
Online One-Shot Learning for Indoor Asset Detection

by Adith Balamurugan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for
the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee

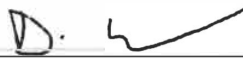


Professor Avidah Zakhor
Research Advisor

May 24, 2019

(Date)

★ ★ ★ ★ ★ ★ ★



Professor David Wagner
Second Reader

5/17/2019

(Date)

Abstract

Building floor plans with locations of safety, security and energy assets such as IoT sensors, thermostats, fire sprinklers, EXIT signs, fire alarms, smoke detectors, routers etc. are vital for climate control, emergency response, security, safety, and maintenance of building infrastructure. Existing approaches to building survey are tedious, error prone, and usually involve an operator with a clipboard and pen, or a tablet enumerating and localizing assets in each room. We propose an interactive method for a human operator to use an app on a smart phone to expedite such a task. One major component of this semi-automated building survey method is to accurately detect and classify assets of interest. Our approach is to use deep learning methods to train a neural network to recognize assets of interest, and uses human-in-the-loop interactive methods to correct erroneous recognition. These corrections serve to improve the accuracy of the model over time as more assets are recorded. A major hurdle faced when classifying via this method is that the appearance of a single type of asset, e.g. power outlet, can vary greatly from building to building or even from room to room, so a high rate of human correction is required to accurately recognize every asset of interest, since the model is unable to adapt in real time. In this thesis, we propose an online "one-shot learning" approach which combines aspects of long-term and short-term memory to minimize the amount of human correction required for accurate asset detection, even in situations involving never-before-seen asset appearances. We use a Neural Turing Machine (NTM) architecture, a type of Memory Augmented Neural Network (MANN), with augmented memory capacity which allows us to rapidly incorporate new data into our model to improve prediction accuracy after only a few examples, all without compromising the ability to remember previously learned data. This approach greatly improves the training time needed to update the model between building survey sessions. Experiments show that our proposed method matches and sometimes outperforms the prediction accuracy attained using more traditional batch processing deep learning methods where new data is used in conjunction with all old data to train the model. The advantage is especially pronounced for assets in new buildings that the model has never seen prior.

Contents

1. Introduction	5
2. Related Works	7
2.1. Interactive Asset Detection	7
2.2. Incremental Learning for Mobile Object Detection	9
2.3. Few-Shot and One-Shot Classification	9
2.4. Online Learning	10
2.5. Neural Turning Machine	11
2.6. Memory Augmented Neural Network	12
3. Method	14
3.1. Model	14
3.1.1. External Memory	14
3.1.2. Object Localization	16
3.1.3. Controller	18
3.2. Data	18
3.3. Training	20
3.4. Evaluation	23
4. Experimental Results and Analysis	24
4.1. Train on Negatives Only	26
4.2. Train on Positives and Negatives	27
4.3. All Memory Retained	28
4.4. Weighted Memory Retained	29
4.5. Comparison of Methods	30
4.6. Timing Results	33
5. Conclusion and Future Work	34

References	36
A. Appendix	38
A.1. Train on Negatives Only	38
A.2. Train on Positives and Negatives	41
A.3. All Memory Retained	42
A.4. Weighted Memory Retained	46

1. Introduction

Building floor plans with locations of safety, security and energy assets such as Internet of Things (IoT) sensors, thermostats, fire sprinklers, EXIT signs, fire alarms, smoke detectors, routers etc. are vital for climate control, emergency response, security, safety, and maintenance of building infrastructure [10]. Existing approaches to building survey are tedious and error prone. They usually involve an operator with a clipboard and pen, or a tablet enumerating and localizing assets in each room as they traverse from room to room and from building to building.

Kostoeva et al. recently proposed a semi-automated approach to solving this problem by creating a human operated smartphone app to create the 2D layout of a room, detect assets of interest using a deep learning method, and localize them within the layout [4]. In this thesis, we focus on improving the second task, detecting assets of interest. Authors of [4] use deep learning methods to train a neural network to recognize the assets of interest in a captured image, and use human-in-the-loop interactive methods to correct erroneous recognition. These corrections serve to improve the accuracy of the model over time as more assets are recorded. One major shortcoming in this approach is the high latency between a human correction and the ability of the model to reflect the new information learned. In choosing a more traditional learning approach for asset classification, the model requires a lengthy training session in order to update its weights to incorporate the newly collected information. Depending on the number of asset images to be trained upon, this training process could take hours or even days. Hence, in a new environment, where the assets in the building do not resemble assets previously seen in the training data, the human operator finds himself or herself correcting the categorization of the asset at a very high rate.

It is important to note that the appearance of a single type of asset, e.g. power outlet, can vary greatly from building to building or even from room to room, so we elect to implement a model that is able to adapt in real time to avoid high training latency. In this thesis, we propose an online "one-shot learning" approach which combines aspects of long-term and short-term memory to minimize the amount of human correction required for accurate asset detection, even

in situations involving previously unseen asset appearances. We use a Neural Turing Machine (NTM) [2] architecture, a type of Memory Augmented Neural Network (MANN) [15], with augmented memory capacity which allows us to rapidly incorporate new data into our ability to make accurate predictions after only a few examples, all without compromising the ability to remember previously learned data. This architecture lends itself nicely to the problem at hand. The NTM combines the ability to slowly learn abstract representations of raw image data, through gradient descent, with the ability to quickly store bindings for new information, after only a single presentation, by utilizing an external memory component. This combination enables us to tackle both a long-term category recognition problem where we can identify 10 different classes of objects across different buildings as well as an instance recognition problem where the model can quickly learn to recognize a particular never-before-seen instance of an asset as belonging to a certain category.

Whereas in [4], the operator of the app would be required to completely retrain the model on all the collected asset data before the performance would reflect the added information, now the new information is assimilated almost instantly and can be robustly trained later on to be reflected in a long-term capacity. Experiments show that our proposed method matches and sometimes outperforms the prediction accuracy attained in [4], when detecting 10 different classes of assets. The advantage is especially pronounced when we evaluate the performance on asset images from new buildings that the model has never seen prior.

The outline of this thesis is as follows: In Section 2, we describe related work in this area and how they relate to and inspire the approach we present in this thesis. Section 3 contains the detailed view of the method we use. This includes the specifics of the model, the dataset we use, and the training and evaluation pipelines. In Section 4, we present the results and of our experiments and analyze how this approach performed against the approach used in [4]. Lastly, in Section 5, we discuss our findings, limitations in the experimental design and execution, and next steps to take in putting this approach into practice.

2. Related Works

In this Section, we review existing work in six areas: Interactive Asset Detection, Incremental Learning, One-Shot and Few-Shot Classification, Online Learning, Neural Turing Machine (NTM), Memory Augmented Neural Networks (MANN).

2.1. Interactive Asset Detection

The automation of the building survey process to be able to create a layout of rooms, detect assets, and plot them within the room is critical in speeding up the otherwise manual process. The authors in [4] propose an interactive human operated smartphone application using Augmented Reality (AR) technology, which allows the placement of virtual anchors in the real world to detect location of the assets. This is possible since phones nowadays are equipped with powerful processors and many sensors, such as cameras and inertial measurement units. [4] also incorporates an object detection pipeline residing on the smartphone itself used to classify each asset on the screen into one of 10 classes. As mentioned in section 1, we intend to provide an alternative method to the object detection pipeline presented in [4] so as to update the recognition model in near real time.

In Figure 2.1, screenshots of the application are visible where the operator points at an asset and taps, following which the captured image is passed through a Single Shot Detector (SSD) neural network and a class prediction is made along with the confidence level, which are both presented on the screen. At this point, the application user has the opportunity to either move on to the next asset if correctly classified or dis-validate the prediction by tapping "UNDO", selecting the correct class label, and drawing a bounding box around the region of the screen containing the asset of interest. This tuple of image, bounding box, and ground truth label is later used to update the model during training.

The approach used in [4] for detecting assets is the Single Shot Detector (SSD) model which has been pre-trained on the MSCOCO dataset [7]. For every session of new asset data collected by the app operator, the last fully connected layer of the model is retrained on the entirety of the



(a)



(b)

Figure 2.1.: 3D Indoor Smartphone Application screenshots, highest probability classification of the asset on the screen along with confidence displayed. (a) Router correctly classified. (b) Light switch correctly classified.

dataset, including all past images. This is an extremely long process and only grows in duration as the amount of sample data grows in size.

To reduce the size and complexity of the model to operate on a smartphone, the Tensorflow neural network is frozen and the inference graph is converted into a much smaller 22 MB TFLite model, which is an offline model optimized for smartphone devices.

In this thesis, we present the MANN approach which uses an external memory component to learn short-term bindings, allowing it to adapt to new data in real time without robust training of the model weights. Additionally, after every session of new asset data collection, the model weights can be incrementally updated by training on only the new images, since the model weights are primarily used to extract general features and an abstract representation of the raw input image. The approach we present in this paper is meant to replace the existing deep learning architecture presented in [4].

2.2. Incremental Learning for Mobile Object Detection

Currently, we focus on 10 object classes of interest, but it is possible a new type of asset is developed or we encounter a new type of asset crucial to the building survey, which the model does not account for. Generally, this problem is solved by incrementally adding objects of the new class to the model's understanding and by fine-tuning the model with training data from the existing classes as well as the new one. However, this is not practical in many cases where new classes must be incorporated there and now. Learning to detect new objects incrementally in a timely and quality conscious manner is crucial in many application scenarios. Researches from Samsung, Apple, and USC have proposed a solution to this incremental learning problem by adding new object classes using only new class examples for training [5]. In order to prevent the model from "catastrophically forgetting" [3, 16] its understanding of the old classes in lieu of the new, [5] developed a data collection and annotation pipeline along with a novel loss function which allows the model to learn without forgetting [6]. This concept is similar to the short-term versus long-term memory approach we use in our approach.

2.3. Few-Shot and One-Shot Classification

In few-shot classification, the objective is to train a classifier from only a few labeled examples. Recent advances in few-shot classification have involved meta-learning approaches where a parameterized model is defined and trained in episodes representing different classification problems. In [12], training episodes also include unlabeled examples which may either belong to one of the same set of classes as the rest of the training data or from a completely new class. Through an extension of Prototypical Networks [17], the models can learn to leverage the unlabeled data during training to improve the classification accuracy on the labeled data, much as in a semi-supervised learning environment.

These techniques are not needed for our current problem since there is a finite number of categories that persist throughout the different buildings we test in. However, for future work, the model might extend to new asset classes that may prove noteworthy but are not currently included in our enumeration. In this case, including them during training should still increase the robustness of our model even if the class itself has never been seen before.

The exploitation of an additional big dataset with different categories can be used to improve the accuracy of few-shot classification over a different "target" dataset [8]. This idea is founded

upon the observation that images can be decomposed into different objects, which many different datasets may contain in common. Using this object level relation learned from the supplemental dataset, the similarity of images from the target dataset can be better determined. The approach presented in this thesis uses a similarity function to improve classification accuracy by generating similarity key to asset category bindings, in external memory. Our approach does not introduce an auxiliary dataset and the similarity key is generated by the model using the images it has already seen prior.

