Matching Renders with Real Images using Learned Features for Industrial Visual Inspection

Pamir Ghimire

Masters in Computer Vision
University of Burgundy

Supervisors:
Dr. Igor Jovančević
Dr. Ludovic Brèthes
Diotasoft, Toulouse

A Thesis Submitted for the Degree of
MSc Erasmus Mundus in Vision and Robotics (VIBOT)

· 2018 ·

Abstract

Automatic inspection of mechanical assemblies for conformity with CAD using images from RGB cameras is an important problem in quality control industry. In model based inspection, for each inspection image, we have a simplistic CAD (lacking in deformable elements like cables and plastic caps, devoid of texture information, and having a reduced face-count) along with noisy relative camera pose and intrinsics. Further, the element we want to inspect (control) can belong to one of several classes.

We want to create a system that can make judgments on the real inspection images about a diverse set of controlled elements by using the simplified geometric information in CAD. The decisions we are interested in are absence/presence of controlled elements, and misalignment.

We propose to address this problem by comparing the real image with a simplistic render of the CAD which can be produced in real-time. We develop deep-learning based approaches to perform the cross-domain comparison (real vs synthetic), while working with a very small number (~ 100) of real images for which we have model-relative camera poses. We also propose techniques to train our approach when no real images corresponding to a CAD are available (using realistic renders), so that ultimately, inspection might be performed using only CAD.

Our main contribution is a method to train, with few image pairs, a cross-domain rotation invariant feature descriptor which can be used to find local correspondences between simple (non photo-realistic) renders and real images by measuring euclidean distances between the descriptors. Descriptors such as SIFT and SURF can not be used for such cross-domain comparisons because they vary with texture information. Our descriptors can be coupled with off-the-shelf detectors such as FAST (for efficient performance). We also propose a simple shader which can quickly produce simple renders that capture geometric information, for real-time inspection.

We show that the learned descriptors can be used to model an arbitrary geometric shape in a render (as a BoW model) which we can then look for in the real image. Further, homography between the local correspondences can be used to check for misalignment. The local correspondences may be used for other applications such as augmented reality.

To the best of our knowledge, at least in the context of 2D-vision based industrial inspection, this is the first work of its kind that reports local-feature based matching between renders and real images.

Contents

Acknowledgments **viii**

1 Introduction **1**

1.1 Context 2

1.2 Motivation 3

1.3 Design Problem 4

1.4 Approach and Milestones 4

1.5 Thesis Outline 5

2 Review of Related Literature **7**

2.1 Vision based Inspection 7

2.2 Rendering CAD 9

2.3 Deep Learning and Discriminative Feature Learning 11

3 Design Choices and Overview of Proposed Method **15**

3.1 Design Choices 15

3.2 Rationale Behind Design Choices 15

3.2.1 Feature type for comparison with CAD 15

3.2.2 Rendering 3D CAD 16

3.2.3 Deep Learning of Cross-domain Descriptor 17

3.3 Overview of Proposed Method 17

3.3.1 Test-time Workflow . . . 18
3.3.2 Train-time Workflow . . . 19

4 Synthetic Data Generation System **23**
4.1 Rendering Procedure Corresponding to One Real Inspection Image . . . 24
4.1.1 Transformation of Camera Pose from Right-handed to Left-handed System 24
4.2 Automatic Coloring and Texturing of Plain CADs . . . 27
4.3 Simple Shader for Generating 'Expected' Image . . . 28
4.4 Generation of Binary Inspection Mask . . . 30

5 Dataset of Patch Pairs **32**
5.1 Manual Registration . . . 33
5.2 Patch-pairs from a Single Registered Image pair . . . 34

6 Domain Invariant Descriptor Learning **36**
6.1 Training Deep Networks for Comparing Images . . . 36
6.2 Initial Insights Behind Proposed Approach (from Transfer Learning) . . . 37
6.3 Two-Stage Training Approach For Learning Domain-Invariant Descriptor . . . 40
6.3.1 Network Architecture . . . 40
6.3.2 Training Step I : BootStrapping . . . 41
6.3.3 Training Step II : Triplet-Loss Training . . . 42

7 Results and Evaluations **46**
7.1 ROC Curves for Learned Descriptors . . . 46
7.2 Bag of Visual Words using Learned Features . . . 48
7.3 Classification with fine-tuned CNN features . . . 48

8 Conclusion and Future Work **53**

Bibliography **63**

List of Figures

1.1 Two inspection images, one for which inspection result is OK (no defect), and one for which it is NOK (with defect). Inspection decision can be easily inferred by visually comparing the simple render (leftmost column) with the real image. The inspection mask specifies the image region where comparison is desired. . . . 3

2.1 Methods encountered in visual-inspection literature, left-to-right, methods from [39], [1], and [17] 9

2.2 Methods encountered in literature regarding using synthetic images for learning systems that work on real ones, left-to-right, figures from [36] and [67] 10

2.3 Methods encountered in literature regarding learning to compare patches, top-to-bottom, figures from [32] (MatchNet) and [95] 14

3.1 Proposed inspection workflow at test time. In the real inspection image, texture patches are also detected as containing interest points due to use of FAST detector. 18

3.2 Proposed method for creating patch pair dataset for training a descriptor. . . . 19

3.3 Proposed method for training a descriptor using a dataset of patches created as described in Figure 3.2 21

3.4 Top Row : Examples of 128x128 patch pairs generated as matching pairs. Not all supposedly matching pairs actually match due to the CADs being simplistic. Bottom Row: Examples of 'texture' patches generated from real images (refer Figure 3.2). As can be seen, these usually come from areas in the real image with scratches, text, or elements such as wires and ties that are not present in the CADs. The patch-pair dataset created in a semi-supervised manner is not perfect. This has implications on learning of the descriptor. 22

4.1 Synthetic data generation system designed in Unity 3D. 23

4.2 World and camera frames in OpenGL (Right handed), as in Diota's Kernel, and Unity 3D (Left Handed) 25

4.3 (a) UV mapping for a cube, a texture can be defined after the cube has been cut open, figure from [93]. (b) Examples of used texture patches from DAGM Industrial dataset [42] 28

4.4 Proposed simplistic shading for rendering expected view in real-time during inspection. The fragment color is based on the fragment's normal in camera-space and its distance from the camera. $\{^{C}n = ((^{C}T_{W}.^{W}T_{M})^{-1})^{T}.^{M}n\}$ relates fragment normal in model-space and camera-space. 29

4.5 Effect of coefficients attached with shading intensity from fragment normal and fragment depth. Combining intensity from both enables a more accurate perception of edges. 30

4.6 Examples of binary inspection masks. 31

5.1 All renders produced for one real inspection image. 32

5.2 Manual registration of simplistic render with real image. (a) and (b) show manually selected corresponding points between the two images. (c) is the simplistic render, and (d) is the corresponding real image. (e) and (f) show overlapped visualization of the two images before and after registration respectively. Effects of registration are clearly visible within the pink circle. 34

6.1 Principal Component Analysis [92] on off-the-shelf features extracted from synthetic and real positive (containing screws) and negative (missing screws) patches. a, b : PCA on features extracted from ensemble of real and synthetic features, c: pair-wise distance between feature vectors, d: PCA on features extracted only from synthetic patches, e: examples of real and synthetic positive and negative patches 38

6.2 Principal Component Analysis [92] on features extracted from synthetic and real positive and negative patches after the network was fine tuned on the mixed (synthetic-real) dataset. 39

6.3 Patch-pairs organized for bootstrapping. All patches containing 'geometric features', whether real or synthetic, are put in one class, and all 'texture' patches are put in another. Texture patches from real images (Class 0) were converted to grayscale, although they have been shown in color here.) 43

6.4 (a) Triplet loss minimization brings anchor and positive closer while increasing separation between anchor and negative. [76] (b) A hard triplet ($d_{an} < d_{ap}$) can be made harder still by swapping the anchor and the positive (in-triplet hard negative mining [6]) 45

7.1 False negative (top row) and False Positive (bottom row) pairs of size 224xx224 obtained using the learned descriptor trained with margin size 5.0 and distance threshold between descriptors = 16.56 (refer Fig. 7.2) 47

7.2 ROC curves for learned descriptor based on patches of size 128x128 and 224x224. Descriptor based on bigger patches clearly performs better. Values at which distances between patch descriptors were thresholded to get a true positive rate of 95% is denoted as 'th. Dist'. The deep architectures for the descriptor network are those mentioned in Table 6.1. 50

7.3 Examples of Nearest neighbour matching between renders (left) and real images (right). The interest points were detected using the ORB implementation of FAST detector in OpenCV [75]. Matches were found using Euclidean distance between the descriptors. 51
7.4 Comparison of the learned features (LF) with ORB features (rotated BRIEF) [75] using Bag of Visual Words [90]. The plot in (e) shows distances between BoW histograms of matching image pairs only computed using ORB based and learned features based dictionary. 52

List of Tables

2.1 Different loss terms encountered in literature regarding descriptor learning. . . . 13

3.1 Design choices and decisions. 16

4.1 RGB color values for different metals. 27

6.1 Architectures of different deep networks used. 41

7.1 FPR95 rates observed for learned descriptors trained with patches of different sizes and different margin parameters α. The threshold distances related to the FPR95 rates are different, refer Figure 7.2 47

7.2 Classification accuracies of SVM's trained on features from fine-tuned CNN. . . . 49

Acknowledgments

Thanks to Diota for funding this thesis work, and to its team in Toulouse (DiotaControl) for hosting me during the span of the work. Special thanks to Igor for valuable discussions throughout the thesis work, Yannick for helping me transition smoothly to Toulouse, and Benoit for guidance regarding transformations between frames and insightful critique and suggestions. Thanks to Nour Islam for being a sounding board during times of confusion. Thanks to Lilian Calvet of EnCOV for his valuable comments on this work during my visit to his group. Thanks also to Professor Jean-José Orteu of Ecole des Mines d'Albi for his detailed comments on this thesis.

Thanks to my parents for their unconditional love and support, without whom none of my achievements would have been possible.

Finally, thanks to the MSCV and VIBOT team at Le Creusot for guiding me through the many procedures involved in carrying out studies in France, and providing me with the basis in mathematics, and image and signal processing that enabled me to carry out this work, and initiated me in the world of computer vision and robotics.

Chapter 1

Introduction

The complexity of man-made tools in terms of number of unique parts and assembly sequence has steadily increased over time. From few unique parts in tools like hammer stones and hand axes in stone age, chariots in bronze age and wooden lathes in iron age [3], complexity of man-made tools exploded in industrial age to thousands of unique parts. Steam engines, ocean going ships and long-distance trains were intricate and were assembled by thousands of people. Further, the advent of mass manufacturing started rapid creation and delivery of complex assemblies to common households. Finally, the current information age has enabled creation of incredibly complex machines; an Airbus A380 has millions of unique parts, which are made by hundreds of companies, distributed over tens of countries [80].

As the complexity of our machines and the pace of their creation increases, so does the room for error in their creation and challenge to ensure their safety. These errors can be very costly. Many reports are encountered about big car manufacturers recalling tens of thousands of shipped vehicles due to manufacturing issues, and air-crafts facing accidents due to ill-maintained sub-systems [91] [54].

Phases of a complex assembly that are most prone to errors are those where humans are involved. This is true for the phase of conformity checking as well, which is checking an assembly for compliance with a specification following an assembly step. To minimize errors, automatic inspection is desirable mainly due to following reasons: [70]

1. Human inspectors are inconsistent, partly because of tiredness.
2. Manual inspection is time consuming, which adds to production costs.
3. Automatic inspection is feasible in places that are dangerous for humans.
4. Constant increase in production pace demands proportional decrease in inspection time.

1.1 Context

Automatic inspection of manufactured goods is an industry with many technologies and applications. Popular technologies have included Coordinate Measurement Machines (CMMs) [96], Ultrasound Probes [5] , Laser Radars [57], 3D Time-Of-Flight sensors [16], and 2D cameras [59] [39] [82] . Contexts of applications have popularly been assembly lines for vehicles [82], boats, aircrafts [87], sea-floor infrastructure [57], electrical equipments [5] and heavy machinery. Inspection tasks can be dimensional measurements [96], photometric inspections (for cracks and scratches) [13] , inspection for presence/absence of parts [39] [18], etc.

CMMs are precise, but they are slow, expensive and limited to inspections that can be performed by sparsely sampling surface points [96]. Laser Radar (LIMAR) systems are affected by reflectivity of surfaces [43]. Ultrasound probes are best suited for non-destructive measurements of the insides of structures [5], and are not useful for photometric inspections. Finally, Time-Of-Flight and structured-light 3D sensors consume a lot of power, require heavy computation for data processing, and are prone to scattering errors when working with high aspect-ratio scenes [58] [28].

2D or 3D vision systems are both attractive for automatic visual inspection since they provide rich information about the inspected part without contact with it, from a distance, and with low latency. For inspection tasks that do not require precise dimensional measurements, and rather pose simple queries like absence/presence or bad-orientation of visible parts or inspection of surface defects (like cracks and scratches), passive vision sensors (2D cameras) are ideal. However, while the 3D information provided by previously mentioned modalities can be directly used for making comparisons with specifications in a 3D CAD, use of CAD is not simple when working with 2D images. This is specially true when the CAD lacks texture information.

If the viewpoint of a 2D camera can be tracked relative to a part manufactured according to a CAD specification, the CAD model can be projected onto the image plane of the camera with the same perspective for different applications [9]. CAD model based tracking is used by Diota, a French company, to provide augmented reality solutions on the factory floor. Based on its core technology of CAD based tracking, Diota has stepped into the industry of vision-based inspection. The tracked viewpoint information is used for rendering a flat projection of the element to be inspected which provides a binary 'inspection' mask for the 2D image. Within the mask area, image processing algorithms are applied on inspection image for making inspection decisions. Algorithms are designed with apriori knowledge about expected appearance of the part. The use of information in the CAD is thus limited.

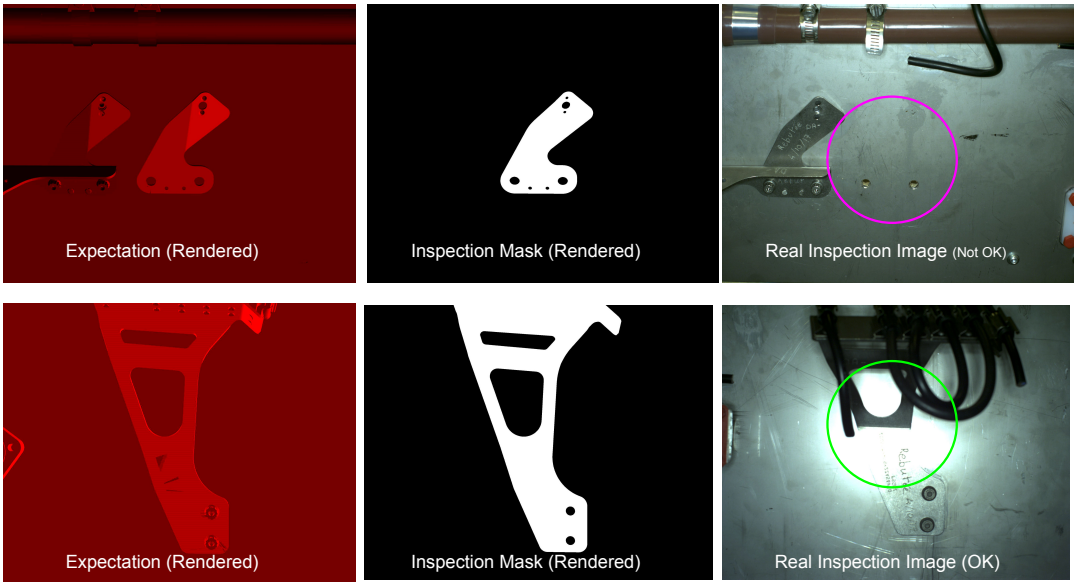


Figure 1.1: Two inspection images, one for which inspection result is OK (no defect), and one for which it is NOK (with defect). Inspection decision can be easily inferred by visually comparing the simple render (leftmost column) with the real image. The inspection mask specifies the image region where comparison is desired.

1.2 Motivation

When we humans look at a simple render of a CAD (containing only basic lighting, and lacking any texture), we are quickly able to relate it with a real image with a similar viewpoint and tell if the real image is in accordance with the render. When we do this, we make connections between simplistic forms we see in a render with complex and noisy manifestations of these forms in a real image. We do a similar task when inspecting a building or a work-piece against a design sketch as well.

Further, when we inspect a new kind of thing, we do not need to have seen a 'perfect' real example of it before to tell if it is faulty; specifications are enough.

What if we could create a system that could also make connections between a simple representation of a specification (concretely, a simple render of a CAD) and a real manifestation of the same. Such a system would, first, alleviate the need for detailed (texture-mapped, high poly, realistically illuminated) renders when only geometry based inspections need to be performed, and second, be able to work with real instances of never-before seen mechanical pieces

by relating primitives in the simple render with those in the real image.

Currently, when an algorithm needs to be designed for inspecting a new class of element, several real image examples are first collected corresponding to its good and bad inspection states which are then used in a template matching setup or a machine learning setup. This process is time consuming, expensive, and not always feasible; real examples of an 'ideal' assembly may not always be available for collecting training data.

In contrast, if a system exists that 'understands' how geometric primitives in a render translate to manifestations in a real image, full element images would not be necessary to tune the system. Further, if the system is able to make local correspondences, between the render and the real image, in addition to inspection, it can also be used for registering the render with the real image and performing precisely registered augmented reality.

1.3 Design Problem

Develop a framework that can perform visual inspection on real images of assemblies given a simple render of the assembly CAD from the same perspective. A small set of images (~ 100) with associated viewpoint information are available which can be used for developing the system.

