Universiteit Leiden

Opleiding Informatica

Local Feature Detection using Neural Networks

Name:	Umut Özaydın
Date:	17/08/2018
1st supervisor:	Dr. Michael Lew
2nd supervisor:	Dr. Erwin Bakker

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Feature detectors and descriptors have been successfully used for various computer vision tasks, such as image stitching and content-based image retrieval. Many methods use image gradients in different stages of the detection-description pipeline to describe local image structures. Recently, some of these stages have been replaced by convolutional neural networks, in order to increase their performance. Replacing orientation estimators and descriptors is a relatively straightforward task, mapping a single image patch to a single output. A detector, however, is defined as a selection problem, which makes it more challenging to implement as a neural network. They are therefore generally defined as regressors, converting input images to score maps. Keypoints can be then be selected from this map by applying non-maximum suppression. This thesis discusses several methods that use neural networks and defines different metrics to determine their performance. Experiments are performed on a selection of conventional methods, as well as a detector network that is trained from scratch, a pre-trained image classifier that is used to generate feature descriptors and a method that replaces the complete pipeline with three separate networks. In addition to qualitative measures defined on keypoints and descriptors, the bag-of-words model is used to implement an image retrieval application, in order to determine how the methods perform in practice. The results show that, in some cases, neural networks can outperform traditional methods.

Contents

1 Introduction **5**

2 Local Feature Detection and Extraction **6**
- 2.1 History . . . 6
- 2.2 Conventional Feature Detectors and Extractors . . . 8
  - 2.2.1 SIFT . . . 8
  - 2.2.2 SURF . . . 9
  - 2.2.3 CenSurE . . . 11
  - 2.2.4 AGAST . . . 12

3 Neural Networks **13**
- 3.1 History . . . 13
- 3.2 Fully-Connected Neural Network . . . 14
- 3.3 Backpropagation . . . 15
- 3.4 Convolutional Neural Network . . . 17
  - 3.4.1 Convolutional Layer . . . 17
  - 3.4.2 Activation . . . 17
  - 3.4.3 Pooling . . . 18
  - 3.4.4 Fully-Connected Layer as Convolutional Layer . . . 18
- 3.5 Optimisation . . . 18
- 3.6 Regularisation . . . 20
- 3.7 Normalisation . . . 21

4 Learning Feature Detectors and Descriptors **23**
- 4.1 TILDE . . . 23
- 4.2 Covariant Feature Detector . . . 24
  - 4.2.1 Network Output . . . 24
  - 4.2.2 Learning . . . 24
- 4.3 Quad Detector . . . 26
  - 4.3.1 Learning . . . 26
  - 4.3.2 Data Generation . . . 26
- 4.4 Orientation Estimator Network . . . 27
- 4.5 LIFT . . . 27
  - 4.5.1 Training . . . 27
  - 4.5.2 Descriptor . . . 28
  - 4.5.3 Orientation Estimator . . . 28
  - 4.5.4 Detector . . . 29

5 Experiments **30**
- 5.1 Repeatability . . . 30
- 5.2 Descriptor Matching . . . 32
- 5.3 Transformations . . . 32
  - 5.3.1 Rotation . . . 33
  - 5.3.2 Scale . . . 33
  - 5.3.3 Blur and Noise . . . 34
  - 5.3.4 Compound . . . 34
- 5.4 Non-Redundancy & Coverage . . . 34
- 5.5 Image Retrieval . . . 35
- 5.6 Data Sets . . . 36
- 5.7 Implementation Details . . . 37
  - 5.7.1 Conventional Feature Detectors & Descriptors . . . 37
  - 5.7.2 The Quad Detector . . . 37

5.7.3 LIFT . . . 38
5.7.4 The Covariant Detector . . . 38
5.7.5 VGG16 Descriptors . . . 38
5.7.6 Feature Cardinality . . . 39

6 Results **40**
6.1 Repeatability and Matching . . . 40
6.2 Non-Redundancy and Coverage . . . 47
6.3 VGG16 . . . 49
6.4 Image Retrieval . . . 50

7 Discussion **52**

1 Introduction

As billions of images and millions of hours of video are uploaded on the internet every year, and the amounts are ever increasing, the ability to decode their contents is a challenge for computers that has great value. There are numerous domain and application areas which over the past decade have required content based analysis of imagery. These include, but are not limited to, applications in face recognition [69], human activity recognition and human-computer interaction (HCI) [15, 28, 53, 59], medical image understanding [22] and image and video retrieval [15,29,52,57]. However, taking raw pixel data and extracting high-level information that's useful is a non-trivial task and has been the topic of a lot of research, almost as long as computers have been around. One method is the detection and extraction of salient features, which can be used in a wide variety of computer vision tasks, such as object classification, image retrieval and image stitching. Many methods to detect and extract features have been proposed and a lot of papers have been published over the past decades describing feature detectors and descriptors, some of which are optimised for a specific domain, while others are intended for general application. One of the most popular methods, still today, is SIFT [31], which was first introduced in 1999. SIFT uses an approximation of the *Laplacian*, which describes gradients in the image. Many methods approximate image gradients, as directly calculating them is an expensive operation and the added performance of calculating exact gradients is usually not worth the extra processing time. Other methods work by directly comparing pixel intensities and do not approximate gradients, which generally means an even higher computational efficiency, but a loss of performance, as higher levels of information are unused. These methods may be preferable in online applications that run multiple times per second and low computational time is critical.

In recent years, researchers have proposed using *neural networks* for detecting and extracting features. Neural networks have been around for some time and they have been successfully used to solve complex problems, such as image classification or speech recognition, but their application to feature detection and description is relatively new. These networks learn by processing large amounts of data, which in the case of visual media should not be a problem. However, the learning process can generally be divided into two classes. The most common class is *supervised* learning, which means that the input samples are coupled with *labels*, which are the expected or desired outputs. These can be used determine the performance of the network and change its parameters to increase this performance. In some cases the labels may need to be generated by hand, which costs a lot of time and effort and makes useful data more scarce. In the case of feature detection, labels could also be calculated, but doing so may build a bias into the network by preferring certain features above others. On the other hand, when learning is done without knowing the explicit outputs, it is called *unsupervised* learning. This, however, makes the task more challenging and requires a specifically designed algorithm for learning. The goal of this thesis is to review how neural networks can be used for feature detection and extraction and compare the performance of the resulting networks to the classical detectors and extractors.

To get an idea of the problem domain, in Section 2 local feature detection and extraction and a couple of successful algorithms are discussed and Section 3 gives a brief overview of neural networks. Then, in Section 4 these two fields are combined and methods that use neural networks to detect features are examined. Section 5 describes the techniques to evaluate the performance of the detectors, which are then used in the experiments to do the performance evaluation. Finally, the results of the experiments are presented in Section 6 and discussed in Section 7.

2 Local Feature Detection and Extraction

2.1 History

Images contain a lot of information and their contents can greatly vary, which makes understanding them a challenging task for computers. *Local feature detectors* and *extractors* arose from the need to computationally process images and were a means of separating the useful parts from the noise. Naturally, it's much easier to work with a simplified representation of the data than it is to use the raw pixel values. The idea is not to process images *globally*, in their entirety, but to look for certain structures at the *local* scale. Some local structures may be distinct and provide useful information, while others are less significant. The goal of local feature detection is to look for the types of structures that are distinguishable and can be precisely localised in sets of images. Some of the earliest structures that were defined to be useful are *edges* and *corners*, where corners are defined as the intersection of two edges.

One of the first corner detectors, defined by Moravec [38] in 1977, moves a sliding window over the image in several directions to determine which directions result in a significant change. The change in intensity E, given a shift (u, v), is defined as

$$E(u,v) = \sum_{(x,y)} w_{(x,y)} |I_{(x+u,y+v)} - I_{(x,y)}|^2, \tag{1}$$

where $w_{(x,y)}$ is the sliding window, $I_{(x,y)}$ is the image intensity at location (x, y) and the shifts (u, v) consist of horizontal, vertical and diagonal movements. Along an edge, the change in intensity will be small, while it is large across it. For a corner, all directions will yield a significant change in intensity, whereas a flat patch will approximately stay the same, no matter which direction the window moves. Corners are found by means of non-maximum suppression, at the location that gives the highest rate of change, or the local maximum. Some of the drawbacks of this method are that it gives many responses along edges and, since it considers changes in direction at steps of $45°$, the results are dependent on the orientations of the edges.

Harris and Stephens improved Moravec's corner detector in 1988 [13] by addressing these issues. To calculate the changes in pixel intensity over a patch, the image gradients are approximated using Taylor expansion. The gradients are then rewritten into matrix form, such that the change in intensity E is defined as

$$E(u,v) = (u,v)M(u,v)^T, \tag{2}$$

where M is the gradient matrix. Eigenvalue decomposition can then be performed on M and, since the eigenvalues are independent of the coordinate system, the direction of the corner does not affect the result and the detector is rotationally invariant. The Harris detector also aims to eliminate edge responses by using the eigenvalues to calculate a response

$$R = det(M) - k \cdot tr(M)^2, \tag{3}$$

where k is some constant and det and tr are the determinant and the trace of matrix M. The patch is considered a corner when the response has a large positive value. Finally, noise is reduced by applying a circular Gaussian window, instead of a rectangular one, which makes the Harris corner detector isotropic.

Another method similar to the Harris corner detector is the one introduced in 1978 by Beaudet [4]. It uses the Hessian matrix

$$H(x,y) = \begin{bmatrix} \dfrac{\partial^2 I(x,y)}{\partial x^2} & \dfrac{\partial^2 I(x,y)}{\partial x \partial y} \\ \dfrac{\partial^2 I(x,y)}{\partial x \partial y} & \dfrac{\partial^2 I(x,y)}{\partial y^2} \end{bmatrix}, \tag{4}$$

which contains the second-order partial derivatives of the image. As in the Harris detector, the second-order derivatives are calculated with Taylor expansion. Interest points are found by applying non-maximum suppression to the determinant of the Hessian matrix, which is why it's also called the Determinant of Hessian (DoH) detector.

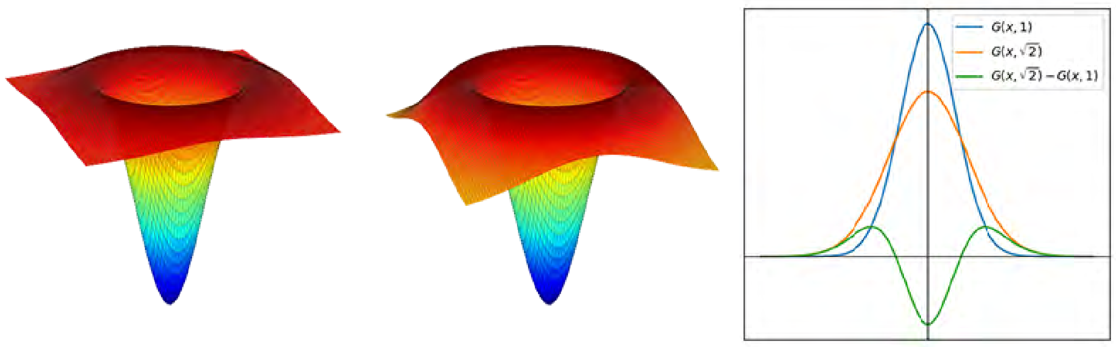


Figure 1: The the Laplacian of Gaussian (LoG) function is depicted on the left. Its computationally more efficient approximation difference of Gaussian (DoG) is shown in the middle. The graph on the right shows how the DoG's shape is derived by subtracting 2 Gaussians with a different standard deviation σ.

While the Harris and Hessian detectors are rotationally invariant, they are not able to assign a scale to the interest points, which puts some constraints on matching. Witkin [65] and Koenderink [20] proposed the use of a *scale-space* to search over small and large scales by applying varying degrees of Gaussian blur to the image to remove details at the finer levels. In 1998 Lindeberg defined the Laplacian of Gaussian (LoG) [30], a *blob* detector that implements the scale-space. Images are incrementally convolved with the Gaussian kernel and for each blurred image, the Laplacian is calculated. The Laplacian is the trace (the sum of the diagonal) of the Hessian matrix, which contains the image derivatives of the second order. Non-maximum suppression is not only applied per image, but also over the scale-space, in order to determine the scale of each interest point.

In the same year Lowe proposed to approximate the LoG with the *difference of Gaussian* (DoG) [31]. In his method called SIFT, the scale-space is built by incrementally convolving the input image by the Gaussian kernel. By subtracting two images with two different amounts of Gaussian blur applied, the LoG function is approximated. This approximation is illustrated in Figure 1. Since images are already blurred for the scale-space, interest points can be found only by a simple subtraction and non-maximum suppression, which is computationally efficient. Besides a feature detector, SIFT also includes an orientation assignment and a feature extractor.

In 2006 Bay et al. defined a feature detector and extractor based on the Hessian matrix. Their method, called Speeded-Up Robust Features (SURF) [3], aims to speed up the process by using box-filters and integral images. The idea of finding features, however, is similar to many other methods and is done by looking for points that have a large gradient in two perpendicular directions. Scale-invariance is achieved by applying different sizes of box-filters. An orientation assigner and a special SURF descriptor are defined as well.

Since the interest points of many feature detectors have a position, a scale and an orientation, the resulting features are defined in an oriented circular region. However, in the case of wide-baseline stereo, images are taken under a large change in viewpoint. Therefore, interest points defined as circles are not descriptive enough. In the early 2000s a new set of *affine invariant* detectors were defined, some of which are the Harris Affine and Hessian Affine detectors [34] and Maximally Stable Extremal Regions (MSER) [33]. These methods give interest points an elliptical shape by approximating the affine transformations in the regions of those points.

In contrast to all of the methods discussed so far, the method published in 2006 named Features from Accelerated Segment Test (FAST) by Rosten and Drummond [45], is not based on image derivatives. However, corners are found by direct pixel comparisons. The intensities of pixels in a circular region are compared to that of the centre pixel and a point is said to be a corner if the circle contains at least n consecutive pixels that are all lighter or all darker than the centre. This is a very intuitive definition of a corner. FAST has a relatively high computational performance, since a couple of pixel comparisons is less expensive than gradient approximation. It is further increased by learning a good order to evaluate the pixels. Its derivatives, FAST-ER [46] and AGAST [32], aim to improve it, but use the same principle.

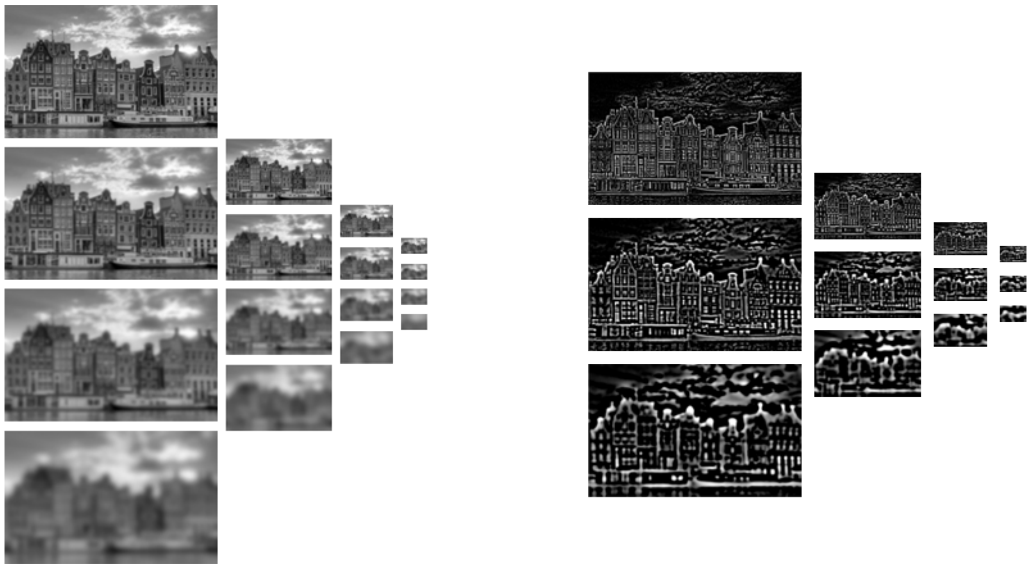


Figure 2: On the left, an example of the SIFT scale-space with 4 octaves and 4 scales per octave. Each octave is down-sampled to half the size and each scale is blurred progressively with a constant factor. In practice there are 6 scales per octave and the number of octaves depends on the image dimensions. On the right, the resulting difference of Gaussians. Since each comparison needs 2 neighbours, only 1 scale per octave will yield interest points.

2.2 Conventional Feature Detectors and Extractors

In general, the feature matching pipeline consists of several stages, namely *feature detection*, *feature extraction* and *feature matching*. Feature detectors select locations from images according to some metric. The detected features consist of at least a position, but may include a scale, an orientation, and even a more complex shape. For each of these detected features, descriptors are calculated to simplify the data contained in the detected region. It is then easier for a matcher to compare these descriptors, as their size is relatively small and constant. In the next few sections several local feature detectors and extractors are discussed. Over the years many detectors and descriptors have been defined with varying approaches and performances. The application for which the features are to be used might influence the selection of a method. Some applications might require a very high stability of features, whereas for others the time performance may be most crucial and approximating the gradients may be too expensive.

2.2.1 SIFT

In order to find keypoints, the *Scale Invariant Feature Transform* (SIFT) [31] uses the difference of Gaussian, which is an approximation to the Laplacian of Gaussian (LoG). Figure 1 shows the LoG and DoG functions, and functions like these have a high response when they are centred on blobs, which is the type of structure mainly targeted by SIFT. What makes SIFT scale-invariant is that a scale is assigned to each interest point and these scales are calculated by utilising the scale-space. SIFT's scale-space has a pyramid-like structure that is made up of octaves and scales. In an octave, every image is incrementally blurred, which removes details from the image and at the same time increases the size of the approximated LoG function. For the first octave, the image is doubled in size, using linear interpolation, and for each following octave the image is sub-sampled to half the size of the previous octave. The first scale in each octave is subjected to prior smoothing and each successive scale is blurred with a constant factor $k = \sqrt{2}$. A visual representation of the scale-space can be seen in Figure 2. The DoG is calculated as

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y), \tag{5}$$

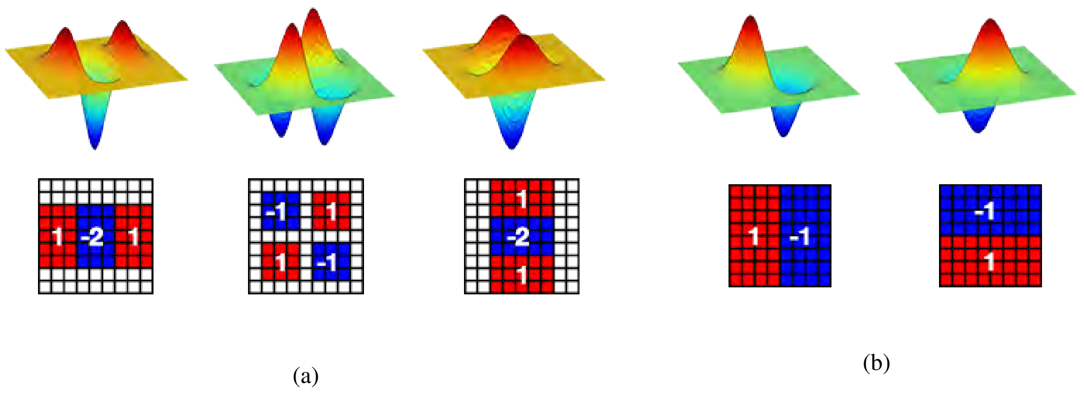


Figure 3: (a) The top row shows the 2D second order Gaussian derivatives $\frac{\partial^2 G(x,y,\sigma)}{\partial x^2}$, $\frac{\partial^2 G(x,y,\sigma)}{\partial xy}$ and $\frac{\partial^2 G(x,y,\sigma)}{\partial y^2}$, respectively. On the bottom the corresponding 9×9 box filters can be seen, which are discretised and cropped and approximate the Gaussian with $\sigma = 1.2$. (b) First order derivatives of the 2D Gaussian in the horizontal direction $\frac{\partial G(x,y,\sigma)}{\partial x}$ and the vertical direction $\frac{\partial G(x,y,\sigma)}{\partial y}$. The bottom shows corresponding box filters of size 8×8.

where $G(x, y, \sigma)$ is the Gaussian kernel with scale σ, $I(x, y)$ is the image and $*$ is the convolution operator. This means that the DoG is calculated for all pairs of neighbouring images in each octave.

From the resulting pyramid of DoG images, interest points are selected by applying non-maximum suppression in and across scales. Every point in the DoG images is compared to its 26 neighbours, 8 of which are in the same scale and 9 in each neighbouring scale. Only those points that are all smaller or all larger than every neighbour pass the first test. To improve the results, interest points are localised by fitting a 3-dimensional function centred at the point and calculating the exact location of the extremum by equating the derivative of the function to 0. The derivatives are approximated with Taylor expansion to increase performance. Finally, points with low contrast are removed and edge responses are eliminated by thresholding the ratio between the eigenvalues of the Hessian at the interest point, similar to the Harris and Hessian corner detectors.

In order to assign orientations to keypoints, a patch centred at the point is extracted at the keypoint's scale. Image gradients are then calculated for every point in the Gaussian-weighted patch. The gradients are added into 36 orientation bins, giving a window of $10°$ per bin. The bin with the largest value is then used to calculate the orientation, which is accurately localised by interpolation to find the peak. An additional keypoint is added for every bin with a value within 80% of the maximum, that has the same location, but a different orientation.

The last step is extracting a descriptor for every keypoint. The descriptors are calculated on patches centred at the keypoint, given the keypoints scale and rotated according to the orientation assigned in the previous step. A Gaussian-weighted patch is divided into a grid of 16×16, and, similar to the orientation assignment step, gradients are calculated and accumulated into 8 bins in a grid of 4×4. This results in a $4 \times 4 \times 8 = 128$ dimensional descriptor.