2.4. Online Learning

The umbrella of online learning refers to a method of machine learning where the data becomes available in a sequential order and the model trains on this information in a stream. Offline training, which is more typical in object detection frameworks occurs when the entire static dataset is collected beforehand and the training occurs on the whole data in one session. Online training gives us the opportunity to quickly incorporate new information such that we can improve our model rapidly to accommodate new situations.

Working systems using unsupervised and an online learning framework to detect moving objects utilize an "automatic labeler" that uses motion information to supply labeled images directly from a video feed. This video feed becomes a stream of images that an online learner uses to train a classifier. With the development of an effective labeler, the classifier can be trained in an online fashion using Winnow algorithm [11].

The problem with traditional object detection is amassing large annotated datasets and the significant time required to train them offline. While this is a viable starting point, we need our system to react to new inputs and make decisions immediately. Some applications require training in real-time on live video streams with a human-in-the-loop. This class of problem is referred to as time ordered online training (ToOT) [19]. ClickBAIT-v2 takes a human annotated approach where a quick single click by a human on an incoming video frame can provide the bounding box annotation required to train an object detector in real time. They leverage the time-ordered nature of the video input to track the object once an initial position is provided via a user click. In this thesis, we leverage an actual human produced ground truth as part of the training pipeline.

In [9], a novel pipeline for object detection training yielding 60x training speedup is proposed. The pipeline combines (i) the Region Proposal Network and the deep feature extractor from [13] to efficiently select candidate RoIs and encode them into powerful representations, with (ii) the FALKON [14] algorithm, a novel kernel-based method that allows fast training on large scale problems. The approach promises learning tasks involving 10 classes of objects with 10000 images of each class in a few seconds on a machine equipped with Intel(R) Xeon(R) E5- 2690 v4 CPUs @2.60GHz, and a single NVIDIA(R) Tesla P100 GPU, with FALKON set to not use more than 10GB of RAM. While this is tremendous improvement over traditional deep learning approaches using CNNs, we are interested in a more nuanced training that can make incremental changes to an existing model so that it does not forget prior data, but adapts quickly to the new incoming data.

2.5. Neural Turing Machine

A Neural Turing Machine (NTM) is a fully differentiable computer that can be trained using gradient descent [2]. It closely resembles a "working memory system," defined by having a capacity for short-term storage of information and its rule-based manipulation [1]. This is evident because the architecture is built with a process to read from and write to memory selectively. The NTM architecture consists of two major components, a neural network controller and a *memory bank*. The NTM model is pictured in Figure 2.2. At every step, the controller network receives inputs from the external environment and emits outputs in response. It also reads to and writes from a memory matrix via a set of parallel read and write heads [2].

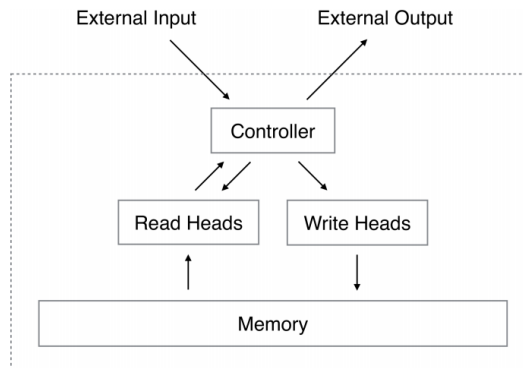


Figure 2.2.: Block diagram of Neural Turing Machine 2.2

The important part of the architecture in Figure 2.2 is that every component is differentiable and this includes the read and writes to memory. This is accomplished via "blurry" read and write

operations that interact to a greater or lesser degree with all the elements in memory. Because of the differentiability, the weights of the entire model can be updated via backpropagation. We describe the way we utilize this external memory for few-shot image classification in the following sections.

2.6. Memory Augmented Neural Network

Traditional gradient-based neural networks, much like the one used in [4], require a large amount of data to learn, through extensive iterations of training. As we have pointed out, when we encounter new data, the network must completely relearn its parameters to adequately adapt to the new information, which is very inefficient. Architectures with augmented memory capacity (MANNs) enable rapid encoding and retrieval of new information, which can eliminate the downsides of the more traditional approach [15].

In this approach, rather than attempting to determine parameters θ to minimize a learning cost \mathcal{L} across some dataset D , parameters are chosen to reduce the expected learning cost across a distribution of datasets $p(D)$:

$$\theta^* = \arg \min_{\theta} E_{D \sim p(D)}[\mathcal{L}(D, \theta)] \quad (2.1)$$

To properly set this up, one must define an episode, which involves the presentation of a dataset $D = \{\mathbf{x}_t, y_t\}_{t=1}^T$ where in the classification case, y_t is the class label for image \mathbf{x}_t . In this setup, y_t is both a target, and is presented as input along with \mathbf{x}_t , in a temporally offset manner; that is, the network sees the input sequence $(\mathbf{x}_1, \text{null}), (\mathbf{x}_2, y_1), \dots, (\mathbf{x}_T, y_{T-1})$. Thus, at time t the correct label for the previous data sample (y_{t-1}) is provided as input along with a new query \mathbf{x}_t . The network is tasked to output the appropriate label for \mathbf{x}_t , i.e. y_t , at the given timestep. In order to prevent the model from learning sample-class bindings during training, the samples from the dataset are shuffled before being fed to the model. It must appropriately learn to hold data samples in memory until the appropriate labels are presented at the next timestep, after which sample-class information can be bound and stored for later use. The ideal performance would be a random guess for the first presentation of a particular class since the appropriate label cannot be inferred from previous episodes, due to shuffling. However, beyond the first presentation, memory is used to achieve better accuracy. The model attempts to capture the predictive distribution $p(y_t | \mathbf{x}_t, D_{1:t-1}; \theta)$ [15].

We utilize this pipeline in the approach presented in this thesis with modifications to better suit our use case, but the crux of the implementation lies in this meta-learning approach which does not attempt to directly fit the provided dataset but rather fit a more general distribution. Note that this episodic form for training suits our real time asset detection problem particularly well since even during an "evaluation" phase, which would be when an operator of the smartphone app is in a never-before-seen building, he or she is still providing corrections to the model if the incorrect asset class is predicted. This means even during evaluation, the system receives ground truth labels in the subsequent timestep to be used in short term sample-class bindings in external memory to improve the model's performance using the new information.

The NTM, shown in Figure 2.2, is a fully differentiable implementation of a MANN [15]. It consists of a controller, such as a feed-forward network or LSTM, which interacts with an external memory module using read and write heads [2]. The NTM is perfect for few-shot prediction since memory encoding and retrieval is rapid and can be done efficiently using vector representations at potentially every timestep. A NTM can learn a good long-term strategy, via model weight updates, that determines the sample representations it should place into short-term memory. It later uses these representations when making predictions, so accurate predictions are possible even for classes that it has only seen once.

The work by Santoro et al. focused on testing this pipeline on the Omniglot dataset, which consists of over 1600 separate classes of handwritten characters from over 50 different alphabets with only a few examples per class. Two major differences in the problem we are solving are that we have far fewer classes to handle and we are dealing with far more complex images. The images of the assets can be taken from varying angles, tilts, lighting conditions, and, as mentioned prior, the assets themselves may not have the same appearance due to intraclass variability. Additionally, we are not necessarily interested in the entire image, but just a particular region of interest where the asset exists. We take these factors into consideration when we adapt this implementation to fit the problem at hand.

3. Method

In this section we outline the model specifications we use in our approach, the dataset used to for the asset detection problem we are addressing, as well as the training and evaluation pipeline. Specifically, the model details are highlighted in Section 3.1, Section 3.2 details the dataset we use to train and evaluate our approach, and in Sections 3.3 and 3.4 we describe the training and evaluation pipelines respectively.

3.1. Model

We propose to solve this asset detection problem using a MANN, specifically the fully differentiable NTM implementation. As described in Section 2.6, the NTM consists of a controller, chosen to be a Long Short-term Memory (LSTM), which interfaces with an external memory module using read and write heads [2]. The LSTM controller interacts with the memory using read and write heads, which rapidly retrieve representations from memory or place them into memory, respectively.

3.1.1. External Memory

Let \mathbf{M}_t be the contents of the $N \times M$ memory matrix at time t , where M is the dimension of the condensed representation created for an input image and N denotes the number of rows in the matrix. We let N equal 10, the number of classes in our system. Having a finite, fixed N allows us to limit the amount of space required for the external memory while also allowing the system to learn sample representation-class bindings for each of the different asset classes we intend to classify. Given an input, \mathbf{x}_t , which in our case is a vectorized raw image, the controller produces a $(1 \times M)$ -dimensional vector key, \mathbf{k}_t , which is used to quickly read from memory in a specific manner we describe below.

We index into the memory matrix \mathbf{M}_t using the cosine similarity measures between each key and each row of the matrix

$$K(\mathbf{k}_t, \mathbf{M}_t(i)) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\|\mathbf{k}_t\| \|\mathbf{M}_t(i)\|} \quad (3.1)$$

where $\mathbf{M}_t(i)$ denotes row i of the memory matrix. The similarity metric computed with each row, equivalent to a class representation, in the memory matrix is then used to compute a read weight for each row in memory, $\mathbf{w}_t^r(i)$, using a softmax:

$$\mathbf{w}_t^r(i) = \frac{\exp K(\mathbf{k}_t, \mathbf{M}_t(i))}{\sum_{j=1}^N \exp K(\mathbf{k}_t, \mathbf{M}_t(j))} \quad (3.2)$$

The $(1 \times M)$ -dimensional "memory" vector \mathbf{r}_t that is ultimately read is a weighted sum of all the rows using the $(1 \times N)$ -dimensional read weights vector \mathbf{w}_t^r .

$$\mathbf{r}_t = \mathbf{w}_t^r \mathbf{M}_t^\top \quad (3.3)$$

The retrieved memory vector \mathbf{r}_t is returned to the controller and is then used as input to the classifier which makes an asset class prediction \hat{y}_t for the original input \mathbf{x}_t . The class prediction \hat{y}_t is also passed as an additional input to the next controller state, since reconciliation steps, error calculation, are taken when the true label of the image is provided at the subsequent timestep.

Encoding and writing new information to memory is inspired by input and forget gates of an LSTM [2]. Each write is broken into two parts, *erase* and *add*. We write into memory when the model receives the true class label, y_t , for a particular image \mathbf{x}_t in the following timestep ($t + 1$). In the current implementation, at time $t + 1$, the controller generates a new erase vector \mathbf{e}_{t+1} of M random values in range (0,1) during each write step and a scalar erasure weight w_{t+1} , which is set as a constant hyperparameter for the entire model. The write update to the appropriate row of the memory matrix occurs as follows:

$$\tilde{\mathbf{M}}_{t+1}(y_t) = \mathbf{M}_t(y_t)[\mathbb{1} - w_{t+1}\mathbf{e}_{t+1}] \quad (3.4)$$

Above, we add some noise to the row corresponding to the true class label we just received. This is to partially "forget" the sample representation-class binding the model has already developed to make room for the new information. In future improvements, we can have the erasure weight chosen based on the sample and the class itself. This can improve the quality of the sample representations stored in memory.

