Improving AR Semantic Awareness Using Realistic Scenes in Game Engines

Brianna Butler
Computer Science B.S., Duke University
Intelligent Interactive Internet of Things (I3T) Lab @ Pratt
Durham, NC
bb274@duke.edu

***Abstract*—Using Unreal Engine's 3D game rendering work space, we generated photorealistic environments to test the YOLOv4 deep neural network. Through Unreal Engine, we were able to test the deep neural network based object recognition classifier performance under conditions found in a previous study where the classifier greatly decreased in accuracy. We utilized a scene as close as possible to being photorealistic with features such as reflections, shadows, and windows. We ran the YOLOv4 object recognition classifier on images of this scene, and then went through the training process in hopes of increasing the accuracy of the classifier in areas where it previously failed. Through the training process, we showed that the game-based Unreal Engine engine allowed us to test areas in which YOLOv4 faltered in accuracy and areas in which YOLOv4's training process faltered in efficacy.**

***Index Terms*—Deep Neural Networks, Object Recognition, Augmented Reality, Game Engine, Bounding Boxes**

I. Introduction

In the modern era of technology, mankind has progressed past amazement at simple computers, moving pictures, and holograms. These days, the latest technology of note have been the type of tech to modify our living world. Whether this technology is creating literal virtual worlds for the user or registering the real world to add artificial objects or actions, modern technology has taken a turn for the better.

This technology is not automatically perfect, nor easy to implement. In order for users to have a flawless experience with virtual reality, the world they are placed in needs to be immersive with low latency and 360 capabilities. For augmented reality to be effective for a user, there must be low latency and accurate mapping of the virtual world onto the real one. In this paper and study, we will focus on the latter.

Examples of the uses of augmented reality range from tasks as fun as catching Pokemon in your room in Pokemon Go, to as educational as training new workers in hands on jobs, to as critical as military combat battlefield assistance.

Augmented reality has a variety of recreational, purposeful, and professional uses. Augmented reality games like Pokemon Go and Harry Potter: Wizards Unite have recently provided entertainment to millions of users world wide. Self-driving cars have begun to transports more and more individuals each day. Even the military is using augmented reality technology to increase the mobility and navigation of soldiers on the battlefield.

The military is beginning to utilize tactical AR (TAR) to push the limits of what AR can do on the battlefield. The military has been using AR in combat for years, but TAR takes it to another level.

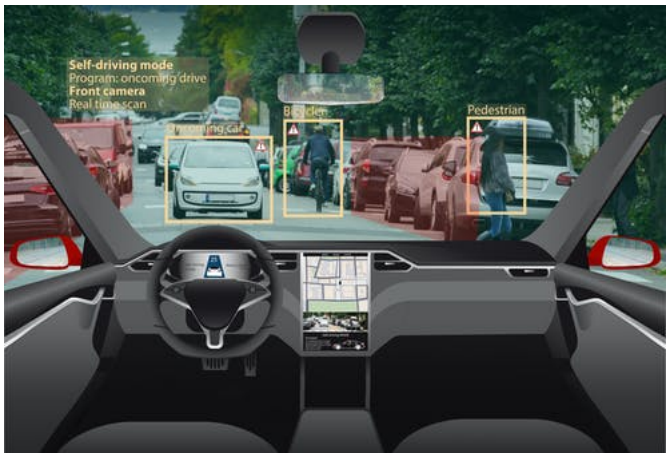

Fig. 1. Example of the object recognition in a self-driving car.

The TAR system provides high quality displays the size of a small eyepiece that relays massive amounts of critical information at once for soldiers to wear on the battlefield [21]:

- enhances soldiers' night vision
- provides shared vision between squad members
- relays topographical information to soldiers

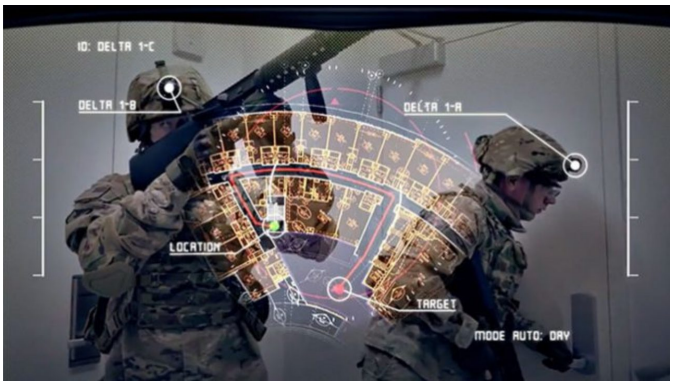

Fig. 2. Example of U.S. Military's deployment of tactical AR.

A well known example of AR technology that is currently being studied and developed is the self-driving car. The self-driving car technology was originally pioneered by Uber and recently introduced to Tesla, but Uber has recently sold their self-driving car research and technology to Aurora [6].

One of the reasons why Uber sold their technology was a lawsuit from Waymo, Another one of the reasons, and the main reason why Uber gave up their self-driving car dream was the death of Elaine Herzeberg in 2018. Herzeberg sadly lost her life in an incident where a self-driving Uber (with a driver behind the wheel) could not recognize Herzeberg on her bike as she was riding in the street at night. The system for the car identified her as an unidentifiable object and continue driving - the driver behind the wheel was on her phone, so she was unable to manually stop the car. If the car was able to recognize Herzeberg on the street, her life would have been saved [16].

Rather than using Uber's self-driving tech to make self-driving taxis, Aurora is using the billion dollars worth of research from Uber to make self-driving trucks. These trucks will not have to rely on passengers nor have multiple passengers and be used for long-haul truck driving. Aurora believes that these trucks will be easier to implement due to the predictability of the long stretch of the highway [6].

