Random Forest Induced Similarity Measures for Image Template Matching

Ramapriya Kyathanahally Janardhan
ID - 396501

In partial fulfillment of the requirements
for the degree of
Master of Science

Date of submission: February 15, 2019

Examiners : Prof. Dr. Olaf Hellwich, Prof. Dr. Axel Küpper
Advisors : Dr. Ronny Hänsch

Technical University of Berlin
Faculty of Electrical Engineering and Computer Science
Department of Computer Vision & Remote Sensing

Declaration

I hereby declare in lieu of an oath that I have produced this work by myself. All used sources are listed in the bibliography. All content taken directly or indirectly from other sources is marked as such. This work has not been submitted to any other board of examiners and has not yet been published. I am fully aware of the legal consequences of making a false declaration.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig ange- fertigt habe. Andere als die angegebenen Quellen wurden nicht verwendet. Die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen wurden als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vor- gelegt und auch noch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ort/Datum	Ramapriya Kyathanahally Janardhan

Abstract

Random forests are one of the strongest modern methods of machine learning and are applied in many different applications. They are an ensemble of decision trees. The basic idea is to generate many decision trees, where tree creation is subject to a certain amount of randomization. The final system answer is the fusion of the result of individual trees. Other than conventional machine learning tasks such as classification, regression, and clustering, random forests can also be used to identify the similarity between data points. In general, each data point follows a specific path through the decision tree. The more similar two data points are, the more likely will their paths overlap. The overlap of their paths in multiple decision trees can, therefore, be used as a similarity measure. In this paper, we used the random forest to identify the similarity between image patches. We compared random forest similarity measure using two applications: (i) we replaced the rectangle filter similarity measure between image patches described in [27] with the random forest similarity measure, and performed template matching, and (ii) we used random forest similarity distance instead of the SURF descriptor distance to evaluate template matching using SURF key point matching. The results we obtained show that template matching using random forest similarity measure performed better when compared to template matching technique proposed in [27]. The results also show that template matching from key point matching using random forest similarity distance performed very close to the method which utilized SURF descriptor distance for key point matching. Hence, we can say, random forests can be potentially used to determine similarities between data points. The derived distance measure thus obtained can be utilized in computer vision tasks such as clustering, image retrieval, classification, or key point matching.

Kurzfassung

Random Forest ist eine der modernsten Methoden im maschinellen Lernen und werden in vielen verschiedenen Anwendungen eingesetzt. Die Grundidee besteht darin, viele einzelne Entscheidungsbäume per Zufall zu generieren und anschließend die Ergebniße einzelner Bäume zusammenzuführen. Neben konventionellen maschinellen Lernaufgaben wie Klaßifizierung, Regreßion und Clustering können auch Random Forest dazu verwendet werden, die Ähnlichkeit zwischen mehreren Datenpunkten zu ermitteln. Allgemein folgt jeder Datenpunkt einem bestimmten Pfad durch die Entscheidungsbäume. Je ähnlicher zweier Datenpunkte sind, desto wahrscheinlicher überlappen sich ihre Pfade. Die Überlappung der Pfade in Entscheidungsbäume kann daher als Ähnlichkeitsmaß verwendet werden. In dieser Arbeit benutzten wir Random Forest um die Ähnlichkeit zwischen Image Patches zu bestimmen. Außerdem vergleichen wir das Ähnlichkeitsmaß mit zwei Anwendungen: (1) Wir ersetzten das Ähnlichkeitsmaß für den Rechtecksfilter in [27] durch das Random Forest Ähnlichkeitsmaß und führte einen Template Matching durch. (2) wir verwendeten Random-Forest Ähnlichkeitsabstand anstelle des SURF-Deskriptorabstands, um den Template Matching mithilfe des SURF-Key Points zu evaluieren. Die erzielten Ergebniße zeigen, daß der Template Matching mit Random-Forest Ähnlichkeitsabstand beßer war als die in [27] beschriebene Template Matching. Weiterhin zeigen die Ergebniße, daß der Template Matching von Key Points unter Verwendung von Random Forest fast genauso gut abschneidet wie die verwendete Methode bei SURF Deskriptor. Daraus kann man schlußfolgern, daß die Methode mit Random Forest geeignet ist, Ähnlichkeiten zwischen Datenpunkten zu bestimmen. Die daraus gewonnene Abstandswerte kann ausgenutzt werden, um Aufgaben im Bereich Computer Vision wie Clustering, Image Retrieval, Klaßifizierung oder Key Point Matching durchzuführen.

Acknowledgements

I would genuinely like to express my sincere thanks to all those who supported me while writing this thesis. It would have not been possible for me to complete my thesis without the support and guidance of my mentor, family and friends. I would like to express my deepest gratitude to:

Dr. Ronny Hansch, for being my advisor and mentor

Pruthvi Sreenath, for his extraordinary guidance and motivation

Rucha and Berlin friends, for their excellent advice and support

My Parents and Sisters, for their eternal love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Subject and Motivation

The ability to identify similarities between images has been the basis of many computer vision problems such as patch recognition, texture and scene categorization, face detection, and object recognition. It is often necessary to define a similarity function or measure between the data points in these images. In computer vision applications, these data points can be individual image pixels, image patches or a large template. Methods which deal with pixel level similarity are slow and doesn't take into account the semantics associated with the images. Methods which use large templates are sensitive to local variations and partial occlusion. Hence image patches which come in between pixels and the large template can be used to identify the similarity between images. But before we use these image patches to identify the similarity between images, we need a sophisticated function to identify the similarity between patches themselves.

One way to compare patches is by comparing the intensity of each pixel in them using some distance function such as Euclidean distance or Mahalanobis distance. But as the patch size and the number of patches increase, the computation complexity in comparison will also increase. The other methods such as [27] use rectangle filters and integral image representations to identify the similarity between patches. They are fast but are not accurate, which we will discuss as part of this work. Can we use any other tool that accurately identifies the similarity between patches with few similarity tests? Yes [37]. We can use Random forest to accurately identify the similarity between patches.

Random forests [9] are a popular and powerful ensemble machine learning tools for classification, regression and clustering problems. They possess several properties that make them particularly interesting for computer vision applications. They are fast in training and classification. They can be easy parallelized, which makes them interesting for multi-core and GPU implementations [65]. The other advantage of using them is, we need not work with few precomputed and simple features. Instead, features can be computed on the fly by the internal nodes of the decision trees [37]. This provides a virtually infinite amount of possible tests at each node. But only a very small subset is required to yield a good classification rate. One approach to identifying the similarity between image patches using random forests are by dropping patches through them and check whether they end up in the same leaf node. But through this approach, we cannot accurately determine by what percentage the patches are similar. Through this approach, we might lose some information in

comparing similarity. So is there a way we can calculate the similarity without losing information? Can we use the decisions resulted in classification to identify the similarity? Many machine learning approaches don't provide this flexibility of analysing the decisions resulted in classification and use them for solving other problems. But random forest has this advantage. We can track the decision paths taken by each patch. If the patches are more similar, the more likely their paths will overlap. The overlap of their paths can, therefore, be used as a similarity measure. Hence with all these advantages, in this work, we are using random forests to identify the similarity between image patches.

Once we identify the similarity between image patches, they can be used to identify the similarity between images. This approach is called Template matching. It is a technique for finding the region of an image that best matches the template. In this paper, we perform template matching using random forest similarity measure and show how the random forests can be used to determine similarities between data points. We compare the performance of random forest similarity measure using two applications - template matching using image patch matching and template matching using key point (patch) matching. In both applications, we use random forest similarity score to compute the similarity between image patches.

1.2 Introduction to Random forests

1.2.1 Decision trees

Random forests are based on the idea of decision trees. In decision analysis, a decision tree can be utilized to represent decisions and visualize the decision-making process. Decision trees are top-down hierarchical structures built on answers to one or more yes or no questions. They are mainly designed to address classification or a regression problem. Classification decision trees predict the class to which the data belong. For example, it predicts if the outcome of a therapeutic test is positive or negative. Regression decision trees predict continuous values such as the population of a state, price, range etc. A decision tree begins with a root node, followed by split nodes and ends with terminal leaf nodes. Root node and each split node in the decision trees lead to branches depending on the number of decision split conditions used in the node. A binary split condition leads to a true or false branch. A ternary split condition leads to three or more branches. Binary trees are utilized more often. Finally, leaf nodes predict the class to which the data belong in the case of classification, or predict a number in case of regression. Figure 1.1 shows an example decision tree trained to classify a fruit.

To generate a decision tree, dataset (S) whose properties are known before is split recursively. The properties of each data sample are called features. Each data sample has a label associated with it. During tree generation, all features are considered and different split conditions are tested such that each split condition leads to subsets that are as pure as possible regarding the label. In general, we want the split function to divide the data in a meaningful manner. We can define a

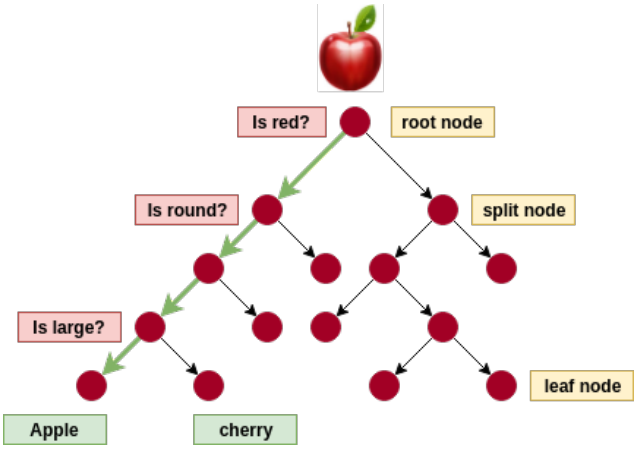


Figure 1.1: A sample decision tree trained to classify a fruit

measure of purity for each subset obtained by split condition, such that the purer the subset, the more meaningful the split. The idea is tantamount to partition the data into subsets that contain instances with similar values (homogenous). The homogeneity of such partitioned subsets is computed using Shannon Entropy $H(S)$. If the subset is completely homogeneous, the entropy is zero and if the subset is equally divided it has an entropy one. Shannon entropy is defined as follows:

$$H(S) = \sum_{c \in Y} -P(c)log(P(c)) \tag{1.1}$$

where Y is the set of labels. The entropy above is for a single subset obtained using the split condition. For K different subsets obtained using the split condition, we define Expected Entropy EH as follows:

$$EH = \sum_{i=1}^{K} \frac{|S^i|}{|S|} H(S^i) \tag{1.2}$$

Once we have defined $H(S)$ and EH, we measure the information obtained from a split condition about the class using Information Gain I. The information gain is based on a decrease in entropy after a dataset is split on a test. During decision tree construction at each split node, the split test is chosen such that it returns the highest information gain using that split test. The information gain is defined as follows:

$$I = H(S) - EH \tag{1.3}$$

To summarize, the training of trees begins at the root node, followed by split nodes. At the root node and at each split node, the split condition is determined to obtain the best information gain. This procedure is repeated recursively until a

stopping criterion is met (that is information gain obtained from the split condition is zero or the maximum Depth D of the tree is reached).

There are various decision tree algorithms. The most widely used among these are ID3 (Iterative Dichotomiser 3), C4.5 [57, 58] and CART (Classification and Regression Tree) [10]. ID3 uses entropy based impurity measurement and CART uses Gini impurity measurement to find a good split. Even though decision trees are widely used for classification and regression tasks, they have few disadvantages. They are very inefficient in finding appropriate splits when the data is highly dimensional. The other disadvantage is that they have a strong tendency to overfit as the depth of the tree increases. We can overcome these shortcomings of decision trees using ensemble machine learning methods such as random forests.

1.2.2 Random forests

Random forests are an ensemble of decision trees where each tree creation is subject to a certain amount of randomization. The main idea behind Random forest is to try and mitigate problems shown by decision trees such as overfitting and lack of generalization, by (i) injecting randomness into the training of the trees, and (ii) combining the output of multiple randomized trees. Since the output of random forests is the combined output of multiple independently trained decision trees, the output will generalize better if diversity/randomness is introduced during tree generation. The randomness can be (i) randomness in data, and (ii) randomness in choosing split conditions at different decision nodes of decision trees. Randomness in the data is achieved by bagging techniques introduced in 1996 [7]. In this technique, multiple decision trees are trained on the randomly chosen subset from the training set and the output is averaged. Randomness in split conditions can be obtained by choosing the best split condition among K split conditions at each decision node in decision trees. The work [16] used this technique in 1998. In [2], a large number of geometric features are defined and searched over a random selection of these for the best split at each node in 1997. Breiman combined these ideas of introducing randomness in the training set selection and introducing randomness in split condition selection for training multiple independent decision trees, and discussed the basic properties of what he called Random forests in his paper [9] in 2001.

In order to train a random forest, initially, a bootstrap sample of size N is drawn randomly from the training dataset. Using this bootstrap sample, a decision tree is grown to a maximum depth D or until all samples reach a leaf node. At each decision node, the best split is chosen among K random split conditions. In a similar line, T independent decision trees where $t \in \{1, 2....T\}$ are trained. After training a random forest, it can be utilized to predict the class to which a data sample belongs in case of classification or to predict a numerical value in case of regression. The prediction/outcome of random forest can be calculated as follows: (i) A test point v is simultaneously pushed through all the T decision trees starting at the root until it reaches the corresponding leaves. (ii) The prediction P_t for v is recorded from all the

T decision trees. (iii) The final prediction is obtained by averaging the prediction from all the T decision trees.

$$P = \frac{1}{T}\sum_{i=1}^{T} P_t \tag{1.4}$$

1.3 Introduction to Template matching

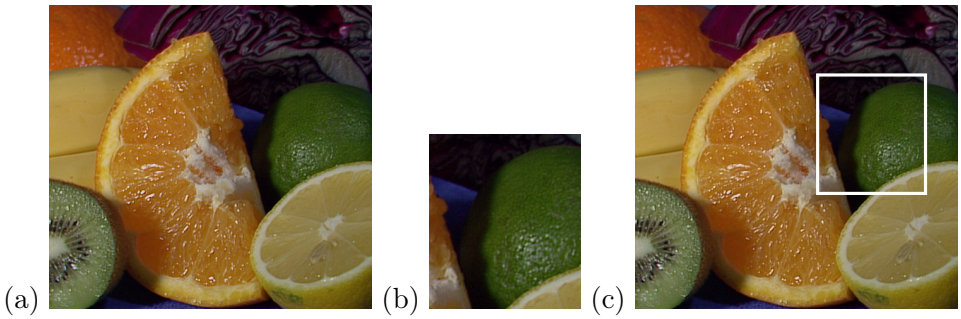


Figure 1.2: Template matching

(a) Source image (b) Template image (c) Match result
Template matching example from [27]

Template matching is a technique for calculating each position of the image under consideration a function that determines the degree of similarity between a template and a portion of the image. Finding a template patch in the target image is a core component in different computer vision applications such as tracking, object detection, image stitching and 3D reconstruction. In real-world scenarios, a bounding box containing a region of interest in the source image undergoes complex deformations such as background changes, rotational changes and partial occlusions. Template matching identifies a region in the target image that best matches with the template. Figure 1.2 taken from [27] shows an example for template matching. It has a source image, template and match result that identifies the region of the source image that best matches with the template.

Template matching approaches are generally classified into two categories: feature-based approaches and template or area-based approaches. The feature-based approach is appropriate when both source and template images have more correspondence with respect to features and control points. Features include curves, points and a surface model that have to be matched. The aim here is to locate the pairwise connection between source and template images using their feature descriptors or spatial relations. Area-based approaches are sometimes called correlation-like methods to involve, establishing similarities between source and template images using correlation techniques such as the sum of squared difference (SSD) and normalized cross-correlation (NCC).

Similarity of template image $t(i,j)$, and test image, $f(i,j)$ using SSD and NCC [27] are defined as follows:

$$SSD(m,n) = \sum_{i,j}[f(i,j) - t(i-m, j-n)]^2 \tag{1.5}$$

$$NCC(m,n) = \frac{\sum_{i,j}[f(i,j) - \bar{f}_{mn}][t(i-m,j-n) - \bar{t}]}{\sqrt{\sum_{i,j}[f(i,j) - \bar{f}_{mn}]^2}\sqrt{\sum_{i,j}[t(i-m,j-n) - \bar{t}]^2}} \tag{1.6}$$

where $\bar{t}$ is the mean of the template image and $\bar{f}_{mn}$ is the mean of the test image $f(i,j)$ in the region centered at (m,n).

Both SSD and NCC work well when the template image is small. The computational cost is really high when the template image is large [74]. Hence various techniques were used for speeding up of correlation computations. A few to mention are integral image representations and fast Fourier transform (FFT). In [38], integral image representation which was first developed in [14] was used for texture mapping in order to quickly compute the denominator in equation 1.6. The numerator in equation 1.6 can be computed by fast Fourier transform (FFT).

Numerous applications of template matching techniques include object detection, tracking, face detection, medical imaging and image stitching. Viola and Jones [75] used the integral image representation for fast face detection. They proposed a framework to provide viable object detection rates in real-time terms. Jurie and Dhome [33] used fast template matching for tracking. They proposed an efficient, robust template matching algorithm that can track targets in real-time. Uenohara and Kanade [73] matched a large set of templates with the test image, and they represented the template set in a PCA subspace. An FFT was utilized to speed up the correlation. [27] described a patch-based approach for rapid image correlation or template matching by representing the template image as an ensemble of patches. They used rectangle filters on image patches for fast filtering based on integral image representation.

1.4 Introduction to Keypoint matching

The concept, of detecting interest points also called key points and matching these key points with reference images are two important problems in computer vision. The main idea of detecting key points is instead of looking at the image as a whole, some special points in the image are extracted and local analysis is performed on them. This technique works well as long as there is a sufficient number of such points. These points are stable and distinguishable and can be localized accurately. An ideal key point detection technique is robust to image transformations such as illumination, rotation, scale, noise and affine transformations. Some of the most commonly used feature detectors nowadays are scale invariant feature transform (SIFT), speed up robust feature (SURF), binary robust independent elementary features (BRIEF) and features from accelerated segment test (FAST).

Figure 1.3: SIFT key point detection

An example from OpenCV tutorials [47] showing SIFT key points

SIFT is a feature detector developed in 2004 [40]. The SIFT algorithm has 4 basic steps. First, a scale-space extrema is estimated using Laplacian of Gaussian with Difference of Gaussian (DoG). Next, localization is performed where the key point candidates are localized and refined by eliminating the low contrast points. Thirdly, a key point orientation assignment is done based on a local image gradient. Finally, a descriptor generator is created to compute the local image descriptor for each key point based on the image gradient magnitude and orientation. Even though SIFT performs really well on key feature detection, it requires large computational complexity which is a major drawback, especially for real-time applications. Figure 1.3 taken from OpenCV tutorials [47] shows the key points detected using SIFT feature detector.

SURF key point detection [3] published is an approximation of SIFT and is a speeded-up version of SIFT. It performs faster than SIFT without reducing the quality of the detected points. SURF approximates the DoG with box filters for finding scale-space extrema. In order to find the points of interest, SURF uses a BLOB detector which is based on the Hessian matrix. For orientation assignments, wavelet responses are used in both horizontal and vertical directions by applying adequate Gaussian weights. Wavelet responses are utilized to feature descriptions. A region around the neighbourhood of a key point is selected and divided into subregions. For each subregion, wavelet responses are taken and represented to get a SURF feature descriptor. Laplacian signs, which were already computed during

detection, can be used to distinguish bright blobs from dark backgrounds from the reverse situation. In short, SURF adds a lot of features to improve the speed in every step. The analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. BRIEF is another alternative for SIFT which requires less complexity than SIFT with almost similar matching performance [13].

SIFT, SURF and BRIEF are really good. But due to their computational complexity, they are not fast enough for real-time applications. SLAM (Simultaneous Localization and Mapping) mobile robot which has limited computational resources is one best example where feature detection has to be fast and robust. In 2006, FAST [61] feature detection was proposed as a solution to detect key features in real time. In this algorithm, a machine learning approach is used which is adaptive in processing. Here, only a few points inside the range are identified and processed whereas the points that fall outside the scope of interest are rejected.

Figure 1.4: Brute-Force key point matching

An example from OpenCV tutorials [47] showing ORB key point matching using Brute-Force matcher

Once the interesting key points are detected and descriptors are computed from above-described feature detection algorithms in two images, these descriptors can be used for key point matching using different matching methods. Various descriptor matching methods exist to identify the best matching descriptors. Few among these are Brute-Force matcher and FLANN based matcher.