2.2.2 SURF

Speeded-Up Robust Features (SURF) [3] is based on the Hessian matrix $H(x, y, \sigma)$, which is defined as

$$H(x,y,\sigma) = \begin{bmatrix} \frac{\partial^2 L(x,y,\sigma)}{\partial x^2} & \frac{\partial^2 L(x,y,\sigma)}{\partial x \partial y} \\ \frac{\partial^2 L(x,y,\sigma)}{\partial x \partial y} & \frac{\partial^2 L(x,y,\sigma)}{\partial y^2} \end{bmatrix}, \tag{6}$$

where $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$ is the image convolved with the Gaussian kernel with scale σ and centred at point (x, y). Since calculating second-order derivatives is expensive, they are approximated with box filters, which are 2D Haar wavelets. The Gaussian second order partial derivatives and their corresponding box filters can be seen in Figure 3. SURF searches over the scale-space by using box filters of variable size. This means that the image does not need to be sub-sampled, which increases performance. A big advantage of using box filters is that they can be calculated very efficiently by using integral images [61]. In an integral image I_Σ of an image I, the value at a pixel (x, y) is calculated by

$$I_\Sigma(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j). \tag{7}$$

In other words, an integral image is the cumulative sum of the image and each pixel is the sum of itself and all pixels preceding it. Calculating a rectangular region of any size then becomes the trivial task of adding and subtracting four values.

The SURF scale-space is, just like SIFT's, divided into octaves and scales, where each scale uses a slightly larger filter and each octave doubles the filter size of the previous octave. The determinant of the Hessian is approximated by

$$det(H) \approx D_{xx}D_{yy} - (wD_{xy})^2, \tag{8}$$

where D_{xx}, D_{xy} and D_{yy} are the responses of the box filters and $w = 0.9$ is a weight to balance the terms. The points in the responses are then compared to their 26 neighbours in the same and neighbouring scales and only extrema are selected. Finally, the interest points are interpolated for exact localisation.

For orientation assignment, SURF approximates gradients in an area around each keypoint by applying horizontal and vertical Haar wavelets that correspond to the first-order derivatives of the Gaussian, which can be seen in Figure 3b. These computations are efficient, as they can also be done with the integral images. The responses of the horizontal and vertical Haar wavelets are interpreted as x- and y-coordinates, respectively, centred around the keypoint centre. The sum of the responses is then calculated in a sliding window of $60°$ (see image 4). The orientation of the keypoint is set to the direction of the sliding window that resulted in the largest sum of responses.

As Haar wavelets are the foundation of the SURF algorithm, they are used to calculate the descriptors as well. A region around the keypoint is selected according to its scale, rotated by the orientation assigned in the previous step and weighted by a Gaussian window in order to increase robustness. The region is then divided into a grid of 4×4 sub-regions and for each sub-region, horizontal and vertical Haar wavelets are applied in a grid of 5×5. This results in 25 horizontal and 25 vertical gradient values per sub-region. A descriptor of size 64 is then constructed by calculating the values $(\sum d_x, \sum |d_x|, \sum d_y, \sum |d_y|)$ for each of the 16 sub-region, where d_x and d_y are the horizontal and vertical gradients, respectively. The sum of the gradients describes the general direction of the gradients in the sub-region, while the sum of the absolute values of the gradients describes the gradients' strength.

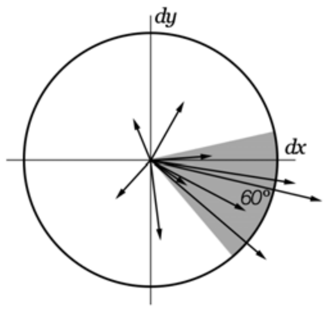


Figure 4: For the orientation, SURF slides a window of $60°$ around the centre of the keypoint, while accumulating the responses of horizontal and vertical Haar wavelets. The orientation is determined to be the position in which the sum of the responses is the largest.

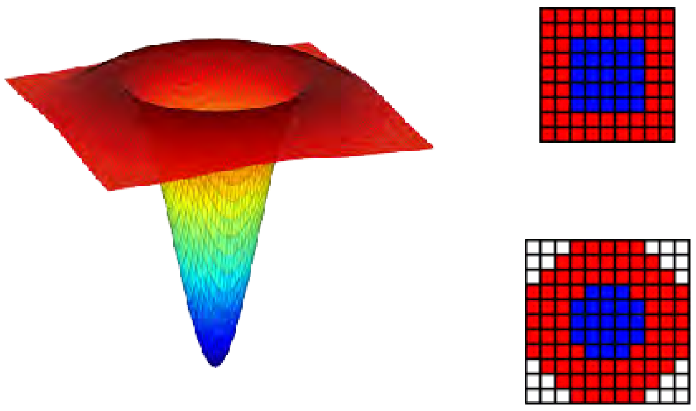


Figure 5: CenSurE uses the Laplacian of Gaussian, shown here on the left. The box and octagon filters on the right approximate the Laplacian with $\sigma = 1.885$.

2.2.3 CenSurE

CenSurE [2] is a feature detector that uses bi-level centre-surround filters, which approximate the Laplacian of Gaussian. A good approximation of the Laplacian would be a circular bi-level filter. However, calculating a circular filter is computationally expensive, especially at the larger scales. On the other hand, using a square-shaped filter, or *difference of boxes* (DOB), results in a loss of rotational invariance. SURF also uses box filters and suffers from a decrease in repeatability at odd multiples of $45°$ of rotation. CenSurE aims to find the middle ground of using circular filters to get a high accuracy, and using box filters to get a high computational performance, by using octagonal and hexagonal filters. The difference between SURF and CenSurE is that SURF approximates each element of the Hessian matrix separately, while CenSurE approximates the LoG, which is the sum of the diagonal of the Hessian matrix. In that regard, CenSurE may be closer to SIFT, which also approximates the LoG. An example of the LoG and its approximations, the DOB and the octagonal filter, can be seen in Figure 5.

A high computational performance can be achieved by using slanted integral images, which calculate the sum over an area in which one of the sides is slanted. A slanted integral image is calculated as

$$I_{\Sigma_\alpha}(x, y) = \sum_{j=0}^{y} \sum_{i=0}^{x+\alpha(y-j)} I(i, j), \tag{9}$$

where α determines the slope of the slant. With these modified integral images, octagonal and hexagonal filters can be calculated in a constant time for all scales. However, by adding more sides to the filters to make them more circular, the number of computations needed is also increased. After the interest points are detected at all levels, only the extrema of all direct neighbours in the same and two adjacent scales are selected. So only those points that are higher than all or lower than all 26 neighbours pass the non-maximum suppression. Finally, the Harris measure at the interest points is used to calculate the ratio of principal curvatures to threshold out responses along edges. The CenSurE method does not include an orientation assignment and the resulting features do not have an orientation.

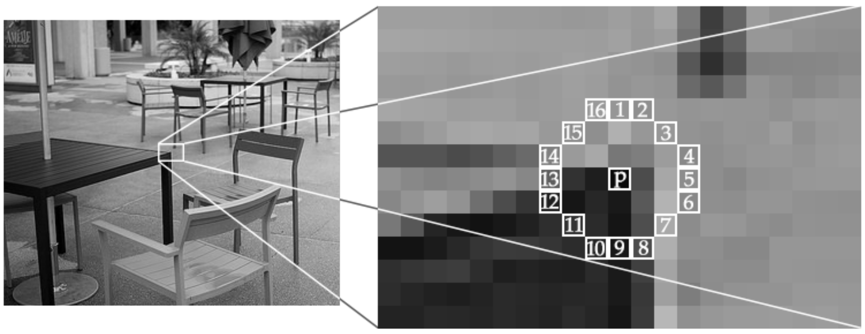


Figure 6: The accelerated segment test works by comparing 16 pixels in a circle around reference pixel $\mathbf{p}$ and determining whether at least n consecutive pixels are all lighter or all darker than $\mathbf{p}$ by some threshold.

2.2.4 AGAST

While methods that approximate gradients are much faster than those that calculate them, the improvement in speed may not be enough for some applications. In order to gain an even bigger speed-up, some methods propose to skip gradients altogether and directly compare pixel values. A range of methods use the so-called *Accelerated Segment Test* (AST), that, in order to determine whether a point $\mathbf{p}$ is a corner, compares its value to the 16 pixels in a circle around it, as shown in Figure 6. The point is said to be a corner if the circle contains n consecutive pixels that are all lighter or all darker than $\mathbf{p}$ by some threshold. The method *Features from Accelerated Segment Test* (FAST) [45] uses this principle to detect corners by determining for each pixel whether the centre pixel is lighter, darker or the same. FAST aims to minimise the amount of pixel value comparisons by using the ID3 machine learning algorithm [42]. Applying ID3 results in a decision tree, which dictates the order in which pixels are compared, depending on the previous comparison results. FAST, however, has some limitations, some of which are the fact that the decision tree has to be learned with a set of training images and the suboptimal decision tree that is known to be the result of the ID3 algorithm.

One of FAST's derivatives, *Adaptive and Generic Accelerated Segment Test* (AGAST) [32], was introduced to address these drawbacks. First of all, the decision tree is made binary instead of ternary, by breaking down each ternary question (whether a pixel in the circle is darker than, similar to or lighter than the centre pixel) into two binary questions. Second, two decision trees are built, one for homogeneous and one for heterogeneous areas, and the tree to be used is determined with *adaptive tree switching*. And finally, the decision tree is built by calculating a cost function for each node in a bottom-up manner. The cost of a leaf node is defined as 0, and for each internal node P the cost c_P is calculated by

$$c_P = \min_{C \in \{C_1,\ldots,C_n\}} c_{C_+} + p_{C_+} c_T + c_{C_-} + p_{C_-} c_T = c_{C_+} + c_{C_-} + p_P c_T, \tag{10}$$

where C_+ and C_- are the positive and negative test results for child node C corresponding to a pixel, p_{C_+}, p_{C_-} and p_P are the probabilities of the pixel configurations at the child and parent nodes and c_T is the cost of accessing the memory (register, cache or main memory, depending on the position of the pixel). The result is a two-part decision tree, where each leaf node leads to the appropriate part of the tree depending on the pixel configuration.

FAST and AGAST do not assign a scale to the found interest points. Since this is necessary to ensure scale invariance, the scale can be calculated as in ORB [47] (which uses FAST) or BRISK [27] (which uses AGAST). Both methods use a scale pyramid and apply non-maximum suppression on the corner measure, which is, like other methods, done by comparing it to 26 neighbours. ORB uses the Harris corner measure to rate the saliency of corners, whereas BRISK uses the FAST score function.

3 Neural Networks

The last decade has seen a lot of advancement in artificial intelligence (AI) and deep learning in particular is increasingly popular. It is a method that enables computers to learn to perform a specific task and the range of its applications has grown larger every year. Many big companies like Google, Microsoft and Apple have invested in the development of deep learning techniques and architectures, and these AI systems are used in the services and products they provide. Even though deep learning is a term that has been popularised in recent years, the neural networks that are used in deep learning have existed for several decades. Instead of programming a certain behaviour, which gets more unmanageable as behaviours get more complex, neural networks aim to learn tasks by taking in a lot of data and incrementally adjusting their parameters. They work similarly to the animal brain in that they transform data from input to output by making abstractions. They combine smaller elements into larger, more complex ones, which is made possible by the layered structure of the networks.

3.1 History

The first neural networks were developed before modern computers existed and one of the first implementations was made by Rosenblatt [44] in 1957 on hardware made especially for this purpose. He called this network a *perceptron*, which is a single-layer neural network with a binary output. The perceptron calculates a function by multiplying the inputs with weights and adding a bias. Given a set of transformed inputs, the perceptron can learn to discriminate between two classes of inputs by changing the weights with an update rule. In 1960 Widrow and Hoff [64] introduced the Adaptive Linear Neuron (ADALINE), which is similar to the perceptron, but as the name suggests, has a linear output, instead of a binary one. In 1969 Minsky and Papert [37] showed that neural networks with a single layer could not solve non-linear problems and didn't see a future in them. However, *multi-layer perceptrons* were able to solve more complex non-linear problems, but there was not yet a good method to learn them. In 1980 the *neocognitron* was introduced by Fukushima [11], which is a multi-layer neural network that can learn to recognise visual patterns and is the precursor to the convolutional neural network, discussed in Section 3.4. It was then suggested in the '70s and '80s that the weights of neural networks be updated with the use of the backpropagation algorithm by

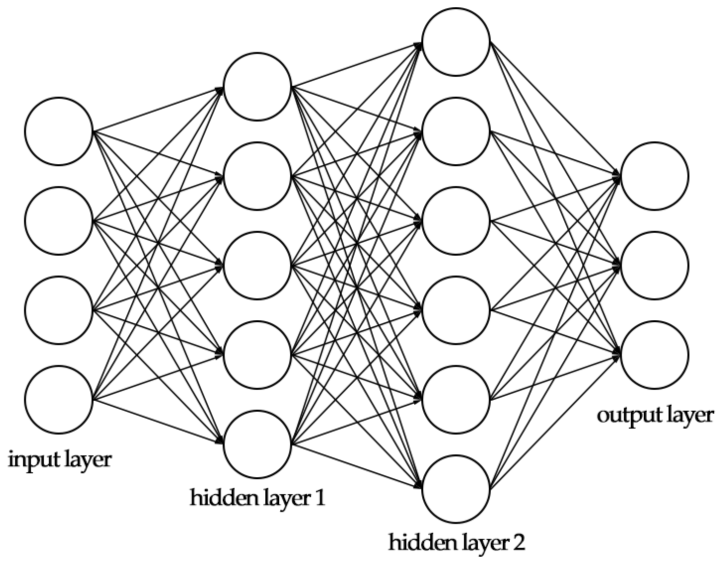


Figure 7: An example of a simple 3-layer neural network with 2 hidden layers. This is a fully-connected neural network, since all nodes in every layer are connected to all nodes in the neighbouring layers.

Rumelhart et al. [48] and others before them [23, 41, 63]. In their paper, Rumelhart et al. also proposed the use of a *momentum* term to speed up learning, which will be discussed in detail in Section 3.5. In the following years progress slowed down again, as people lost confidence in neural networks and didn't think large architectures were feasible. Backpropagation worked well for shallow networks, but not so much for deeper models, although many improvements were suggested. Convolutional neural networks were first defined in 1989 by LeCun et al. [24], named LeNet, and were applied to handwritten digit recognition. It was the fifth iteration, called LeNet-5 [25], that would be very successful. After another period of slow progress, Krizhevsky et al. designed AlexNet [21] in 2012 and won the ImageNet Large Scale Visual Recognition Competition (ILSVRC) of that year. It was also designed to be run on the GPU, which meant much faster learning as computations could be highly parallelised. This was when neural networks really took off and the application domain started to grow even larger. Neural networks, now often referred to as *deep learning architectures*, have even exceeded human performance in certain tasks.

3.2 Fully-Connected Neural Network

A neural network is made up of *neurons* that are organised in layers, the first of which is called the *input layer*, and the last the *output layer*. In general, the input layer is not actually considered a layer, since it does not perform any calculation, but simply provides the first real layer with input. A neural network with an input and output layer is therefore called a single-layer network. In between the input and output layers there may be zero or more layers that are named *hidden layers*. Neurons in consecutive layers may or may not be connected and generally there exist no connections between neurons in layers that are not direct neighbours, although there are some exceptions. There is a weight associated with every connection that determines how much the neurons influence each other. A neuron takes as input a subset of the outputs of the neurons in the previous layer. How inputs are propagated through the network and transformed by each layer will now be discussed. The following definitions are inspired by the ones presented in [6].

There are different types of layers, but the most basic type is the *fully-connected* (FC) layer, also called the *dense* layer, in which every neuron in one layer is connected to every neuron in the next. A network that consists solely of fully-connected layers is called a *fully-connected neural network* or *dense neural network*. An example of such a network can be seen in Figure 7. A neuron n in layer l has for every input $x_1^l, x_2^l, \ldots, x_M^l$ a set of associated weights $w_{n1}^l, w_{n2}^l, \ldots, w_{nM}^l$ and calculates the so-called *pre-activation* as

$$z_n^l = w_{n1}^l x_1^l + w_{n2}^l x_2^l + \cdots + w_{nM}^l x_M^l + b_n^l, \tag{11}$$

where b_n^l is the bias of neuron n in layer l. The bias is a learnable parameter and its goal is to add a value to increase or decrease the input to the activation function, which changes the decision boundary and adds complexity to the neuron. The value of a bias is always 1, but the weight can be learned, just like the weights of other neurons. The pre-activation is then subjected to an *activation function*, or *transfer function*, f, and the output of the neuron, the *activation*, becomes

$$a_n^l = f(z_n^l) = f(w_{n1}^l x_1^l + w_{n2}^l x_2^l + \cdots + w_{nM}^l x_M^l + b_n^l). \tag{12}$$

The goal of the activation function is to introduce nonlinearity into the network, which enables the network to learn complex problems. There are many different types of activation functions, such as the sigmoid, the hyperbolic tangent (tanh) and the rectified linear unit (ReLU), which are depicted in Figure 8. New activation functions may yet be proposed, as this is still an active field of research.

The representation of neurons as nodes and connections helps us intuitively understand how neural networks function, as they were inspired by biology. However, in practice inputs, weights and biases are represented by matrices and vectors. The inputs and biases of layer l can be seen as vectors

$$\mathbf{x}^l = \begin{bmatrix} x_1^l \\ x_2^l \\ \vdots \\ x_M^l \end{bmatrix}, \ \mathbf{b}^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_N^l \end{bmatrix}. \tag{13}$$

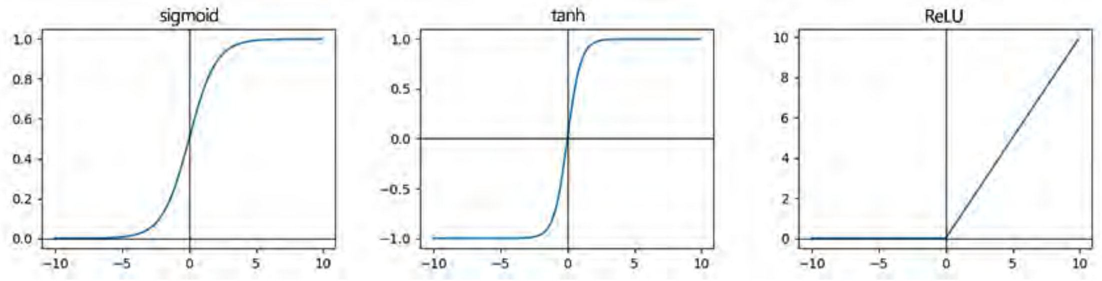


Figure 8: Three different activation functions. The most commonly used in convolutional neural networks is the ReLU activation function.

The weights of a layer l are represented by a matrix

$$\mathbf{W}^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1M}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2M}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^l & w_{N2}^l & \cdots & w_{NM}^l \end{bmatrix}, \tag{14}$$

where every column is the set of weights associated with one input element and every row is the set of weights of one neuron. The output of a layer can now be written as the matrix multiplication

$$\mathbf{a}^l = f(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l). \tag{15}$$

In practice, multiple inputs can be calculated in one pass by also representing the inputs as a matrix, where every column is an input instance. Since the values are propagated from one layer to the next, the input to a layer is the activation of the previous layer and $\mathbf{x}^l = \mathbf{a}^{l-1}$.

As an example, let's consider a two-layer network with a single hidden layer that calculates a function $\mathcal{N}(\mathbf{X})$. The input to the network is the matrix $\mathbf{X}$ with multiple input instances, both layers have weights $\mathbf{W}_i$ and biases and $\mathbf{b}_i$ and each layer is followed by an activation function, which may be different for each layer. The output of the network can then be calculated by

$$\mathcal{N}(\mathbf{X}) = \mathbf{A}^2 = f^2(\mathbf{W}^2 f^1(\mathbf{W}^1 \mathbf{X} + \mathbf{b}^1) + \mathbf{b}^2). \tag{16}$$

The network outputs a matrix $\mathbf{A}$ that contains the output for every input instance. This is the forward pass and is simply a chain of matrix multiplications, additions of biases and element-wise applications of activation functions.

3.3 Backpropagation

In order to function properly, a network has to have the appropriate weights. In general, weights are initialised randomly and are updated in incremental steps by using some set of *update rules*. As previously mentioned, the most frequently and successfully used set of weight update rules is the backpropagation algorithm, although other update methods exist, such as genetic algorithms. In order to learn, a *loss function*, also called a *cost* or *objective function*, is defined. The loss determines how well the networks performs and can be used to modify the weights in such a way that it decreases. One of the simplest loss functions is the *Mean Squared Error* (MSE), which is known as

$$\mathcal{L}_{MSE} = \frac{1}{S}\sum_{s=1}^{S} ||\mathbf{y}_s - \mathbf{a}_s||^2, \tag{17}$$

where $\mathbf{y}_s$ is the expected output of input sample s and $\mathbf{a}_s$ the network's activation. There are different loss functions that can be used and, since backpropagation uses derivatives in order to minimise, an important property that the loss function must possess is differentiability. Unsupervised methods do not have expected values and therefore the loss function must be defined in some other way, and this will be discussed in Section 4.

The idea of backpropagation is to use the derivative of the loss function to take small steps to incrementally decrease it. The derivative is calculated with respect to the weights and biases, so the search-space is highly multidimensional. The algorithm is done in two steps, the forward pass and the backward pass. In the forward pass the outputs of the neurons of all layers are computed, as explained in the previous section. In the backward pass these values are used to compute the gradients at each neuron, starting with the output layer and working its way back, and the gradients are then used to update the weights to minimise the loss function. To train properly, a large data set is needed, which is divided into *training set*, *validation set* and *test set*. The training set is used to calculate the gradients and update the weights, while the validation set is used to compute the performance of the network during training and adjusting hyperparameters. This is done because the performance on the training set may not be representative of the overall performance and the network may be overfitted. The test set is used after training to evaluate the network on unused data. Backpropagation is an iterative method and training is performed for several *epochs*, until some requirements are met. In each epoch, the network learns on the whole training set, and the hyperparameters are adjusted at the end of the epoch.

