ARL-TR-9112 ● Nov 2020

DEVCOM ARMY RESEARCH LABORATORY

Synthetic Dataset Generation and Adaptation for Human Detection

by Jie Yan, Eung Joo Lee, Damon Conover, and Heesung Kwon

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9112 ● Nov 2020



Synthetic Dataset Generation and Adaptation for Human Detection

Jie Yan
HBCU Summer Faculty Research Fellow, Bowie State University

Eung Joo Lee
University of Maryland

Damon Conover and Heesung Kwon
Computational and Information Sciences Directorate, DEVCOM Army Research Laboratory

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*
November 2020

2. REPORT TYPE
Technical Report

3. DATES COVERED (From - To)
1 May 2020–31 August 2020

4. TITLE AND SUBTITLE
Synthetic Dataset Generation and Adaptation for Human Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jie Yan, Eung Joo Lee, Damon Conover, and Heesung Kwon

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-CI
Adelphi, MD 20783

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-9112

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Threat Reduction Agency (DTRA)
Counter WMD Technologies Department
8725 John J Kingman Rd #6201, Fort Belvoir, VA 22060

Department of Defense Research and Education Program for Historically Black Colleges and Universities and Minority-Serving Institutions (HBCU/MSI)
Department of the Army -- Materiel Command, Building 4400
Redstone Arsenal, AL 35898

10. SPONSOR/MONITOR'S ACRONYM(S)
DTRA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID(s): Eung Joo Lee, 0000-0001-6386-8470

14. ABSTRACT
In this report, we introduce an effort to use synthetic imagery to test machine learning classifiers as an alternative to large-scale unmanned aerial vehicle–based data collections. The benefits of synthetic data are that it allows for the generation of a highly diverse dataset, gives you the ability to finely control experiments, and provides automatic annotation. We have demonstrated that synthetic data can be generated using a game engine and then used to perform experiments to validate machine learning classifiers. In these experiments, we visualized the expected performance of three classifiers—Tiny-YOLO, YOLOv3, and RetinaNet—and compared the performance of the classifiers to one another.

15. SUBJECT TERMS
machine learning, classifier benchmark, synthetic data, Unity, unmanned aerial vehicle, UAV

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Damon Conover
Unclassified	Unclassified	Unclassified	UU	30	**19b. TELEPHONE NUMBER (Include area code)** (301) 394-0240

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures iv

List of Tables v

Acknowledgment vi

1. Introduction 1

2. Synthetic Data Generation 3

2.1 3-D Animated Characters 4

2.2 Animation Sequences 6

2.3 Virtual Environments 7

2.4 Synthetic Datasets 8

2.5 Water in Unity 11

2.5.1 Water Setup 11

2.5.2 Creating Water from Scratch 11

2.5.3 Water Shader Properties 12

2.6 Particle System in Unity 13

3. Analysis 13

4. Conclusions 16

5. References 17

Appendix. 3-D Models 18

List of Symbols, Abbreviations, and Acronyms 21

Distribution List 22

List of Figures

Fig. 1 (Left) Training data image with bounding box and (right) annotations associated with the image 1
Fig. 2 Civilian vehicle models purchased from Unity Asset Store 2
Fig. 3 Military vehicle models purchased from Unity Asset Store 2
Fig. 4 Models of human characters purchased from Unity Asset Store 2
Fig. 5 Rural and urban terrain models purchased from Unity Asset Store 3
Fig. 6 Camera-target relationship 3
Fig. 7 Characters from Free3d.com 5
Fig. 8 Characters from Mixamo.com 5
Fig. 9 Characters from Unity Asset Store 6
Fig. 10 Unity animation sequence 6
Fig. 11 Forest terrain downloaded from Unity Asset Store 7
Fig. 12 Forest terrain downloaded from Unity Asset Store 7
Fig. 13 Flooded terrain downloaded from Unity Asset Store 8
Fig. 14 Virtual character overlaid upon real image 8
Fig. 15 Virtual character captured at multiple poses, camera-to-character distances, and camera pitch angles 9
Fig. 16 (Left) synthetic image and (right) image segmentation mask 10
Fig. 17 (Left) training data image with bounding box and (right) annotations associated with the image 11
Fig. 18 Unity particle system 13
Fig. 19 Circular histograms showing detection accuracies for YOLOv3 with respect to the angle of the camera relative to the character at different camera heights (5, 15, 25, and 35 m) 14
Fig. 20 Detection accuracy of YOLOv3 binned by camera height and the radii of the circular paths of the camera 15
Fig. 21 Comparison of the detection accuracy of Tiny-YOLO, YOLOv3, and RetinaNet 15

List of Tables

Table A-1 Free3D models 19
Table A-2 Unity models 19

Acknowledgment

This work was supported by the Defense Threat Reduction Agency, Counter WMD Technologies Department, and the Department of Defense Research and Education Program for Historically Black Colleges and Universities and Minority-Serving Institutions under Funding Opportunity Number: W911NF-19-S-0013.

