MyWatson: A system for interactive access of personal records

Pedro Duarte
pedro.m.duarte@tecnico.ulisboa.pt

Supervisor: Prof. Arlindo Oliveira

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

With the number of photos people take growing, it's getting increasingly difficult for a common person to manage all the photos in its digital library, and finding a single specific photo in a large gallery is proving to be a challenge. In this thesis, the MyWatson system is proposed, a web application leveraging content-based image retrieval, deep learning, and clustering, with the objective of solving the image retrieval problem, focusing on the user.

MyWatson is developed on top of the Django framework, an high-level Python Web framework, and revolves around automatic tag extraction and a friendly user interface that allows users to browse their picture gallery and search for images via query by keyword. MyWatson's features include the ability to upload and automatically tag multiple photos at once using Google's Cloud Vision API, detect and group faces according to their similarity by utilizing a convolution neural network, built on top of Keras and Tensorflow, as a feature extractor, and a hierarchical clustering algorithm to generate several groups of clusters.

Besides discussing state-of-the-art techniques, presenting the utilized APIs and technologies and explaining the system's architecture with detail, a heuristic evaluation of the interface is corroborated by the results of questionnaires answered by the users. Overall, users manifested interest in the application and the need for features that help them achieve a better management of a large collection of photos.

Keywords: Content-based image retrieval, Google Cloud Vision, Clustering, Face detection, Convolutional neural network

1. Introduction

The recent increase in popularity of smartphones [23] caused a boom in the number of digital pictures in the world [7], as smartphone cameras have improved a lot and are a lot more practical to carry than the bulky DSLR cameras, which means a lot more "common" people take pictures. The rise of social media platforms like Facebook and Instagram also seem to have helped the rise of smartphone cameras [7] and thus the number of pictures a common person takes.

One's popularity in any social media platform is directly translated by the number of profile "followers" or "views", or the number of "likes" a photo has. Profiles and photos that are easier to find usually get more views and likes, which means increasing the accessibility of a profile or media is in one's best interest in the pursuit of popularity. In most social media, **hashtags** are what make a certain photo or other types of media popular: when a user searches for a given keyword, photos without hashtags containing that keyword will probably not show up. In other words: more tags means greater chance of appearing in other user's results, consequently increasing the chances of popularity. Often, the tags may not even be related to the content of the photo itself, as in this social media setting, quantity often triumphs quality when dealing with tags.

The assignment of tags is often boring and slow, especially when in great quantity. The process of manually assigning labels is motivated, in this social media setting, by the pursuit of popularity (Ames [5]). However, in a more personal environment, e.g. a person that likes to take photographs but does not want to show them to the public, tagging is also important to make the photos searchable and manageable, and this is a crucial necessity once a collection of photos starts getting bigger and bigger. In spite of thinking that tagging is use-

ful and a necessity, people still do not bother with manual tagging (as presented by Ames) because the cons largely outweigh the pros: it is a very boring process, it takes a lot of time and the tagging usually needs to be exhaustive in order to cover all the content in the photo.

Ultimately, what users want is to easily retrieve images via queries. There are two main techniques for retrieving photos this way: Content-Based Image Retrieval and Text-Based Image Retrieval. Content-Based Image Retrieval (CBIR) approaches focus on "reading" the content of the image and extracting information (features) that can later be checked to see if it matches a query. On the other hand, Text-Based Image Retrieval approaches completely ignores the content of the image itself, and tries to find useful information in the "textual context" of the image, such as captions or other types of surrounding text in a web page where the image is located. Due to the fact that text-based approaches still requires manual work and are dependant on other information that may or may not be available, they will not be further discussed in this document.

Eakins [9] defines CBIR as the "process of retrieving desired images from a large collection on the basis of features (such as colors, texture and shape) that can be automatically extracted from the images themselves". Note that the word "automatically" plays a huge role in the definition – if the feature extraction is not automatic, then the retrieval is not CBIR, even if the keywords describe the content of the image. A general CBIR system comprises image crawling, image representation and image indexing in an offline phase, and the online phase starts when the user translates his intention into a query, where the system will then retrieve and rank the results to return to the user. The most relevant steps to this thesis are the image representation and the retrieve and rank steps. All the other steps are very linear and simple with almost no challenge. However, it is still important to define what kind of query users can formulate in a CBIR system.

Queries are split into five different categories according to Zhou et al. [25], depending on how they are formulated: query by keyword, example, sketch, color layout or concept layout. The type of query this thesis will focus is the query by keyword, as it is the most simple and convenient way for a user to perform a query. In a query by keyword, the user introduces one or more keywords that describe the content of the images he wants to retrieve. Eakins also sub-categorizes this type of query by levels [9], according to its semantic level, from level 1 to level 3. A **level 1** query comprises primitive elements of an image, e.g. "find pictures with a large uniform blue region on top". On the other hand, a **level 2** query jumps the semantic gap, because the system needs to have at least a minimal understanding of objects and their names, and includes types of objects or even individual objects or entities. Finally, a **level 3** query is comprised of abstract concepts, such as activities or emotional or religious significance. The last two levels are classified as semantic image retrieval.

