A Implementation details: retrieval based scene understanding

A.1 How to store memories?

The memory bank only needs to be calculated once per dataset and can then be re-used for each of the images in the evaluation set. To populate the memory bank, each image in the dataset's training set (i.e. the "prompt") is encoded using the frozen backbone of the pretrained network to evaluate. We encode each of the training set images into a spatial map $\boldsymbol{k}_i = f_\theta(\boldsymbol{x}_i) \in \mathbb{R}^{H \times W \times D}$, where a feature $\boldsymbol{k}_i^j \in \mathbb{R}^D$ at a given spatial location j is aligned with the local label $\boldsymbol{l}_i^j$ created by averaging the pixel labels $\boldsymbol{y}_i^j$ in that patch. These features $\boldsymbol{k}_i$ are then L_2-normalized.

When the memory bank length is not large enough to accommodate all features for all images, it is necessary to subsample and only store a subset of the features of each image. For a concrete example using ADE20K, training set images have a resolution of 512×512 which when encoded by a ViT-B/16 results in a 32×32 grid of features (i.e. 1,024 features per image). To store every feature from each of ADE20K's 20,120 training images would require a memory bank length of $20{,}120 \times 32 \times 32 = 20{,}695{,}040$. When using data augmentation to increase the number of training images, the required length is even higher.

Our subsampling strategy for semantic segmentation works as follows. We define the number of features to take per image as $n_{\text{features\_per\_image}} = \frac{|\mathcal{M}|}{|\mathcal{D}| * \text{num\_augmentation\_epochs}}$ where $|\mathcal{D}|$ refers to the number of images in the training dataset. We thus sample the same number of features for each training image. Rather than sampling this number of features per image from the grid uniformly, we attempt to sample the most salient features using a simple strategy: upweighting patches containing class labels that appear less frequently in the image. Following the notation of Section 3.1, let $\boldsymbol{l}^j$ refer to the label attached to the patch indexed by j in the image and let $\mathbb{1}_{c \in \boldsymbol{l}^j} = 1$ if a given class $c \in \boldsymbol{l}^j$ and 0 otherwise. Then for each class c we define $\kappa_c = \sum_j \mathbb{1}_{c \in \boldsymbol{l}^j}$ (i.e. a count of how many patches the class c appeared in). We define a "class score" for each patch indexed by j as $\text{class\_score}^j = \sum_{c \in \mathcal{C}} \kappa_c \cdot \mathbb{1}_{c \in \boldsymbol{l}^j}$. Finally, we take the $n_{\text{features\_per\_image}}$ from the spatial map $\boldsymbol{k}_i$ with the lowest final scores using

$$\text{final\_score}^j = (\text{class\_score}^j \cdot x) + (10^6 \cdot \mathbb{1}_{\boldsymbol{l}^j = \emptyset}) \tag{5}$$

where $x \sim \mathcal{U}_{[0,1]}$. The first term introduces some stochasticity into the sampling process and the second term deprioritizes locations that have no class label. The chosen features serve as the memory bank keys and their associated labels are the memory bank values.

The subsampling strategy used for depth estimation is simpler since there are no classes involved. We opted not to use data augmentation for this task making $n_{\text{features\_per\_image}} = \frac{|\mathcal{M}|}{|\mathcal{D}|}$. We first randomly order each patch in the image, then place all patches that contain no valid pixel labels after any patch with valid pixel labels, and then take the first $n_{\text{features\_per\_image}}$ from the list.

There are many possible alternative strategies for sampling the most salient patches within an image in the event that the memory bank length cannot fit every feature from every image. We leave exploration of these possibly better sampling strategies for future work because in general we found this technique to perform well and wanted to show that nearest neighbor evaluation does not require complicated, hand-crafted strategies but rather works well out of the box with a simple heuristic calculated per image. For a complete listing of the hyperparameters involved in building and retrieving from the memory bank, see Appendix A.2.

A.2 How to recall memories?