Once the system is developed, it should be able to operate on images of parts/assemblies that were not seen during the training. Because the context of development is the inspection business of Diota, we want our approach to be able to satisfy following constraints:

- Rendering and comparison of real image with render should be real-time
- Resources available on a standard laptop computer should suffice
- Reuse of as much information as possible that is already available from Diota's inspection kernel (camera extrinsics and intrinsics) is desirable
- Intermediate results of the framework should be accessible and useful
- It should be possible to fine tune the system for a new inspection dataset, if it is available

1.4 Approach and Milestones

To compare a render with a real image, we can engineer features to be extracted from images and develop a comparison scheme that can correctly relate parts of a render with parts of a real image. Alternatively, we can we can learn domain-invariant features automatically to make the correspondences. We choose the latter, specially in light of recent successes Convolutional

Neural Network (CNN) based methods have had for pattern recognition tasks on images [73] [36] [67]. CNN features have proven to be widely useful.

To perform a comparison between two images for inspecting an element, we can either compare whole crops, real and synthetic, i.e., do a crop-level comparison, or we can compare patches from crops (patch-level comparison) to match primitives (local regions with interesting features, like corners). We choose the latter approach, because first, we do not have enough real images with associated camera poses for which we can produce renders to train a system at the crop level, and second, in the images we have, not all possible kinds of defects are captured, and it is hard to simulate them in 3D. However, when we work at the patch level, even from a single real-synthetic image pair, it is possible to generate hundreds of patch pairs. Additionally, a lot of literature exists on training CNNs for comparing patches and learning a feature descriptor to do so [95] [32] [94].

To execute our approach that intends to utilize deep learning for performing patch-level comparison between real and synthetic images, we identify the following milestones:

- Creation of a rendering pipeline that can load CAD models corresponding to real inspection images and use the associated camera parameters to produce a set of simple and realistic renders
- Creation of a dataset of patch pairs coming from synthetic and real images, and containing meaningful geometric information (like corners, segments, etc.)
- Training a CNN to extract features from patches such that similar patches are mapped nearby and dissimilar ones far away (in Euclidean sense)
- Use of the learned feature descriptor to perform patch-level comparison between renders and real images, and aggregation of patch level decisions to make a crop-level inspection decision

1.5 Thesis Outline

This thesis is organized around the suggested approach to inspection problems mentioned above as follows:

- Chapter 2 reviews the related literature in areas of simple and realistic rendering, inspection approaches using 2D images, discriminative feature learning, and deep learning when using sythetic and real data

- Chapter 3 describes the synthetic images generation system, and the simplistic shader suggested for generating expected inspection image that captures geometric details in real time
- Chapter 4 describes the process by which a dataset of patch pairs was created using the synthetic and real image pairs with same camera poses
- Chapter 5 describes the method by which a cross-domain rotation-invariant feature extractor was learned. This feature extractor maps patches coming from different domains to nearby locations if they are geometrically similar
- Chapter 6 benchmarks the learned descriptor and showcases its application to model previously unseen parts in a render as a Bag of Words (BoW) model that is searched for in the real image
- Chapter 7 finally discusses the merits and short-comings of the proposed approach and the learned descriptor, and proposes directions for future work

Chapter 2

Review of Related Literature

The main theme in this thesis is comparison of real 2D images with 3D models through renders and learned features. Hence, two areas of literature feature prominently:

- 1. Rendering of 3D models
- 2. Discriminative feature learning

Rendering was studied for producing synthetic data for the feature learning stage, and developing a simple shader program that could render an expected view capturing geometric details at run-time. Feature learning techniques, mainly those employing deep learning, were studied for matching images coming from 2 domains, real and synthetic, while working with relatively few real images ($\sim$100).

The final purpose of this work is to use learned domain-invariant features for inspecting certain elements (henceforth auto-control elements) in an industrial assembly.

Below, Section 2.1 reviews methods for automatic vision-based inspection, Section 2.2 discusses rendering of CADs in inspection and related areas, and Section 2.3 discusses deep learning when using synthetic data, and discriminative feature learning. The following chapter introspects our application requirements and presents our choices and proposed methods in light of the reviewed literature.

2.1 Vision based Inspection

Agin [1] in 1980 presented one of the earlier works on computer vision for industrial inspection. He described identification of industrial parts by nearest-neighbour classification of features derived from connected components in an image, and even remarked about desirability of

generic local features over part-specific ones. Chin et. al. [15] in their 1986 review of robot vision for bin-picking problem identified feature extraction, modeling and matching as issues central to the problem. Bin-picking ($\sim$ object detection) is closely related to inspection, the difference being in inspection, we know where the inspected object should be. They classified features into 3 dimensional categories: 2D, $2\frac{1}{2}$D and 3D, and 2 scope-based categories: local and global features. $2\frac{1}{2}$D features were defined as object features derived from a 2D image that are viewpoint dependent, and were remarked to be more robust than 2D features due to preserving information such as surface reflectance and orientation unlike 2D features. Matching approaches were proposed for different model representations: statistical pattern-recognition for those using 2D global features, syntactic matching [84] for those using local features, and graph-matching for those using combination of local and relational features. Further, it was recommended to compare sets of surface patches for $2\frac{1}{2}$D matching.

In more recent literature on industrial inspection, graph matching methods are encountered whereby a 3D CAD is available and inspection is carried out using 2D cameras [24] [39] [87] ; CAD model renders are represented by relational graphs of local 2D features such as segments, corners, ellipses,etc., image crop of a query part is also transformed into a similar graph, and comparison is performed by bipartite matching. However, in most CAD based inspection literature, use of 2D cameras is not imposed as a constraint. Hence, 3D sensors are exploited and matching is performed between 3D descriptors (such as Point Feature Histogram (PFH), 3D Shape Context (3DSC), Rotation Invariant Feature Transform (RIFT), etc. [65]) extracted from range data and CAD [69] [60] [70] [7], as also remarked in survey of Newman et. al. [59].

In inspection of 3D structures using 2D images, it is popular to use information in edges and silhouettes, i.e., 2D features [38] [50] [87]. Feature descriptors computed from patches centered around interest points (view-point dependent features) are relatively rare in inspection related literature; Cusano et. al. [17] inspect interior of an aircraft for absence/presence of certain parts by using a library of templates consisting of real image crops of auto-controlled parts and matching SURF keypoints between the templates and query images. However, in the closely related area of object detection, keypoint matching is a de facto [41] [22]. When the inspection task can be performed purely optically, deep learning based methods are plentiful [89] [55]. However, when decision from image processing has to be conditioned on information in a 3D CAD model, inspection literature lacks deep learning approaches.

As is evident, a promising direction of research in industrial inspection can be use of $2\frac{1}{2}$D features computed from patches centered at interest points to use information in viewpoint, with use of deep learning when decisions on 2D images have to be conditioned on 3D CAD specifications.

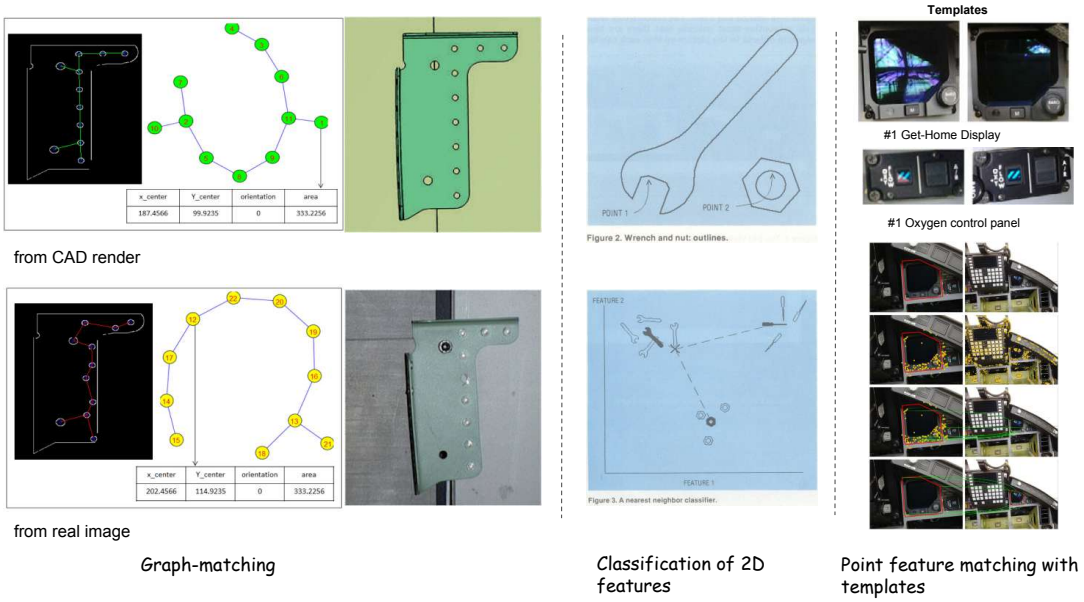


Figure 2.1: Methods encountered in visual-inspection literature, left-to-right, methods from [39], [1], and [17]

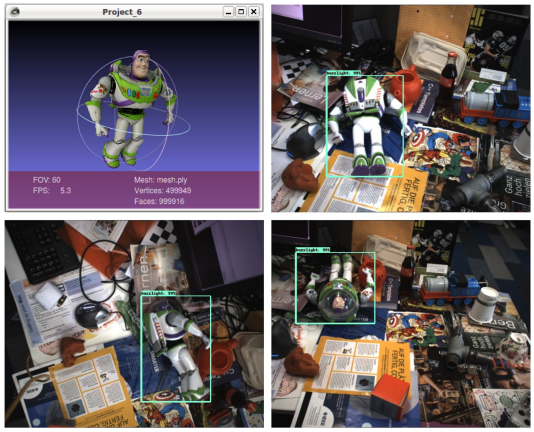
2.2 Rendering CAD

Recent literature on many tasks in computer vision makes use of synthetic data for training deep learning algorithms. This is because it is possible to render physical scenes with varying conditions to generate virtually unlimited number of images, and for each image, it is possible to produce pixel-wise semantic labeling.

Ros et. al. [73] use synthetic images of urban scenes to train deep learning algorithms for semantic segmentation of real scenes. [67] [36] [34] use synthetic images for training object detectors that work on real data. Similarly, [85] use synthetic humans for learning human pose estimation, [19] use procedural synthetic videos for training action recognition system, [44] use deformable face models to generate synthetic faces for training a face recognition system, [33] use synthetic scenes with pedestrians to train a pedestrian recognition system, [37] train a text detection system using synthetic dataset with text in the wild, and so on, and all of these systems finally work on real images.

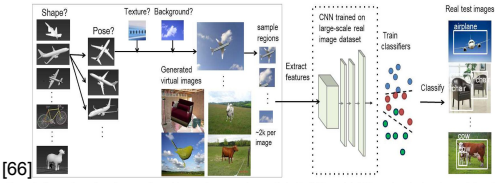
Use of 3D game engines is a popular method for producing synthetic images. [73] [19] use Unity 3D, [85] [34] use Blender, and [36] use Mesh Lab, which is not a game engine. In game development community, Unreal Engine [40] is also a popular choice. Regarding licenses, Unity

3D is not open-source and is a commercial software, however, it provides a free license for non-professional or academic use. Blender, POVRay and Opposite Renderer [66] are some freely available open-source rendering systems. Opposite Renderer [66] was used in SceneNet [56] for producing a large (5M) synthetic image dataset of photorealisitc indoor scenes. Shading languages also differ between rendering technologies; Open GL and Mesh Lab support GLSL, Unity 3D supports HLSL (High Level Shading Language) as well as GLSL, Blender OSL (Open Shading Language), and Unreal Engine HLSL.



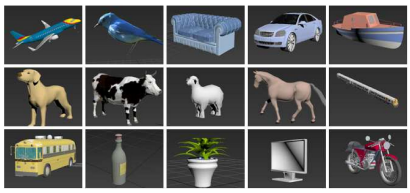
[36]

Figure 1. We show that feature extractor layers from modern object detectors pre-trained on real images can be used on synthetic images to learn to detect objects in real images. The top-left image shows the CAD model we used to learn to detect the object in the three other images.



[66]

Figure 2. Can we learn deep detectors for real images from non-photorealistic 3D CAD models? We explore the invariance of deep features to missing low-level cues such as shape, pose, texture and context, and propose an improved method for learning from synthetic CAD data that simulates these cues.



[66]

Figure 1. We propose to train few-shot object detectors for real images by augmenting the training data with synthetic images generated from freely available non-photorealistic 3D CAD models of objects collected from 3dwarehouse.sketchup.com.

Figure 2.2: Methods encountered in literature regarding using synthetic images for learning systems that work on real ones, left-to-right, figures from [36] and [67]

3D object models used in most modern literature are texture mapped [73] [36] [33]. Google's 3D warehouse [29] provides a good collection of such models. Interestingly, [67] inspect invariance of deep CNN features to low level cues like color and texture when training an object detector by using different object textures for an object class when producing training images. [81] show that HOG based object detector models can be effectively trained in the synthetic domain with uniform object and background textures. The Visual Domain Adaptation Challenge (ICCV, 2017) [68] provided as datasets 2D renders of different textureless 3D models belonging to 12 classes. Participants [25] trained classification and segmentation algorithms to work on real images using these renders that had no texture information. Learning domain invariant

features is a key for doing well in such cross-domain scenarios [26].

Rendering CAD with details is relatively rare in inspection related literature, partly because producing a realistic render can be too slow for real-time inspection. [87] compare graphs of features like circles and ellipses extracted independently from a render and a real image with the same viewpoint by graph matching. Jovancevic et. al. [39] compare 3D descriptions of similar features projected onto the 2D image plane with such features extracted from the real image using image processing techniques also through graph matching. They work under the constraint of simplistic CADs different from most object detection literature mentioned before that freely use texture mapped renders.

Type of shading used in [87] can be interesting for producing simple renders for generating an expected image in real-time that captures geometric details. Although they do not mention the shading model used, visually, edges within the element are visible, although faces of the element that are at different depths do not appear different. In [35], silhouette rendering schemes are proposed; edges can mark discontinuities in face depth, face normals or element identity. A shading that can assign intensity to rendered faces based on face depth, face normal and element identity but does not have to consider pose relative to a light source, like in the Phong or Gouraud models [10], can reveal inspected geometry while requiring little computation.

2.3 Deep Learning and Discriminative Feature Learning

Deep Neural Networks, particularly Convolutional Neural Networks (CNNs) [46], have in recent past revolutionized image understanding tasks such as image classification [45] [79], object detection [72] [49] and semantic segmentation [51]. Successes have been seen mainly under supervised learning paradigm and were mainly attributable to availability of large amounts of annotated data, some notable datasets being Imagenet [20], PASCAL [23], Ms COCO [48], Open Images [30] ([71] provides a rich list of datasets for different vision tasks).

For many image understanding tasks, off-the-shelf features extracted from last layer of a pre-trained CNN can be highly effective [71]. Such features can be used to train a simple classifier, like SVM with a linear kernel, instead of a neural network with few images per class. This practice is also called transfer learning [42] [63]. Instead of an SVM, if a fully connected neural network (FCN) is trained on features from a pre-trained CNN, the practice is called fine tuning. Fine tuning may include learning weights only for an FCN by transferring weights up to convolutional layers [63], or transferred weights may also be tuned further [42].

Transfer learning without fine tuning (off-the-shelf features) works best when test images are similar (in terms of image sensor and image content) to the images on which the network was pre-trained. In such cases, similar images are mapped to nearby locations in the feature space

by the pre-trained network without any fine tuning [53] (Fig. 3), permitting use of a simple classifier. However, when images come from different domains, like synthetic and real, they are mapped far away regardless of image content. This problem is known as domain adaptation or data-shift problem. The origin of this problem is the fact that real and virtual cameras are different sensors, and it is faced even when working with real images that come from different datasets. A simple way to deal with it is to train jointly on a mixed dataset [73] [86].

Transfer learning relies on availability of weights of networks trained on suitable datasets. Popular network architectures for transfer learning have included AlexNet [77], VGG-16/19 [42] [63], OverFeat [71] [53] and GoogLeNet [2] [51]. To be able to terminate a network architecture at the desired network depth/layer and add new layers for fine tuning, it is important that the network architecture is fully detailed in literature and open-sourced. VGG 16/19 [79] and AlexNet [45] are for these reasons more popular architecture choices than others. VGG networks, for example, use 3x3 convolutional filters in all layers, and pooling layers that are always of size 2x2.

While much of the literature that uses CNNs is focussed on image scale analysis, patch level matching or finding local correspondences receives less attention despite being a core problem in computer vision for tasks like Structure From Motion (SFM), Simultaneous Localization and Mapping and Augmented reality and virtuality. For these tasks, engineered image descriptors like SIFT [41] and SURF [47] have long been features of choice. Recently, learned feature detectors and descriptors have outperformed these benchmarks, most notably, LIFT [94]. Learning descriptors for patch-matching was made possible by datasets of natural patches, like the Photo Tour Dataset [11] which enabled [95] [32] [78] and [6]. This dataset contains millions of patches that come from 3 different subsets : Yosemite, Notre Dame and Liberty.

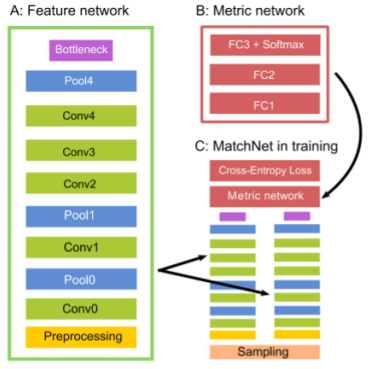
In patch comparison literature, it is common to learn the descriptor and the similarity metric jointly, like in [32] [95], both of which use a 2 layer fully connected neural network as the comparison metric. [32] minimize the cross-entropy error of the fused descriptor-discriminator network while [95] minimize hinge loss. To train descriptors that can be matched using Euclidean distance like SIFT and SURF, [31] minimize contrastive loss and [6] minimize triplet loss. Minimization of euclidean distance based losses is encountered very frequently in modern face recognition literature like [64] and [76] that try to learn discriminative embeddings. In these two works, input images were of sizes 224x224 and 220x220, and networks were 16 and 10 layers deep respectively. Such deep networks are not seen in descriptor learning literature where input patch sizes are of 64x64 (Photo Tour Dataset [11]) and network depths are 3 and 4 in [95], 6 in [32], and only 3 in [78] and [6].

