Exploring the Sim2Real Gap using Digital Twins

Sruthi Sudhakar[1]* Jon Hanzelka[2] Josh Bobillot[2]
Tanmay Randhavane[2] Neel Joshi[2] Vibhav Vineet[2]
[1]Columbia University [2] Microsoft Research

Abstract

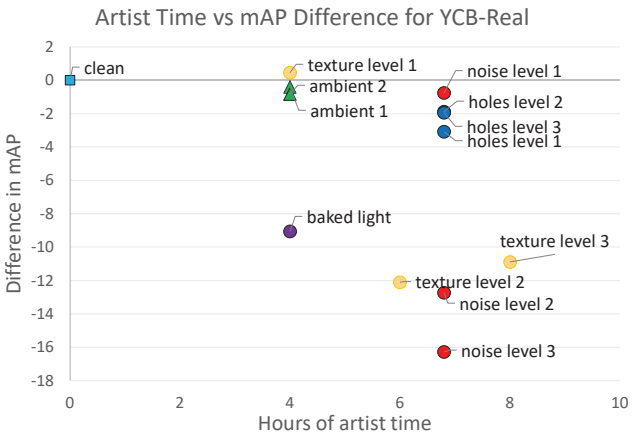
It is very time consuming to create datasets for training computer vision models. An emerging alternative is to use synthetic data, but if the synthetic data is not similar enough to the real data, the performance is typically below that of training with real data. Thus using synthetic data still requires a large amount of time, money, and skill as one needs to author the data carefully. In this paper, we seek to understand which aspects of this authoring process are most critical. We present an analysis of which factors of variation between simulated and real data are most important. We capture images of YCB objects to create a novel YCB-Real dataset. We then create a novel synthetic "digital twin" dataset, YCB-Synthetic, which matches the YCB-Real dataset and includes variety of artifacts added to the synthetic data. We study the affects of these artifacts on our dataset and two existing published datasets on two different computer vision tasks: object detection and instance segmentation. We provide an analysis of the cost-benefit trade-offs between artist time for fixing artifacts and trained model accuracy. We plan to release this dataset (images and 3D assets) so they can be further used by the community. Link to dataset[1]

1. Introduction

A computer vision model must be trained on a large dataset that samples across different environments and conditions so what is learned from it generalizes well. The traditional process is to train models on large real world image datasets, which require a lot of time and cost to create [18]. Image collection alone is difficult, time consuming, has low scalability, and is sometimes impossible to do [2, 16, 26]. Furthermore, annotating the data is time-intensive.

An interesting alternative is to use computer graphics methods to render large sets of synthetic images for training vision models. Synthetic data provides many benefits – it is easy to create a vast number of data variations with minimal human effort, and the annotation of the data is essentially free. These properties have lead an increased use of synthetic data in academia [20, 25, 28, 2] and industry (e.g. Microsoft AirSim, NVIDIA's Carla and Omniverse platforms, Unity's Computer Vision API, Datagen, among many others).

Figure 1: For each artifact that can arise in synthetic data creation, we show how long it takes to fix it vs. the drop in mAP of the model trained with that data. This provides actionable insights as to how to balance the time and cost for synthetic data generation.



Despite these great benefits, broad use is still limited, as models trained on synthetic data often under-perform on real world test data relative to models trained on real world data [29]. Furthermore, many publicly available object models, such as ShapeNet [3], contain meshes which render very poorly in 3D graphics software like Blender, Arnold, etc., leading to difficulties in training for 3D vision tasks. This performance gap, known as the "Sim2Real" gap, is well known, and it is it caused by numerous factors that make synthetic data a bad match for the data distribution seen in the real world test data [29]. However, while the problem is well known, there is actually quite little under-

*Correspondence to Sruthi Sudhakar at ss6638@columbia.edu.

[1]Dataset: https://github.com/SruthiSudhakar/Exploring-the-Sim2Real-Gap-using-Digital-Twins-Dataset.git

standing of what exactly is causing the gap in the first place. It is the goal of this paper to explore this question. In particular, we focus on issues related to the quality of the 3D content used to create the synthetic data.

The cases where synthetic data performs well are generally the result of great care and highly skilled, human effort taken to manually bridge the Sim2Real gap. One common process is to set up the synthetic data to look as close as possible to the real world test data [24]. The extensive time and skills needed for this process is prohibitive for most typical computer vision research teams and thus effectively cancels out the potential gains over using real data. This is a road block to the broad use of synthetic data in computer vision.

However, if one can understand what aspects of realism have the largest affect on the performance, then they can reduce the human effort needed to create highly-performing synthetic data. One can steer more limited resources to fixing the most important factors alone, thus reducing the barriers for using synthetic data effectively.

To help understand the relevant processes and what aspects of it are most critical, we partnered with a team of artists and data-scientists in a large corporation that performs authoring of and training with synthetic data for a number of large commercial clients. The following is their process, and it is common in the industry: Step 1) Digitize the objects and environments to use as building blocks for the synthetic scenes using automated photogrammetry and manually clean up of model artifacts and Step 2) use these realistic 3D assets to create scenes by placing them in different environments and rendering them with a variety of cameras positions under various lighting environments and export metadata (object labels, masks, etc.).