Next we add the representation for the image \mathbf{x}_t generated by the controller, \mathbf{k}_t , to the row:

$$\mathbf{M}_{t+1}(y_t) = \tilde{\mathbf{M}}_{t+1}(y_t) + w_{t+1}\mathbf{k}_t \quad (3.5)$$

Ideally, the model updates the row of the memory matrix M_t corresponding to the class label in order to incorporate the representation of the image x_t since we know it belongs to that class. For future inputs, this added information should help accurately categorize assets belonging to the same class.

3.1.2. Object Localization

Unlike existing one-shot and few-shot learning approaches, we face the additional challenge that the raw image samples may contain assets of interest that occupy only a small portion of the entire image. The methods mentioned in Sections 2.3 and 2.6 focus on the Omniglot dataset, shown in Figure 3.1, which contains images consisting entirely of pixels pertaining to the characters to be classified.

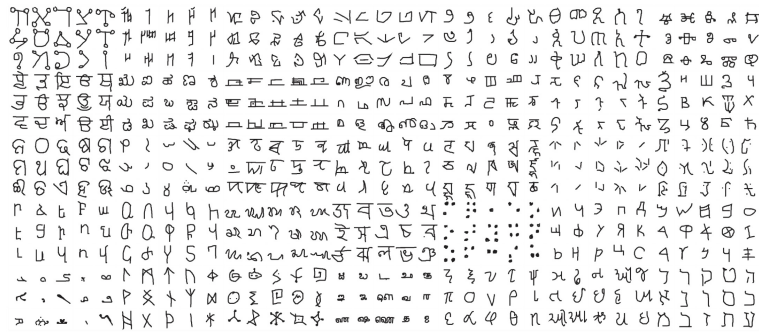


Figure 3.1.: Omniglot data example: grayscale 28 x 28 pixel images of characters

In our asset detection pipeline, we must deal with classifying objects with drastically lower object to image ratio such as the fire sprinkler shown in Figure 3.2 below.



Figure 3.2.: Fire sprinkler asset training image example

Rather than processing the entire raw image with no additional information on the pixel location of the asset, we localize the problem to a region of interest before the model classification.

In order to accomplish this we use classical image segmentation techniques [18] such as edge detection and clustering algorithms to identify a single 400×400 pixel region within the image with the most significant pixel values which we assume pertains to the asset object of interest.

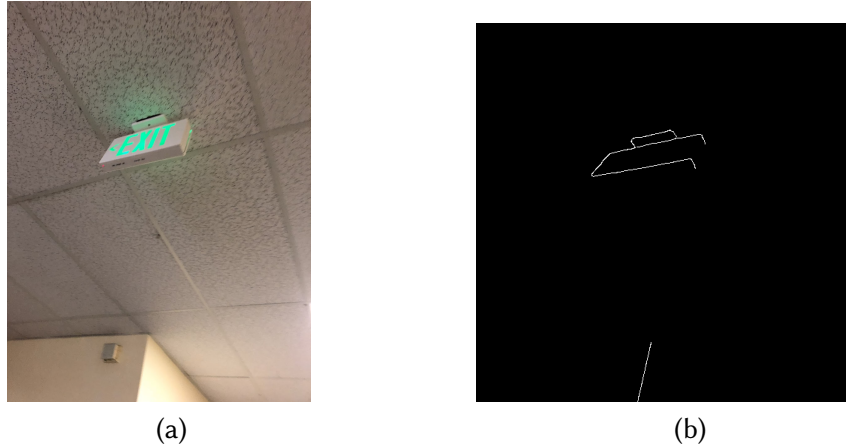


Figure 3.3.: (a) Original image (b) Canny edge detection run on the on the original image, and the pixels with greatest "activity" correspond to the exit sign

As can be seen in Figure 3.3, the edge detection algorithm finds the most significant differences between pixel values along the contours of the exit sign. Using the Canny edge detection output, we select the 400×400 pixel crop containing the most edges and instruct the model to focus on this region when predicting the correct asset class. While in many cases, this approximation to object localization correctly bounds the asset within the image, whenever the cropped image does not contain the entire asset, the model creates a bad sample representation and the accuracy can be punished. We can alleviate this issue slightly by providing the model with the true bounding box in the following timestep. We describe this in more detail in Section 3.3.

An even more ideal solution for reducing the size of the image is to capture the tap location within the smartphone application itself when the operator taps on the intended asset. The location of the tap is the best indicator of where within the captured image the asset of interest lies. This will be incorporated in future iterations of the application. The precision of this tap requires some additional attention and possible training for the phone application operator. However, it is important to note that the amount of skill and time required to operate this

application is still negligible compared to the time and training required to survey a building the old-fashion way.

3.1.3. Controller

In our approach, the controller of the NTM is mostly a LSTM, which connects the inputs and outputs of subsequent timesteps. However, there are a few other components comprising the controller which we describe in this section. The controller takes in the (raw image, true label, bounding box coordinates) tuple as input and interfaces with the external memory in order to update the sample representation-class bindings we have stored. In our implementation, we use a LSTM with 200 hidden units, which worked well for our input.

Figure 3.4 visualizes how the inputs to the controller are used. At time t , the controller receives a 921614-dimensional vector consisting of 640×480 pixel raw image \mathbf{x}_t , one-hot encoded class label y_{t-1} , and bounding box coordinates b_{t-1} . We focus initially on the raw image, the first 921600 entries of the input vector. The controller crops the image using object localization methods described in Section 3.1.2 above. This 400×400 pixel cropped image, is passed into the LSTM and a key representation, \mathbf{k}_t , is outputted. This key is used to read memory \mathbf{r}_t from the memory matrix, \mathbf{M}_t , via Equations 3.1-3.3. The memory, \mathbf{r}_t , is used to make a class prediction, \hat{y}_t , for the input image, using a softmax layer. The raw image, key, and class prediction are passed via the LSTM as additional inputs in the following timestep.

Meanwhile, the true label, y_{t-1} , and bounding box information, b_{t-1} , at time t are used in combination with the additional inputs (\mathbf{x}_{t-1} , \mathbf{k}_{t-1} , \hat{y}_{t-1}) from the previous timestep to compute the cross entropy loss for that particular class prediction using prediction \hat{y}_{t-1} and ground truth y_{t-1} . This loss is used to update the pertinent weights of the LSTM via backpropagation. Additionally, we update the sample representation-class binding for class y_{t-1} by writing to memory using Equations 3.4-3.5. In Section 3.3, we discuss the exact update performed when writing to memory.

3.2. Data

We use the same dataset created in [4] for training and testing purposes, consisting of the following ten categories of assets: router, fire sprinkler, fire alarm, fire alarm handle, EXIT sign, card-key

reader, light switch, emergency lights, fire extinguisher, and outlet.

The breakdown of the data collected through the use of the smartphone app is shown in Table 3.1, which includes images of assets that were both correctly and incorrectly classified.

Day of Collection	Location	Counts of Sample Images by Asset										
		A	B	C	D	E	F	G	H	I	J	Total
Day 0 (Before Training)	Cory Hall	60	35	8	7	57	26	8	7	2	8	218
Day 1	Cory Hall	35	13	20	15	4	7	5	2	2	4	107
Day 2	Cory Hall	25	6	1	0	1	7	10	3	0	2	55
Day 3	Sutardja Dai Hall	6	7	6	3	7	8	9	0	4	2	52
Day 3	Evans Hall	0	8	10	6	3	9	0	14	1	7	58
Day 4	Cory Hall	31	11	15	6	9	6	10	3	4	4	99

Table 3.1.: A = Fire Sprinkler, B = Fire Alarm, C = Outlet, D = Light Switch, E = Router, F = EXIT sign, G = Card-key Reader, H = Emergency Lights, I = Fire Extinguisher, J = Fire Alarm Handle. Distribution of the classes of objects we trained and tested on, over 4 days of data collection.

Every image in the dataset described above is accompanied by the true label of the pictured asset as well as the top left and bottom right coordinates of a bounding box enclosing the asset of interest.

In [4], a smartphone application was used to collect 5 rounds of data, over a span of 4 days, in 3 distinct buildings, as shown in Table 3.1. The Day 0 data was all collected before use of the app and the labels and bounding boxes were assigned manually. Afterwards, a traditional SSD deep learning model was trained on all the data from Day 0 and loaded onto the phone. Then an operator took the phone with the trained model to Cory Hall and attempted to detect 107 asset images on Day 1 using the application. The collected data includes both the images for which the model correctly predicted the class, as well as those that were misclassified where the operator had to provide the true label manually.

Following Day 1, the model was again trained on the entirety of images from Day 0 along with all the images the model had misclassified during Day 1, less than 107 total. This newly updated model was loaded onto the phone for Day 2 and the process repeated through Day 4.

The performance of the model in [4] on each day is presented in Table 4.5 for comparison against the approach presented in this paper. We use the data in the dataset in a manner which

best matches how the traditional deep learning model approach used it.

Since this online learning approach should be able to adjust to new incoming data in real time, we expect it to outperform the traditional approach most noticeably in the two previously unseen buildings, Sutardja Dai and Evans Hall, since this is where the model is able to best utilize both its long-term and short-term memory components to learn new asset appearances before undergoing a long training phase. In the other cases, we expect to reach approximately the same prediction accuracy as the traditional method since the assets from Cory Hall would be familiar to the model already and the one-shot approach has no marked advantage.

3.3. Training

The dataset is denoted $D = \{\mathbf{x}_t, (y_t, b_t)\}_{t=1}^T$ where in our classification case, y_t is the class label for image \mathbf{x}_t and b_t consists of the bounding box information pertaining to the location of the asset within the image \mathbf{x}_t . In this setup, y_t is both a target, and is presented as input along with \mathbf{x}_t , in a temporally offset manner; that is, the network sees the input sequence $(\mathbf{x}_1, \text{null}), (\mathbf{x}_2, (y_1, b_1)), \dots, (\mathbf{x}_T, (y_{T-1}, b_{T-1}))$. So, at time $t + 1$ the correct label and bounding box coordinates for the previous data sample, (y_t, b_t) , are provided as input along with a new query \mathbf{x}_{t+1} . This is a single concatenated input vector of dimension 921614, 921600 for 640×480 RGB image, 10 for one-hot encoded label, 4 for bounding box coordinates, passed into the LSTM controller. For timestep $t + 1$, the network is tasked to output the appropriate label for \mathbf{x}_{t+1} , i.e. y_{t+1} , at the given timestep. Simultaneously, the model is updating its class representation for class y_t since it is now given information pertaining to the true category of the image sample from the previous timestep. Thus, in external memory, the model incorporates image data from \mathbf{x}_t into the sample representation-class binding for class y_t . In order to prevent the model from learning sample-class bindings during training, the samples, and their corresponding label and bounding box information, from the dataset are shuffled before being fed to the model. We want the model to learn to hold data samples in memory until the appropriate labels are presented at the next timestep, after which sample representation-class information can be bound and stored for later use.

