U.PORTO

FEP FACULDADE DE ECONOMIA
UNIVERSIDADE DO PORTO

# Image visual similarity with deep learning: application to a fashion e-commerce company

by

Rui Pedro da Silva Rodrigues Machado

Dissertation for achieving the degree of

**Master in Data Analytics**

Dissertation supervisor: Professor João Manuel Portela da Gama

**June 2017**

# Biography

Rui Pedro da Silva Rodrigues Machado was born on June 13th, 1989 at Porto Portugal. He is an Information Systems manager, entrepreneur and highly oriented for innovative solutions, always looking for technical and business solutions to fix all sort of problems. Over five years of experience in the design and implementation of business intelligence and analytics solutions, leadership and defining the vision of data oriented teams.

He has a background in e-commerce, working with multiple technological and e-commerce platforms in the field of analysis, business intelligence and search optimization. He is currently working as Head of Business and Artificial Intelligence at the HUUB company.

# Abstract

Deep learning is a very trendy topic now, showing high accuracy in image based systems that can go from image segmentation to object detection and image retrieval. Because of this, multiple researchers and companies have been building and sharing work in the community, including pre-trained convolutional neural networks, available for public use.

This work follows the trend and delivers an experimental study, comparing three different deep neural network architectures and six distance indexes in a real-world image retrieval problem, using real data from a fashion e-commerce platform from Morocco.

After testing all the different combinations, we can conclude that for this dataset, Vgg19 combined with a correlation coefficient for similarity calculation is the tuple that best maximizes the similarity between a search image and its retrieved neighbors.

**Keywords:** Deep learning; Image-based search; convolutional Neural networks; fashion image retrieval

# Table of Contents

# Table of figures

# 1 Introduction

This section contains the introductory aspects of the thesis, specifically the motivation for choosing the theme, the main proposed objectives for the construction of the visual search engine, the proposed methodology to be used and the problem description to be solved.

## 1.1 Motivation

Setting up and maintaining an e-commerce company is getting easier every day thanks to the current easiness of buying and selling products from and to anywhere in the world but also to the democratization of technologies that allow an entrepreneur to go from zero to a working solution in a few days if not hours.

On top of this, existing fashion brands are migrating and creating their own online stores due to the fact that customers are increasingly spending more time online and thus creating the opportunity to use social networks and apps to be continuously connected to them and perform marketing one-on-one campaigns with user-oriented content. (Cecilio, 2015)

However, when analyzing the way these platforms allow people to search for fashion items in their catalogs it's possible to realize that it's still a primitive task with most websites only supporting basic text-based querying.

While sometimes describing a product is an easy task, when seeing a shirt with a peculiar pattern or an innovative texture, translating it into words, might result in a frustrating operation that will lead the customer to quit and consequently decrease a shop's conversion rate.

This, sustained by the studies of (Goswami, Chittar, & Sung, 2011; Wei Di, Neel Sundaresan, Robinson Piramuthu, 2014), leads to the hypothesis that if a fashion e-commerce shop upgrades its website and platform to support image based search and navigation, where users can use photos containing products to search for other visually similar products, or click on a product image and navigate to visually similar ones, it will affect their conversion and bounce rates as users quicker and more interactively find the items they were looking for.

Image-based search engines can also be the missing link between online and offline fashion shops and not only help companies optimizing their conversion accuracy, as also allow customers to go to a physical store, take a picture of a product they like and quickly compare online, prices of other visually similar products.

## 1.2 Objectives

Using deep learning models, previously trained over large datasets of images, such as ImageNet or FashionNet build a visual search engine, that will be able to, given a query image as an input, search and retrieve the most similar images in terms of visual features from the companies' catalog. This engine will be tested against the online catalog of a fashion e-commerce company from Morocco.

As a secondary objective, this works will provide an empirical comparison of the usage of multiple deep learning architectures, delivered through different pre-trained convolutional neural networks, as well as the impact of different image preprocessing techniques and similarity measures in the quality of the obtained models.

## 1.3 Questions of study

This work will combine multiple deep learning architectures for visual feature encoding that will later be used for visual similarity calculation. This structure will allow answering the following questions:

- How can deep learning be applied to the construction of a visual similarity based image retrieval system for a fashion e-commerce platform?
- What similarity measures best fit the fashion e-commerce catalogs for image retrieval?
- Is deep learning for image retrieval an enterprise option, available to any company or still a research branch?

# 2 Literature review

This chapter begins by giving some initial definitions and historical context regarding image retrieval as well as main challenges researchers faced since the first appearance of such technologies.

After, it's presented an overview of some main methods used for image retrieval. Consequently, Deep Learning is selected and presented in this chapter as the most relevant one for this work and different architectures for implementing it are demonstrated. Finally, some applications of deep learning for image retrieval in the fashion industry are checked and the main results obtained compared.

## 2.1 Image retrieval

Image retrieval can be defined as the task of searching for images in an image database. This is not a novel concept, instead, it has been researched since the 1970s with combining database management and computer vision, which study the topic from two different perspectives, the first being text-based and the second one visual-based. (Young Rui, Huang, & Chang, 1999)

Initially, the systems were composed only by manual annotations that were stored in a database to perform the retrieval step, however, when the size of the image collections started to increase the effort required to label them was unsustainable. Motivated by this problem, in the early 1990s content-based image retrieval was proposed. Since then many research lines have appeared using one or the other isolated or combining them. (Yong Rui, Huang, & Chang, 1999)

### 2.1.1 Typical image retrieval system

In the image retrieval community, text based features and visual features, when both available, are not a replacement for each other but rather complementary. Taking this into consideration, a typical image retrieval system, will all its components is shown in Figure 1 Typical image retrieval system. (Yong Rui et al., 1999)

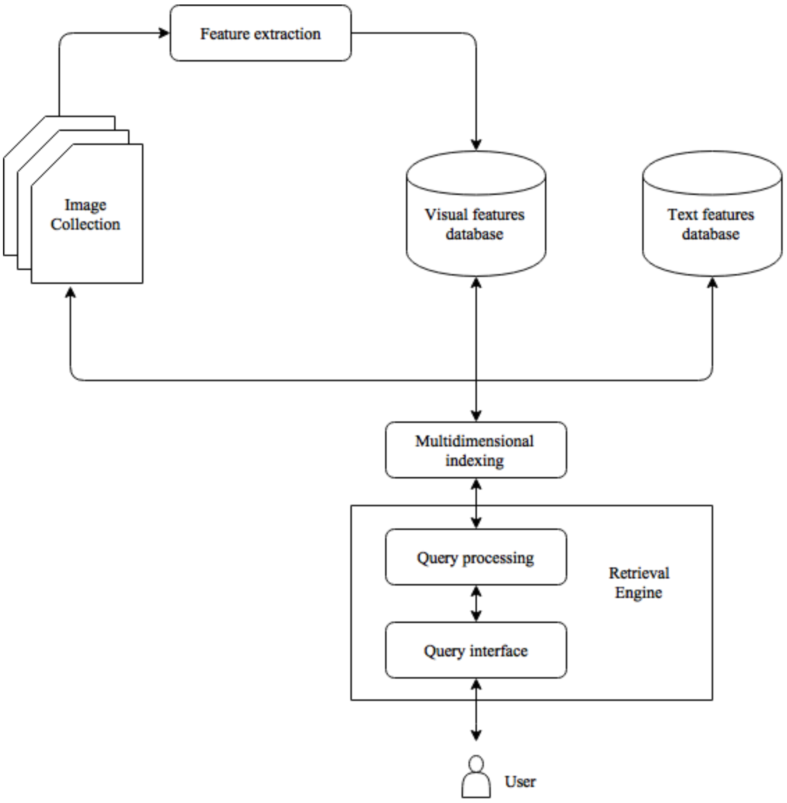


Figure 1 Typical image retrieval system

In the previous figure, we can segment the basic organization of an image retrieval system in two big groups. The first is the feature extraction module, where we define a set of techniques to extract visual features from an image collection and index them in a database that will further be used for retrieval. This step can also be combined with text based features that enrich the search process. This is a content-based combined with the text-based process of feature engineering applied to images, that typically happens offline, meaning it's used to train models that will further be deployed.

The second big group is the retrieval module, where we create the online environment, meaning the process of a user, in real time, submitting an image or text to be used as an input to the search process, query the database for similar image retrieval and result in the ranked result images.

Using the combination of these two modules with its components multiple applications can be developed on top of it. These involve random browsing of images in a given catalogue, search by example where we give an input image and retrieve a set of similar ones, search by sketch, similar to the previous one but instead of an image we provide a simple sketch, search by text that uses sentences or words to find images that contain in a certain way a reference to it and lastly navigate through customized image categories. (Yong Rui et al., 1999)

### 2.1.2 Types of image retrieval systems

As previously mentioned, text and content based methods are the best well known in the scientific community. However, there is another stream of methods, denominated semantic ones, that intent to solve an identified gap between high-level and low-level features in images, that is what image features can distinguish and what people perceive from the image. An overview of these families of methods is given in the following sub-chapters. (Zhao & Grosky, 2002)

Taking these three types of families in consideration, (Alkhawlani, Elmogy, & Bakry, 2015) suggested, as can be seen in Figure 2 Image retrieval categories, an organization for image retrieval methods.

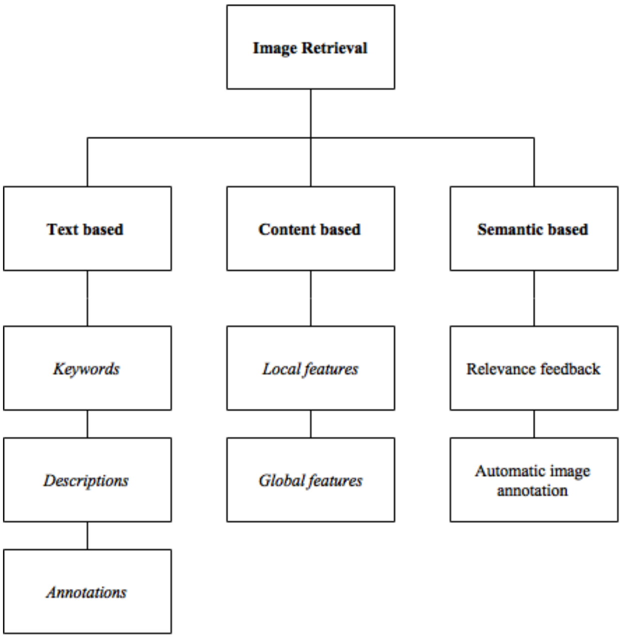


Figure 2 Image retrieval categories

### 2.1.3 Text based methods

Text based image retrieval systems are those that use manually inserted annotations, keywords or descriptions, combined with metadata like image name, the size of dimensions to store information regarding a given image in a database. When a new search is to be done, a user creates a search criterion with filters that will retrieve images satisfying that search.

Although conceptually easy to implement these methods are not scalable in situations where the image collection is big or the users consuming the engine speak different languages, requiring in such cases translations for each annotation.

Another drawback is that they create a gap between what the interpretation of the visual features of an image by the annotator and the interpretation of the user querying, which might have a different one. (Alkhawlani et al., 2015)

While, as previously mentioned, both text and content-bases methods are complementary for an IR search engine, in some cases text features due to their handcrafted nature are not available or are wrongly inserted in databases, forcing the engine to rely only on visual features. Content based image retrieval methods try to solve this issue.

### 2.1.4 Content based methods

Content-based image retrieval system is the process of querying and retrieving similar images from a given data source, such as an e-commerce catalog. The similarity in these systems is defined in terms of colors, textures and shapes identified in images using a visual feature extraction algorithm and a similarity metric to assess with ones match the queried one. (Chaudhari & Patil, 2012; Huu, Thu, & Quoc, 2012)

As previously mentioned, this family of methods, without questioning its accuracy and efficiency, typically suffer from the so called semantic gap. That is if a user what’s to search for images of “turtles in a swamp”, these systems won´t be able to map this human concept in a set of visual features that can be used to query a CBIR system.

In order to solve this gap, researchers are now suggesting the integration of deep learning in content-based image retrieval as a way to overcome the semantics of images and fix this problem, by being able to work with so called regions of interest with are given as input to deep learning models using bounding boxes that represent regions with concepts in an image. (Manzoor & Balubaid, 2015)

On top of it, multiple surveys and benchmarks have been done by researchers and recent conclusions tend to suggest that when it comes to content based image retrieval, the latest generation of deep learning methods in particular convolutional neural networks outperform by a large margin the shallow methods. (Chatfield, Simonyan, Vedaldi, & Zisserman, 2014)

### 2.1.5 Semantic methods

As a complementary explanation to understand semantic methods, these try to learn a high-level feature of an image using low-level features, that is using color, texture, shape, edges and any other already mentioned content-based features, automatically infer textual annotations that describe what a given image contains. This set can then be stored in a database, and a query system can be built on top of it, for based on textual words retrieve images that relate to that semantic concept.

These systems rely on ontologies based image retrieval that is more focused on capturing semantic content, mapping image features to concepts, because this can help in satisfying user requirements in a better way (Manzoor & Balubaid, 2015). In other words, we can input to model both images but also meta data about was is present in the image so that the model can learn to correlate visual features with semantic concepts. As is was previously stated Deep learning is one methodology that can be used for building such systems. The properties and relationships between concepts of ontologies can assume multiple formats such as Individuals, Classes, Attributes, Relations, Function terms, Restrictions, Rules, and Axioms.

## 2.2 Deep learning and Convolutional Neural Networks

Deep learning is a branch of machine learning composed by algorithms that attempt to represent high-level abstractions of data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations. Using this field of knowledge, it is possible to given an input image, extract the set of numeric features that represent its combination of color, texture and shape representations. (Bengio, 2009)

Different deep architectures have been used for solving state-of-the-art speech recognition, visual object recognition, object detection and image retrieval problems. (LeCun et al., 2015)

An instantiation of deep learning is the so called deep convolutional neural networks that have brought improvements in processing images, video, speech, and audio. These typically called convnets are functions for processing images, consisting of a number of layers, placed in sequence, such as convolution, pooling, and rectification, where the parameters of each stage are learned to optimize performance on some task, given training data. (Chatfield et al., 2014)

This concept of convnets was introduced in the 1960s however its processing times were too high to make this an attractive and useful technique. Recently, with the introduction of graphical processing units in video cards, training this model became much faster are now proving its value in different fields of computer vision.

The way convnets learn features from an image follow the sequence of layers it contains, where the first layer of representation will learn high-level, abstract features like edges and the deeper we go, the lower will be the features learned. We can go from edges to topics by spotting particular arrangements of edges deeper combinations that would correspond to parts of familiar objects or whole objects as combinations of these parts. (LeCun et al., 2015)

### 2.2.1 Convolution layer

These nets differentiate from a normal artificial neural network on the way they search for objects in an image. They allow changing the paradigm from feeding a full image as a grid of numbers but instead, break the image into equally-sized tiles and learn visual features from each by applying visual filters.

The learned features are then the consequence of applying the so-called filters, which are learned by applying a mathematical convolution between the filter and the input image. This operation consists in computing element wise multiplication (between the two matrices) and sums the multiplication outputs to get the final integer which forms a single element of the output matrix. This step is called convolution and when training a convnet, it requires as parameters the number of filters it will use and its size and stride.

Filters work as learners of the presence of specific characteristics in images. When applied at the early stage convolutional layers they allow detection of low-level features in images while layers at early stage detect specific characteristics which represent high-level features with a higher semantic meaning.

Filters are not more than matrixes of pixel values that come with a predefined size and depth and are multiplied against the input image in a certain stride cadence in order to learn the presence or not of these certain characteristics. They are learned automatically, although often they start with a random filter matrix they are able to adapt and improve in multiple dimensions such as color, shape, and texture, and other learnings related to pattern angle and opacity, for example, depending on the most prominent aspect of the patch being analyzed, by optimizing the filter values with the so-called backpropagation technique. (Wan et al., 2014)

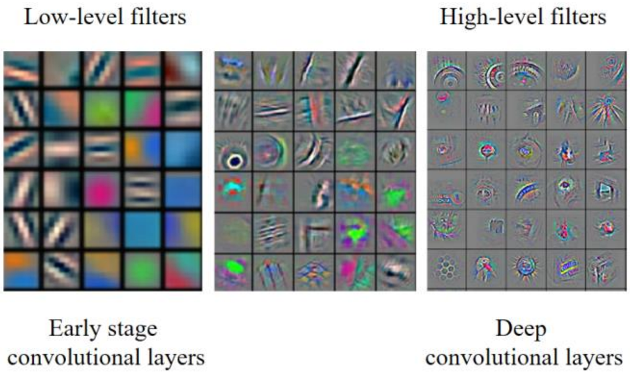


Figure 3 Example of learned filters on a given image

Figure 4 Example of convolution applied to a simple image, with one filter with size 2x2 pixels and a stride of 1 pixel at a time.

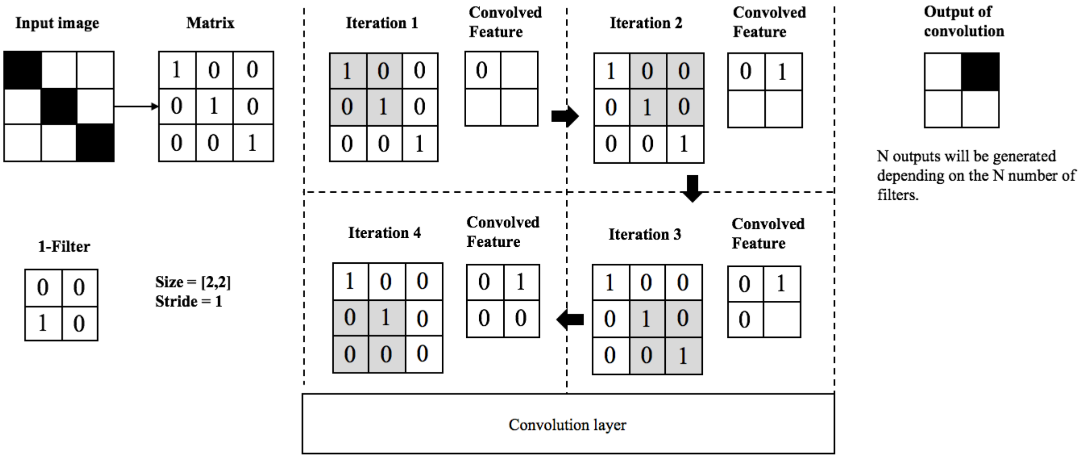


Figure 4 Example of convolution

The convolution step allows to both learn features within these tiles and highlight those that reveal an interesting pattern. In the previous figure, it was possible to identify tiles that contain a predefined texture shape however these filters can be any of the ones mentioned for content-based image retrieval, for color, texture or shape feature learnings such as the edge detectors. (Wan et al., 2014)

### 2.2.2 Pooling layer

After applying convolutions in images, while training the convnet, to optimize the learning process, it’s important to reduce the size of the target matrix and eliminate redundant and noisy convolutions. For that, we apply a step called max-pooling. These steps consist of finding the max value in each grid square (output of convolution filters) in the target matrix, which will isolate the inner sections of the image highlighted by the convolution step.

