Supplementary Material: Consistency driven Sequential Transformers Attention Model for Partially Observable Scenes

Samrudhdhi B. Rangrej[a], Chetan L. Srinidhi[b], James J. Clark[a]
[a]McGill University, Canada. [b]Sunnybrook Research Institute, University of Toronto, Canada.
samrudhdhi.rangrej@mail.mcgill.ca, chetan.srinidhi@utoronto.ca, james.clark1@mcgill.ca

We organize the supplementary material as follows. We include additional results in Section 1, additional visualization in Figure 5-6, and a pseudo code for training our STAM in Algorithm 1.

1. Additional Results

1.1. Analysis of Consistency Loss with Baseline Attention Policies

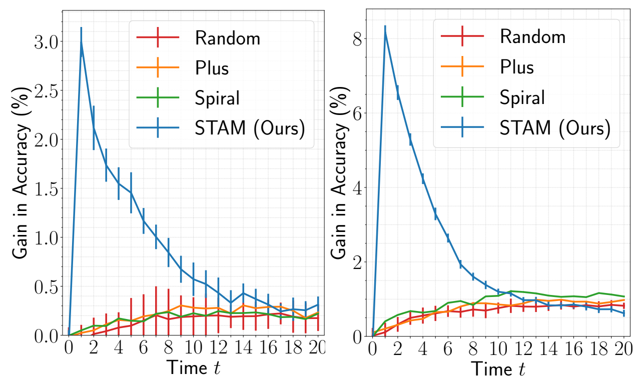
In the main paper, we analyzed the gain in accuracy of STAM when the proposed consistency loss (Equation 3 in the main paper) is included in the training objectives. Here, we analyze the same for the agents with baseline attention policies, namely, the Random, the Plus, and the Spiral. We train baseline agents with and without the proposed consistency objective and plot the difference in their accuracy in Figure 1. We observe that the consistency training objective yields a positive gain in the accuracy for all baseline agents.

Furthermore, the gain achieved with learned policy (i.e., STAM) is higher than the heuristics-based baseline policies. The gain in accuracy is highest for STAM as it learns to attend to the most discriminative glimpses early in time. These results align with the recent findings showing that minimizing the distance between the predictions made from two views of the same image improves model performance the most when the views optimally share the task-specific information [1].

1.2. Effect of Glimpse Size

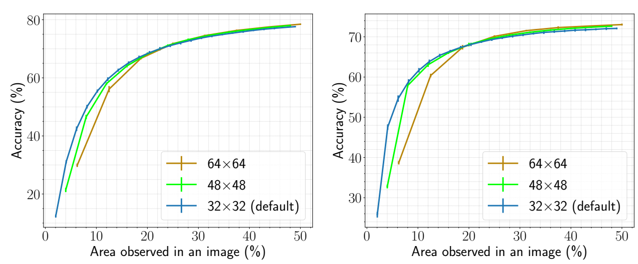
We compare the performance of our agents with glimpses of sizes 32×32, 48×48, and 64×64. To extract the non-overlapping glimpses, we resize the image to 224×224, 240×240, and 256×256 for the three glimpse sizes stated above, respectively.

For the image-size 224×224, we use the teacher models as discussed in the main paper. To train teacher models for images of sizes 240×240 and 256×256, we finetune the pretrained DeiT[1] on images of respective sizes, following the procedure suggested by Touvron *et al.* [2]. We train all agents following the same experimental setup discussed in the main paper, except for the following. We train the agents for image sizes 240×240 and 256×256 using batch sizes of 2000 and 1600, and they observe a maximum of 16 and



(a) (b)

Figure 1. Comparison of gain in accuracy of various baseline agents with inclusion of consistency loss in their training objectives. (a) ImageNet; (b) fMoW. Results for the Random and STAM are presented as mean $\pm$ std computed across ten independent runs.



(a) (b)

Figure 2. Accuracy of STAM with different glimpse sizes presented as a function of % area observed in an image (a) ImageNet; (b) fMoW. The results are presented as mean$\pm 5\times$std computed across ten independent runs.

[1] https://github.com/facebookresearch/deit

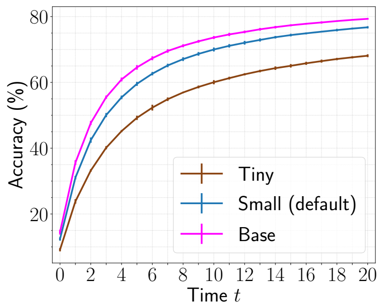


Figure 3. Accuracy of STAM with core of different capacity. We compare DeiT$^{\mathcal{D}}$-Tiny, DeiT$^{\mathcal{D}}$-Small, DeiT$^{\mathcal{D}}$-Base architectures for the core module. The results are presented as mean$\pm 5\times$std computed across ten independent runs.