For all t from 1 to T , the total number of training images for this episode, we repeat the process depicted in Figure 3.4. In Figure 3.4, we can see our NTM transition from time t to time $t + 1$. For just this depiction, we assume asset light switch is associated with class label 1. The label and bounding box information are used in combination with the information fed forward through the

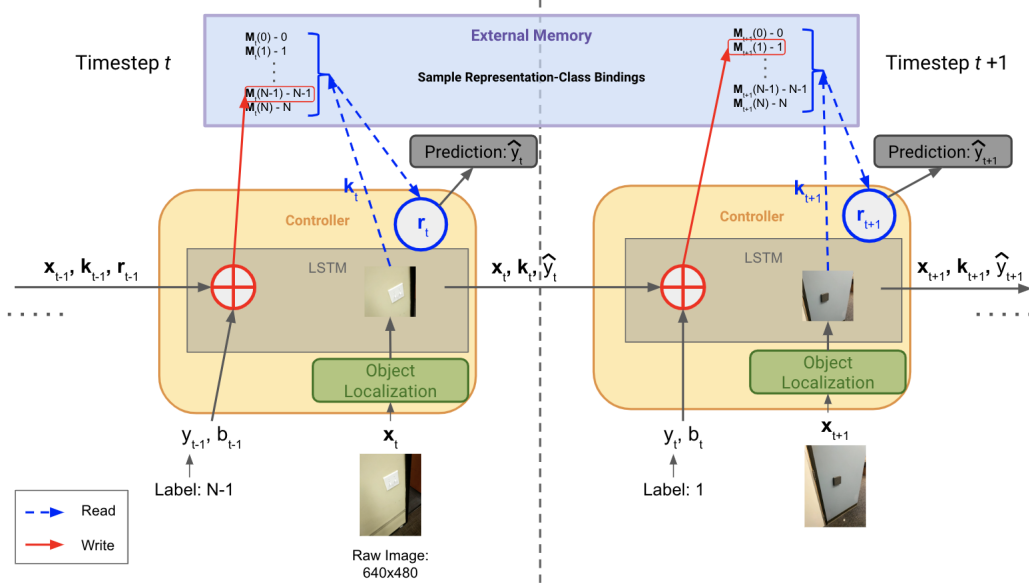


Figure 3.4.: The NTM transition from time t to timestep $t + 1$. For just this depiction, we assume as-set light switch is associated with class label 1. The label and bounding box information is used in combination with the information fed forward through the LSTM controller from the previous timestep to update the corresponding sample representation-class binding in external memory. The asset in the raw image is localized and cropped to be used in the memory read r_t , which is then used to make a class prediction \hat{y}_t .

LSTM controller from the previous timestep to update the corresponding sample representation-class binding in external memory. The asset in the raw image is localized and cropped to be used in the memory read r_t , which is then used to make a class prediction, \hat{y}_t .

Notice that if the external memory component does not yet have a robust sample representation-class binding for a class, because it is the first presentation of this particular class after clearing the memory, the model's inference is close to a random guess. At the following timestep it receives the true label and subsequent appearances of objects of the same class are classified with more accuracy. In practice, we can eliminate this phenomenon by not clearing the external memory component after a round of testing, as long as we know that the model architecture and the number of classes have not changed. This does not work if the model is run on different phones, but for the same user repeatedly surveying different buildings, this reduces the number of erroneous predictions on even the first appearance.

At the same time as the retrieval from memory, the model is also processing the other part of the input, which is the label and bounding box information (y_{t-1}, b_{t-1}) passed in at time t alongside image \mathbf{x}_t . The model combines the true label and bounding box with their corresponding image from the previous timestep. The label is passed in as a one-hot vector, where each different class is assigned an integer label ranging from 0 to $N - 1$, 9 in our case. This one hot vector determines which row of the external memory we alter in order to incorporate the sample representation \mathbf{k}_{t-1} from the image \mathbf{x}_t . The sample representation, \mathbf{k}_{t-1} , is passed from the previous timestep to the current one so the memory module can be updated. However, it is desirable and advantageous to take into account the provided ground truth bounding box from the user during an erroneous detection. Specifically, the controller computes a new key, \mathbf{k}'_{t-1} , using the true bounding box b_{t-1} , expanded or reduced to 400×400 pixels, to crop \mathbf{x}_{t-1} rather than the approximation computed through techniques for object localization. Since we are not updating the method by which the object localization approximation is achieved, we cannot completely omit the key, \mathbf{k}_{t-1} , computed using approximate localization. We update the memory matrix row specified by label y_{t-1} , following the write steps outlines in Equations 3.4-3.5. Instead of using \mathbf{k}_{t-1} computed using approximate object localization, we *add* the sample representation defined by the mean of the two keys we computed, that is $\frac{\mathbf{k}_{t-1} + \mathbf{k}'_{t-1}}{2}$.

$$\tilde{\mathbf{M}}_t(y_{t-1}) = \mathbf{M}_{t-1}(y_{t-1})[\mathbb{1} - w_t e_t] \quad (3.6)$$

$$\mathbf{M}_t(y_{t-1}) = \tilde{\mathbf{M}}_t(y_{t-1}) + w_t \frac{\mathbf{k}_{t-1} + \mathbf{k}'_{t-1}}{2} \quad (3.7)$$

Later, when a sample from this same class is observed, it should retrieve the stored binding pertaining to that class from the external memory to make a prediction. During the training phase, we compute the loss using the true label of the asset arriving in the following timestep and the model's prediction, which is passed in as additional input. The error is backpropagated from this prediction step to update the LSTM weights from the earlier steps in order to promote a better binding strategy [15]. Note that the LSTM weights affect the generated key representation, \mathbf{k}_t , for a provided sample image, so when the model weights are updated, the model moves away from a binding strategy that yielded an incorrect prediction, or it enforces a binding strategy which yielded in a correct prediction. This update of the LSTM weights enforces the long-term memory behavior of learning a good general binding strategy. We note that the backpropagation of error has no effect on the approximate object localization since that is accomplished purely through classical image processing techniques.

3.4. Evaluation

The evaluation of the model is meant to represent the model's ability to adapt to never-before-seen appearances of assets on the fly. The evaluation process should emulate how well a model can perform while an operator of the smartphone application is out surveying a new building for the first time. This means, the model does not make changes to its binding strategy, i.e. no weight updates are performed, and the model is evaluated on its classification accuracy for new data, only adapting via changing the external memory for short-term learning.

Hence, we define our evaluation pipeline to be exactly the same as our training flow with the single change that there is no backpropagated signal updates during the prediction step. In other words, the model weights remain fixed and the model is assessed on how well the long-term strategies it has learned thus far allow it to adjust to accurately classify the new dataset.

We take a moment to point out that it makes sense to maintain the same pipeline as for training because in our use case, when the operator is using the app to survey buildings, the model is provided with a true label and a bounding box for every collected sample image. If the model initially incorrectly classifies the asset, then the operator provides a correction in the form of a label and bounding box. If the model correctly classifies the asset, then no correction is made, indicating to the model that it correctly predicted the ground truth label and the approximate object localization it computed serves as a valid bounding box. Unlike the original application of NTM to the Omniglot digit dataset, in our application, the testing process behaves as though the model receives an input image, and the label and bounding box are provided "at the next timestep." This further justifies our choice to use this episodic learning format to train and evaluate the model.

4. Experimental Results and Analysis

In this section, we evaluate our approach with 4 different training and evaluation schemes. In Section 4.1, we train our model weights using only the negative, or incorrectly classified, images from the previous Day of data before evaluating on the next Day's dataset. This most closely resembles the training data used in [4], where the SSD model was trained after each day of data collection on all the previous training images, plus the misclassified images from the current day. In our approach, we do not retrain the model on all the old data, only the new. In Section 4.2, we train the model weights after each Day of data using the entire dataset, including both positively and negatively classified objects. In Sections 4.1-4.2, we always wipe the external memory after each training phase and after each evaluation phase to determine the model's ability to adapt to new buildings and asset appearances using primarily short-term memory. Section 4.3 explores the model's performance when we train on both positive and negative samples, but we never wipe the external memory matrix. After each training phase, the memory persists into the following testing phase and similarly, after each testing phase the memory matrix persists into the next training phase. This mitigates the first appearance problem where the model is forced to make a random guess for the class of an asset on the first time it is presented due to an empty memory matrix. By maintaining the memory throughout, the model always has a binding stored in memory for each class it has seen, even for the first appearance of a class during evaluation. While memory retention does lessen the first appearance problem, it prevents the model from best utilizing its short-term learning capability since the information from all the previously seen old data is still in external memory, placing less emphasis on the new incoming data. In Section 4.4, we take a similar approach as before where the memory is retained after each training and evaluation phase, but in order to place emphasis on the new data and the short-term learning of the model, the retained memory is down-weighted by a "decay factor" after training, before it is passed on to the evaluation phase. In Section 4.5, we compare the overall performance of all 4 of our approaches on each Day of data with the method used in [4]. We note specifically where we see improvement and analyze the reasons for variations in prediction accuracies. Lastly, in Section 4.6, we discuss the advantages of our approach in terms of offline training time as well as inference

speed compared to that of [4].

The training phases in between each of the evaluation phases consist of training for 30000 episodes on the previous Day's images, along with augmented versions of these input images. The augmentations applied to these images include (a) flip the original image horizontally; (b) flip the original image vertically; (c) adjust the brightness of the original image by a factor in the range $[0, 0.2]$; (d) rotate the original image by 90 degrees. These augmentations were accomplished using options native to the Tensorflow Object Detection API and bounding box information was also augmented accordingly, if applicable. The augmented images were passed into the model along with the original images, in a shuffled order. Note that image augmentations were only used during the training phase and not during the evaluation phase since we want that to be as close to real time as possible.

During evaluation of the model, we take all the samples in the current dataset and shuffle them into a random order. Since the model receives the data in a sequential fashion, we want to measure how well the model adapts to the new data after it sees object of a certain class one time, two times, three times, etc. The " k^{th} appearance" of a particular class is when the model has already encountered, and incorporated into short-term memory, $k - 1$ previous images of that particular class of object. So the accuracy of the model on the k^{th} appearance of a class is computed by averaging the prediction accuracies for each of the 10 classes only on the k^{th} appearance of that particular class. Because we can capture the asset images in different orders, we want to see how well the model performs on the k^{th} appearance, across different permutations of the same dataset. We found that with 100 permutations, we converge towards a single accuracy. Consequently, in our results tables, we present the accuracies for each of the 4 training schemes on each of the 4 Days of data, where we provide the k^{th} appearance accuracies, for k from 1 to 10, averaged over 100 different permutations of the dataset.

In Sections 4.1-4.4, we provide the k^{th} appearance accuracies for each of the 4 methods of training, averaged over the 10 different classes of objects. In some cases, the k^{th} appearance accuracy for higher k , i.e. 8, 9, 10, decreases in comparison to the smaller k . This is due to the fact that in some of the datasets, some classes do not have 8, 9, or 10 appearances, so the accuracies presented in the tables in this section are averaged over fewer classes, some of which may be underrepresented. The k^{th} appearance accuracies found in the tables in Sections 4.1-4.4 can be computed by averaging the k^{th} appearance accuracies of the 10 different classes. For the detailed

k^{th} appearance accuracy results for each class separately, refer to Appendix A.

4.1. Train on Negatives Only

In this first training setup, we train the one-shot approach model on Day 0 data, call this model v0, prior to all evaluation. Then, we freeze the model weights and evaluate on Day 1 data. We then do an episodic training on the previously saved model, v0, using the misclassified examples from Day 1 only. Now we call this model v1. We then evaluate the model on never-before-seen Day 2 data. Continuing this pattern, we train the saved model v1 on incorrectly classified Day 2 images only, now calling it model v2, and evaluate it separately on the two different Day 3 collections. Then, we train model v2 on both Day 3 collections’ incorrectly classified images, dubbing it model v3, and evaluate it on the Day 4 data. We wipe the external memory component of the model after every training and evaluation phase.