In Convolutional Neural Networks, a pooling layer is typically present to provide invariance to slightly different input images and to reduce the dimension of the feature maps. Inside these methods, max pooling is preferred as it avoids cancellation of negative elements and prevents blurring of the activations and gradients throughout the network since the gradient is placed in a single location during backpropagation.

The spatial pooling layer is defined by its aggregation function, the high and width dimensions of the area where it is applied, and the properties of the convolution (e.g. padding, stride). (LeCun et al., 2015)

### 2.2.3 Rectified Linear Unit layer

It's important to mention that these nets still require activation functions or thresholds to activate or not a given neuron. Traditional approached like the sigmoid or hyperbolic tangent are valid options but the most typical one is image processing problems is the ReLu, rectified linear unit, a technique that uses the value of the output as the activation value if it is positive otherwise will transform it to zero.

The rectifier activation function is usually preferred due to their efficiency in terms of computation without affecting the quality of the accuracy. (Wan et al., 2014)

### 2.2.4 Fully Connected layer

Fully connected layers are the latest layers of a convolutional neural network, coming before the output layers and the task is to deliver the proper computations for the class scores. This is the equivalent to have an ordinary neural network at the end of the convolutional steps so that each neuron in this layer will be connected to all the numbers in the output layer of the previous layer. (Wan et al., 2014)

These FC layers (along with the deep learning model as one) are trained with gradient descent so that the class scores that the Convnet processes are consistent with the labels in the training set for each image. Because the output of these nets is an array of size 1x1x[Number of output units], is simplifies the handling of the learned parameter for further project developments such as image retrieval as instead of manipulating tensors, we manipulate one-dimensional arrays. (Wan et al., 2014)

### 2.2.5 Convnet Architectures

These three steps, convolution, pooling, and ReLu, can be combined multiple times in sequence and thus, as previously mentioned, improve the findings and features the convnet is able to learn. This possibility has been leading to different architectures being studied and used by the community.

LeNet, a 7 layers convolutional neural network is still one of the most famous ones, due to the improvement of handwritten digits recognition and for being one of the first architectures to prove the value of convnets in image processing problems, but other more recent networks have been using for deep learning with improved results. (He, Zhang, Ren, & Sun, 2015)

Amongst many, we can find VGG-16, a 16-layer CNN, VGG-19, a 19 layer CNN, both developed by the University of Oxford Visual Geometry Group (VGG) (Simonyan, Andrew Zisserman, & Zisserman, 2015) and also ResNet50, a very deep convolutional neural network with 50 layers, created by Microsoft, that won the ILSRVC 2015 competition and surpass the human performance on the ImageNet dataset. (He et al., 2015)

These three are currently available as pre-trained convolutional neural networks in multiple programming libraries which have been leading researchers to rely on them as the foundation for deep learning studies.

Although convnets are more famous for their object recognition and location purposes, image retrieval and visual similarity as also been an application field of such technique. By extracting the features learned by the net before it reaches the output layer, it’s possible to index to each image an array of visual features that characterize its contents. It’s important to mention that for this step we should use the same convnet, previously trained to extract features from the input image and the search catalog.

After creating the database of indexed visual features per image, we unblock the possibility to apply a similarity metric and thus retrieve all images, that are similar in terms of visual contents to an input one, building an image retrieval engine. (Wan et al., 2014)

### 2.2.6 Convnet Training Methods

As mentioned by (Wan et al., 2014) there are three options when it comes to choosing how to adopt a convolutional neural network:

- The first option is if the data used to train the convnet is from a similar domain to the problem under study and simply use a pre-trained one;
- The second option, is to use a pre-trained convent but on top of option 1, evolve the training with some domain specific images, leveraging all the previous learning to converge on our problem under study;
- The last option would be retraining the convolutional neural network with a new dataset that would fit better the domain of the problem under study. As we can infer, the similarity of the domain of the training data to the domain of the problem and thus target data will define the strategy to use in terms of convnet usage.

## 2.3 Pinterest architecture for deep learning based image retrieval

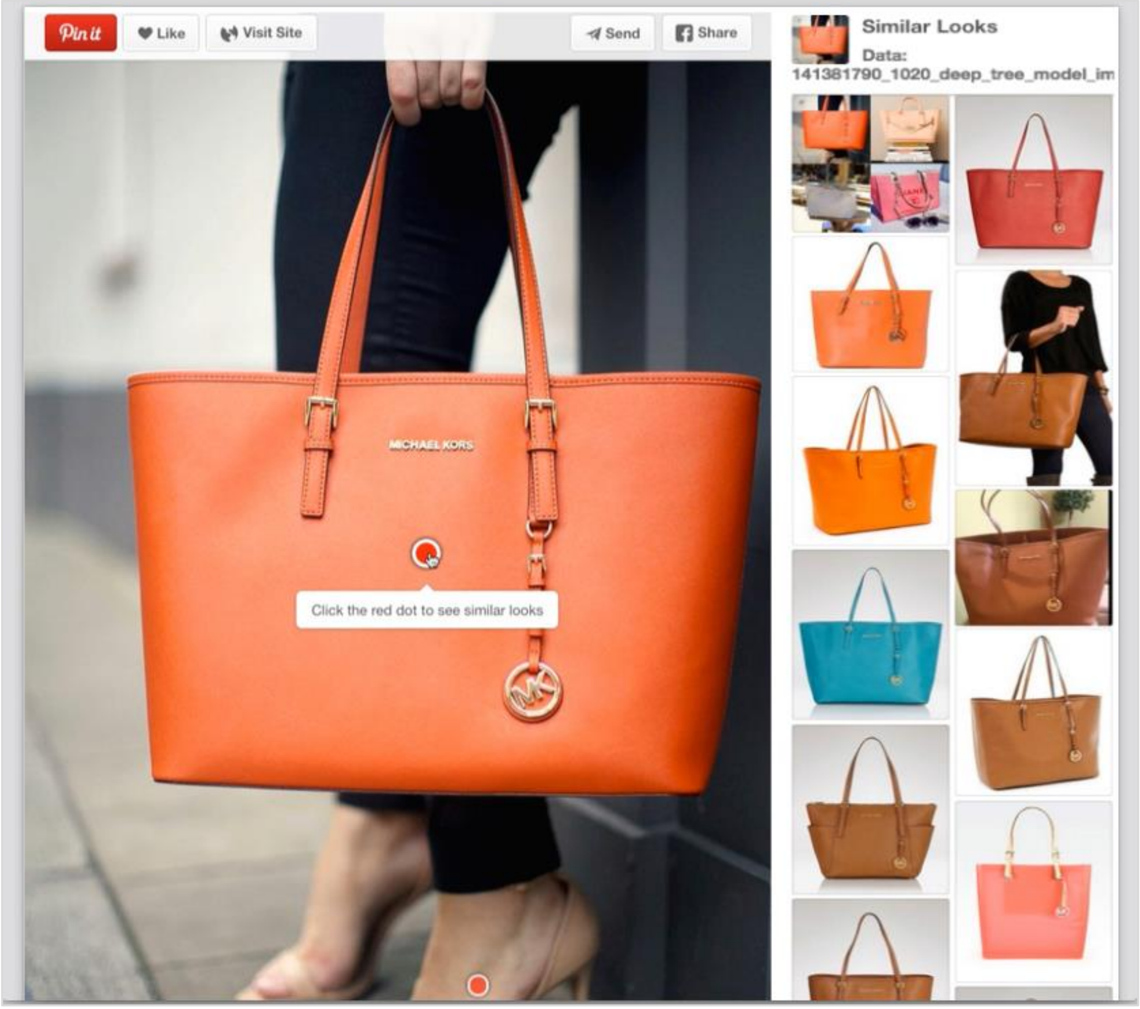
As previously mentioned, multiple solutions have been addressed to tackle the image retrieval problem. Although recent, the usage of deep convolutional neural networks, have been increasingly adopted, mainly caused by recent studies that show evidence of an increased value in computer vision when compared to other methods.

Inside the computer vision problem, the closest system to the problem under study is image retrieval engines, mainly directed toward image classification that use however deep convolutional neural networks to extract and encode visual features from images. There have been already some developments in this area and this chapter gives an overview of a set of published papers related to image retrieval with deep learning in the fashion industry.

The visual bookmarking tool Pinterest decided to invest in learning visual features that could allow the recommendations based on the user's visual preferences. They developed a prototype of a visual search engine, that uses convolutional neural networks, to allow users to click on automatically tagged objects such as bags and shoes to view similar-looking products in other pins using both images and curated information about the products, making this a semantic image retrieval system.

The proposed architecture of Pinterest (Jing et al., 2015a) covers not only the visual search problem but also the object detection and location one as they are related when it comes to image retrieval. For this work only the visual similarity was studied and for such the process was using a convolutional neural network to extract and encode visual features vectors from images and lastly apply a distance metric to compute visually similar ones.

While developing their system, Pinterest has concluded that fully training it to learn a good representation can be time-consuming and requires a very large training set. This way they have decided to use a pre-trained convnet, available online, from which they retain low-level features extracted from different depths of the net. It's interesting to mention that in March 2017 Samsung launched its new smartphone Galaxy S8 which incorporates an AI based assistant called Bixby. This assistant comes with a visual search engine that uses Pinterest architecture and research on image based retrieval systems. (Yeung, 2017)



Click the red dot to see similar looks

Figure 5 Pinterest semantic image retrieval system

Figure 6 Abstract architecture of the Pinterest solution, illustrated the architecture of the solution assembled by Pinterest for an image retrieval system.

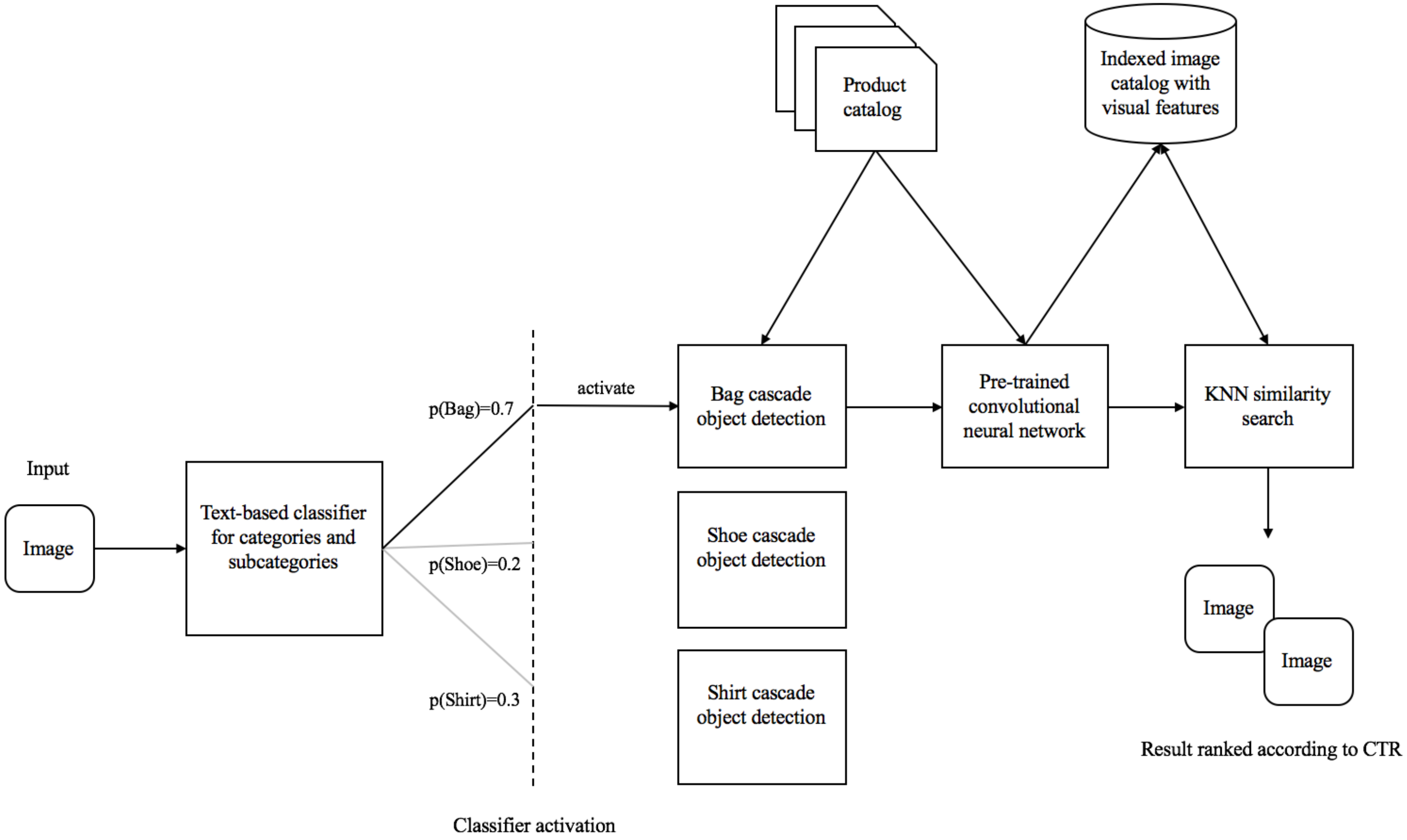

Product catalog
Indexed image catalog with visual features
Input
Image
Text-based classifier for categories and subcategories
p(Bag)=0.7
p(Shoe)=0.2
p(Shirt)=0.3
activate
Classifier activation
Bag cascade object detection
Shoe cascade object detection
Shirt cascade object detection
Pre-trained convolutional neural network
KNN similarity search
Image
Image
Result ranked according to CTR

Figure 6 Abstract architecture of the Pinterest solution (Jing et al., 2015a)

Because of the simplicity of usage, selecting a pre-trained net is a good option that will depend however on available domain related ones. An example of an existing one specific of the fashion domain, came from Liu, Qiu, & Wang (2016) that when presenting their fashion data set DeepFashion, a large-scale clothing data set with over 800,000 images, and to prove its value, developed a deep model, named FashionNet.

This model trained with convolutional neural networks learn clothing features by jointly predicting landmark locations and attributes, available in this public data set. Being trained specifically for Fashion it potentially increases its accuracy for Fashion problems when compared to VGG16 or LeNet (Chatfield et al., 2014).

The last step in an image retrieval system is the similarity computation. Pinterest started by indexing the visual features learned with the convnet in a distributed computing platform for fast retrieval, Hadoop, where each image has an associated feature vector. After that, a K-Nearest Neighbor method is used to retrieve the most visually similar images, where different similarity measures[1] have been studied and presented for the community. (Jing et al., 2015b)

As mentioned by (Chechik, Sharma, Shalit, & Bengio, 2009), visual similarity methods can go from learning a Euclidean, Minkowski, Hammington, Chebyshev or Manhattan distance, to using the cosine similarity (Iglesias & Kastner, 2013). Chechik, Sharma, Shalit, & Bengio, (2009) introduced their own method called OASIS that learn in an online matter, a linear model of similarity between images.

Also, (Sergy, 2008) and (Goshtasby, 2012) has delivered research on such topic, however following the same logic of using unsupervised KNN as a means to calculate image similarity. It suggests the use of distance metrics such as the Euclidean distance, Manhattan or the inner product of two vectors.

---

[1] In similarity measures, high values denote high similarity between elements.

Table 1 Distance measures available for image feature vector similarity, demonstrates the formulas and definitions for the mentioned similarity measures as presented by Iglesias & Kastner, (2013).

| Name | Formula | Description |
|---|---|---|
| Manhattan (City Block) | $d_{CB} = \sum_{i=1}^{d} \| P_i - Q_i \|$ | Sum of the absolute values of two points. (L1 Minkowski order 1) |
| Euclidean | $d_{Euc} = \sqrt{\sum_{i=1}^{d} \| P_i - Q_i \|^2}$ | Square root of the sum of the squared distances between points (L2 Minkowski order 2) |
| Chebyshev | $d_{Cheb} = \max_{i} \| P_i - Q_i \|$ | Max distance between points along any coordinate dimension. ($L_\infty$ Minkowski order ∞) |
| Hammington | Compare the first two bits in each string. If same, record a "0", else “1” | Sum of differences between two binary strings. |
| Cosine | $s_{Cos} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sqrt{\sum_{i=1}^{d} P_i^2}\sqrt{\sum_{i=1}^{d} Q_i^2}}$ | Measures the cosine of the angle between two vector points. |

Table 1 Distance measures available for image feature vector similarity

# 3 Methodology

This chapter will give an overview of the proposed deep learning based architecture to build a visual similarity engine for a Morocco fashion e-commerce company, respecting the functional requirements that were listed by them.

## 3.1 Development Workflow

Inspired by the recent studies on convolutional neural networks and their high efficiency and accuracy of learning visual features about images plus the capacity that they brought to overcome the existing semantics gap on CBIR engines, this work will follow the use of convnets for the task of visual feature extraction. Multiple architectures of these will be used to assess the one that achieves a higher accuracy. The programming language to be used will be python due to the recent machine learning libraries that have been released and that ease the implementation of such systems.

