Synthetic Examples Improve Generalization for Rare Classes: Supplementary

1. Architecture Selection

To select a single classification architecture to use across our experiments, we trained three classifiers: ResNet-101 V2, Inception V3, and Inception-ResNet V2. All three classifiers were pretrained on *no-animal ImageNet* then trained on the Caltech Camera Traps (CCT) training set (described in the main paper, Section 3.1) with no added simulated images. We found that Inception-ResNet V2 performed best on deer in cis and trans scenarios (see Table 1), so we decided to use Inception-ResNet V2 as the base architecture for all further experiments.

	Cis Test		Trans+ Test	
Architecture	Deer	Other	Deer	Other
Resnet 101 V2	47.86	11.18	88.63	**29.76**
Inception V3	50.00	11.74	81.73	32.74
Inception Resnet V2	**29.28**	**10.17**	**77.69**	31.07

Table 1: Error for different architectures. Error is defined as the number of incorrectly identified images divided by the number of images for each test set, where "Deer" contains only deer images and "Other" contains all non-deer images.

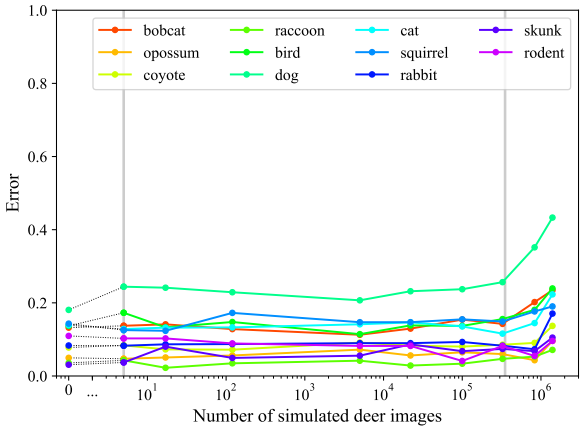
2. Additional analysis

2.1. Per-class analysis of the effect of adding simulated deer images

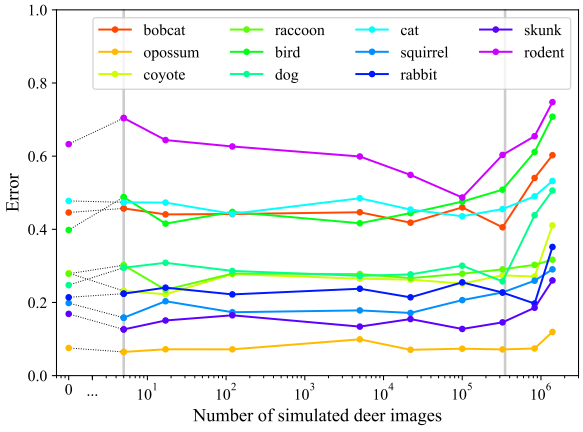
By averaging over the performance of the non-deer classes in Figure 5 in the main paper, we have not changed the overall trend. The performance on each non-deer class stays reasonably constant until the number of added deer images goes above 325K.

2.2. Analyzing the value of real images

We find that our simulated data is sufficient to learn to recognize some deer even without real examples, though the real examples give a large boost in performance. The performance breakdown can be seen in Table 2. These results are promising for both researchers studying zero-shot learning and biologists studying highly endangered species: it is possible to learn a species with no real training data. This avenue remains open for further study.



(a) Cis test



(b) Trans test

Figure 1: **Per-class performance on non-deer classes when adding simulated deer images.** The trends seen in Figure 5 in the main paper when averaging across classes hold for each individual class. Performance stays relatively constant until the number of added simulated deer images starts to bias the classifier, above 325K added images.

