StarCraftImage: A Dataset For Prototyping Spatial Reasoning Methods For Multi-Agent Environments

Sean Kulinski† Purdue University
Nicholas R. Waytowich ARL‡
James Z. Hare ARL‡
David I. Inouye† Purdue University

Abstract

Spatial reasoning tasks in multi-agent environments such as event prediction, agent type identification, or missing data imputation are important for multiple applications (e.g., autonomous surveillance over sensor networks and subtasks for reinforcement learning (RL)). StarCraft II game replays encode intelligent (and adversarial) multi-agent behavior and could provide a testbed for these tasks; however, extracting simple and standardized representations for prototyping these tasks is laborious and hinders reproducibility. In contrast, MNIST and CIFAR10, despite their extreme simplicity, have enabled rapid prototyping and reproducibility of ML methods. Following the simplicity of these datasets, we construct a benchmark spatial reasoning dataset based on StarCraft II replays that exhibit complex multi-agent behaviors, while still being as easy to use as MNIST and CIFAR10. Specifically, we carefully summarize a window of 255 consecutive game states to create 3.6 million summary images from 60,000 replays, including all relevant metadata such as game outcome and player races. We develop three formats of decreasing complexity: hyperspectral images that include one channel for every unit type (similar to multispectral geospatial images), RGB images that mimic CIFAR10, and grayscale images that mimic MNIST. We show how this dataset can be used for prototyping spatial reasoning methods. All datasets, code for extraction, and code for dataset loading can be found at https://starcraftdata.davidinouye.com/.

1. Introduction

Spatial tasks in multi-agent environments require reasoning over both agents’ positions and the environmental context such as buildings, obstacles, or terrain features. These complex spatial reasoning tasks have applications in autonomous driving, autonomous surveillance over sensor networks, or reinforcement learning (RL) as subtasks of the

RL agent. For example, to predict a car collision, an autonomous driving system needs to reason about other cars, road conditions, road signs, and buildings. For autonomous surveillance over sensor networks, the system would need to reason over the positions of objects, buildings, and other agents to determine if a new agent is normal or abnormal or to impute missing sensor values. An RL system may want to predict the cumulative or final reward or impute missing values given only an incomplete snapshot of the world state, i.e., partial observability. Yet, collecting large realistic datasets for these tasks is expensive and laborious.

Due to the challenge of collecting real-world data, practitioners have turned to (semi-)synthetic sources for creating large clean datasets of photo-realistic images or videos [11, 12, 24, 40]. For example, [11] leveraged the Grand Theft Auto V game engine to collect a synthetic video dataset for pedestrian detection and tracking. [4] overlays aerial images with crowd simulations to provide a crowd density estimation dataset. Yet, despite near photo-realism, these prior datasets focus on simple multi-agent environ-
ments (e.g., pedestrian-like simulations \cite{11, 40}) and thus lack complex (or strategic) agent and object positioning. In sharp contrast to these prior datasets, human-based replays of the real-time strategy game StarCraft II capture complex strategic and naturally adversarial positioning of agents and objects (e.g., buildings and outposts). Indeed, the human player provides thousands of micro-commands that produce an overall intelligent and strategic positioning of agents and building units. The release of the StarCraft II API and Python bindings \cite{38} significantly reduces the barrier to using this rich data source for multi-agent environments. Yet, the StarCraft II environment still requires significant overhead including game engine installation, looping through the game engine, understanding the API, etc. This greatly limits the broad adoption of this very rich source of multi-agent interactions as a benchmark dataset—in contrast to the classic and extremely easy-to-use MNIST \cite{27} and CIFAR10 \cite{25} benchmark datasets that drove image classification research in the early years and continue to be used for prototyping new ML methods. In summary, prior multi-agent datasets either lack complex strategic behavior or require significant implementation overhead.