7 glimpses per image.

As the glimpse and the image sizes are different, we compare the accuracy of the three agents as a function of the area observed in the image (see Figure 2). Initially, when an area observed in an image is less than 20%, the agent with smaller glimpses achieves higher accuracy than the agent with larger glimpses. The reason is that the agent explores more regions using smaller glimpses than the larger ones while sensing the same amount of area. Once the agents have observed sufficient informative regions (nearly 20% of the total image area), their performance converges. We use glimpse size 32×32 with image size 224×224 as our default setting.

1.3. Effect of Model Capacity

To study the effect of model capacity on the performance, we compare DeiT$^{\mathcal{D}}$-Tiny, DeiT$^{\mathcal{D}}$-Small, and DeiT$^{\mathcal{D}}$-Base architectures as the core of our agent. The three agents are trained using the same procedure as discussed in the main paper expect for the following. We train agent with DeiT$^{\mathcal{D}}$-Base core using batch size of 512. We use pretrained DeiT$^{\mathcal{D}}$of respective capacity as the teacher model. Results for ImageNet are presented in Figure 3. We observe increasing accuracy with increasing model capacity. However, training an agent with DeiT$^{\mathcal{D}}$-Base is computationally expensive. To achieve a good trade-off between efficiency and accuracy, we use DeiT$^{\mathcal{D}}$-Small as a default architecture for our agent.

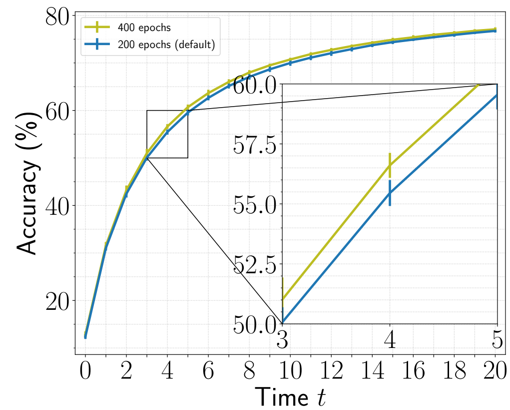


Figure 4. Accuracy of STAM on ImageNet when trained for 200 and 400 epochs. The results are presented as mean$\pm 5\times$std computed across ten independent runs.

1.4. Longer Training on ImageNet

We demonstrate that longer training improves the performance of STAM on ImageNet. We compare performance of STAM trained for 200 and 400 epochs in Figure 4. When STAM is allowed to observe only five glimpses, longer training yields 1.15% improvement in the accuracy. In contrast, we observe overfitting and reduced performance with longer training on fMoW.

References

[1] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? In *Advances in Neural Information Processing Systems*, volume 33, pages 6827–6839, 2020. 1