1. Introduction

Rapid and accurate recognition and tracking of objects of interest using size, weight, and power (SWaP)-constrained platforms at the edge has become an increasingly important component of technical solutions for various civilian and military applications. These solutions often use machine learning algorithms and therefore have unique challenges, such as how to obtain data and the associated annotations that can be used to train the algorithms, as shown in Fig. 1.



image: t72_20200711_105614605.png
annotations:
    category: vehicle
    label: t72
    coordinates:
        x: 215
        y: 205
        width: 430
        height: 202

Fig. 1 (Left) Training data image with bounding box and (right) annotations associated with the image

To produce accurate results, the training dataset needs to be large and diverse. For example, a typical dataset could contain thousands of images of various targets, with different backgrounds, captured at different distances, viewing angles, and orientations, and under different weather and illumination conditions. To rigorously account for all combinations of these attributes at high levels of granularity is a laborious and costly endeavor. Additionally, manually annotating each image requires additional costs. Thus, in this work, we have focused on an alternative approach that can effectively substitute for the large-scale collection of real data.

As an alternative to large-scale unmanned aerial vehicle (UAV)-based data collections, synthetic data can be generated using a game engine (Valohai 2020). We have demonstrated that this synthetic data can then be used to validate machine learning classifiers. The game engine that we used was the Unity Real-Time Development Platform (version 2019.3.5f1) (Unity 2020), and we purchased models of civilian and military vehicles, people, and terrains from the Unity Asset

Store (Unity Asset Store 2020). The models are shown in Figs. 2–5. Within Unity we created various scenes where we could select different terrains, buildings, vehicles, and people. Unity provided the flexibility to adjust the visibility and lighting within the scene, as well as add water to produce flooded areas. C# scripts were written to move the scene camera and lighting (Hayes 2020), and then extract images of the scene to be used as synthetic data.



Fig. 2 Civilian vehicle models purchased from Unity Asset Store (2020)



Fig. 3 Military vehicle models purchased from Unity Asset Store (2020)



Fig. 4 Models of human characters purchased from Unity Asset Store (2020)

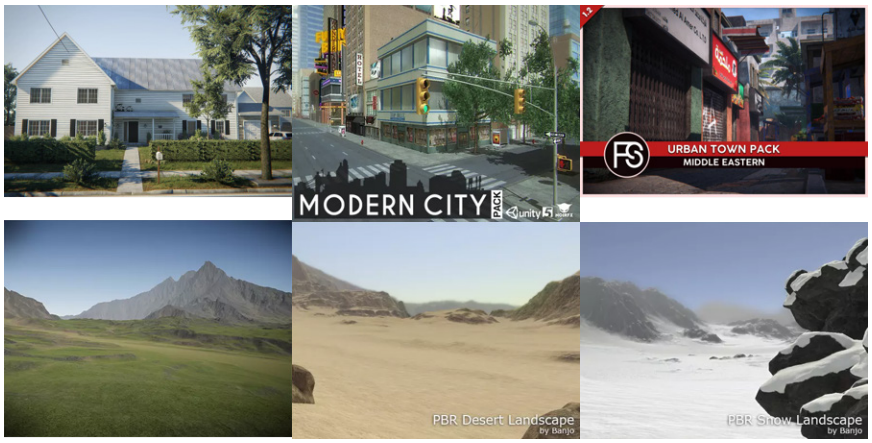


Fig. 5 Rural and urban terrain models purchased from Unity Asset Store (2020)

The synthetic data can be used to benchmark classifiers. In our initial experiments, we characterize the performance of three classifiers with different levels of architectural and model complexity, Tiny-YOLO, YOLOv3, and RetinaNet, with respect to the camera position. This allowed us to visualize the expected performance for each classifier for a given distance (d) and angle (θ) between the camera and the target, as shown in Fig. 6. Additionally, we also compared the performance of the classifiers to one another.

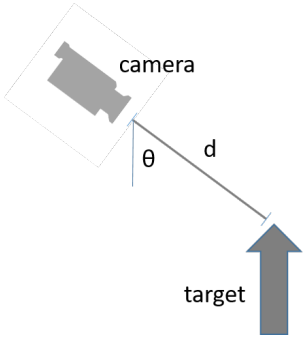


Fig. 6 Camera-target relationship

2. Synthetic Data Generation

Developing accurate machine learning classifiers requires training data that represents the expected imagery well. This presents a challenge when developing classifiers for rare events or uncommon environments. One example that we have considered is detecting people escaping a flood by standing on their roofs. This is a situation that occurs infrequently, and therefore little data exists that could be used to train a classifier; however, such a classifier would be useful for humanitarian aid and disaster recovery (HADR) scenarios. In these rare scenarios, it may be useful to be able to generate high-fidelity and highly diverse synthetic data to augment the

training set. To do this, we first need to be able to build real lifelike animated characters that look and behave like real people. This includes representing and controlling behaviors and facial expressions of animated characters to make them believable, personable, and emotional. Behaviors and expressions include facial expressions, body postures, hand gestures, gaze, and emotions. Second, we need to generate a complex real natural environment, such as deserts, forests, and mountains, as well as meteorological events, such as flooding, blizzard, hurricanes, and wildfires. Once the characters and terrains are generated, they can be imported into a game engine to generate an image dataset that can be used for machine learning model training and testing purpose.

