Hard-Attention for Scalable Image Classification

Athanasios Papadopoulos [1] **Paweł Korus** [1 2] **Nasir Memon** [1]

Abstract

Deep neural networks (DNNs) are typically optimized for a specific input resolution (e.g. 224×224 px) and their adoption to inputs of higher resolution (e.g., satellite or medical images) remains challenging, as it leads to excessive computation and memory overhead, and may require substantial engineering effort (e.g., streaming). We show that multi-scale hard-attention can be an effective solution to this problem. We propose a novel architecture, TNet, which traverses an image pyramid in a top-down fashion, visiting only the most informative regions along the way. We compare our model against strong hard-attention baselines, achieving a better trade-off between resources and accuracy on ImageNet. We further verify the efficacy of our model on satellite images (fMoW dataset) of size up to 896×896 px. In addition, our hard-attention mechanism guarantees predictions with a degree of interpretability, without extra cost beyond inference. We also show that we can reduce data acquisition and annotation cost, since our model attends only to a fraction of the highest resolution content, while using only image-level labels without bounding boxes.

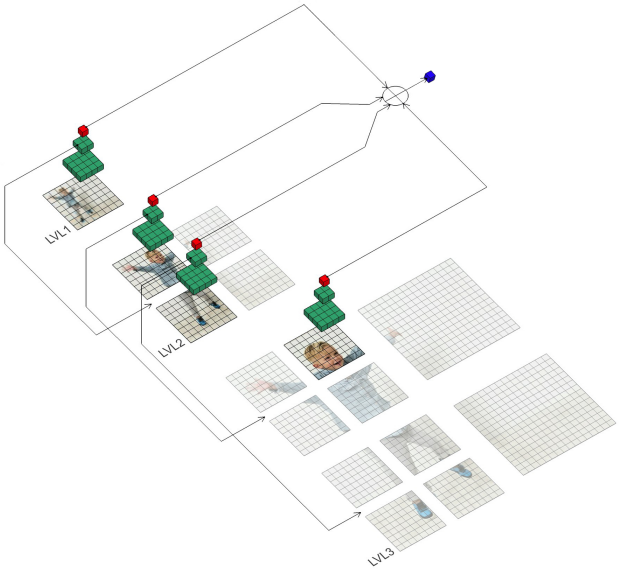

Figure 1. Multi-scale processing in the proposed TNet architecture. Starting at level 1, the image is processed in low resolution to get a coarse description of its content (red cube). Extracted features are used for (hard) selection of image regions worth processing in higher resolution. The process is repeated recursively (here to 2 additional levels). Features from all levels are combined (arrows on the right) to create the final image representation used for classification (blue cube).

1. Introduction

In image classification, deep neural networks (DNNs) are typically designed and optimized for a specific input resolution, e.g. 224×224 px. Using modern DNNs on inputs of higher resolution (as happens e.g., in satellite or medical imaging) is a non-trivial problem due to the subtlety of scaling model architectures (Tan & Le, 2019), and rapid increase in computational and memory requirements.

A linear increase in the spatial dimensions of the input results in a quadratic increase in computational complexity and memory, and can easily lead to resource bottlenecks. This can be mitigated with careful engineering, e.g., streaming (Pinckaers et al., 2019) or gradient checkpointing (Marra et al., 2020). However, such solutions are content-agnostic, and don't take advantage of the fact that discriminative information may be sparse and distributed across various image scales, deeming processing of the whole input unnecessary.

We show that multi-scale hard-attention can be an effective solution to this problem, because it focuses on selected image regions and dispenses with the necessity to process the entire input. To this end, we propose a novel architecture, TNet, which traverses an image pyramid in a top-down fashion, visiting only the most informative regions along the way. Our method draws its intuition from the way humans use saccades to explore the visual world, and an outline of its processing flow is presented in Figure 1.

Compared to other hard-attention models (Elsayed et al.,

[1]New York University [2]AGH University of Science and Technology. Correspondence to: Athanasios Papadopoulos <tpapadop@nyu.edu>.

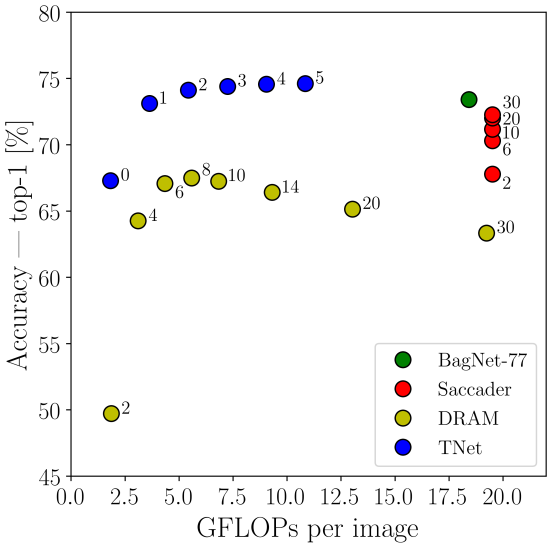

Figure 2. Experimental results on ImageNet (Deng et al., 2009) with baselines based on (Elsayed et al., 2019). Numeric annotations correspond to the number of attended locations ('0' means that only features from the 1st processing level are used). Our model offers the best trade-off between accuracy and complexity (FLOPs), while it surpasses a comparable fully convolutional baseline (green circle).

2019; Sermanet et al., 2014), our goal is to offer a method that effectively scales with the input size and available computational budget, leading to a better trade-off between complexity and accuracy as the number of attended locations changes. In addition, hard-attention explicitly reveals the image regions that our model values the most, and inherently provides a certain degree of interpretability to the model's predictions. This kind of attribution automatically happens during inference, without any additional post-processing cost (which can be large for some attribution methods, e.g., based on occlusions (Ancona et al., 2018)). Finally, our model can reduce the cost of data acquisition (Uzkent & Ermon, 2020) and annotation (Lin et al., 2014), because it attends only to a fraction of high resolution content (the rest can be acquired in lower resolution), and learns its attention policy from image-level labels without any bounding box annotations.

We evaluate our method on two datasets. First, we assess performance on ImageNet (Deng et al., 2009), by following the experimental setting of (Elsayed et al., 2019) to compare our method against strong hard-attention baselines. A summary of our results is depicted in Figure 2. As we can see, our model (blue circles) can achieve a better trade-off between computational cost (FLOPs; see Table 1 for time measurements) and accuracy compared to other hard-attention models (yellow and red circles). TNet yields higher accuracy at a lower cost, while it attends to fewer locations. In

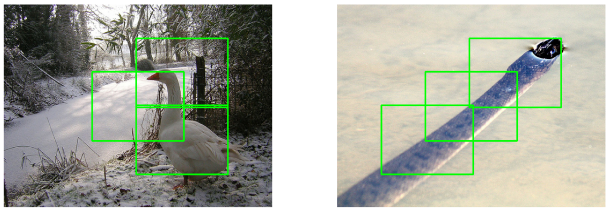

Figure 3. Image regions attended by the attention policy (using 3 locations) learned on ImageNet.

Section 5, we show that benefits in FLOPs directly translate to lower run time, and that hard-attention leads to reduction of memory requirements as well. Interestingly, our model achieves higher accuracy even compared to the fully convolutional baseline (green circle) with the same feature extraction architecture that processes the whole image.

We attribute this behavior to a novel regularization method that encourages classification based on individual attended locations. This has similar effect to cropping-based data augmentation, but uses informed and not random crops. Ablation study confirms that this mechanism was crucial to achieve high classification accuracy. We also analyze the learned attention policy both qualitatively and quantitatively; two examples are provided in Figure 3.

Finally, we test our model on the fMoW dataset (Christie et al., 2018) with high-resolution satellite images. We compare TNet against fully convolutional baselines (Tan & Le, 2019) trained on images of various fixed resolutions, with and without bounding box annotations. We show that our model can process images of high resolution, e.g., 896×896 px, with significant gains in computational cost (FLOPs and memory) and accuracy compared to baseline models operating on the same resolution.

2. Related Work

Attention. Attention has a long history in the artificial neural networks literature (Itti et al., 1998), and in the modern era of deep learning it has been used very successfully in various problems (Larochelle & Hinton, 2010; Denil et al., 2012; Bahdanau et al., 2014; Xu et al., 2015; Gregor et al., 2015; Wang et al., 2018; Shen et al., 2020). Two main classes of attention models include: *soft attention* which processes everything but weighs various regions differently; and *hard attention* which selects only a fraction of the data for processing.