This process leads to synthetic data that works quite well, but is very time consuming to create. While "Step 2" is fairly automated and requires less manual effort, "Step 1" can take easily over a day for even small, fairly straightforward objects. The process also requires considerable skills and intuition that are typically the domain of highly trained 3D artists. In this paper, we focus on these manual, skilled steps – the ones that fix artifacts that appear in the automated photogrammetry pipeline – to understand what is critical to the performance of the synthetic data.

We study these properties on objects from the YCB dataset [30]. We have created a synthetic training dataset and a real test dataset where all conditions of the environment and object match including camera angles, lighting, foreground, background, and textures – also known as a "Digital Twin".

From these twins, we create new datasets where we synthetically introduce artifacts in a controlled manner to understand which 3D-model artifacts cause the greatest drops in performance when their rendered data is used to train a computer vision model. Isolating which factors cause drops in performance requires training and testing in very controlled settings where *only one* factor changes at a time, and thus we have constructed our dataset to have this property. The artifacts we study are 5 factors that our partner team identified most typically arise during photogrammetry and are corrected to create good synthetic data: noise in the 3D mesh, holes in the mesh, texture blur, and variations in diffuse vs non-diffuse (or "baked") lighting.

We then train on these modified datasets and test on our real dataset to understand which factor causes the greatest drops in performance. Further, we test these models on 2 more real-world datasets including YCB-In-the-wild [10] and YCB-Video [34] and show that these findings hold.

For each artifact, we provide an estimate for the time taken by an artist to correct that artifact, so that we can understand the trade-off between how long it takes to fix each artifact to obtain a clean model and the accuracy benefit provided by that fix (summarized in Figure 1). This provides time and cost constrained researchers with actionable insights to prioritize which factors to address.

In summary, our main contributions include:

- Discovery of which factors of variation in 3D model quality between simulated and real data are important for computer vision model performance
- A cost-benefit analysis between artist time for correcting an artifact and trained model accuracy
- A new `YCB-Real` and digital twin `YCB-Synthetic` dataset to study adaptation and generalization in controlled environments

2. Related Work

2.1. Synthetic Data for Robustness Measures:

Several works have aimed to generate data with varying types of image corruptions to understand model failures. In [12], authors introduce 75 common corruptions (categorized by noise, blur, weather, and digital categories) on ImageNet [5] images to create ImageNet-C. Similarly, in Foggy Cityscapes [23], synthetic fog is applied to Cityscapes [4] data at three different levels of severity to understand the impact of weather changes. To assess distortion robustness in object detection, [19] introduces PASCAL-C, COCO-C, and Cityscapes-C containing 15 distortions, following those proposed in [12], each spanning five levels of severity.

These corruptions are all introduced in the pixel space and usually applied over the entire image. Such corruptions can look unrealistic and the images may not mirror typical issues that arise when creating synthetic data. Conversely, our work takes a more realistic approach by creating datasets where our 5 identified factors are applied on the 3D assets, and later rendered into a scene with various camera positions and lighting environments. Furthermore, this method of data generation allows us to introduce changes in object shape and 3D pose, which are not captured by

corruptions introduced in prior work. Finally, many types of corruptions in prior work's datasets are a composite of different factors of the object and environment changing including textures, lighting, shapes, and clutter. In our work, we take a more rigorous approach to test each of these factors in isolation such that generalizations can be then made to higher-level variations in the data.

We introduce two novel datasets, `YCB-Synthetic` and `YCB-Real` that provide a strong test-bed to analyze whether current robustness algorithms are successful. With our dataset, one can discover the precise situations wherein their model fails since each factor or variation is isolated.

2.2. Synthetic Data Generation

There are many synthetic datasets that are generated via graphic engines and physics simulators such that they can create synthetic data that matches the real data as closley as possible. For example, Virtual KITTI [8] was designed to match the KITTI dataset as close as possible. Many other self-driving synthetic datasets aim to achieve the same including CARLA [6], SYNTHIA [22], GTA-V [21], and VIPER [11]. However, many works have shown performance drops when training on these virtual datasets and testing in real environments, showing that these graphic simulators are unable to reconstruct these complex scenes perfectly in the virtual world.

Another explored approach is domain randomization [27] which involves defining a set of parameters that will be perturbed at random while generating the synthetic data. These perturbations include aspects like camera angle, illumination, and object pose which also can only be achieved with sophisticated graphics software.

Other work has aimed to leverage the power of generative models to create synthetic data. Some common generative algorithms include GANs [14], VAEs [1], and more recently, diffusion models [13]. However, these algorithms are currently limited in their power to generate high-quality, high-resolution images with the level of detail that match reality. Furthermore, it is not possible to specify the type of data one wishes to generate. While you can condition outputs on some label or text, there is not much controlability of precisely how the image looks in terms of lighting, object orientation, camera viewpoints, etc.

Since our work requires controlling low-level details including lighting, camera viewpoints, object textures, etc., we take the approach of using 3D scan technologies and rendering software with the help of specialized data artists to create these synthetic datasets. Furthermore, unlike any prior work, we develop digital twins of the synthetic and real data, ensuring that no sim2real gap exists, allowing deeper exploration into the root causes of these gaps.

2.3. Sim2Real Adaptation methods

Domain Adaptation (DA) and Domain Generalization (DG) are two very standard paradigms to allow a model trained on a(many) source distribution(s) to adapt to a target distribution that is different but related. In DA, the model has access to the target domain data and many frameworks focus on aligning the target to the source domain via feature re-weighting [17], domain discrepancy minimization [7], adversarial learning [9], and invariant feature learning [31]. In Domain Generalization, the target distribution is unknown so prior DA methods do not work perfectly in these scenarios, rather approaches such as domain randomization [27], data augmentation [15], and domain-invariant learning [33] are commonly used.