2.1 3-D Animated Characters

We integrated approximately twenty 3-D articulated human models in the project. Each of the models was given a distinct and diverse design. Figures 7–9 show some sample models that were used in the project. They were from three sources: 1) Free3d.com (Free3D 2020), 2) Mixamo.com (Mixamo 2020), and the Unity Asset Store (Unity Asset Store 2020). Specific models used in the report are listed in the Appendix. Each source provides a collection of high-quality 3-D characters ranging from realistic to the cartoon (high-polygon to low-polygon). Each character comes fully textured and rigged so you can import immediately in your projects.

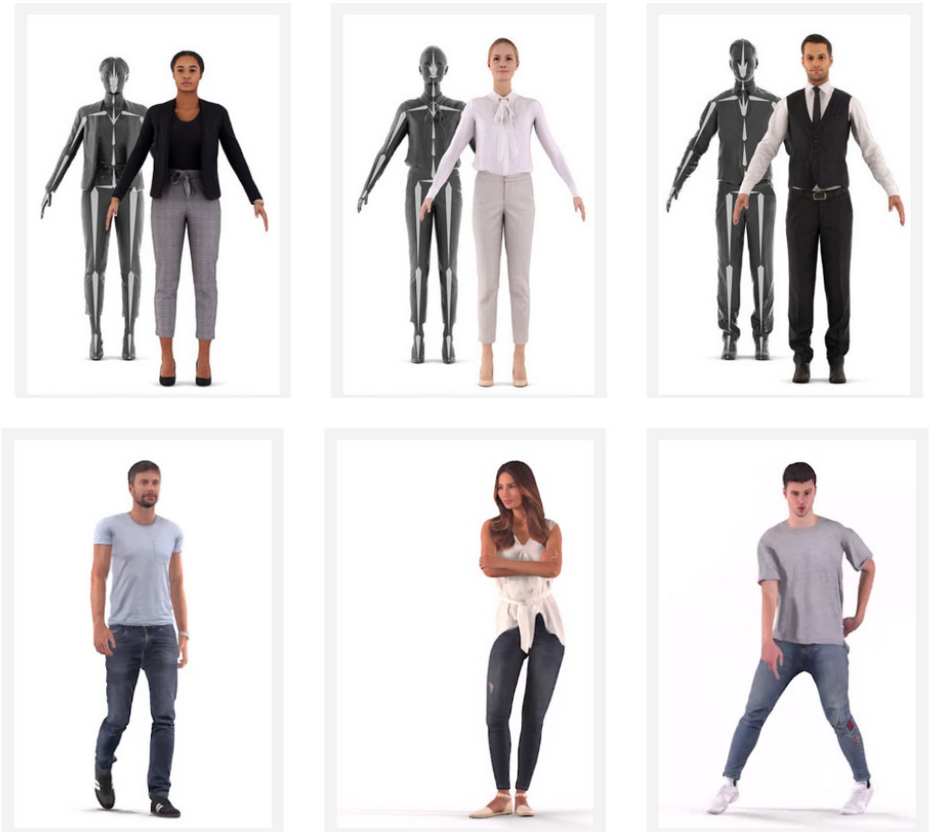


Fig. 7 Characters from Free3d.com (Free3D 2020)

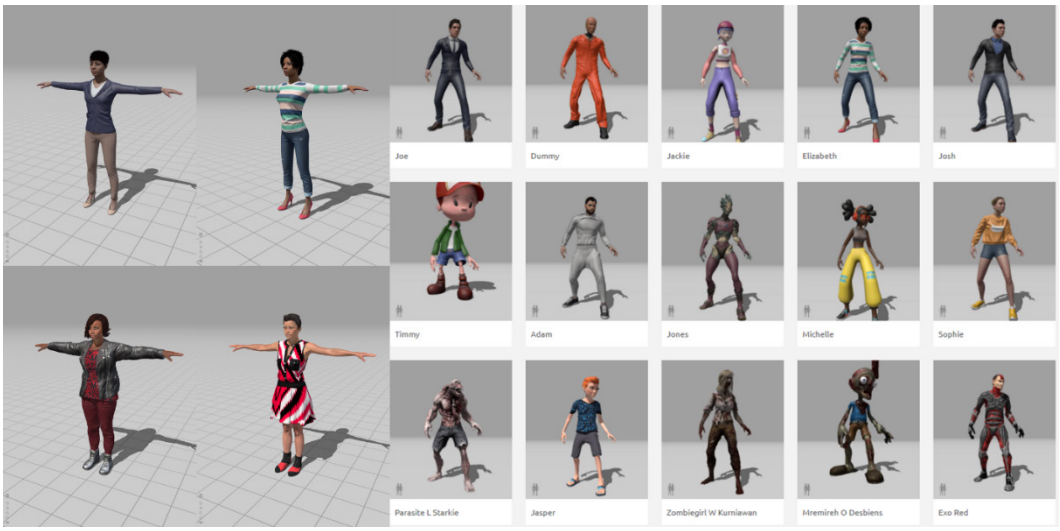


Fig. 8 Characters from Mixamo.com (Mixamo 2020)