The combined-class results of this procedure are found in Table 4.1. For the class specific k^{th} appearance accuracies, refer to Tables A.1-A.5 in Appendix A. The last row of each of those tables match the results presented in Table 4.1.

Results by Appearance (Trained on Negatives Only)										
	Appearance (average over 100 permutations)									
Evaluation Set	1	2	3	4	5	6	7	8	9	10
Day 1 (Cory Hall)	0.151	0.413	0.55	0.609	0.632	0.632	0.632	0.64	0.658	0.655
Day 2 (Cory Hall)	0.169	0.458	0.554	0.683	0.71	0.705	0.66	0.595	0.6	0.625
Day 3 (SDH)	0.2	0.514	0.618	0.664	0.723	0.733	0.803	0.775	0.67	–
Day 3 (Evans Hall)	0.204	0.486	0.58	0.62	0.672	0.662	0.706	0.69	0.62	0.495
Day 4 (Cory Hall)	0.18	0.592	0.622	0.702	0.716	0.723	0.754	0.75	0.768	0.813

Table 4.1.: k^{th} appearance prediction accuracy, averaged over all 10 classes, averaged over 100 permutations of each evaluation dataset. Training method using negative samples only.

In Table 4.1, we observe the prediction accuracies improve as the model sees more instances of the same class. Additionally, we can see that even in new buildings such as Sutardja Dai Hall and Evans Hall, the model is able to adapt to the different appearance of objects despite not seeing them before. Since the memory is cleared, the first appearance prediction is clearly problematic

in this method as can be seen by examining the first column.

4.2. Train on Positives and Negatives

In the second training setup, we train the one-shot approach model on Day 0 data, call this model v0, prior to all evaluation. Then, we freeze the model weights and evaluate on Day 1 data. We then do an episodic training on the previously saved model, v0, using both the misclassified examples from Day 1 as well as the correctly classified samples. Call this model v1. We then evaluate the model on never-before-seen Day 2 data. Repeating this process, we train the saved model v1 on both positive and negative Day 2 images, now calling it model v2, and evaluate it separately on the two different Day 3 collections. Next, we train model v2 on both Day 3 collections, positive and negative images, dubbing it model v3, and evaluate it on the Day 4 data. We wipe the external memory component of the model after every training and evaluation phase.

The combined-class results of this procedure are found in Table 4.2. For the class specific k^{th} appearance accuracies, refer to Tables A.6-A.9 in Appendix A. The last row of each of those tables match the results presented in Table 4.2.

Results by Appearance (Trained on Positives + Negatives)										
	Appearance (average over 100 permutations)									
Evaluation Set	1	2	3	4	5	6	7	8	9	10
Day 1 (Cory Hall)	0.151	0.413	0.55	0.609	0.632	0.632	0.632	0.64	0.658	0.655
Day 2 (Cory Hall)	0.166	0.518	0.606	0.7425	0.768	0.775	0.72	0.66	0.67	0.675
Day 3 (SDH)	0.201	0.617	0.65	0.699	0.738	0.762	0.843	0.82	0.73	–
Day 3 (Evans Hall)	0.143	0.564	0.617	0.648	0.675	0.68	0.712	0.703	0.63	0.525
Day 4 (Cory Hall)	0.148	0.653	0.685	0.731	0.727	0.739	0.74	0.762	0.766	0.823

Table 4.2.: k^{th} appearance prediction accuracy, averaged over all 10 classes, averaged over 100 permutations of each evaluation dataset. Training method using all, positive and negative, samples.

There is general improvement across the board in Table 4.2, compared to the previous method since we are providing the model with more training data. By providing both correctly classified and incorrectly classified images, we can enforce better binding strategies, by positively enforcing

the ones yielding correct classifications and moving away from the misclassifications.

4.3. All Memory Retained

We observe that the training and testing schemes presented in Sections 4.1 and 4.2 suffer from the random prediction on the first appearance and the 1th appearance accuracies are very low. In this section, we maintain the memory matrix and the problem is reduced.

In this method, we train the one-shot approach model on Day 0 data, call this model v0, prior to all evaluation. Then, we freeze the model weights, load the external memory state from the end of training, and evaluate on Day 1 data. We then do an episodic training on the previously saved model, v0, using both the misclassified examples from Day 1 as well as the correctly classified samples. Call this model v1. We then load the memory matrix saved at the end of training and evaluate the model on never-before-seen Day 2 data. Repeating this process, we train the saved model v1 on both positive and negative Day 2 images, now calling it model v2, and evaluate it separately on the two different Day 3 collections, starting both evaluations with the restored memory matrix from the end of training. Next, we train model v2 on both Day 3 collections, positive and negative images, dubbing it model v3, retain the memory matrix, and evaluate it on the Day 4 data.

The combined-class results of this procedure are found in Table 4.3. For the class specific k^{th} appearance accuracies, refer to Tables A.10-A.14 in Appendix A. The last row of each of those tables match the results presented in Table 4.3.

Results by Appearance (All Memory Retained)										
	Appearance (average over 100 permutations)									
Evaluation Set	1	2	3	4	5	6	7	8	9	10
Day 1 (Cory Hall)	0.488	0.554	0.654	0.668	0.701	0.722	0.714	0.708	0.725	0.743
Day 2 (Cory Hall)	0.6	0.695	0.704	0.775	0.805	0.808	0.763	0.755	0.765	0.735
Day 3 (SDH)	0.537	0.629	0.634	0.669	0.735	0.747	0.803	0.77	0.75	–
Day 3 (Evans Hall)	0.494	0.573	0.599	0.66	0.672	0.672	0.718	0.7	0.603	0.52
Day 4 (Cory Hall)	0.528	0.666	0.703	0.746	0.727	0.744	0.754	0.77	0.762	0.795

Table 4.3.: k^{th} appearance prediction accuracy, averaged over all 10 classes, averaged over 100 permutations of each evaluation dataset. Training method using all, positive and negative, samples and memory is never wiped between training and evaluation.

The performance improves significantly with the retention of memory, as can be seen in Table 4.3. There is not a significant improvement in the new buildings, SDH and Evans Hall, from Section 4.2. This may be due to the fact that Cory Hall data bindings are still heavily present in the memory matrix when we evaluate Day 3 data. In the Section 4.4 method, we avoid that problem by decaying the older memory so as to not overpower the new data when making predictions.

4.4. Weighted Memory Retained

In Section 4.3, we retain the exact memory matrix after each training phase and evaluation phase. This does not best utilize the model’s short-term learning functionality which can quickly adapt to new information, since all the old information is still stored in memory. We overcome this by down-weighting the existing memory each time we retain it, i.e. multiply the memory matrix by a positive scalar less than 1, so that the older memory is gradually reduced in presence. This way the model is forced to place more weight on the new incoming data, but still retain old bindings so that it can make informed predictions even on the first few appearances of each class. In order to determine the best scalar to use when down-weighting the memory, we treated it as a hyperparameter of the model. By repeatedly running the algorithm on the data with different decay factors, we empirically determine that the ideal decay weight for our model is 0.684.

In this weighted memory retention method, we train the one-shot approach model on Day 0 data, call this model v_0 , prior to all evaluation. Then, we freeze the model weights, load the down-weighted external memory state from the end of training, and evaluate on Day 1 data. We then do an episodic training on the previously saved model, v_0 , using both the misclassified examples from Day 1 as well as the correctly classified samples. Call this model v_1 . We then load the memory matrix saved at the end of training, multiply it by 0.684, and evaluate the model on never-before-seen Day 2 data. Repeating this process, we train the saved model v_1 on both positive and negative Day 2 images, now calling it model v_2 , and evaluate it separately on the two different Day 3 collections, starting both evaluations with the decayed, restored memory matrix from the end of training. Next, we train model v_2 on both Day 3 collections, positive and negative images, dubbing it model v_3 , retain the memory matrix, multiply it by our decay factor, and evaluate it on the Day 4 data.

The combined-class results of this procedure are found in Table 4.4. For the class specific k^{th} appearance accuracies, refer to Tables A.15-A.18 in Appendix A. The last row of each of those tables match the results presented in Table 4.4.

Results by Appearance (Weighted Memory Retained)										
	Appearance (average over 100 permutations)									
Evaluation Set	1	2	3	4	5	6	7	8	9	10
Day 1 (Cory Hall)	0.488	0.554	0.654	0.668	0.701	0.722	0.714	0.708	0.725	0.743
Day 2 (Cory Hall)	0.608	0.708	0.724	0.793	0.805	0.815	0.767	0.76	0.785	0.795
Day 3 (SDH)	0.561	0.656	0.662	0.703	0.75	0.768	0.818	0.77	0.79	–
Day 1 (Evans Hall)	0.564	0.634	0.693	0.745	0.752	0.758	0.796	0.765	0.693	0.61
Day 4 (Cory Hall)	0.579	0.725	0.745	0.783	0.78	0.79	0.818	0.822	0.82	0.835

Table 4.4.: k^{th} appearance prediction accuracy, averaged over all 10 classes, averaged over 100 permutations of each evaluation dataset. Training method using all, positive and negative, samples and memory retained between training and evaluation, but gradually decaying.

Using this method, we see in Table 4.4 the model reaches higher prediction accuracy with less appearances of a class even in new buildings. This means the short-term adaptability of the model is being utilized properly. There is also a general improvement across all the buildings when we slowly decay the older memory. We see improvement in the Day 4 Cory Hall accuracies as well, despite having decayed the memory matrix three times since the Day 1 and Day 2 Cory Hall training phases. This is an interesting result and shows that this approach may be the most powerful in both new buildings and in old buildings. If retaining the external memory is a possible in practice, then opting to do so could provide the best results.

4.5. Comparison of Methods

Now we compare the performance of the approaches presented in this thesis with that of the approach implemented in [4]. In order to best compare our results to their metrics, for each dataset, we compute an overall prediction accuracy. This is computed by taking the total number of objects classified correctly in the entire dataset divided by the total number of total images in the dataset. This metric treats objects of all classes equally. The accuracy provided for each dataset in [4] was computed by taking pictures of each asset in one particular order and classi-

ifying each object exactly once. In order to replicate this procedure, we shuffle the dataset into a random order, attempt to classify every single object once and record the overall accuracy. We repeat this accuracy computation on 100 permutations of the same dataset. In Table 4.5, we provide the overall accuracy computed averaged over all 100 permutations as well as the standard deviation for the accuracy metric. We note that at 100 permutations, the accuracy converges at a single number with low variance, so this number is comparable to the results produced in [4]. Since we are looking at the distribution of accuracies over 100 permutations of each dataset, we also provide the standard deviations of the overall accuracies measured as well as the minimum and maximum accuracies in the 100 measurements collected for that dataset.

We observe that for a familiar building such as Cory Hall, the approach [4] outperforms our variation with no memory retention. However, when it comes to generalizing to new buildings and adapting to new information, our one-shot approach delivers impressive results in unfamiliar surroundings such as in SDH and Evans Hall. When we incorporate the memory retention option, our approach outperforms the traditional SSD approach in [4] even in previously seen buildings despite not retraining on old data.