Our dataset can serve as a benchmark and in-depth analysis for any of these adaptation methods. Since we have identified the main variations between object models in synthetic and real data that cause performance drops, one can train their adaptation method on any one of our synthetic datasets and see if it transfers to our real dataset. This will indicate not only the strength of the method, but also since we have isolated each factor in these datasets, it highlights which artifact their algorithm is prone to failures on.

3. General Synthetic Data Creation Pipeline

A typical synthetic data creation pipeline has two steps: asset creation and scene composition. While the artist can author everything from scratch, for the sake of time and accuracy, the common approach is to first digitize the assets and scene and then perform 3D modeling op top of these assets to create clean, accurate versions.

3.1. Asset Digitization

In the first step, an artist creates the building blocks of the synthetic scenes which include objects and environments.

Object Creation: A variety of different laser or light-based 3D scan technologies exist for object digitization, the most common of which is photogrammetry. In this technique, an object is photographed (typically with a DSLR or phone camera) from a variety of angles ensuring there is an adequate overlap between photos. Automated photogrammetry (structure from motion and dense multi-view stereo) is then used to reconstruct a textured 3D mesh from these images that provides an initial 3D model the real object.

Environment Creation: To digitize the 3D environment with accurate lighting conditions observed in the real world, we capture a 360-degree HDR (high dynamic range) image of the real environment using a specialized panoramic camera rig that pivots about the camera's nodal point while capturing bracketed exposure stops [32]. A specialized software and a color management pipeline is then used to com-

bine the exposures to create a 32 bit floating point HDR map. This HDR environment accurately represents the color and exposure values and can be used to derive ambient and directional lighting contribution to the 3D scene.

3.2. Scene Composition

After the objects and the environments are digitized, the artist configures the scene layout based on the specific goals for the synthetic dataset.

This includes: (1) Targeted placement of the object(s) in the environment/scene. (2) Programmatic distribution of 3D camera positions to obtain a variety of viewpoints. (3) Configuration of the environments' material and lighting parameters to align with the real world. (4) Assignment of labeling information for ground truth computation

When the scene configuration is completed, it is rendered, typically using a ray-traced engine, that produces highly photorealistic images and pixel perfect annotations.

3.3. Model Quality Factors

Data generated using this process works quite well, however, this process is time consuming and often requires highly skilled 3D artists to fix several model quality issues that arise during object digitization using photogrammetry. The typical model quality factors are:

- **Holes in Geometry:** The 3D mesh may have holes due to the capture process such as the lack of angular coverage in the photography or the inability to capture occluded areas. The surface properties of the object may also result in holes in the 3D mesh. For example, photogrammetry often has difficulty in reconstructing shiny, black, and featureless areas of an object.
- **Texture Artifacts:** Image anomalies that are present in the source photography will also be present in the resulting texture. This is commonly caused by blurry images due to motion or focal depth. Misalignments in the computed camera positions can also result in texture artifacts and can lead to errors in the texture re-projection. This is observed as the doubling of projected details in the texture.
- **Lighting Artifacts:** The lighting conditions when capturing photogrammetry imagery are baked into the resulting texture. If consistent diffuse lighting is not ensured during the capture process, shadows, light directionality, and specular/reflective properties may get embedded in the resulting texture.
- **Mesh Noise:** High-frequency noise in the mesh reconstruction is a common issue that is often caused by lacking enough imagery to properly reconstruct the object. This can also be caused by surface properties like dark or shiny materials that lead to holes in geometry.

Figure 2: We obtain object models and create synthetic data for these 20 objects in the YCB Objects dataset. We show each object's real and synthetic version side-by-side.

4. Datasets

To isolate which model quality factor causes a drop in performance, it is necessary to ensure that all other aspects of the synthetic data we train on and real data that we test on remain the same. To create a digital twin of this nature, we required a dataset that allowed us high control in both synthetic and real. Therefore, we chose to use the 20 daily life objects (depicted in Figure 2) of different shapes, sizes, and textures from The YCB Object and Model Set [34]. The dataset provides us with high-resolution RGBD scans, physical properties, and geometric models to generate synthetic data of these objects. We additionally purchased 16/20 of the objects for which there exists products in real life that look like the objects in the dataset (apart from a few differences due to the changes in product packaging since the release of YCB). These objects include banana, bleach, block, bowl, cheezit, coffee, drill, gelatin, marker, meat, mug, mustard, pitcher, pudding, soup, sugar.

Figure 3: Example of cheezit box from each dataset: digital twin in `YCB-Synthetic`, from `YCB-Real`, from `YCB-In-the-wild`, and from `YCB-Video`.

4.1. Creating YCB-Real

We obtained a staging table and placed it in the capture space. We ensured that the real environment exactly matched the synthetic environment by digitizing the staging table and the capture space and using them to create YCB-Synth. The scene was lighted using an area light in the ceiling which provided ambient lighting and two floor lamps which provided directional lighting. We put markers on the floor to ensure that the table and the floor lamps were in the exact same position in all captures. We captured the different combinations of lighting environment maps using light probes that were placed at the center of the table. This point was treated as the origin of the coordinate system for the synthetic scene ensuring that the lighting in both scenes was exactly the same.