To address these issues, we created StarCraftImage: a simplified image-based representation of human-played StarCraft II matches to serve as a large-scale multi-agent spatial reasoning benchmark dataset that is as easy to use as MNIST and CIFAR10 while still exhibiting complex and strategic object positioning. As seen in Fig. 1, each image in StarCraftImage is akin to a detailed snapshot of the StarCraft II minimap and includes the locations of all units (both moveable units and buildings), the units’ IDs, as well as important metadata like which player won that match, player resource counts, the current map name, player ranking, etc. We made two key design decisions when developing StarCraftImage. First, we chose to represent the matches by snapshot images that summarize a window of approximately 10 seconds of gameplay rather than a video. This design choice was motivated both by the ease-of-use criteria (as images are easier to load and manipulate than videos) and by the goal of performing spatial reasoning rather than temporal reasoning tasks—though a video dataset for complex temporal reasoning is a natural direction for future work.

Our second design choice was to represent the matches via minimap-like images rather than photo-realistic renderings of the game state. This choice was motivated by two reasons. First, minimap images are easy to use because they are small yet still represent of the whole environment. By using a minimap representation, we can encode the most crucial game information (unit types, recent troop movements, building locations, environmental features, etc.) in a naturally compact representation. Indeed, the minimap representation is critical for playing StarCraft II as evidenced by the following quote from the famous StarCraft II player Day\cite{9} (Sean Plott): “...the two most important things are the minimap and your money” \cite{32}. Second, the minimap representation allows for us to have many diverse samples while still maintaining a small data footprint. The resultant smaller disk size allows for rapid prototyping via quick dataset downloads and swift data consumption. Compared to prior common spatial reasoning datasets, our proposed 3.6 million image dataset has a total disk size of 8.4Gb while the MOTSynth-MOT-CVPR22 dataset \cite{11}, which consists of 1.3 million images, has a disk size of 167Gb (20x larger, while containing half the number of samples). Ultimately, we construct three different image representations with decreasing complexity: Hyperspectral images which give precise game state information by encoding the unit ids and last-seen timestamps at each spatial location (mimicking the hyperspectral geospatial representations), RGB images that mimic CIFAR10, and grayscale images that mimic MNIST. Thus, our dataset is compatible with common ML frameworks with minimal overhead or preprocessing effort. Overall, we use 60k StarCraft II replays to create 3.6 million summary images (not multi-counting different representations) and corresponding metadata.

To demonstrate how multi-agent spatial reasoning tasks can be easily prototyped using StarCraftImage, we also provide a series of benchmark tasks. We perform target identification (i.e., determining unit type from only knowing unit locations) where the input is either an RGB or grayscale image and the target image is hyperspectral with each channel corresponding to a unit type. We also perform more complex tasks such as map event prediction (i.e., game outcome and StarCraft race prediction) which serve as canonical image-level reasoning problems. To show how our image representations can be easily manipulated for other tasks (like Rotated MNIST \cite{26} or Color MNIST \cite{2}), we map missing data imputation as an image inpainting task using both simulated sensor network faults and the fog-of-war from the game engine. Ultimately, we hope to provide a large-scale and rich multi-agent spatial reasoning dataset that is very easy to use yet exhibits complex and strategic placement of agents for complex spatial reasoning applications. We summarize our contributions as follows:

- We design and extract StarCraftImage as an easy-to-use multi-agent spatial reasoning dataset under three representations: 1) Hyperspectral images that encode all unit ids and lasts seen timestamps for each spatial location, 2) RGB images that mimic CIFAR10, and 3) grayscale images that mimic MNIST.
- We apply StarCraftImage on tasks such as target identification, movement prediction, and more. We also propose several noise simulation models and discuss several task modifiers such as domain generalization.
- We publicly release the datasets with a permissive CC
2. Dataset Extraction and Construction

In this section, we describe how we extract observational data from the simulated yet complex environment of the StarCraft II (SC2) game. We then transform the raw data into the hyperspectral, CIFAR10, and MNIST formats that are readily usable in ML tools.

2.1. Extracting Raw Data From SC2 Replays

Due to SC2 being an almost entirely deterministic game, an SC2 replay file contains an entire list of actions from both players that can be used to re-simulate an entire match by passing the actions back to the SC2 game engine. Each replay file also contains metadata from the match such as: the length of the match, the map/arena the match took place in, and per-player statistics such as the match making rating (MMR) which can be thought of as the skill level of that player), the actions-per-minute (APM) the player took, and whether that player won, lost, or tied the match. Additionally, Activision Blizzard (the maker of SC2) bundles large sets of these replays together as a Replay Pack for others to use. We used Replay Pack 3.16.1 - Pack 1 from [6].