Our approach outlined in the paper does accomplish the goal we had in mind. The higher overall accuracies in Table 4.5 indicate that in new environments, we reduce the amount of human correction required when detecting assets of interest compared to the SSD approach used in [4]. Using our method, if we were to train on more samples of differing appearances and conditions, we could learn a very robust sample representation-class binding strategy which would generalize far better to never-before-seen instances of these assets than the traditional approach [4] could.

The Section 4.4 approach presented in the table had the best performance out of the 4 variations we tested. We attribute this to this gradual decay of the memory matrix, which enforces that the model retains older information from prior buildings while incorporating newer information with relatively higher weightage. Again, we note that retention of memory may not always be feasible in practice, but if we have that option, Table 4.5 indicates that performance can be improved by retaining it.

We can attribute some of our shortcomings in Table 4.5 to the fact that certain assets such as emergency lights and fire extinguishers were greatly underrepresented and brought down the overall prediction accuracy for our model, hiding the fact that the prediction accuracy for assets

Model	Evaluation Set				
	Day 1 (Cory Hall)	Day 2 (Cory Hall)	Day 3 (SDH)	Day 3 (Evans Hall)	Day 4 (Cory Hall)
SSD [4]	67.2%	81.7%	74.3%	54.7%	73.6%
Ours (Train on Negatives)					
Accuracy (%)	58.6	60.9	63.5	62.9	69.78
Standard Deviation (%)	0.11	0.08	0.05	0.14	0.03
Min Accuracy (%)	54.1	55.7	60.6	56.2	60.1
Max Accuracy (%)	62.0	64.1	65.8	67.5	70.3
Ours (Train on Positives + Negatives)					
Accuracy (%)	66.7	63.9	67.6	64.33	76.2
Standard Deviation (%)	0.21	0.11	0.08	0.13	0.02
Min Accuracy (%)	59.9	61.3	63.4	60.1	74.0
Max Accuracy (%)	70.0	66.2	69.3	66.7	77.3
Ours (All Memory Retained)					
Accuracy (%)	73.1	74.3	76.2	69.3	77.1
Standard Deviation (%)	0.12	0.13	0.01	0.10	0.008
Min Accuracy (%)	68.8	70.2	75.1	68.8	76.0
Max Accuracy (%)	75.2	75.1	77.1	70.1	77.8
Ours (Weighted Memory Retained)					
Accuracy (%)	73.1	82.3	74.7	69.8	79.0
Standard Deviation (%)	0.07	0.01	0.04	0.08	0.10
Min Accuracy (%)	71.8	77.8	70.9	67.1	77.2
Max Accuracy (%)	76.2	84.1	76.3	71.5	83.6

Table 4.5.: % Correctness using a traditional deep learning approach versus the approach presented in this paper. SSD results were reproduced on the exact datasets we use in this paper. We evaluated under four training schemes. One where the model weights were updated by training only on negative samples from the previous day. Another where the model weights trained using both positive and negative samples from the previous day. A third where the model weights were trained on both positive and negative samples from the previous day, and the short-term memory was never wiped. Lastly, the model weights were trained on both positive and negative samples from the previous day, and the short-term memory was retained after each dataset, but down-weighted by a factor of 0.684, chosen empirically.

such as the fire alarm and fire alarm handle were almost perfect.

We can also observe from the tables in Appendix A that the accuracies improve significantly after the first and second appearances of a particular class. If we look at the performance after

the model has adjusted to the building, the predictions of the model match or outperform the SSD [4] approach, even in the previously seen buildings.

Additionally, our model allows for the addition of new asset classes without needing to recreate the architecture and retrain from scratch. This could prove very useful in a practical application of this approach. If an asset we had not accounted for was present in a building, we could learn on the fly that the asset does not fall into any one of the categories and dynamically allocate more external memory space to construct a binding for this new asset type.

4.6. Timing Results

Arguably, the most significant result we find is that due to our approach's long-term and short-term learning capacities, we avoid the need to retrain on all the old data and can focus on training the model on only the additional new data. The results in Table 4.6 show the drastic difference in offline training times between the SSD model in [4] and the approach described in this thesis.

Data Set	Offline Training			Online Training		Online Inference	
	SSD (Kostoeva et al.)	Ours (Negatives)	Ours (All)	SSD [4]	Ours	SSD [4]	Ours
Day 0	18hr 29m	5hr 27m	5hr 27m	–	–	–	–
Day 1	20hr 11m	2hr 29m	4hr 17m	–	1.13 s	0.084 s	0.064 s
Day 2	20hr 56m	1hr 33m	3hr 4m	–	1.06 s	0.082 s	0.041 s
Day 3	22hr 40m	2hr 50m	4hr 13m	–	1.00 s	0.081 s	0.050 s
Day 4	–	2hr 1m	4hr 9m	–	1.21 s	0.084 s	0.034 s

Table 4.6.: Timing information for our approach versus the traditional deep learning approach. Note the SSD method has no Online Training component.

In Table 4.6, we present the timing results for each important function of the model compared to that of the traditional approach. Note that we do not lose any speed in generating a prediction and have less than 1 second of added latency when the model updates the memory matrix in its "Online Training" step, but, in practice, this time is comfortably less than the time it takes for an operator to move from one asset to the next. Most importantly, we cut down the total offline training time by a significant amount while still yielding comparable, if not better, performance.

5. Conclusion and Future Work

We have improved upon the traditional deep learning methods in the ability to detect and accurately classify indoor assets of interest. Our proposed method takes advantage of new information as it is presented in order to minimize the instances of human intervention needed to correct a misclassified example. The nature of the NTM allows us to take advantage of an external memory module which need not take up vast amounts of space and grows in size linearly with the number classes we can differentiate, rather than the number of sample images. This means even as we increase the number of different assets we are able to classify, the additional memory needed does not approach the memory capacity of a smartphone, which means the entire short-term memory system can reside on the smartphone itself. The long-term memory training, mainly the slow update of model weights, takes more time, but can be conducted offline and then the original model can be replaced.

The approach mentioned in this paper suffers from having to approximate the object's position within the image. When the model resides on the phone, we can eliminate this problem by providing the model with the raw image as well as the coordinates of the point the app operator tapped on the screen. It will be a much better approximation to use this tap location as the center of the 400×400 pixel bounding box we choose to approximate the ground truth. Another limitation of the model is the fact that on the very first appearance of a particular class during the evaluation phase, the model has to resort to guessing since the memory may be wiped. We simulated a case where the model retains its memory from the previous offline training phase for just the first prediction, and this did improve the prediction accuracy significantly from before. However, the option to preserve the memory for just the first prediction may not always be available because of memory limitations, changes in hardware, etc.

Future work includes: (a) migrating this approach onto the smartphone application and replacing the current TFLite model in [4], which has high training latency, (b) improving the accuracy of the system via pseudo-realistic augmentation of training examples, (c) extension to greater number of classes without modification to the architecture of the model [5], by choosing an

encoding schema other than one-hot, we can represent more than 10 classes, (d) improve the robustness of the model via collection of data from a wide array of different environments with differing asset appearances, and (e) utilizing the location of the finger tap on the phone screen in localizing the position of the asset within the captured sample image.

References

1. A. Baddeley, M. Eysenck, and M. Anderson. *Memory*. 2009.
2. A. Graves, G. Wayne, and I. Danihelka. “Neural Turing Machines”. In: *CoRR* abs/1410.5401, 2014. arXiv: 1410.5401. URL: <http://arxiv.org/abs/1410.5401>.
3. J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114:13, 2017, pp. 3521–3526. ISSN: 0027-8424. DOI: 10.1073/pnas.1611835114. eprint: <https://www.pnas.org/content/114/13/3521.full.pdf>. URL: <https://www.pnas.org/content/114/13/3521>.
4. R. Kostoeva, R. Upadhyay, Y. Sapar, and A. Zakhor. “INDOOR 3D INTERACTIVE ASSET DETECTION USING A SMARTPHONE”. In: *ISPRS. Indoor 3D workshop*. 2019. URL: <http://www-video.eecs.berkeley.edu/papers/revekka/revekka-revised-submission.pdf>.
5. D. Li, S. Tasci, S. Ghosh, J. Zhu, J. Zhang, and L. P. Heck. “Efficient Incremental Learning for Mobile Object Detection”. In: *CoRR* abs/1904.00781, 2019. arXiv: 1904.00781. URL: <http://arxiv.org/abs/1904.00781>.
6. Z. Li and D. Hoiem. “Learning without Forgetting”. In: *CoRR* abs/1606.09282, 2016. arXiv: 1606.09282. URL: <http://arxiv.org/abs/1606.09282>.
7. T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312, 2014. arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
8. L. Long, W. Wang, J. Wen, M. Zhang, Q. Lin, and B. C. Ooi. “Object-Level Representation Learning for Few-Shot Image Classification”. In: *CoRR* abs/1805.10777, 2018. arXiv: 1805.10777. URL: <http://arxiv.org/abs/1805.10777>.
9. E. Maiettini, G. Pasquale, L. Rosasco, and L. Natale. “Speeding-up Object Detection Training for Robotics with FALKON”. In: *CoRR* abs/1803.08740, 2018. arXiv: 1803.08740. URL: <http://arxiv.org/abs/1803.08740>.
10. D. Minoli, K. Sohraby, and B. Occhiogrosso. “IoT Considerations, Requirements, and Architectures for Smart Buildings—Energy Optimization and Next-Generation Building Management Systems”. In: *IEEE Internet of Things Journal* 4, 2017, pp. 269–283.

11. V. Nair and J. J. Clark. “An Unsupervised, Online Learning Framework for Moving Object Detection”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. CVPR’04. IEEE Computer Society, Washington, D.C., USA, 2004, pp. 317–325. URL: <http://dl.acm.org/citation.cfm?id=1896300.1896346>.
12. M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. “Meta-Learning for Semi-Supervised Few-Shot Classification”. In: *CoRR* abs/1803.00676, 2018. arXiv: 1803.00676. URL: <http://arxiv.org/abs/1803.00676>.
13. S. Ren, K. He, R. Girshick, and J. Sun. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.
14. A. Rudi, L. Carratino, and L. Rosasco. “Falkon: An optimal large scale kernel method”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3888–3898.
15. A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap. “One-shot Learning with Memory-Augmented Neural Networks”. In: *CoRR* abs/1605.06065, 2016. arXiv: 1605.06065. URL: <http://arxiv.org/abs/1605.06065>.
16. K. Shmelkov, C. Schmid, and K. Alahari. “Incremental Learning of Object Detectors without Catastrophic Forgetting”. In: *CoRR* abs/1708.06977, 2017. arXiv: 1708.06977. URL: <http://arxiv.org/abs/1708.06977>.
17. J. Snell, K. Swersky, and R. Zemel. “Prototypical networks for few-shot learning”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4077–4087.
18. G. Stockman and L. G. Shapiro. *Computer Vision*. 1st. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001. ISBN: 0130307963.
19. E. Teng, R. Huang, and B. Iannucci. “ClickBAIT-v2: Training an Object Detector in Real-Time”. In: *CoRR* abs/1803.10358, 2018. arXiv: 1803.10358. URL: <http://arxiv.org/abs/1803.10358>.