The most basic variant of backpropagation is *gradient descent* (GD), or more commonly *stochastic gradient descent* (SGD). SGD updates the parameters after each sample in the training set, or per mini-batch of samples, whereas the standard *batch gradient descent* accumulates gradients and updates the parameters once every epoch for the whole data set. Every weight w_{nm} of node n from input m is updated into the new value $\hat{w}_{nm}$ according to the update rule

$$\hat{w}_{nm} = w_{nm} - \alpha \frac{\partial \mathcal{L}}{\partial w_{nm}}, \tag{18}$$

where α is the learning rate, $\mathcal{L}$ is the loss function and $\frac{\partial \mathcal{L}}{\partial w_{nm}}$ is the partial derivative of the loss function with respect to the weight w_{nm}. This partial derivative gives the direction of steepest ascent for this parameter and it is therefore subtracted from the original value, to decrease the value of the loss function. Note that the same definition applies to biases as well, since a bias is nothing more than a weight with input value -1. The updates start at the final layer, because the output of this layer is directly correlated to the output of the loss function. Since the loss function is a summation over multiple input samples, the derivative also contains a summation. However, in the derivative of the loss function over multiple input instances

$$\frac{\partial \mathcal{L}}{\partial w_{nm}} = \frac{1}{S} \sum_{s=1}^{S} \frac{\partial \mathcal{L}_s}{\partial w_{nm}} \tag{19}$$

the summation and mean will be dropped, since this simplifies the notation. The derivative of the whole can simply be calculated by taking the mean of each individual derivative.

To calculate the partial derivative of the loss function with respect to a single weight of node n in the output layer, the chain rule is applied successively as

$$\frac{\partial \mathcal{L}}{\partial w_{nm}} = \frac{\partial \mathcal{L}}{\partial z_n} \frac{\partial z_n}{\partial w_{nm}} = \frac{\partial \mathcal{L}}{\partial a_n} \frac{\partial a_n}{\partial z_n} \frac{\partial z_n}{\partial w_{nm}}. \tag{20}$$

Each of the components can be rewritten as

$$\frac{\partial z_n}{\partial w_{nm}} = \frac{\partial}{\partial w_{nm}} \left(\sum_{\mu=1}^{M} (w_{n\mu} x_\mu) + b_n \right) = x_m, \tag{21}$$

$$\frac{\partial a_n}{\partial z_n} = \frac{\partial}{\partial z_n} f(z_n) = f'(z_n), \tag{22}$$

$$\frac{\partial \mathcal{L}}{\partial a_n} = \Delta_n. \tag{23}$$

Finally, the derivative then becomes

$$\frac{\partial \mathcal{L}}{\partial w_{nm}} = \Delta_n f'(z_n) x_m. \tag{24}$$

The definitions of Δ_n and f' depends on the choice of the loss function and the activation function on node n. Once every weight in the output layer is updated, the same method can be applied in a backwards manner to calculate derivatives with respect to weights in the hidden layers.

3.4 Convolutional Neural Network

The fully-connected neural networks that have been discussed so far have some limitations when it comes to certain types of input data, such as images. First of all, fully-connected layers are 1-dimensional, which means that pixels need to be serialised and most structural information in the image is lost. Second, since every neuron in one layer is connected to every neuron in the next, the number of connections grows rapidly as the number of neurons increases, and so does the number of computations needed. Finally, because the number of parameters directly corresponds to the number of input values, the size of the input needs to be fixed, which is often not desirable for images. *Convolutional neural networks* (CNNs) were inspired by the visual cortex, and are designed to work with images, although they are also used in natural language processing. Instead of assigning a single parameter to each pixel, CNNs learn a set of small filters that are shared by all parameters. Each filter looks at a small part of an image at a time, which is called its *receptive field*. One filter may respond to different structures in the image, such as lines and curves, than another. The filters are *convolved* over the image, which means that they are applied to each image location and the resulting activation maps show where in the image the structures associated with each filter are found and to what degree. By layering filters on top of each other, more complex structures can be recognised. Convolutional layers are generally followed by some non-linearity, a pooling layer and possibly some other layers.

3.4.1 Convolutional Layer

The main operator of the CNN is the *convolutional layer*. It is made up of f filters, each of which is 3-dimensional, with a width u and height v, and spanning the whole depth d of the input volume. For example, if the input is an RGB image and the first layer has 32 square filters of size 9, the filters have a shape of $32 \times 3 \times 9 \times 9$. Each filter slides over the image and at each position the dot product of the filter and part of the image is computed, resulting in a single value per filter per position. This principle is illustrated in Figure 9a. The layer has a stride s, which mean that after each calculation the filters slide s pixels over the image (in the horizontal and vertical directions). If the input of a convolutional layer has dimensions $n \times d \times w \times h$, the output of the layer has dimensions $n \times f \times ((w-u)/s+1) \times ((h-v)/s+1)$. Taking the same example as before, if the input is an RGB image of size $1 \times 3 \times 120 \times 90$ and the stride is 1, the output of the first layer has a shape of $1 \times 32 \times 112 \times 82$. With a stride of 3, the output becomes $1 \times 32 \times 38 \times 28$. Even if the stride is 1, convolutional layers shrink the input volume. To counteract this, *zero-padding* may be applied to inputs, which means that a border of zeroes is added to them. This may reduce or stop the amount of shrinking, depending on the size of the zero-padding.

3.4.2 Activation

The output of the convolutional layer is generally subjected to an activation function, which introduces non-linearity into the network. Like those in fully-connected networks, the activation function is applied element-wise to every parameter in the input. The most often used activation function in CNNs is the *Rectified Linear Unit* (ReLU). The ReLU layer applies the function $ReLU(x) = \max(0, x)$. Other variants of this function, such as the Leaky ReLU and the ELU exist as well. It is also possible to use the sigmoid or the hyperbolic tangent functions, although this is not very common.

3.4.3 Pooling

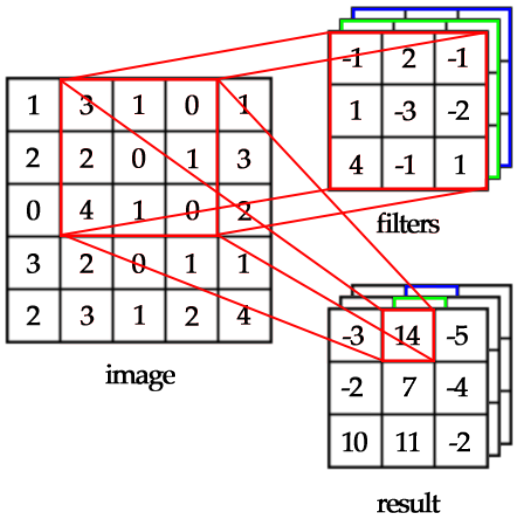
Another type of layer, which is often used, is the *pooling layer*. There are different types of pooling layers, such as max-pooling, average-pooling and L2-pooling. Just like convolutional layers, pooling layers have a size $p \times q$ and stride s, which determines how the input is transformed. A window of size $p \times q$ slides over the width and height dimensions of the input with steps of s. At each location, the given operation is performed over the window and output is a single value. For example, a max-pool layer of size 2×2 and stride 2 slides a window of 2×2 over the input elements and outputs the maximum value per position, which is depicted in Figure 9b. This action will cut the width and the height of the input volume in half. Downsampling is one of the reasons pooling layers are used, in order to increase the computational performance. Another important aspect of pooling layers is that they introduce some invariance to small transformations of features, which increases robustness.

3.4.4 Fully-Connected Layer as Convolutional Layer

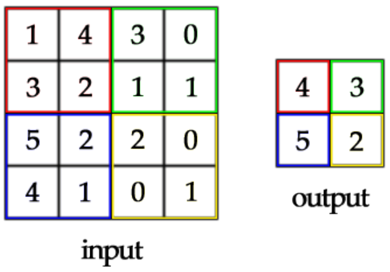
Many CNN architectures start with several convolutional layers that are followed by a number of fully-connected layers. In order for these layers to follow convolutional layers, the data need to be flattened, which is done by serialising the depth, height and width dimensions. However, fully-connected layers are often implemented as convolutional layers. This is done by adding a convolutional layer with n filters of size 1×1, where n is the number of desired nodes. In doing this, the CNN's ability to process images of variable size is conserved. Note, however, that a variable input size means a variable output size, as a CNN shrinks the width and height of the image by a set amount.

3.5 Optimisation

The optimisation algorithm described in Section 3.3 is also called *vanilla* gradient descent, as it is the most standard and straight-forward method of updating weights and biases. At every time-step the parameters are



(a)



(b)

Figure 9: (a) A convolution operation slides every filter over the image, calculating the dot product at each position. (b) A max-pooling operation with size 2 and stride 2 takes input elements of 2×2 and outputs the maximum value.

updated in the direction of steepest descent without taking any other information into account, and this may result in erratic movements caused by over-sensitivity to local structures in the multidimensional search-space. To update any parameter $p_i \in \{w_{11}, \ldots, w_{NM}, b_1, \ldots, b_N\}$ its new value at time step t is simply calculated as

$$p_i^t = p_i^{t-1} - \Delta p_i^t = p_i^{t-1} - \alpha g_i^t, \tag{25}$$

where g_i^t is the gradient of parameter p_i at time step t.

As mentioned before, one of simplest improvements to the standard method is the use of a momentum term, as it was first described by Rumelhart et al. [48]. By adding this term, the movements through the search-space are smoothed out and pulled into a general direction, much like a moving object builds up momentum. The momentum conserves the velocity of the object and it will keep moving in that direction until it is affected by external forces. As such, the momentum term adds velocity to the gradient descent by remembering how much and in which direction the parameters are moving through the search-space. The update rule is modified by calculating the parameter update value Δp_i^t as

$$\Delta p_i^t = \alpha(g_i^t + \mu \Delta p_i^{t-1}), \tag{26}$$

where μ is a decay factor between 0 and 1. By changing this factor, the contribution of the new direction on the momentum can be changed and it is normally set to around 0.9. The momentum term has the same dimensions as the network parameters, so every parameter has its own momentum value. In general, adding a momentum value helps the network converge more quickly.

Duchi et al. introduced another optimiser called AdaGrad [8], which aims to normalise the contribution of each parameter in the update and reduce the effect of large gradients and increase that of small ones. The idea is that features that are seen less often are more discriminative and should contribute more to the output. Every parameter gets its own learning rate by dividing it by the total magnitude of all past gradients and its update value is

$$\Delta p_i^t = \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} |g_i^\tau|_2}} g_i^t, \tag{27}$$

where $|g_i^\tau|_2$ is the L2-norm of gradient i at time-step τ per dimension. It can be seen that the denominator accumulates all previous gradients and grows with each step. This means that with each update, the step size gets smaller and smaller, until it eventually becomes nearly 0 and progress halts. Zeiler aims to improve this with AdaDelta [68], which is directly based on AdaGrad. Instead of taking the complete gradient history, the running average is decayed over time, similar to the momentum term. The decaying running average $E[|g_i|_2]^t$ is defined as

$$E[|g_i|_2]^t = \rho E[|g_i|_2]^{t-1} + (1-\rho)|g_i^t|_2, \tag{28}$$

where ρ is the decaying factor. By using this value to scale the gradients, the problem of progress slowing down over time is solved, but Zeiler finds another issue with the previously described methods. Because the gradients are divided by the running average of the gradients, there is a mismatch between the hypothetical units of the parameters and those of the parameter updates. In order to fix this, another term is added to the update value, which is the running average of the parameter updates. The update value then becomes

$$\Delta p_i^t = \frac{\sqrt{E[|\Delta p_i|_2]^{t-1} + \epsilon}}{\sqrt{E[|g_i|_2]^t + \epsilon}} g_i^t. \tag{29}$$

The constant ϵ is added both to start off the algorithm when the numerator is 0 in the first time-step, and to avoid division by 0. Adding the running average of the parameter updates to the numerator fixes the unit mismatch, and since it replaces the learning rate α, there is no need to spend time finding a good value for it. This makes AdaDelta stand out from the other optimisation algorithms.

More recently, the optimiser Adam was introduced by Kingma and Ba [19]. Adam works by keeping track of the first and second moment of the gradient. The second moment is, similar to AdaGrad and

AdaDelta, the average magnitude of the gradients per dimension. However, Adam additionally accumulates the first moment, which is the mean gradient value per dimension. At each time-step t, the first moment m_i^t and second moment v_i^t of parameter p_i are updated according to

$$m_i^t = \beta_1 m_i^{t-1} + (1 - \beta_1) g_i^t, \tag{30}$$

$$v_i^t = \beta_2 v_i^{t-1} + (1 - \beta_2) |g_i^t|_2, \tag{31}$$

where β_1 and β_2 are hyperparameters that control the decay rate. However, since the values for m_i^t and v_i^t are initialised to 0, these values are biased toward 0, especially in the first few iterations. Therefore, the first and second moment values are bias-corrected, as

$$\hat{m}_i^t = m_i^t / (1 - \beta_1^t), \tag{32}$$

$$\hat{v}_i^t = v_i^t / (1 - \beta_2^t). \tag{33}$$

Once the bias-corrected first and second moments are calculated, the update value of parameter p_i then is defined as

$$\Delta p_i^t = \alpha \frac{\hat{m}_i^t}{\sqrt{\hat{v}_i^t} + \epsilon}. \tag{34}$$

The value $\hat{m}_i^t$ is the decaying average of the past gradients, which is similar to the momentum method. The value $\hat{v}_i^t$ is the decaying average gradient magnitude, which is similar to the AdaDelta method. Therefore, Adam seems to combine the two methods.

3.6 Regularisation

Training data are divided into training set, validation set and test set. This is done to measure the performance, not only on the data that the network is trained on, but also on new data that the network has not seen yet. If there exists a discrepancy between the loss of the training set and the loss of the validation set, the network is likely *overfitting*. This means that the network has a good performance on the training samples, but does not generalise well for new data. In other words, the network *memorises* the correct outputs for the given inputs, instead of finding a general solution. In order to prevent overfitting, some form of *regularisation* can be implemented. There are many different types of regularisation and two common methods will be discussed here.

One of the most basic types of regularisation is L1 and L2 regularisation. The magnitudes of the weights are restricted by adding a penalty term to the loss function. This should decrease the complexity of the model and make it less likely to overfit to the training data. Given a loss function $\mathcal{L}$, the L1 regularised loss becomes

$$\mathcal{L}_1 = \mathcal{L} + \lambda \sum_{w \in W} |w|, \tag{35}$$

where W is the set of all weights in the network and λ is the scaling factor that determines how much the weights are penalised. Analogously, the L2 regularised loss is given by

$$\mathcal{L}_2 = \mathcal{L} + \lambda \sum_{w \in W} w^2. \tag{36}$$

The difference is that L2 regularisation minimises the magnitude of all weights, whereas L1 regularisation may lead to sparse weights. It is also possible to use a mix of both.

Another method that is often used is called *dropout* [14]. During training, at every step of the iteration, a random subsample of neurons in the network is deactivated. This means that they neither participate in the forward step, nor in the backward step of the learning process. The reason this prevents overfitting is that one subset of neurons might start to overfit, which makes the network biased toward the training data. However, when a new subset of neurons is chosen, the bias is no longer present and will not be strengthened. Another way to look at it is that each subset is a separate neural network. When training is done and all neurons are active, the result is the average of all the separately trained networks.

3.7 Normalisation

The distribution of the input data affects how the network is trained, how well it converges and how it performs. Therefore, it is common to normalise the data in some ways before or as it is processed by the network. Here are several method to normalise input data.

One of the most basic types of normalisation is input normalisation. The first step is usually mean centring, which balances the data around 0. Then, the data are normalised by dividing by the standard deviation, so that the standard deviation is 1. This can be done in several ways. Assuming the data are images, the normalisation can be done per image, such that each image has a mean of 0 and a standard deviation of 1. What is also common, is normalising per pixel. In that case, the mean and standard deviation of the complete data set is stored and each input image is normalised before processing. The normalisation can also be done per channel or for all values. Which method is preferable depends on the data and the application.

A type of normalisation that is used in between neural network layers is *Local Contrast Normalisation* (LCN) [18]. The goal is to enforce a local competition of features. A Gaussian-weighted square window (in the paper they use a size of 9×9) spanning the whole depth of the input with a sum of 1 is applied at each position. For an input value x_{ijk}, where i, j and k are the depth, height and width indices, respectively, the mean centred value m_{ijk} is calculated as

$$m_{ijk} = x_{ijk} - \sum_{ipq} w_{jk} x_{i,j+p,k+q}. \tag{37}$$

Here, p and q are the indices of the window in the width and height dimensions. Then, to normalise the values, at each position the standard deviation is calculated as

$$\sigma_{jk} = \sqrt{\sum_{ipq} w_{jk} m^2_{i,j+p,k+q}}. \tag{38}$$

The normalised value n_{ijk} is then defined as

$$n_{ijk} = m_{ijk} / \max(c, \sigma_{jk}), \tag{39}$$

where the value c is set to the mean value of σ_{jk} per sample.

Similar to LCN, Local Response Normalisation (LRN) [21] normalises response values over the depth of the response after the application of the ReLU activation. Each value x_{ijk} is normalised as

$$n_{ijk} = x_{ijk} / (k + \alpha \sum_{p} x^2_{pjk})^{\beta}, \tag{40}$$

where p iterates over a number of adjacent responses at the same width and height position and k, α and β are hyperparameters. In the original paper, $p = [-2, 2]$, $k = 2$, $\alpha = 10^{-4}$ and $\beta = 0.75$. The difference with LCN is that no mean centring is done, there is no window, but only the values at the same position are taken and the normalisation is not done over the whole depth.

Perhaps the most prevalent type of normalisation in recent years is *batch normalisation* [16]. Like LCN and LRN it normalises activations in between convolutional layers. The difference is that batch normalisation does this per parameter over a complete batch. For a parameter x_{ijk}, a batch with B samples is defined as $\mathbf{x}_{ijk} = \{x_{1ijk}, \ldots, x_{Bijk}\}$. Then, the mean over the batch is defined as

$$m_{ijk} = \frac{1}{B} \sum_{b=1}^{B} x_{bijk}. \tag{41}$$

The variance of the mean-centred batch is then calculated as

$$\sigma^2_{ijk} = \frac{1}{B} \sum_{b=1}^{B} (x_{bijk} - m_{ijk})^2. \tag{42}$$

The parameter is then normalised as

$$n_{ijk} = \frac{x_{bijk} - m_{ijk}}{\sqrt{\sigma_{ijk}^2 + \epsilon}}. \tag{43}$$

However, since this normalisation restricts the outputs of the previous layer, two additional learnable parameters γ and β are added. The final normalised result becomes

$$\hat{n}_{ijk} = \gamma n_{ijk} + \beta. \tag{44}$$

These two parameters enable the layer to restore the original values by setting their values to the mean and standard deviation.

4 Learning Feature Detectors and Descriptors

In Section 2 several conventional, hand-crafted methods for feature detection and description were presented and in Section 3 the definitions of NNs and CNNs were discussed. In this section, we consider a selection of methods that have been published in recent years that combine feature detection and neural networks, to produce NNs that have learned to find stable keypoints. Whereas manually defined feature detectors generally focus on one or several kinds of structures, neural networks may learn to find structures that are more complex and are stable with exactly those transformations that are desirable. Naturally, a neural network will produce keypoints that are stable to those transformations that are presented during training. And this is perhaps where the strength of learning lies, as the NN can learn to perform for any specific task given the right data, whereas for hand-crafted methods a completely new method may have to be designed.

The feature detection and matching pipeline is made up of multiple steps, each of which may be replaced by a neural network, although the main categories discussed here are the detection, orientation assignment and feature description.

4.1 TILDE

TILDE [60] is a method by Verdie et al. and is one of the first ones that can be learned by a neural network to detect keypoints. The method focuses on keypoints that are stable under different lighting and weather conditions. It uses supervised learning to train a CNN, as the ground truth for what should and should not be a keypoint is determined by using some manual method, which is SIFT in the original paper, but the authors mention that any other method may be used. The data set used contains images of the same scenes in different conditions and the training data consist of pairs of positive and negative samples. Positive samples are created at locations where in most conditions a keypoint is located and negative samples are locations where there are no keypoints near in any condition. Using the sample patches, network inputs $\mathbf{X}$ are retrieved by calculating the horizontal and vertical image gradients, the gradient magnitude and the components of the LUV colour space for each pixel, which results in a total of 6 image features per pixel. The network outputs a score map, which indicates how well each point satisfies the network's definition of an interest point. Actual interest point are then selected from the score map by applying non-maximum suppression.

The TILDE regressor is a single-layer network that is defined by the Generalised Hinging Hyperplanes (GHH) [62], which is a piecewise linear function. The GHH in the context of CNNs consists of a set of parameters $\omega = [\mathbf{W}, b]$, which are the filters $\mathbf{W} = \{\mathbf{W}_{11}, \ldots, \mathbf{W}_{NM}\}$ and the biases $b = \{b_{11}, \ldots, b_{NM}\}$. The output of the GHH is defined as

$$GHH(\mathbf{X}, \omega) = \sum_{n=1}^{N} \delta_n \max_{m=1}^{M} \mathbf{W}_{nm} * \mathbf{X} + b_{nm}, \tag{45}$$

where $\delta_n = +1$ if n is odd and -1 if it's even. The GHH is a generalisation of activation functions, such as ReLU and PReLU, which can be defined as a GHH. Since the TILDE network is a single-layer network, given the set of parameters, its activation is simply calculated as

$$\mathcal{N}(\mathbf{X}) = \mathbf{A}(\mathbf{X}, \omega) = GHH(\mathbf{X}, \omega). \tag{46}$$

In the paper $N = 4$ and $M = 4$, which means that there are 16 filters of size 21×21 per image feature. The objective function $\mathcal{L}$ used to train the TILDE network consists of three parts, the classification-like loss $\mathcal{L}_c$, the shape regulariser loss $\mathcal{L}_s$ and the temporal regulariser loss $\mathcal{L}_t$, and is defined as

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_s + \mathcal{L}_t. \tag{47}$$

The classification-like loss $\mathcal{L}_c$ helps the networks distinguish between the positive samples and negative samples and takes the form

$$\mathcal{L}_c = \gamma_c ||\omega||_2^2 + \frac{1}{K} \sum_{\mathbf{x}_i \in \mathbf{X}} \max(0, l_i \mathbf{A}(\mathbf{x}_i, \omega))_2, \tag{48}$$

where γ_c is a hyperparameter, $\mathbf{X}$ is the set of all input samples and K its cardinality and $l_i \in \{-1, +1\}$ is the label for the negative and positive samples. This means the network is encouraged to return positive values for patches that contain a keypoint and negative values for patches that don't and a simple non-maximum suppression can be used to find local maxima.

