Human-in-the-Loop Extension to Stream Classification for Labeling of Low Altitude Drone Imagery

Jeffrey Schulz[a], Andrew Buck[a], Derek T. Anderson[a], James M. Keller[a], Grant Scott[a], and Robert H. Luke III[b]

[a]Department of Electrical Engineering and Computer Science, University of Missouri, Columbia MO, USA
[b]U.S. Army DEVCOM C5ISR Center, Fort Belvoir, VA, USA

ABSTRACT

In general, there is a severe demand for, and shortage of, large accurately labeled datasets to train supervised machine learning (ML) algorithms for domains like smart cars and unmanned aerial systems (UAS). This impacts a number of real-world problems from standing up ML on niche domains to ML performance in/across different environments. Herein, we consider the task of efficiently, meaning requiring the least amount of human intervention possible, converting large UAS data collections over a shared geospatial area into accurately labeled training data. Herein, we take a human-in-the-loop (HITL) approach that is based on coupling active learning and self-supervised learning to efficiently label low altitude UAS imagery for the goal of training ML algorithms for underlying tasks like detection, localization, and tracking. Specifically, we propose an extension to our stream classification algorithm StreamSoNG based on human intervention. We also extend StreamSoNG to rely on a second and initially more mature, but assumed incomplete, ML classifier. Herein, we use the Unreal Engine to simulate realistic ray-traced low altitude UAS data and facilitate algorithmic performance analysis in a controlled fashion. While our results are preliminary, they suggest that this approach is a good trade off between not overloading a human operator and circumventing fundamental stream classification algorithm limitations.

Keywords: active learning, self-supervised learning, human-in-the-loop, HITL, drone, unmanned aerial vehicle, unmanned aerial system, object detection, Unreal Engine, stream classification, StreamSoNG

1. INTRODUCTION

In today's machine learning (ML) and artificial intelligence (AI) landscape, deep learning (DL) is the reigning king. Different flavors of DL, like convolutional neural networks (CNNs) and recurrent NNs (RNNs), generally require large amounts of labeled training data. In the case of a CNN and object detection, many look to data sets like ImageNet and Coco; which have 14+ million and 300K images respectively. Realistically, labeling data sets at this scale is a daunting task. In many applications, it is the bottleneck. Modern ML is overly dependent on supervised learning and its not clear if this ideology scales in our pursuit of next generation AI. Herein, we explore the idea of having a human help a machine learn and refine concepts. Specifically, we focus on an extension of self-supervised stream classification using a human-in-the-loop (HITL). The result has the potential to significantly reduce the time and cost required to label large low altitude aerial data sets and build ML/AI models on specialized domains that have insufficient labeled training data. Figure 1 illustrates our motivation.

Current DL models cannot predict classes that they are not trained on. For example, if a CNN is trained to find people and cars, then it will not find aliens or hamburgers. We refer to this hereafter as a "closed world" model and our desire to detect new classes as the "$N + 1$" problem (where N is the current number of known classes). In the research community, these concepts are often called *open set recognition* (see Ref. 1 for a recent survey). A number of questions arise as we attempt to add a new class. For example, how does the model see the new class? Does it incorrectly classify it as one of its known N classes? Can the model say "I don't know?"

Send correspondence to Jeffrey Schulz
Jeffrey Schulz: E-mail: jtsmbb@mail.missouri.edu

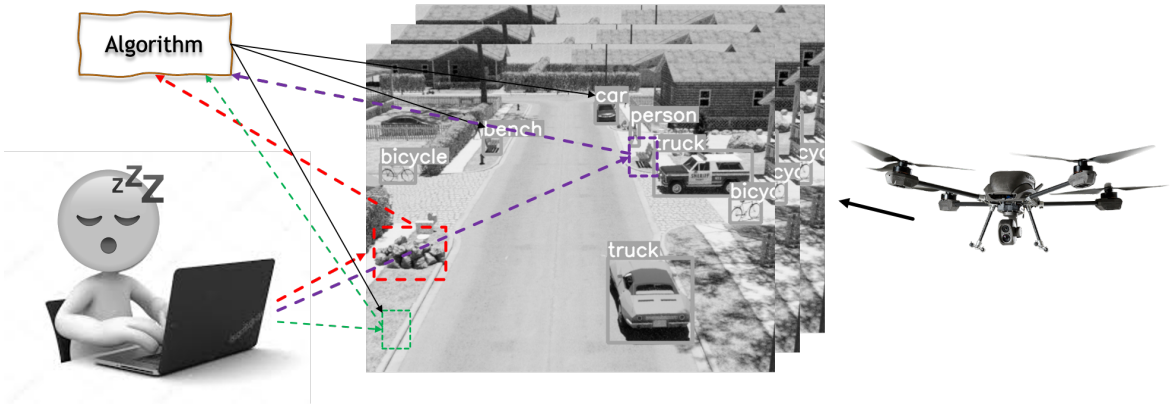

Figure 1: Illustration of our application of interest. Low altitude UAV imagery on niche domains is collected and subsequently labeled automatically by a stream classification algorithm. However, algorithms are not perfect; they need help getting started and achieving a desired steady state. A human is in the loop and helps teach the algorithm. However, the image stream is large and the human does not want to analyze every candidate image. The goal is to strike a balance of active and self-supervised learning to ultimately reduce the time and cost associated with labeling large UAV collected data sets. The image shows correct AI/ML detection's for known classes (gray boxes), algorithm detection's for new patterns that the machine does not know but a user might want labeled (red), algorithm mistakes that need correcting (green), and detection's that a human catches but the algorithm misses (purple). Our aim is to create a collaborative human-machine coupling that results in a small amount of effort to kick start high quality subsequent data labeling.

Furthermore, does the current model even have the potential to detect the new class, e.g., are its features good enough to detect and discriminate this class? Herein, we explore the case of a human working with a CNN that has a fixed vocabulary (e.g., pre-trained convolutional weights) in an online fashion. Instead of using the pre-trained CNN decision making layers, we use stream classification. The machine can now label things it knows (one of its N classes), it can be extended to new classes (aka N+1), it can be used to identify outliers, and it can be used to combat "concept drift". Of course, all of the above is contingent on the pre-trained CNN feature weights being able to extend to class N+1. In a final step, the newly labeled data can be used offline, if desired, to learn a new set of weights, achieving a form of self-supervised learning.