Our model is conceptually similar to glimpse-based models (Ba et al., 2014; Mnih et al., 2014; Ranzato, 2014; Sermanet et al., 2014; Eslami et al., 2016; Ba et al., 2016). A major difference is that we don't use recurrent neural networks (RNNs) to combine extracted features from different

image locations, but instead, we simply average them (see Section 3). This way, gradients flow directly to the extracted features, without the need to backpropagate through RNN steps, avoiding the vanishing gradient problem. In addition, our processing flow is organized in levels (Figure 1), to increase parallelism in processing individual locations.

Hard-attention models address various use-cases. They can be motivated by interpretability (Elsayed et al., 2019), reduction of high-resolution data acquisition cost (Uzkent & Ermon, 2020), or computational efficiency (Katharopoulos & Fleuret, 2019). Our goal is to offer a single model that takes advantage of all these benefits.

Recent work explores soft attention mechanisms based on transformers, which originated from the natural language processing community (Vaswani et al., 2017). Transformers have already been used in machine vision to replace convolutional neural networks (CNNs) (Zhao et al., 2020; Dosovitskiy et al., 2020; Carion et al., 2020).

Multi-scale representations. We identify four broad categories of multi-scale processing methods. (1) *Image pyramid methods* extract multi-scale features by processing multi-scale inputs (Eigen et al., 2014; Pinheiro & Collobert, 2014; Ke et al., 2017; Najibi et al., 2018). Our model belongs to this category, and due to its recursive nature, it can extract features from an arbitrary number of pyramid levels (see Section 3). (2) *Encoding schemes* take advantage of the inherently hierarchical nature of deep neural nets, and reuse features from different layers, since they contain information of different scale (He et al., 2014; Liu et al., 2016; Chen et al., 2018). (3) *Encoding-Decoding schemes* follow up the feed-forward processing (encoding) with a decoder, that gradually recovers the spatial resolution of early feature maps, by combining coarse and fine features (Ronneberger et al., 2015; Lin et al., 2017). (4) *Spatial modules* are incorporated into the feed forward processing, to alter the feature extraction between layers (Yu & Koltun, 2015; Chen et al., 2017; Wang et al., 2019).

Computational efficiency. There are multiple ways to adjust the computational cost of deep neural networks. We organise them into four categories. (1) *Compression methods* aim to remove redundancy from already trained models (LeCun et al., 1990; Hinton et al., 2015; Yu et al., 2018). (2) *Lightweight design strategies* are used to replace network components with computationally lighter counterparts (Jaderberg et al., 2014; Iandola et al., 2016; Rastegari et al., 2016; Howard et al., 2017; Wang et al., 2017). (3) *Partial computation methods* selectively utilize parts of a network, creating paths of computation with different costs (Larsson et al., 2016; Figurnov et al., 2017; Zamir et al., 2017; Huang et al., 2017; Wu et al., 2018). (4) *Attention methods* selectively process parts of the input, based on their importance for the task at hand (Ramapuram et al., 2018; Levi & Ullman, 2018; Katharopoulos & Fleuret, 2019). This is the strategy we follow in our architecture.

3. Architecture

3.1. Processing Flow

We present our architecture by walking through the example in Figure 4, where we process an image with original resolution 896×896 px (① in the top left corner). In the fist level, we downscale the image to 224×224 px and pass it through the *feature extraction module*, in order to produce a feature vector V_1 that contains a coarse description of the original image.

To proceed to the next level, we feed an intermediate feature map, F_1, from the *feature extraction module* to the *location module*, which considers a number of candidate locations described by F_1, and predicts their importance (in this particular example, the candidate locations form a 2×2 regular grid (②), and the *location module* yields 4 predictions). We express region importance as attendance probability, which parametrizes a categorical distribution used for sampling without replacement; in our current example we sample 2 locations (③).

In the 2nd processing level, we crop the selected regions from the full-resolution image, resize them to 224×224 px, and feed them to the *feature extraction module* to obtain the corresponding feature vectors (here V_{21} and V_{23}).

The original input resolution allows us to move to a 3rd processing level, where we feed F_{21} and F_{23} to the *location module*, leading to 2 categorical distributions. We sample 1 location from each one of them to get V_{35} and V_{39}.

Features extracted from all levels are passed through a *positional encoding module*, which injects information about the spatial position and scale of the image regions the features describe. The resulting vectors, $\{V_*^{'}\}$, are averaged (④) into a single comprehensive representation, $V^{'}$, that is fed to the *classification module* for the final prediction.

The presented processing architecture is flexible and can be used for making predictions after each processing level and from a various numbers of attended locations. It seamlessly combines all information encountered so far and allows for dynamic early stopping based on current computational constraints.

3.2. Modules

The **feature extraction module** receives an image of fixed size as input, and outputs a feature vector V and an intermediate spatial representation F. The input size, a hyperparameter we call *base resolution*, defines the minimum amount of information that can be processed. Hence, it constraints

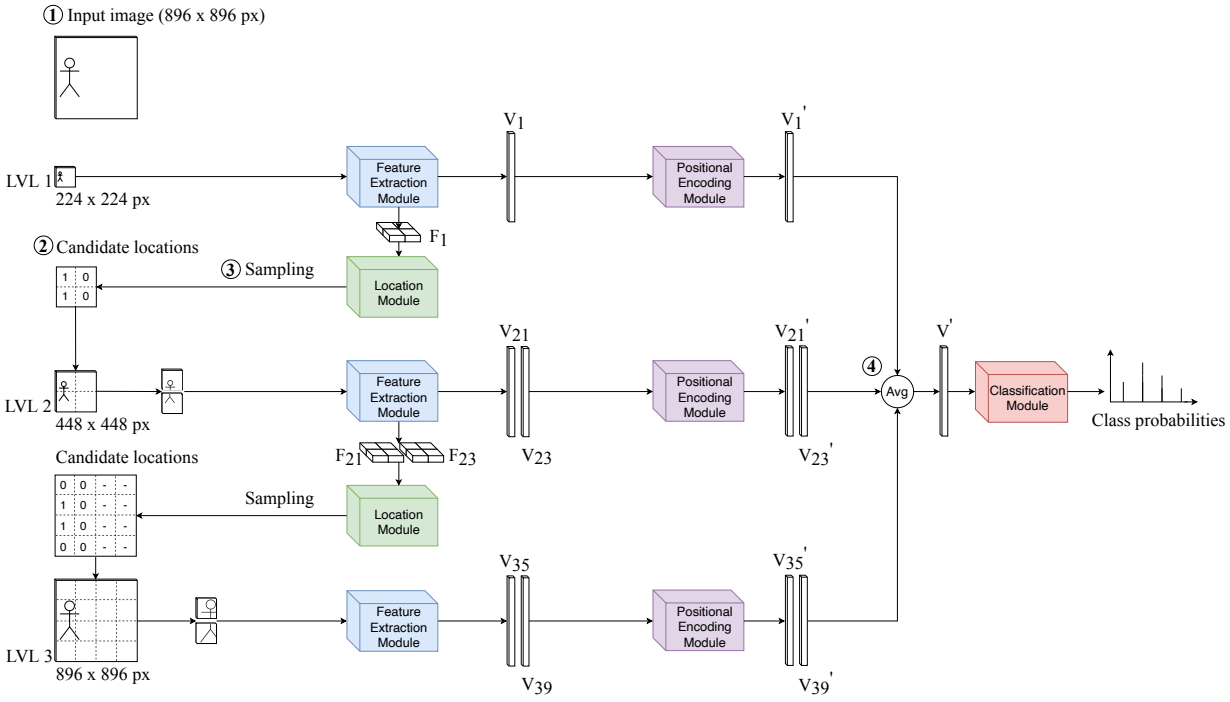

Figure 4. Three unrolled processing levels of our architecture. Starting at level 1, the image is processed in the coarsest scale (*Feature Extraction Module*), and the extracted features are used to decide which image locations should be processed in finer detail (*Location Module*). This process is repeated for each selected location to reach level 3, where features from the highest resolution are extracted. All features are enriched with positional information (*Positional Encoding Module*), and then are averaged before the final classification (*Classification Module*).

the minimum cost that our model has to pay in computational and memory resources. In our experiments, feature extraction modules are implemented using CNNs.

The **location module** predicts $K = n^2$ probabilities of a Categorical distribution over the locations within a given $n \times n$ grid of candidate image regions. It receives a feature map of size $n \times n \times c$ as input, where each $1 \times 1 \times c$ vector describes the corresponding location of the grid. The feature map is passed through a series of 1×1 convolutions (contextual information is infused as well, e.g., via squeeze and excitation (Hu et al., 2018)), yielding K logits, which are transformed to relative region importance via a softmax layer.