The next part of the loss function, the shape regulariser $\mathcal{L}_s$, is meant to give the shape of local maxima a higher peak and is defined as

$$\mathcal{L}_s = \frac{\gamma_s}{K_p} \sum_{\mathbf{x}_i \in \mathbf{X}_p} \sum_{n=1}^{N} \left|\left|\mathbf{w}_{n\eta_i(n)} * \mathbf{x}_i - (\mathbf{w}_{n\eta_i(n)} \mathbf{x}_i)\mathbf{h}\right|\right|_2^2 \tag{49}$$

where γ_s is another hyperparameter, $\mathbf{X}_p$ is the set of all positive samples and K_p its cardinality, $\eta_i(n) = \operatorname{argmax}_m \mathbf{w}_{nm}^\top \mathbf{x}_i$ adds only the filters that contribute to the response and $\mathbf{h}(x, y) = e^{\alpha(1-\frac{\sqrt{x^2+y^2}}{\beta})}$ is a function that has the desired shape. The shape regulariser term is then adjusted by using approximations of the 2D Fourier transform.

Finally, the last term of the loss function is the temporal regulariser $\mathcal{L}_t$, which is defined as

$$\mathcal{L}_t = \frac{\gamma_t}{K_p} \sum_{i=1}^{K_p} \sum_{\mathbf{x}_j \in \mathbf{C}_i} (\mathbf{A}(\mathbf{x}_i, \omega) - \mathbf{A}(\mathbf{x}_j, \omega))^2, \tag{50}$$

where γ_t is another hyperparameter and $\mathbf{C}_i$ is the set of images at the same location as image $\mathbf{x}_i$, but in different conditions . This term helps the network to look for keypoints located at structures that are present in all of the different conditions.

4.2 Covariant Feature Detector

The previously defined method learns to detect features by using SIFT or other keypoints as ground-truth and the resulting detector is therefore biased towards the structures present in those keypoints. Lenc and Vedaldi were one of the first to define a truly unsupervised feature detector [26]. They define a covariance constraint that enables a neural network to learn to produce outputs that are covariant to any affine transformation.

4.2.1 Network Output

Where many other learned detectors output a score map that indicates the quality of each point as a keypoint, the network $\mathcal{N}$ of this method transforms an image patch $\mathbf{p}$ into a transformation matrix as

$$\mathcal{N}(\mathbf{p}) = \begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{x} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_u & b_u & x_u \\ a_v & b_v & x_v \\ 0 & 0 & 1 \end{bmatrix}, \tag{51}$$

where $\mathbf{x}$ is the centre of the keypoint relative to the centre of the patch and $\mathbf{a}$ and $\mathbf{b}$ encode other properties of the keypoint, such as shape and orientation. A very simple implementation would only output the keypoint coordinates $\mathbf{x}$ and interpret $\begin{bmatrix}\mathbf{a} & \mathbf{b}\end{bmatrix}$ as the identity I. In more complex cases $\mathbf{a}$ and $\mathbf{b}$ could, for example, encode the keypoint orientation, the shape of an ellipse or even a combination of both. These transformation matrices can be interpreted to transform some base keypoint at the centre of the patch to the location and shape of the most prominent keypoint. What these values actually encode is enforced by the way network is trained and the transformations in the data presented to it.

4.2.2 Learning

Training data consist of triplets $(\mathbf{p}^1, \mathbf{p}^2, t)$, where $\mathbf{p}^1$ and $\mathbf{p}^2$ are patches that overlap at least partially, $t \in \mathbb{T}$ is a set of transformations and the patches are related by the transformation, such that $\mathbf{p}^2 = t\mathbf{p}^1$ and the inverse $t^{-1}\mathbf{p}^2 = \mathbf{p}^1$. The set of transformations $\mathbb{T}$ can be separated into two subsets $\mathbb{P} \subseteq \mathbb{T}$, the transformations that are resolved by the detector, and $\mathbb{Q} \subset \mathbb{T}$, those that are not resolved by the detector. In other words, $\mathbb{P}$

is the set of transformations that the detector outputs and $\mathbb{Q}$ is the set of transformations that the detector is covariant with, but does not output. An example of this is a corner detector that is covariant with rotations. The network only outputs coordinates, but points to the same location, even if the image is rotated. In this case, the set of resolved transformations $\mathbb{P} = T$ equals translation and the set of residual transformations $\mathbb{Q} = R$ equals rotation.

First, let's consider the case that the network resolves all transformations in the set $\mathbb{T}$ and the set of residual transformations $\mathbb{Q} = I$ is empty and only contains the identity. Since $t^{-1}\mathbf{p}^2 = \mathbf{p}^1$ and the outputs of the networks are related by the same transformation t, the network outputs should satisfy the constraint $\mathcal{N}(\mathbf{p}^2)t^{-1} = \mathcal{N}(\mathbf{p}^1)$. This can be rewritten into the covariance constraint

$$\mathcal{N}(\mathbf{p}^2)\mathcal{N}(\mathbf{p}^1)^{-1}t^{-1} = I. \tag{52}$$

This constraint is only satisfied if the network returns the same keypoint in both patches.

However, in the case that the network resolves only a subset of $\mathbb{T}$, applying the inverse transformation t^{-1} to $\mathcal{N}(\mathbf{p}^2)$ also applies the inverse of the transformation q, which is not resolved by the network. Therefore, only tq^{-1}, the part of t without q, should be applied to $\mathcal{N}(\mathbf{p}^2)$. Thus the network should satisfy the constraint $\mathcal{N}(\mathbf{p}^2)(tq^{-1})^{-1} = \mathcal{N}(\mathbf{p}^1)$. Finally, the covariance constraint becomes

$$\mathcal{N}(\mathbf{p}^2)q\mathcal{N}(\mathbf{p}^1)^{-1}t^{-1} = I. \tag{53}$$

A network that satisfies this constraint is thus covariant to all transformations in $\mathbb{T}$. The learning objective is then defined as

$$\mathcal{L}(\mathbf{p}^1, \mathbf{p}^2, t, q) = \frac{1}{K}\sum_{k=1}^{K}||\mathcal{N}(\mathbf{p}_k^1)t - \mathcal{N}(\mathbf{p}_k^2)q||_2^2. \tag{54}$$

Once the network is trained, it gives the location of the most prominent keypoint for each patch. Given an input image, the network is convolved over the image and a a transformation matrix is output per patch. Keypoints are then extracted from images by accumulating votes from each of these outputs. For example, each output gives the coordinates of the best keypoint in the local area and votes are added at those coordinates by using bilinear interpolation. Doing this with every patch output results in a sort of score map, on which non-maximum suppression and thresholding can be applied to find keypoints. This method can also be used for other properties of the keypoints, although an exact method is not given in the paper. There are two architectures defined for the detector, which are both depicted in Table 1.

DetNet-S

Conv 1 $40 \times 5 \times 5$	Conv 2 $100 \times 5 \times 5$	Conv 3 $300 \times 4 \times 4$	FC 1 500	FC 2 500	FC 3 2
ReLU	ReLU	ReLU	ReLU	ReLU	
Max-Pool 2×2	Max-Pool 2×2				

DetNet-L

Conv 1 $40 \times 5 \times 5$	Conv 2 $100 \times 5 \times 5$	Conv 3 $300 \times 4 \times 4$	FC 1 500	FC 2 500	FC 3 500	FC 4 2
ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	
Max-Pool 2×2	Max-Pool + LRN 2×2					

Table 1: The two architectures of the covariant feature detector, with the small network on top and the large network on the bottom.

4.3 Quad Detector

Another method that learns a feature detector, without using points from other methods to focus on, is the quad detector by Savinov et al. [50]. It's called the quad detector, because it learns by processing quadruples of input patches. Similar to many other learned detectors, it takes an image as input and outputs a score map that gives a higher score to points that are invariant or covariant to a set of transformations. This only describes the detection stage and invariance to scale and rotation is achieved by using the SIFT pipeline and replacing the detector. The quad detector is applied to a scale-space and keypoints are selected by performing non-maximum suppression and second-order Taylor expansion.

4.3.1 Learning

Learning to detect keypoints that are repeatable under certain transformations is done by presenting the detector four image patches from two images. The two images of the same scene are separated by a set of transformations, mainly a change in viewpoint and illumination, but any desired transformations can be included. A quadruple of images is extracted as

$$Q = (\mathbf{p}_d^1, \mathbf{p}_d^2, \mathbf{p}_{t(d)}^1, \mathbf{p}_{t(d)}^2), \tag{55}$$

where $\mathbf{p}_d^i$ and $\mathbf{p}_{t(d)}^i$ are two corresponding points in image d and its transformed counterpart $t(d)$. Given a set of quadruples, the detector $\mathcal{N}$ is to learn a response that satisfies a ranking constraint

$$R(\mathbf{p}_d^1, \mathbf{p}_d^2, \mathbf{p}_{t(d)}^1, \mathbf{p}_{t(d)}^2) = (\mathcal{N}(\mathbf{p}_d^1) - \mathcal{N}(\mathbf{p}_d^2))(\mathcal{N}(\mathbf{p}_{t(d)}^1) - \mathcal{N}(\mathbf{p}_{t(d)}^2)) > 0. \tag{56}$$

This constraint is only satisfied if the detector preserves the order of the points. In other words, if $\mathcal{N}(\mathbf{p}_d^1)$ is larger than $\mathcal{N}(\mathbf{p}_d^2)$, then $\mathcal{N}(\mathbf{p}_{t(d)}^1)$ is also larger than $\mathcal{N}(\mathbf{p}_{t(d)}^2)$, and vice versa. To make this constraint differentiable, it is embedded in the hinge loss and the objective function of the detector becomes

$$\mathcal{L} = \sum_{d \in D} \sum_{t \in \mathbb{T}} \sum_{i,j \in C} \max(0, 1 - R(\mathbf{p}_d^i, \mathbf{p}_d^j, \mathbf{p}_{t(d)}^i, \mathbf{p}_{t(d)}^j)), \tag{57}$$

where D is the set of all images, $\mathbb{T}$ is the set of all transformations and C is the set of all corresponding points. Since it's not practical to take all of the combinations of images, transformation and corresponding points, quadruples are generated by random sampling.

4.3.2 Data Generation

Data generation as described above depends on all desired transformations being included in the data set used. Since it may be difficult to find data sets that include all transformations that are desired, it is also possible to apply some transformations manually. To make the detector invariant to a set of transformations $\mathbb{T}_a$, for each quadruple two transformations $t_a^1, t_a^2 \in \mathbb{T}_a$ are sampled at random and applied to the patches as

$$Q_a = (t_a^1(\mathbf{p}_d^1), t_a^1(\mathbf{p}_d^2), t_a^2(\mathbf{p}_{t(d)}^1), t_a^2(\mathbf{p}_{t(d)}^2)). \tag{58}$$

This makes the detector learn to additionally preserve the order under the applied transformations. For example, if the data set does not contain any images that are rotated, applying manual rotations of patches in this manner ensures rotational invariance is learned by the detector. Another set of transformations $\mathbb{T}_b$ can be used to augment the training images by applying transformations $t_b^1, t_b^2 \in \mathbb{T}_b$ as

$$Q_b = (t_b^1(\mathbf{p}_d^1), t_b^2(\mathbf{p}_d^2), t_b^1(\mathbf{p}_{t(d)}^1), t_b^2(\mathbf{p}_{t(d)}^2)). \tag{59}$$

In this case, the same transformation is applied to the same point in each corresponding image. This does not enforce invariance to the applied transformations, but does increase robustness to them. An example in which this may be desirable is scaling. If certain structures are only present at very large or very small scales, including patches extracted at these scales will ensure the detector processes these structures and possibly learns to detect them. Invariance in this case is not desirable, however, since a structure present at one scale may not be visible at another.

4.4 Orientation Estimator Network

Not only feature detection, but also orientation assignment can be done with neural networks. A method by Yi et al. is designed exactly for that purpose [67]. In order to estimate the orientation of a given patch $\mathbf{p}$, the orientation needs to be parameterised. The network outputs two values that are interpreted as the normalised sine and cosine of the patch's angle. These values can then be used to calculate the angle in degrees.

Given two input patches $\mathbf{p}_1$ and $\mathbf{p}_2$, centred at the same point with different orientation, the loss is defined as

$$\mathcal{L} = ||\mathrm{d}(\mathbf{p}_1, \mathcal{R}(\mathbf{p}_1)) - \mathrm{d}(\mathbf{p}_2, \mathcal{R}(\mathbf{p}_2))||_2^2, \tag{60}$$

where $\mathrm{d}(\mathbf{p}, \mathcal{R}(\mathbf{p}))$ is the descriptor of patch $\mathbf{p}$ with the orientation output by the orientation network $\mathcal{R}(\mathbf{p})$. The network thus aims to choose orientations for pairs of patches that minimise the difference between the resulting descriptors. The orientation network also uses the GHH activation defined in equation 45. The network consists of 5 layers, of which the first 3 are convolutional layers with the ReLU activation and max-pooling and the last 2 are fully-connected layers with the GHH activation. The architecture of the network can be seen in Table 2.

Conv 1 $10 \times 5 \times 5$	Conv 2 $20 \times 5 \times 5$	Conv 3 $50 \times 3 \times 3$	FC 1 100	FC 2 2
ReLU	ReLU	ReLU	GHH $N = 4, M = 4$	GHH $N = 4, M = 4$
Max-Pool 2×2	Max-Pool 2×2	Max-Pool 2×2		

Table 2: The architecture of the orientation estimator network. The last 2 layer are fully-connected and use the GHH activation with 4 filters in the maximum and 4 filters in the summation.

4.5 LIFT

Another method by Yi et al. titled Learned Invariant Feature Transform (LIFT) [66] learns to perform the complete feature detection pipeline, from detection to orientation assignment to description. It contains three separate networks, the detector $\mathcal{N}$, the orientation estimator $\mathcal{R}$ and the descriptor $\mathcal{D}$. The detector takes as input an image and produces a score map that indicates the distinctiveness of each point and selects keypoints by performing non-maximum suppression. Small patches are cropped out at the locations found by the detector, which are in turn used as input for the orientation estimator. The orientation network then produces orientations that are applied to the patches. Finally, the rotated patches are processed by the descriptor network, that outputs a multidimensional vector for each input patch.

4.5.1 Training

Training data are generated using SIFT keypoints to determine positive and negative training samples. The data are comprised of quadruples of image patches $(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4)$, where $\mathbf{P}^1$ and $\mathbf{P}^2$ are centred at the same SIFT keypoint, but from a different 3D perspective, $\mathbf{P}^3$ are centred at another SIFT keypoint and $\mathbf{P}^4$ are located where there is no keypoint. After they are processed by the detector $\mathcal{N}$, the patches $\mathbf{P}$ are cropped into smaller patches $\mathbf{Q}$ centred at the maximum score value. The orientation network then calculates orientations, that are applied to the cropped patches $\mathbf{Q}$, to produce rotated patches $\mathbf{R}$. These rotated patches are then transformed into descriptors $\mathbf{D}$.

Even though the flow goes from detector to descriptor, the network is trained in reverse order, from descriptor to orientation estimator to detector. The descriptor is trained separately as first using SIFT keypoints as training data. Then the orientation estimator is trained together with the already trained descriptor, replacing the SIFT orientations with the ones output by the network $\mathcal{R}$. Finally, the detector is trained using the complete CNN pipeline. To make the maximum selection differentiable, the $\mathrm{softargmax}$ function [5] is used during training, which is defined as

$$\text{softargmax}(\mathbf{S}) = \sum_{\mathbf{x}} \frac{\exp(s_{\mathbf{x}}/\sigma)}{\sum_{\mathbf{y}} \exp(s_{\mathbf{y}}/\sigma)} \mathbf{x}, \tag{61}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ are coordinates of the score map $\mathbf{S}$, $s_{\mathbf{x}}$ and $s_{\mathbf{y}}$ are the scores at those coordinates and σ is a smoothing factor. As σ goes to 0, the softargmax function converges to the argmax function and outputs the coordinates of the maximum value. To make the image transformations differentiable, the cropping and rotation of patches is done using *spatial transformer* layers [17]. A spatial transformer layer takes as input a transformation matrix that contains the desired transformations, such as rotation, scaling and even affine transformations. It generates a sampling grid that contains coordinates of the input values and applies the transformations to get the coordinates of the output values. A sampling kernel can then generate the output using some type of interpolation, such as bilinear interpolation.

4.5.2 Descriptor

The descriptor network of LIFT is the one presented by Simo-Serra et al. [54]. The network is made up of 3 convolutional layers, each of which is followed by an L2-pooling layer, a hyperbolic tangent (Tanh) activation layer and a Local Contrast Normalisation (LCN) layer (discussed in Section 3.7). The input patches are 64×64 pixels and are transformed by the network into 128-dimensional descriptors. The descriptor network processes triplets of rotated patches $(\mathbf{R}^1, \mathbf{R}^2, \mathbf{R}^3)$, where $\mathbf{R}^1$ and $\mathbf{R}^2$ are corresponding pairs and $\mathbf{R}^3$ are non-corresponding. The network learns by calculating the L2-norm between pairs of positive and negative samples and the loss function takes the form

$$\mathcal{L}_{\mathcal{D}}(\mathbf{R}^r, \mathbf{R}^t) = \frac{1}{K_{\mathcal{D}}} \sum_{k=1}^{K_{\mathcal{D}}} l(\mathcal{D}(\mathbf{r}_k^r), \mathcal{D}(\mathbf{r}_k^t)), \tag{62}$$

where $K_{\mathcal{D}}$ is the number of patch pairs, $\mathcal{D}(\mathbf{r}_k^r)$ and $\mathcal{D}(\mathbf{r}_k^t)$ are the computed descriptors for the reference patch and its pair and

$$l(\mathcal{D}(\mathbf{r}^r), \mathcal{D}(\mathbf{r}^t)) = \begin{cases} ||\mathcal{D}(\mathbf{r}^r) - \mathcal{D}(\mathbf{r}^t)||_2, & \text{for positive pairs,} \\ \max(0, C - ||\mathcal{D}(\mathbf{r}^r) - \mathcal{D}(\mathbf{r}^t)||_2), & \text{for negative pairs,} \end{cases} \tag{63}$$

is the hinge embedding loss. This encourages the network to give a similar output for positive samples and penalises negative samples that are closer than C in terms of L2-distance. An important technique that is used in learning is the *mining* of hard samples, which means that in each batch of size K_b, the K_h samples with the highest hinge embedding loss are selected and backpropagation is only performed on those hard samples. This reportedly improves the performance significantly.

4.5.3 Orientation Estimator

The next step is to learn the orientation estimator, together with the already learned descriptor. This orientation estimator is very similar to the one presented in Section 4.4. At this stage, only positive pairs of cropped patches $(\mathbf{Q}^1, \mathbf{Q}^2)$ are considered. The objective function is simply defined as

$$\mathcal{L}_{\mathcal{R}}(\mathbf{Q}^1, \mathbf{Q}^2) = \frac{1}{K_{\mathcal{R}}} \sum_{k=1}^{K_{\mathcal{R}}} ||\mathcal{D}(\varrho(\mathbf{q}_k^1)) - \mathcal{D}(\varrho(\mathbf{q}_k^2))||_2, \tag{64}$$

where $\varrho(\mathbf{q})$ is the rotation operator that rotates the cropped patch $\mathbf{q}$ by $\mathcal{R}(\mathbf{q})$, the angle output by the orientation estimator. This simply is the mean L2-distance between the resulting descriptors and the network is encouraged to learn orientations that minimise the distance of the descriptors for corresponding patches.

4.5.4 Detector

The detector of LIFT is very similar to the TILDE detector and the network also uses the GHH (described in Section 4.1). The difference is that LIFT applies the network directly to the image data, where TILDE extracts features from the image. The detector's loss function has two parts:

$$\mathcal{L}_{\mathcal{D}}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) = \gamma \mathcal{L}_{\text{class}}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) + \mathcal{L}_{\text{pair}}(\mathbf{P}^1, \mathbf{P}^2). \tag{65}$$

Here γ is a hyperparameter that balances the two parts. The class-loss is defined as

$$\mathcal{L}_{\text{class}}(\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \mathbf{P}^4) = \frac{1}{K_{\mathcal{N}}} \sum_{k=1}^{K_{\mathcal{N}}} \sum_{i=1}^{4} \alpha_i \max(0, (1 - \operatorname{softmax}(l_i \mathcal{N}(\mathbf{p}_k^i))))^2, \tag{66}$$

where $l_i = 1$ if the patch is centred at a keypoint ($i \in \{1, 2, 3\}$) and $l_i = -1$ if the patch is not centred at a keypoint ($i = 4$), α_i is another hyperparameter to balance the two cases and the $\operatorname{softmax}$ function is the differentiable version of the $\max$ function. The pair-loss is

$$\mathcal{L}_{\text{pair}}(\mathbf{P}^1, \mathbf{P}^2) = \frac{1}{K_{\mathcal{N}}} \sum_{k=1}^{K_{\mathcal{N}}} ||\mathcal{D}(\varrho(\varsigma(\mathbf{p}_k^1))) - \mathcal{D}(\varrho(\varsigma(\mathbf{p}_k^2)))||_2, \tag{67}$$

where $\varsigma(\mathbf{p})$ is the crop operator that crops the patch $\mathbf{p}$ at the coordinates of the maximum value in the score map $\operatorname{softargmax}(\mathcal{N}(\mathbf{p}))$. The class-loss makes the network learn to discriminate keypoints from non-keypoints and the pair-loss makes the network select keypoints that are invariant to the transformations in the training data. The data for the detector are also mined for hard samples. To help the detector learn, the network is pre-trained by replacing the pair-loss $\mathcal{L}_{\text{pair}}$ is by the function

$$\tilde{\mathcal{L}}_{\text{pair}}(\mathbf{P}^1, \mathbf{P}^2) = \frac{1}{K_{\mathcal{N}}} \sum_{k=1}^{K_{\mathcal{N}}} (1 - \frac{\mathbf{p}_k^1 \cap \mathbf{p}_k^2}{\mathbf{p}_k^1 \cup \mathbf{p}_k^2} + \frac{\max(0, ||\mathbf{x}_k^1 - \mathbf{x}_k^2||_1 - 2s)}{\sqrt{\mathbf{p}_k^1 \cup \mathbf{p}_k^2}}), \tag{68}$$

where $\mathbf{x}_k^i = \operatorname{softargmax}(\mathcal{N}(\mathbf{p}_k^i))$ and $s = 64$ is the width and height of the patches $\mathbf{p}_k^i$. This outputs 0 for patches that are centred at the same location and increases with the distance between the patches. After pre-training this function is replaced with the normal pair-loss function.

