HUMAN-ASSISTED SELF-SUPERVISED LABELING OF LARGE DATA SETS

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Jeffrey Schulz

Dr. Derek T. Anderson, Thesis Supervisor

MAY 2022

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

HUMAN-ASSISTED SELF-SUPERVISED LABELING OF
LARGE DATA SETS

presented by Jeffrey Schulz,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Derek T. Anderson

James M. Keller

Dr. Grant J Scott

Dr. Mikhail Popescu

ACKNOWLEDGMENTS

I would like to thank my parents, Tom and Lori, for their never-ending love and support in my journey. They have always inspired and motivated me, including starting my path in computer engineering by encouraging me to attend that computer-building camp all those years ago. Thank You. Without you, none of this is possible.

I would like to thank Dr. Anderson for his help along my journey to a graduate degree. He turned what I first expected to be simply a piece of paper into a rich and unforgettable period of my life. His patience and expertise enabled learning that went far beyond the scope of the classroom. I can safely say I will be guided by his wisdom for years to come.

And finally, thanks to my brother Justin for listening to me ramble on about computer engineering. I know you couldn't have cared less, but it helped more than you can imagine!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

There is a severe demand for, and shortage of, large accurately labeled datasets to train supervised computational intelligence (CI) algorithms in domains like unmanned aerial systems (UAS) and autonomous vehicles. This has hindered our ability to develop and deploy various computer vision algorithms in/across environments and niche domains for tasks like detection, localization, and tracking. Herein, I propose a new human-in-the-loop (HITL) based growing neural gas (GNG) algorithm to minimize human intervention during labeling large UAS data collections over a shared geospatial area. Specifically, I address human driven events like new class identification and mistake correction. I also address algorithm-centric operations like new pattern discovery and self-supervised labeling. Pattern discovery and identification through self-supervised labeling is made possible through open set recognition (OSR). Herein, I propose a classifier with the ability to say "I don't know" to identify outliers in the data and bootstrap deep learning (DL) models, specifically convolutional neural networks (CNNs), with the ability to classify on N+1 classes. The effectiveness of the algorithms are demonstrated using simulated realistic ray-traced low altitude UAS data from the Unreal Engine. The results show that it is possible to increase speed and reduce mental fatigue over hand labeling large image datasets.

Chapter 1

INTRODUCTION

1.1 SO MUCH DATA, SO LITTLE TIME

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 neural networks (RNNs), generally require large amounts of labeled training data, which in most cases require manual human labeling. Also, our modern era of computational intelligence (CI), ML, AI, and related, is focused intensely on highly specialized domain/task specific learners. In this era of increasingly specialized algorithms for domains such as self-driving, medical diagnosis, and unmanned aerial systems (UAS), labeling training data sets has become a daunting endeavor. As a result, numerous companies have emerged and raised tens of billions of dollars to label data for ML [1]. 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.

While leaps in recognition performance have been made via DL applications, one drawback is that DL generally abides by a *closed world* assumption, meaning the network was trained to regress or predict a certain finite number of variables or classes that were present in the training data. This means that closed-world DL models like CNNs cannot predict classes they were not trained on. In applications with large or quickly populating data sets, online learning is a popular approach to training

ML models. In online ML, data is received sequentially and the knowledge of the model most closely represents recent data. Algorithms in online ML operate without information such as number of total classes or total data points. More specifically, a classifier implemented with online learning in mind (e.g. a stream classifier [2]) could have the ability to identify beyond a finite number of possibilities, and as such would not be inhibited by the closed-world assumption. This *open set recognition* (OSR) (see ref. [3] for a recent survey) has begun to formalize this concept of open versus closed set.

Herein, I first discuss the possibility for online ML extensions of current DL applications for OSR and allowing DL classifiers to say "I don't know." Then, I discuss the application of a user interface for labeling of large data sets for supervised ML tasks.

1.2 OPEN SET RECOGNITION FOR DEEP LEARNING

A number of challenges and questions arise as we attempt to add a new class to a pre-trained DL model. 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?" 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? I discuss this in further detail in Chapter 2. The desire to detect new classes is hereafter referred to as the "$N + 1$" problem (where N is the number of known classes).

OSR formalizes Known-Knowns (KK), Known-Unkowns (KU), Unknown-Knowns (UK), and Unknown-Unknowns (UU). Most DL operates on a *closed set* premises and data sets are comprised of only KKs and KUs. Closed set DL models (e.g. ResNet50, a CNN) have a hard time predicting on data they were not trained on, and therefore the introduction of UKs and UUs in the data should expect to result in undefined model behavior. In offline applications of $N+1$, transfer learning can be implemented

to greatly increase the speed of adopting new classes into the system. The point is, there needs to exist a more elegant way to extend DL models, in my case CNNs, to class $N + 1$ without requiring extensive and expensive offline retraining. In this dissertation, I frame the problem as such that machines should possess a sufficient way to say "I don't know what this is". From there, we can implement proven online ML methods for folding new knowledge into the system.

Distance-based classification methods, such as the k-nearest neighbor (KNN), compute the classification of a query point using the class label of nearby known points. Unlike the fully connected nature of multi-layer perceptron (MLP) classification layers of a CNN, distance-based classification methods have no fundamental limit to number of known classes. Alternative methods to KNN, including fuzzy k-nearest neighbor (FKNN) [4] and possibilistic k-nearest neighbor (PKNN) [5], compute typicalities on queries of ownership to each of N classes. These typicalities can be thresholded (FKNN), or are already thresholded (PKNN), to allow the classifier to say "I don't know what this is" when the query does not have high enough typicality to any known class. Herein, I implement a distance-based classification approach in PKNN to allow $N + 1$ classification at decision time in an online ML application for labeling large data sets.

1.3 ONLINE LEARNING AND HITL TO OBTAIN SUPERVISED LEARNING DATA

Today's supervised learning applications require tremendous amounts of data to perform well on large scale image recognition tasks. 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. This issue has been of importance in the field as of late as many try to get around this bottleneck. Different approaches are be-

ing explored, from transfer learning large models, building a generative adversarial network (GAN) structured automated labeled data augmentation method [6], and unsupervised domain adaptation with a human-in-the-loop (HITL) [7]. While approaching the problem uniquely, each attempt has sights set on eventually being able to automatically label data (that's the dream, right?).

Labeling data automatically, though, can be a tricky problem. Labeling is not a problem that can be *mostly* solved, no, training data sets are no use to an algorithm unless they are completely accurate. A common adage is, "ML/AI models are only as good as the data you give them." With the fact that no current ML/AI algorithms are smart enough to run un-tethered, it is difficult to imagine a solution at our current time where data labeling can be a truly unsupervised task.

Modern ML/AI is not good enough to label full data sets without supervision. In all cases, a human expert is necessary to label large data sets. This HITL approach has proven application in the field for data labeling. We are not the first to investigate online learning, HITL, and OSR to train CI algorithms or label data. Many real-world applications are impacted by these needs, e.g., autonomous driving, drones, security and defense, healthcare, and medicine [8] [9] [10]. For example, Telsa's data-labeling approach uses a HITL to improve the accuracy of their autonomous labeling algorithms. In a 2021 talk, Andrej Karpathy describes Tesla's self-supervised labeling system and reinforces the importance and need of human intervention in large-scale autonomous data labeling [11].

The HITL approach to data labeling requires a human expert to *teach* the algorithm the correct labels *as it runs.* This teacher and student approach is the current state of the art of HITL algorithms, where the student algorithm is assumed at some point to become smart enough to run mostly without the help of the teacher human. This relationship raises an interesting question, does the algorithm to ever reach a point where it can run unsupervised, without any oversight from the human? Can

the human trust the design and performance of the algorithm to label data sets untethered and far quicker than any human could be able to supervise? My answer to this, as explored in this paper, is "no". In labeling data, as discussed above, we should expect near perfect classification results. Herein, I implement a HITL approach for improving the self-supervised classification of training data for large scale image recognition applications.

1.4 CONTRIBUTIONS

This dissertation focuses on two challenges. The first is how to *efficiently* extend CNNs to class N+1 to support tasks like explosive hazard detection (EHD) and automatic target recognition (ATR). To this end, Chapter 2 investigates different classifiers, namely fuzzy set theoretic methods, and their performance versus the conventional CNN fully-connected MLP classifier. Specific classifiers explored herein include the FKNN and the PKNN, including examinations into their ability to say "I don't know". To this end, Chapter 2 proposes a way to learn the *bandwidth* (the mechanism by which to say "I don't know") in the PKNN. Next, various CNN models are challenged to predict on N+1 classes. The first challenge listed here is concluded by proposing an approach based on cluster validity assessment to measure the degree, in $[0, 1]$, to which various CNN models can add class N+1 (or not).

In the second challenge, the idea of having a human help a machine learn and refine concepts is proposed. Specifically, this dissertation focuses on an extension of self-supervised online classification using a 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. To begin, Chapter 3 details the first application of streaming soft neural gas (StreamSoNG) [12] on a streaming computer vision task [13]. 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. Secondly, Chapter 4 extends PKNN and an online growing neural gas (GNG) [14] to create **S**elf-**S**upervised and **H**uman-**I**n-the-loop Growing **N**eural **G**as for HITL labeling of large data sets with a self-supervised online growing distance-based classifier [15]. This dissertation concludes by demonstrating an application of SSHING for labeling large data sets of simulated UAS data which results in quicker labeling speeds and a reduction of labeling loads from manual labeling alone.

Chapter 2

EXTENDING DEEP LEARNING TO NEW CLASSES WITHOUT RETRAINING

Jeffrey Schulz[a], Charlie Veal[a], Andrew Buck[a], Derek T. Anderson[a],
James M. Keller[a], Mihail Popescu[a], Grant J. Scott[a], Dominic K. C. Ho[a],
Timothy Wilkin[b]

[a] Department of Electrical Engineering and Computer Science, University of Missouri, Columbia MO, USA

[b] School of Information Technology, Deakin University, Geelong, Victoria, AU

Abstract

The focus of this article is extending classifiers from N classes to N+1 classes without retraining for tasks like explosive hazard detection (EHD) and automatic target recognition (ATR). In recent years, deep learning has become state-of-the-art across domains. However, algorithms like convolutional neural networks (CNNs) suffer from the assumption of a closed-world model. That is, once a model is learned, a new class cannot usually be added without changes in the architecture and retraining. Herein, we put forth a way to extend a number of deep learning algorithms while keeping their features in a locked state; i.e., features are not retrained for the new N+1 class. Different feature transformations, metrics, and classifiers are explored to assess the degree to which a new sample belongs to one of the N classes and a decision rule is used for classification. Whereas this extends a deep learner, it does not tell us if a network with locked features has the potential to be extended. Therefore, we put forth a new method based on visually assessing cluster tendency to assess the degree to which a deep learner can be extended (or not). Lastly, while we are primarily focused on tasks like aerial EHD and ATR, experiments herein are for benchmark community data sets for sake of reproducible research.

Keywords: convolutional neural network, explosive hazard detection, EHD, automatic target recognition, ATR, possibilistic K nearest-neighbor, PKNN, clustering, clustering in ordered dissimilarity data, CLODD

2.1 INTRODUCTION

Our modern era of computational intelligence, machine learning, artificial intelligence, and related, is focused intensely on highly specialized domain/task specific learners. A well-known example is the convolutional neural network (CNN) and deep learning. While leaps in performance have been made via deep learning, a drawback is that they generally abide by a "closed world" assumption, meaning the network was trained to regress or predict a certain finite number of factors or classes that were present in the training data. However, what happens if a user desires to add a new class/object, e.g., new target in automatic target recognition (ATR)? Current practice is to retrain a network from scratch—which is generally an offline and resource expensive operation—or the feature extraction layers can be retained and the classification layers retrained (a type of transfer learning). The point is, we need a more elegant way to extend models like a CNN to class "N+1" without requiring extensive and expensive offline retraining. Furthermore, most CNNs utilize a normalization strategy like soft max. Whereas this often helps during training (and perhaps testing), it turns a network into a *one of N* classifier. The point it, machines should possess a sufficient way to say "I don't know what this is".

Herein, we focus on two challenges. The first is how to *efficiently* extend modern deep learning to class N+1 to support tasks like explosive hazard detection (EHD) and ATR—see Figure 2.1. To this end, we investigate different classifiers from computational intelligence, namely fuzzy set theoretic methods, versus the conventional CNN approach of using a multi layer perceptron (MLP). Specific classifiers explored herein include the fuzzy K nearest neighbor (FKNN) and the possibilistic K nearest neighbor (PKNN). We explore the performance of these classifiers and their ability to say "I don't know". To this end, we propose a way to learn the *bandwidth* (the mechanism by which we say "I don't know") in the PKNN. Next, we investigate the

Table 2.1: Acronyms and Notation

KNN	K nearest neighbor
PKNN	Possibilistic KNN
CLODD	Clustering in ordered dissimilarity data
K	Number of neighbors for evaluation
$\hat{K}$	Number of neighbors to fit in PKNN
$\mathbf{x}_i \in X$	Train dataset of N samples and M dimension
$\mathbf{f}_i \in \Re^{\hat{M}}$	Neural feature encoding
$[D]_{ij} = D(i,j)$	Dissimilarity of $\mathbf{x}_i$ and $\mathbf{x}_j$

challenge of predicting our models ability to go to N+1—see Figure 2.2. The reality is, a CNN is generally at least two parts, feature extraction and classification. What features did the network learn? Instead of assuming that we can just add "N+1", we propose an approach based on cluster validity assessment to measure the degree, in $[0, 1]$, to which we can add class N+1 (or not).

The remainder of this article is organized as follows. First, in Section 2.2 we discuss features and feature extraction, in Section 2.3 we discuss our PKNN implementation and how to learn the PKNN from data, Section 2.4 discusses our method of assessing if a deep learning model can be extended to class N+1, and Section 2.5 discusses our experiments and results.

2.2 FEATURE EXTRACTION AND DIMENSIONALITY REDUCTION

To no surprise, artificial intelligence, machine learning, pattern recognition, etc., are only as good as the information (e.g., data and associated features) they are provided. For decades our research community has exerted significant effort in manually identifying "manual or hand crafted features". In signal, and specifically image, processing, this typically equates to information like color, texture, and shape. An array of features have been proposed to date, from Gabor filters to histogram of gradients, fractals, Harris, etc. However, the last decade has witnessed a shift from hand crafted

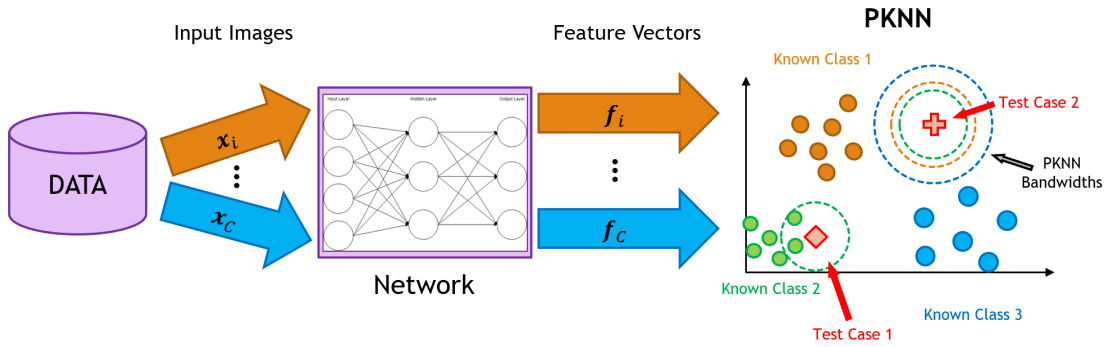


Figure 2.1: High-level illustration of how we extend a deep learner to class N+1. First, a network, e.g., CNN, is trained. Second, we strip off the classification layers (the MLP). As a result, each input is presented to a CNN for feature extraction. These features are then passed to a classifier like the KNN, FKNN, or PKNN. Based on the resultant typicalities, a sample is classified into a known class or the system reports "I don't know".

