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

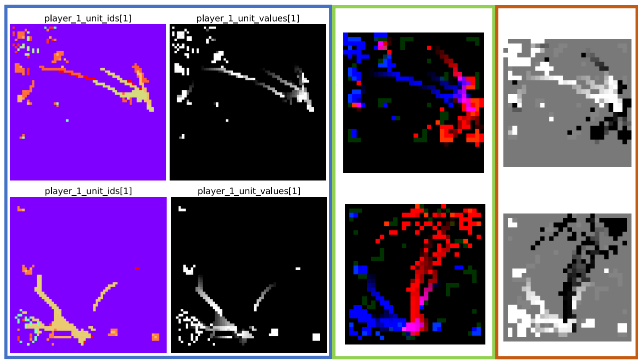


Figure 1. Two samples (one per row) showing (Blue box/left) our 64 x 64 StarCraftHyper dataset which contains all unit IDs and corresponding values for both players (color for unit IDs denotes categorical unit ids), (Green box/middle) StarCraftCIFAR10 (32 x 32) which is easy to interpret where blue is player 1, red is player 2, and green are neutral units such as terrain or resources, and (Orange box/right) StarCraftMNIST (28 x 28) which are grayscale images further simplified to show player 1 as light-gray, player 2 as dark-gray, and neutral as medium-level shades of gray.

[‡]DEVCOM Army Research Laboratory

[†]Corresponding Authors: Sean Kulinski skulinsk@purdue.edu and David I. Inouye dinouye@purdue.edu.

This work appeared in the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 22004-22013

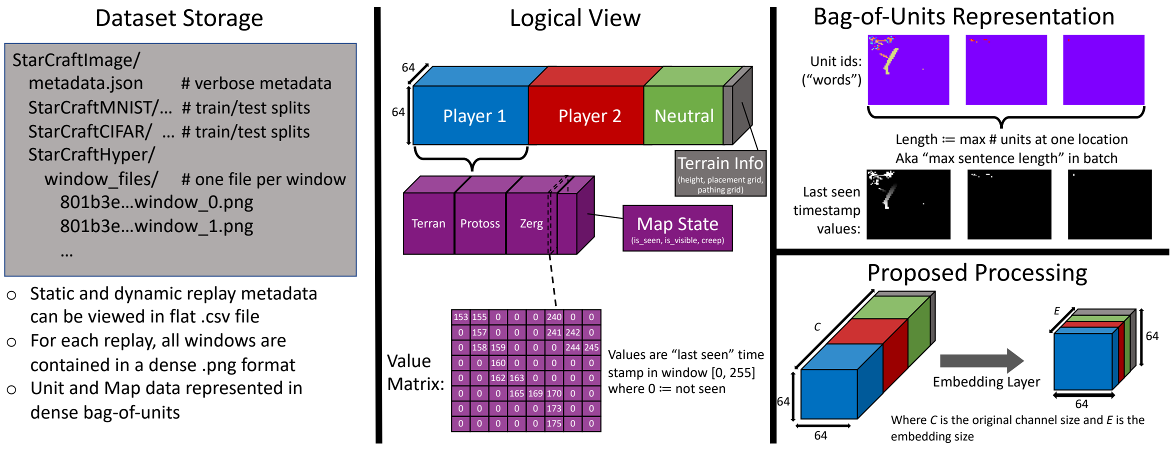
density estimation dataset. Yet, despite near photo-realism, these prior datasets focus on simple multi-agent environments (e.g., pedestrian-like simulations [11, 40]) and thus lack complex (or strategic) agent and object positioning. In sharp contrast to these prior datasets, human-based replays of 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 [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 [27] and CIFAR10 [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* 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[9] (Sean Plott): "...the two most important things [are] the minimap and your money" [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 10.6Gb while the MOTSynth-MOT-CVPR22 dataset [11], which consists of 1.3 million images, has a disk size of 167Gb (16x 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 [26] or Color MNIST [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.


Dataset Storage
Static and dynamic replay metadata can be viewed in flat .csv file
For each replay, all windows are contained in a dense .png format
Unit and Map data represented in dense bag-of-units
Logical View
64
64
Player 1
Player 2
Neutral
Terrain Info
(height, placement grid, pathing grid)
Terran
Protoss
Zerg
Map State
(is_seen, is_visible, creep)
Value Matrix:
Values are "last seen" time stamp in window [0, 255] where 0 ≔ not seen
Bag-of-Units Representation
Unit ids: ("words")
Length ≔ max # units at one location
Aka "max sentence length" in batch
Last seen timestamp values:
Proposed Processing
C
E
64
64
64
64
Embedding Layer
64
Where C is the original channel size and E is the embedding size

```
StarCraftImage/
  metadata.json        # verbose metadata
  StarCraftMNIST/…  # train/test splits
  StarCraftCIFAR/  … # train/test splits
  StarCraftHyper/
    window_files/     # one file per window
      801b3e…window_0.png
      801b3e…window_1.png
      …
```

153	155	0	0	0	240	0	0
0	157	0	0	0	241	242	0
0	158	159	0	0	0	244	245
0	0	160	0	0	0	0	0
0	0	162	163	0	0	0	0
0	0	0	165	169	170	0	0
0	0	0	0	0	173	0	0
0	0	0	0	0	175	0	0

Figure 2. An overview of our hyperspectral dataset from different perspectives. The raw image data is stored in texttt.png files using the bag-of-units representation. A logical view of the dataset is a (sparse) hyperspectral image with many channels that include unit information and visibility per player, resource information (neutral units), and map information. The bag-of-units representation enables processing this very high-dimensional dataset using dense matrices only and leveraging embedding layers that are often used for processing sequences of IDs; importantly, because the unit order does not matter, an order-invariant reduction such as max or sum should be used to arrive at a representation with a fixed number of embedding channels E.

- We publicly release the datasets with a permissive CC BY 4.0 license. We also release the StarCraft II dataset extraction code and the relevant data loaders and modules for using the data as a Python package with an MIT license, and provide matainance as laid out in our dataset nutrition label: Table 4.

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, Activison 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, solider, 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 $H_f[u_{PID}, x_h, y_h] = \mathbb{1}(u_{PID}, x_h, y_h)$ where $\mathbb{1}$ 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: $(x_h, y_h) = \left\lfloor \frac{(x_{raw}, y_{raw})}{\max(x_{raw}, y_{raw})} \right\rfloor$. We also

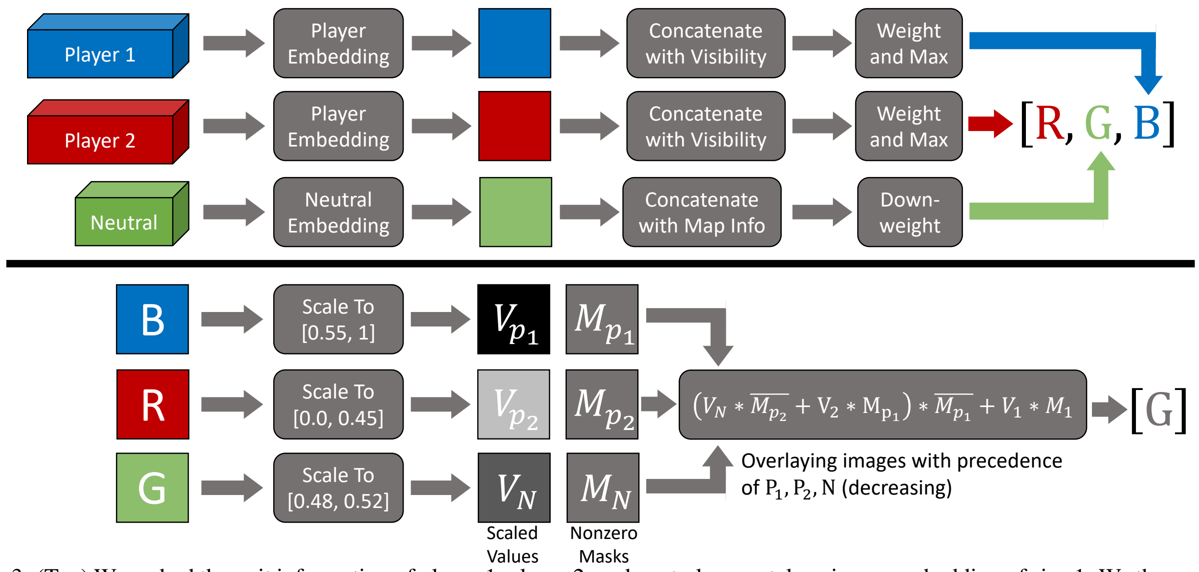


Figure 3. (Top) We embed the unit information of player 1, player 2, and neutral separately using an embedding of size 1. We then combine with other dense features (visibility for players and terrain info for neutral). Finally, we concatenate each output into a 3-channel 32x32 px RGB image where the 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.

crop to only the playable area of the map. There are 170 unit IDs for Player1, 170 IDs for Player2 units, and 44 IDs for Neutral units, which is 384 PIDs (i.e., channels).

Next, to allow for video-like spatial movements, we form a stack of 255 consecutive hyperspectral images $H_{stack} = [H_f]$ where $f \in [0, 255]$.[1] Given H_{stack}, which is a tensor with shape $(255, 384, 64, 64)$, we simplify this from a video format to a static image format by collapsing the time axis to create the summarized hyperspectral image H. We do this by recording the frame index of the most recent frame where a unit was present for each (PID, x, y) coordinate (i.e., $H[c, x, y] = \arg\max_f H_{stack}[f, c, x, y] \neq 0)$ and if $H_{stack}[f, c, x, y] = 0 \; \forall f \in [0, 255]$, then $H[c, x, y] = 0)$. Another possibility would be to average over the window of frames instead of the last seen timestamp; however, the last seen timestamp enables simple visualization of movement via a ghosting-like effect, and the last seen timestamp preserves time information (albeit only a compressed amount).

While this condensed representation does have the trade-off that if a unit with the same PID crosses the same (x, y) location more than once in a frame stack, only the last crossing will be recorded in H, this is rare and seems a reasonable trade-off for a much simpler representation. At this step, the non-zero entries of H are saved as the raw representation of our dataset—i.e., a sparse tensor representation. We can compress H into a dense "bag-of-units" representation, which is similar to representing a sequence of words by their IDs rather than by very high-dimensional one-hot vectors, but where the order of the IDs does not matter (hence, the term "bag" as in bag-of-words representations). Just as the number of words of a sentence can vary in NLP, the number of units at each location can vary. Therefore, as 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 PID 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

[1] We chose 255 to ensure that the values fit in an unsigned 8-bit integer and captures roughly 10 seconds of real time.