5 Experiments

Determining the performance of feature detectors and extractors is not trivial, as it is dependent on the application and the data. There are different measures that are defined to quantify the quality of detected features in a general sense. On the other hand, finding the performance of a specific application can give an indication of how well features can be used in practice. This section discusses several methods that have been used to test the performance of a selection of feature detectors and descriptors. The detector-descriptor pairs that are used in the experiments are shown in Table 3. The results of these experiments are discussed in Section 6.

Detector	**Descriptor**	**Distance**	**Dimensionality**	**Memory Cost**
SIFT	SIFT	L2	128	512 B
SIFT	VGG16	L2	256	1 KB
SURF	SURF	L2	64	256 B
AGAST	BRISK	Hamming	512	64 B
CenSurE	SIFT	L2	128	512 B
Quad	SIFT	L2	128	512 B
Quad	VGG16	L2	256	1 KB
LIFT	LIFT	L2	128	512 B

Table 3: A list of detectors-descriptor pairs, together with the distance measure used to find matches, the dimensionality and memory cost per descriptor.

An important property of feature detectors and descriptors is the computation time, which is listed in Table 4 for the detectors (including orientation assignment) and descriptors. Features were detected on the first 1 000 images of the ILSVRC 2017 data set. Descriptors were calculated for only 300 keypoints.

Detector	**Computation Time (s)**
SIFT	74.86
SURF	64.19
CenSurE	68.21
FAST	6.08
AGAST	90.26
Quad	958.12
LIFT	1972.86

Descriptor	**Computation Time (s)**
SIFT	60.84
SURF	46.24
BRISK	3.04
LIFT	220.59
VGG16	910.55

Table 4: The total computation time for detectors on the left and descriptors on the right, calculated over the first 1000 images of the ILSVRC 2017 test set.

5.1 Repeatability

Repeatability is one of the most frequently used measures of stability and robustness in detected features, since it was first defined for local features by Schmid et al. [51]. It measures the amount of features that are repeated when images are subjected to geometric and photometric transformations. In the second part of the feature matching pipeline, when the descriptors are calculated and matched, it is important that there are as many correspondences as possible. For a set of images of the same scene that is separated by some transformation, the number of correspondences dictate the possible number of descriptor matches. If there are no correspondences in the detected features, a correct match cannot be found. Therefore a high stability and robustness of detected features is desirable and this quality is tested by the repeatability measure. If the repeatability is high for a group of transformations, then the detector is said to be *covariant* to those transformations. The advantage of using repeatability as a scoring function is that the process can be fully automated if the parameters of the transformation are known. Before repeatability was used, user input was required, for example to determine the *ground truth* of what should be a feature.

To calculate the repeatability, the detected interest points in two images have to be compared, where the images are related by a transformation t and are called the reference image I_d and the transformed image $I_{t(d)}$. According to the original definition, an interest point in the reference image is repeated when one is also found at the corresponding point in the transformed image. Since the detector can have a certain deviation, the correspondence is not expected to be in the exact same position, but there is a small margin of error ϵ where the point is allowed to be, and is said to be within the neighbourhood around the point. But it does not seem reasonable to use the same neighbourhood size for interest points of all sizes and it would be more appropriate to use a margin that is relative to the sizes of the keypoint.

Mikolajczyk et al. [36] addressed this issue and improved Schmid's definition, while generalising it for elliptical interest points by calculating the area of intersection between two ellipses. The overlap error ϵ_o is relative to the size of the interest points and is defined as

$$\epsilon_o(p_i, p_j) = 1 - \frac{p_i \cap p_j}{p_i \cup p_j}, \tag{69}$$

where $p_i \in P_d$ and $p_j \in P_{t(d)}$ are the regions of the interest points. As in the original paper, a set of points is considered to be a match if the overlap error is at most 40%. But by using the overlap error to find matches an unfair advantage is given to detectors that find larger interest points. The number of matches, and so the repeatability, can be increased simply by scaling up the size of the points. To counter this bias, for every comparison the two interest points are rescaled, such that the smaller point has a radius of 30 pixels. By matching features in this manner, only the position and the relative difference in size between the two keypoints dictates which features match. However, there is a problem with matching features like this. When comparing two very small keypoints, for example of radius 1 pixel, that are located at a distance of 10 pixels from each other and don't overlap at all, the result should not be a match. But with the given definition, the two keypoints are scaled up to radius 30 pixels and do actually result in a match. Therefore, an additional criterion is defined as

$$|p_i - p_j| \leq 4r_{min}, \tag{70}$$

where r_{min} is the smallest of the two radii. Hence, two keypoints of radius 1 that are more than 4 pixels away from each other do not give a match.

To calculate the repeatability r, the set of corresponding points C has to be determined. This is done by matching every pair of points as described above and calculating the number of correspondences as

$$C = \{(p_i, p_j) | \epsilon_o(p_i, p_j) \leq 0.4, p_i \in P_d, p_j \in P_{t(d)}\}. \tag{71}$$

Note that every keypoint may only be matched once. Both Schmid and Mikolajczyk use the ratio between this set and the minimum number of interest points in the two images to calculate the repeatability as

$$r = \frac{|C|}{min(|P_d|, |P_{t(d)}|)}. \tag{72}$$

There is another issue with this definition of the repeatability, that is best illustrated with an example. If the reference image contains a single keypoint, the transformed image has 99 keypoints and the single point in the reference image has a match in the transformed image, the resulting repeatability is 1, even though there are 98 keypoints in the transformed image that don't have a match. To counter this problem, Ehsan et al. [10] made an adjustment to the calculation of the repeatability by defining it as

$$r = \frac{2 \cdot |C|}{|P_d| + |P_{t(d)}|}, \tag{73}$$

where each correspondence counts as 2 points, one in each image, and instead of the minimum, the sum of the number of keypoints in both images is used to divide the number of correspondences by. In the example, the resulting repeatability is 0.02, which seems more appropriate, since most keypoints don't have a match.

Since the number of keypoints affects the repeatability score, as more keypoints result in a higher probability of finding a match, the number of keypoints is fixed for every comparison. This can be easily done by selecting K_k keypoints with the highest response value. In general, a high response value means a high confidence that a point is a good keypoint and every method used in the experiments assigns a response value to keypoints. The repeatability was computed with $K_k \in 300, 600, 1000$. If, for a certain image a detector found less than K_k keypoints, the results of that comparison was omitted.

5.2 Descriptor Matching

While repeatability gives a good indication of the quality of the interest points with respect to invariance and covariance with a given set of transformations, it is not necessarily a perfect indicator of good matching performance. If a method has a very low repeatability it is unlikely to find many correct matches, since very few keypoints have an actual correspondence. On the other hand, a high repeatability does not automatically result in a good matching performance. Therefore, the descriptors can be matched like they would be in a practical application, and the results will give an indication of how well the methods perform. However, the results of the matching experiments depend on the type of descriptor used. Some detectors have a coupled descriptor, while others don't. Since it is impractical to test different combinations of detectors and descriptors, a single descriptor was chosen for each detector (see Table 3).

To see how well each detector-descriptor pair performs, several values are calculated for pairs of images related by a known transformation, similar to Mikolajczyk and Schmid [35]. As with the repeatability test described in the previous section, the set of *correspondences* C are calculated by doing an intersection test between every pair of keypoints, and a pair is a correspondence if the overlap error $\epsilon_o < 40\%$. Then the descriptors in the reference image and transformed image are matched with a brute force matcher to find the total set of *matches* M. The matcher calculates the distance between the descriptors and a match if found if a pair is each other's nearest neighbour. The type of distance measure used depends on the descriptor and is listed in Table 3. Finally, from the set of matches the set of *correct match* M_+ is found by checking if the keypoints of the two descriptors are correspondences as well, and thus overlap each other by at least 60%. Then, the set of *false matches* $M_- = M - M_+$. To plot the findings, two new values will be calculated from these collected results. First the recall is calculated as

$$\text{recall} = \frac{|M_+|}{|M|}, \tag{74}$$

which determines how many of the found matches are actually correct. Then the precision is calculated as

$$\text{precision} = \frac{|M_+|}{|C|}, \tag{75}$$

that determines what portion of the actual correspondences are found by the method. For a feature detector to perform well, both of these measures need to by high. Having a high recall and a low precision means that the matches found by the detector are correct, but it misses many matches. In the extreme case it may only find a single correct match, while there are many potential matches. Having a low recall and a high precision means that a detector finds a lot of matches that are correct, but also finds many matches that are not correct.

5.3 Transformations

Features are extracted from images so that they may be matched to features found in other images. In the example of image retrieval, the goal might be to find an image of an object in a large database of images, given a reference image. However, the object in the reference image may look different from the one in the target image, as it is rotated, blurred, under different lighting conditions etc. The two images are separated by a set of transformations and this has an impact on the matching success. Therefore, it is important to determine the performance of detectors with respect to these transformations. For this task, the parameters of the transformation need to be known, so that it can be determined computationally whether matches are correct.

The transformation parameters are included in some data sets, which are designed especially for repeatability experiments, which usually come in the form of homographies. Homographies can also be calculated, which is called homography estimation. Recently, neural networks have been used to accomplish this task as well [7, 40]. The downside of using homographies is that they give the transformation of a plane, so unless the content of the image is a flat surface, the homography is not entirely accurate. Some data sets contain 3D information, which can be used to transform each keypoint with high accuracy, but these data sets are scarce and their application is not trivial. One could also generate pairs of images from 3D scenes, for which the transformations can be precisely controlled, but the downside of this is that the resulting images may not accurately represent real-life images and so the results could be unreliable.

Another method that can be used on any data set is to manually apply transformations to images, in which case the parameters of the transformations are already known and can be used to calculate the homography. This is very easy to implement and any data set can be used. Although not every transformation can be manually applied to images, this method has been thoroughly used and has been found to give a good representation of the detector performance. The transformations that have been manually applied to calculate repeatability and matching performance will now be discussed.

5.3.1 Rotation

An important property for feature detectors to have is rotational invariance, as objects in images are often at least somewhat rotated. Input images are rotated by steps of θ, the angle in degrees, and the keypoints in the original image are compared to those in the rotated image. Images and keypoints can be rotated by applying the transformation matrix to the coordinates, which in case of rotations is defined as

$$R(\mathbf{x}, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ 1 \end{bmatrix}, \tag{76}$$

where $\mathbf{x} = \begin{bmatrix} x_u & x_v \end{bmatrix}^\top$ are the coordinates of the point being rotated. However, the points first need to be translated, such that they rotate around the origin. This is done by the translation

$$T_1(\mathbf{x}, \mathbf{c}) = \begin{bmatrix} 1 & 0 & -c_u \\ 0 & 1 & -c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ 1 \end{bmatrix}, \tag{77}$$

where $\mathbf{c} = \begin{bmatrix} c_u & c_v \end{bmatrix}^\top$ is the centre of the image. After rotation the points are translated back, but the bounding box of the rotated image is larger than that of the original and to make the whole image fit the points must be translated back by a different amount. The new translation is

$$T_2(\mathbf{x}, \mathbf{c}, \theta) = \begin{bmatrix} 1 & 0 & \tilde{c}_u \\ 0 & 1 & \tilde{c}_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ 1 \end{bmatrix}, \tag{78}$$

where $\tilde{c}_u = ||\cos(\theta)c_u|| + ||\sin(\theta)c_v||$ and $\tilde{c}_v = ||\sin(\theta)c_u|| + ||\cos(\theta)c_v||$. The final transformation matrix that rotates points around the origin and translates to the new centre is found by concatenating these transformations as

$$R_0(\mathbf{x}, \mathbf{c}, \theta) = T_2 R T_1. \tag{79}$$

Some manually defined feature detectors, such as SURF, are known to work well with some angles of rotation (mainly every $\frac{1}{2}\pi$ or $\frac{1}{4}\pi$), but not others. This is usually caused by the shape of the filters, as SURF, for example, uses box filters. Therefore, in the experiments repeatability is calculated for rotations with steps of $\theta = \frac{1}{12}\pi$.

5.3.2 Scale

Another important transformation that detectors should be covariant with is scaling. An object may be projected in two different images at two completely different scales. Being covariant up to a certain scale can therefore be very important, depending on the application. The transformation matrix for scaling is defined as

$$S(\mathbf{x}, \sigma_s) = \begin{bmatrix} \sigma_s & 0 & 0 \\ 0 & \sigma_s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ 1 \end{bmatrix}, \tag{80}$$

where σ_s is the scale. The value of σ_s was varied from 0.5 to 3, with steps of 0.25. For each scale, the keypoints in the scaled image was compared to those in the unscaled original image.

5.3.3 Blur and Noise

Two more transformations that are more anomalies of photography than actual transformations are blur and noise. Most images will have at least some degree of blur and noise and being robust to these transformations can be important if the target images contain either or both. The coordinates of points are not changed by these transformations and therefore the transformation matrix is simply the identity I. For blurring images, Gaussian blur was applied with a standard deviation σ_b of 0.5 to 5, with steps of 0.25. To apply noise, images values were converted to lie within the range $[0, 1]$. Then, a random value drawn from the normal distribution was added to each pixel. The standard deviation σ_n was varied from 0.1 to 1, with steps of 0.1.

5.3.4 Compound

Each of the previously defined transformations can occur in a pair of corresponding images and each of the experiments give an idea of how well the detectors respond to each of the transformations. Yet, in practice these transformations will generally not exist in isolation, but some degree of each will be present. Therefore, it might be interesting to determine the performance of the detectors when multiple transformations are applied at once. This will be referred to a a compound transformation. Compound transformations consist of rotation, scale, blur, noise and an additional random change in contrast and brightness. The transformation matrix can be found by simply multiplying those of rotation and scale as

$$C(\mathbf{x}, \mathbf{c}, \theta, \sigma_s) = R_0(\mathbf{x}, \mathbf{c}, \theta) S(\mathbf{x}, \sigma_s). \tag{81}$$

Transformations were sampled randomly, such that $\theta \in [0, 2\pi)$, $\sigma_s \in [\frac{2}{3}, 2]$, $\sigma_b \in [\frac{1}{2}, 3]$ and $\sigma_n \in [0.1, 0.4]$. The contrast and brightness changes were applied to image I to get the transformed image $\hat{I}$ according to the function

$$\hat{I}(x, y) = \alpha I(x, y) + \beta, \tag{82}$$

where α is a multiplicative factor in the range $[\frac{1}{2}, 2]$, and β is an additive factor between 0 and 100, assuming pixel values are in the range $[0, 255]$.

5.4 Non-Redundancy & Coverage

While repeatability and matching scores represent a qualitative measure on a per keypoint basis, there are other criteria that say something about the quality of the keypoints in relation to each other. Two such criteria are *non-redundancy* [43] and *coverage* [9].

Non-redundancy is a measure of how much the keypoints overlap and is defined as

$$\text{non-redundancy} = K_{\text{nr}}/K, \tag{83}$$

where K is the cardinality of the keypoints and K_{nr} the non-redundant cardinality. In order to calculate the non-redundant cardinality, a mask function $f_k(\mathbf{x})$ is applied to each keypoint k as

$$f_k(\mathbf{x}) = \begin{cases} K e^{-\frac{1}{2\varsigma^2}(\mathbf{x}-\mathbf{x}_k)^T \Sigma_k^{-1} (\mathbf{x}-\mathbf{x}_k)}, & \text{if} (\mathbf{x}-\mathbf{x}_k)^T \Sigma_k^{-1} (\mathbf{x}-\mathbf{x}_k) \leq \rho^2, \\ 0, & \text{otherwise}, \end{cases} \tag{84}$$

where $\mathbf{x}_k$ is the keypoint centre, Σ_k a matrix containing its shape and ρ and ζ describe the window of the keypoint descriptor. The parameter ρ corresponds with the size of the window and ζ is the standard deviation of the Gaussian window, if one is used. The values of these parameters differ per method and can usually be found in the paper. The values of the masks are normalised, such that the sum for each keypoint is 1. After the mask function is applied to each keypoint, the cardinality and non-redundant cardinality are calculated as

$$K = \int_\Omega \left(\sum_{k \in \mathcal{K}} f(\mathbf{x}) \right) d\mathbf{x}, \tag{85}$$

$$K_{\text{nr}} = \int_\Omega \left(\max_{k \in \mathcal{K}} f(\mathbf{x}) \right) d\mathbf{x}, \tag{86}$$

where Ω is the image domain. In other words, the cardinality is calculated by taking the sum of all masks, which is true because the sum of each mask is 1. The non-redundant cardinality is then the maximum value per pixel.

Related to non-redundancy, coverage describes the spatial distribution of the keypoints. This is done by measuring the mean distance for each pair of keypoints. However, simply taking the arithmetic mean results in a high sensitivity to outliers, which may skew the results. Therefore, the coverage is calculated using the harmonic mean H_i, given a reference keypoint i, which is defined as

$$H_i = \frac{K-1}{\sum_{j=1, j \neq i}^{K} \frac{1}{d_{ij}}}, \tag{87}$$

where K is the number of keypoints and d_{ij} is the distance between keypoints i and j. This is done with each keypoint as a reference and the coverage is calculated as

$$\text{coverage} = \frac{K}{\sum_{i=1}^{K} \frac{1}{H_i}}. \tag{88}$$

Keypoints that are detected at the exact same location at different scales are excluded from the calculations.

5.5 Image Retrieval

While the previously discussed measures define some qualitative properties of the detected keypoints, it is not guaranteed that detectors and descriptors that have a good repeatability and matching performance will also perform well in practical applications. In order to give some idea about the practical performance of the conventional and neural network-based methods, there will be an implementation of an image retrieval application.

In real-world image retrieval systems, the database may contain hundreds of thousands, or even millions of images. While the comparison of every pair of descriptors is a very precise method, it is too costly to perform with these numbers of images. One method to achieve real-time performance is the *bag-of-words* (BoW) model [56, 70], which is used in these experiments. The descriptors of all images in the database are (partially) used to perform *k-means clustering*. The descriptors are fitted to a predetermined number of *clusters* (words), and this set of clusters is called the *dictionary*. Every descriptor can then be assigned to one of the cluster simply by calculating the distance between the vectors. By doing this to every descriptor in an image, the number of times each word occurs in that image can be stored in a histogram of occurrences. This results in a single histogram vector per image, which is now much easier to compare. The dictionary and the histograms can be stored in a database and given a query image, its histogram can be calculated and compared to those in the database. The comparison between histograms can again be done by calculating a simple distance measure, which in these experiments is the L1-distance.

Figure 10: Some example images from the used data sets. Two examples from the Oxford data set are on the left, from the ILSVRC 2017 test set in the middle and the ALOI data set on the right.

The bag-of-words trainer and histogram extractor implemented in the OpenCV library are used. The number of database images is set to 250 000 and the number of clusters to 8 000. The dictionary is calculated by taking the descriptors of the 10 most prominent keypoints in each image, resulting in 2 500 000 descriptors. For the histograms, 200 keypoints per image are used. In the final step, 2 500 query images are constructed by applying a random scale, rotation, blur, noise, contrast and brightness. The query images are then compared to the database and all database images are ranked by their distance to the query image. If the highest ranked image the the same as the untransformed query image the result is considered correct. In order to get a more detailed look at the performance, the rank of the correct images are also recorded.

The transformations used in the image retrieval are slightly different than those used in the repeatability and matching experiments. Rotations are still in the range $[0, 2\pi]$ and scale changes are in the range $[\frac{2}{3}, 2]$. For blurring, σ_b is parameterised between $[0, 2]$. For noise, σ_n lies in the range $[0, 0.2]$. For the contrast and brightness changes, α was randomly generated within the range $[0.85, 1.15]$ and β in the range $[-20, 20]$.

5.6 Data Sets

The repeatability experiments were performed on two different data sets. The first is the Oxford Graffiti data set [36], which contains sets of images, between which there is one of several different transformations. The transformations include, amongst others, change in viewpoint, change in illumination and progressive blurring. Homographies between sets of images are given, which describe the change in viewpoint in the images. A homography is nothing more than a transformation matrix that maps 2-D points from one image to the other. The second is part of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2017 test data set [49]. Of the total of 5 500 test images, the first 2 000 were used to manually create pairs of transformed images. The transformations applied are those described in Section 5.3. The non-redundancy and coverage experiments were performed on the complete data set of 5 500 images. Some examples of the Oxford data set and the ILSVRC 2017 test data set can be seen in Figure 10 on the left and in the middle, respectively.

To test the detectors' performance with changing illumination, experiments were performed on the Amsterdam Library of Object Images (ALOI) [12]. This data set is a collection of images made of 1 000 objects under varying illumination and viewpoint angle. For each object, images were made with 3 different cameras and 8 different lighting conditions, to give a total of 24 images. For the purpose of these experiments, only pairs of images made with the same camera, but under different lighting conditions were compared. The setup contained 5 light sources illuminating the objects from different angles, from left to right. The first 5 illumination conditions were with one of the five light sources turned on. The next two were with two light sources from the left and two from the right, respectively. The final lighting condition was with every light source turned on simultaneously. In the experiments, this last lighting condition was used as the reference and each of the other 7 lighting conditions were compared to this reference condition. Some examples of this data set are shown in Figure 10 on the right.

Finally, the image retrieval experiments were performed on the large ILSVRC 2012 training data set, containing more than 1.2 million images. Like in the 2017 test data set, there are many different categories of images, in all different shapes and sizes. In order to make the computation time feasible, only 250 000 images of the data set were used. Mainly the computation of the methods using neural networks took a very long time, even with parallel computation on the GPU.

5.7 Implementation Details

5.7.1 Conventional Feature Detectors & Descriptors

All of the experiments were implemented in C++. As some of the networks were implemented in python, the Boost.Python library was used to call python functions from C++. Most image manipulation was performed using the OpenCV 3 library, which is available for both C++ and Python. This library also contains many implementations of local feature detectors, such as SIFT and SURF. All of the implementations of the conventional feature detectors used in the experiments were the ones contained in the OpenCV library. The parameters of each method were set to values as close as possible to those given in the original papers, and otherwise to the standard values present in the library. As CenSurE only defines a feature detector without orientation assignment, SURF's implementation of orientation assignment and SIFT's descriptors were used in the experiments.