For this paper, we focus on a subset of the desired functionality outlined above. Herein, we make the following specific contributions. This is the first application of StreamSoNG[2] on a real streaming computer vision task. To date, StreamSoNG has been developed using a combination of theory, controlled synthetic data sets (e.g., mixtures of Gaussians with noise), and real-world texture image data sets. Second, we extend StreamSoNG to take into account a HITL. A fundamental limit of algorithms like StreamSoNG are that they must make very complicated decisions using little information. For example, when has a new pattern emerged? Herein, we take a first step to couple StreamSoNG with a HITL to improve the N+1 problem. Third, we explore a use case of automatically labeling low altitude drone imagery. In order to maintain control, we generate aerial imagery from ray tracing in the Unreal Engine. As the reader will see, the imagery is extremely similar to real aerial data, providing a wonderful testbed and potential source of training data. Simulation at this level allows us to more rapidly explore the user interface, find break cases, and develop new algorithms.

In section 2.1, we discuss our implementation of StreamSoNG. In section 2.2, we introduce our user interface with which the HITL supervision is performed. In section 3, we discuss preliminary experiments and results. Last, in section 4 we discuss future work. Table 1 shows acronyms and our notation.

Table 1: Acronyms and Notation

HITL	Human-In-The-Loop
StreamSoNG	Streaming Soft Neural Gas
PKNN	Possibilistic KNN
PCM	Possibilistic C-Means
GNG	Growing Neural Gas
$\mathbf{x}_t$	Sample at time t
$\mathbf{p}_{ik}$	kth closest prototype to ith class label
t'_{ik}	Typicality of sample $\mathbf{x}$ to the kth closest prototype of the ith class

2. METHODS

2.1 Stream Classification

As mentioned above, our HITL labeling problem can be cast as an instance of stream classification. In Ref. 2, Wu et al. proposed the Streaming Soft Neural Gas (StreamSoNG) algorithm. StreamSoNG makes the assumption that data cannot be stored, e.g., a reality for many Big Data applications. For example, consider the case of imagery streaming at multiple frames per second for hundreds of cameras monitoring traffic in a city 24-7. Problems such as these break the majority of existing supervised and unsupervised learning algorithms. StreamSoNG addresses problems like these by combining unsupervised learning and classification into a streaming algorithm that specializes in extending the knowledge of a system with new data. In short, StreamSoNG first utilizes growing neural gas (GNG) to initialize prototypes for a training set of data and saves out the prototypes. Then during data streaming, typicality is computed on new data with possibilistic k-nearest neighbor (PKNN), separating streaming data as either belonging to an existing class (one of N classes) or as an outlier (with the potential to be class N+1). The list of outliers are then run through the possibilistic c-means (PCM) algorithm with an attempt to find new patterns. If patterns are found in the outlier list, they are extracted, added to the knowledge base of the algorithm as class N+1 (a generic label), and initialized with another GNG to establish cluster prototypes. The algorithm now has a knowledge base for the N+1 class, and new streaming data can now get this classification. The algorithmic flow for how we drop our implementation of StreamSoNG into the system is shown in Figure 2.

To go into more detail, the StreamSoNG algorithm can either assign an incoming streaming data vector as part of an existing pattern or as an outlier. StreamSoNG utilizes the PKNN algorithm to calculate typicality and determine classification for incoming data points. During the PKNN step, computing typicality has historically

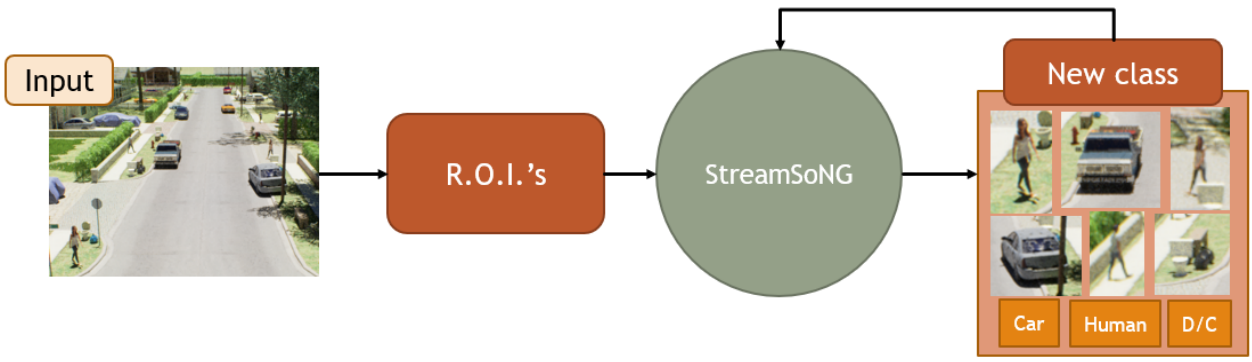

Figure 2: After training data is provided to initialize the StreamSoNG algorithm, the streaming test data is streamed into StreamSoNG where clusters and outliers are determined and new classes can emerge. When a new class is detemined to exist, the user is prompted with a "New Class" GUI where he/she can select the objects belonging to the new class. The StreamSoNG algorithm is then updated to reflect those truths.

been contentious, so in this paper we define our typicality equation by,

$$t'_{ik}(\mathbf{x}_t, \mathbf{p}_{ik}) = \frac{1}{1 + [\max(0, \|\mathbf{x}_t - \mathbf{p}_{ik}\| - \eta)]^{2/(m-1)}}, \tag{1}$$

where if the distance of the sample $\mathbf{x}_t$ and the nearest k prototypes is within η distance, the typicality is greater than zero. Any sample outside of η from any prototype is given a typicality of zero. The typicality is leveraged w.r.t. to the distance with a factor m. In our implementation, estimate a unique η per pattern and choose a m of 2.0.

If the sample is determined to part of an existing pattern (typicality of greater than 0 to any prototype), then the system updates prototypes belonging to that class as follows,

$$\mathbf{p}_{ik}^{t+1} = \mathbf{p}_{ik}^{t} + \alpha * t'_{ik}(\mathbf{x}_t) * e^{-k/\lambda}(\mathbf{x}_t - \mathbf{p}_{ik}^{t}), \tag{2}$$

where $\mathbf{p}_{ik}^{t}$ is the kth closest prototype to sample $\mathbf{x_t}$ at time t for the ith pattern label, α is the learning rate (0.2 in this paper), $t'_{ik}(\mathbf{x_t})$ is the typicality of the sample $\mathbf{x_t}$ to the kth closest prototypes for the ith pattern label, and λ is the "neighborhood range parameter" which determines the drop off for prototype update. Note, we exclude the S-function presented in the original paper.

If the sample is determined to be an outlier (typicality of 0 to every prototype), then after a certain minimum amount of outliers, StreamSoNG tries to automatically identify clusters in the outlier list with PCM clustering. If any clusters are found, they are added to the patterns in StreamSoNG as a new and separate class. This algorithm flow is discussed in Figure 3.

In this paper, PCM is initialized to find one cluster and is simplified for StreamSoNG for sequential computation. As per Wu, et al,[2] the algorithm is modified to become Sequential Possibilistic One-Means (SP1M). For more information on the specific implementations of GNG, PKNN, and SP1M, see Ref. 2.