Image representation consists of extracting low-level features from images, such as corners, edges, or colors, and transforming them in a single vector of high-level tags, such as "sky" or "beach". This process is not trivial: there is a gap between what the computers "sees" and the actual name of the objects present in a picture, called **semantic gap**. Therefore, extracting a singular, compact representation of an image is a two-fold process: reading the low-level features and then bridging the semantic gap. Some techniques, such as Convolutional Neural Networks (CNNs), perform both steps by themselves, by learning high-level descriptions of low-level features using classifiers.

In order to develop a fully functional system that can automatically tag photos and offer additional features for the management of a large image gallery, some challenges must be overcome.

The first challenge is to solve the automatic tagging of the photos. In parallel with the steps previously addressed, the most challenging part is how to extract the high-level features of an image.

The second challenge corresponds to making a user-friendly and accessible interface in order to facilitate the life of the user.

Finally, the last challenge is what can be done to further enhance the user experience when managing photos, i.e. finding a way to help the user find the information he needs faster. This thesis proposes utilizing facial detection for face clustering, by classifying face images in a CNN pre-trained on a large face dataset and then removing the last classification layer in order to extract facial features.

The main contribution of this thesis falls on the general area of CBIR. By focusing on the automatic tagging of photos, I will be testing the usefulness of a system that eases the process of managing a photo gallery without having to waste a potential user's time and effort. Furthermore, machine learning also plays a very important role in this goal: by utilizing a CNN which was pre-trained to classify faces and cutting off the final classification layer, I will be able to recognize faces in photos and cluster them – by utilizing the output from the feature extraction layer as features for the clustering –, increasing the user's available information, which further enhances the ability to man-

age a photo gallery.

Implementing a web interface is also a contribution in itself, as it should be interesting to see the usefulness that an easily accessible system may have on a user's habit, i.e. how often would a user use the system to manage or look through his photo gallery.

After this introductory section briefly explaining some CBIR general terms and techniques, challenges, objectives and contributions, the thesis will cover the algorithms and technologies used, as well as the system architecture and evaluation, in the following sections:

Section 2 The methods and algorithms used in the MyWatson system will be discussed, namely the automatic tagging, the facial recognition, convolutional neural networks and clustering. The objective is to provide a general understanding of the techniques used.

Section 3 Discusses the technologies used in the development of the MyWatson system. Focuses on the frameworks and APIs used.

Section 4 Provides an overview of the system architecture, i.e. how its modules are connected and how common use cases are processed, such as the automatic tagging of photos or the clustering by faces.

Section 5 This section provides a heuristic evaluation of the user interface and discusses the results of a system evaluation done by real users, as well as possible reasons for those results and a comparison between the heuristic evaluation and these results.

Section 6 In the final section, conclusions will be drawn from the current status and evaluation of MyWatson, as well as future improvements and work.

2. Methods and Algorithms

Low-level feature extraction is the core of a CBIR system: it's how a computer sees an image and extracts any kind of low-level information from it, which allows for further refinement into high-level semantic objects and concepts. Currently, low-level features can be extracted in a manual or an automated manner, and manually extracted features can be local or global features, depending on the focus of the algorithm.

Manual or hand-crafted features are not extracted manually by a human, despite its name – instead, they are derived via an algorithm which was specially engineered (by humans) to extract those specific features. This is the difference versus automatically extracted features: in the latter, it is not known what type of features are extracted, because they are automatically learned. Of this type, the most common techniques nowadays include the usage of neural networks with deep architectures to learn the extraction process with high-level abstractions close to those used by humans, which will be further discussed.

The main difference between global and local features is that the resulting output of extracting global features is a vector that describes the entire image. On the other hand, the output of extracting local features is a vector of vectors of regions, and some sort of aggregation is needed to compact those vectors into a single, fixed size vector. In short: global features describe the whole image in terms of colors, shapes and textures, and the local features describe multiple regions that belong to an image.

Global features are attractive, because they are very compact in terms of representation, and as such any standard classifier can be utilized to produce high-level features. However, they are often not directly related to any high-level semantics, and are also very sensitive to background clutter, so they should mainly be utilized in images that only have one object in a front-plane, or with a good and clear segmentation. Examples of global feature extraction techniques include computing a color histogram (Wengert et al. [24]), or GIST (Oliva and Torralba [18]) which represents a scene with a *Spatial Envelope* – i.e. the shape of the space that is present in a certain image – using perceptual dimensions such as the naturalness, openness, roughness, expansion and ruggedness of an image.