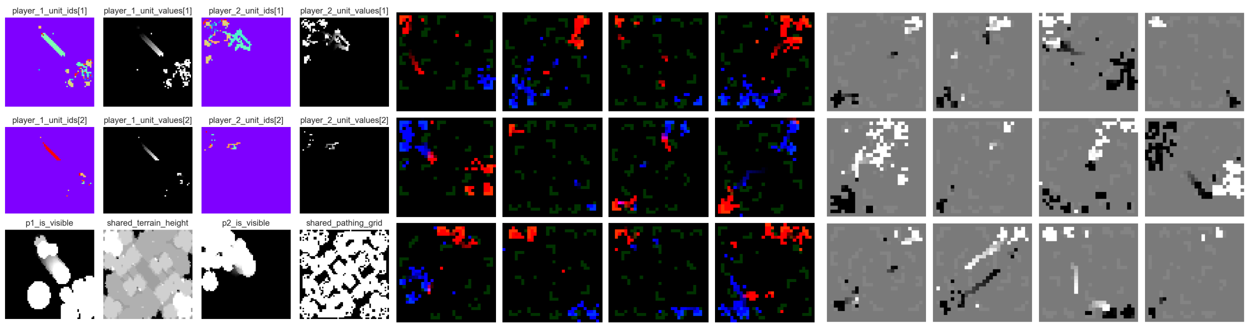


Figure 4. (Left) Our 64 x 64 StarCraftHyper dataset contains all unit IDs and corresponding values for both players (color for unit IDs denotes categorical unit ids) where visibility is player specific but the terrain and pathing grid are shared (a few other layers are not shown, see appendix). (Middle) StarCraftCIFAR10 (32 x 32) is easy to interpret where blue is player 1, red is player 2, and green are neutral units which are usually just resources. (Right) MNIST (28 x 28) grayscale images are further simplified to show player 1 as white to white-gray, player 2 as black to black-gray, and neutral as shades of gray.

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_{p_1}, V_{p_2}, V_N scaled maps on top of each other such that any non-zero elements of V_{p_2} will overwrite the nonzero elements of V_N and nonzero V_{p_1} 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 affiliation (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

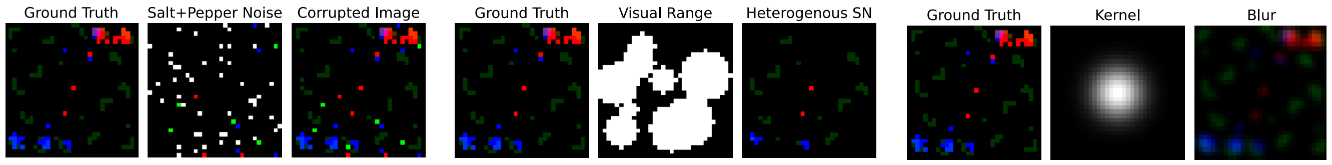


Figure 5. Three example noise corruption models which are simulated on top of the StarCraftCIFAR10 dataset, where (left) simulated random additive noise, (middle) simulates observations via a heterogeneous SN, and (right) simulates limited precision (blurry) observations.

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.

Robustness to adversarial attacks (Adversarial training) While uncommon, adversarial attacks are a genuine concern 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.

Table 1. Benchmark Evaluations on Unit Type Identification and Next Hyperspectral Window Prediction with clean data and simulated data corruptions.

	Unit Identification (Acc)			Next Wind. (MSE)		
Placement ⇒	Clean	Grid	Rand.	Clean	Grid	Rand.
Unet-ResNet18	56.6%	40.3%	30.1%	3.97	4.11	4.15
Unet-ResNet34	58.5%	40.2%	30.8%	3.99	4.12	4.17
Unet-ResNet50	62.5%	44.0%	32.8%	4.00	4.06	4.15

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).

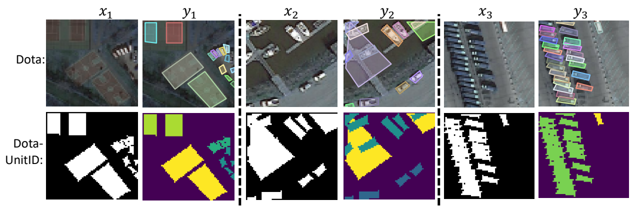


Figure 6. DOTA dataset examples, where the top row shows three original (input, annotations) pairs from the DOTA dataset [41], while the bottom row shows the three corresponding (input, label) pairs from our DOTA-UnitID dataset. The DOTA-UnitID task is to colorize the grayscale annotation mask.

In addition to the Unet-ResNet models seen above, we trained two state-of-the-art segmentation models, a

SegFormer transformer model [42] (12th place in the CityScapes Test leaderboard [7]) and a Lawin transformer model [44] (3rd place in [7]) on the clean Unit Identification 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.

Model	Lawin [44]	SegFormer [42]	R18	R34	R50
SCII	27.0%	27.9%	56.6%	58.5%	62.5%
DOTA	34.1%	35.0%	52.4%	52.8%	53.6%

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 scences 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 behvaiors 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".

Dataset	Frame Count	Agent Positioning	Over-head	GT	Noise Sim
USD [5]	2K	Real Pedestrian	Some		
GCC [40]	15K	Simulated Crowd	Low	✓	
GTA [24]	250K	Simulated Ped/Drive	Some	✓	
MOTSynth [11]	1.4M	Simulated Pedestrian	Some	✓	
TAO [8]	2.2M	Real YouTube	Some		
PySC2[38] + Replays	n/a	Complex / Strategic (from human player)	High	✓	
StarCraftImage (ours)	3.6M	Complex / Strategic (from human player)	Low	✓	✓

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).

References

[1] Naveed Ahmad, Mureed Hussain, Naveed Riaz, Fazli Subhani, Sajjad Haider, Khurram S Alamgir, and Fahad Shinwari. Flood prediction and disaster risk analysis using gis based wireless sensor networks, a review. *Journal of Basic and Applied Scientific Research*, 3(8):632–643, 2013. 6

[2] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019. 2

[3] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011. 6, 8

[4] Matthias Butenuth, Florian Burkert, Florian Schmidt, Stefan Hinz, Dirk Hartmann, Angelika Kneidl, André Borrmann, and Beril Sirmacek. Integrating pedestrian simulation, tracking and event detection for crowd analysis. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 150–157. IEEE, 2011. 1, 8

[5] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–7. IEEE, 2008. 8

[6] P.W.D. Charles. S2clinet-proto repository. `https://github.com/Blizzard/s2client-proto/#replay-packs`, 2015. 3

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 8

[8] Achal Dave, Tarasha Khurana, Pavel Tokmakov, Cordelia Schmid, and Deva Ramanan. Tao: A large-scale benchmark for tracking any object. In *European conference on computer vision*, pages 436–454. Springer, 2020. 8

[9] Riham Elhabyan, Wei Shi, and Marc St-Hilaire. Coverage protocols for wireless sensor networks: Review and future directions. *Journal of Communications and Networks*, 21(1):45–60, 2019. 7

[10] Riham Elhabyan, Wei Shi, and Marc St-Hilaire. Coverage protocols for wireless sensor networks: Review and future directions. *Journal of Communications and Networks*, 21(1):45–60, 2019. 16

[11] Matteo Fabbri, Guillem Brasó, Gianluca Maugeri, Orcun Cetintas, Riccardo Gasparini, Aljovsa Ovsep, Simone Calderara, Laura Leal-Taixé, and Rita Cucchiara. Motsynth: How can synthetic data help pedestrian detection and tracking? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10849–10859, 2021. 1, 2, 8

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 1, 6, 8

[13] Di Guo, Xiaobo Qu, Lianfen Huang, and Yan Yao. Sparsity-based spatial interpolation in wireless sensor networks. *Sensors*, 11(3):2385–2407, 2011. 6

[14] James Z Hare, Junnan Song, Shalabh Gupta, and Thomas A Wettergren. Pose. r: Prediction-based opportunistic sensing for resilient and efficient sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 17(1):1–41, 2020. 7

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 7, 17, 19, 23

[16] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 17

[17] Jeremy Howard and Sylvain Gugger. Fastai: a layered api for deep learning. *Information*, 11(2):108, 2020. 17

[18] Pavel Iakubovskii. Segmentation models pytorch. `https://github.com/qubvel/segmentation_models.pytorch`, 2019. 17

[19] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016. 17

[20] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017. 8

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 19

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 18

[23] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021. 6

[24] Philipp Krähenbühl. Free supervision from video games. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2955–2964, 2018. 1, 8

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[26] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480, 2007. 2

[27] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 2

[28] Wenwen Li and Chia-Yu Hsu. Geoai for large-scale image analysis and machine vision: Recent progress of artificial intelligence in geography. *ISPRS International Journal of Geo-Information*, 11(7):385, 2022. 16

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In

European conference on computer vision, pages 740–755. Springer, 2014. 8

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 7

[31] Ali Masjedi and Melba M Crawford. Prediction of sorghum biomass using time series uav-based hyperspectral and lidar data. In *IGARSS 2020-2020 IEEE International Geoscience and Remote Sensing Symposium*, pages 3912–3915. IEEE, 2020. 6

[32] Sean Plott. Starcraft ii mental checklist, 2011. 2

[33] Joaquin Quiñonero-Candela, Masashi Sugiyama, Neil D Lawrence, and Anton Schwaighofer. *Dataset shift in machine learning*. Mit Press, 2009. 6

[34] Caleb Robinson, Le Hou, Kolya Malkin, Rachel Soobitsky, Jacob Czawlytko, Bistra Dilkina, and Nebojsa Jojic. Large scale high-resolution land cover mapping with multi-resolution data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12726–12735, 2019. 8

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 7, 16, 17

[36] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 7

[37] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 8

[38] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. 2, 3, 6, 8, 13, 18

[39] Angtian Wang, Yihong Sun, Adam Kortylewski, and Alan L Yuille. Robust object detection under occlusion with context-aware compositionalnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12645–12654, 2020. 6

[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

[41] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 7

[42] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021. 8

[43] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 17

[44] Haotian Yan, Chuang Zhang, and Ming Wu. Lawin transformer: Improving semantic segmentation transformer with multi-scale representations via large window attention. *arXiv preprint arXiv:2201.01615*, 2022. 8

[45] Mohamed Younis, Izzet F Senturk, Kemal Akkaya, Sookyoung Lee, and Fatih Senel. Topology management techniques for tolerating node failures in wireless sensor networks: A survey. *Computer networks*, 58:254–283, 2014. 16

[46] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person re-identification: Past, present and future. *arXiv preprint arXiv:1610.02984*, 2016. 8

[47] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 8

[48] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 6

Table 4. Dataset Nutrition Label for the StarCraftImage dataset.