Fig. 9 Characters from Unity Asset Store (2020)

2.2 Animation Sequences

To create a diverse synthetic dataset, the position and pose of the characters must change. For example, we wish to avoid having each character stand with straight legs with their arms at their side in every image. Instead, the dataset should include images of characters exhibiting a wide range of motions. Therefore, it is beneficial to animate the characters by implementing workflows to model different behaviors. First, in the modeling step, we produce a precise description of the character, in terms of graphics primitives. Second, in the rendering step, we generate animation sequences in Unity animator, as shown in Fig. 10.

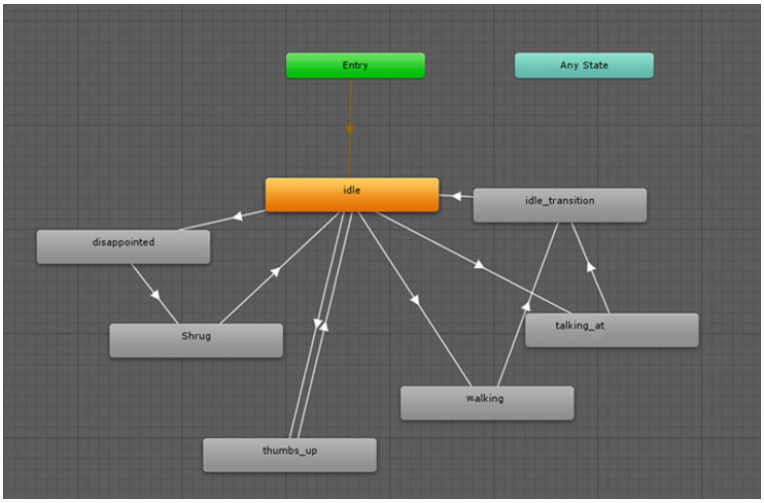


Fig. 10 Unity animation sequence

2.3 Virtual Environments

Machine learning models are often sensitive to the background; therefore, it is important that training datasets include background terrain that is similar to what is expected when the models will be used in the field. Alternatively, if there is a requirement for a general-purpose model that can be used in a wide variety of terrains, then the training dataset must also include a wide variety of backgrounds. In this project, we used four terrains: desert, forest, mountain, and water. Examples of these terrains can be seen in Figs. 11–13. We imported virtual characters into these terrains to generate the image datasets for training and testing purposes. Additionally, training images were also generated using real image data as the background, as shown in Fig. 14.



Fig. 11 Forest terrain downloaded from Unity Asset Store (2020)



Fig. 12 Forest terrain downloaded from Unity Asset Store (2020)



Fig. 13 Flooded terrain downloaded from Unity Asset Store (2020)



Fig. 14 Virtual character overlaid upon real image

2.4 Synthetic Datasets

The process of generating a synthetic dataset starts with placing a virtual character on one of the terrain assets. A C# script then controls the position of a camera as it moves in a circle above the character. The script iterates the altitude of the camera, the radius of the circle, and the angle of the camera relative to the character to produce a range of poses, camera-to-character distances, and camera pitch angles, as shown in Fig. 15. For example, in one synthetic data generation trial, for a single virtual character, the altitude of the camera was varied from 5 to 50 m in 5-m increments, the radius of the circle was varied from 5 to 30 m in 5-m increments, and the angle of the camera relative to the character was varied from 0° to 358° in 2° increments. This produced 10,800 images.



Fig. 15 Virtual character captured at multiple poses, camera-to-character distances, and camera pitch angles

The relevant code for the variations in Fig. 15 is as follows. First, the position of the character (target_asset) is defined as the center of the bounding cube that encompasses the character (position0).

```
player = GameObject.Find(target_asset);
playerTransform = player.transform;
rend = player.GetComponent<SkinnedMeshRenderer>();
bounds = rend.bounds;
position0 = bounds.center;
```

Next, the initial position of the camera (position1) is identified.

```
position1.x = position0.x + radius;
position1.y = position0.y - bounds.size.y/2 + height;
position1.z = position0.z;
```

The next position of the camera (position2) is then determined within a nested loop, which increments the size of the circle (radius), the altitude of the camera (height),

and camera position along the circle (theta). Then the previous position (position1) gets set to the current position (position2).

```
position2.x = position0.x + radius*Mathf.Cos(theta*Mathf.PI/180.0f);
position2.y = position0.y - bounds.size.y/2 + height;
position2.z = position0.z + radius*Mathf.Sin(theta*Mathf.PI/180.0f);
transform.position = Vector3.Slerp(position1, position2,
    Time.deltaTime*smooth);
position1 = position2;
```

Finally, the camera is pointed at the center of the character.

```
transform.LookAt(position0);
```

To synthesize the images from virtual scene, we used a repository called Image Synthesis for Machine Learning (U3DC 2020). The repository contains C# code for generating annotated training sets in Unity. Specifically, the code produces an image segmentation mask where each object is assigned a unique color, as well as the synthetic image, as shown in Fig. 16.