After the memory bank has been populated as described in Appendix A.1, we sequentially make predictions for each image in the evaluation set. Evaluation was done on a single Nvidia V100 GPU per downstream task and takes approximately 20 minutes for PASCAL VOC, 40 minutes for ADE20K, and 75 minutes for NYUv2. Each image $\boldsymbol{x}$ is encoded as a grid of features $\boldsymbol{q} = f_\theta(\boldsymbol{x})$ and each of the features from this grid will serve as the query that we will look up the nearest neighbors for. We use the open-source ScaNN library [28] to perform the approximate nearest neighbor search efficiently. ScaNN natively provides the functionality to return both the top-k nearest neighbors for a given query as well as scores for the similarity that can be used as the attention logits. These scores are then divided by a temperature scaling value before having a softmax applied to them to obtain the final attention values (see Equation 1).

Throughout the paper, we use ScaNN in asymmetric hashing (AH) mode as opposed to brute-force mode. We find that there is little to no negative impact on the evaluation from using approximate nearest neighbor search as opposed to a brute-force exact search, despite the approximate search being several orders of magnitude faster. We use cosine similarity (L_2-normalized dot product) as a distance measure throughout this work. We also attempted some experiments using squared Euclidean distance and found it to have no benefits to performance for any of the models evaluated.

Table 6: **NN retrieval hyperparameters.** Note that no training is involved with NN evaluation, hence there are no hyperparameters such as learning rates or training epochs.

	Section 4.2	Everywhere else
$\|\mathcal{M}\|$ (Memory bank length)	20,480,000	10,240,000
k (nearest neighbors)	90	30
Temperature	.1	.02
Augmentation epochs	8	2
ScaNN dimensions_per_block	4	4
ScaNN num_leaves	512	512
ScaNN num_leaves_to_search	256	32
ScaNN reordering_num_neighbors	1800	120

Table 6 summarizes the hyperparameters used for NN evaluation throughout this work. For every section except for Section 4.2, we use a flat set of hyperparameters detailed in the "Everything else" column of Table 6. Because Section 4.2 is concerned with small subsets of the data (i.e. training on the order of hundreds of images), hyperparameter sweeps are extremely cheap to run and it is computationally fast to find nearest neighbors even with minimal approximations, hence we used a slightly different set-up in this regime. In general, we found nearest neighbor retrieval to be surprisingly robust to the choice of hyperparameters, with temperature and reordering_num_neighbors being the most relevant to performance. The same set of hyperparameters were used for the semantic segmentation tasks (PASCAL VOC and ADE20K) as for the monocular depth estimation task (NYUv2), with the exception of the number of augmentation epochs (we did not use augmentations for depth estimation). For a complete description of the meaning of the ScaNN hyperparameters, please see `https://github.com/google-research/google-research/blob/master/scann/docs/algorithms.md`.

Table 7 details the parameters used for augmenting the training dataset for semantic segmentation tasks. Note that the augmentations used to augment the training set when evaluating downstream tasks differ from the augmentations used for creating different views of the same image during contrastive pretraining described in Appendix C.1. When augmentations are enabled, the image is first scaled between the minimum and maximum scale factor, from which a random crop is selected. Then photometric augmentations are applied independently with the probabilities and maximum intensities provided.

Table 7: **Evaluation augmentations.** Parameters used to augment the training dataset for semantic segmentation.

Parameter	
Random crop probability	1.0
Minimum scale factor	0.5
Maximum scale factor	2.0
Brightness jittering probability	0.5
Contrast jittering probability	0.5
Saturation jittering probability	0.5
Hue jittering probability	0.5
Brightness adjustment max	0.1
Contrast adjustment max	0.1
Saturation adjustment max	0.1
Hue adjustment max	0.1