We captured the real data in multiple sessions. At the beginning of each session, we set up the staging table and the lights at their marker positions. We then put the real objects on the table and captured multiple images from various camera positions using an Android phone. We ensured that the camera positions covered a variety of distances from the object ranging from 0.2 to 5 meters. After obtaining the images, we labeled them manually using a labeling software. As a result, we obtained a real dataset containing $\sim$100 images per object that has the same lighting conditions and environments as YCB-Real.

4.2. Creating YCB-Synthhetic

To understand the effect of the 5 artifacts (model quality factors), we identified a series of experiments that would allow us to artificially add these issues back into a clean version of the models and independently analyze their impact.

- **Clean Dataset (clean):** Our first step is to generate an initial dataset where all 20 objects are staged and rendered to match YCB-Real as closely as possible. Once the environment and lighting are set up, the objects can then be brought into the scene and placed on the table. We then fine tune any materials, asset placements, and lighting, to ensure the datasets are as close to the real-world captures as possible. All subsequent datasets are based on this initial dataset.
- **Ambient Lighting (ambient 1/2):** We created this dataset to understand the effect of varied light exposure levels on model performance. Using the captured HDR images of the real-world space, we generated datasets with the light set to 50%, and 10% exposure intensity (where 100% is simply the clean dataset).
- **Baked Lighting (baked light):** We swap the HDR environment map captured from our lab setup with a completely different HDR environment (from a public HDR library) that has very different, harsh lighting. When the objects are rendered with this environment map, the lighting appears very different than in the real scene. This was done to mimic photogrammetry captures that were not taken in a diffuse lighting environment. When this happens shadows and lighting information can be much different than the environment the objects are rendered in.
- **Holes in Geometry (holes 1/2/3):** To evaluate the impact of holes in the object's geometry, we purposefully select parts of the mesh to delete. We target areas that have flat featureless sections and solid colors. This mimics patterns observed in photogrammetry. We created 3 datasets by adding more and more holes progressively at 3 different levels which we term level 1 for least severity, up to level 3 for most severity. Where Level 0 is simply the clean dataset.
- **Blur in Texture (texture 1/2/3):** By adding blur into the textures we mimic photogrammetry that has misaligned cameras or poor-quality photography (motion blur, out of focus). Object textures are modified in Photoshop to apply layer offsets and opacity changes that mimic these artifacts. Similar to holes, we created datasets with 3 levels of severity.
- **Noise in mesh (noise 1/2/3):** For these experiments we introduce different amounts of noise into the object's mesh. We achieve this by adding random amounts of curl and fractal noise computations into the vertex positions of the mesh, this causes the surface of the mesh to become deformed and noisy. Similar to holes and blur, we created datasets with 3 levels of severity.

We render 1000 images/object for each object in all of these datasets with closely matching camera ranges, object ranges, and camera angles as YCB-Real. Figure 4 shows an example of each type of artifact dataset on the cheezit object. Figure 3 shows a sample of a cheezit box image in real and its digital twin in synthetic.

Figure 4: Example of cheezit object with each type of artifact applied from the YCB-Synthetic artifact datasets.

5. Experiments

Our dataset provides the community with a systematic way to study how model quality artifacts in synthetic

Table 1: Object Detection results (mAP) from training on each type of artifact in YCB-Synthetic and testing on YCB-Real. Notice that when training on clean and testing on YCB-Real, we see near perfect mAP50 as expected. However, training on various artifacts causes the performance to drop. From this table, we can see that the biggest performance impactors include high levels of noise in the mesh, high texture blurs, and baked lighting. Conversely, ambient lighting does not affect performance, and holes only have mild affects. We highlight the lowest 3 performing attributes in red.

train set	all	sugar	mug	gelatin	banana	bowl	drill	bleach	block	meat	marker	cheezit	pitcher	mustard	pudding	coffee	soup
clean	81.93 ± 0.98	86.69	84.55	82.14	77.03	86.71	86.82	67.84	90.20	86.90	38.17	92.16	90.44	82.65	83.75	90.18	84.64
ambient 1	81.53 ± 0.75	86.51	83.20	80.70	73.34	85.50	85.75	74.49	91.52	87.02	38.34	90.81	90.48	81.46	80.26	91.95	83.20
ambient 2	81.10 ± 1.08	85.10	84.96	80.01	75.17	85.61	85.17	73.03	89.06	80.44	36.60	90.64	91.14	81.97	84.77	91.78	82.11
holes 1	78.85 ± 2.14	81.79	83.14	80.18	75.94	84.43	86.76	65.38	91.17	85.91	37.64	90.67	90.84	82.58	53.33	86.11	85.69
holes 2	80.05 ± 1.30	82.55	84.34	81.74	76.09	83.65	85.56	63.47	90.65	82.75	43.62	91.04	90.46	82.19	71.57	86.65	84.48
holes 3	79.99 ± 1.35	86.85	85.01	80.48	80.03	81.78	86.61	58.02	89.98	83.43	35.32	91.10	91.50	81.90	78.55	85.50	83.84
texture 1	82.29 ± 1.11	86.31	86.38	81.81	77.08	87.21	86.68	69.72	92.53	87.20	35.96	92.41	91.26	82.80	85.93	89.11	84.29
texture 2	**69.82 ± 4.47**	85.65	84.78	13.72	78.23	83.59	86.20	56.98	87.03	34.55	31.74	61.11	92.10	83.29	75.22	77.25	81.39
texture 3	71.04 ± 4.74	83.47	85.24	74.49	82.81	82.56	85.70	63.20	93.19	67.63	33.16	21.37	90.39	82.18	43.62	66.25	81.41
baked light	72.87 ± 1.23	83.03	84.47	79.60	70.79	87.11	83.60	60.99	90.02	80.01	21.68	74.93	91.77	83.83	4.82	87.30	81.88
noise 1	81.17 ± 1.17	84.81	85.08	80.59	78.55	85.29	87.08	65.69	89.58	79.64	37.25	91.23	90.44	84.65	84.13	91.14	83.51
noise 2	**69.20 ± 4.92**	84.71	36.10	71.39	82.41	80.44	81.96	63.60	89.52	51.01	37.38	87.19	9.26	84.15	80.13	84.27	83.63
noise 3	**65.65 ± 4.84**	83.38	63.59	78.77	81.76	68.18	80.64	70.31	87.45	31.09	38.07	71.28	4.67	85.04	85.76	36.52	83.94

