# A DETAILS FOR METHOD

## A.1 BOED PSEUDOCODE

**Algorithm 1** Outline of optimal sequence generation.

**procedure** GENERATENOGS
  Initialise Adam. ▷ Train AVP-CNN.
  **for** each AVP-CNN training steps **do**
    $\lambda \leftarrow \lambda + \text{Adam}(\frac{\partial}{\partial \lambda}\mathcal{L}_\lambda)$
  **for** entry $i$ in $1, \ldots, N_{img}$ **do** ▷ Create objects needed for $r_{\text{img}}$.
    $\mathcal{I}_i \sim$ generative model.
  Perform PCA on $\mathcal{I}_{1,\ldots,N_{img}}$ and save results.
  **for** $i$ in $1, \ldots, N_{seq}$ **do** ▷ Perform BOED (using AVP-CNN and $r_{\text{img}}$).
    **for** $t$ in $1, \ldots, T$ **do**
      $l_t^i \leftarrow \text{SELECTLOCATION}(\mathbf{y}_{1:t-1}^i, l_{1:t-1}^i)$
      $\mathbf{y}_t^i \leftarrow f_{\text{fovea}}(\mathcal{I}_i, l_t^i)$
  **return** $l_{1:T}^{1:N_{seq}}$ ▷ Return near-optimal sequences.

.

Algorithm 1 presents pseudocode for the full procedure to generate near-optimal glimpse sequences, including training the AVP-CNN and constructing an empirical image distribution. Once this been called, and near-optimal glimpse sequences have been generated, they can be used repeatedly to train hard attention neural networks. This involves initializing a hard attention network (as in Fig. 1) and training it as described in Section 5.

When performing BOED, Algorithm 1 references Algorithm 2 at each BOED time step. This selects a near-optimal glimpse location for a single timestep. Repeating this for $t = 1, \ldots, T$ yields a near-optimal glimpse sequence for a particular image. Figure 7 provides a visualisation of the various stages of this algorithm.

**Algorithm 2** BOED pipeline to select $l_t$. This roughly follows the three-step procedure given in Section 4: $N$ images are sampled in line 2; these are used to estimate expected posterior entropies for each $l_t \in L$ in lines 3-6. Finally, the minimising value of $l_t$ is found on line 7.

1: **procedure** SELECTLOCATION($\mathbf{y}_{1:t-1}, l_{1:t-1}$)
2: $\mathcal{I}^{(1)}, \ldots, \mathcal{I}^{(N)} \sim r_{\text{img}}(\mathcal{I}|\mathbf{y}_{1:t-1}, l_{1:t-1})$ ▷ stochastic image completion
3: **for** $l_t \in L$ **do** ▷ grid search for $l_t$
4: **for** $n \leftarrow 1, \ldots, N$ **do**
5: $\text{PE}_{l_t}^{(n)} \leftarrow \mathcal{H}\left[g_{\text{AVP}}(\theta|f_{\text{fovea}}(\mathcal{I}^{(n)}, l_{1:t}), l_{1:t})\right]$ ▷ estimate posterior entropy
6: $\text{EPE}_{l_t} \leftarrow \frac{1}{N}\sum_{n=1}^{N} \text{PE}_{l_t}^{(n)}$ ▷ average over Monte Carlo samples
7: $l_t^* \leftarrow \text{argmin}_{l_t} \text{EPE}_{l_t}$
8: **return** $l_t^*$

.

## A.2 EIG ESTIMATOR IN EQ. (5)

This section derives the form of the expected posterior entropy presented in Eq. (5). Starting from Eq. (3), we introduce an inner expectation over $\mathcal{I}$:

$$\text{EPE}_{\mathbf{y}_{1:t-1}, l_{1:t-1}}(l_t) = \mathbb{E}_{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, l_{1:t})}\left[\mathcal{H}\left[p(\theta|\mathbf{y}_{1:t}, l_{1:t})\right]\right] \tag{9}$$

$$= \mathbb{E}_{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, l_{1:t})}\mathbb{E}_{p(\mathcal{I}|\mathbf{y}_{1:t}, l_{1:t})}\left[\mathcal{H}\left[p(\theta|\mathbf{y}_{1:t}, l_{1:t})\right]\right] \tag{10}$$

$$= \mathbb{E}_{p(\mathbf{y}_t, \mathcal{I}|\mathbf{y}_{1:t-1}, l_{1:t})}\left[\mathcal{H}\left[p(\theta|\mathbf{y}_{1:t}, l_{1:t})\right]\right]. \tag{11}$$

Since, according to the probabilistic model defined in Eq. (4), each $\mathbf{y}_i$ is a deterministic function of $\mathcal{I}$ and $l_i$ we can substitute $\mathbf{y}_{1:t}$ using the definition $\mathbf{y}_{1:t} = f_{\text{fovea}}(\mathcal{I}, l_{1:t}) =$

