Learning deep domain-agnostic features from synthetic renders for industrial visual inspection

Abdelrahman Abubakr, Igor Jovančević, Nour Islam Mokhtari, Hamdi Ben Abdallah, Jean-José Orteu

Learning deep domain-agnostic features from synthetic renders for industrial visual inspection

Abdelrahman G. Abubakr,[a] Igor Jovančević,[a,b,*] Nour Islam Mokhtari,[a,c] Hamdi Ben Abdallah,[c] and Jean-José Orteu[c]

[a]Diota, Labège, France

[b]University of Montenegro, Faculty of Natural Sciences and Mathematics, Podgorica, Montenegro

[c]Université de Toulouse, Institut Clément Ader, CNRS, IMT Mines Albi, INSA, UPS, ISAE, Albi, France

Abstract. Deep learning has resulted in a huge advancement in computer vision. However, deep models require an enormous amount of manually annotated data, which is a laborious and time-consuming task. Large amounts of images demand the availability of target objects for acquisition. This is a kind of luxury we usually do not have in the context of automatic inspection of complex mechanical assemblies, such as in the aircraft industry. We focus on using deep convolutional neural networks (CNN) for automatic industrial inspection of mechanical assemblies, where training images are limited and hard to collect. Computer-aided design model (CAD) is a standard way to describe mechanical assemblies; for each assembly part we have a three-dimensional CAD model with the real dimensions and geometrical properties. Therefore, rendering of CAD models to generate synthetic training data is an attractive approach that comes with perfect annotations. Our ultimate goal is to obtain a deep CNN model trained on synthetic renders and deployed to recognize the presence of target objects in never-before-seen real images collected by commercial RGB cameras. Different approaches are adopted to close the domain gap between synthetic and real images. First, the domain randomization technique is applied to generate synthetic data for training. Second, domain invariant features are utilized while training, allowing to use the trained model directly in the target domain. Finally, we propose a way to learn better representative features using augmented autoencoders, getting performance close to our baseline models trained with real images.

1 Introduction

Industrial inspection and quality control are major tasks in modern industries. With more complex mechanical systems being developed, automation of the inspection process becomes crucial. It helps to increase production speed and decrease human error rates. With the huge advancement in computer vision algorithms in recent years, industrial inspection is one of the important fields to apply its state-of-the-art methods on, the objective being to automate tiresome quality control operations.

Our work is addressing various problems of automating the process of visual industrial inspection. We develop algorithms that receive two-dimensional (2D) images and provide a diagnostic on the state of mechanical assemblies. Computer-aided design models (CAD) is a standard way to describe mechanical assemblies in industrial systems. For each mechanical assembly part, there is a 3D CAD model with real dimensions and geometrical properties. Whenever available, we exploit the CAD models of the assemblies.

*Address all correspondence to Igor Jovančević, igorjovan@gmail.com

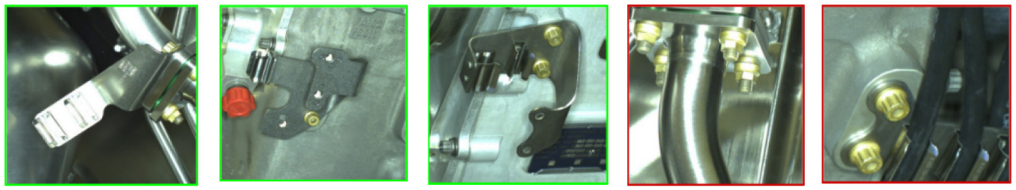

Fig. 1 (a) (c) Sample images of the three target objects used in this work, and (d), (e) the background class.

Particular objective treated in this paper is so called conformity check, i.e., verification of the presence of assembled parts at the locations predefined by the CAD model of the assembly. As a use case, we aim to verify the presence of three mechanical supports, shown in Fig. 1. More precisely, we need to either confirm the presence of a mechanical support (classes 1, 2, or 3 from Fig. 1) or report its absence (defect). Subfigures (d) and (e) in Fig. 1 show two examples of a background - absent of all three objects of interest.

We perform the control process with a robotic arm equipped with multiple high-quality 2D cameras. These cameras enable two main functionalities of our system: localization and inspection. Localization is being performed using the wide field camera. We precisely localize the effector with respect to the assembly it controls. To do that, we rely on an in-house-developed CAD-based 2D 3D alignment method. Therefore, we obtain a relative pose of our sensor with respect to the assembly being inspected (pose estimation). Figure 2 shows an example of a CAD model (left) and the robot position relative to the CAD model in a simulation environment (right). Therefore, while operating, our robotic system has information about

which mechanical support is being inspected at each moment. In other words, our system has information about the part whose presence is being verified from a particular point of view.

After the localization phase, the inspection is done with a high-resolution camera with a reduced field of view (FOV) that allows to capture the details and to observe the target elements very finely. Knowing an approximate camera pose and the camera intrinsic parameters, CAD model can be projected onto the image plane of the camera. This projection of the target object's CAD model provides an expected region of interest (ROI) around the target object. Figure 3 shows an example of RGB image captured by the 2D camera from the robot, and the corresponding synthetic image from approximately the same point of view obtained by the CAD model. Further, ROI of a target part can be observed on both images, in the form of two rectangles.

Starting from a full RGB image and using the ROI obtained by the 3D 2D CAD projection, we crop a real image containing the area where the inspection item is expected to be found. This cropped image is then fed to a classifier to output a label for one of our target objects,

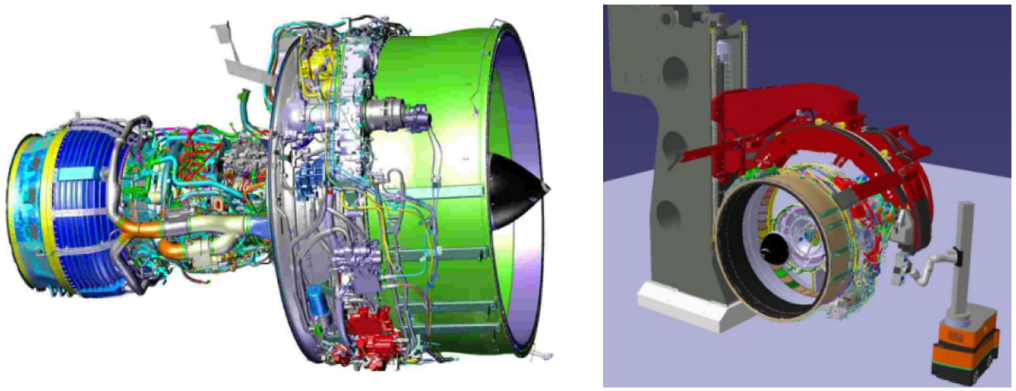

Fig. 2 (a) Example of a CAD model for a complex mechanical assembly, (b) the simulation system where the planned positions of the robot relative to the mechanical assembly are calculated.

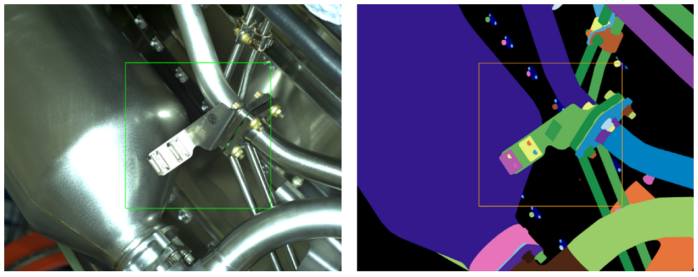

Fig. 3 (a) An example of a real image captured by the robot and (b) the corresponding synthetic image generated from the reference 3D CAD model using estimated camera pose and intrinsic parameters. ROI around the target part is indicated with the rectangles in both images.

or background which represents the fact that none of the known parts (classes) is recognized. Figure 4 shows the general pipeline of the inspection phase of our approach.

In this work, our goal is to make use of the huge advancement in the field of deep learning. Particularly, we strive to use state-of-the-art deep convolutional neural networks (CNN) to recognize the presence/absence of target objects in 2D images. The typical approach to achieve this goal would be to manually collect and annotate a lot of training data and use it to train state-of-the-art object detection or classification models (supervised learning approach).

However, in the context of industrial inspection, data collection is challenging for several reasons. First, the huge number of different objects in mechanical assemblies makes it difficult to manually collect high quality images for each part. Second, most of the parts are very specialized for certain mechanical assemblies and industries, which will not be helpful in different problems in the future, and will require collecting new data for each new case. Third, there is a class imbalance problem caused by the different number of elements in mechanical assemblies. For example, screws and clamps are parts that can be found almost everywhere in an airplane engine, while some other parts may appear only once in the engine. In addition, there is no easy and practical way to acquire large amounts of images with defects, i.e., when the object is missing (false classes). Finally, manually annotating and preparing the data for training is a laborious, time-consuming, and expensive task.

For all these problems, we need to find a way to train CNN models with a small number of collected images or to find a different approach for data collection or generation. As mentioned, we have a CAD model for each element in the assembly with the real dimensions and its position relative to all the other elements in the mechanical system. Thus, trying to use 2D renders of 3D CAD models to generate training data is an intuitive approach as it already shares many attributes

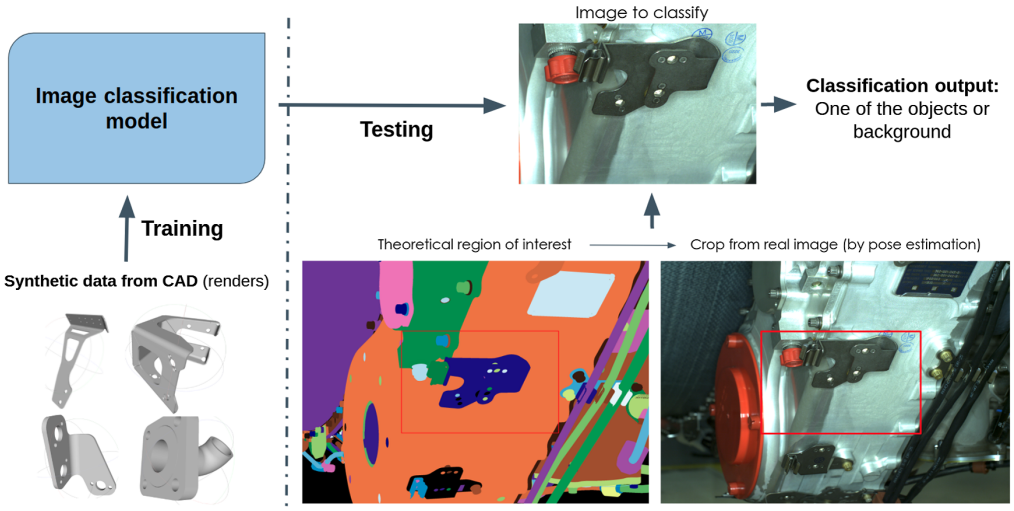

Fig. 4 Inspection pipeline based on multiclass classification.

with real objects. In addition, by rendering training images from a reference 3D model, we implicitly obtain perfect annotations for each object in the generated data.