Table 2: Object Detection results from training on various artifact datasets in YCB-Synthetic and testing on (a) YCB-Real, (b) YCB-In-the-wild, and (c) YCB-Video. Notice that many overall trends that we see on YCB-Real hold for YCB-In-the-wild and YCB-Video. We highlight the lowest 3 performing attributes in each dataset, note the consistency across datasets.

(a) YCB-Real

train set	mAP	mAP50	mAP75
clean	81.93 ± 0.98	99.40 ± 0.66	93.90 ± 0.78
ambient 1	81.53 ± 0.75	99.36 ± 0.34	94.03 ± 0.55
ambient 2	81.10 ± 1.08	99.20 ± 0.71	93.73 ± 0.69
holes 1	78.85 ± 2.14	96.74 ± 1.82	90.84 ± 2.21
holes 2	80.05 ± 1.30	97.94 ± 1.44	92.12 ± 1.39
holes 3	79.99 ± 1.35	98.23 ± 1.53	90.63 ± 1.70
texture 1	82.39 ± 1.11	99.31 ± 0.81	93.91 ± 0.76
texture 2	**69.82 ± 4.47**	85.68 ± 5.26	79.86 ± 5.14
texture 3	71.04 ± 4.74	88.69 ± 5.16	81.72 ± 5.12
baked light	72.87 ± 1.23	90.19 ± 1.17	84.46 ± 1.11
noise 1	81.17 ± 1.17	98.72 ± 1.26	92.88 ± 1.36
noise 2	**69.20 ± 4.92**	85.85 ± 5.14	79.50 ± 5.36
noise 3	**65.65 ± 4.84**	81.32 ± 4.96	75.71 ± 5.13

(b) YCB-In-the-wild

train set	mAP	mAP50	mAP75
clean	67.15 ± 2.56	77.47 ± 2.69	75.25 ± 2.98
ambient 1	65.76 ± 2.05	76.20 ± 2.12	74.28 ± 2.02
ambient 2	66.35 ± 2.17	76.86 ± 2.28	74.25 ± 2.36
holes 1	60.88 ± 3.12	70.40 ± 3.36	68.35 ± 3.08
holes 2	58.19 ± 3.64	67.49 ± 4.00	65.16 ± 3.84
holes 3	55.19 ± 3.93	64.13 ± 4.28	61.85 ± 4.14
texture 1	65.18 ± 2.09	75.45 ± 2.54	72.90 ± 2.44
texture 2	**40.92 ± 5.81**	47.91 ± 6.57	45.75 ± 6.47
texture 3	44.85 ± 2.91	52.96 ± 3.04	50.42 ± 3.08
baked light	62.65 ± 4.50	73.65 ± 5.01	70.57 ± 5.14
noise 1	61.00 ± 1.78	70.27 ± 2.13	68.56 ± 1.96
noise 2	**33.68 ± 7.58**	40.68 ± 9.03	38.62 ± 8.67
noise 3	**20.30 ± 6.93**	25.11 ± 7.98	23.37 ± 8.01

(c) YCB-Video

train set	mAP	mAP50	mAP75
clean	37.83 ± 2.28	51.74 ± 3.13	44.48 ± 2.77
ambient 1	39.45 ± 2.44	54.30 ± 3.63	46.78 ± 2.89
ambient 2	38.18 ± 2.46	51.85 ± 3.51	45.16 ± 2.80
holes 1	33.95 ± 1.93	46.17 ± 2.37	40.42 ± 2.37
holes 2	30.69 ± 4.57	41.60 ± 6.21	36.25 ± 5.59
holes 3	27.35 ± 2.21	37.13 ± 3.09	32.44 ± 2.70
texture 1	37.37 ± 3.18	51.84 ± 4.71	43.98 ± 3.43
texture 2	**22.54 ± 2.17**	30.06 ± 2.96	25.61 ± 2.45
texture 3	28.03 ± 3.04	39.29 ± 4.44	32.30 ± 3.90
baked light	33.59 ± 1.60	45.97 ± 2.44	39.85 ± 1.98
noise 1	35.67 ± 1.63	48.50 ± 2.47	42.02 ± 1.86
noise 2	**24.02 ± 2.76**	33.54 ± 3.52	28.02 ± 3.30
noise 3	**18.88 ± 3.18**	28.17 ± 4.32	21.80 ± 3.77

training data can result in performance drops when testing on real data. As a first step into investigating the causes of the sim2real gap, we study object detection when training on YCB-Synthetic and testing on real-world datasets including the matched captures, YCB-Real, and two publicly available datasets YCB-In-the-wild [10] and YCB-Video [30], to see how results generalize outside of our own data. YCB-In-the-wild contains YCB objects in real environments with various poses and scales, and YCB-Video includes multiple YCB objects per frame, with different arrangements, spatial configurations, poses, and with severe occlusions (Fig 3). As a second task, we train an instance segmentation model on YCB-Synthetic and test on YCB-Video.