Fig. 3. Augmented reality relies on accurate object and bounding box recognition: here is an example of its labeling process.

As shown in this tragic event and past studies, augmented reality relies heavily on accurate object recognition, and when these recognition technologies fail, the AR tech involved fails as well, which can result in drastic results.

Because of the mass use of this, and how important it can be in certain dangerous situations, AR (augmented reality) needs to be accurate for users [17].

A big proportion of this study was highly motivated and inspired by [4] and [13]. In their study, they also used self-created scenes to test object recognition algorithms. This will be further discussed in the related works section of the paper.

II. Related Work

Two articles inspired the motivation and structure of our study were [4], [13], [24], [25].

In [13], they introduce the idea of apply deep neural networks to point clouds, artificial representations of 3D objects. In the paper they present a method using the PointNet architecture to map tool placement correctly in virtual reality. PointNet takes raw point clouds as input for object recognition tasks [7].

The ultimate goal of this study was to find a way to automate some of the grittiest jobs in the construction and food industries. As explained in the study, humanity has had thousands of years of evolution to advance their motor skills and movement. We have advanced it to the point that it is second nature to us. However, it is not second nature to machines. We have been able to structure machines to accomplish complex math and data acquisition, but we struggle with having machines perform tasks involving perception and motor skills, things that are second nature to us. Similar to our study, they focus on machine perception using machine learning.

The gritty job they focus on in this study is the removing of pig's ears using hydraulic scissors in slaughterhouses. This task is still performed manually due to previous failures to develop technology to automate this process. Some of these failures involved difficulty processing the variations in shape and size of pig's ears and ear orientation and position of the ears. One of these orientations included the posture of the ear, which can be erect of flopped.

Using point clouds captured by a Kinect V2, the creators of this studied created a controlled validation set and a training set for their machine to process all parts of their 3D objects.

Through this study, they were able to reveal limitations and potential further improvements for machine learning using point clouds. They also discovered that model predictions were heavily dependent on the accuracy levels of the human demonstrators. Humans are flawed and cannot repeat the same act with the same accuracy and same movement over and over again. Due to this, results significantly differed in accuracy because the difference in applications by those who the machines were modeled after made comparisons difficult.

Overall, the article found flaws with PointNet application while showcasing a lot of the system's strengths: collecting the ground truth , obtaining positional information of objects, the color of the object, etc. They also touched upon how the training process for these types of systems should become more efficient and labeling must be intuitive.

In [4] they proposed a utility function based on a 3D convolutional neural network that predicts the usefulness of future viewpoints.

They tested this method on realistic scenes - high-fidelity 3D reconstructions of buildings captured with the Matterport camera. These were detailed images with very easily recognizable and clear differences in depth. They found that their model was able to generalize and work with these real, detailed scenes.

Their data consisted of large urban environments with a good variety of structures that are commonly seen in human-made environments. They obtained these images from the 3D Street View dataset, and they chose a large scene from the Unreal Engine marketplace, which is a very similar action that we took when conducting our study. They evaluated different