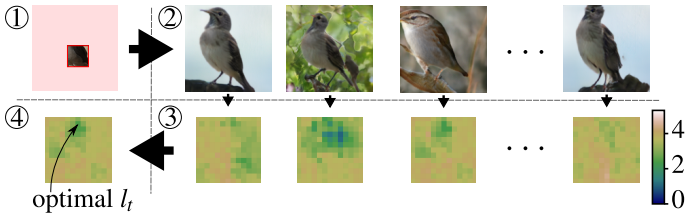


Figure 7: BOED procedure to select $l_2$ given $\mathbf{y}_1$ and $l_1$, corresponding to Algorithm 2 with $t = 2$. Panel 1 shows a visualisation of the single previous glimpse, $\mathbf{y}_1$, taken at $l_1$, with all unobserved image pixels displayed in pink. Panel 2 shows $\mathcal{I}^{(1)}, \ldots, \mathcal{I}^{(N)}$, Monte Carlo samples of the full image drawn on line 2 of Algorithm 2. For each of these images, panel 3 shows a heatmap. Each heatmap plots the estimated posterior entropy after conditioning on $\mathbf{y}_t = f_{\text{fovea}}(\mathcal{I}, l^*)$ and $l_t = l^*$ for each $l^* \in L$, plotted against the $x$ and $y$ coordinates of $l^*$. These posterior entropies are estimated on line 5, and then averaged over the Monte Carlo samples on line 6, giving the *expected* posterior entropies displayed by a heatmap in panel 4. Finally, $l_t$ is chosen to minimise $\text{EPE}_{l_t}$.

$\{f_{\text{fovea}}(\mathcal{I}, l_1), \ldots, f_{\text{fovea}}(\mathcal{I}, l_t)\}$.

$$\text{EPE}_{\mathbf{y}_{1:t-1}, l_{1:t-1}}(l_t) = \mathbb{E}_{p(\mathbf{y}_t, \mathcal{I} | \mathbf{y}_{1:t-1}, l_{1:t})} \left[ \mathcal{H} \left[ p(\theta | f_{\text{fovea}}(\mathcal{I}, l_{1:t}), l_{1:t}) \right] \right] \tag{12}$$

$$= \mathbb{E}_{p(\mathcal{I} | \mathbf{y}_{1:t-1}, l_{1:t-1})} \left[ \mathcal{H} \left[ p(\theta | f_{\text{fovea}}(\mathcal{I}, l_{1:t}), l_{1:t}) \right] \right] \tag{13}$$

as presented in Eq. (5).

### A.3 Stochastic image completion

This section provides more detail on our method for stochastic image completion and the alternatives we considered, as discussed in Section 4. Three sampling mechanisms for $r_{\text{img}}(\mathcal{I} | \mathbf{y}_{1:t-1}, l_{1:t-1})$ were compared qualitatively, based on both the diversity of samples produced and how realistic the samples are (taking into account the previous observations). These were a conditional GAN, HMC in a deep generative model, and the image retrieval-based approach used in our pipeline. We now provide more detail on each.

#### A.3.1 Conditional GAN

We considered a conditional GAN, motivated by recent state-of-the-art performance on conditional image generation problems (Nazeri et al., 2019; Pathak et al., 2016). In particular, we considered using pix2pix (Isola et al., 2017) to map from an embedding of $\mathbf{y}_{1:t}$ and $l_{1:t}$ (similar to that in Fig. 3 but without concatenating the mask, and with unobserved pixels replaced with Gaussian noise instead of zeros) to the completed image. We used the U-net architecture (Ronneberger et al., 2015) with resolution $256 \times 256$. Fig. 8 shows samples when all other hyperparameters were set to the defaults (Isola et al., 2017). We tried varying both the generator architecture and the weight given to the L1-norm, although neither significantly improved the quality of the generated images. Although the GAN can learn to generate realistic images for CelebA-HQ, they have very little diversity when the observations are fixed. An additional issue is that the outputs looked considerably less realistic for the Caltech-UCSD birds dataset.

#### A.3.2 HMC in a deep generative model

We also considered taking a pre-existing, unconditional, GAN (StyleGAN (Karras et al., 2018)) with publicly available weights and using Hamiltonian Monte Carlo (Duane et al., 1987; Hoffman & Gelman, 2014) (HMC) to sample images from it conditioned on the observations. Specifically, we sample from $p(\mathcal{I} | \mathbf{y}_{1:t}, l_{1:t}) \propto p(\mathcal{I}) \prod_{i=1}^{t} p(\mathbf{y}_i | \mathcal{I}, l_i)$, where $p(\mathcal{I})$ is the prior over images defined by the GAN. Deviating from the model specified in Eq. (4), we define $p(\mathbf{y}_i | \mathcal{I}, l_i)$ to be an isotropic Gaussian centred on $f_{\text{fovea}}(\mathcal{I}, l_i)$, rather than a Dirac-delta on $f_{\text{fovea}}(\mathcal{I}, l_i)$. This change is necessary to make inference with HMC feasible. We used an implementation of HMC in Pyro (Bingham et al., 2018), a probabilistic programming language. Samples are shown in the second row of Fig. 8. These were taken using a likelihood with standard deviation 0.4 applied to the normalised pixel values,


Figure 8: Sampled image completions using various techniques. The glimpse locations are marked with red squares, and close-ups of each glimpse are shown to the right of each image. The leftmost column shows the 'true' image from which the observed glimpses are taken and the others show samples conditioned on these glimpses.

**Algorithm 3** SAMPLEIMAGES calls SAMPLEPROPOSAL to select $K_1$ images to load into memory, based on approximations of their importance weights. These are then reweighted using their exact likelihood. $K_2$ images are sampled from the resulting weighted categorical distribution, and returned.