The **positional encoding module** receives a feature vector f and a positional encoding vector p, and combines them (e.g., through a fully connected layer) to an output feature vector f'. We use a variant of the fixed positional encodings based on sine and cosine functions introduced by (Vaswani et al., 2017). Instead of a single dimension of time, we have three: two spatial dimensions and scale.

The **classification module** projects the final feature vector (e.g., via a linear layer) to classification logits. We provide the exact architecture in Appendix B.1., along with justification of the design choices we made for every module.

4. Training

4.1. Learning Rule

Our model is not end-to-end differentiable because of location sampling. We address this problem using a variant of the *REINFORCE* (Williams, 1992) learning rule:

$$L_F = \frac{1}{N \cdot M} \sum_{i=1}^{N \cdot M} \Big[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \lambda_f (\log p(y_i|l^i, x_i, w) - b) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \Big] \quad (1)$$

where x_i is the ith image, y_i is its label, and w are the parameters of our model. $p(l^i|x_i, w)$ is the probability that the sequence of locations l^i is attended for image x_i, and $p(y_i|l^i, x_i, w)$ is the probability of predicting the correct label after attending l^i.

$N \cdot M$ is the total number of examples used for each update. The size of our original batch B is N, and we derive (1) using a Monte Carlo estimator with M samples to approx-

imate the expectation $\sum_{l^i} p(l^i|x_i, w)\Big[\frac{\partial \log p(y_i|l^i,x_i,w)}{\partial w} + \log p(y_i|l^i, x_i, w)\frac{\partial \log p(l^i|x_i,w)}{\partial w}\Big]$ for each image x_i in B.

b is a baseline used to reduce the variance of our estimator, while we replace $\log p(y_i|l^i, x_i, w)$ with a discrete indicator function R equal to 1 for correct predictions and 0 otherwise. λ_f is a weighting hyperparameter.

The first term of L_F is used to update the parameters in order to maximize the probability of the correct label. The second term is used to update the location selection process, according to the utility of the attended sequence l^i in the prediction of the correct label.

We provide a detailed derivation of learning rule (1) in Appendix A.1.

4.2. Per-Feature Regularization

When our model attends to a location sequence l_i and makes a correct prediction, we positively reinforce the probability to attend every location in the sequence. This is expressed in the second term of (1), where we use the probability $p(l^i|x_i, w)$ of the whole sequence l_i. However, some of the locations may have missed the object of interest, and the correct prediction may be due only to a subset of locations that successfully located the object. In such cases, we would backpropagate an update signal that encourages attending uninformative regions (the ones that missed the object), leading to a sub-optimal attention policy.

To mitigate this problem, for every attended location, we use its feature vector to make a separate classification prediction. Then, we use these predictions to complement our assessment on whether the corresponding image regions have useful content. Per-feature regularization modifies our learning rule in the following way:

$$L_r = L_F^s + \frac{1}{|l^i|}\sum_{k=1}^{|l^i|} L_F^k \tag{2}$$

$$L_F^s = \frac{1}{N\cdot M}\sum_{i=1}^{N\cdot M}\Big[\lambda_c \frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \lambda_f\lambda_r(R^s - b)\frac{\partial \log p(l^i|x_i, w)}{\partial w}\Big] \tag{2a}$$

$$L_F^k = \frac{1}{N\cdot M}\sum_{i=1}^{N\cdot M}\Big[(1-\lambda_c) \frac{\partial \log p(y_i|l_k^i, x_i, w)}{\partial w} + \lambda_f(1-\lambda_r)(R^k - b)\frac{\partial \log p(l_k^i|x_i, w)}{\partial w}\Big] \tag{2b}$$

where L_F^s is learning rule (1) with additional weighting hyperparameters $\lambda_c, \lambda_r \in [0, 1]$. L_F^k is learning rule (1) when we attend only to the kth location (l_k^i) from sequence l^i. Also, we introduce weighting factors $(1-\lambda_c)$ and $(1-\lambda_r)$. $|l^i|$ is the number of locations in sequence l^i. R^s and R^k are discrete indicator functions equal to 1 when a prediction is correct, and 0 otherwise. L_F^s updates the parameters of our model based on attending l^i, while L_F^k updates them based on l_k^i; λ_c and λ_r specify the relative importance of these updates.

Even though our initial motivation was to improve the attention policy, the first term in (2b) updates feature extraction parameters based on independent predictions from attended image regions. We empirically observed that such updates boost performance, potentially because they lead to features that co-adapt less and generalize better.

5. Experimental Evaluation

5.1. Experiments On ImageNet

Data. ImageNet (Deng et al., 2009) consists of natural images from $1,000$ classes. We use the ILSVRC 2012 version with $1,281,167$ training and $50,000$ validation images, including $544,546$ examples with bounding box annotations.

Models. We use Saccader and DRAM (Elsayed et al., 2019) as hard-attention baselines. We use the version of Saccader with BagNet-77-lowD (Brendel & Bethge, 2019) as its backbone for feature extraction. BagNet-77-lowD is based on modified ResNet-50 (He et al., 2016) with receptive field constrained to 77×77 px. DRAM uses the standard ResNet-50 for feature extraction, but it applies it to glimpses of 77×77 px, which restricts its receptive field accordingly.

For fair comparison, we use BagNet-77-lowD as the feature extraction module of TNet, and set the base resolution to 77×77 px. We introduce 3 modifications to the model and also use it as a separate baseline, referred to as BagNet-77, that processes 224×224 px images in a fully convolutional manner. First, we replace the "valid" padding of some convolutional layers with "same" to obtain less aggressive reduction of the spatial dimensions. Second, we removed batch-normalization due to technical issues in preliminary experiments (batch norm was successfully used in later experiments on the fMoW dataset; Section 5.2.). Third, we use Leaky ReLU instead of ReLU activations, to avoid 0 gradients for negative inputs. In the location module, we use a uniform 5×5 grid of overlapping candidate regions. The dimensions of each grid cell span 34.375% of the corresponding image dimensions. This way, for an image of size 224×224 px, the image patches within the grid cells at the 2nd processing level are 77×77 px. More details of the TNet architecture are provided in Appendix B.1.

Training. We train TNet with 2 processing levels on images of 224×224 px using class labels only. We train for 200 epochs using the Adam optimizer (Kingma & Ba, 2014)

Table 1. Results on ImageNet: TNet surpasses all baselines by attending to just 2 locations. Higher accuracy can be achieved for less FLOPs against the best performing versions of the baselines, and translates to lower actual run time. Memory savings are obtained compared to BagNet-77 and the Saccader as well. TNet has slightly more parameters compared to BagNet-77 due to the additional modules, but has significantly fewer parameters than other hard-attention models.

Model	# Locs	Top-1 Acc.	Top-5 Acc.	FLOPs (B)	# Params (M)	Time (msec/im)	Memory (GB)
Saccader	2	67.79%	85.42%	19.5	35.58	7.74	2.58
	6	70.31%	87.8%	19.5		7.66	2.52
	20	72.02%	89.51%	19.51		7.75	2.53
	30	72.27%	89.79%	19.51		7.72	2.51
DRAM	2	49.72%	73.27%	1.86	45.61	3.83	0.45
	4	64.26%	84.84%	3.1		4.29	0.44
	8	67.5%	86.6%	5.58		5.1	0.45
	20	65.15%	84.58%	13.03		7.65	0.46
BagNet-77	-	73.42%	91.1%	18.42	20.55	5.99	2.62
TNet	0	67.29%	87.38%	1.82	21.86	0.74	0.46
	1	73.12%	90.56%	3.63		1.43	0.57
	2	74.12%	91.18%	5.43		2.09	0.69
	3	74.41%	91.4%	7.24		2.74	0.95
	5	74.62%	91.35%	10.84		3.97	1.47

with initial learning rate 10^{-4}, that we drop once by a factor of 0.1. We use dropout (keep probability 0.5) in the last layer of feature extraction. We use per-feature regularization with $\lambda_c = \lambda_r = 0.3$. We attend to a fixed number of 3 locations.

We train BagNet-77 from scratch, on 224×224 px images. Compared to TNet, we reduced dropout keep probability to 0.375, and we early-stop after 175 epochs. We don't train our own Saccader and DRAM, we use the results reported in (Elsayed et al., 2019). We provide additional details about training in Appendix B.2.

