KING'S College LONDON

University of London

6CCS3PRJ Final Year Project
Image-Based GPS Verification

Final Project Report

Author: Preslav Kisyov

Supervisor: Dr. Michael Spratling

Programme Title: Computer Science (BSc)

Student ID: 1726137

April 23, 2020

Abstract

This project attempts to solve the problem of image-based GPS verification. Given an image with some GPS coordinates, the software should be able to identify whether that specific image was taken at the given location.

The approach taken in this paper, in order to solve that problem, uses Computer Vision and Deep Learning methods to compare a query image with a reference one. The objective of the project is to be able to make a valid and accurate comparison, by taking an image of either a normal or panoramic type and comparing areas of it against a provided query image. If an area matches the image to be verified, then it would be safe to say the query image was taken at the given location.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Preslav Kisyov

April 23, 2020

Acknowledgements

I would like to thank and acknowledge Dr. Michael Spratling of the Faculty of Natural and Mathematical Sciences at King's College London for all the help and supervision that he has provided me with during my work. In addition, I would like to thank Professor Spratling for coming up with the topic as well as believing in my ability to complete this task.

Contents

Abbreviations

AI Artificial Intelligence.

BBP Best-Buddies Pairs.

BBS Best Buddies Similarity.

BGR Blue Green Red.

BUPM Bottom-Up Pattern Matching.

CNN Convolutional Neural Network.

DDIS Deformable Diversity Similarity.

DIS Diversity Similarity.

GPS Global Positioning System.

GPU Graphics Processing Unit.

HOG Histogram of Oriented Gradients.

IDE Integrated Development Environment.

NCC Normalized Cross-Correlation.

ResNet Residual Networks.

RGB Red Green Blue.

SAD Sum of Absolute Differences.

SIFT Scale Invariant Feature Transform.

SSD Sum of Squared Differences.

SURF Speeded-Up Robust Features.

SVM Support Vector Machines.

URL Uniform Resource Locator.

VGG Visual Geometry Group.

Chapter 1

Introduction

Image Verification is a well-known Computer Vision problem that tries to verify the similarity between two images. This project tackles that exact problem, and more specifically, verifying whether an image or an area of that image is present in another. The paper describes different approaches taken in the field of Computer Vision and presents a solution using various methods and techniques.

1.1 Background and Motivation

Nowadays, the use of Artificial Intelligence is substantially bigger than even a few years ago. From it being used to predict data in different sectors of business to self-driving cars and autonomous robots, AI, undoubtedly, plays a vital role in the modern world.

One of the most researched areas of Artificial Intelligence is Computer Vision. It is used for understanding digital images and video streams. This includes, but it is not limited to, recognizing objects as well as processing and analysing images. Computer Vision is also responsible for today's security in the likes of facial recognition and reconstruction of damaged images, through the "generative adversarial networks" technique [1]. The applications of it are infinite, thus making it, one of the most important and significant fields to explore. However, with the increasing need for fast, yet highly accurate solutions, many Computer Vision methods are being disregarded as not feasible in practice. With the introduction of Convolutional Neural Networks, Computer Vision has moved to more accurate ways of solving different problems, as it could be observed in figure 1.1. Despite that, there is still a lot of room for improvement.

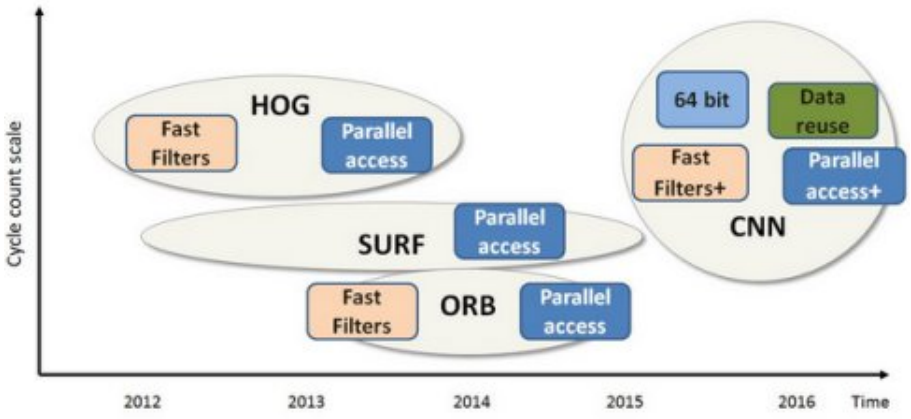


Figure 1.1: The evolution of Object Recognition in Computer Vision throughout the years. The x-axis presents the timeline for each method. The y-axis shows the computational intensity of each method. The higher on the axis, the more computationally expensive the method; Design&Reuse, Evolution of Object Recognition in Embedded Computer Vision [2]

There have been many solutions that have tried to solve the problem of Image Verification. However, the results are far from perfect and the problem continues to be one of the most difficult ones in Computer Vision. The importance of having a fast and accurate solution is becoming more immense. For instance, if a facial recognition system wrongfully verifies a person with another or if it takes too long to detect their face at all, this might lead to wrongfully accusing people of doing something they have not done. If solved efficiently, Image Verification will affect our everyday lives even more than it is currently. From having better security with robust facial recognition, removing the so-called "Fake News" from media platforms by verifying the places on the posts, to even having better navigation systems by providing images of places to navigate to. Classical solutions that try to solve these problems are Scale-Invariant Feature Transform (SIFT) [3], Speeded-Up Robust Features (SURF) [4] or Histogram of Oriented Gradients (HOG) [5]. As seen in figure 1.1, methods are evolving, trying to become more accurate with the introduction of CNNs. This new approach was first explored in [6]. However, there has always been a trade-off between accuracy and speed, and the problem remains unsolved, where a solution would provide exceptional performance and not being computationally expensive. This statement is backed by the results in table 1.1, where CNNs perform significantly slower than their predecessors.

Method	SIFT	ImageNet CNN	Unsup. CNN
Time	$2.95\text{ms} \pm 0.04$	$11.1\text{ms} \pm 0.28$	$37.6\text{ms} \pm 0.6$

Table 1.1: The table shows the feature computation times on a single CPU. These runtimes represent the computational performance of SIFT compared against two CNN methods on the same patch of 91 by 91 pixels; Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT [7]

On the other hand, other approaches like Best Buddies Similarity (BBS) [8] and Deformable Diversity Similarity (DDIS) [9] try to provide a balanced implementation between CNNs and classical techniques. They use Template Matching to match, as the name suggests, a template to another image. However, as it can be observed in figure 1.2, the accuracy is not at the level of CNN methods such as BUPM [10].

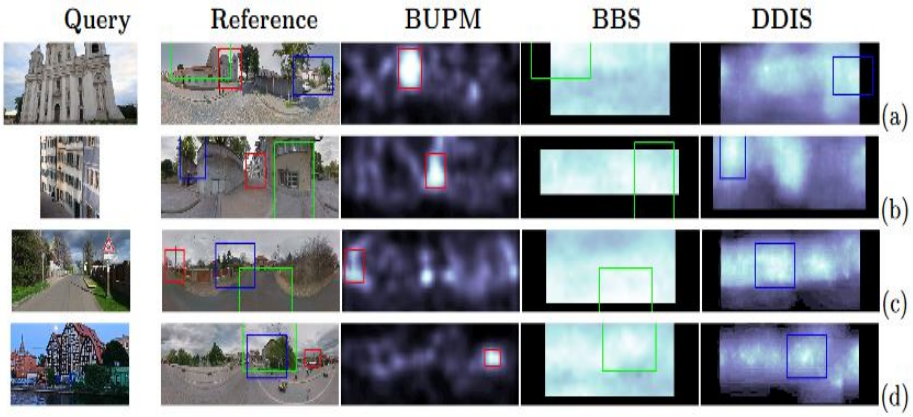
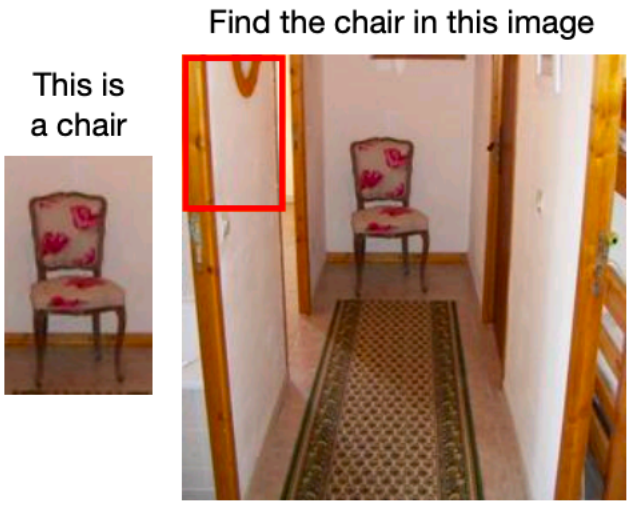
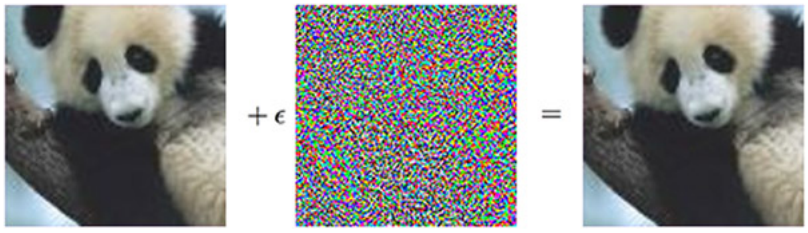


Figure 1.2: The results of performing the CNN BUPM [10] method against BBS [8] and DDIS [9] on a pair of two images. Column 1 shows the query image and column two the reference image. The task is to find the object from the query image in the reference one. Column 3 to column 5 shows the likelihood maps of performing the three different methods respectively. It could be observed, by looking at the colored boxes, that the accuracy of the CNN method outperforms the other two proposed techniques. The red box represents the BUPM [10] method, the green one is the BBS [8] and the blue one represents the DDIS [9] method. Note that zooming on the image might be required for more details; Image-to-GPS Verification Through A Bottom-Up Pattern Matching Network [10]

Even though CNNs record the highest accuracy between methods, as stated before, there is still a lot of room for improvement overall. Figure 1.3 shows the results of performing a Template Matching and a CNN method on two images. Although the images that the CNN has been tested on are essentially for classification problems, the network is a representative of the robustness of current CNNs. This applies to Image Verification because in both problems the extracted features by the network play vital part in the final overall result. This concept is further covered in Chapter 2.



(a)



(b)

Figure 1.3: Figure a) shows misclassification by using a Template Matching method on an image. The template is misplaced, rendering the results unusable [11]. Figure b) shows the performance of a CNN method on an image of a panda, where it is misclassified as another animal [12]. The pixelated image shows that by adding a disruption to the original image, CNNs can easily be tricked to give inaccurate predictions.

Although the results in the figure do not seem that problematic, methods that can produce inaccurate results even when the images are visible does prove the previous point made that AI techniques are not at the desired level of performance. For instance, if a method manages to verify two images wrongly when trying to detect ”Fake News” on media platforms, it might even result in classifying an actual news post as ”fake”. Such media ”fake” posts can be seen in figure 1.4.

Figure 1.4: This figure shows different media posts that prove the problem of Image Verification is of great importance, and that it could be even applied to social media platforms. The images are to provoke the reader of asking the question "Are these images taken at the claimed locations?" [10].

All of the points discussed above prove that Image Verification remains one of the hardest, but yet most important, problems in Computer Vision. Nevertheless, even humans are not perfect when it comes to Image Verification. However, AI methods have already proven to outperform humans in some tasks, so it is just a matter of time and resources to surpass the threshold of just "good" solutions and solve one of the most pressing Computer Vision problems.

Therefore, this paper tries to solve problems like "Fake News" in social media by using a GPS estimation approach. It also tries to explore different ways of solving the problem of Image Verification and Image Segmentation, which is the process of dividing a digital image into multiple segments. There are many applications that this solution can be applied to. One of them, more specifically, is checking whether an object, like a building on an image, is present in another image. Where the latter is obtained by given GPS coordinates. Thus, verifying both images are similar enough, leading to the conclusion that the query image was taken at the location provided.

Motivated by the above-mentioned approaches, and more specifically the BUPM [10] method, this paper tries to solve the challenge of image-based GPS verification by implementing both Deep Learning and measure-of-similarity methods like ResNets [13] and Correlation Coefficient [14]. Besides, this project has the aim to provide a fast, yet accurate solution that requires no training of a Convolutional Neural Network.

1.2 Report Structure

- Chapter 1:
  - This section of the report covers the introduction as well as the background and motivation behind the project.
- Chapter 2:
  - This chapter outlines an in-depth literature review on the problem of Image Verification in Computer Vision. It covers different methods and techniques, as well as such that are used in this project. In addition, the chapter describes their benefits and applications.
- Chapter 3:
  - In Chapter 3, a new combined Image Verification method is described, which adopts both Deep Learning and measure-of-similarity techniques. The methods in use, as well as the model itself, are covered and described in detail.
- Chapter 4:
  - Chapter 4 covers an in-depth testing and evaluation of the proposed model. Moreover, a comparison between other existing methods and techniques is conducted. This includes approaches like BUPM [10] and SIFT [3].
- Chapter 5:
  - This chapter outlines possible future improvements as well as different research topics that can be further explored.
- Chapter 6:
  - Chapter 6 covers the legal actions taken into account during the implementation process of this project, as well as the ethical issues that apply to it.

Chapter 2

Literature Review

2.1 Image Processing

Digital images can contain a lot of information. That data is essentially represented as three-dimensional points, where each number in these points is in the range between *0* and *255* as it could be seen in figure 2.1. The points are represented as (R, G, B), and by giving numbers to each position of a point, different color combinations are created. For instance, the combination of (225, 255, 255) represents the white color.

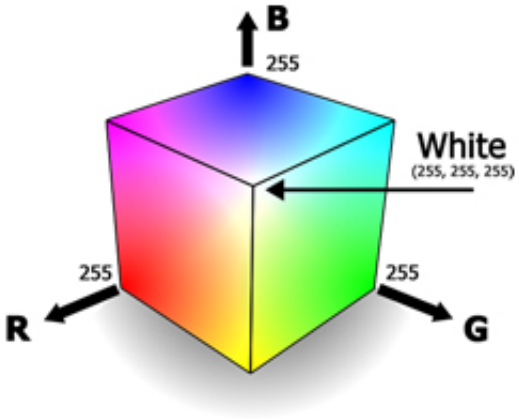


Figure 2.1: This figure shows a representation of an RGB cube, showing that each point corresponds to a different color. That is, depending on the combination of the three main additive primary colors - red, green and blue; The New Wave of Color Calibration Technology: Cube Calibration [15]

The best way to use an image to its full potential, without having to go over large amounts of data, is to process the image by enhancing and analysing it. Image Processing does that with the help of different algorithms.

2.1.1 Feature Detection

This is the first step of Image Verification. Distinguishing between the relevant and irrelevant parts of an image is a low-level type of Image Processing, called Feature Detection. It includes methods and algorithms that deal with finding only the important bits of information in an image. By making a local decision, the algorithms can decide whether there is an interesting point on an image or not. These exact points are also referred to as *features*, which could be of type edges, corners, blobs or ridges. Once detected, a vector of features can be used for further image manipulation. The process of getting these vectors is called Feature Extraction, which is described more thoroughly in the next subsection.

2.1.2 Feature Extraction

It must be noted that in Image Processing, the Feature Extraction process is merely reducing the information of an image and grouping detected features into feature vectors. Once a Feature Detector finds the points of interest, Feature Extraction is performed. The detected points of interest are grouped into vectors, which can be later used for Image Verification. After reducing the dimensions, algorithms can perform Similarity Matching to find how similar two of these feature vectors are. By doing so, the risk of matching unimportant features is mitigated. Also, the computation time is reduced significantly as algorithms will use only that small set of features. Classical Feature Detectors like SIFT [3] or HOG [5] are considered to be manual Feature Extraction algorithms. Meaning, they need to specifically detect features like edges, blobs, etc. On the other hand, CNNs are automatic Feature Extractions, that find patterns in images to describe features. This is done by adjusting weights between layer connections. Such Feature Extraction CNNs are covered in later sections. The following section describes such feature detection and extraction algorithms, as well as the benefits of performing them.

2.2 Feature Detection & Extraction Algorithms

The methods described in this section are one of the first that have been used to find points of interest in an image, and eventually, compare these points to verify it against another image. Such techniques are in the basis of more sophisticated CNNs, that later prove to be an important part of the proposed method by this paper. It is necessary to understand how such detectors work, as different combinations of the methods implemented have been used in this project. This is, either as part of the image processing or the actual Feature Extraction

process.

2.2.1 Canny Edge Detector

The Canny algorithm is a detector that finds edges. Such features represent points between two regions in an image with a strong gradient magnitude. The gradient is the change in the intensity of a pixel in an image. It is represented as a vector, where the partials are derivatives with respect to x and y, as it could be seen in formula (2.1) [16].

$$\vec{f} = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.1}$$

In addition, the magnitude is represented as $\sqrt{g_u^2 + g_x^2}$. Finding the intensity gradients is one out of 5 steps that the Canny algorithm follows. The steps are:

1. Apply Gaussian filter [17]. This step smooths the image and removes the noise to prevent false detection. It blurs edges and reduces contrast by applying a Gaussian mask to each pixel of an image. The size of the mask is determined by the Gaussian filter kernel [17]. The effect of applying such a filter can be seen in figure 2.2.

2. Finding the intensity gradients. As stated before, finding the gradient magnitude is an important step in edge detection.

3. Applying edge thinning technique in order to find the biggest edge. The technique is called non-maximum suppression as per [18] and it helps to set all gradient values to 0 except the local maxima.

4. Filtering out edge pixels with a weaker gradient value. This step performs a double threshold. Meaning, if a gradient value is smaller than the high threshold and bigger than the low one, it is marked as a weak or "bad" value, where if a gradient is smaller than the low threshold, it is suppressed.

5. The last and final step is to remove weak edges caused by true edges in the image. This is done by doing hysteresis [19]. It is essentially another threshold method, where any pixel above the high threshold is turned white. In addition, the surrounded pixels are also searched and if their values are greater than the lower threshold, the pixels are turned white as well. This method results in preserving weak edges if there is at least one strong edge pixel involved in the hysteresis analysis [19].

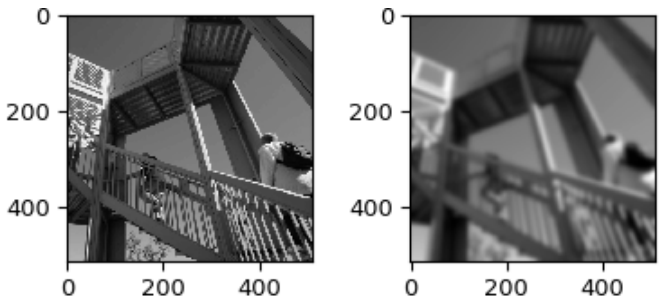


Figure 2.2: The results of applying Gaussian filter [17] to an image; SciPy Gaussian Filter Library [20]

In conclusion, the Canny edge detector can effectively remove noise in an image, detect edges and give good localization in various environments. An image with an applied Canny edge detector can be seen in figure 2.3. However, it is an algorithm that is difficult to implement and compute, due to its complexity.

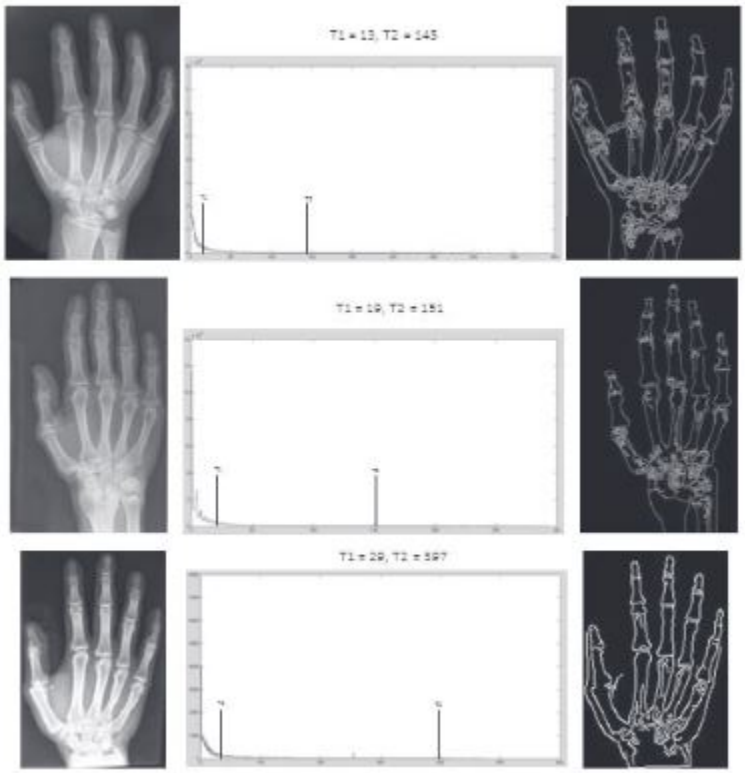


Figure 2.3: The results of applying the Canny edge detector on several images. The diagrams represent the low and high thresholds used in the algorithm; An Improved Canny Edge Detection Algorithm Based on Type-2 Fuzzy Sets [19]

2.2.2 Harris & Stephens Corner Detector

This detector is used to find corner features. These are points with rapid changes in directions. The word "corner" is not necessarily in the traditional sense, but rather it means an interest point in an image. The basic idea behind that algorithm is to be able to recognize a point by looking through a small window as it can be observed in figure 2.4.

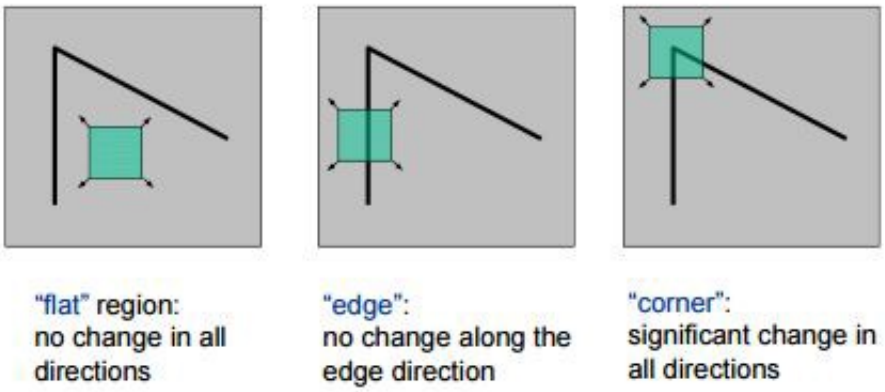


Figure 2.4: The basic idea of Harris & Stephens Corner Detector; Introduction to Harris Corner Detector [21]

The detector uses small windows around each pixel on a grayscale image and tries to identify uniqueness by shifting these windows and measuring the change that occurs in the pixel values. The change function is defined by the *Sum of Squared Differences* (SSD) of the pixels before and after the shift has been performed for eight directions. The SSD is a similarity measure that is represented as $\sum_{i}^{n}(a_i - b_i)^2$, where a and b represent two vector inputs. However, the algorithm takes an image path over an area *(u, v)* and shifts it by *(x, y)*. The weighted sum of squared differences of these two patches, as per equation 2.2 [22], is taken and maximised for corner detection. This is done by Taylor expansion [23] approximation $I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y$, which results in equation (2.3) [22], where the A is a tensor represented as in (2.4) [22], and the *w(u, v)* is the type of sliding window.

$$S(x,y) = \sum_{u}\sum_{v} w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{2.2}$$

$$\begin{aligned} S(x,y) &\approx \sum\sum w(u,v)(I_x(u,v)x + I_y(u,v)y)^2 \\ S(x,y) &\approx (x \quad y)A\begin{pmatrix} x \\ y \end{pmatrix} \end{aligned} \tag{2.3}$$

$$A = \sum_{y} \sum_{v} w(u,v) \begin{bmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{bmatrix} = \begin{bmatrix} < I_x^2 > & < I_x I_y > \\ < I_x I_y > & < I_y^2 > \end{bmatrix} \tag{2.4}$$

The detector characterizes a corner by a large variation of S in all directions of the vector *(x y)*. By computing A $\lambda 1$ and $\lambda 2$ can be extracted, which are the eigenvalues of the matrix. Given these values, the following rules can be defined:

1. When $\lambda 1$ and $\lambda 2 \approx 0$, *(x, y)* has no points of interest.
2. When $\lambda 1 \approx 0$ and $\lambda 2$ has a large positive value, *(x, y)* has an edge.
3. When $\lambda 1$ and $\lambda 2$ has large positive values, *(x, y)* has a corner.