[2] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357, 2021. 1

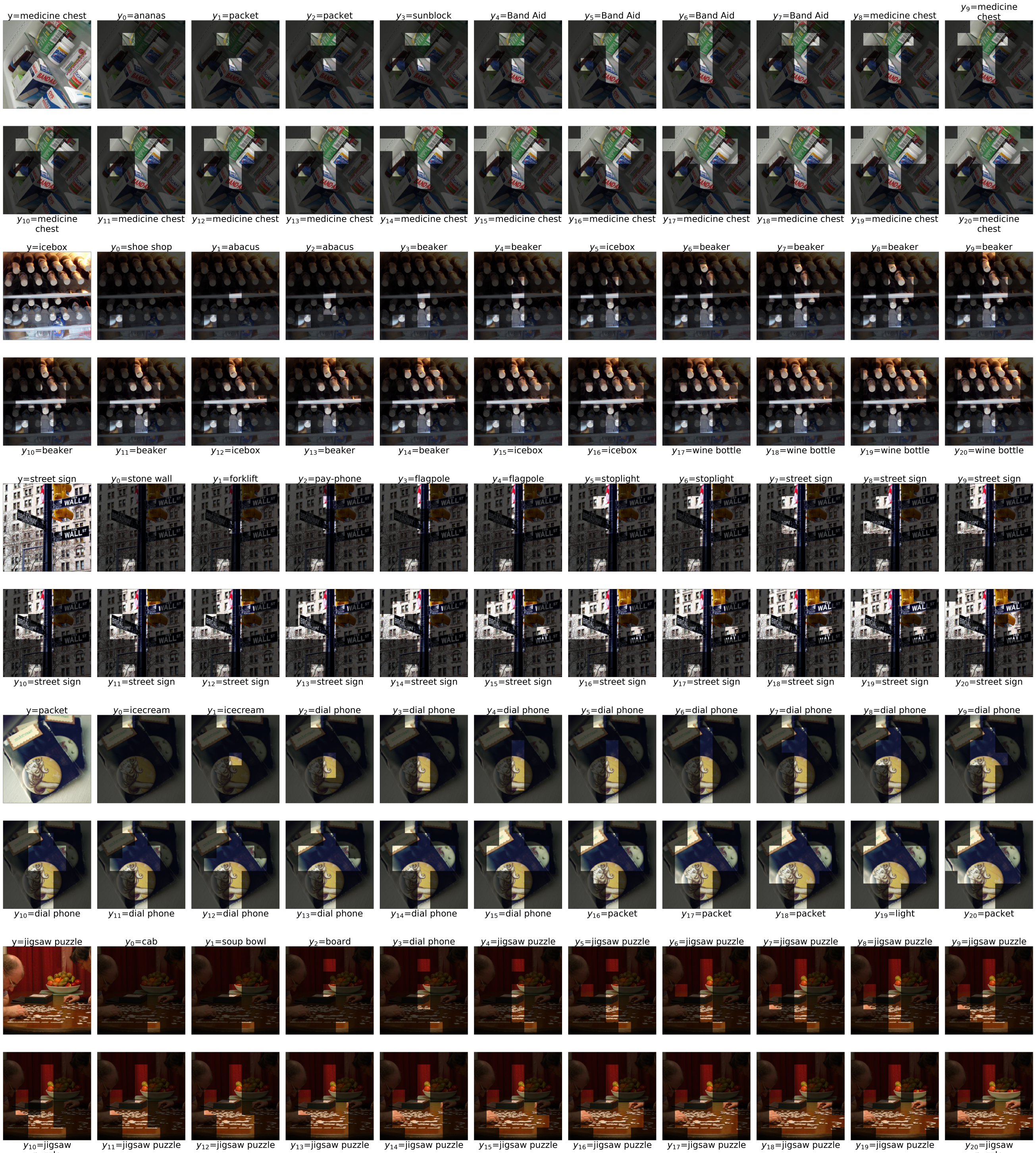


Figure 5. Glimpses selected by STAM on example images from the ImageNet dataset and the predicted labels. Complete images are shown for reference only. Note that STAM does not observe the complete image. Ground truth labels are displayed above complete images.