However, using CAD models of mechanical assemblies to generate synthetic data is a challenging task. The most difficult problem is the domain gap between features learned from synthetic data and those extracted from real images. This is a nontrivial problem, and many approaches were proposed to overcome this issue by means of domain adaptation, domain randomization, domain generalization, and more.[1–6] The problem can be considered as a severe case of dataset bias; however, realistic might be the renders, there will still be a difference between renders and real photographs, which will affect the learned features and prevent the model from generalizing knowledge on real images.

Another challenge is the nature of the provided 3D CAD models. Unlike 3D models for artistic scenes, CAD models of mechanical assemblies lack many crucial visual details, such as color, texture, material properties, etc. In addition, any deformable or disposable parts such as cables or plastic caps will be missing in a CAD model. The simplistic nature of 3D CAD models leads to nonrealistic rendering, which widens the domain gap and makes it more difficult for deep learning models to generalize the features learned from synthetic data to real photographs.

In this work, two approaches are adopted simultaneously to solve the problem of domain gap between synthetic and real images. The first approach we call the "data-based approach," i.e., narrowing the domain gap by improving the synthetic data used for training. In Sec. 3, we will introduce our rendering pipeline and discuss the details of the domain randomization approach,[4,7–10] which is a key-stone in our proposed solution.

The second approach is what we call "model-based approach," i.e., to improve deep CNN models and explore different ways for training to learn domain-invariant features that generalize well between synthetic and real domains. Section 4 shows the details of training deep image classification models with synthetic data and draws some conclusions about the limitations of these models. After that, we try to dig more and understand the features learned by our classification model. This leads to our proposed approach to learn better features representations by means of self-supervision, namely, augmented autoencoder (AAE), which helped to achieve competitive results to our baseline model trained on real images. Finally, our findings and results are discussed in Sec. 5, and the conclusion is given in Sec. 6.

2 Related Work

Our research group is aiming to automate the industrial visual inspection procedures by proposing artificial vision algorithms able to perform such tasks. Our inspection algorithms are exploiting 2D images and 3D point clouds and are running on a robotic platform or on a handheld tablet. In previous works, we were dealing with industrial visual inspection challenges by employing conventional image processing[11,12] and 3D point cloud processing techniques[13,14] or recent deep learning architectures on 3D point clouds.[15,16] In this work, as well as in Ref. 17, we are focusing on using deep CNN models on 2D images (both real and synthetic). This paper is an extension of our previous paper published in Ref. 18.

Inspection based on 2D image analysis has been of interest in many works. The authors in Ref. 19 have presented a CAD-based conformity check of mechanical parts by comparing an image generated from a CAD model with an image acquired with a 2D camera, using primitives extracted from the contours. The work in Ref. 20 presented a visual detection and verification of exterior aircraft elements.

Machine learning modules that perform classification and object detection can be very useful bricks to solve various problems in vision-based industrial inspection. The authors in Ref. 21 presented a method for automatic visual inspection of dirties, scratches, burrs, and wears on surface parts. The authors in Refs. 22 and 23 have presented a new approach to detect and inspect screws on aircraft fuselage images acquired by an UAV. The work in Ref. 24 has presented a deep CNN-based method to identify and classify four types of visible surface defects on semiconductor wafers. Main challenge for deep CNN systems in industrial inspection is lack of data, because assemblies are not often available for data acquisition. To the best of our knowledge, this challenge is yet to be solved.

2.1 *Deep Domain Adaptation*

Domain adaptation can be considered as a particular case of transfer learning[25] that leverages labeled data in a source domain to learn a classifier for unseen or unlabeled data in a target domain.[1] It is assumed that the task is the same, i.e., class labels are the same in both domains. The source domain is assumed to be related to the target domain but with different data distribution, causing what is called "domain gap" or "domain shift," which significantly degrades performance at test time. Domains can be images from different cameras, images from cameras versus sketches, artistic images, and clip arts or images from visible spectrum versus near-infrared sensors.[26–31]

The authors in Refs. 2, 3, and 32 noted that the results obtained with deep convolutional activation features even without any adaptation to the target domain are significantly better than the results obtained with domain adaptation methods based on handcrafted features. This suggests that deep neural networks learn more abstract and robust representations that are general and can decrease the domain bias.[1,3,29,32,33]

The authors in Ref. 3 utilized deep features directly to train and test a classifier on different domains. Comparing the results to state-of-the-art methods using SURF handcrafted features, they proved how deep features are more general and can decrease the domain bias without explicit adaptation. The work in Ref. 30 proposed a framework for domain adaptation using a sparse and hierarchical network. It jointly learns a hierarchy of deep features together with transformations that address the mismatch between different domains. The proposed approach in Ref. 34 extracted the convolutional activations from a CNN as the tensor representation, then performed tensor-aligned invariant subspace learning to realize domain adaptation, outperforming state-of-the-art approaches based on traditional handcrafted features.

Another kind of solution is to embed domain adaptation into the training process to learn a deep feature representation that is semantically meaningful and domain invariant. One of the intuitive approaches is to combine the source and labeled target data and train the model with them. However, when only few labeled data in the target domain are available, this approach results in overfitting to the source distribution.[33] Another intuitive approach is to pretrain the deep network with source data, then fine-tune the network with labeled target data to decrease the shift between the two domains. The authors in Refs. 31, 32, and 35 experimented for best practices to fine-tune model layers with target domain data. The work in Ref. 36 proposed a two-stream network for source and target data and considered that the weights in corresponding layers are not shared but related by a weight regularizer to account for the differences between the two domains.

Autoencoders[37] found their way in the application of domain adaptation as means of learning shareable deep features.[38–40] For example, in Ref. 38 the authors proposed extracting a high-level representation based on stacked denoising autoencoders that can represent both the source and target domain data. Thus, a linear classifier that is trained on the labeled data of the source domain can make predictions on the target domain data with these representations.

2.2 *Deep Convolutional Neural Networks with Synthetic Data*

The use of synthetic data has a long history in computer vision. For example, Refs. 41 and 42 used 3D models as the primary source of information to build object recognition models. More recently, Refs. 43 to 46 used renders of 3D CAD models as a source of labeled data. Usually, they were trying to design special features for matching synthetic 3D object models to real image data or to use HOG and SIFT features and linear SVMs for classification.

Due to the data-hungry nature of deep CNNs, synthetic renders of 3D models are a very attractive source of training data in many applications, such as object recognition, detection, instance segmentation, optical flow estimation, action recognition, and more.[5,6,31,47–49] Unfortunately, synthetic rendering pipelines are usually unable to reproduce the statistics of their real world counterparts due to the "domain gap" between synthetic and real data, resulting in a poor performance as observed in Refs. 4 6 and 31, for example. Existing approaches focus either on mapping feature representations from one domain to the other or learning to extract generic

features that are invariant to the domain from which they were extracted. Our work is part of the second approach as we try to learn domain-invariant features directly from synthetic data.

One of the intuitive solutions could be to generate very realistic renders so the network is confused between rendered and real images.[47,50–52] However, this approach is very time-consuming and computationally demanding. Also, it requires full knowledge of the target domain and modeling for all the details, such as color, texture, material, lighting, etc., which are not available in many applications.[4,5] In addition, even when having such expensive realistic renders, trained models will still suffer from the domain gap.[4,6]

Therefore, many approaches were proposed to solve the problem of domain gap with non-realistic renders using a combination of model-based and data-based approaches. For example, the authors in Ref. 5 used synthetic data only to train deep object detection models for Pascal VOC classes and investigated the importance of low level cues, such as color, texture, pose, and context. In Refs. 50 and 51, the authors tried to have more photorealistic rendering and simulated the context using real images as background for their images. Similarly, Refs. 48 and 49 used photorealistic renders and tried to place target objects in the right context in real scenes.

The authors in Ref. 6 showed that it is feasible to train modern object detectors with synthetic images. Using faster-RCNN,[53] they trained the base model with ImageNet, freezed feature extraction layers to these generic layers pretrained on real images (ImageNet), and trained only the remaining layers of object detector with plain OpenGL rendering. Using this idea, the features learned from the target domain (real images) were used to train the model on the source domain (synthetic images), which helped them to get competitive results to models trained on real images. This trick is tested in our work, as will be discussed in Sec. 4.1.

2.3 *Domain Randomization*

One of the promising techniques for domain adaptation using the data-based approach is domain randomization,[4,7–10,31] which is a simple technique for training deep models on synthetic images that can transfer to real images directly. This can be achieved by randomizing rendering appearance parameters that describe the objects, such as color, texture, lighting conditions, camera positions, and more.[4,7,8] With enough variability in the rendered training data, the real world may appear to the model as just another variation of what it saw during training, and hence it can be considered as a domain generalization technique.[4]

The authors in Ref. 4 proposed the first successful application for domain randomization that was able to train a deep model with simulated synthetic data and test on a real-world situation. They trained an object localization model in a robotic simulation environment with nonphotorealistic renders and tested the trained model in a real robot with real objects. This was the first trial that closed the domain gap without using any real images during training. The work published in Ref. 10 successfully applied a similar technique in a robotic simulation environment and tested in real scenes, but they used the weights of models pretrained on real images.

In Ref. 7, the authors further demonstrated this approach by training an object detection model to recognize a set of simple geometric shapes (sphere, cylinder, cube, etc.). They proved that an object detection model can be trained using a synthetic dataset that is not photorealistic and can perform well on real images with completely different appearance. Furthermore, they proved that the model trained with synthetic data using domain randomization outperforms the model, which is fine-tuned on small domain-specific dataset.

To test the domain randomization idea on nontrivial objects in outdoor scenes with a real-world environment, the work in Ref. 8 investigated the problem of car detection. They trained an object detection model on synthetic data and tested it on KITTI dataset.[54] When training only on synthetic domain-randomized data, they achieved competitive results on the real world task, but it could not be better than training on real images (from KITTI training data). However, fine-tuning the model trained on synthetic data with real images yields better results than training on real KITTI data alone. In this work, they used random textures from real images and added some random false objects that they called "flying distractors." The authors in Ref. 31 applied the same ideas for people detection and human pose estimation.

The work in Ref. 9 extended the idea of domain randomization and introduced structured domain randomization for object detection. It is simply a variant of domain randomization

that takes into account the context of the scene, unlike normal domain randomization that randomizes everything including the position and scale of the object. Structured domain randomization places objects and distractors randomly according to probability distributions that arise from the specific problem at hand. Therefore, structured domain randomization enables the neural network to take the context around an object into consideration during detection.[9]

3 Synthetic Data Generation

As discussed in Sec. 1, we are adopting two approaches for bridging the reality gap between synthetic and real domains, namely, (1) model-based approaches and (2) data-based approaches. This section discusses the data-based approach by explaining the techniques used to generate the synthetic training dataset.

3.1 *Rendering Pipeline*

We adopt the method of domain randomization,[4,7–10,31] which is a simple technique for training deep models on synthetic images that can transfer to real images directly. Domain randomization is our choice due to its simplicity, generality, and suitability for our problem as we do not have texture or material information in the CAD models used. In addition, domain randomization achieved state-of-the-art results for object localization and detection when training the models completely with synthetic data without freezing weights of pretrained models or further fine-tuning the trained model with real images.[4,9]

To apply the domain randomization approach, a rendering pipeline is implemented using OpenGL[55] that allows to control all variants of rendering parameters, such as color, camera position and FOV, lighting condition, object texture, and material modeling (diffuse and specular reflection parameters).