**procedure** SAMPLEPROPOSAL($\mathcal{I}, l_{1:t}$)
  Select relevant weight matrix columns $\tilde{\mathbf{W}} \leftarrow \text{SLICE}(\mathbf{W}, l_{1:t})$
  Select relevant columns of mean vector $\tilde{\mu} \leftarrow \text{SLICE}(\mu, l_{1:t})$
  Reconstruct observations $\hat{\mathbf{y}}_{1:t} \leftarrow \tilde{\mu} + \tilde{\mathbf{W}}^\top \mathbf{W}\mathcal{I}$
  **for** $i = 1, \ldots, N^{\text{data}}$ **do**
    Approximate observed patches $\hat{\mathbf{y}}^i_{1:t} \leftarrow \tilde{\mathbf{W}}^\top \mathbf{z}^i$
    Compute approximate likelihood $w_1^i \leftarrow \mathcal{N}\left(\hat{\mathbf{y}}_{1:t} | \hat{\mathbf{y}}^i_{1:t}, \sigma_q^2\right)$
  $j^{(1)}, \ldots, j^{(K_1)} \leftarrow \text{RESAMPLE}(w_1^1, \ldots, w_1^{N^{\text{data}}})$
  **return** $\{i^{(k)}, w_1^{i^{(k)}}\}$ for $k = 1, \ldots, K_1$

**procedure** SAMPLEIMAGES($\mathcal{I}, l_{1:t}$)
  **Input:** $\mathcal{I}, l_{1:t}$
  $\{i^{(1)}, w_1^{i^{(1)}}\}, \ldots, \{i^{(K_1)}, w_1^{i^{(K_1)}}\} = \text{SAMPLEPROPOSAL}(\mathcal{I}, l_{1:t})$
  **for** $k = 1, \ldots, K_1$ **do**
    Load $\mathcal{I}^{i^{(k)}}$
    $\mathbf{y}^k_{1:t} \leftarrow \text{Glimpse}(\mathcal{I}^{i^{(k)}}, l_{1:t})$
    Compute exact likelihood $p(\mathbf{y}_{1:t} | \mathcal{I}^{i^{(k)}}, l_{1:t}) = \mathcal{N}\left(\mathbf{y}_{1:t} | \mathbf{y}^k_{1:t}, \sigma_p^2\right)$
    Compute weight $w_2^{(k)} \leftarrow \frac{p(\mathbf{y}_{1:t} | \mathcal{I}^{i^{(k)}}, l_{1:t})}{w_1^{i^{(k)}}}$
  $j^{(1)}, \ldots, j^{(K_2)} \leftarrow \text{RESAMPLE}(w_2^1, \ldots, w_2^{K_1})$
  **return** $\mathcal{I}^{j^{(k)}}$ for $k = 1, \ldots, K_2$

which was set to trade-off the 'closeness' of samples to the true image and the feasibility of inference. The integration was performed with 200 steps of size 0.05 and all other parameters set to the defaults. Although the samples mostly look realistic, they suffered from low sample diversity; the HMC chains became stuck in certain modes and did not explore the full posterior. Additionally, sampling was slow: drawing each sample took approximately a minute on a GPU.

### A.3.3 IMAGE RETRIEVAL

Our stochastic image retrieval procedure proceeds as follows. We begin with a large dataset of images $\mathcal{I}^1, \ldots, \mathcal{I}^{N^{\text{data}}}$ independently sampled from $p(\mathcal{I})$ (or an approximation of it). We are then given some observations, $\mathbf{y}_{1:t}, l_{1:t}$ and, roughly speaking, want to approximate $K_2$ samples from $p(\mathcal{I} | \mathbf{y}_{1:t}, l_{1:t})$ using $K_2$ images from the dataset. We begin by assigning each image $i$ in the dataset a weight,

$w_1^i$, which approximates the likelihood $p(\mathbf{y}_{1:t}|\mathcal{I}^i, l_{1:t})$. This approximation is efficiently computed using PCA, as we describe later. We then construct a categorical proposal distribution over these images, where the probability of each is proportional to the weight. We draw $K_1$ samples from this proposal. These images are then retrieved from the database. To make up for the inexact weights computed previously, we reweight these samples using exact likelihoods. These weights are used to compute a categorical distribution over the $K_1$ images. This distribution approximates $p(\mathcal{I}|\mathbf{y}_{1:t}, l_{1:t})$ increasingly well as $N^{\text{data}}$ (the dataset size) and $K_1$ tend to infinity. $K_2$ samples from this distribution are returned.

Principal component analysis allows for a memory-efficient representation of the dataset through a mean image, $\mu$, a low-dimensional vector for each image, $\mathbf{z}^i$, and an orthogonal matrix, $\mathbf{W}$, which transforms from an image, $\mathcal{I}^i$, into the corresponding $\mathbf{z}^i$ as follows:

$$\mathbf{z}^i = \mathbf{W}\mathcal{I}^i \tag{14}$$

Denoting the dimensionality of $\mathbf{z}$ as $L$, the number of images as $N^{\text{data}}$, and the number of pixels per image as $P$, these objects can be stored with memory complexity $\Theta(N^{\text{data}}L + PL)$, compared to $\Theta(N^{\text{data}}P)$ for the entire dataset. Since $\mathbf{W}$ is orthogonal, image $i$ can be approximated using $\mathbf{z}^i$ as

$$\hat{\mathbf{x}}^i = \mathbf{W}^\top \mathbf{z}^i \tag{15}$$

and $\hat{\mathbf{x}} \approx \mathcal{I}$ for large enough $L$. If only certain pixels of the reconstructed image are required, $\mathcal{I}^i_{p_1:p_C}$ these can be obtained efficiently by using $\tilde{\mathbf{W}}^\top$, a matrix made up of rows $p_1$ to $p_C$ of $\mathbf{W}^\top$:

$$\mathcal{I}^i_{p_1:p_C} = \tilde{\mathbf{W}}^\top \mathbf{z}^i. \tag{16}$$

This allows us to construct an approximation of the observed portion with each image in the dataset in a time and memory-efficient manner. Additionally, we found that our proposal was improved when the approximate likelihood was calculated with a reconstruction of the observations as $\hat{\mathbf{y}}_{1:t} = \tilde{\mathbf{W}}^\top \mathbf{W}\mathcal{I}$, rather than the true observations $\mathbf{y}_{1:t}$. Although this requires access to the full true image, this is acceptable as the full image was always available when we carried out our experimental design. Also, since this is only used to calculate the proposal distribution, it should have limited effect on the samples returned after new weights are calculated with the exact likelihood.