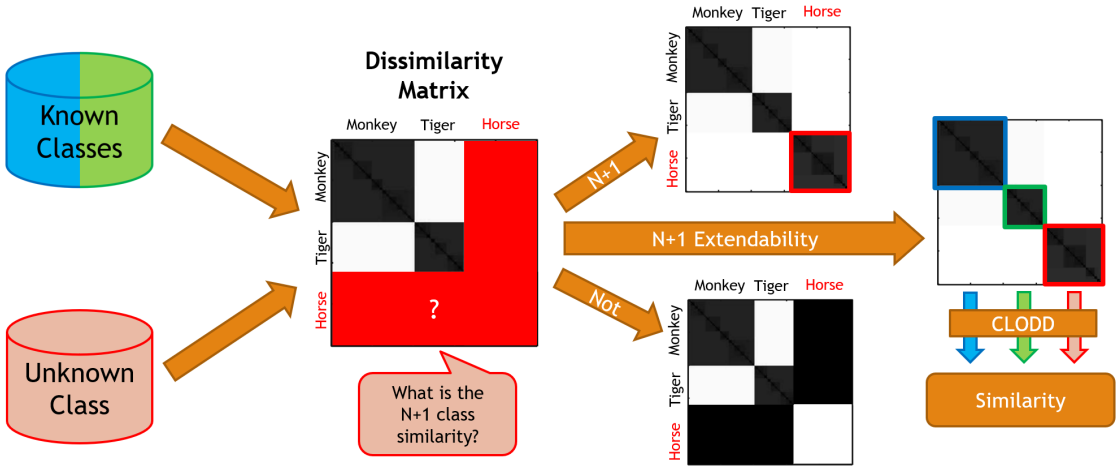


Figure 2.2: Illustration of our procedure for measuring the degree to which a model can be extended to class N+1. We start with known classes that the model has been trained to recognize. Next, a set of instances from class N+1 is provided. A dissimilarity matrix is constructed, which we can visually assess to determine how well the model can extend. Furthermore, aspects of the CLODD algorithm are used to formally extract a number, or the $[0, 1]$ degree to which the network is extendable.

features to "machine learned" features.

In the context of neural networks, specifically CNNs, the story is that they learn basic shape, color, and/or texture features in lower layers, deeper layers are associated with more complicated and compound shapes/objects, followed eventually by higher-level relationships and semantic information. Regardless of what is learned, the fact is that CNNs consist of layers of filters, which when presented with data provide a response field. Consider the popular Resnet51. If one removes the "classification layers" of Resnet51, then they obtain a final response field of $2,048$ *features*. Specifically, the standard Resnet51 has a $7x7$ response field per feature, which means that Resnet51 has a dimensionality of $2,048*7*7$. Herein, in our pursuit of extending our models to class $N+1$, we investigate nearest neighbors versus an MLP classifier layer. This presents a dilemma as we have to compute, and use, distances between samples in high dimensional spaces. The reader can refer to [16] for a discourse on the challenges of assessing proximity in high dimensional spaces. Herein, in an effort to combat the curse of dimensionality, we explore a few reduction strategies to empirically assess if they help classification.

2.2.1 Reduction Strategy 1: Pooling

The first strategy that we explore is pooling. In the context of a CNN, pooling helps us combat a number of factors, from dimensionality reduction to combating scale and even noise. The output feature vector for Resnet51 is of size $2,048 * 7 * 7$. In an attempt to side step affine transformations that we are not concerned with, we pool within each feature map. Herein, we use max pooling, i.e., we take the largest response per feature (response field). A demonstration of max pooling across a $4x4$

response field is

$$\begin{matrix} 0.2 & 0.1 & 0 & 0.6 \\ 0.1 & 0.3 & 0.4 & 0.5 \\ 0.7 & 0.6 & 0.8 & 0.5 \\ 0.4 & 0.3 & 0.4 & 0.2 \end{matrix} \rightarrow 0.8.$$

Whereas we use max pooling, the reader can engage in a variety of aggregation operations for pooling, i.e., min, median, or more complicated and unique operators like the ordered weighted average[17] or fuzzy integral[18, 19]. Max, a generalized union operator, is an appropriate fit for our task as it captures the idea of what was our strongest response per feature. In summary, for a CNN like Resnet51, pooling is a simple procedure that helps us reduce dimensionality from $2,048x7x7$ to $2,048$, at the expense of where in the spatial domain the feature occurred and how many instances of that feature were observed.

2.2.2 Reduction Strategy 2: Principal Component Analysis

A second, yet simple, dimensional reduction technique explored herein is principal component analysis (PCA). PCA is an unsupervised method that is a linear transformation whose basis vectors are identified on the premises of maximizing variance. We perform an eigen decomposition, which results in D eigenvectors with corresponding eigenvalues for a D dimensional space. Herein, we select the fewest number of eigenvectors whose eigenvalue sum corresponds to more than %99 of the overall data variation. However, PCA is a linear transformation and there is no guarantee that patterns or clusters in the original high dimensional space are preserved in the reduced low dimensional space. For additional methods, the reader can refer to additional unsupervised methods like random projections or supervised methods like Fishers linear discriminate analysis.

2.3 CLASSIFICATION: POSSIBILISTIC KNN

Next, we discuss our strategy for taking neural features and realizing a classifier that can help extend to $N+1$ and say "I don't know". To this end, the PKNN improves the KNN by improving the semantics of the nearest neighbor memberships and enabling outlier detection. A membership value can be calculated from the KNN via

$$\mu_{knn}^{i}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_i(\mathbf{y}_k), \tag{2.1}$$

$$\mathbb{1}_i(\mathbf{y}_k) = \begin{cases} 1 & \text{if sample } \mathbf{y}_k \text{ belongs to class } i \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Clearly, $\mu_{knn}^{i}(x) \in [0,1]$ is the number of points in class i relative to all of the nearest neighbors (K). In [4], Keller et al. proposed an extension to the KNN, the fuzzy KNN (FKNN),

$$\mu_{fknn}^{i}(\mathbf{x}) = \frac{\sum_{j=1}^{K} \tilde{\mu}^{i}(\mathbf{y}_j)\left(\frac{1}{||\mathbf{x}-\mathbf{y}_j||^{\frac{2}{m-1}}}\right)}{\sum_{j=1}^{K}\left(\frac{1}{||\mathbf{x}-\mathbf{y}_j||^{\frac{2}{m-1}}}\right)}, \tag{2.3}$$

which is a membership weighted and inverse distance ratio of the K nearest neighbors. For each training sample data point, a fuzzy membership is computed offline,

$$\tilde{\mu}^{i}(\mathbf{x}) = \begin{cases} 0.51 + (\frac{n_i}{K}) \times 0.49, & \text{if } \mathbf{x} \text{ belongs to class } i \\ (\frac{n_i}{K}) \times 0.49, & \text{otherwise,} \end{cases} \tag{2.4}$$

where $1 \leq n_i \leq K$ is the number of neighbors that belong to class i. Thus, a membership degree is computed per data point and per class relative to its local K nearest neighbors. If the sample point ($\mathbf{x}$) belongs to class i, then a *reward* of 0.51 is added to start, plus the ratio of K points in class i. Otherwise, the membership is restricted to $0 \leq \tilde{\mu}^{i}(\mathbf{x}) \leq 0.49$. Imagine a sample surrounded by data points from

the opposing class. The assumption here is that our sample (that belongs to class i) is not *noise*. The resultant membership degree (i.e., $\tilde{\mu}^i(\mathbf{x})$) is 0.51, whereas it would have little-to-no voice in the KNN. In general, one expects the FKNN to be the most beneficial in overlapping regions. During test time, we calculate $\mu^i_{fknn}(\mathbf{x})$, which exploits these membership degrees. Consider a scenario in which a test sample has one retrieval from class i and the other nearest neighbors belong to a different class. In the KNN, we would not select class i. However, in the FKNN, depending on the proximity of $\mathbf{x}$ to the sample in class i, it is possible to generate $\mu^i_{fknn}(\mathbf{x}) = 1$ when $\mathbf{x}$ is sufficiently *close* to the ith sample. This is one story. There are clearly other FKNN benefits that are more complicated, e.g., related to memberships and relative neighbor distances.

In [5], Frigui and et al. created the possibilistic KNN (PKNN), an extension of the FKNN and KNN,

$$\mu^i_{pknn_1}(\mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K} \tilde{\mu}^i(\mathbf{y}_j) w^p(\mathbf{x}, \mathbf{y}_j), \tag{2.5}$$

where $\mu^i_{pknn_1}(\mathbf{x})$ is the typicality–e.g., degree of similarity–of a test sample to a known class. Equation 2.5 sums over K neighbors the product of the training data fuzzy memberships and the possibilistic score w^p (explained below). Herein, we also explore an unweighted version of the PKNN,

$$\mu^i_{pknn_2}(\mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K} w^p(\mathbf{x}, \mathbf{y}_j). \tag{2.6}$$

The possibilistic score, $w^p(\cdot)$, is where the PKNN truly differs from the FKNN and KNN. The PKNN uses a *bandwidth factor*, η, to take into account the distance of the test case to the prototypes. If a data point is within the bandwidth, then the membership value is 1. Otherwise, the membership value decays, allowing points to

have low-to-no membership. The possibilistic scoring is

$$w^p(\mathbf{x}, \mathbf{y}_j) = \frac{1}{1 + [\max(0, \|\mathbf{x} - \mathbf{y}_j\| - \eta)]^{2/(m-1)}}. \tag{2.7}$$

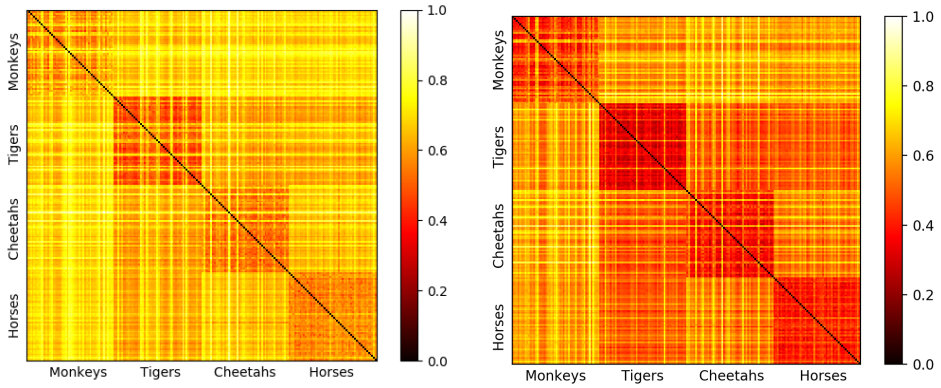
In[5], Frigui et al. derived η from data. Specifically, Frigui let $\eta = \frac{\eta_1}{\eta_2}$, where η_1 is the mean μ of the distances of the K closest neighbors in the training data and η_2 is three times the standard deviation of the same set.

2.3.1 Bandwidth Parameter Optimization

Not all data and patterns are created equal, which poses problems for the η parameter presented in Equation 3.1. Whereas the parameter estimation method proposed by Frigui et al. led to improvement for their explosive hazard detection task, it does not necessarily generalize. In our experiments, datasets like ImageNet have relatively large standard deviations which result in tiny η values. This required us to scale Frigui's η to produce positive results. While this is useful, it increased the amount of hand-picking and fine tuning required to get a PKNN classifier running. To combat this problem, we propose to learn the bandwidths from data. To this end, we formulate a cost function, $J(w^p, \eta, \alpha)$, and a genetic algorithm is used for optimization as the equation does not have a nicely differential or analytical solution. The cost function explored herein is

$$J(w^p, \eta, \alpha) = (1 - \frac{w^p_{i,i}}{C}) + \frac{w^p_{i,j}}{C} + \alpha \sum_{i=1}^{C} |\eta_i| \,, \tag{2.8}$$

where w^p is the typicalities of our validation samples, $w^p_{i,i}$ is the sum of true-positive typicalities (where only the typicalities belonging to the correct class are totaled), and $w^p_{i,j}$ is the sum of false-positive typicalities (where only the typicalities belonging to incorrect classes are totaled). The cost function is rounded out with a summation factor, scaled by α (a user defined parameter), meant to punish larger bandwidths. This



(a) No Dimensionality Reduction (b) Max-Pooling Reduction

Figure 2.3: ODM displaying 50 images per class of Monkeys, Tigers, Cheetahs, and Horses (the N+1 class). The displayed data is the feature activation from ResNet101 max-pooled to size 1x2048. For this class balanced example, the best case scenario is four dark diagonal squares with white in the off diagonals.

factor prevents bandwidths from ballooning farther than they need to be, increasing the chance for future $N+1$ classes to fall outside of η in feature space.

2.4 ASSESSING EXTENDING TO CLASS N+1 AND SAYING "I DON'T KNOW"

The goal of this section is two fold. First, we desire a visual way to see and understand the quality of different deep learning algorithms and models with respect to the N classes it was trained on and new unknown classes. Second, we desire a way to move this qualitative process into a quantitative procedure. The following section outlines these two approaches.

2.4.1 Qualitative: Visualizing Architectures, Models, and Features

First, we focus on qualitative assessment. If we, as humans, can determine separability between the features of known and unknown classes when displayed in certain data visualization strategies, then it might be good news for our deep learning models. In order to qualitatively determine if a model is able to separate the features of a new

class from an existing class, we will use an ordered dissimilarity matrix (ODM). ODMs are matrices built from our data set by computing a distance function, e.g., l_p norm, between all possible pairings of image feature vectors. For sake of description, assume we have N classes. Furthermore, assume, without loss of generality that there is an equal number of samples per class. In the ideal case, all samples in a class are similar and have a low distance. Conversely, samples in a class are dissimilar to samples from other classes. As a result, when one "looks" at an ODM they will see "dark blocks" along the diagonal per class and "white rectangles" in the opposing classes. Whereas all distances are positive, in order to draw an ODM one can engage in a strategy like range compressing the ODM between the min and max value for visual display. A dissimilarity matrix (not ordered), like what we will use in this paper, is shown in Figure 2.3.

In the context of clustering (unsupervised learning), one typically uses a procedure like VAT or iVAT[20], for ordering the samples and enhancing the visual structure of an ODM. Herein, we have class labels, we are doing supervised learning. While we could reorder samples in each class via VAT/iVAT, one can regardless make out the structure and class separation in Figure 2.3. However, for larger sets of samples, we recommend running VAT per class and the enhancement step of iVAT on the permuted VAT ODM/image. As Figure 2.3 shows, our N+1 class, horses, has a distinguishable dark box, indicating similarity to itself and dissimilarity to the rest of the data set (aka other classes). As a result, it suggests that even though the network was not trained on horses, it learned visual features to recognize and distinguish horses. This is a positive indicator that the model can be extended, i.e., the network might not need to be retrained.

2.4.2 Quantitative: Numerical Value Indicating "Goodness"

In order to rank models according to their ability to extend to N+1, we need to establish a measure that somehow determines the separability of the N+1 class features to known class features. This takes the human out of the loop and it enables automating the ranking for large numbers of trained models. Herein, we leverage the measure component of the Clustering in Ordered Dissimilarity Data (CLODD) [21] algorithm, which is a method to automatically discover the number of clusters in a VAT/iVAT image. The equation for CLODD is

$$E_{\alpha}(U; D*) = \alpha E_{sq}(U; D^*) + (1 - \alpha) E_{edge}(U; D^*); 0 \leq \alpha \leq 1, \tag{2.9}$$

where $E_{\alpha}(U; D*)$ is the weighted score between the "edginess" $E_{edge}(U; D^*)$ of the clusters and the "squaredness" $E_{sq}(U; D^*)$ of the clusters. The weight, a user defined parameter, of the "edginess" and "squaredness" factors are determined by the mixing factor α. The equation for squaredness in CLODD is

$$E_{sq}(U; D^*) = \left(\frac{\sum_{i=1}^{c} \sum_{s \in i, t \notin i} d^*_{st}}{\sum_{i=1}^{c} (n - n_i) n_i} \right) - \left(\frac{\sum_{i=1}^{c} \sum_{s,t \in i, s \neq t} d^*_{st}}{\sum_{i=1}^{c} (n_i^2 - n_i)} \right), \tag{2.10}$$

where the average dissimilarity within dark regions is subtracted from the average dissimilarity between dark and non-dark regions. In the first factor, dissimilarity between classes and all other classes ($d^*_{s \in i, t \notin i}$) is averaged. In the second factor, dissimilarity between classes and themselves ($d^*_{s,t \in i, s \neq t}$) is averaged. The equation for edginess in CLODD is

$$E_{edge}(U; D^*) = \frac{1}{c-1} \sum_{j=1}^{c-1} \frac{\sum_{i=m_{j-1}}^{m_j} \left| d^*_{i,m_j} - d^*_{i,m_j+1} \right| + \sum_{i=m_j+1}^{m_{j+1}} \left| d^*_{i,m_j} - d^*_{i,m_j+1} \right|}{n_j + n_{j+1}}, \tag{2.11}$$

where the dissimilarity between one cluster and the next is summed up over all clusters, averaged over the number of samples. Equations 2.10 and 2.11 are combined to create Equation 2.9, which is maximized in the CLODD algorithm to find good clusters.