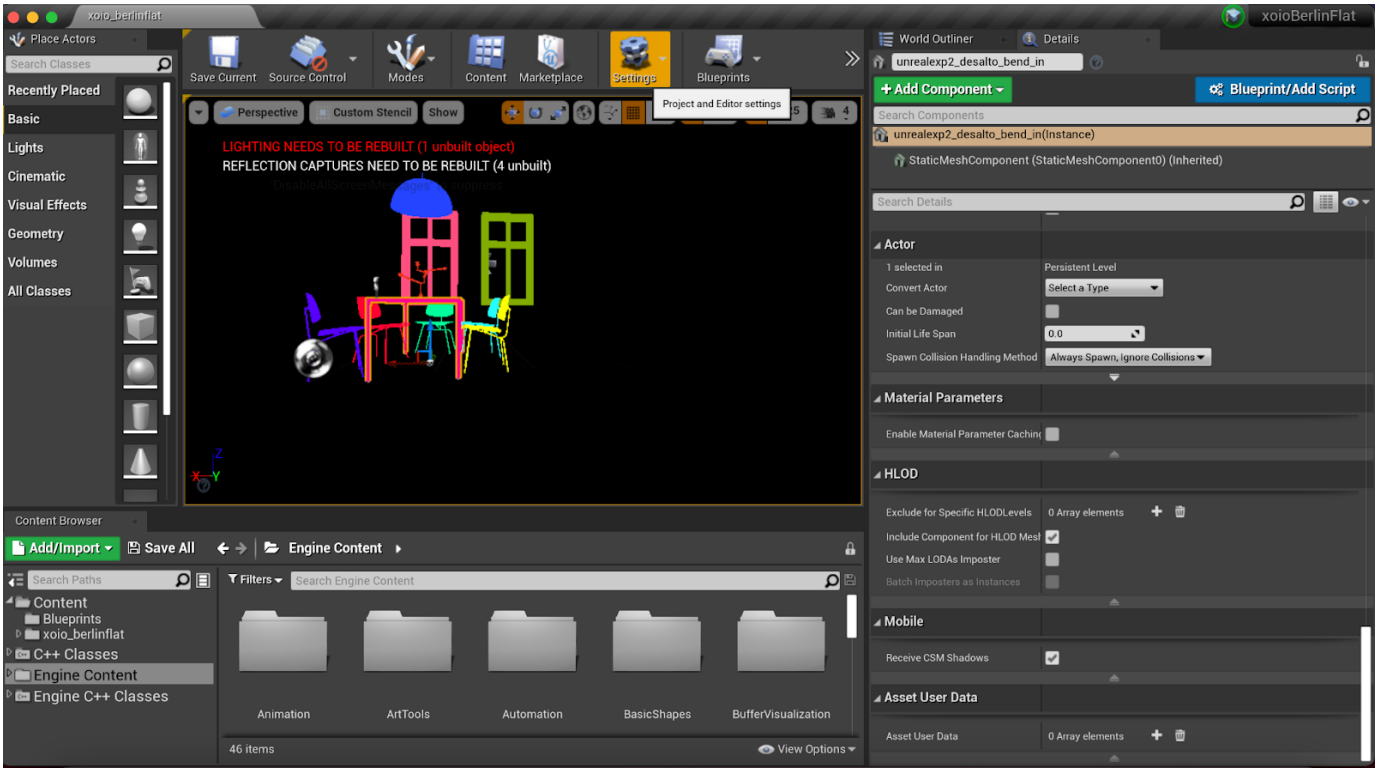

Fig. 4. Image of the Unreal Engine editor window.

ConvNet variants and tried modifications like as using residual units

In this paper, they accomplished efficient exploration of unknown 3D environments, and most importantly, were able to use their model in a way that generalizes scenes other than scenes used for the training data. According to them, they improved upon previous methods of reconstruction efficacy.

The study did have some limitations, which they mentioned in their writing. Their method depends on the surface voxel distribution in the training data. They also only tested on urban environments and wanted to try these methods with landscapes with more natural objects like rock formations and parks. Their model is also bound to the map resolution and parameters used by their training data.

In [24], the idea of using synthetic scenes to test object-recognition deep neural networks is presented. Using the Inception v3 DNN from the PyTorch model zoo and the ModernGL renderer, they constructed realistic 3D scenes and found areas where the classifier misclassified objects in the scene. The paper tested the DNN in many out-of-distribution positions, transformations, and backgrounds and identified possible biases in the datasets themselves using v3. One downside of this study was that the scenes were not photo-realistic, and more so just 3D. Their studies helped identify big holes in the accuracy of deep neural networks and further areas where these algorithms needed to be trained/adjusted.

In [25], they discuss ways to work with the Unreal Engine game engine to build photo-realistic virtual worlds. UnrealCV, the program highlighted in this study, is trained to extend the functionality of Unreal Engine to be the ultimate tool for researchers. UnrealCV has two main features that enabled the environment to be full optimized: a communication layer that gives statistical information and parameters to the researcher while the game is running, and a way to find the ground truth easily is provided as well. With these two main features, they were able to make Unreal Engine an even more powerful too to reconstruct 3D scenes, with the ability to provide information about this environment that researchers would have no way to record or find in scenes in the real world.

III. Implementation

A. Unreal Engine

In the previous semester, we developed our scene of testing in the Unity game engine editor. This time around, we ran our project in Unreal Engine, which is another one of the gaming industries leaders in game engines to create apps, mobile games, and video games. Unlike Unity, which specializes in 2D and 3D games, Unreal Engine specializes in 3D games, which results in a better output of 3D games than Unity. Rather than using C like Unity, Unreal Engine allows users to script in C++ or blueprints (which will be explained more in depth later on in the paper). Like most game engines, Unreal allows users to import assets, drag and drop them in environments and import animated scenes and objects to use within thee user's game [20].

Fig. 5. Image of the bedroom in the Xoio Berlin Flat Scene

The benefit to using a game engine for the study is the ability to manipulate nearly every feature of a scene and animate the scene in ways that are beneficial to our study. Because of this, we were able to collect data from our environments not available in scenes in the real world (like depth values) and camera movements and rotations not easily duplicated in real environments.

The main features of Unreal [18] that we utilized the most in our project were:

- UE Marketplace: The Unreal Engine Marketplace is where we were able to find and download the realistic Xoio Berlin Flat scene for out study
- Blueprint scripting: We used Unity scripts to create depth masks for our objects, enable camera rotation around a scene, and automatically obtain camera parameters.
- Unreal's Materials: Through the use of materials and the custom stencil buffer in Unreal Engine, we were able to create depth masks for the objects in our scene.

B. Scene

Similar to the scene we used in our study in Unity's game engine, we chose a premade scene in Unreal Engine's marketplace [26].

Using a premade scene allowed us to save a significant amount of time that would have been spent creating our own 3D objects, masks, environments, and placing everything in the scene. In addition to this, creating 3D objects takes a massive learning curve, especially when the objects need to be photo-realistic.

Fig. 6. Black and white, object defining view of our Xoio Berlin Flat scene.

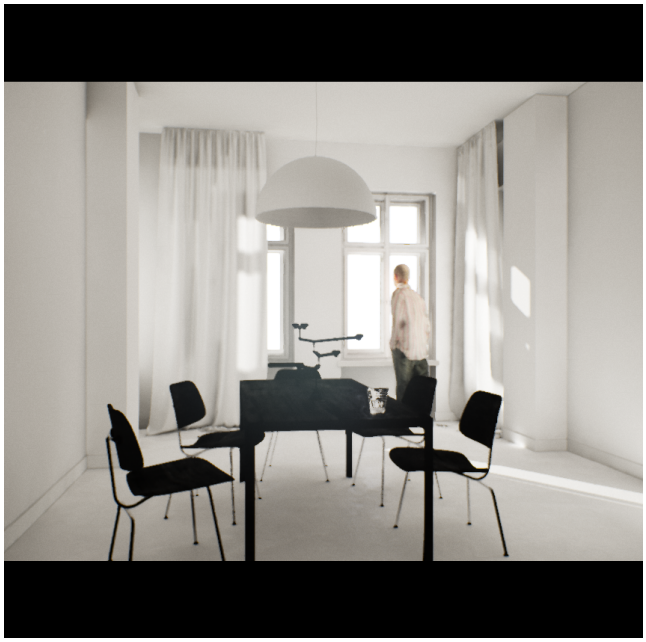

Fig. 7. Image of the bedroom in the Xoio Berlin Flat Scene

The time spent to train to make 3D objects was also cut from the time spent on conducting the study. The scene we utilized was the Xoio Berlin Flat scene by Unreal 3D artist $xoio$ from the Unreal Engine Marketplace.