In the inspection literature, to the best of the author's knowledge, there is no precedence of local feature matching or patchwise comparison between renders and real images, neither any

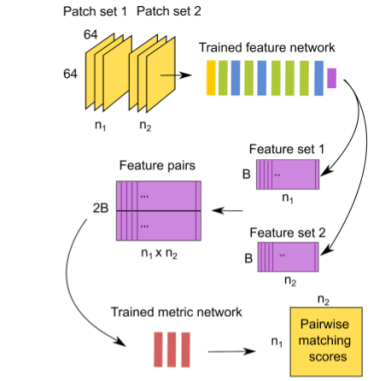
Loss	**Type**	**Expression**	**Details**
Hinge Loss [95]	Classification / Discriminative	$\sum_i max(0, 1 - y_i o_i^{net})$	o_i^{net} is network output for i^{th} training pair, $y_i \in \{-1, 1\}$ the corresponding label, -1 $\Rightarrow$ non-matching
Softmax cross-entropy loss [32]	Classification / Discriminative	$E = \frac{1}{n}\sum_i [y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})]$	$y_i \in \{0, 1\}$ is label for i^{th} training pair, and $\hat{y_i}$ the predicted softmax probability
Contrastive Loss [31]	Metric	$L_i = \frac{1}{2}\{(1 - y_i) D_W^2 + y_i \{max(0, m - D_W)\}^2\}$	$y_i \in \{0, 1\}$ is the label for i^{th} training pair, 1$\Rightarrow$ non-matching, D_W is the Euclidean distance between deep-features of image pair
Ratio Triplet Loss [6]	Metric	$L(d_p, d_n) = \left(\frac{e^{d_p}}{e^{d_p}+e^{d_n}}\right)^2 + \left(1 - \frac{e^{d_n}}{e^{d_p}+e^{d_n}}\right)^2$	d_p is the Euclidean distance between anchor and positive images and d_n that between anchor and negative images in a triplet
Margin ranking Triplet Loss [6]	Metric	$L(d_p, d_n) = max(0, d_p - d_n + \mu)$	d_p and d_n are same as above, μ is the margin
Quadruplet Loss [14]	Metric	$L_{quad} = [d_{ij}^2 - d_{ik}^2 + \alpha_1] + [d_{ij}^2 - d_{lk}^2 + \alpha_2]$	d_{xy} is a learned distance metric between images x and y, i, j, k, and l are anchor, positive, negative and probe images respectively, α_1 and α_2 are two different margins

Table 2.1: Different loss terms encountered in literature regarding descriptor learning.

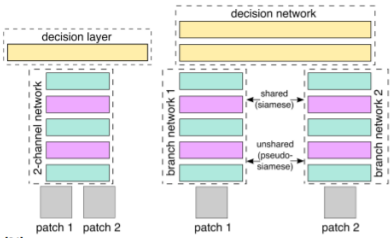
precedence of learned local descriptors. However, patch-level comparison with renders exist elsewhere; [4] establish part-wise correspondences between 2D images of chairs and their 3D models, [47] match features between renders and real images of objects belonging to 2 classes: cars and motorbikes; each render is associated with a pose information and is encoded as a set of SURF features, same features are extracted from a query real image, features from the real image vote for the object class as well as the object pose. In both these works, [4] [47], for each object class, numerous texture-mapped 3D models are available, and the objective is to learn class-wise models to locate them in images. In contrast, in inspection, we know what object needs to be where, as specified in a CAD, and we just want to check if it is actually there as specified.



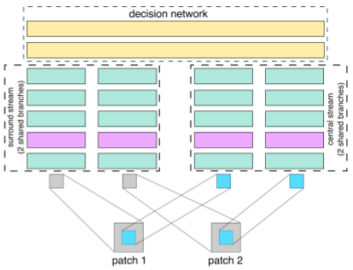
[32]
Figure 1. The MatchNet architecture. A: The feature network used for feature encoding, with an optional bottleneck layer to reduce feature dimension. B: The metric network used for feature comparison. C: In training, the feature net is applied as two "towers"



[32]
Figure 3. MatchNet is disassembled during prediction. The feature network and the metric network run in a pipeline.



[94]
Figure 2. Three basic network architectures: 2-channel on the left, siamese and pseudo-siamese on the right (the difference between siamese and pseudo-siamese is that the latter does not have shared branches). Color code used: cyan = Conv+ReLU, purple = max pooling, yellow = fully connected layer (ReLU exists between fully connected layers as well).



[94]
Figure 3. A central-surround two-stream network that uses a siamese-type architecture to process each stream. This results in 4 branches in total that are given as input to the top decision layer (the two branches in each stream are shared in this case).

Figure 2.3: Methods encountered in literature regarding learning to compare patches, top-to-bottom, figures from [32] (MatchNet) and [95]

Chapter 3

Design Choices and Overview of Proposed Method

3.1 Design Choices

In review of related work in Chapter 2, we identified 3 main components regarding which design choices have to be made to develop an approach for comparing a 2D inspection image with a 3D CAD through renders. They were :

- 1. Feature type for comparison (2D vs $2\frac{1}{2}$D)
- 2. Rendering technology
- 3. Deep Learning of cross-domain descriptor

We summarize our design choices in Table 3.1 and present the rationale behind these choices below.

3.2 Rationale Behind Design Choices

3.2.1 Feature type for comparison with CAD

In section 2.1, we saw that $2\frac{1}{2}$D features computed from patches are more informative than 2D features like contours and edges because they capture information in viewpoint (pose + intrinsics) and local geometric information. Such features are attractive to us at Diota since we have viewpoint of the inspection camera relative to the inspected object for all our inspection

Components	**Choices**	**Decisions**
Inspection Approach	2D features, $2\frac{1}{2}$D features (local patch-based)	$2\frac{1}{2}$D features (based on patches centered at interest points)
Rendering Technology	Unity 3D, Blender, OpenGL, Opposite Renderer	Unity 3D
Deep Learning of Descriptor	Dataset of Natural patches vs Custom dataset, Shallow vs Deep Network, Transfer Learning vs Training from scratch, Deep-network metric vs l_p metric	Custom dataset, Deep network, Transfer Learning, l_2 metric

Table 3.1: Design choices and decisions.

images which allows us to render the 3D CAD from the same viewpoint as the inspection images.

Ideally, Diota wants a system based on modern (deep-learning) approaches that can work on real images of previously unseen parts without having to take numerous images of these parts for a 'learning' stage. We want a CAD render to be all that is needed to perform inspection on a new assembly. Viewpoint dependent features that can relate local geometric information in a render and a real image can be a powerful tool for these goals. Once local correspondences can be established, global decisions (at image level) can be made by aggregating information from local matches.

Finally, when working at patch level, it is possible to generate numerous patch pairs (real, synthetic) from relatively few pairs of real-synthetic images. This is a key constraint for us since at the time of this work, only a small number of images with CAD-model relative viewpoint were available.

3.2.2 Rendering 3D CAD

In section 2.2, we identified 3 major rendering technologies: Unity 3D, Blender and Opposite Renderer. Although Blender and Opposite Renderer are free and open source, the Unity 3D environment is simplest to work with. It supports programming shaders in GLSL, the shading language of OpenGL, and has numerous free plugins in the Unity Asset Store for handling 3D models. For non-professional use, a free license of Unity is available. Additionally, Unity 3D is used at Diota for augmented reality applications. For these reasons, Unity 3D was chosen for rendering our 3D CAD models.

3.2.3 Deep Learning of Cross-domain Descriptor

Desired descriptor characteristics: Because of our choice to compare renders with real images using local features, we need a descriptor that is domain invariant, invariant to texture in real images since we do not have texture mapped CAD models, and rotation invariant since inspected parts can be rotated differently from CAD.

Challenges in learning desired descriptor: Descriptors like SIFT and SURF are sensitive to object texture, and therefore are not useful for our application. Learning a descriptor is reasonable for us, however, using a dataset of natural patches (like [11]) is not an option since our matching scenario is unique, i.e., matching renders with real images. This also rules out using networks pre-trained on fully real datasets. Most literature on discriminative feature learning, like [32] [95] [6] [78], use small patches (of size 64x64) since they want a descriptor for wide baseline matching. However, larger patches are more favorable for us since the baseline between our renders and real images is ideally zero, and the context in larger patches is helpful for matching because our CAD models are simplistic. For large patches, it is not enough to use small networks like those in the literature mentioned before.

Decisions regarding learning a descriptor: For the aforementioned reasons, we created **our own dataset** of patch pairs from a small set of real and corresponding synthetic images. Use of **large patch sizes** makes us use **deep networks** for feature extraction. Since our patches are comparable in size to the face-images used in [64], we use a descriptor network based on **VGG 16**. Finally, unlike the Photo Tour dataset [11] that contains millions of patches, ours contains tens of thousands. So, training a deep network from scratch is not feasible for us. We hence apply **transfer learning** using weights of VGG 16 trained on ImageNet [20], which are available online [52].

Finally, we want to be able to use the learned descriptor at test-time using matchers available in OpenCV (brute force matcher). Because we don't want to evaluate a deep-network based metric for all descriptor-pairs, we choose to learn a descriptor which can work with l_2 **distances (Euclidean) for matching**.

3.3 Overview of Proposed Method

We propose to learn a descriptor such that it can be used for finding local correspondence between a render and a real image the same way a descriptor like SIFT can be used for matching natural images. We illustrate the proposed test-time and train-time workflows below.

3.3.1 Test-time Workflow

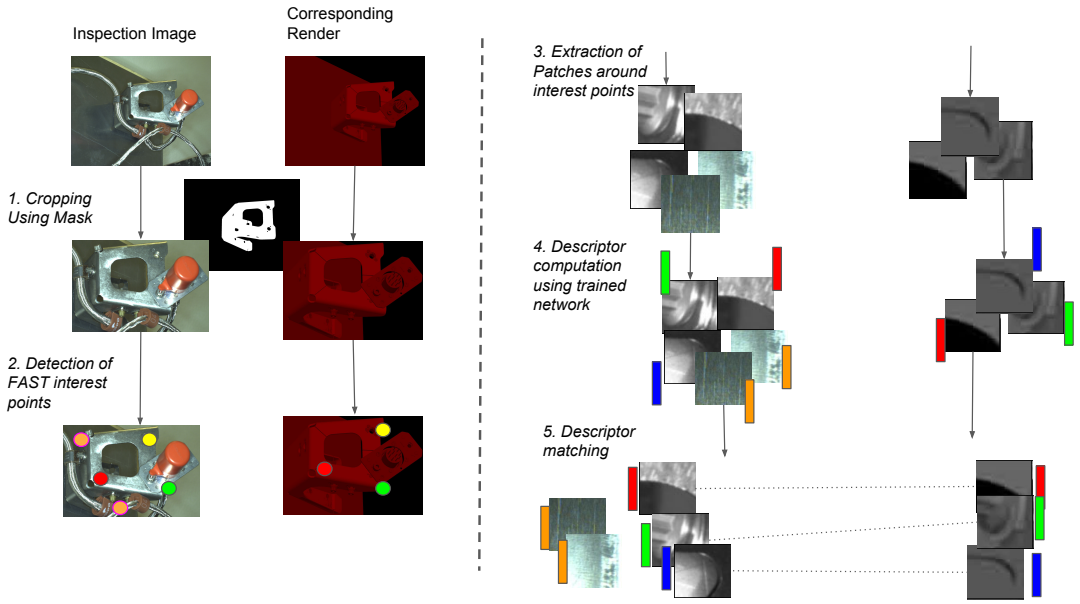


Figure 3.1: Proposed inspection workflow at test time. In the real inspection image, texture patches are also detected as containing interest points due to use of FAST detector.

Figure 3.1 describes the proposed workflow for matching an inspection image with a corresponding simple render generated at run-time using the estimated inspection camera pose.

Step 0 : The render appears red because the shader program outputs the computed gray-level for each fragment only into the red channel. When extracting patches and computing the descriptor, only the red channel of the render is used. Details of the simple shader can be found in Chapter 4.

Step 1: A binary mask for cropping the two images around where the inspected element is expected to be is created by projecting the inspection element in CAD using estimated viewpoint of the inspection camera. We use the dilated version of the mask shown in Figure 3.1.

Steps 2, 3, 4 : Unlike SIFT [41] and LIFT [94], we do not have a dedicated detector coupled with a descriptor. We use the FAST detector [74], and always extract patches of a fixed size

centered at interest points at original image scale. This is because our matching does not need to be scale invariant, the render and the real camera are at similar distances from the virtual and the real inspection objects respectively.

Once the patch descriptors have been computed, a crop level inspection decision has to be made, either by nearest-neighbour matching of descriptors as shown in Figure 3.1 and making a decision using number of good matches and homography between corresponding feature points, or some other approach, like matching feature triplets or searching for a BoW (Bag of Words) model constructed from descriptors computed only on rendered image in the real image. These approaches have been explored further in the final chapter (Chapter 7).

3.3.2 Train-time Workflow

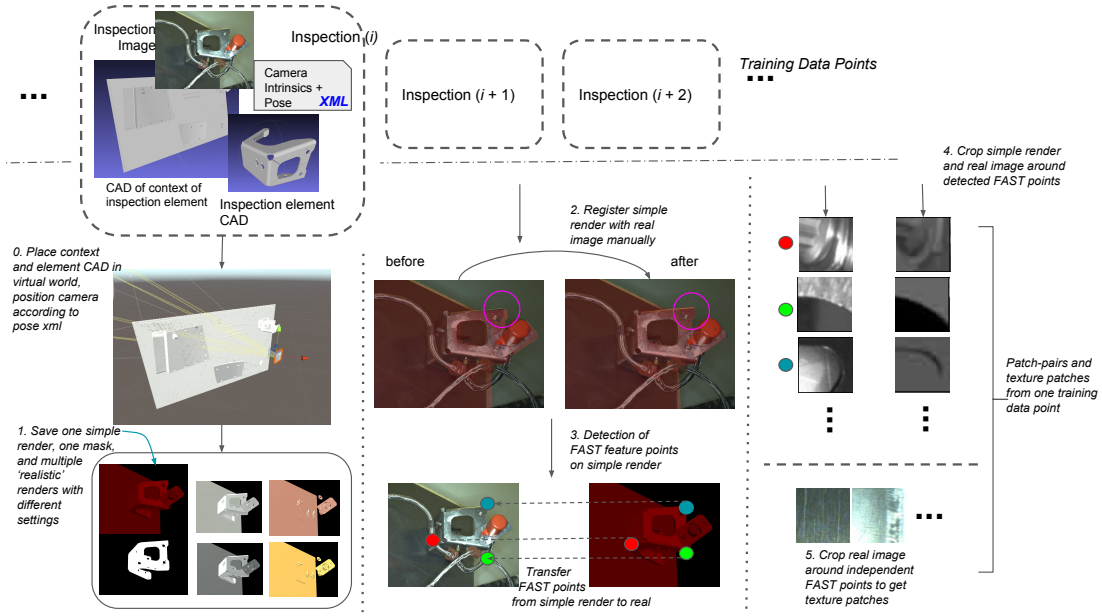


Figure 3.2: Proposed method for creating patch pair dataset for training a descriptor.

Figure 3.2 illustrates the proposed procedure for generating a set of corresponding real and synthetic patches from one inspection instance. An inspection instance is a set which consists of one real inspection image, one CAD of the inspected element, one CAD of the context around inspection element, and one XML that describes the extrinsic and intrinsic parameters of the inspection camera for the inspection image. We explain below the steps illustrated in Figure 3.2. The proposed approach for training a neural network descriptor using the patch-pairs

generated is illustrated in Figure 3.3 and explained thereafter.

Proposed Work-flow for creating patch-pairs

Step 0 : The context CAD and the CAD of the inspection element are loaded onto the rendering platform, and the virtual camera is positioned relative to the CADs using the viewpoint of the real inspection camera available in an XML file.

Step 1: A simple render, like the one to be used at test-time (Figure 3.1), is produced using the viewpoint in the XML. A simple shader is used for this render, described in section 4.3. A binary mask is created by projecting only the inspection element CAD, all areas outside this projection are set to zero intensity (see Chapter 4 for details).

A set of 'realistic' (meaning randomly colored and textured) renders are also produced along with the simple render and inspection mask using different lighting and materials. These renders are planned for use instead of real images in future work as described in Chapter 8. Some work using these renders is presented in section 6.2.

Step 2: The measured pose of the inspection camera has some noise. Because of this, the renders and the real inspection image do not have exactly the same viewpoint. In order to generate matching patch-pairs by cropping image pairs at same image locations, the images need to be registered. Because the images come from different domains, automatic registration based on intensity or features like SIFT, SURF [97] is not feasible. Further, many elements that are present in the real assembly are missing in CAD (specially, deformable elements like wires, and plastic caps). For these reasons, renders are registered manually with real inspection images by manually selecting corresponding points between the real inspection image and the simplistic render. See Chapter 5 for details.

Step 3: FAST feature locations are detected in the simplistic render, and these locations are considered in the corresponding real inspection image as well for producing crops. The inspection element mask and ground truth corresponding to the inspection image (element present/absent) are used for deciding whether a patch that lies inside the inspection mask should be added to the patch-pair dataset. See Chapter 5 for details.

Step 4: Patch-pairs are created by cropping the registered renders and the real inspection image at same locations that are derived from the list of FAST feature locations as mentioned above. In addition to patches containing FAST corners, we also want texture patches from the real image in the patch-pair dataset. Hence, FAST features are detected independently on the

real inspection image as well. See Chapter 5 for details. Samples from 128x128 patch-pair-dataset have been presented in Figure 3.4.