To extract the game state, we used the PySC2 [38] Python library developed for RL applications that interfaces with the SC2 game engine. PySC2 exposes the raw game state while re-simulating a match based on replay files. Each raw game state consists of information such as the location, allegiance, size, unit type ID, and health of every unit (character, building, worker, soldier, etc.) which currently exists for that specific frame (where a frame is a single unit of time in a game). The raw frame data also contains dynamic map information including the visibility for each player (the locations on the map that the player can see due to friendly units/scouts being in that area, versus areas which are undiscovered and thus hidden) and the current creep state (which is a terrain feature consisting of purple slime in which most Zerg structures must be built and upon which Zerg units will move faster). However, since the PySC2 interface was designed for interacting with StarCraft II, it comes with a steep learning curve and a complex data representation which greatly hinders our goal of having clean observed game states that can be represented in a standard form. Thus, we use PySC2 to extract raw game state observations and process these into standard image formats.

2.2. StarCraftHyper: Construction and Processing of Hyperspectral Representation

Our most general format is a hyperspectral image format where each channel represents information for each unit type for each player in SC2. To do this, we first use PySC2 to extract raw frame data and for the \( f \)th frame observation, we record the location of each unit present via \( H_f [u_{PID}, x_h, y_h] = 1(u_{PID}, x_h, y_h) \) where 1 is an indicator function that returns 1 if a unit is present, else 0, \( u_{PID} \) is the player-specific unit ID (PID), and \( x_h, y_h \) is the spatial location of the unit. Since the raw data gives spatial information in raw game-map spatial coordinates, we must perform a coordinate transform to our square hyperspectral image coordinates: \( (x_h, y_h) = \left( \frac{x_{raw}, y_{raw}}{\max(x_{raw}, y_{raw})} \right) \). We also...
Figure 3. (Top) We embed the unit information of player 1, player 2, and neutral separately using an embedding of size 1. We then combine with other dense features (visibility for players and terrain info for neutral). Finally, we concatenate each output into a 3-channel 32x32 px RGB image where the neutral channel is down-weighted for visual clarity. (Bottom) We take the RGB color image, rescale the values of each channel, and overlay each channel into a single grayscale 28x28 px image where precedence is given to P1, then P2, and finally neutral or background. We use precedence combinations as linear combinations of the layers could lead to unit information being canceled.

in processing word sequences, we pad the channels of the dense representation with zeros (representing no unit) up to the max number of units at any location (either in a single sample or in a batch of samples), denoted by $k$. Concretely, the bag-of-units representation collapses the channel axis into $k$ ID matrices and $k$ timestamp matrices of size $(64, 64)$, where the ID matrix contains the $PIDs$ of the units present at each $(x, y)$ coordinate, the timestamp matrices contain the corresponding timestamp that the unit was last seen, and $k$ is the max number of units present at one $(x, y)$ location in $H$. This highly-compressed bag-of-units representation for the StarCraftHyper dataset can be seen in the top right of Fig. 2 and is the default representation for the StarCraftHyper dataset.

2.3. StarCraftCIFAR10 and StarCraftMNIST: RGB and GrayScale Representations

To further simplify dataset usage and prototyping ability, we develop datasets that mimic CIFAR10 and MNIST in terms of image size, number of channels, number of classes, and number of train/test samples as seen in (middle) and (right) of Fig. 4. Thus, our StarCraftCIFAR10 and StarCraftMNIST datasets can be used for rapid initial prototyping of new spatial reasoning methods just as these ubiquitous datasets have been used for prototyping image classification. These can model situations where agent and building positions are known but the agent type is unknown (e.g., low resolution satellite images or a network of pressure sensors). One natural task is to infer unit types given only unit location information, which is discussed in more detail in future sections.

