distances)
        subset1 = list()
        subset2 = list()
        for feature_schema in feature_subset:
            if feature_schema.distance_from(pivot) <= median:
                subset1.append(feature_schema)
            else:
                subset2.append(feature_schema)
        node.set_median(median)
        node.set_pivot(pivot)
        node.add_left(Node())
        node.add_right(Node())
        nodes += 2
        if len(subset1) <= bucket_size:
            build(node.get_left(), subset1, leaf=True)
        else:
            build(node.get_left(), subset1)
        if len(subset2) <= bucket_size:
            build(node.get_right(), subset2, leaf=True)
        else:
            build(node.get_right(), subset2)
```

RANGE SEARCH

During the Range Search, at each step the distance between the query object and the pivot is evaluated to decide if it is necessary to access the children nodes. The condition to access the left node is

$$d(q,p) - r \leq m$$

instead, the condition to access the right node is

$$d(q,p) + range \leq m$$

If the node considered in the step is a leaf, then all the distances from the objects contained in its bucket are evaluated to decide if they have to be added to the result.

```python
def range_search(node, query, range):
    if node.is_leaf() == True:
        for object in node.get_subset():
            if query.distance_from(object) <= range:
                range_search_return_list.append(object)
        return
    if query.distance_from(node.get_pivot()) <= range:
        range_search_return_list.append(node.get_pivot())
    if query.distance_from(node.get_pivot()) - range <= node.get_median():
        recursive_range_search(node.get_left(), query, range)
    if query.distance_from(node.get_pivot()) + range >= node.get_median():
        recursive_range_search(node.get_right(), query, range)
    return
```

K-NEAREST NEIGHBORS SEARCH

In the k-NN Search a Priority Queue is used to maintain the retrieved objects ordered by the distance. This Priority Queue is initialized with the first objects encountered while traversing the Vantage Point Tree. At each step, if the node is an internal node then the distance between the query and its pivot is computed. If this distance is lower than d_{MAX}, the current highest distance in the Priority Queue, the pivot is added to the queue. Hence, if

$$d(q,p) - d_{MAX} \leq m$$

then the k-NN Search is called on the left node and if

$$d(q,p) + d_{MAX} \geq m$$

then the k-NN Search is called on the right node. Instead, if the node is a leaf, the distances from all the objects are computed to evaluate if they have to be added in the Priority Queue.