In the Brute-Force matcher method, the descriptor of one key point in the template is matched with all other key point descriptors from the source image using some distance calculation. Normally L2 norm distance is used as a distance measure for SURF and SIFT keypoint descriptors. For binary string based descriptors like ORB (Oriented FAST and Rotated BRIEF) and BRIEF, norm hamming distance is used.

Figure 1.4 shows an example of key point matching using Brute-Force matcher taken from OpenCV tutorials [47]. ORB descriptors are used in this matching example. ORB [62] is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance.

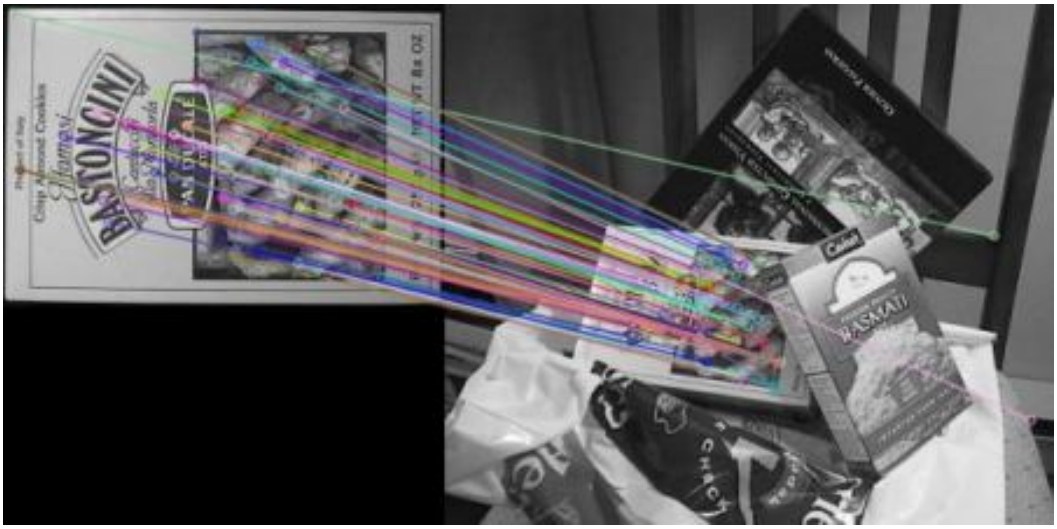


Figure 1.5: FLANN key point matching

An example from OpenCV tutorials [47] showing SIFT key point matching using FLANN matcher

The other method which can be used to match key points is FLANN. FLANN (Fast Library for Approximate Nearest Neighbors) [50, 49, 48] is a collection of algorithms optimized for fast nearest neighbor search in high-dimensional features and large datasets. It works faster than Brute-Force matcher for large datasets. FLANN is part of OpenCV and is one of the most popular libraries for nearest neighbour matching. FLANN uses a randomized k-d forest and priority search k-means tree to find the best matching key point from the source for each key point in the template. Figure 1.5 shows an example of key point matching taken from OpenCV tutorials [47] where SIFT key points detected on source image is matched to the reference image using FLANN.

1.5 Outline

The remainder of the work is structured as follows: In the next chapter, a review of related work regarding random forests and its role in the context of computer vision domain will be presented. We also discuss related work regarding template and key point matching. Chapter 3 will introduce and explain the proposed approach in detail. In Chapter 4, implementation is discussed. In Chapter 5, the experimental setup, evaluation methods and results are discussed. In the last chapter, conclusions and future work are presented.

Chapter 2

Related Work

The first section in this chapter presents work regarding template matching. In the second section, works that use random forests for computer vision tasks are listed. In the last section works that use the random forest for similarity computations and works that discuss different methods to compute a similarity score based on random forests are presented.

2.1 Template matching in Computer Vision

Template matching algorithms are based on the similarity measure between the template and a matching window in the target image. Many different similarity measures are employed for this purpose. As discussed in section 1.3, few popular similarity measures widely used are the Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD) and Normalized Cross-Correlation (NCC), mostly due to their computational efficiency [52]. Many variants of these measures were proposed which deals with noise and illumination changes [28, 20]. In [28], a fast pattern matching scheme called Matching by Tone Mapping (MTM) was introduced which allows matching under non-linear tone mappings. Similarly, in [20], an efficient and noise robust template matching method based on asymmetric correlation (ASC) was presented. This function is invariant to changes in affine illumination and robust to extreme noise. Other families of similarity measures comprise of robust error functions like M-estimators [70] or Hamming-based distance [68], which are less affected by additive noise and outliers than cross-correlation related methods.

As discussed in section 1.3, many works were published to speed up SSD and NCC calculations for template matching. In [12], an algorithm for fast calculation of the normalized cross-correlation (NCC) and its application to the problem of template matching was presented. The algorithm represents the template and the reference image as a sum of rectangular basis functions used for correlation computation. The work showed that the algorithm can outperform Fourier transform based implementations of the normalized cross-correlation algorithm based on approximations.

In [44], a novel edge gradient-based template matching method for object detection was proposed. The proposed template matching based on edge features utilizes the orientation and intensity of edges in both the template and query images directly for similarity measures. In [32], a novel method for template matching named THTM - a template matching algorithm based on HOG (histogram of the gradient)

and two-stage matching was proposed. The algorithm relies on the fast construction of HOG and the two-stage matching which jointly lead to a highly accurate approach.

In [45], a novel, a fast, resolution-independent silhouette area-based similarity measure was developed for template matching to compare model templates and segmented input images. The developed method approximated the silhouette area using a small set of axis-aligned rectangles that yielded a very memory-efficient representation of templates. In addition to this, the method utilized integral image representation for input images. Exploiting these techniques together helped to achieve a significant increase in template matching speed.

In [27], a patch-based approach for rapid image correlation or template matching was proposed. The template image is decomposed into a set of non-overlapping patches. The best matching window on the source image can be identified by comparing each patch in the template image with patches extracted from the reference window. Rectangle filters are used as features of patches to compare the similarity between patches. More details about the rectangle filters and the similarity method used in this paper will be presented in the next chapter. We choose this work to compare our work which uses random forest similarity measure to compute the similarity between image patches. Our main idea is to show whether random forests can be used to determine the similarity between data points especially similarity between image patches and test their behaviour under various testing conditions such as rotational changes, occlusions, blur and noise conditions.

Similar to [27], many other works exist which uses image patches to find the best matching window in the source image that matches the template. In [79], a CNN-based approach to compare image patches for computing costs in small baseline stereo problem was proposed and shown the best performance in KITTI dataset. In [53], a method which improves the state of the art image patch matching using a sparse-overcomplete representation of an image was presented. This is based on the idea that sparse-overcomplete representation of an image posses statistical properties of natural visual scenes that can be utilized for patch matching. The method in this paper initially encodes image patch details by encoding the patch and then subsequently uses this sparse representation as input to a neural network to compare image patches.

The work in [78] showed how to learn a general similarity function for patches directly from raw image pixels, which is of basic importance for many computer vision problems. The similarity function was encoded in the form of a CNN model that was trained to account for a wide variety of changes in image appearance. Several neural network architectures that were specifically adapted to this task were studied to identify the best CNN model that exhibits extremely good performance, significantly outperforming the state-of-the-art on various problems and benchmarking datasets.

The work in [15] introduced a new measure, Best-Buddies Similarity(BBS), for template matching in the wild. BBS is based on counting the number of points in source and target sets, where each point is the nearest neighbour to the other. BBS

is robust against complex geometric deformations, and high levels of outliers arose from background clutter and occlusion.

2.2 Random forests in Computer Vision

The work [2] in 1997 used an ensemble of trees for optical character recognition. This was one of the earliest applications of an ensemble of decision trees in computer vision. In 2005, ensembles of extremely randomized decision trees were used for image classification in [41]. In 2006, random forests were used for key point recognition in [37]. In this work, decision trees were generated by using pixel comparisons as node tests. Node tests were randomly generated on the fly to bring randomization into tree generation. In our work, we use a similar approach to generate random forests which would then be used to derive similarity measures.

In [76], random forests were built to distinguish between patches from different parts of the object as well as from the background. They treated this as a pure classification problem by splitting the object into a predefined number of parts treated as independent classes and used random forests to solve this classification task. In [5], random forests and ferns were used for image classification. Classifying images by the object categories when the image contains a large number of object categories was the problem explored in this work. Random forests and random ferns were used as a multi-way classifier, which made training and testing phase easier when compared to other multi-way classifiers such as SVM. In [24], random forests were used for the detection of instances of an object class, such as cars or pedestrians, in natural images. A class-specific hough forest, a random forest that mapped the image patch appearance to the probabilistic vote about the possible location of the object centroid was trained in this work.

In [69], random forests were used to quickly and accurately predict 3D positions of body joints from a single depth image. Realistic synthetic depth images of humans in varying shapes and sizes at different poses sampled from a large motion capture database were generated and a deep randomized decision forest classifier on these sampled images to estimate body parts invariant to pose, body shape and clothing was trained. In [77], an object detection using generalized Hough transform in combination with randomized decision trees was presented. The work proposed an interactive object annotation method that incrementally trains an object detector while the user provides annotations. In [59], methods that address different aspects of image processing at a scale where thousands of object categories need to be recognized in millions of images were introduced. All the methods were prepared within the framework of random forests, which is particularly suitable for such large-scale settings.

In 2008, random forests were used for semantic image segmentation in [64]. The work showed the ability of random forests to combine multiple features to increase segmentation performance when HOG features, colour, textons and filterbanks were used simultaneously. In [29], a novel learning-based single image super-resolution

method using the random forest was proposed. The random forest was used to classify the training low resolution-high resolution (LR-HR) patch pairs into a number of classes. In 2017, the same authors of [29] proposed a set of decision tree strategies for fast and high-quality single image super-resolution (SR) in their work [30]. By inspiration from random forests which combines regression models from an ensemble of decision trees, they proposed super-resolution hierarchical decision trees (SRHDT) to exploit fast image patch classification. Another work on image super-resolution was [39]. The authors have proposed a novel feature-augmented random forest for image super-resolution, where conventional gradient-based features were augmented with gradient magnitudes and different feature recipes were formulated on different stages in a random forest.

Random forests are widely used in face detection and recognition systems. In [4], a system for detecting and recognizing faces in images in real-time was presented in 2008. The work used random forests for both face detection and recognition to realize real-time performance on modest computing hardware. In [63] in 2012, a fast face recognition system was proposed where HOG descriptor (Histograms of Oriented Gradients) with random forest classifier was used to achieve a higher face recognition rate with lower computational cost. Recently, the work [31] used random forests to identify the driver's facial expression. In this work, a fast facial expression recognition algorithm for monitoring a driver's emotions that are capable of operating in low specifications devices installed in vehicles was presented. They used hierarchical Weighted Random forest (WRF) classifier for this purpose.

In [6], a novel method to automatically recognize pictured dishes was proposed. The work introduced a method to mine discriminative parts using random forests, which enabled them to mine for parts simultaneously for all classes used and to share knowledge among them. Another application of random forests is in Image colourization, which is the process of assigning colour values to grayscale images. In [43], a fully automatic method for image colourization based on random forests was proposed. Random forests were used for the task of estimating the proper colour values when presented with a grayscale image patch. Initially, a random forest was used to analyze the local structure of a grayscale image patch and was then used to employ a spatial colour prior model learned from training data.

Random forests are also used for pedestrian detection. In [71], a novel Random decision forests (RDFs) to simultaneously classify pedestrians and their directions were proposed. In [42], a pedestrian detection method to combine multiple local experts using random forests was proposed. The work used the random forest framework to improve robustness and generalization capabilities of the pedestrian detection process. Few other areas where random forests are widely used are codebook generation [46], edge detection [17, 18], traffic sign recognition [21, 26] and medical imaging [25, 35, 36].

2.3 Similarity computations using Random forest

Qi et al. [56] used the random forest to identify similarity for protein-protein interaction prediction. They proposed a method to compute similarities for the task of classifying pairs of proteins as interacting or not. They faced the following issues when they wanted to identify similarity between protein-protein interactions : (i) the data they used was noisy and contained many missing values, (ii) some of the data sources used were categorical and some were continuous and, (iii) some data sources were given more weights compared to others. Random forests are good at handling missing values, categorical data and data sources with variable weights, due to the decision trees and randomization strategy within them. Hence they used random forests to overcome these issues and to compute the similarity matrix. The similarity between two pairs of protein pairs X_1 and X_2 was computed in the following way. For each of the two pairs, they propagated their values down all the trees within the random forest. Next, the terminal node position for each pair in each of the trees was recorded. Let $Z_1 = (Z_{11}, ..., Z_{1K})$ be these tree node positions for X_1 and similarly Z_2 was defined. Then the similarity between pair X_1 and X_2 was set to $S(X_1, X_2) = (1/K) \sum_{i=1}^{K} I(Z_{1i} == Z_{1K})$, where I is the indicator function and K is the number of decision trees. This similarity measure gave a similarity matrix for all protein pairs used for testing. Then this matrix was used to rank samples by a weighted k-nearest-neighbour approach. The proposed method claimed to work well for the protein-protein interaction prediction in yeast. This work used the random forest to identify similarity matrix or proximity matrix after classification to determine interactions between protein pairs. Similar to [56], in [54] similarity matrix computed using random forest was used to learn similarities in RNA sequencing data. But in our work, we use don't compute the similarity using proximity matrix but instead use the decision tree paths to identify the similarity between image patches.

In [51], an algorithm that learns a similarity measure for comparing never seen objects was proposed. Three steps were used for computing the similarity measure. (a) From a pair of images, several random pairs of patches which forms a best-normalized cross-correlation match are sampled. (b) A binary vector representing features of image pairs is derived by quantizing the space of image patch pairs. Each patch pair is assigned to several clusters via an ensemble of extremely randomized binary decision trees. (c) The cluster memberships are combined to make a global decision about the pair of images. The similarity measure derived is a simple linear combination of the binary feature vector indicating cluster membership. The decision trees are not used as classifiers, but as quantizers by only considering which patch pairs reaches the leaves and discarding the predictions from the leaf nodes. In our work, instead of using random forest decision trees as quantizers to generate feature vectors, we use them to calculate the similarity between patches by considering the decision path of the patch pairs.

In random forests based data mining, including data clustering, visualization, outlier detection, substitution of missing values, and finding mislabeled data sam-

ples, a data proximity matrix is an important information source. In [22], a novel approach to estimate proximity was proposed. The approach is based on measuring the distance between two terminal nodes of a decision tree occupied by observations X_i and X_j. The proximity of observations X_i and X_j is given by $p_{ij} = (1/K)\sum_k^K 1/(e^{w.g_{ijk}})$ where k runs over the K trees, for which both X_i and X_j are among the OOB samples, w is a parameter, and g is the number of tree branches between the two terminal nodes occupied by X_i and X_j. If X_i and X_j occupy the same terminal node, then $g = 0$ and p_{ij} will be increased by one as in the original way to assess data proximity. The parameter w controls the influence of the distance between two terminal nodes occupied by the observations on the proximity values. To assess the quality of data proximity estimate, the proximity matrix can be used as a kernel matrix in a support vector machine (SVM) classifier. An SVM is trained using the proximity matrix as a kernel and the generalization error of the SVM is evaluated. The lower the error, the higher is the quality of the matrix. In the same manner, the optimal value of the parameter w can be found by varying w and again using the proximity matrix in SVM. This work used a different approach using random forest and SVM to calculate the proximity matrix when compared to [56].

Puggini et al. [55] demonstrated the use of random forest similarity distance measurement as a metric for anomaly detection within a semiconductor manufacturing process. They proposed that a random forest similarity distance can be used in an unsupervised way to predict faults within a semiconductor manufacturing process. For a given wafer dataset X_0, they labelled all samples in X_0 as class 0. They created a synthetic second class X_1 of the same size by sampling at random from the univariate distributions of the original data. They labelled all samples in this synthetic dataset as class 1. They trained random forest on this two-class dataset. Then they computed proximity between samples as follows : (i) they ran all test samples through the decision trees, (ii) they estimated the proximity matrix P_r. This was calculated by increasing the proximity by one of the cases i and j if they end up in the same terminal node of the tree, and (3) finally they normalised the estimated proximity by dividing by the total number of trees and then replacing each value with its square. They computed the average proximity of a sample z to a class k as $P_{r_k}(z) = (1/|k|)\sum_{j:class(j)=k} P_r(j, z)$. They then used this proximity to calculate the distance of a sample from a class k and eventually determined whether it was an outlier. In this work, random forests were used to compute proximity /similarity between samples and eventually the proximity matrix was used for anomaly detection.

In [80], random forest proximity matrix was used to measure variable importance. The difference of proximity matrices was used to compute the variable importance which was used for gene selection on DNA microarray data. In this work, proximity was taken as a special kernel measure, and the differences of samples proximity instead of out-of-bag error were used as the criterion to select the information genes. The work showed that compared with the original variable importance analysis of random forest, this method was more sensitive to information gene and yielded small

sets of genes while preserving predictive accuracy. For n training pairs $S = (x_i, y_i)$, where $i = 1, ..., n$, and x_i is a feature vector representing ith sample, and y_i is the class label of x_i, and for a binary problem $y_i \in -1, 1$, and for a $k - class(k > 2)$ problem $y_i \in 1, 2, ..., k$, the variable importance of proximity difference is computed as follows. Random forest is trained and proximity matrix $P = p_{ij}, i, j = 1, .., n$ of all data is computed similar to [56]. The proximity ratio between inner-class and inter-class is computed : $C = P_s/P_d$, where $P_s = \sum_{i,j=1}^{l} p_{ij}(if y_i = y_j)$ and $P_d = \sum_{i,j=1}^{l} p_{ij}(if y_i \neq y_j)$. Now some variables are permuted and proximity matrix $C_i, i = 1, ..., l$ is recomputed again. The final measure is the difference between these : $C - C_i, i = 1, ..., l$. Proximity matrix can be taken as a special measure that reflects the distribution of samples in a "proximity space" [80]. A variable which contributes to the sample's distribution is noised up, then the inner-class proximity will decrease while the inter-class proximity will increase, so the ratio C_i will degrade noticeably. On the other hand, if a variable is not relevant or redundant, noising it should have little effect on the performance. In our work, the features are computed on the fly using different node tests. Hence we don't compute variable importance in our work. But this paper [80] is worth mentioning here since it is a very good example which shows how random forest proximities can be computed and utilized for determining variable importance.

In the above works, we saw the applications of random forest proximity matrix. The proximity matrix obtained from the random forest can be transformed into a random forest dissimilarity matrix. If SIM is a random forest similarity matrix, then random forest dissimilarity matrix is defined as $DISSIM_{ij} = (\sqrt{1 - SIM_{ij}})$ [66]. The random forest dissimilarity has been successfully used in several unsupervised learning tasks involving genomic data: Breiman and Cutler [11] applied random forest clustering to DNA microarray data and Allen et al. [1] applied it to genomic sequence data. Shi et al.[67] successfully used random forest unsupervised learning for tumour class discovery based on immunohistochemical tumour marker expression. The random forest dissimilarity matrix was used as input for partitioning around medoid (PAM) clustering to separate patients into two groups. Another use of the proximity matrix is for missing data imputation. Breiman and Cutler [11] illustrated data imputation by weighting the frequency of the non-missing values with the proximity values.

From all the works mentioned above, we saw how random forest can be utilized to compute the similarity between data points. We also saw various techniques to compute proximities. But none of the above works computes similarity using decision tree paths in the random forest. In our work, we use this approach to compute the similarity between the image patches. We believe that computing similarity using the decision paths prevents the loss of information when computing similarity, and it helps to compute similarity accurately. In the next chapter, we discuss the technique to compute similarity using the random forest.

Chapter 3

Method

The main aim of this paper is to use random forests to identify the similarity between image patches. We test the behaviour of random forest similarity measure using two applications: (i) we replace the rectangle filter similarity measure between image patches described in [27] with the random forest similarity measure, and perform template matching to evaluate its performance, and (ii) we use random forest similarity distance instead of the SURF descriptor distance to evaluate template matching using SURF key point matching. In this chapter, we describe the patch similarity, rectangle filters and template matching technique used in [27]. We describe random forest generation and redefine the patch similarity used in [27] using random forest similarity measure. The same random forest similarity measure can also be used to perform key point matching, which we would describe in the last section of this chapter.

3.1 Patch-based correlation

The ability to identify similarities between data points has been the basis of many computer vision problems such as pattern recognition, object recognition, texture and scene categorization. In many computer vision applications, it is often necessary to define a similarity measure between objects or images. One classic method used to identify the similarities between images and objects is template matching. Template matching is a technique for finding areas of an image that match (are similar) to a template image (patch). Template matching involves translation of the template to every possible position in the test image and evaluation of matching score between the template and the image at that position [74].