	StarCraftImage		
Motivation	Existing multi-agent spatial reasoning datasets have focused on hyper-realism, leading to large image/video sizes which are expensive to prototype on. Therefore, the ML community lacks an easy-to-use, yet complex behaviored, dataset for prototyping new spatial reasoning algorithms. To fill this gap, we construct a StarCraftImage, a dataset based on StarCraft II replays that has intelligent human behaviors, while being as easy to use as MNIST and CIFAR10 and still enabling complex spatial reasoning tasks.		
Collection Process	We construct this dataset based on 60K StarCraft II replays (taken from Replay Pack 3.16.1 - Pack 1), and use PySC2 to create 3.6 million images which summarize a window of 255 game states from the replays. We additionally collect all relevant metadata such as game outcome and player races. Each window is represented as an image and processed with standard computer vision tools.		
Possible Uses	Since our dataset contains diverse, interesting, and intelligent (even adversarial) behaviors, and thus, it should provide a relevant simple benchmark for rapid prototyping of multi-agent spatial reasoning tasks and algorithms before trying on realistic data. For example, we map common spatial reasoning tasks to event prediction (i.e., game outcome and StarCraft race prediction), target identification (i.e., determining unit type from only knowing unit locations), and missing data imputation using both corruption models and the fog-of-war from the game engine.		
Availability	The StarCraftImage dataset is available for download at https://starcraftdata.davidinouye.com/ and available under a permissive CC BY 4.0 license. The code to recreate (or extend) the dataset extraction and processing, as well as code to load the data, rerun benchmarks, and recreate demos is maintained on GitHub with an MIT license.		
Metadata	Each window in the StarCraftImage dataset is paired with relevant metadata collected during the extraction process. The metadata for each window consists of 52 entries which belong to one of three main subgroups: dynamic metadata (which is information that is specific to each window), static metadata (which is information that is specific to each replay, but does not change across windows within a replay), and computed metadata (which is information added by a user e.g., a label for that window for a classification task). For details please seen "Metadata Description" subsection in the Appendix.		
Dataset Composition	The dataset comprises of three main formats: StarCraftHyper, StarCraftCIFAR10, and StarCraftMNIST		
	StarCraftHyper	StarCraftCIFAR10	StarCraftMNIST
Description	A 340x64x64 px hyperspectral image which has the most information of the three representations. Specifically, there is a channel for each unit type as well as creep and visibility, all collected for each player. Additionally there are channels which contain map info such as map height, placement grid, and unit pathing grids. Additionally, each image is paired with corresponding metadata.	A 32x32 RGB image dataset which was synthesized from the StarCraftHyper dataset such that the Player1 information is embedded into the Blue channel, Player2 to the Red channel, and Neutral information to the Green channel. Each RGB image comes with a corresponding label from one of 10 classes. This was done so it exactly matches the format of the CIFAR10 dataset.	A 28x28 Grayscale image dataset which was synthesized from the StarCraftCIFAR10 dataset such that the Player1 information is pushed into the range of [0.55, 1] px values, Player2 to the [0.0, 0.45] px values, and Neutral information to the range of [0.48, 0.52], and overlaid with decreasing precedence of P1, P2, N. Each grayscale image comes with a corresponding label from one of 10 classes. This was done so it exactly matches the format of the MNIST dataset.
How to access	After downloading the dataset, one can use the StracraftImage dataset class from the code repo. E.g., for accessing training data: sc2_train = StarCraftImage(data_root, subdir, train=True)	This can be used similar to CIFAR10 via unpickling the compressed StarCraftCIFAR10.tar.gz file. Or, the whole StarCraftImage dataset can be loaded in RGB format using the StarCraftCIFAR10 python class.	This can be used similar to MNIST via uncompressing the StarCraftMNIST.gz file. Or, the whole StarCraftImage dataset can be loaded in grayscale format using the StarCraftMNIST python class.
Data structure	If in a sparse format: sparse matrix with shape 340x64x64 If in a dense format: dense matrix with shape Nx64x64 where N is the max # of units at one location in a batch	Tuple with (3 x 32 x 32 numpy matrix (aka, RGB image), integer label between 0-9) -- matching the exact format of CIFAR10	Tuple with (28 x 28 numpy matrix (aka, grayscale image), integer label between 0-9) -- matching the exact format of MNIST
Number of Samples	3,607,787 images	50K training images, 10K test images	60K training images, 10K test images
Recommended Data Splits	Depends on task, see Metadata Description, Splitting dataset in corresponding paper.	Already split along into 10 balanced classes based off of (map_name, is_from_begining_of_game). See "Splitting dataset to k Classes" in paper for details.	Already split along into 10 balanced classes based off of (map_name, is_from_begining_of_game). See "Splitting dataset to k Classes" in paper for details.
Noise levels	No noise, but noise can be added as a preprocessing step (see "Simulated Data Corruption Models") in paper. However, if multiple units of the same type cross the same location within a window, only the most recent crossing will be recorded.	In addition to the StarCraftHyper noise, the unit type ID information is removed from this representation.	In addition to the StarCraftHyper noise, the unit type ID information is removed from this representation.
Contains Confidential/ Person Identifiable Data?	None	None	None

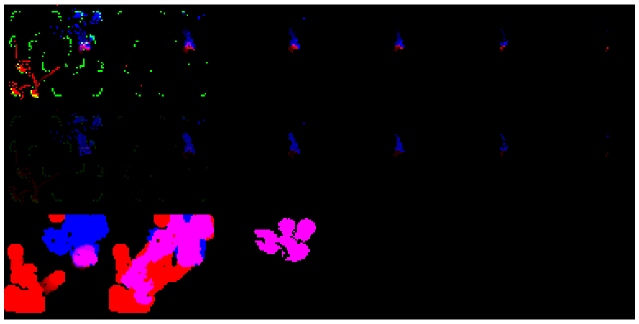
A. Dataset Availability, Licensing, and Management

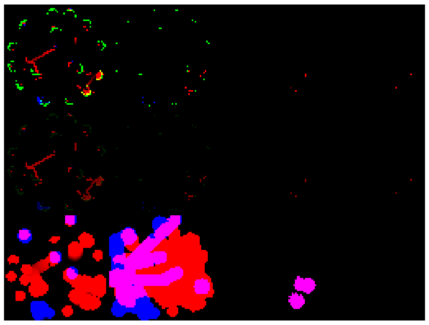
The StarCraftImage dataset is available for download at https://starcraftdata.davidinouye.com/ which contains the full extracted data from the 3.6 million windows, the metadata for all windows, the StarCraftMNIST train/test datasets, and the StarCraftCIFAR10 train/test datasets. The code to recreate (or extend) the dataset extraction and processing, as well as code to load the data, rerun benchmarks, and recreate demos, can be found at the previous link, and is maintained on GitHub. Instructions for loading and using the dataset can be found in the README in the dataset as well as in the code repository. The dataset has been openly published under a permissive CC BY 4.0 license, and the code has been openly published on GitHub with an MIT license. The authors bear all responsibility in case of violation of rights and confirm the CC license for the provided datasets.

B. Direct Loading of Window Data

While we encourage using the corresponding PyTorch dataset classes that we have developed (one class for each representation) to load in StarCraftImage data, one can also directly access the data by loading in the relevant `.png` file and metadata row for each window. To assist with this direct data access, we now describe the data structure used to store the image data for each window (i.e., how to correspond each `.png` to the hyperspectral format H discussed in subsection 2.2).

As a reminder, the bag-of-units representation collapses the channel, axis of our hyperspectral image H into k ID matrices and k timestamp matrices of size $(64, 64)$, where the ID matrix contains the PID 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, seen in the top right of Fig. 2. We can further compress this bag-of-units representation by stacking the bag-of-units for player 1, player 2, and neutral to match the RGB structure of a `.png` image, where the red channel corresponds to player 2, the blue channel to player 1, and the green channel corresponds to the neutral units (e.g., mineral deposits). To fit the structure of a RGB image, we can tile the bag-of-units into rows where the first RGB row corresponds to the timestamp matrices and the second row corresponds to the PID matrices. Finally, we add a third row to record the map state information for each player, specifically, the map state row contains: [RGB 'is_visible', RGB 'is_seen', RGB 'creep']. This leaves us with a RGB `.png` image with height $3 * 64$ and width $k * 64$. Examples of this can be seen in Fig. 7.


(a) Max number of units overlapped is 6.


(b) Max number of units overlapped is 4.

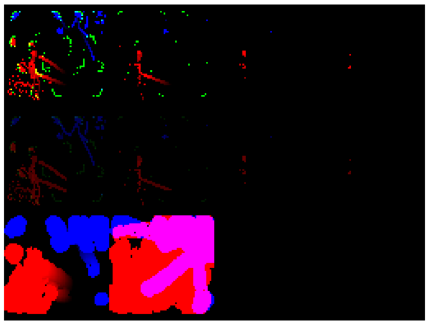

(c) Max number of units overlapped is 4.

Figure 7. Three examples of the dense bag-of-units `.png` show how the hyperspectral image data for a window is stored in a simple `.png` file. The hyperspectral information is represented by tiling 64 x 64 RGB images. The first row is the unit timestamps (0-255), the second row is the unit ids (0-255), and the third row contains map state information (`is_visible`, `is_seen`, and `creep`). The blue channel encodes player 1, the red channel encodes player 2, and the green channel encodes neutral elements. We note how the width of the image varies, as it is determined by how many overlapping units at the same location there are in that window. For example, the top example had 6 units overlapping at one location, so it has a width of six 64x64 images whereas the other two only had a max of 4 units overlapping at one location.

C. Broader Impact

We introduced this spatial reasoning dataset to allow for quick prototyping of complex multi-agent spatial reasoning ML models and easy benchmarking to compare models (similar to the use cases of MNIST and CIFAR10). While our dataset contains complex dynamics that are based on real human actions, it is still a simulation-based dataset, and thus methods tested on this dataset should be further tested in real-world cases before a real-world deployment. Additionally, since this dataset is a general spatial reasoning dataset that can either be directly applied or easily adapted to real-world cases, there is an opportunity for this dataset to be used for tasks that have a negative societal impact (e.g., unauthorized surveillance/tracking). We do not condone the usage of this dataset for the development of harmful models for such negative tasks. Furthermore, since our replays are created by humans and have personal metadata like the actions per minute (APM) and match-making rating (MMR) for each player, this could possibly be used to uniquely attribute a replay to a player. However, this likely is only possible for extreme APM, MMR values (e.g., the top MMR value), and even then, APM is match-specific and a player's MMR updates with each match. Finally, all replays were freely uploaded in an open-source manner and (to the best of our knowledge) contain no personally identifying information (e.g., name of the uploader, upload IP address, etc.).

D. Metadata Description and Suggested Class-wise Splits

In this section we discuss the metadata collected alongside the image data for each window in StarCraftImage.