On the other hand, local feature focus on keypoints, the salient image patches that contain the rich local information of an image. Generally, there are two steps in local feature extraction: the keypoint detection and the region description. A feature is comprised of a keypoint and a descriptor: the former is the location of a certain region or interest point in an image, and the latter is some set of values that describe the image patch around the keypoint. Descriptors should be independent from the keypoint position, and should also be independent from scale and transformations. Keypoints are detected such as that even with some kind of change – a scaling or rotation, for example – they can still be identified. Usually, some kind of detector is used, and one such example is the Difference of Gaussians (DoG) [15] detector.

With the interest points detected, descriptors of the area around those points are also extracted. The most well-known choice of descriptor is the Scale-Invariant Feature Transform (SIFT) by Lowe [15]. SIFT is a patented algorithm which extracts both keypoints and descriptors independent from scale. An alternative to SIFT is the Speeded Up

Robust Features (SURF), presented by Bay et al. [6], which is a more robust and faster alternative to SIFT, and includes feature matching and orientation assignment.

After low-level features (keypoints and descriptors) are extracted, the semantic gap must be bridged in order to obtain high-level labels that describe the content of the image. Liu et al. [14] discusses five state-of-the-art techniques to bridge the semantic gap and implement high-level semantic image retrieval: object-ontology, machine learning, relevance feedback, semantic template and web image retrieval.

The basic idea of an object ontology system is to map low-level features to high-level keywords based on our knowledge. On the other hand, when using machine learning, high-level concepts are learned based on a set of annotated images.

Relevance feedback is not very interesting in this context, as this technique requires user interaction, and so does semantic template. Finally, in web image retrieval, textual clues about the image are used, for example, title information, text surrounding the picture, HTML tags, hierarchical structure of links, etc. This information, however, is usually not good enough. These techniques can also be combined, complementing each other.

Convolutional Neural Networks are, not only another paramount technique to solve the semantic gap problem, but also to solve the face detection problem which is, in fact, what MyWatson utilizes to do so. Convolutional Neural Networks (CNN) are a class of Artificial Neural Networks (ANN) with an architecture specifically made to analyze visual data. This means that any kind of data that can be arranged to look like an image can successfully be learned with a CNN.

Artificial Neural Networks are systems that are inspired in actual biological neurons in animal brains: computational units (neurons) have scalar inputs and outputs, with each input having an associated weight. Neural networks are trained to predict the class of an input by repeatedly adjusting the weights associated with each input according to the error, i.e. the difference between their prediction and the correct class. The error serves as the basis to make adjustments on the weights, in a process called backpropagation.

Each main building block of the CNN, i.e. a layer, has a singular purpose, and belongs to one of the three following types: a convolution layer, a pooling layer, or a fully connected layer. Different CNN architectures arrange these layers in an sequential fashion in different manners, but the idea is generally the same. First, the input layer transforms an image into a 3D vector: one dimension for the width, other for the height, and other for the color channels. For example, an RGB image with 224x224 pixels will be transformed into a $(224, 224, 3)$ vector. Then, to recognize patterns, filters are used in the **convolution layer**: each filter that corresponds to a pattern is convoluted around the entire image and the scalar product between the image pixels and the filter is computed. Because multiple filters are convoluted, one images becomes a stack of images, i.e. the matrices containing the scalar product. In order to shrink the image stack of the previous layer, a technique known as **pooling** is used. Pooling consists of walking across the image with a window of a given size and with a specific stride – commonly 2 for both – and saving a combination of the values from the image within the window in a new, smaller matrix. Common types of pooling include the **max pooling** and **average pooling**. Another important layer in CNNs is the normalization layer, which prevents negative values. This is commonly done by a **ReLU layer**, which uses a Rectified Linear Unit function that simply swaps all negative values to 0. Finally, a **fully connected layer** flattens the input image and treats it as a vector with one dimension. Each element of the vector is assigned a weight as a product of the training and, during the classification, each element essentially casts a weighted vote on a given class. Thus, the highest voted class is assigned to the input image.

Another useful technique in the field of deep learning is **transfer learning**, defined as using knowledge of one domain in another domain. More specifically, by training a neural network in one dataset corresponding to a given domain and then use it another dataset by fine-tuning the same network. In practice, this technique is used to speed up the training process, by obtaining a pre-trained model on a very large dataset and then use it as initialization or as a feature extractor for the task of interest. It is common to tweak the weights by continuing the backpropagation process on the new dataset. In MyWatson, this technique is leveraged as a fixed feature extractor: a CNN pre-trained on 2.6 million images of 2.6K different people (Parkhi et al. [19]) is used to extract the features from images containing faces, which are then utilized for clustering purposes.