Results. We present our results in Table 1 and Figure 2. TNet outperforms Saccader and DRAM by attending to only 1 location, while it surpasses BagNet-77 by attending to 2. Saccader was designed with accuracy and interpretability in mind, which leads to sub-optimal computational efficiency as it processes the entire full-resolution image before attending to locations. As a result, FLOPs stay nearly constant and remain similar to BagNet-77, more that 5 times higher than TNet with 1 attended location. DRAM offers the expected gradual increase in computational cost as the number of attended locations increases, but for FLOPs comparable with TNet - e.g., DRAM for 8 locations (maximum accuracy) and TNet for 2 - our model is superior by more than 6.5%.

TNet has slightly more parameters than BagNet-77, because of the location and positional encoding modules. Saccader and DRAM have significantly heavier attention mechanisms in terms of parameters.

We performed detailed model profiling to validate correspondence between theoretical FLOPs and real processing speed and to assess memory requirements. For all models we profile the inference phase on batches of 64 images (early batches are discarded to discount code optimization). We used a single NVIDIA Quadro RTX 8000 GPU, with 64 GB of RAM, and 20 CPUs to mitigate data pipeline impact. For Saccader and DRAM we used public implementations (google research, 2019) in TensorFlow (TF) 1. TNet and BagNet-77 were implemented in TF 2. The difference in TF versions may be a confounding factor in the obtained results. However, TNet and BagNet-77 use the same environment and yield the expected difference in latency.

DRAM attends locations sequentially, while TNet processes all locations from the same level in parallel. This leads to fixed memory use in DRAM and monotonic increase in TNet, as number of attended locations increases. We could trade this for latency by switching to fully sequential processing.

5.2. Experiments On fMoW

Data. Functional Map of the World (fMoW) (Christie et al., 2018) consists of high-resolution satellite images from 62 classes ($363,572$ training, $363,572$ validation and $53,473$ images). All images have bounding box annotations.

Models. We use EfficientNet-B0 (Tan & Le, 2019) as the feature extraction module of TNet, with base resolution of 224×224 px. In the location module we use a 3×3 grid with overlapping cells (50% overlap). Hence, by processing

Table 2. Results on fMoW: TNet effectively scales to inputs of resolution 448×448 px and 896×896 px. It surpasses in accuracy the EfficientNet-B0 baselines trained on inputs of the same resolution, while it requires less FLOPs. Differences in FLOPs translate to differences in actual run time, while memory requirements are reduced as well. TNet has more parameters compared to EficientNet-B0 due to the additional modules.

Model	Input Size	# Locs	Top-1 Acc.	Top-5 Acc.	FLOPs (B)	# Params (M)	Time (msec/im)	Memory (GB)
EfficientNet-B0 (with bboxes)	224	-	69.7%	89.22%	0.39	4.13	0.81	0.76
EfficientNet-B0	224	-	62.8%	84.97%	0.39	4.13	0.79	0.76
	448		69.83%	90.22%	1.54		3.02	2.6
	896		70.6%	90.81%	6.18		12.21	10.35
TNet	224	0	47.79%	81.14%	0.39	4.56	0.83	0.76
	448	1	70.17%	90.99%	0.77		1.76	1.3
		2	71.46%	91.58%	1.16		2.46	1.55
		3	71.57%	91.72%	1.55		3.19	2.03
	896	4	72.16%	91.98%	1.94		4.72	3.13
		6	71.92%	91.83%	2.71		6.13	3.47

2 levels, we can use inputs of 448×448 px, and for 3 levels, we can use inputs of 896×896 px. Details of the TNet architecture are provided in Appendix B.1.

We create our first 3 baselines by training Efficient-B0 with inputs of different size; 224×224 px, 448×448 px, and 896×896 px. We do not crop the images, but merely resize them to the target resolution. These baselines represent a straightforward way of scaling a model to images of considerably higher resolution than the one it was designed for. For the last baseline, we use the available bounding boxes to crop the regions of interest, resize them to 224×224 px, and train a corresponding EfficientNet-B0 model. This is a common practice for handling high-resolution images when bounding box annotations are available (Uzkent & Ermon, 2020).

Training. We train TNet using only class-level labels. The training proceeds in 2 stages: (1) training from scratch for 40 epochs on 448×448 px inputs, using 2 processing levels, and a fixed number of 2 locations; (2) finetuning for 10 more epochs on images of 896×896 px, which allow us to extend processing to 3 levels. We use 4 locations; 2 in the 2nd processing level, plus 1 for each of them in the 3rd. We use per-feature regularization in both stages. All 4 baselines are trained independently from scratch. The exact hyperparameters are provided in Appendix B.3.

Results. We present our results in Table 2. We see that the baseline trained with bounding boxes achieves top-1 accuracy 69.7%[1], which is marginally lower than EfficientNet-B0 trained without bounding boxes on 448×448 px images. The baseline accuracy for 896×896 px images is higher, 70.6%, showing that there still is valuable information in higher resolution.

TNet surpasses all baselines by attending to 2 locations on the 2nd processing level (top-1 accuracy 71.46%). This requires less FLOPs than the corresponding baseline at the same input size (448×448 px). Extending processing to 3 levels (attending to 4 locations) further increases accuracy to 72.16%. This requires less than $1/3$ of the FLOPs of the corresponding baseline operating on 896×896 px inputs.

We profiled all models described in Section 5.1 and show that gains in FLOPs translate to gains in actual run time. TNet also requires less memory, although it has more parameters because of the location and positional encoding modules.

5.3. Attention Policy Analysis

In the 1st processing level of our architecture, we extract features from a low resolution version of the entire input image, and as a result, the final representation always contains global information. This is deliberately avoided by the Saccader, because global information could undermine interpretability of the attended locations; it's not clear whether the attended locations or the global content were more im-

[1]ResNet-50 trained on fMoW with bounding box crops resized to 224×224 px, achieves 67.3% top-1 accuracy (Uzkent & Ermon, 2020). The resulting difference in accuracy with our EfficientNet-B0 baseline, resembles the reported difference in accuracy between the two models on ImageNet (Tan & Le, 2019). This is an indication that our baselines are sufficiently trained.

Figure 5. Image regions attended by the attention policy (using 2 locations) learned on fMoW.

portant for the final prediction.

To examine the relative importance between the coarse context and the information from the attended locations, we evaluate TNet on ImageNet, by using features only from attended locations. With 2, 3 and 5 attended locations, TNet achieves top-1 accuracy 67.95%, 69.66% and 71.05% respectively (with contextual information included, the corresponding accuracy values are 74.12%, 74.41% and 74.62%). The Saccader achieves similar accuracy for 2, 4 and 8 locations, with 67.8%, 69.51% and 70.08%. This shows that features extracted from attended locations have discriminative value even without global context. As a result, the attention policy can be valuable for interpreting predictions.

Hard-attention is also used for reduction of high-resolution data acquisition cost (Uzkent & Ermon, 2020). When our model attends to 1 location on ImageNet, it is processing only 11.71% of the input in the highest resolution. For 2, 3, 4, and 5 attended locations, this percentage grows to 19.73%, 26.49%, 32.62%, and 38.37% respectively. On fMoW, when our model attends to 4 locations, it covers 37.91% of the input in resolution 448×448 px, and less than 12.5% in the highest resolution of 896×896 px.

We show examples of attended locations in Figure 3 (ImageNet) and Figure 5 (fMoW). On ImageNet, the learned policy is highly content dependent and informative as it overlaps with intuitively meaningful regions (see Appendix C for more examples). On fMoW, our model predominantly attends to locations at the center, where nearly all objects of interest are located. While this is a correct heuristic implied by the data, it leads to a biased attention policy that is not sensitive to content changes. This demonstrates that sufficient diversity is needed to train fully informative policies.

5.4. Ablation Study

We examine the effect of per-feature regularization by training TNet on ImageNet without it. With 1 attended location, top-1 accuracy drops from 73.12% (Table 1) to 65.21%, while for 3 and 5 locations, accuracy drops from 74.41% and 74.62%, to 67.33% and 68.55% respectively. The drop in accuracy is substantial, to the degree that TNet is placed below both Saccader and BagNet-77 in terms of accuracy.

Per-feature regularization may have similar impact as cropping-based data augmentation, since it forces the model to make independent predictions with features from every attended location. However, the attention policy is not random, but learned, which is crucial for the quality of the crops. In addition, we don't get one image crop per epoch, but multiple crops in the same training iteration. We hypothesize that this is important to prevent feature co-adaptation, since the model learns to recognize the same object from multiple crops simultaneously.

6. Conclusion

We proposed a novel multi-scale hard-attention architecture, TNet, that can efficiently scale to images of high resolution. By controlling the number of attended locations, the method can adjust the accuracy-computation trade-off dynamically, e.g., based on currently available resources (or level-of-service). We demonstrated the efficacy of our method on ImageNet against strong hard-attention baselines, and we further verified its behavior with high-resolution satellite images (fMoW). The attention policy reveals the image regions deemed more informative by our model, and makes its predictions inherently interpretable.