In many computer vision applications, images can be analyzed at the patch level rather than at the individual pixel level. Image patches contain contextual information and have many advantages in terms of computation and generalization. For example, patch-based methods produce better results and are much faster than pixel-based methods for texture synthesis [19].

But some applications of computer vision represent a complete object using a large template, especially for object recognition. In many instances, templates of face contain the whole face to be used in face detection [75] or face recognition [72]. Though large templates are useful for representing the whole object, they are sensitive to local variations and partial occlusion. In order to use large templates, an image can be represented by an ensemble of patches. In template matching or

image correlation, there are three levels of approaches: pixel level, patch level, and complete template, as shown in figure 3.1.

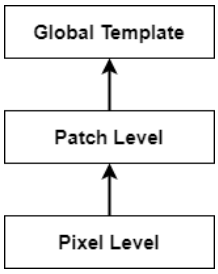


Figure 3.1: Image correlation

Three possible levels of methods for image correlation [27]

SSD and NCC measures actually work at the pixel level and use the whole template for matching as discussed in section 1.3. Hence standard SSD and NCC are not robust to variations and are slow. As discussed in section 1.3 many methods like [38] speed up NCC by using integral images. Few other methods like [73, 60] are sensitive to occlusions and local variations as they used complete templates only.

Figure 3.2: Template image

A template image divided into non-overlapping patches [27]

Hence motivated by the advantages of representing an image by a set of patches, a patch-based method for image correlation was proposed in [27]. In this work, a new approach for template matching using an ensemble of patches instead of a single, large template was proposed. Along with this, a new method for feature selection using rectangle filters given a single image was developed and issues related to robustness, scale change and partial occlusion were addressed. The rest of this section describes the patch similarity defined in [27].

The template to be identified on the source image is first decomposed into over-

lapping or nonoverlapping patches. In all the experiments, the template is decomposed into nonoverlapping patches, as showed in figure 3.2.

Now, the template image can be represented by $t = [p_1^t, p_2^t, ..., p_k^t]$ when it is divided into k patches. The similarity between source image $f(i,j)$ and template $t(i,j)$ is defined as follows:

$$D(t, f, m, n) = \sum_{r=1}^{k} ||p_r^t - p_r^f(m,n)|| \tag{3.1}$$

where $p_r^f(m,n)$ is the patch in the source image $f(i,j)$ matching the patch p_r^t in the template, as the template $t(i,j)$ is translated to a position (m,n) in the source image. $||.||$ is the norm, and 1-norm is used in all the experiments.

$D(t,f,m,n)$ is non-negative. The smaller the value of $D(t,f,m,n)$, the more similar the template t and the source image f at position (m,n). The image correlation problem is to find the minimum of the objective function:

$$(m^*, n^*) = arg \min_{m,n} D(t, f, m, n) \tag{3.2}$$

Now, the similarity of patches is measured using rectangle filters extracted from patches.

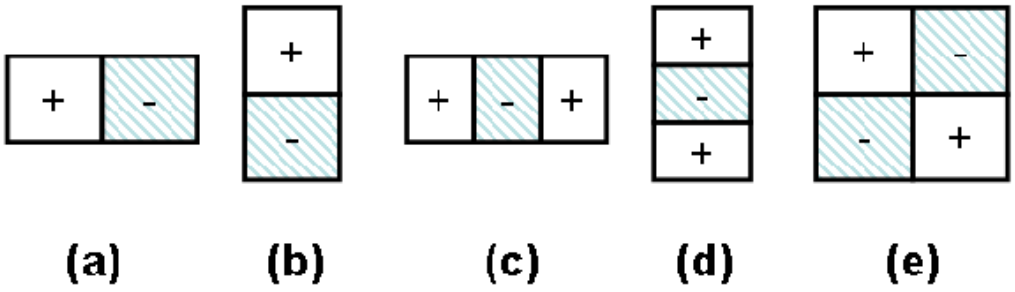


Figure 3.3: Rectangle filters

Rectangle filters applied to each image patch [27]

Instead of directly using the raw pixel intensities on each patch, a set of filters is used on them to extract the features. The advantages of using features over raw pixels for template matching are : (i) compared to pixel based system, the feature-based system is faster in operation [75], and (ii) the image correlation or template matching can be made faster by reducing the number of features which encode some important image structure information.

A variety of filters can be used for filtering in each patch, including Gabor filters and Laplacian-of-Gaussian filters. In [27] rectangle filters are used for extracting features which were originally used by Viola and Jones for rapid face detection [75].

The advantage of using rectangle filters is that they can be evaluated quickly with simple additions without compromising on filter size, based on the integral image representation [14].

Similar to the filters in [75], five different types of rectangle filters are used in [27] as showed in figure 3.3. In this figure, the sum of pixels that lie in dark rectangles (with -) is subtracted from the sum of pixels in light rectangles (with +). Filters (a) and (b) are useful for edge features, (c) and (d) for line features, and (e) for diagonal structures. Instead of applying filters to complete template, filters are applied to each individual patch. This is the only difference between these filters and the filters used in [75]. The similarity equation 3.3 can be rewritten using rectangle filters as:

$$D(t, f, m, n) = \sum_{r=1}^{k} \sum_{l=1}^{L} ||g_l \otimes p_r^t - g_l \otimes p_r^f(m, n)|| \tag{3.3}$$

where g_l are rectangle filters, with $l \in \{1, 2, ..., L\}$ and $\otimes$ is convolution.

The rectangle filters are applied to each patch in the template. The number of features extracted from the template is given by $\#features = \#patches \times \#filters$. For example, if a template is decomposed into 60 patches and 100 filters are applied to each patch, then the total number of features is 6000. This number is smaller than the number of pixels in the template. Thus compared to pixel-based SSD or NCC, patch-based matching is faster. But still, the number of features extracted is too large and slows down the matching. Hence the next idea is to reduce the number of features extracted from the template.

In order to reduce the number of features extracted, in [27], only salient features are selected from the extracted patches and used for matching. Here regions with "salient" image structures are given importance when compared to "flat" regions in the template. But how to choose these salient features? One way would be to detect edges or corners, but it comes with a lot of complexity computations. The other way is to look at the rectangle filter values extracted from the template, compare these filter values and retain only the filter values whose response is maximum. These rectangle filters respond strongly to image structures such as edges, lines, and diagonal structures. Here strong response means large values in the filtered results.

Figure 3.6 shows the magnitude of the filtered values after sorting in descending order. The filter values are extracted from two templates - figure 3.4 (b) and figure 3.5 (b) respectively. Based on observation in figure 3.6, only a small number of features that provide maximum response to the filters can be retained for image correlation, and to make matching much faster.

Since the filter values in figure 3.6 are already in descending order of maximum response, only top $n\%$ of these filter values are retained for matching. This makes matching much faster.

Let $v_{l,r}^t = g_l \otimes p_r^t$ be the filtered value using the filter g_l on patch p_r^t. A small number of features can be selected by using the below equation:

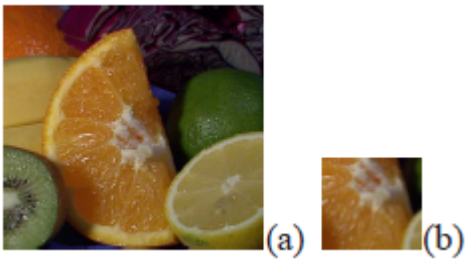


Figure 3.4: Fruit source image and template

A sample fruit source image and template [27]

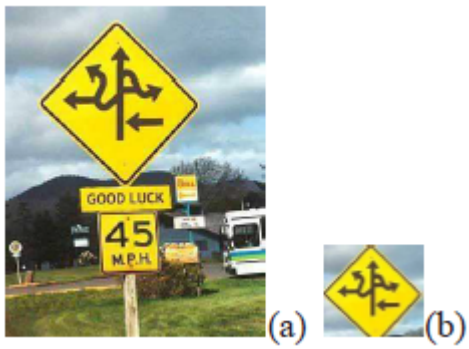


Figure 3.5: Traffic sign source image and template

A sample traffic sign source image and template [27]

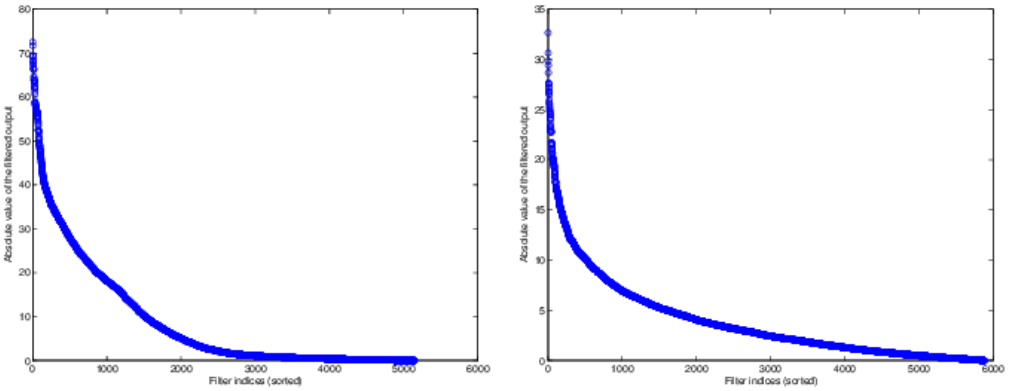


Figure 3.6: Rectangle filter responses from templates

Rectangle filter responses from figures 3.4 (b) and figure 3.5 (b) respectively [27]

$$v^t_{l',r'} > \alpha \times \max_{l,r} v^t_{l,r} \tag{3.4}$$

where α is a constant to control the number of selected features, and thus influences the computation time in matching.

After retaining the filters that provide a maximum response on the template, the same filters are applied to a region of source image whose size is same as template size, and the matching score is calculated using equation 3.3. The sliding window approach is used for template matching. Image patches from a source image with the same size as template are extracted by sliding over the source image pixel by pixel. Now both template and large source image patch are decomposed into non-overlapping smaller patches. Rectangle filters are applied to template patches and filters with maximum response are retained. Now the same filters which returned maximum value for the template are also applied to source patches and image correlation score/ template matching score is computed. The region in the source image that provides the least value for this matching score is the region that best matches with the template as given in equation 3.2.

3.2 Random forest similarity

Random forests are one of the most efficient machine learning methods whose usage can be typically found in many different applications. In this specific section, we properly describe how random forests can be beneficially used to identify the similarity between patches.

Random forests can be used to solve classification tasks or regression tasks. In this paper instead of utilizing random forests to predict the class to which the data sample belongs to or to predict the price or a continuous number for any regression task, we are more interested in the decision path taken by data sample to reach the decision tree leaf nodes. We use these decision paths to identify the similarity between image patches. The general idea behind this is as follows - when data samples from an object under consideration are run through the random forest, they follow a specific path through the decision trees. When two samples are more similar, the more likely their paths will overlap. This overlapping of decision paths can, therefore, be used to measure the similarity between data samples. We use the same idea in the context of image patches.

Let p_a and p_b are two image patches of the same size. Let T be the number of decision trees that the random forest is trained on. The similarity between image patches using random forest decision paths is given by:

$$S_{a,b} = \frac{1}{T} \sum_{i=1}^{T} \frac{overlap(n_a, n_b)}{max(n_a, n_b)} \tag{3.5}$$

where n_a and n_b are the decision nodes visited by image patches p_a and p_b in a given decision tree. $overlap(n_a, n_b)$ gives the overlapping nodes or total initial

common nodes visited by p_a and p_b. $max(n_a, n_b)$ gives the maximum of the total nodes visited by p_a and p_b. The average of the ratio of these quantities over all the decision trees in the random forest gives the similarity score $S_{a,b}$ between patches p_a and p_b. The higher the similariy score $S_{a,b}$, the higher the similarity between image patches. $S_{a,b} = 1$ is an ideal case where p_b visits only the same set of decision nodes visited by p_a in the same order. Figure 3.7 shows an illustration of calculating the similarity measure between image patches.

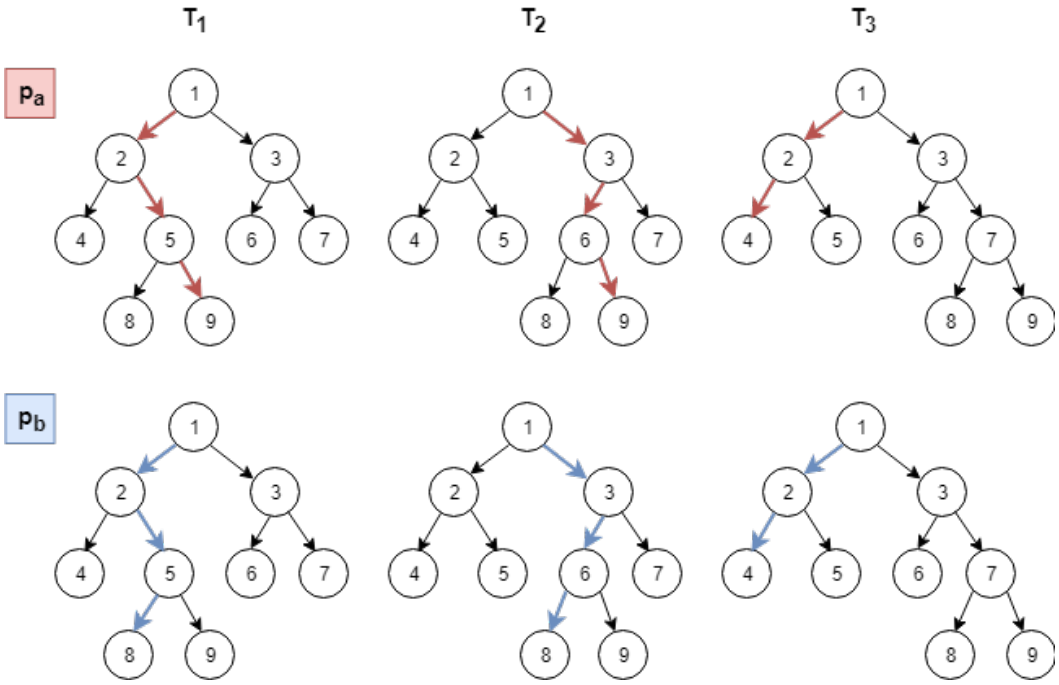


Figure 3.7: Random forest similariy

p_a and p_b are image patches. T_1, T_2 and T_3 are decision trees. The decision nodes visited by patches are highlighted with the respective patch color. In T_1 only first 2 nodes overlap. Hence $overlap(n_a, n_b)/max(n_a, n_b) = 2/max(3, 3) = 2/3$. Similary, from T_2 the ratio is 2/3. From T_3 the ratio is $2/2 = 1$. Therefore $S_{a,b} = (2/3 + 2/3 + 1)/3 = 0.77$.

We can also compare the image patches by comparing the leaf nodes they visit. This approach is faster and easier compared to comparing the decision paths of image patches. But this approach gives only a binary result - true or false indicating whether the image patches are completely identical or not. It doesn't take into consideration the percentage of similarity between the image patches. Even if the image patches are 80-90% similar, using the leaf node visiting approach, we get a complete mismatch between the image patches. Hence by taking the decision paths of image patches into consideration, we retain this 80-90% similarity.

We now know how to compute the similarity between image patches using random forests. The next idea is to replace rectangle filter similarity defined in the previous section with this random forest similarity and compare the performances

of these two similarities in the context of template matching.

The similarity between image patches using rectangle filters is given in equation 3.3. Using the random forest similarity defined in equation 3.5, we can now rewrite the equation 3.3 using this similarity as follows:

$$D(t, f, m, n) = \sum_{r=1}^{k} S(p_r^t, p_r^f(m, n)) \tag{3.6}$$

Using equation 3.5 we can rewrite the term $S(p_r^t, p_r^f(m, n))$ as follows:

$$S(p_r^t, p_r^f(m, n)) = \frac{1}{T} \sum_{i=1}^{T} \frac{overlap(n_r^t, n_r^f(m, n))}{max(n_r^t, n_r^f(m, n))} \tag{3.7}$$

where T is the number of decision trees in the random forest and n is the decision nodes visited by image patches. In this equation similarity score between two patches are computed by dropping patches through T decision trees. Each decision tree gives a single similarity score. The overall similarity score of a single patch pair is the average of the similarity scores obtained by T decision trees. By combining the similarity scores of patch pairs, we get the overall similarity between the template and a region of the source image with the same size as the template as shown in equation 3.6.

Equation 3.6 always returns a positive number. When the template $t(i, j)$ is translated to a position (m, n) in the source image, then the best matching position returns a maximum value for $D(t, f, m, n)$. So now the image correlation problem is to find the maximum of the objective function:

$$(m^*, n^*) = arg \max_{m,n} D(t, f, m, n) \tag{3.8}$$

Similar to template matching using rectangle filters as explained in the previous section, now template matching can be performed the same way but using this random forest similarity. Again the sliding window approach is used for this approach. Image patches whose size is same as template are extracted by sliding over the source image pixel by pixel. The large image patch and template and decomposed into non-overlapping small patches. For each patch pair from template and source, the random forest similarity score is computed using equation 3.6. The region of the source image that returns the maximum similarity score is the region in the source that best matches the template as given in equation 3.2. The steps to perform template matching using random forest similarity score is shown in figure 3.8.

Now we know how to determine the similarity between image patches using random forests and perform template matching. The next problems to address are as follows - on what data should we train random forest on? What tasks should we solve using this random forest - a classification task or a regression task? How can we use this random forest to solve the template matching problem?

In order to use the image patch as input to decision trees in the random forest, the random forest has to be trained to perform certain proxy tasks on image patches.

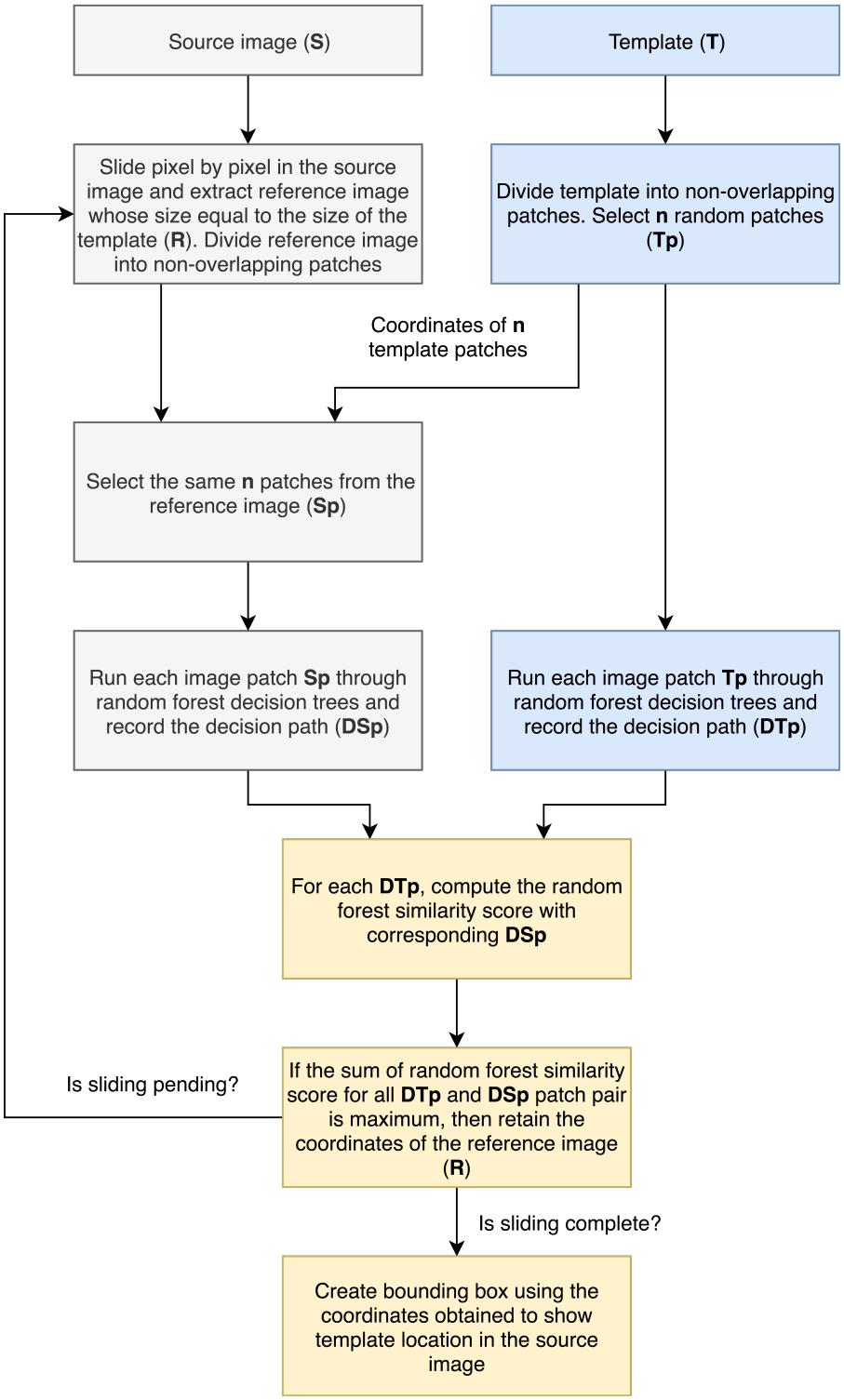


Figure 3.8: Random forest template matching

Steps to perform template matching from using random forest similarity score

The proxy task can be any machine learning task such as classification, regression or clustering. Since we only use the decision paths of image patches to determine the similarity, the proxy task at hand can be optimized for any of these machine learning tasks.