B Implementation details: contextual pretraining

The contextual pretraining module takes as input an image representation (i.e. query) $\boldsymbol{q} = \boldsymbol{h} = f_\theta(\boldsymbol{x}) \in \mathbb{R}^{B \times H \times W \times D}$ from the ViT encoder f_θ, where $B = 4096$ is the batch size, $H = W = 14$ are the height and width of the spatial feature map and $D = 768$ for ViT-B and $D = 1024$ for ViT-L is the feature dimension. Keys and values for the contextualization cross-attention operation are entries of the memory bank $\mathcal{M}_p = \{(\boldsymbol{k}_i, \boldsymbol{v}_i), i = 1, ..., |\mathcal{M}_p|\}$, where keys $\boldsymbol{k}_i$ are taken from previous batches by spatially averaging $\boldsymbol{h}$ (see Equation 2) and values $\boldsymbol{v}_i$ are obtained by applying a two-layer MLP ϕ_θ to the keys, where we use batch norm after the first layer and the hidden dimension is set to 4096. Each feature $\boldsymbol{q}^i$ of the image representation is then updated as $\boldsymbol{c}^i = \psi_\theta((1-\lambda)\frac{\boldsymbol{q}^i}{\|\boldsymbol{q}^i\|} + \lambda \frac{\hat{\boldsymbol{v}}^i}{\|\hat{\boldsymbol{v}}^i\|})$, where ψ_θ is a linear layer and $\|\boldsymbol{x}\|$ is the L_2 norm. Preliminary analysis showed $\lambda = 0.2$ to work well across datasets, so we use it for all our experiments, with higher values $\lambda \geq 0.5$ degrading performance.

We populate the memory bank with all batch entries of ImageNet-1k / -22k at each step, using the representations from the target network. The memory bank is spread across 128 Cloud TPU v3 workers with 1200 entries on each TPU for ImageNet-1k (256 TPUs with 600 entries for ImageNet-22k), resulting in total memory length of 153,600.

C Implementation details: self-supervised pretraining

C.1 Data augmentation

Each image is randomly augmented twice, resulting in two views $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The augmentations are constructed as compositions of the following operations, each applied with a given probability:

1. random cropping: a random patch of the image is selected, whose area is uniformly sampled in $[0.08 \cdot \mathcal{A}, \mathcal{A}]$, where $\mathcal{A}$ is the area of the original image, and whose aspect ratio is logarithmically sampled in $[3/4, 4/3]$. The patch is then resized to 224×224 pixels using bicubic interpolation;
2. horizontal flipping;
3. color jittering: the brightness, contrast, saturation and hue are shifted by a uniformly distributed offset;
4. color dropping: the RGB image is replaced by its grey-scale values;
5. gaussian blurring with a 23×23 square kernel and a standard deviation uniformly sampled from $[0.1, 2.0]$;
6. solarization: a point-wise color transformation $x \mapsto x \cdot \mathbb{1}_{x<0.5} + (1-x) \cdot \mathbb{1}_{x \geq 0.5}$ with pixels x in $[0, 1]$.

The augmented images $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ result from augmentations sampled from distributions $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. These distributions apply the primitives described above with different probabilities and different magnitudes. Table 8 specifies these parameters for the BYOL framework [27], which we adopt without modification.

Table 8: **Pretraining augmentations.** Parameters used to generate different views of a single image for contrastive pretraining.

<table>
<tr><th>Parameter</th><th>$\mathcal{T}_1$</th><th>$\mathcal{T}_2$</th></tr>
<tr><td>Random crop probability</td><td colspan="2">1.0</td></tr>
<tr><td>Flip probability</td><td colspan="2">0.5</td></tr>
<tr><td>Color jittering probability</td><td colspan="2">0.8</td></tr>
<tr><td>Color dropping probability</td><td colspan="2">0.2</td></tr>
<tr><td>Brightness adjustment max</td><td colspan="2">0.4</td></tr>
<tr><td>Contrast adjustment max</td><td colspan="2">0.4</td></tr>
<tr><td>Saturation adjustment max</td><td colspan="2">0.2</td></tr>
<tr><td>Hue adjustment max</td><td colspan="2">0.1</td></tr>
<tr><td>Gaussian blurring probability</td><td>1.0</td><td>0.1</td></tr>
<tr><td>Solarization probability</td><td>0.0</td><td>0.2</td></tr>
</table>