OpenGL is a cross-platform library for interfacing with programmable GPUs for the purpose of rendering real-time 3D graphics. Its use is common in games, CAD, and data visualization applications.[55,56] The whole OpenGL rendering process is out of the scope of this paper, but we will discuss the appearance parameters we control in the vertex and fragment shaders using the OpenGL Shading Language (GLSL).

3.1.1 *Randomized model parameters*

The reason why materials look the way they do in real life is often the result of very complex interactions between light and the microscopic structure the material objects are made of. It would be too complicated to simulate these interactions in a computer graphics model, therefore, we use mathematical models to approximate them instead. We use low-fidelity rendering, i.e., we are not trying to model the material of the rendered objects with many details or generate photo-realistic renders. Therefore, two simple shading models are used to generate our data, namely, Phong shading[57] and Cook-Torrance shading[58] models.

Phong shading model is an empirical model of the local illumination of points on a surface.[57] It treats reflection as consisting of three components: ambient, diffuse, and specular. The ambient component represents light that is assumed to be uniformly incident from the environment and that is reflected equally in all directions by the surface. The diffuse and specular components are associated with light from specific light sources. The diffuse component represents light that is scattered equally in all directions. The specular component represents highlights, light that is concentrated around the mirror direction.

The basis of the Cook Torrance model is a reflectance definition that relates the brightness of an object to the intensity and size of each light source that illuminates it.[58] The model predicts the directional distribution and spectral composition of the reflected light. Therefore, this algorithm can model metallic materials better than Phong and avoid what the author called “plastic appearance” of the Phong model.[58] Figure 5 shows sample output of our objects using both shading models.

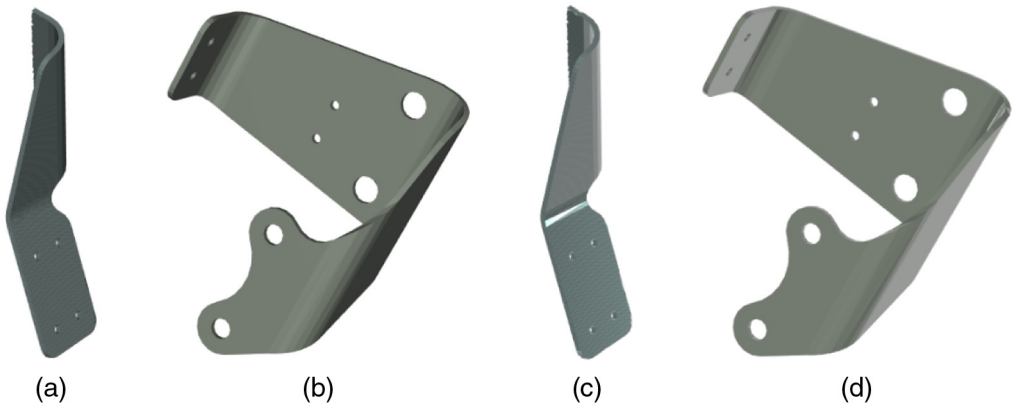

Fig. 5 Sample renders with Phong and Cook Torrance models. (a), (b) Phong model and (c), (d) Cook Torrance model. We can notice that the effect of the specular component is stronger in the Cook Torrance model.

Using these shading models, we randomize the following parameters in the rendered objects:

- **Object color:** We sample a color value for each of the RGB channels from a uniform random distribution. The values of color channels in OpenGL are scaled values (sRGB), which means the minimum is 0.0 and maximum is 1.0. We use what we call "grayish color." To obtain grayish colors, the value of one channel is randomly sampled from the range of [0.0, 1.0]. Let us call this value R for the red channel. Then, the blue and green channels values are randomly sampled so they are within a distance of 0.1 from the chosen R value, i.e., $B = R + rand(-0.1, 0.1)$, where $rand(-0.1, 0.1)$ uniformly samples a value between -0.1 and 0.1. The reason for choosing the colors to be grayish is that our objects are all metallic objects, and grayish colors represent metallic materials.[59,60]
- **Specular reflection coefficient:** This can be described as the ratio of light reflecting from the surface. For example, metals reflect a higher ratio of incident light than wood, that is why metals are shiny. Uniform random value is chosen in the range of [0.1, 0.5] for Cook Torrance model and range of [0.5, 1.0] for Phong model.
- **Camera FOV:** This value represents the zoom in the camera model. In our application, a uniform random value is chosen from the interval [20, 35], knowing that higher FOV value means the object is further (smaller).
- **Shininess of the surface:** The higher the shininess value of an object, the more it properly reflects the light instead of scattering it all around, and thus the smaller the highlight becomes. Uniform random value is chosen from the range of [2.0, 16.0].
- **Roughness of the surface:** This is used in Cook Torrance model only and sampled from a range of [0.2, 0.8].
- **Fresnel complex coefficients:** Light reflecting off metallic surfaces is described by the Fresnel equations,[61,62] which are controlled by the complex index of refraction $\eta = n + ik$. In the Cook Torrance model, we randomly simulated metallic and nonmetallic behaviors by controlling the values of n and k. n is randomly sampled from the range [0.5, 2.0], and k from the range [0.5, 7.0].

3.1.2 *Procedural texturing*

Textures are a central part in rendering, they can provide a greater level of detail to surfaces. In computer graphics, textures are 2D images, and the process of texture mapping (or UV mapping) is to wrap the 2D image to the surface of the 3D model. In our application, no such texture information is available; hence, we use the approach of generating random procedural textures.

Procedural textures take an entirely different approach than UV mapping. Instead of creating an image by defining a large, unchanging block of pixels, procedural texturing creates the texture from the ground up.[63] This is where the term "procedural" comes from. The texture is defined

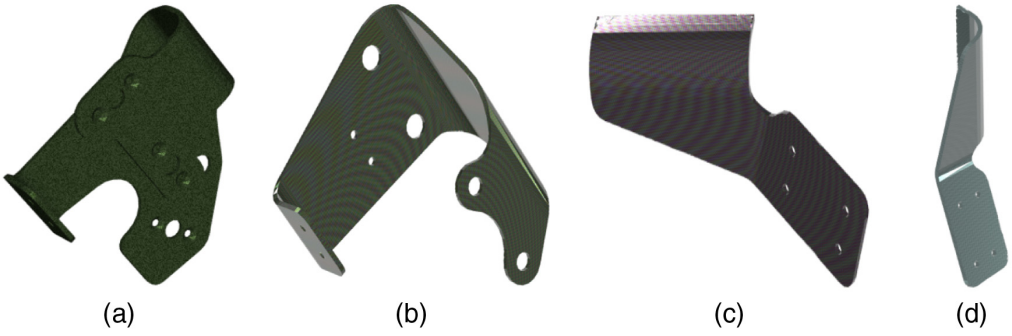

Fig. 6 (a) (d) Samples of rendered objects after applying domain randomization.

only by the mathematical formula (procedure) needed to create it. With this formula, the algorithm is able to create the texture at any scale, in any orientation, and extending as far as needed. The advantage of this approach is low storage cost, unlimited texture resolution, and easy texture mapping.[63,64] In our application, the procedural function will take a 3D coordinate and give a color back.[63] For each object, we randomly select one of the following textures:

1. 3D Perlin noise (Simplex).[65,66]
2. Random uniform color disturbance (like salt and pepper noise).
3. No noise (just color, which is by itself randomly selected as mentioned before).

Perlin noise is a procedural texture primitive, a type of gradient noise used by visual effects to increase the appearance of realism in computer graphics. It is often used in computer graphics to make computer-generated visual elements (such as object surfaces, fire, smoke, or clouds) appear more natural, by imitating the controlled random appearance of textures in nature.[65–67] The texture of the two middle objects in Fig. 6 is examples of Perlin noise.

In our application, we use the 3D Perlin noise function, which takes the 3D coordinate of the vertex and returns a color from the procedure. The Perlin noise implementation by Stefan Gustavson[68] was used in this work, as it is written in C, and can be used directly in our GLSL shader. Inside Perlin functions, scale and orientations are also randomized, and there is a possibility to have intersection between different scales of noise, so we have a bigger variety of random textures.

3.1.3 *Visibility sphere to sample camera views*

Final parameter controlled during rendering is the virtual camera position. In our application, we simulate the camera movement while assuming the object position is fixed. We aim to perform robotic-based inspection, where the robot camera pose is known relative to the reference CAD model, and hence, relative to the real mechanical assembly. To choose the camera positions during rendering, we simulate the robot camera pose by sampling the camera positions from a sphere around the target object that we want to render (Fig. 7).

By sampling the spherical coordinates parameters in a certain range and with reasonable step size, we obtain camera poses that simulate all possible positions of the robot camera. This approach helps to obtain objects' viewpoints distributions similar to real images. For example, it is not possible for the robot to look at some objects from a certain point of view, because there are other objects that occlude the target object when observed from these views.

In our application, we fix the radius of the sphere to 60 cm, which is the maintained distance of the robot to the real assembly, and sample the azimuthal and polar angles with certain step size.[69] The range of the angles differs between objects as their default orientations in CAD model are different, so these ranges are set manually for each target object. Figure 7 shows a visualization of the sampled sphere around the CAD model of one sample mechanical assembly. Note that we are randomizing the FOV values while rendering, so even when using constant radius of the sphere, objects scales are still randomized in the generated renders.

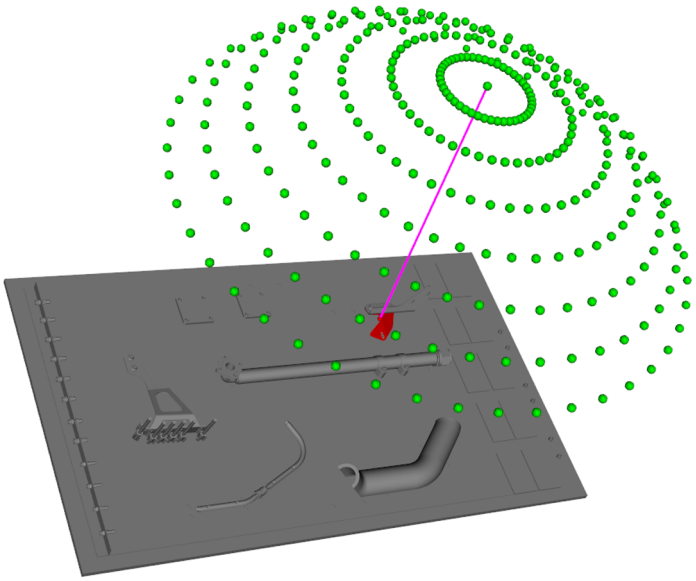

Fig. 7 Example of visibility sphere to sample camera viewpoints during rendering. Green dots are sampled camera locations, and the red object is the target object to be rendered. While rendering, the CAD of the target object is loaded and the context (here in gray) is not loaded for rendering.