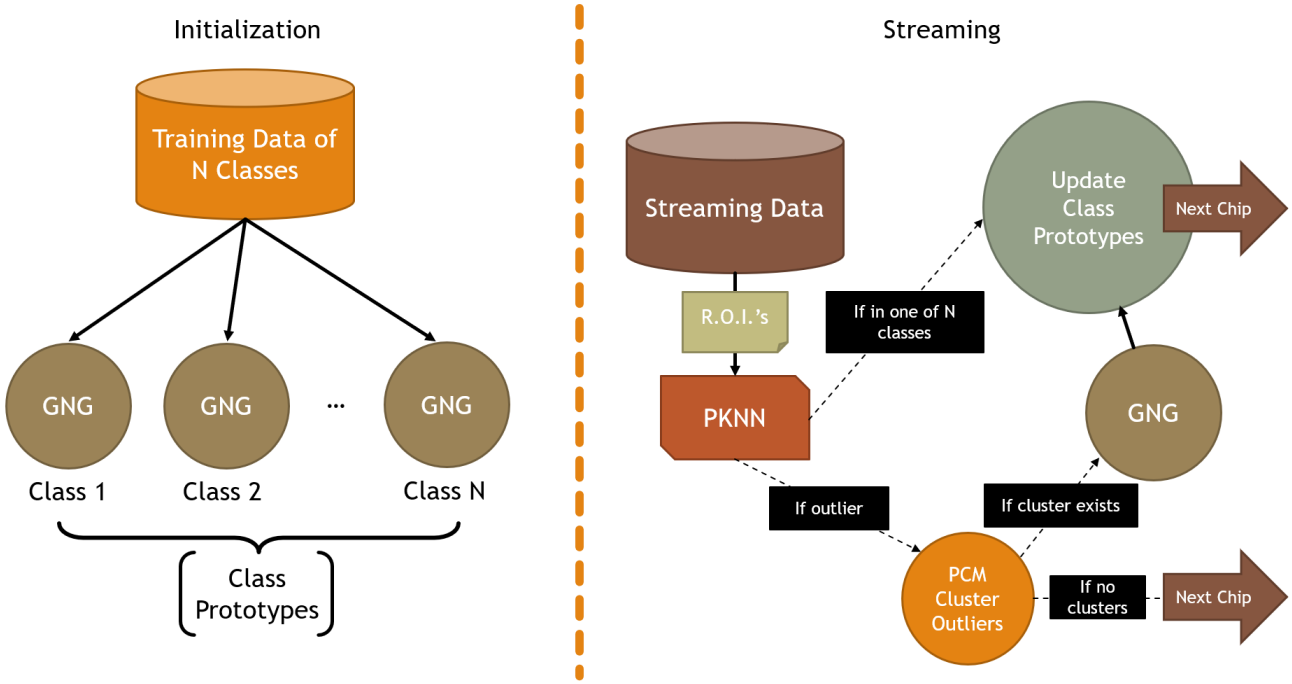

Figure 3: StreamSoNG flowchart. During initialization, Growing Neural Gas (GNG) networks are trained on each known data cluster and prototypes are saved. During streaming, data is introduced to the system one chip at a time. Herein, chips are image space regions of interest (R.O.I.) identified by a change detection algorithm. Classification is determined by the PKNN algorithm and outliers are clustered via PCM. If a new pattern appears, the class is initialized with GNG to find prototypes and the pattern is added to the known patterns in StreamSoNG.

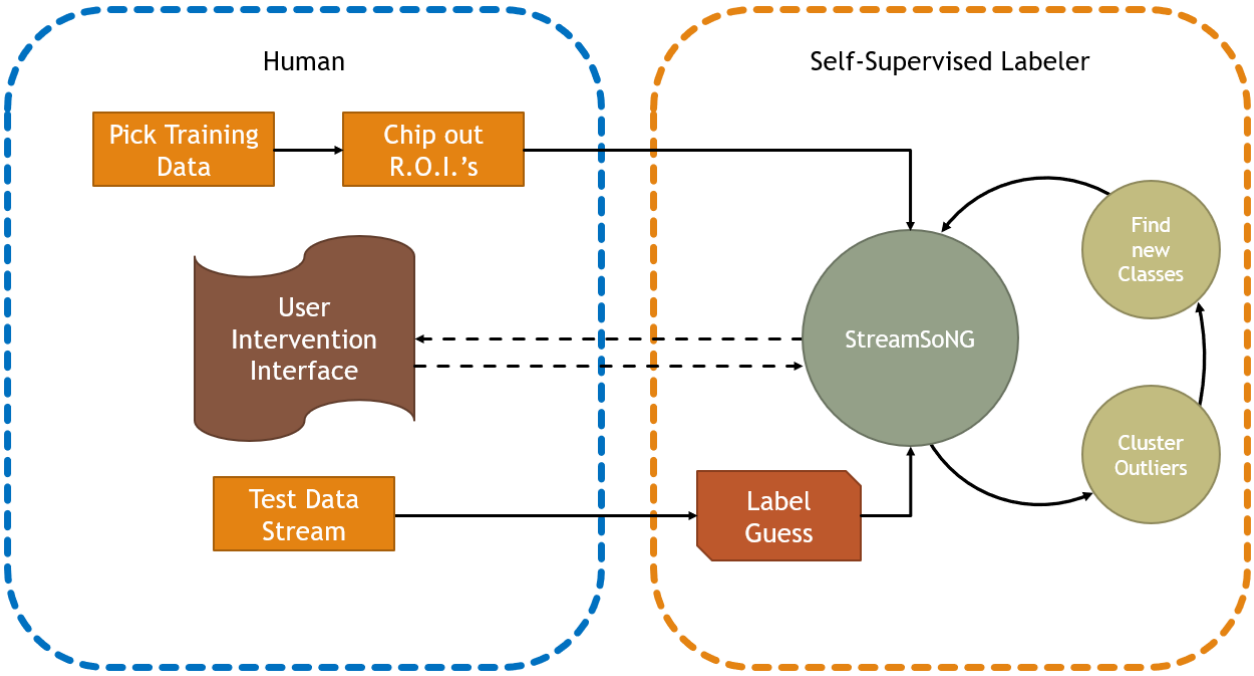

Figure 4: This diagram shows the relationship between the human operator and the self-supervised labeling algorithm as proposed in this paper. After providing the necessary initialization data, the human operator takes a back seat and monitors the self-supervised labeler. As the labeler streams data, we propose that it is possible to interrupt the data stream to correct wrong labels and preempt new class creation.