To construct StarCraftCIFAR10, we first follow the approach in subsection 2.4 to subsample our StarCraftHyper dataset to 50,000 train windows and 10,000 test windows, to
match the dataset size of CIFAR10. To transform each hyperspectral window into a CIFAR10 format, we separate $H$ into player-specific images and follow the process shown in Fig. 3 (top). To construct the StarCraftMNIST dataset, we similarly subsample from the full StarCraftHyper dataset, but to a size of 60,000 train and 10,000 test images as in MNIST. We process the images in the manner seen at the top and bottom of Fig. 3, where the last step is a function that overlays the $V_{p1}$, $V_{p2}$, $V_N$ scaled maps on top of each other such that any non-zero elements of $V_{p2}$ will overwrite the non-zero elements of $V_N$ and non-zero $V_{p1}$ values will overwrite both. We decided to overwrite rather than average because having a unit of player 1 and player 2 at the same location would average to a gray background value but that is in fact one of the most interesting locations. In the next section, we discuss the creation of the 10 classes for each window via a combination of the variables: Player 1 race, Player 2 race, and Player 1 outcome.

### 2.4. Dataset Exploration and Analysis

All in all, the StarCraftImage dataset consists of 3,607,787 windows extracted from 60,000 replays which are readily available in three representations (examples in Fig. 4). The image data for each window is stored as a .png file in the bag-of-units representation. The data can be accessed via directly loading in the relevant .png file and metadata row, or more simply by using the corresponding PyTorch dataset classes that we have developed (one class for each representation). 

Jointly with the image data collection, we also aggregated relevant metadata for each window, such as the temporal location of the window in the overall match (e.g., 75th window of 130), which player won the match, the races of the players, the name of match’s map, etc. (for a full list of the metadata keys, please see Appendix D). This metadata has many uses for filtering replays based on conditions for a specific application, e.g., training on a subset of maps and testing on the held out set. Additionally, for canonical class labels, we use the race of each player (Terran, Zerg, or Protoss) and player 1’s outcome (Win and NotWin where NotWin includes the rare Tie outcome) to split the overall dataset into 18 classes (3 races for player 1, 3 races for player 2 and 2 outcomes). We chose these three variables (Player 1 outcome, Player 1 race, Player 2 race) because though outcome prediction is a canonical task, readily available ground truth for race prediction with this dataset is akin to behavior or tactical strategy prediction, as unit type information is hidden in the StarCraftCIFAR10 and StarCraftMNIST versions of the dataset. For StarCraftCIFAR10 and StarCraftMNIST, we select only classes that have at least one player as Zerg (5 total) with both outcomes to get exactly 10 balanced classes to match the setup of CIFAR10 and MNIST—this could be done similarly for Terran and Protoss but Zerg is the easiest to understand because some Zerg-specific units are often spread across the battlefield.

### 3. Multi-Agent Spatial Reasoning Applications

In this section, we list examples of spatial reasoning tasks on our datasets (e.g., global reasoning as a classification task). We will also discuss simple noise models that simulate more complex scenarios on top of the clean data representations. Finally, we discuss natural task modifiers such as domain generalization or adversarial contexts. In all cases, we aim for a compromise between realism and simplicity as this dataset is meant as an initial prototyping dataset for complex or strategic agent and object positioning rather than a fully realistic spatial reasoning dataset. Given space constraints, we provide demos of these tasks in the supplementary material both in Appendix F and as IPython notebooks in our code repository.

#### 3.1. Spatial Reasoning Examples

**Target identification (Image colorization)** The goal here is to identify the unit type (e.g., marine unit) or af-
filiation (player one, two or neutral) for every detected unit. This can be seen as a setting where an image only shows if a unit exists in its field of view (e.g., an aerial photo from a UAV or a post-processed output from a LiDAR scanner). For the task, we cast this problem as an image colorization problem in which the input is either a StarCraftCIFAR10 or StarCraftMNIST and the target output is the corresponding StarCraftHyper or StarCraftCIFAR10 image.

Movement prediction (Simplified Multi-Object Tracking) Predicting what is going to happen next is clearly an important task especially in time-critical applications such as autonomous driving [12], disaster relief [1], or, more generally, optical flow [3]. While we do not generally consider the time dimension after we summarize the window, for this task, we can use the metadata to create pairs of adjacent window summary images where the input is the current summary image and the target output is the next summary image in the same match.