A. Appendix

A.1. Train on Negatives Only

The following 5 detailed tables, A.1-A.5, pertain to the training scheme used where the model weights were updated by training only on negative samples from the previous day.

Day 1 - Cory Hall (Trained on Day 0)											
	Appearance (average over 100 permutations)										
Asset	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.21	0.29	0.45	0.53	0.54	0.62	0.62	0.65	0.65	0.66	
Fire Alarm	0.14	0.45	0.63	0.72	0.69	0.65	0.73	0.78	0.75	0.73	
Outlet	0.18	0.45	0.54	0.55	0.6	0.59	0.57	0.58	0.61	0.62	
Light Switch	0.18	0.4	0.43	0.48	0.49	0.51	0.47	0.55	0.62	0.61	
Router	0.15	0.35	0.49	0.57	–	–	–	–	–	–	
EXIT sign	0.23	0.65	0.74	0.76	0.81	0.79	0.77	–	–	–	
Cardkey Reader	0.17	0.58	0.63	0.65	0.66	–	–	–	–	–	
Emergency Lights	0.02	0.04	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.13	0.4	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.1	0.52	0.53	0.61	–	–	–	–	–	–	
Total	0.151	0.413	0.55	0.609	0.632	0.632	0.632	0.64	0.658	0.655	11.531

Table A.1.: Model v0 evaluated on Day 1 data from Cory Hall

Day 2 - Cory Hall (Trained on Day 1 negative predictions)											
	Appearance (average over 100 permutations)										
Asset	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.3	0.35	0.48	0.5	0.53	0.53	0.54	0.56	0.56	0.59	
Fire Alarm	0.16	0.52	1	0.98	1	1	–	–	–	–	
Outlet	0.12	–	–	–	–	–	–	–	–	–	
Light Switch	–	–	–	–	–	–	–	–	–	–	
Router	0.21	–	–	–	–	–	–	–	–	–	
EXIT sign	0.19	0.54	0.63	0.65	0.7	0.68	0.81	–	–	–	
Cardkey Reader	0.13	0.53	0.58	0.6	0.61	0.61	0.63	0.63	0.64	0.66	
Emergency Lights	0.08	0.1	0.08	–	–	–	–	–	–	–	
Fire Extinguisher	–	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.16	0.71	–	–	–	–	–	–	–	–	
Total	0.169	0.458	0.554	0.683	0.71	0.705	0.66	0.595	0.6	0.625	10.146

Table A.2.: Model v1 trained on negatives, evaluated on Day 2 data from Cory Hall

Day 3 - SDH (Trained on Days 1-2 negative predictions)											
	Appearance (average over 100 permutations)										
Asset	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.2	0.37	0.5	0.57	0.58	0.63	–	–	–	–	
Fire Alarm	0.15	0.87	1	1	1	1	1	–	–	–	
Outlet	0.14	0.51	0.5	0.51	0.56	0.55	–	–	–	–	
Light Switch	0.09	0.31	0.54	–	–	–	–	–	–	–	
Router	0.23	0.44	0.47	0.63	0.7	0.74	0.73	–	–	–	
EXIT sign	0.18	0.7	0.79	0.81	0.86	0.84	0.82	0.89	–	–	
Cardkey Reader	0.15	0.36	0.61	0.63	0.64	0.64	0.66	0.66	0.67	–	
Emergency Lights	–	–	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.4	0.51	0.53	0.5	–	–	–	–	–	–	
Fire Alarm Handle	0.26	0.56	–	–	–	–	–	–	–	–	
Total	0.2	0.514	0.618	0.664	0.723	0.733	0.803	0.775	0.67	–	12.562

Table A.3.: Model v2 trained on negatives, evaluated on Day 3 data from Sutardja Dai Hall

Day 3 - Evans Hall (Trained on Days 1-2 negative predictions)											
	Appearance (average over 100 permutations)										
Asset	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	–	–	–	–	–	–	–	–	–	–	
Fire Alarm	0.29	0.68	1	1	1	1	1	1	–	–	
Outlet	0.14	0.52	0.51	0.52	0.57	0.56	0.58	0.55	0.61	0.63	
Light Switch	0.13	0.41	0.44	0.49	0.5	0.52	–	–	–	–	
Router	0.4	0.4	0.54	–	–	–	–	–	–	–	
EXIT sign	0.18	0.63	0.72	0.74	0.79	0.77	0.75	0.82	0.84	–	
Cardkey Reader	–	–	–	–	–	–	–	–	–	–	
Emergency Lights	0.17	0.19	0.17	0.2	0.39	0.33	0.42	0.39	0.41	0.36	
Fire Extinguisher	0.07	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.25	0.57	0.68	0.77	0.78	0.79	0.78	–	–	–	
Total	0.204	0.486	0.58	0.62	0.672	0.662	0.706	0.69	0.62	0.495	9.973

Table A.4.: Model v2 trained on negatives, evaluated on Day 3 data from Evans Hall

Day 4 - Cory Hall (Trained on Days 1-3 negative predictions)											
	Appearance (average over 100 permutations)										
Asset	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.18	0.55	0.52	0.58	0.64	0.68	0.79	0.74	0.74	0.79	
Fire Alarm	0.28	1	1	1	1	1	1	1	1	1	
Outlet	0.05	0.62	0.61	0.62	0.67	0.66	0.68	0.65	0.71	0.73	
Light Switch	0.12	0.54	0.57	0.62	0.63	0.65	–	–	–	–	
Router	0.14	0.4	0.54	0.52	0.55	0.57	0.6	0.66	0.68	–	
EXIT sign	0.23	0.68	0.77	0.79	0.84	0.82	–	–	–	–	
Cardkey Reader	0.3	0.6	0.65	0.67	0.68	0.68	0.7	0.7	0.71	0.73	
Emergency Lights	0.19	0.26	0.24	–	–	–	–	–	–	–	
Fire Extinguisher	0.15	0.59	0.63	0.65	–	–	–	–	–	–	
Fire Alarm Handle	0.16	0.68	0.69	0.87	–	–	–	–	–	–	
Total	0.18	0.592	0.622	0.702	0.716	0.723	0.754	0.75	0.768	0.813	8.641

Table A.5.: Model v3 trained on negatives, evaluated on Day 4 data from Cory Hall

A.2. Train on Positives and Negatives

The following 4 detailed tables, A.6-A.9, pertain to the training scheme used where the model weights trained using both positive and negative samples from the previous day.

Day 2 - Cory Hall (Trained on Day 1 positive + negative samples)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.28	0.42	0.56	0.6	0.61	0.62	0.62	0.64	0.63	0.63	
Fire Alarm	0.17	0.62	1	1	1	1	–	–	–	–	
Outlet	0.12	–	–	–	–	–	–	–	–	–	
Light Switch	–	–	–	–	–	–	–	–	–	–	
Router	0.2	–	–	–	–	–	–	–	–	–	
EXIT sign	0.16	0.59	0.68	0.69	0.75	0.78	0.81	–	–	–	
Cardkey Reader	0.15	0.58	0.64	0.68	0.71	0.7	0.73	0.68	0.71	0.72	
Emergency Lights	0.11	0.14	0.15	–	–	–	–	–	–	–	
Fire Extinguisher	–	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.14	0.76	–	–	–	–	–	–	–	–	
Total	0.166	0.518	0.606	0.7425	0.768	0.775	0.72	0.66	0.67	0.675	12.616

Table A.6.: Model v1 trained on positives **and** negatives, evaluated on Day 2 data from Cory Hall

Day 3 - SDH (Trained on Days 1-2 positive + negative samples)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.18	0.39	0.52	0.58	0.59	0.64	–	–	–	–	
Fire Alarm	0.13	1	1	1	1	1	1	–	–	–	
Outlet	0.17	0.56	0.59	0.61	0.66	0.65	–	–	–	–	
Light Switch	0.11	0.35	0.57	–	–	–	–	–	–	–	
Router	0.16	0.49	0.52	0.59	0.65	0.71	0.77	–	–	–	
EXIT sign	0.21	0.74	0.79	0.85	0.84	0.88	0.89	0.94	–	–	
Cardkey Reader	0.25	0.45	0.65	0.68	0.69	0.69	0.71	0.7	0.73	–	
Emergency Lights	–	–	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.29	0.57	0.56	0.58	–	–	–	–	–	–	
Fire Alarm Handle	0.31	1	–	–	–	–	–	–	–	–	
Total	0.201	0.617	0.65	0.699	0.738	0.762	0.843	0.82	0.73	–	13.782

Table A.7.: Model v2 trained on positives **and** negatives, evaluated on Day 3 data from Sutardja Dai Hall (SDH)

Day 3 - Evans Hall (Trained on Days 1-2 positive + negative samples)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	–	–	–	–	–	–	–	–	–	–	
Fire Alarm	0.24	1	1	1	1	1	1	1	–	–	
Outlet	0.11	0.56	0.54	0.55	0.58	0.56	0.55	0.57	0.59	0.63	
Light Switch	0.12	0.46	0.52	0.51	0.57	0.59	–	–	–	–	
Router	0.14	0.31	0.51	–	–	–	–	–	–	–	
EXIT sign	0.17	0.73	0.71	0.78	0.76	0.78	0.79	0.83	0.86	–	
Cardkey Reader	–	–	–	–	–	–	–	–	–	–	
Emergency Lights	0.09	0.18	0.21	0.24	0.34	0.35	0.41	0.41	0.44	0.42	
Fire Extinguisher	0.05	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.22	0.71	0.83	0.81	0.8	0.8	0.81	–	–	–	
Total	0.143	0.564	0.617	0.648	0.675	0.68	0.712	0.703	0.63	0.525	12.108

Table A.8.: Model v2 trained on positives **and** negatives, evaluated on Day 3 data from Evans Hall

Day 4 - Cory Hall (Trained on Days 1-3 positive + negative samples)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.22	0.65	0.67	0.68	0.69	0.7	0.68	0.71	0.73	0.79	
Fire Alarm	0.18	1	1	1	0.99	1	1	1	1	1	
Outlet	0.14	0.65	0.64	0.64	0.67	0.71	0.69	0.7	0.69	0.73	
Light Switch	0.1	0.62	0.64	0.66	0.67	0.66	–	–	–	–	
Router	0.2	0.41	0.52	0.55	0.57	0.58	0.62	0.67	0.66	–	
EXIT sign	0.14	0.69	0.78	0.77	0.82	0.86	–	–	–	–	
Cardkey Reader	0.07	0.64	0.67	0.63	0.68	0.66	0.71	0.73	0.75	0.77	
Emergency Lights	0.15	0.26	0.29	–	–	–	–	–	–	–	
Fire Extinguisher	0.11	0.61	0.64	0.65	–	–	–	–	–	–	
Fire Alarm Handle	0.17	1	1	1	–	–	–	–	–	–	
Total	0.148	0.653	0.685	0.731	0.727	0.739	0.74	0.762	0.766	0.823	10.229

Table A.9.: Model v3 trained on positives **and** negatives, evaluated on Day 4 data from Cory Hall

A.3. All Memory Retained

The following 5 detailed tables, A.10-A.14, pertain to the training scheme used where the model weights were trained on both positive and negative samples from the previous day, and the short-term memory was never wiped.