C.2 Optimization

We pretrain the model for 300 epochs on ImageNet-1k or 100 epochs on ImageNet-22k using AdamW [46] with a batch size of 4096, split across 128 Cloud TPU v3 workers for ImageNet-1k and 256 Cloud TPU v3 workers for ImageNet-22k. Training a ViT-B / ViT-L for 300 epochs on ImageNet-1k takes roughly 21 hours / 53 hours, while 100 epochs on ImageNet-22k takes approximately 60 hours / 128 hours. We update the online parameters θ with a cosine learning rate schedule with a base learning rate of 0.001, weight decay of 0.1 and gradient clipping with a maximum norm of 1. We update the target parameters ξ as an exponential moving average of the online parameters with a decay rate of 0.99.

Following [15] the projections and predictions in Equation 4 are normalized and rescaled such that their norm is equal to $1/\sqrt{\tau}$ where the contrastive loss temperature τ is equal to 0.1. When using additional supervision we set the supervised loss weight α to 0.25 for the supervised ViT-B trained on ImageNet-22k and $\alpha = 0.05$ for all other experiments.

D Supplementary analysis

D.1 Data efficiency

In Table 2 we compared *Hummingbird* with several leading representation learning techniques in the low-data regime. Here we provide the complete analysis from $1/128$ to 100% of the data, as well as results for our ViT-L model trained on ImageNet-22k to show the scalability properties of *Hummingbird*. Note that there is a difference between the experiments run here and those found in Section 4.4 of the main paper; that section uses an UperNet [73] decoder and this section uses a linear decoder for all of the finetuned rows in each table.

For PASCAL VOC (Table 9), *Hummingbird* performs very well not only in the low-data regime but in the full-data regime, with the apples-to-apples comparison (ViT-B self-supervised on IN1K) competitive with all other techniques even as the dataset fraction increases. This table also demonstrates the clear benefit of supervision as well as model-size and dataset size scaling—with only nearest neighbors (no finetuning), *Hummingbird*++ trained on IN22K with a ViT-L backbone beats all of the other finetuned variants for every dataset fraction. *Hummingbird*++ using a ViT-B and IN1K predictably lies inbetween the other two models for every dataset fraction.

For ADE20K (Table 10), the same general trends from above hold. Backbone and dataset scaling are once again beneficial as *Hummingbird*++ with ViT-L and IN22K training outperforms the other *Hummingbird* models, however this time the absolute performance relative to the finetuned competition in the high-data regime is less favorable since the end-to-end finetuned versions of other techniques start to outperform the nearest neighbors only ViT-L *Hummingbird*++ at $1/16$ of the data.

Table 9: **PASCAL VOC data efficiency analysis.** After pretraining, models are applied to downstream tasks with the indicated fraction of the dataset size. Models perform the task either with end-to-end fine-tuning with a linear head (E2E FT) or with our mechanism for in-context scene understanding using nearest neighbors at evaluation time (NN). All fine-tuning runs are averaged over five different seeds. The metric reported is mean IoU (higher numbers are better). † denotes models trained on ImageNet-22k; all other models were trained on ImageNet-1k.

			Fraction of dataset							
Method	Decoder	Backbone	1/128	1/64	1/32	1/16	1/8	1/4	1/2	1/1
DeiT-III [64]	E2E FT	ViT-B	41.8	53.8	63.1	67.7	70.7	72.2	73.4	75.2
DINO [14]	E2E FT	ViT-B	36.1	44.3	54.3	57.8	61.7	64.8	68.2	72.2
MoCo-v3 [19]	E2E FT	ViT-B	19.9	33.4	47.0	54.8	61.5	67.1	70.7	73.4
MAE [29]	E2E FT	ViT-B	34.2	44.1	53.0	58.7	62.7	67.4	70.8	73.5
LOCA [12]	E2E FT	ViT-B	40.1	53.9	63.1	67.8	70.7	72.8	74.4	75.5
Hummingbird	NN	ViT-B	50.5	57.2	60.1	62.6	64.3	65.9	68.9	71.8
Hummingbird++	NN	ViT-B	52.4	57.3	61.5	64.6	66.2	67.9	70.5	73.2
Hummingbird++†	NN	ViT-L	**61.8**	**65.3**	**68.0**	**70.7**	**71.4**	**73.2**	**75.3**	**77.2**