In conclusion, the Harris & Stephens Corner Detector [24] can identify points of interest on an image with good accuracy. However, computing it could be quite expensive. That is especially when calculating the eigenvalues, as it requires computing a square root. Luckily, the authors' Harris and Stephens suggest a solution to that problem in their paper [24].

2.2.3 Scale-Invariant Feature Transform (SIFT)

SIFT [3] is a corner and region feature detector algorithm. It identifies images even when scaled, hence its name Scale-Invariant. The algorithm is based on the Difference of Gaussian [25] edge detector, which is the subtraction of a blurred image from another blurred image. The first stage of the algorithm is to apply the previously described Gaussian filter [17] at different scales. Then subtraction of the Gaussian-blurred images is performed. This process of subtracting is done for different octaves, or levels, as shown in figure 2.5.

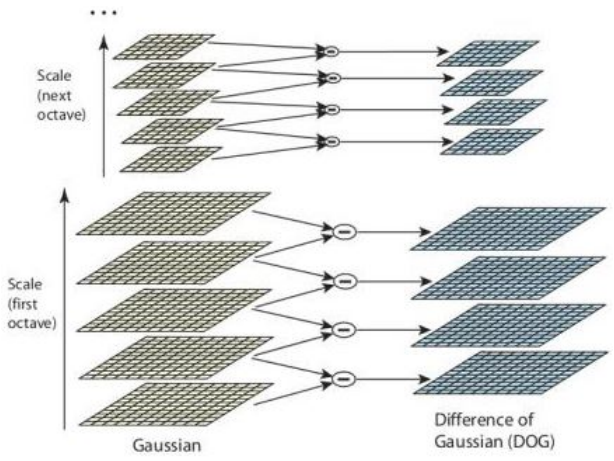


Figure 2.5: The process of performing the Difference of Gaussian for different octaves of an image in a Gaussian Pyramid; Distinctive Image Features from Scale-Invariant Keypoints [3]

After this process is complete, potential key points are found, by searching for local minima/maxima of the Difference of Gaussian [25] images across scales. This is done by comparing pixels to their 8 neighbors on the same scale and their 9 neighbors at each different neighboring scale. The process of finding key points can be observed in figure 2.6.

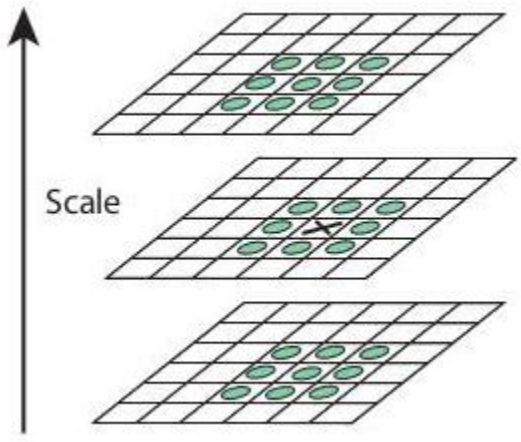


Figure 2.6: The process of keypoint selection by comparing pixels at different octaves; Distinctive Image Features from Scale-Invariant Keypoints [3]

However, this method produces a lot of key points, some of which are unstable. To increase the accuracy, each key point is compared against a threshold. Also, an orientation is assigned to every point. This step is vital, as it is key to achieving invariance to image rotation. The process produces a histogram with 36 bins for every point, where each bin covers 10°. The peaks in this histogram are also the dominant orientations, which are assigned to the key point. The orientation corresponding to the highest peak and to all local peaks within 80% of that highest peak are assigned. This could lead to multiple orientations belonging to one key point. In that case, an identical key point is created at the same location for each additional orientation assigned. The entire key point selection process can be seen in figure 2.7.

The final step of the algorithm is to group blocks of these points into a feature key point vector. The same process can be performed on another image and then having both key point vectors compared by identifying their closest neighbors. Thus, resulting in finding the similarity of features between two images.

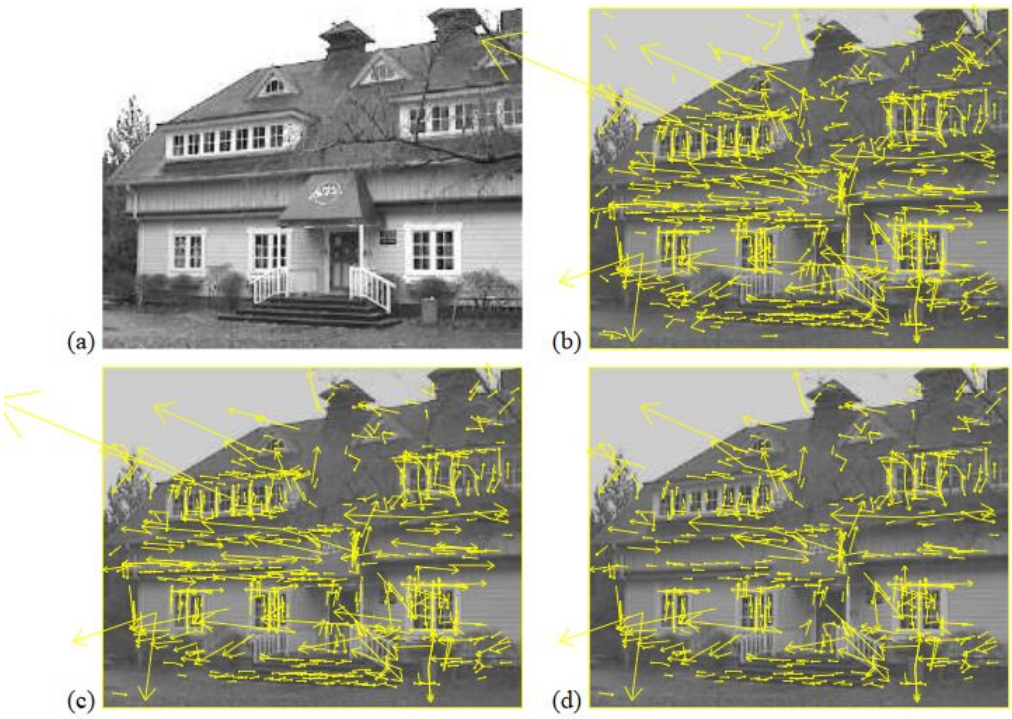


Figure 2.7: The process of keypoint selection in 4 stages. Stage a) shows the original image. Stage b) presents 832 key points locations at the minima/maxima of the Difference of Gaussian [25] function. Stage c) presents the image after applying a threshold, which results in reducing the points to 729. The last stage d) shows the final image of 536 remaining key points after an additional threshold on the ratio of principal curvatures has been applied; Distinctive Image Features from Scale-Invariant Keypoints [3]

2.2.4 Speeded-Up Robust Features (SURF)

SURF [4] is another feature detector and descriptor that was partly inspired by the above-mentioned SIFT [3] algorithm. However, instead of using the Difference of Gaussian [25], the algorithm uses square-shaped filters. An advantage of that approximation over the one used in SIFT [3], is that the former makes filtering the image much faster with the use of integral images. An integral image is a data structure that efficiently generates the sum of values in a rectangular grid. It is also known as summed-area tables. The formula used can be seen in equation (2.5) [26].

$$S(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \tag{2.5}$$

The detection step is solemnly based on a blob detector that uses a Hessian matrix [27] to find points of interest. The determinant of that matrix is used for measuring the local change

around points, as well as selecting the scale. The points with the maximal determinant are chosen. The Hessian matrix [27] is represented as equation (2.6) [26]. Where p represents a point in an image with coordinates *(x, y)*, and σ is the scale. $L_x x(p, \sigma)$ is the convolution of the second-order derivative of Gaussian at point p. In this case, a convolution is a mathematical operation that results in a function expressing the modification of one function by another.

$$H(p, \sigma) = \begin{pmatrix} L_{xx}(p, \sigma) & L_{xy}(p, \sigma) \\ L_{yx}(p, \sigma) & L_{yy}(p, \sigma) \end{pmatrix} \tag{2.6}$$

SURF's scale space is divided into several octaves. It is analyzed by up-scaling rather than reducing the size of the image. To localise interest points in the image over scales, the algorithm applies a non-maximum suppression in a *3x3x3* neighborhood. In addition, a scale interpolation is performed on the maxima of the determinant of the Hessian matrix [27]. This is so, because the difference in scale between the different layers is considered to be relatively large.

As in SIFT [3], SURF [4] also assigns orientation in order to achieve rotational invariance. A wavelet response in both x- and y- directions is computed. These responses are further weighted by a Gaussian function. A local orientation vector is computed by estimating the dominant orientation. That orientation is the sum of the horizontal and vertical responses within a sliding window. The longest orientation vector defines the orientation of the point.

In conclusion, SURF [4] proves to be a faster and more robust solution to the previously described SIFT [3], as it can be seen in table 2.1 and in figure 2.8.

	U-SURF	SURF	SURF-128	SIFT
time (ms):	255	354	391	1036

Table 2.1: The computation times of different SURF variations against SIFT [3]; SURF: Speeded Up Robust Features [4]

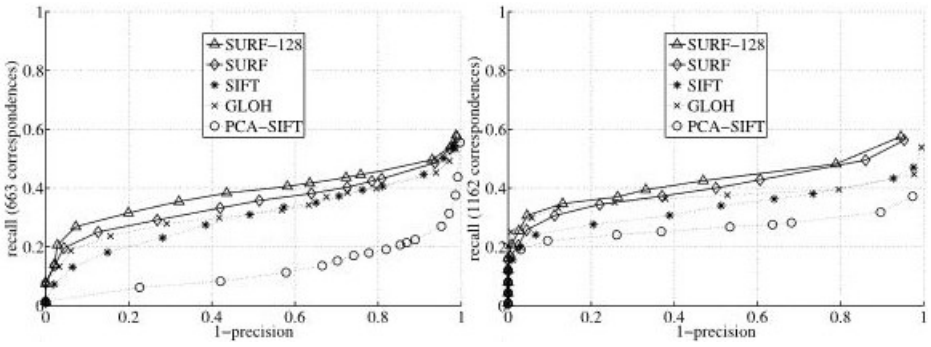


Figure 2.8: The precision metrics of SURF compared against other methods on different view angles, changes in scale or brightness; SURF: Speeded Up Robust Features [4]

2.2.5 Histogram of Oriented Gradients (HOG)

HOG [5] is a feature descriptor for object detection. The idea behind that algorithm is that an object or a shape can be described by the distribution of intensity gradients or the directions of edge points. An image is divided into cells or regions, for which a histogram of gradient directions is created. The descriptor itself is the concatenation of these histograms.

Instead of pre-processing the image, the HOG [5] algorithm computes the gradient values by applying a 1-D centered derivative mask in one or both directions - horizontal or vertical. These masks are small filters with kernels [-1, 0, 1] and [-1, 0, 1]. Using more complex masks was found to perform worse when detecting people in images, as per [5].

The next step in the algorithm is to create those cell histograms mentioned above. Each pixel in every cell has a weighted vote for an orientation histogram channel. That is based on the gradient values that have been computed. These histogram channels are spread either between *0* and *180* degrees or *0* and *360* degrees. This distribution is dependent on whether the gradient is "unsigned" or "signed". The HOG [5] algorithm uses "unsigned" gradients, meaning that a gradient arrow and the one that is 180° opposite are considered to be the same.

Following the steps described above, the algorithm locally normalizes the gradients. This is done in order to account for illumination and contrast changes. The created cells are grouped into big connected blocks. These blocks usually overlap, meaning that cells can contribute more than once to the final descriptor. There are two main block geometries. The first type is the *R-HOG* blocks, which are represented as square grids. They are computed in dense grids without orientation alignment or multi-scaling. The second type is the *C-HOG* blocks. These blocks are circular and can be either with a single, central cell or with an angularly divided central cell. Besides, the *C-HOG* blocks contain cells with several orientation channels.

After grouping the cells into blocks, HOG [5] implements four different block normalization factors. They can either be L2-norm, L2-hys, L1-norm or L1-sqrt. All of these normalization forms can be observed in equations (2.7) [28]. In addition, their performance is quite similar, with L1-norm proving to be slightly less reliable, as per [5]. However, in [5] it was tested that normalizing the blocks significantly improved performance over non-normalized data.

$$L2-norm: f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$

L2-hys: Same as L2-norm but the v value is limited to a maximum of 0.2 (2.7)

$$L1 - norm : f = \frac{v}{(\|v\|_1 + e)}$$
$$L1 - sqrt : f = \sqrt{\frac{v}{\|v\|_1 + e}} \tag{2.7}$$

In conclusion, the HOG [5] algorithm can be used for object recognition by providing features to different algorithms. In terms of advantages, it is invariant to geometric transformations. This makes it suitable for human detection even while in motion. However, the descriptor is not invariant to object orientation. The result of performing the HOG [5] algorithm can be seen in figure 2.9.

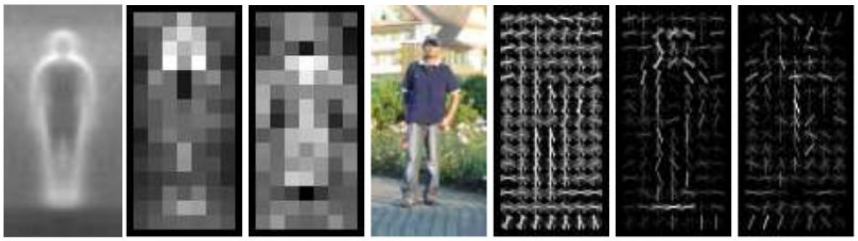


Figure 2.9: The results of performing the HOG detectors. Image a) reflects the average gradient picture over the training examples. Image b) shows the maximum positive SVM weight in the block centered on the pixel. Image c) displays weights that are negative to SVM. Image d) is a test image. Picture e) is computed with R-HOG. And finally images f) and g) show the R-HOG descriptor balanced by the positive and negative SVM weights; Histograms of Oriented Gradients for Human Detection [5]

2.3 CNNs for Feature Detection & Extraction

Convolutional Neural Networks, as stated before, play a vital role in the implementation of this project. The proposed model uses a CNN to perform Feature Detection as well as Feature Extraction. CNNs' performances may vary between different types of data, and that is why, for this project, it is vital to understand the difference between networks.

CNNs are fully connected networks. It means that every neuron in one layer is connected to all neurons in the next layer. As mentioned before, CNNs have become extremely popular when it comes to image analysis, segmentation or verification. Similar to some Feature Detection algorithms, CNNs can also be a shift or space invariant. Their design could be quite complex, consisting of an input and output layer, as well as multiple hidden layers. These hidden layers have their inputs and outputs masked by an activation function. Typically, such layers consist of a series of Convolutional layers. Furthermore, an activation layer is present. Usually, it is followed by more additional layers. Such could be either pooling layers, fully connected layers

or normalization layers. An overview of a CNN architecture can be seen in figure 2.10.

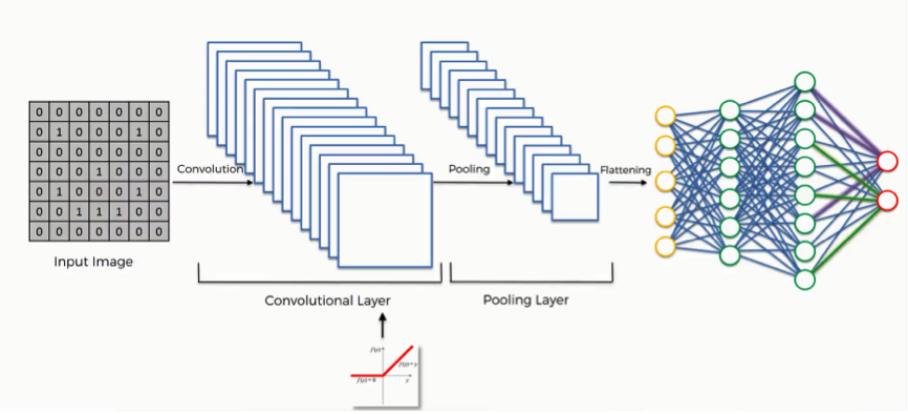


Figure 2.10: A typical CNN architecture model with convolutional, pooling and flattening layers; Convolutional Neural Networks (CNN): Summary [29]

CNNs can also apply different filters to detect and extract points of interest from an image. The first layer of a CNN, which is always a Convolutional Layer, describes low-level features. Going deeper and through more Convolutional Layers results in extracting more complex features, as the input gets different filters applied on top. In addition, a Convolutional Layer's output is another layer's input. This results in each layer describing the locations of certain features of an image.

Combined with their relatively small pre-processing process, these deep neural networks hold a significant advantage over the classical algorithms covered in the previous section. However, they come with certain disadvantages as well. CNNs tend to overfit because of their multi-layer connectivity. Also, a substantial amount of data could be required in order to provide reasonable accuracy even for small networks. Despite that, to overcome these disadvantages, this project uses a pre-trained CNN to extract image features. The specifics of the CNN used are covered in the upcoming subsections. Furthermore, different CNN methods are examined closely, providing an overview as well as the pros and cons of using them.

The intuition behind original CNNs is to progressively learn more features with every layer. It was thought that the more layers a CNN has, the better the results. Despite that being, in theory, correct, CNNs has proven to have a threshold of possible layers being stacked as it can be observed in figure 2.11.

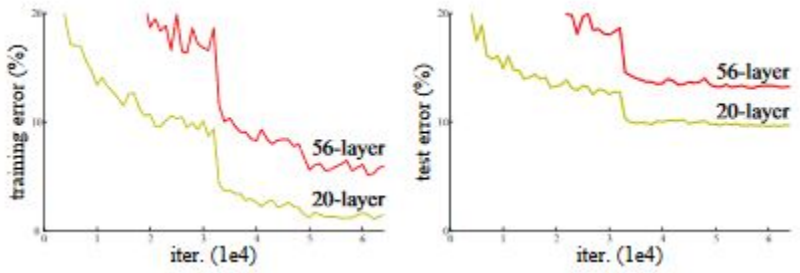


Figure 2.11: The training (left) and testing (right) errors on the CIFAR-10 dataset with respectively 20 and 56 layers. It could be observed that the network with 56 layers has higher error; Deep Residual Learning for Image Recognition [13]

Residual Networks or ResNets [13] solve that exact issue. They introduce a new layer called Residual Block. This block implements a Skip Connection step, also referred to as identity mapping. It puts the output from the layer before to the layer ahead as it can be observed in figure 2.12.

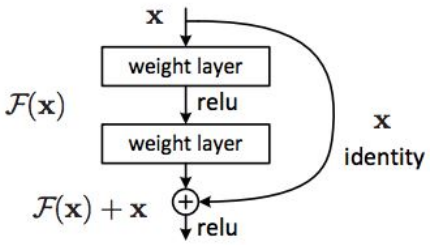


Figure 2.12: The architecture of a Residual Block Layer. *F(x)* is a residual mapping of the original mapping *H(x)* and subtracting *x* ; *F(x)* = *H(x)* - *x*. Therefore, *F(x)* + *x* denotes the original mapping; Deep Residual Learning for Image Recognition [13]

It must be noted that sometimes the dimension of *x* and *F(x)* can be different, and that is why the identity mapping uses a linear projection to expand its channels. This leads into having both *x* and *F(x)* as input to the next layer. The formula $y = F(x, \{W_i\}) + W_s x$ is the result of applying this linear projection, as per [13]. With the addition of a Residual Block, deeper networks were able to be trained. This leads to decreasing error rates compared to the "plain" networks, which can be seen in table 2.2. These "plain" networks are such that they mostly have 3x3 filters and have two very specifically defined rules. The first one states that the layers must have the same number of filters for the same output map of features. And the second rule suggests that the number of filters must be doubled if the size of the features map is halved. That is done in order to preserve the time complexity of each layer. The architecture of these networks is covered with more details in [13]. Based on the "plain" networks described above, ResNet [13] inserts a Skip Connection in order to overcome the original limitations. Residual

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 2.2: A table showing the error rates of different Residual Network models on the ImageNet dataset [30]. From the data above it can be concluded that deeper networks perform better. The VGG-16 [31], which is also described in [13], GoogleNet [32] and PReLU-net [33] networks have been used in the test shown and are described in-depth in their corresponding papers; Deep Residual Learning for Image Recognition [13]

Networks can be used to extract features from images and classify them, as per [13]. Because ResNets [13] are essentially CNNs, as mentioned above, they can be used to detect, extract or segment images. Such Residual Networks and CNNs are described below.

2.3.1 ResNet50/101/152

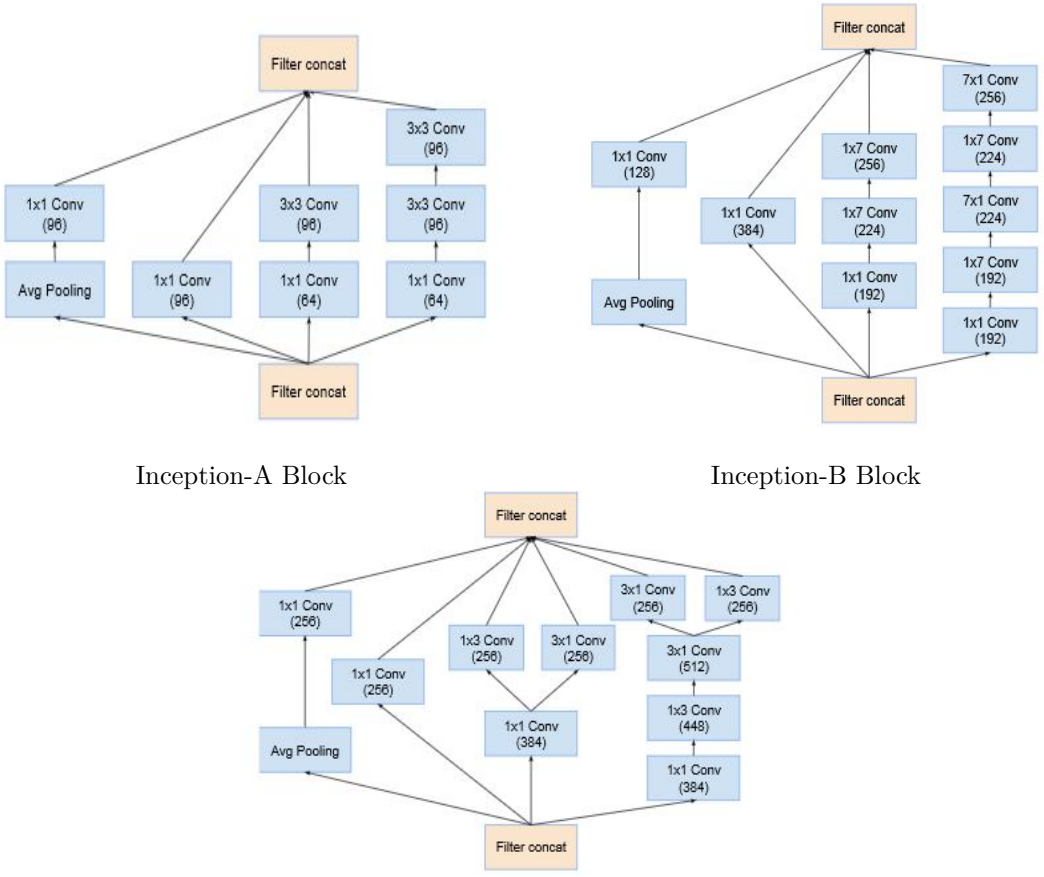
This specific residual network, as the name suggests, has implementations of 50, 101 and 152 layers. The architecture follows the one described above, with the only difference being the number of layers. Using Residual Blocks, ResNet [13] manages to increase the depth of the network without increasing the complexity. The 50-layer implementation replaces every 2-layer block from the initial implementation with a 3-layer block. Furthermore, the 101- and 152- layer implementations simply use more 3-layer blocks. Table 2.3 depicts how the different implementations of the ResNet [13] have been constructed. As table 2.2 shows, ResNet [13] improves upon other implementations as well as its "plain" counterpart.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 2.3: This table shows the Building blocks for different architectures of the ResNet. *conv3_1, conv4_1, and conv5_1* are used to perform down-sampling; Deep Residual Learning for Image Recognition [13]

2.3.2 Inception

Inception [34] differs from the ResNet [13] model described above. Instead of making the network as deep as possible, Inception [34] makes it wider. This is done to address the issue of objects in images of varying sizes and locations. In practice, a bigger kernel is preferred for information distributed more globally where a smaller one is usually used for information that is distributed more locally. Instead of settling for only one, Inception [34] implements filters of multiple sizes on the same level, thus making the network wider rather than deeper. The two most recent distributions of that CNN are Inception-v4 (Version 4) and Inception-ResNet, as described in [34]. Inception-v4 [34] introduces "pure" Inception blocks. These blocks are more uniformed than the ones in previous versions of the model. The architecture of these blocks can be seen in figure 2.13.