D.1. Metadata Description

For each window in StarCraftImage, we also collected relevant match/window metadata, which can be seen in Fig. 8. Each entry belongs to one of three main subgroups: dynamic metadata (which is information that is specific to each window), static metadata (which is information that is specific to each replay but does not change across windows within a replay), and computed metadata (which is information added by a user e.g., a label for that window for a classification task). Namely, the dynamic metadata contains a vector of tabular features for both player1 and player2 such as resource counts for each player. Specfically, these tabular features correspond to: `['player_id', 'minerals', 'vespene', 'food_used', 'food_cap', 'food_army', 'food_workers', 'idle_worker_count', 'army_count', 'warp_gate_count', 'larva_count']`. Additionally, the dynamic metadata contains: `date_time_str` which is a string representing the date that window was added to the dataset, `frame_idx` which is the frame index within a replay which corresponds to the last frame included in a window (e.g., if a window's `dynamic.frame_idx`=1000 then that window summarizes frames 745 to 1000 of the given replay). The dynamic `window_percent` corresponds to how far into a match that window takes place, represented as a fraction. For the `static` metadata, this is broken into `game_info` (which corresponds to information that is mostly match specific such as map information) and `replay_info` (which replays to information about the replay file and the players contained in the file). In the `game_info`, the race information is encoded following the PySC2 convention where Terran = 1, Zerg = 2, Protoss = 3, and Random = 4. The player-level information can be found in the `replay_info.player_stats` section where APM corresponds to the player's Actions Per Minute for that match and the player's MMR is the player's Match Making Rating (which can be thought of as a skill-level determined by Blizzard, where higher is more skilled). We include these metadata to give more details about each window, but most importantly to allow a user to split the StarCraftImage dataset along these features for a specific task. For example, if one is developing a model which should generalize to new environments, a user can split this dataset on the `static.game_info.map_name` feature, and use windows from five of the seven maps for training/validation and test on windows from the remaining two maps. To aid in determining filtering methods, histograms for numeric entries within the metadata can be seen in Fig. 9.

D.2. Splitting dataset to k Classes

When working with global-spatial reasoning tasks (e.g., whole-image classification), the question of how to split this dataset into k classes arises. Thus, we suggest some possible ways to split the dataset along with simple benchmark accuracy values for comparing the difficulty of the splits for two of the most common classification schemes ML: binary classification (k=2) and 10-way classification (k=10). For all splitting experiments we mention running below, we use the same ConvNet architecture of two convolutional layers with max-pooling in-between, three fully connected layers, all with ReLU activations, and train for 20 epochs using SGD with a learning rate of 0.001 and momentum of 0.9.

For binary classification, the obvious choice is to perform match outcome prediction (i.e., "did Player 1 win?"), as mentioned in the main body of this work. While an important task, this can be difficult even for human experts watching a StarCraftII E-sports event as well as difficult for an AI to solve/ For example, the best model in [38] can only achieve 65% outcome prediction training accuracy for frames taken 15 minutes into a game, and when tested on the grayscale StarCraftMNIST and RGB StarCraftCIFAR10 datasets split on the match outcome variable (which include windows throughout all points in the game rather than just mid-to-end game), we report 57.9% and 59.4% test accuracy, respectively, on the same task. A binary prediction task that is more easily interpreted is the task of predicting if a window comes from the first half or second half of a replay ("is `dynamic.window_percent` ¡ 0.5?"). This is also somewhat easier to solve (we report a testing ac-

Metadata for Window: 3,607,787

dynamic.date_time_format	%Y-%m-%d_%H-%M-%S
dynamic.date_time_str	2023-03-16_19-06-53
dynamic.frame_idx	14855
dynamic.num_windows	58
dynamic.timestamp	1679008013
dynamic.window_idx	57
dynamic.window_percent	0.982758621
static.extracted_image_size	[64, 64]
static.game_info.map_name	Odyssey LE
static.game_info.mod_names	['Mods/Core.SC2Mod', 'Mods/Liberty.SC2Mod', 'Mods/Swarm.SC2Mod', 'Mods/Void.SC2Mod', 'Battle.net/Cache/f1/9f/f19f56b...8ea3a5.s2ma']
static.game_info.options.raw	TRUE
static.game_info.options.score	TRUE
static.game_info.player_info.player_1.race_actual	3
static.game_info.player_info.player_1.race_requested	3
static.game_info.player_info.player_2.race_actual	1
static.game_info.player_info.player_2.race_requested	1
static.game_info.start_raw.map_size.x	168
static.game_info.start_raw.map_size.y	184
static.game_info.start_raw.pathing_grid.bits_per_pixel	8
static.game_info.start_raw.pathing_grid.size.x	168
static.game_info.start_raw.pathing_grid.size.y	184
static.game_info.start_raw.placement_grid.bits_per_pixel	8
static.game_info.start_raw.placement_grid.size.x	168
static.game_info.start_raw.placement_grid.size.y	184
static.game_info.start_raw.playable_area.p0.x	8
static.game_info.start_raw.playable_area.p0.y	8
static.game_info.start_raw.playable_area.p1.x	160
static.game_info.start_raw.playable_area.p1.y	164
static.game_info.start_raw.start_locations.x	143.5
static.game_info.start_raw.start_locations.y	24.5
static.game_info.start_raw.terrain_height.bits_per_pixel	8
static.game_info.start_raw.terrain_height.size.x	168
static.game_info.start_raw.terrain_height.size.y	184
static.num_frames_per_window	255
static.replay_info.base_build	55958
static.replay_info.data_build	55958
static.replay_info.data_version	5BD7C31B44525DAB46E64C4602A81DC2
static.replay_info.game_duration_loops	14855
static.replay_info.game_duration_seconds	663.2159424
static.replay_info.game_fps_calculated	22.39843624
static.replay_info.game_version	3.16.1.55958
static.replay_info.player_stats.player_1.apm	242
static.replay_info.player_stats.player_1.mmr	3192
static.replay_info.player_stats.player_1.result	Loss
static.replay_info.player_stats.player_1.result_int	2
static.replay_info.player_stats.player_2.apm	103
static.replay_info.player_stats.player_2.mmr	3120
static.replay_info.player_stats.player_2.result	Win
static.replay_info.player_stats.player_2.result_int	1
static.replay_name	378bfb2ce94.....e9451dc93.SC2Replay
computed.target_id	9
computed.target_label	('Odyssey LE', 'End')

Figure 8. An example of the metadata collected for each window of a replay. Descriptions of the key, value pairs are given in Appendix D.

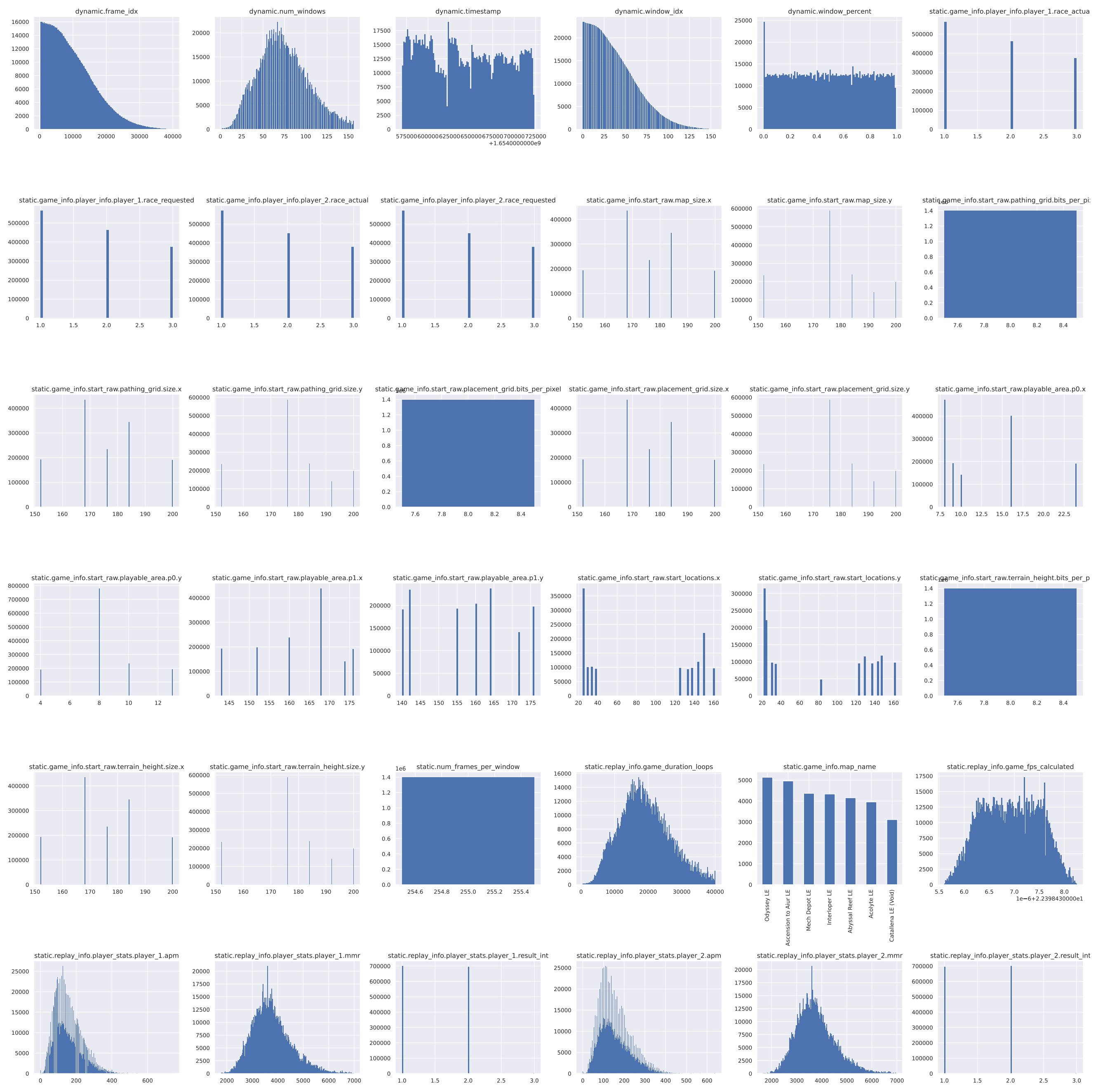


Figure 9. Histograms of the numerical values entries in the metadata (best viewed zoom in). To keep the x-axis interpretable, we excluded any outliers which have a game_duration_loop > 40,000 and any replays which hold a negative MMR value for either player (which is likely a result of a bug in the `.SC2Replay` file). Any histograms which are fully rectangular are static values at the center point of the bin (e.g., `static.num_frames_per_window`=255 for all windows).

curacy of 74.3% and 76.9% for grayscale StarCraftMNIST and RGB StarCraftCIFAR10 datasets with this split, respectively), while still requiring a model to learn environmental dynamics to solve.