There are multiple research directions that can address current limitations of our method. First, we would like the decision on the number of attended locations to stem from a learned policy. This way, we could adapt the number of attended locations according to the content of the input. In addition, the simple averaging of feature vectors before the final classification, can be replaced with a weighted average, potentially with weights predicted during inference. Per-feature regularization is based on predictions with features from individual attended locations, however, we could incorporate predictions with various combinations of these features.

Finally, we hypothesize that under a number of assumptions, hard-attention has the potential to be useful to any intelligent agent that navigates through the immense complexity of the natural visual world. These assumptions are that (1) the available resources are limited (2) there are performance constraints e.g. maximum response time, (3) the available information is practically infinite (4) all bits of information are not equally useful, and can even be misleading, e.g. noise. In this context, it would be beneficial for an intelligent agent to prioritize the expenditure of its resources according to the utility of the available information, in order to reach its performance goals; this is what a learned hard-attention mechanism can facilitate.

References

Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations*, 2018.

Ba, J., Mnih, V., and Kavukcuoglu, K. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pp. 4331–4339, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Brendel, W. and Bethge, M. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. *arXiv preprint arXiv:1904.00760*, 2019.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229. Springer, 2020.

Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

Christie, G., Fendley, N., Wilson, J., and Mukherjee, R. Functional map of the world. In *CVPR*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Eigen, D., Puhrsch, C., and Fergus, R. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pp. 2366–2374, 2014.

Elsayed, G. F., Kornblith, S., and Le, Q. V. Saccader: improving accuracy of hard attention models for vision. *arXiv preprint arXiv:1908.07644*, 2019.

Eslami, S. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Hinton, G. E., et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pp. 3225–3233, 2016.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1039–1048, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

google research. *GitHub repository that contains the official Saccader and DRAM implementations*, 2019. URL https://github.com/google-research/google-research/tree/master/saccader.

Gregor, K., Danihelka, I., Graves, A., Rezende, D., and Wierstra, D. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pp. 1462–1471, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision*, pp. 346–361. Springer, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Howard, A. G. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Itti, L., Koch, C., and Niebur, E. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20 (11):1254–1259, 1998.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Katharopoulos, A. and Fleuret, F. Processing megapixel images with deep attention-sampling models. *arXiv preprint arXiv:1905.03711*, 2019.

Ke, T.-W., Maire, M., and Yu, S. X. Multigrid neural architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6665–6673, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Larochelle, H. and Hinton, G. E. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pp. 1243–1251, 2010.

Larsson, G., Maire, M., and Shakhnarovich, G. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Levi, H. and Ullman, S. Efficient coarse-to-fine non-local module for the detection of small objects. *arXiv preprint arXiv:1811.12152*, 2018.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Lin, T.-Y., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. Feature pyramid networks for object detection. In *CVPR*, volume 1, pp. 4, 2017.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Luo, W., Li, Y., Urtasun, R., and Zemel, R. Understanding the effective receptive field in deep convolutional neural networks. *arXiv preprint arXiv:1701.04128*, 2017.

Marra, F., Gragnaniello, D., Verdoliva, L., and Poggi, G. A full-image full-resolution end-to-end-trainable cnn framework for image forgery detection. *IEEE Access*, 8: 133488–133502, 2020.

Mnih, V., Heess, N., Graves, A., et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pp. 2204–2212, 2014.

Najibi, M., Singh, B., and Davis, L. S. Autofocus: Efficient multi-scale inference. *arXiv preprint arXiv:1812.01600*, 2018.

Pinckaers, H., van Ginneken, B., and Litjens, G. Streaming convolutional neural networks for end-to-end learning with multi-megapixel images. *arXiv preprint arXiv:1911.04432*, 2019.

Pinheiro, P. H. and Collobert, R. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, number EPFL-CONF-199822, 2014.

Ramapuram, J., Diephuis, M., Webb, R., and Kalousis, A. Variational saccading: Efficient inference for large resolution images. *arXiv preprint arXiv:1812.03170*, 2018.

Ranzato, M. On learning where to look. *arXiv preprint arXiv:1405.5488*, 2014.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Sermanet, P., Frome, A., and Real, E. Attention for fine-grained categorization. *arXiv preprint arXiv:1412.7054*, 2014.

Shen, Y., Wu, N., Phang, J., Park, J., Liu, K., Tyagi, S., Heacock, L., Kim, S. G., Moy, L., Cho, K., et al. An interpretable classifier for high-resolution breast cancer screening images utilizing weakly supervised localization. *Medical Image Analysis*, pp. 101908, 2020.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 648–656, 2015.

Uzkent, B. and Ermon, S. Learning when and where to zoom with deep reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12345–12354, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Wang, H., Kembhavi, A., Farhadi, A., Yuille, A. L., and Rastegari, M. Elastic: Improving cnns with dynamic scaling policies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2258–2267, 2019.

Wang, M., Liu, B., and Foroosh, H. Factorized convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 545–553, 2017.

Wang, X., Girshick, R., Gupta, A., and He, K. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7794–7803, 2018.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826, 2018.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057, 2015.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Zamir, A. R., Wu, T.-L., Sun, L., Shen, W. B., Shi, B. E., Malik, J., and Savarese, S. Feedback networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1308–1317, 2017.

Zhao, H., Jia, J., and Koltun, V. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10076–10085, 2020.

Appendix

A. Training

A.1. Learning Rule Derivation

The *REINFORCE* rule naturally emerges if we optimize the log likelihood of the labels, while considering the attended locations as latent variables (Ba et al., 2014). Given a batch of N images, for the log likelihood we get:

$$\sum_{i=1}^{N} \log p(y_i|x_i, w) = \sum_{i=1}^{N} \log \sum_{l^i} p(l^i|x_i, w) p(y_i|l^i, x_i, w) \tag{3}$$

where x_i is the ith image in the batch, y_i is its label, and w are the parameters of our model. $p(l^i|x_i, w)$ is the probability that the sequence of locations l^i is attended for image x_i, and $p(y_i|l^i, x_i, w)$ is the probability of predicting the correct label after attending l^i. Equation 3 describes the log likelihood of the labels in terms of all location sequences that could be attended. $p(y_i|l^i, x_i, w)$ is computed by the classification module, and $p(l^{i}|x_i, w)$ is computed by the location module (see Section A.2.)

We use Jensen's inequality in equation 3 to derive the following lower bound on the log likelihood:

$$\begin{aligned} &\sum_{i=1}^{N} \log p(y_i|x_i, w) \geq \\ &\sum_{i=1}^{N} \sum_{l^i} p(l^i|x_i, w) \log p(y_i|l^i, x_i, w) = F \end{aligned} \tag{4}$$

By maximizing the lower bound F, we expect to maximize the log likelihood. The update rule we use, is the partial derivative of F with respect to w, normalized by the number of images in the batch. We get:

$$\begin{aligned} \frac{1}{N}\frac{\partial F}{\partial w} =& \frac{1}{N} \sum_{i=1}^{N} \sum_{l^i} \Big[p(l^i|x_i, w) \frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \\ & \log p(y_i|l^i, x_i, w) \frac{\partial p(l^i|x_i, w)}{\partial w} \Big] = \\ =& \frac{1}{N} \sum_{i=1}^{N} \sum_{l^i} p(l^i|x_i, w) \Big[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \\ & \log p(y_i|l^i, x_i, w) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \Big] \end{aligned} \tag{5}$$

To derive (5) we used the log derivative trick. As we can see, for each image x_i we need to calculate an expectation according to $p(l^i|x_i, w)$. We approximate each expectation with a Monte Carlo estimator of M samples:

$$\begin{aligned} \frac{1}{N}\frac{\partial F}{\partial w} \approx \frac{1}{N}\frac{\partial \tilde{F}}{\partial w} =& \\ = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{M} \sum_{m=1}^{M} \Big[& \frac{\partial \log p(y_i|l^{i,m}, x_i, w)}{\partial w} + \\ & \log p(y_i|l^{i,m}, x_i, w) \frac{\partial \log p(l^{i,m}|x_i, w)}{\partial w} \Big] \end{aligned} \tag{6}$$

$l^{i,m}$ is the sequence of locations attended during the m-th sample from $p(l^i|x_i, w)$ (we get samples by repeating the processing of image x_i).