5.7.2 The Quad Detector

The implementations of the quad detectors described in Section 4.3 for the purpose of these experiments were made using the PyTorch neural networks library, made for Python. The network architecture was one from the original paper, which is a non-linear version with 32 convolutional filters in the first layer, followed by an ELU activation layer, a dropout layer and finally a dense layer. The networks were trained with the DTU Robot Image Point Feature Data Set [1], using only illumination changes as correspondences. Attempts to use viewpoint changes given the 3D point data were not very successful. The networks trained with illumination changes performed approximately as well or even better than the ones trained with viewpoint changes, and therefore only the former were used in the experiments. The networks were trained for 2 000 epochs, each consisting of 10 240 newly generated quadruples and with a batch size of 256. Optimisation was done with the Adam optimiser. The resulting filters are depicted in Figure 11.

The quad detector only defines the calculation of a score map and the rest of the pipeline, the orientation assignment and feature description, are done according to the SIFT procedure. Therefore, the OpenCV implementation of SIFT was used and modified accordingly. The main difference with SIFT is the size of the images in the scale-space. Where SIFT has equally sized images in each octave, comparing two neighbouring images is simple. However, the filters of the quad detector have a constant size and therefore, the image size needs to be adjusted to calculate at different scales. An octave consists of 5 scales and each scale is $\sqrt[3]{\frac{1}{2}}$ times smaller than the previous and each octave is twice as small as the last. By choosing these factors, the scales over which are searched are evenly spread out. Because neighbouring scales are different sizes, comparing them is not as straightforward as in SIFT. Several approaches were tested, such as resizing every image in an octave to the largest scale in that octave or calculating the position in neighbouring scales with bilinear interpolation. The best method, however, was for each scale to resize the dimensions of its neighbouring scales to that of itself. In other words, for every scale, the larger neighbour was scaled down and the smaller neighbour was scaled up. Since the largest and smallest scales only need to be scaled in one direction, this resulted in 3 more rescalings per octave. There was, however, no noticeable difference in computation time.

5.7.3 LIFT

In the experiments, there are two different implementations of LIFT used, both of which are available on GitHub. The first version includes all 3 stages of the method, but the orientation estimator is trained on images that do not contain large rotations. Therefore, it does not work well when images do contain large rotations. The orientation assignment of SIFT, also included in the code, was used instead of the orientation network. This version was used for the repeatability and matching experiments. A second version, in which the orientation estimator was trained with images containing rotations, was used for the retrieval experiments. Ideally, the second version would have been used for all experiments. However, the second version was not found until after the repeatability and matching experiments had been performed and time constraints did not allow a recalculation of the results.

5.7.4 The Covariant Detector

An implementation of the covariant network can be found online. However, the networks provided only detect keypoint locations without scale and orientation. A complete keypoint detector was implemented by applying the scale-space and orientation assignment of SIFT. This detector did however not produce stable keypoints and was therefore omitted from the experiments. Other attempts at training a detector that outputs a location and scale were also without success.

5.7.5 VGG16 Descriptors

Recently, some well-known pre-trained networks, such as VGG and AlexNet, have been used to generate image descriptors [39, 58]. These networks have already been trained on many training images and they have good filters that are known to perform well. The first several layers have been shown to be appropriate for this task, as these layers are trained to recognise local structures. There are different methods to extract features from the network outputs. However, in this case the network outputs are used directly as feature descriptors.

To test whether such a pre-trained network can produce qualitative descriptors, some of the experiments will be performed with two more detector-descriptor pairs. For SIFT and the quad detector, both of which use the SIFT descriptor, the descriptor calculation part will be replaced by the output of the VGG16 network's 7^{th} layer. All other components will be left intact, meaning that the detection and orientation assignment will be exactly the same as in the original methods. After patches are cropped and rotated according the the location, scale and orientation of the keypoints, they are resized to 40×40 pixels and fed into the VGG16

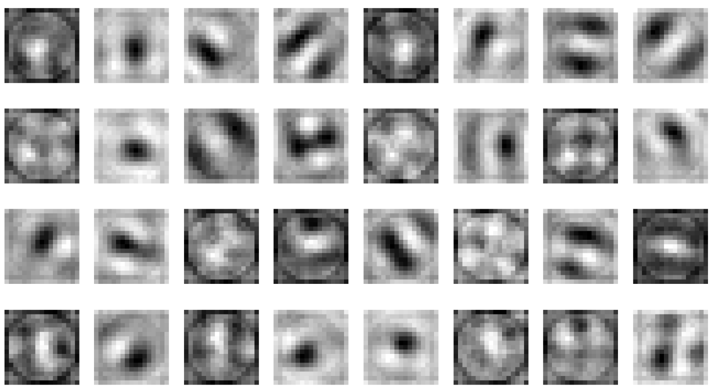


Figure 11: The filters in the first layer of the non-linear quad detector. The network seems to look at different gradients in order to determine which points are covariant with the given transformations.

network (as shown in Table 5). The output of layer 7, which has depth 256, is then used as the descriptor. The padding in each layer is set to 0, such that the volume shrinks to size $256 \times 1 \times 1$. The results will show if these descriptors improve the robustness of the methods over the original, hand-crafted SIFT descriptors. Since the keypoints are the same, only the matching and retrieval experiments will be done.

VGG16 Descriptor

Conv 1 $64 \times 3 \times 3$	Conv 2 $64 \times 3 \times 3$	Conv 3 $128 \times 3 \times 3$	Conv 4 $128 \times 3 \times 3$	Conv 5 $256 \times 3 \times 3$	Conv 6 $256 \times 3 \times 3$	Conv 7 $256 \times 3 \times 3$
ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	
	Max-Pool 2×2		Max-Pool 2×2			

Table 5: The first 7 convolutional layers of the VGG16 network used to calculate descriptors.

5.7.6 Feature Cardinality

As discussed in Section 5, to make a fair comparison, the number of features should be the same for every detector. For the repeatability and matching experiments, every detector detected as many features as possible. The sets of features, with cardinality K, were then sorted according to the response value. Each experiment was then performed 3 times, by using only the $\{K_n | n \in \{300, 600, 1000\}\}$ features with the highest response. This resulted in 3 different values for each experiment. Since the desired number of features depends on the specific application, any value for n may be preferable. Therefore, the values reported in the next section are the mean of the 3 different values. However, if a detector did not detect enough features for a certain image ($K < K_n$), the results for that image are omitted. For the manual transformations, the results are completely omitted if there were fewer than 100 images that resulted in enough features. For the non-redundancy and coverage experiments, only the cases with $K_n = 300$ were used, since the number of keypoints influence the results.

6 Results

The results of the experiments described in the previous section will be presented here. First, the results of the repeatability and matching experiments are considered together, as these measures are correlated. Next, we look at the results from the non-redundancy and coverage experiments. Then, the matching results of the SIFT and Quad detectors together with the VGG16 descriptor are compared to those with the original SIFT descriptor. Finally, the results of the image retrieval experiments are presented.

6.1 Repeatability and Matching

The repeatability and matching experiments were performed on the methods listed in Table 6. The repeatability scores of the Oxford data set are presented in Figure 12. The repeatability of the manual transformations applied to the ILSVRC data set, together with those of the lighting changes from the ALOI data set, can be found in Figure 13. Then, the recall and precision from the feature matching experiments can be found in Figure 14 for Oxford and in Figure 15 for the manual transformations. Now, a brief discussion of the results per method.

Method	Detector	Orientation Estimator	Descriptor
SIFT [31]	SIFT	SIFT	SIFT
SURF [3]	SURF	SURF	SURF
CenSurE [2]	CenSurE	SURF	SIFT
AGAST [27, 32]	AGAST	BRISK	BRISK
Quad [50]	Quad	SIFT	SIFT
LIFT [66]	LIFT	LIFT	LIFT

Table 6: The combinations of detector, orientation estimator and descriptor used in the repeatability and matching experiments.

SIFT

It is immediately noticeable that SIFT does rather poorly compared to the other methods when it comes to repeatability. In the Oxford data set, except in the bark set and with small changes in the graffiti set, SIFT has a relatively low repeatability. It does reasonably well with manual rotations and scaling, but worse than all other methods for the other transformations. However, when looking at the precision and recall, SIFT seems to do better, especially when it comes to scale and rotation changes. There does seem to be a sensitivity to blurring and noise, which may stem from the fact that SIFT uses blurring in the DoG to approximate the image gradients. Large amounts of blur in the image seem to interfere with the detection and matching.

SURF

On the other hand, SURF seems to dominate the repeatability for most data sets, except when it comes to scaling. SURF appears to be especially robust to blurring and noise, exactly where SIFT struggles to perform. The box filters may be responsible for this, as a bit of blurring or noise should not affect its output values. Unfortunately, this high repeatability does not necessarily translate to a good matching performance. SURF's matching scores are mediocre for scaling and rotation, although it does have a much higher recall for blur, noise and compound transformations than the other conventional methods.

CenSurE

While CenSurE is based on SIFT and SURF, the experiments reveal that it often does not work as well as either method. In some cases it does outperform one, but usually not both. In both repeatability and matching scores, it's mostly either at or near the bottom. However, this might be due to the combination of the detector, the orientation assignment and the descriptor that was used, since CenSurE only defines a detector without orientation assignment.

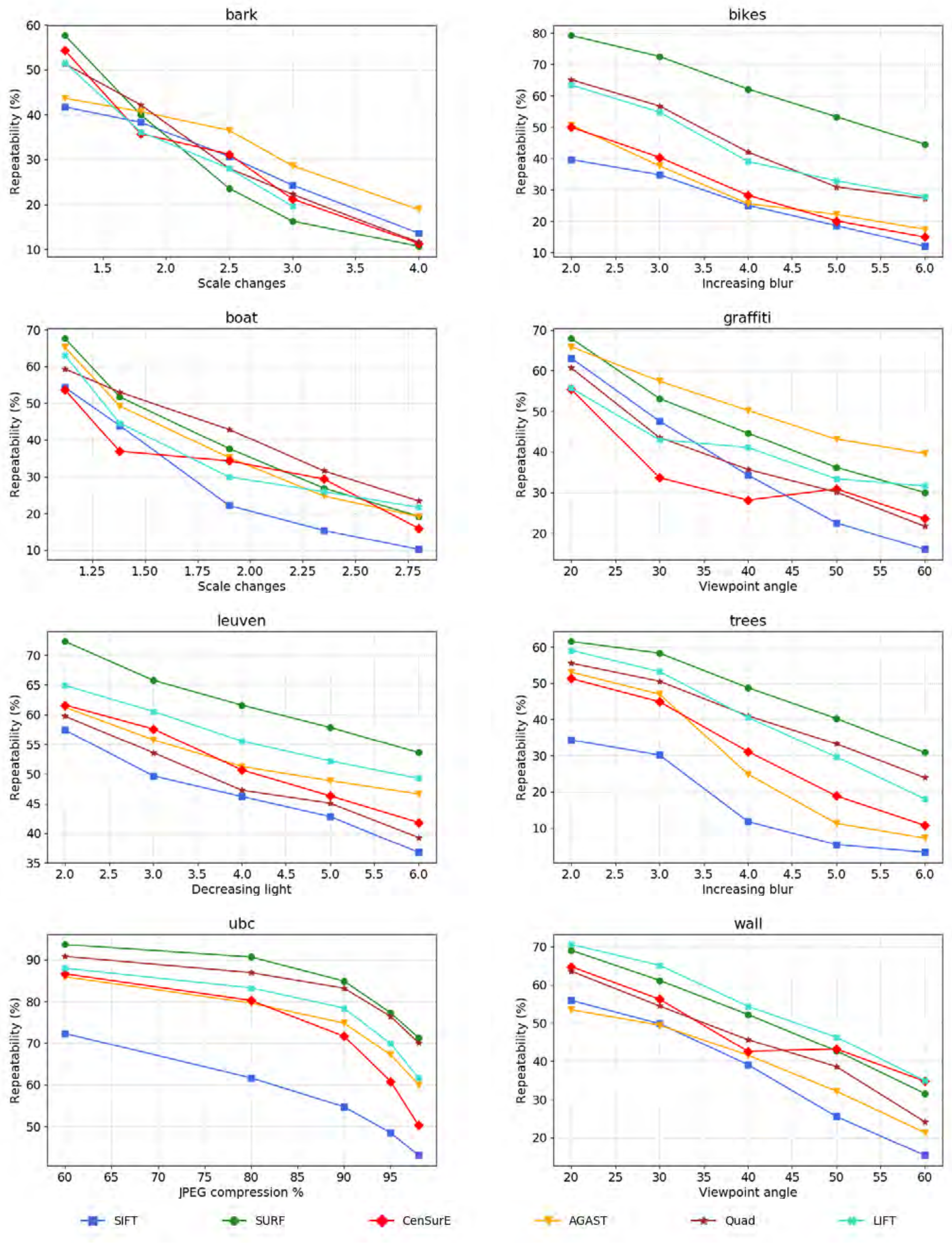


Figure 12: The repeatability scores of the oxford data set.

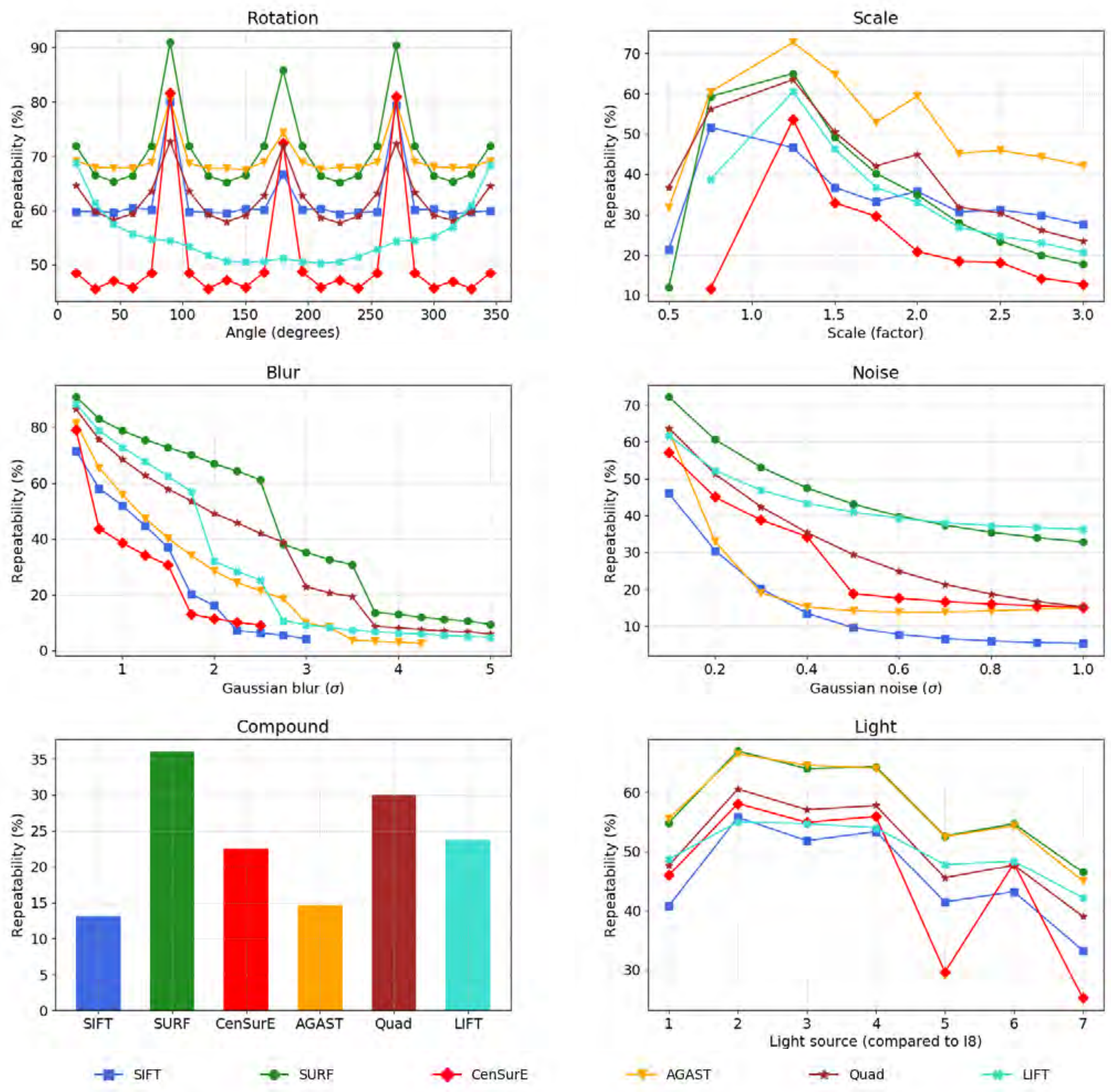


Figure 13: The repeatability scores of the manual transformations and the lighting changes.

AGAST

The results show that AGAST has a high repeatability when the relative values between neighbouring pixels stay the same, such as rotation and scaling. However, when the image structures are disrupted at a pixel-level, for example by noise or blur, the repeatability suffers. This could very well be because keypoints are found by comparing the values of pixels to those of their neighbours. Looking at the matching scores, AGAST does not do very well. However, for a method that does not work with gradients and by doing so gains a lot in terms of computational performance, this may not be very surprising. In some cases, such as the boat scene of the Oxford data set or the manual scale changes, AGAST outperforms some of the more complex methods.

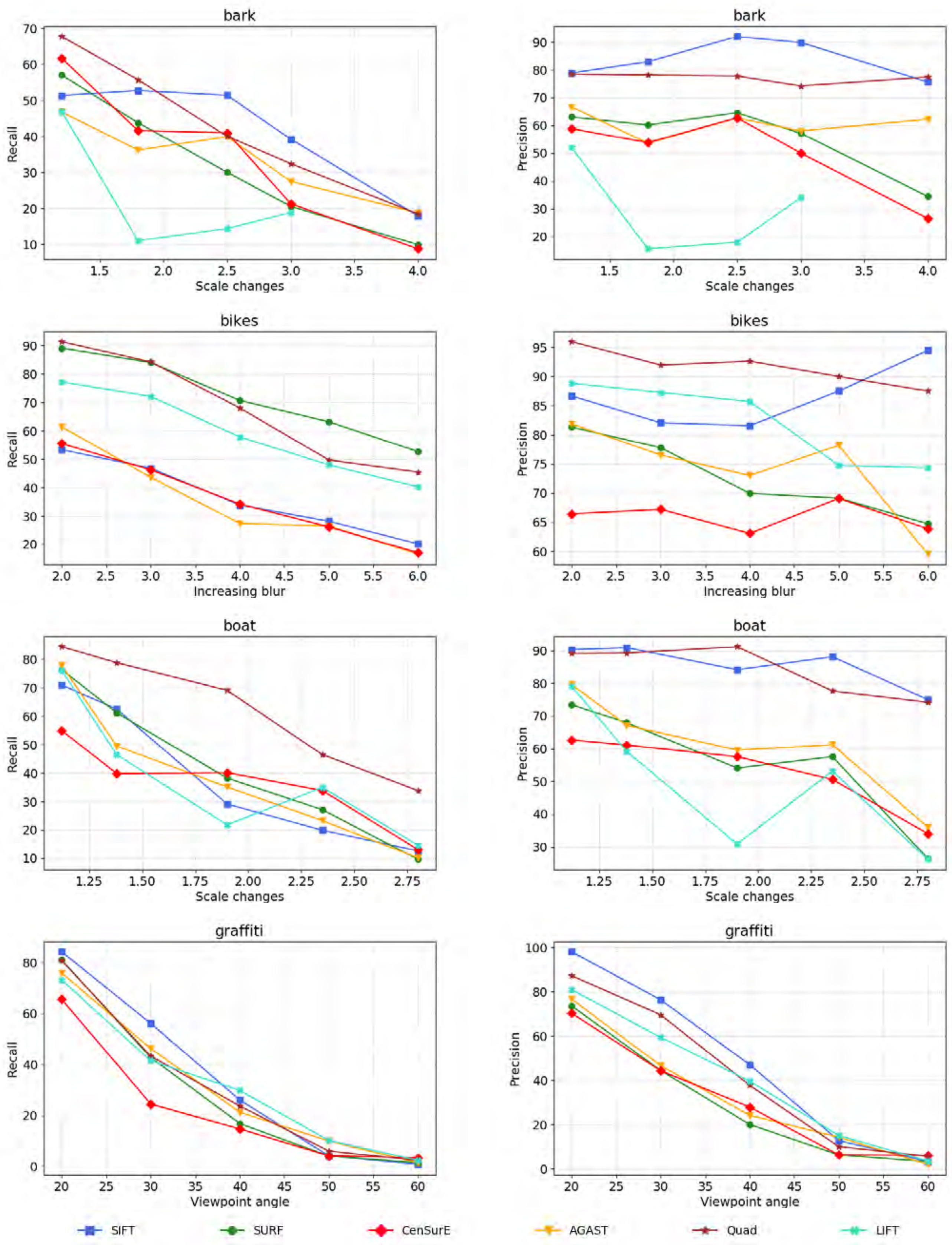


Figure 14: The matching scores of the Oxford data set.