2.2 User Interface

The purpose of this section is to discuss our considerations for a user interface (UI) prototype. First, we discuss how the user should be able to interact with our stream classification algorithms. We intend to have clear divisions of responsibility between the user and algorithmic labeler to improve the ease of use for this potential implementation. This relationship is shown in Figure 4. After the human operator provides the necessary initialization data and a source for the streaming data, they can sit back and only intervene at certain, clearly defined moments. However, our current article is about extending stream classification via a HITL interaction. Even though we do not focus on optimal user interface design nor human factors, achieving our algorithmic goal requires us to at least explore a UI prototype. The UI outlined in this section is focused on the idea of extending StreamSoNG. Figure 5 shows the proposed UI.

Figure 5 consists of the following parts. As data is streaming in, the user can specify a "play rate". The UI also has "arrows" for go backward and forward in time. This supports the need to take small steps or to go back and correct something that the user saw as wrong and it rotated off the UI. Here, we focus on the aspects related to giving the human operator the ability to understand and make decision on what is happening on screen as quickly as possible. Things we keep in mind include: not forcing the operator to make decisions with time constraints, keeping information in consistent locations on screen, and having clear and uncluttered controls. Again, while this paper is not about UI design, we try to consider at least a few good design principles in our prototype.

Factors like the above led us away from having a real-time cluttered video feed that requires a high cognitive investment, e.g., Figure 1. Instead, we run an existing detection and localization algorithm to find candidate regions of interest (R.O.I.). Specifically, we run YOLOv5.[3] Non-open set algorithms like YOLOv5 have the advantage that their community default models (set of weights) have been trained on many image data sets. The problem is, these algorithms can be wrong or incomplete when adapted to new domains and applications. We use an algorithm like YOLOv5 to bootstrap our stream classification. In the long run, we expect the stream

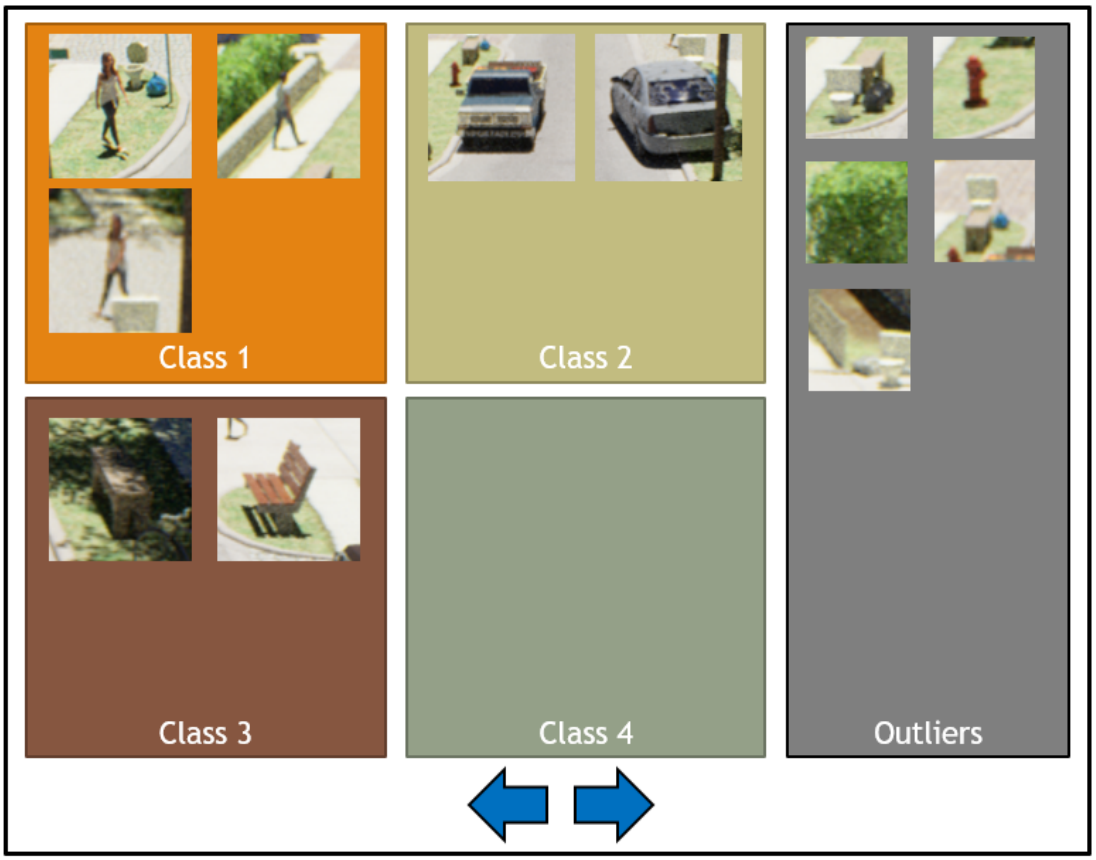

Figure 5: Prototype of the user interface explored herein. See the text for a full description.

classifier to perform the majority of work in our approach. However, using an algorithm like YOLOv5 leaves us vulnerable. It only knows what it knows. Meaning, people in a new data can look different and we often desire to find new classes. Therefore, we also rely on change detection for R.O.I. identification. The reader can refer to the literature for decades of image (2D) and world (3) space methods from mixtures of Gaussians[4] to modern change detection in deep learning.[5,6] In this initial paper, we simulate change detection from one flight or day to the next. We do not consider frame-2-frame change detection, but it could be added. Last, it should be noted that while we focus on the above two methodologies for finding R.O.I.s for HITL assisted StreamSoNG, other strategies exist. For example, the modern computer vision literature has a heavy investment in visual attention modeling[7–9] and lower level algorithms like optical flow (e.g., FlowNet[10]) can be used to find temporal movement in a video sequence. In summary, our UI is driven by a stream of video R.O.I.s.

Keeping the above in mind, Figure 5 is based on a few simple design ideas. Each class gets its own real estate in the UI. For example, "Class 1" is people, "Class 2" is cars, and "Class 3" is benches. The goal of the UI is not to show the internal representation of StreamSoNG, e.g., graphical depictions of the underlying neurons. Instead, the goal is to show a rotating set of examples that the algorithm feels belong to that class. At any moment in time a human watching the rotating UI can pause the interface if a chip (R.O.I.) is wrong. Our idea was its perhaps simpler for a human to watch these categories and stop–to provide feedback–when they notice mistakes. Thus, the human is sitting over the algorithm letting it do its thing until errors are encountered. Furthermore, there are R.O.I.s that do not belong to a class that a user cares about. We have a section of the UI dedicated to this. These are examples on the "watch list" and StreamSoNG. The next sub-sections of this article go into depth on the a human monitoring the watch list and reacting to StreamSoNG inquiries. Overall, the UI is simple and its about supporting interactivity with the user.

The point is, Figure 5 is a proof-of-concept. Future work will focus more on the human angle, e.g., ergonomics.