3.2 *Synthetic Dataset Generation*

After generating the renders of target objects (Fig. 6), we create a dataset to be used in the training of our classification models. Following, Refs. 6 and 7 we are randomizing the background of rendered objects by placing our renders on top of randomly selected real images that are related to some industrial environments. In addition, we tried using the idea of "flying distractors" introduced in Ref. 7. Namely, we randomly position some of the rendered "context objects" around the rendered target objects. We will discuss the effect of adding these context background renders in Sec. 5. In addition to the classes of target objects, we also generate images for a "background" class.

The steps to generate our synthetic dataset are shown in Fig. 8, and examples of the generated synthetic dataset are shown in Fig. 9. Having rendered target objects, and real images as background, we form our dataset for classification as follows:

1. Randomly rotate the background image, then crop a patch in a randomly selected position, with random box size. Then, the cropped patch is resized to 1296×1024.
2. Randomly place rendered context objects (flying distractors) in the background image.
3. From the background image, randomly crop four patches (from random locations, and with random scales and aspect ratios). Resize the cropped patches to 500×500.
4. Randomly place the renders of target objects on the patches. We add one class to each crop, in addition to one crop without target objects representing the "background" class (Fig. 8).

In general, we can generate enormous amount of synthetic images for each object from our rendering pipelines. However, we have limited time and hardware resources to process the data and train our models. Therefore, we limited the number of our training images for each object to the number of real images we have for the real background images. In our experiments, we used 13,680 real background images, each of which was used to generate four patches (three classes, and background). Therefore, we have a total of 54,720 images for training with perfectly balanced number of images for each class. In our experiments, we call this set of images 55K. For more experiments, we randomly sampled half the images from our 55K set, and we call it 27K set. The 27K set has 27,360 images with balanced number of images for each class. Finally, to have an even smaller set, we randomly sampled 15,000 images from 55K set, and we call

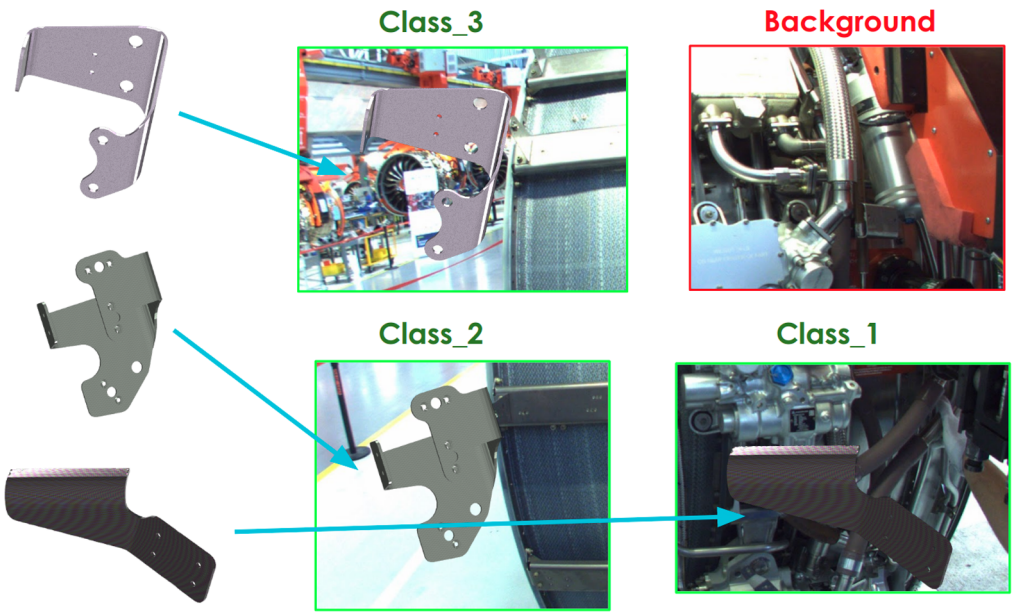

Fig. 8 Synthetic dataset generation process.

Fig. 9 Samples of the synthetic dataset used for training: (a) (c) rendered target objects, and (d) an example of the "background class" with no target objects.

this set 15K, which also has a balanced number of images for each class. In all our experiments, we always keep randomly sampled 1000 images as validation set and use the rest for training.

4 Searching for Domain Invariant Features for Domain Generalization

As discussed in Sec. 1, just training the deep models with synthetic data will cause a "domain gap" between source domain (synthetic images) and target domain (real images). One of the solutions to tackle this problem is using the domain randomization technique for data generation as explained in Sec. 2.3. In this section, we try to close the domain gap using the model-based approach, i.e., searching for techniques to train our image classification model to learn domain invariant features that can generalize well between the real and synthetic domains.

4.1 *Utilizing Target Domain Features*

The authors in Ref. 6 showed that it is possible to effectively train modern object detectors with synthetic images only. Using faster-RCNN, they trained the base model (feature extractor) with ImageNet, then froze all feature extraction layers to these generic layers pretrained on real images, and trained only the remaining layers of object detector with plain OpenGL renderings.[6] Using this trick, the features learned during pretraining from target domain (real images) were used as an initialization for training the model on source domain (synthetic images),

so the detection of objects in target domain is easier as now the features used are from the same domain. The mentioned authors obtained results competitive to the models trained on real images.[6]

In our approach, we applied this idea on image classification and evaluated its effectiveness for our problem. However, we found that this approach could not solve the domain gap problem for our classification model as will be discussed in Sec. 5.3.1. The difference between our work and the work in Ref. 6 is that their CAD models contain the colors of the real objects, and their objects have different shapes and colors, so they are easier to discriminate than our supports which have the same metallic texture and similar geometry. Nevertheless, we did use the idea of domain randomization, which helped to overcome the problem of lack of textures and colours.

4.2 *What Deep Classification Models Really Learn*

To find better features as initialization for our model, we need to first understand the features learned when pretraining with ImageNet and why such features are not good to represent our objects. As deep CNNs are learning their weights directly from the huge amount of data seen during training, many researchers tried to understand what these learned features are, through different layers. The work in Ref. 70 suggests that along different layers, networks seek to identify increasingly larger patterns in input image. For example, having a car image, the first layer will learn simple edges and contours, then deeper layers learn more complex shapes such as a car wheel until the object can be recognized by the final layer.[70]

Recent works tried to investigate this assumption, especially the work in Ref. 71, which tried to answer an important question: how do neural networks classify images: based on shape or texture? The authors came up with an interesting experiment. Using style transfer,[72] they generated images with different textures than the object in the image and tested the performance of several deep classification models. As an example, when tested on an image of texture of an elephant and on an image of a cat, state-of-the-art image classification models are able to correctly classify the texture image as "elephant" and the image with the cat as "cat." However, when using the texture of an elephant projected on a cat, all state-of-the-art classifiers recognize the image as "elephant," which clearly shows the bias of deep models towards the texture and not toward the shape of objects.[71]

To overcome this issue, the authors proposed to generate training data with random textures for all objects using the style transfer methods. With this idea, they could overcome the issue of classifying the objects depending on the texture, in addition to improving the overall accuracy of classification when testing with original testing images.[71]

The work in Ref. 73 proved the findings from Ref. 71 by introducing a deep network that has the same idea as bag-of-features and comparing it to state-of-the-art image classification models. They found that the network that models a bag of local deep features worked well and had comparable results to state-of-the-art classification models trained end-to-end with ImageNet. This proved the idea that deep models are not necessarily looking at the global shape of the object and using local patches are enough for classification.[73] They further proved this by testing on a set of scrambled images that are difficult for humans to recognize and found that state-of-the-art deep neural networks can still recognize scrambled images very well, which further proves the assumption that deep models do not actually recognize objects by their shapes.[73]

4.3 *Augmented Autoencoders*

Autoencoders are a family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, then reconstructing the output from this representation. Therefore, it can be considered as a dimensionality reduction technique for high dimensional data.[74] Convolutional autoencoder consists of an encoder and a decoder, both are CNNs. The loss function is a sum of the pixelwise L2 distance. The goal of using such architecture is to learn the latent representation, which is considered as the most important feature set that the model can use to reconstruct the image.[74]

Denoising autoencoder[75] defines a modified training process. Artificial random noise is applied to the input images while the reconstruction target image stays clean. The main assumption is that denoising autoencoder produces latent representations which are invariant to noise because it facilitates the reconstruction of clean images.[76]

Using this idea, the authors in Ref. 76 introduced the concept of AAE. Their goal was to learn 3D orientation of textureless objects for 6D object detection from RGB images. Using synthetic rendering and the idea of domain randomization, they generated a high variability of source images with extensive augmentation, while keeping the clean image to be reconstructed (clean target image). The idea behind this approach is to control what the latent representation encodes and which information to be ignored.[76] Their goal was to force the model to learn geometrical transformation for the rendered objects, then apply it to real images to estimate the orientation of objects in real images.

4.4 *Learning Better Features Representation by AAE*

Inspired by the approach of AAE,[76] we try to adapt this idea to our use case of learning geometrical representation of the mechanical parts. Ideally, we hope that latent features learned by the autoencoders can model the important geometrical features of the objects. Further, when training our classifier, these features will be discriminative enough so the classifier can recognize objects in real images depending on their shape and ignore all uninformative cues such as texture, color, and background.

Figure 10 shows our training strategy for the AAE. The input images are similar to what was used in classification, a rendered object on top of random real background. Output images contain only the object centered in the image with black background.

Figure 11 shows some examples of the input images (first row) and output images (second row). We note that both input and output images have the same size, and the object can have random location and scale in input images, but objects in target images are centered in the middle of the images with the maximum possible scale, and without any background (black background). Figure 12 shows sample input and output images after training our AAE for 15 epochs.

The goal of using AAE is to learn better features representation for our objects. After learning such features, we can use the weights of the encoder network as pretrained weights better than the weights learned from ImageNet. To do so, we remove the decoder network and replace it with a classification layer (softmax layer), then train this model for our classification task. Figure 13 shows this approach, and the results of this method will be discussed in Sec. 5.4.

The main goal of our approach is to find a domain agnostic features that can be learned from the synthetic dataset while training, to help our classification model discriminate between our mechanical supports in real images. As mentioned in Sec. 4.1, the deep classification models pretrained with ImageNet can be used as a generic feature extractors which belong to the target domain (real images), which can close the domain gap between synthetic and real domains. However, after testing this approach, we found that these generic ImageNet features are not helpful to differentiate between our mechanical objects, and there is still a big domain gap (see Sec. 5.3.1). After investigating what deep classification models really learn while training, we discovered that all state-of-the-art classification models are looking for the features that make their task easier. They are focusing on the texture of objects as the best way to discriminate between different classes in ImageNet dataset.

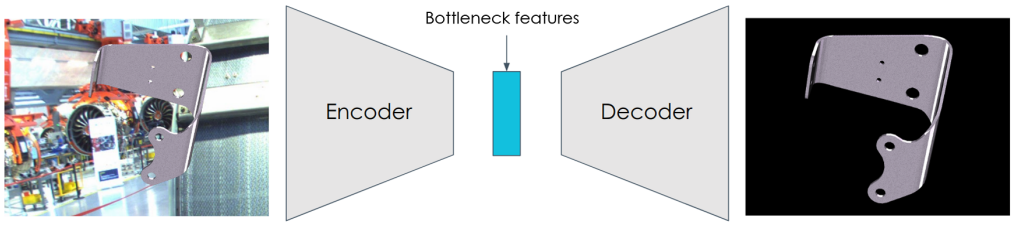

Fig. 10 AAE to learn better geometrical features representation.