However, face detection is not face recognition: in the first, the faces and their landmarks (nose, eyes, mouth, etc) are detected, and only in the latter the faces from an individual are matched or verified against a database. In MyWatson, both techniques are used: the faces are first detected and cropped, and then are clustered together, which acts as facial recognition. Face clustering is a good alternative for facial recognition, specially in the case where actual identities – i.e. names – are not

needed and only some kind of distinction between faces is required.

The main purpose of clustering or cluster analysis is to transform a large set of raw, unstructured data into groups of semi-structured data with related properties or features – i.e. data within a cluster is more semantically close to each other than with data outside of a cluster. Some authors, such as Mann and Kaur [16] and Fahad et al. [10], divide clustering algorithms into five general categories: hierarchical, partitioning, density-based, grid-based and model-based.

In partitioning-based clustering algorithms, the number of clusters must be specified *a priori*, and data is divided according to these partitions. The most well-known example of this category is the K-means algorithm. Hierarchical-based algorithms seek to build hierarchies of clustering and may do it in two different ways: agglomerative and divisive. In agglomerative approaches, also called "bottom-up", each observation starts with its own cluster, and pairs are merged as the hierarchy moves up. Divisive or "top-down" approaches begin with a cluster containing all the data points, and as the hierarchy moves down the clusters split in pairs. A dissimilarity measure between sets of observations is required, generally using a distance metric and a linkage criterion which specifies the dissimilarity measure as a function of the distances between pairs of observations. Common linkage criteria include **maximum or complete linkage**, **minimum or single linkage**, **average linkage** and **ward linkage**. An advantage of hierarchical-based approaches is their flexibility, as the level of granularity of clusters is embedded in the approach itself. On the other hand, most algorithms of this type do not look back to a cluster for improvement purposes once it has been built, and termination criteria can sometimes be vague.

In density-based clustering, data points are separated according to regions of density, connectivity and boundaries. Sparse areas are required, with the purpose of separating clustering, and data points in those sparse areas are usually seen as noise or border points and thus are not assigned a cluster. Grid-based algorithms divide the data space into grids, and have a fast processing time, as the dataset is visited once to build the statistical values for the grids. The idea is to find clusters from the cells in the grid structure. Finally, model-based methods optimize the fit between the given data and some predefined mathematical model, based on the assumption that data is generated by a mixture of underlying probabilistic distributions. Clusters are then identified by seeing which objects are more likely to belong to the same distribution. The Expectation-Maximization (EM) algorithm [8] is an example of model-based clustering that, although commonly used as a clustering algorithm, is a general methodology for finding distributions parameters. The idea behind the EM is to find parameters that best fit the data given, following two steps iteratively: the expectation step (E) and the maximization step (M).

When finding out in how many k clusters should the data be split in, some kind of metric is needed to evaluate the quality of the clusters, much like in supervised learning to find out the precision or recall of a certain algorithm. However, this is not so trivial when utilizing clustering algorithms to do so. Considerations about this are done in the Scikit-learn [20] website, particularly that the separation of data should be compared to some ground-truth set of classes rather than cluster labels. Another alternative is to take into consideration if the data points are more similar to each other within a cluster than with other points outside of it. Metrics that require ground-truth labels cannot be used in MyWatson, as no previous knowledge can be obtained. An interesting metric that does not require ground-truth labels is the **silhouette coefficient**, which evaluates the level of definition of a cluster. For a single data point, the silhouette score s is defined as

$$s = \frac{b - a}{\max(a, b)}$$

where a is the mean distance between the sample and all the other points in the same class, and b is the mean distance between the sample and all the other points in the next nearest cluster. The score of the whole cluster is the mean of all the scores for each sample, and is bounded between -1 and 1, indicating incorrect clustering and dense clustering, respectively. Furthermore, scores near 0 indicate overlapping clusters.

3. Technologies

The decision of developing a web application supports the idea that the user should have as little trouble as possible when using the system, and having to install a program in his personal computer to do so may not be the best approach. The usage of existing frameworks implies several advantages, such as efficiency, less bugs and good integration, as well as generally good documentation.

The APIs and frameworks that are a part of MyWatson are **Django**, **Google Cloud Vision**, and **Keras + Tensorflow**.

Django [1] is a Python web framework that simplifies most aspects of web development. Django rules itself by the Don't-Repeat-Yourself (DRY) principle, which aims at reducing repetition of soft-

ware patterns, leveraging abstractions and using data normalization. The Django framework is based on the Model-View-Controller (MVC) architectural pattern, although Django's architectural pattern is called the Model-View-Template (MVT) [3] since the controller is handled by the framework itself:

- **Model (M):** the same as in the original MVC architecture. In this layer, Django's Object-Relational Mapping (ORM) provides an interface to the MySQL database.
- **Template (T):** contrarily to the standard MVC, the template is the presentation layer, which means that it controls how a given set of data is displayed and in what form.
- **View (V):** this layer, while similar to its homonym in the MVC, has more characteristics of a controller, because while in the MVC the view controlled *how* the data was seen, here the view controls *which* data is seen. In other words, the view fetches the content and the template presents that content.