Inception-A Block

Inception-B Block

Inception-C Block

Figure 2.13: This figure presents the architecture of Inception-A Block, Inception-B Block, and Inception-C Block respectively; Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning [34]

Besides, the initial set of operations, before the above-mentioned blocks, has been modified. These operations are referred to as the "stem" of the network, as per [34]. By doing all these changes with the addition of Reduction Blocks, Inception-v4 [34] boosts its performance compared to its predecessors. Two Reduction Blocks are used to change the height and width of the grid.

On the other hand, Inception-ResNet [34] introduces several changes to the previously-described model. Residual connections, as described previously, are implemented in order to create a hybrid model. For these residual connections to work, the input and output after convolution have to be of the same dimension. Therefore, Inception-ResNet [34] adds additional Convolutional Layers with kernels 1x1 after the original ones. Furthermore, the pooling operations are removed from the Inception blocks [34]. However, they are replaced with the residual connections, leaving only the Reduction blocks to do the pooling operation.

In conclusion, the results described in [34] suggest that the Inception model [34] has better performance to the pure Residual Network ResNet152 [13]. However, the Inception [34] CNN has a trade-off between being computationally expensive or with better recognition performance, that is between different versions of the model [34]. The architecture of both Inception-v4 and Inception-ResNet [34] can be observed in figure 2.14.

Inception-v4 Inception-ResNet

Figure 2.14: The network architecture of Inception-v4 and Inception-ResNet respectively; Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning [34]

2.4 Similarity Matching Techniques

The previous sections described the first step of Image Verification, as well as various techniques and methods used for feature detection and extraction. This current section covers the next step of verifying images. It describes different techniques used to extract similarity values from two images and making verification decisions based on those values. It must be noted that the methods covered, either act as or use feature detectors. This is unlike the previous sections, where the techniques in use had to be implemented in conjunction with other methods to verify an image.

Similarity matching is used to make predictions based on whether how similar two images or objects are. It compares feature vectors using different algorithms and techniques. However, because images can have similar features, algorithms can be deceived. For this project, comparing buildings can be a difficult task, especially if objects have similar shapes, edges, etc. Thus, choosing a robust enough method is crucial. The following subsections describe different methods used in similarity matching, as well as the benefits of using them.

2.4.1 Best-Buddies Similarity (BBS)

BBS [8] addresses the issue of template matching. Where a bounding box containing a region of interest is used to have a path found in a source image. However, there is the problem of taking all features into account even when there are changes in the background between the template and the target image. Thus, BBS [8] proposes a solution to these limitations.

The main idea behind this technique is to measure the similarity between two groups of points using properties of the Nearest-Neighbor matches. As the name of the method suggests, it uses only a subset of these points, also known as Best-Buddies Pairs (BBPs) [8]. BBP is a pair of points where each point has a corresponding nearest neighbor in the same set of points. As defined in [8], since BBS takes into account only BBPs, the method is robust for different outliers as well as accurate for matching features from the same distribution. Pixels from the same distribution are called inliers since their adjacent points are more likely to be detected. Where pixels from various environments are considered to be outliers. The BBS [8] method proves to be more robust and more accurate than other methods, as described by the [8] analysis. That is, especially when the target's background differs from the template's one. The performance of this technique can be seen in figure 2.15.

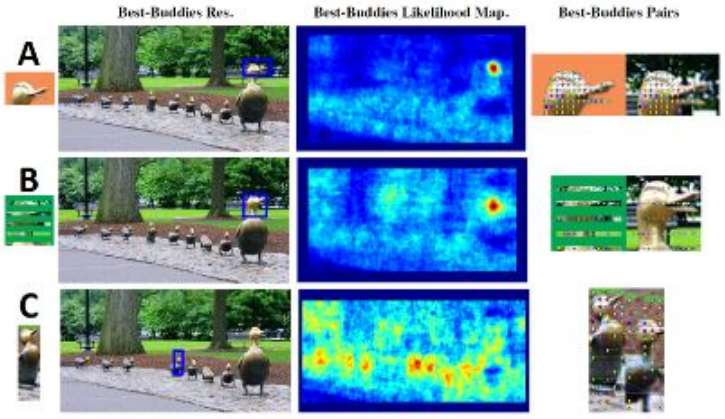


Figure 2.15: This figure presents the result as well as the different steps' representations from performing the BBS method. The first column shows the template, where the second shows the target image with the matched BBS template. The third column is a likelihood map of well-localized distinct points. The final column shows the BBPs; Best-Buddies Similarity for Robust Template Matching [8]

2.4.2 Deformable Diversity Similarity (DDIS)

Similarly to the above-mentioned BBS [8] method, DDIS [9] tackles the challenging task of template matching by using Nearest Neighbor as measure of similarity. A noticeable difference between both methods is that DDIS [9] proves to be computationally less expensive as tested in the paper on *"Template Matching with Deformable Diversity Similarity" [9]*. Furthermore, the method accounts for the deformation implied by the Nearest-Neighbor matches, making it a robust technique that can operate in different environments. In addition, the algorithm follows two main ideas:

1. Account for the diversity of matching pairs.

2. Account for the amount of deformation.

DDIS [9] accounts for both diversity and similarity by examining pairs of both template and target points. The method reaches a maximum of 1 when a pair is found to be a unique Nearest Neighbor, and lower value when the Nearest Neighbor pair is shared by other points. Moreover, every pair is weighted by the length of its implied deformation vector in order to account for that as well.

To summarize, DDIS [9] is a robust template matching algorithm that is based on the previously explored BBS [8]. It accounts for the diversity of matches as well as the deformation implied by them, as described above and in [9]. A comparison in the performance of the method against BBS [8] and DIS [9] can be observed in figure 2.16. The DIS [9] method mentioned

above is a simplified version of the explored DDIS [9] algorithm. The main difference is that it does not account for any deformations.

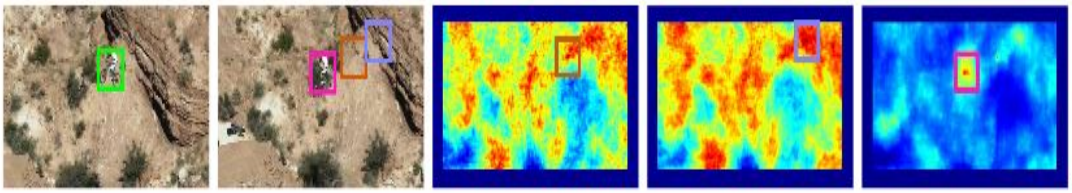


Figure 2.16: This figure shows the performance windows of BBS [8] (in orange), DIS [9] (in blue) and DDIS [9] (in pink) methods. The template is represented in green color in the first image. The last three columns show the likelihood of the three methods used respectively. It can be observed that the DDIS [9] map is more robust than the others; Template Matching with Deformable Diversity Similarity [9]

2.4.3 Bottom-Up Pattern Matching (BUPM)

The BUPM [10] method uses a CNN module in order to accurately verify an image, unlike previously described methods. This approach is divided into three steps.

Step 1, like most methods, is to detect and extract features from two images. This step proves to be useful for this very project as well. BUPM [10] implements a new approach compared to BBS [8] and DDIS [9], that uses a ResNet [13] to extract features. Moreover, it uses the previously described ResNet50 model [13].

Step 2 uses the Cosine Similarity measure in order to detect similar features between both newly extracted feature vectors. Also, it implements a CNN that performs pooling to detect image masks, as it can be seen in figure 2.17. The CNN itself is trained on a big dataset and adopts the architecture shown in figure 2.18.

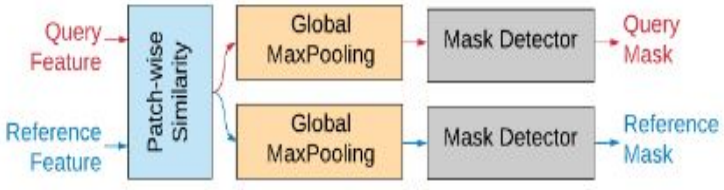


Figure 2.17: This image shows an overview of the steps taken after the features have been extracted. The Patch-wise Similarity module uses Cosine Similarity to get similarity patches between both vectors. Then this is followed by a CNN that extracts the masks for both images; Image-to-GPS Verification Through A Bottom-Up Pattern Matching Network [10]

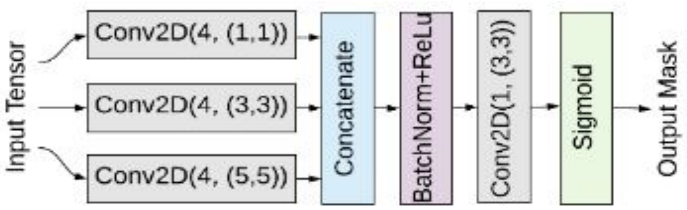


Figure 2.18: The figure shows the architecture choices that are taken for the Mask Detector module in 2.17; Image-to-GPS Verification Through A Bottom-Up Pattern Matching Network [10]

The last and third step is for the method to verify if the two images are similar enough. This is done with the help of 3 Dense Layers followed by the Sigmoid function. The result of this verification module is in the range of 0 or 1. This module is trained separately to the Mask Detector one on a different dataset.

Therefore, following these three steps, BUPM [10] achieves an accuracy of around 0.78% on the *Shibuya* dataset, as per [10]. The architecture of the method, as well as the result of performing it, can be seen in figure 2.19.

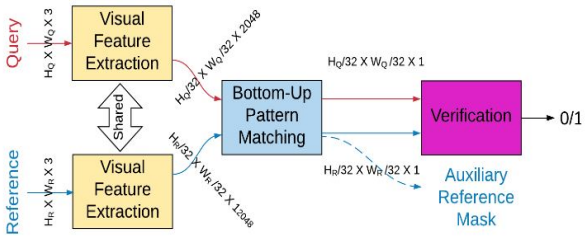


Figure a)

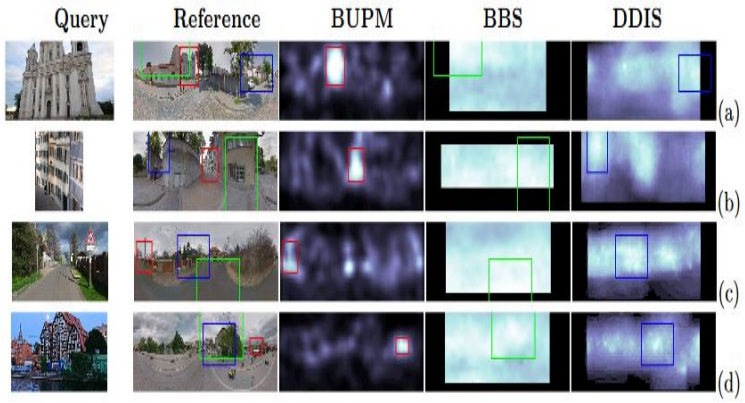


Figure b)

Figure 2.19: Figure a) shows the architecture of the BUPM [10] method. It includes the Feature Extraction process, the Bottom-Up Pattern Matching, which includes detecting the masks of both images, and the Verification module. Figure b) shows the results and likelihood maps of applying the BUPM [10] method on two images against BBS [8] and DDIS [9] methods. Note that zooming on the image might be required for more details; Image-to-GPS Verification Through A Bottom-Up Pattern Matching Network [10]

2.5 Similarity Measures

Similarity measures are widely used in Computer Vision. They are functions that determine the similarity value between two elements. The following subsections cover such measures.

2.5.1 Normalized Cross-Correlation (NCC)

Cross-Correlation [35] is a similarity measure of two vectors or series. It tests the way one feature is displaced relative to another, which is essentially the convolution of two functions, as covered previously. In Computer Vision, the Cross-Correlation is described as performing the Dot product [35] on two vectors a and b, resulting in the equation $\sum_i a_i b_i$. In cases when amplitudes or dimensions are different, Cross-Correlation must be normalized. This results in NCC [36]. The equation is shown in (2.8). By normalizing the Cross-Correlation [35] formula, it becomes usable for images that differ in dimensions. The equation can also be referred to as Cosine Similarity.

$$\frac{\sum_i a_i b_i}{\sqrt{(\sum_i a_i^2)}\sqrt{(\sum_i b_i^2)}} \tag{2.8}$$

2.5.2 Correlation Coefficient

Correlation Coefficient [14] is very similar to the NCC method [36]. The difference is that instead of simply using the two vectors a and b, Correlation Coefficient uses the difference between these vectors and their mean. This results in Pearson's Correlation Coefficient [14]. The equation is shown in (2.9) [14]. The result of performing that formula is between +1 and -1, where +1 represents a perfect correlation, 0 shows no correlation, and -1 is a perfect negative correlation.

$$\frac{\sum_i (a_i - \overline{a})(b_i - \overline{b})}{\sqrt{\sum_i (a_i - \overline{a})^2}\sqrt{\sum_i (b_i - \overline{b})^2}} \tag{2.9}$$

2.5.3 Sum of Absolute Differences (SAD)

Similarly to the previously-described sum of squared differences in the *Harris & Stephens Corner Detector* [24] section, SAD calculates the difference between pixels. However, instead of squaring that difference, SAD takes the absolute one. This is represented as $\sum_i^n |a_i - b_i|$. In conclusion, SAD could be unreliable because it does not take into account changes in lighting, size, orientation or color. Despite that, due to its simplicity, it is an extremely fast measure.

Chapter 3

ResNet and Similarity Measure Approach for Image Verification

3.1 Objectives

This project, as previously mentioned, takes on the challenge of Image Verification. More specifically, verifying if a query image has been taken at a specific GPS location by comparing it to another reference image. There are a lot of methods and techniques used to solve that problem, as it was already covered in Chapter 2. However, this project tries to improve on already existing methods as measured by the following metrics:

- Accuracy
- Computational Time

The first point is the most important, however the hardest to improve. The methods used to solve the problem of Image Verification can be arranged in two high-level groups. The first one being methods that do not use CNNs to match features between two images. Such algorithms are BBS [8] and DDIS [9]. It is very important to note that these methods use CNNs to extract features but not to verify how similar both images are. The use of CNNs is preferred for the purpose of detecting and extracting points of interest as they are more robust than classical Feature Detectors, as stated in Chapter 2. As covered before, such methods like BBS [8] and DDIS [9] are a good solution in order to remove the long and exhaustive training step that CNNs require. However, they are not as robust or accurate.

The second group consists of methods that use CNNs to both extract features and make verification predictions. Such methods, as already stated in Chapter 1 and Chapter 2, require huge datasets and a lot of computational power. Despite that, their accuracy proves to be higher than methods from the first group.

The efficiency of such techniques can be improved by modifying their different modules. This includes, but it is not limited to, detecting and extracting more robust features or utilising different similarity measures. This particular project combines different methods in order to increase accuracy and decrease computational times and power. The approach taken is to use the already pre-trained ResNet50 [13] on the ImageNet dataset [30] for the extraction of features, as well as a similarity measure for verification. Moreover, in this way, there is no need for training a complex CNN in order to obtain verification results. This approach has been inspired by the BUPM method [10], where the authors implement the same pre-trained ResNet [13] to extract features. Therefore, the project can be put as belonging more to the first group explored. However, it also differs from it in the process of extracting features from images. Methods like BBS [8] and DDIS [9] use outdated and not as accurate CNNs, where this project implements a new approach of using a Residual Network [13]. Such a technique was inspired by the previously covered BUPM [10] method.

3.2 Technologies Used

For the implementation of this project, different technologies have been used. The code was written in python 3.6 with the help of various libraries to support the algorithm. Such libraries include, but are not limited to, *Keras 2.3.1*, *Tensorflow 2.0.0*, *OpenCV 3.4.2*, and *NumPy 1.16.0*. The use of Keras with backend engine Tensorflow provides the additional support of different AI methods and algorithms. In addition, it includes different pre-trained ResNets [13] which were used in this very project.

The algorithm was tested on multiple Residual Networks [13] as Feature Extraction engines, as it is described in future chapters. However, the ResNet50 [13] was kept as the default one, mainly because of its good accuracy and computational time. In the *Design & Implementation* section, a more in-depth description of why and how the ResNet [13] was used can be found.

The NumPy and OpenCV libraries were primarily used to manipulate and work with images as vectors. Besides, they were used for re-scaling images by using different interpolations, as well as performing and implementing the similarity measure for the Verification process of the project. Furthermore, different libraries were used in order to visualise, collect and save the data

produced by the project. As mentioned previously, this algorithm adopts techniques used by already existing methods. The Feature Extraction process is based on the *Bottom-Up Pattern Matching BUPM [10]* paper. However, the Verification process uses a similarity measure, which was used as an approach by various other methods. Such are BBS [8] and DDIS [9], as per Chapter 2. Therefore, although the project implements already existing methods, the algorithm itself is original.

The project was created and tested in both local and cloud IDEs. Google Colab [37] was mainly used for testing as it provides better computing power than the already available local machine. Moreover, in the following section, a detailed description of the design is shown in conjunction with the implementation steps taken for the project.

3.3 Design & Implementation

For the implementation of this project three main steps have been taken. Each step covers implementation and design milestones for the algorithm. By dividing it into different modules, the project can be easily modified, resulting in a more modular solution that can be adjusted for different types of problems. The project is designed in such a way that every parameter can be changed. Therefore, the user can manually choose the best options for their problem dataset. All parameters can be found in table 3.1. It must be noted that a more detailed description of these parameters as well as a user manual on how to use the model is provided in the appendices of the paper. The next subsection describes the implementation and design of the project step by step.

Parameter	Default Value	Description
Threshold	0.585	Used for the verification process. If the similarity of the pair of images is higher or equal to the threshold, then the query image has been taken at the location of the reference image. In addition, this parameter can be used to specify a custom threshold value.
Extract	False	If this parameter is toggled, then the model will begin extracting similarity values from a specified training dataset. The values will be saved to a file which can be later used for further testing or examination.
Extraction Dataset	...	This parameter is used to specify the location of the query and reference images, as well as the labels that will be used in extracting.
Extraction Path	...	It is used for specifying the location of a file that contains the results of extracting the similarity values from the specified dataset.
Plot	False	The "plot" parameter is used for plotting the data from a file. The file that will be plotted is specified by the "extraction_path" parameter.
Print Mask	False	By toggling this parameter to True, the model shows a likelihood mask for both images. This mask is shown only if the pair of images is found to be similar enough. This is a restriction made in order to reduce computational power and time.
Predict	False	By using this parameter, the user can provide the method a never-seen pair of images and get a result that includes the similarity value between both images.
Test	False	If the "test" parameter is invoked, then the user can specify a test dataset that contains the path to query images, reference images and labels. In that way, the model will test automatically pairs of images and then compare the results to the actual labels, giving an accuracy value at the end.
Match Method	Template Matching	This is a parameter that has two options. One is to use the Template Matching method implemented in the model, and the other is to use the Patch Matching technique.
SURF Comparison	False	If this parameter is toggled, then the model will use SURF [4] to verify the query, reference image pair. This parameter toggles the code used for the evaluation against the SURF [4] method.
Model	ResNet50 [13]	This parameter specifies the CNN that will be used for feature extraction. The user can pick from the following options - ResNet50, ResNet101 ResNet152 [13], VGG-19 [31], Inception-ResNet [34].
Similarity Measure	Correlation Coefficient [14]	This parameter specifies the similarity measure that will be used in the model. The choices are - Correlation Coefficient [14], Normalized Cross-Correlation [36].

Table 3.1: This table provides an overview of all parameters that are available to use in the model. Every parameter is customisable and they can be used in conjunction with other parameters.

3.3.1 Project Modules

Reading & Pre-processing Images

This is the very first module implemented in the project. It has the task of reading images from a specified location, as well as performing various pre-processing techniques in order to make them ready to be passed over to the other modules. The model accepts inputs of two images - query and reference. To perform Image Verification, a query and reference image must be provided to compare the features of both against each other. Nonetheless, the Reading step is merely a way to represent these images as vectors or arrays of numbers, so they can be further modified and manipulated. The images passed to the model are of RGB color mode. The dimensions of the images are kept as well. They, essentially, become vectors of *(height, width, channels)* shape. However, because the images are of RGB color mode, the *channels'* value will always be 3. Each channel represents the red, green and blue colors respectively. That is vital for the "Future Extraction" module that is covered later on, as it requires input images to have 3 channels. Also, the dimensions of every image are increased along the zeroth axis. That step is necessary because the input for the ResNet [13] in place must be of four dimensions, not three. After this modification, the shape of the image vector is as follows - *(1, height, width, channels)*.

The next part of this module is to pre-process the input images even further. This is done to enhance the images and remove any unwanted distortion. For this purpose, the Keras *preprocess_input* method is used. This function is necessary to be used before sending the images to the actual Residual Network [13]. It has two different modes - "tf" or "caffe". For this project, the latter is used. The mode converts an RGB image to a BGR one. As the name suggests, BGR is the same as RGB but with reversed channels. Furthermore, the "caffe" mode zero-centers each channel with respect to the ImageNet dataset [30]. This mode was chosen for that exact reason. If "tf" mode was used it would affect the performance of the Residual Network as it was not designed for networks trained on the ImageNet dataset [30]. However, the "tf" mode normalizes the pixels between -1 and 1. In addition, an important note to take is that usually ResNet50 [13] takes an image as of *(224, 224, 3)* shape. Although, for this project, a new approach is taken where the original images are not resized to a specific size. This is done because the ResNet [13] is not used as a classifier but as a feature detector. Therefore, the bigger an image is, the more features will be extracted. ResNet [13] was trained on images with shape *(224, 224, 3)* to do classification, but this project only requires local feature extraction to be performed. Therefore, there is no need for the size of the images to be restricted.

To summarise, this module reads an input of two RGB images into two vectors. Then it pre-processes the images by expanding their dimensions as well as performing the previously described ”caffe” mode. Additionally, a diagram of how this specific module works can be observed in figure 3.1. It must be also stated that the query and reference images undergo this module at different times. It is further explained why and how this is performed in the next section.

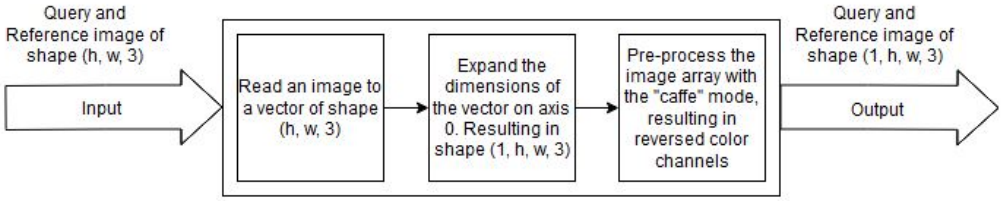


Figure 3.1: This diagram shows the steps taken in the ”Read & Pre-process” module. This step is required in order to accommodate the input to be passed to the ResNet [13].

