Computer Vision Aided Unexploded Ordnance (UXO) Detection using Synthetic Data

Russell Craig, Joe Mercado

Unity Technologies

Orlando, FL

russell.craig@unity3d.com,
joe.mercado@unity.3d.com

Chris Kawatsu, Ben Purman

Soar Technology

Ann Arbor, MI

chris.kawatsu@soartech.com,
ben.purman@soartech.com

ABSTRACT

The objective of Computer Vision (CV) research is to equip computers to sense the environment, understand the sensed data, take appropriate actions, and learn from this experience to improve future performance. Machine Learning is the primary driver of CV and requires hundreds of thousands of data points. Collecting real world data for ML is both expensive and difficult. Also, real-world data does not have the diversity required to train ML models. As a result, access to sufficient amounts of data is the biggest challenge facing CV applications. Beyond access to data, annotating real-world data is resource-intensive, expensive, and error-prone. Training using real-world data is also challenging because datasets are often biased and insufficient, affecting our ability to validate ML models. A viable solution to solve the data problems mentioned above is synthetic data.

This paper discusses how a combination of real world and synthetic data is used to train a CV model to detect Unexploded Ordnances (UXOs) on an airfield. Specifically, this paper will illustrate the approach SoarTech and Unity took to generate synthetic data at scale and train multiple Convolutional Neural Networks (CNNs) for UXO detection. It will present data showing the CNNs' performance improvements using a "real-world only" dataset vs. a combination of 25% real world and 75% synthetic dataset. It will also discuss the use of Domain Randomization to vary lighting (time of day, brightness), building and tree arrangement, concrete asphalt and paint striping, rubble, camera parameters, and backgrounds. Lastly, this paper will discuss techniques used to avoid overfitting to one kind of data.

ABOUT THE AUTHORS

Russell Craig is a Senior Software Engineer at Unity Technologies. He has almost 10 years of professional Unity simulation experience in the areas of application development and product hardware/firmware simulation.

Joe Mercado is a Senior Product Manager at Unity Technologies. Joe has over 10 years of experience in Modeling and Simulation.

Chris Kawatsu specializes in deep learning as a Lead Scientist at Soar Technology. Since joining Soar Technology he has served as principal investigator on over $6M in research contracts.

Ben Purman is a Lead Scientist at Soar Technology. He has 20 years of experience developing computer vision and multi-sensor exploitation algorithms, phenomenology modeling and simulation, and human robot interaction.

Computer Vision Aided Unexploded Ordnance (UXO) Detection using Synthetic Data

Russell Craig, Joe Mercado

Unity Technologies

Orlando, FL

russell.craig@unity3d.com,
joe.mercado@unity.3d.com

Chris Kawatsu, Ben Purman

Soar Technology

Ann Arbor, MI

chris.kawatsu@soartech.com,
ben.purman@soartech.com

INTRODUCTION

The objective of CV is to equip computers to sense the environment, understand the sensed data, take appropriate actions, and learn from this experience to improve future performance. Over the past decade, Convolutional Neural Networks (CNNs) have greatly improved state-of-the-art performance for identifying and localizing objects in images. Early CNNs such as AlexNet (Krizhevsky et al., 2012) and VGG (Simonyan et al., 2014) significantly improved performance on image classification compared to prior approaches. More recent architectures such as ResNet (He et al., 2016) and Inception (Szegedy et al., 2017) achieved human-level performance on image classification tasks. These CNNs form the basis for object detection architectures such as Faster R-CNN (Ren et al., 2015), YOLO (Redmon et al., 2016), and SSD (Liu et al., 2016), which localize the position of objects within an image, in addition to classifying the object. Two-stage approaches such as Faster R-CNN often achieve slightly higher performance but require significantly more computation power. Since the goal of this study was to process a large amount of small unmanned aircraft system (sUAS) imagery in a short amount of time, we focus on recent single-stage approaches such as RetinaNet (Lin et al., 2017) and EfficientNet (Tan et al., 2019), which require significantly less computation power with only a slight reduction in performance. Regardless of the CV model we chose we knew we were going to have a data problem.

When rapidly assessing airfields and taxiways after a bombing attack, there are three primary challenges, especially given the thirty-minute time requirements given to Explosive Ordnance (EOD) teams. Runways and airfields are large and time-consuming to inspect, damage and Unexploded Ordnance (UXO) can be small, and EOD teams struggle to collect, process, and review assessments of EODs in thirty minutes. This paper discusses how combining real world and synthetic data can train a Computer Vision (CV) model to detect Unexploded UXOs on airfields and runways. Specifically, this paper will illustrate the approach SoarTech and Unity took to generate synthetic data at scale and train multiple Convolutional Neural Networks (CNNs) for UXO detection (see Figure 1).