If Django is the scaffolding that supports the whole MyWatson system, the Google Cloud Vision API [2] is its heart, as it provides fully automatic tags given a photo, which is the most important aspect of the solution. By using the Google Cloud Vision API, the focus can be shifted from solving the automatic tagging problem, and rather build a working application that fully utilizes the advantages of an automatic tagging system.

The job of the Google Cloud Vision API is simple: given an image, return information about the contents of the image, which include objects, properties, landmarks, logos, faces and web entities. Another advantage of Google Cloud Vision is that it gets better annotation results with time as new concepts are introduced. Furthermore, because Google is known for designing advanced and sophisticated systems, Google Cloud Vision is likely very scalable and should improve in the future.

Finally, Keras is a high-level neural network API used as a way of easily leverage a CNN without having to design or train it. Keras can be seen as an interface to TensorFlow, offering more intuitive abstractions of higher level. Another advantage of Keras is that it already implements several known CNN architectures developed for general image classification, such as the VGG16 – the 16-layer model used by the VGG team in the ILSVRC-2014 competition [22] – and ResNet50 [11].

Another library on top of Keras, `keras-vggface`, provides implementations of the VGG16, ResNet50 and Senet50 models and comes with weights pre-trained on the dataset by Parkhi et al. [19] with over 2.6M images and 2.6K different people. With this, all the building blocks for extracting facial features are in place, and no training is necessary.

The most prominent feature of `keras-vggface` and Keras itself is the ability to import and download models and pre-trained weights, providing the ability to utilize a functional and trained CNN to classify images without the need to train it, fully bypassing the most prominent disadvantage of using neural networks. Another advantage is the ability to customize or tweak the model: parameters and activation functions can be changed and layers can be added or removed, which is an important feature that allows the removal of the fully-connected classification layer – the last layer. This way, by leveraging CNN features "off-the-shelf" [21], features are the output of face "classification", turning a CNN into a feature extractor for face images.

Finally, Tensorflow [4], which functions as the backend to the Keras libraries, is an open-source software library created by the Google Brain team. At its core, TensorFlow is a framework used to build Deep Learning models easily as data flow directed **graphs**. These graphs are networks of **nodes**, each one representing an **operation** that can range from a simple addition to a more complex equation, and represent a data flow computation, allowing some nodes to maintain and update state. The values that flow through the nodes are called **tensors**, which are arrays with arbitrary dimension that treat all types of input uniformly as n-dimensional matrices.

4. Architecture

Based on the previously presented discussion of the problems and techniques of content-based image retrieval, clustering, face detection and recognition, as well as deep learning as a powerful tool to solve some of the previous problems, the details of how MyWatson leverages the presented techniques in order to build an environment that the user can fully utilize to upload, manage and automatically tag their photos will be discussed.

When using MyWatson, the user first should create an account and log-in. Only after can the user upload photos, browse the gallery, edit or add tags, perform queries and, last but not least, view aggregated similar faces.

MyWatson's architecture is graphically described in the diagram shown in picture 1, and is composed of three different main components:

- **The front-end:** this is what the user sees. It is essentially comprised of the MyWatson website, which hosts the application. The most relevant detail of this component is the web-

site itself, i.e. the user interface. Despite not being the most important element of the system, it is still a very important one: a good user interface facilitates the life of the user, avoids wasting his time by not hiding crucial elements and information and is accessible and simple to use.

- **The Django controller:** the Django framework is what receives requests and returns responses from and to the front-end, respectively. It is also what implements the whole server-side business logic, including the management of the database – which, despite making the database also an element in this component, could be done without Django – and the pre-processing of some data to send to the main MyWatson application, the core. Because of this, it is seen as the mediator between the front-end and the modules that perform the tagging, retrieving and learning operations.
- **The core:** divided into four modules – the main application, the retrieve & rank, the automatic tagger and the learning modules –, the core is where the previously discussed CBIR and learning techniques are implemented. The main module, however, is the intermediary between the three other modules and the rest of the application and therefore, in the future, will not be referenced as a "module", unlike the other three. This approach – which roughly follows a facade software-design pattern – makes adding new functionality very easy, avoids a lot of complexity and provides a singular, simplified interface that allows communication with the rest of the application.