Splitting the StarCraftImage dataset for 10-way classification (e.g., StarCraftMNIST and StarCraftCIFAR10) is more difficult since there are no natural way to split the dataset 10 ways. The splitting method which most closely aligns with spatial reasoning problems is likely splitting via player race information + match outcome prediction, as this requires learning battlefield strategy/dynamics (for outcome prediction) and understanding of unit information (for player race prediction). Thus, this is the 10-way splitting method suggested in subsection 2.4 in the main body of this work. However, from a purely ML perspective, this is an extremely difficult classification problem; which is supported by our testing accuracy of 26.4% and 28% for StarCraftMNIST and StarCraftCIFAR10 datasets created with the split detailed in subsection 2.4. Thus, for purposes with a stronger abstract ML focus, we suggest a 10-way split that combines the "is_beginning_or_end" binary variable from above with a prediction of the map_name. This task requires the model to learn environment information for the map_name and battlefield dynamics for the beginning/end prediction and is more solvable than a split requiring match outcome prediction. Specifically, since there are 7 maps in total, we suggest subsampling to only 5 maps, then further splitting each of these 5 map groups into "beginning " and "end" groups (based on whether or not the window takes place in the beginning 50% of the match), to get 10 classes. Examples from such a split can be seen in figure Fig. 13, and in our experiments, we received a testing accuracy of 77.2% and 77.9% for StarCraftMNIST and StarCraftCIFAR10 datasets, respectively. Heuristically, we have found this 10-way split to be a good balance between problem realism and difficulty/human interpretability, and thus we will be using it for the following task demonstrations.

E. Benchmark Evaluations On Multi-Agent Spatial Reasoning Tasks

In this section, we report benchmark results on 4 benchmark multi-agent spatial reasoning tasks, which incorporate training 60 U-Net-based [35] models. Unlike the benchmark results in the main paper, (e.g., Table 1), the results seen here and throughout the rest of the appendix are trained on a smaller StarCraftImage dataset (specifically, these results are generated from a random 1.8 million window subset, i.e. a random 50% subset of the main dataset). This was done to allow for faster model training, thus allowing us to add more models beyond the three ResNet models seen in the main paper (specifically, 60 models were trained on this smaller dataset).

The four benchmark tasks consist of two tasks on unit type identification and two tasks for unit tracking (next hyperspectral window prediction). Both task sets consist of first training and evaluating on "clean" (unaltered) data as well as a second task of training on data which is first passed through a simulation of a noisy sensor network. This simulation consists of 50 sensors with a radius of 5.5 pixels with different sensor placement methodologies (e.g., grid, random, quasi-random, and diagonal barrier) and communication failures during sensor fusion. For reference, grid-based placements are commonly used in environmental monitoring data sets where they are optimally placing sensors to cover the environment space [28]. Random and Quasi-random deployments are typical when sensors cannot be placed optimally (e.g., when they are dropped out of planes/helicopters). The barrier placements come from the well-studied barrier coverage problem, which is commonly used for border surveillance, road monitoring, etc. For further examples studying these different coverage types see [10] which provides a taxonomy for different coverage protocols including the ones mentioned above. For a study on different failure types, including the ones seen here, see [45]. Examples of the different placement types can be seen in Fig. 10.

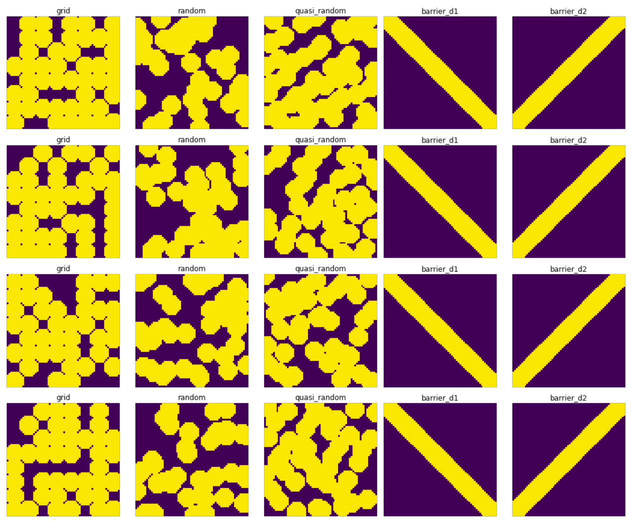


Figure 10. Example masks for sensor network simulations containing 50 sensors with five different placement strategies [grid, random, quasi-random, barrier-d1, barrier-d2] (where yellow is a location that is visible). Each sensor has a radius of 5.5. pixels and a 20% chance of being dropped during the sensor fusion process (e.g., due to a communication failure).

E.1. Unit Type Identification

As discussed in section 3, the goal of this task is to train a model to take a 64 x 64 RGB image (similar to the format of the StarCraftCIFAR10 images) as input and to output a 64 x 64 matrix corresponding to the unit ids for each location. This problem is analogous to fine-grained multi-object detection, where given raw images (e.g., satellite images), our goal is to predict what kind of unit is present at each location (if there is a unit present at all). For example, for a given window, if there is a non-zero value at location, (Red, i, j), then we know an enemy unit passed through location (i, j) – our goal now is to figure out that unit's type (e.g., ZERG_QUEEN, PROTOSS_ORACLE, etc.).

We used the unaltered RGB images as input and synthesized the 64 x 64 unit id label matrix from the StarCraftHyper dataset. In cases where multiple units were present at

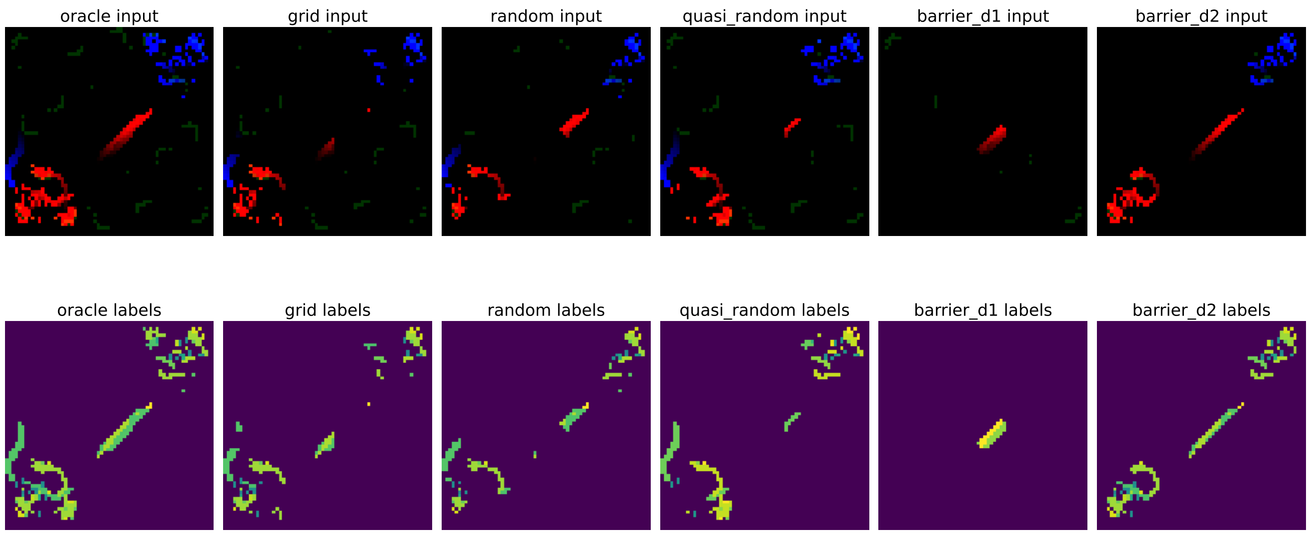


Figure 11. Example input-output pairs for the sample from the Unit Type Identification task with the clean representation as well as representations which are prepossessed by corruption simulations via faulty sensing networks five different placement strategies [grid, random, quasi-random, barrier-d1, barrier-d2] (where yellow is a location that is visible). Note that during training, the labels are also masked to simulate training on noisily labeled data.

the same location (which is possible since a window covers a span of 255 in-game frames/seconds), we set the id for that location to that of the most recent unit. We used the FastAI library [17] to train six U-Net models [35] with backbones: ResNet18, ResNet34 [15], Squeezenet1_0, Squeezenet1_1 [19], XResNet18, XResNet34 [16], on this dataset for 10 epochs with cross-entropy loss, a batch size of (512 for ResNet*, 256 for Squeezenet*, and 512 for XResNet*), and default FastAI configuration settings. We trained each of the above models on the clean (i.e., noiseless) dataset as well as on all five sensor placement variations where both the input images and output label matrices were masked by the generated sensor masks (see Fig. 11 for examples), yielding 36 models in total. During testing, we tested all models on held-out *clean* data, which simulates the situation where one has noisy training data but wants to evaluate an algorithm with respect to clean ground truth data for final evaluation. We report the Cross-Entropy error, Unit Accuracy (was the unit type correctly predicted), and the averaged Dice coefficient for all models in Table 6. As expected, there is significant performance derogation across models when moving from the clean data to the noisy data, and this is most evident in the diagonal barrier placements. As seen by the unit accuracy metric, this is a hard problem (there are 340 possible unit ids for each location), which we hope future work will be able to innovate upon.

E.2. Next Hyperspectral Window Prediction

Here our goal is to use the StarCraftHyper dataset to train a model on the common spatial reasoning task of object tracking. For this, we frame this task as: a given replay, we want to take the k^{th} hyperspectral window as input (with shape 340 x 64 x 64) and have a model forecast how all units will move to their locations in the $k+1$ hyperspectral window. Specifically, the model must output the *difference* (i.e., movement) between the two hyperspectral windows ($ground\_truth = (H_{k+1} - H_k)$ and has shape (340, 64, 64)).