Proposed Work-flow for Learning a Cross-domain Descriptor

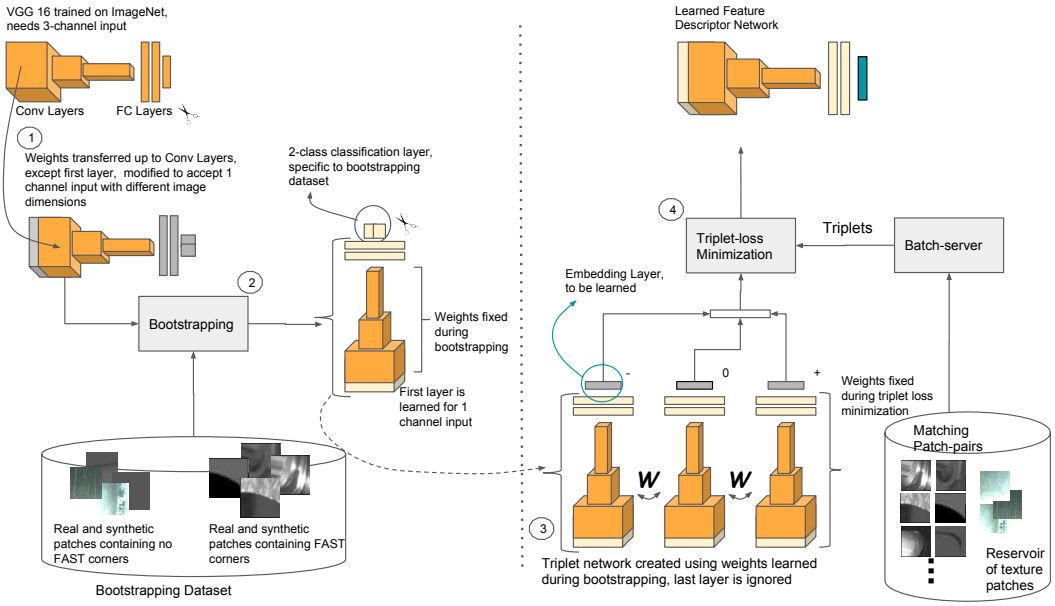


Figure 3.3: Proposed method for training a descriptor using a dataset of patches created as described in Figure 3.2

Step 1 : The convolutional layers of the descriptor network have the same architecture as VGG 16 and are initialized using ImageNet weights [52]. The weights for the first convolutional layer are initialized from scratch to contain 2D filters because unlike VGG 16 (containing 3D filters), we want to work with grayscale images. See Chapter 6 for details.

Step 2 : 3 fully connected layers are added ahead of the convolutional layers, the last of which is followed by a softmax layer. The softmax layer outputs probabilities of a n input patch being a texture patch or a patch containing geometric details, regardless of the domain (real or synthetic) that the patch comes from. We name this step bootstrapping, following the terminology in [64], and illustrate that it is important for addressing domain shift (more details in Chapter 6).

Step 3 : A triplet network is created using weights arrived at at the end of the bootstrapping process, except those for the last 2-class classification layers (linear + softmax). A linear layer (without bias) is added, initialized from scratch, and set as the only layer whose weights can be updated during the triplet-loss training process. This layer is the one responsible for extracting discriminative embeddings. The weights of the triplet towers are tied, training using back-propagation updates weights associated with all towers. More details are in Chapter 6.

Step 4: To train a triplet network, a batch server produces triplets from the patch-pair dataset and a collection of texture patches. To produce 'good' triplets, we perform hard negative mining, which is described along with other details in Chapter 6. The weights obtained after training by minimizing triplet loss are the one that are used in the final descriptor network.

The neural-network based descriptor obtained at the end of the proposed training process can be used for matching renders with real images as described in Figure 3.3 or in other ways as described in the Chapter 7.

In the following chapter, we detail the rendering pipeline.

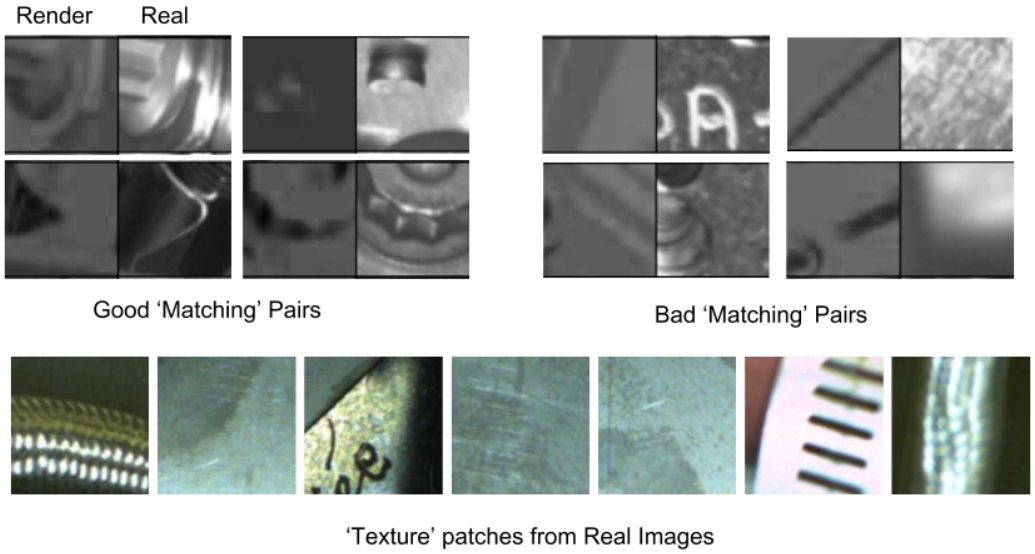


Figure 3.4: Top Row : Examples of 128x128 patch pairs generated as matching pairs. Not all supposedly matching pairs actually match due to the CADs being simplistic. Bottom Row: Examples of 'texture' patches generated from real images (refer Figure 3.2). As can be seen, these usually come from areas in the real image with scratches, text, or elements such as wires and ties that are not present in the CADs. The patch-pair dataset created in a semi-supervised manner is not perfect. This has implications on learning of the descriptor.

Chapter 4

Synthetic Data Generation System

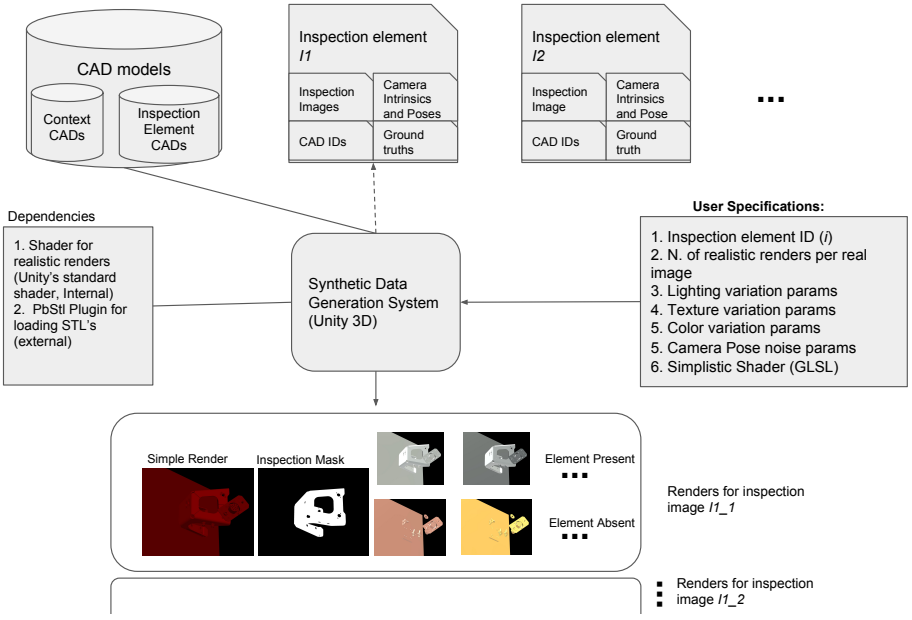


Figure 4.1: Synthetic data generation system designed in Unity 3D.

As described in Section 2.2, we use Unity 3D to render our 3D CAD models to generate training data. Figure 4.1 presents a simple illustration of the implemented system.

An Inspection Project at Diota: At Diota, for every project, there exists a collection of CAD models in a database. A project is a set of inspection tasks for a client consisting of a set of elements to be inspected and the kinds of inspections to be performed, which can be presence/absence of parts, misalignment, etc. For every element to be inspected, a CAD model is specified along with a context CAD, which can be common to several inspection elements.

Data for producing renders: The context of this thesis work is a robotic inspection project consisting of several inspection elements, like supports, screws, etc., for each of which we have a set of real inspection images along with camera calibration, pose, inspection ground-truth (OK/NOK) and inspection and context CAD IDs for each inspection image. The synthetic data generation system can be 'pointed' at the database corresponding to a particular inspection element, and for each inspection image therein, the system produces a set of renders using viewpoint and other associated information mentioned before.

Inspection element central to this work: To avoid complications due to small sizes of inspection elements and simplistic nature of our CAD models, in this work, the focus is on 'supports'. Supports are very common elements in mechanical assemblies, and in an assembled piece, they are often present against a cluttered background along with deformable elements like wires and plastic caps on top of them, which are not present in the CAD. Also, they have many points with interesting geometry such as corners, curves, etc.

4.1 Rendering Procedure Corresponding to One Real Inspection Image

For each inspection image, a set of colored and textured renders (termed realistic renders) are produced with inspection element CAD present and absent against the Context CAD. The reasons behind this choice will be explained in Chapter 6 and 8.

Besides the colored and textured renders, for each inspection image, one 'expected' (or simplistic) render utilizing the simple shader, and one binary inspection mask are also rendered, both of which are described below.

4.1.1 Transformation of Camera Pose from Right-handed to Left-handed System

From Diota's tracking kernel (existing software with protected source code), for each real inspection image, we have an estimated transformation from the camera frame to the (CAD)

Algorithm 1 Rendering Procedure for a real inspection image

```
for Each Real Inspection image do
    Get IDs of Context CAD and Inspection CAD
    Get camera pose and camera calibration
    Position virtual camera according to Pose in Step 3
    Apply camera calibration fetched in Step 3 (as FOV and image resolution)
    for Number of desired realistic renders do
        Apply random color and texture to Context CAD
        Apply a different random color and texture to Inspection CAD
        Apply perturbations to global lighting
        Save Render
    Apply simplistic shader to Context and Inspection CADs
    Turn-off all lighting
    Save expected/simplistic render
    Apply a diffuse-white material to inspection CAD
    Render only the Inspection CAD
    Save the render as Inspection Mask
```

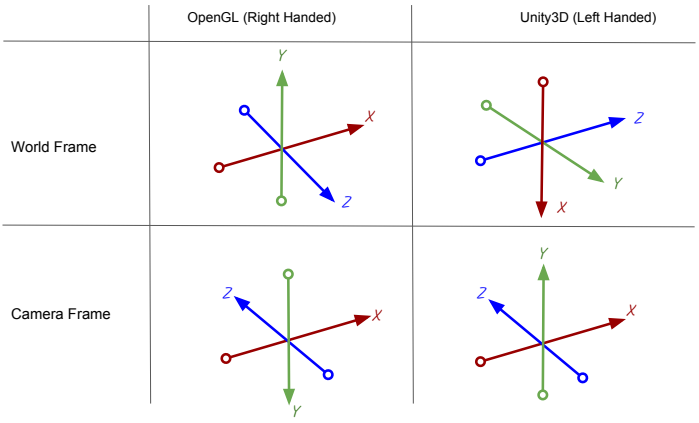


Figure 4.2: World and camera frames in OpenGL (Right handed), as in Diota's Kernel, and Unity 3D (Left Handed)

model frame (${}^{M}O_{C}$). Our world frame is the same as the model frame. Additionally, all the CAD models (context and inspection) for a particular inspection are in the same frame. All this information is in the right handed frame as detailed in the OpenGL column of Figure 4.2. Our goal is to place and orient the camera in the left-handed virtual world frame (of Unity 3D) using the information in ${}^{M}O_{C}$.

Say ${}^{M}O_{C} = \begin{bmatrix} {}^{M}R_{C} & {}^{M}T_{C} \\ 0 & 1 \end{bmatrix}$ Then, a point P represented in the camera frame as ${}^{C}P$ is related to its representation in the model/world frame ${}^{M}P$ by :

$$ {}^{M}P = ({}^{M}R_{C}).{}^{C}P + {}^{M}T_{C} \tag{4.1} $$

So,

$$ {}^{C}P = ({}^{M}R_{C})^{T}.{}^{M}P - ({}^{M}R_{C})^{T}.{}^{M}T_{C} \tag{4.2} $$

In (4.2), $(-{}^{M}R_{C}^{T}.{}^{M}T_{C})$ is the translation from the world/model frame to the camera frame in the right-handed system of Diota's tracking software. To convert the position (translation) of the camera to a left-handed system, we multiply it with a right-hand to left-hand linear transformation matrix.

$$ P_{L} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} .P_{R} \tag{4.3} $$

P_L and P_R are the positions of the same point P in left and right handed world frames respectively. Next, we want to change the rotation of the camera from right-handed convention in OpenGL to the left-handed convention in Unity 3D. In OpenGL, the camera-to-world rotation matrix contains in its columns the camera right, camera up and camera front axes respectively. We want to transform these three axes to the left handed unity system.

$$ {}^{Rw}R_{CR} = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} \tag{4.4} $$

${}^{Rw}R_{CR}$ is the rotation from right-handed camera to right handed world frame. We want ${}^{Lw}R_{CL}$, rotation from left-handed camera to left-handed world. ${}^{Lw}O_{Rw}$ denotes transformation from right-handed world to left-handed world.

$$ {}^{Lw}R_{CL} = {}^{Lw}O_{Rw}.{}^{Rw}R_{CR}.{}^{CR}R_{CL} $$

$$ {}^{Lw}R_{CL} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} . \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

Metal	Red	Green	Blue
Iron	0.560	0.570	0.580
Silver	0.972	0.960	0.915
Aluminium	0.913	0.921	0.925
Gold	1.00	0.766	0.336
Copper	0.955	0.637	0.538
Chromium	0.550	0.556	0.554
Nickel	0.660	0.609	0.526
Titanium	0.542	0.497	0.449
Cobalt	0.662	0.655	0.634
Stainless Steel	0.878	0.874	0.859
Bronze	0.714	0.428	0.181
Brass	0.714	0.428	0.181

Table 4.1: RGB color values for different metals.

$$^{Lw}R_{CL} = \begin{bmatrix} -x_1 & y_1 & -z_1 \\ x_2 & -y_2 & z_2 \\ x_0 & -y_0 & z_0 \end{bmatrix}$$

In $^{Lw}R_{CL}$, we have the camera right, up and front axes in the left-handed world as linear transformations of the same camera axes in right-handed system.

4.2 Automatic Coloring and Texturing of Plain CADs

The CAD models that the clients of Diota provide lack corresponding materials (shader programs and textures). For providing realistic metallic colors to our CAD renders, we use the color values detailed in Table 4.1. [21]

While it is sufficient to specify the same color values as vertex attributes to color the render of a plain CAD, in order to texture it, a procedure termed UV-unwrapping is necessary. In UV-unwrapping, each vertex is assigned a (u, v) coordinate corresponding to a location in a texture map, which is a 2D image. This process can be manual whereby an artist decides how a mesh is to be cut-open, or automatic whereby a software cuts a mesh open at the seams. A texture is then defined for each triangle of the mesh projected onto a flat surface. [93].

In Unity 3D, an automatic solution exists whereby UV's are generated automatically for each triangle (Unwrapping.GeneratePerTriangleUV in UnityEditor namespace [83]). A texture map can then be assigned to the material associated with the mesh. We choose randomly one of 6000 textures available in the DAGM industrial datasets [42]. Some of them have been presented in Figure 4.3 b. The colored and textured renders do not contribute to discriminative

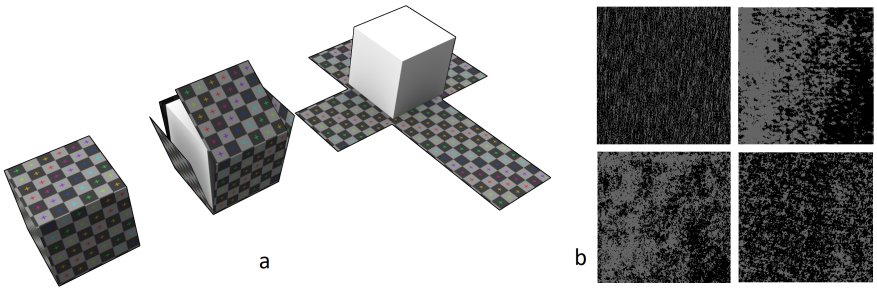


Figure 4.3: (a) UV mapping for a cube, a texture can be defined after the cube has been cut open, figure from [93]. (b) Examples of used texture patches from DAGM Industrial dataset [42]

feature learning in this work and are of importance mainly for future work.

For producing the colored renders shown in Figure 5.1, Unity 3D's standard shader was used. The textures were applied as Metallic Gloss Maps. We produce textured renders in order to have patches that can be used to augment real patches obtained from real images when there are not many of them, as described in future work. This choice is in view of the ideal goal of being able to train an inspection system for a new project (with a previously unseen set of CAD models) when no real images are available for the inspection elements therein.

4.3 Simple Shader for Generating 'Expected' Image

The colored and textured renders described previously utilize a complex shader program. It is not feasible to generate such renders in real-time using tracked pose of the inspection camera to produce for every real image an expected image with which it might be compared.

Hence, we propose a simple shader, similar to Phong Shading [10], which assigns an intensity to each fragment based on the orientation of its normal relative to the camera optical axis, its depth along the optical axis, and the CAD to which it belongs (context or inspection element), without considering any lighting. These choices were made in order to have edges in the render at locations where the CAD has discontinuities in surface normals, surface depth and/or element identity (inspection element has to stand out from the context at locations where the inspection and context surfaces are parallel and at same depth from the inspection camera). These choices were inspired by [35]. In practice, we observe that even when we do not transform the normals from model space to view space, the shading still captures edges due to normal discontinuities.