Herein, we have a slightly different problem than CLODD. Specifically, we are not working with unlabeled data and we do not permute the dissimilarity matrix with respect to underlying cluster structure. Instead, the dissimilarity matrix is organized according to know class labels, which alters the semantics of edginess. As such, two procedures are explored herein.

In Method 1, only squaredness is calculated, not edginess, with respect to class N+1. The idea is to subtract the average of the class N+1 to not N+1 classes from the average of the class N+1 to class N+1 instances. However, we first normalize the dissimilarity matrix by subtracting its minimum and then dividing by the maximum. This normalization is performed for sake of interpretability, i.e., a value of one is best and negative one is the worst possible outcome. In Method 2, we compute Equation 2.10, just squardness on the entire matrix. The difference between Method 1 and Method 2 is, Method 1 says "how well can we separate class N+1 from the other classes", and Method 2 says that plut "how well does the not N+1 classes separate from one another". It might be important to consider both, as the goal would be an ODM with all dark blocks across the diagonal, indicating that class N+1 can be detected and discriminated, but it does not come at the expense of any of the existing classes. This nuance is subtle and elaborated on via example in the results section.

2.5 EXPERIMENTS AND RESULTS

In this section, we investigate three questions: what is our classification accuracy relative to different models and projections; does optimal bandwidth parameter selection lead to performance gain; and is it possible to rank order approaches based on our

Figure 2.4: Our Imperfect Dataset consists of oranges, bananas, and donuts. Oranges and bananas are pulled from ImageNet and donuts are pulled from Food101.

predictive metric. While our primarily goal is detection relative to aerial EHD and ATR, our experiments are performed on benchmark community data sets for sake of reproducible research. Figures 2.4 and 2.5 show samples of the data used for each class.

2.5.1 PKNN-Based Classification

In this subsection, we focus on two data sets with varying levels of complexity, noise, and class similarity.

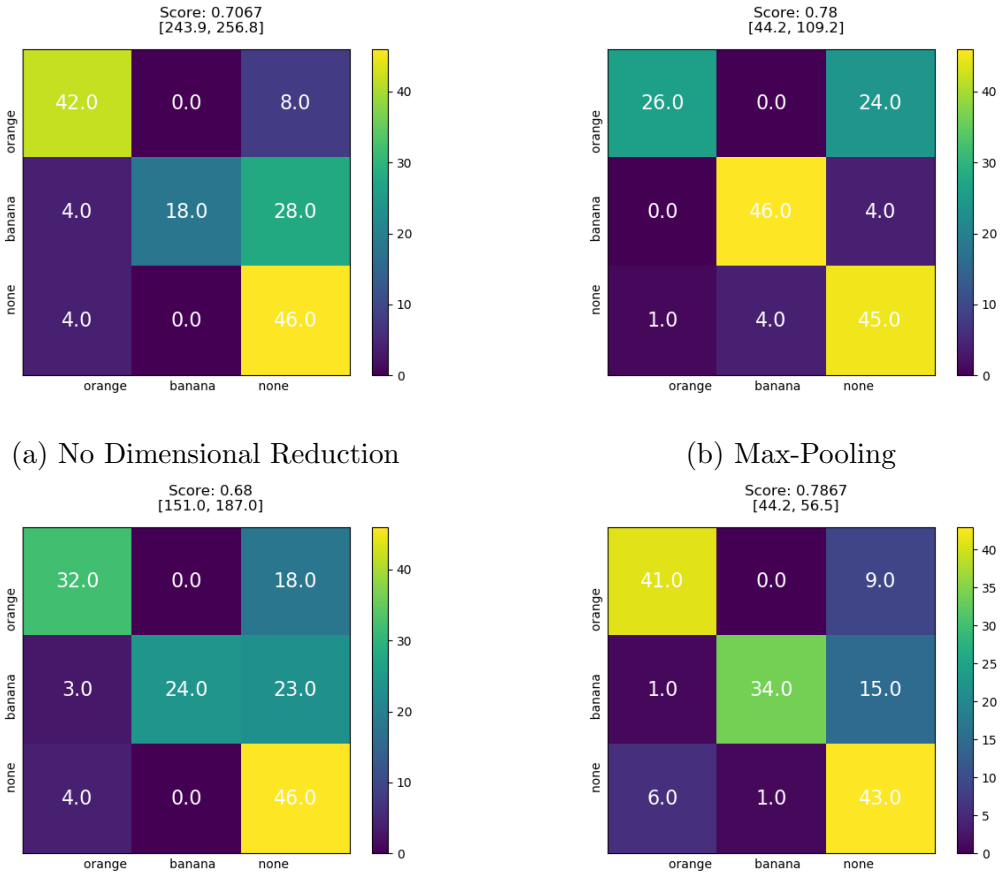
Experiment 1: Visually Challenging Classes and Imperfect Dataset

In Experiment 1 (see Figure 2.6), we demonstrate the classification accuracy of the PKNN on a three class dataset; Oranges, Bananas, and Donuts. Oranges and Bananas are from ImageNet but Donuts (the "class N+1" here) are from the Food101 dataset. The Food101 dataset has noisy labels, at an estimated level of twenty percent. The model under test is ResNet101, pretrained on ImageNet. Since the model is pretrained on ImageNet and ImageNet does not include donuts, presumably our model has not

Figure 2.5: A simpler dataset consists of monkeys, cheetahs, tigers, and horses. Monkeys, cheetahs, and tigers are pulled from ImageNet and horses are hand-picked.

seen class N+1 and it may not be equipped with features to recognize donuts. The weight factor α, from the cost function 2.8 is 0.001 and the typicality threshold for classficiation is 0.34. Optimized bounds are found using a DEAP genetic algorithm in 10 generations with 100 individuals, a 0.5 cross-over rate, and 0.2 mutation rate. In the case of PCA, the data is reduced to 1x200 per sample. When reducing the max-pooled data with PCA, the data is reduced to 1x180.

Experiment 1 tells the following story. First, when no dimensionality reduction is performed, donuts worsen banana classification. For sake of page count, we do not focus on why these miss-classifications occur—e.g., shape, background versus foreground object features, etc.—the reader can refer to methods like GradCAM or convolution matrix transpose (e.g., "deconvolution") for visual explainable AI if desired. PCA is the worst performer and the best solution is max pooling with PCA. That is, the best answer is to reduce dimensionality after performing an optimistic



(a) No Dimensional Reduction

(b) Max-Pooling

(c) PCA

(d) Max-Pooling & PCA

Figure 2.6: Confusion matrices of test data after bandwidth parameter estimation using the raw (high dimensional) data, max-pooled, PCA, and max-pooled data reduced with PCA.

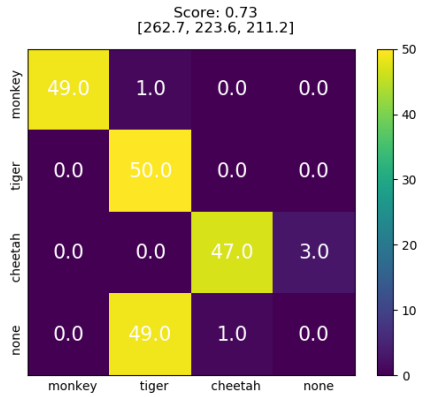
pooling step per feature map. The reader should recall that Food101 has noisy labels and complex backgrounds (aka, image content that is not our class of interest). This is a major reason for selecting these datasets versus a dataset like MNIST; which consists of foreground digits with no background complexity. Furthermore, we determined that it was important to start with a relatively hard visual task, e.g., distinguishing foods, versus something simpler like different animals (Experiment 2).

Experiment 2: More Relevant, Albeit Simpler Dataset

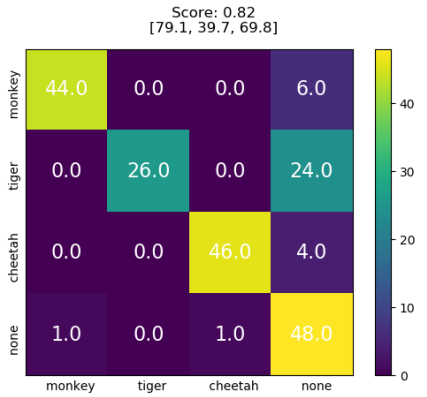
In Experiment 2 (Figure 2.7), we demonstrate the PKNN on a four animal class dataset; Monkeys, Tigers, Cheetahs, and Horses. Monkeys, Tigers, and Cheetahs are from ImageNet, whereas Horses are not, their reference imagery were handpicked by us. The model under test is ResNet101, pretrained on ImageNet. Since the model is pretrained on ImageNet and ImageNet does not include horses, the question is, did our model learn features that can help detect and distinguish horses. The weight factor α, from the cost function 2.8 is 0.001 and the typicality threshold for classficiation is 0.34. Optimized bounds are found using a DEAP genetic algorithm in 10 generations with 100 individuals, 0.5 cross-over rate, and 0.2 mutation rate. When reduced via PCA, the data is reduced to 1x300 per sample and the max-pooled data with PCA is reduced to 1x264.

A limitation of only reporting classification rates and confusion matrices is that the reader cannot see the typicality degrees. For example, a confusion matrix is built with respect to which class has the highest typicality. In Figure 2.7, we show the classification rates, confusion matrices, and stem plots of the typicalities. The PCA stem plot paints a picture where the machine is almost always certain about the known classes, but class N+1 has many high typicalities. On the other hand, max-pooling shows varying confidences in the class examples with little-to-no typicalities in the class N+1. The reader can see that max-pooling reduction led to an accuracy of 82%, while PCA led to a drop of 9%.

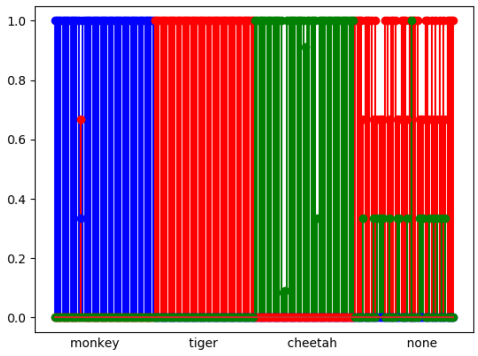
Furthermore, we would like to stress that just because the model does not have a class does not mean that a model has not seen, and possibly built features for class N+1. For example, two unrelated objects can share features and learning features across classes can lead to discriminatory potential. Furthermore, even though ImageNet does not have a class for Horses, it does have a synset for "horse-cart" and other horse-drawn vehicles. We discovered this after identifying and running this



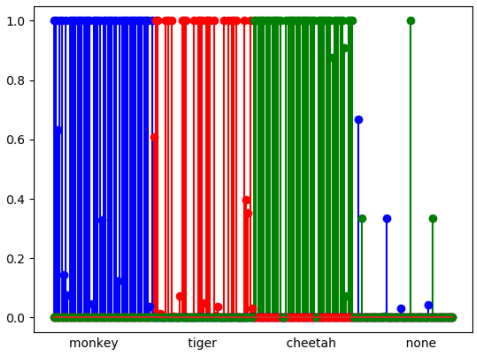
(a) PCA



(b) Max-Pooling



(c) Stem Plot - PCA



(d) Stem Plot - Max-Pooling

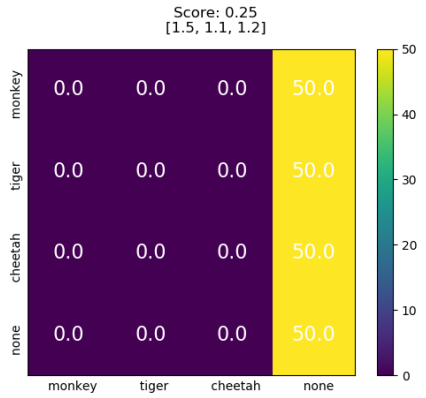
Figure 2.7: Confusion matrices and corresponding typicality stem plots for test data after bandwidth estimation.

experiment. Hence, our success with respect to this experiment—i.e., our ability to add and discriminate class N+1—could be due to the fact that a model pretrained on ImageNet learned/remembers features for horses, even though the specific class is not an option for classification in a traditional CNN pretrained on ImageNet. It is an interesting tidbit and something that the reader should be aware of.

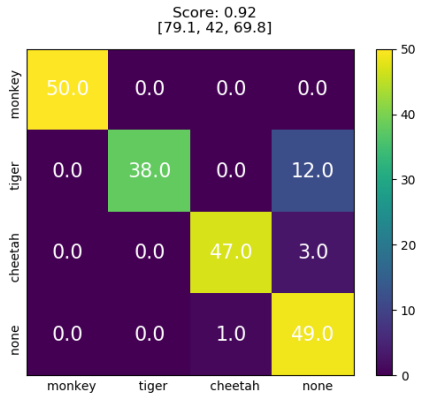
Experiment 3: Bandwidth Optimization

In Experiment 3, we demonstrate that it is important to individually optimize bandwidths for each class as the Frigui et al.[5] method of calculating η based on class statistics does not hold for every dataset. The results of the originally proposed η

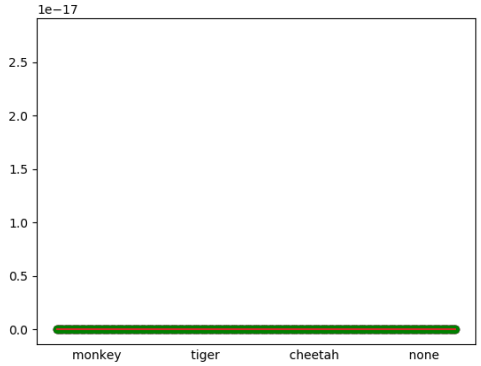
are shown in Figure 2.8, with respect to the same setup as outlined in Experiment 2, except the data was max-pooled then reduced with PCA to 1x264.



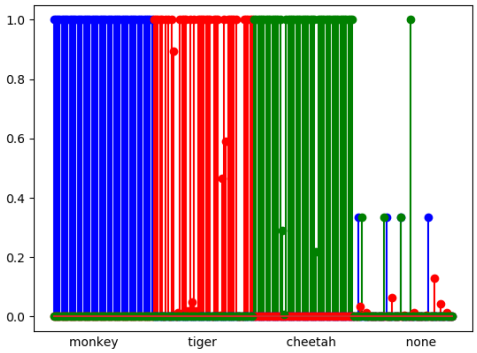
(a) $\eta = \mu/(3 * \sigma)$ Parameters



(b) GA Optimized η



(c) Stem Plot: $\eta = \mu/(3 * \sigma)$



(d) Stem Plot: GA Optimized η

Figure 2.8: Visualization of the confusion matrices and typicality stem plots for the two bandwidth estimation procedures. Figures (a) and (c) show that the data-derived bandwidths are too small, resulting in total classification of "none". The optimized bandwidths for this dataset are clearly higher, as shown in (b) and (d).

Clearly, the bandwidths calculated from class statistics are too small. While they lead to perfect classification for class N+1, they are so tiny that they do not generate any high typicalities for the known N classes. While this selection scheme worked for the explosive hazard detection problem and set of features that Frigui et al. investigated, it does not hold across datasets. This is not alarming as we did expect the optimized bandwidths to perform better, as it has the ability to adapt to

the underlying data/needs.

We want to make sure that the following is clear. The bandwidth optimization achieves a score of 92%, at the expense of mistaking some tigers as horses. While horses are almost perfectly classified, same with cheetahs and monkeys, the neural network has not learn enough features to properly distinguish all of its classes. We can optimize the bandwidth parameters all we like, but we cannot overcome this limitation with the features.