The current UI is a real-time class clustered rotating stream of R.O.I.s and outliers in support of HITL enhanced StreamSoNG. The next few sections go into greater depth on the UI relative to three use cases.

3. USE CASES

Figure 6: Simulated Modular Neighborhood Pack[11] environment on the Unreal Marketplace[12] for the Unreal Engine.[13] Example (left) view in the editor and (right) imagery we generated for this paper using a Camera (Cinematic Actor), pre-scripted flight sequence via a Cinematic Track, and ray tracing offline using the Movie Render Queue. See Section 3.1 for additional details.

3.1 Simulated Data and Experimental Design

Herein, our primary focus is the extension and exploration of stream classification (StreamSoNG) with respect to a HITL and an additional classifier. In order to experiment with a wide and controlled range of conditions to speed up algorithmic prototyping, the Unreal Engine[13] is used as a surrogate to a real low altitude drone equipped with a visual spectrum (aka RGB) camera. We use a combination of free and for purchase content (3D models, animations, and textures) from the Unreal Marketplace.[12] Figure 6 is example imagery that we generated in Unreal. It is worth noting that Unreal's ray tracing, which can be approximated in real time using NVIDIA hardware like the GeForce RTX 3090, provides access to high fidelity rendering capable of mimicking real cameras. For example, a user has control over features like motion blur, fstop, focal distance, FOV, pixel resolution, noise, and much more. This provides flexibility in mimicking different systems, making simulated data more like real-world data. However, if a user desires real time simulation, then the AirSim[14] Unreal Editor plug in can be used, and Ref. 15 outlines a plug in for extended cameras models and effects.

Specifically, we use a Camera (Cinematic Actor), a pre-scripted flight sequence is configured (versus a real-time autonomous flight in AirSim) as a Cinematic Track and last, imagery is generated offline using the Movie Render Queue. An advantage of this offline route in the short research term is it gives us greater control, e.g., use of Deferred Rendering (via Path Tracer), multiple spatial and temporal samples for anti-aliasing, specification of maximum number of ray bounces, etc. This offline procedure has allowed us to achieve a desired level of image quality for our experiments. Furthermore, it is our belief that our algorithms and codes can be migrated without major effort to real data from a drone next. A further advantage of simulation is we know the truth. The reader can refer to our articles on visual guided autonomy,[16] meta data enabled contextual fusion,[17] or simulated augmentation data for explosive hazard detection[18] for details about how to use stencil buffers to automatically generate labeled bounding boxes or per-pixel semantic labels.

Herein, we simulate a neighborhood in Unreal via the Modular Neighborhood Pack[11] that includes people, cars, bicycles, and miscellaneous objects (outlined below). As already discussed, we assume that StreamSoNG is operating on alarms generated by a region of interest (R.O.I.) algorithm, e.g., change detection between consecutive flights over the same region, frame-to-frame change detection, etc. As the focus of our current article is StreamSoNG and not a change detection algorithm, a human curated the R.O.I.'s using the visual object tagging tool (VoTT).[19] A set of simulated streets containing people and cars were held back for StreamSoNG

initialization. Additional streets were held back for testing or stream evaluation. These streets have the following. First, we consider duplicates of known people and cars in different contexts (locations and poses). Second, we add objects that belong to these classes that the algorithm has not seen before, e.g., new people and cars. Third, we add R.O.I. that an algorithm has not seen before but might be interested in, e.g., trash bags, bicycles, toilets, etc. Last, we add R.O.I.'s that change detection algorithms frequently find that a user likely does not care about, e.g., algorithm mistakes, nature (bushes, trees), etc. The point is, our curated data set has a mixture of challenges. The human chipped varying size bounding boxes around each R.O.I., as this is likely the reality for an imperfect change detection algorithm.

Now that our scene is set up and data is generated, we use the following experimental design. First, we use an existing deep learning model for feature extraction on R.O.I.'s. Second, we use a method to reduce the dimensionality of these neural features. R.O.I. features are generated using a modified ResNet50 (no classification layer) pretrained on the ImageNet dataset. The result is initially of size 2048x(7x7) per sample; 2048 features with response fields of size 7x7. As we are primarily interested in the degree to which a feature is present or not, versus where spatially these features exist, max pooling was used per feature map, which results in 2048 features. These features are then reduced to size 128 using an Autoencoder trained on the ImageNet dataset. Experimentally, we tried different sizes and we determined that 128 was a good balance for our data set. We did this visually by generating a sorted dissimilarity matrix and we looked at similarity between objects within, and across classes. For an ideal situation, dark diagonal blocks should appear for each sorted class, indicating low dissimilarity (aka high similarity) for objects of the same class. Conversely, off diagonal blocks should be high in value, indicating low similarity (high dissimilarity) between objects in the different classes. In the end, StreamSoNG operates on our features in this reduced 128 dimensional space. In future work, we will study the effect of running StreamSoNG in different dimensions and we will explore different transformations than what is outlined herein. The above was used because it is somewhat common operating practice nowadays; i.e., ML on neural features with reduced dimensionality.

Last, before we can start streaming data into our interface, we need to initialize StreamSoNG. This initialization includes a training set of features for all "known" classes and their labels. In our case, our held back training data streets had 30 samples for each known class (cars and people). The interface is now ready for streaming data. In the following sections, we demonstrate three use cases.

3.2 Use Case 1: StreamSoNG Recommended Emergent Patterns

Use case one is driven by the following need. A user would expect a stream classification algorithm to prompt them when something new has been detected, i.e., a new class (or sub-class) has been found. We expect that this is one of the most important problems to handle when developing a truly self-supervised algorithm. Figure 7 shows our HITL desktop interface. When StreamSoNG identifies new patterns in the streaming data, we want the interface to interrupt the stream and alert the user for input. This is demonstrated in Figure 8. The user is now responsible for selecting images that are similar to each other–or they can simply accept all recommended by StreamSoNG–and they must provide a class label. In the case of Figure 8, the user has determined that these are examples of bushes and they provide a new label ("bush"). However, inner class variation can exist and it is possible that a newly detected pattern belongs to an existing class, e.g., a new type of car, for which the user can provide an existing class label. After the user submits their changes, streaming continues but StreamSoNG now has knowledge of this new class and it can leverage this to increase its classification accuracy.

While effective in many scenarios, this use case can falter in real world deployment. In order for StreamSoNG to find and recommend a new pattern, StreamSoNG had to determine that a new pattern (cluster) has emerged. This requires a few factors. Namely, a sufficient number of samples with satisfactory similarity. This is where stream classification algorithms are at a disadvantage. It is not trivial to answer these question. It is one thing to address questions like these in controlled settings like compact well separated Gaussian clouds, but how does it perform on high dimensional neural features where classes are likely multi modal? The next use case was designed with this shortcoming in mind.