2.3. Comparing night and day performance

We further analyze the effect of day and night simulation by comparing three experiments: one trained with only simulated daytime images, one trained with only simulated nighttime images, and one trained with half day and half

night (see Fig 2). We find that the models trained on only day and only night perform similarly on trans deer, and that the 50/50 split performs best on trans deer (highlighted region in Fig 2). Training on day or night alone gives us a 20% performance boost on trans deer, while training on both gives us a 40% performance boost. This suggests that the day and night simulated images help the classifier in complementary ways: day helps with day images and night helps with night images. Performance on other classes is not strongly effected. Cis performance is quite noisy, and performs best with no added simulated data, see Fig. 2 in the main paper for further analysis.

	Cis Test		Trans+ Test	
Real Training Data	Deer	Other	Deer	Other
CCT train w/o deer	94.29	18.64	68.56	34.42
CCT train w/ deer	52.14	10.91	44.05	30.47
% decrease from real deer	44.7	41.5	35.7	11.5

Table 2: Error with and without the 44 real deer examples when adding 100K simulated deer images. Error is computed as in Table 1.

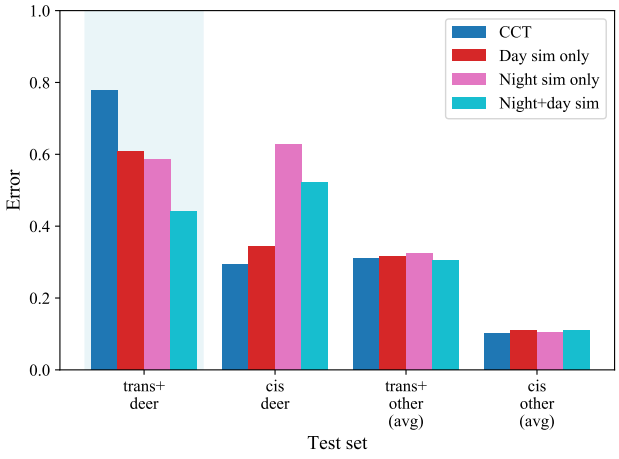


Figure 2: **Error as a function of day or night simulated images: 100K simulated deer images.** Error is calculated as in Fig. 4 in the main paper. Trans+ deer performance is highlighted. Models trained on added night- or day-only simulated data perform better on trans deer than CCT alone, but the best trans deer performance comes from the 50/50 day/night split of added simulated data.

2.4. Investigating the effect of adding simulated data for a common class

In order to investigate how added simulated data might effect a common class, as opposed to a rare one, we created "coyote" simulated data with TrapCam-Unity, using

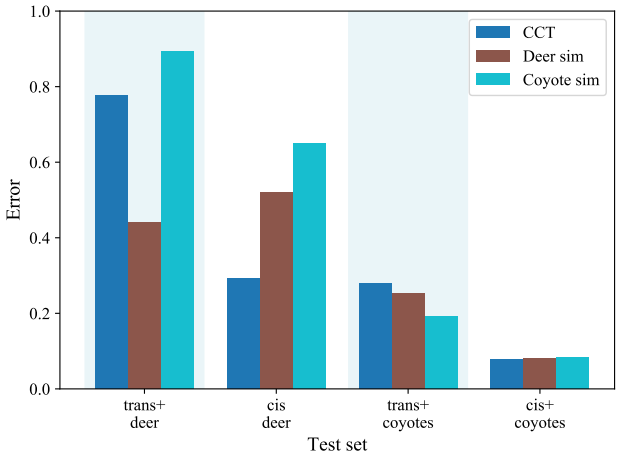


Figure 3: **Error as a function of deer or coyote simulated images: 100K simulated images.** Error is calculated as in Fig. 4 in the main paper. Trans+ deer and coyote performance are highlighted.

(a) Coyote [4] (b) Wolf [16]

Figure 4: **Wolves and coyotes are visually similar.**

rendered models of wolves as a proxy for coyotes. Off-the-shelf, high-quality wolf models were more widely available, and wolves and coyotes are visually very similar (see Fig.4). This is a coarse-grained experiment, and it remains to be seen what would happen if simulated data from two visually similar classes (*e.g.* wolves and coyotes) was added at the same time.