As illustrated in Figure 4.4, the gray-level of each fragment, L, is based on the orientation of its normal n in the camera/view space, and its distance from the camera along the optical

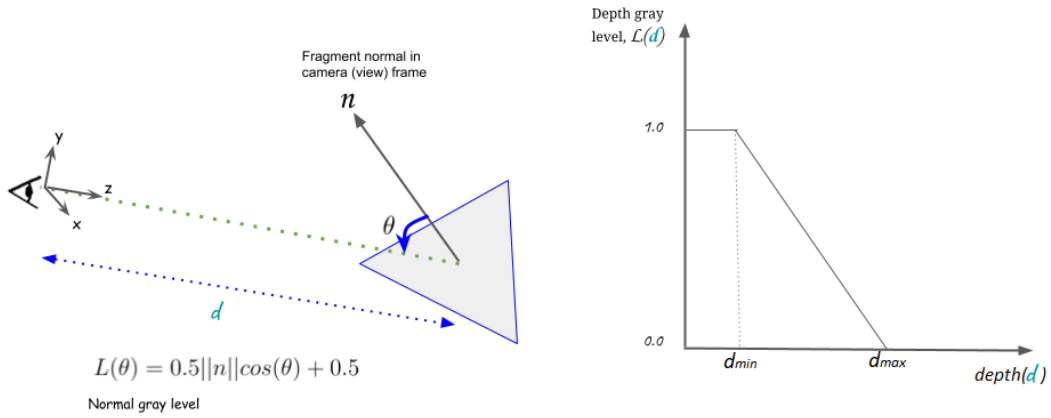


Figure 4.4: Proposed simplistic shading for rendering expected view in real-time during inspection. The fragment color is based on the fragment's normal in camera-space and its distance from the camera. $\{^{C}n = ((^{C}T_{W}.^{W}T_{M})^{-1})^{T}.^{M}n\}$ relates fragment normal in model-space and camera-space.

axis. It is computed as an affine combination of two gray-levels as follows:

Shading from fragment normal :

$$L(\theta) = 0.5||n||cos(\theta) + 0.5$$

where 'n' is the fragment normal in camera (or view) space. If ^{M}n is the normal in model space, $n = ((^{C}T_{W}.^{W}T_{M})^{-1})^{T}.^{M}n$, where $^{W}T_{M}$ is the model matrix and $^{C}T_{W}$ the view matrix. Shading from fragment depth :

$$L(d) = 1 - min\left(1, max\left(0, \frac{d - d_{min}}{d_{max} - d_{min}}\right)\right)$$

All fragments closer than d_{min} are assigned maximum depth intensity and all further than d_{max} 0. Finally, $L(\theta)$ and $L(d)$ are combined as :

$$L = \alpha L(\theta) + (1 - \alpha)L(d) \ , \ 0 \leq \alpha \leq 1 \tag{4.5}$$

The weights associated with intensities from normal and depth determine where an edge is perceived in the render as illustrated in Figure 4.5. We set α at 0.5. The distances 'd_{min}' and

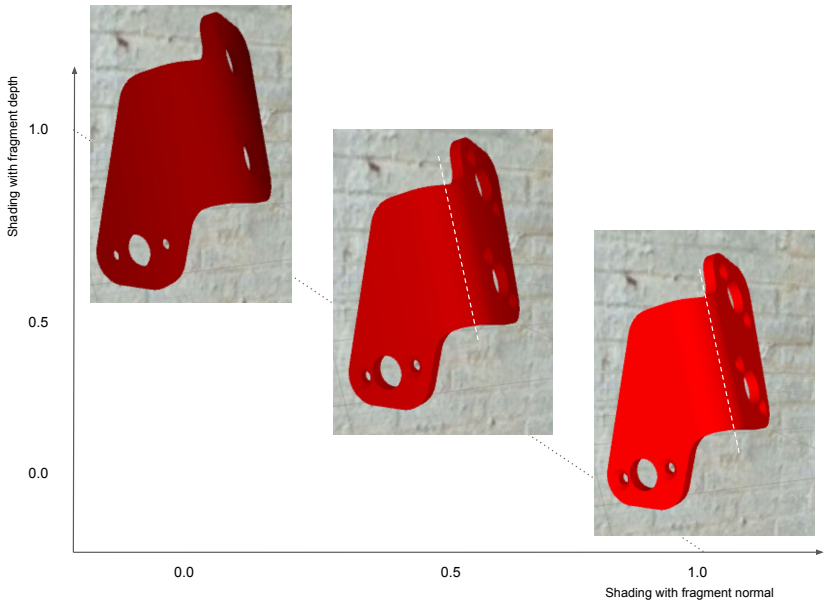


Figure 4.5: Effect of coefficients attached with shading intensity from fragment normal and fragment depth. Combining intensity from both enables a more accurate perception of edges.

'd_{max}' affect the contrast achieved from shading using depth. In Diota's software, when using a depth camera, these distances are selected automatically for optimal contrast based on model tracking relative to the camera. However, when producing renders for creating training data, we set these values at 10 cm and 1 m respectively.

4.4 Generation of Binary Inspection Mask

For producing the inspection mask, all lights are 'turned off' in the rendering environment. A material with ambient white color 'FFF' is applied to the inspection mesh. No other meshes are rendered. This produces a black and white render which is a shadow-projection of the inspection element. No details of the elements are visible except the contours (see Fig. 4.6).

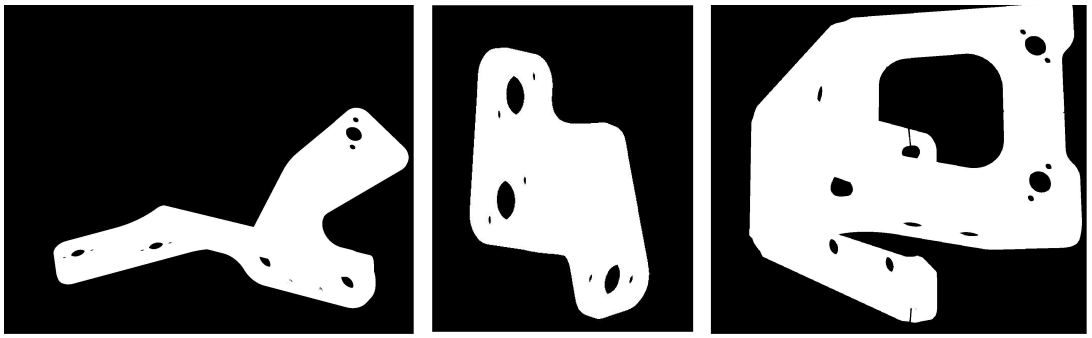


Figure 4.6: Examples of binary inspection masks.

Chapter 5

Dataset of Patch Pairs

The synthetic data generation system described in Chapter 4 produces for each real inspection image a simple render, an inspection mask and a set of realistic renders as illustrated below in Figure 5.1.

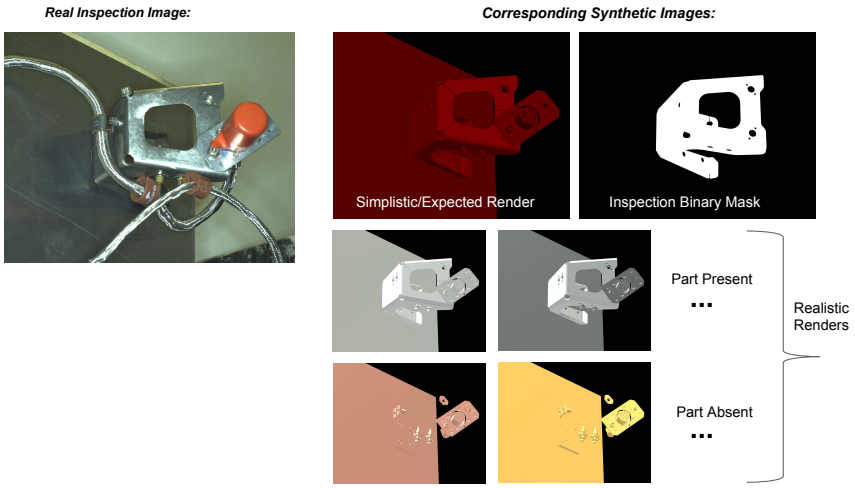


Figure 5.1: All renders produced for one real inspection image.

Terminology : 'Crops' are created from 'Images' by using the binary inspection mask. 'Patches' are created either from 'crops' or 'images' by making a fixed size crop centered at an interest point. Hence, Images > Crops > Patches. We are interested in comparing crops.

We have $\sim$ 100 real inspection images with corresponding camera poses and calibration. Although we may think about a system that compares crops of real and expected images directly, we do not have enough images to train such a system. Additionally, at crop level, differences between real images and renders are more prominent since deformable elements such as wires and plastic caps are present in the real image and completely absent in the renders due to simplistic CADs. Therefore, we choose to compare images at patch level rather than directly at crop level.

5.1 Manual Registration

The camera pose that we have for each real image is noisy. Hence, the renders that are generated using these poses are not perfectly registered with real images. Because we want to generate patch pairs from image pairs (real, synthetic) by cropping the image pairs at same image locations, perfectly registered image pairs are necessary.

Methods based on local features, intensity, mutual information, etc. have been proposed for automatically registering images [97]. However, these methods are not applicable in the context of this work. This is mainly due to the simplistic nature of our CAD models (no texture and deformable elements). Therefore, we manually register our image pairs by selecting corresponding control points in every image pair.

Given a set of manually selected corresponding locations P_R and P_S between two images I_R and I_S, we estimate an affine transform between them. An affine transformation between 2D images has 6 parameters, so we need at least 3 corresponding control points.

If $\{(x_{Ri}, y_{Ri})\} \in P_R$ and $\{(x_{Si}, y_{Si})\} \in P_S$, we can define an affine transformation between the two sets as :

$$x_{Ri} = a.x_{Si} + b.y_{Si} + c$$

$$y_{Ri} = d.x_{Si} + e.y_{Si} + f$$

Considering all corresponding 2D points in P_R and P_L, we can write a linear system, $A\theta = b$, to solve for the transformation parameters $\theta = [a, b, c, d, e, f]^T$ as follows:

$$\begin{bmatrix} x_{S1} & y_{S1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{S1} & y_{S1} & 1 \\ x_{S2} & y_{S2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{S2} & y_{S2} & 1 \\ & & \vdots & & & \end{bmatrix} . \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_{R1} \\ y_{R1} \\ x_{R2} \\ y_{R2} \\ \vdots \end{bmatrix}$$

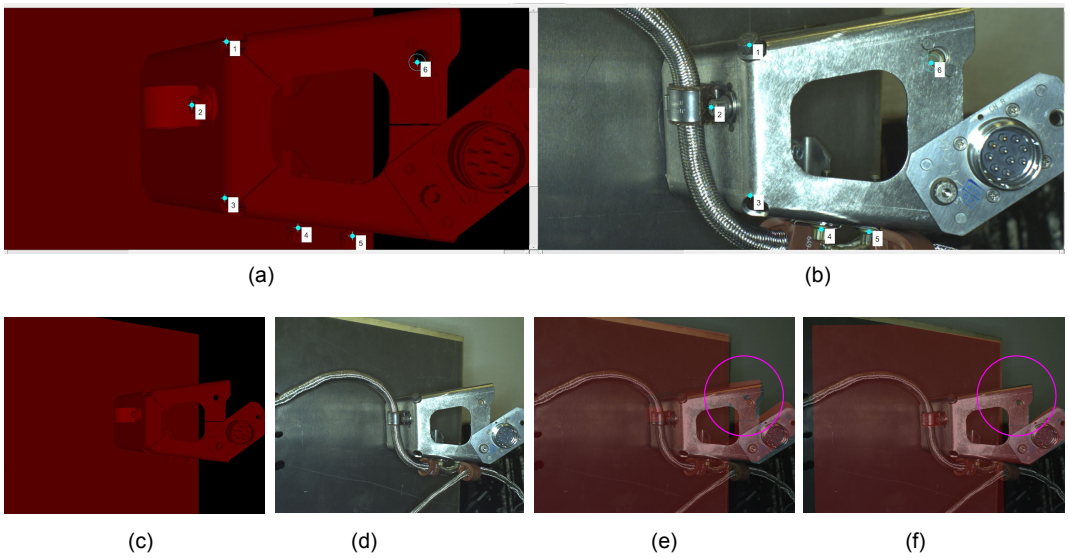


Figure 5.2: Manual registration of simplistic render with real image. (a) and (b) show manually selected corresponding points between the two images. (c) is the simplistic render, and (d) is the corresponding real image. (e) and (f) show overlapped visualization of the two images before and after registration respectively. Effects of registration are clearly visible within the pink circle.

The transform parameters are solved for by using linear least squares; $\tilde{\theta} = A^T(A^TA)^{-1}A^Tb$. The estimated affine transform is used for warping all synthetic images (simplistic and realistic renders, and inspection mask) corresponding to a real inspection image.

5.2 Patch-pairs from a Single Registered Image pair

From every image pair (like (a, b) in Figure 5.2), after registration, a set of patch-pairs are produced as described in Algorithm 2 below. For each image pair, FAST points are detected only in the synthetic simplistic render. This is because locations where FAST points are detected in the simplistic render are locations with edges due to geometry (fragment depth and normal discontinuities), and not edges due to texture since simplistic render has no texture. We are interested in matching patches using such geometric information and being invariant to texture.

Crops are made at locations of the FAST points in both synthetic and real images. Crop at each location results in a pair (or set, if realistic renders are also used, see Figure 5.1) of

patches. If the location of the FAST point lies inside the corresponding inspection mask, the set of patches is retained only if the inspection ground truth for the real image is True/OK signifying conformity of inspection image with CAD.

In addition to saving patches in the real image that have 'geometrically meaningful' information, we are also interested in saving patches where the FAST detector fires due to texture. This is to have a dataset of texture patches for training a descriptor that can ignore texture information in real images. These patches are created as described in Algorithm 3. Concretely, FAST points are independently detected in the real image, and those points at or near which the FAST detector also fired in the registered simplistic render are discarded.

Algorithm 2 Patch-pairs from one Image Pair

procedure PATCH_PAIRS($R, S, M, G(R)$)
  $S \leftarrow Simple\_Render, R \leftarrow Inspection\_Image, M \leftarrow Inspection\_Mask$
  $F \leftarrow$ FAST_Points(S)
  for `f in F` **do** ▷ f is an image location (u, v)
    if not M(f) **then**
      $C_S \leftarrow S(W_f)$ ▷ W_f:Cropping window centered at f
      $C_R \leftarrow R(W_f)$
      Save C_S, C_R
    else
      if $G(R)$ **then** ▷ G(R) : Inspection ground-truth for R
        $C_S \leftarrow S(W_f)$
        $C_R \leftarrow R(W_f)$
        Save C_S, C_R
      else
        Continue

Algorithm 3 Real Texture-Patches from one Image Pair

procedure TEXTURE_PATCHES(R, S, M)
  $S \leftarrow Simple\_Render, R \leftarrow Inspection\_Image, M \leftarrow Inspection\_Mask$
  $F_S \leftarrow$ FAST_Points(S), $F_R \leftarrow FAST\_Points(R)$
  $F'_R \leftarrow [\,]$
  for `f in` F_R **do** ▷ f is an image location (u, v)
    if f not in F_S **then**
      $F'_R \leftarrow F'_R.append(f)$
  for f' `in` F'_R **do**
    $C_R \leftarrow R(W_f)$ ▷ W_f:Cropping window centered at f'
    Save C_R

Chapter 6

Domain Invariant Descriptor Learning

6.1 Training Deep Networks for Comparing Images

Comparing images and image patches using deep-learning approaches has been already seen many meaningful contributions [95] [31] [32]. Two network architectures are prominent in learning features and metrics for image-comparison : siamese network, and triplet network.

[95], [32] use siamese architecture to jointly train a feature-extractor network and a metric network by using discriminative (classification) losses; [95] use hinge loss and [32] use cross-entropy loss (see Table 2.1). [8] also train a siamese-network for object tracking (using a patch-comparison approach) by minimizing logistic loss.

Similarly to [95] [32] and [8], [31] also use a siamese architecture to train a feature network for image comparison, but dissimilarly from them, they aim directly at learning a feature network that produces discriminative embeddings (features) in Euclidean sense by minimizing contrastive loss. [95] report that minimizing hinge loss at the output of the classification network connected to the output of the feature network causes the feature network to extract embeddings that are discriminative in the l_2 sense. Learning such embeddings is interesting due to possibility of easy (Nearest Neighbour) matching between descriptor sets using existing methods (like brute-force matcher in OpenCV).

For learning discriminative embeddings, the triplet network architecture is more favored than the siamese network [6] [78] [76] [64]. In the patch-comparison literature, [6] and [78] learn feature networks for discriminative embeddings by using a triplet architecture and minimizing triplet loss which consists of two distances, one between a matching pair, and one between

a non-matching one. 2-pairs come from 3 patches (a triplet of patches), popularly termed 'anchor', 'positive' and 'negative' patches, one pair being (anchor, positive) and the other being (anchor, negative). In face recognition literature, where images are usually larger than in patch comparison literature, using triplet architecture and minimizing triplet loss are de facto for learning discriminative embeddings [76] [64].

In all the works mentioned above, networks are trained from scratch using datasets that contain upwards of a million images. When such large datasets are not available, transfer learning is an attractive approach. However, in the literature for patch comparison, works that use transfer learning are scarce. One example is [8], who use a siamese network for comparing an exemplar with a candidate image for tracking objects. Their feature extractor network is trained on the ImageNet object detection dataset for object detection task, i.e., the networks in the siamese tower are not trained directly to produce discriminative embeddings.