The proposed steps to build the visual search engine, a consequence of the state of the art demonstrated before and highly influenced by the Pinterest architecture are the following:

1. Extract visual features from the training data (search catalog) using a pre-trained convolutional neural network;
    a. Pre-process images so that all have the same size;
    b. Extract visual features from the last fully connected layers;
2. Index the feature vectors in a multidimensional database organized based per category classification obtained in the previous step;
3. Build a simple search engine to preprocess an input query image and extract the visual features of it using the same pre-trained convnet from step 1.
    a. Pre-process image so that it has the same size of the training data;
    b. Extract visual features from different depth layers in the convent;
    c. Calculate the similarity with target images of the same category using multiple measures;
    d. Show the most visually similar images;
4. Evaluate results

## 3.2 High-level solution architecture

The basic idea of this project is to use deep learning as the method for describing images in terms of visual features, then store the results of extracting the feature vector of the fully connected layer before classification, in a database so that they can be used for retrieval. After this process, using a similarity measure, compare a new image as input with the previously trained data, saved results, and extract the top five similar ones.

To illustrate the proposed system architecture that illustrates the previous methodology, inspired by Pinterest, Figure 7 Proposed system architecture based on Pinterest architecture shows the offline environment, used for extracting visual features from the training dataset and storing them in a database as well as the online environment, used to retrieved similar images are given a new image as input (search query).

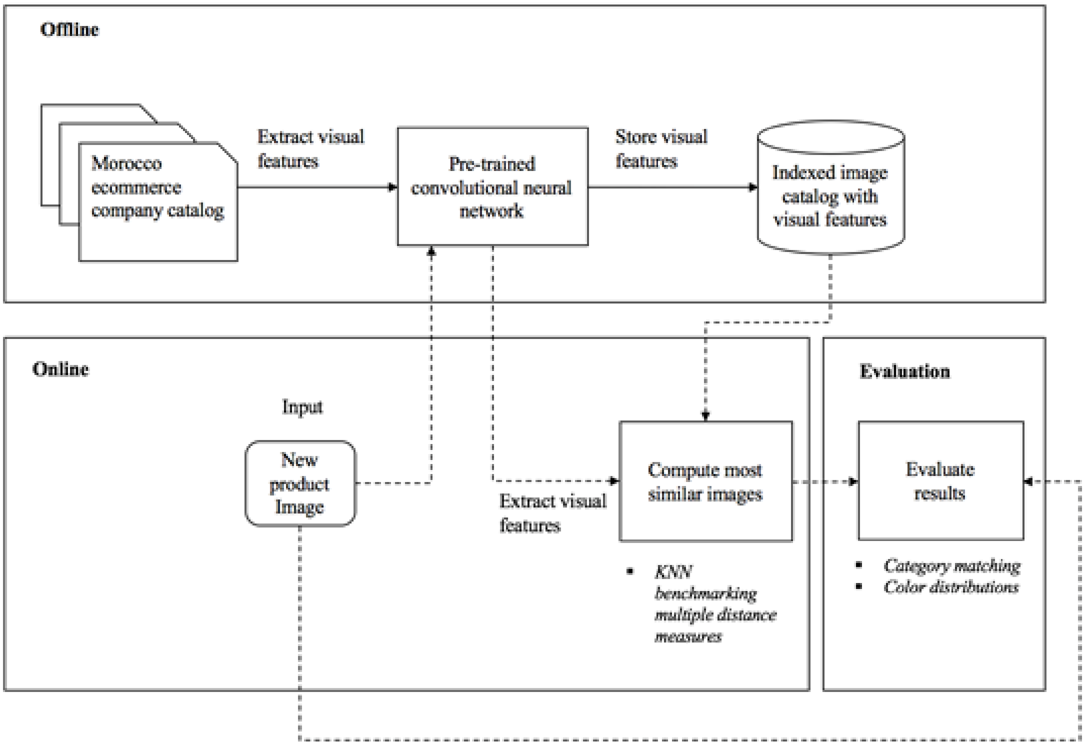

Offline
Morocco ecommerce company catalog
Extract visual features
Pre-trained convolutional neural network
Store visual features
Indexed image catalog with visual features
Online
Input
New product Image
Extract visual features
Compute most similar images
KNN benchmarking multiple distance measures
Evaluation
Evaluate results
Category matching
Color distributions

Figure 7 Proposed system architecture based on Pinterest architecture

## 3.3 Technology Stack

With the previously described architecture as a baseline, in this chapter, the main technologies and development lines will be explained. Overview wise, the main technologies used to build and deploy this deep learning based image retrieval system were the following:

- Python[2] as the main Programming Language, orchestrating the whole logic;
- Keras[3], a simplified wrapper to rapidly build, test, and deploy deep learning architectures, built around more complex numerical computation engines such as TensorFlow and Theano that abstracts the user of the complexity of building a deep neural network. For this project, Theano was selected[4];
- SQLite[5], an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

To support minor operations like string manipulation, data frames or CSV files, other Python libraries were used. All of them are available out-of-the-box, by installing the data science package manager Anaconda[6] on top of python.

Using the architecture presented in 3.1 as a base line for the system implementation, both offline, online and evaluation components were developed and combined they form the architecture that can be seen in the following Figure 8 Final result of the implemented system. In this hybrid diagram the main use cases available actor is detailed, namely:

- Offline user:
    - Run offline feature extraction for the train data (Fashion items catalog);
    - Run offline feature extraction for the test data (Evaluation);
    - Run evaluation of the system
- Online user:
    - Retrieve similar images to a given input;

---

[2] https://www.python.org/
[3] https://keras.io/
[4] http://deeplearning.net/software/theano/
[5] https://www.sqlite.org/
[6] https://docs.continuum.io/

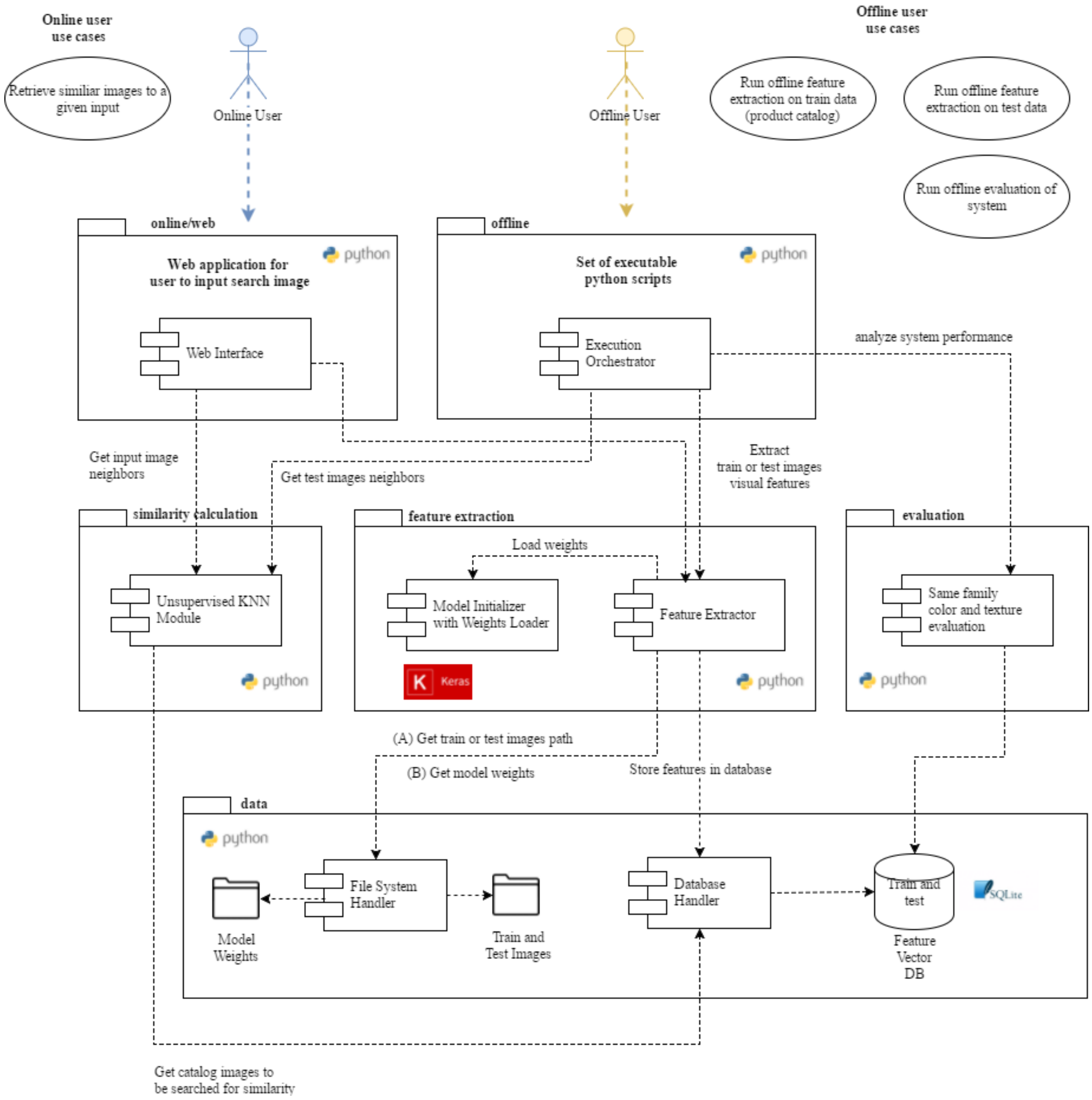


Figure 8 Final result of the implemented system

Regarding the feature extraction, similarity, and evaluation packages (as seen in the previous Figure 8), the following subchapters will detail the implementation process for each of them.

# 4 Implementation

## 4.1 System requirements

In this section, the functional requirements acquired and agreed with the fashion e-commerce company, for building its deep learning based image retrieval system are described. On top of the company requirements, some others were added by the researcher to be able to understand the model behavior in deeper detail. Table 2 Requirements listed by the e-commerce company, lists the requirements acquired from the company.

| **ID** | **Source** | **Name** | **Description** |
|---|---|---|---|
| R01 | Company | Application | As a user, I want to be able to interact with the image retrieval system through a web application that allows me to input an image and see the results on a single page. |
| R01 | Company | Visual feature extraction | As a user, I want to be able to give a set of product catalog images, be able to extract these features as a float vector and store them in a database for further mining. |
| R02 | Company | Image similarity | As a user, I want to be able to give a new input image be able to mine the catalog feature vector database and retrieve the top 5 most visually similar images. |
| R03 | Company | Evaluation | As a user, I want to be able to understand the product category and sub category accuracy of the returned similarity results. |

Table 2 Requirements listed by the e-commerce company

Table 3 Research requirements for evaluation and results analysis, complements the requirements listed by the e-commerce company with a set of others that allow a better comprehension of the system performance.

Since the company requires its evaluation method to be based on the product tree, namely category and sub category accuracy between the input image and retrieved ones, the following requirements focus on texture and color correlation. The product tree gives a better business evaluation perspective since the company is interested in always suggest images that come from the same product branch of the input image and not so much with the color or texture correlation.

By looking at the product tree we are in a certain way evaluating the shape of the input and output images since an image of the sub category t-shirt will always have the same shape of the other t-shirts. However as previously mentioned, the traditional content-based image retrieval systems focus on shape, color, and texture and following this mindset we believe to be useful to evaluate these three dimensions from the deep learning method perspective as well. This way, besides categories analysis as a shape descriptor it's also interesting to add from the research perspective evaluation requirements texture and color. With this in consideration Table 3 Research requirements for evaluation and results analysis, adds these two evaluation dimensions to the system.

| **ID** | **Source** | **Name** | **Description** |
|---|---|---|---|
| R04 | Researcher | Evaluation | As a researcher, I want to be able to understand the color correlation between the input image and the returned similar ones, to understand how related they are in terms of color. |
| R05 | Researcher | Evaluation | As a researcher, I want to be able to understand the texture correlation between the input image and the returned similar ones, to understand how related they are in terms of color. |

Table 3 Research requirements for evaluation and results analysis

## 4.2 Train and test data

Since the methodology uses pre-trained convolutional neural networks, no training data will be required for such purpose.

On the other hand, for the image retrieval process which includes image similarity based on visual features extracted using the pre-trained convents, a set of images are required for the retrieval process. In other words, whenever a new input image is given, this will be the target data set to retrieve the similar images. From now on this will be named train data.

Also, for testing purposes on the similarity module, a different set of different images from the ones used for training is required so that we can access accurately (without the bias of the training set already knowing the test image) if the similar images suggestion is business relevant. This set will be from now on called the test data.

To build the train and test data images from the catalog of items of a Morocco e-commerce platform were used. Because of confidentially requirements, the name of the company won´t be mentioned.

The train and test data was provided by the company directly and represent around 10% of their entire catalog, sampled across their main product categories using the same factor of 10% of each. Two datasets were provided, one for testing and another for training.

Within the training data set two natures of data were given, one classified as train data, containing only fashion related images, used as a target for the similarity retrieval, but also validation data, non-fashion items, to assess if the category and sub category of the retrieved images are the same as of the input image. This was one of the most import requirements mentioned by the company and its described in Table 2 Requirements listed by the e-commerce company.

On the test data, only fashion items were provided. Their expectation was that all test images should have five automatically retrieved similar images and their category and sub category matching evaluation should be assessed.

| **Domain** | **Category** | **Sub-category** | **# of Images** |
|---|---|---|---|
| Train | Men’s Clothing | Hoodies | 159 |
| Train | Men’s Clothing | Sweatshirts | 229 |
| Train | Men’s Clothing | T-shirts | 14 |
| Train | Women Clothing | Bras | 23 |
| Train | Women Clothing | Dresses | 22 |
| Train | Women Clothing | Nightdresses | 62 |
| Train | Women Clothing | Nightwear | 47 |
| Train | Women Clothing | Pants | 59 |
| Train | Women Clothing | Pajamas | 83 |
| Train | Women Clothing | Sweatshirts | 71 |
| Train | Women Clothing | Traditional wear | 55 |
| Train | Shoes | Shoes | 102 |
| Validation | Car | Car | 28 |
| Validation | Home and decoration | Home and decoration | 94 |
| Validation | Smartphones | Smartphones | 49 |
| Validation | TV | TV | 37 |
| **Total** | | | ***1051*** |

Table 4 Number of train images provided by category and sub category

As described in the previous

Table 4 Number of train images provided by category and , the company provided 12 sub-categories images under 3 product categories, with 4 validation categories. In total 1051 images were given, aggregated in 843 fashion images and 208 non-fashion images.

The following Table 5 Number of test images provided by category and , described for test data, the distribution of images per product category and sub category. As it can be seen, the company provided 8 sub-categories under 4 categories. Furthermore, in total, 97 test images were provided.

| **Domain** | **Category** | **Sub-category** | **Nb. of Images** |
|---|---|---|---|
| Test | Men’s Clothing | Hoodies | 10 |
| Test | Men’s Clothing | Sweatshirts | 22 |
| Test | Women Clothing | Bras | 5 |
| Test | Women Clothing | Dresses | 5 |
| Test | Women Clothing | Sweatshirts | 20 |
| Test | Women Clothing | Traditional wear | 5 |
| Test | Shoes | Shoes | 15 |
| Test | Pants | Pants | 15 |
| **Total** | | | ***97*** |

Table 5 Number of test images provided by category and sub category

## 4.3 Model layers and weights

The process of feature extraction as previously mentioned relies on using pre-trained convolutional neural networks. The way these convnets are shared in communities is by posting online the weights that were produced at the end of the training phase. In our case, the weights of the Vgg16, Vgg19 and ResNet50 architectures are available in the Keras library by default, under a callable object that can be used almost immediately for feature extraction. To speed up the model loading process, the weights can be downloaded and its path can be used instead of the online versions of the weights.

The previous show that Keras is a library that abstracts the user of the effort of building the sequential models and train the networks with training data. However, it's interesting to understand how Keras is organized internally and what hyper parameters can be changed for performance optimization. The deep learning library follows the same logic of layers presented in 2.2 with the addition of orchestration and preprocessing layers that increase the quality of the resulting model.

Table 6 Main layers used in Keras sequential model, summarizes the main layers used to assemble the three architectures, adopted in this work.

| Layer Name | Description | Key Parameters |
|---|---|---|
| Conv2D | Creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs | filters, stride, activation |
| ZeroPadding2D | Add rows and columns or zeros at the top, bottom left and right side of an image tensor | padding |
| MaxPooling2D | A 2D pooling operation on AxB pixels neighborhood, by step size of CxD | pool_size, stride |
| Dense | Fully connect neural network | output_units, activation, name |
| Dropout | Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. | drop_out_percentage |
| Flatten | Flattens the input Without affecting the batch size. | - |

Table 6 Main layers used in Keras sequential model (Source: Keras [7])

---

[7] https://keras.io/layers/about-keras-layers/#about-keras-layers

The Conv2D layer is a convolution layer that uses a filters parameter, being the number output of filters in the convolution, a stride parameter as the strides of the convolution along the width and height of the tensor object and an activation function that can differ from architecture to architecture, but in our selected ones, all fall under RELU activation function. RELU can also be called separately of the convolutional layer by using a special Activation layer.

The ZeroPadding2D is an optimization payer, that adds a padding of zeros around and image to optimize the operation of the filter in a way that avoids losing information. The only input in the padding size (Number of lines of zeros to be added).

The MaxPolling2D is a pooling layer and it works with the purpose of reducing the spatial size of the representation to reduce the number of parameters and computation inside the network, and hence to also control overfitting. Its parameters are the stride, previously explained and a pool size which is a factor by which to downscale the representation.

The Dense layer is a normal artificial neural network that receives a set of input features and learns an activation function that produces "output_units" features using a specific activation function type also required as a parameter. In the Keras architecture, if we want to access a dense layer we must name it, thus we have done it for all the fully connected layers.

The dropout function consists of randomly setting a set of input units to 0 at each update during training time, which helps prevent overfitting. Keras by default uses a value of 0.5 (50%).

Lastly, we have a flatten layer that transforms a multidimensional output into a single vector of features without affecting the batch size. This layer doesn't have any parameter in the Keras architecture.

Keras contains a bigger set of layers that can be used for building a deep learning model but for the used models in this works, only the previous were used. By combining them under a Keras sequence model, we have an empty convolutional neural network that can be fully trained from the ground base or load downloaded weights into it. As previously stated, the latest was the approach followed in this work.