To do this, we use the FastAI library [17] and the SMP library [18] to train four U-Net [35] models with backbones: ResNet18, ResNet34, ResNet50 [15], and ResNext50_32x4d [43] on both clean versions of the dataset and five noisy versions of the dataset matching the five sensor network simulations (see Fig. 12 for examples). Due to the large size of the samples, we use a batch size of 20 across all models, and to accelerate the training process we randomly subsample the overall training data to 60K window pairs. We train all models for 10 epochs using the Mean-Squared Error loss and otherwise default FastAI configuration settings. We then test our models on 10K held out *clean* window pairs, and report the MSE loss. Additionally, we bin the $ground_t ruth$ test data into $[-1, 0, 1]$ where location (u_{id}, i, j) is -1 if a specific unit type *left* location

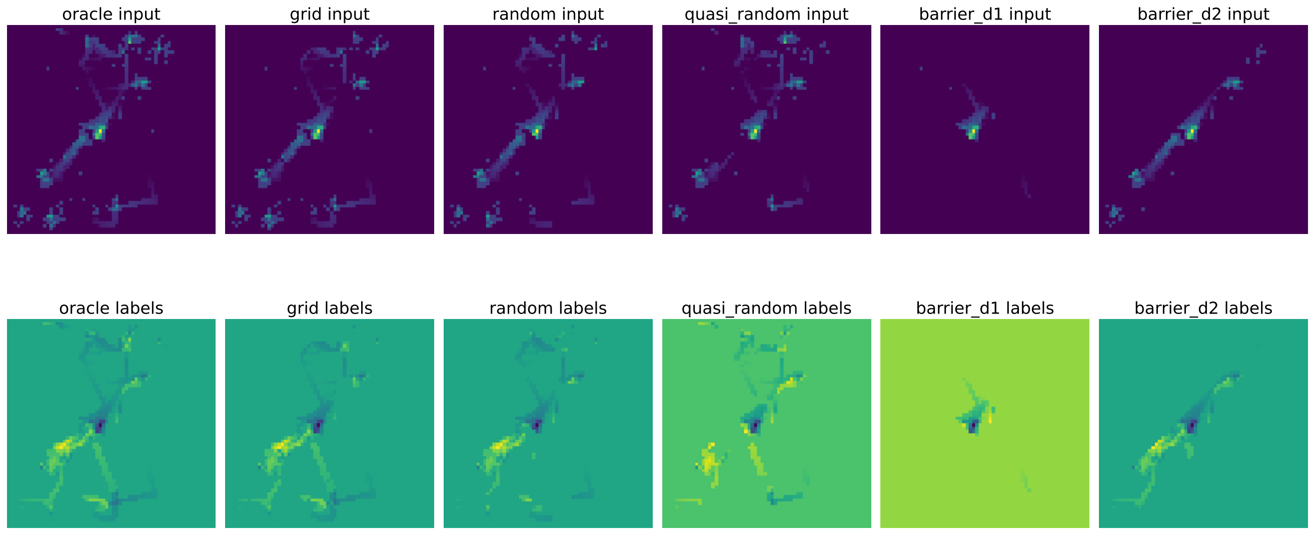


Figure 12. Example input output pairs of the same sample from the Next Hyperspectral Window prediction task with the clean representation as well as representations which are prepossessed by corruption simulations via faulty sensing networks with five different placement strategies [grid, random, quasi-random, barrier-d1, barrier-d2] (where yellow is a location that is visible). Note that during training, the labels are also masked to simulate training on noisily labeled data.

(i, j) from window k to window $k + 1$ (i.e. the unit moved away from that spot), 0 means no movement happened, and 1 means unit u_{id} moved *into* location (i, j). With this, we report the False Positive Rate (% of times a model predicts movement happens when it doesn't), True Positive Rate (+) (% of the time a model correctly predicts $+1$, a unit moved into a location), and the True Positive Rate (-). The results for this can be seen in Table 7, where while the MSE is lowest for the models trained on the clean data, these models also tend to have a higher false positive rate.

F. Additional Demonstrations of ML tasks on the StarCraftImage Dataset

Predicting match outcome As mentioned previously, this is a very difficult task which can be hard even for human experts. [38] performed the match outcome prediction task on features extracted from PySC2, and report only 65% outcome prediction training accuracy for frames taken as far as 15 minutes into a game. For this task, we use two datasets: one which has the grayscale window formats of StarCraftMNIST and the other with the RGB window formats of StarCraftCIFAR10. These datasets are constructed such that the train/test splits for the positive class (Player1IsWinner) and negative class (Player1IsNotWinner) are evenly balanced with (60k, 10k) and (50k, 10k) train, test examples for the grayscale and RGB datasets, respectively. We found the performance of our ConvNet model (mentioned in the previous section) to be similar to that of [38], where we see only 57.9% test accuracy on the grayscale window dataset and 59.4% test accuracy on the RGB window dataset.

10 class classification Here we use the map_name + is_beginning_or_end 10-way class split mentioned in subsection D.2 and seen in Fig. 13. We apply our ConvNet model to the StarCraftMNIST and StarCraftCIFAR10 datasets. After training for 20 epochs, we received a testing accuracy of 77.2% and 77.9% for StarCraftMNIST and StarCraftCIFAR10, respectively.

Imputing occluded objects For this task, we simulate occluded objects via randomly masking out a circle of pixels with a radius 5px, which can be seen to approximate cloud coverage or a fused image with missing data (see left of Fig. 15 for examples). For this task, our goal is to impute the missing information and to do this we implement a VAE model [22] to denoise the inpainted image. Our VAEImputer encoder and decoder both consist of six convolutional layers (with a 3x3 kernel, stride of 2, and padding of 1), each with batch norm and leakyReLu activations and with one fully connected layer in-between. For the encoder, the convolutional layers consist of $[3, 32, 64, 128, 128, 512]$ channels, the decoder convolutional layers consist of $[512, 128, 128, 64, 32, 3]$ channels, and the fully connected layer takes in the 512 channels and projects this to the 64 dimensional μ and σ latent parameter space. We train the model for 100 epochs on StarCraftMNIST (which consists of the same 60k training images as in the 10-class map-based split), with Adam optimizer [21] with a learning rate of 5e-3 and $\beta = (0.9, 0.999)$. The results can be seen in Fig. 15, where the model does well at imputing the missing data, at the expense of blurring the image due to artifacting from the VAE.

Unit type identification In this task we simulate the case where a model is only given a raw image showing the presence of a unit, but not what type of unit it is. This can be mapped to a colorization task by first taking an RGB sample from StarCraftCIFAR10 (where each channel corresponds to a specific player's units) and averaging along the color dimension to get a grayscale image that has no owner information. Then, to recover the owner information (i.e. whether a unit belongs to Player 1, Player 2, or Neutral), we colorize the image by predicting which channel each unit should belong to. To do this, we use a ResNet-101 model [15], which has been adapted to have an output dimensionality of 32x32x3 (the number of pixels in a StarCraftCIFAR10 image). We train this model on the StarCraftCIFAR10 dataset for 100 epochs using the Adam optimizer [21] with a learning rate of 5e-3 and $\beta = (0.9, 0.999)$. The results can be seen in Fig. 15, where the model correctly identifies between neutral and non-neutral units, but has trouble determining whether a unit belongs to Player 1 or Player 2 due to both players having random starting corners of the map.

Table 5. A per-window frequency table for the non-neutral units across all windows in the StarCraftImage dataset, where Avg. Per Win. corresponds to the average number of times that unit is present per window, Perc. is the number of times that unit appeared divided by the total unit appearances, and Cum Perc. is the cumulative percentage up to that row. Note, this analysis was performed on the 30k replay subset but should be quite similar to the 60k frequencies.