The goal is to understand how training a model on each dataset of 14 artifacts in YCB-Synthetic (as detailed in Sec 4.2), affect the real-world performance of the model. A large drop in performance indicates that a certain factor can have a high impact on the how well the model learns, and therefore this factor is important to address when generating synthetic data. Conversely, if a certain artifact does not affect model performance, we can say that it is not important for data artists to correct this artifact as model performance will not degrade even in the presence of the artifact.

5.1. Training

We used the Detectron2 library and start with a pre-trained Faster-RCNN model for detection and Mask-RCNN for instance segmentation, with a ResNet50 + FPN backbone, initialized with COCO weights. We fine-tuned the pre-trained model on clean synthetic data and all of the 14 types of corrupted synthetic data we have created in the YCB-Synthetic dataset. To eliminate all possible confounding variables and understand the source of the performance discrepancies, we trained all models with no augmentations. We then take each of these trained models and test it on real test data in YCB-Real, YCB-In-the-wild, and YCB-Video. For each experiment, we trained 10 times and we report the average and standard deviation of mAP, AP50, and AP75 metrics.

Figure 5: Results comparing relative performance of Detection vs Segmentation on YCB-Video.

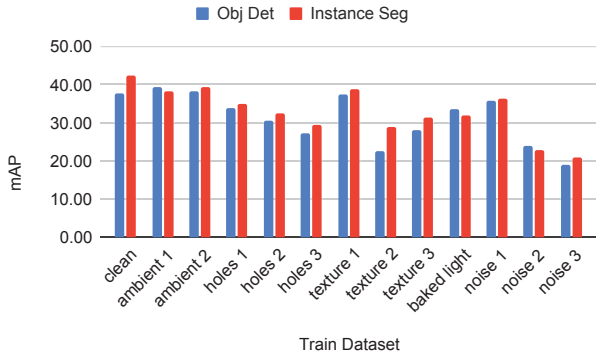
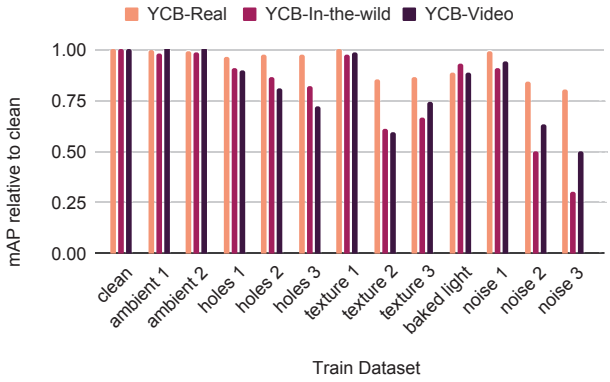


Figure 6: Comparing model trained on clean vs artifacts datasets in`YCB-Synthetic` We divide these two performance numbers to get a relative performance number.



5.2. Baseline

First as a sanity check, we ensure that the detection model trained on the clean synthetic data in `YCB-Synthetic` and tested on the twin, `YCB-Real`, should achieve high performance. We expect this because by design, all environmental conditions match between clean `YCB-Synthetic` and `YCB-Real`, thus we do not anticipate a sim2real gap. As expected, we see excellent performance with an overall mAP of 82% (see Table 1), AP50 of 99%, and AP75 of 93%. Further, we trained on YCB-Real and testing on YCB-Real, and we observe mAP of 87.6% which is similar to training YCB-Synthetic and testing on YCB-Real (the slight increase due to the differences in bounding box accuracies in YCB-Synthetic), indicating that YCB-Synthetic clean is a good digital twin for the real data.

5.3. `YCB-Real` Object Detection

Next, we trained on the 12 (artifact) datasets in `YCB-Synthetic` and tested on `YCB-Real`. As shown in Table 1, overall we found that ambient lighting had almost no impact on performance at both levels. Additionally, holes in the mesh at all levels (1-3) had mild to no impact on model performance. However, baked lighting, texture blur levels 2-3, and noise in the mesh levels 2-3, all caused major drops in model performance with noise being the worst.

5.4. `YCB-Video` and `In-the-wild` Detection

While training on `YCB-Synthetic` and testing on the `YCB-Real` gave us a strong understanding of how each factor specifically affected model performance, we wanted to see if our observed trends generalize to other existing datasets. In Table 2 and Figure 6, we provide detailed results comparing training on the artifacts datasets in `YCB-Synthetic` and testing on (a) `YCB-Real`, (b) `YCB-In-the-wild`, and (c) `YCB-Video`. Firstly, we note that there is a significant domain shift from `YCB-Real` to `YCB-In-the-wild` and `YCB-Video` since these are not digital twins, and this is reflected in the large performance drops. When we train on `YCB-Synthetic` clean and test on these various datasets, we can see decreasing mAPs with `YCB-Real` = 81.93%, `YCB-In-the-wild` = 67.15%, and `YCB-Video` = 37.83%. The lower performance on `YCB-Video` compared to `YCB-In-the-wild` is expected as multiple objects appear in each image with various occlusions, orientations, and backgrounds while `YCB-In-the-wild` has various backgrounds but only has one object per image and is mostly un-occluded.