In this work, we decided and trained random forest on an image classification task. A data set of thousands of random image patches (whose size is same as the patches we use as input to the random forest to perform template matching) where each single image patch represents an image class was chosen to train the random forest. The random forest was then trained to optimize this image classification task.

Once random forest on proxy task is ready, the next step is to run image patches through the decision trees in them. If the image patches are similar, then they take a similar path in the decision trees. The overlapping of decision paths can then be used as the similarity measure between image patches. Hence image patches of the same size as the image patches used to train random forest are extracted from both template and source images. Then each patch from template and source image is run through the random forest, and their decision paths are recorded. The similarity score between patches can then be computed using equation 3.5 and template matching can be performed using equation 3.6.

3.3 Template matching from Key point matching

In section 1.4, we saw a brief overview of various key point detection algorithms. The most commonly used of these are SURF, SIFT and FAST. In this work, we perform template matching by matching SURF key points in template and source image.

Initially, SURF key points are identified on both template and source images. For each key point, its associated SURF descriptor is extracted. Then for each key point from the template, the best matching key point from the source image is identified using some distance calculation. Then using these matched key points, the region on the source image that best matches the template is identified.

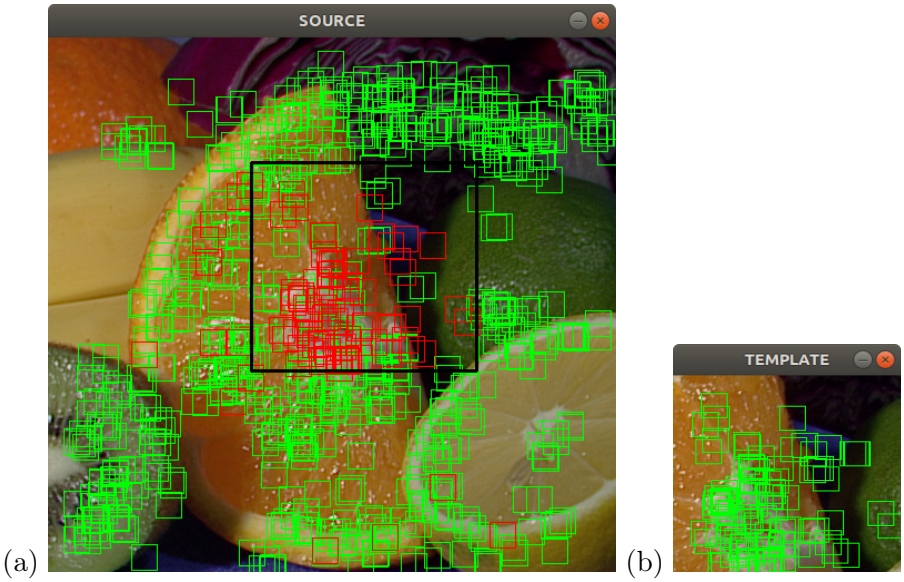
In order to match the key points in source and template image, we use FLANN key point matcher. FLANN matcher is one of the most popular libraries for nearest neighbour matching and is part of OpenCV. The more details about the FLANN matcher is discussed in section 1.4. FLANN matcher uses SURF descriptors to perform key point matching. The difference in SURF descriptor distance is used to identify the best matching key point pairs in source and template images.

Using these matched key point pairs, a perspective transformation between the template and the source image can be estimated. This is called homography estimation. A Homography is a transformation (a 3x3 matrix) that maps the points in one image to the corresponding points in the other image. Various algorithms are available to estimate homography. We use RANSAC (RAndom SAmple Consensus) [23] based approach for estimating homography. After computing the homography

matrix, this matrix can be used to identify coordinates of the region on the source image that best matches the template.

The steps used in RANSAC to estimate homography is as follows - (i) four feature pairs (key point pairs) are selected at random, (ii) homography H is computed using these feature pairs, (iii) inliers are computed and only those inliers which doesn't exceed certain threshold are retained, and (iv) using all of the inliers least-squares of homography H is recomputed. These steps are executed in the loop to estimate the best homography H between source and template images.

After computing homography, using this homography matrix H, the coordinates of the region of the template on source image can be estimated. OpenCV [47] has these inbuilt functions findHomography and perspectiveTransform to detect homography matrix H, and perform perspective matrix transformation of vectors respectively. The function perspectiveTransform takes two arguments - homography matrix H, coordinates of template ((0,0),(0,template-width),(template-height,0),(template-height,template-width)) and returns the coordinates of region on source image corresponding to template.



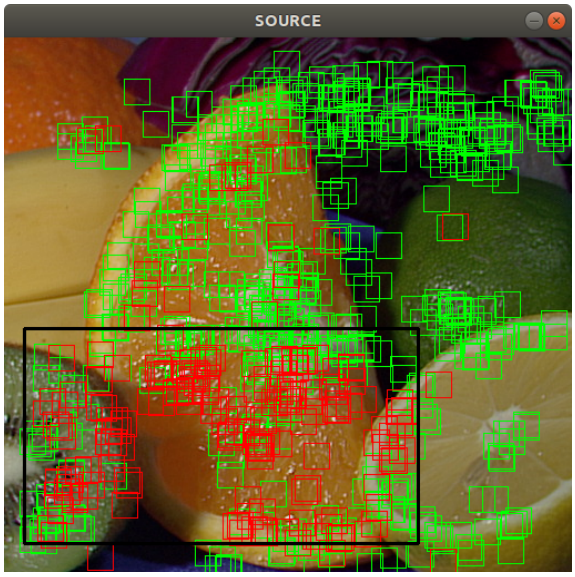
(a) (b)

Figure 3.9: Template matching - example 1

(a) Source image (b) Template
Template matching from key point matching using random forest distance on the same fruit image used in [27]

After performing above mentioned template matching steps using SURF descriptor distance, we replaced SURF descriptor distance with random forest distance and re-performed template matching to test its accuracy compared to SURF descriptor distance.

The random forest can also be used to identify the similarity between key points.

(a) 

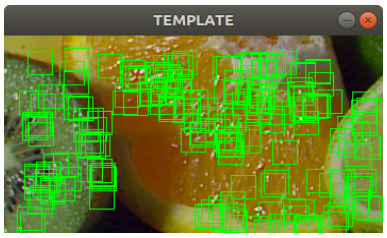
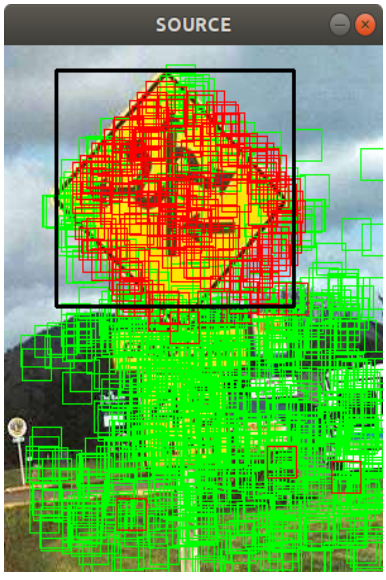
(b) 

Figure 3.10: Template matching - example 2

(a) Source image (b) Template
Template matching from key point matching using random forest distance on the same fruit image used in [27] with larger template size

(a) 

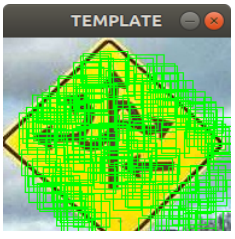
(b) 

Figure 3.11: Template matching - example 3

(a) Source image (b) Template
Template matching from key point matching using random forest distance on the same traffic sign image used in [27]

Instead of using SURF descriptors, we can use random forest similarity measure to match key points between template and source image. As explained in the previous section, a random forest is trained on a proxy classification task to classify image patches. Key patches centred at SURF key points are extracted from both template and source image. For each key patch from the template, the random forest similarity score is calculated for all key patches from source image using equation 3.5. The best matching key patch pair would be the one that returns the maximum value for this equation. This way the best matching key patch from the source image for all key patches in the template is calculated. Since these key patches are nothing but patches centred at key points, again RANSAC based approach mentioned earlier is used to identify the region on the source image that best matches the template.

Figures 3.9, 3.10 and 3.11 show examples of performing template matching from key point matching using random forest distance. The green rectangles in source and template images are image patches centred by SURF key points. The number of SURF key points detected can be controlled using hessian constant used for detecting key points. If the key points are more, then accuracy will be high. But it, in turn, affects the performance since each individual key points in the source has to be matched with each individual key points in the template to find the best match. Hence hessian constant is wisely chosen to balance accuracy and performance. For the image patches in the template, the best matching source patches are identified using random forest similarity. These are highlighted in the source image with red rectangles. The SURF key points in the centre of these red rectangles can be used to identify the region in the source image that best matches the template using RANSAC based approach. Figures 3.9, 3.10 and 3.11 has a black rectangle which shows the detected region of template on source image by RANSAC based approach using random forest similarity distance. The steps to perform template matching from SURF key point matching using random forest distance is shown in figure 3.12.

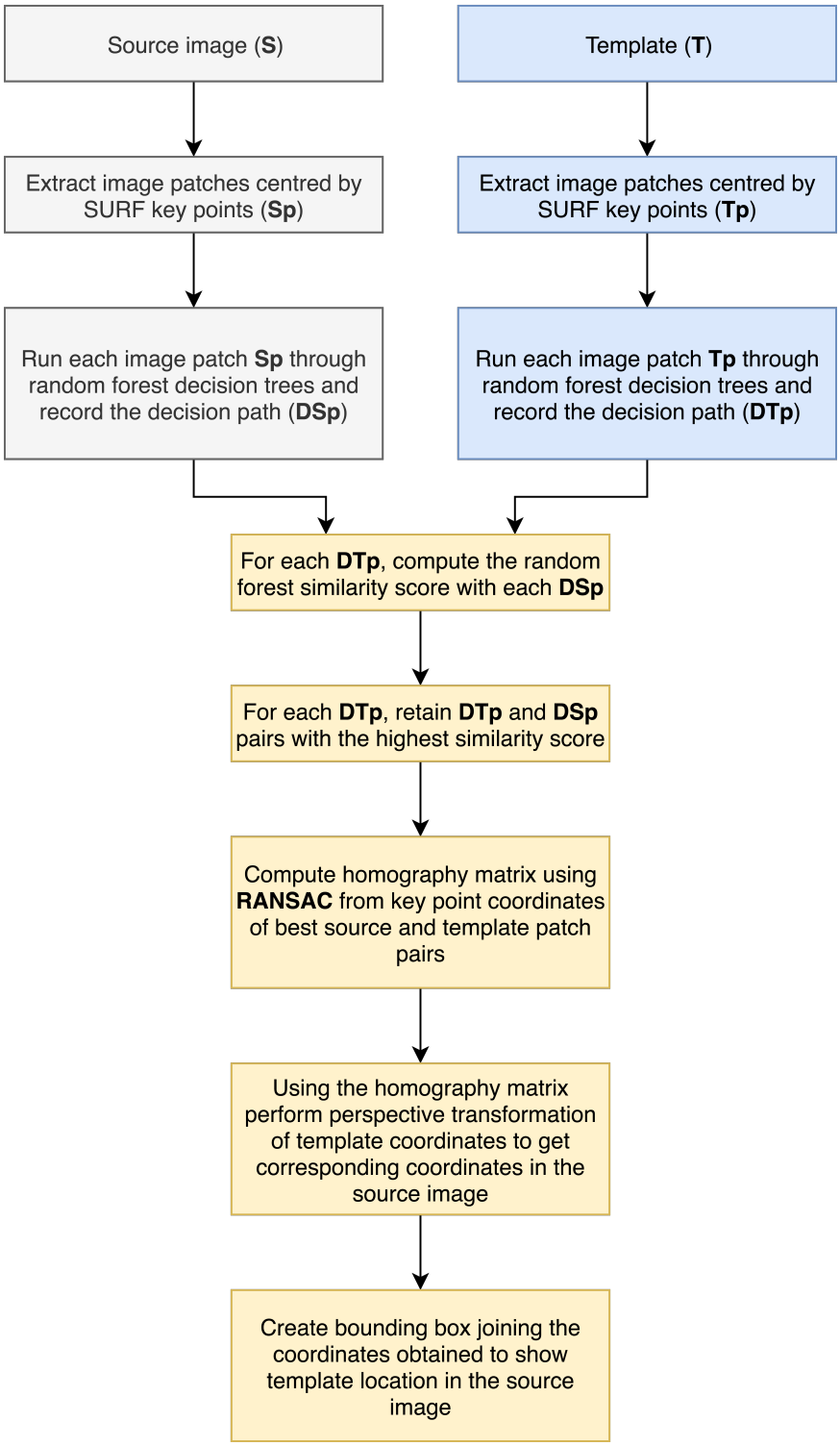


Figure 3.12: Random forest key point matching

Steps to perform template matching from SURF key point matching using random forest similarity distance

Chapter 4

Implementation

In this section, we discuss random forest generation for the proxy classification task which we will use in order to identify the similarity between image patches. We compare random forest similarity, (i) by replacing the rectangle filter similarity measure between image patches described in [27] **(Application A)** with the random forest similarity measure **(Application B)**, and (ii) by replacing SURF descriptor distance for key point matching **(Application C)** with random forest similarity distance **(Application D)**. We present an evaluation method for the applications mentioned above. In the last section of this chapter, we discuss the parameters that we used for applications A, B, C and D.

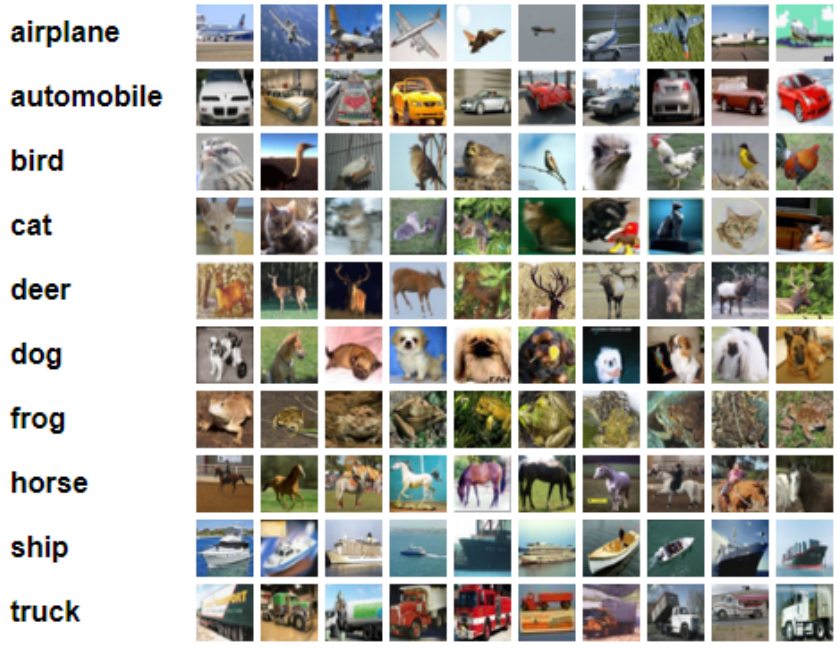


Figure 4.1: An example from the CIFAR 10 dataset

The figure shows the different categories of image patches in CIFAR-10 dataset [34]

4.1 Random forest generation

As discussed in section 3.2, we need to train random forest on a proxy task to use it to get similarity score between image patches. The images we use for random forest training should be of the same size as the image patches we use for testing. For testing, we use only RGB image patches of size 24x24 pixels. Hence we need to train random forest with RGB images of size 24x24 pixels. Also, the random forest should be trained on a generic dataset such that it can be utilized to perform template matching on different sets of images. Hence, we decided to train random forest on CIFAR-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [34]. We found this dataset as appropriate to train the random forest for our proxy classification task as it consists 60000 32x32 generic coloured images in 10 classes, with 6000 images per class. An example image set by CIFAR-10 dataset is shown in figure 4.1.

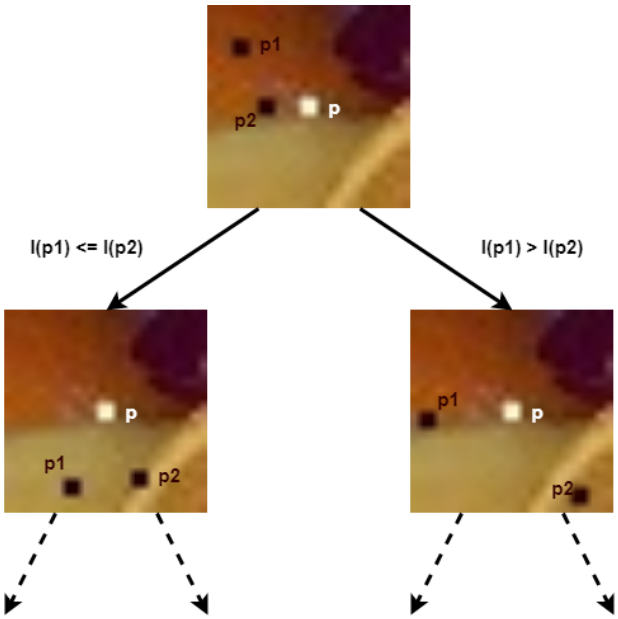


Figure 4.2: A generic tree

Each patch is centred by key point p. $p1, p2$ are pixels around the neighbourhood of the key point p. I is the intensity of the pixel. At each node test, two random pixels around the key point are extracted and their intensities are compared. If $I(p1) <= I(p2)$, the patch follows the left branch of the tree, else it follows the right branch of the tree. Each leaf node includes the posterior probabilities.

In all the applications - A, B, C and D, we use an image patch of size 24 x 24 pixels. Hence in order for the random forest to accept image patches of size 24 x 24 pixels, we resized all patches in CIFAR-10 dataset to 24x24 pixels and used them for training random forest. In order to prevent overfitting and to get better classification accuracy, we increased the number of patches in the data set by

performing the following modifications on the data set. Two additional patches were generated for each original patch (i) by flipping each patch and, (ii) by rotating each patch at 90 degrees. There were 180000 image patches in total for training. For training, we used the same class label of the original patch for transformed patches.

We used the same process and node tests used in [37] to generate the random forest. Figure 4.2 shows a generic tree in the random forest. Since each tree is generated by bagging and also the node tests are generated on the fly at each tree depth, the trees generated performs a different partition depending on the specific tests they contain. Combining the output of all the trees yield a better classification result. Also, the classification accuracy of trees increases with increasing depth and the number of trees. But it also increases the computational cost and memory efficiency. Hence an appropriate number of trees and tree depth should be chosen for training. We will discuss the hyperparameter tuning in the last section of this chapter.

Since each image patch used for training is 3 channel RGB 24x24 pixel patch, there are 24x24x3 = 1728 pixel intensities in each patch that can be used for node testing. Each internal node of the decision tree contains a simple test that divides the space of image patches. Node tests are generated on the fly by comparing the intensities of two random pixels extracted in the neighbourhood of the key point. Since we are training image patches which directly represents the entire image, we can consider the key point as the midpoint of the image patch. But when we use image patches for template matching, these key points will be the SURF key points.