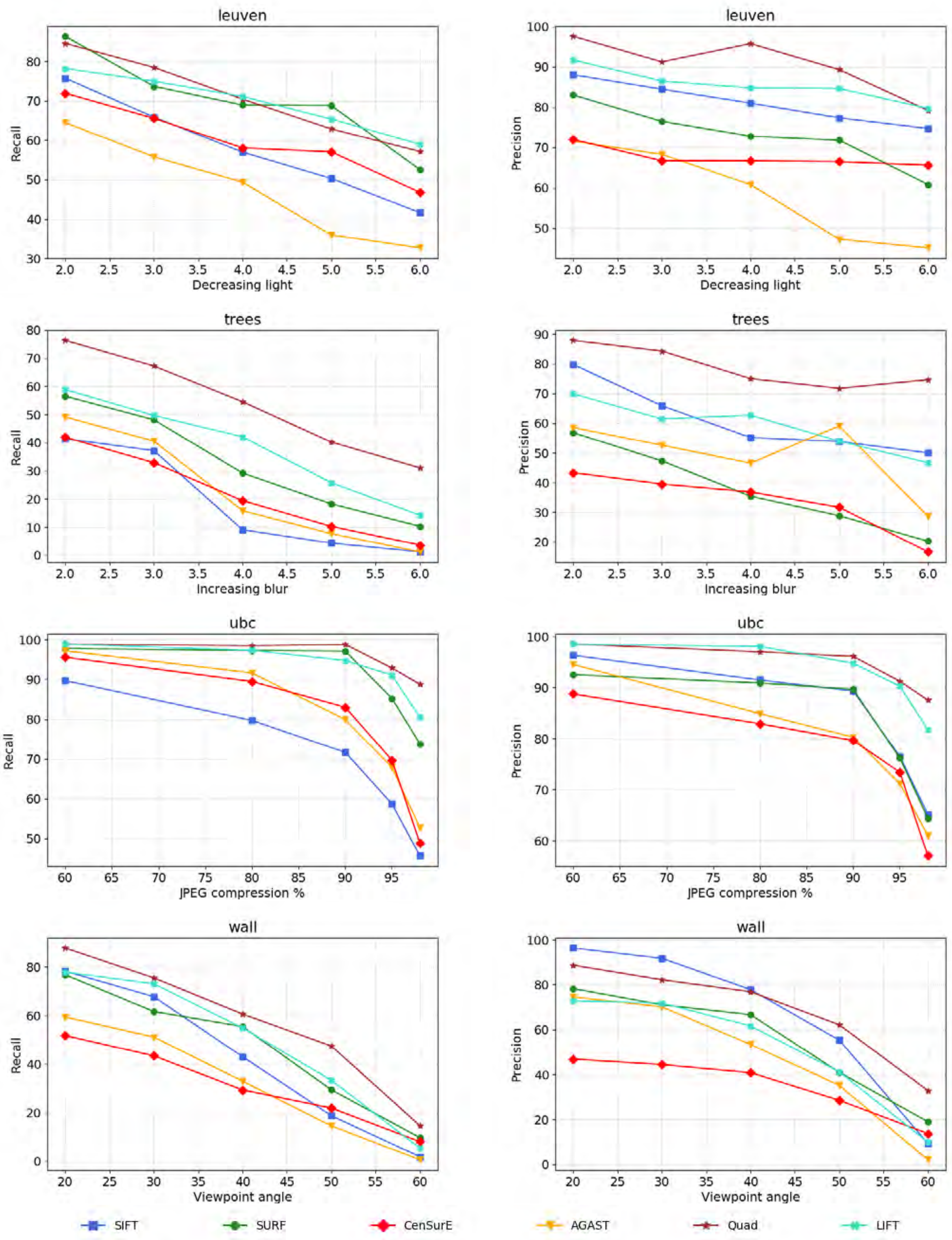


Figure 14: The matching scores of the Oxford data set.

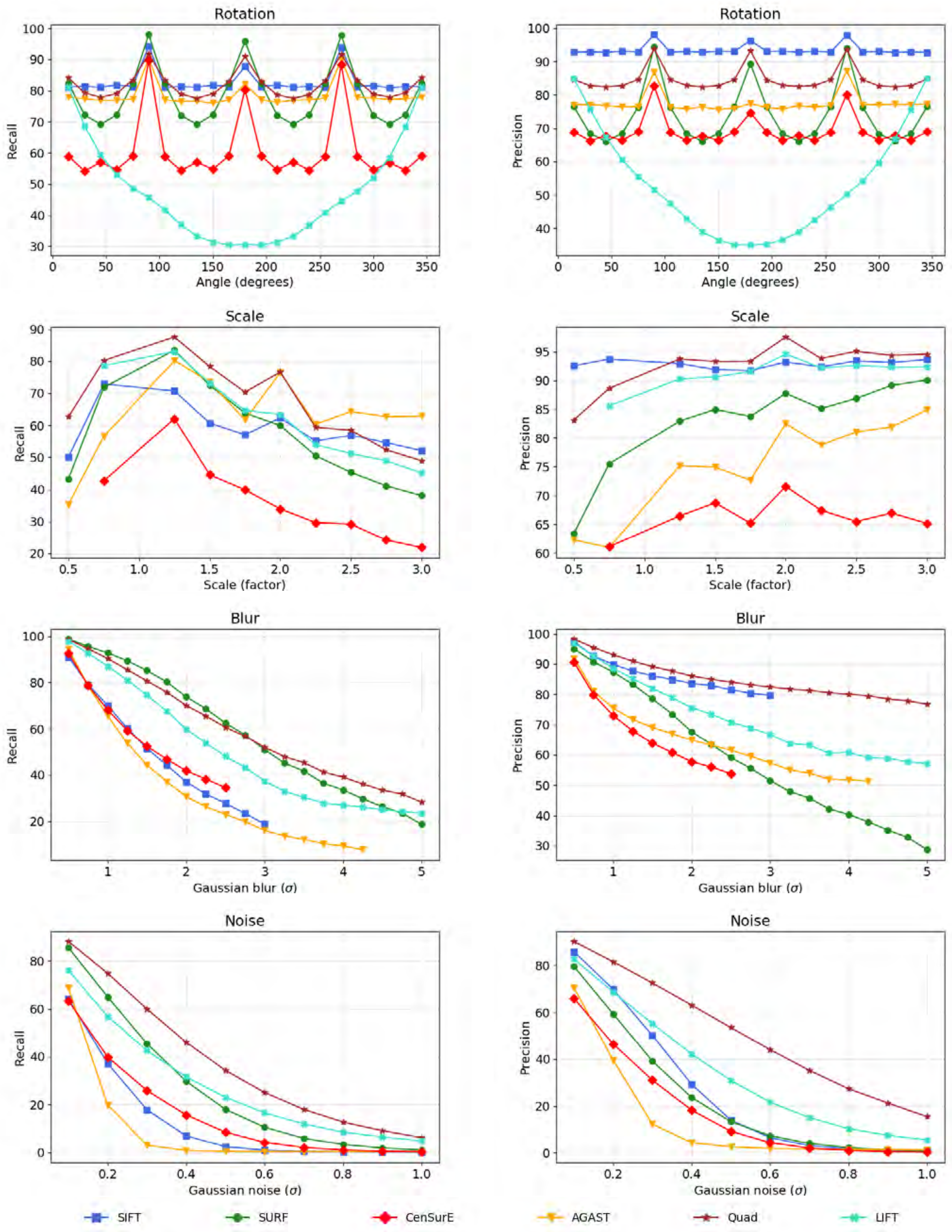


Figure 15: The matching scores of the manual transformations and the lighting changes.

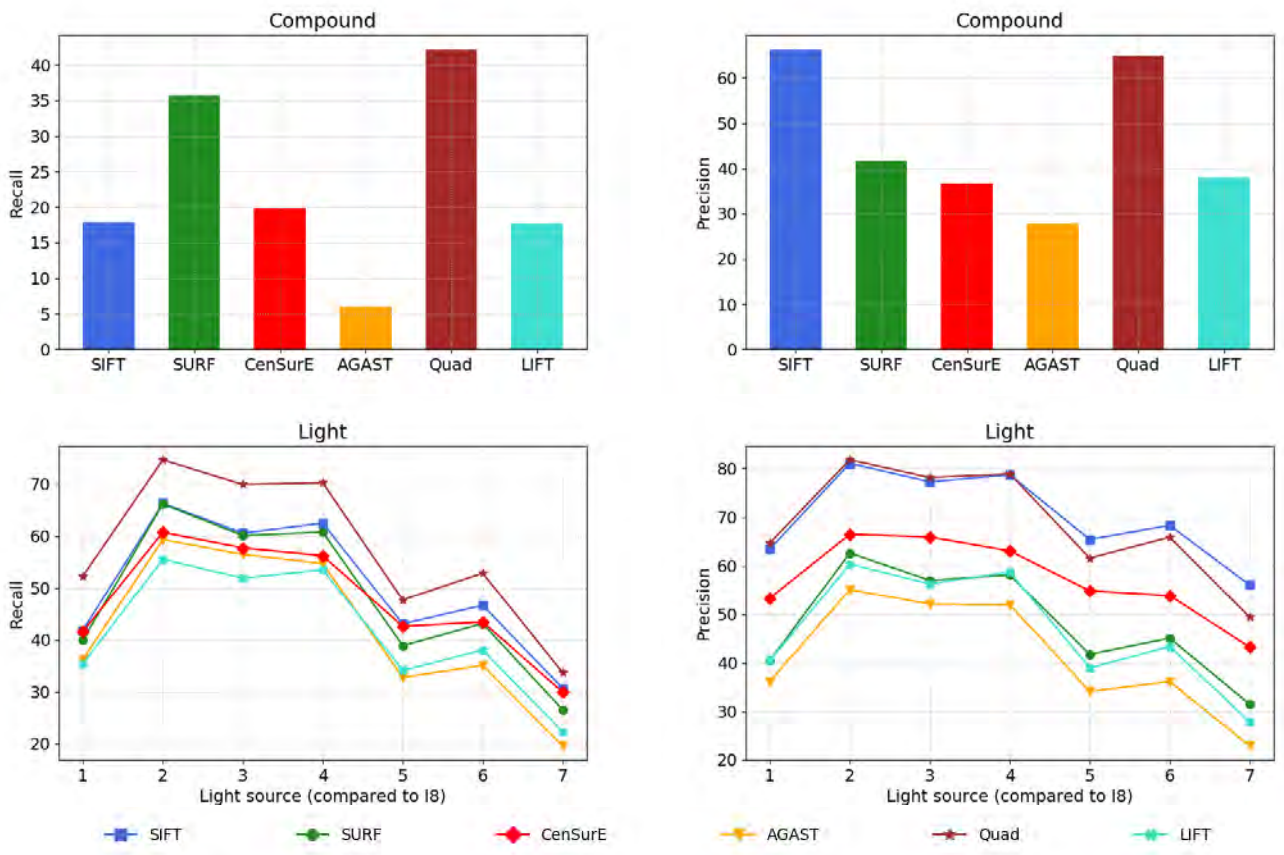


Figure 15: The matching scores of the manual transformations and the lighting changes.

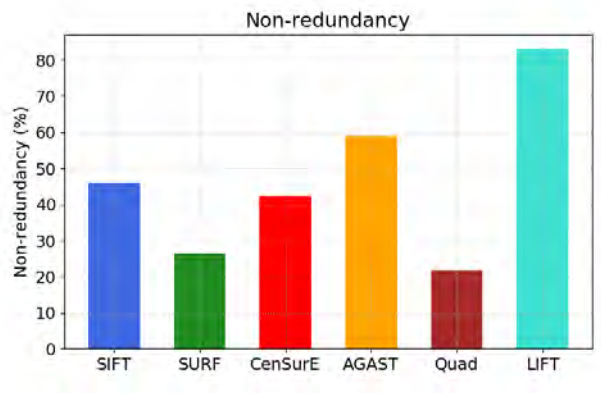
Quad

The non-linear quad detector with 2 layers gives reasonably high repeatability scores. In a few cases, such as in Oxford's leuven set and in the manual rotation changes, the repeatability is not as high as the other methods, but in general it's either in the middle or at the top. What is really interesting is the results of the matching experiments. Here, we can see that the quad network outperforms the other methods in a lot of the cases, sometimes together with SIFT or SURF, but more consistently over all of the experiments. It also performs very well in the compound transformations and the lighting changes. The question is whether these results will translate in a practical application.

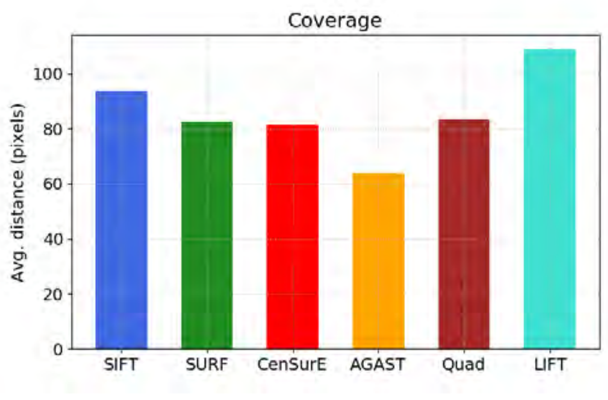
LIFT

The version of LIFT used in these experiments is the one without the rotation network, but with SIFT's rotation assignment. This may be why the performance is not very high with rotations, as can be seen in both the Oxford sets that contain rotations and the manual rotations. In both these cases LIFT yields low repeatability and matching results. Because the networks were trained without any rotational variations, some loss of performance may be expected. However, since SIFT orientations are assigned to keypoints, the drop in performance should not be very dramatic. The extreme loss of performance that can be seen for manual rotations in Figures 13 and 15 may result from the fact that all networks are trained together. What should be LIFT's strength is in this case its weakness. The networks are trained to work well together and replacing the orientation network may have resulted in the drop in performance that can be seen. For all other transformations, except lighting changes, LIFT performs reasonably well.

6.2 Non-Redundancy and Coverage



(a)

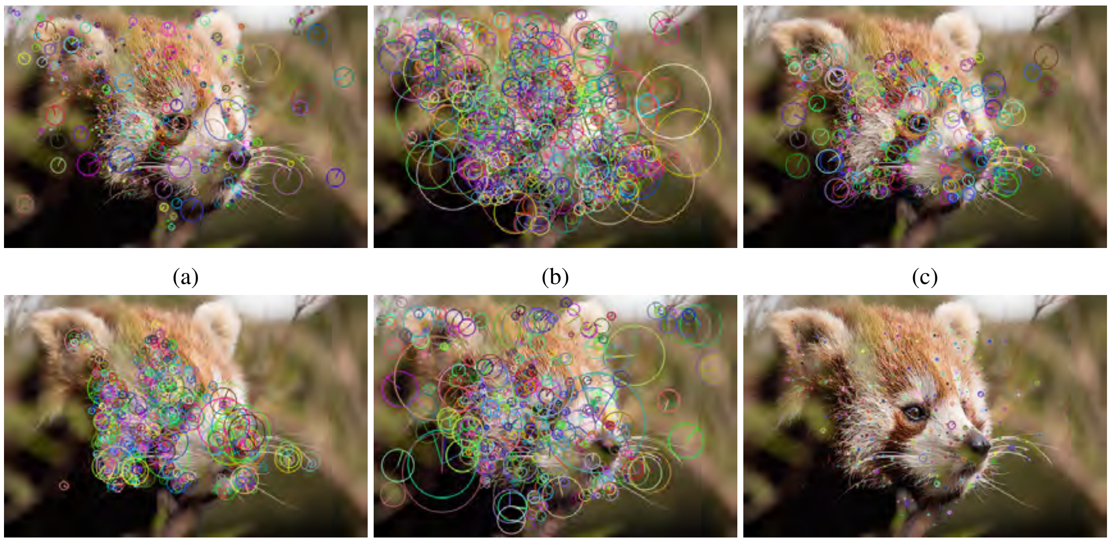


(b)

Figure 16: (a) Non-redundancy. (b) Coverage.

The combinations of detector, orientation estimator and descriptor in these experiments are the same as in the repeatability and matching experiments, shown in Table 6. Non-redundancy is a measure of how much (or how little) the keypoints overlap each other. A score of 100% means that the keypoints don't overlap at all and this means every keypoint adds new information. If a keypoint highly overlaps another, then its descriptors will not add any new information, as it encodes the same region. Therefore, it is desirable that a set of keypoints overlap as little as possible, such that the information contained in them is maximised. Coverage, on the other hand, shows how close the keypoints are to each other. A good detector will produce keypoints that are evenly spread out over the image.

The results of the non-redundancy experiments are shown in Figure 16a. In both instances LIFT has the highest score. After LIFT, AGAST performs best when it comes to non-redundancy and SURF and the quad detector have the lowest scores. AGAST has the smallest descriptors and SURF the largest descriptors, which heavily influences how much the descriptors overlap. Figure 17 shows an example of keypoints detected by



(a) (b) (c) (d) (e) (f)

Figure 17: Keypoints detected by (a) SIFT, (b) SURF, (c) CenSurE, (d) AGAST, (e) Quad and (f) LIFT.

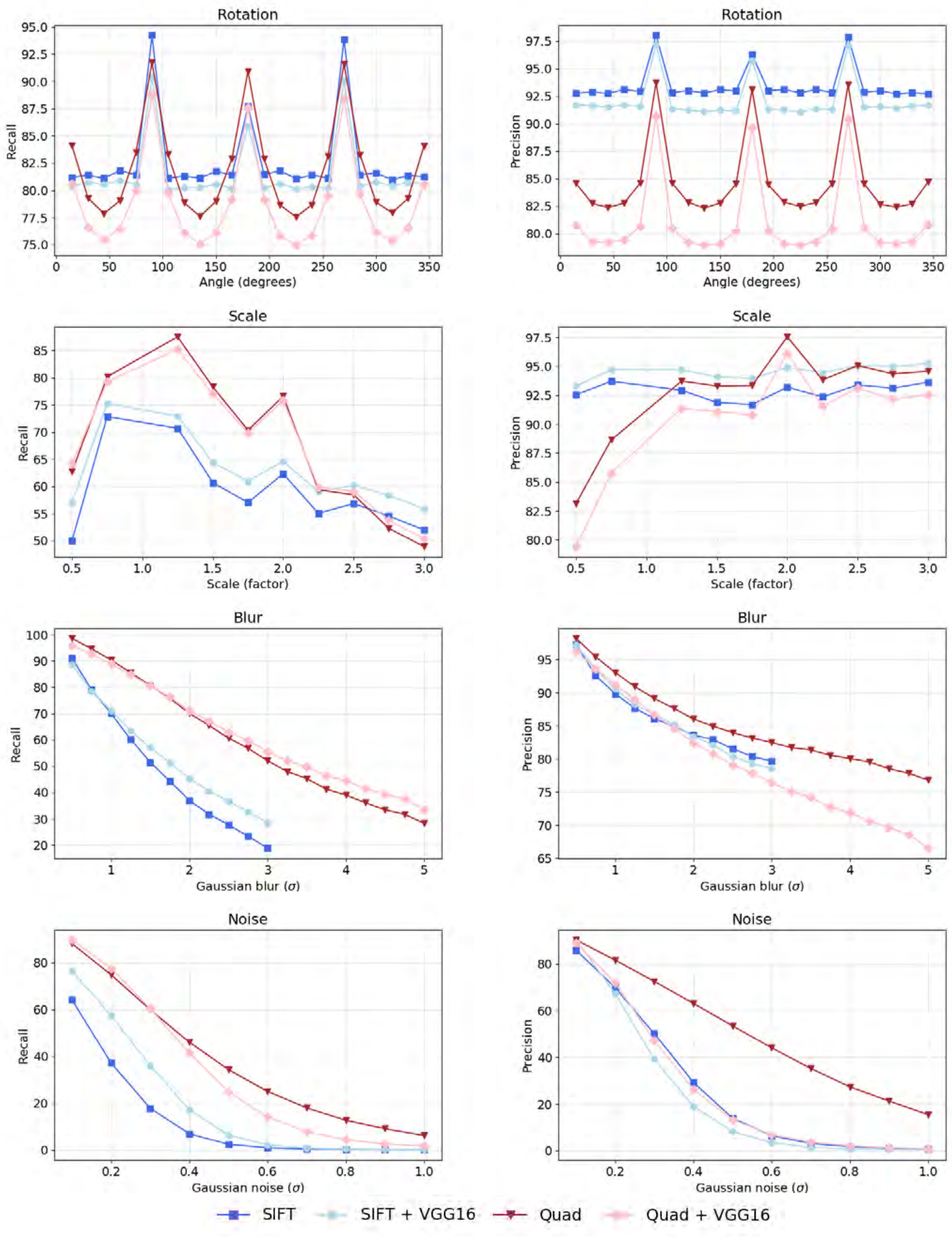


Figure 18: The matching scores, comparing the SIFT and VGG16 descriptors.

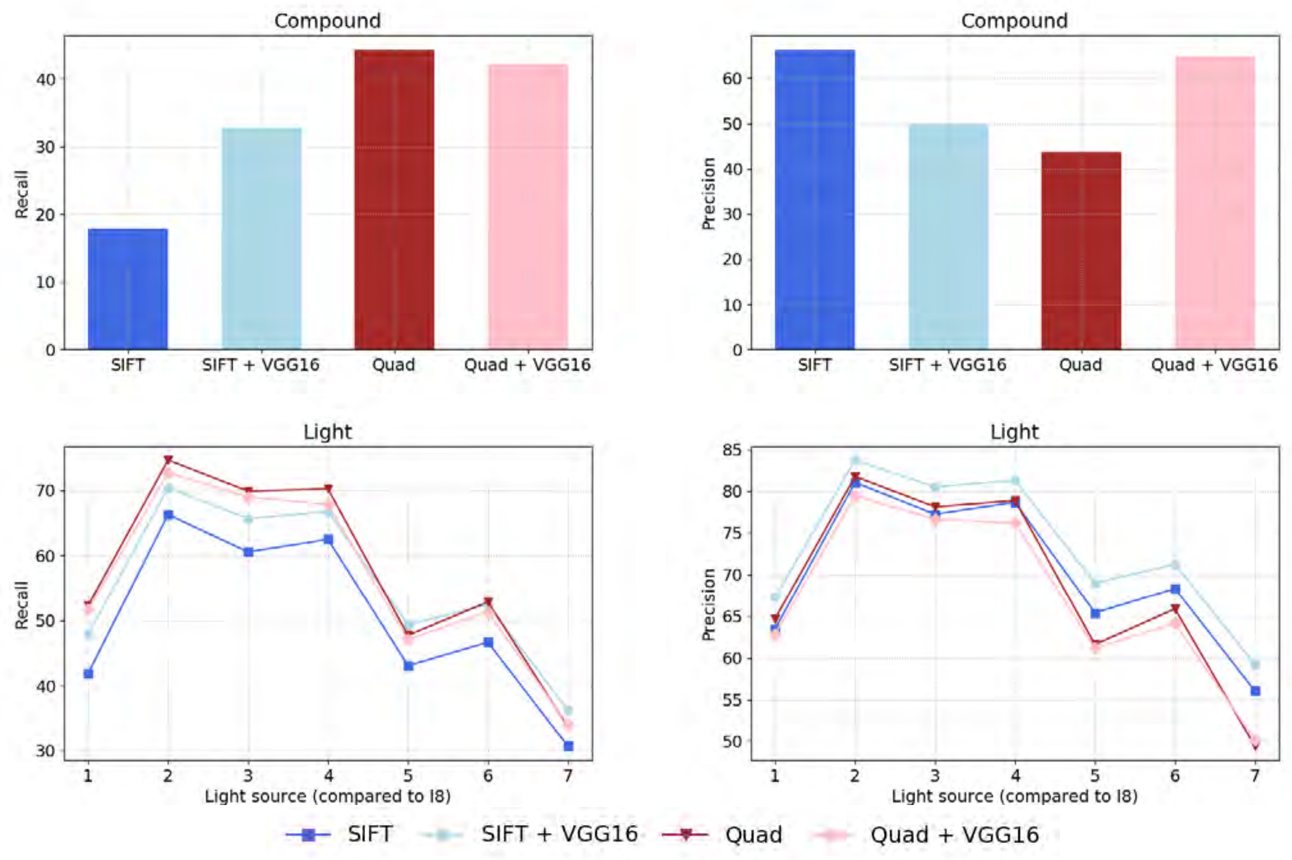


Figure 18: The matching scores, comparing the SIFT and VGG16 descriptors.

each detector. SURF and the quad detector find large keypoints, whereas the keypoints found by LIFT are very small, which would also explain the non-redundancy scores. The non-redundancy seems very sensitive to the keypoint and descriptor size. One could easily increase the non-redundancy by decreasing the size of the keypoints.