Table 10: **ADE20K data efficiency analysis.** After pretraining, models are applied to downstream tasks with the indicated fraction of the dataset size. Models perform the task either with end-to-end fine-tuning with a linear head (E2E FT) or with our mechanism for in-context scene understanding using nearest neighbors at evaluation time (NN). All fine-tuning runs are averaged over five different seeds. The metric reported is mean IoU (higher numbers are better). The results for other techniques between $1/32$ and $1/1$ are sourced directly from [12], the rest are reproductions. † denotes models trained on ImageNet-22k; all other models were trained on ImageNet-1k.

			Fraction of dataset							
Method	Decoder	Backbone	1/128	1/64	1/32	1/16	1/8	1/4	1/2	1/1
DeiT-III [64]	E2E FT	ViT-B	10.8	14.3	20.9	27.1	32.7	38.3	42.0	47.3
DINO [14]	E2E FT	ViT-B	11.7	14.4	18.4	24.5	29.5	35.2	39.5	44.1
MoCo-v3 [19]	E2E FT	ViT-B	4.6	7.9	17.7	25.2	30.8	36.5	40.7	45.4
MAE [29]	E2E FT	ViT-B	8.2	12.2	18.4	25.3	30.5	36.1	40.6	45.5
LOCA [12]	E2E FT	ViT-B	11.2	15.5	22.2	**30.0**	**34.4**	**39.1**	**42.8**	**47.9**
Hummingbird	NN	ViT-B	11.7	15.1	17.3	20.0	22.3	24.9	27.9	29.6
Hummingbird++	NN	ViT-B	12.7	16.4	18.9	21.5	24.0	26.8	29.9	32.0
Hummingbird++†	NN	ViT-L	**16.6**	**20.5**	**24.0**	27.4	30.2	33.1	36.0	37.8

D.2 Correlation of NN retrieval and finetuning performance

In this section, we study the relation between NN retrieval performance and end-to-end finetuning. To that end, we collect 14 *Hummingbird* models trained with different architectures (ViT-B vs ViT-L), datasets (ImageNet-1k vs ImageNet-22k), learning objectives (self-supervised or with additional supervision), and training lengths. Figure 5 plots the performance of these models when equipped with NN retrieval decoders (x-axis) and fully-finetuned UperNet decoders (y-axis). For both PASCAL VOC and ADE20K semantic segmentation, performance using one decoding scheme is highly predictive of the other (Person's $\rho = 0.80$ for PASCAL VOC, $\rho = 0.89$ for ADE20K). As such, even in cases where NN retrieval underperforms end-to-end finetuning, it can still be used as a powerful diagnostic tool. As illustrated in Section 4.3, evaluating with NN retrieval is much simpler and faster than with end-to-end finetuning, even when using a linear decoder. End-to-end finetuning often requires sweeping over optimization hyperparameters and averaging across multiple seeds, making it unsuitable for online evaluation, whereas NN retrieval is 10x less variable across runs and doesn't require any hyperparameter sweeps. As such NN retrieval can be used as an online evaluation that is highly predictive of performance obtained with more expensive finetuning protocols.