	Avg.				Avg.				Avg.		
	Per		Cum		Per		Cum		Per		Cum
Unit Name	Win.	Perc.	Perc.	Unit Name	Win.	Perc.	Perc.	Unit Name	Win.	Perc.	Perc.
TERRAN_SCV	29.4	12.4%	12.4%	PROTOSS_CARRIER	0.7	0.3%	92.3%	PROTOSS_DARKSHRINE	0.1	0.0%	99.5%
ZERG_DRONE	21.5	9.1%	21.5%	TERRAN_RAVEN	0.7	0.3%	92.6%	PROTOSS_FLEETBEACON	0.1	0.0%	99.5%
PROTOSS_PROBE	19.7	8.3%	29.8%	PROTOSS_GATEWAY	0.6	0.3%	92.9%	TERRAN_THORAP	0.1	0.0%	99.6%
ZERG_ZERGLING	16.0	6.7%	36.5%	ZERG_SPAWNINGPOOL	0.6	0.2%	93.1%	TERRAN_FUSIONCORE	0.1	0.0%	99.6%
TERRAN_MARINE	14.1	5.9%	42.5%	PROTOSS_WARPPRISM	0.6	0.2%	93.4%	TERRAN_VIKINGASSAULT	0.1	0.0%	99.6%
ZERG_OVERLORD	9.0	3.8%	46.2%	TERRAN_HELLIONTANK	0.6	0.2%	93.6%	ZERG_LOCUSTMPFLYING	0.1	0.0%	99.6%
TERRAN_MEDIVAC	5.7	2.4%	48.7%	TERRAN_COMMANDCENTER	0.6	0.2%	93.9%	ZERG_CHANGELING	0.1	0.0%	99.7%
ZERG_ROACH	5.1	2.2%	50.8%	ZERG_EVOLUTIONCHAMBER	0.6	0.2%	94.1%	TERRAN_STARPORTFLYING	0.1	0.0%	99.7%
ZERG_CREEPTUMORBURROWED	5.0	2.1%	52.9%	TERRAN_FACTORYTECHLAB	0.6	0.2%	94.4%	TERRAN_REACTOR	0.1	0.0%	99.7%
ZERG_HYDRALISK	4.6	1.9%	54.9%	TERRAN_THOR	0.5	0.2%	94.6%	ZERG_LOCUSTMP	0.0	0.0%	99.7%
PROTOSS_STALKER	4.4	1.9%	56.7%	PROTOSS_COLOSSUS	0.5	0.2%	94.8%	ZERG_ROACHBURROWED	0.0	0.0%	99.7%
TERRAN_MARAUDER	4.3	1.8%	58.6%	PROTOSS_HIGHTEMPLAR	0.5	0.2%	95.0%	ZERG_ULTRALISKCAVERN	0.0	0.0%	99.8%
PROTOSS_PYLON	4.1	1.7%	60.3%	PROTOSS_CYBERNETICSCORE	0.5	0.2%	95.2%	PROTOSS_ORACLESTASISTRAP	0.0	0.0%	99.8%
ZERG_LARVA	3.8	1.6%	61.9%	PROTOSS_DARKTEMPLAR	0.5	0.2%	95.4%	TERRAN_GHOSTACADEMY	0.0	0.0%	99.8%
TERRAN_SUPPLYDEPOT	3.4	1.4%	63.3%	ZERG_SPINECRAWLER	0.4	0.2%	95.6%	TERRAN_TECHLAB	0.0	0.0%	99.8%
ZERG_QUEEN	3.3	1.4%	64.7%	TERRAN_WIDOWMINEBURROW	0.4	0.2%	95.7%	ZERG_SPINECRAWLERUPROOTE	0.0	0.0%	99.8%
PROTOSS_ZEALOT	2.9	1.2%	65.9%	TERRAN_BATTLECRUISER	0.4	0.2%	95.9%	ZERG_LURKERDENMP	0.0	0.0%	99.8%
TERRAN_SIEGETANK	2.8	1.2%	67.1%	PROTOSS_INTERCEPTOR	0.4	0.2%	96.1%	PROTOSS_WARPPRISMPHASIN	0.0	0.0%	99.8%
TERRAN_REFINERY	2.7	1.2%	68.3%	TERRAN_STARPORTREACTOR	0.4	0.2%	96.2%	PROTOSS_PYLONOVERCHARGE	0.0	0.0%	99.9%
ZERG_MUTALISK	2.7	1.1%	69.4%	PROTOSS_STARGATE	0.4	0.2%	96.4%	ZERG_OVERLORDCOCOON	0.0	0.0%	99.9%
ZERG_EXTRACTOR	2.3	1.0%	70.3%	PROTOSS_FORGE	0.4	0.2%	96.6%	ZERG_CHANGELINGZEALOT	0.0	0.0%	99.9%
TERRAN_BARRACKS	2.3	1.0%	71.3%	TERRAN_ARMORY	0.4	0.2%	96.7%	ZERG_RAVAGERCOCOON	0.0	0.0%	99.9%
ZERG_BANELING	2.3	0.9%	72.2%	PROTOSS_ROBOTICSFACILITY	0.4	0.1%	96.9%	ZERG_CHANGELINGZERGLINGWI	0.0	0.0%	99.9%
TERRAN_VIKINGFIGHTER	2.2	0.9%	73.2%	ZERG_BANELINGNEST	0.3	0.1%	97.0%	ZERG_SPORECRAWLERUPROOTE	0.0	0.0%	99.9%
PROTOSS_ADEPT	2.2	0.9%	74.1%	TERRAN_STARPORTTECHLAB	0.3	0.1%	97.1%	ZERG_GREATERSPIRE	0.0	0.0%	99.9%
ZERG_EGG	2.0	0.8%	74.9%	ZERG_SWARMHOSTMP	0.3	0.1%	97.2%	ZERG_CHANGELINGMARINESHIE	0.0	0.0%	99.9%
PROTOSS_ASSIMILATOR	1.9	0.8%	75.7%	PROTOSS_ADEPTPHASESHIFT	0.3	0.1%	97.3%	TERRAN_AUTOTURRET	0.0	0.0%	99.9%
TERRAN_SUPPLYDEPOTLOWERE	1.9	0.8%	76.5%	PROTOSS_TWILIGHTCOUNCIL	0.3	0.1%	97.4%	ZERG_NYDUSNETWORK	0.0	0.0%	99.9%
ZERG_OVERSEER	1.9	0.8%	77.3%	ZERG_LAIR	0.3	0.1%	97.6%	ZERG_ZERGLINGBURROWED	0.0	0.0%	99.9%
TERRAN_REAPER	1.8	0.8%	78.1%	ZERG_LURKERMP	0.3	0.1%	97.7%	ZERG_LURKERMPEGG	0.0	0.0%	100.0%
TERRAN_MISSILETURRET	1.7	0.7%	78.8%	ZERG_ROACHWARREN	0.2	0.1%	97.8%	TERRAN_KD8CHARGE	0.0	0.0%	100.0%
TERRAN_HELLION	1.7	0.7%	79.5%	TERRAN_FACTORYREACTOR	0.2	0.1%	97.9%	ZERG_CHANGELINGMARINE	0.0	0.0%	100.0%
ZERG_HATCHERY	1.6	0.7%	80.2%	TERRAN_ORBITALCOMMANDFL	0.2	0.1%	97.9%	ZERG_SWARMHOSTBURROWED	0.0	0.0%	100.0%
PROTOSS_WARPGATE	1.6	0.7%	80.9%	ZERG_CREEPTUMOR	0.2	0.1%	98.0%	PROTOSS_DISRUPTORPHASED	0.0	0.0%	100.0%
TERRAN_MULE	1.5	0.6%	81.5%	ZERG_BROODLING	0.2	0.1%	98.1%	ZERG_BROODLORDCOCOON	0.0	0.0%	100.0%
PROTOSS_IMMORTAL	1.5	0.6%	82.2%	ZERG_BANELINGCOCOON	0.2	0.1%	98.2%	ZERG_DRONEBURROWED	0.0	0.0%	100.0%
PROTOSS_OBSERVER	1.4	0.6%	82.8%	TERRAN_BUNKER	0.2	0.1%	98.3%	ZERG_NYDUSCANAL	0.0	0.0%	100.0%
TERRAN_ORBITALCOMMAND	1.3	0.6%	83.3%	PROTOSS_TEMPEST	0.2	0.1%	98.3%	ZERG_BANELINGBURROWED	0.0	0.0%	100.0%
PROTOSS_NEXUS	1.3	0.5%	83.9%	TERRAN_GHOST	0.2	0.1%	98.4%	ZERG_CHANGELINGZERGLING	0.0	0.0%	100.0%
TERRAN_CYCLONE	1.2	0.5%	84.4%	ZERG_BROODLORD	0.2	0.1%	98.5%	ZERG_INFESTORTERRAN	0.0	0.0%	100.0%
TERRAN_LIBERATOR	1.2	0.5%	84.9%	TERRAN_PLANETARYFORTRESS	0.2	0.1%	98.6%	ZERG_TRANSPORTOVERLORDCC	0.0	0.0%	100.0%
TERRAN_WIDOWMINE	1.2	0.5%	85.4%	TERRAN_BARRACKSFLYING	0.2	0.1%	98.6%	TERRAN_POINTDEFENSEDRONE	0.0	0.0%	100.0%
PROTOSS_PHOENIX	1.1	0.5%	85.9%	ZERG_SPIRE	0.2	0.1%	98.7%	ZERG_HYDRALISKBURROWED	0.0	0.0%	100.0%
PROTOSS_ORACLE	1.1	0.5%	86.3%	ZERG_INFESTATIONPIT	0.1	0.1%	98.8%	ZERG_INFESTEDTERRANSEGG	0.0	0.0%	100.0%
ZERG_CORRUPTOR	1.1	0.5%	86.8%	PROTOSS_DISRUPTOR	0.1	0.1%	98.8%	TERRAN_NUKE	0.0	0.0%	100.0%
PROTOSS_MOTHERSHIPCORE	1.0	0.4%	87.2%	TERRAN_COMMANDCENTERFLY	0.1	0.1%	98.9%	player_(Unknown)	0.0	0.0%	100.0%
PROTOSS_PHOTONCANNON	1.0	0.4%	87.7%	ZERG_INFESTOR	0.1	0.1%	98.9%	ZERG_QUEENBURROWED	0.0	0.0%	100.0%
ZERG_RAVAGER	1.0	0.4%	88.1%	ZERG_HYDRALISKDEN	0.1	0.1%	99.0%	ZERG_PARASITICBOMBDUMMY	0.0	0.0%	100.0%
TERRAN_BARRACKSREACTOR	1.0	0.4%	88.5%	ZERG_VIPER	0.1	0.1%	99.0%	PROTOSS_SHIELDBATTERY	0.0	0.0%	100.0%
TERRAN_FACTORY	1.0	0.4%	88.9%	TERRAN_SENSORTOWER	0.1	0.0%	99.1%				
TERRAN_STARPORT	0.8	0.4%	89.2%	ZERG_INFESTORBURROWED	0.1	0.0%	99.1%				
ZERG_SPORECRAWLER	0.8	0.4%	89.6%	TERRAN_LIBERATORAG	0.1	0.0%	99.2%				
PROTOSS_SENTRY	0.8	0.3%	89.9%	ZERG_OVERLORDTRANSPORT	0.1	0.0%	99.2%				
PROTOSS_VOIDRAY	0.8	0.3%	90.2%	ZERG_CREEPTUMORQUEEN	0.1	0.0%	99.3%				
TERRAN_BANSHEE	0.8	0.3%	90.6%	PROTOSS_MOTHERSHIP	0.1	0.0%	99.3%				
ZERG_ULTRALISK	0.7	0.3%	90.9%	PROTOSS_TEMPLARARCHIVE	0.1	0.0%	99.3%				
PROTOSS_ARCHON	0.7	0.3%	91.2%	ZERG_HIVE	0.1	0.0%	99.4%				
TERRAN_BARRACKSTECHLAB	0.7	0.3%	91.5%	ZERG_LURKERMPBURROWED	0.1	0.0%	99.4%				
TERRAN_ENGINEERINGBAY	0.7	0.3%	91.8%	PROTOSS_ROBOTICSBAY	0.1	0.0%	99.4%				
TERRAN_SIEGETANKSIEGED	0.7	0.3%	92.1%	TERRAN_FACTORYFLYING	0.1	0.0%	99.5%				

Table 6. Results for the Unit Identification Benchmarks.

	Cross Entropy						Unit Accuracy (ignoring non-units)						Multi-class Dice					
Model \ Placement	oracle	grid	random	quasi	diag1	diag2	oracle	grid	random	quasi	diag1	diag2	oracle	grid	random	quasi	diag1	diag2
unet_resnet18	0.087	0.255	0.355	0.257	1.155	0.948	56.0%	37.8%	28.0%	35.2%	9.3%	17.7%	0.172	0.119	0.094	0.111	0.050	0.074
unet_resnet34	0.078	0.257	0.370	0.262	1.087	0.852	55.2%	34.1%	27.9%	33.9%	9.3%	17.8%	0.159	0.095	0.097	0.103	0.051	0.075
unet_squeezenet1_0	0.161	0.291	0.380	0.227	1.170	1.308	49.3%	30.0%	22.6%	32.5%	8.8%	16.7%	0.126	0.081	0.062	0.103	0.045	0.068
unet_squeezenet1_1	0.086	0.288	0.315	0.254	1.060	1.106	49.3%	29.1%	24.6%	29.9%	8.3%	15.2%	0.136	0.078	0.078	0.086	0.042	0.058
unet_xresnet18	0.076	0.261	0.316	0.269	1.318	1.030	56.3%	37.3%	25.0%	35.8%	9.5%	18.1%	0.169	0.111	0.073	0.115	0.051	0.080
unet_xresnet34	0.083	0.244	0.331	0.235	1.256	0.984	56.5%	38.7%	27.8%	32.8%	9.6%	18.2%	0.173	0.125	0.085	0.094	0.052	0.079
Average over models	*0.095*	*0.266*	*0.344*	*0.251*	*1.175*	*1.038*	*53.8%*	*34.5%*	*26.0%*	*33.4%*	*9.1%*	*17.3%*	*0.156*	*0.102*	*0.081*	*0.102*	*0.048*	*0.072*

Table 7. Results for the Next Hyperspectral Window Prediction Benchmarks.