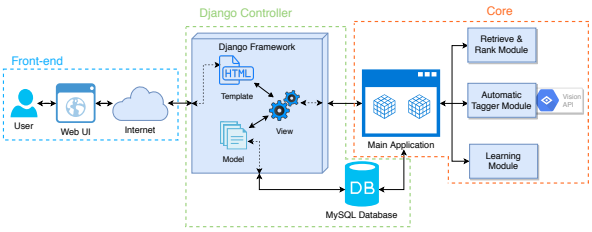

Figure 1: An informal diagram picturing the architecture of the MyWatson system.

The most relevant implementation is done in the core modules, where the main logic of the application is, i.e. the code that is directly related to all the previously discussed problems and techniques is presented in the retrieve and rank, automatic tagger and learning modules. The former two implement techniques that try to solve the CBIR problem by leveraging the Google Cloud Vision API and a text-based information retrieval technique called TF-IDF, while the last module implements and utilizes a convolutional neural network to extract features of face images and leverages a hierarchical clustering algorithm to generate groups of face clusters.

The main module is a file called `MyWatsonApp.py` (MWA), and essentially has functions that operate like a direct channel for each of the other three modules, i.e. if a view requests the MWA to tag a batch of images, the MWA will simply ask the automatic tagger module to tag the batch of images. Then, the automatic tagger module will return the tags, and the MWA will pass the result along to the view that requested it. The job of MWA is simply to be a facade to the rest of the core in order to provide an abstraction for the Django controller and make it easier to decouple.

The automatic tagger module is probably the most important module of the whole MyWatson core. It is what powers the image retrieval by keywords and implements the face detection so it is, consequently, a hard prerequisite for the other two modules. The role of the automatic tagger is to attach textual information that describes the content of the image or is in any way relevant, i.e. to extract tags. To further improve the efficiency of the whole process, the extraction of tags is done in batches. The process of obtaining a list of tags itself is divided into three steps: building the requests for the Google Cloud Vision API, grouping the requests in batches and, finally, requesting the API client to annotate the batches, one by one.

Additionally, tags that are associated with faces in the image, i.e. tags that have the "`person`" description and have bounding boxes, will trigger the creation of the face objects, which will be used in the learning module for clustering.

The work of the retrieve and rank module begins when the user executes a query in the search box, located in the navigation bar. The objective of the module is to return a list of Django photo objects that are relevant to the given query set. To this end, the TF-IDF model from text-based information retrieval is leveraged, as the photos can be approximated as documents containing words (tags). The retrieval process begins by transforming the query terms into a query vector. For each query term, a weight is derived, describing the importance of a term, ranging from 0 to 1, respectively indicating low or high relevance. As a very basic weighting strategy, the original query terms are assigned the weight 1, and the stemmed and lemmatized terms have a 0.7 weight. The idea is that terms directly input by the user should have higher importance than their expanded forms. Similarities are calculated, by applying the cosine similarity function over the BoW vectors (query and image vectors) and then,

in the end, the results are sorted according to their similarity values in order to display the most relevant photos first.

The learning module is the last and most complex module of the MyWatson core. Its job is to, given a list of face objects, compute a configurable number of sets of clusters, each with a different number of clusters, by grouping faces according to their similarity. The general idea of the process is to dynamically change the list of values of $n\_clusters$, the number of clusters to compute. This way, even if the user has, as an example, 10000 photos, instead of trying to compute 10000 clusters – it is guaranteed that there is a cluster group where the number of clusters corresponds the number of different faces in the user's photos (optimally, because this also depends on the face features and clustering algorithm) –, the module will apply a variation of a local search algorithm in order to optimally find the best cluster groups with a high probability.

The algorithm acts in accordance with the following steps:

1. The order of the number of clusters in which the module will compute the groups is initialized as a list, called `cluster_try_order`, of three numbers, corresponding to the first, second and third quartiles. These values are calculated according to equations 2, 3 and 4.

$$offset = \left\lfloor \frac{\#faces}{4} \right\rceil \quad (1)$$

$$mid\_n\_c = \left\lfloor \frac{\#faces}{2} \right\rceil \quad (2)$$

$$min\_n\_c = mid\_n\_c - offset \quad (3)$$

$$max\_n\_c = mid\_n\_c + offset \quad (4)$$

   Then, a balance threshold b_thresh is also initialized (equation 5), corresponding to the number of times the score must drop more than it rises for the algorithm to stop computing cluster groups. The ratio r is a negative real value that defines the rate at which the necessary number of clusters decreases with the number of faces, and the `min_clusters` describes the minimum number of clusters that should always be computed. The ratio must be a value between 0 and 1, and should not be too high, because if the amount of face images is too large a lot of clusters will have to be computed. For the same reason, the minimum number of clusters should not be too high either. After some informal, empirical analysis, the parameters $r = -0.25$ and $min\_clusters = 25$ were deemed as acceptable, as they allow the computation of all cluster groups in cases where there are not a lot of face images, and still compute a good number of clusters to have a good chance of finding a good maxima when the number of face images is high.

$$b\_thresh = \lfloor r \times \#faces - min\_clusters \rceil \quad (5)$$

2. The three initial cluster groups, corresponding to the groups with n_1 clusters within the initial vector, i.e. $n_1 \in cluster\_try\_order_0$, are computed, along with their respective silhouette score. From these three, the cluster with the highest silhouette score is chosen as the reference cluster group, i.e. because it has the highest score, it is deemed likely that a decent local maxima is closer to it that to the other groups.