In order to reduce the variance of the estimator, we replace $\log p(y_i|l^{i,m}, x_i, w)$ with a reward function $R_{i,m}$, which is equal to 1 when the prediction for x_i in the m-th sample is correct, and 0 otherwise. In addition, we use the baseline technique from (Xu et al., 2015), which corresponds to the exponential moving average of the mean reward $R_{i,m} \forall i, m$, and is updated after processing each training batch. Our baseline is initialized to 0.5, and after the n-th batch we get:

$$b_n = 0.9 \cdot b_{n-1} + 0.1 \cdot \frac{1}{NM} \sum_{i=1}^{NM} R_i^n \tag{7}$$

where R_i^n is the reward for the i-th image in the n-th batch. Since we use M samples for the Monte Carlo estimator of each image, we simply consider that our batch has size NM to simplify notation. Our updated learning rule becomes:

$$\begin{aligned} L_F =& \frac{1}{NM} \sum_{i=1}^{NM} \Big[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \\ & \lambda_f (R_i - b) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \Big] \end{aligned} \tag{8}$$

For simplicity, we drop the subscript of b_n that indicates the batch we are processing. Also, we add a weighting hyperparameter λ_f. Equation 8 is the learning rule we presented in Section 4.1. of our paper, and this concludes our derivation.

A.2. Sampling Approximation

Attending a sequence of locations l^i results from sampling (without replacement) from a series of categorical distribu-

tions. For the probability of the sequence we get:

$$p(l^i|x_i,w)=\prod_{j=1}^{N^{l^i}}\prod_{r=1}^{L_j^{l^i}}\prod_{k=1}^{g}\Bigg[\frac{p_j^{l^i}(l_k|x_i,w)^{u_{j,k,r}^{l^i}}}{\sum_{k=1}^{g}\left[p_j^{l^i}(l_k|x_i,w)\prod_{r'=1}^{r-1}(1-u_{j,k,r'}^{l^i})\right]}\Bigg] \tag{9}$$

where N^{l^i} is the number of categorical distributions (equal to the number of times the location module is applied), $L_j^{l^i}$ is the number of samples we draw from the jth distribution, and g is the total number of candidate locations per distribution. In the example of Fig. 4 in our paper, we consider 3 distributions ($N^{l^i}=3$), $L_1^{l^i}=2$ in the 2nd processing level and $L_2^{l^i}=L_3^{l^i}=1$ in the 3rd, and $g=4$ since we consider a 2×2 grid.

l_k is the k-th out of the g candidate locations, and $p_j^{l^i}(l_k|x_i,w)$ is the probability of selecting l_k in the j-th distribution. $u_{j,k,r}^{l^i}$ is an indicator function that is equal to 1 when location l_k is attended as the r-th sample of the j-th distribution, and 0 otherwise. The denominator in (9) is applicable for $r>1$, and normalizes the distribution after each sample to account for the lack of replacement. $p_j^{l^i}(l_k|x_i,w)$ is computed by the location module, and $u_{j,k,r}^{l^i}$ is the outcome of sampling from the jth categorical distribution.

In order to simplify our implementation of sampling dictated by (9), we introduce two modifications. First, we disregard the normalization factor (denominator in (9)) for each $p_j^{l^i}(l_k|x_i,w)^{u_{j,k,r}^{l^i}}$. This changes the relative weighting between different probabilities, but the identity of the locations that are reinforced to be attended again, remains the same. Second, instead of sampling, we just pick the locations with the $L_j^{l^i}$ highest probabilities. Potential downside is that we miss the opportunity to attend to less probable locations that may have valuable information (less exploration). However, at the beginning of training, all locations start with practically equal probability, and even by picking the top $L_j^{l^i}$ locations, we are able to explore the location space.

B. Experimental Evaluation

B.1. Architectures

In Table 3 we define the building blocks of our model. The complete TNet architecture is defined in Table 4 and 5 for the ImageNet and fMoW datasets, respectively. The baselines BagNet-77/EfficientNet-B0 correspond to the TNet feature extraction module followed by the classification module.

B.1.1. Location Module

The location module (Table 4) receives two inputs. The first one is a feature map of size $5\times 5\times 1024$, which originates from an intermediate layer of the feature extraction module. The spatial dimensions of the feature map are equal to the dimensions of the candidate location grid. Each $1\times 1\times 1024$ vector of the feature map, describes the image region within the corresponding grid cell.

To achieve this, we aim for the receptive field of each pixel in the feature map to align with the image region that it is supposed to describe. In the specific architecture of Table 4, we assume a 5×5 grid of overlapping cells, and an input to the feature extraction module of fixed size 77×77 px. Each grid cell occupies 34.375% of the corresponding input dimension. Based on that, when the 5×5 grid is superimposed onto the 77×77 px input, each cell is approximately of size 27×27 px.

The layer of the feature extraction module with the closest receptive field size, is in the 8th ConvBlock with 29×29 px. However, the effective receptive field size is usually smaller that the actual receptive field size (Luo et al., 2017), as a result, we pick the output feature map of the 13th ConvBlock with receptive field 45×45 px. The spatial dimensions of this feature map are 9×9, and we need to downsample it to 5×5 px. To this end, we calculate the image level coordinates of the receptive field centers of the feature map pixels, and we pick the 25 of them with receptive fields that better align with the assumed candidate image regions. Based on our previous remarks about the effective receptive field size, we don't consider perfect alignment to be crucial.

The second input to the location module provides contextual information, and it is the output feature vector of the feature extraction module. This vector is of size $1\times 1\times 512$, and we concatenate it across the channel dimension at each spatial position of the input feature map, increasing its size to $5\times 5\times 1536$.

We pass the combined feature map through two 1×1 convolutional layers. The first one fuses the features with the concatenated context. The second one projects each fused vector to a logit value, which represents the relative importance of the corresponding candidate location.

We use the same weights to estimate the importance of each candidate location (1×1 convolutions). We don't want to use different sets of weights (e.g., to have 25 output heads (Uzkent & Ermon, 2020)), because this doesn't allow information learned for one location to transfer to other locations. Also, less attended locations (e.g., corners) could lead to a partially trained model with erratic behavior.

The downside of 1×1 convolutions is that they disregard the spatial information. To mitigate this problem, we enrich the

Table 3. Building blocks of our architectures. ConvBlock and MBConv$F$ have residual connections that add the input to the output. If $s > 1$ or the number of input channels is not equal to the output channels, MBConv$F$ drops the residual connection. If the same conditions hold true for the ConvBlock, it applies an 1×1 convolution with stride s and channels $4 \cdot C$ to the input before it is added to the output. Also, if p is VALID, a margin of total $(k-1)^2$ pixels is dropped from the spatial dimensions of the input before it is passed through the residual connection. The first layer in MBConv$F$ is performed only if $F > 1$. Both for SE and MBConv$F$, C_{in} corresponds to the number of input channels and is not a parameter of the blocks. Batch Normalizatio (Ioffe & Szegedy, 2015) is applied before the activation. GAP stands for Global Average Pooling, and DWConv for depthwise convolution.

Block Type	Layer/Block Type	Kernel Size	# Channels	Stride	Padding	Batch Norm	Activation
ConvBlock (He et al., 2016)	Conv	1×1	$C/4$	1	SAME	-	Leaky ReLU
	Conv	$k \times k$	$C/4$	s	p	-	Leaky ReLU
	Conv	1×1	C	1	SAME	-	Leaky ReLU
Squeeze and Excitation (SE) (Hu et al., 2018)	GAP	-	C_{in}	-	-	-	-
	Conv	1×1	C_r	1	SAME	-	SiLU
	Conv	1×1	C_{in}	1	SAME	-	Sigmoid
MBConv$F$ (Tan et al., 2019)	Conv	1×1	$C_{in} \cdot F$	1	SAME	$\surd$	SiLU
	DWConv	$k \times k$	$C_{in} \cdot F$	1	SAME	$\surd$	SiLU
	SE	-	$C_{in} \cdot 0.25$	-	-	-	-
	Conv	1×1	C	1	SAME	$\surd$	-

Table 4. TNet architecture used on ImageNet. For simplicity, we provide only the spatial dimensions of the feature extraction module's output. The location module receives two inputs and combines them into a single input feature map of size $5 \times 5 \times 1538$; the process is described in Section B.1. The positional encoding module receives a feature vector and a positional encoding vector, and concatenates them to a 1×1024 input vector.