There are two rooms in the scene we chose: a bedroom and a dining room. We chose to use the dining room in order to keep the scene similar to the dining room scene we used last year in our Unity project/experiment.The part of the scene we are focusing on consists of a closed room with 2 windows, 6 chairs, 1 lamp, 1 table, 1 centerpiece, and 1 cup:

- Windows: Three generic white windows are featured in the scene. Two of the windows are partially obscured by curtains. Nothing is visibly through the windows, just light.
- Centerpiece: There is an abstract centerpiece featured on the table. It resembles a black tree-like figure.
- Chairs: Black wooden-like chairs surround the table featured in the scene.
- Cup: There is a clear glass cup placed on the table.
- Table: The table is a similar material to the chairs and is placed in the center of the room.
- Lamp: Above the table is a semi-circle black lamp. No light is emitting from the lamp

Additional features of the indoor scene is a man standing in front of one of the windows, artwork, a shelf, and a large pink bug statue in the back corner of the room.

C. Scripts

Unity allows users to create scripts using UnityScript or C, but Unreal allows users to use Blueprints or C++ [19].

Blueprints are Unreal Engine's "complete gameplay scripting system based on the concept of using node-based interface

Fig. 8. Image of the scene taken by the scene camera from behind the table.

to create gameplay elements from within Unreal Editor." It can define object-oriented objects and classes as well.

Important Blueprint [5] elements of note that we used in our study are:

- Class: An asset that allows content creators to add functionality to camera or objects in their scene. The parent classes we used in our project were: actor and camera component.
- Components: These are nodes allowing different actions within your blueprint script. Examples of components we used were GetActorLocation (get current location of parent actor) and Play Tick (begin an inner clock within the game)
- Variables: As in regular computer science, variables hold values and references within your actor or object for your blueprint to use.

.

1) Screenshot Automation: In order to automate the process of taking screenshots, we decided to capture photos in a similar way as we did in the Unity study - we had the camera rotate around the center of the scene and take a photo every second the game was running [15]. For the center of the scene, we chose the centerpiece to be the object of origin, because, in addition to being the centerpiece, it was a good estimate for the center of the scene, and it allowed the camera to rotate enough that we could get every lateral angle of the scene.

In addition to taking the screenshot every second and having the camera rotate around the centerpiece in the scene, we had the parameters of the camera saved within the image title name. This allowed us to easily find out which angles/positions of the camera our trained dataset struggled with.

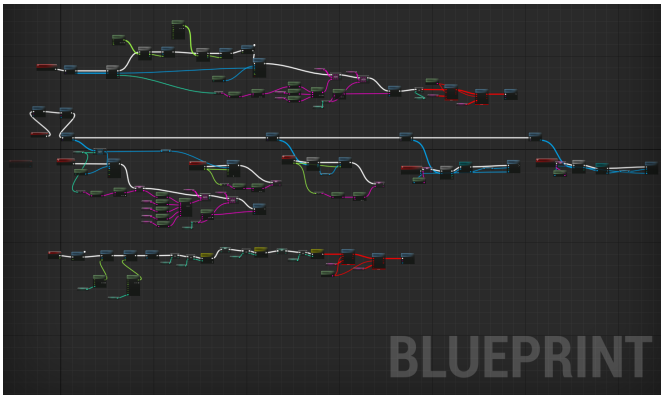

Fig. 9. Image the automatic screenshot script from Unreal Engine.

2) Depth Values: In addition to allowing the movements of the camera for taking screenshots, we utilized Unreal Engine object mask feature to apply depth values d=to different objects in the scene and create depth masks for them. We were unable to accomplish this with Unity, because it was impossible or near impossible. With Unreal Engine it was much simpler [8].

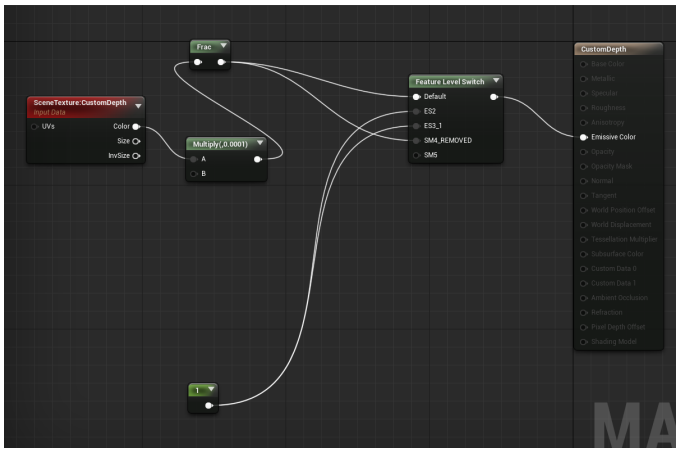

Fig. 10. Image the depth value material script from Unreal Engine.