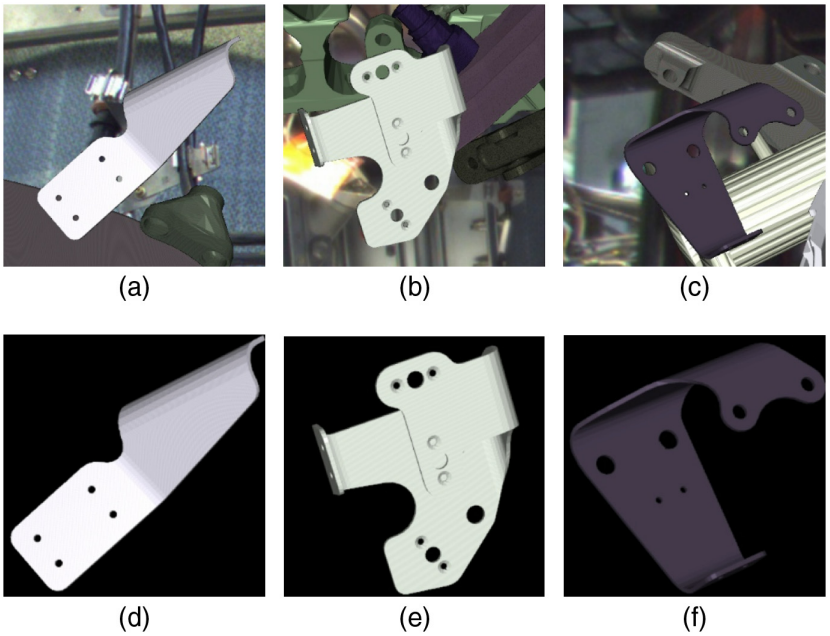

Fig. 11 Examples of data used to train our AAE: (a) (c) input images and (d) (f) the corresponding output images to be reconstructed by the autoencoder.

Fig. 12 Examples of reconstructed images using the trained AAE. Input images (top row) are from the validation set, and bottom row shows the corresponding output images generated by the AAE.

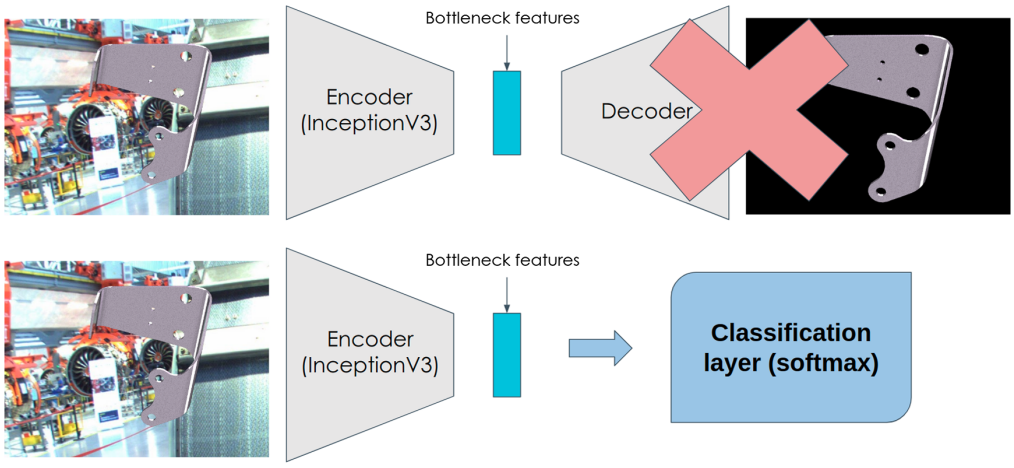

Fig. 13 AAE is trained, then the decoder is replaced by a classification layer, and the weights of the encoder network are used as initialization for training image classification models.

Learning the color and texture features to discriminate between objects can be good for certain applications; however, for our application these are the worst features to learn as our synthetic renders are missing these cues. Therefore, we introduced the AAE idea as our approach to force the model to focus on more helpful features to discriminate between mechanical objects. The task and the input and output data fed to the AAE (Figs. 10 and 11) are designed to force the AAE to focus on the most discriminative features in our mechanical supports: geometrical features. Our AAE is learning good features while trying to perform its task: reconstruct an output image containing the object alone, with no background. In addition, the model is forced to ignore any features related to the background, the color and the texture of the objects as all these cues are rendered with highly randomized variations. Namely, there is almost no two samples with the same color or texture or background. Sample output from our AAE model shown in Fig. 12 demonstrates that our strategy is working as expected. Namely, it can be noted that the objects in the output images (second row) do not preserve the color or texture of input objects (first row), but the model rather tries to reconstruct the shape of the objects in white and gray colors.

5 Results and Discussion

In this section, we explain all our experiments using the data-based and model-based approaches and discuss the results of each experiment. We start by explaining our experimental setup, then perform a series of experiments to test some assumptions. For each experiment, we build on the conclusion from its results and use the best setup in the following experiments.

5.1 *Experimental Setup*

5.1.1 *Real images for testing*

All the tests are done on the same dataset made entirely of real images. In this section, we describe this dataset. We collected a set of 179 RGB images, each with an ROI around the target object. ROIs are obtained from the reference CAD as explained in Sec. 1. To generate images that contain our objects for classification, the ROI boxes were used to crop parts of the images that contain the target objects. To increase the number of samples, in addition to the original crop of the object, four transformations are applied: horizontal and vertical flip, rotation 90 deg clockwise and counterclockwise.

In addition, to make a testing set for the "background" class, a set of seven boxes with random positions and sizes are cropped from each image for each object in the image. The positions of "background" boxes are selected so they have no overlap with any of the target objects.

Finally, all crops are resized to 500×500. The total number of real images for testing is 2148. Figure 1 shows samples of the generated test set.

5.1.2 *Training and testing setup*

For the classification model, InceptionV3[77] is used as the base model for all the experiments. The reason for choosing InceptionV3 is that it is one of the state-of-the-art models for image classification, in addition to its reasonable size, allowing for fast training with different configurations. All experiments were done using Keras[78] with Tensorflow backend.

Training is done using Nvidia GeForce GTX 1060 with 6 GB of RAM. Unless otherwise stated, batch size of 32 images is used when freezing feature extractor layers, and batch of 20 images when training all layers. Input image size is set to 256×256. SGD optimizer was used with learning rate of 0.0001 and momentum of 0.9 for all experiments, and training run for maximum three epochs. For the learning rate value, we experimented with few different values and picked those which gave the best behavior of our models. We used a small learning rate for the classifier as our batch size is relatively small. Another reason is preserving the old features from the pretrained model while fine-tuning.

For the AAE, we used InceptionV3 as the encoder, while for the decoder we used a simple network consisting of successive convolutional and upsampling layers that can reconstruct the image with the same input size.

Training of the AAE is done using Adam optimizer with learning rate 0.0003, and the training runs for 15 epochs. For AAE, the learning rate and number of epochs are higher than those for the classifier, as the AAE is trained from scratch. Input and output image size is 256×256. While training, further image augmentation is done for source images, which includes randomly changing the brightness, and adding random colored patches on top of the image to introduce random occlusion.

To evaluate the classifier quality, we use precision, recall, and $F1$-score metrics.[79] All evaluation results are obtained using the Scikit-Learn Python library.[80] To get the average results between all classes, results reported in this paper are using "macro" averaging between all classes.[81] Macroaveraging calculates metrics for each label and finds their unweighted mean, which treats all classes equally. For comparison between different methods, we consider $F1$-score, i.e., when stating that method X is better than method Y, we mean that $F1$-score of X is higher than Y.

5.2 *Training with Real Images as Baseline*

To get baseline results, we first train the model with real images. This will be our reference, and all models trained with synthetic data will be compared to it. The training is done on crops of real images as discussed in Sec. 5.1.1. For training and testing, the data are split to two parts (50% 50% split), training is done with first split and testing with the other and vice versa, then the results from the two splits are averaged to get the final baseline results (two-fold cross-validation).

The InceptionV3 model is pretrained with ImageNet. We test two configurations, one is training all layers of the InceptionV3 network (initialized with ImageNet pretrained weights), and the other is freezing all convolution layers of the network, i.e., utilizing the ImageNet pretrained weights, and just letting the final fully connected layers to train. The evaluation metrics used are described before (precision/recall/$F1$-score), and they will be shown in this order in all our tests.

Table 1 shows the results when training with real images. It is evident from the table that freezing the ImageNet pretrained layers is not helpful relative to training all the layers, especially in "class 1" where we obtain 0.0% when freezing all layers. These results are expected, as the shape of our mechanical supports are very different from the images in ImageNet dataset, so the deep features learned from ImageNet are far from the features needed for our classes. In addition, training and testing splits are from the same domain (real images domain). Therefore, allowing the model to freely train all its layers gave better results. It is also important to note that "class 1" got a low $F1$-score even when training all layers, which indicates the difficulty of this class. As training all layers with real images produced the best results (79% $F1$-score), we consider this model as our baseline and compare all methods to it.

5.3 *Utilizing ImageNet Pretrained Features*

In this section, we discuss our experiments to test the effectiveness of different configurations to apply standard image classification by training with synthetic data. Following Ref. 6, we test the effect of freezing different layers of our model and compare it to training all layers. Then, we test the effect of changing some parameters in the synthetic data used for training to find the best configurations for domain randomization.

Table 1 Results of training InceptionV3 with real images (two splits).

Training data	Train/freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
Real (two splits)	Train all	0.9/0.77/**0.79**	0.87/0.99/0.92	0.87/0.46/0.53	0.99/0.74/0.82	0.86/0.90/0.88
Real (two splits)	Freeze all	0.48/0.49/0.45	0.91/0.88/0.89	0.00/0.00/0.00	0.53/0.91/0.67	0.48/0.2/0.23

Note: The bold value represents the best F1-score (higher is better) in the experiment.

Table 2 Results of training different stages of InceptionV3. Training all layers shows the best result even with synthetic data, which is the opposite of what was found in Ref. 6 with object detection.

Training data	Train/freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
Real (two splits)	Train all	0.9/0.77/0.79	0.87/0.99/0/92	0.87/0.46/0.53	0.99/0.74/0.82	0.86/0.90/0.88
Phong 55K	Freeze all	0.42/0.39/0.28	0.99/0.19/0.31	0.11/0.27/0.15	0.43/0.33/0.38	0.16/0.75/0.26
Phong 55K	Train last block	0.45/0.39/0.29	0.98/0.10/0.18	0.07/0.33/0.11	0.57/0.63/0.60	0.19/0.48/0.27
Phong 55K	Train last two blocks	0.51/0.56/0.51	0.95/0.59/0.73	0.16/0.30/0.21	0.62/0.86/0.72	0.31/0.49/0.38
Phong 55K	Train all	0.76/0.51/**0.54**	0.76/0.98/0.86	0.89/0.32/0.47	0.75/0.67/0.71	0.66/0.08/0.15

Note: The bold value represents the best F1-score (higher is better) in the experiment.