6.2 Initial Insights Behind Proposed Approach (from Transfer Learning)

It has been shown that a CNN trained on a large dataset of natural images for a generic image-related task (like classification) can be used as a discriminative feature extractor without any further training on target natural images that come from classes not seen during training [71] [53] [8]. Imagenet [20] weights are widely used for this practice and the VGG16 architecture is a popular choice due to its simplicity [42] [53]. The 'features' are often extracted from the fully connected layer that is right before the final classification layer (softmax layer) or the one right after the final convolutional layer (bottleneck).

When using above-mentioned off-the-shelf features [71], images that contain similar patterns are mapped to nearby locations in the feature space, and those that contain dissimilar ones are mapped far away. This is interesting in our context of industrial inspection where we do not have images of OK and NOK cases for every element to train dedicated classifiers. However, for most elements to be inspected, we do have CAD specifications. It is therefore interesting if features from renders with expected configurations and actual images with coforming configurations can be mapped close together, and those with non-conforming ones mapped far away.

We hence test off-the-shelf features extracted from image crops that either contain or do not contain a screw, an omnipresent part in industrial assemblies. The crops come from real images and corresponding realistic renders. We thus have 2 classes and 2 domains, and we want to see separation only between classes, not between domains, in the mapped features. We used VGG16 pretrained on Imagenet [20] whose weights were fetched from [52]. Features were extracted from the fully connected layer preceding the final classification (softmax) layer.

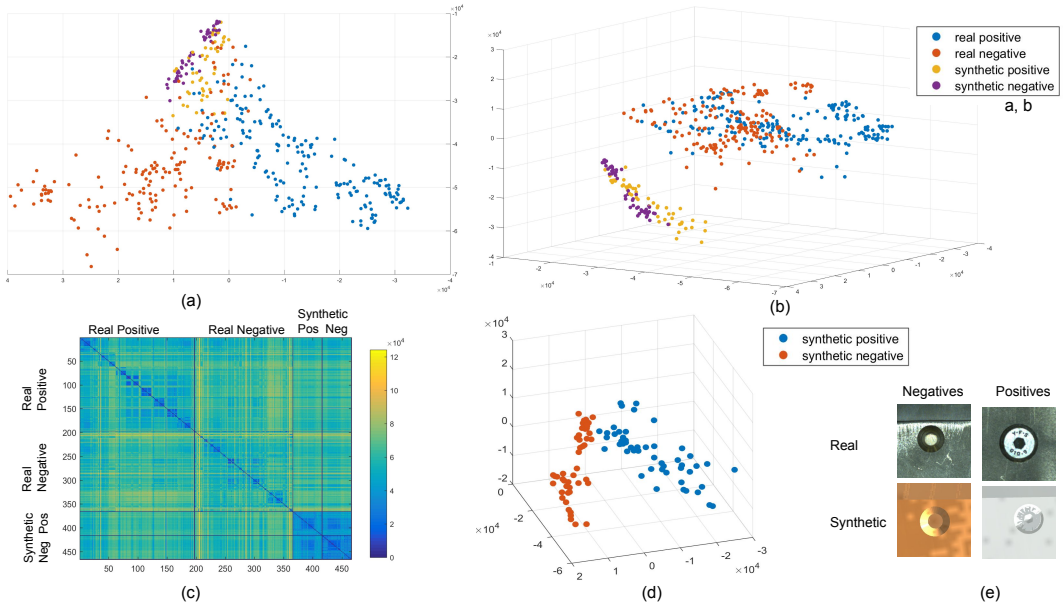


Figure 6.1: Principal Component Analysis [92] on off-the-shelf features extracted from synthetic and real positive (containing screws) and negative (missing screws) patches. a, b : PCA on features extracted from ensemble of real and synthetic features, c: pair-wise distance between feature vectors, d: PCA on features extracted only from synthetic patches, e: examples of real and synthetic positive and negative patches

As visible in Figure 6.1 (a), off-the-shelf features clearly separate positive and negative real patches (positives being the ones containing screws). No further training of the convolutional network was necessary to achieve such features. However, as visible in Figure 6.1 (b), the non-fine-tuned network also separates synthetic patches from real ones, regardless of the content of the patches. Figure 6.1 (c) shows the pair-wise distances between features extracted from all test patches. As can be seen, the non-fine-tuned network maps all synthetic patches very close to each other. However, in the features extracted only from synthetic patches, differences are still visible between the two classes in Figure 6.1 (d). So, the network pre-trained on natural images does make some distinction between synthetic images with different features.

In order to be able to train a classifier that works with real image patches using only synthetic data, a feature network that maps patches from different domains with similar contents to nearby locations in the feature space is necessary. In the case of the patches shown in Figure 6.1, it was possible to have such a feature extractor by fine-tuning the convolutional layers and the fully connected layers of the VGG16 initialized with ImageNet weights [52]. The final

linear+softmax layer with 1000 neurons was replaced with a 2-neuron linear+softmax layer and the network was trained to minimize cross entropy loss, see Table 2.1, for real and synthetic patches that were labeled either 0 or 1 (all synthetic and real patches that contained a screw were labelled 1, and all that did not were labeled 0). Weight updates were applied to the last three convolutional layers and all the fully connected layers, while the rest of the network was 'frozen' during the fine tuning procedure. Figure 6.2 presents the results.

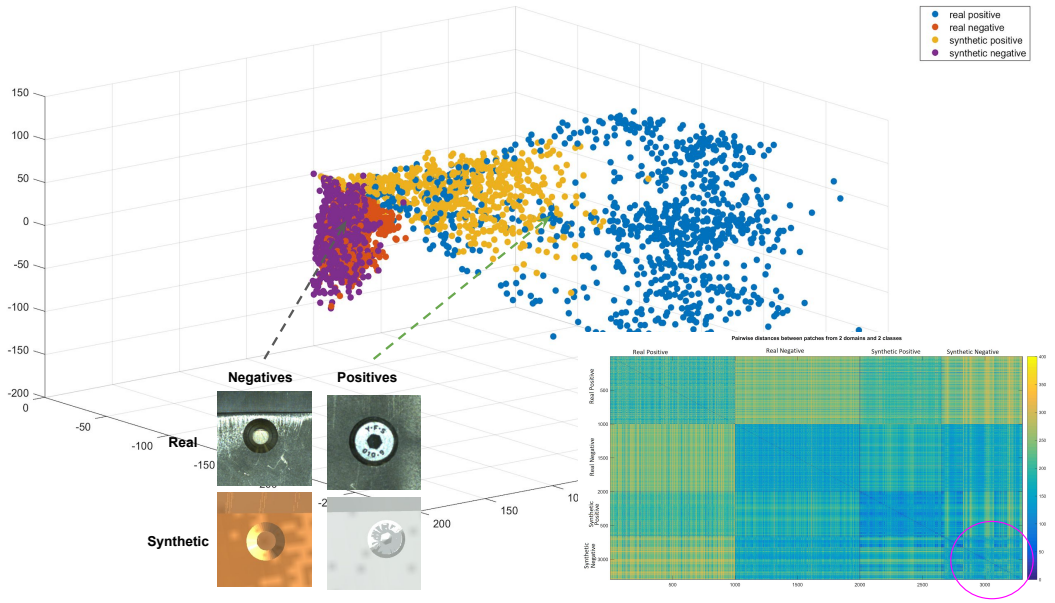


Figure 6.2: Principal Component Analysis [92] on features extracted from synthetic and real positive and negative patches after the network was fine tuned on the mixed (synthetic-real) dataset.

As seen in Figure 6.2, after fine-tuning by minimizing 2-class classification loss on the mixed dataset, the VGG16 network was able to extract features from synthetic and real patches such that patches with similar content are mapped in the same region of space regardless of their domain of origin.

This method of tackling domain shift is evidenced else-where as well; [73] mention training on batches that contain both real and synthetic images, [42] report more discriminative features on target dataset of industrial textures, very unlike source dataset ImageNet that consists of natural images, after fine-tuning using a classification.

If the above mentioned approach is to be used for learning a good feature extractor for random patches centered at interest points, it would imply selecting a 'good' subset of such

patches (which could be in tens of thousands) and training a classifier to classify these patches into the above-said thousands of classes, while having only 2 (or 10's in case realistic renders corresponding to simplistic ones) patches per class. This is an ill posed problem. We address this problem with a 2-stage training strategy, consisting of a bootstrapping step for addressing domain shift by learning classification over only 2 classes, as described in the following section.

6.3 Two-Stage Training Approach For Learning Domain-Invariant Descriptor

As illustrated by Figures 6.1 and 6.2, domain shift is a serious problem when dealing with synthetic and real images, and it is imperative to address this issue in order to learn a descriptor that is sensitive only to geometry within patches, not domain-specific intensity or texture.

A straight-forward approach for addressing this problem might be to train a network from scratch using a large number of patch pairs (millons, as suggested by [6] [95] [78]) that come from synthetic and corresponding real images. However, since in this work, we have less than 100 image pairs from which we can create patch pairs for training, we have only 19000 patch pairs in the training set (except texture patches). A dataset of this size is not enough for training a deep-network based descriptor, necessary when patches are big, as evidenced by [79] [64]. Hence, we use transfer learning approaches as described later in this section.

6.3.1 Network Architecture

We choose to work with VGG16 architecture, detailed in Table 2 of [79] (architecture D). This is because the VGG16 architecture is very simple, used in many references that this work draws from, has easily available ImageNet (ILSVRC 2014) weights [52], and our patch sizes are comparable in size to the image sizes on which VGG16 was trained.

Differently from the original architecture however, we are interested in working with grayscale patches because our simplistic render (as described in Chapter 4) contains intensity values only in one color channel (red, and only red channel is considered at both train and test time). To account for these differences, we modify the VGG16 architecture as detailed in Table 6.1.

Note that the filters in the first layer of the modified architecture are 2-dimensional (different from 3-dimensional filters in the original VGG16 architecture) since we use 1-channel inputs. The number of filters, however, is the same. The 2-dimensional filters are initialized by averaging the 3-dimensional filters along the 3^{rd} dimension, as suggested in [88] (section 2.2). The 3-dimensional mean to be subtracted from each input image to VGG16 was averaged to produce a scalar mean (equal to 114.799).

_128 Bootstrapping	**_224 Bootstrapping**	**_128 TripletLoss**	**_224 TripletLoss**
16 weights layers	16 weights layers	16 weights layers	16 weights layers
conv3-64 (3x3x1)	conv3-64 (3x3x1)	conv3-64 (3x3x1)	conv3-64 (3x3x1)
conv3-64	conv3-64	conv3-64	conv3-64
maxpool	**maxpool**	**maxpool**	**maxpool**
conv3-128	conv3-128	conv3-128	conv3-128
conv3-128	conv3-128	conv3-128	conv3-128
maxpool	**maxpool**	**maxpool**	**maxpool**
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
maxpool	**maxpool**	**maxpool**	**maxpool**
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
maxpool	**maxpool**	**maxpool**	**maxpool**
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
maxpool	**maxpool**	**maxpool**	**maxpool**
FC-1024	FC-4096	FC-1024	FC4096
FC-1024	FC-4096	FC-1024	FC4096
———	———	l_2 normalization	$l_2$normalization
FC-2	FC-512	FC-2	FC-1024
softmax	softmax	———	———

Table 6.1: Architectures of different deep networks used.

Following the notation in [64], we denote the deep-architecture (Convolutional layers followed by 2 fully connected layers, see Table 6.1) with 'ϕ'. Given an input patch $P_i, 1 \leq i \leq N$, the output of the deep-network is a score vector $X_t = \phi(P_i) \in \mathbb{R}^D$. The network ϕ is trained in two steps, similarly to the procedure in [64] (discriminative embeddings for face recognition), as described below.

6.3.2 Training Step I : BootStrapping

We start with the network ϕ, whose architecture is described in Table 6.1 (_128 or _224, both denoted here with ϕ). The convolutional layers of the network are initialized with weights obtained for VGG16 training for 1000-way ImageNet classification [52], and the first convolutional layer is initialized as described in section 6.3.1 to accept 1-channel input. The fully connected layers are initialized from scratch using Xavier initialization [27].

Next, similarly to the bootstrapping procedure described in [64], we append to ϕ a linear layer $(W, b), W \in \mathbb{R}^{N \times D}, b \in \mathbb{R}^N$. However, in our case, $N = 2$; $N = 2622$ in [64], 2622 training identities with 1000 faces per-identity. Similarly to fine-tuning experiments described in section 6.2, the two classes in our bootstrapping dataset consist of mixed patches that were either cropped around those locations where FAST features were detected in the (simplistic) renders (patches with interesting geometry) or those where they were detected only in the real image but not in the corresponding (simplistic) render (texture patches). Next, soft-max cross entropy loss was minimized for the 2-class classification task on predictions $x_i = W\phi(P_i) + b \in \mathbb{R}^2$ of the classification layer (W, b).

$$E = \frac{1}{n} \sum_i [y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})]$$

Where $y_i \in [0, 1]$ and $\hat{y_i} = \frac{e^{x_i}}{e^{x_0} + e^{x_1}}$. The bootstrapping dataset has been illustrated in Figure 6.3.

We also tried bootstrapping similarly to [64]; 5000 patch sets (real and corresponding synthetic (simple and realistic)) were selected randomly, a distinct class label was associated with each set, and the 5000-way classification loss minimized. However, this was a failure; no improvement was seen in classification accuracy even on the training set, and initial weights for triplet loss training (part II) were bad. This was due to a very small number of training patches (only 6: one real, one simplistic, 4 realistic) per class unlike in [64] where 1000 faces for each of 2622 classes were available.

Training Details

We use 38,000 patches in each class for bootstrapping. Batch-size was set to 128, and mini-batch gradient descent optimization was used with an initial learning rate of 0.005. Network was regularized using dropout after each of the two FC layers in ϕ with a rate of 0.5. Weights associated with all convolutional layers except the first were 'frozen'. Hence, only 4 layers received weight updates, namely, the first convolutional layer, the 2 FC layers in 'ϕ' and the classification layer (W, b).

Training was performed for only 2 epochs, which resulted in a classification accuracy of 0.98 on the training set. For the bootstrapping stage, we are not concerned with performance on a test or validation set since it is only an intermediate step.

6.3.3 Training Step II : Triplet-Loss Training

The goal of the deep learning approach developed in this thesis is to obtain discriminative embeddings that can be used for matching patches between real and synthetic images with

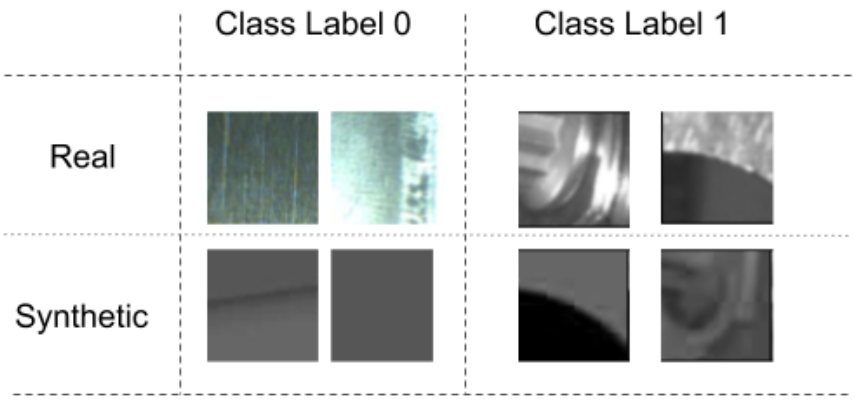


Figure 6.3: Patch-pairs organized for bootstrapping. All patches containing 'geometric features', whether real or synthetic, are put in one class, and all 'texture' patches are put in another. Texture patches from real images (Class 0) were converted to grayscale, although they have been shown in color here.)

similar viewpoints. Although the bootstrapping process described in the previous sub-section presumably helps in addressing domain-shift problem, the deep-network ϕ at the end of the bootstrapping process does not produce discriminative embeddings.

Hence, again following the technique in [64], we follow the bootstrapping process with a triplet loss minimization training with the explicit goal of minimizing the Euclidean distance between embeddings of matching patches and maximizing that between non-matching patches.

The classification layer (W, b) appended to 'ϕ' is removed and replaced with a l_2 normalization layer and an embedding layer, which is an affine projection layer $W' \in \mathbb{R}^{L \times D}$, $L << D$. Together, they implements $e_i = W'\phi(P_i)/||\phi(P_i)||_2$. We set $L = 1024$. The embedding layer is trained to minimize the triplet loss denoted below:

$$E(W') = \sum_{(a,p,n) \in T} max\{0, \alpha - d_{an} + d_{ap}\} \tag{6.1}$$

Here, $d_{an} = ||e_a - e_n||_2$, $d_{ap} = ||e_a - e_p||_2$, and T is the set of triplets in each training mini-batch as explained below. $\alpha > 0$ is the margin parameter that describes the desired difference between average distances between matching and non-matching patches. We test with α equal to 2, 5, and 10.

Triplets The patch-pairs and the real texture patches produced using the procedure described in Chapter 3 (Figure 3.4) are sourced to produce triplet of patch pairs using a batch server (see Figure 3.3); each triplet consists of an anchor patch, a positive patch and a negative patch, the anchor and the positive patch correspond and anchor and negative don't. In each batch, we create a triplet by picking a patch from a simplistic render as anchor, a corresponding patch from real image as positive, and a non-corresponding 'geometrically meaningful' real patch or a texture real patch as negative.

From ∼60 manually registered images, we produced 19,000 patch pairs. Additionally, we also produced 10's of thousands of texture patches from these and other real inspection images. Thus, we have 19,000 corresponding patches while many more non-corresponding ones. We can, however, produce only as many triplets as number of matching patch-pairs.

Training

The feature network, which is the concatenation of deep-architecture 'ϕ' and the l_2 normalization and embedding layer W' produces for each input patch 'P' a feature vector (or embedding) $e_P \in \mathbb{R}^L$. All layers except the final embedding layer W' are initilized using weights learned during the bootstrapping process. These weights are frozen during the triplet-loss training and only the embedding layer is learned. The goal is hence to learn a discriminative affine projection (like in [64]). Note that the embedding layer has no bias. This is because a bias term would cancel out while computing triplet loss as detailed in Equation 6.1.