D. Deep Neural Network

Similar to last semester's study, we centered our research on YOLOv4. We chose YOLOv4 to continue work with the deep neural network, but we chose YOLOv4 last semester due to its reputation back than and now. Rather than see how accurate YOLOv4 was this semester, which we found with plenty of examples of inaccuracy, we wanted to see if this training would help solve these extreme deficiencies we found last semester.

YYOLOv4 is much faster than its predecessor YOLOv3, can predict multiple objects at a time and draw bounding boxes around the objects it detected during its image inspection process [1]. YOLOv5 is also available, but many users have found it problematic, so stuck with YOLOv4 [10].

E. Training Process

A key feature of YOLOv4 is that it can be trained to detect custom classes as well [1]. For the training process, we followed Jacob Solawetz's tutorial for training YOLOv4 on a custom dataset. the tutorial also had us use Google CoLab, which is where we conducted a large part of training. CoLab included a built-in GPU and built in NVIDIA features needed for the study. Having a Macbook pro slightly hindered the process, as many features of the projects are not innately compatible with apple products, but we found ways to circumvent these issues (for example, CoLab).

We had to train YOLOv4 with the Darknet framework, because training with TensorFlow, Keras, and PyTorch is still under construction [2]. Last summer, we used the Tensorflow conversion of YOLOv4 because we were only testing the object recognition accuracy, not training. Darknet is a custom framework created by Josephn Redmon, which is known to not be intuitive but very flexible in its uses.

The training process consisted of these main steps, directly from the Solawetz tutorial [11]:

1) We configured our GPU environment for YOLOv4 on Google CoLab
2) We installed the YOLOv4 training environment in CoLab
3) We downloaded our custom YOLOv4 and set up directories (more of the process of making this dataset is included late in the paper)
4) We configured a custom YOLOv4 training config file for Darknet in CoLab
5) We trained our custom YOLOv4 Object Detector
6) Finally, to test our new weights, we used our custom YOLOv4 detector for inference

.

F. Bounding Boxes

In order to create bounding boxes, we had to upload all of our images captured automatically from Unreal Engine to a folder on our local machine, and then to Roboflow, a very popular site to create custom machine learning models on [11].

For our first training process, we annotated 100 images within the dataset of 245 images. With these images, we included drawing bounding boxes around occluded and partially visible objects.

For our second training process, we annotated another 100 images within the dataset of 245 images. With these photos we only drew bounding boxes around non-occluded and non-partially visible objects.

IV. Experiments and Results

The total dataset of images consisted of 245 images.

Before we ran the official experiments, we assigned depth values to each item in the scene in order to better analyze our results.

Here are the depth values and colors assigned to the objects in the scene (in regards to the photo above):

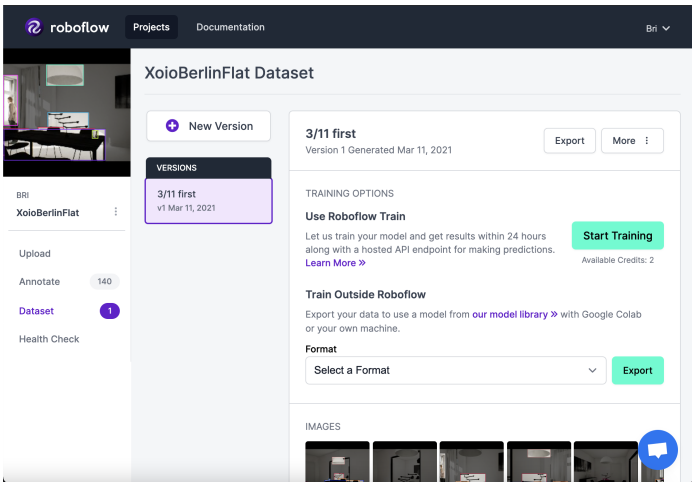

Fig. 11. The Roboflow environment where we created our custom dataset for YOLOv4 training.

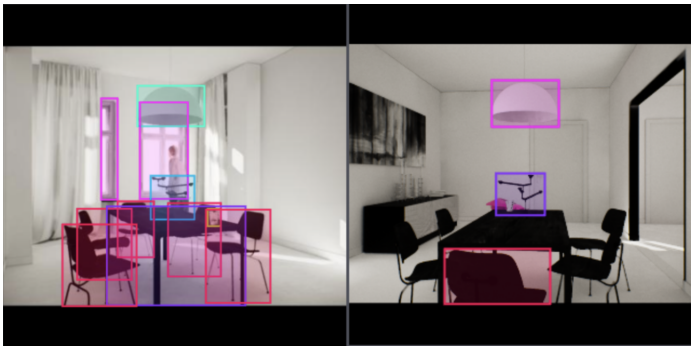

Fig. 12. Left: Bounding boxes drawn including occluded objects. Right: Bounding boxes drawn without occluded objects included.

Fig. 13. Left: Bounding boxes drawn including occluded objects. Right: Bounding boxes drawn without occluded objects included.

1) Front Right Chair (Yellow)
2) Back Right Chair (Light Blue)
3) Back Chair (Light Green)
4) Back Left Chair (Red)
5) Front Left Chair (Purple)
6) Centerpiece (Dark Pink)
7) Left Window (Pink)
8) Right Window (Forest Green)
9) Lamp (Blue)

The only item not included in these depth values is the cup. This is because the cup is a non-physical object in the scene that is transfixed to the table.

We ran two experiments in total: testing the weights including occluding bounding boxes during training on our dataset and testing the weights not including occluded boxes during training on our dataset.