We find that adding simulated "coyote" data improves trans+ coyote performance slightly, while cis coyote performance remains the same. Unsurprisingly, for the deer class (which has few training examples) adding a large amount of simulated coyote data harms both cis and trans+ deer performance.

3. Creating Sim and Real on Empty Data

Alternative to the full synthetic methods of data generation with AirSim and Unity, we generated synthetic images by overlaying either simulated deer or real cropped deer on real empty background images from the CCT dataset (see Fig. 5).

For the *Sim on Empty* dataset generation, we posed either a stag or a doe deer from the GiM model set in front of a

simulated camera in Unity. We randomized the animation, orientation in azimuth (0-360 degrees), position, direction of light orientation in azimuth (0-360 degrees), and elevation (20-90 degrees).

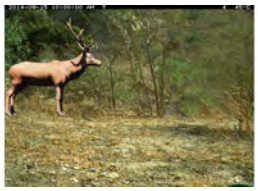
For the *Real on Empty* dataset, we manually segmented and cropped out the 44 instances of deer from the CCT training set. Then we pasted the cropped deer foreground images on top of empty camera trap images in random locations. It is worth noting that we use real empty background to investigate the effect of real versus sim foreground deer, it is possible in future work to combine either type of foreground with sim background images.

(a) Simulated deer foreground

(b) Cropped real deer foreground

(c) Empty background from CCT

(d) Empty background from CCT



(e) Sim on empty overlay

(f) Real on empty overlay

Figure 5: **Sim and Real on Empty Generation.** (a),(c),(e) demonstrate the process of overlaying a simulated deer on top of an empty background image from the CCT dataset. (b),(d),(f) show the process of overlaying a cropped real deer on top of an empty background image from the CCT dataset.

4. TrapCam-AirSim Details

It took time and thought to derive the overall requirements for the AirSim TrapCam environment. With a sizable number of potential biomes globally, we narrowed the scope of what we intended to build to a SW United States environment similar to what is seen in the CCT data. Eventually we settled on a sub-alpine woodland scene that is readily found across most of the Western/ Southwest US. A major requirement and challenge was how to get the most data out of a relatively small, but detailed, area - this was key to the project without expanding the size of the area of interest. The overall intent was to leverage Microsoft AirSim's computer vision mode to move a pre-configured camera around the scene, providing varied background.

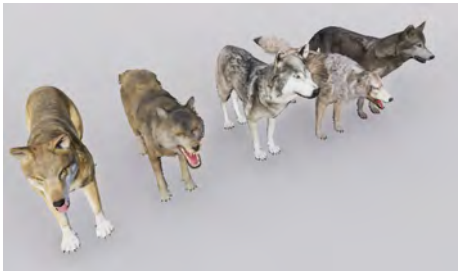
We used various off-the-shelf components such as an animal pack from Epic Studios [5] (Animals Vol 01: Forest Animals by GiM [6]), background terrain from Unreal Marketplace [14], vegetation from SpeedTree [13], and rocks/obstructions from Megascans [11]. In other AirSim environments, the general scenery is fairly static with exception of particle effects (snow/rain/dust/etc). For this effort we wanted a method to vary the background, to replicate a variety of terrains within a single environment (see Fig.6). The actual area of the environment is small, at 50 meters long, but the modularity allows many possible scenes to be constructed. The randomization was designed to facilitate artists by allowing them to make a list of different objects to randomize from. Those objects are prioritized based on their order on the list. The BiomeTerrain class generates them by tracing random areas across the field based on a global seed. If there's space available it spawns the desired object. There are a number of object types available in TrapCam-AirSim; animal type, rocks, logs, grasses, shrubs, trees, and each type can be varied by density and distribution. Additionally, we provide 9 GiM animal models: deer (doe/stag), wolf, fox, rat, spider, bear, raccoon, and buffalo. The doe model was created by removing the antlers from the stag model with Maya [9], a common modeling tool. All animal objects were assigned segmentation IDs for efficient ground truth extraction.