Machine Learning is the primary driver of CV and requires hundreds of thousands of data points. Collecting real world data for Machine Learning (ML) is both expensive and difficult. Also, real-world data does not have the diversity required to train ML models. As a result, access to sufficient amounts of data is the biggest challenge facing CV applications. Beyond access to data, annotating real-world data is resource-intensive, expensive, and error-prone. Training using real-world data is also challenging because datasets are often biased and insufficient, affecting our ability to validate ML models. Synthetic data is emerging as an answer to solving many of these challenges. Researchers at OpenAI (Tobin et al., 2017) and Google (Hinterstoisser et al., 2019) have successfully demonstrated the efficacy of synthetic data for real-world tasks such as object detection. With advances in graphics processing and reduction in scalable computing costs, it has become possible to leverage the same tools and systems that are used to develop immersive video games and movies, to simulate photorealistic synthetic images of the real-world.

Throughout the rest of this paper we discuss how we developed a synthetic data pipeline and then conducted experiments with synthetic and real world data on a number of target deep learning model architectures and we discuss our results.

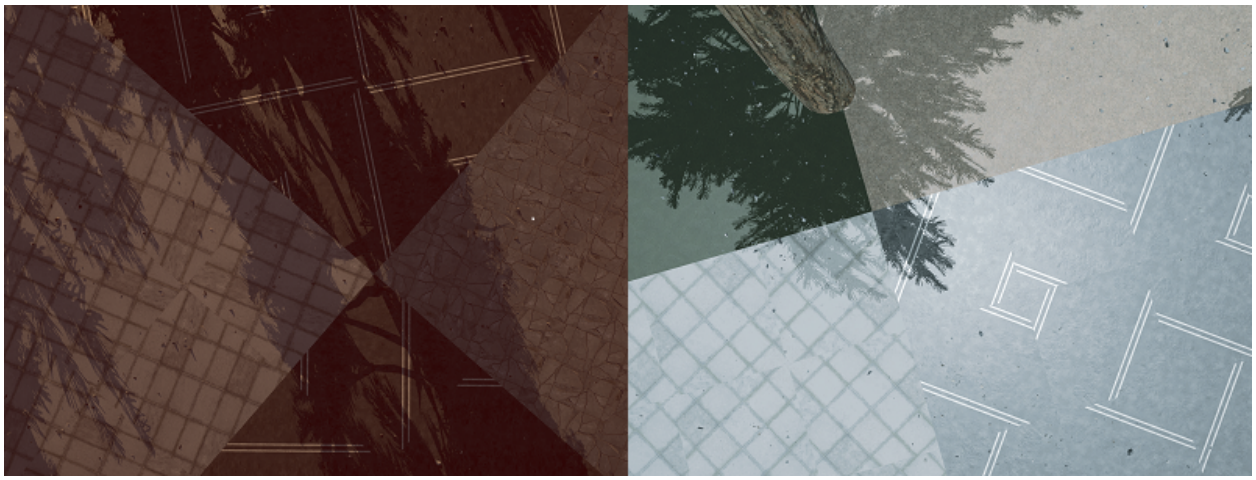

Figure 1. Example Synthetic Data Produced for This Project

METHODOLOGY

The first step our team needed to take was to setup a synthetic environment that could be used to generate large data sets. While this task could be accomplished in many of the available rendering/game engines on the market, we chose to utilize the Unity game engine as that is what the bulk of our team's experience consists of. Once the primary engine was selected, further decisions had to be made regarding how to render the environment, how to annotate the output data, and how to scale the data generation to efficiently produce large amounts of data.

By generating these large data sets and automatically annotating (labeling) the output data with 100% accuracy, this avoids the need for manual data labeling which is costly, time-consuming, and prone to human error. Producing large amounts of data helps to avoid ML model underfitting, or poor performance simply due to lack of available training data. Another benefit is this approach also helps to avoid ML model overfitting, where the model performs well on the training data sets but performs poorly on new unseen data. Overfitting is avoided by employing a technique known as Domain Randomization, which generalizes the synthetic training data by randomizing variables within given ranges such that evaluating the model against new unseen data produces similar accuracy results to the training data.