2.5.2 Predicting a Models Ability to Extend to Class N+1

In this subsection we switch gears and we explore both qualitative and quantitative ways to assess if its possible to extend a model to class N+1. The above subsection approached this challenge with respect to the PKNN and classification accuracy. The aim of this subsection is to weaken our assumptions. We desire to determine if it is possible to take an ODM and directly predict a networks extension potential.

Experiment 4: Dimensionality Reduction Technique Assessment

In Experiment 4, we use squaredness (Equation 2.10) on the full ODM (Method 1) and only the N+1 rows (Method 2) to assess the different dimensionality reduction techniques explored herein. That is, we desire to observe if its advantageous to retain the full set of original features or their reduced and more efficient counterparts; the latter being our intuition. For this experiment, we use features from ResNet101 on the animal dataset from Experiment 2. The results for both measures are reported in Table 2.2 and Figure 2.9.

Figure 2.9 and Table 2.2 tell the following story. Overall, max-pooling with PCA is the best. However, PCA achieves a better score with respect to Method 1, Equation 2.10 for just class N+1. That is, PCA alone does the best job rejecting class N+1 samples. However, it does the such at the expense of the other N classes. As the

Table 2.2: Dimensionality reduction techniques on ResNet101 features.

Model	Squaredness, Full Matrix	Squaredness, Sub Matrix
No reduction	0.1106	0.1523
Max-pooling	0.1652	0.4253
PCA	0.2889	0.2344
Max-pooling & PCA	0.2257	0.6732

overall squaredness tells us, max-pooling does a better job discriminating between the N classes and with class N+1. We included this example to illustrate the fact that only listening to rejecting unknown samples is not enough, it cannot come at the expense of the N classes.

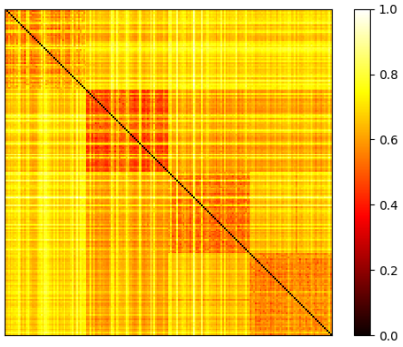
Experiment 5: Assessing Different Architectures and Models

In Experiment 5, we use our squaredness measure to compare and rank various models with respect to their ability to extend to class N+1. For this experiment, we max-pool the features coming out of the network on the animal dataset from Experiment 2. The results for each of the models are reported in Table 2.3 and Figure 2.10.

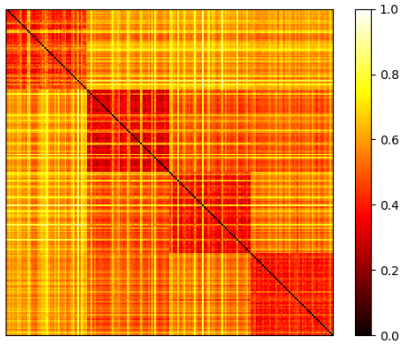
Table 2.3: Experiment 5 results.

Model	Squaredness, Full Mat.	Squaredness, Sub Mat.	Rank	# Features
ResNet101	0.1652	0.2565	1	2048
AlexNet	0.0992	0.0733	4	256
VGG19	0.1237	0.1511	3	512
DenseNet201	0.1333	0.1992	2	1920

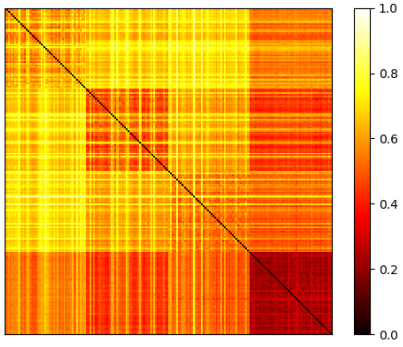
Table 2.3 and Figure 2.10 tell the following story. First, the measure scores align with how we visually would rank the four architectures. Namely, ResNet101 was best, followed by DenseNet201, VGG19, then AlexNet. While this is reinforcing, there is a trend. Namely, the rank ordering of our models align with the number of features in the respective architectures. This might lead one to believe, in general, that the more features the better. However, we are not able to deduce such a conclusion based on



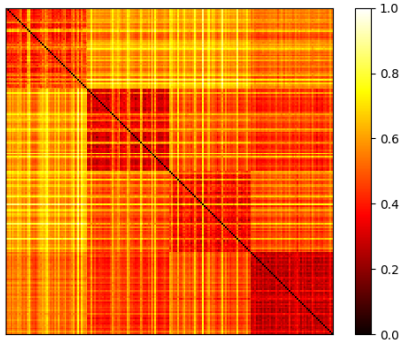
(a) None



(b) Max-pooling



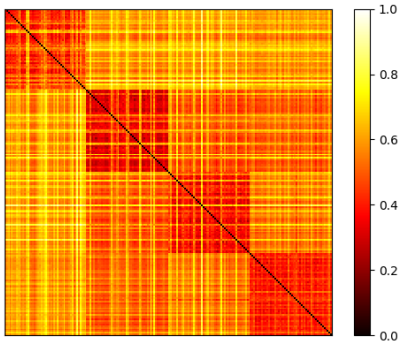
(c) PCA



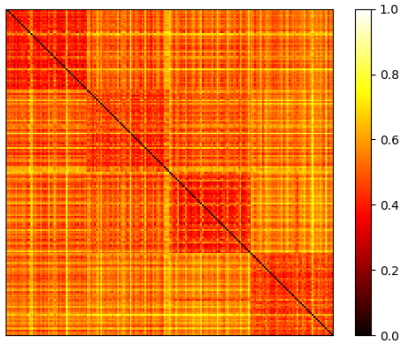
(d) Max-pooling & PCA

Figure 2.9: Dissimilarity matrices (normalized to $[0, 1]$) for Experiment 4.

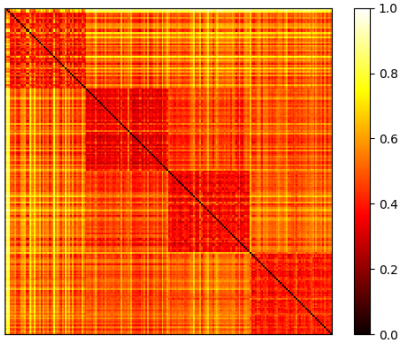
such a simple basis; set of experiments. What Table 2.3 really highlights is the fact that this experiment is apples-2-oranges. That is, each model has a different number of features and as such they are challenging to compare. Again, our results are nice in the respect that they help support the validity of our qualitative and quantitative processes, however the reader would benefit from using the proposed measure across a wider range of models or perhaps apples-2-apples experiments, e.g., like architectures trained on different data, different initializations, etc. All we can logically extract from Experiment 5 is that our measure lines up with what a human would assess and that more features, up to some limiting or diminishing point, possibly result in a richer visual vocabulary that help with extending to class N+1.



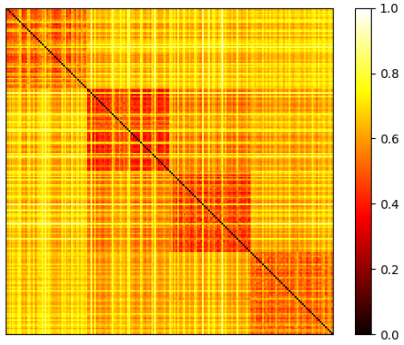
(a) ResNet101



(b) AlexNet



(c) VGG19



(d) DenseNet201

Figure 2.10: Dissimilarity matrices of max-pooled feature activations from each CNN, normalized to $[0, 1]$.

2.6 SUMMARY AND FUTURE WORK

Herein, we explore how to extend artificial neural networks to class N+1 (aka a new class that the network has not been trained on). To this end, we investigate different dimensionality reduction methods to remediate the impact of undesired affine transformations and the curse of dimensionality relative to the proposed possibilistic k nearest neighbor classifier (PKNN). As the PKNN depends on bandwidth parameters, we optimize them using a genetic algorithm. We couple these methods with the generation of ordered dissimilarity matrices and automatic scoring based on the notion of squaredness in CLODD. Our experiments show that the combination of

max pooling and PCA lead to the best classification accuracy, typicalities, and ordered dissimilarity matrices. All of the above results were observed on community benchmark datasets for sake of reproducible research versus some underlying EHD or ATR dataset that cannot be shared. The particular experiments were selected to highlight classes of varying visual complexity.

In future work, we will investigate the following. First, we will explore how to extend the notion of edginess in a supervised context. Second, while the PKNN is useful for rejecting outliers, the mechanism (equation) needs improvement, beyond parameter (bandwidth) optimization. Next, our procedures, e.g., the PKNN and bandwidth selection, are learned independent of the neural network, which is merely performing feature extraction. Ideally, these would be learned in conjunction—parallel or simultaneously—with one another. From an experimental standpoint, a deeper and more thorough analysis is needed across existing architectures and models. Last, this article focuses on detection. A next step will be assessing how to extend the proposed ideas to detection and localization. In summary, we are excited about the preliminary results but more work is needed before a robust real-time solution is in hand.

This work is partially funded by the Army Research Office (ARO) grants numbered W911NF-18-1-0153 and W911NF-19-1-0181 to support the U.S. Army RDECOM CERDEC NVESD.

Chapter 3

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 J. Scott[a], 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

3.1 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 4.1 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. [3] 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?" 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

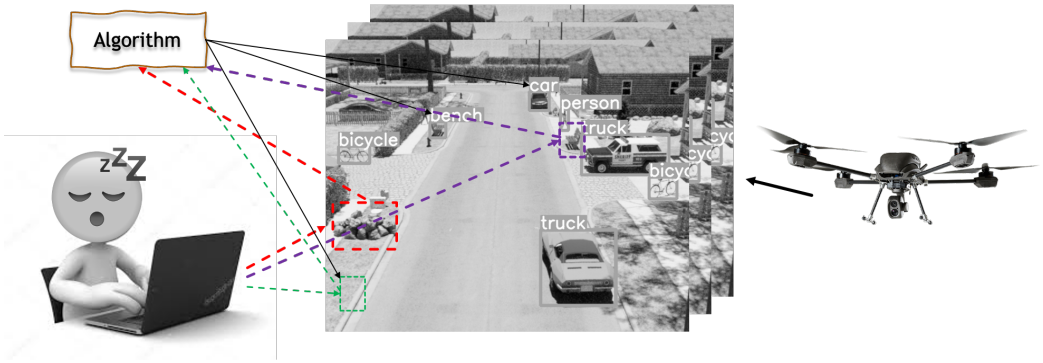


Figure 3.1: Illustration of our application of interest. Low altitude UAV imagery on niche domains is collected and subsequently labeled automatically by a stream classification algorithm. However, algorithms are not perfect; they need help getting started and achieving a desired steady state. A human 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.

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 [12] 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 3.2.1, we discuss our implementation of StreamSoNG. In section 3.2.2, we introduce our user interface with which the HITL supervision is performed. In section 3.3, we discuss preliminary experiments and results. Last, in section 3.4 we discuss future work. Table 3.1 shows acronyms and our notation.

Table 3.1: 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

3.2 METHODS

3.2.1 Stream Classification

As mentioned above, our HITL labeling problem can be cast as an instance of stream classification. In Ref. [12], 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 3.2.

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

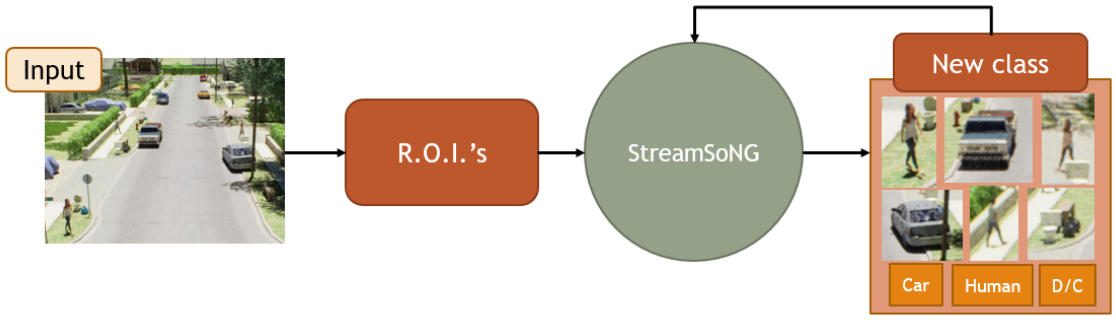


Figure 3.2: After training data is provided to initialize the StreamSoNG algorithm, the streaming test data is streamed into StreamSoNG where clusters and outliers are determined and new classes can emerge. When a new class is detemined to exist, the user is prompted with a "New Class" GUI where he/she can select the objects belonging to the new class. The StreamSoNG algorithm is then updated to reflect those truths.

incoming data points. During the PKNN step, computing typicality has historically been contentious, so in this paper we define our typicality equation by,

$$t'_{ik}(\mathbf{x}_t, \mathbf{p}_{ik}) = \frac{1}{1 + [\max(0, \|\mathbf{x}_t - \mathbf{p}_{ik}\| - \eta)]^{2/(m-1)}}, \tag{3.1}$$

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,

$$\mathbf{p}_{ik}^{t+1} = \mathbf{p}_{ik}^{t} + \alpha * t'_{ik}(\mathbf{x}_t) * e^{-k/\lambda}(\mathbf{x}_t - \mathbf{p}_{ik}^{t}), \tag{3.2}$$

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.

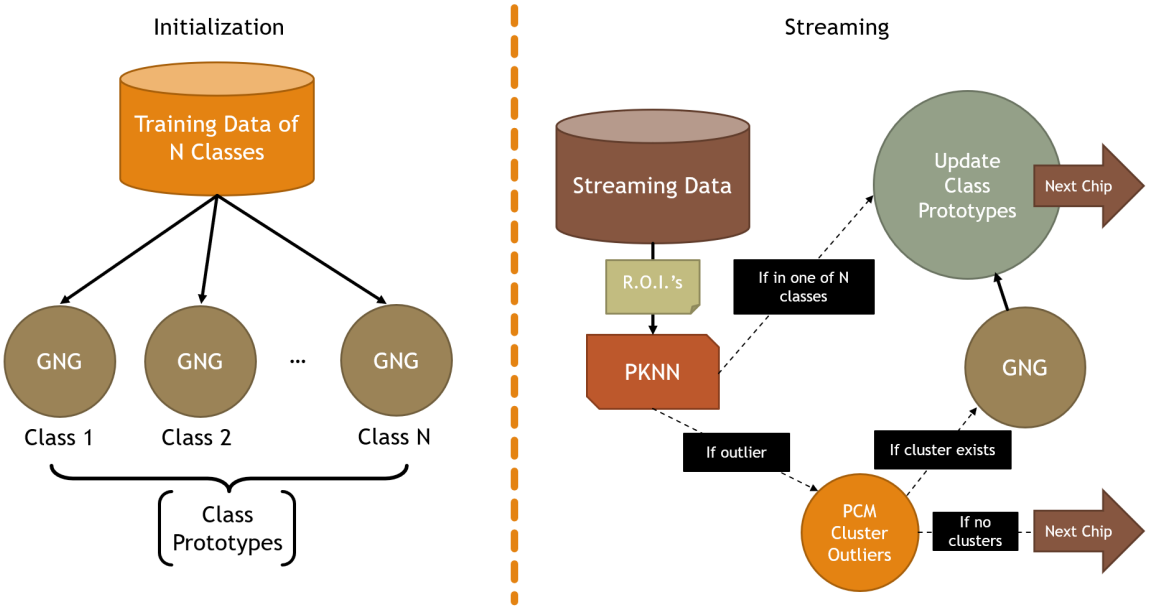


Figure 3.3: StreamSoNG flowchart. During initialization, Growing Neural Gas (GNG) networks are trained on each known data cluster and prototypes are saved. During streaming, data is introduced to the system one chip at a time. Herein, chips are image space regions of interest (R.O.I.) identified by a 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.

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 3.3.