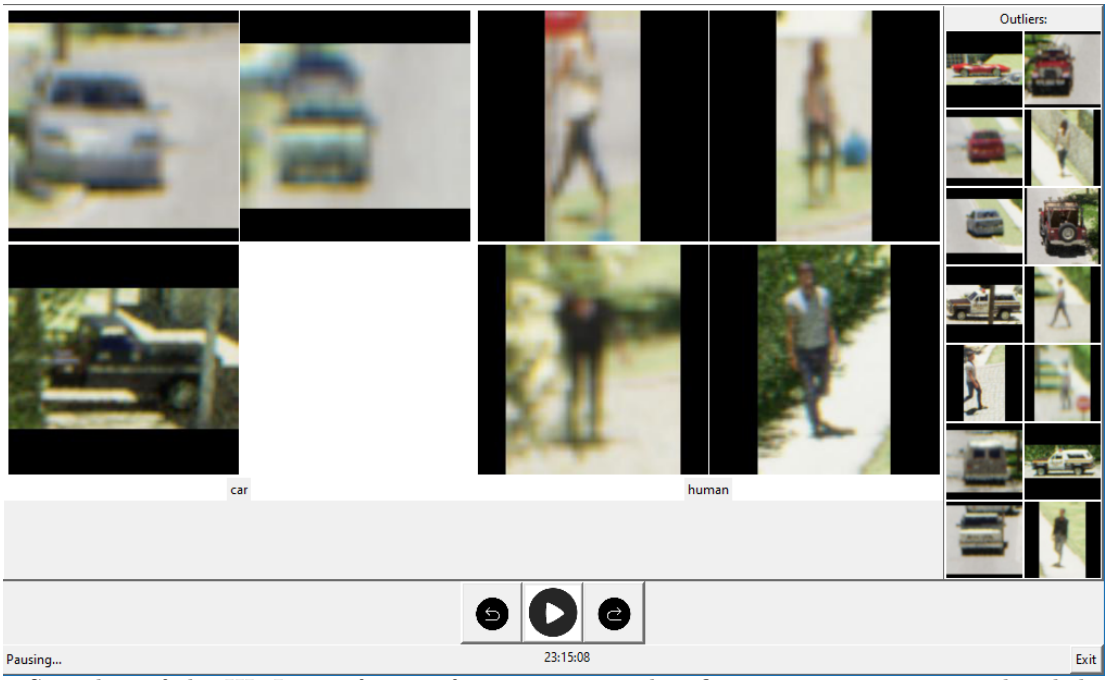

Figure 7: Snapshot of the HITL interface performing stream classification on incoming simulated data. Cars and humans that StreamSoNG has confidently classified populate in their respective regions. All chips that StreamSoNG determines are outliers populate in the right column.

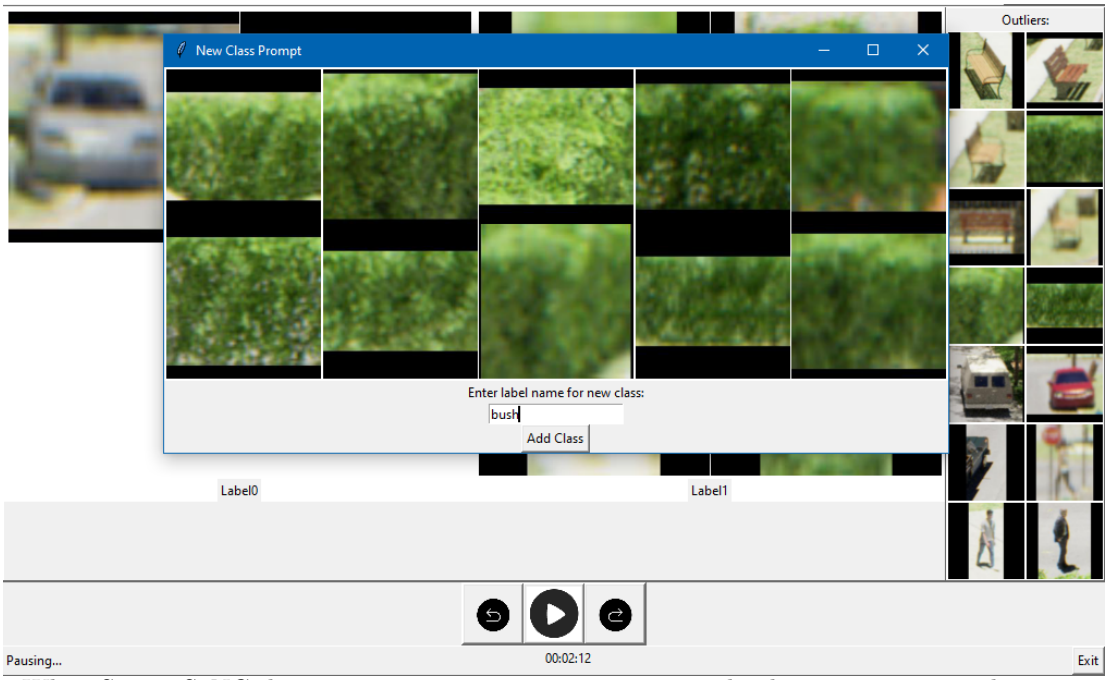

Figure 8: When StreamSoNG determines a new pattern is present in the data, it interrupts the streaming data and presents this dialog box for the user. The user selects similar images, or simply accepts all, and provides a label for the emergent pattern. Once streaming resumes the system now classifies images using this new knowledge.

3.3 Use Case 2: Preemptive Identification of a Pattern

Another use case we expect our system to handle is the preemptive identification of a pattern by the human operator. The idea is as follows. Say the user is monitoring the streaming data and they see an important pattern of data start to populate in the outliers column. The user should be able to preempt the identification of this class as they please. Figure 9 highlights this interactive process.

To preempt the identification a new pattern, the user can click any image chip on screen to bring up a dialog box showing the clicked image and a rank ordered set of similar images to help aid in the process of quickly identifying imagery for a new pattern. The user can then select the images they think belong in the new pattern and provide a label. This process can be repeated as much as needed. Similar to the behavior in Use Case 1, the new pattern is now added to the knowledge of StreamSoNG and then system will now attempt to classify images accordingly. Note, our underlying implementation includes adding a neural gas neuron for each user identified selection. While this is likely more neural gas neurons than what StreamSoNG would find for an emergent pattern, we chose to keep the users resolution of sampling. Alternatively, the reader could select to run growing neural gas on the user identified samples.

This use case aims to remedy shortcomings in Use Case 1. Namely, StreamSoNG might be too slow to react. If objects are rare, it might take a lot of time to see enough examples before StreamSoNG is willing to declare a new emergent pattern. This specifically addresses the number of samples challenge in StreamSoNG. However, it is also not trivial to determine similarity in high dimensional spaces driven learned neural features. If StreamSoNG is unable to detect a cluster, but the user has already made this connection, then it make sense to have StreamSoNG include this preemptive strike, as its a feature of having a HITL.