### 4.3.1 VGG16 Structure in Keras

To assemble a VGG16 architecture in Keras we have followed the same approach used for ResNet50, building a sequential model and adding all the required layers.

The resulting structure, as implemented in Keras [8], of such architecture can be seen in Appendices A - VGG16 Structure in Keras.

### 4.3.2 VGG19 Structure in Keras

To assemble a VGG19 architecture in Keras we have followed the same approach used for ResNet50 and VGG16, building a sequential model and adding all the required layers.

The resulting structure, as implemented in Keras [9], of such architecture can be seen in Appendices B - VGG19 Structure in Keras (19 Layers)

### 4.3.3 ResNet50 Structure in Keras

To initialize a ResNet50 architecture, Keras starts by instantiating what it's called a sequential model since a convolutional neural network is such a kind of a model and adding all the required layers.

The resulting structure, as implemented in Keras [10], of such architecture can be seen in Appendices C – ResNet50 Structure in Keras (50 Layers)

It's important to mention that the three models were pre-trained using images from ImageNet[11], an open and free image database. At mid-2010 this database had around 15 million URLs of images which have also been hand-annotated by ImageNet to indicate what objects are pictured plus bounding boxes of such object in around one million of them. (Source: ImageNet[12])

---

[8] https://keras.io/applications/#vgg16

[9] https://keras.io/applications/#vgg19

[10] https://keras.io/applications/#resnet50

[11] http://www.image-net.org/

[12] http://image-net.org/about-stats

After using the previous process to initialize the three models, we have used the model object returned by Keras to load the previously downloaded weights file. After the later, the model is ready to be used to extract visual features.

## 4.4 Extraction and storage of visual features

The extraction and storage of visual features process consisted of building a mechanism, using the Keras library, that could, given a set of images, run the visual feature extractor for each convolutional neural network architecture in use, namely Vgg16, Vgg19, and ResNet50, and store the feature vector in an SQLite database for later be used in the image retrieval step.

The previously described process was applied first to the training data, the sample of the catalog of images of the Morocco e-commerce company with focus on fashion items but also some random images of minor categories to later better assess the quality of the system by comparing the product category of the input images and the same for the suggested similar ones.

After this it was also applied the same logic for the test images, in both cases, the feature vector of every image was stored in the SQLite database, however in different destination tables. The purpose is to ease the process of image retrieval, by accessing the feature vectors in the same place, the target database.

With this flow in mind, the extraction algorithm was implemented with the following steps:

```
get all images in a folder (train or test)
for each model in list(Resnet50,Vgg16,Vgg19) do
    initialize model layers
    load previously downloaded weights
    for each image in the folder do
        pass image through the network
        extract a feature vector
        save in database
```

To better understand how the implemented flow works, the following sequence diagram (Figure 9) exemplifies an interaction example with this module of the search system.

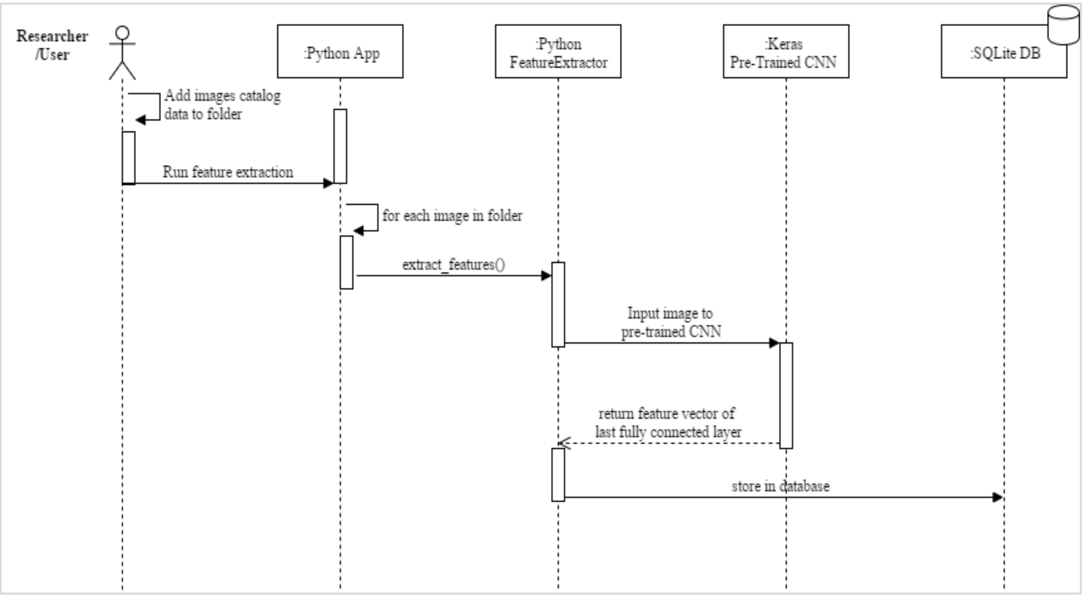


Figure 9 Sequence diagram of feature extraction and storage step

It's also important to notice that, regarding the number of features that were extracted per model, as in 4, it depends on the layer used to extract them. In our case, we extracted the last fully connected layer of each one which means that each VGG16 and the VGG19 feature vector of each image will contain 4096 features (output units defined in the architecture), while ResNet50 will return a feature vector of 1000 features.

After the process ended, this stage outputs a fully populated database, with both trains test images, feature vectors extracted for each of the selected Convnet architectures and target layer, stored as tables and ready to be mined in terms of image similarity.

## 4.5 Image similarity calculation

After the previous step of extracting a feature vector for each train and test data, we could start working on image similarity calculation. For this step and using the same approach created by Pinterest and in consequence the studies of Jing et al. (2015b), an unsupervised k-nearest neighbor strategy was implemented.

Because of the chosen programming language, Python, we also had access to a library called scikit-learn[13], a machine learning library that included brings the nearest neighbor module with both supervised and unsupervised implementations[14].

For this work and due to the nature of the problem where no dependent variable is being predicted, the unsupervised sub-module was selected and used for similarity calculation.

The idea behind is that by using a distance metric we can evaluate how close two data point are close to each other in the feature space, meaning two vectors that are closely related will have a small difference and a large similarity.

Unsupervised KNN, from now on called U-KNN, implementation in scikit-learn allows multiple distance metrics to be used. However using the studies of Jing et al. (2015b) and the research of Sergy (2008) we were able to compile a set of most used similarity measures for image retrieval. Table 7 Distance metrics used in image similarity calculation, describes the metrics selected and applied in this work.

| Metric | Type | Interpretation in Similarity |
|---|---|---|
| Euclidean | Dissimilarity | The lower the value the higher the similarity |
| Manhattan | Dissimilarity | The lower the value the higher the similarity |
| Minkowski | Dissimilarity | The lower the value the higher the similarity |
| Chebyshev | Dissimilarity | The lower the value the higher the similarity |
| Correlation | Similarity | The higher the value the higher the similarity |
| Cosine | Similarity | The higher the value the higher the similarity |

Table 7 Distance metrics used in image similarity calculation

---

[13] http://scikit-learn.org/

[14] http://scikit-learn.org/stable/modules/neighbors.html

Although a deeper description is given to these variables in chapter 2.3, it’s important to complement with information related to image retrieval, namely the fact that Euclidean, Manhattan, Minkowski and Chebyshev distances when applied allow us to evaluate if different to images are being the fact that they are dissimilarity measures while correlation and cosine measures are true similarity ones, allowing us to evaluate how close two images are. Although it’s just a matter of semantics it affects the way we read the value of them. (Goshtasby, 2012)

The flow for similarity calculation started by training a new U-KNN model with the training set containing the feature vectors extracted from the training data detailed in Table 4 Number of train images provided by category and . With this flow in mind the U-KNN training algorithm was the following:

```
set train_data = get feature vectors of the training data from the database
for each model in the list (Resnet50, Vgg16,Vgg19) do
    for each distance_metric in (Cosine, Manhattan, Euclidean, Minkowski,
                                 Correlation, Chebyshev) do
        train new knn model(train_data, distance_metric, neighbors=5)
```

Using the previous algorithm, and after executing it in python using the scikit-learn library, 18 models were trained, one for each pair of {Convnet Architecture; Distance Metric}.

With the training stage complete we started the retrieval of the neighbor images where, using the trained models, each test image was given as an input for each model and 5 neighbors were extracted from it and store in the database.

This process was repeated for each of the 18 models and when finished we had in the database a correspondence between each test image and its five neighbors for each architecture and distance metric in a way that eases the evaluation process, due to the easiness of, with such structure, extract the results from the database.

With this flow in mind the U-KNN based image retrieval algorithm was the following:

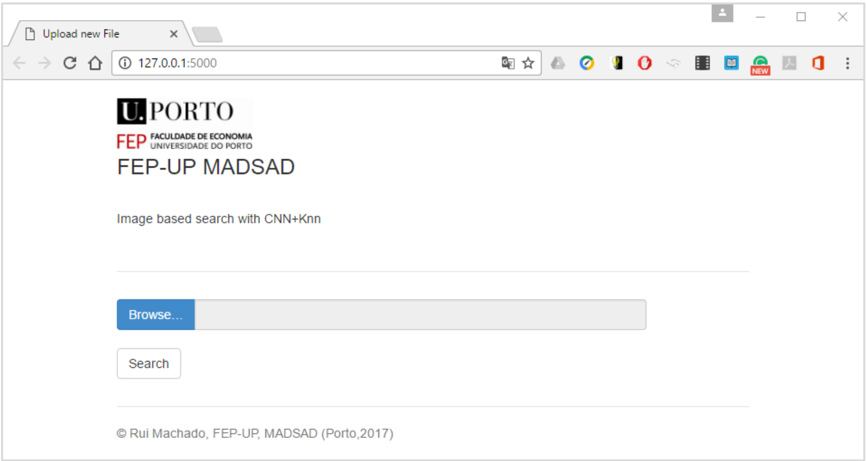
```
set test_data = get feature vectors of the test data from the database
for each model in range(1,18) do
    for each image in test_data do
        extract neighbors top five from model
        store results in the database
```

With this subchapter, we explained how we took all the train and test images feature vectors, extracted in 4.2 and trained a U-KNN model for visually similar images retrieval which was used to get neighbors of each test image in the database.

Having all the tuples {input image; neighbor1, neighbor2, neighbor3, neighbor4, neighbor5} stored in the database, we are ready to start the evaluation process on one hand and on the other build the web application that will be able to demonstrate the results.

## 4.6 Image retrieval web application

To validate in a visual way, the results of the model created, a web application was developed, using Python once again, that is able to receive an input image, run the extraction module to get the feature vector of this new image and output a set of five neighbors. When started the user has immediately the option to input a new image as shown in Figure 10 Search application running.



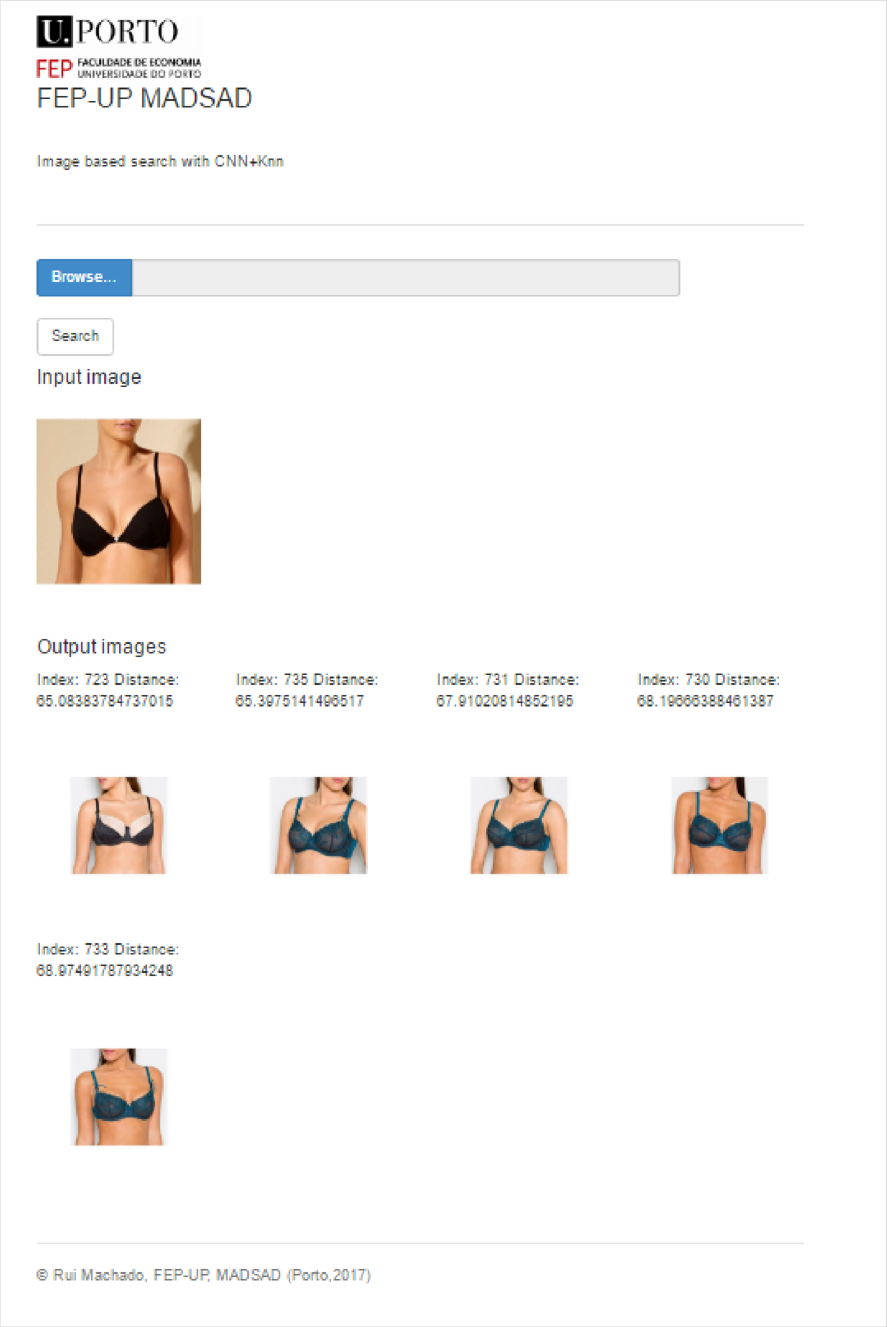
FEP-UP MADSAD

Image based search with CNN+Knn

© Rui Machado, FEP-UP, MADSAD (Porto,2017)

Figure 10 Search application running

When an image is given as an input, the internal flow will convert the image into the standard expected format, 224 pixels of height and 224 pixels of width, load the convolutional neural network architecture that is defined in the application backend, extract the visual features of it using the loaded model and lastly use the distance metric also defined in the backend to calculate the top five most similar images from the product catalog, whose features were previously extracted and stored in the database. To illustrate it Figure 11 Example of image retrieval using the web application, shows the results of an image retrieval process.



FEP-UP MADSAD

Image based search with CNN+Knn

Input image

Output images

Index: 723 Distance: 65.08383784737015

Index: 735 Distance: 65.3975141496517

Index: 731 Distance: 67.91020814852195

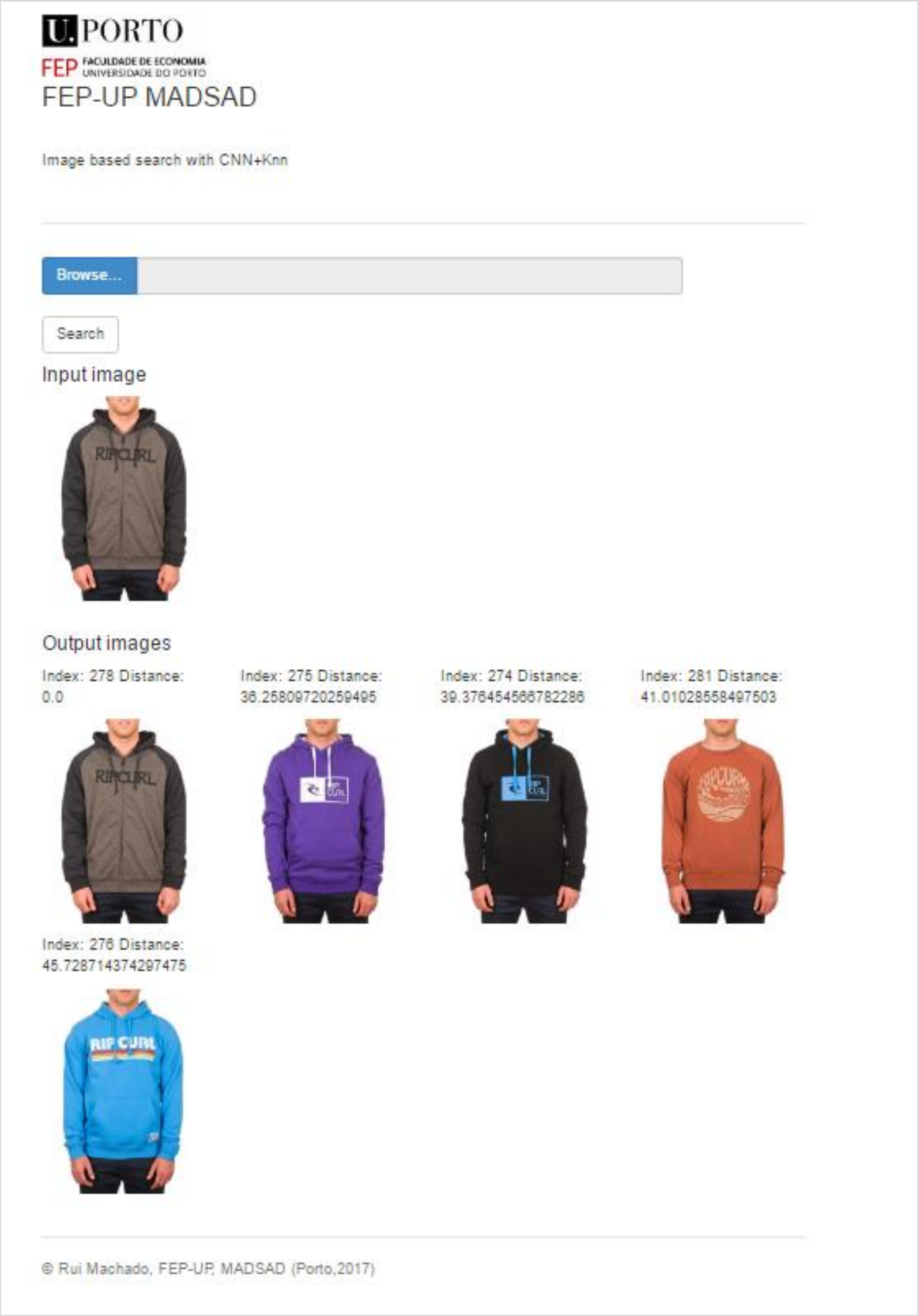
Index: 730 Distance: 68.19666388461387

Index: 733 Distance: 68.97491787934248

© Rui Machado, FEP-UP, MADSAD (Porto,2017)

Figure 11 Example of image retrieval using the web application using a Bra image as an input

Using a different product category, which in this case it’s a man’s hoodie we can see, in Figure 12 Example of image retrieval using the web application using a hoodie image as an input, that the application returned the top five most similar products and in this case the test image was also present in the train data (catalog).