Once the data set is generated, it is then transformed and fed into the model for training. From there the model is evaluated against new data and results can be examined and validated. A benefit of using a synthetic environment setup such as this is rapid iteration can be achieved during the training and validation steps. By simply adjusting domain ranges, adding more variance to objects, and/or adjusting randomization seeds, an entirely new annotated training data set can be quickly produced. This allows for rapid hypothesis testing, answering questions such as "What happens if I position the data slightly differently? How would that impact the training?".

Universal Rendering Pipeline (URP)

The team utilized the Universal Rendering Pipeline to solve the problem of rendering the environment. We chose this because it allows for use across multiple platforms, which we leveraged when developing the application on Windows-based machines but running the application to generate datasets on cloud-based Linux instances. Also, it ensured the viability and compatibility of as many materials and post-processing effects as possible.

Perception Package

Once we selected the URP, our next interest was generating large datasets for CV model training. We selected the Perception package because it provides a ready-made toolkit for generating these large-scale datasets for computer vision training and validation. It is focused on a handful of camera-based use cases for now and will ultimately expand to other forms of sensors and machine learning tasks. The primary features of this package that were utilized

are ground-truth object labeling, domain randomization, RGB camera image capture, and object metadata capture (see Figure 2).

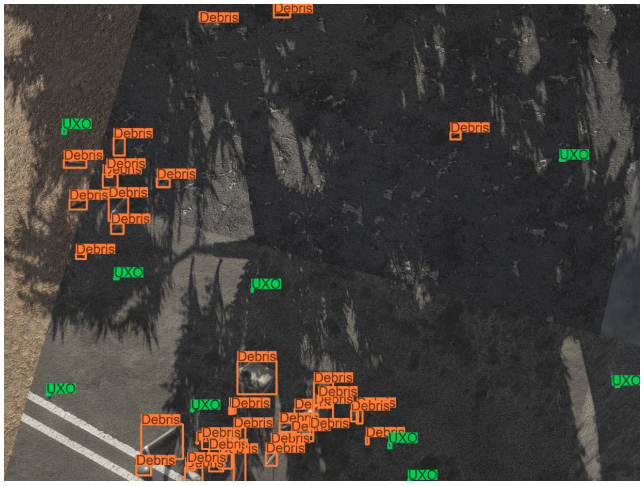

Figure 2. Perception RGB Output with Bounds

Simulation

Given the need to generate very large datasets, we needed a way to scale. Unity Simulation was chosen because it is a managed cloud service that enables product developers, researchers, and engineers to easily and efficiently run thousands of instances of parameterized Unity projects in batch in the cloud. By developing our project in Unity, parameterizing it for simulation, and running many simulated scenarios in the cloud faster than real time, our overall data generation times were greatly reduced.

DOMAIN RANDOMIZATION

Several facets of the project were broken down into groups for the randomizer to help generate accurate imagery to feed into the AI.

Camera

Our main camera was positioned over the tiles facing down at the estimated height of our drone at about 15 meters (see Figure 3). The focal length, field of view, and shutter speed were set to match the drone camera as close as possible. For each captured frame, the randomizer generated a position and rotation of the drone over the runway.

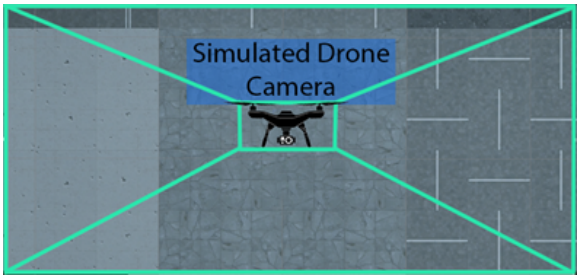

Figure 3. Drone Camera Setup

Runway/Outer tiles

The team created 20x20 meter tiles for the environment. The group of inner tiles represent the runway, which would sample a random concrete/asphalt material from our library. The outer tiles would mostly sample from grass/dirt materials, but would sometimes randomly select concrete/asphalt as well from the randomizer (see Figure 4). For each captured frame, the randomizer selected an appropriate material for each tile, and UV coordinates were offset and rotated randomly.