We created a simple UI to vary parameters, along with a command line API for parameter configuration. The UI was constructed with Unreal Motion Graphics (UMG) Widgets and allows for future flexibility for modifications, DPI resolutions and platforms. The main core functionalities were created with C++ for better performance as a parent class for data-only blueprints, which allows the technical artists to easily swap assets for different environments without recompiling the C++ code.

We started the requirements and scoping in mid-August 2018 with a go-ahead approximately 6 September, and produced a working prototype two weeks later, with continued development and refining through mid-October. A second phase late in the year modified the camera system to include flash capability, and animals were updated to provide eye-shine, and the UI was modified to include variability for that eye-shine.

Figure 6: **TrapCam-AirSim environment.** The TrapCam-Airsim envionment was designed to be modular and randomizeable, which allows a variety of biomes to be synthesized within a limited simulated area.

(a) Models of deer



(b) Models of wolves

Figure 7: **Models of deer and wolves.** In TrapCam-Unity we used 17 different models of deer from 5 different artists and 5 models of wolves from 5 different artists. We used the wolf models as proxies for coyotes (see Section 2.4). Model details are available in Section 5.

5. TrapCam-Unity Details

5.1. Simulation

The overall goal of our simulation is to take advantage of off-the-shelf components crafted for game development as much as possible so that we minimize manual labor and make the method more scalable and generalizeable. Specifically, we used off-the-shelf animal models and environment.

The "Book of The Dead" environment [3] we use is published for free by Unity. As shown in Fig.8, the near-photorealistic environment simulates a large patch of forest in a valley with volumetric grass, a variety of high definition trees, logs, and bushes, as well as rocks and terrain. The environment is a irregular area of roughly 20,000 m^2. It runs on a desktop PC in real time and enables us to generate large amounts of images efficiently.

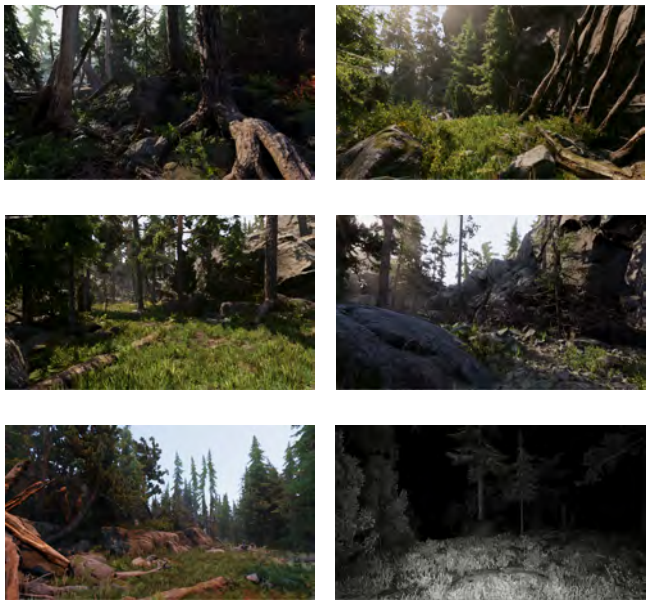


Figure 8: **TrapCam-Unity environment.** The Book of The Dead environment is a large natural environment with diverse sub regions.

To create daytime images we varied the orientation of the simulated sun in both azimuth and elevation. To create images taken at night we created a spotlight attached to the simulated camera to simulate a white-light or IR flash and qualitatively match the low color saturation of the night time images. To simulate animals' eyeshine (a result of the reflection of camera flash from the tapetum lucidum), we placed small reflective balls on top of the eyes of model animals (see Fig.9).

For deer simulation, we used 17 animated deer models from 5 publishers on Unity (GiM[7], 4toon[1],

Protofactor[10], Red Deer[12], Janpec[8]). For coyote simulation, we used 5 models from 5 publishers (GiM[7], 4toon[1], Protofactor[10], Janpec[8], WDallgraphics[15]). We created the GiM doe model by removing the antlers of the GiM stag model with Blender[2]. For each of the animated models, we included an animation controller that contains several animation clips ranging from commonly seen behavior episodes like walking and eating, to rare occurrences like attacking and sleeping. During dataset generation, we randomly picked a clip for each instance of animals and freeze it at a random time point, then we move the cameras around to sample a static scene with animals and environment.