FEP-UP MADSAD

Image based search with CNN+Knn

Input image

Output images

Index: 278 Distance: 0.0

Index: 275 Distance: 36.25809720259495

Index: 274 Distance: 39.376454566782286

Index: 281 Distance: 41.01028558497503

Index: 276 Distance: 45.728714374297475

© Rui Machado, FEP-UP, MADSAD (Porto,2017)

Figure 12 Example of image retrieval using the web application using a hoodie image as an input

As it was expected the distance between the input image and the equal one in the training set has a distance of 0. For this example, it also used the Vgg16 model with a Euclidean distance.

# 5 Case study and results evaluation

To evaluate the quality of the results, we need to compare the input image characteristics to the set of ranked output results. For such task, and as previously explained all the test images were used in sequence against the U-KNN model and the top 5 neighbors were extracted for each Convnet architecture and distance metric. Will all the metadata required for analysis in the database we can start comparing the input image with the resulting images.

For this task, multiple evaluation metrics were used, the first and most important for the company stakeholder is category matching, considering how much of the resulting images match the input category and sub category.

The second method is based on content-based image retrieval approaches, comparing color descriptions, by extracting the RGB distributions, by color histograms, of each image and applying a statistical independence test on top on them. Also, a texture matching test was executed, using a technique called linear binary patterns on each image and comparing input images with resulting ones by applying a correlation test between the obtained LBPs.

In the next sub-chapters, the results obtained using the previous techniques are illustrated and at the end of it, a global evaluation assessment can be analyzed.

## 5.1 Relevant product category retrieval analysis

This analysis consisted of comparing the product category of each test image with the category of the resulting neighbors. On other words, if a user inputs an image of shoes, it's expected from a functional perspective that an image retrieval system would suggest shoes as well and not any other category. The difference between the category and the sub category in only in terms of granularity being the category the highest level of product clustering and the sub category the lowest.

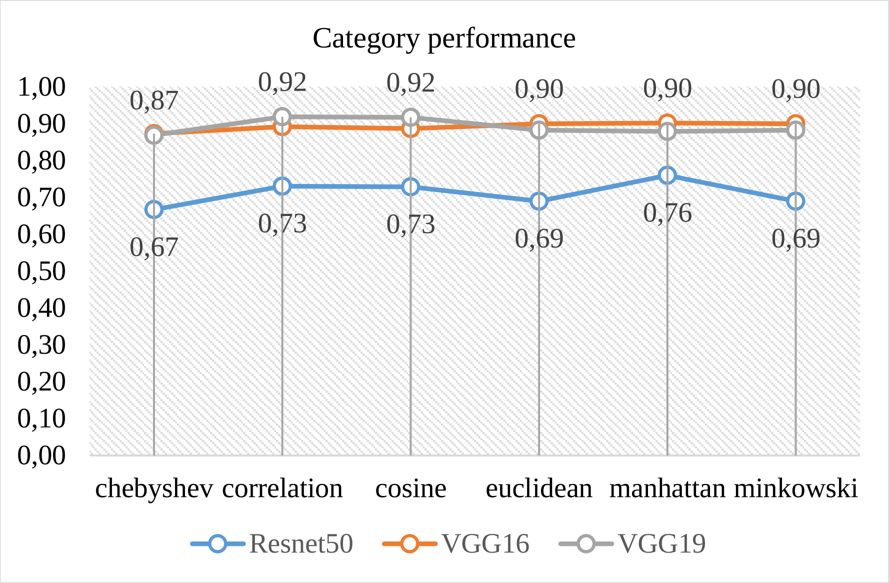


Figure 13 Category performance by convent model and distance metric

As it is demonstrated in metric the tuple {convnet architecture; distance_metric} that demonstrated the best performance in terms of a category matching between the input image and output neighbors was VGG19 as a convolutional neural network architecture and either correlation or cosine similarity as distance metrics. Both with an accuracy of 92%. From a business perspective, this means that 92 times out of 100 we are suggesting visually similar images that correspond to the same product category. It is also interesting to understand that for the data used in this work, ResNet50, even being the model with the highest number of layers, is the worst model for image retrieval, regardless of the distance metric.

Moreover, a Precision-Recall curve was also created to assess the evolution of the correctly retrieved images, by product category from the universe of possible neighbors at each step of the test. The observed results are in line with main conclusions highlighted before. Precision give us a good indication of the ratio of the number of relevant images you have retrieved to the total number of irrelevant and relevant images retrieved while Recall indicates how many of the relevant images we have retrieved so far out (evolutionary perspective of the test process) of the total number of relevant images that exist.

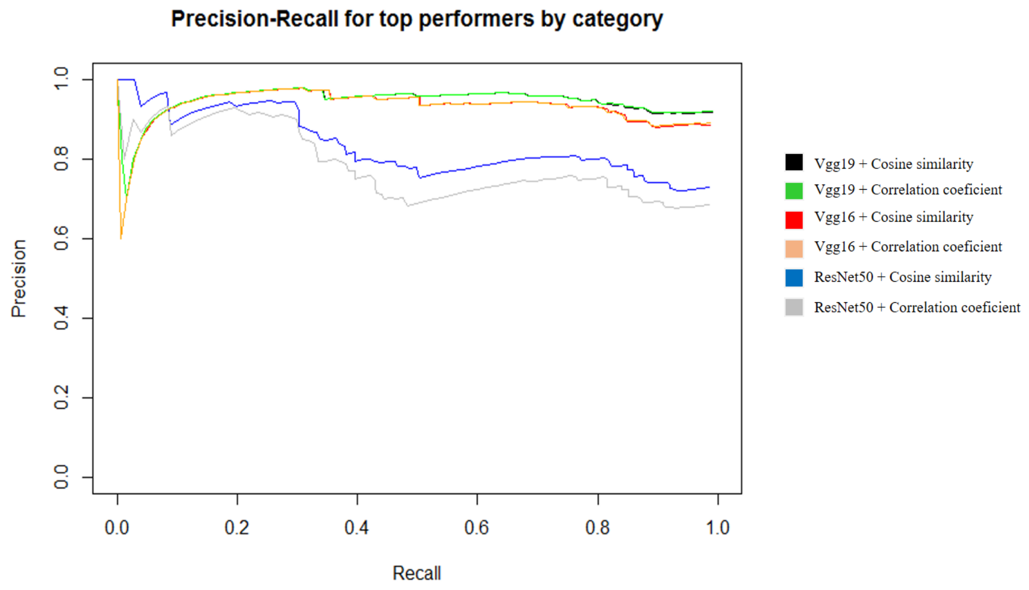


Figure 14 Precision-recall plot for top performers by category

Looking at the precision-recall curve we can understand when during the test process, the number of retrieved images was more significant. Focusing on the orange line, respective of using Vgg16 combined with correlation coefficient we can say that at the beginning we were retrieving a successive chain of irrelevant images, which made precision decrease. However, beginning in the recall ratio higher than ~0.01, the number of relevant images started to increase, stabilizing the precision around 90-95%.

These results indicate that solely using this type of category matching evaluation we would select for the final production system VGG19 as the feature extraction deep learning the model and either cosine or correlation similarity as distance metrics.

Understanding the results from the category perspective is important but if we take into consideration that “Men clothing” is a category we could still be making the mistake of suggesting a shirt when the user is searching for a hoodie as both belong to the same category but have different sub-categories.

## 5.2 Relevant product sub category retrieval analysis

This analysis consisted of comparing the product sub category of each test image with the sub category of the resulting neighbors. On other words, if a user inputs an image of a Hoodie, it's expected from a functional perspective that an image retrieval system would suggest a Hoodie as well and not any other sub category like a Shirt or a Sweatshirt.

To assess the sub category performance, we must execute the same type of analysis done at category level but now select the sub category parameter in the query being done against the database. Figure 15 demonstrates the obtained results.

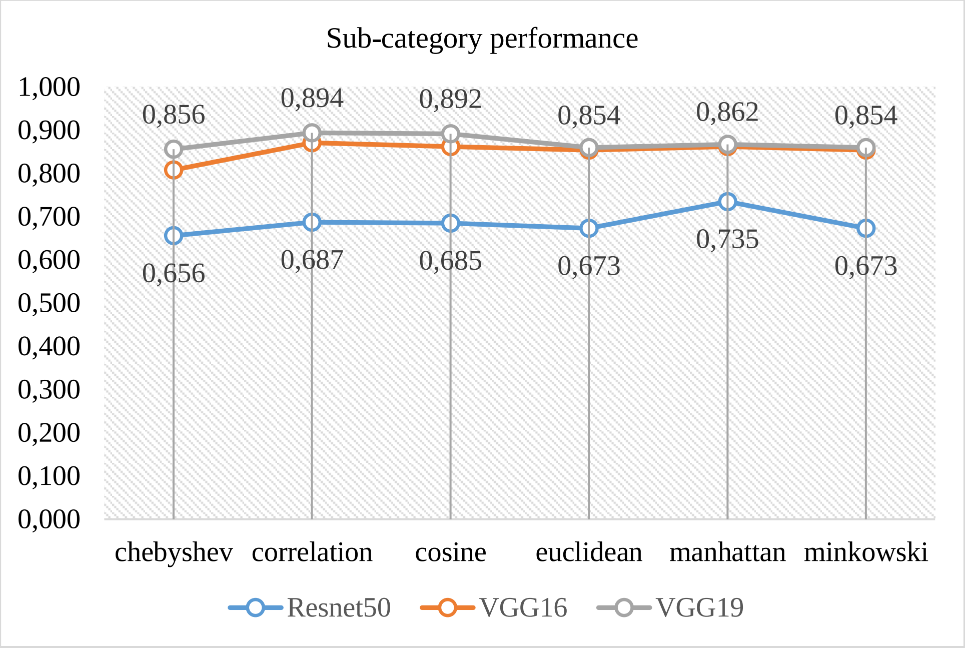


Figure 15 Sub-category performance by convent model and distance metric

Moreover, a Precision-Recall curve was also created to assess the evolution of the correctly retrieved images, by product sub category from the universe of possible neighbors at each step of the test. Once again, the results are in line with main conclusions highlighted before and the behavior in terms of relevant retrieved images is like the category analysis.

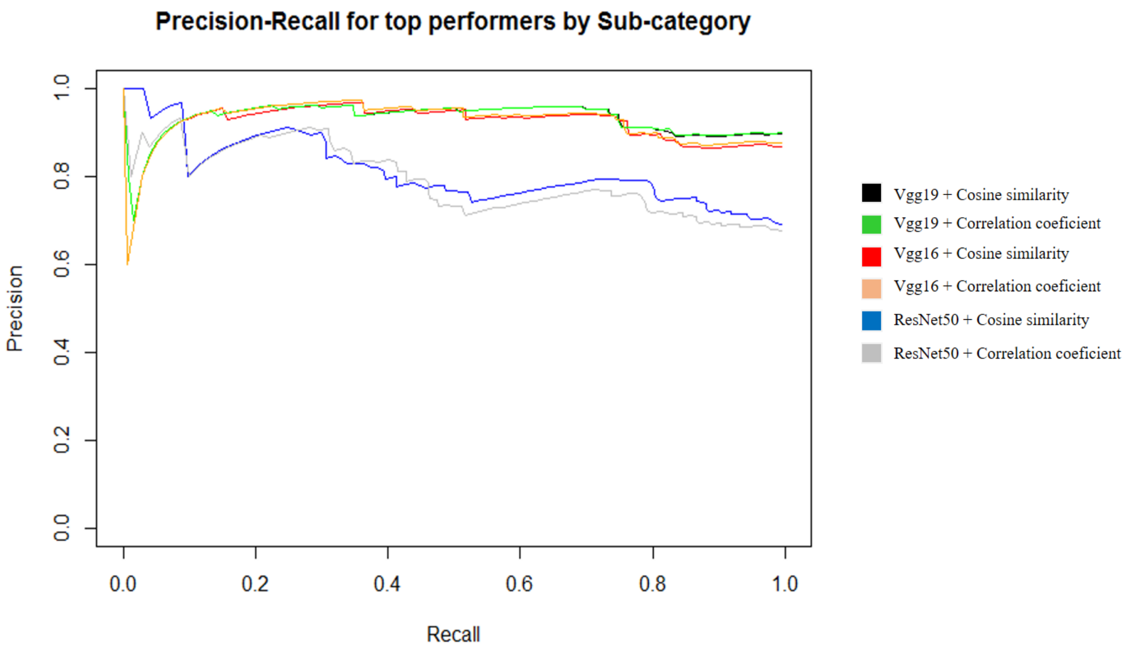


Figure 16 Precision-recall plot for top performers by subcategory

As it is demonstrated by both sub category matching analysis and the precision-recall curve, VGG19 with correlation or cosines similarity as distance metrics is still out-performing when comparing to the others, however, the accuracy is much lower when analyzing the result at this granularity.

The result of 89.4% of sub category matching is the best-obtained accuracy, that even so it can be considered a positive result, shows that there is a higher set of the mistakes in our image retrieval system at the sub category level.

ResNet50 demonstrates the same behavior as before and is the worst choice so far with 68.7%, regardless of the distance metric used for similarity calculation.

To better understand where these errors are coming from, we have also created an error analysis at the product sub category level. Due to the high number of records, we have selected VGG19 and cosine similarity for this analysis, as being the best models, surely help us identify more accurately the sub-categories with more errors.

Table 8 Success rate per category and sub category, shows the success rate distribution per sub category.

| Model | Distance Metric | Category | Sub Category | % Correct |
|---|---|---|---|---|
| Vgg19 | cosine | Men’s Clothing | Hoodies | 0.933 |
| Vgg19 | cosine | Men’s Clothing | Sweatshirt | 0.973 |
| Vgg19 | cosine | Pants | Pants | 0.92 |
| Vgg19 | cosine | Shoes | Shoes | 0.987 |
| Vgg19 | cosine | Women Clothing | Bras | 0.88 |
| *Vgg19* | *cosine* | *Women Clothing* | *Dresses* | *0.36* |
| Vgg19 | cosine | Women Clothing | Sweatshirt | 0.85 |
| Vgg19 | cosine | Women Clothing | Traditional wear | 0.8 |

Table 8 Success rate per category and sub category

As it can be observed the highest error is coming from the sub category “dresses” where the percentage of correct similar images suggestion has only 36% of accuracy. One hypothesis relies on the fact that the e-commerce company only provided 22 images of dresses. Further tests could be made, upon new images gathering, increasing the number of dresses and re-running these analyses to evaluate the impact of it.

Regardless of the decrease of performance from the category level analysis to the sub category, which can be considered residual, these results start to indicate a case, that VGG19 shows the best results of the conducted tests when combined with the correlation coefficient or with the cosine similarity for building deep learning based image retrieval systems in a fashion context. Since category and sub category analysis is a company requirement, we have also decided to detail its performance per neighbor, also using the VGG19 architecture combined with the cosine similarity.

| Neighbor rank | Category Matching | # Correct categories | # of Cases | Average distance |
|---|---|---|---|---|
| 1 | 91% | 88 | 97 | 0,19 |
| 2 | 95% | 90 | 95 | 0,18 |
| 3 | 88% | 84 | 96 | 0,17 |
| 4 | 90% | 86 | 96 | 0,16 |
| 5 | 96% | 92 | 96 | 0,12 |

Table 9 Correct category cases per neighbor using Vgg19 and cosine similarity

As it is possible to observe in similarity the percentage of correct categories between the input image and the retrieved neighbor is high across all the neighborhood, however there it is not possible to observe an increase of errors while the distance increases.

For example, neighbor 1 is in average the closest one to the input according to the cosine similarity index however when compared to the farthest average 5th neighbor, we see that the distance has decreased meaning is more distant of the input, however, the category matching percentage is higher at this level.

This is related to the fact that deep neural networks such as convolutional neural networks don´t look at one attribute only but combine color, texture, and shape when retrieving feature vectors (Bengio, 2009). It also means that the closest neighbor in terms of distance metric might be the one with a highest error rate in one of the analysis dimensions ( a category to evaluate shape, color, and texture).

## 5.3 Cross-validation evaluation for category and sub category matching

To complete the product tree matching analysis, we have also conducted a cross-validation test to demonstrate the stability of the observed results while the test set changes.

| **CNN Model** | Vgg19 | Vgg19 | Vgg16 | Vgg16 | ResNet50 | ResNet50 |
|---|---|---|---|---|---|---|
| **Distance Index** | Correlation | Cosine | Correlation | Cosine | Euclidean | Correlation |
| **Number of Folds** | 10 | 10 | 10 | 10 | 10 | 10 |
| **Images p/ fold** | 20 | 20 | 20 | 20 | 20 | 20 |
| **Avg Cat Accuracy** | 91% | 93% | 86% | 87% | 64% | 63% |
| **Avg Subcatg Acc.** | 86% | 89% | 86% | 84% | 67% | 64% |
| **Std Cat Acc.** | 0,08 | 0,05 | 0,09 | 0,09 | 0,15 | 0,16 |
| **Std Subcat Acc.** | 0,11 | 0,10 | 0,08 | 0,12 | 0,14 | 0,12 |
| **Var. Coef. Cat** | 9% | 5% | 11% | 11% | 24% | 25% |
| **Var. Coef. Subcat** | 13% | 11% | 10% | 14% | 21% | 19% |

Table 10 Summary of evaluation results after cross-validation test

With the cross-validation test executed, running 10 folds, each with 20 different test images, we have observed as showed in Table 10, a category accuracy of 93% in average and a sub category accuracy of 89% in average for Vgg19 with a cosine similarity as distance index, proving to be the tuple that best maximizes accuracy for the performed tasks using the previously referred data set.