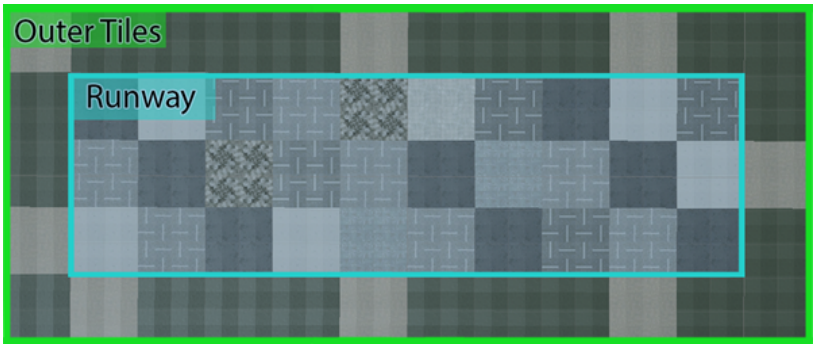

Figure 4. Runway Tile Layout

Buildings/Trees

The team created basic outlines of buildings and trees and randomly placed them on the outside tiles to generate shadows on the ground for the camera whenever its position changed. For each captured frame, the randomizer chose a random number of buildings and trees to place. Then for each object, the randomizer selected a random model from a distribution of models and placed it off of the runway in a random position, and its rotation and scale were also randomized (see Figure 5).

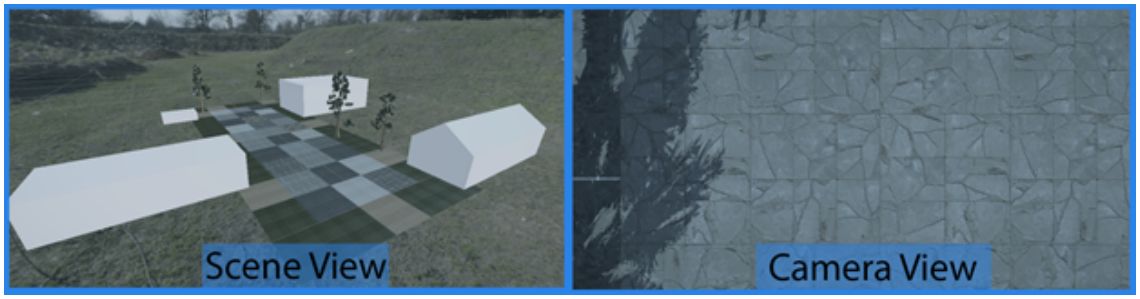

Figure 5. Buildings and Shadows

Debris/Clumps

The application used shattered primitives from Maya for the basis of most chunks found in the imagery. Much the size of the UXO with some other larger chunks for variation (see Figure 6). For each captured frame, a random number of debris to place was chosen. Then for each debris piece, a random value is checked against a threshold value to determine if this piece should generate as a clump of debris. If below the threshold, only a single debris piece would be placed. If above the threshold, a random number of debris are placed in a circular distribution around the original placement point. This mimics an explosion or impact on the runway. All debris pieces have a random runway material applied, as well as a random rotation and scale.

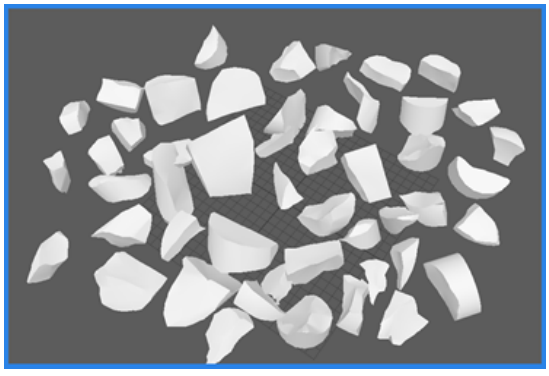

Figure 6. Debris Clumps

Unexploded Ordnance (UXO)

For the primary object to detect, the team created Dual-Purpose Improved Conventional Munition (DPICM) models in two states, one with a ribbon and the other without (see Figure 7). Subtle color variations (green/gray) were added for the randomizer. The ribbon served as a challenge. Each image created, we wanted them to have a more organic feel. First, we considered making the ribbon physics-based, but given how many images were to be generated, it increased the risk of a crash between sets. We opted to sample keyframe data from an animation created on the ribbon, and the randomizer would choose a keyframe on each new image created.