Predict final outcome or race (Classification) Spatial reasoning systems are often used to predict the global properties of a system (e.g., crop yield predictions [31] or reward predictions for RL models [38]), which can be cast as classification. The most canonical task is to predict the final outcome of the game (i.e., which player will win), which requires reasoning over both fighting units and environmental factors such as buildings and resources (e.g., even if there is little movement/few fighting units in a window, a model can still predict who will win based off of who has the strongest base). Another canonical task for the simplified datasets StarCraftCIFAR10 and StarCraftMNIST (which give only unit location information rather than unit type information) is to predict both players’ races, which requires recognizing the common placement configurations for each race.

Imputing missing data (Image inpainting) Another critical task in spatial reasoning is imputing missing values for areas that lack coverage due to occlusion, data collection failures, or adversarial attacks [13, 39]. Here the input image is a corrupted version of a sample from one of the three datasets, and the target output image is the uncorrupted sample. Due to StarCraftImage’s simple minimap representation, simulating spatial corruptions (e.g., noisy measurements or partial observability) is simple to do—unlike in photo-realistic settings which would require editing the images or videos to hide or remove information. In the next section, we go over examples of spatial corruption models.

3.2. Simulated Data Corruption Models

Random additive noise This corruption model is relevant for settings where images are taken using noisy equipment or a hierarchical system where reasoning happens on a (potentially noisy) abstracted spatial representation. We can implement this as a type of salt and pepper noise where the salt noise can randomly add units to locations that do not have units (i.e., false positives) and each real unit could possibly become missing (i.e., false negatives), as seen in the left of Fig. 5.

Heterogeneous partial observations (Image masking) Here the images can be seen as the fusion of irregular heterogeneous sensor networks. This can be simulated by producing a mask that is based on static sensor locations and detection ranges, see Fig. 5 (middle) for example. Furthermore, detailed sensor models can be used to pre-process the masked observations to provide an accurate representation based on the type of sensor implemented at a particular location, e.g., acoustic sensors may only return a range of the unit relative to its position. Sensor faults as above could be implemented on top of this heterogeneous sensor network (e.g., masking over a set of sensors’ visible range). For examples and benchmark results on such heterogeneous sensor placements with aggregation failure simulations, please see Appendix E.

Imprecise sensors (Blur) Low resolution imaging will yield imprecise unit locations. Thus, we can implement this noising process by performing blur operations on top of the original datasets. This corruption is simplest to apply to StarCraftCIFAR10 (e.g., Fig. 5, right) and StarCraftMNIST via standard CV packages but could also be applied to StarCraftHyper (albeit with more computation).

3.3. Spatial Reasoning Task Modifiers

Robustness to distribution shift (Domain generalization) A key challenge in applying ML to real-world settings is training a model in one context but applying it to another context [48]. This is known as the domain generalization problem in which the goal is to perform the task well on an unseen test domain [33]. The metadata that we provide can provide natural segmentations of the dataset into domains. One of the most canonical examples of distribution shift in real-world settings is a change in the environment settings [23]. While greatly simplified, we can simulate changes in location by splitting the dataset based on the SC2 map and holding out one or more maps for testing. Other excellent domain splits could be players’ MMR or APM, which correspond to their skill level and frequency of actions. Player two’s race (Terran, Protoss, or Zerg) is also another way to split the dataset into 9 domains such as Terran vs. Terran, Protoss vs. Zerg, or Terran vs. Protoss.
for reasoning methods, especially those which involve humans such as autonomous driving. We can simulate this idea by applying adversarial training methods under different adversarial attack models such as L0 pixel-wise attacks [36] for attacking individual units. The adversarial training literature already benchmarks using MNIST as a key difficult example [30], and thus, these StarCraft datasets could be immediately relevant and provide a more realistic benchmark for the adversarial training literature.

### Equipment usage optimization (Active learning)

Optimizing sensor location and power usage are key challenges in sensor networks [9, 14]. Following the simulated sensor network seen in the previous section, constrained power usage could be framed as an active learning problem in which the algorithm can only query a fixed number of sensors for each prediction problem. For optimizing sensor location, the algorithm could attempt to determine where to place the next sensor (i.e., to uncover information at a certain location) to optimize the downstream task such as outcome classification. A more complex case is moving sensors from their original locations to another location under a budget on geographic movement (e.g., a sensor on a robotic device).