Fig. 16 (Left) synthetic image and (right) image segmentation mask

To generate the annotations, we have written a Python script to parse each mask file, identify each target, and measure the center coordinates, width, and height of the smallest rectangle that encompasses the target, as shown in Fig. 17. The results are then written to a single JavaScript Object Notation (JSON) file for each dataset.



image: data_20200714_122123325.png
annotations:
category: person
label: juliet
coordinates:
x: 258
y: 237
width: 188
height: 420

Fig. 17 (Left) training data image with bounding box and (right) annotations associated with the image

2.5 Water in Unity

Water rescue is one of the motivating scenarios for this project, so it was important for us to study Unity's capabilities for adding water to projects. Fortunately, Unity includes several water prefabs (including the necessary shaders, scripts, and art assets) within the standard assets packages. Separate daylight and nighttime water prefabs are also provided.

2.5.1 Water Setup

Place one of the existing water prefabs into your scene. Make sure you have the standard assets installed. Choose from simple or more complex water prefabs.

- Simple Daylight Simple Water and Nighttime Simple Water in Standard Assets > Water.
- Fancier water: Daylight Water and Nighttime Water in Pro Standard Assets > Water (this needs some assets from Standard Assets > Water as well). Water mode (Simple, Reflective, Refractive) can be set in the Inspector.

The prefab uses an oval-shaped mesh for the water. If you need to use a different mesh, change it in the Mesh Filter of the water GameObject.

2.5.2 Creating Water from Scratch

Simple water requires attaching a script to a plane-like mesh and using the water shader.

1) Create a mesh for the water. This should be a flat mesh, oriented horizontally. UV coordinates are not required. The water GameObject should use the water layer, which you can set in the Inspector.

2) Attach the WaterSimple script (from Standard Assets/Water/Sources) to the GameObject.

3) Use the FX/Water (simple) Shader in the Material, or tweak one of the provided water Materials (Daylight Simple Water or Nighttime Simple Water).

Reflective/refractive water requires similar steps to set up from scratch.

1) Create a mesh for the water. This should be a flat mesh, oriented horizontally. UV coordinates are not required. The water GameObject should use the water layer, which you can set in the Inspector.

2) Attach the Water script (from Pro Standard Assets/Water/Sources) to the GameObject. (Water rendering mode can be set in the Inspector: Simple, Reflective, or Refractive.)

Use the FX/Water shader in the Material, or tweak one of the provided water Materials (Daylight Water or Nighttime Water).

2.5.3 Water Shader Properties

For this project, we downloaded the water shader from Unity Asset Store, and then altered the water shader properties to accommodate the needs of the virtual scene. The water shader properties used in the project are as follows:

```
Shader "Flooded_Grounds/PBR_Water" {
	Properties {
		_Color ("Main Color", Color) = (1,1,1,1)
		_Emis("Self-Ilumination", Range(0,1)) = 0.1
		_Smth("Smoothness", Range(0,1)) = 0.9
		_Parallax ("Height", Range (0.005, 0.08)) = 0.02
		_MainTex ("Base (RGB) Gloss (A)", 2D) = "white" {}
		_BumpMap ("Normalmap", 2D) = "bump" {}
		_BumpMap2 ("Normalmap2", 2D) = "bump" {}
		_BumpLerp("Normalmap2 Blend", Range(0,1)) = 0.5
		_ParallaxMap ("Heightmap", 2D) = "black" {}
		_ScrollSpeed("Scroll Speed", float) = 0.2
		_WaveFreq("Wave Frequency", float) = 20
		_WaveHeight("Wave Height", float) = 0.1
	}
}
```

2.6 Particle System in Unity

Another motivating scenario for this project is degraded visual environments; therefore, it is useful to be able to simulate falling leaves, fire, smoke, and bubbles. To that end, Unity has a powerful and versatile particle system implementation. Figure 18 is the script interface for the Unity Particle System. The Particle System's general parameters are kept inside a special Main module. These parameters are visible in the Inspector above all the other modules.