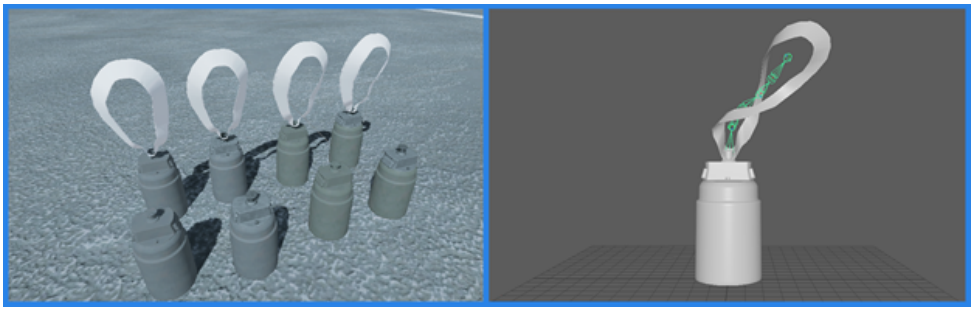

Figure 7. Unexploded Ordnance

Substance Designer

Our goal was to develop a random set of texture models in order to avoid overfitting on the ground itself. We chose Substance because it allowed the randomization we needed without having to manually create many individual textures. From one Substance material, we could generate hundreds of combinations of textures changing any number of parameters in the randomizer depending on the base material settings (see Figure 8). To facilitate easier development and iteration, every Substance material was organized in such a way that the parameters used for randomization were always contained in the same named group. For each captured frame, a random Substance material is chosen. Then, for each parameter in the named group, that parameter was randomized within minimum and maximum values.

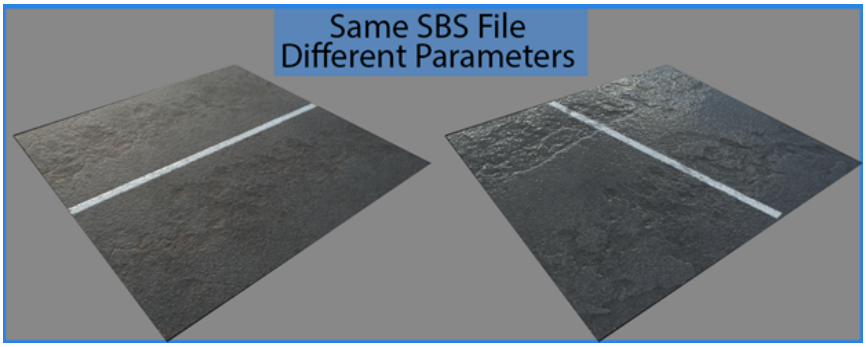

Figure 8. Substance Materials

Lighting/Skybox

The team then created positions for the directional light to coincide with the sky boxes time of day. Light intensity would change along with the time of day options. We had multiple post processing configurations/skyboxes for morning, afternoon, and evening. For each captured frame, a random time frame was chosen (see Figure 9). From there, an appropriate random skybox material was selected and rotated. The sunlight color, rotation, and intensity was then further randomized within the constraints of the chosen time frame.

Figure 9. Skyboxes for Time of Day

Machine Learning

For the machine learning architecture, we primarily experimented with specific deep learning architectures, different strategies for training with synthetic and real data, and architecture performance with increasing scene complexity. To simplify experimentation with different deep learning architectures, we used the Tensor Flow 2 Detection Model Zoo. This provided access to a number of standard implementations of high-performing architectures. To train the architectures, we explored training strategies that included different ratios of synthetic to real data, including 25% synthetic/75% real, 50% synthetic/50% real, 75% synthetic/25% real, and training on all synthetic with a fine-tuning step on real data. With respect to scene complexity, we considered a validation dataset with high similarity to the training data (grassy and parking lot backgrounds), a relatively constant background scene on asphalt runway, and a relatively cluttered concrete background. The cluttered background contained significant damage and debris that were similar in size to UXO. Figure 10 below, shows examples of synthetic training data (left) and real training data from an SUAS platform (right).