While the above helps, it still often results in outlier lists containing many instances of known classes. For

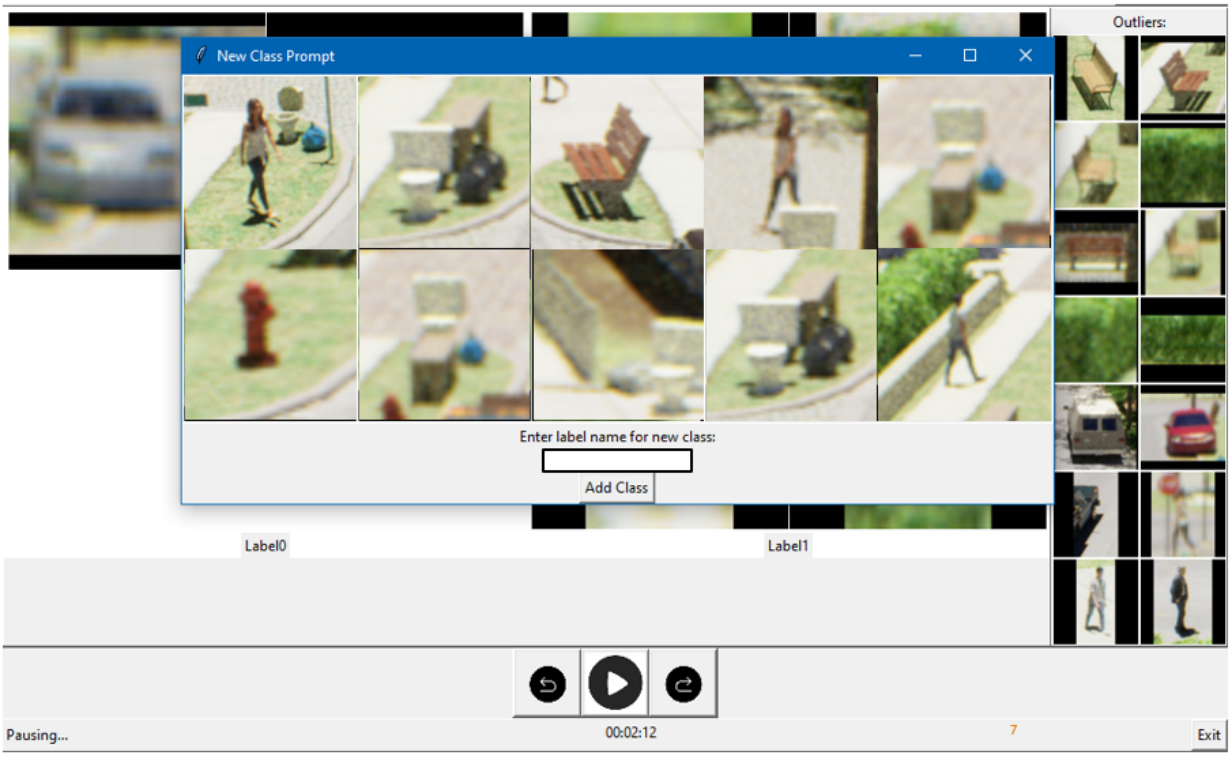

Figure 9: If a user chooses to preempt the creation of a new class, or extend an existing one, this dialog box pops up showing the target image similar imagery. The user then selects chips and they provide a label. In this figure, the user noticed a class of toilets being thrown away. They clicked on a toilet from the outlier list and a rank ordered list of similar chips (according to the underlying neural feature space) are presented. The user picks which chips are toilets, they provide the label, and streaming resumes.

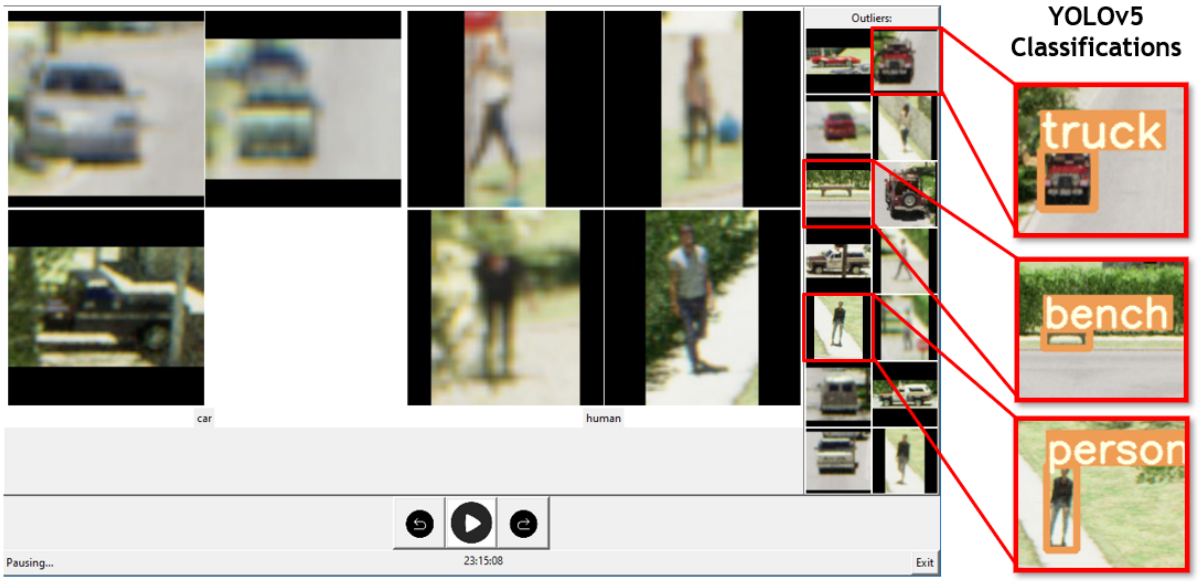

Figure 10: Example of image chips that StreamSoNG determined were outliers but YOLOv5 correctly classified.

example, consider Figure 9. The outlier list has many cars and people still, regardless of the fact that is has classes supposedly covering those classes. A challenge that StreamSoNG has to face is, its a streaming classification algorithm and it takes time for it to "come up to speed." Meaning, in the early stages the user will likely have to help the algorithm more than desired. In the next setion we address this challenge.