Re-scaling & Feature Extraction

As covered in Chapter 2, there are many techniques and methods used to detect and extract features from an image. In this particular project, the Residual Network ResNet50 [13] is used to find interesting points in both the query and reference image. This model implements two different techniques in order to match the query with reference image:

1. Template Matching
2. Patch Matching

The first method is implemented in such a way that only the query image is re-scaled to different dimensions multiple times. By doing that, a more accurate similarity value can be obtained. As the size of the object being searched on the reference image is unknown, by re-scaling the query image the similarity measure algorithm can find better matching features. Essentially the size factor is removed to make the method more robust. This statement is backed by various tests in a later chapter. The scaling that is used is done by resizing both the height and width of the query image. This is the case because resizing only one dimension will result in distortion of the image. Because this process is repeated for every scale from a list, only the best size can be picked. The best scale size is decided depending on the produced similarity value by the similarity measure. It must be noted, that the query image skips the previously described module in terms of pre-processing and it is performed at a later stage. This is done because every different query scaling must be individually pre-processed. Because the

whole pre-processing process, as covered in the previous module, expands the dimensions of the images, the query image must be re-scaled before that. This is not absolutely necessary, and if done otherwise it will not affect the quality of the image or features. However, it simplifies the computational process significantly by removing the need of reducing the dimensions at every scale. Unlike the query, the reference image is not being resized for the reason that it must be up-scaled to have the object of interest match the query image size, rather than down-scaled. This approach is worse, as up-scaling reduces the quality of the image more than down-scaling. With down-scaling, the features become more condensed, where with up-scaling the points of interest will get further away from each other resulting in worse features.

The second, Patch Matching technique, implements a different approach to the previously described Template Matching method. It disassembles the reference image into patches, where each patch is of a fixed size. This is performed with the help of *"scikit-learn"* library. The idea is that at least one of these images will contain the query object. The number of taken patches is controlled by a parameter. Extracting more patches would be overall better, however, it will significantly increase the system resources required. Thus, a good balance must be found, which is done in Chapter 4. Furthermore, after obtaining the patches, the algorithm iterates over them, passing each patch with the query image to the Residual Network [13]. Essentially, the feature extraction part is the same as in the Template Matching technique. The only difference being that instead of having the query image re-scaled into different sizes, the reference image is decomposed into a list of patches. Each extracted patch becomes the reference image for the specific iteration. Additionally, in the beginning, the query image is resized to a smaller size to speed up the process. Because images may be of various sizes, in order to remove the risk of memory issues, resizing of the query image is performed. That smaller size is not fixed but dynamic. This means that the query image is resized a certain percentage of its original size. This approach is taken because the query image might have a smaller size than the potential fixed one. Therefore, the resizing process will be up-scaling rather than down-scaling the image, which is worse as stated before.

After either of the above-mentioned techniques are completed, the newly modified images are passed through the previously described ResNet50 [13] network. The output of which is two feature vectors with *(h/32, w/32, 2048)* shape. Furthermore, the *2048* channels dimension is produced by ResNet's [13] last Convolution layer applying 2048 filters. Additionally, the denominator 32 is because of the five times of factor 2 down-sampling that the Residual Network implements. The two vectors are then reshaped into two dimensions to make further

modifications easier. The new shape is produced by the product of the *height* with the *width* of the vector. By doing this, the algorithm will have to work with 2-dimensional vectors rather than 3. Therefore, making it easier to compute. Once these feature vectors are obtained, the next step is for them to be compared by the Verification module. Additionally, the implemented techniques are used separately, with the user choosing the best one for their data. If the reference image is of big dimensions, then the Patch Matching method might be of better use. This is the case because reference images during Template Matching are not re-scaled, resulting in a lot of unimportant features. That is especially if the reference image is a panorama. Additionally, these points of interest can match the ones in the query image, although, in reality, the two images might not be the same. However, if in that case, the Patch Matching method can compare only small patches to the query image, resulting in more uniformed features. If the features from the patch match the query image then it is more likely for both to be similar, rather than just compared against a panorama image. A diagram explaining the steps taken in this module can be seen in figure 3.2.

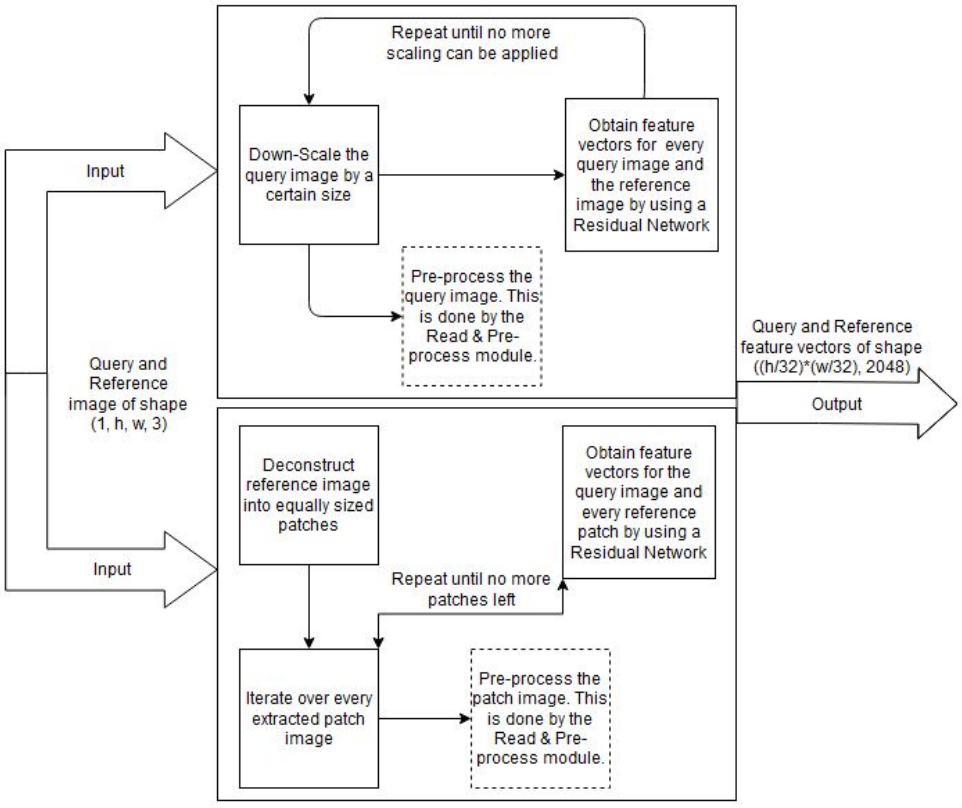


Figure 3.2: This diagram shows the steps taken in the "Re-scaling & Feature Extraction" module. Two methods can be used for the purpose of this module. The first one being the Template Matching technique where the query image is re-scaled. And the second Patch Matching method, which decomposes the reference image into patches. The produced vectors from these methods are later used for verification.

Verification

The following module deals with these several tasks:

- Getting similarity value between query and reference feature vectors.
- Verifying the similarity given different methods.

As described before, the previous module produces two vectors. Each vector corresponds to either the query or reference image that was provided initially as an input. The "Verification" module uses these two vectors in order to compare them and extract a similarity value. This is performed with the help of a similarity measure. The one used by the project is Correlation Coefficient [14], which was previously covered in Chapter 2. It is used to measure the strength of the relationship between both images.

First, the vectors are reduced by their means along with their *height* and *width* dimensions. In this case, it is the *0th* axis as the output of the previous module is of shape *((h/32)*(w/32), 2048)*. This is the case because the algorithm is interested in the actual image pixel values, that is, the rows and columns. Thus, the depth of the images is ignored and only the height and width are taken into consideration. Moreover, the dimensions are kept the same, which means that no reduction is performed on the image. This whole process of reducing by the means of both images is done to center the pixels. It is also referred to as Local Centering, where the distribution of the pixel values is centered on the value of zero.

The Correlation Coefficient [14] formula is performed as described in Chapter 2, with the two vectors as input. However, the query feature vector must be transposed before passed over to the formula. Transposing a matrix means that the column will be swapped with the row vectors. This measure is necessary because the result of the Dot product [35] has entries that are the inner product of a row of the reference matrix with a column of the query matrix.

Additionally, after transposing, the columns of the query matrix become essentially its rows. Therefore, the entry corresponds to the product of two rows, resulting in a product symmetric matrix. A symmetric matrix is such that it is equal to its transpose. After transposing the query feature vector, it will have a shape of *(2048, (h/32)*(w/32))*. Once the Correlation Coefficient [14] is performed, the result is a matrix where the maximum value in it represents the best matching point between both feature vectors. To extract that value as well as the location of that point, OpenCV's *minMaxLoc* method is used.

As mentioned before, the "Re-scaling & Feature Extraction" module either iterates over all scales or produces a list of reference patches. This module, being performed immediately after

the previous one, loops as well. Which results in many maximum values and locations for each iteration. However, only the best ones are kept.

A verification process needs to be performed to verify if both images are similar enough. If positive, it is safe to say that the query image is present in the reference one. Using a threshold proves to be an accurate method for that task, as seen in a later chapter. The obtained best maximum value is compared against a pre-specified threshold. If the value is bigger or equal to the threshold, then both images will be classified as being similar enough. By having a general threshold that classifies images as either similar or not similar, the proposed method can produce valid results within a small margin error. This is the case because negative and positive samples should produce similar results respectively most of the time. However, it could be the case that samples get misclassified because of the size or quality of the images. Therefore, using a threshold is not a perfect method and further ways on how to improve it are suggested in future chapters.

To summarise, this module extracts and compares the similarity value of two feature vectors against a threshold. It can be either used as the output of the model, or it could be used as an input to the "Outputs Visualisation" module, which is described in the next section. Additionally, a diagram of how the "Verification" module works can be observed in figure 3.3.

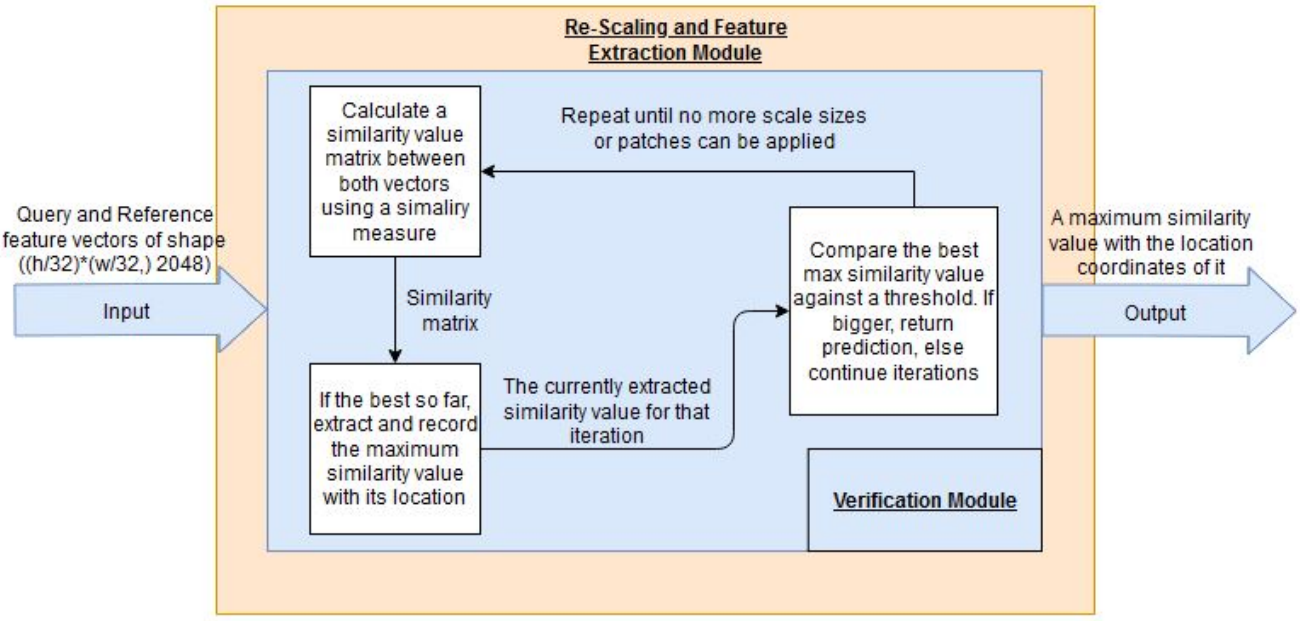


Figure 3.3: The diagram shows the different steps taken in the "Verification" module. It includes performing a similarity measure to extract a similarity value between two vectors. As well as, comparing the best maximum similarity value against a threshold. In that way, if the value is bigger than the threshold, a prediction can be returned resulting in further iterations being omitted.

Outputs Visualisation

The previously described three modules are necessary and performed on every execution of the algorithm. However, this module is optional. It is executed after the "Verification" module. Its inputs are the produced Correlation Coefficient [14] matrix as well as the sizes of the reference image and the best-scaled query image. The sizes after extracting the features are taken, rather than the original image sizes. That is, as covered before, because ResNet50 [13] reduced the dimensions of the image by a factor of 32. Keeping track of the sizes is necessary because the algorithm needs to know how big both images are in order to extract them from the matrix. The process of extracting likelihood maps is performed by taking the maximum values across both axes of the Correlation Coefficient [14] matrix. The matrix is of two dimensions, where the *0th* axis corresponds to the query image and axis *1* to the reference image. The result of extracting the maximum best-matched features has only one dimension for every image. It is represented as of *((h/32)*(w/32))* shape. Therefore, a vector reshaping is performed with the use of the image sizes passed as input. The output of that reshaping process is a two-dimensional array of shape *((h/32), (w/32))*.

If the above-mentioned Template Matching method is used, the newly extracted best-matching feature maps need to be modified to show the best-matching location point. Because the model matches the query image to the reference one, it means that the reference image will hold that best location point that represents the object in the query image. Thus, the OpenCV *minMaxLoc* function is used again to extract the best maximum location from the reference image. Furthermore, because it is expected for that point to be the object from the query image, scaling is performed. The query image's height and width are used in conjunction with a specific scale to represent somewhat accurately the size of the object. Because the best maximum location is represented by *(x, y)* coordinates, each point has the query width or height deducted respectively. In addition, the height and width values are weighed by the scale. This is performed twice because, to draw the object on the image, it requires a starting and an ending point. However, for the ending point, instead of deducting, the height and width are added. After the object is drawn, the original images with their respective maps are shown. The query map shows the best matching features, where the reference map presents only where the best maximum matching location is, also referred to as a likelihood map. Additionally, the formula used to draw the best matching location can be seen in equation (3.1).

$$\text{Start Point:}((x-(w/scale));(y-(h/scale)))$$
$$\text{End Point:}((x+(w/scale));(y+(h/scale))) \tag{3.1}$$

The *(x, y)* coordinates are the points of the best maximum location, and the scale is an optional value bigger or equal to 1. This is the case because it is used as a denominator to weigh the size of each point, as it is shown in figure 3.1. However, if the Patch Matching method is used instead, the likelihood map is simply replaced with the best-matched patch, which is then printed with the query likelihood map.

In conclusion, the "Outputs Visualisation" module uses the Correlation Coefficient [14] matrix to extract and present a best matching feature map and a likelihood map/patch, depending on the matching method. The query map shows the best matching features between the query and reference image, and the other shows where the location of the object from the query has been found to be. Moreover, the whole module process is presented in figure 3.4. Figure 3.5 shows the results of performing this module on an example image.

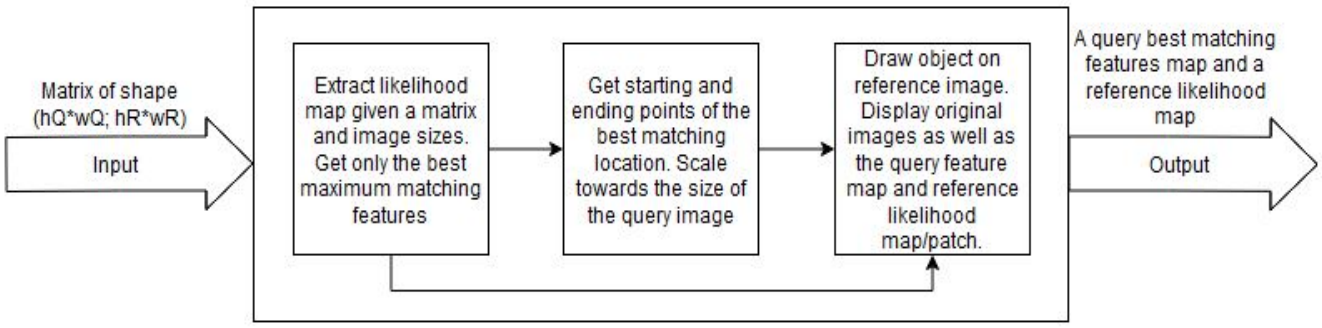
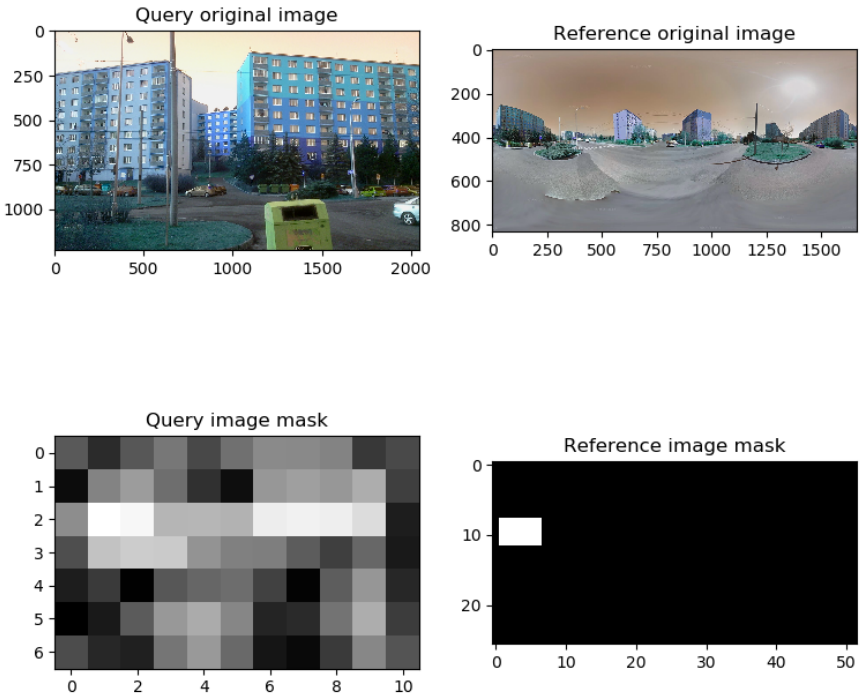
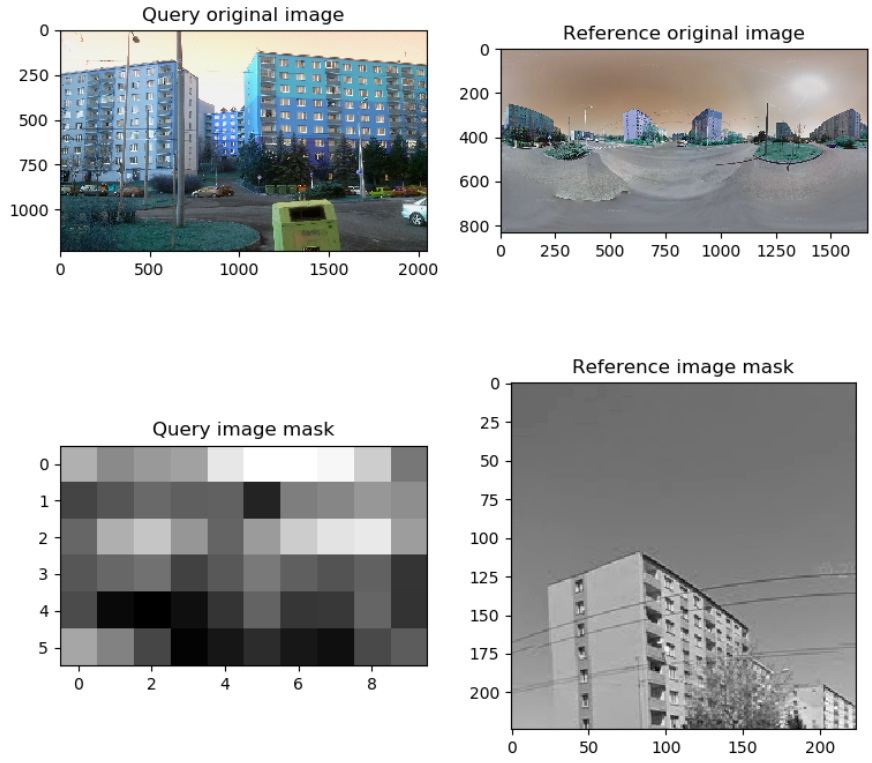


Figure 3.4: This diagram shows the different steps taken in the "Outputs Visualisation" module. It shows the process of extracting a reference likelihood map/image patch and a query map of best-matched features. Furthermore, the diagram shows the use of the previously mentioned scaling function to represent the location of the query image object in the reference image. It must be noted that the input of the module represents the sizes of both query and reference images. That is, not their original sizes but the sizes after the features have been extracted. *(hQ, WQ)* and *(hR, wR)* are equal to the original query and reference image sizes divided by 32, as covered previously.



(a)



(b)

Figure 3.5: Figure a) shows the extracted likelihood maps after performing Template Matching on a pair of query, reference images. Figure b) presents the best-matched patch as the result of performing Patch Matching on the same image as in figure a). The query image masks represent the best matching features between the two images. The whiter the pixel, the better the feature.

3.4 Model Flexibility

This section further explores some of the previously covered model parameters. The project implements helper functions that provide flexibility in terms of testing and visualising data. If the "extract" parameters are invoked, then the model will produce a file that contains a similarity value with a respective label attached to it. This can be used to visualise how the model performs on both positive and negative samples. Furthermore, this newly produced file can be used in conjunction with the plot parameter, to put the data on a graph for better visualisation. Also, these parameters can be used in future project modifications. For instance, they can be used to train a Machine Learning algorithm to adjust the threshold dynamically. Moreover, by using the "extract" parameters in conjunction with the plot parameter, a graph like in figure 3.6 will be produced. Further improvements are suggested in a future chapter.

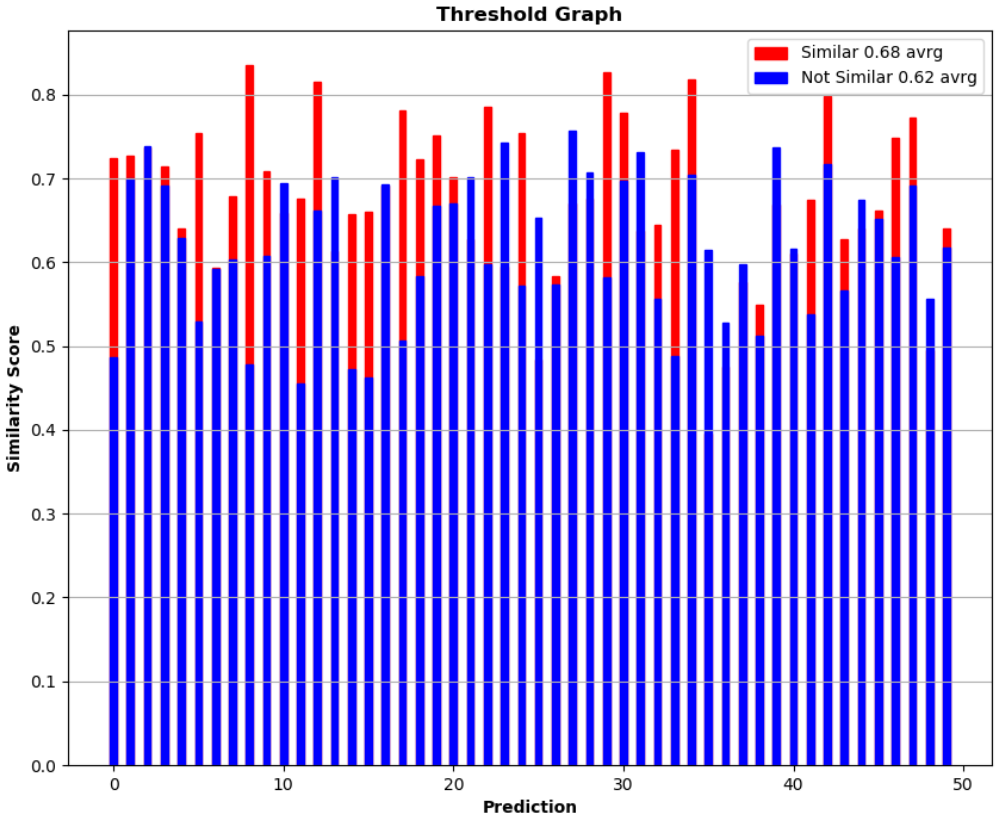


Figure 3.6: This figure shows a graph representing the results of extracting similarity values on the Wiki_Commons [10] Dataset. They are divided into positive and negative samples, depending on the label for each similarity value. This data can be used for further examination and improvement of the model.