The coverage results can be seen in Figure 16b. For these experiments, SIFT performs best, second to LIFT, and AGAST the worst. A possible explanation is that AGAST tends to focus the strongest keypoints in the areas of the image that have the highest contrast, because it ranks points on a cornerness measure. This is also supported by the fact that additional experiments showed that AGAST has a significantly higher coverage when all keypoints are used, as opposed to only the 300 keypoints with the highest response. Other than LIFT and AGAST, the differences between the methods are relatively small and not very significant.

6.3 VGG16

For these experiments, descriptors of the SIFT and Quad detectors were replaced with the VGG16 descriptors, as shown in Table 7. In the case of SIFT, when it comes to rotation the VGG16 descriptors reduce the performance, as both recall and precision go down by about 1%. Given the fact that the VGG16 network is not trained with rotated images, but all images have an upward orientation, this is not very unexpected. In the case of scaling, the VGG16 network improves the recall and precision by about 3–5%, which is a significant improvement. For all other cases, the VGG16 descriptor improves the recall (in some cases by as much as 20%), while the precision goes down significantly. What this means is that there are fewer matches found in total, but a larger portion of those that are found are actually correct.

In the case of the quad network, the VGG16 descriptors do not have as much of a positive effect as with SIFT. In all cases, the precision drops drastically, which means fewer good matches are found, since the

Method	Detector	Orientation Estimator	Descriptor
SIFT [31]	SIFT	SIFT	SIFT
SIFT + VGG16 [31,55]	SIFT	SIFT	VGG16
Quad [50]	Quad	SIFT	SIFT
Quad + VGG16 [50,55]	Quad	SIFT	VGG16

Table 7: The combinations of detector, orientation estimator and descriptor used in the VGG16 matching experiments.

number of correspondences is the same. For rotation, the recall drops by about 2%, which is not a very big difference and again is not very surprising for a network that is not trained with rotated images. For scaling and blurring, the recall may increase or decrease and the difference is not significant. With noise, VGG16 finds fewer correct matches. However, in the case of compound transformations, the VGG16 network performs slightly better by 2–3%. All in all, considering that using a neural network is computationally more expensive, pairing the VGG16 descriptor together with the quad detector may not be justified. However, the method used here to calculate descriptors is very simple and straight-forward. More complex methods could be used [39, 58], which might improve the results. Using the output of an earlier layer might improve the results as well. Perhaps the 7th layer is too deep and the structures are too high-level. Another possibility for improvement is training the networks together, similar to how the separate networks of LIFT were trained together.

6.4 Image Retrieval

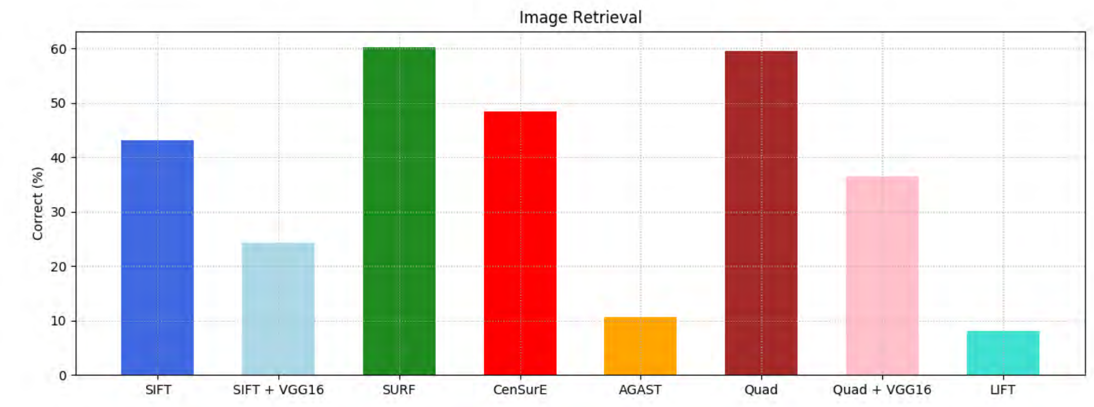
Method	Detector	Orientation Estimator	Descriptor
SIFT [31]	SIFT	SIFT	SIFT
SIFT + VGG16 [31,55]	SIFT	SIFT	VGG16
SURF [3]	SURF	SURF	SURF
CenSurE [2]	CenSurE	SURF	SIFT
AGAST [27,32]	AGAST	BRISK	BRISK
Quad [50]	Quad	SIFT	SIFT
Quad + VGG16 [50,55]	Quad	SIFT	VGG16
LIFT [66]	LIFT	LIFT	LIFT

Table 8: The combinations of detector, orientation estimator and descriptor used in the retrieval experiments.

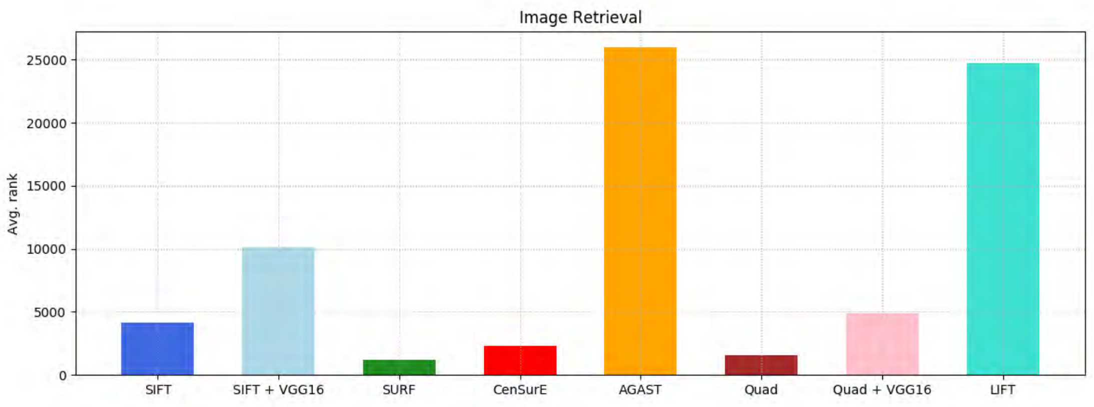
The image retrieval experiments were performed on the methods in Table 8. The results in Figure 19 show how the methods perform in a practical application. The aim of a retrieval system is to retrieve the correct image, and if the correct image is not ranked first, it should be ranked as low as possible. Figure 19a shows the percentage of images that were retrieved correctly and had a rank of 0. Figure 19b shows the average rank of all the 2 500 instances.

The method that has the highest performance when it comes to image retrieval is SURF, as it has the most correct matches and the lowest average rank. However, the difference with Quad is very small and negligible. Both methods predicted 60% of the query images correctly. Compared to SIFT, which matched about 43% correctly, the quad detector did a much better job. This may partially result from the fact that the transformations contain blurring and noise, which SIFT is sensitive to. It is an interesting comparison, though, as the SIFT and Quad pipelines are very similar and only the feature detection is replaced. CenSurE also worked relatively well, with 48% of the predictions correct.

The lowest scores are produced by AGAST, which has the highest average rank, and LIFT, which has the lowest number of correct matches. For AGAST, this may not be very surprising, as it was already clear from the previous experiments that there is a high sensitivity to blurring and noise. LIFT, on the other hand, did have promising results for repeatability and matching experiments without rotations. Since this version



(a)



(b)

Figure 19: The results of the image retrieval experiments. (a) The percentage of cases in which the image retrieved by the system was correct, out of a total of 2 500 instances. (b) The average rank of the retrieval, including the ones that are correct and have a rank of 0.

of the LIFT algorithm did include an orientation estimator network that was trained with rotations, a much better result could have been expected.

Whereas the VGG16 descriptors improved the matching performance of SIFT, they drastically decrease performance when it comes to image retrieval. For the quad detector this may have been expected, as the matching performance was also decreased by the VGG16 descriptors. But for SIFT, it seems that the SIFT descriptor already works very well.

Comparing the results of the matching experiments with compound transformations in Figure 15 with the results of the image retrieval, it can be seen that the performance of the methods is relatively similar. Quad and SURF have the highest performance, then CenSurE, followed by SIFT and finally AGAST. The only big difference is that in the matching experiments, LIFT has a performance similar to that of CenSurE, and this does not at all translate to the case of image retrieval.

7 Discussion

SIFT and SURF perform very well in both the defined metrics and the practical setup. SIFT is particularly strong in rotation and scaling, while it is sensitive to blurring and noise. This may be explained by the fact that it uses Gaussians in different steps of the algorithm. On the other hand, SURF is very robust when it comes to blurring and noise. It is, however, not as good as SIFT in dealing with rotations and scaling. In the image retrieval, SURF had the highest number of correct matches and the lowest average rank and outperformed SIFT significantly. However, in a practical application, images in the data set may not contain as much blurring or noise as was used in these experiments, which would have a big impact on how each detector performs.

AGAST works relatively well when in case of rotation and scaling, but performance drops quickly when transformations like blurring and noise are applied. This likely results from the fact that the AST finds keypoints by direct pixel comparisons and even small disruptions at the pixel level can disturb the process. Perhaps it not so surprising that it does not achieve a high score in image retrieval, as AGAST was mainly designed for tracking. What is remarkable, though, is that AGAST is not faster than SIFT or SURF, as shown in Table 4. Since AGAST does not add any extra computation and it is stated in [32] that it improves the computation time of FAST, it is possible that the OpenCV implementation of AGAST is not as efficient as it could be.

The Quad network replaces the interest point detection of SIFT, while the rest of the detection and description pipeline is the same, where possible. In almost all experiments, the Quad network outperforms SIFT, except perhaps when it comes to rotations. Comparing the Quad network to SURF, the latter generally has a higher repeatability score, while the former does better in matching. In the retrieval experiments, the performance of both is very similar, although SURF does slightly better. Since neural networks are computationally more expensive, SURF is much faster.

Defining a neural network for calculating an orientation or a descriptor is relatively straightforward, as it converts a single image patch to a single multidimensional output. On the other hand, detecting keypoints is a more complex problem, because it is not clear what the number of outputs should be. All of the feature detectors discussed in Section 4 use a different method to find keypoints. However, at one point or another, in each method the input image is converted into a score map, on which non-maximum suppression is then performed to find the most stable keypoints. Most methods calculate the score map directly. The covariant detector does it indirectly, by first determining the location of the best keypoint per patch, and then accumulating votes. It seems necessary to use a score map, at least with CNNs, in order to produce an arbitrary amount of keypoints, given an image of any size.

The results of the neural networks are not as good as might be expected. The same results can be achieved by traditional method that are much faster. However, these results show that neural networks do have a potential to achieve very high performance, as they can be trained to work for specific transformations and data sets. Therefore they may be tuned exactly to the data they need to perform on. The downside of neural networks is that they are difficult to train and can have many parameters that need to be precisely tuned in order to achieve good results.The advantage of neural networks is that they can be trained specifically for a data set, containing any set of transformations. TILDE, for example, was trained to be robust to weather and lighting changes. Manually defining a detector that performs well with these transformations may be challenging, while a neural network mainly requires training on the right data.

References

[1] Aanæs, H., Dahl, A., and Steenstrup Pedersen, K. Interesting interest points. *International Journal of Computer Vision* (2011), 1–18.

[2] Agrawal, M., Konolige, K., and Blas, M. R. *CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 102–115.

[3] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. Speeded-up robust features (surf). *Comput. Vis. Image Underst. 110*, 3 (June 2008), 346–359.

[4] Beaudet, P. R. Rotationally invariant image operators. In *Proceedings of the 4th International Joint Conference on Pattern Recognition* (Kyoto, Japan, Nov. 1978), pp. 579–583.

[5] Chapelle, O., and Wu, M. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval 13*, 3 (Jun 2010), 216–235.

[6] Demuth, H. B., Beale, M. H., De Jess, O., and Hagan, M. T. *Neural Network Design*, 2nd ed. Martin Hagan, USA, 2014.

[7] DeTone, D., Malisiewicz, T., and Rabinovich, A. Deep image homography estimation. *CoRR abs/1606.03798* (2016).

[8] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res. 12* (July 2011), 2121–2159.

[9] Ehsan, S., Kanwal, N., Clark, A. F., and McDonald-Maier, K. D. Measuring the coverage of interest point detectors. In *Image Analysis and Recognition* (Berlin, Heidelberg, 2011), M. Kamel and A. Campilho, Eds., Springer Berlin Heidelberg, pp. 253–261.

[10] Ehsan, S., Kanwal, N., Clark, A. F., and McDonald-Maier, K. D. Improved repeatability measures for evaluating performance of feature detectors. *CoRR abs/1504.07967* (2015).

[11] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 4 (Apr 1980), 193–202.

[12] Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. The amsterdam library of object images. *International Journal of Computer Vision 61*, 1 (Jan 2005), 103–112.

[13] Harris, C., and Stephens, M. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference* (1988), pp. 147–151.

[14] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR abs/1207.0580* (2012).

[15] Hu, W., Xie, N., Li, L., Zeng, X., and Maybank, S. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 41*, 6 (Nov 2011), 797–819.

[16] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR abs/1502.03167* (2015).

[17] Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. Spatial transformer networks. *CoRR abs/1506.02025* (2015).

[18] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision* (Sept 2009), pp. 2146–2153.

[19] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv e-prints* (Dec. 2014).

[20] Koenderink, J. J. The structure of images. *Biological Cybernetics 50*, 5 (Aug 1984), 363–370.

[21] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (USA, 2012), NIPS'12, Curran Associates Inc., pp. 1097–1105.

[22] Kumar, A., Kim, J., Cai, W., Fulham, M., and Feng, D. Content-based medical image retrieval: A survey of applications to multidimensional and multimodality data. *Journal of Digital Imaging 26*, 6 (Dec 2013), 1025–1039.

[23] LeCun, Y. Une procédure d'apprentissage pour réseau à seuil assymétrique. In *Cognitiva 85: A la Frontière de l'Intelligence Artificielle, des Sciences de la Connaissance des Neurosciences* (1985), CESTA.

[24] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput. 1*, 4 (Dec. 1989), 541–551.

[25] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (Nov 1998), 2278–2324.

[26] Lenc, K., and Vedaldi, A. Learning covariant feature detectors. *CoRR abs/1605.01224* (2016).

[27] Leutenegger, S., Chli, M., and Siegwart, R. Y. Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision* (Washington, DC, USA, 2011), ICCV '11, IEEE Computer Society, pp. 2548–2555.

[28] Lew, M., Bakker, E. M., Sebe, N., and Huang, T. S. Human-computer intelligent interaction: A survey. In *Human–Computer Interaction* (Berlin, Heidelberg, 2007), M. Lew, N. Sebe, T. S. Huang, and E. M. Bakker, Eds., Springer Berlin Heidelberg, pp. 1–5.

[29] Lew, M. S., Sebe, N., and Eakins, J. P. Challenges of image and video retrieval. In *Image and Video Retrieval* (Berlin, Heidelberg, 2002), M. S. Lew, N. Sebe, and J. P. Eakins, Eds., Springer Berlin Heidelberg, pp. 1–6.

[30] Lindeberg, T. Feature detection with automatic scale selection. *International Journal of Computer Vision 30*, 2 (Nov 1998), 79–116.

[31] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60*, 2 (Nov. 2004), 91–110.

[32] Mair, E., Hager, G. D., Burschka, D., Suppa, M., and Hirzinger, G. Adaptive and generic corner detection based on the accelerated segment test. In *Proceedings of the 11th European Conference on Computer Vision: Part II* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 183–196.

[33] Matas, J., Chum, O., Urban, M., and Pajdla, T. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing 22*, 10 (2004), 761 – 767. British Machine Vision Computing 2002.

[34] Mikolajczyk, K., and Schmid, C. An affine invariant interest point detector. In *Computer Vision — ECCV 2002* (Berlin, Heidelberg, 2002), A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., Springer Berlin Heidelberg, pp. 128–142.

[35] Mikolajczyk, K., and Schmid, C. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 10 (Oct. 2005), 1615–1630.

[36] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., and Gool, L. V. A comparison of affine region detectors. *Int. J. Comput. Vision 65*, 1-2 (Nov. 2005), 43–72.

[37] Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[38] Moravec, H. P. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford, CA, USA, 1980. AAI8024717.

[39] Ng, J. Y., Yang, F., and Davis, L. S. Exploiting local features from deep networks for image retrieval. *CoRR abs/1504.05133* (2015).

[40] Nowruzi, F. E., Laganiere, R., and Japkowicz, N. Homography estimation from image pairs with hierarchical convolutional networks.

[41] Parker, D. B. Learning logic. In *Technical Report TRâĂŞ47* (1985), Center for Computational Research in Economics and Management Science, MIT.

[42] Quinlan, J. R. Induction of decision trees. *Mach. Learn. 1*, 1 (Mar. 1986), 81–106.

[43] Rey-Otero, I., and Delbracio, M. Is repeatability an unbiased criterion for ranking feature detectors? *SIAM Journal on Imaging Sciences 8*, 4 (2015), 2558–2580.

[44] Rosenblatt, F. The perceptron: A perceiving and recognizing automaton. In *Technical Report 85-460-1* (1957), Cornell Aeronautical Laboratory.

[45] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I* (Berlin, Heidelberg, 2006), ECCV'06, Springer-Verlag, pp. 430–443.

[46] Rosten, E., Porter, R., and Drummond, T. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 1 (Jan 2010), 105–119.

[47] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision* (Washington, DC, USA, 2011), ICCV '11, IEEE Computer Society, pp. 2564–2571.

[48] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature 323* (Oct. 1986), 533–536.

[49] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[50] Savinov, N., Seki, A., Ladicky, L., Sattler, T., and Pollefeys, M. Quad-networks: unsupervised learning to rank for interest point detection. *CoRR abs/1611.07571* (2016).

[51] Schmid, C., Mohr, R., and Bauckhage, C. Evaluation of interest point detectors. *Int. J. Comput. Vision 37*, 2 (June 2000), 151–172.

[52] Sebe, N., and Lew, M. S. Salient points for content-based retrieval. In *In BMVC* (2001), pp. 401–410.

[53] Sebe, N., Lew, M. S., and Huang, T. S. The state-of-the-art in human-computer interaction. In *Computer Vision in Human-Computer Interaction* (Berlin, Heidelberg, 2004), N. Sebe, M. Lew, and T. S. Huang, Eds., Springer Berlin Heidelberg, pp. 1–6.

[54] Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., and Moreno-Noguer, F. Discriminative learning of deep convolutional feature point descriptors. In *2015 IEEE International Conference on Computer Vision (ICCV)* (Dec 2015), pp. 118–126.

[55] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[56] Sivic, and Zisserman. Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision* (Oct 2003), pp. 1470–1477 vol.2.

[57] Thomee, B., Huiskes, M. J., Bakker, E., and Lew, M. S. Large scale image copy detection evaluation. In *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval* (New York, NY, USA, 2008), MIR '08, ACM, pp. 59–66.

[58] Tolias, G., Sicre, R., and Jégou, H. Particular object retrieval with integral max-pooling of CNN activations. *CoRR abs/1511.05879* (2015).

[59] Turaga, P., Chellappa, R., Subrahmanian, V. S., and Udrea, O. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology 18*, 11 (Nov 2008), 1473–1488.

[60] Verdie, Y., Yi, K. M., Fua, P., and Lepetit, V. Tilde: A temporally invariant learned detector. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 5279–5288.

[61] Viola, P., and Jones, M. 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* (2001), vol. 1, pp. I–511–I–518 vol.1.

[62] Wang, S., and Sun, X. Generalization of hinging hyperplanes. *IEEE Transactions on Information Theory 51*, 12 (Dec 2005), 4425–4431.

[63] Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[64] Widrow, B., and Hoff, M. E. Adaptive switching circuits. *1960 IRE WESCON Convention Record* (1960), 96–104. Reprinted in *Neurocomputing* MIT Press, 1988 .

[65] Witkin, A. P. Scale-space filtering. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2* (San Francisco, CA, USA, 1983), IJCAI'83, Morgan Kaufmann Publishers Inc., pp. 1019–1022.

[66] Yi, K. M., Trulls, E., Lepetit, V., and Fua, P. LIFT: Learned Invariant Feature Transform. In *Proceedings of the European Conference on Computer Vision* (2016).

[67] Yi, K. M., Verdie, Y., Fua, P., and Lepetit, V. Learning to assign orientations to feature points. *CoRR abs/1511.04273* (2015).

[68] Zeiler, M. D. Adadelta: An adaptive learning rate method. *CoRR abs/1212.5701* (2012).

[69] Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A. Face recognition: A literature survey. *ACM Comput. Surv. 35*, 4 (Dec. 2003), 399–458.

[70] Zheng, L., Yang, Y., and Tian, Q. SIFT meets CNN: A decade survey of instance retrieval. *CoRR abs/1608.01807* (2016).