   The clustering algorithm is as follows:

   (a) Given a set of faces and an integer n corresponding to how many clusters will the faces be divided into, the algorithm starts by obtaining the features from the face images, which can be retrieved from the face objects. In the case that the features of a certain face object have already been computed, they are simply retrieved from the database. Otherwise, the CNN model, which uses the ResNet50 architecture with no pooling, is used to extract the features.

   (b) An hierarchical clustering algorithm with ward linkage is then used to predict the clusters of the data points corresponding to the faces, agglomerating them via a bottom-up approach and stopping when the number of clusters reaches the requested value. Furthermore, the average silhouette score of this group is also calculated and returned along with the labels of each data point.

3. After the reference cluster group is chosen, its adjacent neighbours are added to the `cluster_try_order` list, and their clusters and silhouette scores are also calculated. The neighbour with highest score defines the direction in which the search will be conducted. As an illustrative example, consider that, from neighbors of the reference cluster group $n = 5$, namely $n = 4$ and $n = 6$, the one with the

highest score is the group with $n = 4$ clusters. In this case, the search would be conducted "downwards", to lower clusters. Finally, all of the n values corresponding to that direction will be added to the `cluster_try_order` until their lower and upper limits (3, 2 and 1).

4. In the main loop, the added groups are computed. Each time a cluster group is computed, the highest score is updated if that cluster group has higher score. If the score is updated, the `balance` variable is incremented by one. Else, i.e. the score keeps dropping, the `balance` is decremented by one. If the balance reaches the threshold, it means that enough evidence has been gathered to indicate that it is not probable that a better cluster group can be found, and the algorithm stops. Otherwise, the algorithm runs until all of the groups in the `cluster_try_order` have been computed.

5. Finally, in the case that the main loop broke due to the balance reaching the threshold, the limit cluster groups are calculated, i.e. the groups with 1 and $\#faces$ clusters.

In summary, the idea of this strategy is to find the spot were clusters groups most likely have the highest score, i.e. better model the face images, and then to find the direction in which the score rises, gathering further evidence of the presence of a better cluster group along the way by keeping score of the balance.

Finally, the MyWatson website follows a very simple design, where a fixed navigation bar is always present on the top of the page and the rest of the page displays the respective content. A relevant note is that all pages contain a "mini MyWatson" in the bottom right corner which, when clicked, displays helpful information about the current page, in order to offer some assistance to users that may have trouble understanding some aspects of the interface.

5. Evaluation

The evaluation methodology for the MyWatson web application is essentially comprised of user feedback, collected after the development of MyWatson was complete. Additionally, an heuristic inspection is performed as a way of evaluating the usability of the user interface, based on the well-known heuristics by Jakob Nielsen [17].

The objective of heuristic analysis is to identify usability problems along the user interface of a given application. For each of the ten Nielsen's heuristics, a rating between 0 and 5 is given in table 1, corresponding to "does not comply" and "fully complies", respectively.

Heuristic	Evaluation
H1. Visibility of system status	5
H2. Match between system and the real world	4
H3. User control and freedom	3
H4. Consistency and standards	5
H5. Error prevention	5
H6. Recognition rather than recall	5
H7. Flexibility and efficiency of use	3
H8. Aesthetic and minimalist design	5
H9. Help users recognize, diagnose, and recover from errors	2
H10. Help and documentation	5
Average	4.2

Table 1: Heuristic evaluation of MyWatson's UI

The user feedback was collected via online forms from Google Forms. The users were asked to experiment with the MyWatson app for a while and perform some tasks, such upload photos, search for photos or analyze the people cluster groups. As such, users were required to have an account to do so. Although the process of registration is very simple and quick, potential users might have been reluctant in creating an account just for trying a website and experimenting for 10 minutes. In order to mitigate this issue, a testing account was previously created, so that users that found registering for an account too troublesome could skip that step and test MyWatson. Some generic photos were uploaded to the testing account beforehand, such as pictures of beaches, cats, dogs, streets and celebrities, to name a few.

The form is comprised of 9 questions, 2 of which ask for the user's to write their opinion and additional observations and thus are not required to be answered. In summary, users are asked their opinion about the user interface, the usefulness of MyWatson and their potential use and recommendation. There were 27 replies to the questionnaires, from which the great majority's age is comprised between 22 and 24 years, inclusive.