Chapter 4

Results/Evaluation

This chapter provides an extensive evaluation report, including test results against other Computer Vision methods used to solve the task of Image Verification. Moreover, it shows the advantages and disadvantages of using this newly proposed method, as well as how it performs under different implementation variations and data. This also includes evaluation between different ResNets [13]. Normalized Cross-Correlation [36] has been tested against the default Correlation Coefficient [14] measure, as well. These results are based not only on the accuracy of the model but also on the computational time needed to perform Image Verification.

The following sections cover an in-depth evaluation of different implementations of the model in use. It is done by picking the best similarity value for every query and reference image pair. ResNet50 [13] was used to extract features, and Correlation Coefficient [14] was used as the similarity measure. This configuration was inspired by the results and evaluation of the BUPM [10] method. Moreover, as stated in Chapter 3, the model uses different scaling provided the method being used, either Template Matching or Pattern Matching. The scale sizes used vary between test runs, as different implementations are tested. However, the scale units are the same throughout. They represent the percentage by which the image will have their dimensions reduced. For instance, if a scaling range is [10, 15, 20], it would mean that the image will be down-scaled by 10, 15, and 20 percent. For Pattern Matching, no scaling ranges have been used. Therefore, this does not apply to that method. However, the query image is still down-scaled by a fixed percentage as well. This is further covered in the sections below.

Moreover, the patch size used is of (224, 224). Such shape has been chosen because the default input size recommended for the ResNet50 [13] is of these dimensions. The ResNet50 [13] was initially trained on images of that shape. Nevertheless, that does not necessarily mean

that the dimensions used will produce the best results. There will be fluctuation depending on the data and on the other parameters of the model.

Additionally, the unit of measure by which every test run is compared with other methods or with other test runs of this model is accuracy. The accuracy is represented by how many accurate predictions a model has made out of the total number of predictions possible. For instance, if the model predicts 5 samples correctly out of 10 samples, then the accuracy would be 50% or 0.5 as shown in the test figures below. The accuracy is connected to the threshold, which was used as a verification method. Finding the best threshold, that splits the data perfectly, will also means finding the best accuracy as they are correlated. The best threshold for a test run is represented with a green line on the graph. Also, two datasets have been used for testing:

- Wiki_Commons - A dataset provided by the authors of the BUPM [10] method. It contains 500 positive and 500 negative samples. However, because some of the images were missing and could not be downloaded, this project tests only 331 pairs. That is, both positive and negative pairs, resulting in 662 total samples. Nevertheless, every pair of query and reference images is of the same format, although the size may differ. The reference images are panoramas where the query images are of normal view type.

- Caltech Dataset - This is a dataset taken for 50 buildings around the Caltech campus [38]. For every building, there are 5 images taken in different angles and distances. Because there are no negative samples provided, for the purpose of testing extensively this project, negative samples have been automatically generated by taking the last image from a 5 image cluster and pair it with an image from another cluster.

Google Colab [37] was used to produce the testing results. The hardware specifications are as follows:

- System Memory - 12.72 GB

- GPU - *Google Colab* [37] changes between different types of GPUs to provide their services for free. Thus, the GPUs that were used for testing this project are Nvidia K80, T4, P4, and P100 [39]. Nevertheless, their performance difference should be negligible as they are all specifically designed for AI testing and training.

4.1 Threshold Evaluation

As covered in Chapter 3, this project implements a threshold validation method. However, choosing the right threshold can be a difficult task. This project implements a python script that takes similarity values from a file and iterates over them, trying different thresholds. The following results have been produced by using that script, which can be found in the appendices of this paper. Also, to produce the file with similarity values, an initial run of the test data is required. First, the algorithm is run on the test data using either the template or patch matching technique and ResNet50 [13] as the feature extraction method. After that, the script is run to find the best threshold and accuracy for the configuration in place. The following results evaluate the model by using a threshold against other solutions, like BUPM [10]. Additionally, they give an overview of how the threshold verification method performs on different types of datasets. That is, finding the maximum accuracy possible given a threshold to distinguish between positive and negative samples.

4.1.1 Template Matching on the Wiki_Commons Dataset

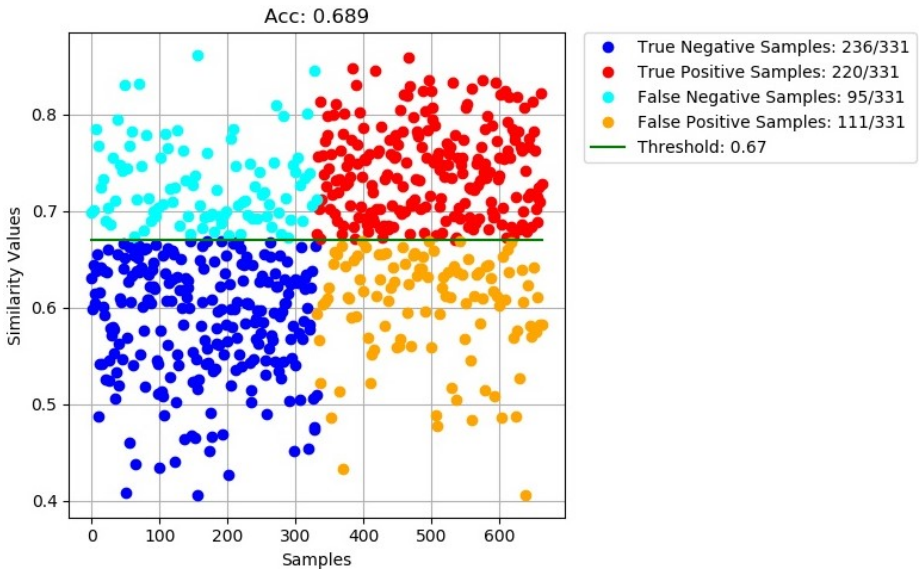


Figure 4.1: This figure shows all the samples obtained by using two different ranges of scaling sizes. The query image has been re-scaled by either [10,12,...,20] percent if it is bigger dimension-wise than the reference image. If the query image is smaller, then the range of scaling sizes is [22, 24,...,30]. This approach has been taken because there is a possibility that the query image is bigger than the given reference one. Thus, it needs to be converted to a reasonable size to represent a template for the reference image. The objects on the reference image will be, most likely, even of smaller size. However, if the query image is already smaller than the reference one, then there is no need to down-scale more than the necessary. The graph shows that there are a lot of low positive and high negative samples, which results in decreased accuracy, although the threshold green line is almost centered.

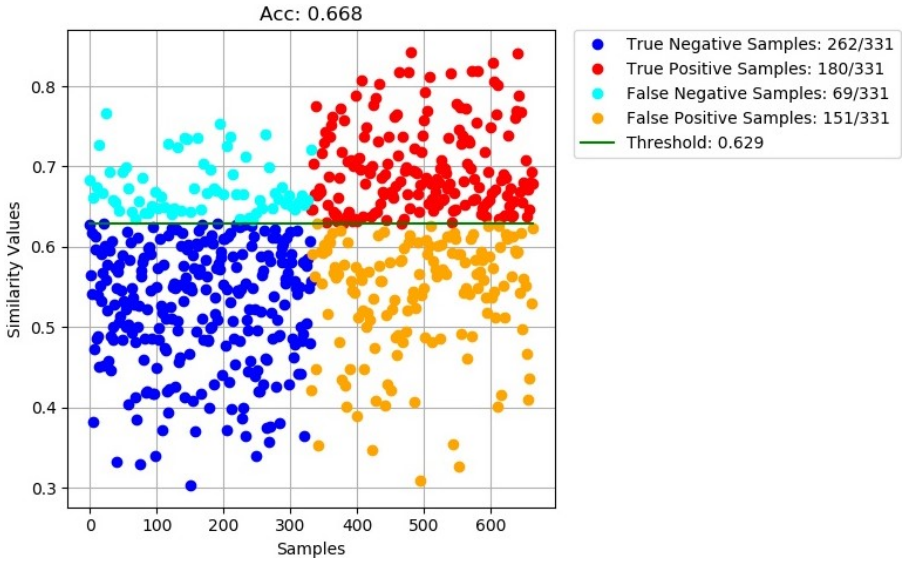


Figure 4.2: For this test run, the query image has been resized by the percentage sizes from the range [5,6,...,10]. This approach has been taken to rule out and prove further the results in figure 4.1. By re-scaling the query image by only small percentages, as it can be observed, does not result in high accuracy. That is because when the query image is of small size already, the amount of features extracted is not enough to obtain a big similarity value when matched against the reference image. Thus, the value cannot go over the assigned threshold. This scaling range would be better if all query images are of very big sizes.

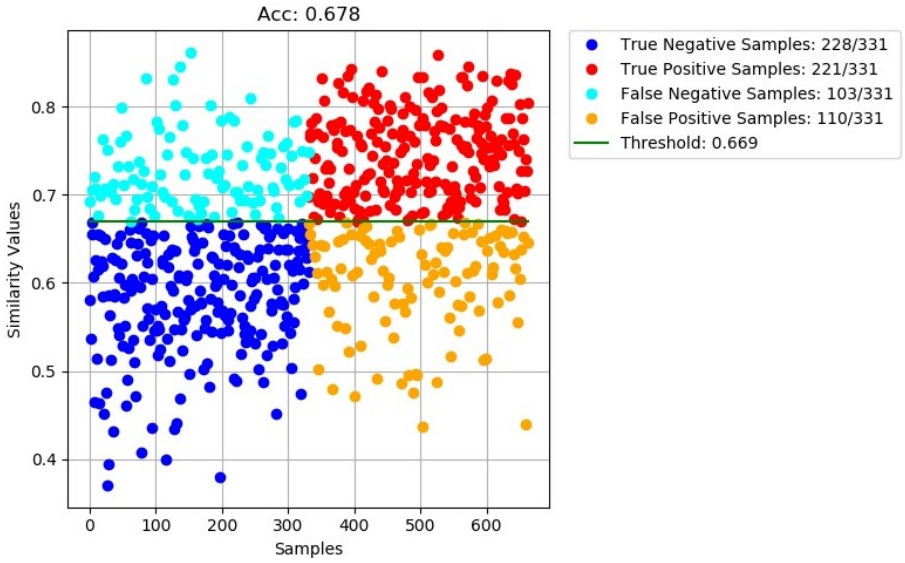


Figure 4.3: The graph presents the results obtained with a query image down-scaled by the sizes from the [10,11,...,20] range. The scales are different by only one percent, or one value. This results in a lot of fluctuation between the obtained values for both positive and negative samples. Because there are so many different sizes used, the similarity values differ a lot between each other. This figure shows the results by using the opposite of the range from figure 4.2. Although it results in higher accuracy, the used range only accounts for query images of small dimensions. As the image would be down-scaled to a bigger size, if it already has big dimensions, the query image would not be able to effectively match the small object on the reference image. That is, it would match either different object that has more features or it would result in lower similarity value because of the difference in extracted features.

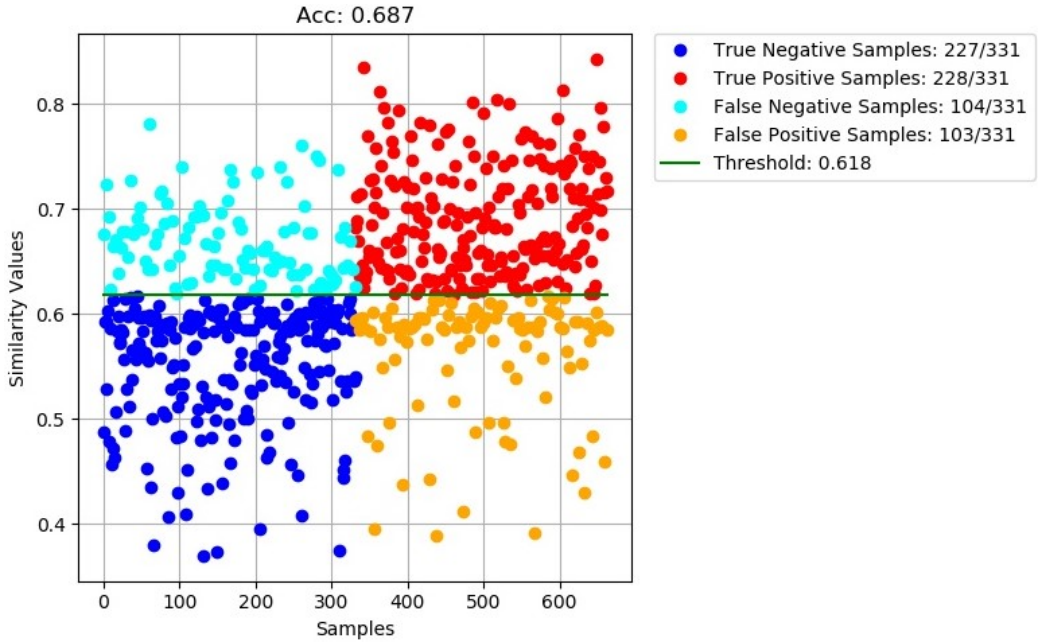


Figure 4.4: For this test run the query image has been resized with scales from the range [13,14,...,17]. As stated before, these scales represent the percentage by which the query image has been re-scaled. These specific sizes have been chosen following the results in figure 4.3. They represent the middle of the range from the previous graph. In that way, it has been accounted for either the very big or the very small query images in the dataset. Thus, resulting in accuracy similar to figure 4.1 that utilises two different ranges. Nevertheless, there are still some positive samples with very low similarity values and negative samples with very high values. However, actual predictions are more than twice as many. The threshold line is also almost in the center of the data, meaning that this range of sizes divides the data effectively.

In conclusion, the proposed method performs worse on the Wiki_Commons dataset [10] than the BUPM [10] one, which recorded an average accuracy of 0.875. However, the authors had to train two Neural Networks on huge image datasets as per [10], taking large amounts of resources and time. The differences in accuracy could be due to the verification technique used in the BUPM [10] method. It takes advantage of a Dense Neural Network, which is a network that utilises Dense Layers. Meaning, the neurons of a normal layer are fully connected to the neurons in the next layer. And as covered before, networks tend to perform better than classical methods like threshold verification. In addition, it must be noted that the dataset was specifically picked for the BUPM [10] method, where the query images are of high quality and big dimensions. For the proposed method in this project, such data could be deceiving for the model. A big query image results in a lot of features, which is good when matching against single object images, but when matching against panoramas some features might be matched as positive where in reality they are not. For instance, buildings can have similar shapes and architecture, so the query image might match a dissimilar object in the reference image.

Additionally, on the graphs shown above, it can be seen that the green threshold line tends to stay around the middle of the similarity value axis. This could mean that the proposed model tends to distinguish well between positive and negative samples. However, a linear threshold cannot account for bad predictions. That is, either very low positive or very high negative samples. Nevertheless, the verification method plays only a partial role when it comes to the accuracy of the model, as it can be seen in the next subsection.

4.1.2 Template Matching on the Caltech Buildings Dataset [38]

The following figures show the accuracy results on the Caltech Buildings dataset [38]. Unlike the Wiki_Commons [10] dataset, this one compares two single-object images, meaning the reference image is not of panoramic type. Because the type of the data is different, there should be a discrepancy between the results of using the Template Matching method when compared to the results from the previous subsection. The reference image has only one building object on it, meaning that the Patch Matching method should perform worse compared to the Template one. This would be the case because dividing a big object into several small ones would result in losing the overall information about that object.

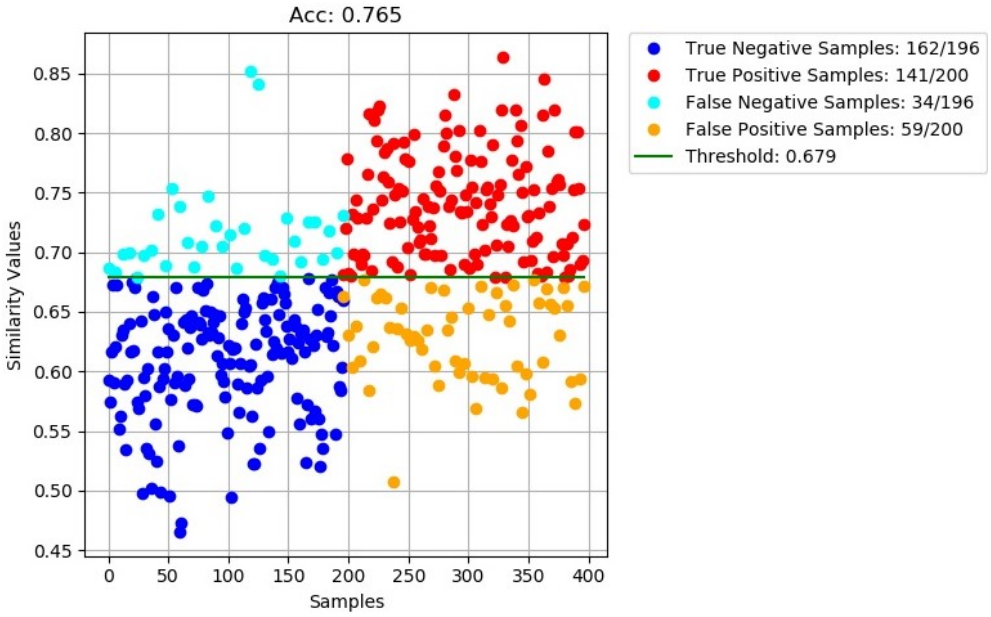


Figure 4.5: The following figure utilises the two ranges used in figure 4.1. It could be observed that these two ranges result in the highest accuracy on both datasets. This is because they account for various different query image sizes. The higher results for this dataset, compared to the Wiki_Commons one [10], are also mainly due to the type of method used. This statement will be further backed when the other Patch Matching method is being tested. Because the type of data is different on this dataset, more generalised sample values can be seen. There are almost no high negative samples and only a few low positive samples. Additionally, the threshold line is perfectly centered, meaning the samples are well separated.

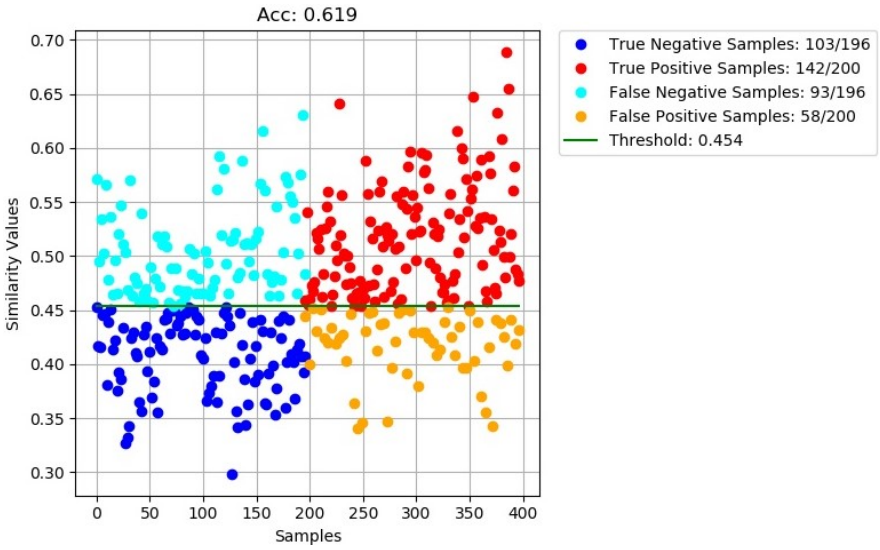


Figure 4.6: This figure shows the accuracy results on the scaling range [5,6,...,10], which was also used in figure 4.2. Because of the small sizes, this dataset does not scale as good as on the previous dataset. The lack of panoramas increases the need for more features. Therefore, small ranges of scaling sizes are not preferable. Thus, it could be concluded that bigger ranges are better when comparing non-panoramic images. There is also a lot of fluctuation between the similarity values of all samples. Therefore, the threshold cannot separate the data linearly. Which results in extremely bad accuracy compared to other runs.

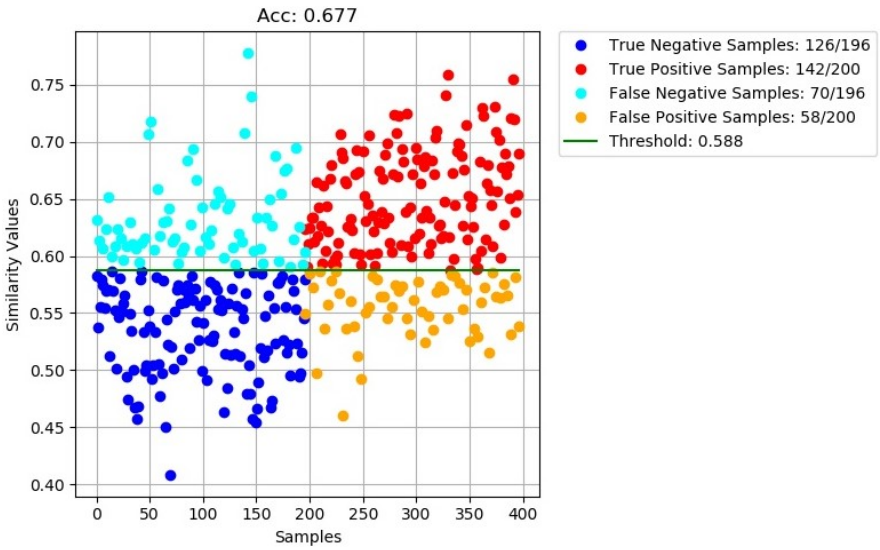


Figure 4.7: This figure shows the accuracy results on the scaling range used also in figure 4.3. It could be seen that the results in figure 4.3 are better for the Wiki_Commons dataset [10]. This is the case because the ranges are not big enough for this particular dataset. For instance, figure 4.5 shows much better results, although the Wiki_Commons [10] ones are very similar. The difference is because of the relatively low scaling ranges. Figure 4.5 uses 10% bigger scaling sizes than the ones used in this figure, resulting in more features. Also, it could be seen that there are a lot of high negative and low positive samples, which means that the method obtains bigger or lower similarity values for both types of samples respectively. That is due to the different query sizes present in the dataset. Because the scaling sizes do not account for either very big or very small image dimensions, this results in highly fluctuating predictions. However, the threshold still manages to separate the data somewhat effectively.

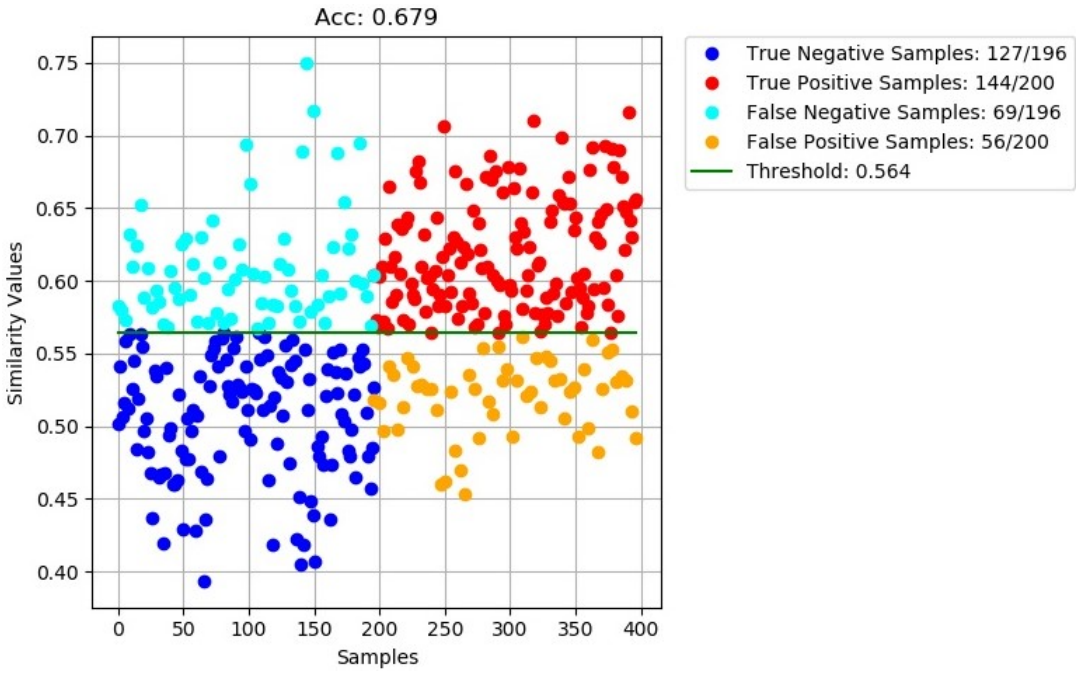


Figure 4.8: This graph shows the accuracy results on the scaling range used also in figure 4.4. It could be observed that the accuracy of this run is essentially the same as the one from figure 4.7. Also, the small difference is that this run has less false positive samples compared to other runs on the same dataset. By removing the low percentage sizes from the range, the method can generalise the data better, meaning there will be less low positive and high negative samples. The threshold is centered and separates the data somewhat effectively. However, it could be seen that there are a lot of negative samples that barely go over the assigned threshold, rendering the accuracy even lower. Therefore, a linear verification method, like a threshold, would perform worse than a non-linear one.