Module	Layer/Block Type	Kernel Size	# Channels	Stride	Padding	Activation	Output Size	Receptive Field
Feature Extraction	Input	-	-	-	-	-	77×77	-
	Conv	3×3	64	1	VALID	Leaky ReLU	75×75	3×3
	ConvBlock	3×3	256	2	SAME	-	38×38	5×5
	ConvBlock	3×3	256	1	SAME	-	38×38	9×9
	ConvBlock	1×1	256	1	SAME	-	38×38	9×9
	ConvBlock	3×3	512	2	SAME	-	19×19	13×13
	ConvBlock	3×3	512	1	SAME	-	19×19	21×21
	ConvBlock ($\times 2$)	1×1	512	1	SAME	-	19×19	21×21
	ConvBlock	3×3	1024	2	VALID	-	9×9	29×29
	ConvBlock	3×3	1024	1	SAME	-	9×9	45×45
	ConvBlock ($\times 4$)	1×1	1024	1	SAME	-	9×9	45×45
	ConvBlock	3×3	2048	1	VALID	-	7×7	61×61
	ConvBlock	3×3	2048	1	SAME	-	7×7	77×77
	ConvBlock	1×1	2048	1	SAME	-	7×7	77×77
	Conv	1×1	512	1	SAME	Leaky ReLU	7×7	77×77
	GAP	-	512	-	-	-	1×1	-
Location	Input	-	-	-	-	- -	$5 \times 5 \times 1024$, $1 \times 1 \times 512$	-
	Conv	1×1	512	1	SAME	Leaky ReLU	$5 \times 5 \times 512$	-
	Conv	1×1	1	1	SAME	-	$5 \times 5 \times 1$	-
	L_2 Normalization	-	-	-	-	-	1×25	-
	Softmax	-	-	-	-	-	1×25	-
Positional Encoding	Input	-	-	-	-	- -	1×512, 1×512	-
	Fully Connected	-	512	-	-	-	1×512	-
Classification	Input	-	-	-	-	-	1×512	-
	Fully Connected	-	1000	-	-	-	1×1000	-

Table 5. TNet architecture used on fMoW. For simplicity, we provide only the spatial dimensions of the feature extraction module's output. The positional encoding module receives a 1×320 positional encoding vector that is projected to 1×1280, and then it is added to the second input of the module, which is a 1×1280 feature vector.

Module	Layer/Block Type	Kernel Size	# Channels	Stride	Batch Norm	Activation	Output Size	Receptive Field
Feature Extraction	Input	-	-	-	-	-	224×224	-
	Conv	3×3	32	2	$\surd$	SiLU	112×112	3×3
	MBConv1	3×3	16	1	-	-	112×112	7×7
	MBConv6	3×3	24	2	-	-	56×56	11×11
	MBConv6	3×3	24	1	-	-	56×56	19×19
	MBConv6	5×5	40	2	-	-	28×28	35×35
	MBConv6	5×5	40	1	-	-	28×28	67×67
	MBConv6	3×3	80	2	-	-	14×14	83×83
	MBConv6	3×3	80	1	-	-	14×14	115×147
	MBConv6	3×3	80	1	-	-	14×14	147×115
	MBConv6	5×5	112	1	-	-	14×14	211×211
	MBConv6 ($\times 2$)	5×5	112	1	-	-	14×14	339×339
	MBConv6	5×5	192	2	-	-	7×7	403×403
	MBConv6 ($\times 3$)	5×5	112	1	-	-	7×7	787×787
	MBConv6	3×3	320	1	-	-	7×7	851×851
	Conv	1×1	1280	1	$\surd$	SiLU	7×7	851×851
	GAP	-	1280	-	-	-	1×1	-
Location	Input	-	-	-	-	-	$3 \times 3 \times 80$	-
	Conv	1×1	80	1	-	SiLU	$3 \times 3 \times 80$	-
	SE	-	40	-	-	-	$3 \times 3 \times 80$	-
	Conv	1×1	80	1	-	SiLU	$3 \times 3 \times 80$	-
	Conv	1×1	1	1	-	-	$3 \times 3 \times 1$	-
	L_2 Normalization	-	-	-	-	-	1×9	-
	Softmax	-	-	-	-	-	1×9	-
Positional Encoding	Input	-	-	-	-	-	1×320	-
	Fully Connected	-	1280	-	-	-	1×1280	-
	Input	-	-	-	-	-	1×1280	-
	Add	-	-	-	-	SiLU	1×1280	-
Classification	Input	-	-	-	-	-	1×1280	-
	Fully Connected	-	62	-	-	-	1×62	-

input tensor with positional information according to (Zhao et al., 2020). In particular, for each spatial location, we calculate horizontal and vertical coordinates in the normalized range $[-1, 1]$. We then pass the coordinates through trainable linear layers that map their values to a learned range. The resulting 2-dimensional vectors are concatenated across the channel dimension, resulting in an input feature map of size $5 \times 5 \times 1538$.

The estimated logits are reshaped to a 1×25 vector, which is first normalized to have L_2 norm equal to 1, and then it is passed through a Softmax layer to get the final parameters of the categorical distribution. The L_2 normalization aims to reduce the variance between logits, because we empirically observe that logit values may be too negative, or very close to zero, leading Softmax outputs to be exactly 0, and thus hindering the backpropagation of gradients.

The architecture of the location module in Table 5 is conceptually the same, but has some technical differences. In particular, we provide only one input, the output feature map of the 8th MBConv block (selected and downsampled according to the process described before). This means that we don't provide the output vector of the feature extraction module as an additional input. The reason is that its size of 1×1280 results in a parameter-heavy location module, which is antithetical to the very light design of the feature extraction module.

To inject contextual information to the input feature map, we pass it through a squeeze-and-excitation (SE) block (Hu et al., 2018). Other than that, we follow the design principles described before. We use 1×1 convolutions, we

augment the SE output feature map with 2-dimensional spatial coordinates' vectors, and we normalize the logits.

B.1.2. Positional Encoding Module

The positional encoding module presented in Table 4 receives two inputs. The first one is the output feature vector of the feature extraction module. The second input is a vector that encodes positional information about the image region described by the first input. The encoded positional information is three dimensional; the first 2 dimensions correspond to spatial coordinates, and the 3rd one to scale.

Given processing level l, we assume that a grid is superimposed onto the input image, where its cells correspond to all possible candidate locations of the level. In the example of Fig. 4 in our paper, in the 1st processing level, the assumed grid is comprised of a single cell. In the second level ($l = 2$) the gird is 2×2, and for $l = 3$ the gird is 4×4.

The spatial coordinates of the grid cells start with $(0, 0)$ in the top left corner, and increase linearly with step 1 both horizontally and vertically. The scale coordinate is equal to $l - 1$. Based on this, each candidate image region has a unique positional triplet (x, y, s) where x, y are the spatial coordinates and s is the scale.

We use sine and cosine functions of different frequencies to encode positional triplets (x, y, s) according to (Vaswani et al., 2017). In particular, for positional encodings of size $1 \times N$, we get:

$$P_s(p, \vec{t}) = \sin(p \cdot \left(\frac{1}{100}\right)^{\frac{\vec{t}}{\lfloor N/6 \rfloor}}),$$
$$P_c(p, \vec{t}) = \cos(p \cdot \left(\frac{1}{100}\right)^{\frac{\vec{t}}{\lfloor N/6 \rfloor}}),$$
$$p \in [x, y, s]$$
$$\vec{t} = [0, 1, 2, ...\lfloor N/6 \rfloor]$$

The final positional encoding for triplet (x, y, s) results by concatenating $P_s(x, \vec{t})$, $P_c(x, \vec{t})$, $P_s(y, \vec{t})$, $P_c(y, \vec{t})$, $P_s(s, \vec{t})$ and $P_c(s, \vec{t})$.

The reason we selected these positional encodings (instead of e.g., learned positional embeddings (Dosovitskiy et al., 2020)), is that they can generalize to scales and spatial dimensions of arbitrary size, while they encode distance between spatial coordinates in a consistent way across scales. This is useful for our model because it has the potential to extend its processing to an arbitrary number of levels.

The positional encoding module adds a learned component to the process of encoding position, by passing its two concatenated inputs through a trainable linear layer. The positional encoding architecture in Table 5 is slightly different, because we want a relatively small number of parameters. To this end, we use positional encodings of lower dimensionality (1×320), and only this vector is passed through a linear layer (it is projected to 1×1280). Then, the projected vector is simply added to the input feature vector (output of the feature extraction module).

B.2. Training On ImageNet

To train TNet, we use a single sample for the Monte Carlo estimators, and we set $\lambda_f = 0.1$ (eq. 8). The BagNet-77 baseline is trained by minimizing the cross-entropy classification loss.