Day 1 - Cory Hall (Trained on Day 0, Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.7	0.71	0.72	0.75	0.78	0.74	0.73	0.72	0.75	0.74	
Fire Alarm	0.81	0.85	0.92	0.93	0.96	0.94	0.96	0.94	0.95	0.98	
Outlet	0.36	0.41	0.54	0.57	0.63	0.63	0.59	0.58	0.6	0.62	
Light Switch	0.31	0.44	0.46	0.47	0.46	0.53	0.51	0.59	0.6	0.63	
Router	0.62	0.65	0.69	0.68	–	–	–	–	–	–	
EXIT sign	0.53	0.63	0.71	0.74	0.76	0.77	0.78	–	–	–	
Cardkey Reader	0.41	0.57	0.62	0.61	0.65	–	–	–	–	–	
Emergency Lights	0.32	0.4	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.31	0.32	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.51	0.56	0.57	0.59	–	–	–	–	–	–	
Total	0.488	0.554	0.654	0.668	0.701	0.722	0.714	0.708	0.725	0.743	10.23

Table A.10.: Model v0 trained on Day 0, with memory retained, evaluated on Day 1 data from Cory Hall

Day 2 - Cory Hall (Trained on Day 1 positive + negative samples, Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.68	0.72	0.76	0.75	0.76	0.72	0.77	0.79	0.81	0.77	
Fire Alarm	0.89	0.94	1	1	0.99	1	–	–	–	–	
Outlet	0.52	–	–	–	–	–	–	–	–	–	
Light Switch	–	–	–	–	–	–	–	–	–	–	
Router	0.63	–	–	–	–	–	–	–	–	–	
EXIT sign	0.66	0.69	0.68	0.64	0.73	0.76	0.79	–	–	–	
Cardkey Reader	0.57	0.62	0.66	0.71	0.74	0.75	0.73	0.72	0.72	0.7	
Emergency Lights	0.31	0.44	0.42	–	–	–	–	–	–	–	
Fire Extinguisher	–	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.54	0.76	–	–	–	–	–	–	–	–	
Total	0.6	0.695	0.704	0.775	0.805	0.808	0.763	0.755	0.765	0.735	12.752

Table A.11.: Model v1 trained on positives **and** negatives, with memory retained, evaluated on Day 2 data from Cory Hall

Day 3 - SDH (Trained on Days 1-2 positive + negative samples, Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.48	0.49	0.57	0.62	0.64	0.68	–	–	–	–	
Fire Alarm	0.91	1	1	1	1	1	1	–	–	–	
Outlet	0.49	0.58	0.62	0.65	0.67	0.69	–	–	–	–	
Light Switch	0.36	0.45	0.57	–	–	–	–	–	–	–	
Router	0.56	0.59	0.62	0.69	0.75	0.76	0.78	–	–	–	
EXIT sign	0.41	0.54	0.59	0.55	0.63	0.68	0.73	0.78	–	–	
Cardkey Reader	0.55	0.59	0.64	0.69	0.72	0.67	0.7	0.76	0.75	–	
Emergency Lights	–	–	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.39	0.47	0.46	0.48	–	–	–	–	–	–	
Fire Alarm Handle	0.68	0.95	–	–	–	–	–	–	–	–	
Total	0.537	0.629	0.634	0.669	0.735	0.747	0.803	0.77	0.75	–	11.253

Table A.12.: Model v2 trained on positives **and** negatives, with memory retained, evaluated on Day 3 data from Sutardja Dai Hall (SDH)

Day 3 - Evans Hall (Trained on Days 1-2 positive + negative samples, Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	–	–	–	–	–	–	–	–	–	–	
Fire Alarm	0.94	1	1	1	1	1	1	1	–	–	
Outlet	0.41	0.57	0.56	0.57	0.54	0.56	0.57	0.59	0.55	0.63	
Light Switch	0.42	0.47	0.5	0.56	0.59	0.54	–	–	–	–	
Router	0.29	0.34	0.36	–	–	–	–	–	–	–	
EXIT sign	0.47	0.66	0.73	0.75	0.78	0.79	0.81	0.82	0.84	–	
Cardkey Reader	–	–	–	–	–	–	–	–	–	–	
Emergency Lights	0.2	0.23	0.26	0.28	0.31	0.35	0.4	0.39	0.42	0.41	
Fire Extinguisher	0.52	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.7	0.74	0.78	0.8	0.81	0.79	0.81	–	–	–	
Total	0.494	0.573	0.599	0.66	0.672	0.672	0.718	0.7	0.603	0.52	10.067

Table A.13.: Model v2 trained on positives **and** negatives, with memory retained, evaluated on Day 3 data from Evans Hall

Day 4 - Cory Hall (Trained on Days 1-3 positive + negative samples, Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.62	0.66	0.68	0.71	0.69	0.71	0.7	0.72	0.71	0.73	
Fire Alarm	0.86	0.97	1	1	1	0.98	1	1	1	0.99	
Outlet	0.43	0.67	0.65	0.66	0.63	0.68	0.66	0.72	0.7	0.72	
Light Switch	0.36	0.6	0.63	0.63	0.66	0.65	–	–	–	–	
Router	0.46	0.51	0.56	0.58	0.59	0.62	0.65	0.66	0.65	–	
EXIT sign	0.46	0.58	0.74	0.75	0.77	0.83	–	–	–	–	
Cardkey Reader	0.66	0.7	0.73	0.74	0.75	0.74	0.76	0.75	0.75	0.74	
Emergency Lights	0.24	0.41	0.43	–	–	–	–	–	–	–	
Fire Extinguisher	0.31	0.57	0.63	0.64	–	–	–	–	–	–	
Fire Alarm Handle	0.88	0.99	0.98	1	–	–	–	–	–	–	
Total	0.528	0.666	0.703	0.746	0.727	0.744	0.754	0.77	0.762	0.795	12.008

Table A.14.: Model v3 trained on positives **and** negatives, with memory retained, evaluated on Day 4 data from Cory Hall

A.4. Weighted Memory Retained

The following 4 detailed tables, A.15-A.18, pertain to the training scheme used where the model weights were trained on both positive and negative samples from the previous day, and the short-term memory was never wiped. The performance for Day 1 matches that in Table A.10.

Day 2 - Cory Hall (Trained on Day 1 positive + negative samples, Weighted Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.67	0.74	0.75	0.74	0.75	0.73	0.78	0.81	0.83	0.83	
Fire Alarm	0.9	0.94	1	1	1	1	–	–	–	–	
Outlet	0.53	–	–	–	–	–	–	–	–	–	
Light Switch	–	–	–	–	–	–	–	–	–	–	
Router	0.64	–	–	–	–	–	–	–	–	–	
EXIT sign	0.66	0.7	0.72	0.71	0.72	0.77	0.8	–	–	–	
Cardkey Reader	0.57	0.64	0.67	0.72	0.75	0.76	0.72	0.71	0.74	0.76	
Emergency Lights	0.34	0.46	0.48	–	–	–	–	–	–	–	
Fire Extinguisher	–	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.55	0.77	–	–	–	–	–	–	–	–	
Total	0.608	0.708	0.724	0.793	0.805	0.815	0.767	0.76	0.785	0.795	13.211

Table A.15.: Model v1 trained on positives **and** negatives, with down-weighted memory retained, evaluated on Day 2 data from Cory Hall

Day 3 - SDH (Trained on Days 1-2 positive + negative samples, Weighted Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.52	0.55	0.59	0.67	0.67	0.69	–	–	–	–	
Fire Alarm	0.95	1	1	1	1	1	1	–	–	–	
Outlet	0.51	0.59	0.64	0.65	0.68	0.71	–	–	–	–	
Light Switch	0.37	0.49	0.56	–	–	–	–	–	–	–	
Router	0.54	0.63	0.66	0.71	0.77	0.77	0.77	–	–	–	
EXIT sign	0.47	0.53	0.57	0.61	0.63	0.67	0.72	0.76	–	–	
Cardkey Reader	0.58	0.64	0.71	0.73	0.75	0.77	0.78	0.78	0.79	–	
Emergency Lights	–	–	–	–	–	–	–	–	–	–	
Fire Extinguisher	0.42	0.52	0.56	0.55	–	–	–	–	–	–	
Fire Alarm Handle	0.69	0.95	–	–	–	–	–	–	–	–	
Total	0.561	0.656	0.662	0.703	0.75	0.768	0.818	0.77	0.79	–	13.338

Table A.16.: Model v2 trained on positives **and** negatives, with down-weighted memory retained, evaluated on Day 3 data from Sutardja Dai Hall (SDH)

Day 3 - Evans Hall (Trained on Days 1-2 positive + negative samples, Weighted Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	–	–	–	–	–	–	–	–	–	–	
Fire Alarm	0.95	1	1	1	1	1	1	1	–	–	
Outlet	0.54	0.64	0.66	0.67	0.67	0.66	0.68	0.67	0.68	0.67	
Light Switch	0.44	0.53	0.56	0.58	0.59	0.57	–	–	–	–	
Router	0.31	0.36	0.42	–	–	–	–	–	–	–	
EXIT sign	0.52	0.68	0.74	0.76	0.79	0.82	0.82	0.83	0.83	–	
Cardkey Reader	–	–	–	–	–	–	–	–	–	–	
Emergency Lights	0.33	0.35	0.56	0.53	0.54	0.55	0.54	0.56	0.57	0.55	
Fire Extinguisher	0.56	–	–	–	–	–	–	–	–	–	
Fire Alarm Handle	0.86	0.88	0.91	0.93	0.92	0.95	0.94	–	–	–	
Total	0.564	0.634	0.693	0.745	0.752	0.758	0.796	0.765	0.693	0.61	12.194

Table A.17.: Model v2 trained on positives **and** negatives, with down-weighted memory retained, evaluated on Day 3 data from Evans Hall

Day 4 - Cory Hall (Trained on Days 1-3 positive + negative samples, Weighted Memory Retained)											
	Appearance (average over 100 reps)										
Category	1	2	3	4	5	6	7	8	9	10	Loss
Fire Sprinkler	0.65	0.69	0.73	0.75	0.74	0.76	0.75	0.76	0.77	0.77	
Fire Alarm	0.98	1	1	1	1	1	1	0.99	1	0.99	
Outlet	0.41	0.71	0.72	0.73	0.73	0.74	0.76	0.77	0.74	0.75	
Light Switch	0.38	0.62	0.61	0.64	0.65	0.64	–	–	–	–	
Router	0.51	0.67	0.66	0.71	0.72	0.73	0.75	0.74	0.75	–	
EXIT sign	0.49	0.62	0.74	0.75	0.77	0.8	–	–	–	–	
Cardkey Reader	0.68	0.82	0.83	0.82	0.85	0.86	0.83	0.85	0.84	0.83	
Emergency Lights	0.34	0.51	0.52	–	–	–	–	–	–	–	
Fire Extinguisher	0.43	0.62	0.64	0.65	–	–	–	–	–	–	
Fire Alarm Handle	0.92	0.99	1	1	–	–	–	–	–	–	
Total	0.579	0.725	0.745	0.783	0.78	0.79	0.818	0.822	0.82	0.835	12.815

Table A.18.: Model v3 trained on positives **and** negatives, with down-weighted memory retained, evaluated on Day 4 data from Cory Hall