The algorithms we use in our experiments vary in the following ways from Algorithm 3. They both expand the effective size of the generated dataset by comparing the observations with a horizontally-flipped version of each dataset image, as well as the original version. Additionally for CelebA-HQ, in order to increase the sample size and impose an invariance to very small translations, the likelihoods (for both the proposal and the exact likelihood) for each image are given by summing the Gaussian likelihoods over a grid of image patches taken at locations close to the observed location. For CUB, a simpler alteration is made to increase the effective sample size: $\sigma_q$ is tuned while creating each proposal distribution so that it will have an effective sample size roughly equal to the number of samples drawn. The attached code contains both variations.

# B EXPERIMENTAL DETAILS

## B.1 AVP-CNN TRAINING

Here, we expand on the training procedure for the AVP-CNN which was described in Section 4. For CelebA-HQ, it was trained to predict each of the 40 attributes simultaneously by outputting a vector parameterising an independent Bernoulli distribution for each, which we found to improve accuracy compared to training a network for each. Similarly for CUB, the AVP-CNN output a vector of probabilities for each of the 312 binary attributes in the dataset, in addition to the 200-dimensional categorical distribution. These outputs were produced by linear mappings (and a sigmoid/softmax) from the same final hidden layer. The predictions for the binary attributes of CUB were not used; this was done purely to improve the training of the classifier.

For both datasets, the AVP-CNN was initialised from weights pretrained on ImageNet, with only the final layer replaced and an additional input channel added (with weights initialised to zero). It was then trained using Adam optimizer with a learning rate of $1 \times 10^{-4}$ and a batch size of 64. Validation was performed and a checkpoint saved every epoch, and the checkpoint which gave the lowest validation cross-entropy was used. This occurred after 183 epochs for the network used to

create near-optimal glimpse sequences on CUB and after $458$ for CelebA-HQ. For both datasets, the training set for the AVP-CNN was chosen to exclude the examples on which Bayesian experimental design was later performed in order to prevent any potential issues with using the AVP-CNN to approximate classification distributions on data which it may be overfitted to.

### B.2 CUB PRETRAINING

As mentioned in the paper, we found a pretraining stage to be important for achieving high accuracy on CUB. During this stage, the parameters of the RNN, glimpse embedder, and classifier were trained, while the location network was not used. After 250 epochs of pretraining, saved parameters from the epoch with highest validation accuracy were used for the next stage of training. These parameters were then frozen while the location network was trained. The only difference between our pretraining loss and our training loss is that pretraining glimpse locations are sampled from some fixed distribution, instead of from the distribution proposed by the location network. As mentioned in the paper, this is a uniform distribution over all $l_t \in L$ for RAM. We now describe the heuristic distribution used to pretrain the other networks. In all our tests, we found that this heuristic gave better performance than the uniform distribution.

The heuristic distribution uses $\text{EPE}_{\emptyset,\emptyset}(l)$ as a measure of the saliency of a location. That is, the expected entropy in the posterior after taking a single glimpse at $l$. Since this is not conditioned on any previous glimpses, it is independent of the image being processed. This is used to create a distribution over locations as:

$$\log p^{\text{loc}}(l) = C - \gamma^{-1} \cdot \text{EPE}_{\emptyset,\emptyset}(l) \tag{17}$$

where $\gamma^{-1}$ is the inverse temperature, which we set to 1. $C = -\log \sum_l \exp\left(-\gamma^{-1} \cdot \text{EPE}_{\emptyset,\emptyset}(l)\right)$ is a constant chosen such that $p^{\text{loc}}$ is a normalised distribution. We estimate $\text{EPE}_{\emptyset,\emptyset}(l)$ for each $l$ using our BOED pipeline.

### B.3 ADDITIONAL BASELINES

In addition to the supervision sequences described in Section 6, we here consider another form of heuristic supervision sequence as a baseline. These were created by sampling glimpse locations independently at each time step from the distribution defined in Eq. (17). We ran experiments with both $\gamma^{-1} = 1$ (denoted PS-H1) and $\gamma^{-1} = 5$ (denoted PS-H5).

Figure 12 shows validation accuracy for these over the course of training. We find that networks trained with PS-H1 and PS-H5 converge quickly: after 240 and 400 iterations respectively, as measured by when they reach within 1% of the highest validation accuracy. This may be due to the simplicity of the policies that the supervision sequences encourage them to learn, with identical and independent distributions over the glimpse location at every time step. Despite their fast convergence, the validation accuracy for these heuristics appears to be almost always lower than that for PS-NOGS throughout training, and never higher by a statistically significant margin.