It’s also interesting to refer the low standard deviation and variation coefficient that denote a certain stability around the observed mean for all the different tuples, along with the tests execution.

## 5.4 Comparing color histograms

Color histograms allow us to compare how each RGB channel behaves across the pixel universe of one or more images. The results of the color histogram correlation don´t necessarily denote the quality of results as the color histograms of two completely different images might be the same, but this might not relate to domain/business significance.

Say the color histogram of a car might be the same as of a book but this correlation doesn´t mean anything in terms of similar object retrieval. For this, product category and sub category analysis will be the more relevant.

This type of analysis allows, however, to better understand the Convnet behavior in terms of color learning and sub consequent impact on the visual feature vector it returns. It is, however, important to note that this type of analysis is highly influenced by using only images with cropped regions of interest versus using images with high non-relevant areas, as it will be shown in this subchapter.

For the practical implementation, the color histograms library of OpenCV was used, which is an open source computer vision and machine learning software library, that’s allows us to both extracts the color histogram but also compare two instances using multiple comparison methods. For this work, a Pearson correlation was selected.

With this in consideration the applied process for testing color correlation between images was the following:

1. **Extract** color histogram for the test image
2. **For each** neighbor, out of five extracted
    3. **Extract** the color histogram of the neighbor
    4. **Compare** color histograms using Pearson correlation
    5. **Store** the correlation in the database

The following Figure 17 helps understanding of this process works by showing the result of extracting the color histograms for two images that are non-correlated in terms of color. To ease the readiness of the chart, we should consider in the Y axis, the number of pixels in the image, while the X axis illustrates the value for each RGB channel.

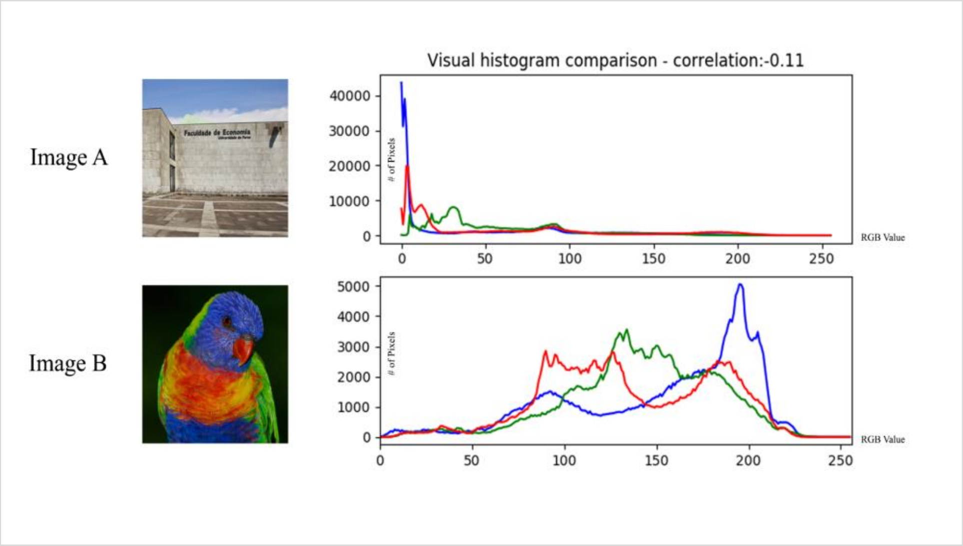


Figure 17 Visual comparison of two histograms for two noncorrelated images.

On the opposite, the following Figure 18 shows an example of two highly positive correlated images in terms of color. As we can see, both histograms look alike.

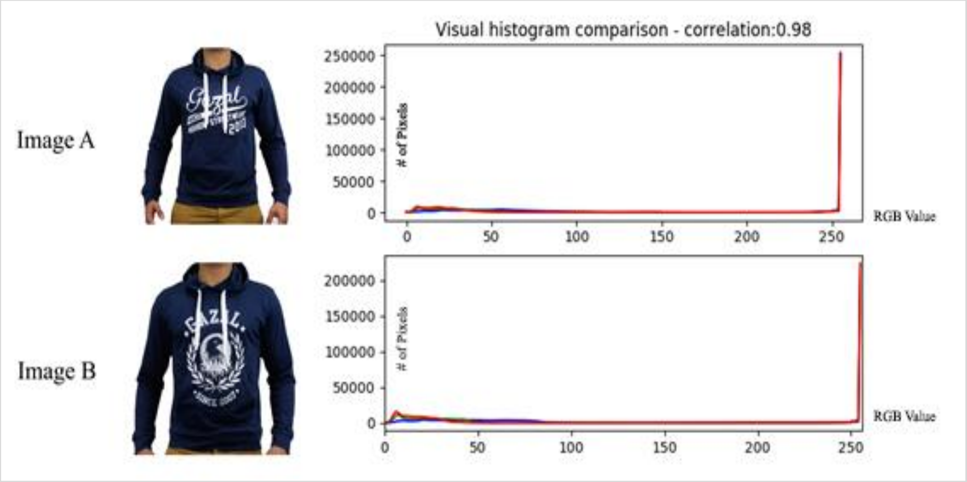


Figure 18 Visual comparison of two histograms for two correlated images

This technique was applied to all the tuples {Test Image; Predicted Neighbor; Prediction Rank}, being the prediction rank the order of similarity between the test image and each of the five predicted neighbors, to understand the impact of color correlation of the obtained results. On other words, it explains how much closer is the test image to its neighbors in terms of color.

After applying such technique to the test data, we have realized that using color to evaluate the performance of an image retrieval system can bring erroneous conclusions. This is because of the background and a number of colors that are present in the image but not in the object under study, which is right at the center of the image. To prove this hypothesis one simple comparison of results between scenarios were conducted. Using the Vgg16 architecture and the Manhattan distance for neighbor selection, one example was selected to illustrate the similarity in terms of color, of the results obtained.

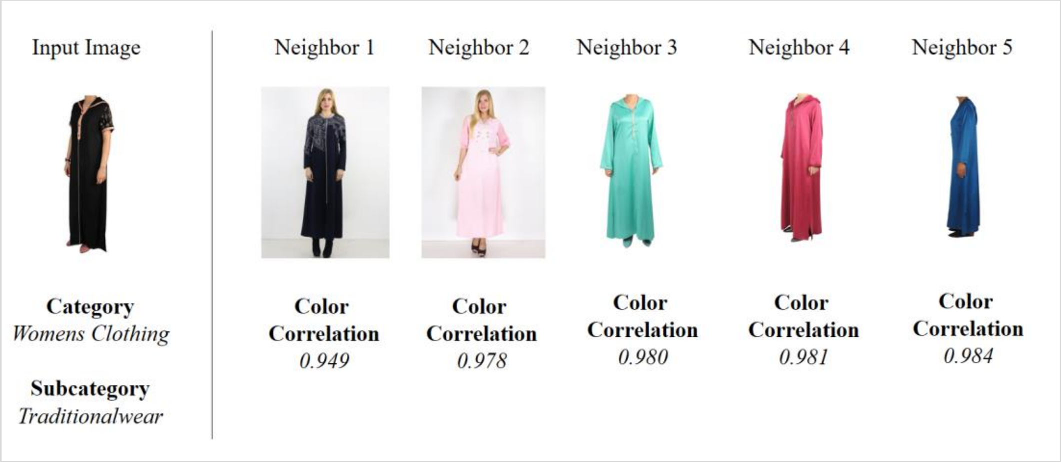


Figure 19 Color correlation example

As we can see in Figure 19, although we obtain high values for correlation which could be interpreted as good results, they don´t seem to reflect the reality. According to the correlation metric, the closer to 1, the highest the similarity in terms of color, and by looking at the input image that is a black traditional dress, and the second neighbor which is a pink traditional dress, it was expected to have as output a low correlation. This is related to a number of non-relevant regions such as the white background that affect the analysis of color on the region of interest, the dress.

As proof of evidence, background and Figure 21 Correlation between two images with cropped background show a comparison between preprocessing the catalog images for cropping images to contain only the regions of interest and not doing it. As we can see the results are different, in the first one, due to with amounts of white, the region of interest becomes non-relevant and both images show a high correlation in terms of color of 0.98.

In the second one, only regions of interest are given to the histogram extractor and the correlation becomes more realistic according to the human eye, resulting in non-correlated histograms with 0.25 of the correlation coefficient.

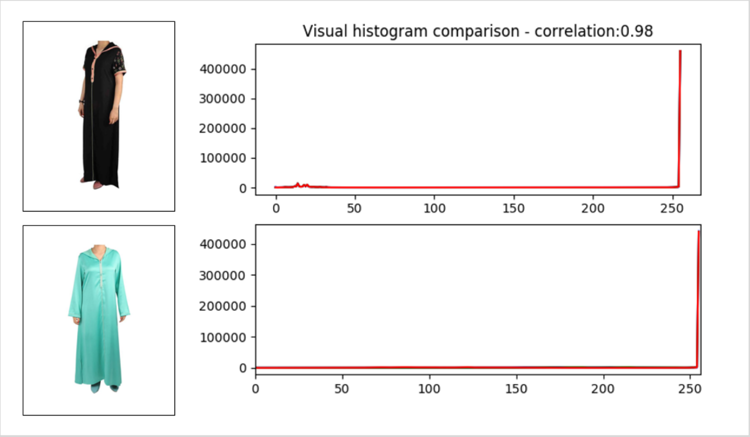


Figure 20 Correlation between two images with large white background

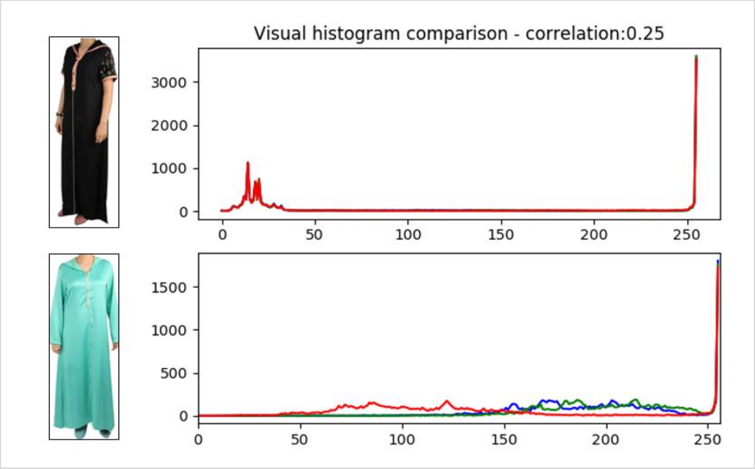


Figure 21 Correlation between two images with cropped background

The latest indicates the need of pre-processing images by either cropping regions of interest of adding bounding boxes that indicate to the learning models, which regions should be considered, otherwise the results might be erroneous when using color as the only means of feature extraction. Even so, color correlation analysis was performed and Table 10 show the obtained results, inflated by the background issue, stated before.

| **Architecture** | **Avg R** | **Min R** | **Max R** |
|---|---|---|---|
| Resnet50 | 0,86256 | 0,99958 | 0,22320 |
| Vgg16 | 0,87156 | 0,99991 | 0,22892 |
| Vgg19 | 0,86781 | 0,99967 | 0,22004 |

Table 11 Avg color histogram correlation per Convnet architecture

Analyzing these high-level values per convolutional network architecture, we could say that the color understanding and learning is homogenous across the three tested ones, since the same similarity measures were used of each convent architecture and these results compile and average between them, however, to be able to trust these conclusions further tests would have to be conducted using only the region of interest of the input and output images. Even so, a deeper analysis was conducted to observe the behavior of the different architectures in terms of color correlation. Keeping in mind the bias caused by the regions of interest, it's still interesting to evaluate which architecture delivers the highest results.

The following Figure 22 shows such similarity in terms of color correlation by comparing the average histogram correlation of each at the granularity of the test image neighbor obtained.

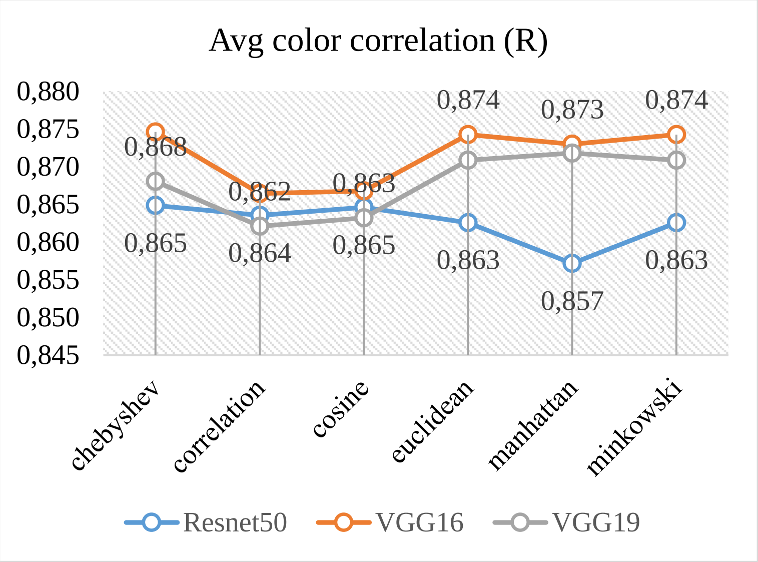


Figure 22 Avg color correlation by CNN architecture and distance metric

As we can see in the previous Figure 22, the average values for color correlation are quite similar across the analysis spectrum, varying between 0.857 as a minimum using ResNet50 with Manhattan distance, to a maximum of 0.874 for Vgg16 using either a Euclidean distance or the Minkowski one. Once again, this high correlation is due to the noise existing in the provided images by the e-commerce company.

Even so, using the available data, we can conclude that Vgg16 architecture is the best retrieving color correlation, using Euclidean or Minkowski as a distance metric. To further detail this evaluation, Table 12, details the color correlation dispersion, for this winning combination for each of the suggested neighbors in average.

| Architecture | Distance Metric | Neighbor Rank | Avg R |
|---|---|---|---|
| Vgg16 | Minkowski | 1 | 0.864302485790451 |
| Vgg16 | Minkowski | 2 | 0.880394166672799 |
| Vgg16 | Minkowski | 3 | 0.876317459100337 |
| Vgg16 | Minkowski | 4 | 0.871238119649183 |
| Vgg16 | Minkowski | 5 | 0.879156777047101 |

Table 12 Color correlation by neighbor for winning tuple {Convnet architecture: distance metric}

As demonstrated, there is no significate variation in terms of color correlation within the different levels of similarity between input image and neighbors. For example, in average, the color similarity of the first neighbor is 0.86 but for the most distance neighbor, the fifth, the correlation is higher, showing a correlation coefficient of 0.87.

## 5.5 Comparing texture and patterns with Linear Binary Patterns and Chi-square Distance

Using the non-parametric LBP descriptor combined with a distance metric is a recognized strategy for summarizing local structures of images, such as texture or pattern analysis. Originally it was proposed to act as means to describe the texture of images, but recently it has been also used for facial recognition and facial expression analysis. (Huang, Shan, Ardabilian, Wang, & Chen, 2011)

The way LBP works is by selecting a neighborhood of a certain size (parameter of the model) for each pixel in the grayscale image and calculating an LBP value of each of these pixels. This value is obtained by analyzing if the RGB intensity (Can vary between 0 and 255) of the center pixel is greater than or equal to its neighbor, which in this case we set its value to 1 otherwise, we set it to 0. This test part of the LBP calculation is called binary test. Figure 23 show how LBP value is calculated for one pixel. (Rosebrock, 2016)

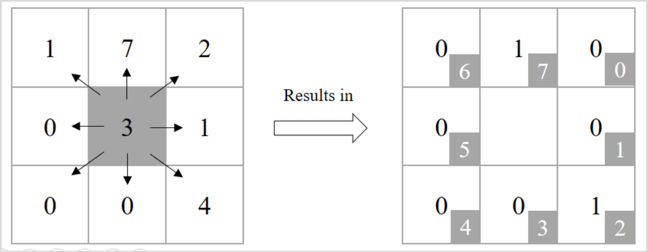


Figure 23 How LBP value is calculated for each pixel in an image

With the LBP matrix calculated for this center “3” pixel, we can then calculate a single LBP value for the pixel. This is done by calculating a decimal value from the binary test array ordered in this case from 0 to 7.