A. Training Trial 1

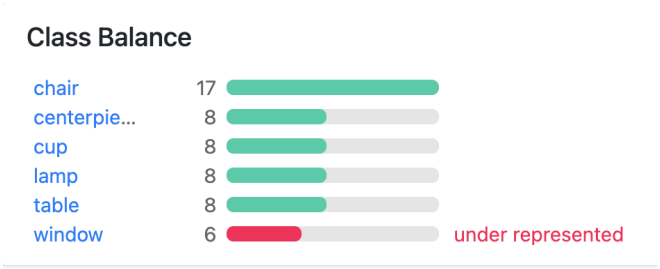

Fig. 14. The distribution of item labels of bounding boxes for the dataset.

As mentioned before, in this first training trial, we used bounding boxes that overlapped with one another.

Rather than recording the accuracy of each bounding boxes, we recorded results of running the training weights on images and recorded observations we made with correct labels and incorrect labels.

With the first training trial, we had very few correct guesses for the objects in the scene, however we did notice trends with the results we did find within the incorrect guesses.

In the first trial, the weights had these tendencies:

- The entire table and chair area was labeled as the table, but more specifically, multiple tables.
- The lamp was commonly mislabeled as the cup.
- The centerpiece was commonly labeled as a chair.
- The windows, centerpiece, and lamp labels never appeared in any of the images.

B. Training Trial 2

For our second trial of training the YOLOv4 weights, we only used training images that contained at least one object that did not overlap with another. These images were very limited. The objects were very limited as well.

There is extreme over representation of the cup object because the cup was one of the very few items in the dataset that was never obscured/overlapping with another object.

Fig. 15. Examples of the results we got from trial 1

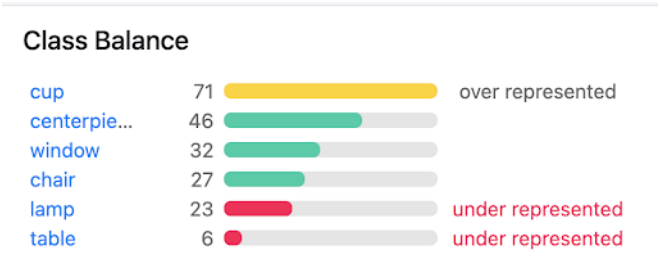

Fig. 16. The distribution of item labels of bounding boxes for the (2nd) dataset.

There is extreme under representation of the table because of the way the scene is organized - most of the chairs are in/under the table, so isolating the table in a bounding box with no overlap was impossible - there were only 6 images in which the table had few overlap, but there was no table with absolutely no overlap.

Trial 1 had a much more even distribution due to the inclusion of items with overlaps.

In the second trial, the weights had these tendencies:

- The lamp was commonly labeled as the cup.
- The cup was commonly labeled as the window.
- The centerpiece was commonly labeled as a chair.
- The centerpiece, table, and lamp labels never appeared in any of the images.

Similar to training trial 1, the guesses for the identities of the objects was still inaccurate, but it was inaccurate in a different way

In addition to this difference this trial also differed in that it had a few correct guesses unlike the first trial.

The trial 2 training set was able to identify chairs, but only when they were relatively isolated from other items in the scene. As you can see above the algorithm was correct when the chairs in the bedroom were visible from the dining room – these chairs are isolated, non-occluded, and very recognizable as chairs.

Fig. 17. Examples of the results we got from trial 2

Fig. 18. Example of when the 2nd trial weights were correct.

V. Conclusions

Overall, we found that YOLOv4, before and after the training process can be very flawed in the object detection process for objects that are varied even slightly differently than normal. Although our dataset was relatively small, the inability of YOLOv4 to even recognize the cup, even when it was highly represented in the training dataset for trial 2, was unexpected. In addition to this, YOLOv4 still continues to struggle identifying glass objects like windows or the cup.

In examples shown by many of the blogs we followed to learn the training process, the training dataset featured 2D objects or very limited angles of objects in the training set. For our training set, we included full 360 representations of the objects and full 360 representations of the objects with occlusions and overlaps. This heavily mirrors the findings we had in our Unity study: YOLOv4 struggles to recognize objects in positions that show the objects in rotations or angles that don't full represent the "obvious" form of the object.

For example, with the second training set and trial, we found that our custom weights for YOLOv4 could recognize the chairs in the back corner of the bedroom, but those chairs are also non-occluded, and they are fairly obvious they are chairs. Because the chairs, the table, and the centerpiece are the same color, in the first trial, our custom weights consistently labeled any legged dark object as a table. This is understandable, because in the dining room scene, the collective grouping of table and chairs looks like a blocky collection of legs, and including the occluding bounding boxes probably did not make it easier for the algorithm to make a clear decision/identification.

In addition to these findings, and interesting finding of note that we had was the situation with the glass cup and the glass windows. In our Unity study, YOLOv4 failed to recognize any windows. Even though we included windows in our training set for the first and second trials, YOLOv4 failed to correctly identify the windows. This was surprising because in both training trials, the windows were the two objects that were probably occluded the least, so it was expected the windows would be recognized easily. As mentioned before, this is also the case with the cup in the second trial. Even though the cup was the least occluded and most labeled object for the second dataset, YOLOv4 failed to recognize it at all. However, it did recognize the cup as a window (in the second trial). This causes one to think about YOLOv4 and its tendencies with glass/translucent objects.

A. Shortcomings

The YOLOv4 training process, as mentioned before, is not very intuitive. As a result, many training tutorials are open-ended and are only understandable by those who have prior experience with the deep neural network or those who have prior experience with neural networks overall [1].

```
data/darknettest.png: Predicted in 7664.972000 milli-seconds.
bicycle: 28%
bicycle: 32%
bicycle: 31%
motorbike: 43%
bicycle: 37%
bicycle: 43%
bicycle: 41%
bicycle: 45%
```

Fig. 19. Example of the labeling error we encountered: YOLOv4 only guessed bicycle or motorbike for our objects due to unnecessary labels hidden in the darknet source folder.