To conclude, bigger scaling is preferred when matching images that are not panoramas. When having panoramas, features of different objects might match the query template features, resulting in a false positive pair match. This proves the initial observations made. However, when matching two images that have, for instance, only one building, it is much easier to verify that both images are the same if the model has more features to work with. Besides, the following graphs in the next subsections present the results on testing with the other proposed Patch Matching method, as well as a comparison between the current Template Matching technique. Moreover, as observed in the results for the Wiki_Commons dataset [10], the threshold line tends to stay more towards the center of the similarity values. Which, to a degree, removes the verification method as the reason behind not having better results. Therefore, big fluctuation of both low positive and high negative samples can be observed. This suggests a lack of robustness of the Template Matching method for specific datasets. Nevertheless, the highest 76.5% accuracy recorded proves that this method performs better on non-panoramic reference images.

4.1.3 Patch Matching on the Wiki_Commons Dataset

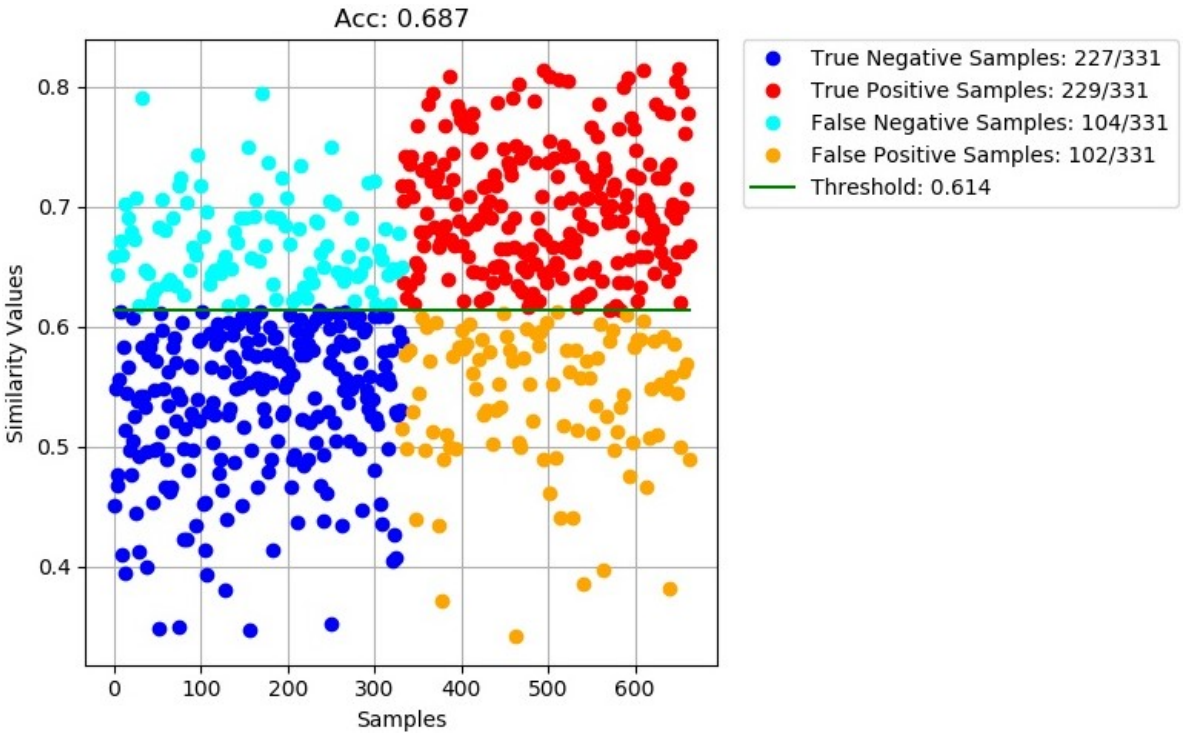


Figure 4.9: This graph presents the accuracy results by resizing the query image to a default pixel size, which is 10% of its original size. The test run uses 150 patches with size (224, 224). Compared to the Template Matching method, this one manages to produce a lot more high positive and low negative sample predictions. This configuration is not optimal, however, it still almost matches the best accuracy of the Template Matching test run on that dataset. Nevertheless, there is still a significant amount of low positive and high negative samples that produce noise, which the threshold line cannot overcome. This is due to the low fixed query size and the low number of patches extracted. The configuration decreases the accuracy but increases the efficiency of the run, which results in less memory and computation time needed.

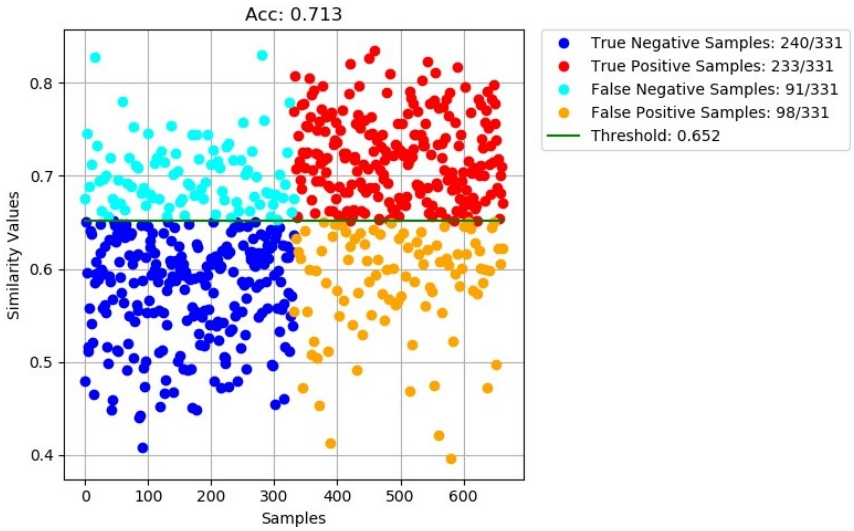


Figure 4.10: This figure presents the results of using the Patch Matching method with the query image resized 15% of its original size. The test run uses 250 patches with size (224, 224). The accuracy value is the highest recorded for the model in general, showing better distinctions between positive and negative samples. This is due to the high volume of extracted patches as well as the somewhat balanced image size. Although there are still low positive and high negative samples, the rest of the predictions are balanced enough for the threshold line to separate them effectively. In other words, there are not many slightly low or high samples that go either over or below the threshold line. If the sample is wrongly predicted, then it has either a very high value, if negative, or very low value, if positive. That makes it easier for the threshold to define a clear separation line between samples.

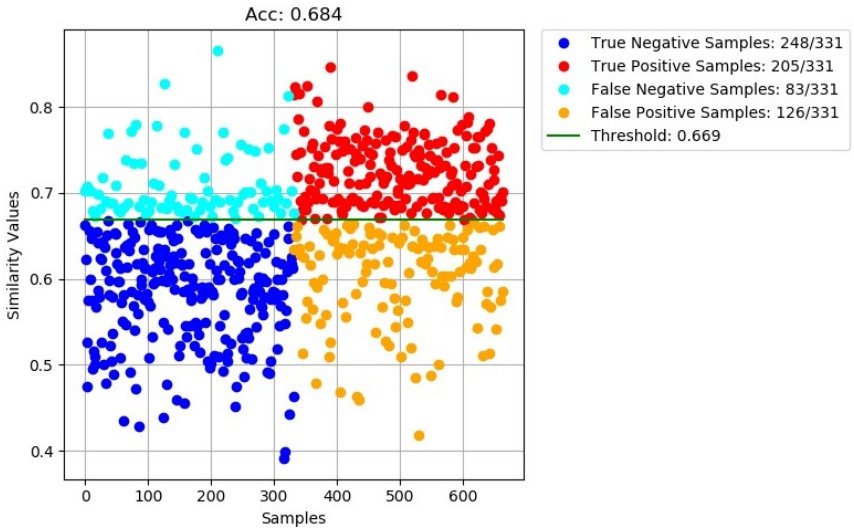


Figure 4.11: This figure shows the results after resizing the query image to 20% of its size. The test run uses 200 patches with size (224, 224). This configuration was picked to prove that the run in figure 4.10 is truly the best one for the model. This test run uses a higher query image size but with less patches. It shows that the query image size is as equally important parameter as increasing the number of patches. However, it could also be observed that the test run in figure 4.9 performed also better, although it extracts less patches. This could be because of the significant difference in the query size or because of the random factor in extracting patches, as they are chosen randomly by the function.

The proposed Patch Matching method outperforms the Template Matching method on various scaling sizes. The Template Matching method, as stated before, results in more high negative and low positive samples when comparing against a panorama. Where matching small patches results in better accuracy. It must be noted that having more patches would usually increase the accuracy, however, it would also increase the memory required. Besides, there will be a higher risk of matching a patch that does not contain the query image object. Nevertheless, that risk is accounted for by the actual way the method is implemented. Because the patch containing the query object will have the biggest similarity value, it will be picked by the method. Additionally, the graphs, presented above, show that there are more low positive samples rather than high negative samples. This could be due to the randomness when extracting patches. If the number of patches is not big enough, then the method could result in matching against patches that do not contain the query object. There is a visible difference between the low positive samples in figure 4.9 and in figure 4.11. The former implements the method with only 150 patches where the latter increases that number to 250. It must be noted that the threshold line tends to be between 0.6 and 0.7 on the similarity value axis. Thus, it can be concluded that the method differentiates well between positive and negative samples but cannot account for the noise, or fluctuation between sample similarity values.

4.1.4 Patch Matching on the Caltech Buildings Dataset

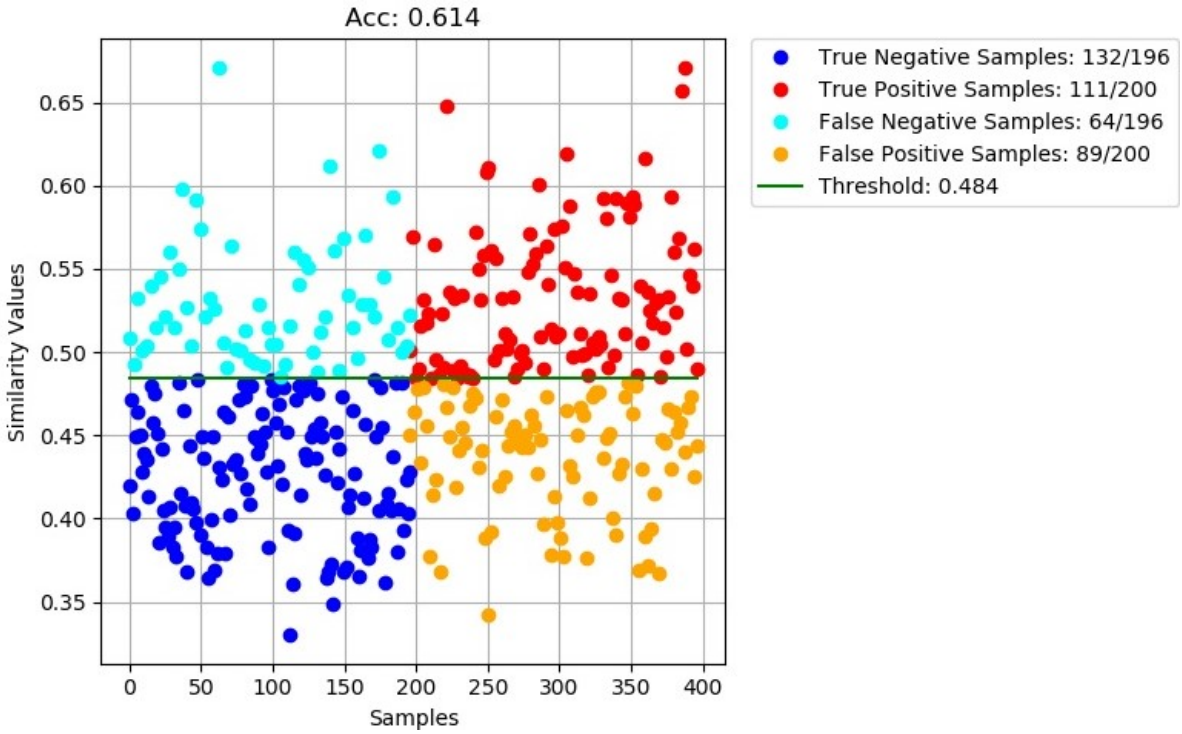


Figure 4.12: The following test run uses 150 patches with size (224, 224), the same as in figure 4.9, with the query image down-scaled to 10% of its size. It can be seen that the accuracy is not as good as the one produced by the Template Matching method. Reducing the size of the reference image results in fewer features. Also, the patch would be smaller than the actual object, as the reference image is not panoramic. The samples are of different values. Especially, it can be seen that there are a lot of very low positive and very high negative samples. Moreover, many wrong predictions are slightly over or below the threshold line. Thus, the threshold cannot split the data effectively. The reasons behind this are the same for all graphs in this subsection and are described extensively in the conclusion.

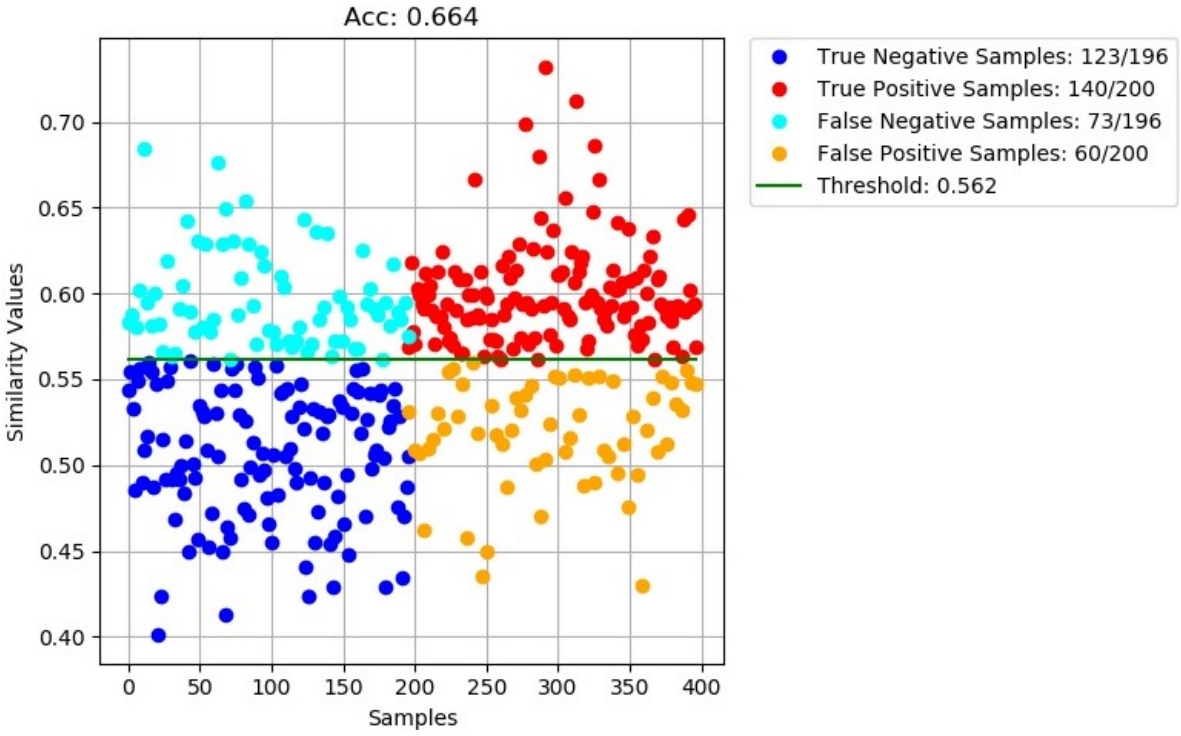


Figure 4.13: The following graph shows the results after having the query image resized to 20% of its size. Additionally, 200 patches have been extracted from the reference image with size (224, 224). This is the same configuration used in figure 4.11. The pattern described in figure 4.12 continues, where the accuracy is not as good as with the Template Matching method. However, it is better than the 4.12 because the query image has more information, as the size is bigger. The fluctuation in similarity values can be explained by the ineffectiveness of small extracted patches to present valuable information about the object on the image. That is why the Patch Matching method is preferred primarily when comparing against panoramas.

In conclusion, the Patch Matching method proves to be better than the Template Matching technique when it comes to verifying pairs of normal and panoramic image type. Because panoramas contain a lot of information about other objects that are not similar to the query one, it is better to disregard these features. Therefore, when matching the query image with different reference patches it results in better accuracy. Because the patches are represented as small-sized squares, there is a high chance that the query object will align exactly in the patch itself. This would result in having the query image match exactly against its object on the reference image. This probability of having the object align within the borders of the patch can be increased by either using bigger patch sizes or extract more patches. However, as it was observed on the results on the Caltech Buildings dataset [38], when there are no panoramas present, it is better to use the Template Matching method. Reducing the reference image into patches results in worse accuracy because these patches would contain only a small amount of information, unlike when performed on panoramas. Additionally, the threshold line

tends to stick to the middle of the data, which again proves that both methods manage to distinguish well between samples. It means that there are not many high negative or low positive predictions. Nonetheless, the previously described BUPM [10] method performs better on the Wiki_Commons [10] dataset. However, there is a trade-off between being accurate and computationally expensive. The BUPM [10] method requires a lot of computation power, resources and time for training, where the proposed model does not. Nevertheless, this project implements two different methods in a flexible way, where the user can pick the best technique for their data.

4.2 Feature Extraction Methods Evaluation

This section shows the evaluation of using different ResNets and CNNs as Feature Extraction methods. The most famous networks are compared on the two previously described datasets. The networks used for testing are ResNet50/101/152 [13], VGG-19 [31], Inception-ResNet [34]. It must be noted that if a network is more complex it does not necessarily mean that it would perform better when extracting features. Because the above-mentioned networks were initially created for classification, their depth would have an impact on the classification results rather than on the quality of the features extracted, as they all use similar filters. Additionally, further passing the data through more convolutional layers might result in extracting more features. However, there is a limit, where after that the extracted features might disrupt the matching process rather than help it. This might lead to errors, where a lot of false positives or false negatives are produced, which is reflected in the results produced below.

4.2.1 ResNets

ResNet101

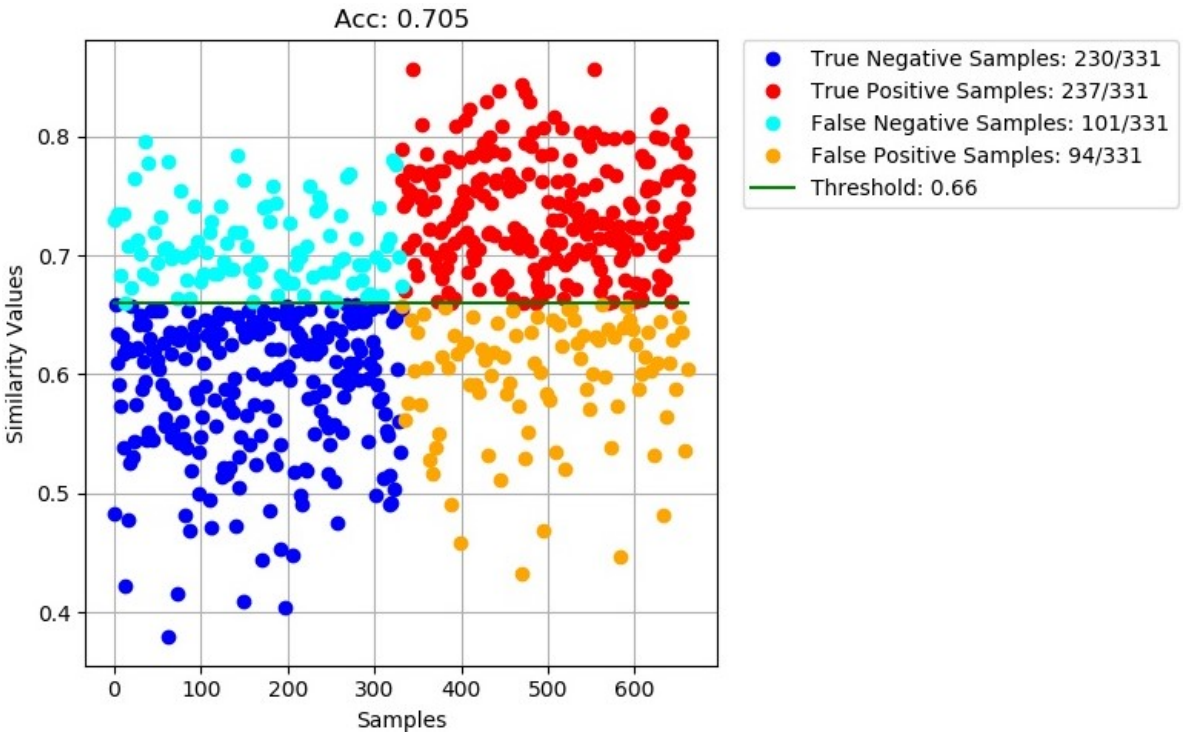


Figure 4.14: This figure shows the performance on the Wiki_Commons [10] dataset by using the Patch Matching method. The configuration is the same as in figure 4.10, where the query image is resized 15% of its size with 250 patches extracted. It could be noticed that the accuracy is worse than on the ResNet50 [13]. That is due to the higher complexity of ResNet101 [13]. Extracting more features does not necessarily result in better accuracy. Quality of the features is as important, thus, extracting features that do not benefit the comparison between both images reduces the accuracy, as the similarity value would not increase. This can be observed in the graph as well, where there are not many very low or very high samples. However, a lot of the samples are either slightly above or below the threshold line. Therefore, this makes it difficult for the threshold to clearly define a separation line between the samples.

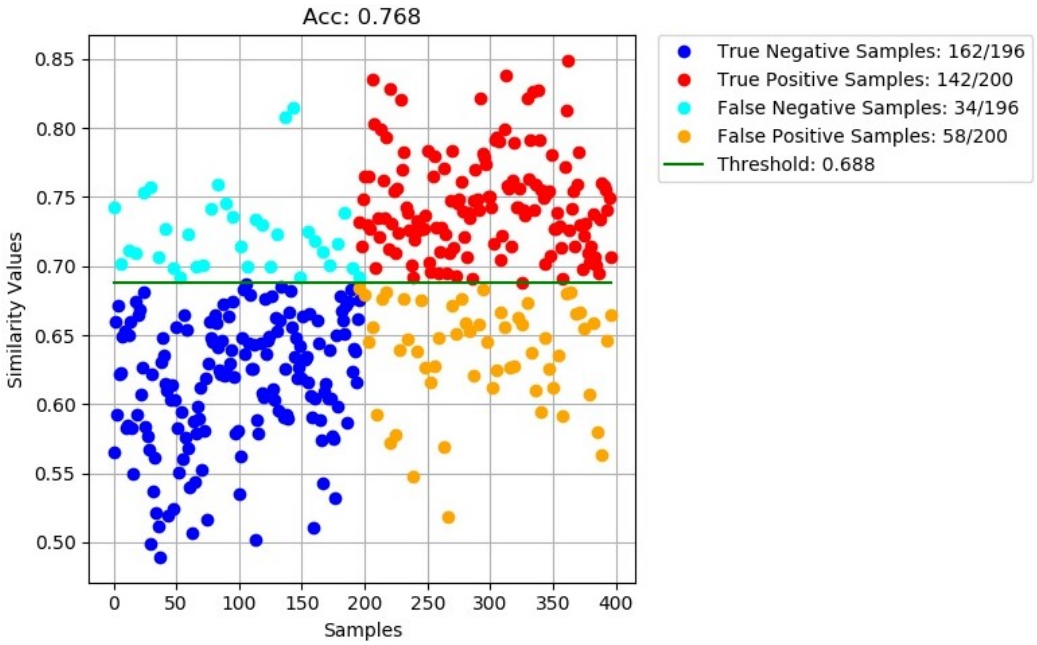


Figure 4.15: The following graph shows the results of using the Template Matching method on the Caltech Buildings dataset [38]. The query image is re-scaled with sizes from two ranges, similarly to figure 4.5. If the query image dimensions are bigger than the reference image ones, then [10,12,...,20] range is used, and if the opposite, the [22,24,...,30] range is implemented. Moreover, for this test run, the accuracy is better than on the ResNet50 [13] test run. As stated before, extracting more features greatly benefits when comparing normal images and not panoramas. Because the reference image contains "only" one object, e.i. a building, it means that the chance of wrongly segmenting it is decreased with more features being extracted.