5.3.1 *Freezing different layers of feature extractor*

Following Ref. 6, we test the effect of freezing and training different stages of the ImageNet pretrained InceptionV3 network while training with synthetic data. Inception network consists of a number of "Inception blocks."[77] Here, we test the effect of freezing all layers, freezing all layers except last Inception block, freezing all layers except last two Inception blocks, and training all layers. In all cases, the last fully connected layers are trained. We used 55K synthetic training images rendered using the Phong illumination model.

Table 2 shows the results of these tests and compares it to the baseline model. We can see that the average $F1$-score is improving when training more Inception blocks, starting from the worst when all layers are frozen, then improving a little when training the last Inception block, and improving by a large margin when the last two blocks are trained. Finally, the best result comes when all layers are trained allowing the model to adapt all its weights to the training data.

Unlike object detection models shown in Ref. 6, it seems that classification models are affected less by dataset bias when trained with synthetic data. The results are even improving as we loosen the constraints in terms of freezing layers. We think that the reasons are twofold. First, there is a big difference in the shape between our mechanical supports and the objects in ImageNet dataset, which is not the case in Ref. 6. Another reason could be the domain randomization strategy used with training data. Therefore, the classification model is learning more discriminating features from synthetic data than the generic ImageNet features, which helped to better classify the objects in the real testing set. Still, a large domain gap remains: the average $F1$-score is 54% for our best model trained on synthetic data versus 79% for the baseline model trained with real images (Table 2).

5.3.2 *Effect of using different shading models for rendering*

Here, we test the effect of using different shading models for training data. We test the data generated using Phong and Cook Torrance shading models, in addition to mixing the data from both shading models by adding them together, resulting in double the number of images (Mixed 110K set). For all tests, the classifier is initialized with ImageNet pretrained weights, and all layers are trained.

Table 3 shows the results of these experiments. We notice that the average results from Cook Torrance rendering are slightly better than Phong, but for some classes Phong is much better, especially for "class 1." However, for "class 3" the results of Cook Torrance rendering are much better than Phong. Mixing the data generated by both shaders did not improve the results, which is against the intuition that increasing the number of training data for deep learning models should lead to better results. We believe that doubling the number of images for training in mixed mode leads to overfitting to the synthetic domain, and hence to worse results on real images. This assumption will be confirmed in Sec. 5.5.

Table 3 Results of using different illumination models for rendering. Overall, Cook Torrance rendering is better, but for some classes Phong is better. Mixing the data degraded the results.

Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
Real (two splits)	Train all	0.9/0.77/0.79	0.87/0.99/0/92	0.87/0.46/0.53	0.99/0.74/0.82	0.86/0.90/0.88
Phong 55K	Train all	0.76/0.51/0.54	0.76/0.98/0.86	0.89/0.32/**0.47**	0.75/0.67/**0.71**	0.66/0.08/0.15
Cook Torrance 55K	Train all	0.78/0.51/**0.55**	0.78/0.97/0.86	0.93/0.14/0.24	0.69/0.66/0.67	0.72/0.29/**0.41**
Mixed 110K	Train all	0.79/0.49/0.50	0.79/0.97/**0.87**	0.94/0.16/0.27	0.65/0.74/0.69	0.80/0.09/0.16

Note: The bold values represent the best F1-score (higher is better) in the experiment.

5.4 *Learning Better Features by Augmented Autoencoder*

In this section, we evaluate the effect of using different configurations to train the AAE and use its weights as initialization to train the image classification model. Once we find the best configuration, we will compare the results to models pretrained with ImageNet and the baseline model.

5.4.1 *Freezing different layers of the AAE pretrained classifier*

As mentioned in Sec. 4.4, we train our AAE with synthetic images along with its corresponding object in black background image. After that, the decoder network is removed, and we use the encoder only followed by a final softmax layer for classification (Fig. 13). This new classification network (encoder plus softmax layer) is trained using our synthetic data for classification. When we mention the AAE pretrained classifier in the following experiments, we mean this combination of encoder network plus softmax classification layer.

The goal of the first experiment is analyzing the effect of using different layers of the AAE pretrained classifier model. In this experiment, the AAE is trained with source images of rendered objects using the Phong model, placed on top of real image crops with no rendered context in the background. The classifier is trained on images of the Phong model with rendered context in the background. First, we freeze all layers, including the fully connected layer, which is the latent representation learned by AAE. This means that we are using the latent features as they are without any changes and just train the final softmax layer to classify the objects depending on these features. Then, we try training the last fully connected layer along with the last Inception block and the last two Inception blocks. Finally, we train the whole network using the weights from AAE as initialization for all layers.

Table 4 shows the results of these experiments. It is very clear that using the weights of the pretrained encoder part of AAE by itself is not helpful. However, using these weights as

Table 4 Results of training/freezing different layers of InceptionV3 pretrained with AAE. Both AAE and classifier trained with renders of Phong model.

Training data	Train/freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
Phong 55K	Freeze all	0.23/0.25/0.19	0.58/1.00/0.74	0.33/0.01/0.01	0.00/0.00/0.00	0.00/0.00/0.00
Phong 55K	Train last block	0.27/0.25/0.19	0.59/1.00/0.74	0.50/0.01/0.02	0.00/0.00/0.00	0.00/0.00/0.00
Phong 55K	Train last two blocks	0.31/0.25/0.19	0.59/1.00/0.74	0.67/0.01/0.02	0.00/0.00/0.00	0.00/0.00/0.00
Phong 55K	Train all	0.65/0.38/**0.40**	0.70/0.98/0.81	0.43/0.16/0.23	0.51/0.31/0.38	0.95/0.09/0.16

Note: The bold value represents the best F1-score (higher is better) in the experiment.

initialization and allowing the classification model to train all layers generated much better results (almost double the $F1$-score). These results share the same trend that we saw in Sec. 5.3.1. Knowing this, in next experiments, we will train all layers of the AAE pretrained classifiers (using the pretrained encoder part of AAE as initialization).

5.4.2 *Training AAE pretrained classifier with different synthetic data*

In this section, we try training the AAE pretrained classifier on different synthetic data rendered by Phong and Cook Torrance shading models, in addition to the mix of both models. The AAE is trained on data from Phong model as explained in Sec. 5.4.1 and the pretrained encoder part is used as initialization for training (fine-tuning) all layers of the classifier.

Table 5 shows the results of these experiments. We can see that rendered data of Cook Torrance model got the best performance between models pretrained with AAE. The reason for getting the best results with Cook Torrance in this experiment and the experiment in Sec. 5.3.2 is that this shading model represents the metallic material and its reflections better than Phong as discussed in Sec. 3.1.1. Mixing Phong and Cook Torrance data is giving better results than using Phong model only, but still worse than Cook Torrance. These results share the same trend as observed in Sec. 5.3.2, and it will be discussed in depth in Sec. 5.5. Comparing these results to the best model pretrained with ImageNet (last row in the table), we can see that pretraining with ImageNet got better performance than any of our models. In the next section, we will try to improve the AAE features by trying different shading models for training data.

5.4.3 *Training AAE with different synthetic data*

In this section, we test the effect of training AAE with synthetic data from Phong and Cook Torrance models. The AAE pretrained classifier in this experiment is always trained on Cook Torrance data as it proved to be the best data for the classifier (Secs. 5.3.2 and 5.4.2). Table 6 shows these results, which proves that training AAE with images from the Phong model produced much better features to be used as pretrained weights.

Table 5 Effect of using different shading models in training the AAE pretrained classifier.

Pretrain	Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
AAE (Phong)	Phong 55K	Train all	0.65/0.38/ 0.40	0.70/0.98/ 0.81	0.43/0.16/ 0.23	0.51/0.31/ 0.38	0.95/0.09/ 0.16
AAE (Phong)	Cook Torrance 55K	Train all	0.59/0.50/ 0.52	0.84/0.95/ 0.89	0.17/0.06/ 0.08	0.50/0.59/ 0.54	0.86/0.40/ 0.54
AAE (Phong)	Mixed 110K	Train all	0.64/0.45/ 0.48	0.75/0.97/ 0.85	0.38/0.13/ 0.19	0.51/0.41/ 0.46	0.92/0.29/ 0.44
ImageNet	Cook Torrance 55K	Train all	0.78/0.51/ **0.55**	0.78/0.97/ 0.86	0.93/0.14/ 0.24	0.69/0.66/ 0.67	0.72/0.29/ 0.41

Note: The bold value represents the best F1-score (higher is better) in the experiment.

Table 6 Effect of training autoencoder on synthetic data with different rendering methods.

Pretrain	Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
AAE (Phong)	Cook Torrance 55K	Train all	**0.59/0.50/ 0.52**	0.84/0.95/ 0.89	0.17/0.06/ 0.08	0.50/0.59/ 0.54	0.86/0.40/ 0.54
AAE (Cook Torrance)	Cook Torrance 55K	Train all	0.43/0.41/ 0.38	0.88/0.79/ 0.83	0.00/0.00/ 0.00	0.40/0.73/ 0.51	0.45/0.10/ 0.17

Note: The bold value represents the best F1-score (higher is better) in the experiment.

Table 7 Training AAE on renders of Phong with rendered context objects in the background.

Pretrain	Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
ImageNet	Cook Torrance 55K	Train all	0.78/0.51/ 0.55	0.78/0.97/ 0.86	0.93/0.14/ 0.24	0.69/0.66/ 0.67	0.72/0.29/ 0.41
AAE (Phong)	Cook Torrance 55K	Train all	0.59/0.50/ 0.52	0.84/0.95/ 0.89	0.17/0.06/ 0.08	0.50/0.59/ 0.54	0.86/0.40/ 0.54
AAE (Phong+cntxt)	Cook Torrance 55K	Train all	0.79/0.56/ **0.61**	0.77/0.98/ 0.86	0.62/0.17/ 0.26	0.82/0.67/ 0.74	0.95/0.42/ 0.58

Note: The bold value represents the best F1-score (higher is better) in the experiment.

Knowing that training AAE with the Phong model has proven to be the best option, we tried to add random rendered context objects in the background of source images (flying distractors) then add the rendered object on top of it. This process was explained in Sec. 3. The motivation for this experiment is to test our assumption that using real images as the background may cause the AAE to ignore any features from the real images domain (background) and focus only on the rendered object. In fact, the whole idea of AAE is to ignore anything but the rendered object to be able to reconstruct it again, but when adding random rendered context in the background, the model learns to ignore background that belongs to both domains, the real and synthetic, which will hopefully improve the performance. Table 7 shows that this intuition was correct, and randomly adding context objects in the background improved the $F1$-score to 61%, which is better than pretraining on ImageNet by 6%.