Interestingly, many of the trends we saw from `YCB-Real` results held true even in these other datasets. We average mAP across the levels of the different artifacts to get a high-level intuition on the results (average result figure in Supplemental). We noticed that like `YCB-Real`, changing the level of ambient lighting had the least drop in model performance from clean. Furthermore, baked lighting had a similar impact level as `YCB-Real` as seen by similar mAP values in Figure 6. Finally, we notice that across all 3 datasets, noise in mesh level 3 results in the greatest performance drops.

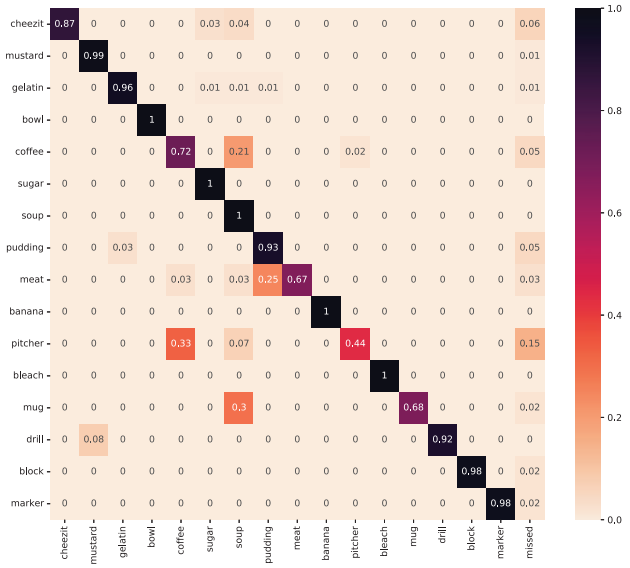
This illustrates that certain artifacts that arise when creating synthetic data have more significant affects on the downstream performance of models trained on this data.

5.5. `YCB-Video` Instance Segmentation

To further test the generalizability of our findings, we ran experiments with another downstream task: Instance Segmentation. In Table 5, we provide results comparing training on various artifacts in `YCB-Synthetic` and testing on `YCB-Video`. Note that the relative results between the Detection Task and Segmentation task are consistent, with the lowest 5 performing artifacts being holes 3, texture 2/3 and noise 2/3 in both tasks.

5.6. Trade-offs between artist time and accuracy

Figure 7: Confusion Matrix of model trained on YCB-Synthetic noise in mesh (averaged across level 1/2/3) and tested on YCB-Real. Notice that noise in mesh causes more class confusions than missed detections.



Our artist team estimated the average time needed on average for a skilled artist to correct each artifact. Model artifacts, such as noise and holes, typically take 6.8 hours to correct, regardless of corruption level as these artifacts are fixed by creating a new, clean model. Lighting and textures issues vary from 4-8 hours based on the severity of the artifact. With these time estimates, we can model the trade-off between how long it takes to fix each artifact and the accuracy benefit provided by that fix for object detection in YCB-Real, as shown in Figure 1 (other tasks/dataset results provided in Supplemental). Our analysis shows that it is less worth it to correct ambient lighting level issues and low-level noise and texture issues, as they have a relatively small affect on mAP. Holes in the mesh are typically also not as much of a factor, while they are very time consuming to fix. The "best bang for one's buck" is fixing baked lighting issues and significant texture and noise errors. With these results, we provide actionable insights as to how to reduce costs of data generation.

6. Discussion

Extreme performance drops: Noise in the mesh seems to have the greatest impact. Rows 'noise 2' and 'noise 3' in Table 1 show this artifact degrades performance significantly, with mAP in YCB-Real falling from 82% to 69% and 66% respectively. Similarly, noise 2 and 3 cause one of the steepest drops in performance for YCB-In-the-wild and YCB-Video as well. Furthermore, in both noise 2 and noise 3, pitcher is most often confused with 'coffee' (see Figure 7), most likely due to their similar shapes and colors.

Model architecture handles certain artifacts: Ambient lighting does not seem to have any impact on model performance in YCB-Real and only a mild impact on YCB-In-the-wild and YCB-Video. One possible reason for this is that lower ambient lighting simply shows up as lower intensity values in the pixel space, which the deep learning training algorithm already accounts for by incorporating normalization and batch norm. (Note we trained these models *without* data augmentations to remove confounding factors).

Synthetic data artifacts can help: From our findings on the twin dataset YCB-Real, we can see that training with some artifacts improve an objects' performance, compared to clean. For example, as we increase the severity of holes in the mesh, we notice that the performance increases on certain classes such as mug, gelatin, and bowl. Additionally, adding texture blur boosts bowl's performance across all levels of severity (the highest being 6% at severity level 1). This highlights a more general finding that some artifacts in the object model are advantageous during testing – perhaps adding some of the benefits of data augmentations such as adding noise Gaussian noise to images. These boosts in performance vary by object and artifact type, and would be interesting to study in more detail in future work using our controlled train/test datasets.