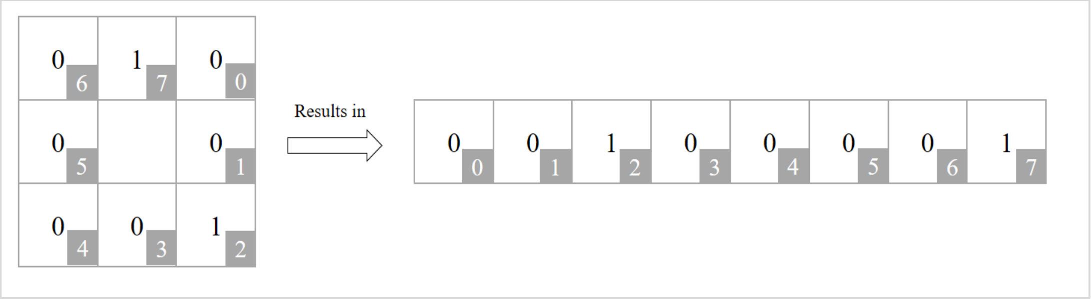
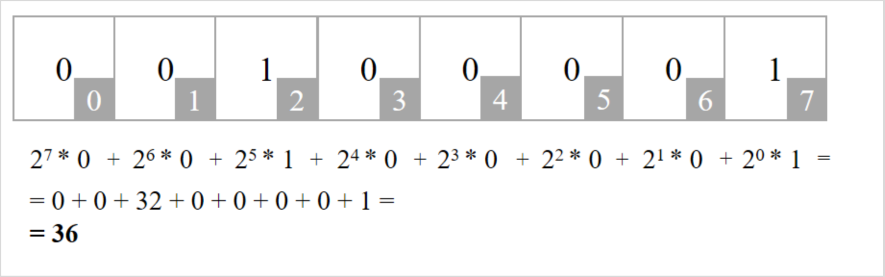


Figure 24 LBP matrix to array

From the binary array, we can transform it into a decimal value by applying Formula 1 Calculating LBP value.

$$\sum_{n=0}^{7}(2^n * x_n)$$

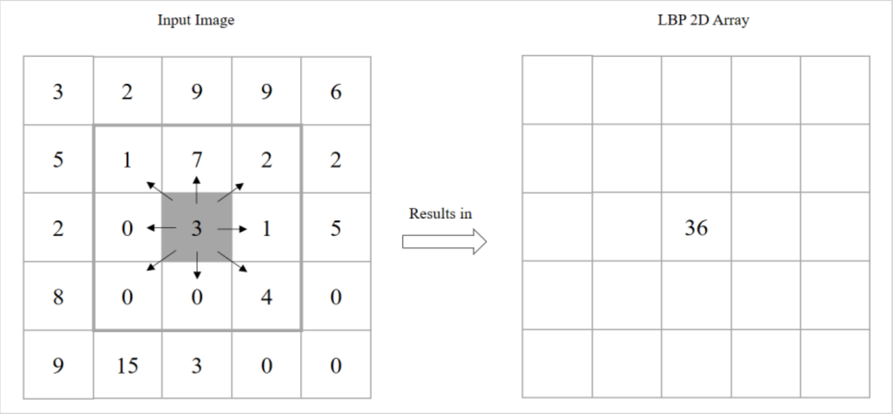
Formula 1 Calculating LBP value



$$2^7 * 0 + 2^6 * 0 + 2^5 * 1 + 2^4 * 0 + 2^3 * 0 + 2^2 * 0 + 2^1 * 0 + 2^0 * 1 =$$
$$= 0 + 0 + 32 + 0 + 0 + 0 + 0 + 1 =$$
$$\mathbf{= 36}$$

Figure 25 Calculating a decimal value for LBP

Using the previous, we obtain the final value for this pixel in terms of LBP pattern. This strategy is then applied to all the pixels in the image and we obtain the final output, the LBP 2D array.



| | | | | |
|---|---|---|---|---|
| 3 | 2 | 9 | 9 | 6 |
| 5 | 1 | 7 | 2 | 2 |
| 2 | 0 | 3 | 1 | 5 |
| 8 | 0 | 0 | 4 | 0 |
| 9 | 15 | 3 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | 36 | | |
| | | | | |
| | | | | |

Figure 26 Example of an LBP 2D Array

As it was stated for color correlation analysis, linear binary patterns analysis might be an interesting test to analyze how close the input image is close to its neighbors in terms of texture but business wise, it says nothing about the category and sub category matching, which is the most relevant requirement of the company.

For the practical implementation, the *local_binary_pattern*[15] library of *scikit-image*[16] package for Python was used, which is a collection of algorithms for image processing, that's allows us to do single image manipulations such as cropping, resizing and cutting but also compare images using multiple comparison methods.

Within the local binary pattern library, we have selected the LBP histogram extraction combined with a chi-squared distance to evaluate how close two images are to each other in terms of texture.

LBP histograms can only be extracted from grayscale images and so each processed one was converted before extracting the histogram.

With this in consideration the applied process for testing texture matching between images was the following:

```
1. Convert test image to grayscale
2. Extract LBP histogram for the test image
3. for each neighbor out of five extracted:
4.         Convert neighbor image to grayscale
5.         Extract the LBP histogram of the neighbor
6.         Compare LBP histograms using Chi-square distance
7.         Store the distance in the database
```

The following Figure 27 helps understanding of this process works by showing the result of extracting the LBP histograms for two images that are closely distant in terms of texture, using the Chi-squared distance. Moreover, we have used a neighborhood based on a radius of 24 pixels.

---

[15] http://scikit-image.org/docs/0.8.0/auto_examples/plot_local_binary_pattern.html

[16] http://scikit-image.org/

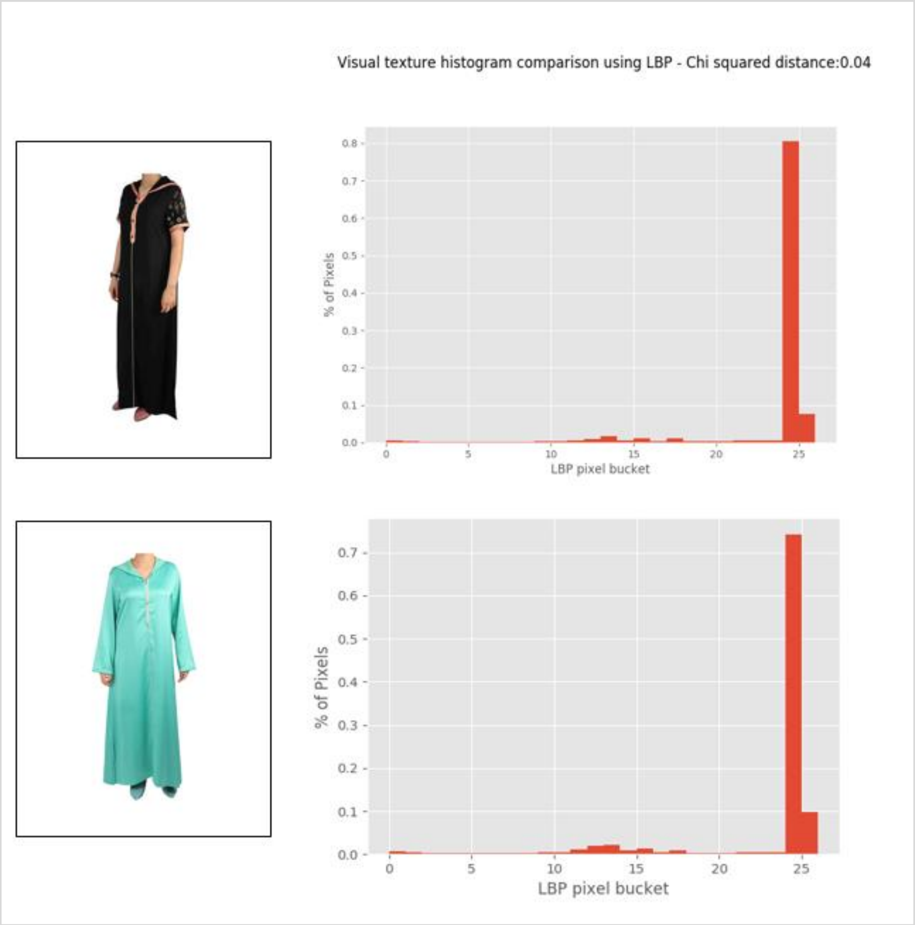


Figure 27 Example of LBP Comparison between two images

Once again and following the results obtained in the color histogram comparison, we went to validate if the background would affect the obtained results. In fact, because LBP works based on neighborhood, it is expected that the results contain the bias of the common background areas that will show a high correlation and thus a shorter distance in terms of pattern matching.

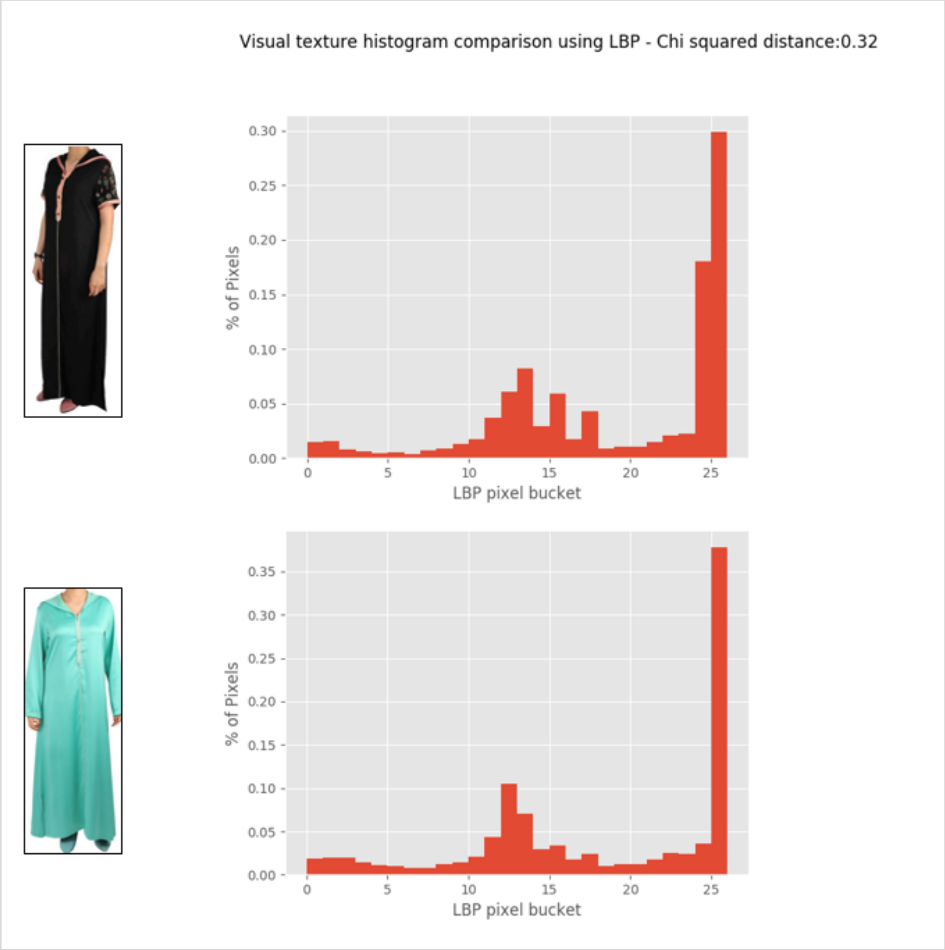


Figure 28 Example of LBP Comparison between two cropped images

As expected, the distance as increased after testing with cropped images. Figure 28 shows that these images have a chi-squared distance of 0.32 with the LBP pattern while Figure 26 shows a 0.04 distance. This test once again proves the need to have images with only regions of interest and previously cropped to fit them. Even so, the main results obtained for texture/pattern analysis will be presented.

| **Architecture** | **Avg, $\chi^2$ Distance** | **Min $\chi^2$ Distance** | **Max $\chi^2$ Distance** |
|---|---|---|---|
| Resnet50 | 0,22459 | 0,00069 | 6,96765 |
| Vgg16 | 0,18489 | 0,00058 | 5,48810 |
| Vgg19 | 0,17050 | 0,00058 | 4,55755 |

Table 13 Texture analysis results obtained by model

Analyzing these high-level values per convolutional network architecture, we could say that the texture understanding and learning is homogenous across the three tested ones, since the same similarity measures were used of each convent architecture and these results compile and average between them, however, to be able to trust these conclusions further tests would have to be conducted using only the region of interest of the input and output images. Even so, by analyzing these results we can conclude that Vgg19 is the model retrieving the most similar results between the input image and retrieved neighbors with an average chi-squared distance of 0.17.

A deeper analysis was conducted to observe the behavior of the different architectures in terms of texture similarity. Keeping in mind the bias caused by the regions of interest, it’s still interesting to evaluate which architecture delivers the highest results.

The following Figure 29 shows such similarity in terms of texture distance by comparing the average texture chi-squared distance at the granularity of the test image neighbors obtained.

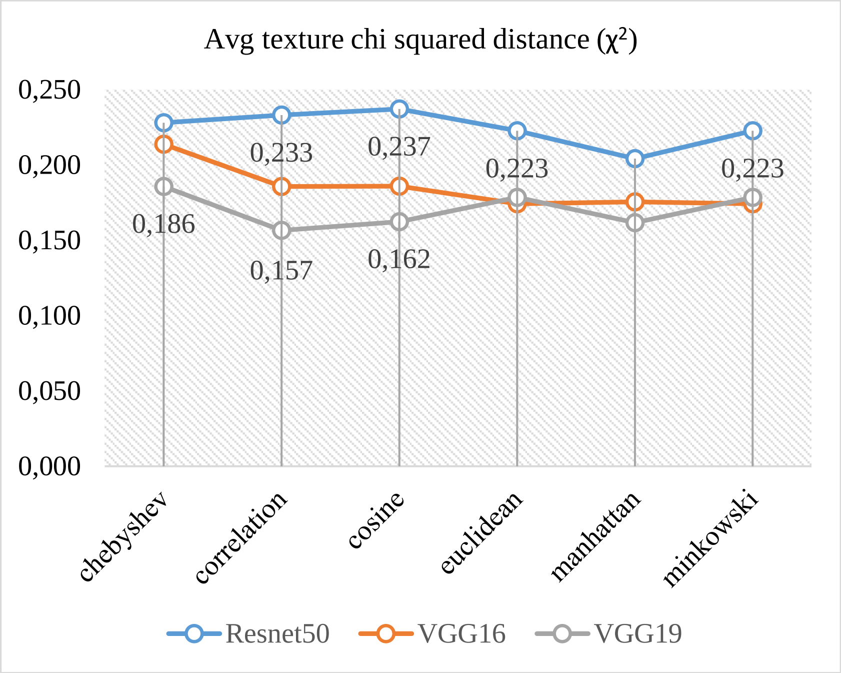


Figure 29 Avg texture correlation by CNN architecture and distance metric

As we can see in the previous Figure 25, the average values for texture similarity are quite similar across the analysis spectrum, varying between 0.157 as a minimum using Vgg19 with correlation similarity, to a maximum of 0.237 for ResNet50 using cosine similarity as a distance metric. Once again, this high similarity is due to the noise existing in the provided images by the e-commerce company.

Even so, using the available data, we can conclude that Vgg19 architecture is the best-retrieving texture related neighbors, using correlation similarity as a distance metric. To further detail this evaluation, Table 14, details the texture similarity correlation dispersion, for this winning combination for each of the suggested neighbors in average.

| Architecture | Distance Metric | Neighbor Rank | Avg $\chi^2$ |
|---|---|---|---|
| Vgg19 | correlation | 1 | 0.124924064633015 |
| Vgg19 | correlation | 2 | 0.170755307423161 |
| Vgg19 | correlation | 3 | 0.159732479954178 |
| Vgg19 | correlation | 4 | 0.174878766962978 |
| Vgg19 | correlation | 5 | 0.153007106600164 |

Table 14 Texture similarity by neighbor for winning tuple {Convnet architecture: distance metric}

As demonstrated, there is no significative variation in terms of texture similarity within the different levels of similarity between input image and neighbors. For example, in average, the texture proximity of the first neighbor is 0.12 but for the most distance neighbor, the fifth, the distance is higher, showing an observed chi-squared distance of 0.15, but smaller values are observed in average for neighbors at 3rd position.

## 5.6 Evaluation summary

After analyzing the three-evaluation dimensions, we have compiled the observed results in a summary table that intends to demonstrate which architectures and distance metrics best performance in an image retrieval problem.

| **Analysis** | **Best Architecture** | **Best Distance Metric** | **Observed value** |
|---|---|---|---|
| Category Match | Vgg19 | Cosine | 93,0% Avg. Accuracy from CV |
| Sub-category Match | Vgg19 | Cosine | 89% Avg. Accuracy from CV |
| Color correlation | Vgg16 | Minkowski or Euclidean | 0,87 Avg. Correlation[17] |
| Texture similarity | Vgg19 | Correlation | 0,157 Avg. Chi-squared distance[18] |

Table 15 Summary of observed results

Table 15 Summary of observed results summarized the observed best performers for each of the evaluation test conducted. The main conclusions are that, for the training set used in this work, Vgg19 and cosine similarity as a similarity measure, are the tuple combination that allowed us to achieve the best performance with a successful category matching, in average of 93% and a sub category matching of 89% in average after cross-validation test was conducted.

Regarding the color and texture similarity between the input image and output neighbors, and again, with the bias injected by the images containing high regions of non-interest, Vgg19 achieved the best results for texture with an average chi-squared distance of 0.157, while Vgg16 achieved the best results for color correlation.

---

[17] Values inflated by bias in regions of interest

[18] Values inflated by bias in regions of interest

# 6 Conclusions and future work

Building a deep learning model using pre-trained convolutional neural networks is an accessible reality using frameworks like Keras, that abstract the researcher from building complex algorithms to rather focus on going directly to the construction of the required models using simple coding structures.

Although they might not fit all purposes, as these models are pre-trained with general images instead of domain specific, they have proven to allow the retrieval of accurate results, in the order of 92% success rate of same product category matching and 90% in sub category matching between input image (search query) and retrieved neighbors, in the fashion industry, with a dataset of images from a Morocco fashion e-commerce company.

These evaluation numbers denote that this is a technology that can be considered for any image retrieval project with a commercial purpose, however requiring specific work depending on the domain of application and initial results accuracy. An example of such are the results obtained in the color and texture evaluation process, where the fact that the catalog (training and testing) images were not cropped to the region of interest only (an image of a dress had a large white background with the dress in the middle), distort the results and make all images highly correlated in terms of color and texture. This demonstrates and highlights the opportunity of researching in future work, how to segment images combined with dynamic cropping so that only regions of interest are given as inputs for both training and testing the image based search system suggested in this work and how it affects the model evaluation.

Even so, after testing three different models, Vgg16, Vgg19 and Resnet50, each combined with four different distance metrics, Euclidean, Manhattan, Minkowski and Chebyshev and two similarity indexes, cosine and correlation, we can conclude that Vgg19 combined with a correlation coefficient for similarity calculation is the model tuple that best maximizes the similarity between a search image and its retrieved neighbors. As future work mentioning, new pre-trained architectures, provided by Google, such as InceptionV3 are now available at the Keras library, which also demonstrates an opportunity to evaluate in future work, how this convnet behaves in the process of feature extraction when compare to our used Vgg and ResNet architectures.