Overall, the reports indicate that users think MyWatson would be a useful application to use, but not constantly, with 37%, 41% and 19% of the inquirees giving usefulness ratings of 3, 4 and 5, respectively, and a third of the inquirees replied with a 3 on a scale of 1 to 5 to the question "How often would you use MyWatson?", with a close 30% giving a rating of 4, and only 11% gave a rating of 5.

Most users also agree that the interface is good

and simple to use, as the ratings 4 and 5 were given to the UI by respectively 48% and 44% of the inquirees, and 67% gave the maximum rating to the ease of use. These results corroborate the heuristic evaluation, where the user interface was also considered to be adequate in terms of usability.

6. Conclusions

Up to this point, the known limitations of the MyWatson system include the lack of customization of the user interface and the lack of possibility of creating albums or new clusters, as pointed out by the users. Additionally, these APIs, despite speeding up the process of development and leveraging their advantages, also include their limitations. Namely, Google Cloud Vision API still has some limitations, such as the fact that the extracted tags correctly label dominant objects in the image, but do not label every object of interest in a photo, as also noted in an article by Chad Johnson [13], with reproducible results.

However, despite its limitations, its clear that Google Cloud Vision is evolving. In 2017, Hosseini et al. [12] reported that the API was not robust to noise, i.e. it could not extract the correct labels from pictures with a small degree of noise introduced. Despite this, I repeated the same experiment with the same photos utilized in the experiment, and this problem was not found.

This thesis aimed to present a practical solution to the problem of Content-Based Image Retrieval. To this end, the development of a modular application that provides an easy and simple way of managing photos was discussed. The MyWatson system leverages the Google Cloud Vision API for automatic tag extraction, including objects, landmarks, colors and others, as well as faces and their coordinates that are present in the content of an image. Detected faces then have their features extracted by a convolution neural network model, pretrained on a large set of faces and built on top of the Keras API using the Tensorflow backend. Face features are then clustered together according to their similarity using a hierarchical clustering algorithm, generating many cluster groups with different numbers of clusters, corresponding to the hypothetical number of faces in a group. Cluster groups with higher silhouette score correspond to stronger and more probable hypotheses, which are then shown to the user as a way of further enhancing the management of the user's photo library.

Looking at the user's opinions for improvement and the limitations of the system, future work should include the inclusion of meta-data to the photos, such as titles, location or the date, which might further help the searching feature. In the scope of further enhancing the image retrieval, which is limited by the tags extracted by the Google Cloud Vision API, query and tag expansion techniques should be a priority, as the low number of tags and information can generate additional relevant information when adding similar or related keywords to both the tags and the query.

References

[1] Django. https://www.djangoproject.com/.

[2] Google Cloud Vision API. https://cloud.google.com/vision/.

[3] The Django Book - The Model-View-Controller Design Pattern. https://djangobook.com/model-view-controller-design-pattern/.

[4] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] M. Ames and M. Naaman. Why We Tag: Motivations for Annotation in Mobile and Online Media.

[6] H. Bay et al. Speeded-Up Robust Features (SURF). Technical report.

[7] C. Cakebread. 1.2 trillion photos to be taken in 2017, thanks to smartphones: CHART - Business Insider, 2017. https://www.businessinsider.com/12-trillion-photos-to-be-taken-in-2017-thanks-to-smartphones-chart-2017-8.

[8] A. Dempster et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B Methodological*, 39(1):1–38, 1977.

[9] J. Eakins and M. Grahan. Content-based Image Retrieval. (October), 1999.

[10] A. Fahad et al. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, 2(3):267–279, Sept 2014.

[11] K. He et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] H. Hosseini et al. Google's cloud vision API is not robust to noise. *CoRR*, abs/1704.05051, 2017.

[13] C. Johnson. My Initial Impressions of Google's Cloud Vision API – Digital Experience Platforms, 2016.

[14] Y. Liu et al. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 2007.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[16] A. Mann and K. Navneet. Survey Paper on Clustering Techniques. *International Journal of Science, Engineering and Technology Research*, 2(4):2278–7798, 2013.

[17] J. Nielsen. Usability inspection methods. In *CHI 95 Conference Companion*, 1994.

[18] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 2001.

[19] O. M. Parkhi et al. Deep Face Recognition. In *Procedings of the British Machine Vision Conference 2015*, pages 41.1–41.12, 2015.

[20] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] A. S. Razavian et al. CNN Features off-the-shelf: an Astounding Baseline for Recognition. mar 2014.

[22] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] Statista. Global smartphone user penetration 2014-2021, 2017.

[24] C. Wengert et al. Bag-of-colors for improved image search. pages 1437–1440, 2011.

[25] W. Zhou et al. Recent Advance in Content-based Image Retrieval: A Literature Survey. 2017.