### B.4 ARCHITECTURAL DETAILS

For all hard attention networks, the classifier is a fully-connected layer mapping from the hidden state to a softmax output and the location network has a single 32-dimensional hidden layer. The GRU hidden dimension is 64 for CelebA-HQ and 1024 for CUB. The glimpse embedder for CelebA-HQ consisted of the following, in order: a $3 \times 3$ convolution to 16 channels with stride 1; a ReLU activation; a $3 \times 3$ convolution to 32 channels with stride 2; a ReLU and $2 \times 2$ max pool; and a linear layer mapping the output of this to the $64$-dimensional RNN input. As mentioned Section 6, the glimpse network for CUB consists of the first 12 convolutional layers of a VGG, and is pretrained on ImageNet (Simonyan & Zisserman, 2014; Deng et al., 2009)

As mentioned, we use a learned baseline to reduce the variance of the REINFORCE gradient estimate. Following Mnih et al. (2014), this is in the form of a linear 'baseline' network which maps from the RNN hidden state to a scalar estimate of the reward.

### B.5 HYPERPARAMETERS

For the Monte Carlo estimation of the expected posterior entropy in Eq. (6), we use $N = 200$ Monte Carlo samples for CelebA-HQ and $N = 100$ for CUB.

**Hard attention network training** All hard attention networks are trained by Adam optimiser (Kingma & Ba, 2014) with a batch size of 64; learning rate $3 \times 10^{-4}$ for CelebA-HQ and $5 \times 10^{-5}$ for CUB; and other hyperparameters set to the recommended defaults (Kingma & Ba, 2014).

**Stochastic image completion** For the stochastic image completion algorithm, we use the following hyperparameters for CelebA-HQ: $K_1 = 1000$; $K_2 = 200$; and $\sigma_p = \sigma_q = 5 \times 2^t$ for each timestep $t$. For CUB, we use: $K_1 = 500$; $K_2 = 100$; $\sigma_p = 80$; and a dynamically adjusted $\sigma_q$ (as descibed in Appendix A.3.3). For both datasets, we use a 256-dimensional latent space for the PCA and $N^{\text{data}} = 1\,500\,000$.

To allow direct comparison between PS-NOGS and the baseline supervision sequences (PS-HGS, PS-H1 and PS-H5), we supervise the same number of training images for each; that is, 600 for tasks on CelebA-HQ and 1000 for CUB classification.

## C ADDITIONAL PLOTS

### C.1 GLIMPSE LOCATIONS

Building on Fig. 4 in the paper, Figures 9 to 11 show the glimpse locations of networks trained with each of RAM and PS-NOGS for all 40 CelebA-HQ attributes.

### C.2 CUB TRAINING

Figure 12 shows the validation accuracy throughout training for CUB, including for the additional baselines from Appendix B.3.

### C.3 SUPERVISION SEQUENCES

Figure 14 shows supervision glimpse sequences given by various methods. These are for a random sample of images.

### C.4 NEAREST NEIGHBOURS OF TEST IMAGES

As mentioned in Section 4, using pre-trained GANs to generate an empirical image distribution incurs a risk of test leakage; specifically, it means that we cannot control the set on which the GANs are trained. We use these GANs in our pipeline to generate near-optimal glimpse sequences, which are later used to train a hard attention network. The test set for this hard attention network can include images used to train the GAN, and so test leakage may occur. We verify that this has no significant effect by checking whether any test images (or uncomfortably similar images) are reproduced in the empirical distribution which we create with the GANs. We check this by displaying nearest neighbours for several test images in Fig. 14. We do not observe any cases where test images are reproduced by the GAN.