### 4. Benchmark Evaluations

While we point the reader to Appendix E, where we give full descriptions and results, here we introduce four benchmark multi-agent spatial reasoning tasks, which incorporate training U-Net-based [35] ResNet [15] models. The four benchmark tasks consist of two tasks on target identification (given a 64x64 RGB image, predict the ID of each unit at each location) and two tasks for unit tracking (given hyperspectral window $k$, predict what will happen in window $k+1$). Both task sets consist of first training and evaluating on “clean” (unaltered) data. To highlight the extendability of StarCraftImage, we also perform both tasks on corrupted data that has been passed through a simulation of a noisy sensor network. The sensor network simulation consists of 50 imaging sensors with a radius of 5.5 pixels with different sensor placement methodologies (e.g., grid, random) and communication failures during sensor fusion (see Fig. 10 for details), and results in noisy training windows. From the results seen in Table 1, it is clear that this is a difficult problem, especially when reasoning over corrupted samples, and hopefully future work can build upon these results.

### 5. Preliminary Real-World Experiment on DOTA Satellite-Image Dataset

In this section, we explore whether performance on StarCraftImage is predictive of performance on real-world datasets. To this end, we use a version of the DOTA dataset [41], which is a benchmark dataset for multi-object detection in satellite images, where the samples have been transformed to match a similar format to StarCraftImage, which we call DOTA-UnitID (see Fig. 6). This format is similar to the scenario when we may have remote sensing or a sensor network that can detect the presence of certain agents or buildings but may not know what they are (e.g., due to cloud cover only synthetic aperture radar data is available).

![Figure 5. Three example noise corruption models which are simulated on top of the StarCraftCIFAR10 dataset, where (left) simulated random additive noise, (middle) simulates observations via a heterogeneous SN, and (right) simulates limited precision (blurry) observations.](image)

![Table 1. Benchmark Evaluations on Unit Type Identification and Next Hyperspectral Window Prediction with clean data and simulated data corruptions.](table)

<table>
<thead>
<tr>
<th>Placement</th>
<th>Unit Identification (Acc)</th>
<th>Next Wind. (MSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clean</td>
<td>Grid</td>
</tr>
<tr>
<td>Unet-ResNet18</td>
<td>56.6%</td>
<td>40.3%</td>
</tr>
<tr>
<td>Unet-ResNet34</td>
<td>58.5%</td>
<td>40.2%</td>
</tr>
<tr>
<td>Unet-ResNet50</td>
<td>62.5%</td>
<td>44.0%</td>
</tr>
</tbody>
</table>

![Figure 6. DOTA dataset examples, where the top row shows three original (input, annotations) pairs from the DOTA dataset [41], while the bottom row shows the three corresponding (input, label) pairs from our DOTA-UnitID dataset. The DOTA-UnitID task is to colorize the grayscale annotation mask.](image)
task for both the StarCraftImage dataset and the DOTA-UnitID datasets. As seen in the second row of Table 2, the model ranking is the same for both the DOTA-UnitID and StarCraftImage-UnitID experiments across all models (e.g., the Unet-ResNet50 had the best unit accuracy across both datasets), thus providing preliminary evidence that performance improvements on our dataset will carry over to real-world datasets. We note that the transformer results are much below the results of the ResNet models. This is likely due to these larger models requiring longer training times than the CNN-based models. Despite this, these results suggest that StarCraftImage is still a difficult dataset even for SOTA models.

Table 2. Unit-ID experiment results on clean data for StarCraftImage and Dota-UnitID. RX is short for a Unet-ResNet-X model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Lawin [44]</th>
<th>SegFormer [42]</th>
<th>R18</th>
<th>R34</th>
<th>R50</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCII</td>
<td>27.0%</td>
<td>27.9%</td>
<td>36.6%</td>
<td>38.5%</td>
<td>62.5%</td>
</tr>
<tr>
<td>DOTA</td>
<td>34.1%</td>
<td>35.0%</td>
<td>52.4%</td>
<td>52.8%</td>
<td>53.6%</td>
</tr>
</tbody>
</table>

6. Related Works

As with any ML task, accessible datasets are critical for making advancements. For spatial reasoning tasks, these include elementary reasoning datasets (e.g., CLEVR [20]), scene understanding (e.g., Places [47]), geospatial datasets (e.g., Chesapeake Land Cover [34]), optical flow datasets (e.g., Middlebury [3]), and more.