A example of the lack of intuitiveness of YOLOv4's training process hindering our study is the lack of information about which files to change. By following most of the tutorials verbatim, you will still run into the issue of having labels show up inn your results that were never in your training set. We were able to circumvent this issue by combining through every file in the darknet folder to eliminate unnecessary labels that we could find. None of the tutorials we found mentioned that we would need to do this, and we had to comb the internet to find how to fix this issues - which was one many of those who wanted to train custom weights ran into, showing a deficiency in tutorials.

As shown in many of the blogs we followed to facilitate the training process and construct the training weights, the

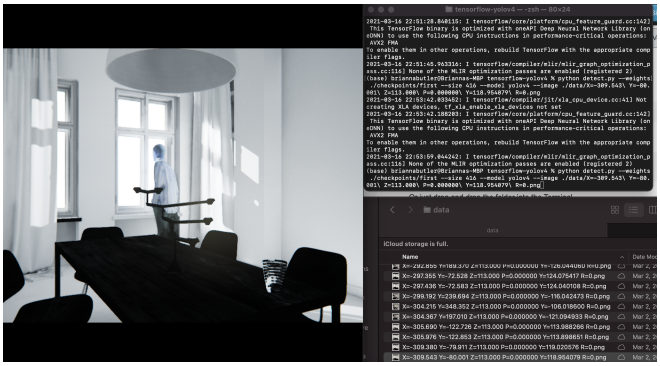

Fig. 20. Another example of an obstacle we faced in the training process: the tutorial failed to mention further file changes necessary to correctly load each picture in the source image.

training dataset was much larger than ours. Many training sets online included more than 500 images, which we were not able to reach due to time and technical limitations. One of these technical limitations was the actual time to run the training process in Co-Lab. Our dataset of 100 labeled images took at least 2 hours for each trial. A dataset of 500 images would take easily double the amount of time [11].

Another limitation we faced was with the version control of Unreal Engine. Many of the scripts and features of Unreal Engine we wanted to replicate from a past study were rendered obsolete or removed by new updates to the software. Because of this, basic tutorials online became somewhat of a hassle to follow. We had to go through each step, and then convert each step to the modern version of Unreal Editor [20].

Another shortcoming we came across was the running power of Unreal Engine. Unreal Engine is a powerful game engine, so it requires a powerful machine to run it smoothly [18]. A laptop, which is what we were using, was a relatively low-powered machine to run such a high CPU-using software. This would cause delays in the running of Unreal Engine, loading the whole scene in the engine, and the running of the game in play mode, which was the mode we needed to run in order to run our automatic screenshot procedure.

Our final limitation was the difference in difficulty of scripts in Unreal Editor vs Unity. Although Unreal is much more powerful than Unity in generating 3D scenes and it is much easier to add depth masks to objects in this engine, the overall scripting process in Unity was far easier than the scripting process in Unreal Engine. Unreal utilizes blueprints, which one would expect to be easier to code with than actual code (C for Unity), but for what took a simple method in Unity took massive amounts of blocks in blue prints [19].

A prime example of this is rotating the camera around in Unity versus Unreal Engine. As shown above the scripting process of making the camera rotate around the scene was quite complicated for Unreal Engine, while it took only defining a single method in Unity.

B. Discussion and Future Directions

It would be fairly interesting to attempt these trials again with these three factors changed from our study:

- Larger training dataset (more than 500 images)
- Table and chairs that are not the same color and material
- More focus/another section focused on the identification of glass objects with YOLOv4

It would be extremely interesting, and beneficial to understanding the efficacy of YOLOv4 to attempt future studies with the occlusion/non-occlusion format of experiment. With future work with this format of study, it will help the improvement of YOLOv4 and deep neural networks to identify gaps in efficacy between these two forms of training sets. Our study's datasets were far too small, so a future study with larger datasets would be extremely informational about training and YOLOv4.

Fig. 21. An image of our original scene in the Unity Editor from last year: Here, each object is very different from each other in texture and color, making it very easy to differentiate them from each other.

In our Unity experiment from last semester, we found that YOLOv4 had an easy time identifying objects that were spread out from another, and it had an easy time recognizing chairs and a table, that were very different materials from each other. The chairs were green and the table was brown, which could be beneficial for a future study on the training process. The YOLOv4 training process may have had difficulty with our scene due to the chairs and table being relatively identical compared to our previous scene with chairs that were drastically different than the table.

The phenomenon of YOLOv4 struggling with windows and glass objects is one that should definitely be explored. Specific details of glass like translucency, glare, color, and texture of glass would be interesting factors to examine in a future study with training and YOLOv4. It would also be beneficial to examine these factors with the default weights of YOLOv4, to see whether it is a training issue or an issue with the deep neural network itself [16].

This is an excellent tie-in to our overall examination of deep neural networks and what these findings mean for object recognition and augmented reality overall. The complexity

of the training process could be affecting the efficacy of current deep neural networks and their use in augmented systems. However, as AR and deep neural networks continue to advance, the process will become easier and easier to replicate and conduct.

However, as mentioned in our previous study, the issue of YOLOv4 struggling with identifying objects rotated around can be slightly concerning for large (and potentially dangerous) uses of augmented reality. In the case of individuals who have lost their lives in self-driving cars because off system failure due to out-of distribution objects, it is owed to them, and future users of these products to find ways to conquer this problem, whether it is with the training process or the testing process.