```
def knn(node, query, k):
    if node.is_leaf() == True:
        for object in node.get_subset():
            distance = query.distance_from(object)
            if not nn.is_full():
                nn.push(priority=distance, item=object)
                d_max = distance if d_max < distance else d_max
            elif abs(distance) < abs(d_max):
                nn.pop()
                nn.push(priority=distance, item=object)
                tmp = nn.data[0][0]
                d_max = abs(tmp)
        return
    distance = query.distance_from(node.get_pivot())
    if not nn.is_full():
        nn.push(priority=distance, item=node.get_pivot())
        d_max = distance if d_max < distance else d_max
    elif abs(distance) < abs(d_max):
        nn.pop()
        nn.push(priority=distance, item=node.get_pivot())
        tmp = nn.data[0][0]
        d_max = abs(tmp)
    if distance - d_max <= node.get_median():
        knn(node.get_left(), query, k)
    if distance + d_max >= node.get_median():
        knn(node.get_right(), query, k)
    return
```

Time Performance

RANGE SEARCH WITH VPTREE

RANGE SEARCH WITHOUT INDEX

K-NN WITH VPTREE

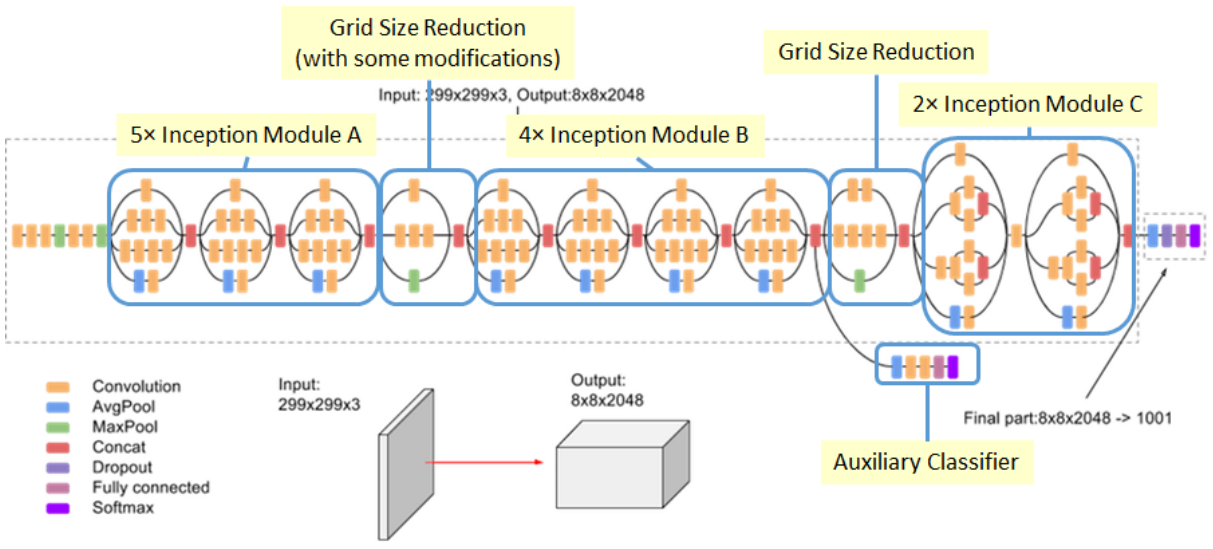
K-NN NOT OPTIMIZED WITH VPTREE

K-NN WITHOUT INDEX

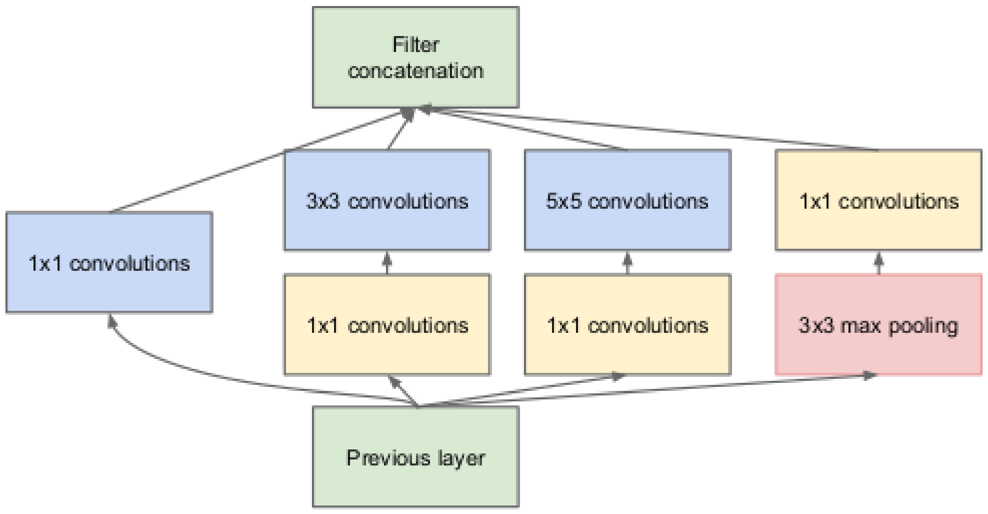
FEATURE EXTRACTION WITH A PRE-TRAINED NETWORK

STRUCTURE OF INCEPTION V3

Inception v3 is a Convolutional Neural Network model based on the paper "Rethinking the Inception Architecture for Computer Vision" presented by Szegedy et al. that showed an 78.1% accuracy on the ImageNet dataset. Its structure is composed by 42 layers split in different modules.



The main module and the core idea of the Inception model that made it one of the most used pre-trained models is the Inception Layer:



This layer is used in the Neural Network to select the filter size that will be relevant to learn the required information of the next layer, to learn the best weights and so automatically select the more useful features. This job is performed by the parallel Convolution Filters and the 1x1 Convolution Filters are used to perform dimensionality reduction.

PERFORMANCE

FEATURE EXTRACTION WITH A FINE-TUNED NETWORK

STRUCTURE

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
inception_v3 (Functional) (None, 5, 5, 2048) 21802784

_________________________________________________________________
gap (GlobalAveragePooling2D) (None, 2048) 0

_________________________________________________________________
dense (Dense) (None, 512) 1049088

_________________________________________________________________
dropout (Dropout) (None, 512) 0

_________________________________________________________________
last_relu (Dense) (None, 512) 262656

_________________________________________________________________
last_dropout (Dropout) (None, 512) 0

_________________________________________________________________
classifier_hidden (Dense) (None, 5) 2565
=================================================================
Total params: 23,117,093
Trainable params: 23,082,661
Non-trainable params: 34,432

_________________________________________________________________
```

TRAINING (EARLY STOP, CLASS WEIGHTS, DATA AUGMENTATION...)