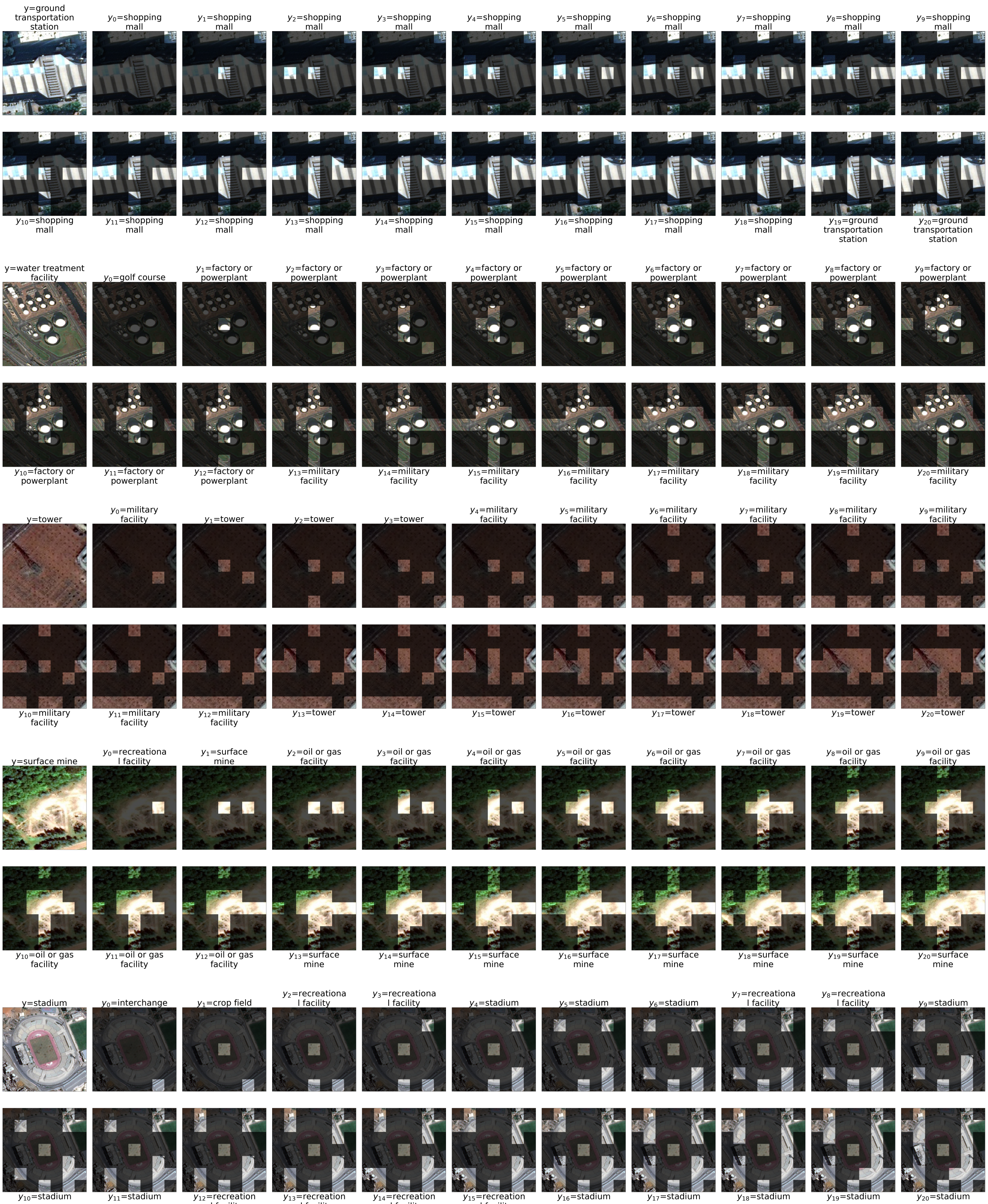


Figure 6. Glimpses selected by STAM on example images from the fMoW dataset and the predicted labels. Complete images are shown for reference only. Note that STAM does not observe the complete image. Ground truth labels are displayed above complete images.

Algorithm 1 Pseudo code for training our Sequential Transformers Attention Model (STAM)

```
'''
Inputs:
    X = complete image X
    y = ground truth for X
'''
def process_one_batch(X,y):
    # STAM collects series of T glimpses from X
    # Parameters of STAM are updated after each additional glimpse
    q = step_one(X)
    l_t = initial_random_location() # Initial glimpse should be captured at a random location
    g_t = extract_glimpse(X, l_t)
    g_upto_t = [g_t]                # A list of all glimpses
    l_upto_t = [l_t]                # A list of all glimpse locations
    for t in range(T):
        # Perform step 2
        p_g_t, p_d_t, V_t, pi_of_l_tplus1, l_tplus1 = step_two(g_upto_t, l_upto_t)
        # Extract one additional glimpse and append it to previous glimpses
        g_tplus1 = extract_glimpse(X, l_tplus1)
        g_upto_t.append(g_tplus1)
        l_upto_t.append(l_tplus1)
        # Perform step 3
        p_tplus1, V_tplus1 = step_three(g_upto_t, l_upto_t)
        # Evaluate losses
        loss = evaluate_losses(y, q, p_g_t, p_d_t, V_t, pi_of_l_tplus1, p_tplus1, V_tplus1)
        # Update model parameters
        loss.backward()
        optimizer.step()

def step_one(X):
    # Teacher predicts soft pseudo-label from a complete image
    with no_grad():
        q = teacher(X)
    return q

def step_two(g_upto_t, l_upto_t):
    # STAM predicts class distributions, state value, attention policy and next glimpse location
    f_g_t, f_d_t, s_t = core(g_upto_t, l_upto_t)          # Core
    p_g_t, p_d_t = classifiers(f_g_t, f_d_t)              # Classifiers
    V_t = critic(s_t)                                     # Critic
    l_unobserved = find_unobserved_locations(l_upto_t)    # Find yet unobserved locations
    pi_of_l_tplus1, l_tplus1 = actor(s_t, l_unobserved)   # Actor
    return p_g_t, p_d_t, V_t, pi_of_l_tplus1, l_tplus1

def step_three(g_upto_tplus1, l_upto_tplus1):
    # STAM computes ensemble class distribution and the state value one step ahead
    with no_grad():
        f_g_tplus1, f_d_tplus1, s_tplus1 = core(g_upto_tplus1, l_upto_tplus1)     # Core
        p_g_tplus1, p_d_tplus1 = classifiers(f_g_tplus1, f_d_tplus1)              # Classifiers
        p_tplus1 = (p_g_tplus1 + p_d_tplus1)/2                                    # Ensemble
        V_tplus1 = critic(s_tplus1)                                               # Critic
    return p_tplus1, V_tplus1

def evaluate_losses(y, q, p_g_t, p_d_t, V_t, pi_of_l_tplus1, p_tplus1, V_tplus1):
    # Evaluate losses
    L_sup = cross_entropy(p_g_t, y)                           # Supervised classification loss
    L_consist = kl_div(p_d_t, q)                              # Consistency loss
    R_tplus1 = - kl_div(p_tplus1, q)                                          # Reward
    L_critic = l1_loss(V_t, R_tplus1 + V_tplus1)                              # Critic loss
    L_actor = pi_of_l_tplus1 * (V_t-(R_tplus1 + V_tplus1)).detach()           # Actor loss
    L_final = (L_sup + L_consist)/2 + L_critic + L_actor                      # Final loss
    return L_final
```