In this paper, PCM is initialized to find one cluster and is simplified for StreamSoNG for sequential computation. As per Wu, et al [12], 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. [12].

3.2.2 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 3.4. 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 3.5 shows the proposed UI.

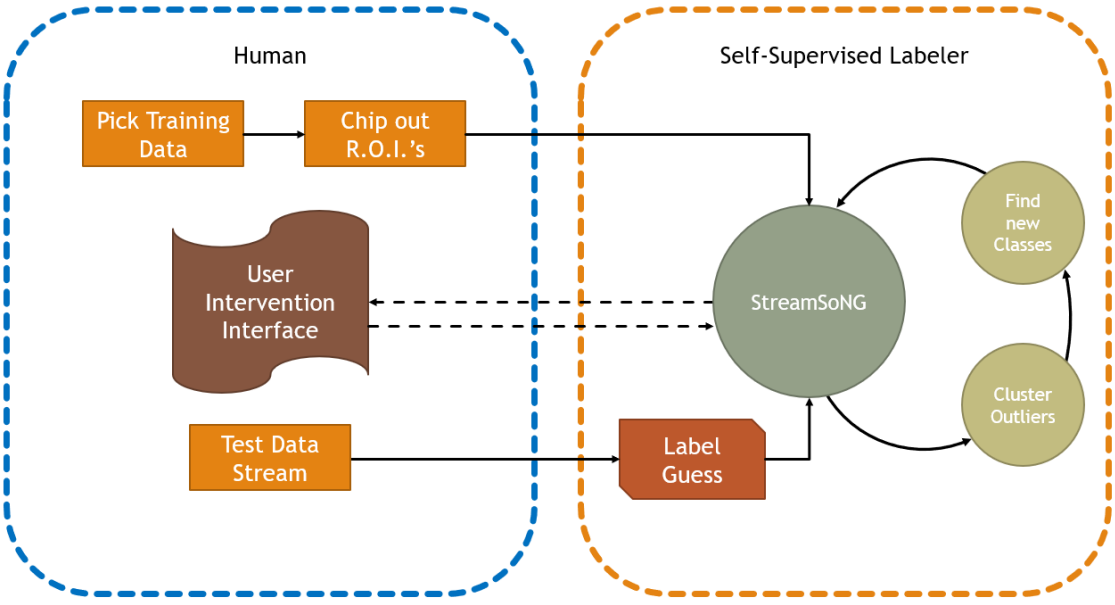


Figure 3.4: This diagram shows the relationship between the human operator and the self-supervised labeling algorithm as proposed in this paper. After providing the necessary initialization data, the human operator takes a back seat and monitors the self-supervised labeler. As the labeler streams data, we propose that it is possible to interrupt the data stream to correct wrong labels and preempt new class creation.

Figure 3.5 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.

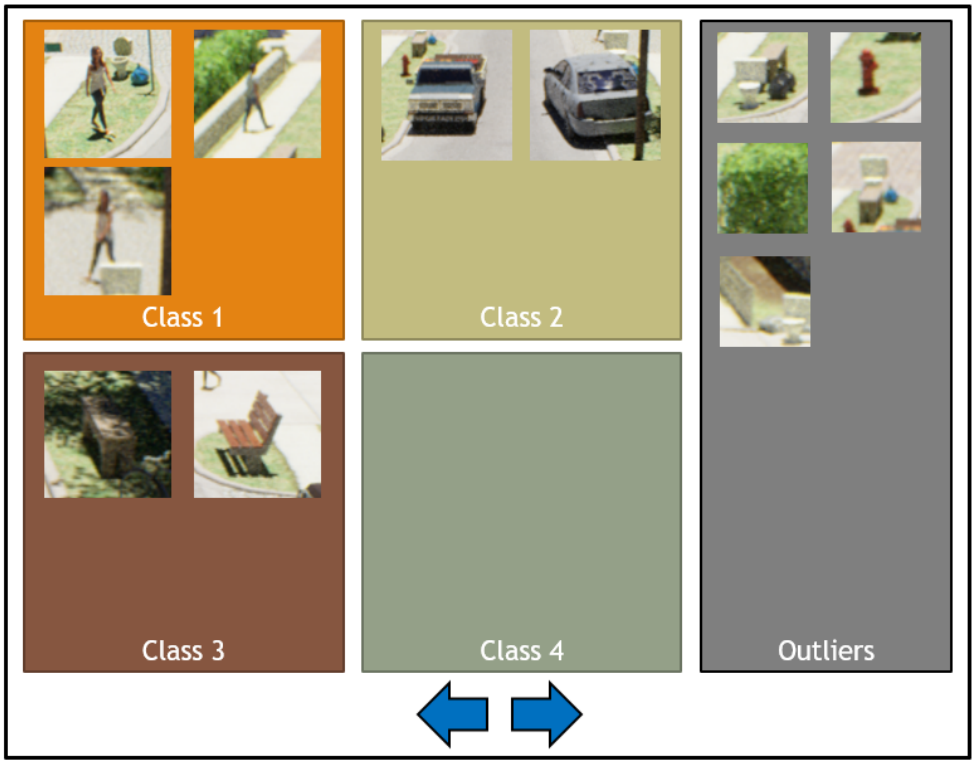


Figure 3.5: Prototype of the user interface explored herein. See the text for a full description.

Factors like the above led us away from having a real-time cluttered video feed that requires a high cognitive investment, e.g., Figure 4.1. Instead, we run an existing detection and localization algorithm to find candidate regions of interest (R.O.I.).

Specifically, we run YOLOv5[22]. 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 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[23] to modern change detection in deep learning[24, 25]. 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[26, 27, 28] and lower level algorithms like optical flow (e.g., FlowNet[29]) 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 3.5 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 3.5 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.

3.3 USE CASES

Figure 3.6: Simulated Modular Neighborhood Pack[30] environment on the Unreal Marketplace[31] for the Unreal Engine[32]. 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 3.3.1 for additional details.

3.3.1 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[32] 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[31]. Figure 3.6 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[33] Unreal Editor plug in can be used, and Ref. [34] 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[35], meta data enabled contextual fusion[36], or simulated augmentation data for explosive hazard detection[37] 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[30] 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)[38]. 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.

3.3.2 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 3.7 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 3.8. 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 3.8, 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 de-

tected 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.

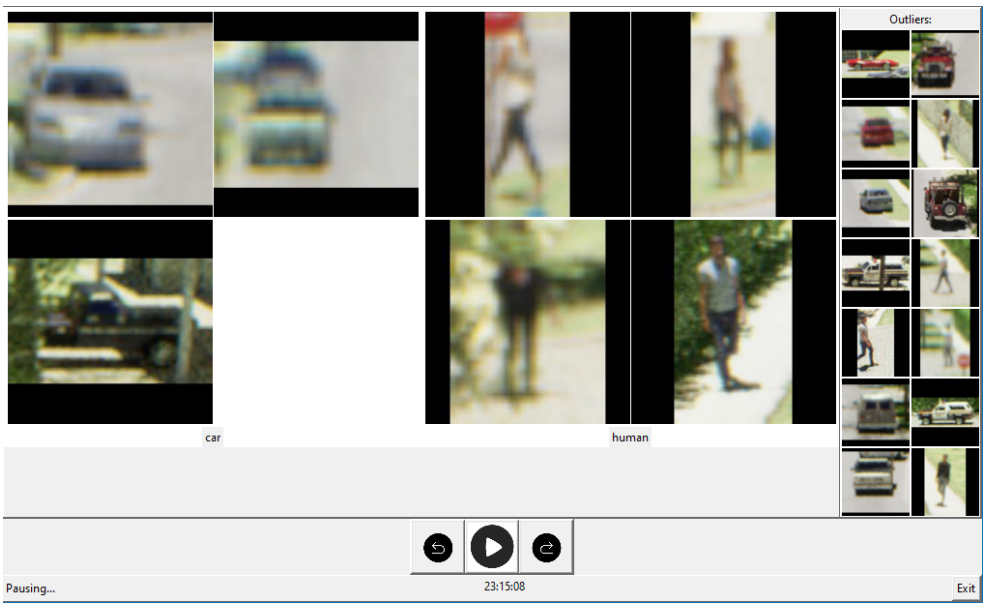


Figure 3.7: Snapshot of the HITL interface performing stream classification on incoming simulated data. Cars and humans that StreamSoNG has confidently classified populate in their respective regions. All chips that StreamSoNG determines are outliers populate in the right column.

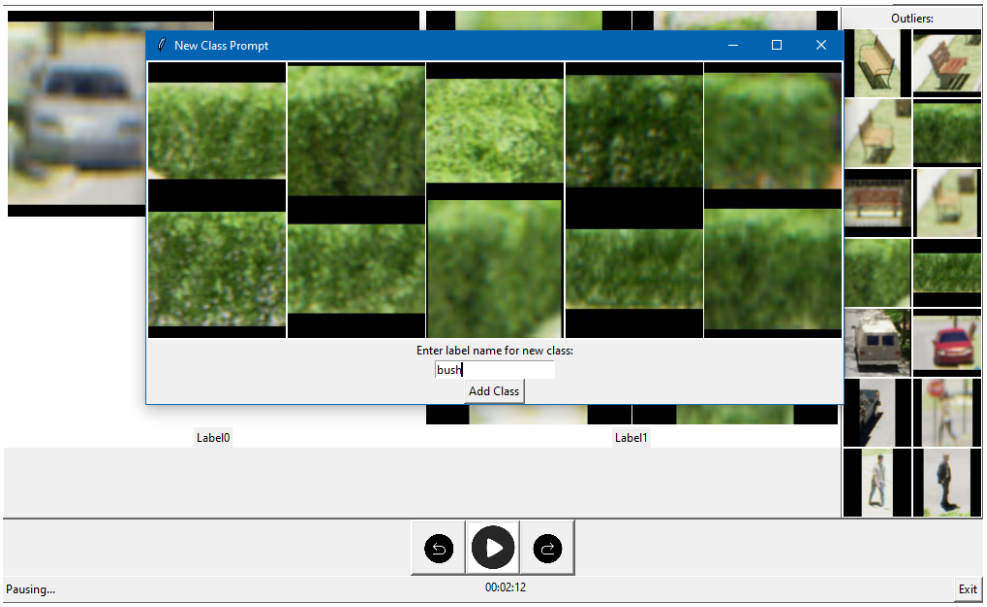


Figure 3.8: When StreamSoNG determines a new pattern is present in the data, it interrupts the streaming data and presents this dialog box for the user. The user selects similar images, or simply accepts all, and provides a label for the emergent pattern. Once streaming resumes the system now classifies images using this new knowledge.

3.3.3 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 3.9 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.

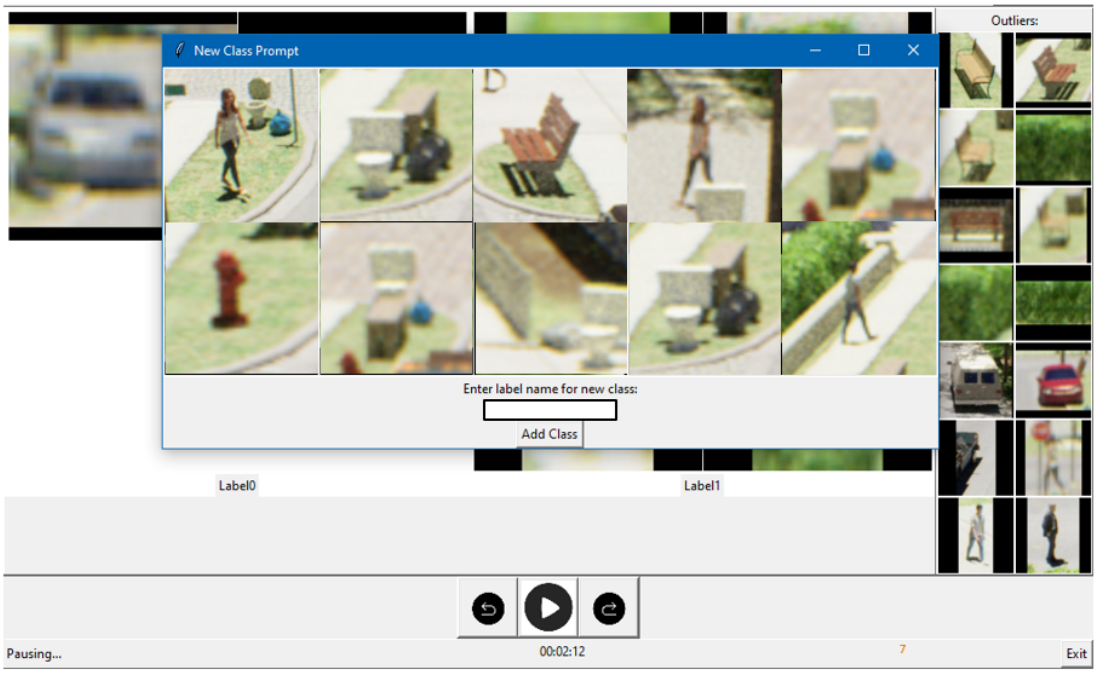


Figure 3.9: If a user chooses to preempt the creation of a new class, or extend an existing one, this dialog box pops up showing the target image similar imagery. The user then selects chips and they provide a label. In this figure, the user noticed a class of toilets being thrown away. They clicked 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.

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 example, consider Figure 3.9. 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.

3.3.4 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 3.10 shows an example.

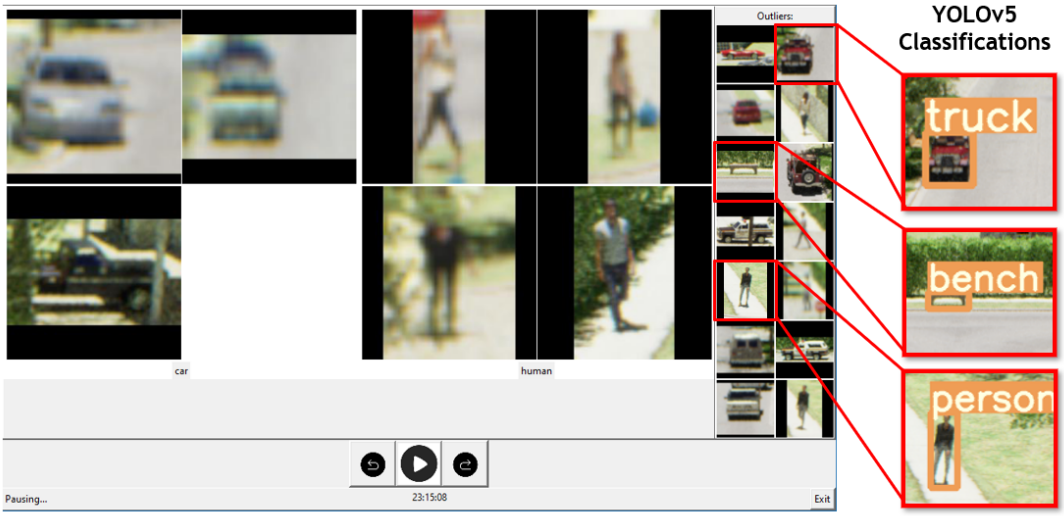


Figure 3.10: Example of image chips that StreamSoNG determined were outliers but YOLOv5 correctly classified.

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.

3.4 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.

Chapter 4

HUMAN-IN-THE-LOOP AUGMENTED GROWING NEURAL GAS FOR LABELING AERIAL IMAGE DATASETS

Jeffrey Schulz[a], Derek T. Anderson[a], James M. Keller[a],
Grant J. Scott[a], 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

There is a severe demand for, and shortage of, large accurately labeled datasets to train supervised computational intelligence (CI) algorithms in domains like unmanned aerial systems (UAS) and autonomous vehicles. This has hindered our ability to develop and deploy various computer vision algorithms in/across environments and niche domains for tasks like detection, localization, and tracking. Herein, we propose a new human-in-the-loop (HITL) based growing neural gas (GNG) algorithm to minimize human intervention during labeling large UAS data collections over a shared geospatial area. Specifically, we address human driven events like new class identification and mistake correction. We also address algorithm-centric operations like new pattern discovery and self-supervised labeling. The effectiveness of our algorithm is demonstrated using simulated realistic ray-traced low altitude UAS data from the Unreal Engine. Our results show that it is possible to increase speed and reduce mental fatigue over hand labeling large image datasets.