	MSE						FPR (nonzero is "positive", zero is "negative")					
Model \ Placement	oracle	grid	random	quasi	diag1	diag2	oracle	grid	random	quasi	diag1	diag2
unet_resnet18	3.85	3.84	3.91	3.88	4.12	4.03	15.0%	17.1%	15.4%	15.9%	9.7%	12.8%
unet_resnet34	3.85	3.86	3.92	3.89	4.12	4.04	16.2%	13.1%	15.0%	15.7%	9.6%	11.9%
unet_resnet50	3.71	3.84	3.91	3.87	4.14	4.06	16.6%	15.6%	15.8%	17.0%	16.6%	16.5%
unet_resnext50_32x4d	3.83	3.87	3.93	3.85	4.12	4.05	16.5%	15.5%	15.9%	17.1%	8.1%	11.4%
Average over models	*3.81*	*3.85*	*3.92*	*3.87*	*4.13*	*4.05*	*16.1%*	*15.3%*	*15.5%*	*16.4%*	*11.0%*	*13.1%*

	TPR(+) (positive diff is "positive")						TPR(-) (negative diff is "positive")					
Model \ Placement	oracle	grid	random	quasi	diag1	diag2	oracle	grid	random	quasi	diag1	diag2
unet_resnet18	60.6%	68.9%	65.8%	62.3%	43.3%	54.0%	55.6%	46.7%	40.8%	47.8%	27.8%	31.5%
unet_resnet34	59.7%	57.4%	63.4%	60.0%	45.8%	52.3%	58.3%	47.5%	42.6%	44.1%	31.4%	33.2%
unet_resnet50	62.3%	67.2%	65.3%	68.0%	45.6%	51.3%	58.0%	46.9%	42.6%	46.4%	29.7%	30.6%
unet_resnext50_32x4d	60.4%	65.5%	69.0%	65.3%	43.5%	52.3%	58.6%	46.0%	39.6%	52.7%	17.3%	23.3%
Average over models	*60.8%*	*64.8%*	*65.9%*	*63.9%*	*44.6%*	*52.5%*	*57.6%*	*46.8%*	*41.4%*	*47.7%*	*26.5%*	*29.7%*

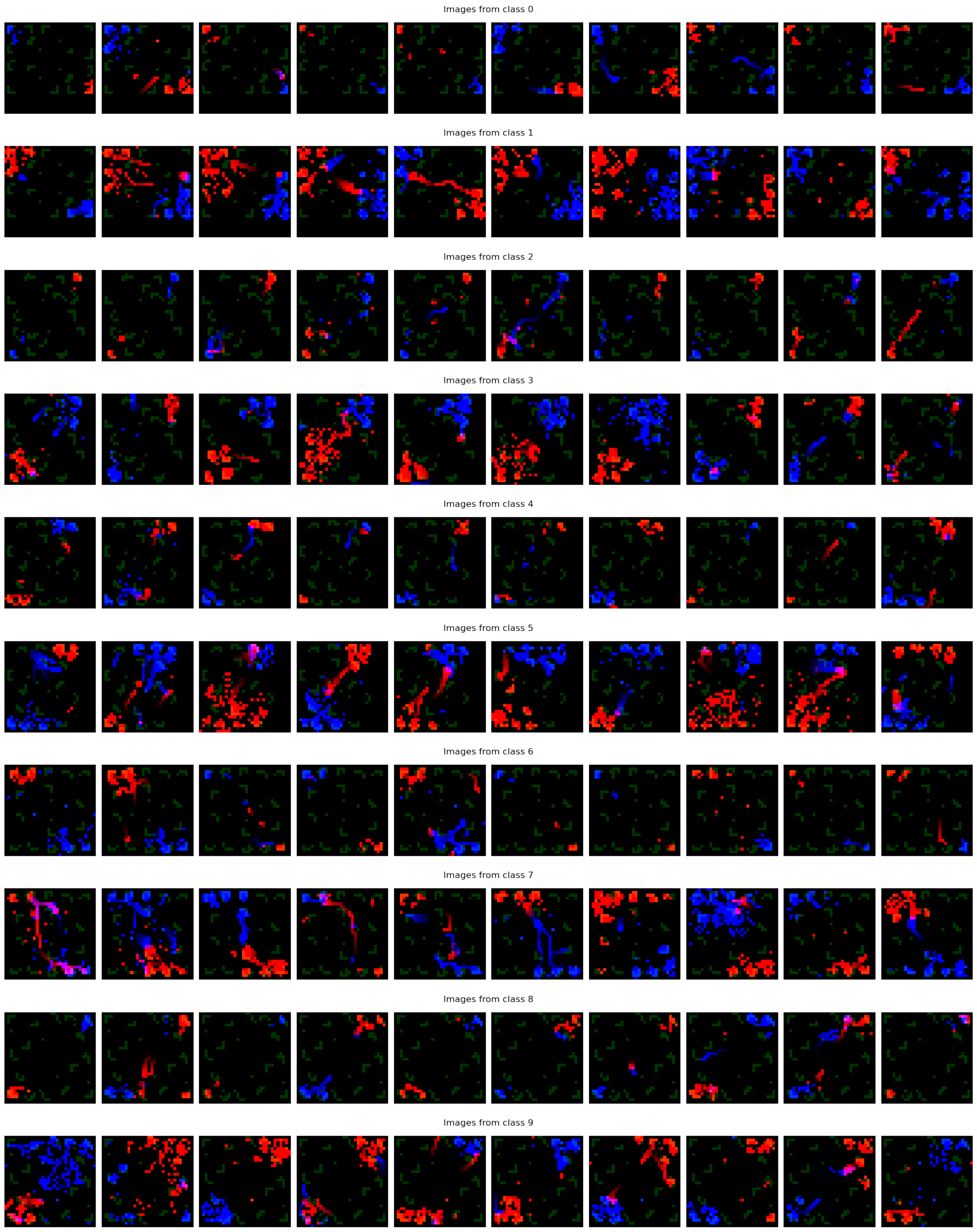


Figure 13. 10 random samples from each class (where each row is its own class), from the map_name + is_begining_or_end 10-way class split. The class label to variable information mapping is as follows: Class_0=('Acolyte LE', 'Beginning'), Class_1=('Acolyte LE', 'End'), Class_2=('Abyssal Reef LE', 'Beginning'), Class_3=('Abyssal Reef LE', 'End'), Class_4=('Ascension to Aiur LE', 'Beginning'), Class_5=('Ascension to Aiur LE', 'End'), Class_6=('Mech Depot LE', 'Beginning'), Class_7=('Mech Depot LE', 'End'), Class_8=('Odyssey LE', 'Beginning'), Class_9=('Odyssey LE', 'End')).

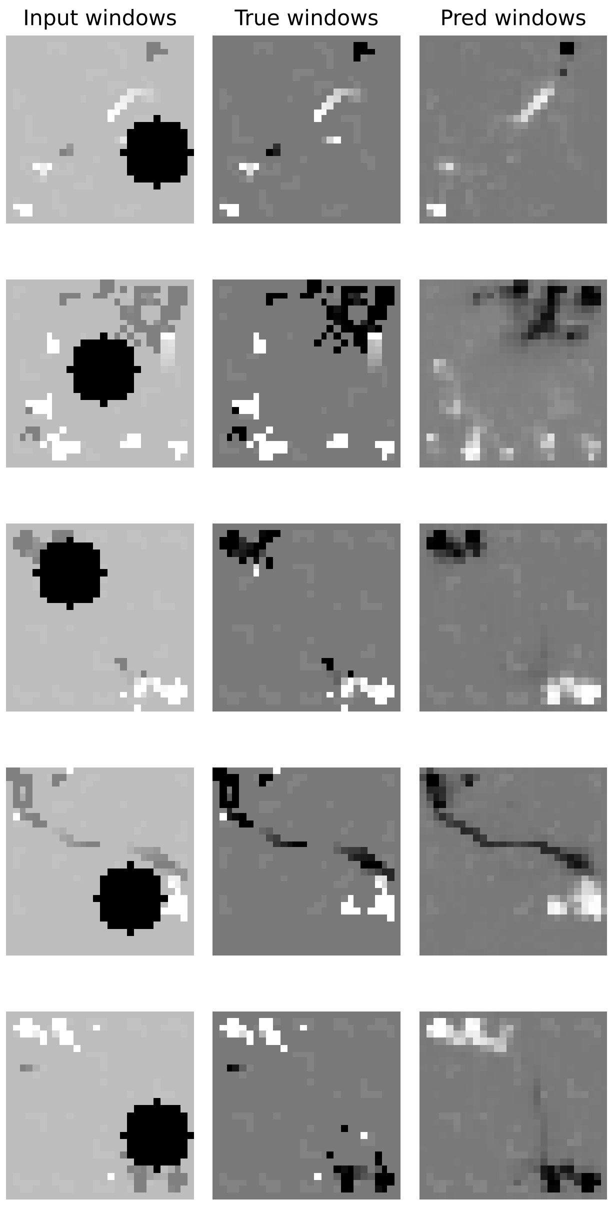


Figure 14. Five examples from the VAEImputer model trained to denoise an imputed corrupted aerial image (where in this case an occlusion with a 5px radius has been simulated). As can be seen. Note, the difference in colorization between the input windows and true windows is simply due to plotting renormalization due to the occlusion.

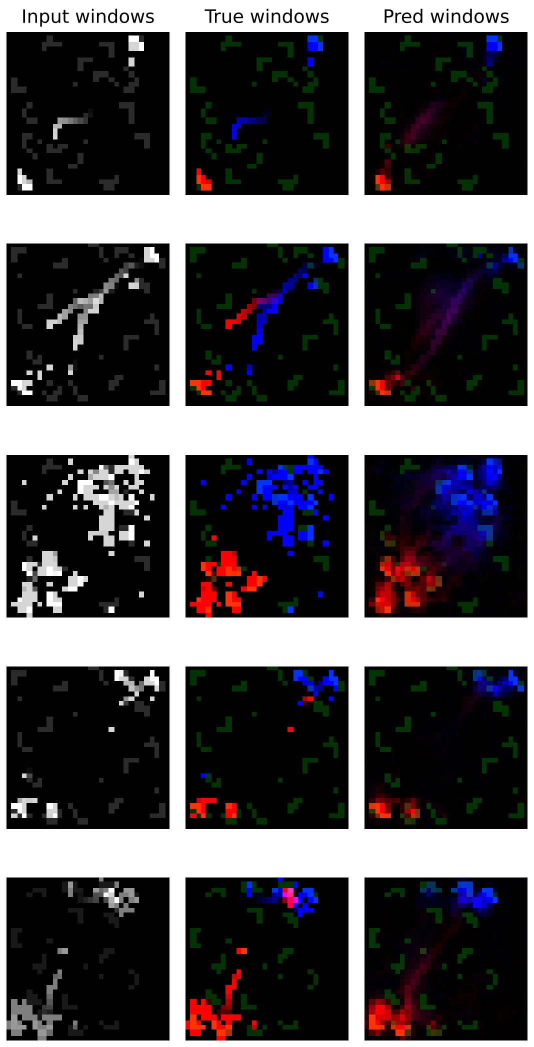


Figure 15. Five examples from the ResNet-101 model [15] trained to identify the owner of each unit in a window, where this task is akin to an image colorization task where each owner (Player1, Neutral, Player2) is placed on a separate channel. As can be seen in the examples here, it is difficult at times for the model to determine the difference between Player 1 and Player 2 due to both players having random starting corners of the map.