Each iteration of training is performed using a mini-batch of triplets, which when served by a batch-server contains 128 triplets. For creating each triplet, the batch-server picks a patch from a simplistic render as the anchor patch, a corresponding real patch as the positive, and a non-corresponding 'geometrically meaningful' real patch or a textured real patch as negative. At train time, we have many more textured real patches (in a negative reservoir) that have no corresponding patches in the simplistic render. This problem, also termed 'class imbalance' [12], was addressed by undersampling the reservoir of texture patches. Ratio between the texture-patches and non-corresponding patches from patch-pairs was set at 3:7 i.e., 3 of 10 negatives in a batch of 10 triplets were sampled from the texture-patch-reservoir. This mimics the situation at test time when FAST points are detected independently in real and synthetic images and many more texture corners are detected in the real image that have no corresponding FAST point in the expected image than those that do.

In triplet-loss based training, it is important to pick 'hard' triplets. If a network already produces embeddings such that $d_{ap} + \alpha < d_{an}$, minimizing the triplet loss for such a triplet leads to no learning. The process of finding 'hard' triplets is known as hard negative mining. We perform such mining for each batch obtained from the batch-server by passing it through the

network to obtain embeddings for each triplet in the batch and retaining only those triplets for which $d_{ap} + \alpha > d_{an}$ for training. Additionally, to make our mined hard triplets even harder, we perform anchor-positive swap as described in [6]. Concretely, for each hard triplet, if $d_{pn} < d_{an}$, anchor and positive patches are swapped. This procedure was important for us in being able to learn a discriminative embedding using the few patch-pairs, in few epochs.

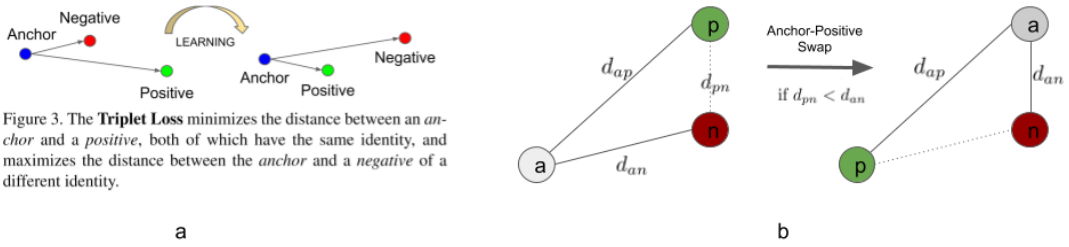


Figure 6.4: (a) Triplet loss minimization brings anchor and positive closer while increasing separation between anchor and negative. [76] (b) A hard triplet ($d_{an} < d_{ap}$) can be made harder still by swapping the anchor and the positive (in-triplet hard negative mining [6]) .

For triplet loss training, we use the Adam Optimizer with initial learning rate of 0.005, β_1 0.9, β_2 0.999 and ϵ 1e-08. Regularization was performed via dropout added before the embedding layer. The margin parameter α was emperically set at 5.0. This resulted in a better descriptor than when margin was 2.0 and 10.0, as presented in the following chapter. After training for 5 epochs, the training loss would stop decreasing. We resumed training with a learning rate reduced by a factor of 10 to reduce it further. Such resumptions did not produce much improvement. Since we want descriptors that are invariant to rotations (since inspected parts in the real image might have different rotation than what is specified in CAD), random rotations were added on the fly by the batch-sever. Scale invariance is not of concern since the renders are generated with same depth from the model as real inspection camera is from the inspection piece.

Chapter 7

Results and Evaluations

In the literature on descriptor learning, the most common benchmarks for assessing the performance of the descriptor are FPR95 rate, Mean Average Precision (MAP) for matching descriptors between image pairs using Nearest-Neighbour matching, and descriptor computation time. FPR95 rate is the false positive rate at 95% true positive rate. Additionally ROC curves are also used for characterizing the descriptor [6] [95]. ROC curves are plots of True Positive Rate vs False Positive Rate when classifying patch pairs using different threshold distances. We present ROC curves for the learned descriptor and examples of successful nearest neighbour matching (without MAP).

Since our application is inspection (presence/absence, misalignment) of industrial parts, crop-level analyses are also of interest to us. For this, we present some results using Bag of Visual Words analysis using oriented BRIEF features in ORB [75] and the learned features.

7.1 ROC Curves for Learned Descriptors

We experimented with descriptors extracted from patches of 2 different sizes : 128x128 and 224x224. The deep-architecture 'ϕ' differs between implementations for two patch sizes in the number of neurons in the two fully connected layers (i.e., the bottleneck size 'D'), and the size of the final embedding ('L', see Table 6.1).

Figure 7.2 presents the ROC curves for descriptors based on the two patch sizes for different values of the margin parameter 'α' used during the training process. We notice that the descriptor based on bigger patches is clearly better than the one based on smaller patches. The best FPR95 rate of 13.8 was obtained for descriptor based on 224x224 patches when the margin parameter was 5.0. For comparison, when matching natural patches using SIFT, FPR95

Margin α	**128x128 Patch Descriptor**	**224x224 Patch Descriptor**
2.0	40.1	17.0
5.0	41.2	**13.8**
10.0	37.8	17.5

Table 7.1: FPR95 rates observed for learned descriptors trained with patches of different sizes and different margin parameters α. The threshold distances related to the FPR95 rates are different, refer Figure 7.2

was reported to be between 26.0 and 30.0 in [6]. For obtaining the ROC curves, patch pairs from a test set with 10K patches were classified as either matching or non-matching based on whether the Euclidean distance between their descriptors was less than a range of thresholds (less than threshold meaning matching), resulting in one (true-positive, false-positive) data-point per threshold.

Figure 7.1 shows some examples of 224x224 patch-pairs classified as false-positive and false-negative using the best learned descriptor (FPR95=13.8, see Tab. 7.1). We see that the false-positive pairs have similar geometry. Some more context (through use of a bigger patch-size still) could have been helpful in rejecting them. In the false-negative pairs, we see some significant differences inside the patch pairs (such as an additional screw or wire snippet in the real image). Here also, more context could have been helpful for accepting them.

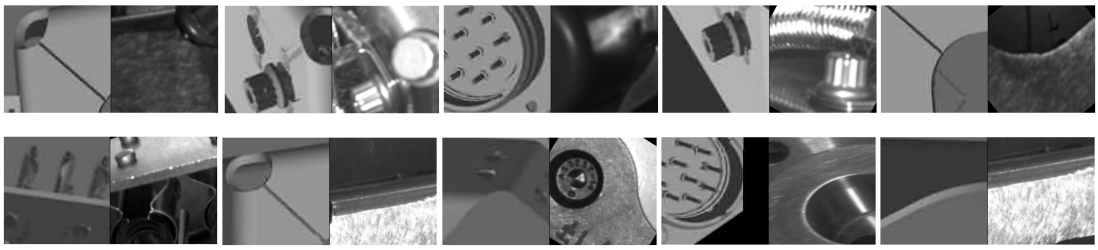


Figure 7.1: False negative (top row) and False Positive (bottom row) pairs of size 224xx224 obtained using the learned descriptor trained with margin size 5.0 and distance threshold between descriptors = 16.56 (refer Fig. 7.2)

Figure 7.3 shows some examples of successful nearest neighbor matching between renders (left) and real images (right) using the learned descriptor. Interest points were extracted using the FAST detector in ORB in OpenCV [62] avoid many detections for clear illustration. We see that the renders are of a poor quality and the local correspondences are not very precise. It is also visible that many details and small parts (like screws and cables) that are present in the real image are not present in the render. This is because they are not available in CAD.

7.2 Bag of Visual Words using Learned Features

Figure 7.4 shows comparison between ORB [75] and the learned features for comparing crops of renders with crops of real images using histograms of visual words. These derived from Bag Of Words (BoW) dictionaries [90] of learned-feature-words and ORB-words learned from a training set of real and syntheticimages.

From the training images, ORB features and learned features were extracted at FAST interest points. The extracted feature vectors were then clustered using K-means clustering with K = 50, resulting in two dictionaries each consisting of 50 words. The choice of 'K' was inspired by work of Mokhtari et. al. [61].

Similarly, from each test crop-pair consisting of a render crop and a real image crop, ORB and learned descriptors were extracted at FAST interest points, and the extracted set of descriptors converted into histograms of words using the learned dictionaries. Two 50-element histograms were thus extracted for each image, 4 histograms for a pair, and l_1 distances computed between ORB-based histograms and learned features (LF in Figure 7.4) based histograms. The distances between the ORB histograms and learned feature histograms for all 'matching' test image pairs have been shown in Figure 7.4 (e). As seen in distribution of pair-wise distances between the histograms in Figure 7.4 (a) and (b), the learned features based histograms for matching pairs differ from each other with a predictable l_1 distance while ORB based histograms for matching pairs have more erratic distances between them. Figure 7.4 (c) and (d) show two matching crop pairs. The distances between both their histograms is similar when Learned Features based BoW model is used. However, the distances between their ORB BoW based histograms is very dissimilar. We do not present a classification metric based on thresholding the distance between histograms because, as is evident in Fig. 7.4, we have very few non-matching crop pairs.

7.3 Classification with fine-tuned CNN features

We also trained SVM classifiers on CNN features extracted from the last fully connected layer of a fine-tuned VGG16 network initialized with ImageNet weights [52] (architecture _224 Bootstrapping in Table 6.1). The last 1000-way soft-max layer of the original architecture was replaced with a 2-way softmax layer and training performed as described in section 6.2.

A training dataset of 1400 positive and negative synthetic patches was created that contained or did not contain a screw respectively and combined with a dataset of 170 real patches, 90 of which contained a screw (positive). The dataset of real patches was augmented by adding random geometric distortions like shear and warping to get a total of 2000 augmented patches [45].

SVM Trained On	**Classification Accuracy on Only Real**	**Classification Accuracy on Only Synthetic**
Only synthetic	0.698	0.519
Only real	0.922	0.676
Both real and synthetic	0.9048	0.801
Only real, but half as many	0.8116	0.5420
Both real and synthetic, but half as many real	0.8831	0.7240

Table 7.2: Classification accuracies of SVM's trained on features from fine-tuned CNN.

After training, features were extracted from 1300 synthetic and 2000 real test patches (augmentated set as before). Features from 1% of the synthetic and real patches each were used for training different SVM classifiers with linear kernel and classification was performed on the remaining 99%. The results have been presented in Table 7.2. The last row shows that when an SVM is trained using half as many real patches by adding synthetic patches, the classification accuracy on real patches is comparable to when the training is performed using (all) only real patches. The improved accuracy due to additional synthetic data when was also observed in [73].

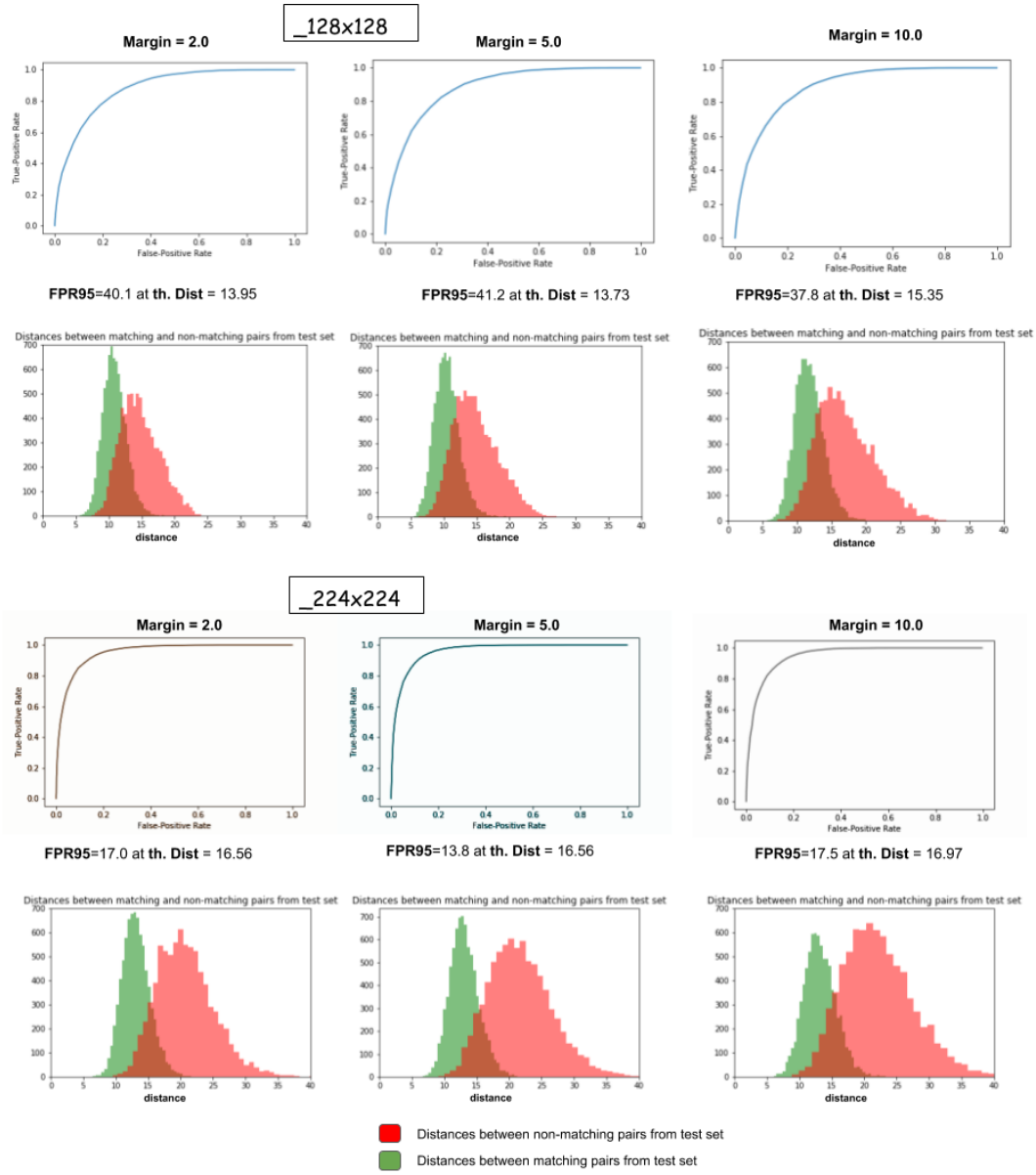


Figure 7.2: ROC curves for learned descriptor based on patches of size 128x128 and 224x224. Descriptor based on bigger patches clearly performs better. Values at which distances between patch descriptors were thresholded to get a true positive rate of 95% is denoted as 'th. Dist'. The deep architectures for the descriptor network are those mentioned in Table 6.1.

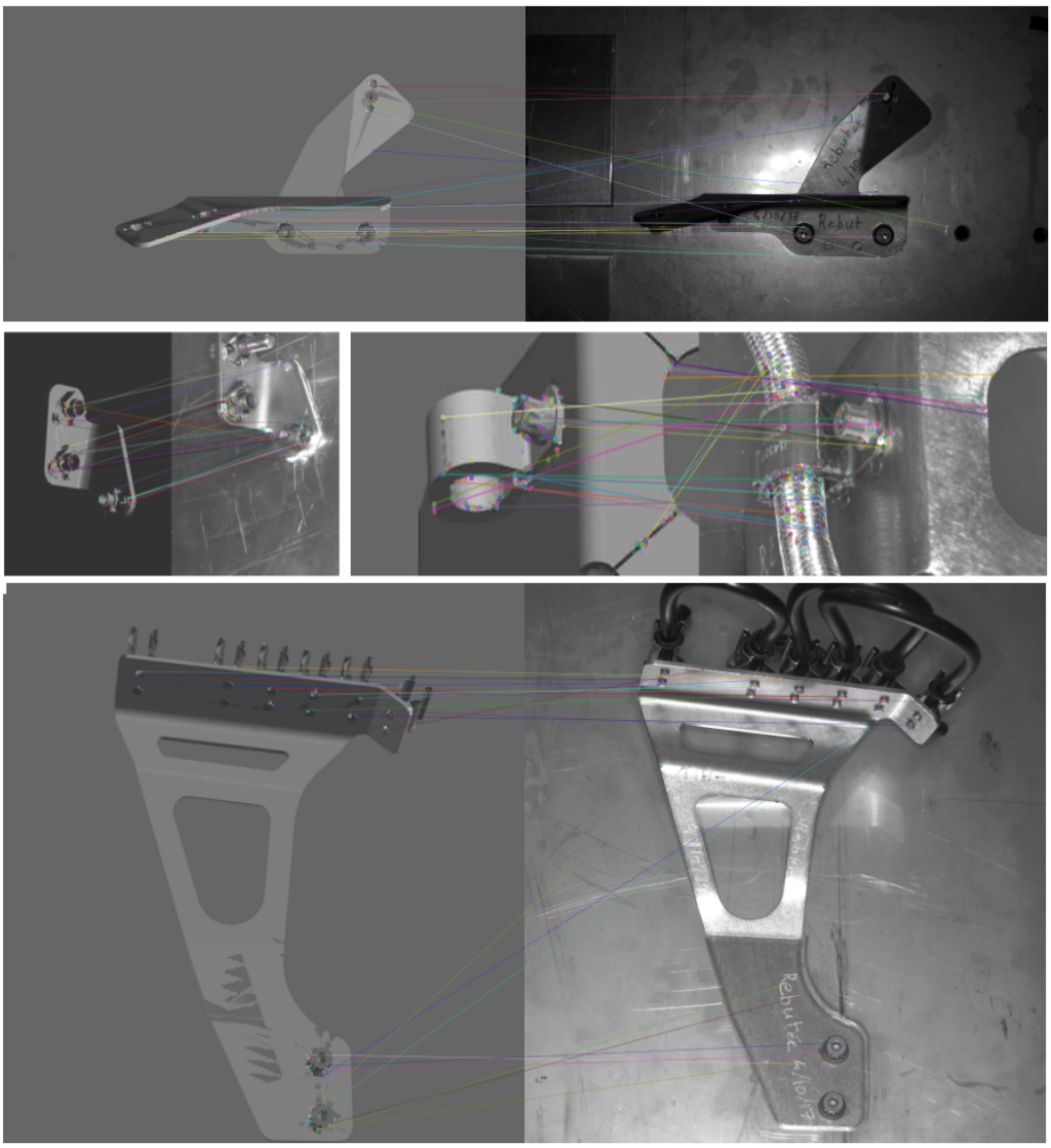


Figure 7.3: Examples of Nearest neighbour matching between renders (left) and real images (right). The interest points were detected using the ORB implementation of FAST detector in OpenCV [75]. Matches were found using Euclidean distance between the descriptors.