As shown in figure 4.2, each node tests contain simple tests that test the randomly selected pixel intensities and sends the patch to the left or right branch of the tree. Instead of using a binary branch, we can use a ternary branch for our trees by altering the node tests. But in this paper, we are using binary branches for the decision trees of the random forest. Coming to the node tests, the node test we used is $I(p1) <= I(p2)$, where $I(p1)$ and $I(p2)$ are random pixel intensities from the possible 1728 pixels intensities in the image patch. Various other node tests can also be used such as $I(p1) - I(p2) <= I(p3) - I(p4)$, where $p1, p2, p3, p4$ are random pixels in the image patch. More details about different kinds of node tests and their performances are discussed in detail in [37]. But in this paper, we decided to use $I(p1) <= I(p2)$ as the node test in different trees.

The number of possible tests in each node is very high. But only a few hundreds of intensity comparisons are enough to classify an image patch [37]. Each patch reaches the leaf node after a certain number of node tests. Each leaf node contains posterior probabilities of the classes based on the number of training patches that reach the leaf node. A new test patch is classified by dropping it down the decision trees and performing the node tests as showed in figure 4.2. When a patch reaches the leaf node, the posterior probabilities stored in the leaf node, recorded during the training will be returned. In a similar manner, the patch is dropped through all the T decision trees and posterior probabilities are recorded. The class to which the new patch belongs to is determined using the equation below:

$$C(p) = arg\max_{c} \frac{1}{T} \sum_{i=1}^{T} P(Y(p) = c) \tag{4.1}$$

where $P(Y(p) = c)$ is the probability of each class c in the data set. The equation returns the class whose average of posterior probabilities from all trees is maximum.

In order to improve the classification rate, we grew T decision trees using two approaches.

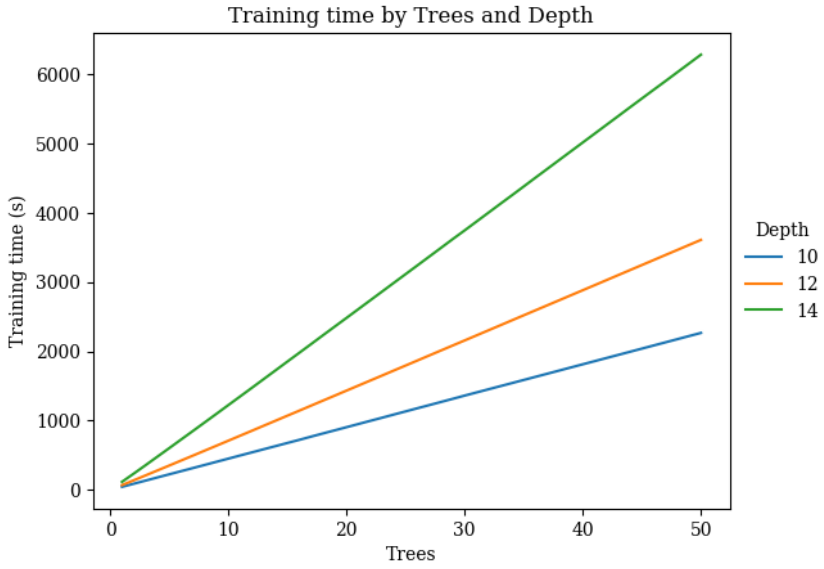


Figure 4.3: Training time - Entropy optimization

In the first method, we constructed trees in a classical, top-down manner, where n node tests are performed at each node and best node test was determined by Shanon's entropy and information gain using equations 1.1,1.2 and 1.3. We chose $n = 10$ [37], a very small number, to reduce the correlation between the resulting trees. For all other nodes, we used $n = 50d$, where d is the depth of the node. The trees were grown by varying depths between 10 and 14, and a maximum of 50 trees was grown. The time to grow trees increased with increasing depth. Figure 4.3 shows the time taken in seconds to generate 50 trees at various depths using this approach. Figures 4.4 and 4.5 shows the number of leaf nodes at different depths in line and 3d bar chart views respectively.

The second approach we used to generate trees was much faster and simpler. Instead of utilising the best test among the set of n tests, we picked a test randomly at each node. This is extremely modest when compared to the first approach. Using this second approach, the time took to grow the trees dropped from tens of minutes to a few seconds on a 2.3 GHz machine. The trees were grown by varying depths between 10 and 20, and a maximum of 100 trees was grown. Figure 4.6 shows the time taken in seconds to generate 100 trees at various depths using this approach.

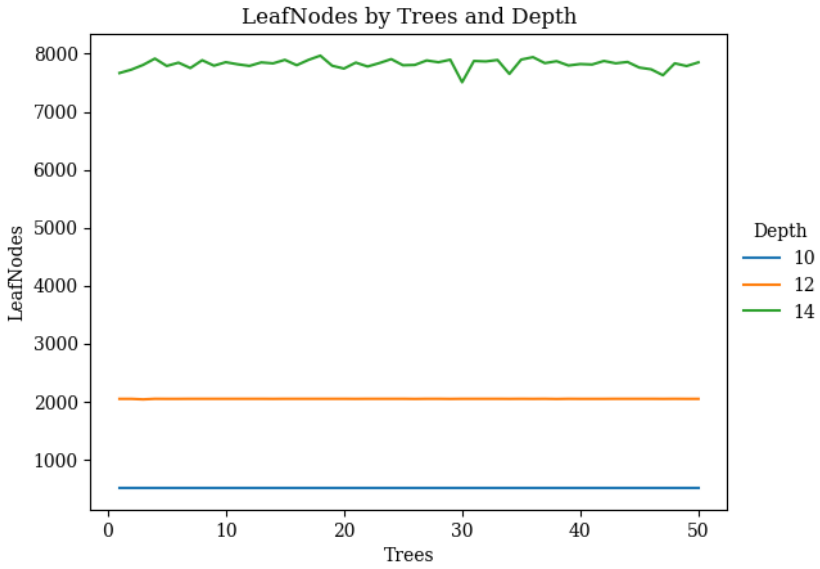


Figure 4.4: Leaf nodes - Entropy optimization

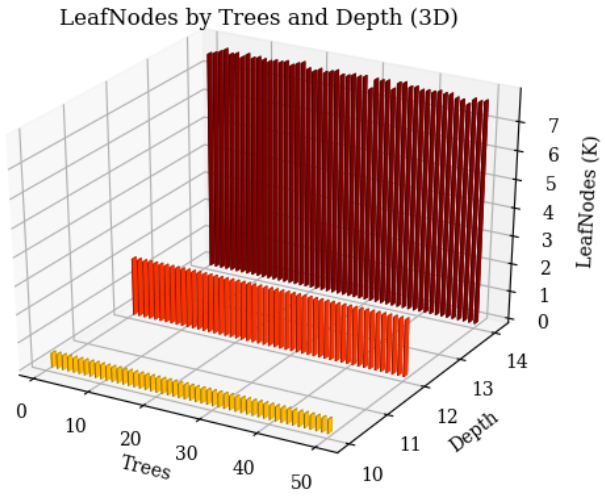


Figure 4.5: Leaf nodes (3D) - Entropy optimization

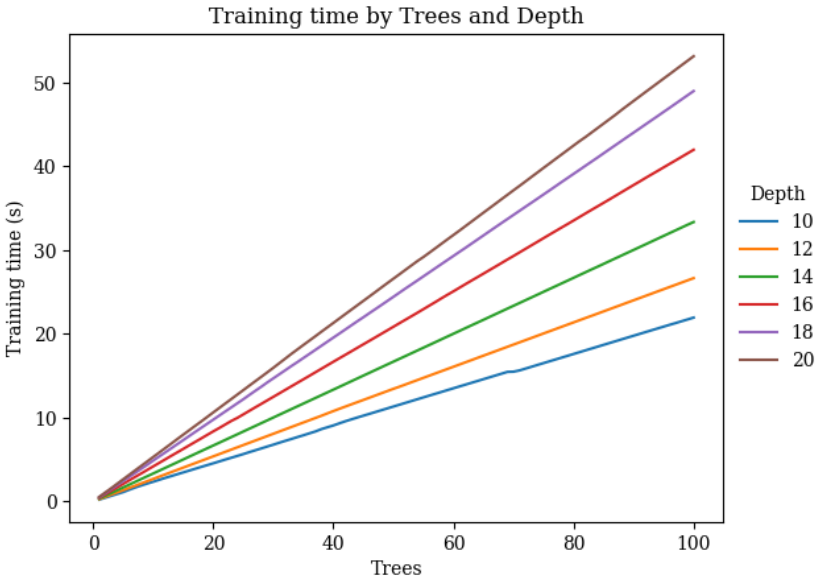


Figure 4.6: Training time - Random test

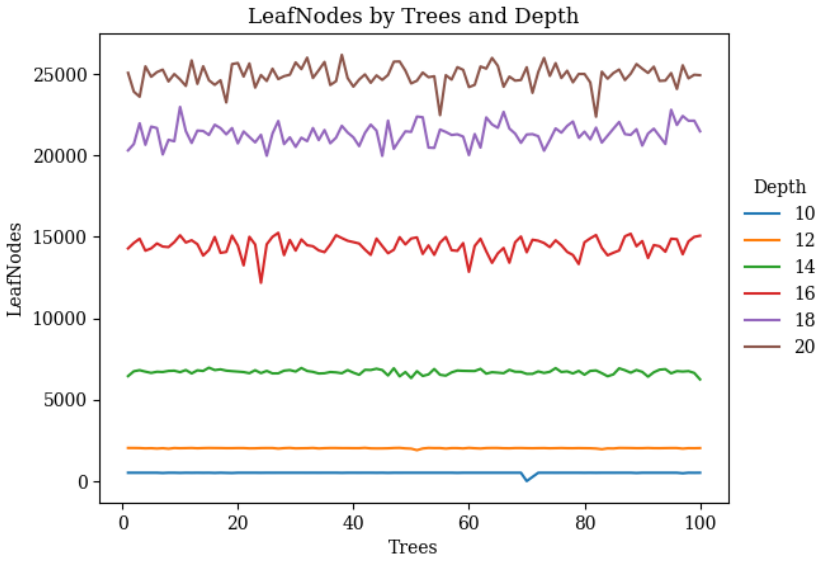


Figure 4.7: Leaf nodes - Random test

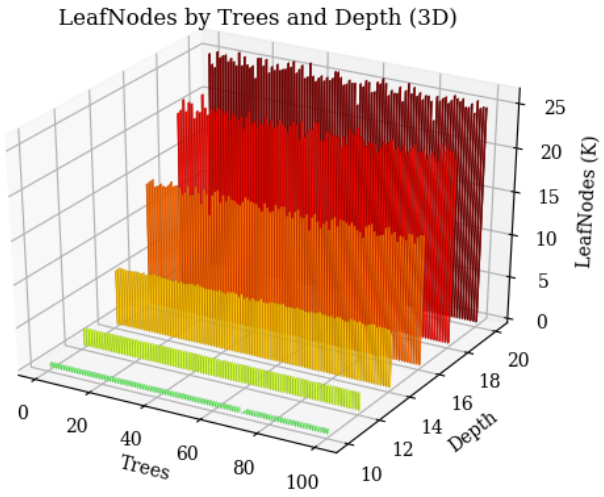


Figure 4.8: Leaf nodes (3D) - Random test

Figures 4.7 and 4.8 shows the number of leaf nodes at different depths in line and 3d bar chart views respectively.

When we build random forests, during bagging, each bootstrap sample generates decision trees by leaving approximately 37% of samples. These left out samples are called Out of Bag (OOB) samples. Each sample in training dataset will be OOB sample of one or more trees. The predictions or class to which these OOB samples belong to can be determined by dropping them through all the trees which are not built using these samples. The ratio of the total number of correctly classified samples to the total number of samples determines the OOB accuracy. OOB accuracy gives much-improved estimates of node probabilities and node error rates in decision trees [8]. We used Out of Bag (OOB) method to measure the testing accuracy of random forests generated using both the approaches.

Figures 4.9 and 4.10 shows the OOB testing accuracy of random forest generated using entropy optimization and random test approaches respectively. OOB testing accuracy using entropy optimization approach was slightly better compared to the random test approach. Nevertheless, the time it took to generate trees using this approach was high when compared to the random test approach. Since in all our experiments, we use only the decision tree paths and not the end estimates from leaf nodes, and since the testing accuracy using random test approach did not vary too much when compared to entropy optimization approach, we used random test approach to generate the random forest in our experiments. The accuracy of random forest similarity measure increases with the increasing number of decision trees. Therefore, the chosen number of decision trees and their depth are a trade-off be-

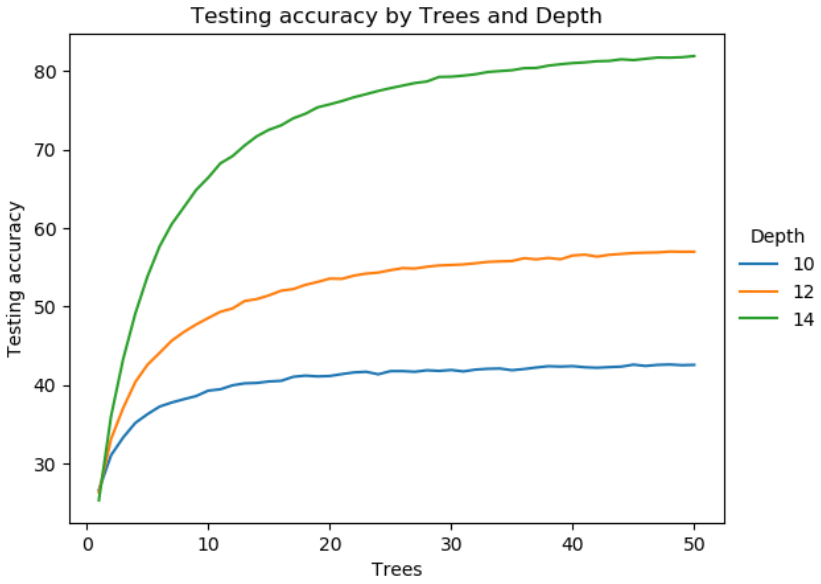


Figure 4.9: Testing accuracy - Entropy optimization

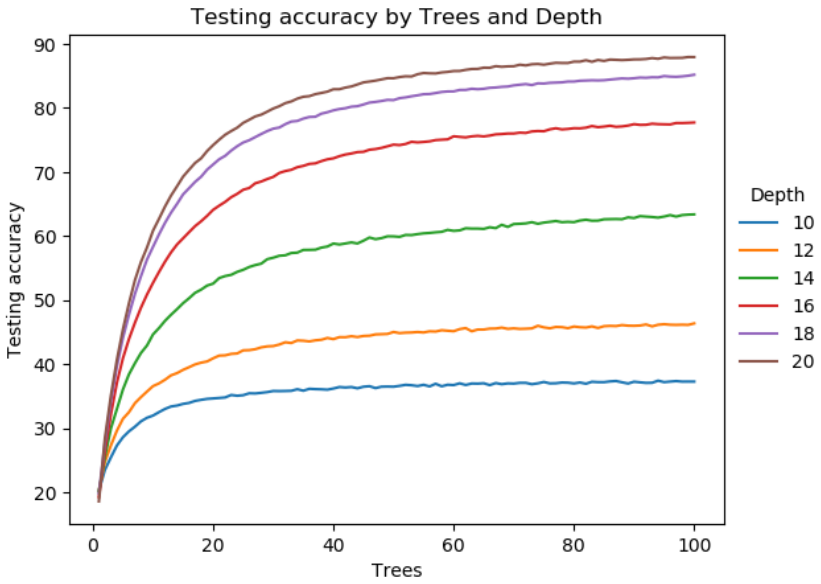


Figure 4.10: Testing accuracy - Random test

tween memory, time and accuracy. For all our experiments we used 8 decision trees with the depth of 10, which gave good results in identifying the similarity between image patches.

4.2 Evaluation method

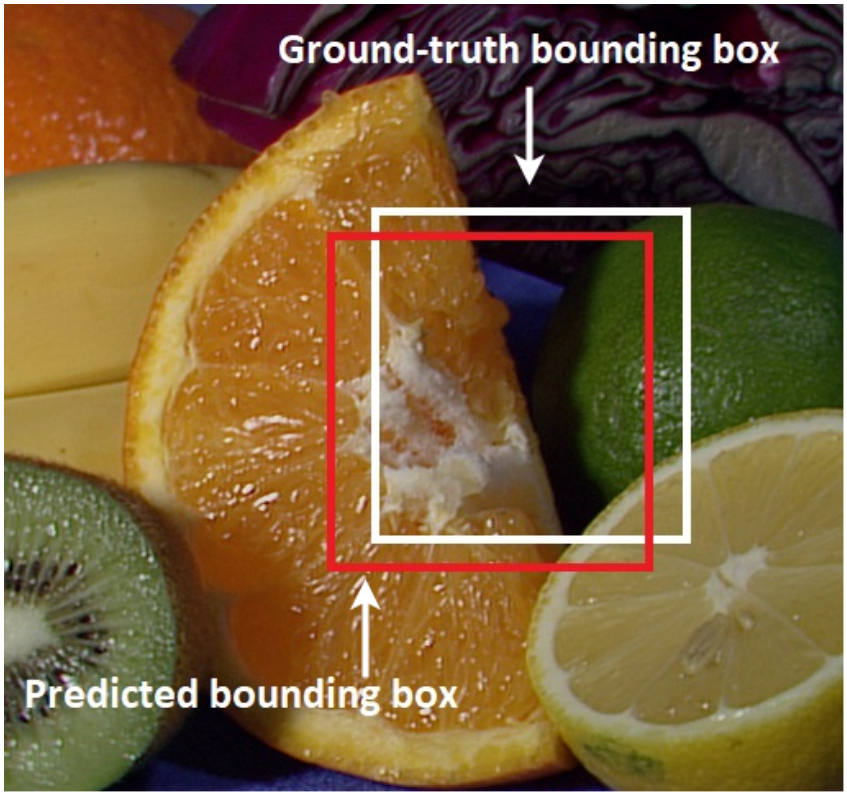


Figure 4.11: Ground-truth bounding box and Predicted bounding box

In the figure, the white rectangle shows the actual location of template and red rectangle shows the predicted template location on fruit image from [27]

We perform template matching using all the applications mentioned in the introduction of this chapter. Each application returns the coordinates of a location in the source image that best matches the template. As discussed in chapter 3, the similarity between image patches are determined using random forest and, template matching is performed. In order to evaluate the accuracy of template matching using these applications, we use Intersection over Union (IOU) method.

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. In this paper, we use Intersection over Union to determine the accuracy of template matching. In order to apply Intersection over Union to evaluate template matching we need: the ground-truth

bounding boxes (i.e., the hand-labelled bounding boxes on the source image that specify where in the source image is the template located exactly) and, the predicted bounding boxes from the applications. As long as we have these two sets of bounding boxes we can apply Intersection over Union. Figure 4.11 shows a visual example of a ground-truth bounding box versus a predicted bounding box. In this figure, the white box is the ground-truth bounding box or the actual location of the template in the source image and the red box is the predicted bounding box or the location of template predicted by any one of the applications (A, B, C or D). Computing Intersection over Union can, therefore, be determined using the equation 4.2. Figure 4.12 shows the same equation visually.

$$IOU = \frac{AreaofOverlap}{AreaofUnion} \tag{4.2}$$

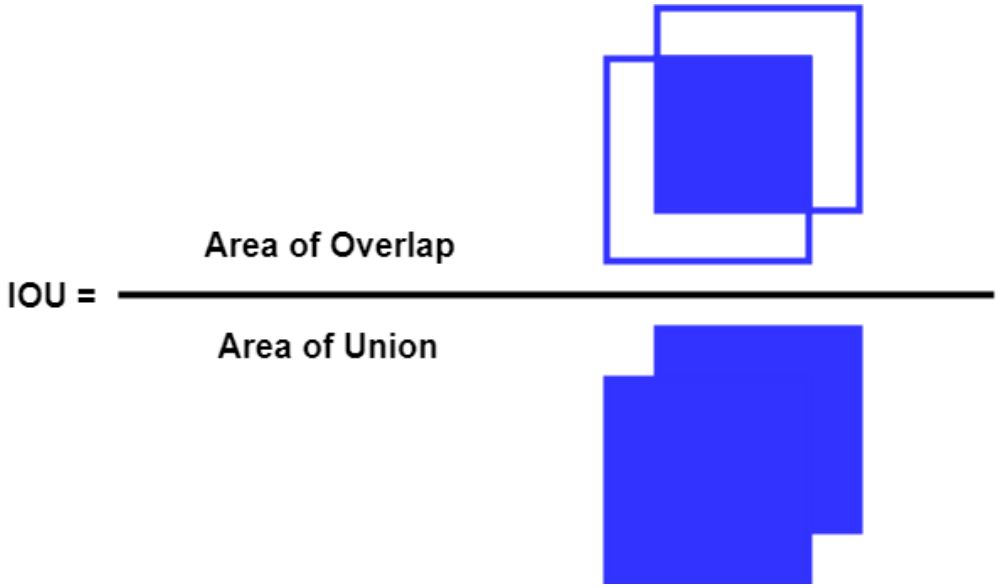


Figure 4.12: Intersection over Union (IOU)

In equation 4.2, the numerator computes the area of overlap between the predicted bounding box and the ground-truth bounding box. The denominator is the area of the union or is the area encompassed by both the predicted bounding box and the ground-truth bounding box. Dividing the area of overlap by the area of union gives the Intersection over Union score. Due to varying parameters such as sliding window size, feature extraction method, key point extracted, etc., finding an exact match of the template on the source image might be unrealistic. Hence Intersection over Union gives a very good estimate of the percentage of overlap of the location predicted by template matching applications with the exact template location on the source image. The higher the score, higher is the matching accuracy. Figure 4.13 shows an example of computing Intersection over Unions for various bound-

ing boxes. As we can see, predicted bounding boxes that highly overlap with the ground-truth bounding boxes have higher Intersection over Union scores compared to those with less overlap. This makes Intersection over Union an excellent metric for evaluating our template matching applications. An Intersection over Union score greater than 0.5 is considered a good match.