PERFORMANCE UNTIL GAP

PERFORMANCE UNTIL DENSE

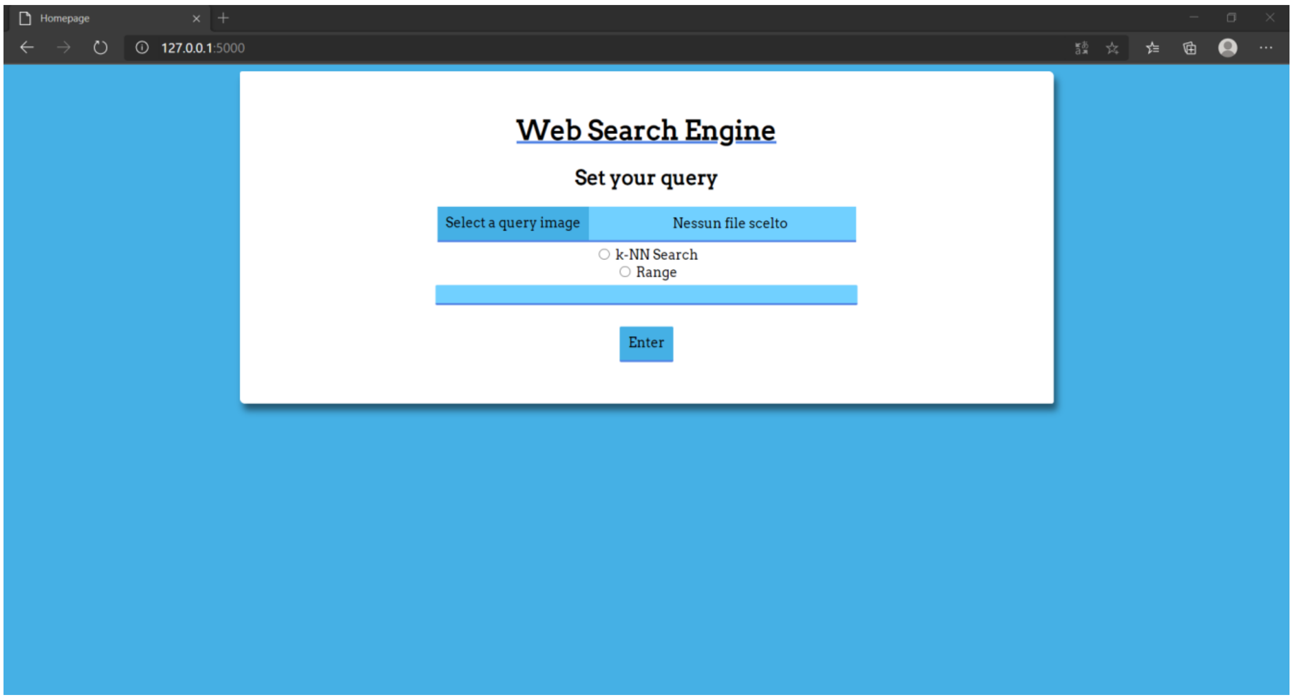
PERFORMANCE COMPARISON BETWEEN NETWORKS

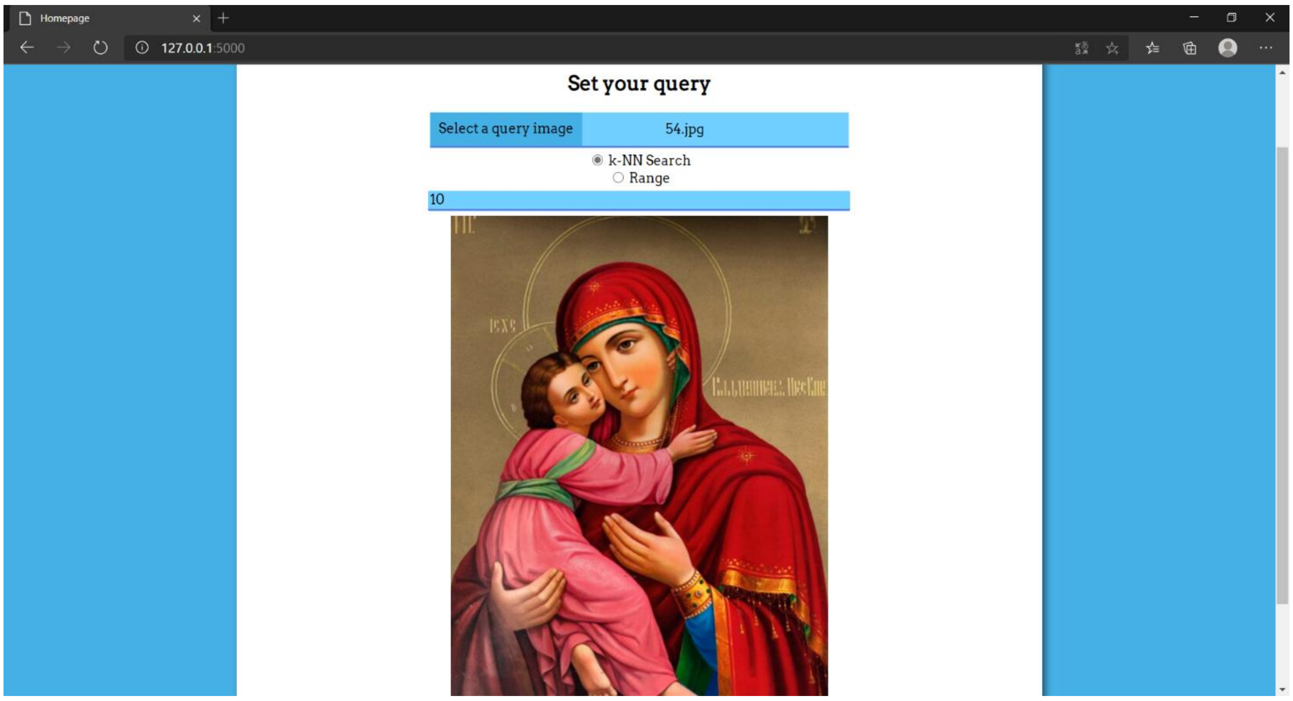
PLOTS OF PERFORMANCE

USER INTERFACE FOR WEB SEARCH ENGINE

ARCHITECTURE OF THE WEB SEARCH ENGINE