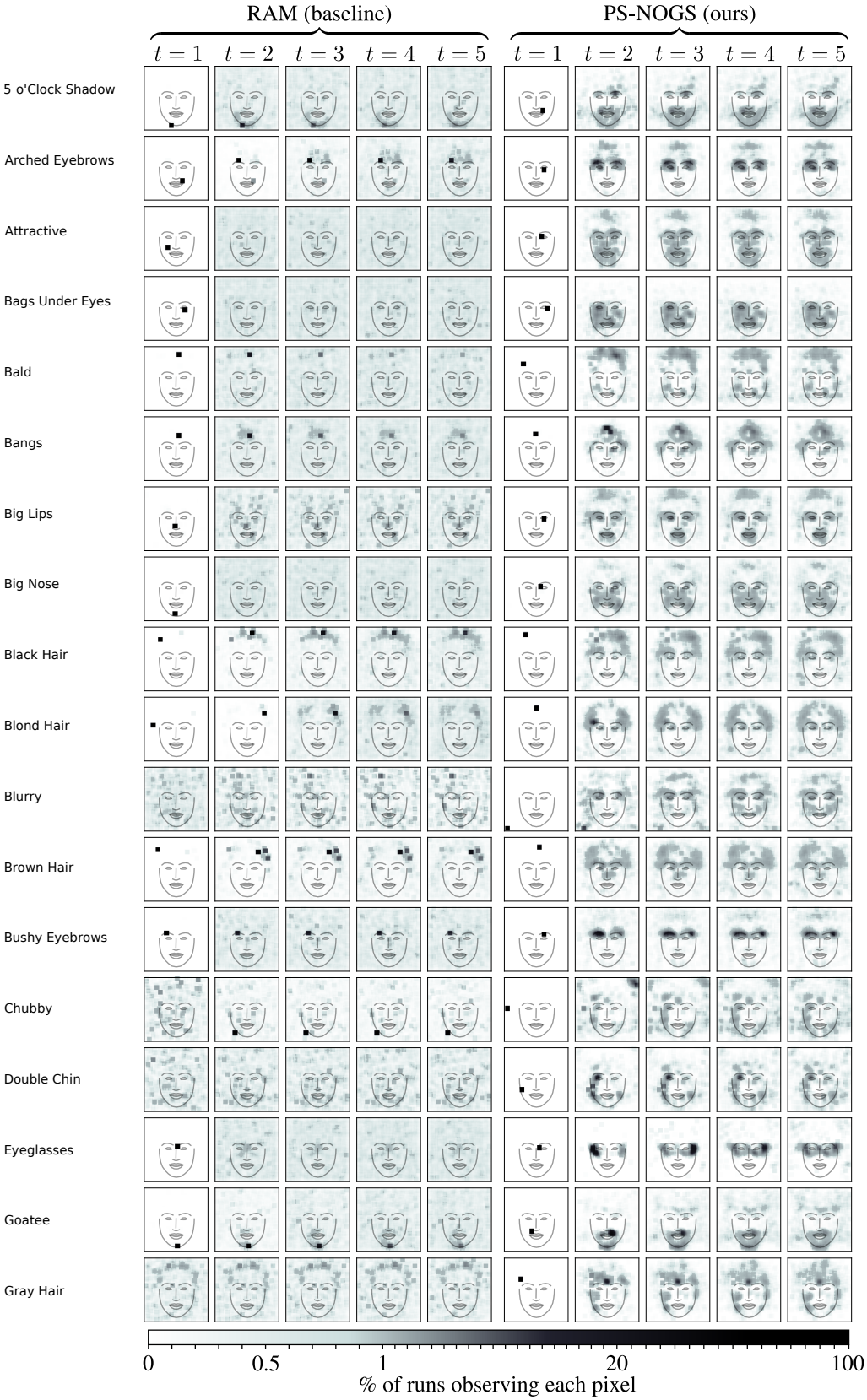


Figure 9: Glimpse locations on CelebA-HQ.

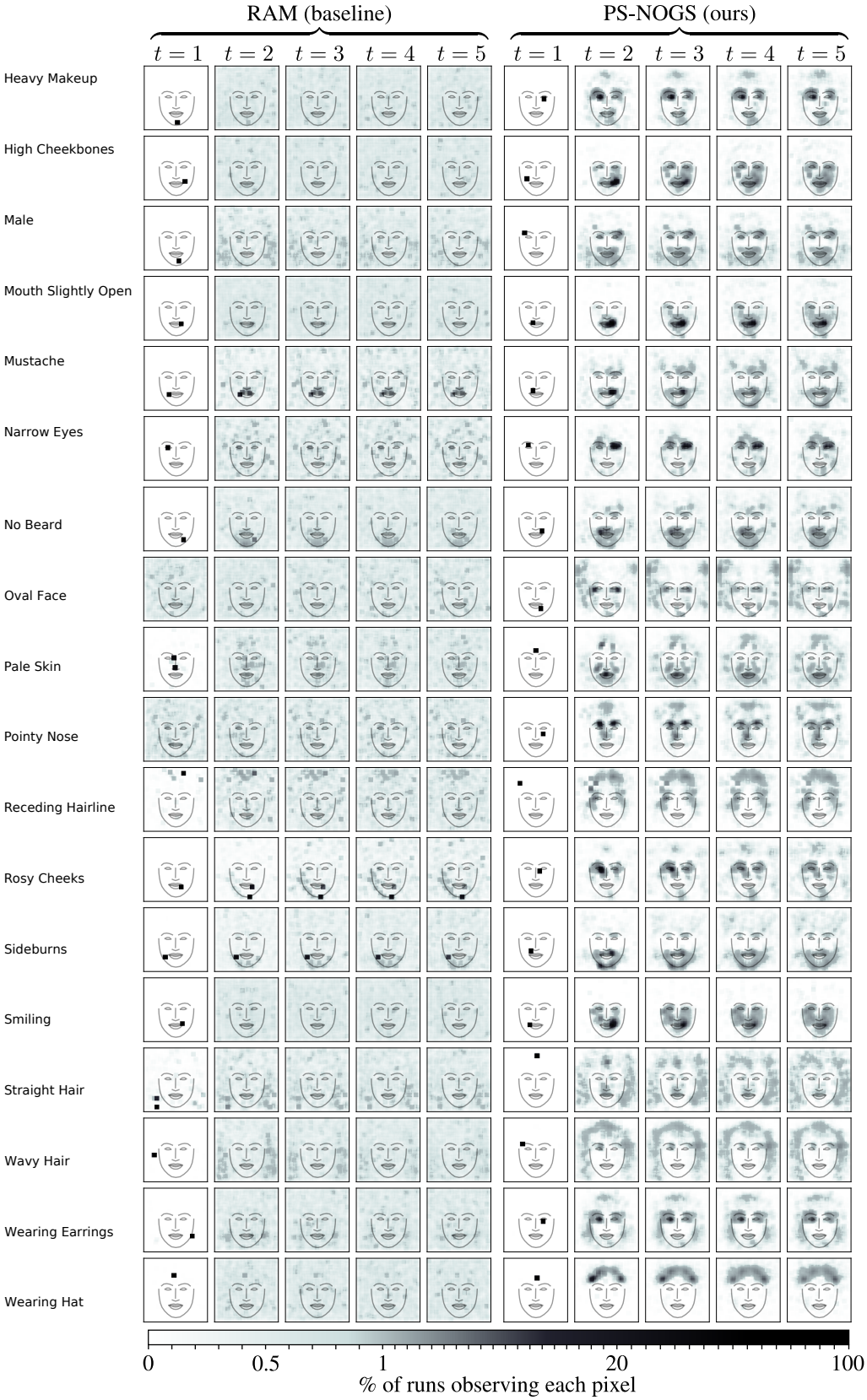


Figure 10: Glimpse locations on CelebA-HQ.