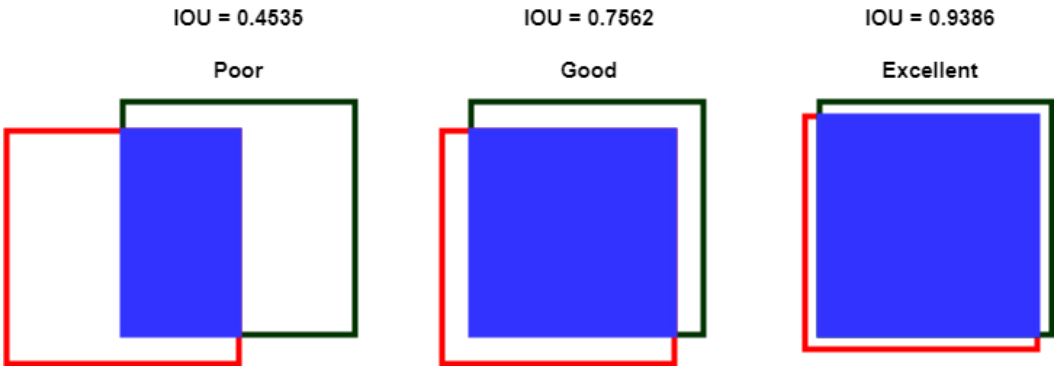


Figure 4.13: An example of computing Intersection over Union

4.3 Parameters

In this section, we present the parameters we use for template matching applications A, B, C and D. In all these applications we use an image patch of size 24x24 pixels.

In section 3.1, we presented the patch based correlation and template matching technique discussed in [27]. The template is divided into non-overlapping patches of the same size as showed in figure 3.2. The patches close to the image boundary of the template are not used if they are smaller than 24x24 pixels. The features are extracted using rectangle filters. The filter size is 8x12 for (a) and (c), 12x8 for (b) and (d), and 8x8 for (e) in figure 3.3. Within each patch, the filters are shifted by a step size of 4 pixels. Therefore 105 features are extracted out of each patch. In order to select the salient features, we set $\alpha = 0.95$ in equation 3.4. In all our experiments utilizing application A, the template is translated with a step size of one pixel in the source image for matching and only grey level intensity values are used.

In section 3.2, we discussed how we can use random forest similarity instead of rectangle filter similarity used in [27] for patch similarity. Similar to application A, in application B, the template is translated with a step size of one pixel in the source image for matching. But RGB pixel intensities are used instead of grayscale intensities since we trained the random forest to process patches with RGB pixel intensities. Also in order to speed up the matching process, we randomly extract n patches from the template as we slide template over the source image instead of using all non-overlapping patches from the template. In all our experiments using application B, we set $n = 5$.

In section 3.3, we discussed how we can perform template matching using key point matching. In both the applications, application C and D, using SURF descriptor distance and random forest similarity distance respectively, SURF key points are used. Once the key points are matched, the template location on the source image is estimated using RANSAC based approach. The template matching accuracy depends on the number of key points extracted from source image and template. Higher the number of SURF key points, greater the accuracy. Nonetheless, it, in turn, affects the key point matching time. Hence a good amount of key points needs to be extracted from the template and source image. In all our experiments using application C and D, we set the Hessian threshold to 300 to extract approximately 300 SURF key points from template and source image. Also in order to speed up the matching process, we can filter the key point matching pair from template and source image whose distances are greater than the certain threshold value. For application C using SURF descriptor distance, we set this threshold at 0.75. We only retain those key point pairs where the template key point descriptor distance is less than 0.75 times the source key point descriptor distance. In all our experiments using application D using random forest similarity distance, we don't set this threshold value and all matching key point pair from template and source image are used.

Chapter 5

Experiments and Results

In this chapter, we provide an outline and an evaluation of the experiments performed. In the first section, we describe the dataset and tests employed to our experiments. In the second and third section, we discuss the template matching results from applications A, B and applications C, D respectively.

5.1 Data sets and tests

The experiments are performed on a local database that is prepared by collecting most of the image objects from the World Wide Web (WWW). Some sample images from the local database are shown in figure 5.1.

Figure 5.1: Sample test images

Some sample images in local database collected randomly from world wide web

Data Set	Image Size	Pattern/ Template Size
class A	480x640	132x134
class B	640x800	202x202
class C	640x800	254x298
class D	960x1280	302x356

Table 5.1: Image data sets
Image data sets of different sizes of scene images and patterns for Local Database

The local database consists of 4 sets consisting of 10 images of different sizes collected from the web. Table 5.1 shows the size of images in dataset and size of the template that will be extracted from each image.

Transformation	Description
ORIGINAL	Source image without any modification / transformation
GNO_0_30	Source image with gaussian noise with $mean = 0$ and $standard\ deviation = 30$
GBR_1.5_0	Source image with gaussian blur with $kernelsize = (3, 3)$ and $standard\ deviation = 1.5$
GBR_2.5_0	Source image with gaussian blur with $kernelsize = (5, 5)$ and $standard\ deviation = 2.5$
ROTATE_5	Source image is rotated around mid point by 5 *degree*
ROTATE_10	Souce image is rotated around mid point by 10 *degree*
OCCLUDED	Randomly chosen part of template, that is less than half of template size is occluded from template

Table 5.2: Image transformations
Image transformations performed on each image from datasets in table 5.1

For each source image in the data sets in table 5.1, transformations specified in table 5.2 are applied and template matching is performed using applications A, B, C and D. So each application will perform template matching on 280 different images with transformations. In the next two sections, we discuss the results of template matching performed by these applications. Figure 5.2 shows a sample source and template images from local database after applying transformations from table 5.2.

The steps followed to conduct template matching tests are as follows:

1. From each source image of the data sets in table 5.1, a template with the size specified in the table is randomly extracted and its coordinates are saved. The quadrilateral joining the coordinates is the ground-truth bounding box.

2. Individual transformations specified in table 5.2 are applied one by one to source image and template.

3. Template matching is then performed using this transformed template and source image using applications A, B, C and D.

4. Each template matching application returns the coordinates of the best matching location of the template on source image using which predicted bounding box can be found.

5. Intersection over Union score is calculated using the equation 4.2, which gives the accuracy of template matching by the applications A, B, C and D. Higher the score, better the template matching application.

Figure 5.2: Image transformations
Sample source and template images from local database after applying transformations from table 5.2

5.2 Results - Template matching

In this section, we show the results from applications A and B (template matching using rectangle filter similarity and random forest similarity respectively).

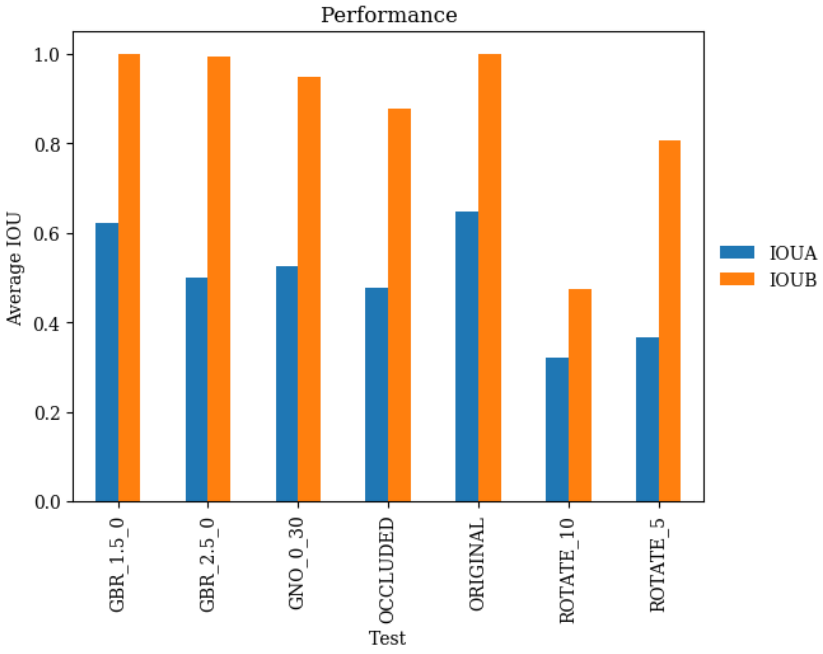


Figure 5.3: Performance - applications A and B

Figure 5.3 shows the performance of applications A and B based on the average IOU score. Each bar represents an average IOU score of 40 images. The figure clearly demonstrates that the template matching using random forest similarity measure outperformed the template matching method using rectangle filters. We can also notice that the recognition rate/template matching accuracy for original source image and image with gaussian blur using random forest similarity measure is close to 1. Application B also outperforms application A in occlusion and Gaussian noise tests. For template matching on rotated images, application A performed poorly and application B performed averagely. Since both applications use the sliding window approach to perform template matching, a sophisticated approach is necessary to perform template matching on rotated images. Figure 5.4 shows the histogram of IOU score for template matching on 280 images using applications A and B. Nearly 250 images out of 280 images have an IOU score above 0.5 using application B, which clearly indicates that random forests can be used to accurately identify the similarity between image patches.

Figure 5.5 presents the average template matching time using applications A and B. Template matching using random forest similarity clearly takes more time compared to rectangle filter similarity. This is caused by recursive function calls

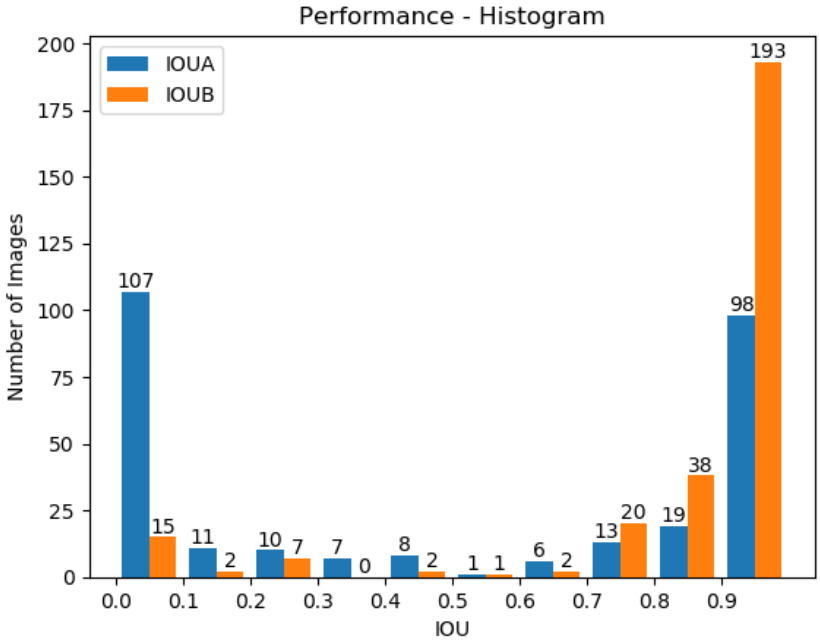


Figure 5.4: Performance histogram - applications A and B

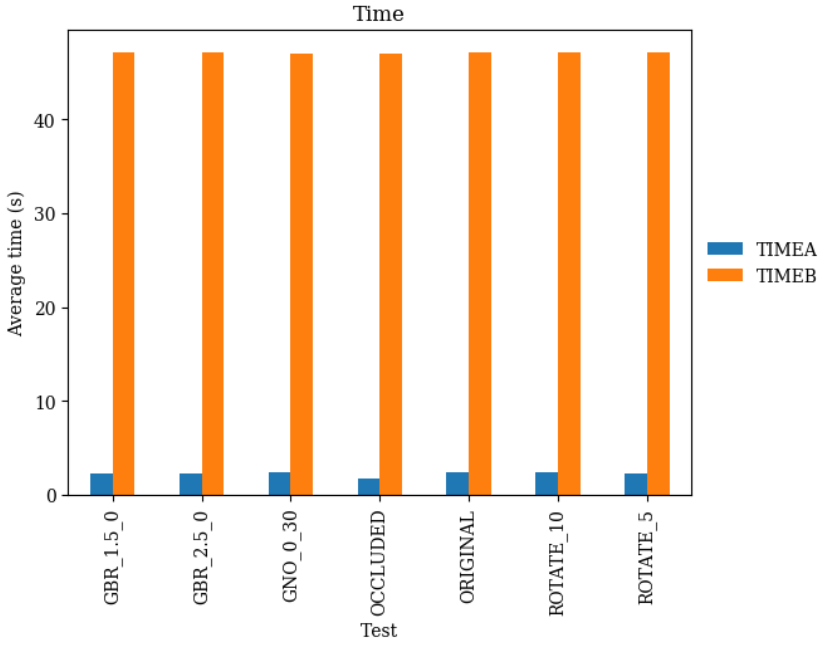


Figure 5.5: Average time - applications A and B

when determining the similarity between image patches using decision tree paths in the random forest. In rectangle filter similarity, the time taken is less due to simple summations, feature filtering and integral image representations. Each bar in the figure indicates the average time taken for 40 images. This includes images of larger sizes (class D) also. Hence template matching using random forest similarity measure takes less time on smaller images compared to larger images. Overall, the template matching accuracy from application B is high compared to application A. But in contrast, application B takes more time to perform template matching compared to application A. Due to this application B cannot be used for performing template matching in real time scenarios. Appendix A presents detailed results on performance and the average time for template matching using applications A and B for different source images and templates given in table 5.1.

5.3 Results - Template matching from Keypoint matching

In this section, we present the results from applications C and D (template matching from key point matching using SURF descriptor distance and random forest similarity distance respectively).

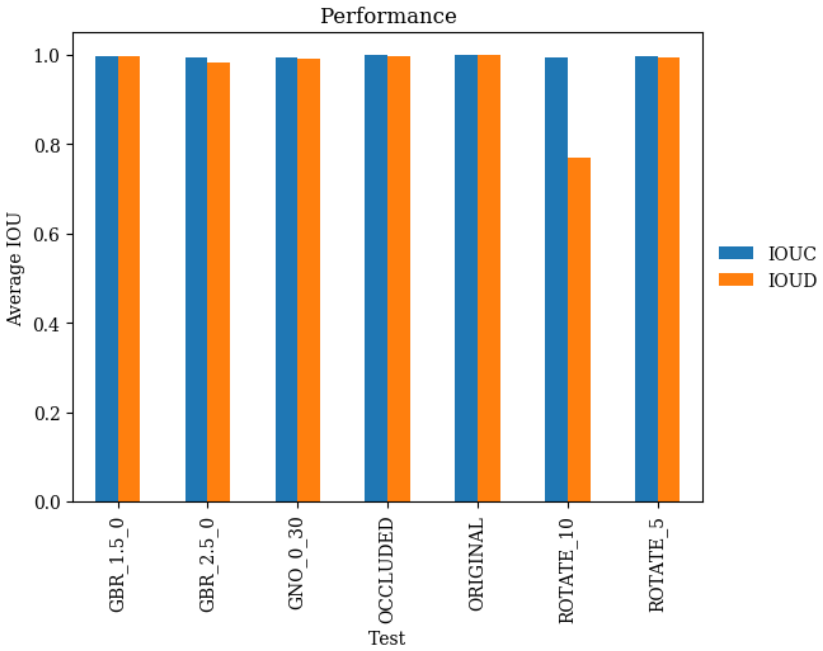


Figure 5.6: Performance - applications C and D

Figure 5.6 shows the performance of applications C and D based on the average IOU score. Each bar represents an average IOU score of 40 images. From the fig-

ure, we can find that template matching from key point matching using both SURF descriptor distance and random forest similarity distance have similar performance. Only in few rotation tests, application C outperformed application D. In all other tests, the IOU score from both the approaches is close to 1, which clearly indicates that random forest can be successfully used to perform key point matching and its similarity distance can be used in place of SURF descriptor distance. Figure 5.7 shows the histogram of IOU score for template matching on 280 images using applications C and D. Application C clearly outperformed application D by performing template matching with high accuracy. Application D performed very close to application C but failed to accurately perform template matching in rotation cases.

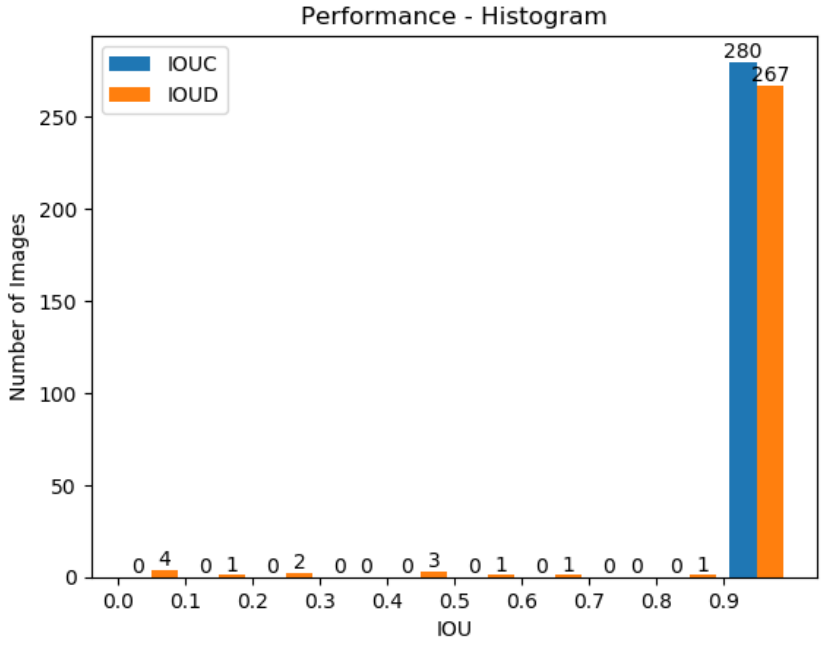


Figure 5.7: Performance histogram - applications C and D

Figure 5.8 presents the average template matching time using applications C and D. Template matching from key point matching using random forest similarity distance clearly takes more time compared to SURF descriptor distance. This is caused by recursive function calls during calculating the similarity between image patches using decision tree paths in the random forest. Also as explained in section 4.3, since application D does not filter key point matching pair using threshold value like application C, this is also a reason for an increase in template matching time. Each bar in the figure indicates the average time taken for 40 images. Overall, the template matching accuracy of application C is high compared to application D. But application D performed very close to application C in terms of accuracy. As we can see in appendix B, for smaller image sizes such as class A images, the accuracy of

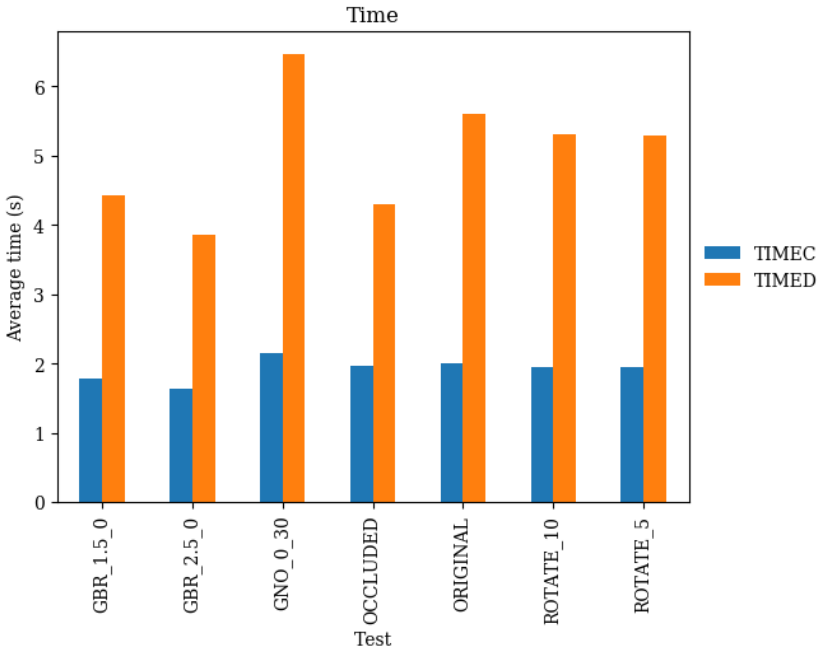


Figure 5.8: Average time - applications C and D

application D is close to application C and it performed faster compared to application C. The time taken to perform template matching using application D increased with increasing image size. However, application C performed faster for larger image sizes compared to smaller image sizes. Appendix B presents detailed results on performance and the average time for template matching using applications C and D for different source images and templates given in table 5.1.