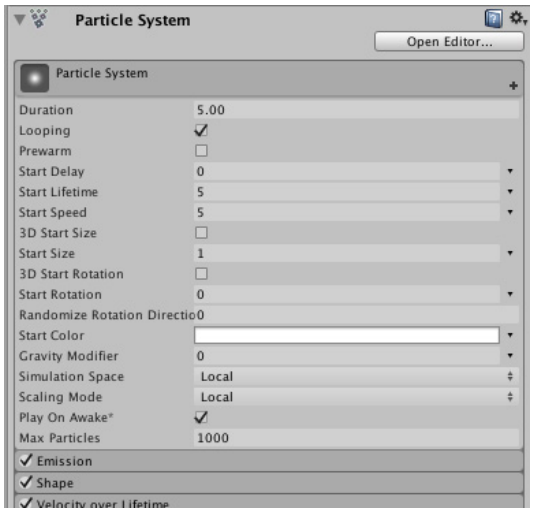


Fig. 18 Unity particle system

3. Analysis

Our primary reason for generating synthetic data was to use it to train and test machine learning classifiers. One aspect of synthetic data that is extremely difficult to replicate with real data is the fine control it allows you to impose upon an experiment. For example, a game engine allows you to vary a single variable, such as camera position, while all other variables, such as camera height/range/pitch angle, lighting conditions, and weather conditions are frozen in time. Because of this, synthetic data can be used to benchmark classifiers relative to a single variable. For example, the plots in Fig. 19 are circular histograms that illustrate the accuracy of YOLOv3 for four different experiments. For each experiment, the positions of a set of eight characters are held constant, as well as the time of day. The camera height is held at heights of 5, 15, 25, and 35 m, and it is moved in a circle with a constant radius of 5 m. The only variable that changes in each histogram is the angle of the camera relative to the character as the camera moves along the circle. An image is generated every 2°. Each histogram shows the number of positive

detections at each angle. This approach allows you to test if a trained classifier has a bias toward specific poses or camera pitch angles.

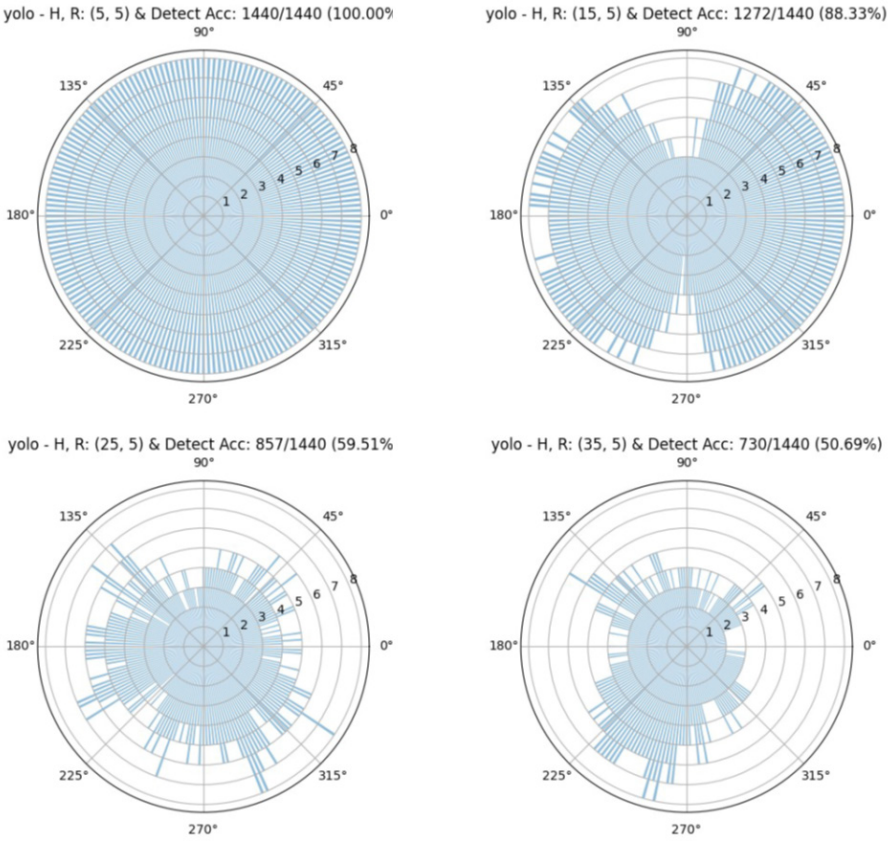


Fig. 19 Circular histograms showing detection accuracies for YOLOv3 with respect to the angle of the camera relative to the character at different camera heights (5, 15, 25, and 35 m)

Alternatively, Fig. 20 shows a plot of the detection accuracy of YOLOv3 binned by camera height and the radii of the circular paths of the camera as it moves around the characters. The plot shows a bias against images collected directly above the character. The detection accuracy increases for a given height as the camera is moved farther away. One explanation for this result is that the classifier was trained using ground imagery, so the detection accuracy increases as the images look more like ground imagery. This result may indicate that the classifier should be retrained if is going to be used on aerial imagery.