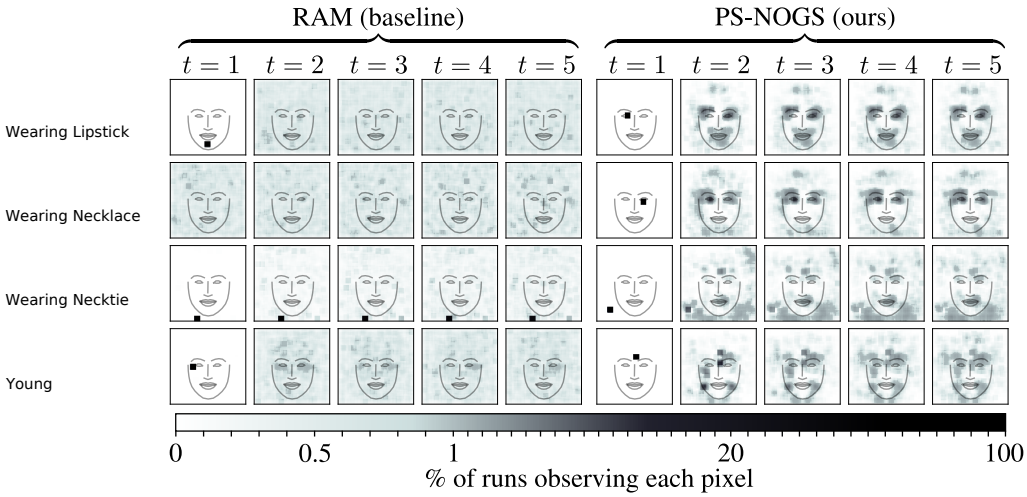


Figure 11: Glimpse locations on CelebA-HQ.

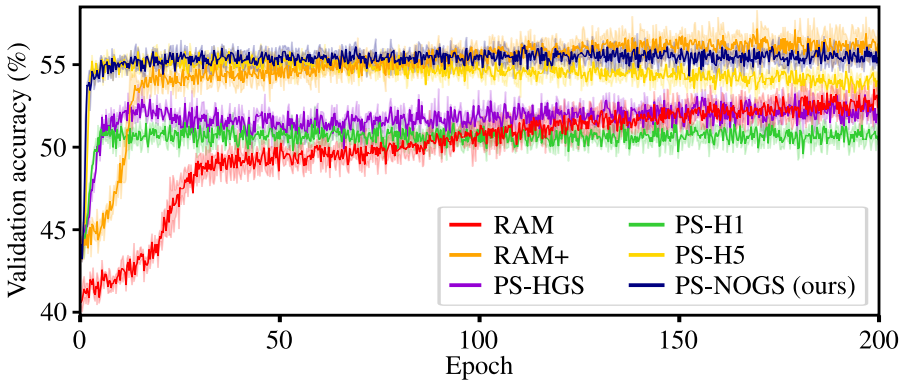


Figure 12: Expanded version of Fig. 6 including the additional baselines described in Appendix B.3.

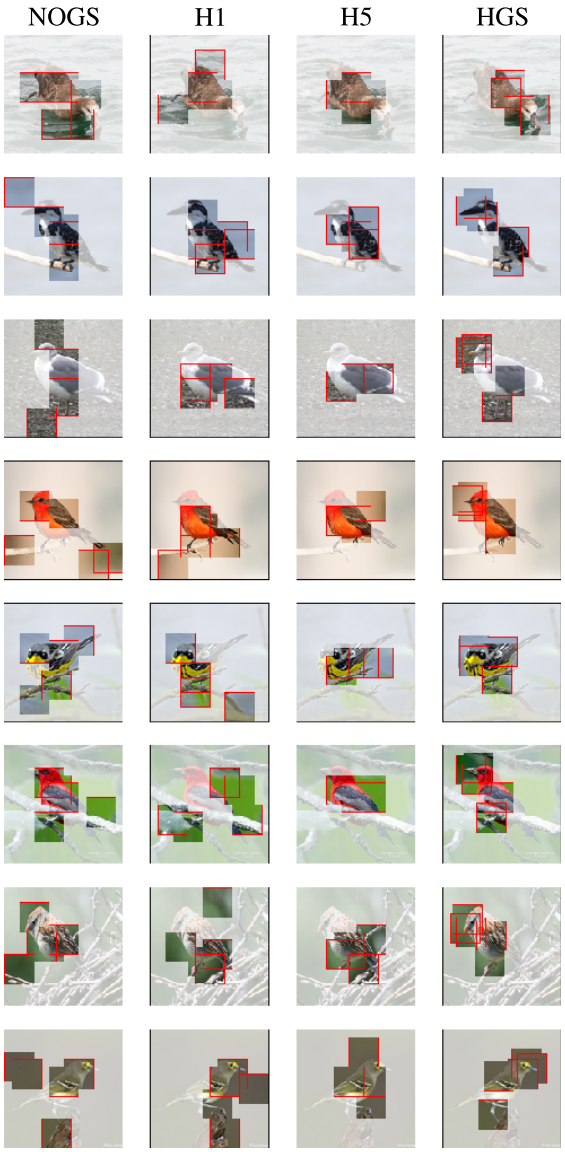


Figure 13: Examples of the image areas observed by near-optimal glimpse sequences (NOGS); sequences generated using the heuristic described in Appendix B.3 with $\gamma^{-1} = 1$ (H1) or $\gamma^{-1} = 5$ (H5); and hand-crafted glimpse sequences (HGS).