SCREENSHOTS





Homepage

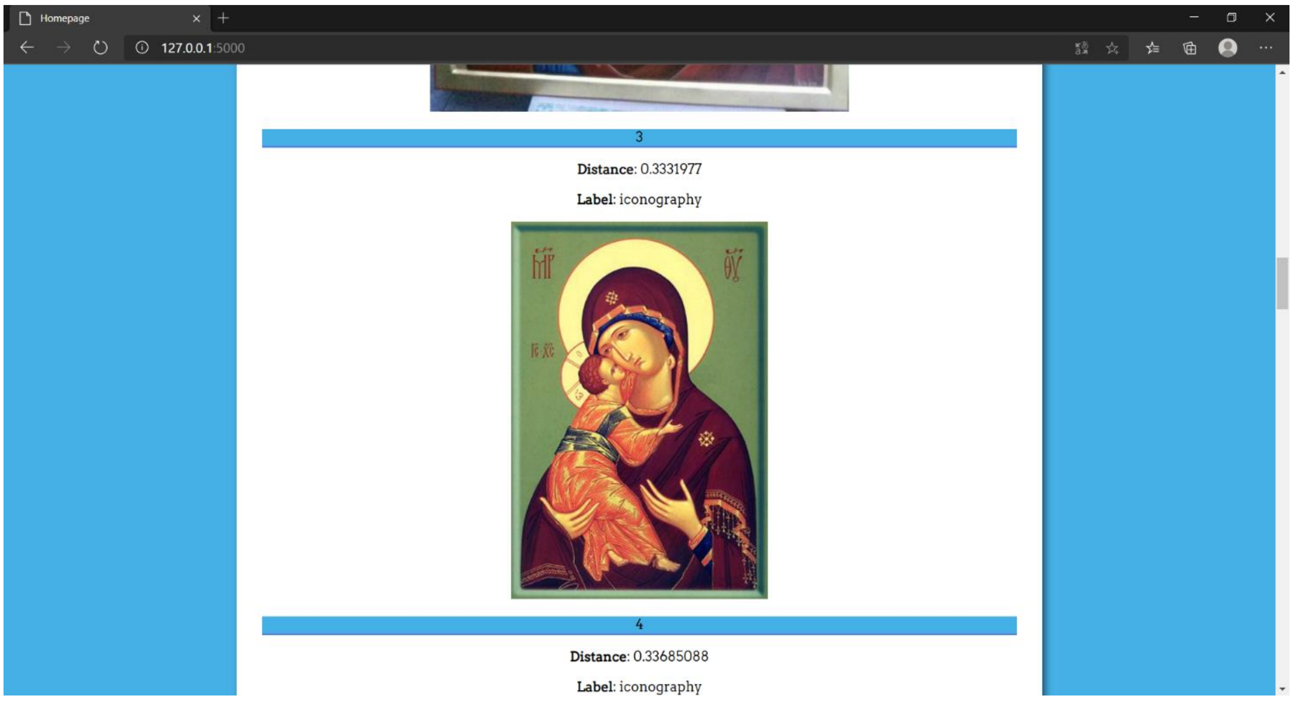
127.0.0.1:5000

Set your query

Select a query image 54.jpg

(● k-NN Search) ○ Range

10



Homepage

127.0.0.1:5000

3

Distance: 0.3331977

Label: iconography

4

Distance: 0.33685088

Label: iconography