Figure 9: **Example of eyeshine simulation.**

We had 300 seed locations and randomly placed animals in the vicinity of a subset of the seed locations. This process was repeated multiple times to simulate animals in random locations within the environment. A similar random placement process was used to determine the locations of the cameras. All images generated are in full HD resolution (1980 x 1080).

For ground truth generation, we turned off the lighting and rendered each instance of the animal in a unique color by replacing the original animal shader with an unlit shader. We then used customized python scripts to extract animal bounding boxes by extracting pixels with these unique colors.

5.2. Scalability and Generalizability

All synthetic examples in this study are generated with off the shelf environments and models. We use our simulators to generate deer images for the sake of this study, but the simulators each currently include up to 30 simulation-ready species.

A large number of high quality assets already exist online in the game development community. For example, Unity Asset Store alone has 1382 items under the Animal category. There are also many environments available online, like the "A Boy and His Kite" environment for Unreal. Despite the abundance of readily made animal models and environments, it might still remain challenging if the species-environment combination is not covered by existing assets as the 3D assets need to be created by artists first. However, recent work in automating 3D model generation [18, 20, 17, 19], might reduce the need for hand-crafted assets in the future.

References

[1] 4toon studio. https://assetstore.unity.com/publishers/3695. Accessed: 2019-03-27. 4, 5

[2] Blender. https://www.blender.org/. Accessed: 2019-03-28. 5

[3] Book of the dead environment. https://assetstore.unity.com/packages/essentials/tutorial-projects/book-of-the-dead-environment-121175. Accessed: 2019-03-27. 4

[4] Coyote in a camera trap. https://www.inaturalist.org/photos/7738216. Accessed: 2019-03-28. 2

[5] Epic studios. http://epicstudios.com/. Accessed: 2019-03-21. 3

[6] Forest animals by GiM. https://www.unrealengine.com/marketplace/en-US/animals-vol-01-forest-animals. Accessed: 2019-03-21. 3

[7] GiM studio. https://assetstore.unity.com/publishers/18347. Accessed: 2019-03-27. 4, 5

[8] Janpec. https://assetstore.unity.com/publishers/1066. Accessed: 2019-03-27. 5

[9] Maya. https://www.autodesk.com/products/maya/overview. Accessed: 2019-03-28. 3

[10] Protofactor inc. https://assetstore.unity.com/publishers/265. Accessed: 2019-03-27. 5

[11] Quixel megascans library. https://quixel.com/megascans. Accessed: 2019-03-21. 3

[12] Red deer studio. https://assetstore.unity.com/publishers/12623. Accessed: 2019-03-27. 5

[13] Speedtree. https://store.speedtree.com/. Accessed: 2019-03-21. 3

[14] Unreal game engine. https://www.unrealengine.com/en-US/what-is-unreal-engine-4. Accessed: 2019-02-05. 3

[15] Wdallgraphics studio. https://assetstore.unity.com/publishers/5060. Accessed: 2019-03-28. 5

[16] Wolf in a camera trap. https://3c1703fe8d.site.internapcdn.net/newman/csz/news/800/2018/cameratrapst.jpg. Accessed: 2019-03-28. 2

[17] T. J. Cashman and A. W. Fitzgibbon. What shape are dolphins? building 3d morphable models from 2d images. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):232–244, 2013. 5

[18] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 371–386, 2018. 5

[19] F. Pahde, M. Puscas, J. Wolff, T. Klein, N. Sebe, and M. Nabi. Low-shot learning from imaginary 3d model. *arXiv preprint arXiv:1901.01868*, 2019. 5

[20] B. Reinert, T. Ritschel, and H.-P. Seidel. Animated 3d creatures from single-view video by skeletal sketching. In *Graphics Interface*, pages 133–141, 2016. 5