5.5 *Effect of the Number of Training Images*

As we are controlling the whole process of data generation, we can theoretically generate an infinite number of synthetic images for training. However, this is not practical as the data generation process takes time for rendering, and using more data than actually needed can cause overfitting to the synthetic data which may increase the domain gap. This was noticed in Secs. 5.3.2 and 5.4.2, as the results of using 110K images by adding together data generated by both shading models, were consistently worse than using Cook Torrance data only (55K images). In this section, we prove our assumption that using too much data can cause overfitting to the synthetic domain and try to find the sweet spot for the best number of images for training our classifier.

In this section, we test this assumption on the classification models pretrained on ImageNet and AAE. We trained all models for 1 epoch, with 55K, 27K, and 15K images. The results of these experiments are shown in Table 8. From the results we can see that the models pretrained with ImageNet are not affected by the number of images. However, AAE pretrained models showed a big improvement when decreasing the number of training data. Our AAE pretrained classifier trained on 15K synthetic images achieved 70% $F1$-score, which is 9% higher than the AAE pretrained model trained on 55K images and 15% higher than ImageNet pretrained model.

Since the performance of the models pretrained with AAE outperformed the model pretrained on ImageNet, we can conclude that the weights learned by AAE are representing our mechanical objects better than the generic features learned from ImageNet. However, training with too much data (or for too many epochs) will cause the model to overfit to the synthetic domain, which increases the domain gap and hurts the overall performance. In other words, we proved that using more synthetic data will not always lead to better results, as the models can overfit to the synthetic domain.

5.6 *Overall Results*

Finally, here we summarize the best classification results from all our experiments. Table 9 shows our baseline results versus the best ImageNet pretrained model and the best AAE

Table 8 Training the classifier with different numbers of synthetic images.

Pretrain	Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
ImageNet	Cook Torrance 55K	Train all	0.78/0.51/ 0.55	0.78/0.97/ 0.86	0.93/0.14/ 0.24	0.69/0.66/ 0.67	0.72/0.29/ 0.41
ImageNet	Cook Torrance 27K	Train all	0.74/0.54/ 0.54	0.87/0.95/ 0.91	0.85/0.06/ 0.11	0.63/0.85/ 0.73	0.61/0.30/ 0.40
ImageNet	Cook Torrance 15K	Train all	0.67/0.52/ 0.55	0.81/0.94/ 0.87	0.77/0.27/ 0.40	0.62/0.69/ 0.65	0.47/0.20/ 0.28
AAE (Phong+cntxt)	Cook-Torrance 55K	Train all	0.79/0.56/ 0.61	0.77/0.98/ 0.86	0.62/0.17/ 0.26	0.82/0.67/ 0.74	0.95/0.42/ 0.58
AAE (Phong+cntxt)	Cook Torrance 27K	Train all	0.80/0.60/ 0.65	0.80/0.96/ 0.87	0.66/0.21/ 0.31	0.80/0.77/ 0.79	0.96/0.47/ 0.63
AAE (Phong+cntxt)	Cook Torrance 15K	Train all	0.79/0.66/ **0.70**	0.83/0.94/ 0.88	0.62/0.28/ 0.38	0.84/0.86/ 0.85	0.85/0.59/ 0.70

Note: The bold value represents the best F1-score (higher is better) in the experiment.

Table 9 Comparison between the baseline models trained on real images, and models trained on synthetic data with different pretraining strategies.

Pretrain	Training data	Train/ freeze	Avg results (P/R/F1)	Background	Class 1	Class 2	Class 3
ImageNet	Real (two splits)	Train all	0.9/0.77/ **0.79**	0.87/0.99/ 0/92	0.87/0.46/ 0.53	0.99/0.74/ 0.82	0.86/0.90/ 0.88
ImageNet	Cook Torrance 55K	Train all	0.78/0.51/ 0.55	0.78/0.97/ 0.86	0.93/0.14/ 0.24	0.69/0.66/ 0.67	0.72/0.29/ 0.41
AAE (Phong+cntxt)	Cook Torrance 15K	Train all	0.79/0.66/ 0.70	0.83/0.94/ 0.88	0.62/0.28/ 0.38	0.84/0.86/ **0.85**	0.85/0.59/ 0.70

Note: The bold value represents the best F1-score (higher is better) in the experiment.

pretrained model trained with synthetic images. It is clear how pretraining with AAE improved the classification results when using synthetic data only.

It is interesting to note that AAE itself is trained completely with synthetic images, then using its features as a weights initialization, the classifier is also trained on synthetic images. Our best AAE pretrained multiclass classification model achieved 70% $F1$-score which is 9% lower than the baseline model trained with real images. However, our best model trained on synthetic data outperformed the baseline model trained with real image in class "class 2" as shown in Table 9.

A final trick to further improve our results is to train binary classifiers for each support against the background class. As explained in Sec. 1, our system aims to verify the existence of each object in its expected position. Therefore, there is no reason to train a multiclass classifier, as in each run the system will search for the particular object (support) that matches the reference CAD model. Then, we can solve the problem via a binary classifier for this particular object. Therefore, the two classes for our first binary classifier would be: "class 1" versus "background." Further, the two classes for our second binary classifier would be: "class 2" versus "background," etc.

Table 10 shows the results of training the binary classification models, which improved the overall performance. Our binary classifiers pretrained with AAE and trained with synthetic images outperformed the baseline model trained with real images in two out of three classes. This is expected as now there is less confusion between similar classes.

Table 10 Results of training binary classifiers with synthetic images. The training of each classifier is done with class versus background images.

Pretrain	Training data	Train/ freeze	Class 1	Class 2	Class 3	Overall
ImageNet	Real (two splits)	Train all	0.94/0.55/ 0.56	0.95/0.87/ 0.89	0.99/0.97/ **0.98**	0.96/0.80/ 0.81
ImageNet	Cook Torrance 27K	Train all	0.76/0.62/ 0.65	0.89/0.85/ 0.87	0.82/0.72/ 0.76	0.82/0.73/ 0.76
AAE (Phong+cntxt)	Cook Torrance 27K	Train all	0.77/0.69/ **0.72**	0.90/0.92/ **0.91**	0.91/0.85/ 0.88	0.86/0.82/ **0.84**

Note: The bold value represents the best F1-score (higher is better) in the experiment.

6 Conclusion

In this work, we proposed an original solution for an automatic vision-based conformity check (presence/absence of elements) for parts in a mechanical assembly. We opted for leveraging the efficiency and robustness of deep CNN. More precisely, we train and evaluate deep CNN classification models of both multiclass and binary type. The main difficulty in training such models for industries is lack of real images. This is due to various reasons, such as unavailability of the assemblies for extensive data acquisition campaigns. To overcome this, we exploit available CAD model to generate a large number of synthetic 2D images. The ultimate goal is to train models entirely on synthetic data and to achieve results comparable to the baseline model trained on real images.

Using data from different domains for training and testing produces another problem called domain gap. To deal with the domain gap, we adopt a two-step approach: data-based approach and model-based approach. First, we propose an OpenGL rendering pipeline for 2D images from simplified 3D CAD. The pipeline is designed to randomize appearance features of rendered objects, to decrease the domain gap. Those features are color, textures, material properties, background, in addition to orientation, scale, and position of the rendered objects. The second step of the approach is our method based on selfsupervised learning that enables better learning of features representation for the objects whose presence we need to verify. Relying on the concept of AAE, our trained models can focus on the most important, geometrical features to represent target objects, while ignoring irrelevant information such as texture, color, or background.

We evaluate our method on a real use case from an industrial context, namely on a complex structure of an airplane engine. We train the models on synthetic data without using a single real image that contains a target object. We test the models on real images. We achieve an $F1$-score comparable to the baseline model trained on real images. Moreover, when we simplify the problem from multiclass to a binary classification, the model trained on synthetic images outperformed the baseline model trained on real images by 3%.

References

1. G. Csurka, "Domain adaptation for visual applications: a comprehensive survey," https://arxiv.org/abs/1702.05374 (2017).
2. B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Thirtieth AAAI Conf. Artif. Intell.* (2016).
3. J. Donahue et al., "Decaf: a deep convolutional activation feature for generic visual recognition," in *Int. Conf. Mach. Learn.*, pp. 647 655 (2014).
4. J. Tobin et al., "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ Int. Conf. Intell. Rob. and Syst. (IROS)*, IEEE, pp. 23 30 (2017).
5. X. Peng et al., "Learning deep object detectors from 3D models," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 1278 1286 (2015).

6. S. Hinterstoisser et al., "On pre-trained image features and synthetic images for deep learning," in *Proc. Eur. Conf. Comput. Vision (ECCV)* (2018).
7. J. Borrego et al., "Applying domain randomization to synthetic data for object category detection," https://arxiv.org/abs/1807.09834 (2018).
8. J. Tremblay et al., "Training deep networks with synthetic data: bridging the reality gap by domain randomization," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit. Workshops*, pp. 969 977 (2018).
9. A. Prakash et al., "Structured domain randomization: bridging the reality gap by context-aware synthetic data," https://arxiv.org/abs/1810.10093 (2018).
10. F. Sadeghi and S. Levine, "Cad2rl: real single-image flight without a single real image," https://arxiv.org/abs/1611.04201 (2016).
11. I. Jovančević et al., "Automated exterior inspection of an aircraft with a pan-tilt-zoom camera mounted on a mobile robot," *J. Electron. Imaging* **24**(6), 061110 (2015).
12. H. Ben Abdallah et al., "Automatic inspection of aeronautical mechanical assemblies by matching the 3D CAD model and real 2D images," *J. Imaging* **5**, 81 108 (2019).
13. I. Jovančević et al., "3D point cloud analysis for detection and characterization of defects on airplane exterior surface," *J. Non Destruct. Eval.* **36**, 74 (2017).
14. H. Ben Abdallah et al., "3D point cloud analysis for automatic inspection of complex aeronautical mechanical assemblies," *J. Electron. Imaging* **29**(4), 041012 (2020).
15. I. Mikhailov et al., "Classification using a three-dimensional sensor in a structured industrial environment," *J. Electron. Imaging* **29**(4), 041008 (2020).
16. A. Boughrara et al., "Inspection of mechanical assemblies based on 3D deep learning approaches," *Proc. SPIE* **11794**, 1179407 (2021).
17. P. Ghimire, I. Jovančević, and J.-J. Orteu, "Learning local descriptor for comparing renders with real images," *Appl. Sci.* **11**(8), 3301 (2021).
18. A. G. Abubakr et al., "On learning deep domain-invariant features from 2D synthetic images for industrial visual inspection," *Proc. SPIE* **11794**, 1179418 (2021).
19. I. Viana et al., "Inspection of aeronautical mechanical parts with a pan-tilt-zoom camera: an approach guided by the computer-aided design model," *J. Electron. Imaging* **24**, 061118 (2015).
20. J. Leiva et al., "Automatic visual detection and verification of exterior aircraft elements," in *IEEE Int. Workshop of Electron., Control, Meas., Signals and Their Appl. to Mechatron. (ECMSM)*, IEEE, pp. 1 5 (2017).
21. J.-K. Park et al., "Machine learning-based imaging system for surface defect inspection," *Int. J. Precis. Eng. and Manuf.-Green Technol.* **3**, 303 310 (2016).
22. J. Miranda et al., "Machine learning approaches for defect classification on aircraft fuselage images aquired by an UAV," *Proc. SPIE* **11172**, 1117208 (2019).
23. J. Miranda et al., "UAV-based inspection of airplane exterior screws with computer vision," in *14h Int. Joint Conf. Comput. Vision, Imaging and Comput. Graphics Theory and Appl.*, Prague, pp. 421 427 (2019).
24. J.-C. Chien, M.-T. Wu, and J.-D. Lee, "Inspection and classification of semiconductor wafer surface defects using CNN deep learning networks," *Appl. Sci.* **10**(15), 5340 (2020).
25. S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.* **22**(10), 1345 1359 (2009).
26. Y. Aytar et al., "Cross-modal scene networks," *IEEE Trans. Pattern Anal. Mach. Intell.* **40**(10), 2303 2314 (2016).
27. L. Castrejon et al., "Learning aligned cross-modal representations from weakly aligned data," in *IEEE Conf. Comput. Vision and Pattern Recognit. (CVPR)*, IEEE (2016).
28. H. Venkateswara et al., "Deep hashing network for unsupervised domain adaptation," in *(IEEE) Conf. Comput. Vision and Pattern Recognit. (CVPR)* (2017).
29. S. Saxena and J. Verbeek, "Heterogeneous face recognition with CNNs," *Lect. Notes Comput. Sci.* **9915**, 483 491 (2016).
30. H. V. Nguyen et al., "DASH-N: joint hierarchical domain adaptation and feature learning," *IEEE Trans. Image Process.* **24**(12), 5479 5491 (2015).
31. S. E. Ebadi et al., "PeopleSansPeople: a synthetic data generator for human-centric computer vision," https://arxiv.org/abs/2112.09290 (2021).