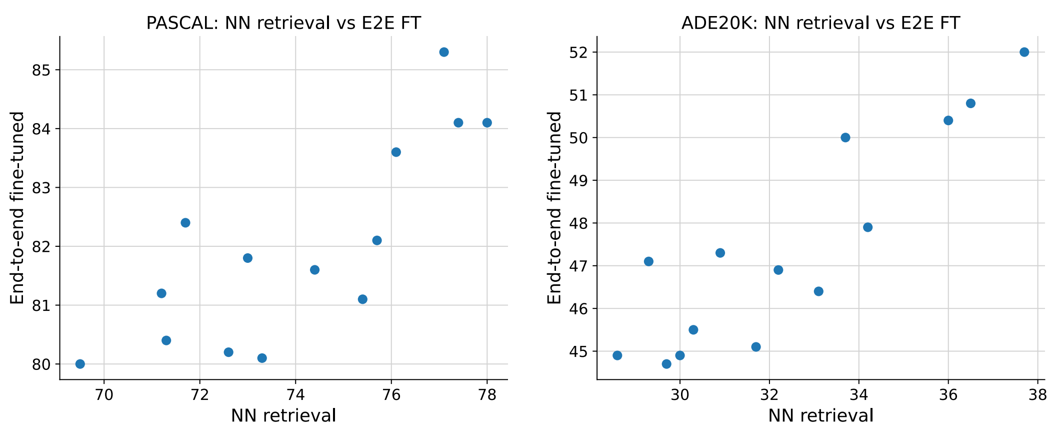

Figure 5: Correlation of NN retrieval vs end-to-end finetuning.

D.3 Effect of pretraining and evaluation memory length for ADE20K

We include the equivalent of Figure 4 on the ADE20K dataset in Figure 6. Similar to what we observe for PASCAL VOC, we benefit from large memory banks at evaluation. Since the ADE20K training set is roughly 2x larger than that of PASCAL VOC, we also observe that sampling which features to store in the memory bank is more important than it is for PASCAL VOC (see Appendix A.1 on the details of the sampling procedure). Similarly, at training time, ADE20K benefits from larger pretraining memory banks than PASCAL VOC, with performance plateauing for memory banks larger than $200{,}000$. Thus, we set the pretraining memory bank length to $153{,}600$ in all our experiments (see Appendix B for details on contextual pretraining).

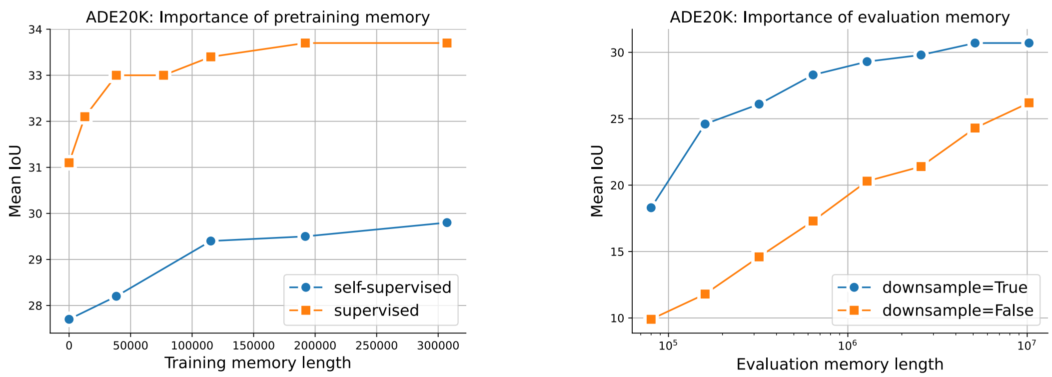

Figure 6: Effect of the pretraining (*left*) and evaluation (*right*) memory length on performance of ADE20K. All models were pretrained with ViT-B on ImageNet-22k. *Left*: Since the retrieval-based supervised objective is only defined for memory banks of non-zero length, for the purpose of this ablation we replace it with a simple linear classifier when $|\mathcal{M}_p| = 0$. *Right*: For downsample=False, we store representations of all patches into the memory bank. If downsample=True, we sample $|\mathcal{M}|/N$ patches per image (N is the length of the downstream training set), allowing for greater memory bank diversity and thus superior performance than when downsample=False.