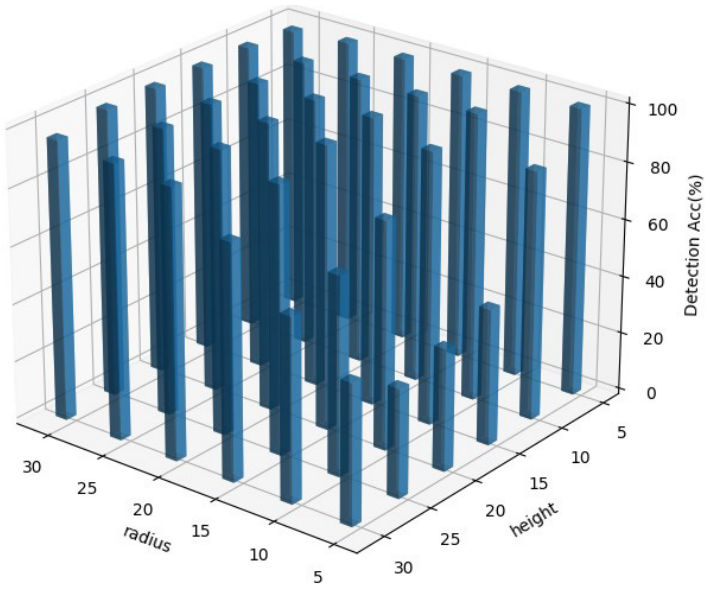


Fig. 20 Detection accuracy of YOLOv3 binned by camera height and the radii of the circular paths of the camera

Additionally, synthetic data can be used to compare the performance of different classifiers with different levels of architectural and model complexity. Figure 21 shows a plot of the detection accuracy of Tiny-YOLO, YOLOv3, and RetinaNet as camera height and the radii of the circular paths of the cameras are varied. This approach would allow someone to select the optimal classifier for a given set of mission parameters.

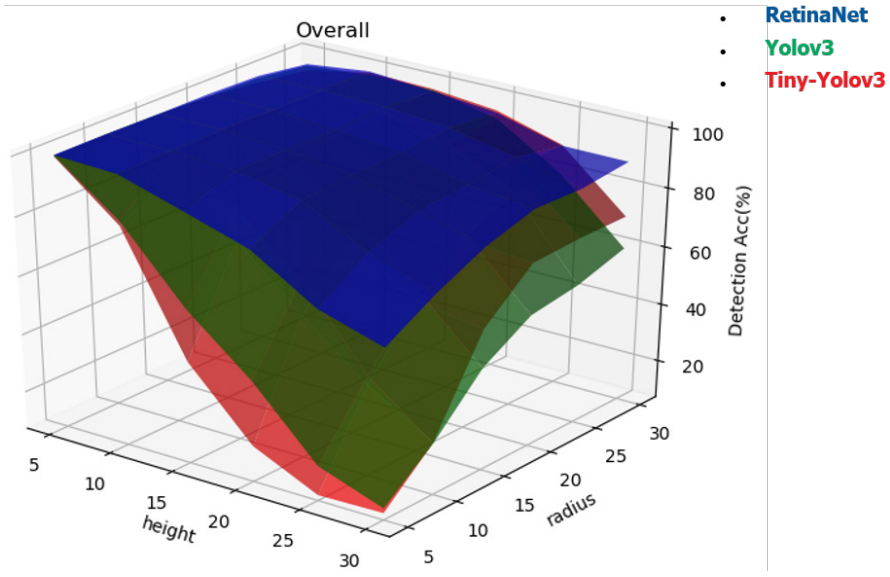


Fig. 21 Comparison of the detection accuracy of Tiny-YOLO, YOLOv3, and RetinaNet

4. Conclusions

During the summer of 2020, we performed an investigation into the use of synthetic imagery to train and test machine learning classifiers, as an alternative to large-scale UAV-based data collections. The benefits of synthetic data are that it allows for the generation of a highly diverse dataset, gives you the ability to finely control experiments (height, range, view angles, lighting/weather conditions, background types, etc.), and provides automatic annotation (bounding boxes, metadata, etc.).

We have demonstrated that synthetic data can be generated using a game engine and then used to validate machine learning classifiers. Unity provided the flexibility to adjust the visibility and lighting within the scene, add water to produce flooded areas, and degrade the visual environment with falling leaves, fire, smoke, and bubbles. In these experiments, we have characterized the performance of three classifiers with different levels of architectural and model complexity: Tiny-YOLO, YOLOv3, and RetinaNet. This allowed us to visualize the expected performance for each classifier for a given distance and angle between the camera and the target and compare the performance of the classifiers to one another. Our analysis has shown that the performance of all three classifiers is lower for imagery collected directly above the character. This may indicate that the classifier should be retrained if is going to be used on aerial imagery.

5. References

Free3D. Free 3D Models. [accessed 2020 Sep 25]. https://free3d.com.

Hayes P. Rotates a unity directional light based on location and time. [accessed 2020 Sep 25]. https://gist.github.com/paulhayes/54a7aa2ee3cccad4d37bb65977eb19e2.