32. J. Yosinski et al., "How transferable are features in deep neural networks?" in *Adv. Neural Inf. Process. Syst.*, pp. 3320 3328 (2014).
33. M. Wang and W. Deng, "Deep visual domain adaptation: a survey," *Neurocomputing* **312**, 135 153 (2018).
34. H. Lu et al., "When unsupervised domain adaptation meets tensor representations," in *Proc. IEEE Intl. Conf. Comput. Vision*, pp. 599 608 (2017).
35. B. Chu et al., "Best practices for fine-tuning visual classifiers to new domains," *Lect. Notes Comput. Sci.* **9915**, 435 442 (2016).
36. A. Rozantsev, M. Salzmann, and P. Fua, "Beyond sharing weights for deep domain adaptation," *IEEE Trans. Pattern Anal. Mach. Intell.* **41**(4), 801 814 (2019).
37. Y. Bengio et al., "Learning deep architectures for AI," *Found. Trends Mach. Learn.* **2**(1), 1 127 (2009).
38. X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: a deep learning approach," in *Proc. 28th Int. Conf. Mach. Learn. (ICML-11)*, pp. 513 520 (2011).
39. M. Ghifary et al., "Deep reconstruction-classification networks for unsupervised domain adaptation," *Lect. Notes Comput. Sci.* **9908**, 597 613 (2016).
40. K. Bousmalis et al., "Domain separation networks," in *Adv. Neural Inf. Process. Syst.*, pp. 343 351 (2016).
41. R. Nevatia and T. O. Binford, "Description and recognition of curved objects," *Artif. Intell.* **8**(1), 77 98 (1977).
42. D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," *Artif. Intell.* **31**(3), 355 395 (1987).
43. J. Liebelt and C. Schmid, "Multi-view object class detection with a 3D geometric model," in *IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recognit.*, IEEE, pp. 1688 1695 (2010).
44. J. Liebelt, C. Schmid, and K. Schertler, "Viewpoint-independent object class detection using 3D feature maps," in *IEEE Conf. Comput. Vision and Pattern Recognit.*, IEEE, pp. 1 8 (2008).
45. M. Stark, M. Goesele, and B. Schiele, "Back to the future: learning shape models from 3D CAD data," in *Proc. BMVC*, pp. 106.1 106.11 (2010).
46. M. Sun et al., "A multi-view probabilistic model for 3D object classes," in *IEEE Conf. Comput. Vision and Pattern Recognit.*, IEEE, pp. 1247 1254 (2009).
47. H. A. Alhaija et al., "Augmented reality meets deep learning for car instance segmentation in urban scenes," in *Proc. British Mach. Vision Conf.* (2017).
48. G. Varol et al., "Learning from synthetic humans," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 109 117 (2017).
49. G. Georgakis et al., "Synthesizing training data for object detection in indoor scenes," https://arxiv.org/abs/1702.07836 (2017).
50. Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh, "How useful is photo-realistic rendering for visual learning?" *Lect. Notes Comput. Sci.* **9915**, 202 217 (2016).
51. A. Gupta, A. Vedaldi, and A. Zisserman, "Synthetic data for text localisation in natural images," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 2315 2324 (2016).
52. B. Planche et al., "DepthSynth: real-time realistic synthetic data generation from CAD models for 2.5D recognition," in *Int. Conf. 3D Vision (3DV)*, IEEE, pp. 1 10 (2017).
53. S. Ren et al., "Faster R-CNN: towards real-time object detection with region proposal networks," in *Adv. Neural Inf. Process. Syst.*, pp. 91 99 (2015).
54. A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *IEEE Conf. Comput. Vision and Pattern Recognit.*, IEEE, pp. 3354 3361 (2012).
55. OpenGL, "OpenGL website," https://www.opengl.org/ (accessed 1 June 2022).
56. J. Groff, "An intro to modern OpenGL," 2010, http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Table-of-Contents.html (accessed 1 June 2022).
57. B. T. Phong, "Illumination for computer generated pictures," *Commun. ACM* **18**(6), 311 317 (1975).
58. R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *ACM Siggraph Comput. Graphics* **15**(3), 307 316 (1981).

59. "OpenGL material," 1994, http://devernay.free.fr/cours/opengl/materials.html (accessed 1 June 2022).
60. B. Stenseth, "OpenGL Light and materials," http://www.it.hiof.no/~borres/j3d/explain/light/p-materials.html (accessed 1 June 2022).
61. M. Born and E. Wolf, *Principles of Optics*, 7th ed., Cambridge University Press (1999).
62. O. Gulbrandsen, "Artist friendly metallic Fresnel," *J. Comput. Graphics Tech.* **3**(4), 64 72 (2014).
63. Shea McCombs, "Intro to Procedural Textures," http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures (accessed 1 June 2022).
64. Scratchapixel 2.0, "Introduction to Shading," https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/procedural-texturing (accessed 1 June 2022).
65. K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.* **19**, 287 296 (1985).
66. K. Perlin, "Improving noise," in *Proc. 29th Annu. Conf. Comput. Graphics and Interact. Tech., SIGGRAPH '02*, ACM, New York, pp. 681 682 (2002).
67. S. Gustavson, "Perlin noise," https://en.wikipedia.org/wiki/Perlin noise (accessed 1 June 2022).
68. "Perlin noise code," https://github.com/stegu/perlin-noise (accessed 1 June 2022).
69. E. W. Weisstein, "Spherical coordinates," https://mathworld.wolfram.com/SphericalCoordinates.html (accessed 1 June 2022).
70. M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *Lect. Notes Comput. Sci.* **8689**, 818 833 (2014).
71. R. Geirhos et al., "Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," https://arxiv.org/abs/1811.12231 (2018).
72. L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 2414 2423 (2016).
73. W. Brendel and M. Bethge, "Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet," https://arxiv.org/abs/1904.00760 (2019).
74. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Technical report, California Univ. San Diego La Jolla Inst. for Cognitive Science (1985).
75. P. Vincent et al., "Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.* **11**(110), 3371 3408 (2010).
76. M. Sundermeyer et al., "Implicit 3D orientation learning for 6D object detection from RGB images," in *Eur. Conf. Comput. Vision (ECCV)* (2018).
77. C. Szegedy et al., "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 2818 2826 (2016).
78. Keras, "Keras applications," https://keras.io/applications/ (accessed 1 June 2022).
79. scikit-Learn, "Precision, recall and F1-score," https://scikit-learn.org/stable/auto examples/model selec-tion/plot precision recall.html (accessed 1 June 2022).
80. "Scikit-learn," https://scikit-learn.org (accessed 1 June 2022).
81. scikit-Learn, "Precision recall supports," https://scikit-learn.org/stable/modules/generated/sklearn.metrics-.precision recall fscore support.html#sklearn.metrics.precision recall fscore support (accessed 1 June 2022).

Abdelrahman G. Abubakr graduated with an engineering degree in electrical engineering from Alexandria University, Egypt, in 2016. He received his master's degree in computer science from the National School of Computer Science and Applied Mathematics of Grenoble, France, in 2019. The work in this paper was done during his master's thesis internship with Diota. He worked for several companies as a computer vision engineer. He is currently working at NVISO, and focusing on developing efficient deep learning models for edge devices for automotive and robotics applications.

Igor Jovančević graduated in 2008 from Faculty of Natural Science and Mathematics within University of Montenegro with a mathematics degree (spec. computer science). He graduated in 2011 from joint Erasmus Mundus Master program in computer vision and robotics (VIBOT) organized by the University of Burgundy (Le Creusot, France), University of Girona

(Girona, Spain), and Heriot Watt University (Edinburgh, United Kingdom). He received his PhD in computer vision in 2016 from IMT Mines Albi (Albi, France), a French "Grande Ecole" specialized in process engineering. He worked for more than 4 years at Diota, Toulouse, as a research engineer focusing on computer vision applications on the problems of inspection and manufacturing process monitoring. He is currently carrying out his research work in the same domain at the University of Montenegro.

Nour Islam Mokhtari graduated with an engineering degree in control and automation from Ecole Nationale Polytechnique of Algiers, Algeria, in 2016. He also graduated with a master's degree in computer vision from the University of Burgundy in France in 2018. He worked in several companies as a machine learning engineer with a focus on computer vision applications. He is currently working as a machine learning engineer at Orpalis, a company that builds document imaging systems.

Hamdi Ben Abdallah graduated with an engineering degree in electronics and industrial control from the National Engineering School of Sousse, Tunisia, in 2016. He received his PhD in computer vision in 2020 from IMT Mines Albi (Albi, France), a French "Grande Ecole" specialized in process engineering. Currently, he is working at Institut Clément Ader, Albi, as a research engineer focusing on computer vision applications on the problems of inspection and manufacturing process monitoring.

Jean-José Orteu graduated in 1987 from a French "Grande Ecole" (ENSEIRB, Bordeaux, France) with an engineering degree in electrical and software engineering and a master's thesis in automatic control. He received his PhD in computer vision in 1991 from the Université Paul Sabatier (Toulouse, France). Currently, he is a full professor at IMT Mines Albi (Albi, France), a French "Grande Ecole" specialized in process engineering. He carries out his research work in the Institut Clément Ader (ICA) laboratory (200 people). For more than 15 years, he has developed computer-vision-based solutions for 3-D measurements in experimental mechanics (PhotoMechanics) and, since 2010, he is more specifically involved in the application of computer vision to NDE, inspection, and manufacturing process monitoring.