5.4 Tests on extremely noisy images

We also experimented template matching on noisy images such as GNO_0_100 (Gaussian noise with mean = 0 and standard deviation = 100) as showed in figure 5.9 using other template matching applications. Application A performed poorly. Application C performed accurately compared to applications B and D.

Applications B and D did not perform template matching accurately on extremely noisy images due to less number of decision trees in the random forest. In all our experiments, we used 8 decision trees with depth 10 in the random forest. We experimented template matching on extremely noisy images by varying the number of trees from 8 to 40. The accuracy of template matching is increased with the number of trees in the forest. Nonetheless, as we increased the number of trees in the forest, the computation time to identify the similarity between patches also increased which in turn increased the template matching time. Hence we cannot use the random forest to identify the similarity between image patches in real time

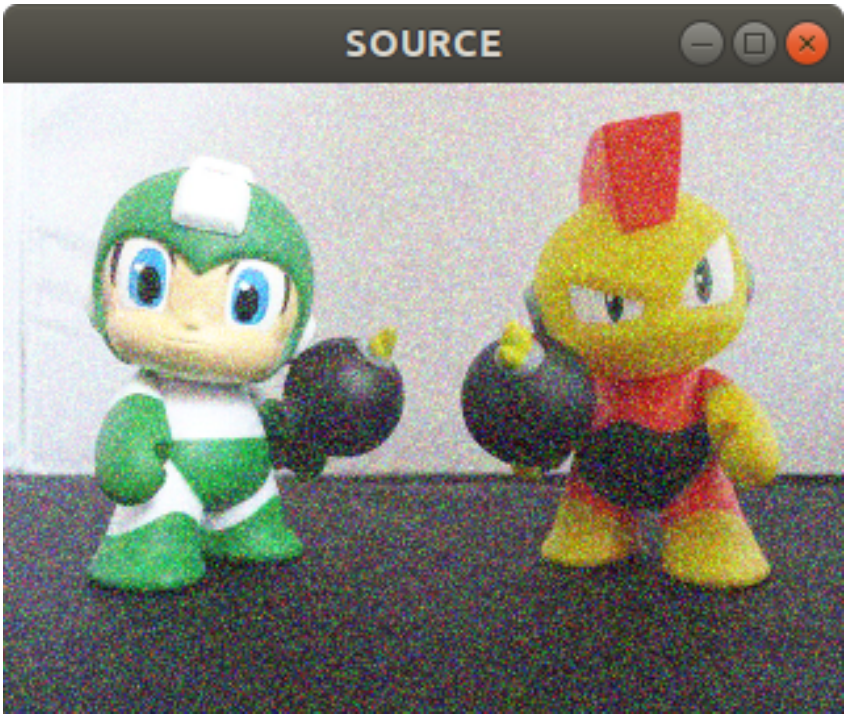


Figure 5.9: GNO_0_100 - Extremely noisy image

for extremely noisy images. For non-real time applications where the time is not a constraint, the random forest can be successfully utilized to perform template matching by identifying the similarity between image patches.

Chapter 6

Conclusion

6.1 Summary

Other than performing tasks like classification, regression and clustering, random forests can be used to determine the similarity between data points. When data points are similar, they naturally take the same decision path in the decision trees. We can use overlapping paths between data points as a similarity measure.

In this paper, we presented how we can potentially utilize random forests to accurately identify the similarity between image patches. We discussed in detail the steps to efficiently generate a proxy random forest and how we can utilize it to accurately calculate the similarity score between image patches and in turn use it in the context of template matching.

We compared the accuracy of template matching using random forest similarity measure with previous work which utilized rectangle filters to identify the similarity between image patches. We showed that template matching using random forest similarity measure outperformed the method which used rectangle filter for identifying the similarity between image patches.

We also presented how we can use random forest similarity measure in the context of key point matching and compared its accuracy with SURF descriptor distance. We showed that template matching from key point matching using random forest similarity distance performed very close to the one which used SURF descriptor distance for key point matching.

We performed template matching experiments using random forest similarity measure on the source and template images with various transformations like noise, blur, occlusion, rotational changes etc and discussed the results in detail. We also discussed how we can improve the accuracy of template matching when the source images used are extremely noisy.

6.2 Future work

We found that random forests perform poorly on identifying the similarity between patches when they are extremely noisy. The reason for this might be due to the kind of split tests used in the nodes of the decision trees and the pixel intensity levels used. One workaround we found was to increase the number of decision trees in the random forest to improve the accuracy. But this increases the computation time. As a future work, different node tests can be tested and decision trees can be optimally built to successfully differentiate and classify the noisy patches.