Mixamo. Animate 3D characters for games, film, and more. [accessed 2020 Aug 10]. https://www.mixamo.com/#/.

U3DC. Image synthesis for machine learning. [accessed 2020 Sep 25]. https://github.com/U3DC/Image-Synthesis-for-Machine-Learning.

Unity. Unity real-time development platform 2019.3.5f1. [accessed 2020 Sep 25]. https://store.unity.com/#plans-individual.

Unity Asset Store. [accessed 2020 June 24]. https://assetstore.unity.com.

Valohai. Imgen: example Unity project for image generation for deep learning training data purposes. [accessed 2020 Sep 25]. https://github.com/valohai/Imgen.

Appendix. 3-D Models

Table A-1 Free3D models

Model name	URL
Nathan animated 003 walking 3D model	https://free3d.com/3d-model/nathan-animated-003-walking-644277.html
Sophia animated 001 idling 3D model	https://free3d.com/3d-model/sophia-animated-001-idling-130876.html
Eric rigged 001 3D model	https://free3d.com/3d-model/eric-rigged-001-771956.html
Claudia rigged 002 3D model	https://free3d.com/3d-model/claudia-rigged-002-167206.html
Manuel animated 001 dancing 3D model	https://free3d.com/3d-model/manuel-animated-001-dancing-256270.html
Carla rigged 001 3D model	https://free3d.com/3d-model/carla-rigged-001-762221.html

Table A-2 Unity models

Model name	URL
Realistic car HD 01	https://assetstore.unity.com/packages/3d/vehicles/land/realistic-car-hd-01-135301
Realistic car HD 02	https://assetstore.unity.com/packages/3d/vehicles/land/realistic-car-hd-02-99489
Realistic car HD 03	https://assetstore.unity.com/packages/3d/vehicles/land/realistic-car-hd-03-113200
Realistic car HD 04	https://assetstore.unity.com/packages/3d/vehicles/land/realistic-car-hd-04-122607
Realistic car HD 05	https://assetstore.unity.com/packages/3d/vehicles/land/realistic-car-hd-05-133027
BMP-1 tank	https://assetstore.unity.com/packages/3d/vehicles/land/bmp-1-tank-28280#description
PRP-4 tank	https://assetstore.unity.com/packages/3d/vehicles/land/prp-4-tank-30668
AS-90 tank	https://assetstore.unity.com/packages/3d/vehicles/land/as-90-tank-28956
BTR-70	https://assetstore.unity.com/packages/3d/btr-70-74502#description
Tank T-80	https://assetstore.unity.com/packages/3d/vehicles/tank-t-80-58997
Tank leopard	https://assetstore.unity.com/packages/3d/vehicles/tank-leopard-64371
Tank K-9 thunder	https://assetstore.unity.com/packages/3d/vehicles/tank-k9-thunder-58907
T-72	https://assetstore.unity.com/packages/3d/vehicles/land/t-72-3260
Customized characters	https://assetstore.unity.com/packages/3d/characters/humanoids/customized-characters-134182
PBR military (pack)	https://assetstore.unity.com/packages/3d/characters/humanoids/pbr-military-pack-118245

Table 2 Unity models (continued)

Model name	URL
PBR customized soldier	https://assetstore.unity.com/packages/3d/characters/humanoids/pbr-customized-soldier-111945
Suburb neighborhood house pack	https://assetstore.unity.com/packages/3d/environments/urban/suburb-neighborhood-house-pack-modular-72712
Modern city pack	https://assetstore.unity.com/packages/3d/environments/urban/modern-city-pack-18005
Urban town pack - middle eastern	https://assetstore.unity.com/packages/3d/environments/urban/urban-town-pack-middle-eastern-99580
PBR mountain landscape pack	https://assetstore.unity.com/packages/3d/environments/landscapes/pbr-mountain-landscape-pack-95865
PBR desert landscape	https://assetstore.unity.com/packages/3d/environments/landscapes/pbr-desert-landscape-56285
PBR snow landscape	https://assetstore.unity.com/packages/3d/environments/landscapes/pbr-snow-landscape-63394
Fantasy forest environment	https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361
Mountain terrain	https://assetstore.unity.com/packages/3d/environments/landscapes/mountain-terrain-rock-tree-97905
Flooded grounds	https://assetstore.unity.com/packages/3d/environments/flooded-grounds-48529

List of Symbols, Abbreviations, and Acronyms

DTRA	Defense Threat Reduction Agency
HADR	Humanitarian Aid and Disaster Recovery
HBCU/MSI	Historically Black Colleges and Universities/Minority Serving Institution
JSON	JavaScript Object Notation
SWaP	size, weight, and power
UAV	unmanned aerial vehicle
UV	ultraviolet
WMD	weapons of mass destruction

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
6 (PDF)	DEVCOM ARL FCDD RLC C BEDELL FCDD RLC C A LADAS FCDD RLC CI R RAO H KWON S HU D CONOVER