Focusing now only on future work, it would be interesting to understand how business rules can be combined in the proposed architecture so that it affects the neighbors returned by the model. Say that the model is extracting the five top similar products from a given input image however the six elements in the list provides a higher profit to the company when bought. The e-commerce company used as a reference in this work identified the need to show this product instead of another neighbor that shoes a lower profit margin.

Finally, it would be interesting to research how to transform this System into a software as a service component, where a user can make a request through a web-API, sending an input image and after running our query on our server, we could retrieve the five top similar neighbors and return them also through the we-API. This is the final test to evaluate if deep learning for image retrieval is at a high maturity level and can be used as an enterprise option.

# 7 Bibliography

Alkhawlani, M., Elmogy, M., & Bakry, H. El. (2015). Text-based, Content-based, and Semantic-based Image Retrievals: A Survey. *International Conference on Machine Vision, Image Processing and Pattern Analysis*, *4*(1), 58–66. Retrieved from http://www.ijcit.com/archives/volume4/issue1/Paper040109.pdf

Bengio, Y. (2009). *Learning Deep Architectures for AI*. *Foundations and Trends in Machine Learning* (Vol. 2). https://doi.org/10.1561/2200000006

Cecilio, D. (2015). E-commerce is changing the fashion industry – it’s time to catch up. Retrieved from https://www.theguardian.com/small-business-network/2015/feb/06/how-mobile-ecommerce-changing-fashion-industry

Chatfield, K., Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Return of the Devil in the Details: Delving Deep into Convolutional Nets. *arXiv Preprint arXiv: …*, 1–11. https://doi.org/10.5244/C.28.6

Chaudhari, R., & Patil, A. M. (2012). Content Based Image Retrieval Using Color and Shape Features. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, *1*(5), 386–392. Retrieved from www.ijareeie.com

Chechik, G., Sharma, V., Shalit, U., & Bengio, S. (2009). An Online Algorithm for Large Scale Image Similarity Learning. *Advances in Neural Information Processing Systems*, *21*, 1–9. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.8937&amp;rep=rep1&amp;type=pdf

Goshtasby, A. A. (2012). *Image Registration*. https://doi.org/10.1007/978-1-4471-2458-0

Goswami, A., Chittar, N., & Sung, C. H. (2011). A study on the impact of product images on user clicks for online shopping. *Proceedings of the 20th International Conference Companion on World Wide Web - WWW ’11*, 45. https://doi.org/10.1145/1963192.1963216

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Arxiv.Org*, *7*(3), 171–180. https://doi.org/10.3389/fpsyg.2013.00124

Huang, D., Shan, C., Ardabilian, M., Wang, Y., & Chen, L. (2011). Local binary patterns and its application to facial image analysis: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *41*(6), 765–781. https://doi.org/10.1109/TSMCC.2011.2118750

Huu, Q., Thu, H., & Quoc, T. (2012). An efficient content based image retrieval method for retrieving images. *International Journal of …*, *8*(4), 2823–2836. Retrieved from http://www.ijicic.org/ijicic-10-11046.pdf

Iglesias, F., & Kastner, W. (2013). Analysis of Similarity Measures in Times Series Clustering for the Discovery of Building Energy Patterns. *Energies*, *6*(2), 579–597. https://doi.org/10.3390/en6020579

Jing, Y., Liu, D., Kislyuk, D., Zhai, A., Xu, J., Donahue, J., & Tavel, S. (2015a). Visual Search at Pinterest. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1889–1898. https://doi.org/10.1145/2783258.2788621

Jing, Y., Liu, D., Kislyuk, D., Zhai, A., Xu, J., Donahue, J., & Tavel, S. (2015b). Visual Search at Pinterest. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1889–1898. https://doi.org/10.1145/2783258.2788621

LeCun, Y., Bengio, Y., Hinton, G., Y., L., Y., B., & G., H. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Manzoor, U., & Balubaid, M. A. (2015). Semantic Image Retrieval : An Ontology Based Approach, *4*(4), 1–8.

Rosebrock, A. (2016). Local Binary Patterns with Python & OpenCV - PyImageSearch. Retrieved from http://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

Rui, Y., Huang, T., & Chang, S.-F. (1999). *Image retrieval: current techniques, promising directions and open issues*.

Rui, Y., Huang, T. S., & Chang, S.-F. (1999). Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, *10*(1), 39–62. https://doi.org/10.1006/jvci.1999.0413

Sergy, S. (2008). Color Histogram Features Based Image Classification in Content-Based Image Retrieval Systems, 221–224.

Simonyan, K., Andrew Zisserman, & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, 1–14. https://doi.org/10.1016/j.infsof.2008.09.005

Wan, J., Wang, D., Hoi, S. C. H., Wu, P., Zhu, J., Zhang, Y., & Li, J. (2014). Deep Learning for Content-Based Image Retrieval. *Proceedings of the ACM International Conference on Multimedia - MM '14*, 157–166. https://doi.org/10.1145/2647868.2654948

Wei Di, Neel Sundaresan, Robinson Piramuthu, A. B. (2014). Is a picture really worth a thousand words?:-on the role of images in e-commerce. *Proceedings of the 7th ACM International Conference on Web Search and Data Mining - WSDM '14*, 633–641. https://doi.org/10.1145/2556195.2556226

Yeung, K. (2017). Samsung's Bixby visual search is powered by Pinterest's Lens tool. Retrieved April 16, 2017, from https://venturebeat.com/2017/03/29/samsungs-bixby-visual-search-is-powered-by-pinterests-lens-tool/

Zhao, R., & Grosky, W. I. (2002). Bridging the Semantic Gap in Image Retrieval. *Distributed Multimedia Databases: Techniques and Applications*, 13–36.

# 8 Appendices

## 8.1 Appendices A - VGG16 Structure in Keras (16 Layers)

A model containing 13 convolutional layers combined with 3 final dense layers.

| Layer ID | Layer | Parameters |
|---|---|---|
| 1 | ZeroPadding2D *(Input Image)* | **padding**=*(1, 1)*,**data_format=**(*channels=3,height=224,width=224*) |
| 2 | Conv2D | **filters**=*64*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 3 | ZeroPadding2D | **padding**=*(1, 1)* |
| 4 | Conv2D | **filters**=*64*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 5 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 6 | ZeroPadding2D | **padding**=*(1, 1)* |
| 7 | Conv2D | **filters**=*128*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 8 | ZeroPadding2D | **padding**=*(1, 1)* |
| 9 | Conv2D | **filters**=*128*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 10 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 11 | ZeroPadding2D | **padding**=*(1, 1)* |
| 12 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 13 | ZeroPadding2D | **padding**=*(1, 1)* |
| 14 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 15 | ZeroPadding2D | **padding**=*(1, 1)* |
| 16 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 17 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 18 | ZeroPadding2D | **padding**=*(1, 1)* |
| 19 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 20 | ZeroPadding2D | **padding**=*(1, 1)* |
| 21 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 22 | ZeroPadding2D | **padding**=*(1, 1)* |
| 23 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 24 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 25 | ZeroPadding2D | **padding**=*(1, 1)* |
| 26 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 27 | ZeroPadding2D | **padding**=*(1, 1)* |
| 28 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 29 | ZeroPadding2D | **padding**=*(1, 1)* |
| 30 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 31 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 32 | Flatten | - |
| 33 | Dense | **output_units**=*4096*, **activation**=*'relu'*, **name**=*'fc1'* |
| 34 | Dropout | **dropout**=50% |
| 35 | Dense | **output_units**=*4096*, **activation**=*'relu'*, **name**=*'fc2'* |
| 36 | Dropout | **dropout**=50% |
| 37 | Dense | **output_units**=*1000*, **activation**=*'softmax'*, **name**=*'predictions'* |

Table 16 VGG16 Structure in Keras

## 8.2 Appendices B - VGG19 Structure in Keras (19 Layers)

A model containing 16 convolutional layers combined with 3 final dense layers.

| **Layer ID** | **Layer** | **Parameters** |
|---|---|---|
| 1 | ZeroPadding2D *(Input Image)* | **padding**=*(1, 1)*,**data_format=**(*channels=3,height=224,width=224*) |
| 2 | Conv2D | **filters**=*64*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 3 | ZeroPadding2D | **padding**=*(1, 1)* |
| 4 | Conv2D | **filters**=*64*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 5 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 6 | ZeroPadding2D | **padding**=*(1, 1)* |
| 7 | Conv2D | **filters**=*128*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 8 | ZeroPadding2D | **padding**=*(1, 1)* |
| 9 | Conv2D | **filters**=*128*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 10 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 11 | ZeroPadding2D | **padding**=*(1, 1)* |
| 12 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 13 | ZeroPadding2D | **padding**=*(1, 1)* |
| 14 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 15 | ZeroPadding2D | **padding**=*(1, 1)* |
| 16 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 17 | ZeroPadding2D | **padding**=*(1, 1)* |
| 18 | Conv2D | **filters**=*256*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 19 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 20 | ZeroPadding2D | **padding**=*(1, 1)* |
| 21 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 22 | ZeroPadding2D | **padding**=*(1, 1)* |
| 23 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 24 | ZeroPadding2D | **padding**=*(1, 1)* |
| 25 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 26 | ZeroPadding2D | **padding**=*(1, 1)* |
| 27 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 28 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |
| 29 | ZeroPadding2D | **padding**=*(1, 1)* |
| 30 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 31 | ZeroPadding2D | **padding**=*(1, 1)* |
| 32 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 33 | ZeroPadding2D | **padding**=*(1, 1)* |
| 34 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 35 | ZeroPadding2D | **padding**=*(1, 1)* |
| 36 | Conv2D | **filters**=*512*; **stride**=*(3, 3)*, **activation**=*'relu'* |
| 37 | MaxPooling2D | **pool_size**=*(2, 2)*, **strides**=*(2,2)* |

| | | |
|---|---|---|
| 38 | Flatten | - |
| 39 | Dense | **output_units**=*4096*, **activation**=*'relu'*, **name**=*'fc1'* |
| 40 | Dropout | **dropout**=50% |
| 41 | Dense | **output_units**=*4096*, **activation**=*'relu'*, **name**=*'fc2'* |
| 42 | Dropout | **dropout**=50% |
| 43 | Dense | **output_units**=*1000*, **activation**=*'softmax'*, **name**=*'predictions'* |

Table 17 VGG19 Structure in Keras

## 8.3 Appendices C – ResNet50 Structure in Keras (50 Layers)

A model containing 49 convolutional layers combined with one final dense layer.

| **Layer ID** | **Layer** | **Parameters** |
|---|---|---|
| 1 | ZeroPadding2D *(Input Image)* | **padding**=*(3, 3)*,**data_format**=(*channels=3,height=224,width=224*) |
| 2 | Conv2D | **filters**=*64*; **stride**=*(7, 7)*, **activation**=*'relu'* |
| 3 | BatchNormalization | - |
| 4 | Activation | **activation**='relu' |
| 5 | MaxPooling2D | **pool_size**=*(3, 3)*, **strides**=*(2,2)* |
| 6 | Conv2D | **filters**=*64*; **stride**=*(1, 1)*, **activation**=*'relu'* |
| 7 | BatchNormalization | - |
| 8 | Activation | **activation**='relu' |
| 9 | Conv2D | **filters**=*64*; **stride**=*(1, 1)*, **activation**=*'relu'* |
| 10 | BatchNormalization | - |
| 11 | Activation | **activation**='relu' |
| 12 | Conv2D | **filters**=*64*; **stride**=*(1, 1)*, **activation**=*'relu'* |
| 13 | BatchNormalization | - |
| 14 | Activation | **activation**='relu' |
| 15 | Conv2D | **filters**=*64*; **activation**=*'relu'* |
| 16 | BatchNormalization | - |
| 17 | Activation | **activation**='relu' |
| 18 | Conv2D | **filters**=*64*; **activation**=*'relu'* |
| 19 | BatchNormalization | - |
| 20 | Activation | **activation**='relu' |
| 21 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 22 | BatchNormalization | - |
| 23 | Activation | **activation**='relu' |
| 24 | Conv2D | **filters**=*64*; **activation**=*'relu'* |
| 25 | BatchNormalization | - |
| 26 | Activation | **activation**='relu' |
| 27 | Conv2D | **filters**=*64*; **activation**=*'relu'* |
| 28 | BatchNormalization | - |
| 29 | Activation | **activation**='relu' |
| 30 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 31 | BatchNormalization | - |
| 32 | Activation | **activation**='relu' |
| 33 | Conv2D | **filters**=128; **stride**=*(2,2)*, **activation**=*'relu'* |
| 34 | BatchNormalization | - |
| 35 | Activation | **activation**='relu' |

| | | |
|---|---|---|
| 36 | Conv2D | **filters**=128; **stride**=*(2,2)*, **activation**=*'relu'* |
| 37 | BatchNormalization | - |
| 38 | Activation | **activation**='relu' |
| 39 | Conv2D | **filters**=512; **stride**=*(2,2)*, **activation**=*'relu'* |
| 40 | BatchNormalization | - |
| 41 | Activation | **activation**='relu' |
| 42 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 43 | BatchNormalization | - |
| 44 | Activation | **activation**='relu' |
| 45 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 46 | BatchNormalization | - |
| 47 | Activation | **activation**='relu' |
| 48 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 49 | BatchNormalization | - |
| 50 | Activation | **activation**='relu' |
| 51 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 52 | BatchNormalization | - |
| 53 | Activation | **activation**='relu' |
| 54 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 55 | BatchNormalization | - |
| 56 | Activation | **activation**='relu' |
| 57 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 58 | BatchNormalization | - |
| 59 | Activation | **activation**='relu' |
| 60 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 61 | BatchNormalization | - |
| 62 | Activation | **activation**='relu' |
| 63 | Conv2D | **filters**=128; **activation**=*'relu'* |
| 64 | BatchNormalization | - |
| 65 | Activation | **activation**='relu' |
| 66 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 67 | BatchNormalization | - |
| 68 | Activation | **activation**='relu' |
| 69 | Conv2D | **filters**=256; **stride**=*(2,2)*, **activation**=*'relu'* |
| 70 | BatchNormalization | - |
| 71 | Activation | **activation**='relu' |
| 72 | Conv2D | **filters**=256; **stride**=*(2,2)*, **activation**=*'relu'* |
| 73 | BatchNormalization | - |
| 74 | Activation | **activation**='relu' |
| 75 | Conv2D | **filters**=1024; **stride**=*(2,2)*, **activation**=*'relu'* |
| 76 | BatchNormalization | - |
| 77 | Activation | **activation**='relu' |

| | | |
|---|---|---|
| 78 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 79 | BatchNormalization | - |
| 80 | Activation | **activation**='relu' |
| 81 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 82 | BatchNormalization | - |
| 83 | Activation | **activation**='relu' |
| 84 | Conv2D | **filters**=1024; **activation**=*'relu'* |
| 85 | BatchNormalization | - |
| 86 | Activation | **activation**='relu' |
| 87 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 88 | BatchNormalization | - |
| 89 | Activation | **activation**='relu' |
| 90 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 91 | BatchNormalization | - |
| 92 | Activation | **activation**='relu' |
| 93 | Conv2D | **filters**=1024; **activation**=*'relu'* |
| 94 | BatchNormalization | - |
| 95 | Activation | **activation**='relu' |
| 96 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 97 | BatchNormalization | - |
| 98 | Activation | **activation**='relu' |
| 99 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 100 | BatchNormalization | - |
| 101 | Activation | **activation**='relu' |
| 102 | Conv2D | **filters**=1024; **activation**=*'relu'* |
| 103 | BatchNormalization | - |
| 104 | Activation | **activation**='relu' |
| 105 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 106 | BatchNormalization | - |
| 107 | Activation | **activation**='relu' |
| 108 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 109 | BatchNormalization | - |
| 110 | Activation | **activation**='relu' |
| 111 | Conv2D | **filters**=1024; **activation**=*'relu'* |
| 112 | BatchNormalization | - |
| 113 | Activation | **activation**='relu' |
| 114 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 115 | BatchNormalization | - |
| 116 | Activation | **activation**='relu' |
| 117 | Conv2D | **filters**=256; **activation**=*'relu'* |
| 118 | BatchNormalization | - |
| 119 | Activation | **activation**='relu' |

| | | |
|---|---|---|
| 120 | Conv2D | **filters**=1024; **activation**=*'relu'* |
| 121 | BatchNormalization | - |
| 122 | Activation | **activation**='relu' |
| 123 | Conv2D | **filters**=512; **stride**=*(2,2)*, **activation**=*'relu'* |
| 124 | BatchNormalization | - |
| 125 | Activation | **activation**='relu' |
| 126 | Conv2D | **filters**=512; **stride**=*(2,2)*, **activation**=*'relu'* |
| 127 | BatchNormalization | - |
| 128 | Activation | **activation**='relu' |
| 129 | Conv2D | **filters**=2048; **stride**=*(2,2)*, **activation**=*'relu'* |
| 130 | BatchNormalization | - |
| 131 | Activation | **activation**='relu' |
| 132 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 133 | BatchNormalization | - |
| 134 | Activation | **activation**='relu' |
| 135 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 136 | BatchNormalization | - |
| 137 | Activation | **activation**='relu' |
| 138 | Conv2D | **filters**=2048; **activation**=*'relu'* |
| 139 | BatchNormalization | - |
| 140 | Activation | **activation**='relu' |
| 141 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 142 | BatchNormalization | - |
| 143 | Activation | **activation**='relu' |
| 144 | Conv2D | **filters**=512; **activation**=*'relu'* |
| 145 | BatchNormalization | - |
| 146 | Activation | **activation**='relu' |
| 147 | Conv2D | **filters**=2048; **activation**=*'relu'* |
| 148 | BatchNormalization | - |
| 149 | Activation | **activation**='relu' |
| 150 | AveragePooling2D | **pool_size**=*(7, 7)* |
| 151 | Flatten | - |
| 152 | Dense | **output_units**=1000, **activation**=*'softmax'*, **name**=*'fc1000'* |

Table 17 ResNet50 Structure in Keras