Findings on classification vs localization: While many of these artifacts cause significant drops in performance, some artifacts cause the model to miss-classify one object for another, while other artifacts result in completely missed detections. In Figure 7, we visualize the confusion matrices of a model trained on noise in mesh (averaged across the 3 severity levels) and tested on YCB-Real. While training on noise in mesh and texture blur artifacts cause poor performance, noise in mesh results in a lot more confusions between objects whereas texture blur mostly results in missed detections. Depending on the downstream application, one type of error may be more acceptable than the other, and we encourage future work to use this dataset to look into which types of artifacts cause which types of classification/localization errors. (Other confusion matrices included in Supplemental)

Scope for further analysis by object size: Furthermore, when observing clean confusion matrix, we see that pudding and marker are the two objects that always have poor performance and pudding is always miss-classified as gelatin while marker is always miss-localized. Both objects are some of the smallest amongst the 18 real objects, with the marker object being the smallest. Therefore even slight

artifacts to the object model cause significant visual differences and may lead to classification and localization issues. This highlights an interesting relationship between model quality and object size.

7. Conclusions

We presented a novel synthetic dataset, `YCB-Real`, and its digital twin, `YCB-Synthetic`. We benchmark models on these datasets to understand the affects of artifacts on model performance. Furthermore, we provide cost estimates to help others understand the value of fixing certain artifacts. While we have done a large number of initial experiments with this datasets, we believe there are many more that can be done. Thus one of our main contributions is the dataset and 3D assets used to create it that we plan to release to the public. We are excited to release our data so that the research community can benefit from the highly controlled training and test environments we have created, and they can also use our 3D assets and environments to create new datasets for research into synthetic data generation and additionally domain generalization and adaptation.

Acknowledgements: We thank Pedro Urbina and the members of the MS Synthetics Team for helping in creating the datasets.

References

[1] Jianmin Bao, Dong Chen, Fang Wen, Houqiang Li, and Gang Hua. CVAE-GAN: fine-grained image generation through asymmetric training. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2764–2773, 2017. 3

[2] Sara Beery, Yang Liu, Dan Morris, Jim Piavis, Ashish Kapoor, Markus Meister, Neel Joshi, and Pietro Perona. Synthetic examples improve generalization for rare classes. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 852–862, 2020. 1

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 3

[7] Zhekai Du, Jingjing Li, Hongzu Su, Lei Zhu, and Ke Lu. Cross-domain gradient discrepancy minimization for unsupervised domain adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021. 3

[8] A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 3

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016. 3

[10] Yunhao Ge, Harkirat Behl, Jiashu Xu, Suriya Gunasekar, Neel Joshi, Yale Song, Xin Wang, Laurent Itti, and Vibhav Vineet. Neural-sim: Learning to generate training data with nerf. *arXiv preprint arXiv:2207.11368*, 2022. 2, 6

[11] Douglas Gray, Shane Brennan, and Hai Tao. Evaluating appearance models for recognition, reacquisition, and tracking. 2007. 3

[12] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *CoRR*, abs/1903.12261, 2019. 2

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 3

[14] He Huang, Philip S. Yu, and Changhu Wang. An introduction to image synthesis with generative adversarial nets. *CoRR*, abs/1803.04469, 2018. 3

[15] Sheng-Wei Huang, Che-Tsung Lin, Shu-Ping Chen, Yen-Yi Wu, Po-Hao Hsu, and Shang-Hong Lai. Auggan: Cross domain adaptation with gan-based data augmentation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 731–744, Cham, 2018. Springer International Publishing. 3

[16] Nidhi Kalra and Susan M. Paddock. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, Santa Monica, CA, 2016. 1

[17] Jingjing Li, Ke Lu, Zi Huang, Lei Zhu, and Heng Tao Shen. Transfer independently together: A generalized framework for domain adaptation. *IEEE Transactions on Cybernetics*, 49(6):2144–2155, 2019. 3

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1

[19] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S. Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming, 2019. 2

[20] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016. 1

[21] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *ArXiv*, abs/1608.02192, 2016. 3

[22] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 3

[23] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Semantic foggy scene understanding with synthetic data. *International Journal of Computer Vision*, 126(9):973–992, Sep 2018. 2

[24] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 2

[25] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1

[26] James Stout. How simulation turns one flashing yellow light into thousands of hours of experience, 2017. 1

[27] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 3

[28] J. Tremblay, T. To, and S. Birchfield. Falling things: A synthetic dataset for 3d object detection and pose estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2119–21193, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. 1

[29] Jean-Baptiste Weibel, Rainer Rohrböck, and Markus Vincze. Measuring the sim2real gap in 3d object classification for different 3d data representation. In Markus Vincze, Timothy Patten, Henrik I. Christensen, Lazaros Nalpantidis, and Ming Liu, editors, *Computer Vision Systems*, pages 107–116, Cham, 2021. Springer International Publishing. 1

[30] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2018. 2, 6

[31] Qizhe Xie, Zihang Dai, Yulun Du, Eduard H. Hovy, and Graham Neubig. Controllable invariance through adversarial feature learning. In *NIPS*, 2017. 3

[32] Greg Zaal. How to create your own hdr environment maps. *Adaptive Samples*, Mar 2021. 3

[33] Han Zhao, Remi Tachet des Combes, Kun Zhang, and Geoffrey J. Gordon. On learning invariant representation for domain adaptation. *CoRR*, abs/1901.09453, 2019. 3

[34] Berk Çalli, Aaron Walsman, Arjun Singh, Siddhartha S. Srinivasa, P. Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. *ArXiv*, abs/1502.03143, 2015. 2, 4