3.4 Use Case 3: Using Another Classifier to BootStrap StreamSoNG

The last use case we discuss is the ability to extend an existing algorithm. As briefly discussed earlier, frameworks like YOLOv5 have an immense knowledge base already for object detection. In this section, we explore the use of an algorithm like YOLOv5 to boostrap StreamSoNG. The idea is, an algorithm like YOLOv5 may fail to work on new domains and its a closed set/world algorithm. However, while StreamSoNG is coming up to speed, an algorithm like YOLOv5 could be used to reduce our set of R.O.I.'s in an outlier list. We would like for our StreamSoNG extension to be able to draw from this capability. When images are streamed, they are first run through YOLOv5 for localization and detection. If a confident classification is achieved and it has a generalized intersection over union (GIOU) with one of our alarms, then the chip is auto-labeled and fed to our algorithm, i.e., its respective neural gas class is updated. An additional benefit of this approach is the user can select if they want all YOLOv5 detection's to be added to StreamSoNG. That is, chips that are not associated with alarms can be found and used for learning. While this sounds redundant, i.e., YOLOv5 already knows about these objects, it allows StreamSoNG to learn from YOLOv5, helping it learn faster. Figure 10 shows an example.

This process of bootstrapping StreamSoNG using another algorithm is not without flaw. Herein, it helped us reduce our outlier, lessening the users amount of desired interactivity. However, in future work we will need to address how to modify StreamSoNG to accommodate YOLOv5 mistakes. That is, YOLOv5 is trusted and if it provides labels that are wrong, then StreamSoNG learns from these examples.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we focused on a HITL extension to a stream classification algorithm, StreamSoNG. In addition we also explored bootstrapping StreamSoNG using a second classifier. Three common use cases were explored relative to controlled scenes generated by simulation. Use case one showed that StreamSoNG can recommend new patterns to a user, use case two allows the user to preemptively declare patterns, and use case three outlined how to reduce the outlier set using an algorithm like YOLOv5. Overall, while qualitative and preliminary, our

use cases show promise for HITL assisted labeling of low altitude aerial data sets. However, more work is required in order to achieve high quality labeling on real world streaming data.

In future work, we will address the following. First, we will continue to rely on simulation in the short term. However, we will integrate our change detection algorithms to remove the human identified R.O.I.'s. Now that a pipeline for generation, labeling, and next segmentation and alarm generation exists, we will identify performance metrics to facilitate quantitative scoring. The user interface will also be improved around human factors. At an algorithmic level, we will explore how to fuse the secondary algorithm (e.g., YOLOv5) with StreamSoNG classification results. We will also explore how a user can provide feedback to scrub or update mistakes made by StreamSoNG. In addition, StreamSoNG currently uses the PKNN and possibilsitic clustering. We would like to explore other ways of updating growing neural gas relative to the desire to generate a membership per sample and to automatically discover new emergent patterns. Finally there are a number of user defined parameters in this system that need sensitivity analysis and studying to determine if they can be analytically understood. Last, while the proposed algorithms can be used in a stream classification setting per run, it would be good to study these algorithms across runs and environments to see their effects on different environments and under conditions like concept drift.

REFERENCES

[1] Geng, C., Huang, S. J., and Chen, S., "Recent advances in open set recognition: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , 1–1 (2020).

[2] Wu, W., Keller, J. M., Dale, J., and Bezdek, J. C., "Streamsong: A soft streaming classification approach," (2020).

[3] Jocher, G., Stoken, A., Borovec, J., NanoCode012, ChristopherSTAN, Changyu, L., Laughing, tkianai, yxNONG, Hogan, A., lorenzomammana, AlexWang1900, Chaurasia, A., Diaconu, L., Marc, wanghaoyang0106, ml5ah, Doug, Durgesh, Ingham, F., Frederik, Guilhen, Colmagro, A., Ye, H., Jacobsolawetz, Poznanski, J., Fang, J., Kim, J., Doan, K., and , L. Y., "ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration," (Jan. 2021).

[4] Stauffer, C. and Grimson, W. E. L., "Adaptive background mixture models for real-time tracking," in [*Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*], **2**, 246–252 Vol. 2 (1999).

[5] Khelifi, L. and Mignotte, M., "Deep learning for change detection in remote sensing images: Comprehensive review and meta-analysis," *IEEE Access* **8**, 126385–126400 (2020).

[6] Varghese, A., Gubbi, J., Ramaswamy, A., and Balamuralidhar, P., "Changenet: A deep learning architecture for visual change detection," in [*Computer Vision – ECCV 2018 Workshops*], Leal-Taixé, L. and Roth, S., eds., 129–145, Springer International Publishing, Cham (2019).

[7] Wang, W. and Shen, J., "Deep visual attention prediction," *IEEE Transactions on Image Processing* **27**(5), 2368–2378 (2018).

[8] Hara, K., Liu, M.-Y., Tuzel, O., and Farahmand, A.-m., "Attentional network for visual object detection," (02 2017).

[9] YuanQiang, C., Du, D., Zhang, L., Wen, L., Wang, W., Wu, Y., and Lyu, S., "Guided attention network for object detection and counting on drones," in [*Proceedings of the 28th ACM International Conference on Multimedia*], *MM '20*, 709–717, Association for Computing Machinery, New York, NY, USA (2020).

[10] Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazirbas, C., Golkov, V., v. d. Smagt, P., Cremers, D., and Brox, T., "Flownet: Learning optical flow with convolutional networks," in [*2015 IEEE International Conference on Computer Vision (ICCV)*], 2758–2766 (2015).

[11] "Modular Neighborhood Pack." https://www.unrealengine.com/marketplace/en-US/product/modular-neighborhood-pack. (Accessed: 1 March 2021).

[12] "Unreal Marketplace." https://www.unrealengine.com/marketplace/en-US/store. (Accessed: 1 March 2021).

[13] "Unreal Engine." https://www.unrealengine.com/. (Accessed: 1 March 2021).

[14] "AirSim." https://github.com/microsoft/AirSim. (Accessed: 1 March 2021).

[15] Pueyo, P., Cristofalo, E., Montijano, E., and Schwager, M., "Cinemairsim: A camera-realistic robotics simulator for cinematographic purposes," (2020).

[16] Buck, A., Deardorff, M., Anderson, D. T., Wilkin, T., Keller, J. M., Scott, G., III, R. H. L., and Camaioni, R., "Vader: A hardware and simulation platform for visuallyaware drone autonomy research," in [*SPIE*], (2021).

[17] Deardorff, M., Alvey, B., Anderson, D. T., Keller, J. M., Scott, G., Ho, D., Buck, A., and Yang, C., "Metadata enabled contextual sensor fusion for unmannedaerial system-based explosive hazard detection," in [*SPIE*], (2021).

[18] Alvey, B., Anderson, D. T., Keller, J. M., Buck, A., Scott, G., Ho, D., Yang, C., and Libbey, B., "Improving explosive hazard detection with simulated and augmented data for an unmanned aerial system," in [*SPIE*], (2021).

[19] "VoTT." https://github.com/microsoft/VoTT. (Accessed: 1 March 2021).