REFERENCES

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020.

[2] AlexeyAB, "AlexeyAB/darknet," Darknet. [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed: 11-Nov-2020].

[3] "Augmented Reality," Information Management and Decision Architectures. [Online]. Available: https://www.nrl.navy.mil/itd/imda/research/5581/augmented-reality.

[4] B. Hepp, D. Dey, S. N. Sinha, A. Kapoor, N. Joshi, and O. Hilliges, "Learn-to-Score: Efficient 3D Scene Exploration by Predicting View Utility," Computer Vision – ECCV 2018, pp. 455–472, 2018.

[5] "Blueprint Visual Scripting," Unreal Engine Documentation. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html. [Accessed: 19-Apr-2021].

[6] C. Metz and K. Conger, "Uber, After Years of Trying, Is Handing Off Its Self-Driving Car Project," The New York Times, 07-Dec-2020. [Online]. Available: https://www.nytimes.com/2020/12/07/technology/uber-self-driving-car-project.html. [Accessed: 22-Apr-2021].

[7] C. R. Qi, H. Su, K. Mo, and L. Guibas, PointNet, 2017. [Online]. Available: http://stanford.edu/ rqi/pointnet/. [Accessed: 22-Apr-2021].

[8] How to Create Masks With the Custom Stencil Buffer — Tips amp; Tricks — Unreal Engine. YouTube, 2020.

[9] J. Choi, H. J. Park, J. Paek, R. K. Balan, and J. G. Ko, "LPGL: Low-power graphics library for mobile AR headsets," Yonsei University, 12-Jun-2019. [Online]. Available: https://yonsei.pure.elsevier.com/en/publications/lpgl-low-power-graphics-library-for-mobile-ar-headsets. [Accessed: 14-Nov-2020].

[10] J. Nelson, "Responding to the Controversy about YOLOv5," Roboflow Blog, 05-Oct-2020. [Online]. Available: https://blog.roboflow.com/yolov4-versus-yolov5/. [Accessed: 15-Nov-2020].

[11] J. Solawetz, "How to Train YOLOv4 on a Custom Dataset," Roboflow Blog, 08-Mar-2021. [Online]. Available: https://blog.roboflow.com/training-yolov4-on-a-custom-dataset/. [Accessed: 19-Apr-2021].

[12] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-time Object Detection for Mobile Augmented Reality," The 25th Annual International Conference on Mobile Computing and Networking, 2019.

[13] M. P. Philipsen and T. B. Moeslund, "Cutting pose prediction from point clouds," Sensors, vol. 20, no. 6, p. 1563, 2020.

[14] Paul Gestwicki. 2019. Unreal engine 4 for computer scientists. J. Comput. Sci. Coll. 35, 5 (October 2019), 109–110.

[15] R. Böhringer, "Postprocessing with the Depth Texture," Ronja's Shader Tutorials, 30-Jun-2018. [Online]. Available: https://www.ronja-tutorials.com/2018/07/01/postprocessing-depth.html. [Accessed: 15-Nov-2020].

[16] R. Jones, "Using physically-based lighting values with Enlighten and UE4: Enlighten - Real Time Global Illumination Solution: Silicon Studio," Using physically-based lighting values with Enlighten and UE4 — Enlighten - Real Time Global Illumination Solution — Silicon Studio, 22-Mar-2019. [Online]. Available: https://www.siliconstudio.co.jp/middleware/enlighten/en/blog/2019/20190322/. [Accessed: 11-Nov-2020].

[17] R. Randazzo, "Who was really at fault in fatal Uber crash? Here's the whole story," The Arizona Republic, 18-Mar-2019. [Online]. Available: https://www.azcentral.com/story/news/local/tempe/2019/03/17/one-year-after-self-driving-uber-rafaela-vasquez-behind-wheel-crash-death-elaine-herzberg-tempe/1296676002/. [Accessed: 09-Nov-2020].

[18] "The most powerful real-time 3D creation platform," Unreal Engine. [Online]. Available: https://www.unrealengine.com/en-US/. [Accessed: 15-Nov-2020].

[19] "Unity vs Unreal: Ultimate Game Engine Showdown," The Ultimate Resource for Video Game Design, 12-Jun-2020. [Online]. Available: https://www.gamedesigning.org/engines/unity-vs-unreal/. [Accessed: 15-Nov-2020].

[20] Unreal Engine 5. [Online]. Available: https://fasraspen586.weebly.com/unreal-engine-5.html. [Accessed: 11-Nov-2020].

[21] Vergun, "Heads-up display to give Soldiers improved situational awareness," www.army.mil. [Online]. Available: https://www.army.mil/article/188088/heads-up-display-to-give-soldiers-improved-situational-awareness. [Accessed: 24-Apr-2021].

[22] W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim, and Y. Wang, "UnrealCV," Proceedings of the 2017 ACM on Multimedia Conference - MM '17, 2017.

[23] X. Ran, C. Slocum, M. Gorlatova, and J. Chen, "ShareAR," Proceedings of the 18th ACM Workshop on Hot Topics in Networks, 2019.

[24] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, "Strike (With) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nov. 2018.

[25] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, and Yizhou Wang. 2017. UnrealCV: Virtual Worlds for Computer Vision. In Proceedings of the 25th ACM international conference on Multimedia (MM '17). Association for Computing Machinery, New York, NY, USA, 1221–1224.

[26] xoio, "Xoio Berlin Flat in Architectural Visualization - UE Marketplace," Unreal Engine. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/xoio-berlin-flat. [Accessed: 26-Apr-2021].