Keywords: active learning, self-supervised learning, human-in-the-loop, unmanned aerial vehicle.

Figure 4.1: High-level overview of SSHING. In our aerial application, change detection is used to identify regions of interest in imagery. SSHING attempts to classify ROIs based on machine learned DNN features. SSHING operates in a self-supervised fashion to identify new classes and evolve its knowledge. SSHING also makes use of a human-in-the-loop to accelerate learning, refine its knowledge, and/or answer questions it cannot otherwise solve.

4.1 INTRODUCTION

In today's Artificial Intelligence (AI) and Machine Learning (ML), Deep Learning (DL) is the reigning king. Some flavors of DL, e.g., Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), typically require large amounts of supervised data for learning. ImageNet, for example, is trained on approximately 14 million images. However, not every application has the luxury of millions of labeled images. Labeling requires a large amount of labor, i.e., time and cost. As a result, numerous companies have emerged and raised tens of billions of dollars to label ML data [1]. Herein, we explore the combination of self-supervision and human assisted labeling (Figure 4.1) to improve machine intelligence and reduce the expense of human labeling.

Modern DL does not identify classes it was not trained on. This is captured in Open-Set Recognition (OSR)[3]. OSR formalizes Known-Knowns (KK), Known-Unkowns (KU), Unknown-Knowns (UK), and Unknown-Unknowns (UU). Most DL operates on a Closed-Set (CS) premises and data sets have just KKs and KUs; e.g., combination of labeled and unlabeled imagery. A problem is, with the introduction of new data and classes, DL models can have undefined behavior. Herein, we are interested in online learning and approaches that can evolve their understanding of KK, KU, UK, and UU. However, there are fundamental limits with respect to what a machine can learn on its own. As a result, we focus on Human-in-the-Loop (HITL) enhanced online learning.

We are not the first to investigate online learning, HITL, and OSR to train Computational Intelligence (CI) algorithms or label data. Many real-world applications are impacted by these needs, e.g., autonomous driving, drones, security and defense, healthcare, and medicine [8] [9] [10]. For example, Telsa's data-labeling approach uses a HITL to improve the accuracy of their autonomous labeling algorithms. In a 2021 talk, Andrej Karpathy describes Tesla's self-supervised labeling system and reinforces the importance and need of human intervention in large-scale autonomous data labeling [11].

Herein, we introduce a **S**elf-**S**upervised and **HI**tl growing **N**eural **G**as (SSHING) algorithm for labeling image datasets. While we address algorithm-centric operations like new pattern discovery and self-supervised labeling, our primary contribution is integration of human events like new class identification and mistake correction. The SSHING algorithm is build on top of well-established CI methods like the Possibilistic K-Nearest Neighbor (PKNN) and Growing Neural Gas (GNG).

4.2 APPLICATION DOMAIN

The focus of this article is SSHING, not unmanned vehicles nor computer vision. However, for context, motivation, and sake of completeness, we summarize our application and data. Our problem consists of labeling, for training and evaluation, aerial data relative to object detection in support of human-robot teaming for augmented reality in collaborative spaces. The objects we need to detect cannot be found in publicly available data sets due to object type and/or context. Our data is based on a geospatial location that we have previously 3D mapped. New video, i.e., site re-visitations, are registered and change detection in point cloud or voxel space is performed. 3D change is then back-projected into image space, thresholded, mathematical morphology is applied for filtering, and the connected components algorithm is used to identify ROIs in image space (see [13] for details). Each ROI is fed to a Deep Neural Network (DNN) that was trained on a large and diverse collection of unrelated imagery and classes. SSHING takes as input the DNN feature embedding for ROIs. Thus, each ROI is transformed from an image chip into a set of machine learned features. To re-iterate, we cannot train a new DNN because there is little-to-no labeled data for our niche domain. Also, our KK-UU sets are changing with respect to new data collections and initially unintended system uses. Herein, we focus on labeling previously collected aerial datasets. If chips were constantly fed to SSHING in real-time during flight, then SSHING would be an online streaming algorithm. However, streaming is not our concern herein as our application does not require it. Our goal is to reduce the human time and cost required to label data in not real-time.

4.3 METHODOLOGY

SSHING, outlined in Figure 4.1 and Algorithm 1, is motivated by StreamSoNG [12], a fuzzy neural network-based streaming classification algorithm. The StreamSoNG

algorithm is a combination of clustering (unsupervised learning) and classification. It can identify new patterns, via clustering, and update its knowledge, realized via neural gas, to account for situations like concept drift. In [12], Wu et al. states that StreamSoNG is designed for large streaming data, and as a result the algorithm cannot store raw data. A difference between StreamSoNG and this article is we do not throw away data. The data and eventual labels are needed to train and/or evaluate CI algorithms. StreamSoNG is also fundamentally limited in what it can achieve as it only exploits data, there is no HITL to guide and correct learning. Next, we outline SSHING, a new online HITL-based GNG algorithm.

```
Algorithm 1: SSHING
-----------------------------------------
0. Initialization
     U is the set of unlabeled images
     K = ∅ (labeled data)
     O = ∅ (outlier set)
1. DO UNTIL user is done labeling
2.   Select image I_k from U and U = U \ I_k
3.   FOR EACH ROI (r_i) in I_k
4.     User generates a label (l_i) for r_i
5.     K = K ∪ {(r_i, l_i)}
6. Initialize GNG on K (see Sec. 4.3.2)
7. Initialize PKNN η values (Sec. 4.3.3)
-----------------------------------------
8. DO UNTIL U = ∅
9.   Sample an image I_k from U
10.   FOR EACH ROI (r_i) in I_k
11.    PKNN(r_i) (Sec. 4.3.3)
12.    If known PKNN class
13.      K = K ∪ {(r_i, l_i)}
14.    Else
15.      O = O ∪ {r_i} (add to outlier set)
16.      Run SP1M on O
17.      If SP1M identifies a cluster
18.        Prompt user for class label
19.        Add cluster to K
20.        Remove cluster from O
21.    Update GNG on K (see Alg. 4.3.2)
22.  IF user provides interaction
23.    DO UNTIL DONE
24.      IF user adds a class (Sec. 4.3.4)
25.        Retrieve neighbors (N) from
26.        {K ∪ O} per Sec. (4.3.4)
28.      IF mistake observed (Sec. 4.3.5)
29.        Retrieve neighbors (N) from
30.        {K ∪ O} per Sec. (4.3.5)
```

4.3.1 SSHING: Initialization

Of all the algorithmic decisions to be made, initialization might be the most contentious. We chose the following due to a few factors, namely a need to label chipped ROIs from UAS captured image data and the goal to reduce human intervention. To initialize the self-supervised labeling, first the knowledge needs to be seeded. The most qualified available *agent* is the human. The first steps involve manual labeling.

SSHING predictions are still made behind the scene and shown to the operator as a suggestion, allowing the human to understand the progression of the knowledge base and decide when to allow the self-supervised system to take over.

Our data collection pipeline outputs ROIs chipped from images that have been feature encoded by a CNN. These vectorized chips populate Euclidean space and are represented at the classification layer by GNG neurons. As patterns in high dimensional Euclidean spaces are known to be sparse and problematic, e.g., difficult to cluster and computationally intensive, we classify the set of GNG neurons rather than the entire known data set. Once the GNG has organized and reached a target max neuron error threshold (γ^*), we initialize unique bandwidth (η) values for each neuron (see sec. 4.3.3).

After initialization, there are three exclusive data set partitions; known class data (K), known-unknown class data (O), and existing unknown-unknown class (not yet seen) data (U). Set K comprises all known data from known classes that have been actively labeled. Set O, our current outlier set, are data that we know are unknown. Classes can emerge from O as SSHING progresses and an operator decides to label.

The big question is, how long should the user manually label until the self-supervised labeler takes over? SSHING gives the user the choice of when to activate the self-supervised portion of the system, as the user has the best idea of when the data set has been thoroughly sampled.

4.3.2 SSHING: Growing Neural Gas

GNG is an unsupervised incremental network that learns data set topology [14]. GNG is used to represent SSHINGs knowledge due to its lack of rigidity and its ability to be tuned for target performance. In this section we discuss our implementation of GNG for self-supervised labeling.

Initialization

We start by running conventional GNG [14] until a desired target error is reached. In our implementation, the target is when the error of all the neurons are below a user defined threshold, γ^*. This threshold can be tuned to directly influence how many neurons the network should produce.

GNG Update

Difficulties in our aerial data set problem, which is collected over a span of hours or even days, are natural inner class variation and concept or class drift. Examples are different visual appearances of objects caused by changes in time of day, environmental conditions, and the objects themselves (deformation, materials, etc.). SSHING needs to account for such factors so it can identifying the same objects from different viewing conditions and contexts.

Our GNG update loop includes the evaluation of the error of neurons in the system, addition of neurons to reduce error, and deletion of old neurons. Our GNG implementation is a slight variant of Fritzke's original algorithm [14]. Our change only impacts Friztke's Step 8 (see Algorithm 2). Instead of creating new neurons every γ iterations, we create new neurons based on a user defined error threshold (γ^*). This threshold is an essential addition for continuous iteration as it enforces the network to maintain ideal topology for a state of the knowledge regardless of number of updates. Without a hard cap on number of iterations to create an ideal map nor the number of neurons that can represent a class at any time in the algorithm, our modified GNG can stay accurate to any moment in the iterative algorithm process.

Algorithm 2: GNG Update Algorithm

Step 8. If a neuron has an error greater than γ^*, insert a new unit:

a. - Determine unit q with maximum accumulated error.

b. - Insert a new unit, r, halfway between q and its neighbor, f, with largest error:

$$w_r = 0.5(w_q + w_f).$$

c. - Insert edges connecting unit r with units q and f, and remove the original edge between q and f.

d. - Decrease the error variables of q and f by multiplying them with a constant α. Initialize the error variable of r with the new value of the error variable of q.

4.3.3 SSHING: Possibilistic K-Nearest Neighbor

Now that we have established the organization of points in space, we need to determine a method to classify new points. We use the Possibilistic K-Nearest Neighbor[5] (PKNN) algorithm. As opposed to the original K-Nearest Neighbor (KNN) algorithm, PKNN allows us to organize our data in an open-set friendly respect. In addition to allowing a null classification, PKNN produces a set of typicalities to each of the known classes. This can further provide information to the algorithm for potentially confusing classifications, which we leverage in section 4.3.5.

Classification

In the PKNN algorithm, a bandwidth parameter η determines the *cut-off distance*. If a query point is not within the bandwidth of any labeled points, it is unclassified.

For reference, the PKNN classification metric is

$$t'_{ik}(\mathbf{x}_t, \mathbf{p}_{ik}) = \frac{1}{1 + [\max(0, \|\mathbf{x}_t - \mathbf{p}_{ik}\| - \eta_{ik})]^{2/(m-1)}} \tag{4.1}$$

where the L2-normalized distance between sample x_t, at time t, and each prototype, p_{ik}, for each neuron (k) of each class (i) is thresholded by the unique bandwidth parameter η_{ik} to produce a typicality value, t'. Typicalities of sample x_t to each class fall off for distances larger than η_{ik}, eventually reaching zero. This mechanic, when the sample typicality to every known class is zero, is what allows null classification, which we use to define "outliers."

Initialization

PKNN η values provide a threshold for possibilistic classification. The initialization is important in dividing up the classification space in our training set. While other approaches to initializing η exist [39], we choose to utilize knowledge already in the system to compute bandwidths. To set unique bandwidths for all neurons, we use the GNG neuron error, which already holds information unique to each neuron relating to how well it represents nearby data in both K and O. This initialization is as follows,

$$\eta_{ik} = \omega * \epsilon_{ik}, \tag{4.2}$$

where bandwidths for each prototype k for each class i are computed as a multiple ω of the GNG neuron errors ϵ for each prototype k for each class i.

Upon unique initialization of the PKNN η bandwidth parameter for each neuron, query points can compute a typicality to neurons using the PKNN metric in Equation 4.1. Over time, neurons without classification can self-organize into emergent classes, discussed next.

4.3.4 SSHING: Human Identifies a New Class

Adding a human element to an online algorithm can increase its labeling speed and accuracy. First, we address the case of when the human identifies a new class. Specifically, we are concerned about two scenarios. The first scenario is when the operator manually identifies a new class of interest that they want to preemptively label, versus let SSHING try to eventually discover and recommend. The second case is when the human notices potential problematic non-target classes that could lead to classifier confusion.

Recall, one of our main objectives is to reduce the human workload. To this end, we try to reduce the number of overall human events by increasing the amount of interaction during an event. ROIs are retrieved from un-labeled regions via the nearest neighbor algorithm. In practice, a user interface may not be able to display all chips (retrievals). Therefore, chips are sampled to rapidly expand the reach of a new class definition. This feature is especially useful in quickly defining new class bounds. When the user feels the class has been adequately expanded, they terminate the interaction.

4.3.5 SSHING: Human Corrects a Labeling Mistake

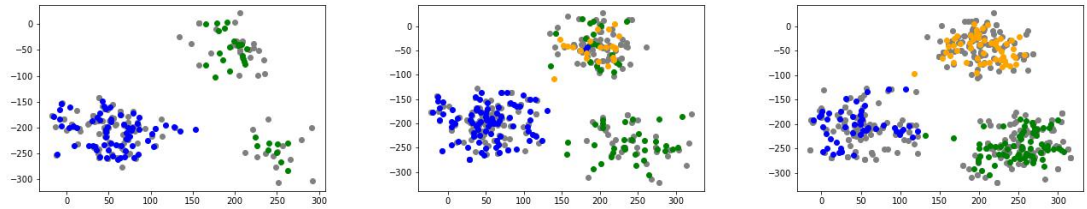
The second scenario is when the human notices a SSHING mistake. Similar to Section 4.3.4, the idea is to take advantage of the fact that the user interrupted SSHING and retrieve related chips. In this interaction, we are concerned with only when the human operator notices misclassified KKs.

The idea is to use human interaction to refine class boundaries around mistakes. Similar to Section 4.3.4, SSHING displays a set of chips sampled from a larger retrieval and the human can choose to label as many chips from that set as they want. This retrieval method is as follows. Starting with a nearest neighbor retrieval of K chips, calculate the PKNN typicalities of each neighbor to each class. To score the set of

(a) First few iterations after manual labeling

(b) New class dialog to define "Class 2"

(c) System knowledge after many iterations



(d) After few iterations after manual labeling

(e) After adding orange cluster as "Class 2"

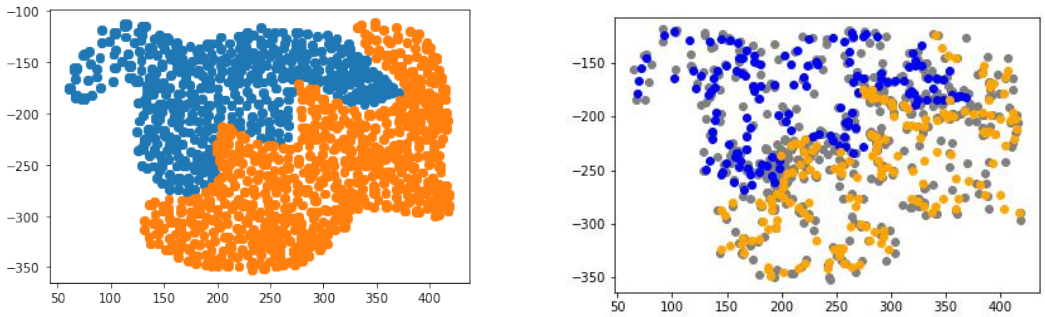
(f) After many self-supervised iterations

Figure 4.2: Example 1. Figure (a) shows the state of SSHING for a single defined class (blue). Aided by the extended period of manual labeling (e.g., subfigure (b)), new classes are quickly established. Subfigure (c) shows the state of SSHING after the 2nd class label has fully claimed the space occupied by the orange class. Row two shows data (gray) and the neural gas memory (colored points) in feature space.

typicalities for each neighbor by most confusing to least confusing, a small amount is added to each typicality score, e.g., .01. Next, we multiply the typicalities for each neighbor to each class. This results in a value that is larger for neighbors with typicalities to many classes; and are therefore more confusing. The scores are then sorted and displayed to the user with respect to their confusion. The objective of this retrieval is to refine the decision boundaries between classes, quickly setting new prototypes in the confusion area.