ResNet152

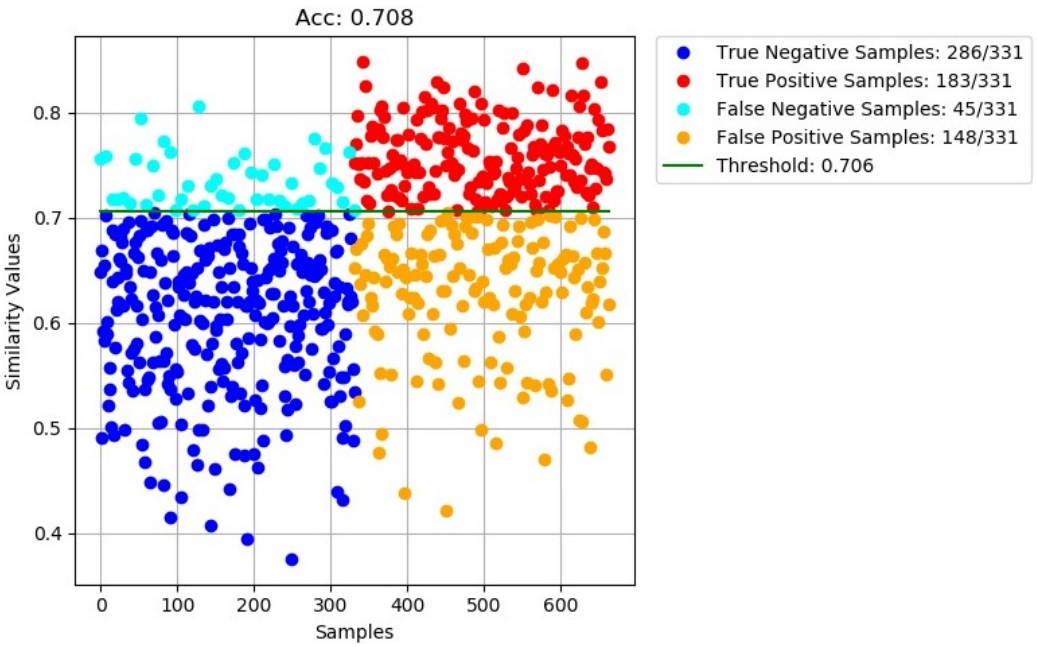


Figure 4.16: The results of performing Patch Matching on the Wiki_Commons [10] dataset.

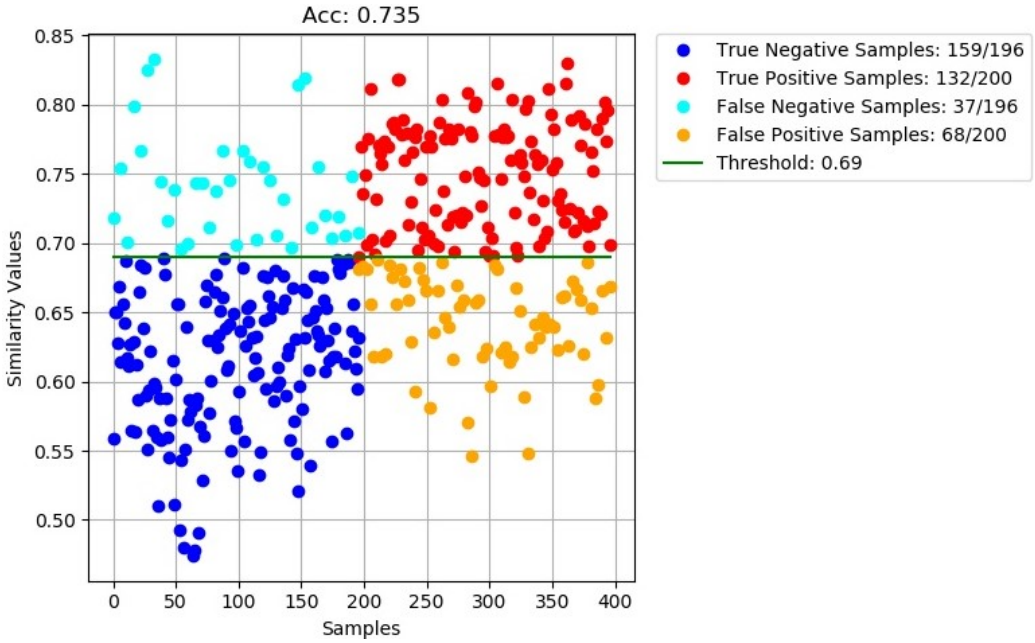


Figure 4.17: The results of performing Template Matching on the Caltech Buildings [38] dataset.

The configuration used in figure 4.16, again, is the same as in figure 4.15 and figure 4.10. Nevertheless, the accuracy is once again worse than the ResNet50 implementation [13] but better compared to the ResNet101 [13] one. That could be because of the better and more filters that ResNet152 [13] uses. Figure 4.17 utilises two ranges, the same as in figure 4.15. It can be observed that the accuracy is worse, which could be because of the large amount of features present. As mentioned before, extracting more features has a limit, after which distortions occur on the extracted vectors. Some features get miss-matched leading to lower similarity value.

In conclusion, from the above-presented graphs, ResNet101 and ResNet152 [13] perform worse than ResNet50 on the Wiki_Commons dataset [10] and similarly on the Caltech Buildings [38] one. Nevertheless, ResNet50 [13] proves to be faster in terms of computational time. The time comparison results can be seen in the "Overall Evaluation" section below. There is a distinct trade-off when it comes to different ResNet [13] implementations. ResNet50 [13] was picked for this project because of its better accuracy on the Wiki_Commons dataset [10], a very close performance on the Caltech Buildings dataset [38], and its faster computation time. ResNet101 and ResNet152 [13] tend to distinguish well between positive and negative samples on the Caltech Buildings dataset [38], as it can be seen in figures 4.15 and 4.5. The positives are very high and the negatives are very low. However, the opposite can be observed for the Wiki_Commons dataset [10]. The results could be such because of the complexity of both

networks. The Caltech Buildings dataset [38] requires more features because it does not have panoramic images, as stated on multiple occasions before. Because ResNet101 and ResNet152 [13] have more convolutional layers, they would extract more features, thus having a bigger margin between positive and negative samples. Although, that is to a limit. The results are not very different from the ones recorded by ResNet50 [13]. Furthermore, having more features when comparing to only a small batch of patches would not necessarily produce better results, as it could be seen in figures 4.14 and 4.16.

4.2.2 Inception & VGG

Inception-ResNet

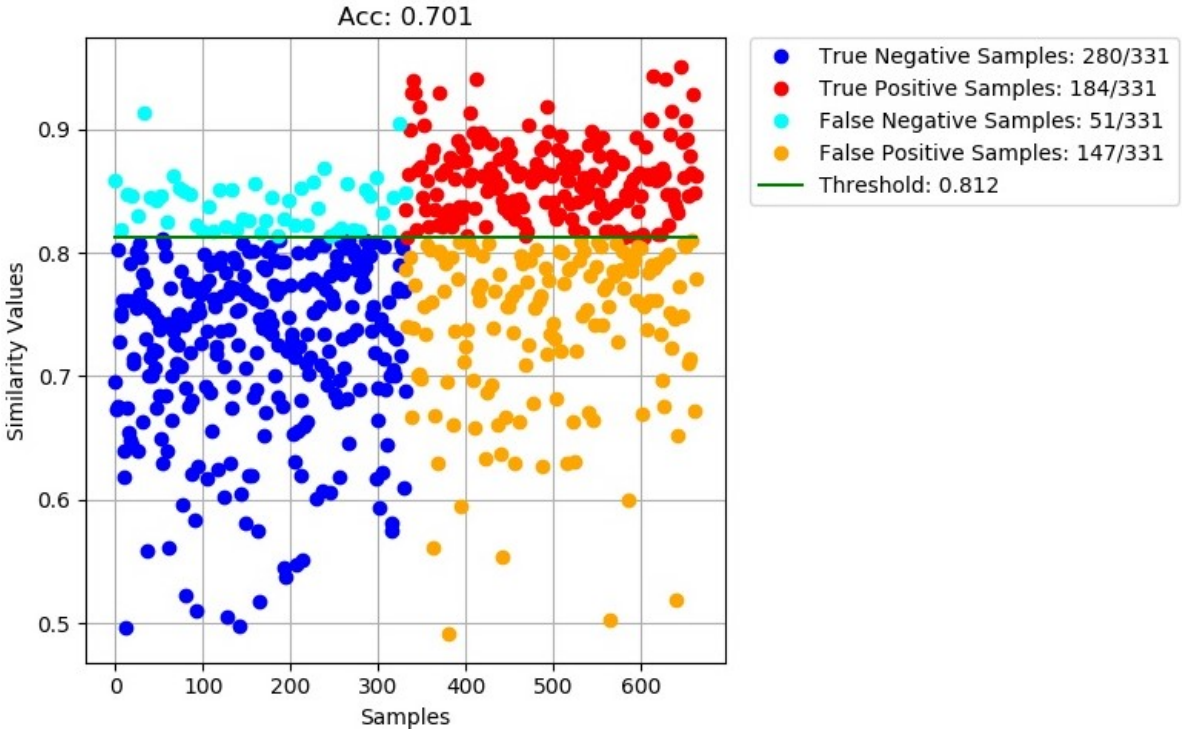


Figure 4.18: This figure provides the results of performing Patch Matching on the Wiki_Commons dataset [10]. The configuration used is the same as in figure 4.10. Despite both Residual Networks being different, the accuracy is almost the same. Although, the threshold differs from the one used in figure 4.10 quite significantly. This shows that Inception-ResNet [34] produces more features, which results in bigger similarity values. Additionally, there are no high negative samples and only low positive ones. This shows that the network extracts good features that match positive matches and do not mismatch negative ones. However, the threshold line is not centered given the y-axis, and that could be why the accuracy is not higher.

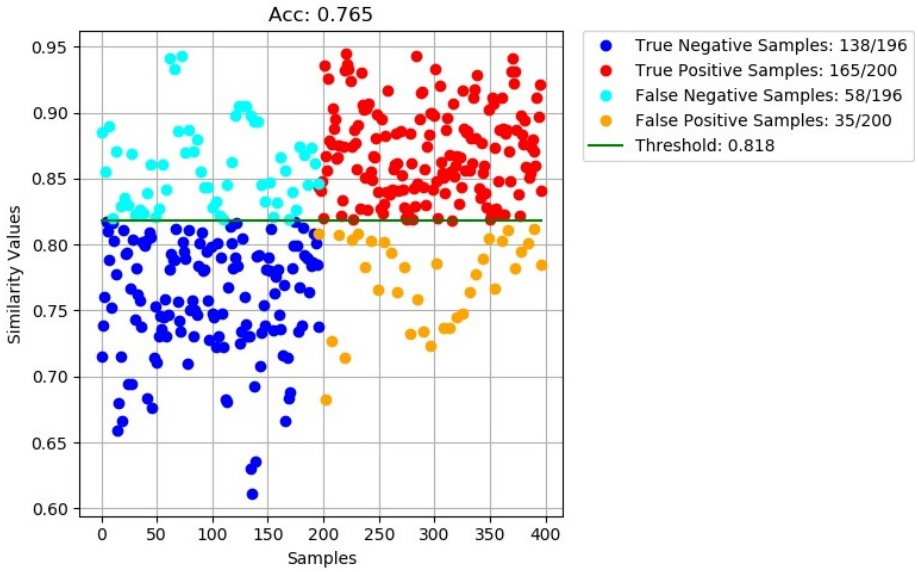


Figure 4.19: This graph shows the results on the Caltech Buildings dataset [38] using the configuration from figure 4.5, producing similar results. Although the accuracy is essentially the same as ResNet50 [13], this network proves to be significantly slower. This can be observed in the "Overall Evaluation and Comparison" section. This also shows that wider networks like Inception-ResNet [34] produce essentially the same results as deeper networks like ResNet50 [13]. However, it also shows that the network extracts robust enough features, which in some Residual Networks could be an issue as seen before [13].

VGG-19

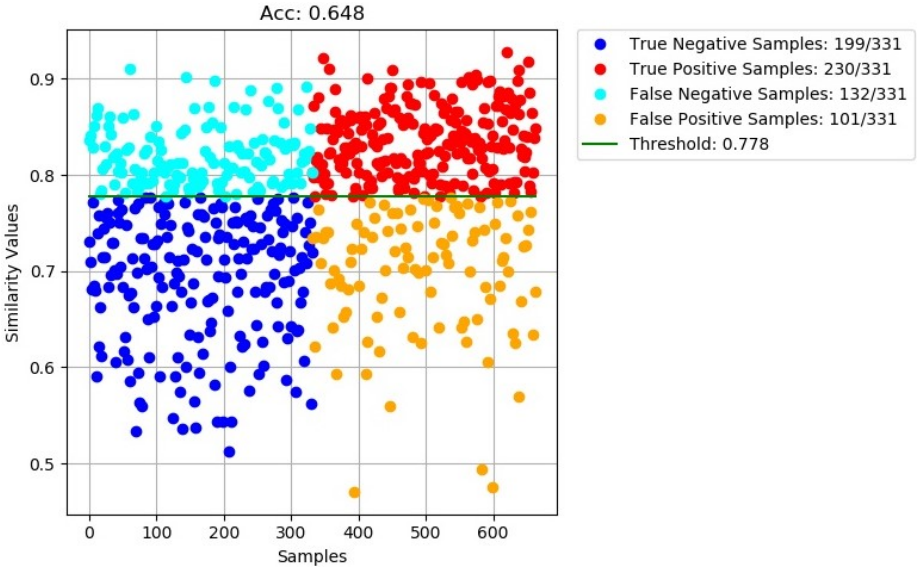


Figure 4.20: This figure provides the results on performing Patch Matching on the Wiki_Commons dataset [10], similarly to figure 4.10. An interesting observation can be made that a lot of positive samples produced low similarity values. That could be because of the more simplistic design of the CNN. Also, it could be because of the worse quality of extracted features compared to the ResNet50 [13]. It can be observed that all samples have generally high similarity value, which results in worse threshold separation of the data, as the data is heavily clustered in a span of 0.2 similarity value difference.

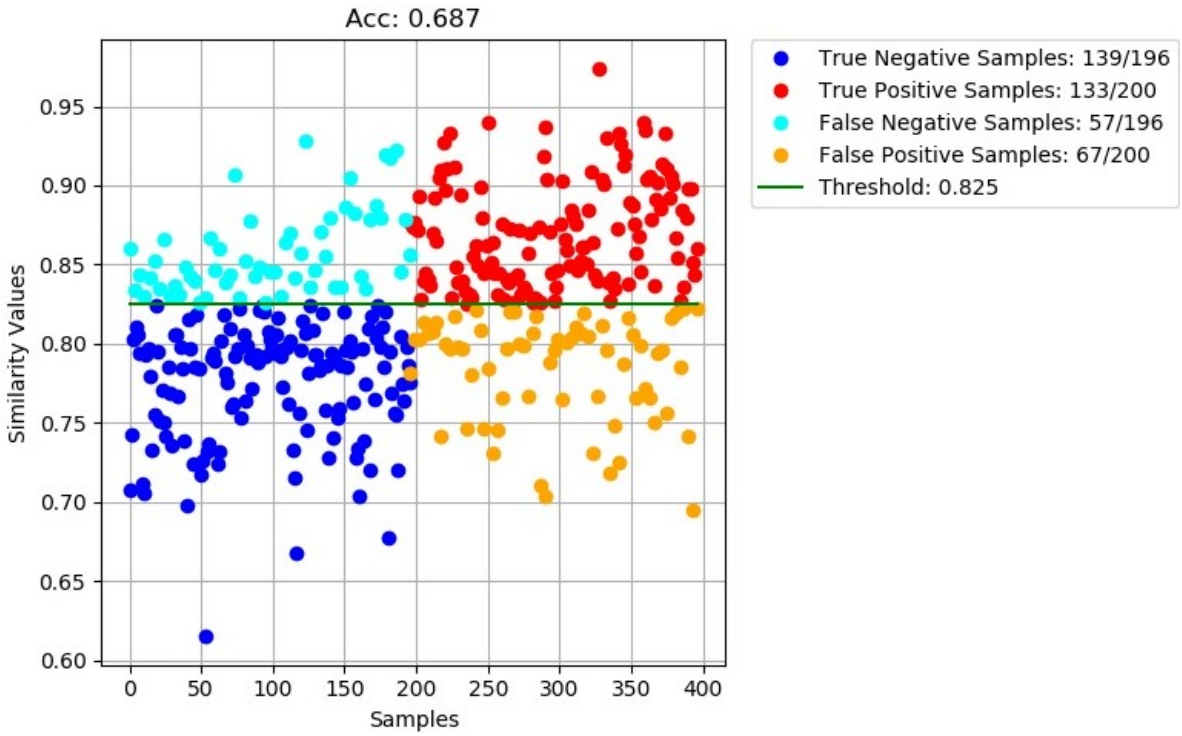


Figure 4.21: The following graph provides the result on the Caltech Buildings dataset [38], by performing Template Matching with the same configuration as in figure 4.5. Although it is the fastest network, as it can be seen in later sections, VGG-19 [31] records a significant drop in accuracy compared to ResNet50 [13]. There are a lot of low positive samples, which again could be because of the bad quality of the extracted features. The network tends to match query features that are not initially present on the reference image, as they are either distorted or simply not robust enough.

4.3 Normalized Cross-Correlation against Correlation Coefficient

This section compares Normalized Cross-Correlation [36] with Correlation Coefficient [14] to find the best measure given the context of the data in use. Only Normalized Cross-Correlation [36] has been picked for testing because it is the closest to the similarity measure used in this project. Also, these two measures are generally preferred when template matching. Additionally, it should perform better than the Cross-Correlation [35] one as it is more complex and robust. The configurations for every test run are the same as in the previous sections. That includes the two different Template Matching and Patch Matching methods as well as the best ResNet50 [13]. The number of extracted patches and scaling sizes are the same as in the best test runs recorded.

4.3.1 Normalized Cross-Correlation Evaluation

This similarity measure is very similar to the Correlation Coefficient [14] one used in this project. The only difference is that the latter centers each pixel around 0, which is represented by subtracting their mean. Normalized Cross-Correlation [36] has been picked as a comparison to showcase the difference of subtracting the mean of each pixel. Additionally, the Correlation Coefficient [14] does provide information about the magnitude and the direction of the relationship. On the other hand, Normalized Cross-Correlation [36] only gives information about the direction, and it is not invariant to the magnitude. As the following results show, this similarity measure does not perform as good as the one used in this project. Nevertheless, it remains an import similarity measure that should be tested.

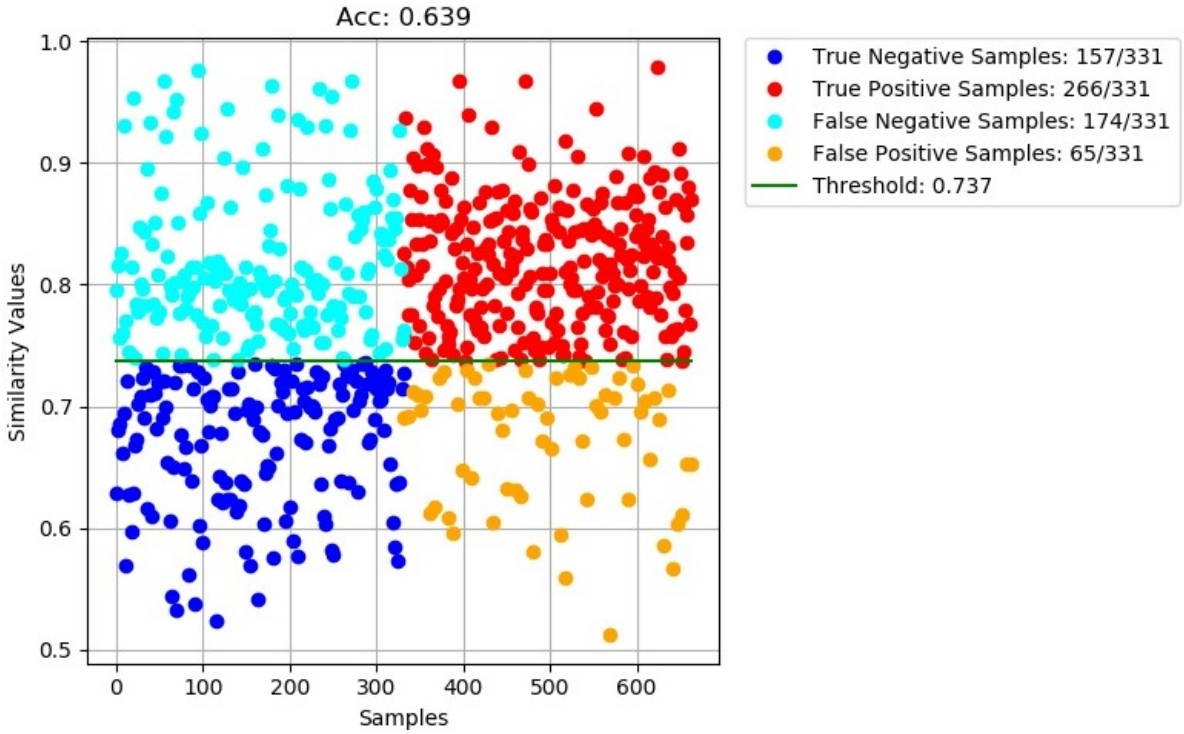


Figure 4.22: This figure shows the results of running the Patch Matching method on the Wiki_Commons dataset [10] with the same configuration as the best-recorded result in figure 4.10. From the graph, it can be observed how inconsistent sample predictions are. There are a lot of very high negative samples. Normalized Cross-Correlation [36] is not invariant to changes scale. Because the query object could be of any size on the reference image, the measure cannot match it properly to the fixed-sized object in the query image. Additionally, the threshold line is position significantly low on the y-axis, which shows further that the data is not easily separable.