Appendix A

Appendix - A

A.1 Template matching using applications A and B

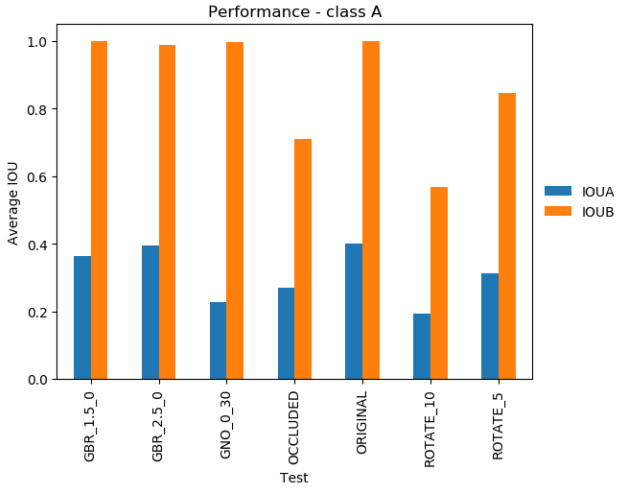


Figure A.1: Performance - applications A and B - class A

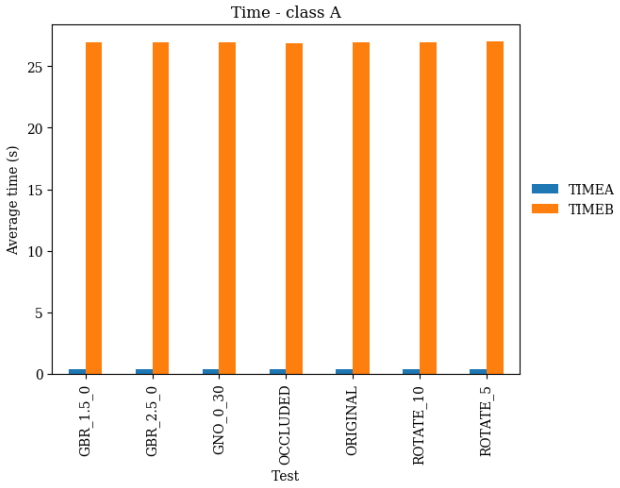


Figure A.2: Average time - applications A and B - class A

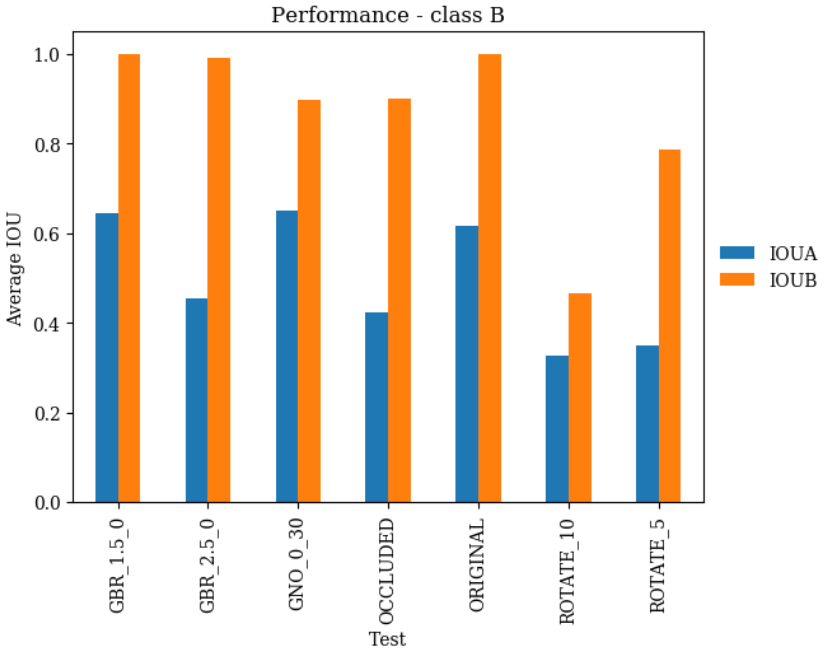


Figure A.3: Performance - applications A and B - class B

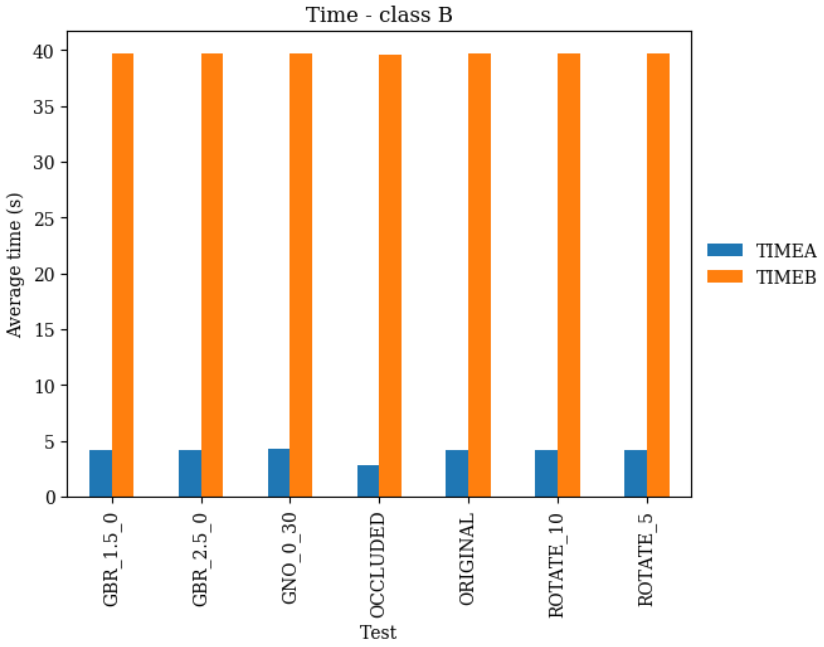


Figure A.4: Average time - applications A and B - class B

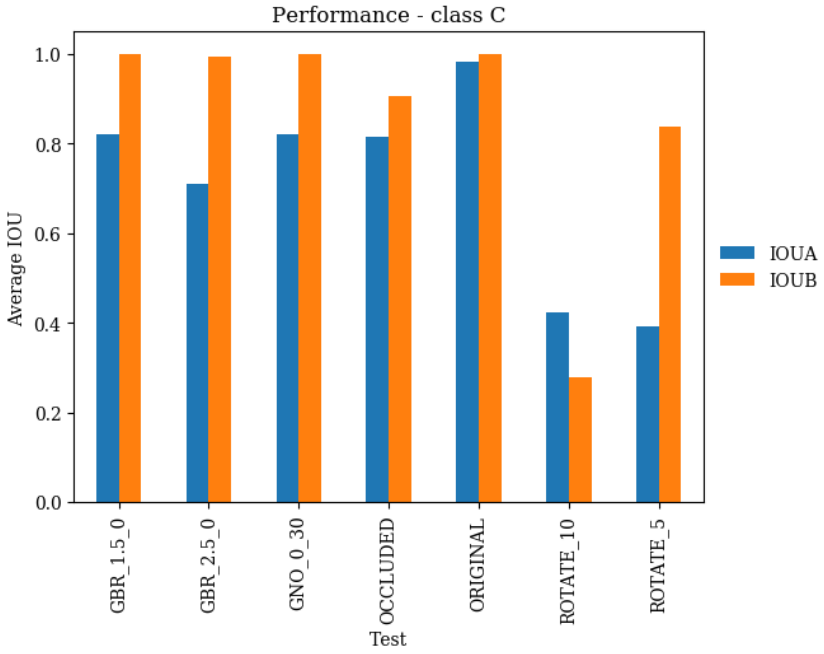


Figure A.5: Performance - applications A and B - class C

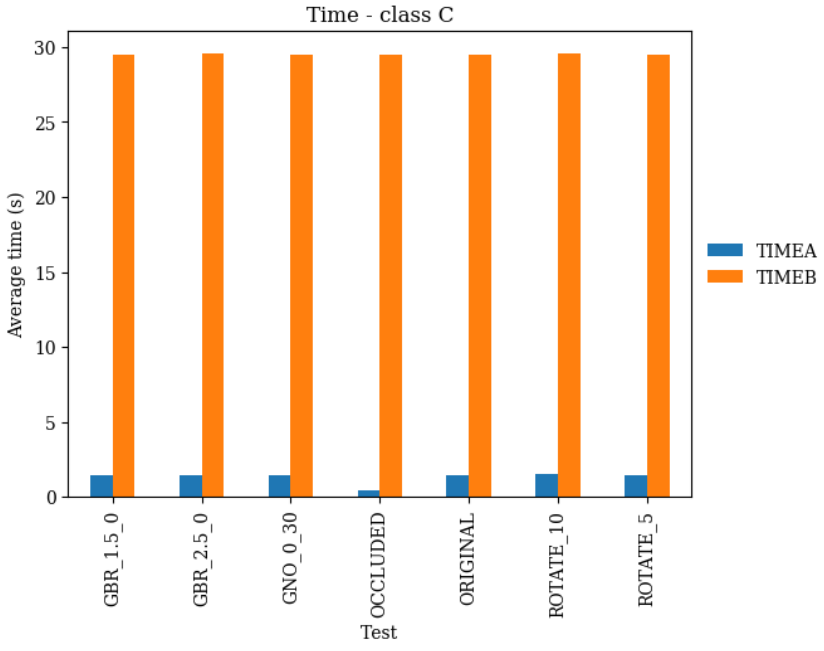


Figure A.6: Average time - applications A and B - class C

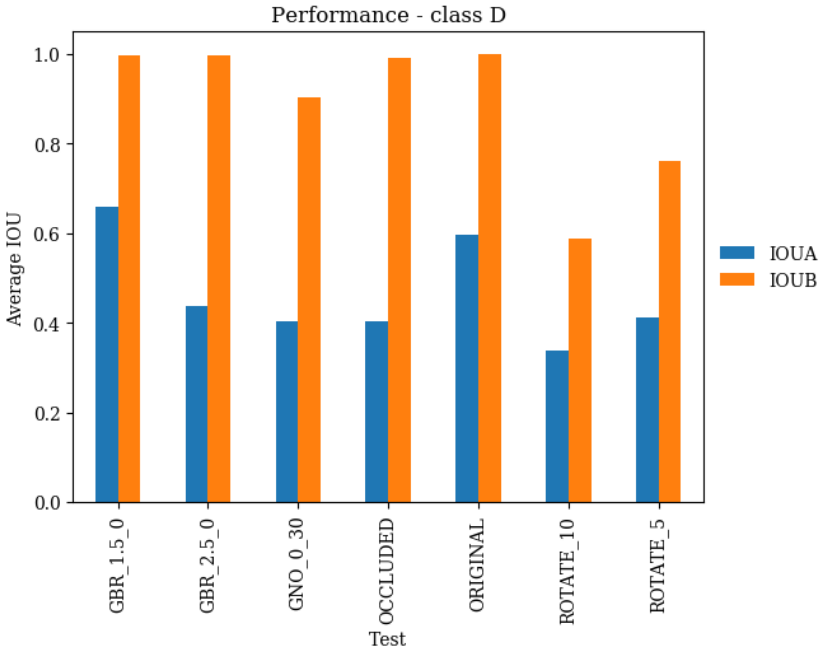


Figure A.7: Performance - applications A and B - class D

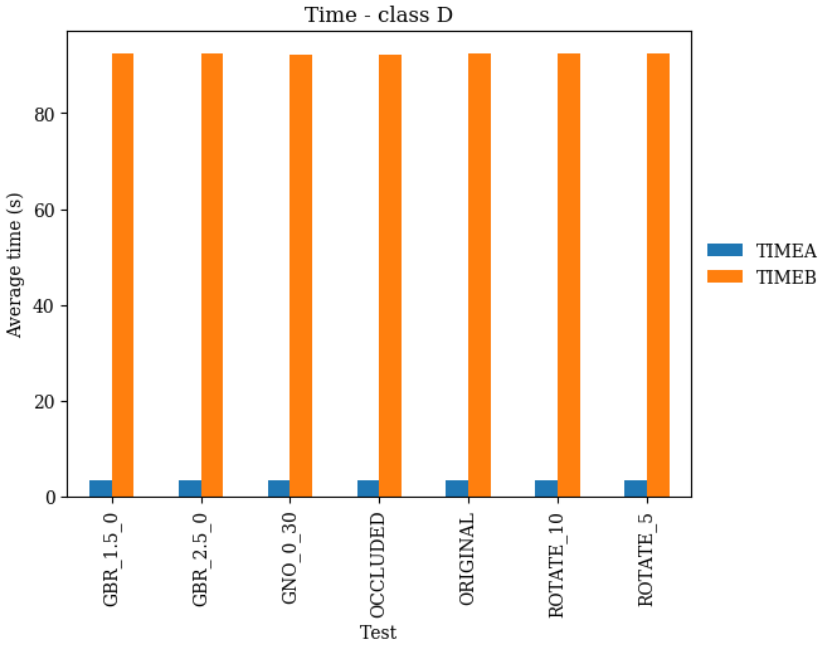


Figure A.8: Average time - applications A and B - class D

Appendix B

Appendix - B

B.1 Template matching using applications C and D

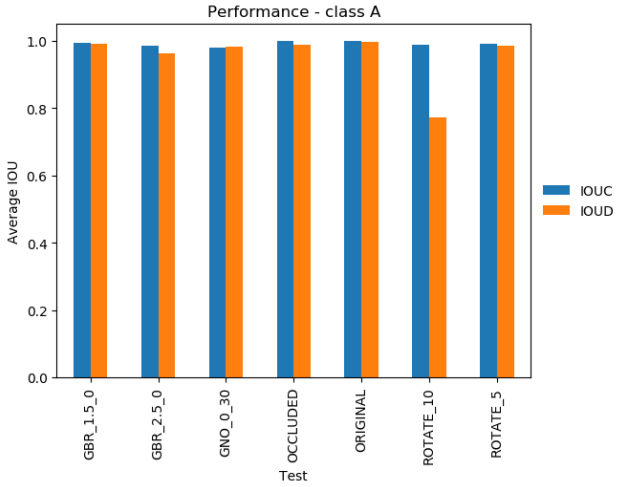


Figure B.1: Performance - applications C and D - class A

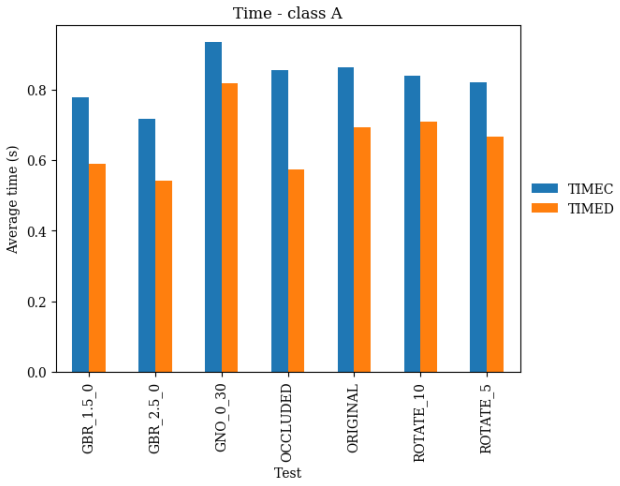


Figure B.2: Average time - applications C and D - class A

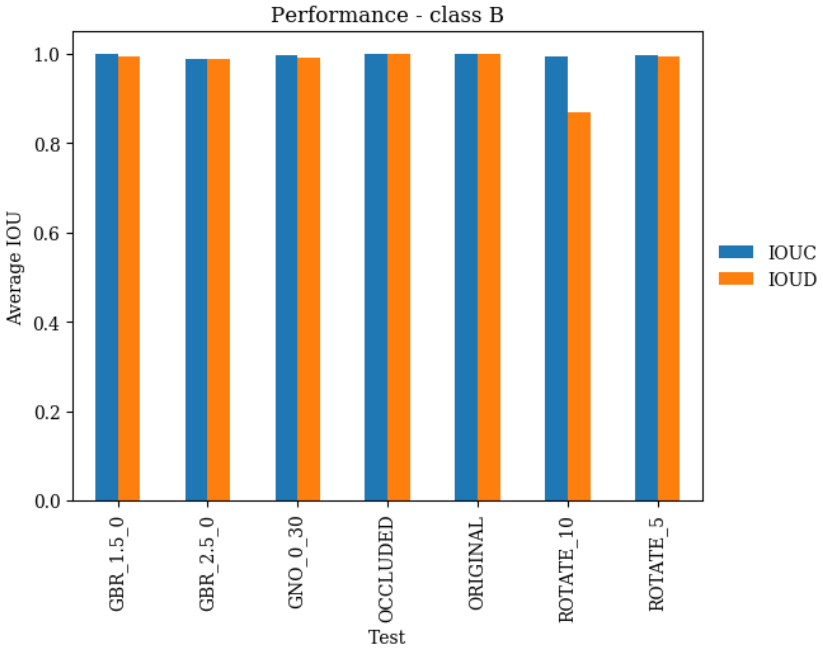


Figure B.3: Performance - applications C and D - class B

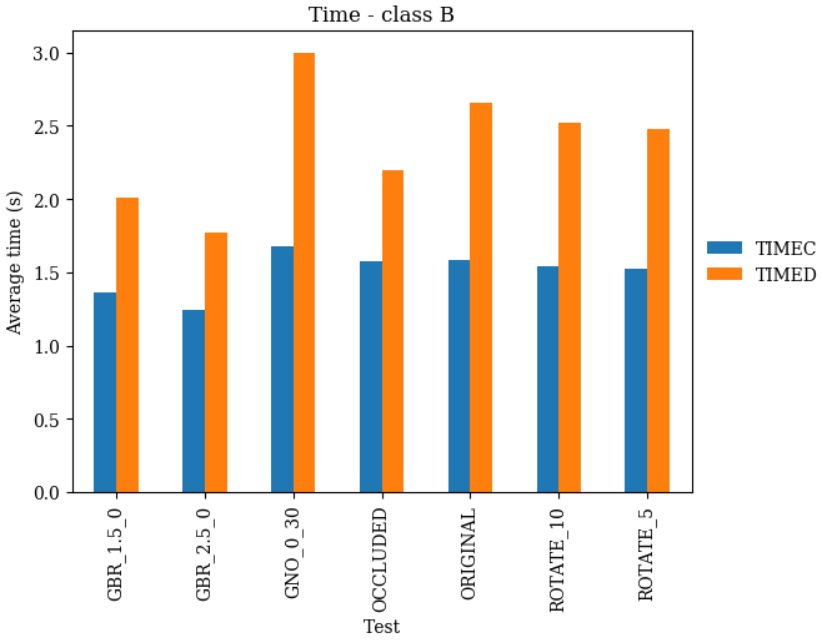


Figure B.4: Average time - applications C and D - class B

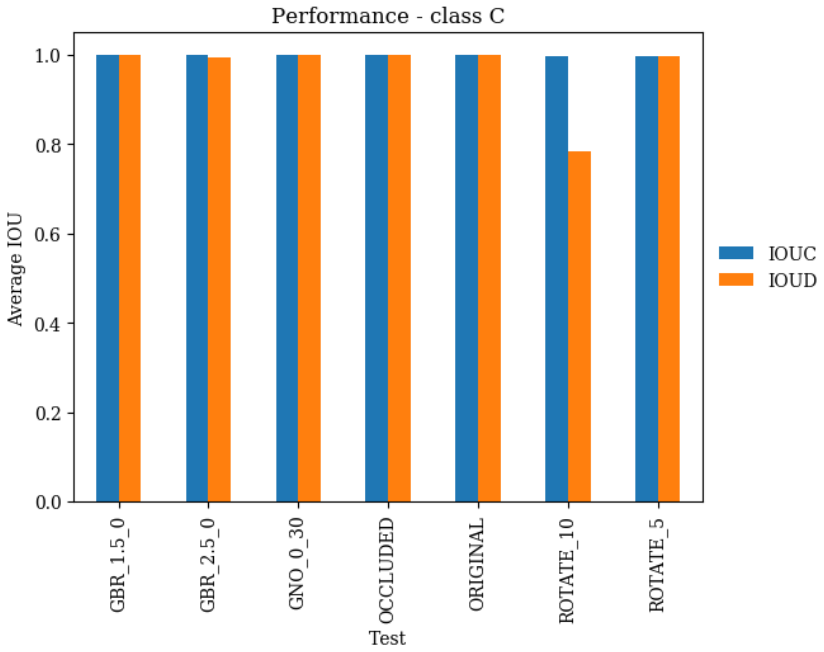


Figure B.5: Performance - applications C and D - class C

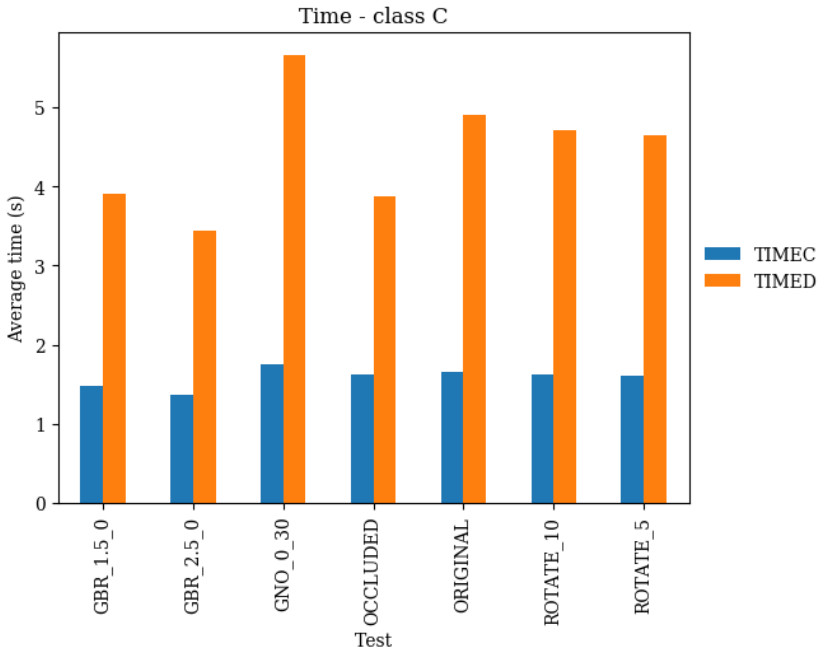


Figure B.6: Average time - applications C and D - class C

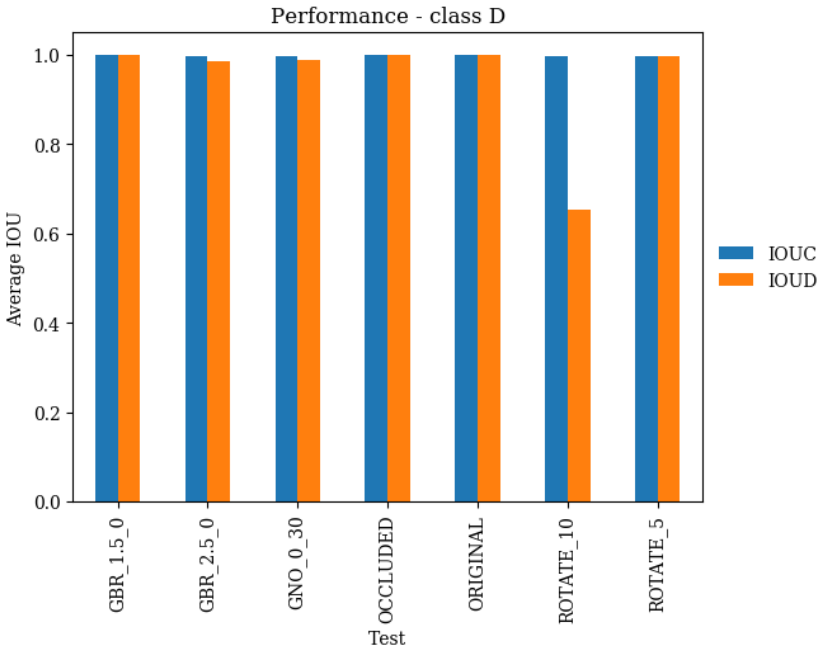


Figure B.7: Performance - applications C and D - class D

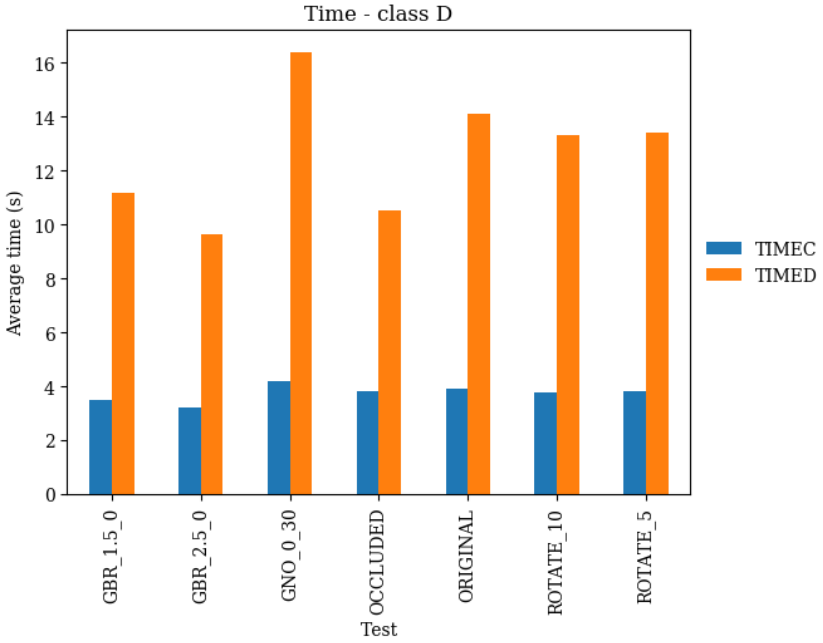


Figure B.8: Average time - applications C and D - class D

Bibliography

[1] E. Allen, S. Horvath, F. Tong, P. Kraft, E. Spiteri, A. D. Riggs, and Y. Marahrens. High concentrations of long interspersed nuclear element sequence distinguish monoallelically expressed genes. *Proceedings of the National Academy of Sciences*, 100(17):9940–9945, 2003. 29

[2] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997. 16, 25

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008. 19

[4] V. Belle, T. Deselaers, and S. Schiffer. Randomized trees for real-time one-step face detection and recognition. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008. 26

[5] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 25

[6] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101–mining discriminative components with random forests. In *European Conference on Computer Vision*, pages 446–461. Springer, 2014. 26

[7] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 16

[8] L. Breiman. Out-of-bag estimation, 1996. 51

[9] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 13, 16

[10] L. Breiman. *Classification and regression trees*. Routledge, 2017. 16

[11] L. Breiman and A. Cutler. Random forests manual v4. In *Technical report*. UC Berkel, 2003. 29

[12] K. Briechle and U. D. Hanebeck. Template matching using fast normalized cross correlation. In *Optical Pattern Recognition XII*, volume 4387, pages 95–103. International Society for Optics and Photonics, 2001. 23

[13] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010. 20

[14] F. C. Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH computer graphics*, volume 18, pages 207–212. ACM, 1984. 18, 34

[15] T. Dekel, S. Oron, M. Rubinstein, S. Avidan, and W. T. Freeman. Best-buddies similarity for robust template matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2021–2029, 2015. 24

[16] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine learning*, 32:1–22, 1998. 16

[17] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1841–1848, 2013. 26

[18] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1558–1570, 2015. 26

[19] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001. 31

[20] E. Elboher and M. Werman. Asymmetric correlation: a noise robust similarity measure for template matching. *IEEE Transactions on Image Processing*, 22(8):3062–3073, 2013. 23

[21] A. Ellahyani, M. El Ansari, and I. El Jaafari. Traffic sign detection and recognition based on random forests. *Applied Soft Computing*, 46:805–815, 2016. 26

[22] C. Englund and A. Verikas. A novel approach to estimate proximity in a random forest: An exploratory study. *Expert systems with applications*, 39(17):13046–13050, 2012. 28

[23] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 40

[24] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *Decision forests for computer vision and medical image analysis*, pages 143–157. Springer, 2013. 25

[25] K. R. Gray, P. Aljabar, R. A. Heckemann, A. Hammers, D. Rueckert, A. D. N. Initiative, et al. Random forest-based similarity measures for multi-modal classification of alzheimer's disease. *NeuroImage*, 65:167–175, 2013. 26

[26] J. Greenhalgh and M. Mirmehdi. Traffic sign recognition using mser and random forests. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pages 1935–1939. IEEE, 2012. 26

[27] G. Guo and C. R. Dyer. Patch-based image correlation with rapid filtering. In *CVPR*, volume 7, page 1, 2007. 3, 4, 13, 17, 18, 24, 31, 32, 33, 34, 35, 41, 42, 45, 53, 55

[28] Y. Hel-Or, H. Hel-Or, and E. David. Matching by tone mapping: Photometric invariant template matching. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):317–330, 2014. 23

[29] J.-J. Huang and W.-C. Siu. Practical application of random forests for super-resolution imaging. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 2161–2164. IEEE, 2015. 25, 26

[30] J.-J. Huang and W.-C. Siu. Learning hierarchical decision trees for single-image super-resolution. *IEEE Trans. Circuits Syst. Video Techn.*, 27(5):937–950, 2017. 26

[31] M. Jeong and B. C. Ko. Driverâs facial expression recognition in real-time for safe driving. *Sensors*, 18(12):4270, 2018. 26

[32] Y. Jiang, L. Ruan, L. Xiao, X. Liu, F. Yuan, and H. Wang. Thtm: A template matching algorithm based on hog descriptor and two-stage matching. In *AIP Conference Proceedings*, volume 1955, page 040131. AIP Publishing, 2018. 23

[33] F. Jurie, M. Dhome, et al. Real time robust template matching. In *BMVC*, pages 1–10, 2002. 18

[34] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 45, 46

[35] S. L. A. Lee, A. Z. Kouzani, and E. J. Hu. Random forest based lung nodule classification aided by clustering. *Computerized medical imaging and graphics*, 34(7):535–542, 2010. 26

[36] V. Lempitsky, M. Verhoek, J. A. Noble, and A. Blake. Random forest classification for automatic delineation of myocardium in real-time 3d echocardiography. In *International Conference on Functional Imaging and Modeling of the Heart*, pages 447–456. Springer, 2009. 26

[37] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1465–1479, 2006. 13, 25, 47, 48

[38] J. P. Lewis. Fast template matching. In *Vision interface*, volume 95, pages 15–19, 1995. 18, 32

[39] H. Li, K.-M. Lam, and M. Wang. Image super-resolution via feature-augmented random forest. *arXiv preprint arXiv:1712.05248*, 2017. 26

[40] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 19

[41] R. Maree, P. Geurts, J. Piater, and L. Wehenkel. Random subwindows for robust image classification. In *Computer Vision and Pattern Recognition,*

2005. CVPR 2005. IEEE Computer Society Conference on, volume 1, pages 34–40. IEEE, 2005. 25

[42] J. Marin, D. Vázquez, A. M. López, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2592–2599, 2013. 26

[43] H. Mohn, M. Gaebelein, R. Hänsch, and O. Hellwich. Towards image colorization with random forests. In *VISIGRAPP (4: VISAPP)*, pages 270–278, 2018. 26

[44] D. Mohr and G. Zachmann. Continuous edge gradient-based template matching for articulated objects. In *VISAPP (2)*, pages 519–524, 2009. 23

[45] D. Mohr and G. Zachmann. Silhouette area based similarity measure for template matching in constant time. In *International Conference on Articulated Motion and Deformable Objects*, pages 43–54. Springer, 2010. 24

[46] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Advances in neural information processing systems*, pages 985–992, 2007. 26

[47] A. Mordvintsev and K. Abid. Opencv-python tutorials documentation. *Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals. pdf*, 2014. 19, 20, 21, 41

[48] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009. 21

[49] M. Muja and D. G. Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV)*, pages 404–410, 2012. 21

[50] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014. 21

[51] E. Nowak and F. Jurie. Learning visual similarity measures for comparing never seen objects. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 27

[52] W. Ouyang, F. Tombari, S. Mattoccia, L. Di Stefano, and W.-K. Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE transactions on pattern analysis and machine intelligence*, 34(1):127–143, 2012. 23

[53] A. Pemasiri, K. Nguyen, S. Sridharan, and C. Fookes. Sparse over-complete patch matching. *arXiv preprint arXiv:1806.03556*, 2018. 24

[54] M. B. Pouyan and D. Kostka. Random forest based similarity learning for single cell rna sequencing data. *bioRxiv*, page 258699, 2018. 27

[55] L. Puggini, J. Doyle, and S. McLoone. Fault detection using random forest similarity distance. *IFAC-PapersOnLine*, 48(21):583–588, 2015. 28

[56] Y. Qi, J. Klein-Seetharaman, and Z. Bar-Joseph. Random forest similarity for protein-protein interaction prediction from multiple sources. In *Biocomputing 2005*, pages 531–542. World Scientific, 2005. 27, 28, 29

[57] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 16

[58] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014. 16

[59] M. Ristin-Kaufmann. *Large-Scale Image Recognition with Random Forests*. PhD thesis, ETH Zurich, 2015. 25

[60] A. Rosenfeld. Coarse-fine template matching. *IEEE Trans. Syst., Man & Cybern.*, 2(2):104–107, 1977. 32

[61] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006. 20

[62] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011. 21

[63] A. I. Salhi, M. Kardouchi, and N. Belacel. Fast and efficient face recognition system using random forest and histograms of oriented gradients. In *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG-Proceedings of the International Conference of the*, pages 1–11. IEEE, 2012. 26

[64] F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. In *BMVC*, pages 1–10, 2008. 25

[65] T. Sharp. Implementing decision trees and forests on a gpu. In *European conference on computer vision*, pages 595–608. Springer, 2008. 13

[66] T. Shi and S. Horvath. Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics*, 15(1):118–138, 2006. 29

[67] T. Shi, D. Seligson, A. S. Belldegrun, A. Palotie, and S. Horvath. Tumor classification by tissue microarray profiling: random forest clustering applied to renal cell carcinoma. *Modern Pathology*, 18(4):547, 2005. 29

[68] B. G. Shin, S.-Y. Park, and J. J. Lee. Fast and robust template matching algorithm in noisy image. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pages 6–9. IEEE, 2007. 23

[69] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts

from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. Ieee, 2011. 25

[70] A. Sibiryakov. Fast and high-performance template matching method. 2011. 23

[71] J. Tao and R. Klette. Integrated pedestrian and direction classification using a random decision forest. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 230–237, 2013. 26

[72] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991. 31

[73] M. Uenohara and T. Kanade. Use of fourier and karhunen-loeve decomposition for fast pattern matching with a large set of templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):891–898, 1997. 18, 32

[74] D. Vernon. Machine vision-automated visual inspection and robot vision. *NASA STI/Recon Technical Report A*, 92, 1991. 18, 31

[75] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001. 18, 31, 33, 34

[76] J. Winn and J. Shotton. The layout consistent random field for recognizing and segmenting partially occluded objects. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 37–44. IEEE, 2006. 25

[77] A. Yao, J. Gall, C. Leistner, and L. Van Gool. Interactive object detection. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3242–3249. IEEE, 2012. 25

[78] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, 2015. 24

[79] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1592–1599, 2015. 24

[80] Q. Zhou, W. Hong, L. Luo, and F. Yang. Gene selection using random forest and proximity differences criterion on dna microarray data. *Journal of Convergence Information Technology*, 5(6):161–170, 2010. 28, 29