4.4 EXAMPLES

In this section, we begin with a controlled synthetic and easy to visualize data set. Next, the Unreal Engine[32] is used to generate a more challenging yet fully ground truthed low-altitude aerial image data set. The UE environment consists of free and



(a) 2D data set with non-linear boundaries

(b) Final SSHING results

Figure 4.3: SSHING for Experiment 2 with more complex non-linear boundaries.

paid content from the Unreal Marketplace[31].

The first simple 2D data set is synthesized using Numpy in Python[40] with the goal of creating poignant problem samples to demonstrate the behavior of SSHING. The data set in Section 4.4.1 is generated from three separable normal distributions. The data in Section 4.4.2, which has more complex non-linear decision classes, was produced by hand in Microsoft Paint.

The simulated aerial data generated in UE requires processing for ROI identification before it is processed by SSHING, see Section 4.2 and [13]. UE was used due to its photorealism and since it is often used to help train ML/AI algorithms. UE is also idea for rapid testing the concepts presented herein as we know the ground truth and its simple to change the scene, e.g., add or remove objects, change properties, vary the time of day, repeat an experiment, etc.

Herein, we focus on the speed with which a human assists labeling and the degree to which we have lowered their mental load. Namely, we track the following metrics. Metric 1 is the number of images manually and automatically labeled; showing the balance of assisted labeling. Metric 2 is the number of clicks; which tracks the physical labeling act. Metric 3 is completion time; which measures labeling efficiency.

4.4.1 Example 1: User Adds New Class

In Example 1, we examine the behavior of the labeler when a new class is identified in a simplified 2D data set with 3 separable clusters. SSHING is first initialized with a period of manual labeling. Specifically, a human manually labeled 50 images consisting of the known (blue) and unknown (green) classes. Neural gas elements are shown color coded in feature space in Figure 4.2. However, colors were slightly perturbed in the user interface to help the user easily determine when a new sample is streamed in. Otherwise, it is challenging to tell when something is updated.

Next, the orange class is introduced. In this instance, the operator is now advancing the labeler from image to image without having to pick labels (see Figure 4.2a), as the system has enough knowledge to correctly label Class 1 (blue) and unknown examples (green) in this easily-separable toy data set. When the orange class is introduced, corresponding data points start to populate in the user interface outlier section. The human sees this data and makes the decision to create a new class, prompting the system to retrieve nearby samples (see Section 4.3.4). These data points are then displayed to the user and it prompts a period of manual labeling (Figure 4.2b). As the user labels, SSHING continues to pull in un-labeled data according to the retrieval method. When the human is finished labeling, they close the labeling prompt and continue advancing images (Figure 4.2c). Figures 4.2d, 4.2e, and 4.2f all show the state of SSHINGs knowledge at each interval in the labeling process respective to the figures above them.

We asked a user to label the same data set ($N = 500$) twice: once with only manual labeling for each data point, and once with respect to SSHING and 50 manually labeled points. The results are shown in Table 4.1.

We discovered (Table 4.1) that SSHING could label Experiment 1 629 seconds faster than a human with no assistance. This was due primarily to the fact that SSHING correctly classified 379 labels. Granted, Experiment 1 is overly simple and this is

Evaluation Measure	Just Human	SSHING
Images Manually Labeled	500	121
Images Automatically Labeled	0	379
Number of Clicks	1000	561
Completion Time (seconds)	770	141

Table 4.1: Experimental results of human manually labeling versus SSHING for Experiment 1.

a good indicator of an upper bound on performance. Experiment 1 is a sanity check and opportunity to ensure that SSHING is operating as advertised. The algorithm automatic labeling also contributed to a decrease in user number of clicks from 1000 (one click to choose label, one click to advance to next image) to 561, with the user only having to make 61 additional clicks aside from advancing each image.

To clarify a possible point of confusion, manual labels include the initialization labels as well as any labeling when defining a new class or correcting a mistake by the algorithm. Automatically labeled images include only images that are labeled by the self-supervised labeling mechanism and allowed to pass through the user interface without intervention.

4.4.2 Example 2: User Identifies a Labeling Mistake

In Example 2, we explore the SSHING scenario of the user identifying a mistake. As in Example 1, SSHING is initialized with 50 iterations of manual human labeling before turning on the self-supervised labeling mechanism. Figure 4.3a shows the 2D non-linear data set, which consists of 2 classes with more complex and potentially confusing class boundaries.

When the user identifies a labeling mistake, SSHING retrieves nearby un-labeled data points with high confusion (see Section 4.3.5) and a subset are displayed in the interface. The human labels as many data points as they see sufficient. By retrieving

the data this way and prompting the user for an extending labeling period, SSHING can quickly learn the space around the misclassification. Figure 4.3b shows a snapshot of SSHINGs knowledge after a *good amount* of self-supervised iterations; meaning SSHING is not in need of much correction, it is correctly classifying most incoming data points. As the reader can see, the incoming data points (gray) are heavily concentrated in and around the class boundary, as our retrieval prioritizes those points. As such, SSHING has concentrated the GNG neurons (blue and orange) in that space to better represent class boundary. As performance results are extremely similar to Example 1, which is encouraging, we omit them. The take away from Experiment 2 is that SSHING is properly focusing on mistakes around class boundaries and human intervention is helping SSHING to adapt and grow.

4.4.3 Example 3: SSHING on Simulated UE Aerial Imagery

Now that we have verified two of the main aspects driving efficient HITL-based self-supervised labeling, we demonstrate SSHING on a harder real-world problem. Labeling objects of interest in real aerial imagery can be time intensive and complex, e.g., what bounding box to assign, when/how to label in light of resolution, occlusion, etc. SSHING can help us address some of these challenges in a quicker and less mentally taxing affair. With respect to tracking an increasing number of labels (new classes), SSHING allows labeling outliers, giving the user the opportunity to label things that are important and unique to each run/collection. SSHING also allows the user to decide when they want to begin labeling; though the longer a user waits the more damage is done and corrections are needed.

In Example 3, we asked the user to label aerial region of interest chips, see Figure 4.4. As before, the human was asked to complete two runs of the data set, one entirely manual (no SSHING) and one pass SSHING assisted. The results are tabulated in Table 4.2.

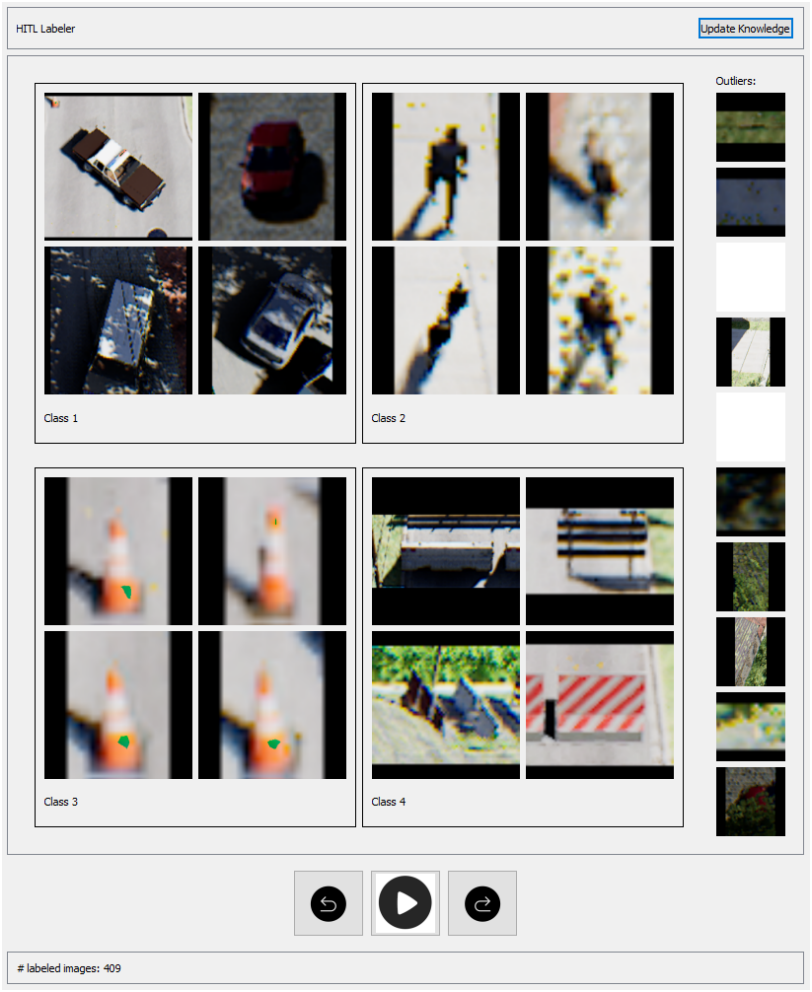


Figure 4.4: Screenshot of SSHING labeling simulated aerial imagery. Known classes in this example include cars, humans, traffic cones, and road barricades. Outliers (KUs) include trees, bushes, benches, trash cans, and furniture.

Evaluation Measure	Just Human	SSHING
Images Manually Labeled	500	258
Images Automatically Labeled	0	242
Number of Clicks	1000	655
Completion Time (s)	870	617

Table 4.2: Experimental results of human manually labeling versus SSHING on a 500 chip data set of simulated drone imagery (Experiment 3).

In a stark difference to Experiment 1 and 2, fewer ROIs (image chips) are automatically labeled and the SSHING-assisted run resulted in only a 29% performance in completing the data set. As a side note, the last 100 images labeled were 60 automatic and 40 manual. At this moment in the labeling process, the user has the knowledge necessary to start to quickly labeling the data set. Labeled data consisted of 79 cars, 31 humans, 38 construction cones, 19 barricades, and 333 outliers. The point being, outliers made up the significant portion of the automatically labeled data, forcing the user to manually label most of the target classes in the data.

4.5 DISCUSSION AND FUTURE WORK

In Example 1 and 2 we see that labeling separable patterns in a simple 2D data set is trivial for SSHING, or any HITL assisted or streaming algorithm at that. After no time at all, SSHING runs almost entirely untethered. On the other hand, labeling real-world higher dimensional image data struggles to achieve the same lofty expectations. Why? Is this a flaw in SSHING, an artifact of our experiment and data set, etc.?

Of the few differences in performance across experiments is the time taken to label Experiment 3 stands out as only slightly improved over manual labeling. A significant portion of this time is the mental ability to quickly identify the content of the image chip. Assisted self-supervised labeling serves to improve, at most, the amount of time necessary for the user to click a label. The time needed to click a

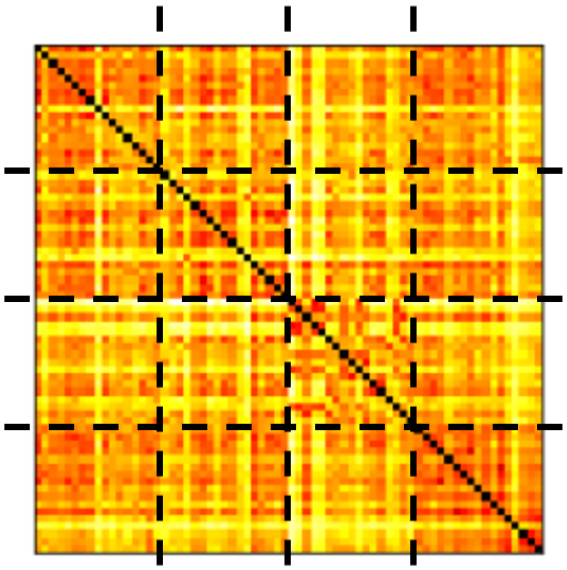


Figure 4.5: Dissimilarity matrix for Example 3 data, showing the differences of 20 instances from each class (car, human, cone, and barricade). Instances are organized by class, i.e., samples 1 to 20 are cars, 21 to 40 are humans, etc. In this plot, darker means more similar and the matrix is normalized between min and max distance for display. This matrix shows that it is difficult for SSHING to distinguish between classes, as indicated by the lack of organized darker shades in the off-diagonal sub-matrices. The ideal dissimilarity matrix has zeros in (class i, class i) and ones in (class i, class j), $i \neq j$.

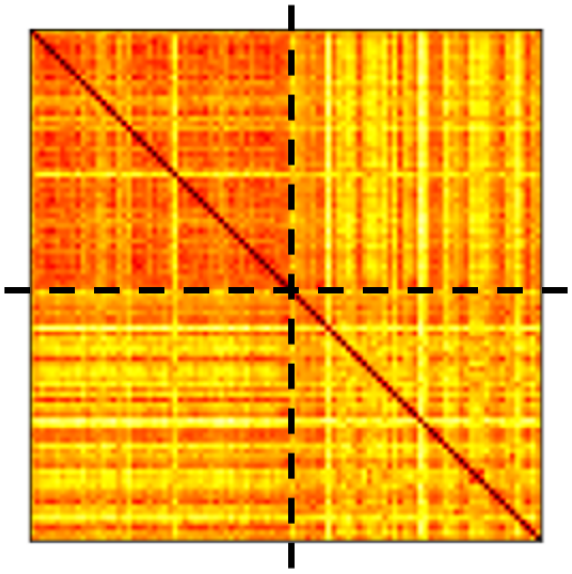


Figure 4.6: Dissimilarity matrix between 50 outliers (first 50 indices) and 50 cars (last 50 indices). This shows that outliers in Experiment 3 have higher intra-class similarity than cars while maintaining low inter-class similarity. This reaffirms that SSHING's strong suit on Example 3 is labeling outliers.

label is far less impactful than the time needed to identify possibly obstructed objects in low-resolution imagery. As it turned out, most automatically classified ROIs were outliers. While this may not be novel by itself, it does contribute to reducing the labeling load on the user. An approach to assist the user in labeling could be to make available multiple cropping of ROI chips to the user. With multiple levels of cropping on target (aka scales), the operator could potentially more quickly establish context of the ROI and identify the target of the chip.

In future work, we will also explore more intelligent methods of encoding our image ROIs. Our current method consists of using pretrained deep neural network features from benchmark computer vision data sets reduced down using an autoencoder to a lower dimensional feature space. A problem with this is it does not express our aerial near nadir imagery well; both in relative pose, environment, and types of objects we wish to classify. At the end of the day, our current approach, while community accepted, is not as separable as needed, which results in more SSHING misclassifications and ultimately less efficiency. These claims are backed by manual analysis of class examples in the context of feature dissimilarity matrices, see Figures 4.5 and 4.6. In particular, we see low inter-class dissimilarity and low intra-class similarity in the encoded features. Furthermore, Figure 4.6 backs the claims of low similarity between outliers and one known class (cars). The point is, our aerial image data set SSHING performance is hindered significantly by our feature encoding.

Another consideration is that our initial experiments do not take into account variability with respect to the number of neural gas neurons used. One of our goals, versus working ith the entire data set, is to maintain a a user defined max or relatively small percentage of the total amount of data as it relates to classification and retrieval efficiency. In future work, we will explore human performance relative to minimizing the number of neurons and more intelligently distributing these neurons in the feature space, e.g., areas of potential confusion like class boundaries.

There are a number of additional future extensions that we will consider related to image processing and computer vision. For example, Experiment 3 was plagued by a significantly large number of not class of interest ROIs generated by our change detection algorithm. A next step will be to improve the change detection algorithm and incorporate domain knowledge to reduce this stream of ROIs. Clearly this will have a large impact on SSHING as it relates to how much data requires supervision. Furthermore, Experiment 3 was shown as a stand alone experiment. However, this is not the intended use of SSHING for aerial image labeling. Our goal is to reuse SSHING on consecutive runs in a same scene and for related environments. As such, SSHING will not start from tabula rasa. This consumes a great deal of human effort. In future work we will use SSHING to label consecutive runs and measure performance over longer periods of time. We are also interested in if there exists additional filters that can be exploited. For example, if the operator sees particular color or altitude or other contextual information about ROIs, is it possible to incorporate this information and abstract the problem to reduce the amount of constant visual inspection that is required. Last, a few measurements were used herein. In future work, we will explore more advanced user interface and cognitive measures and metrics to assess workload.