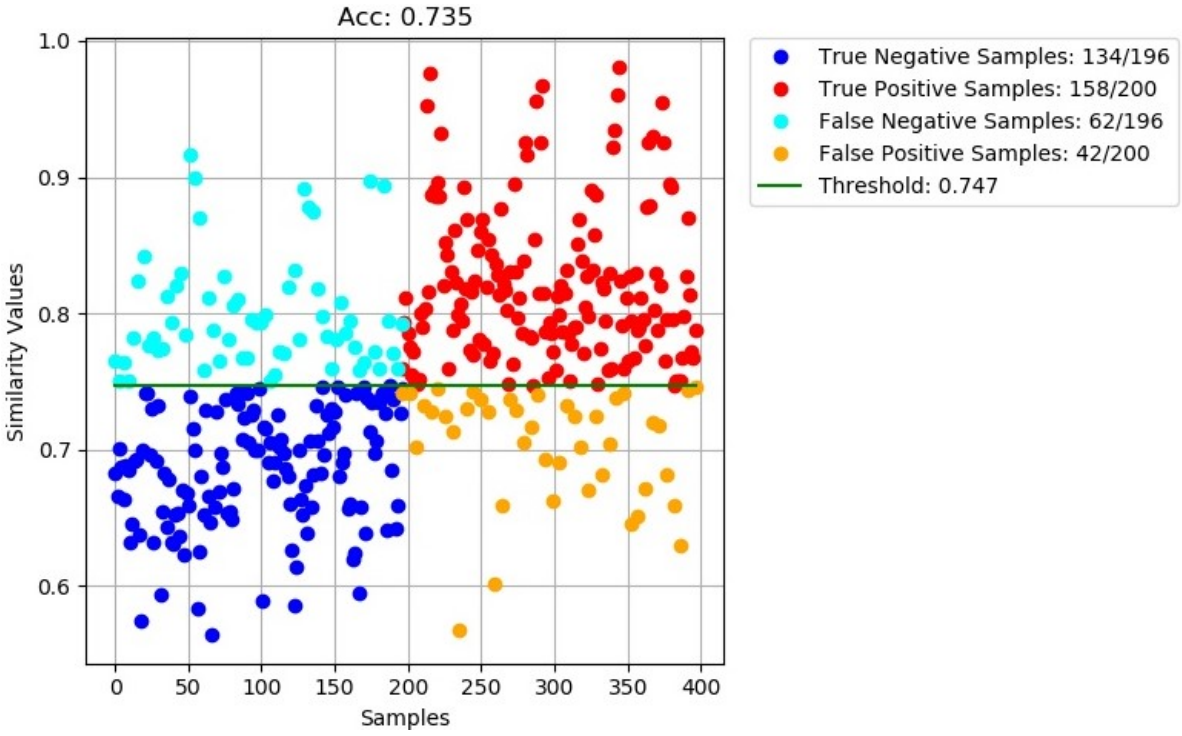


Figure 4.23: This figure presents the results of using the Template Matching method on the Caltech Buildings dataset [38]. The configuration of this test run matches the one from figure 4.5, which was also the best-recorded one for that dataset. The graph shows that this similarity measure performs worse than the Correlation Coefficient [14] one on both datasets. The recorded results could be because of the different sizes of the images or because of the different locations of the pixels given the object. Nevertheless, the threshold line does not manage to separate the data well enough due to the high fluctuation in similarity values.

To conclude, Normalized Cross-Correlation [36] has a known problem when the vectors are of different magnitude. This is further explained and depicted in [40]. Because this project implements methods that re-scale both images and change their dimensions significantly, the Correlation Coefficient [14] metric is preferred to the Normalized Cross-Correlation one [36]. It must be noted, however, that if the scaling factor is removed, Normalized Cross-Correlation [36] could perform on the same level or even better. Nevertheless, completely removing the problem of different magnitudes is infeasible in practice. Therefore, Normalized Cross-Correlation [36] tested worse on both datasets than the currently used Correlation Coefficient [14] similarity measure.

4.4 Overall Evaluation and Comparison

To summarise, the best similarity measure was found to be the Correlation Coefficient [14]. Additionally, Template Matching performs better when matching images of non-panoramic type, where on the other hand, Pattern Matching results in better accuracy when the reference image is a panorama. When there are no panoramas in the data, the Patch Matching method is not preferred because the extracted patches do not contain enough information about the reference object. On the contrary, Template Matching is preferred when there are no panoramas because the reference object is represented as a whole, rather than divided into different patches. In this way, the features extracted from the query image have a better probability of matching with the reference image features. The two best-recorded configurations were:

- Template Matching - the query image is resized depending on its dimensions. If the shape of the query image is bigger than the shape of the reference image, then the range of scaling sizes is [10, 12,...,20]. However, if it is the opposite, the range is [22,24,...,30]. By having two different ranges makes the model more robust to different image sizes. That is true even when Template Matching is used with panoramic images. If the query is bigger than the reference image it would mean that further down-scaling must be applied to perform feature matching, especially when the object of interest in the reference image is of small size.

- Patch Matching - the query image is resized a certain percentage of its size, to make it more suitable for matching the small 224 by 224 patch size. The best configuration for this method is by re-scaling the query image by 15% and producing 250 patches. It must be noted that depending on the hardware more patches could be extracted, which could benefit in better accuracy.

In terms of Feature Extraction methods, the ResNet50 [13] proves to be better overall than the Resnet101/152 [13], VGG [31] or Inception-ResNet [34]. Although the performance varies between different networks, they have all recorded lower accuracy on the Wiki_Commons dataset and almost the same on the Caltech Buildings dataset [38] when compared to the ResNet50 network [13]. However, another point to make is that ResNet50 [13] is less computationally expensive due to its more simplistic design. The time comparison between all networks can be seen in table 4.1. The difference in accuracy can be explained by either the lack of depth and complexity of the VGG [31] and Inception-ResNet [34] networks, or because of the excessive complexity of the other ResNet network implementations [13].

Network	Test Dataset	Relative Time	Average relative sample time
ResNet50	Wiki_Commons	187 min	20s
ResNet50	Caltech Buildings	13 min	2s
ResNet101	Wiki_Commons	215 min	23s
ResNet101	Caltech Buildings	20 min	3s
ResNet152	Wiki_Commons	230 min	24s
ResNet152	Caltech Buildings	33 min	5s
InceptionResNetV2	Wiki_Commons	300 min	32s
InceptionResNetV2	Caltech Buildings	14 min	2s
VGG-19	Wiki_Commons	140 min	15s
VGG-19	Caltech Buildings	7 min	1s

Table 4.1: The following table shows the computational time results of performing different CNNs (Column 1) on two different datasets (Column 2). The third column shows the total time for a test run, where column 4 shows the average time for predicting a single sample. Moreover, the green rows show the network with the best time for a dataset, and the red rows represent the network with the worst times respectively. It can be observed that the model used in this project, ResNet50 [13], comes second, behind the VGG-19 [31]. However, as can be seen in the respective result sections, ResNet50 [13] outperforms VGG-19 [31] in terms of accuracy on both datasets. ResNet152 [13] and Inception-ResNet [34] are the two slowest networks. This project picks ResNet50 [13] as the default Residual Network for Feature Extraction as a good compromise between accuracy and computation time. It must be noted that the times can change depending on the hardware or Google Colab's [37] Servers.

Additionally, the proposed model performs worse on the Wiki_Commons dataset [10] compared to the BUPM [10] method. Although there is a certain trade-off that must be noted. The BUPM [10] method requires to be trained twice on two different huge image datasets, where the proposed model in this report does not.

Moreover, this model performs better than SURF [4], as it could be seen in figures 4.25 and 4.24. However, the SURF [4] method took significantly less time to perform. 4.25 and 4.24 show the results on running the method on the two datasets - Wiki_Commons [10] and Caltech Buildings [38]. The difference in computation time is because SURF [4] is significantly less complex than both the proposed model and the BUPM [10] method. Nonetheless, the average sample time on the Caltech Buildings dataset [38] is the same as when using the pre-trained ResNet50 [13]. Both models recorded an average sample time of around 2 seconds for that dataset. However, there is a difference when testing on the Wiki_Commons dataset [10]. The SURF [4] method recorded an average time of around 1 second where, as can be seen in table 4.1, ResNet50 [13] recorded an average time of 20. Nonetheless, the big difference in accuracy makes the proposed model more preferable. The best accuracy recorded by using the SURF [4] method on both datasets is 0.524 on the Wiki_Commons [10] and 0.73 on the Caltech Buildings [38]. Where the proposed model recorded accuracy of 0.713 from figure 4.10 and 0.765 from

figure 4.5 on both datasets respectively. It must be noted that the configurations for both SURF [4] figures below are different to the ones presented in previous sections. Because SURF [4] utilises different matching techniques (the FLANN-based k-NN [41]) the similarity value is actually the number of matched points by the matcher. The threshold, therefore, tries to separate the data given the number of matched key points. Additionally, the threshold value needs to be adjusted accordingly.

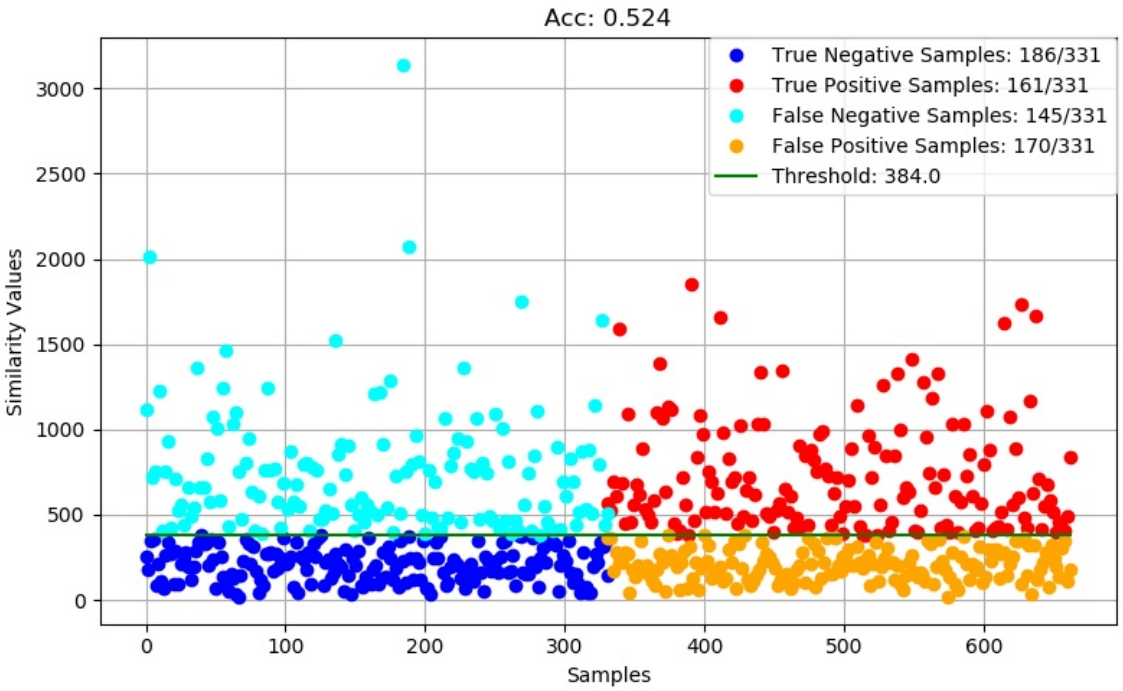


Figure 4.24: This figure shows the results of running the SURF [4] method on the Wiki_Commons dataset [10]. The accuracy recorded is worse than the one produced by any of the configurations used to test the proposed model on this dataset. The query and reference images have been initially resized by a fixed 80% of their size to mitigate the risk of memory overflow, as most of the images in the dataset are of big dimensions. All the samples' predictions are extremely fluctuating, thus the threshold is almost rendered unusable. Additionally, it could be seen that there are a lot of very low positive and high negative samples. The threshold line is out-centered and does not separate the data effectively. These results could be due to the low robustness and complexity of the method for that specific dataset. On the other hand, the computation time was around 662 seconds, which is significantly lower than the time spent by the proposed method. This could be explained by the lack of need to re-scale images as well as extract features using a CNN. It must be also noted that this method has not been initially designed to perform matching on panoramas, but on images with only one object present, [4]. This could be the explanation for the bad accuracy.

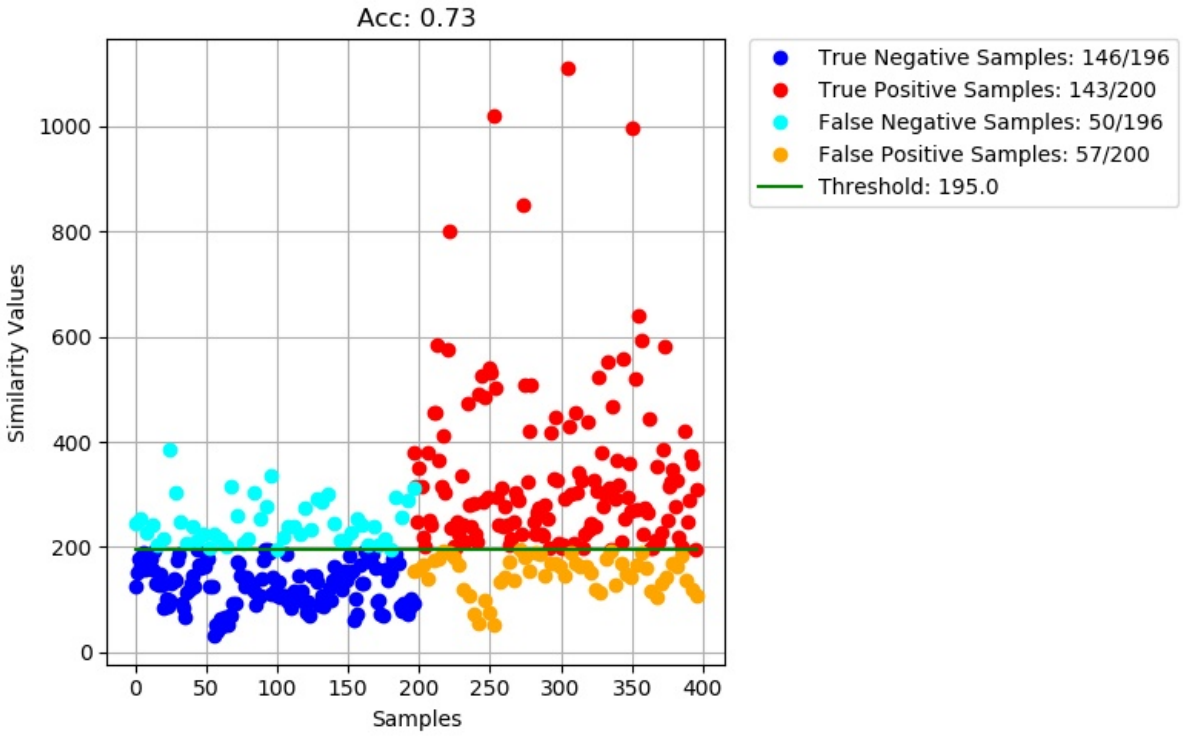


Figure 4.25: The following graph presents the results on the Caltech Buildings dataset [38]. Similarly, as figure 4.24, this test run produced worse results than the model proposed by this paper. The query and reference images have been resized by 80% of their sizes to avoid memory overflow. The computational time was around 792 seconds, which matches the computation time needed for this project. Although this figure shows less high negative samples, it still records a significant amount of low positive predictions, thus resulting in worse accuracy. The threshold is not centered, however, it still manages to separate the data well.

In conclusion, the SURF [4] method performed worse on both datasets compared to the proposed model. This could be due to the lack of robustness and complexity of the method. It must be noted that it extracted, on average, more matches when predicting on the Wiki_Commons [10] dataset. This could be explained by the type of the data used. As mentioned before, SURF [4] has not been designed to match an image with a panorama. Therefore, the method matches a lot of features that are not actually present in the query image. This leads to misclassification of the predicted samples, resulting in the worst accuracy recorded in this report.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Residual Networks in conjunction with similarity measures have shown potential in solving image verification problems in Computer Vision. With the increasing accuracy and robustness of CNNs, this model has shown significant improvements in terms of feature extraction. Therefore, using Residual Networks to find points of interest might be the correct way going forward in trying to solve the problem of Image Verification. However, it is reasonable to assume that similarity measures perform worse than CNNs when it comes to matching features. Nevertheless, there is a significant decrease in resources required when using similarity measures, as no training is needed. The proposed model showed promising results in verifying two images, in terms of both accuracy and computation time. Thus, the combination of pre-trained Residual Networks with the use of different similarity measures might solve the problem of Image Verification in the near future.

This project had the motivation of providing a model that performs fast and accurately, without a huge trade-off between the two. The motivation behind it is covered more in-depth in Chapter 1. This project uses a combination of methods that were previously used by other solutions. Chapter 2 explores such solutions by providing information about their performance as well as the technologies in use. By utilising technologies from several of the best solutions covered in Chapter 2, Chapter 3 presents an explanation and design description of the newly proposed model. It can be said that it is a good balance between classical methods and new methods. In Chapter 4 it can be seen that the proposed model performs better than the classical algorithm SURF [4] in terms of accuracy and worse in terms of speed. However, the opposite

can be observed when comparing to the more recent BUPM [10] model. These results depict this project as a balanced solution to the Image Verification problem, where one does not need to compromise speed for accuracy, and vice versa.

5.2 Future Work

Given that the proposed solution implements different technologies that are not usually combined, leaves the possibility for further research and modifications. In terms of feature extraction, Residual Networks prove to be the best solution, as it can be seen in Chapter 4. Nevertheless, a new and different approach can be taken where the combination of outputs of different layers is taken rather than only from the last layer. This can further increase the robustness and quality of the features extracted. Also, using the method of a similarity measure for matching features can be further adjusted and altered to increase accuracy. Using different combinations of similarity measures is an area where further research can be conducted. The best similarity measure may vary between datasets, hence a method that implements several measures can be implemented. Other types of improvements can include experimenting with different pre-processing algorithms as well as different variations of scaling sizes. However, using more scaling sizes or more patches might either increase or decrease the accuracy, depending on how robust the similarity measure is. Also, different patch sizes can be implemented and tested. This could improve the accuracy as objects on panoramic images are of various sizes.

Moreover, a pre-trained Dense Neural Network can be used to verify a pair of images. Therefore, removing the need for training on huge image datasets, but at the same time utilising the complexity and robustness of networks, which as per Chapter 2, prove to be more accurate. However, more research can be also conducted on verifying with a threshold. This project uses a static threshold, meaning it does not change between samples. A new dynamic threshold can be introduced that adjusts itself, with a marginal error, for different samples. There have been studies on this concept, where a dynamic threshold is introduced with the help of different algorithms. The authors of [42] propose a novel approach by using a dynamic threshold to improve detection sensitivity and to repress the influence of noise fluctuation. Although this method is for signals and also used in a different manner, a similar approach can be implemented for the Image Verification problem in Computer Vision. Nevertheless, this project leaves a lot of possibilities for further research, that can eventually solve or contribute to solving not only Image Verification problems but also other different tasks in Computer Vision.

Chapter 6

Legal, Social, Ethical & Professional Issues

The model implemented by this paper is designed to be used as a professional tool in the respective professional environment. The British Computer Society was taken into strong consideration and was made sure that the project abides their Code of Conduct [43]. Therefore, this chapter takes into account the following issues:

6.1 Plagiarism

The implemented model utilises multiple libraries and different IDEs. The libraries used are all open-source and are fully stated in appendix C.1 of this paper. They have all been used within the permitted boundaries. Additionally, Google Colab [37] has been used as the main IDE when testing. It has a free membership that enables only a limited amount of available hardware to be used openly. This project utilises the free membership provided by Google Colab [37].

Moreover, the proposed model implements different AI, Image Verification and Similarity Measure methods. Most of them have been used previously in other solutions. All of these methods have been referenced accordingly throughout the paper, given credit where applicable. The SURF method [4], which is also available to use in this project, is patented. Thus, it cannot be used for commercial use unless given permission by the authors [4]. Nevertheless, if this project is to be used for commercial use, every library or method implemented must be made sure that allows to be used for commercial use. Additionally, they must be referenced

and given credit to accordingly.

6.2 Competence & Risk

This project has been made with the intention of being open-source. This is done to encourage further improvements and modifications that can be done to the model itself. Additionally, missed or wrongful implementations will be revealed and rectified.

The libraries used in the project should not affect the stability of this project. Execution errors like memory overflow can occur while using the model. This depends on the hardware used as well as the images passed through the model. Certain modifications have been put in place to account for that. However, the error could still occur if the images predicting cannot be stored in memory. Additionally, this would also affect the computation time of the project. It should not affect the performance of the model.

6.3 AI & Ethics

This project implements different AI methods and techniques. The intent of this project is to not be used for any harm, either individually or as a part of another solution, to any business, individual, etc. In terms of AI bias, the initial goal of this project is to be used primarily on buildings or open areas without clearly distinguishable people or faces. However, if this model is to be modified for the use of comparing pictures of human beings or other sensitive data, the risk of the model being biased must be accounted for. This includes, but it is not limited to, race, ethnicity, gender and age. Furthermore, this project is not robust, secure or accurate enough to be fully trusted by either people or machines. It is not to be used as part of any self-driving AI or AI that might affect the health or well-being of any individual. However, it can be used as a valid solution nonetheless, provided that a specific margin of error is presented to its users.

References

[1] Ugur Demir and Gozde Unal. Patch-based image inpainting with generative adversarial networks. *arXiv preprint arXiv:1803.07422*, 2018.

[2] *Design & Reuse, Evolution of Object Recognition in Embedded Computer Vision.* https://www.design-reuse.com/articles/37837/object-recognition-in-embedded-computer-vision.html.

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2), 2004.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*. Springer, 2006.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[6] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[7] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.

[8] Tali Dekel, Shaul Oron, Michael Rubinstein, Shai Avidan, and William T Freeman. Best-buddies similarity for robust template matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[9] Itamar Talmi, Roey Mechrez, and Lihi Zelnik-Manor. Template matching with deformable diversity similarity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[10] Jiaxin Cheng, Yue Wu, Wael Abd-Almageed, and Prem Natarajan. Image-to-gps verification through a bottom-up pattern matching network. In *Asian Conference on Computer Vision*. Springer, 2018.

[11] *Visual Perception-From Human Vision to Computer Vision.* https://medium.com/towards-artificial-intelligence/visual-perception-from-human-vision-to-computer-vision-3152134292.

[12] *Hacking the Brain With Adversarial Images.* https://spectrum.ieee.org/the-human-os/artificial-intelligence/machine-learning/hacking-the-brain-with-adversarial-images.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[15] *The New Wave of Color Calibration Technology: Cube Calibration.* https://hometheaterhifi.com/reviews/video-accessory/video-calibration/the-new-wave-of-color-calibration-technology-cube-calibration/.

[16] *Image Gradient Wikipedia.* https://en.wikipedia.org/wiki/Image_gradient.

[17] Ian T Young and Lucas J Van Vliet. Recursive implementation of the gaussian filter. *Signal processing*, 44(2):139–151, 1995.

[18] Tuan Q Pham. Non-maximum suppression using fewer than two comparisons per pixel. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 438–451. Springer, 2010.

[19] Ranita Biswas and Jaya Sil. An improved canny edge detection algorithm based on type-2 fuzzy sets. *Procedia Technology*, 4:820–824, 2012.

[20] *Gaussian Filter Library.* https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html.

[21] *Harris And Stephens Corner Detector.* https://medium.com/analytics-vidhya/introduction-to-harris-corner-detector-32a88850b3f6.

[22] *Harris and Stephens Corner Detector Wikipedia.* https://en.wikipedia.org/wiki/Corner_detection.

[23] RP Kanwal and KC Liu. A taylor expansion approach for solving integral equations. *International Journal of Mathematical Education in Science and Technology*, 20(3):411–414, 1989.

[24] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[25] Lucas Assirati, Núbia Rosa da Silva, Lilian Berton, Alneu de A Lopes, and Odemir M Bruno. Performing edge detection by difference of gaussians using q-gaussian kernels. In *Journal of Physics: Conference Series*, volume 490, page 012020. IOP Publishing, 2014.

[26] *Speeded-Up Robust Features Wikipedia.* https://en.wikipedia.org/wiki/Speeded_up_robust_features.

[27] Ruan Lakemond, Clinton Fookes, and Sridha Sridharan. Affine adaptation of local image features using the hessian matrix. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 496–501. IEEE, 2009.

[28] *Histogram of Oriented Gradients Wikipedia.* https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.

[29] *Convolutional Neural Networks (CNN): Summary.* https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-summary/.

[30] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[34] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[35] *Dot Product Wikipedia*. https://en.wikipedia.org/wiki/Dot_product.

[36] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 2, pages II–II. IEEE, 2006.

[37] Ekaba Bisong. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. Springer, 2019.

[38] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[39] *Google Colab Resources*. https://research.google.com/colaboratory/faq.html.

[40] Arash Heidarian and Michael J Dinneen. A hybrid geometric approach for measuring similarity level among documents and document clustering. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 142–151. IEEE, 2016.

[41] Cong Fu and Deng Cai. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*, 2016.

[42] Guicai Yu, Chengzhi Long, Mantian Xiang, and Wei Xi. A novel energy detection scheme based on dynamic threshold in cognitive radio systems. *Journal of Computational Information Systems*, 8(6):2245–2252, 2012.

[43] *BCS, THE CHARTERED INSTITUTE FOR IT, CODE OF CONDUCT FOR BCS MEMBERS*. https://cdn.bcs.org/bcs-org-media/2211/bcs-code-of-conduct.pdf.

[44] Mohamed Aly, Peter Welinder, Mario Munich, and Pietro Perona. Towards automated large scale discovery of image families. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–16. IEEE, 2009.