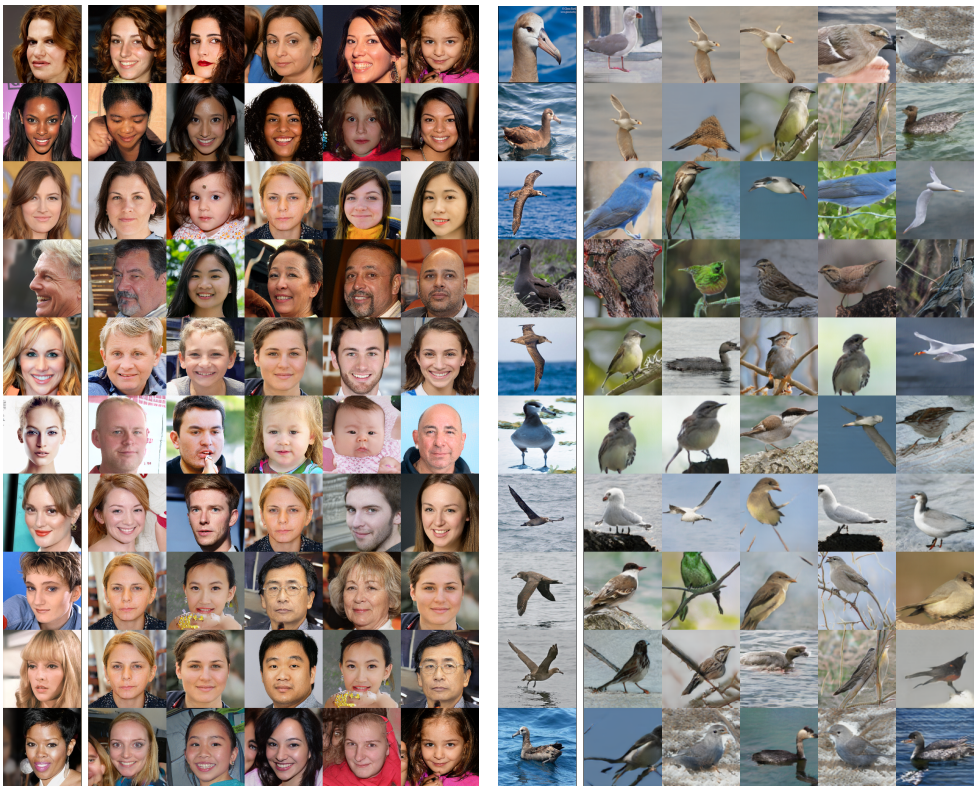

Figure 14: For CUB and CelebA-HQ, we show test images (left column) followed by their nearest neighbours from the artificial dataset. Neighbours are selected based on the L1 distance of a centred crop of the image.

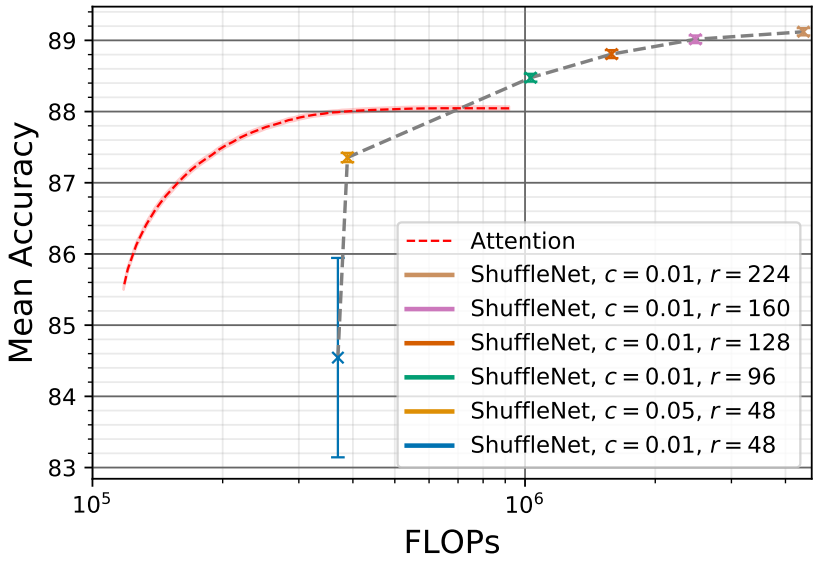


Figure 15: A preliminary plot of performance (mean accuracy over the 40 CelebA-HQ classification tasks) against the average number of FLOPs taken to classify an image. We considered MobileNet (Sandler et al., 2018) and ShuffleNet (Ma et al., 2018) CNN architectures with hyperparameters consisting of a multiplier for the number of channels in each layer ($c$) and the input resolution ($r$). We perform a grid search over these hyperparameters and plot the best six. Interpolations between these provide a baseline for accuracy at given numbers of FLOPs. The curve for the accuracy of the 'Attention' method is obtained as described in the text.

## D PRELIMINARY COMPARISON TO CNN

Figure 15 shows preliminary results suggesting that, in low-power settings, hard attention can outperform CNNs which process the full image. We emphasize that these are very preliminary results which motivate our work, but require a major modification from the setting we describe. In particular, the attention architecture uses a mechanism for "early stopping", where a network module estimates the expected information gain from the next glimpse, and computation can be halted after $t < T$ steps if this is below some threshold. Varying this threshold results in the smooth curve trading off between computation and accuracy. We see that, with a budget below about $7 \times 10^5$ FLOPs, the hard attention network achieves superior performance to the CNNs. It remains to be seen whether different attention architectures will provide an advantage for higher numbers of FLOPs.