Multi-Agent Spatial Datasets A notable area for multi-agent spatial reasoning tasks is reasoning for autonomous driving. For this, the well-known KITTI dataset [12] has driven many advancements since its introduction in 2012, and more recently the Waymo Open dataset [37] has introduced 1.1K additional scenes with LiDAR and Camera measurements for practitioners to benchmark on. More generally, there is TAO [8], a multi-object tracking dataset, which is akin to a video-version of Microsoft COCO [29] and has over 800 object classes. In a similar vein to our work exists pedestrian and crowd analysis (e.g., crowd counting [5, 40], person ReID [46], population density estimation [4]), however, these datasets tend to have simple agent behaviors such as conversing or walking from one point to another across a scene.

Synthetic Datasets Developing multi-agent spatial reasoning datasets can be expensive as they tend to involve humans in the collection process. Thus, practitioners have turned to collecting this data from simulations of the real world. For pedestrian tracking, there is the MOTSynth dataset [11], GCC [40], and the GTA dataset [24] which all use Grand Theft Auto V to produce realistic pedestrian images/behaviors as agents walk across a scripted scene. Following [24], the GTAV’s rendering engine is used to produce exact crowd counts for [40] and bounding boxes, segmentation masks, and depth masks of all agents for [11, 24].

Table 3. An overview of multi-object spatial reasoning datasets. StarCraftImage has the most complex agent positioning, the lowest overhead, and the ability to simulate more complex scenarios (e.g., data corruption, as seen in subsection 3.2). GT stands for “ground truth”.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Frame Count</th>
<th>Agent Positioning</th>
<th>Overhead</th>
<th>GT</th>
<th>Noise Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD [5]</td>
<td>2K</td>
<td>Real Pedestrian</td>
<td>Some</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCC [40]</td>
<td>15K</td>
<td>Simulated Crowd</td>
<td>Low</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TAO [8]</td>
<td>2.2M</td>
<td>Real YouTube</td>
<td>Some</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PySC2 [38] + Replays</td>
<td>n/a</td>
<td>Complex / Strategic</td>
<td>High</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>StarCraftImage (ours)</td>
<td>3.6M</td>
<td>Complex / Strategic</td>
<td>Low</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

7. Conclusion

We introduce StarCraftImage as a multi-agent spatial reasoning dataset with the overarching goal of being as easy to use for prototyping and initial method testing as MNIST and CIFAR10 while capturing complex and strategic unit positioning for advanced spatial reasoning methods. To this extent, we process raw frame data from 60 thousand human StarCraft II replays to formulate 3.6 million summary images in three representations of decreasing complexity: StarCraftHyper which is hyperspectral images that encode the unit ids and last seen timestamps at each spatial location, StarCraftCIFAR10 which is RGB images that mimic CIFAR10, and StarCraftMNIST which is grayscale images that mimic MNIST. We also include relevant metadata for each summary image which can be used to filter the StarCraftImage dataset when performing the tasks, corruption extensions, and modifiers we discuss in section 3. While we hope this work allows for easy prototyping and thus simpler and more systematic advances in developing spatial reasoning methods, we recognize that although this dataset is based on complex human actions, it is still a simplified simulated environment, and thus real-world data (or more realistic data) will always be needed to fully evaluate methods. Additionally, our code for dataset processing, extracting, and loading the data could be used to expand or specialize new StarCraft datasets for multi-agent spatial reasoning applications using the millions of publicly available StarCraft II replays via Blizzard’s developer API without the overhead of starting from scratch. Ultimately, we hope our dataset provides the ML community with an easy-to-use multi-agent spatial reasoning dataset that will significantly reduce the barrier of entry for these important tasks.

Acknowledgements This work was supported by NSF (IIS-2212097) and ARL (W911NF-2020-221).
European conference on computer vision, pages 740–755. Springer, 2014. 8


[40] Qi Wang, Junyu Gao, Wei Lin, and Yuan Yuan. Learning from synthetic data for crowd counting in the wild. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 8198–8207, 2019. 1, 2, 8