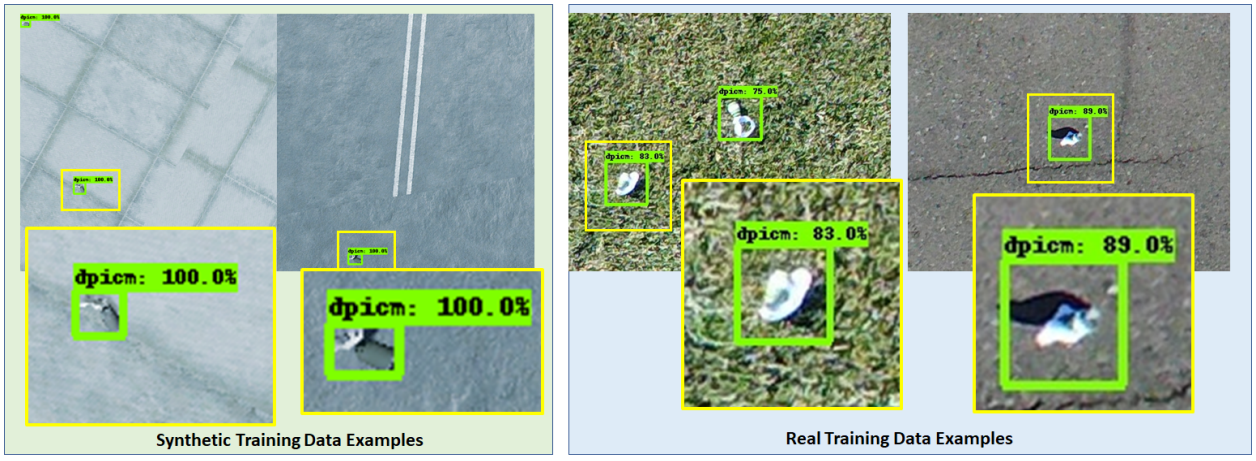

Figure 10. Synthetic vs Real Data

RESULTS

Data Generation

For each dataset, we generated roughly 10,000 images with associated metadata. Generating this on a local developer machine without cloud scaling took 4-5 hours. This generation time was due to Substance randomization and the time it took to generate 4-6 4k-resolution textures per Substance per iteration.

Running the same dataset generation scaled across 20 instances in the cloud resulted in the same data being generated in about 45 minutes, roughly 1/5th the processing time. Were we not using Substance, this time would have been significantly reduced on both fronts: local PC and cloud generation.

Our lesson learned here was twofold: If using Substances, only randomize them for every N captured frames, this greatly reduces generation time at the cost of less-randomized ground materials. Or, ideally, instead of Substance use a custom procedural texture-blending shader. This would produce slightly less variations but would greatly accelerate the data generation time.

Machine Learning

Our experiments demonstrated a significant increase in mAP and decrease in false alarm rate on realistic test datasets. We explored approximately 75 combinations of different training configurations, primarily focused on three different network architectures: RetinaNet (ResNet50), EfficientNet (B0), and CenterNet (Hourglass104). Our best results are summarized in Table 1. From a deep learning architecture perspective, we obtained the best results with EfficientNet. With respect to the mixture of real and synthetic data, we found the best performing strategy was sampling with 25% real data and 75% synthetic data. Performance decreased slightly on the validation dataset. However, we were also able to increase validation dataset performance using a different sampling of synthetic data (75% real, 25% synthetic). The results in Table 1 also illustrate the increasing difficulty of the validation, asphalt, and concrete datasets.

Dataset	**mAP: Real Only**	**False Alarm Rate @ 100% Probability of Detection**	**mAP: Real + Synthetic**	**False Alarm Rate @ 100% Probability of Detection**
Validation	***0.882 mAP***	***0.15 per image***	0.779 mAP	0.30 per image
Asphalt	0.454 mAP	1.2 per image	***0.653 mAP***	***0.50 per image***
Concrete	0.174 mAP	4.8 per image	***0.478 mAP***	***1.1 per image***

Table 1: Comparison of EfficientNet performance when trained on real data and real + synthetic data. Performance on the two realistic datasets increases significantly when synthetic data is used.

CONCLUSION

In summary, the goal of this study was to investigate the effectiveness of using a combination of real world and synthetic data to train a CV model to detect UXOs on an airfield. Specifically, we trained multiple CNNs. The results of this paper show that a combination of synthetic and real world data resulted in a more accurate CV model compared to only real world data. We believe that a key reason behind this result is because we ensured the model did not overfit by utilizing domain randomization tactics, and by providing substantial amounts of ground-truth data to augment the training of the model. While we obtained favorable results, the next steps are to understand how manipulating the mix of real world and synthetic data impacts CV model training.

REFERENCES

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097-1105.

Lin, T., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal Loss for Dense Object Detection. *arXiv preprint arXiv:1708.02002*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Sebe, N., Cohen, I., Garg, A., and Huang, T. S. (2005). *Machine learning in computer vision* (Vol. 29). Springer Science & Business Media.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 31, No. 1).

Tan, M., Le, Q. (2019, May). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv preprint arXiv:1905.11946*

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017, September). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 23-30). IEEE.