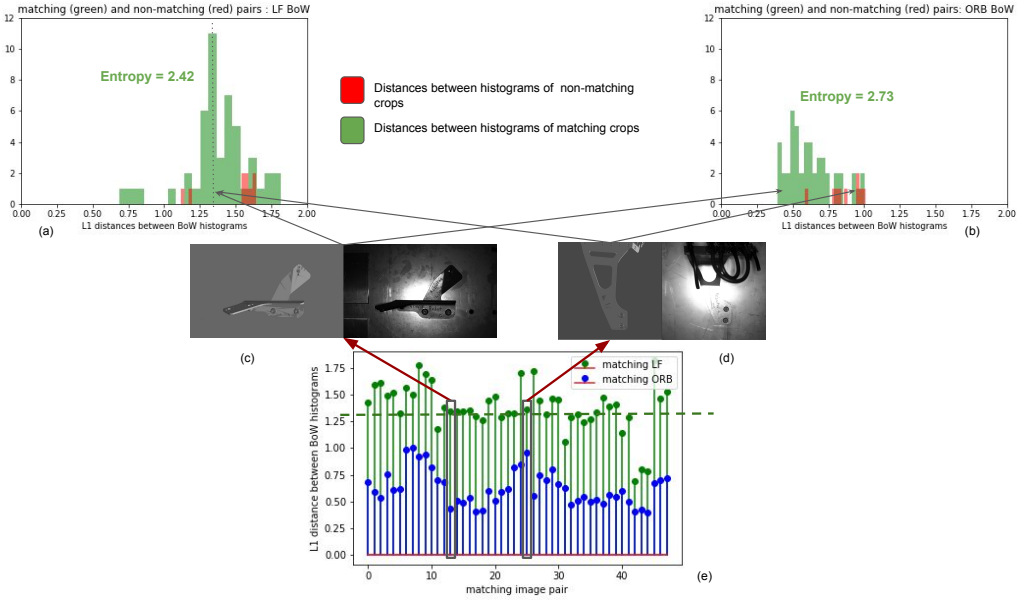


Figure 7.4: Comparison of the learned features (LF) with ORB features (rotated BRIEF) [75] using Bag of Visual Words [90]. The plot in (e) shows distances between BoW histograms of matching image pairs only computed using ORB based and learned features based dictionary.

Chapter 8

Conclusion and Future Work

We showed that it is possible to learn a descriptor that can make correspondences between simple renders of simplistic CADs with a real images with same viewpoint, and the learned descriptors might be used for nearest-neighbour matching or with Bag of Words approach to compare image pairs. Rendering CADs, producing patch-pairs in semi-supervised manner, and discriminative feature learning were the main highlights of the work.

Renders Although we used only the simple renders described in section 4.3 for generating patch pairs for learning the descriptor, using a slightly more complex and realistic shader, like Phong [10], might have produced better results.

We also talked about colored and textured renders and used patches from these renders only in section 6.2 to illustrate the effect of fine-tuning a pre-trained network, but never for descriptor learning. In Table 7.2, we saw that when the fine tuned network was used as a feature extractor for training SVM classifiers, a good classifier for real patches could be trained by using few real patches and additional synthetic ones with color and texture. PCA on features extracted from test patches using the fine tuned network also provided interesting results shown in Fig. 6.2. These hold clues for training a descriptor customized for matching renders of never before seen control elements without any corresponding real images. Concretely, we might skip the bootstrapping stage (Section 6.3.2) by starting with the fine tuned weights as mentioned before obtained from training with renders of other elements for which real inspection images are available. Such 'Fine-tuning' addresses domain-shift, which is what bootstrapping is presumably doing. The triplet-loss training might then be carried out using patches from simplistic and realistic (colored and textured) renders, which substitute for real patches. Of course, the network architecture would now need to be modified for accepting color patches, and some modifications might also be necessary to the simple shader.

Learned Descriptor In learning the descriptor, there were two main challenges :

- Lack of an existing large dataset of real-synthetic patch pairs
- Domain shift

The first problem was solved by manually registering renders with corresponding real images and automatically generating corresponding patch-pairs by cropping around FAST interest points. FAST was chosen for the sake of density of interest points to create a large dataset of patch-pairs from limited number of image pairs. However, it might be interesting to study the best detector in terms of selecting most 'matchable' locations between renders and real images, or even to learn one [94].

The problem of domain shift was addressed using 'bootstrapping' step as described in section 6.3.2. However, the effect of bootstrapping was only evaluated based on whether or not the weights derived from it led to successful triplet loss training. Some independent insights about this stage might be useful for improving it.

The deep architecture used in the descriptor was VGG16. Although the final success of this work would be to use the developed descriptors in real time along with the simple shader that was designed for speed, use of a big network like VGG16 prohibits this. On an NVIDIA GPU with 640 cores (960 M), it took 200 ms to compute the descriptors for a batch of 16 patches. It is thus important to consider smaller architectures, like the one in [6], to move toward real-time performance.

In literature on learning patch descriptors, the 2-channel 2-stream network presented in [95] has delivered the best performance till date. This was attributed to the network using multi-scale information. Feature extraction from patches using such strategy might result in more robust features. Additionally, we saw that use of larger patches resulted in more discriminative features (in terms of FPR95 score, see Table 7.1). We might see how much larger the patches might be made and what that might imply for the network depth necessary.

Matching Finally, in nearest neighbour matching (Figure 7.4), we saw that the correspondences are not very accurate and well localized. Nearest neighbour matching may not be the best matching strategy here because of the simplistic CADs. We might explore matching pairs or triplets of descriptors since correspondence between a simplistic CAD and real assembly is not one to one. The Bag of Words based analysis (Figure 7.2) showed how BoW histograms based on learned features (LF) for matching pairs were at predictable distances from each other as compared those based on ORB (BRIEF) descriptors [75]. However, because of a lack of sufficient non-matching pairs, we did not investigate classification as matching/non-matching based on the BoW histograms. More non-matching pairs have to used for making this evaluation for illustrating better the utility of the learned descriptor for comparing image pairs.

Bibliography

[1] Gerald J Agin. Computer vision systems for industrial inspection and assembly. *Computer*, (5):11–20, 1980.

[2] Samet Akçay, Mikolaj E Kundegorski, Michael Devereux, and Toby P Breckon. Transfer learning using convolutional neural networks for object classification within x-ray baggage security imagery. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 1057–1061. IEEE, 2016.

[3] Ancientcraft. Iron age technology http://www.ancientcraft.co.uk/archaeology/iron-age/ironage-tech.html, 2018.

[4] Mathieu Aubry, Daniel Maturana, Alexei A Efros, Bryan C Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3762–3769, 2014.

[5] DW Auckland, AJ McGrail, CD Smith, BR Varlow, J Zhao, and D Zhu. Application of ultrasound to the inspection of insulation. *IEE Proceedings-Science, Measurement and Technology*, 143(3):177–181, 1996.

[6] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, volume 1, page 3, 2016.

[7] Jacob Barhak. Utilizing computing power to simplify the inspection process of complex shapes. In *The 2004 Israel-Italy Bi-National Conference on Measurements and Uncertainty Evaluation in Coordinate Measuring Machine (CMM) and Scanners and Their Implication on Design and Reverse Engineering*. Citeseer, 2004.

[8] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016.

[9] Mark Billinghurst, Adrian Clark, Gun Lee, et al. A survey of augmented reality. *Foundations and Trends® in Human–Computer Interaction*, 8(2-3):73–272, 2015.

[10] James F Blinn. Models of light reflection for computer synthesized pictures. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 192–198. ACM, 1977.

[11] Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):43–57, 2011.

[12] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *arXiv preprint arXiv:1710.05381*, 2017.

[13] Young-Jin Cha, Wooram Choi, and Oral Büyüköztürk. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 32(5):361–378, 2017.

[14] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In *Proc. CVPR*, volume 2, 2017.

[15] Roland T Chin and Charles R Dyer. Model-based recognition in robot vision. *ACM Computing Surveys (CSUR)*, 18(1):67–108, 1986.

[16] Changhyun Choi and Daniela Rus. Probabilistic visual verification for robotic assembly manipulation. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5656–5663. IEEE, 2016.

[17] Claudio Cusano and Paolo Napoletano. Visual recognition of aircraft mechanical parts for smart maintenance. *Computers in Industry*, 86:26–33, 2017.

[18] Francesca De Crescenzio, Massimiliano Fantini, Franco Persiani, Luigi Di Stefano, Pietro Azzari, and Samuele Salti. Augmented reality for aircraft maintenance training and operations support. *IEEE Computer Graphics and Applications*, 31(1):96–101, 2011.

[19] César Roberto de Souza, Adrien Gaidon, Yohann Cabon, and AM López Pena. Procedural generation of videos to train deep action recognition networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[21] Dep. for Information Technology, stfold University College. Openglmaterials: http://www.it.hiof.no/ borres/j3d/explain/light/p-materials.html, 2018. [Online; accessed 18-May-2018].

[22] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. Ieee, 2010.

[23] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[24] Fabricio Fishkel, Anath Fischer, and Sigal Ar. Verification of engineering models based on bipartite graph matching for inspection applications. In *International Conference on Geometric Modeling and Processing*, pages 485–499. Springer, 2006.

[25] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. In *International Conference on Learning Representations*, 2018.

[26] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In *European Conference on Computer Vision*, pages 597–613. Springer, 2016.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[28] S Burak Gokturk, Hakan Yalcin, and Cyrus Bamji. A time-of-flight depth sensor-system description, issues and solutions. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 35–35. IEEE, 2004.

[29] Google 3D Warehouse. https://3dwarehouse.sketchup.com/?hl=en, 2018. [Online; accessed 12-May-2018].

[30] Google Research. Open images dataset, 2018. [Online; accessed 13-May-2018].

[31] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[32] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3279–3286. IEEE, 2015.

[33] Hironori Hattori, Vishnu Naresh Boddeti, Kris Kitani, and Takeo Kanade. Learning scene-specific pedestrian detectors without real data. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3819–3827. IEEE, 2015.

[34] Bernd Heisele, Gunhee Kim, and Andrew Meyer. Object recognition with 3d models. In *BMVC*, pages 1–11. Citeseer, 2009.

[35] Aaron Hertzmann. Introduction to 3d non-photorealistic rendering: Silhouettes and outlines. *Non-Photorealistic Rendering. SIGGRAPH*, 99(1), 1999.

[36] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. *arXiv preprint arXiv:1710.10710*, 2017.

[37] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*, 2014.

[38] Igor Jovančević, Stanislas Larnier, Jean-José Orteu, and Thierry Sentenac. Automated exterior inspection of an aircraft with a pan-tilt-zoom camera mounted on a mobile robot. *Journal of Electronic Imaging*, 24(6):061110, 2015.

[39] Igor Jovancevic, Ilisio Viana, Jean-José Orteu, Thierry Sentenac, and Stanislas Larnier. Matching cad model and image features for robot navigation and inspection of an aircraft. In *5th International Conference on Pattern Recognition Applications and Methods (ICPRAM'2016)*, 2016.

[40] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 2013.

[41] Yan Ke and Rahul Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.

[42] Seunghyeon Kim, Wooyoung Kim, Yung-Kyun Noh, and Frank C Park. Transfer learning for automated optical inspection. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2517–2524. IEEE, 2017.

[43] John O Klepsvik, Morten Bjarnar, PO Brosstad, DA Braend, and H Westrum. A novel laser radar system for subsea inspection and mapping. In *OCEANS'94.'Oceans Engineering for Today's Technology and Tomorrow's Preservation.'Proceedings*, volume 2, pages II–700. IEEE, 1994.

[44] Adam Kortylewski, Andreas Schneider, Thomas Gerig, Bernhard Egger, Andreas Morel-Forster, and Thomas Vetter. Training deep face recognition systems with synthetic data.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[46] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[47] Joerg Liebelt, Cordelia Schmid, and Klaus Schertler. independent object class detection using 3d feature maps. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[48] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[49] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[50] Angelique Loesch, Steve Bourgeois, Vincent Gay-Bellile, and Michel Dhome. Generic edgelet-based tracking of 3d objects in real-time. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6059–6066. IEEE, 2015.

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[52] machrisaa, Github. tensorflow-vgg: https://github.com/machrisaa/tensorflow-vgg, 2018. [Online; accessed 16-May-2018].

[53] Dimitrios Marmanis, Mihai Datcu, Thomas Esch, and Uwe Stilla. Deep learning earth observation classification using imagenet pretrained networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):105–109, 2016.

[54] David A Marx and R Curtis Graeber. Human error in aircraft maintenance. *Aviation psychology in practice*, pages 87–104, 1994.

[55] Jonathan Masci, Ueli Meier, Dan Ciresan, Jürgen Schmidhuber, and Gabriel Fricout. Steel defect classification with max-pooling convolutional neural networks. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–6. IEEE, 2012.

[56] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth. *arXiv preprint arXiv:1612.05079*, 2016.

[57] Dan McLeod, John Jacobson, Mark Hardy, and Carl Embry. Autonomous inspection using an underwater 3d lidar. In *Oceans-San Diego, 2013*, pages 1–8. IEEE, 2013.

[58] James Mure-Dubois and Heinz Hügli. Optimized scattering compensation for time-of-flight camera. In *Two-and Three-Dimensional Methods for Inspection and Metrology V*, volume 6762, page 67620H. International Society for Optics and Photonics, 2007.

[59] Timothy S Newman and Anil K Jain. A survey of automated visual inspection. *Computer vision and image understanding*, 61(2):231–262, 1995.

[60] Timothy S Newman and Anil K Jain. A system for 3d cad-based inspection using range images. *Pattern Recognition*, 28(10):1555–1574, 1995.

[61] Nour Islam Mokhtari. Msc thesis : Application of state of the art machine learning to various industrial visual inspection problems, 2018.

[62] OpenCV contributors.

[63] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014.

[64] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

[65] PCL Wiki contributors. Pcl/openni tutorial 4: 3d object recognition (descriptors), 2018. [Online; accessed 11-May-2018].

[66] Stian Aaraas Pedersen. Progressive photon mapping on gpus. Master's thesis, Institutt for datateknikk og informasjonsvitenskap, 2013.

[67] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1278–1286. IEEE, 2015.

[68] Xingchao Peng, Ben Usman, Neela Kaushik, Judy Hoffman, Dequan Wang, and Kate Saenko. Visda: The visual domain adaptation challenge. *arXiv preprint arXiv:1710.06924*, 2017.

[69] Flavio Prieto, Tanneguy Redarce, Richard Lepage, and Pierre Boulanger. Visual system for fast and automated inspection of 3d parts. *International Journal of CAD/CAM and Computer Graphics*, 13(4):211–227, 1998.

[70] Flavio Prieto, Tanneguy Redarce, Richard Lepage, and Pierre Boulanger. An automated inspection system. *The International Journal of Advanced Manufacturing Technology*, 19(12):917–925, 2002.

[71] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[72] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[73] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3234–3243, 2016.

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

[75] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[76] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[77] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[78] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 118–126. IEEE, 2015.

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

[80] Howard Slutsken. Four million parts, 30 countries: How an airbus a380 comes together https://edition.cnn.com/travel/article/airbus-a380-parts-together/index.html, 2018.

[81] Baochen Sun and Kate Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *BMVC*, volume 1, page 3, 2014.

[82] Today's Motor Vehicles. Todays inspection methods during automotive assembly, 2014. [Online; accessed 8-May-2018].

[83] Unity 3D. Uunwrapping.generatepertriangleuv, 2018. [Online; accessed 18-May-2018].

[84] T Vamos. Industrial objects and machine parts recognition. In *Syntactic Pattern Recognition, Applications*, pages 243–267. Springer, 1977.

[85] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, 2017.

[86] David Vazquez, Antonio M Lopez, Javier Marin, Daniel Ponsa, and David Geronimo. Virtual and real world adaptation for pedestrian detection. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):797–809, 2014.

[87] Ilisio Viana, Florian Bugarin, Nicolas Cornille, and J-J Orteu. Cad-guided inspection of aeronautical mechanical parts using monocular vision. In *Twelfth International Conference on Quality Control by Artificial Vision 2015*, volume 9534, page 95340I. International Society for Optics and Photonics, 2015.

[88] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[89] Daniel Weimer, Bernd Scholz-Reiter, and Moshe Shpitalni. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Annals*, 65(1):417–420, 2016.

[90] Wikipedia contributors. Bag-of-words model in computer vision — Wikipedia, the free encyclopedia, 2017. [Online; accessed 2-June-2018].

[91] Wikipedia contributors. 200911 toyota vehicle recalls — Wikipedia, the free encyclopedia, 2018. [Online; accessed 8-May-2018].

[92] Wikipedia contributors. Principal component analysis — Wikipedia, the free encyclopedia, 2018. [Online; accessed 2-June-2018].

[93] Wikipedia contributors. Uv mapping — Wikipedia, the free encyclopedia, 2018. [Online; accessed 18-May-2018].

[94] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision*, pages 467–483. Springer, 2016.

[95] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4353–4361. IEEE, 2015.

[96] Fiona Zhao, Xun Xu, and Shen Quan Xie. Computer-aided inspection planningthe state of the art. *Computers in Industry*, 60(7):453–466, 2009.

[97] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11):977–1000, 2003.