Chapter 5

DISCUSSION

Chapter 2 explores how to extend artificial neural networks to class N+1. After proposing a pipeline to input, encode, and predict on image data, the system is optimized using a genetic algorithm. After generating ordered dissimilarity matrices and performing automatic scoring based on squaredness in CLODD, the best pipeline is concluded to be ResNet101 $\rightarrow$ Max-pooling $\rightarrow$ PCA reduction. To simplify the pipeline complexity and increase consistency between data sets and runs, in the following works ResNet101 is substituted for its smaller sister model ResNet50 and PCA is substituted for the repeatable encoding method in a pre-trained Autoencoder. In this chapter, I learn mostly that PKNN is limited. I get around this in future works by testing and picking parameters to allow the best performance from PKNN, but the bane of the usage for PKNN in this chapter is the lack of well-defined context for the value assignment of the bandwidth parameter. This parameter, of course, being integral to the ability of the PKNN-based classifier to say "I don't know." Frigui and Gader, in their work [5], derive the parameter statistically. This did not work without amplification for encoded visual features, and rarely did the amplifying factor remain the same between contexts. In Chapter 2, the bandwidth parameter is defined with the help of a genetic algorithm, brute-forcing the performance of the PKNN classifier to find the optimal bandwidth. Without an explainable and derivable parameter for the bandwidth, it is tough to see PKNN at the core of an OSR classifier.

In Chapter 3, this dissertation outlines the first use of StreamSoNG on a streaming computer vision task. Along with StreamSoNG, I implement a HITL into the data-labeling online classification algorithm. By bootstrapping an established stream-classification algorithm in StreamSoNG, I show that a collection of algorithms can exist to perform quick and self-supervised classification with the assistance of a human operator. In this step, I also propose a graphical user interface (GUI) for intelligently interfacing the human operator with the self-supervised labeling system. In this chapter, I learn mostly that more work is required to achieve high quality labeling on real world streaming data. Among many small algorithmic changes, it became clear that StreamSoNG was not the right algorithm for my HITL labeling application. StreamSoNG, by design, does not store any of the streaming data. The application built here is a data labeler, which should store out every data point that it labels. At this point, it is unclear if a "teacher" algorithm should exist, in Chapter 3 I mention YOLOv5 as an option, to further help steer the online classifier in the correct direction in combination with the human expert. Lastly, it is important that whatever new algorithm is implemented, it must keep in mind concept drift. As UAS data can be collected at any time in the day and include many different variations of objects in the same class, drift in the data *will* occur and must be accounted for.

Chapter 4 proposes and demonstrates the **S**elf-**S**upervised and **H**uman-**I**n-the-loop Growing **N**eural **G**as (SSHING) for human-assisted self-supervised labeling of large data sets. SSHING demonstrates a polished version of the GUI previously interfaced with StreamSoNG and introduces two focal points for human assistance in online self-supervised image classification. This chapter demonstrates that SSHING reduces the labeling load on the human operator in real-data labeling applications. In this chapter, I primarily struggle with encoding image data in a way that features are separable when represented in low-dimensional space. My approach to human-assisted labeling is proven to be advantageous in separable low-dimensional toy data

sets, but SSHING can only work as well as the input features are separable. This chapter proposes the use of well-known methods of feature extraction for image data, such as using modified implementation of different deep CNNs, but exploration into alternative methods of image feature extraction is reserved for future works. Specifically with the SSHING user interface, the most labor-intensive responsibility of the human seems to be in the identification of low resolution imagery and highly obstructed looks on target. While attempting to use simulated imagery akin to that collected from a drone, both of these problem cases should be expected. One approach to improving the time to label tough images could be modifying the change detection and ROI extraction methods to include multiple intensities of cropping on the target to increase the human operator's ability to draw context from the scene. With a larger context to identify targets, the human operator may be able to label imagery quicker in the SSHING application. Finally, even though SSHING's performance on simulated UAS data didn't meet the same lofty performance expectations set in the toy 2-D data set examples, the system provides a noticeable improvement in mental load an labeling speed over manually labeling the same data set. In the simulated UAS data, there is a large amount of non-interesting change (tuned to emulate that of ROI extraction results from real data applications), which is largely automatically labeled by SSHING. SSHING struggles at target classification early in labeling, but it very quickly establishes between the set of target classes and non-target classes (outliers). Automatically labeling the outliers in a data set that consists 50% of outliers is marked improvement over manual labeling, especially when considering the light overhead to initializing SSHING.

Overall, reflecting on the dissertation as a whole, a couple things stand out. Firstly, I needed better features. At all points in this process, I was in need of feature encoding methods that produced separable vectors based on visual features. This problem is not unique by itself, of course, but I feel as if I was close to a passable

solution by using CNN encoded features. In those features exist the separable data I am seeking, I just did not have the bandwidth to find the right encoding method. After all, the focus was on the following, which I am proud of.

After iterating on HITL guidelines and user events for an entire paper (Chapter 3), I was proud to boil every possible event in SSHING down to two well-defined categories: user defines class and user corrects mistake. By finding two main actions that the user can take at any time in the usage of the application, I feel I am laying groundwork for progress on future HITL applications. With a starting philosophy of how to gauge human input on a HITL labeling application, I hope future works can spring board to achieving better results.

Chapter 6

FUTURE WORK

Future work on SSHING and surrounding algorithms include the following. Firstly, the PKNN method is useful for rejecting outliers with its bandwidth component but the mechanism (equation) needs improvement. Classification methods that are distance-based like PKNN suffer from a perfectly circular world-view, a structure in which clusters of target classes hardly manifest. A well-designed classifier for SSHING would likely adopt a topological strategy, molding a kernel to best fit to the known data instead of in circular sub-structures. In addition, a deeper and more thorough analysis is needed across existing architectures and models. Ideally, every part of SSHING from the change detection and ROI extraction to the online GNG and PKNN classification to the application should be interconnected and optimized. As this dissertation focuses on exploration within this topic, this thorough work has not been attempted.

Secondly, the change detection and ROI extraction methods have lots of room for improvement. Improving change detection in the data set can improve classification speeds by simply reducing the amount of data for which the human operator is responsible. At this time, the change detection and ROI extraction pipeline exists as a headless operation. In the future, this process could have a user-facing interface for real-time tuning of change detection parameters. As mentioned above, each step of the system is deeply interconnected. Improving the ROI extraction to include more

meaningful change will succeed in reducing the labeling load on the human operator.

For the time being, SSHING has not been asked to run on data sets with large amounts of concept drift. Large amounts, for example, might include exaggerated color and shadow differences in different hours of the day or different weather conditions. This may also include inner-class concept drift of like objects, including similar car bodies with different paint colors, or classes like people where each sample is unique. Future works surrounding SSHING should identify and test the performance under these varying types of concept drift in the data set. While my augmented GNG algorithm should account for and adapt to any amount of concept drift, the upper bound was not tested. Iterating SSHING on real data collected in the field should test the system's susceptibility to concept drift in classes.

Next, future work developing SSHING should look at better methods of encoding the image data for use in metric learning space. The current method is still largely inseparable, resulting in too many misclassifications by SSHING. While succeeding in defining a methodology for human-assisted labeling, SSHING is still limited by the lack of separable features in the encoded image data. With improvements in the encoding pipeline, either by more intelligent dimensional reduction or better trained CNNs with more separable raw features, I believe the application of simluated UAS can see similar performance improvements to manual labeling as the toy 2-D data set examples. While not being utilized in the same way as a fully-connected MLP classifier, I believe that the data necessary to separate the vectors exists in the encoded visual features.

Finally, the future of SSHING relies on real-time, lightweight, online operation. The idea of SSHING was born from the idea of running a system on-board the drone to identify and classify notable change in the environment and reporting that labeled change to the user. While the execution of SSHING pivoted to an iterative data-labeling application for large supervised learning models in its infancy, the goal

still remains of a simple, light, and powerful ROI classification backbone for drone imagery. This future iteration of SSHING should prove to be CNN and context independent, only requiring that features be separable in some space. Future works on SSHING should aim for lightweight deployment to enable execution on mobile platforms, enabling real-time human-assisted labeling in-flight during data collects. Not only does this enable the system to live on a drone and execute mid-flight, but it continues to enable uses of SSHING for labeling large offline data sets, just as we have explored in this dissertation.

BIBLIOGRAPHY

[1] J. Kahn. *If data is the new oil, these companies are the new Baker Hughes*. Feb. 2020. URL: https://fortune.com/2020/02/04/artificial-intelligence-data-labeling-labelbox/.

[2] J. C. Bezdek and J. M. Keller. "Streaming Data Analysis: Clustering or Classification?" In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2020), pp. 91–102.

[3] C. Geng, S. .-. Huang, and S. Chen. "Recent Advances in Open Set Recognition: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1.

[4] J. A. G. James M. Keller Michael R. Gray. *A fuzzy K-nearest neighbor algorithm*. USA, 1985. URL: https://ieeexplore.ieee.org/document/6313426.

[5] P. G. Hichem Frigui. *Detection and Discrimination of Land Mines in Ground-Penetrating Radar Based on Edge Histogram Descriptors and a Possibilistic K-Nearest Neighbor Classifier*. USA, 2008. URL: https://ieeexplore.ieee.org/document/4610973.

[6] Y.-C. Chou, C.-J. Kuo, T.-T. Chen, G.-J. Horng, M.-Y. Pai, M.-E. Wu, Y.-C. Lin, M.-H. Hung, W.-T. Su, Y.-C. Chen, et al. "Deep-learning-based defective bean inspection with GAN-structured automated labeled data augmentation in coffee industry". In: *Applied Sciences* 9.19 (2019), p. 4166.

[7] I. Shin, D.-J. Kim, J. W. Cho, S. Woo, K. Park, and I. S. Kweon. "Labor: Labeling only if required for domain adaptive semantic segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8588–8598.

[8] A. Holzinger. "Interactive machine learning for health informatics: when do we need the human-in-the-loop?" In: *Brain Informatics* 3.2 (2016), pp. 119–131.

[9] J. Wu, Z. Huang, C. Huang, Z. Hu, P. Hang, Y. Xing, and C. Lv. "Human-in-the-loop deep reinforcement learning with application to autonomous driving". In: *arXiv preprint arXiv:2104.07246* (2021).

[10] D. S. Nunes, P. Zhang, and J. S. Silva. "A survey on human-in-the-loop applications towards an internet of all". In: *IEEE Communications Surveys & Tutorials* 17.2 (2015), pp. 944–965.

[11] A. Karpathy. *[CVPR'21 wad] keynote - Andrej Karpathy, Tesla - YouTube*. URL: `https://www.youtube.com/watch?v=g6bOwQdCJrc`.

[12] W. Wu, J. M. Keller, J. Dale, and J. C. Bezdek. "StreamSoNG: A Soft Streaming Classification Approach". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2021).

[13] J. Schulz, A. Buck, D. T. Anderson, J. M. Keller, G. Scott, and R. H. Luke. "Human-in-the-loop extension to stream classification for labeling of low altitude drone imagery". In: *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2021*. Vol. 11748. International Society for Optics and Photonics. 2021, 117480J.

[14] B. Fritzke. "A growing neural gas network learns topologies". In: *Advances in neural information processing systems* 7 (1994).

[15] J. Schulz, A. Buck, D. T. Anderson, J. M. Keller, G. Scott, and R. H. Luke. "Human-Assisted Growing Neural Gas for Efficient Labeling of Aerial Image Datasets". 2022.

[16] A. H. Charu C. Aggarwal and D. A. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. Berlin, 2001. URL: https://bib.dbvis.de/uploadedFiles/155.pdf.

[17] M. Pagola, J. I. Forcen, E. Barrenechea, C. Lopez-Molina, and H. Bustince. "Use of OWA operators for feature aggregation in image classification". In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. July 2017, pp. 1–6.

[18] C. Dias, J. Bueno, E. Borges, S. Botelho, G. Dimuro, L. Giancarlo, J. Fernandez, H. Sola, and P. Drews-Jr. "Using the Choquet Integral in the Pooling Layer in Deep Learning Networks". In: July 2018, pp. 144–154. ISBN: 978-3-319-95311-3.

[19] S. R. Price, S. R. Price, and D. T. Anderson. "Introducing Fuzzy Layers for Deep Learning". In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. June 2019, pp. 1–6.

[20] L. Wang, U. T. V. Nguyen, J. C. Bezdek, C. Leckie, and K. Ramamohanarao. "iVAT and aVAT: Enhanced Visual Analysis for Cluster Tendency Assessment". In: *PAKDD*. 2010.

[21] e. a. Timothy C. Havens James C. Bezdek. *Clustering in Ordered Dissimilarity Data*. New Jersey, USA, 2009. URL: https://pages.mtu.edu/~thavens/papers/IJIS_2009_24_Havens.pdf.

[22] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, A. Chaurasia, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Durgesh, F. Ingham, Frederik, Guilhen, A. Colmagro, H. Ye, Jacobsolawetz, J. Poznanski, J.

Fang, J. Kim, K. Doan, and L. Y. *ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration*. Version v4.0. Jan. 2021. URL: https://doi.org/10.5281/zenodo.4418161.

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

[24] L. Khelifi and M. Mignotte. "Deep Learning for Change Detection in Remote Sensing Images: Comprehensive Review and Meta-Analysis". In: *IEEE Access* 8 (2020), pp. 126385–126400.

[25] A. Varghese, J. Gubbi, A. Ramaswamy, and P. Balamuralidhar. "ChangeNet: A Deep Learning Architecture for Visual Change Detection". In: *Computer Vision – ECCV 2018 Workshops*. Ed. by L. Leal-Taixé and S. Roth. Cham: Springer International Publishing, 2019, pp. 129–145. ISBN: 978-3-030-11012-3.

[26] W. Wang and J. Shen. "Deep Visual Attention Prediction". In: *IEEE Transactions on Image Processing* 27.5 (2018), pp. 2368–2378.

[27] K. Hara, M.-Y. Liu, O. Tuzel, and A.-m. Farahmand. "Attentional Network for Visual Object Detection". In: (Feb. 2017).

[28] C. YuanQiang, D. Du, L. Zhang, L. Wen, W. Wang, Y. Wu, and S. Lyu. "Guided Attention Network for Object Detection and Counting on Drones". In: *Proceedings of the 28th ACM International Conference on Multimedia*. MM '20. Seattle, WA, USA: Association for Computing Machinery, 2020, pp. 709–717. ISBN: 9781450379885. URL: https://doi.org/10.1145/3394171.3413816.

[29] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. "FlowNet: Learning Optical Flow with Con-

volutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766.

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

[31] *Marketplace - UE marketplace*. https://www.unrealengine.com/marketplace/en-US/store.

[32] *Unreal Engine*. https://www.unrealengine.com/.

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

[34] P. Pueyo, E. Cristofalo, E. Montijano, and M. Schwager. *CinemAirSim: A Camera-Realistic Robotics Simulator for Cinematographic Purposes*. 2020. arXiv: 2003.07664 [cs.RO].

[35] A. Buck, M. Deardorff, D. T. Anderson, T. Wilkin, J. M. Keller, G. Scott, R. H. L. III, and R. Camaioni. "VADER: A Hardware and Simulation Platform for VisuallyAware Drone Autonomy Research". In: *SPIE*. 2021.

[36] M. Deardorff, B. Alvey, D. T. Anderson, J. M. Keller, G. Scott, D. Ho, A. Buck, and C. Yang. "Metadata Enabled Contextual Sensor Fusion for UnmannedAerial System-Based Explosive Hazard Detection". In: *SPIE*. 2021.

[37] B. Alvey, D. T. Anderson, J. M. Keller, A. Buck, G. Scott, D. Ho, C. Yang, and B. Libbey. "Improving Explosive Hazard Detection with Simulated and Augmented Data for an Unmanned Aerial System". In: *SPIE*. 2021.

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

[39] W. Wu, J. M. Keller, and T. A. Runkler. "Sequential possibilistic one-means clustering with dynamic eta". In: *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2018, pp. 1–8.

[40] *NumPy*. https://numpy.org/.