For both models we use batches of 64 images, distributed in 4 GPUs. We use the Adam optimizer with the default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. We use xavier initialization (Glorot & Bengio, 2010) for the weights, and zero initialization for the biases. For regularization purposes, we use data augmentation that is very similar to the one used in (Szegedy et al., 2015). In particular, given a training image, we get a random crop that covers at least 85% of the image area, while it has an aspect ration between 0.5 and 2.0. Since we provide inputs of fixed size to our networks (224×224 px), we resize the image crops accordingly. Resizing is performed by randomly selecting between 8 different methods, which include bilinear, nearest neighbor, bicubic, and area interpolation. Also, we randomly flip the resized image crops horizontally, and we apply photometric distortions (Howard, 2013). The final image values are scaled in range $[-1, 1]$. Finally, the dropout mentioned is Section 5.1. of our paper, is spatial (Tompson et al., 2015).

B.3. Training On fMoW

We first train TNet with inputs of size 448×448, allowing 2 processing levels. We train for 40 epochs with batches of 64 images (om 4 GPUs), with initial learning rate 0.001 that we drop once by a factor of 0.1. We use the Adam optimizer with its default parameter values, and we follow the weight initialization of (Tan & Le, 2019).

We attend a fixed number of 2 locations. We use $\lambda_f = 0.1$ and per-feature regularization with $\lambda_c = \lambda_r = 0.2$. We use a single sample for the Monte Carlo estimators.

We use dropout before the linear layer of the classification module with 0.5 drop probability. We use stochastic depth (Huang et al., 2016) with drop probability that increases linearly to a maximum value of 0.3. We use the data augmentation technique described in Section B.2.

We finetune TNet for 10 epochs on images of 896×896 px, with a fixed number of 2 attended location in the 2nd processing level, and 1 in the 3rd (4 in total). Compared to the previous step, we increase the maximum drop probability of stochastic depth to 0.5, and we set $\lambda_c = \lambda_r = 0.05$. We also use only features extracted until the 2nd processing level in per-feature regularization.

We use different input images to train four EfficientNet-B0 baselines. For the first baseline we use images cropped according to the bounding box annotations, and resized to 224×224 px. We train for 65 epochs with batches of 64 images, on 4 GPUs. Our initial learning rate is 0.001, and we drop it once by a factor of 0.1. We use the Adam optimizer with its default parameter values, and we follow the weight initialization of (Tan & Le, 2019).

We use dropout before the final classification layer with 0.75 drop probability, and L_2 regularization weight 10^{-5}. We use stochastic depth with drop probability that increases linearly to a maximum value of 0.5. We use the data augmentation technique described in Section B.2.

The second baseline is trained on the original images, resized to 224×224 px. The only difference with the training of the previous baseline is that we train for 60 epochs.

The third baseline is trained on the original images resized to 448×448 px. We train for 30 epochs with batches of 32 images. We reduce the stochastic depth maximum drop probability to 0.3. All other training hyperparameters remain the same.

The fourth baseline is trained on the original images resized to 896×896 px. We train for 30 epochs with batches of 32 images. We set the dropout drop probability to 0.3, and the stochastic depth maximum drop probability to 0.2. All other training hyperparameters remain the same.

B.4. Metrics

We calculate the FLOPs of a convolutional layer in the following way:

$$N_{FLOPs} = (C_{in} \cdot k^2) \cdot (H_{out} \cdot W_{out} \cdot C_{out}) \qquad (11)$$

where C_{in} is the number of channels in the input feature map, $k \times k$ are the spatial dimensions of the convolutional kernel, $H_{out} \times W_{out}$ is the spatial resolution of the output, and C_{out} is the number of output channels. Each time the kernel is applied, we make $(C_{in} \cdot k^2)$ multiplications, and we apply the kernel $(H_{out} \cdot W_{out} \cdot C_{out})$ times (number of output pixels). For fully connected layers, simply holds $k = 1$ and $H_{out} = W_{out} = 1$.

Equation 11 accounts only for multiplications. If we consider additions as well, the number of FLOPs approximately doubles. Based on eq. 11, for our EfficientNet-B0 baseline we get almost identical number of FLOPs to those reported in (Tan & Le, 2019)[2].

[2]The only difference in FLOPs between our baseline and EfficientNet-B0 reported in (Tan & Le, 2019), stems from the size of the output layer, due to different number of classes. For the same reason there is observable difference in the number of parameters between the two models.

We measure the processing time of our models by timing 200 batches of 64 images during inference. In the same setting, we measure memory by using the TensorFlow memory profiler. We disregard the first iterations to avoid any computational and memory overhead that stems from the creation of the TensorFlow graph. Finally, TensorFlow automatically calculates the number of our models' parameters.

C. Attention Policy Examples

In Figure 6 we provide examples of the attention policy on the ImageNet validation set with 3 locations. In Figure 7 we provide examples of the attention policy on the fMoW test set with 2 locations at the 2nd processing level.

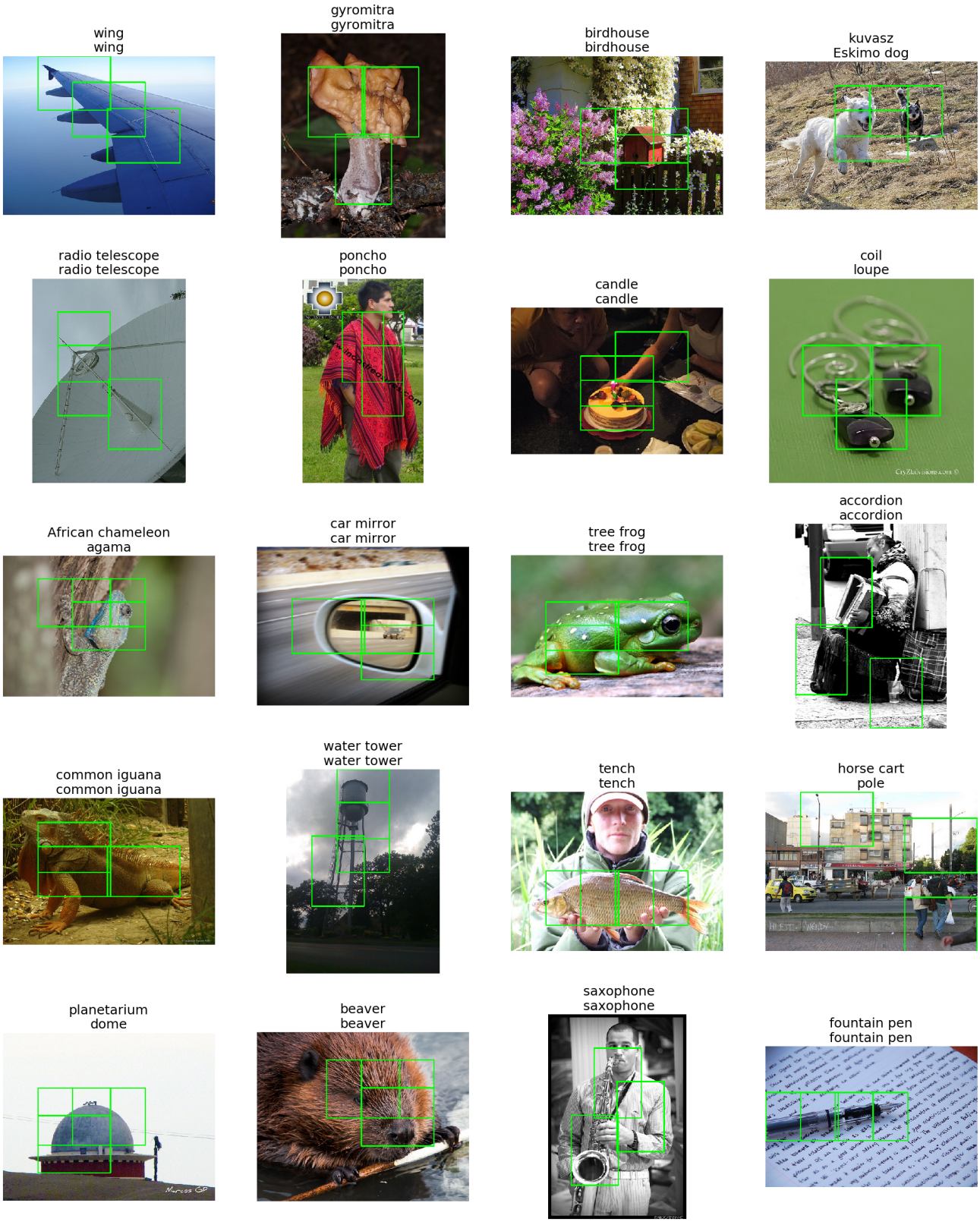

Figure 6. Attention policy examples with 3 locations on the ImageNet validation set. On top of each image we provide the correct and the predicted label.

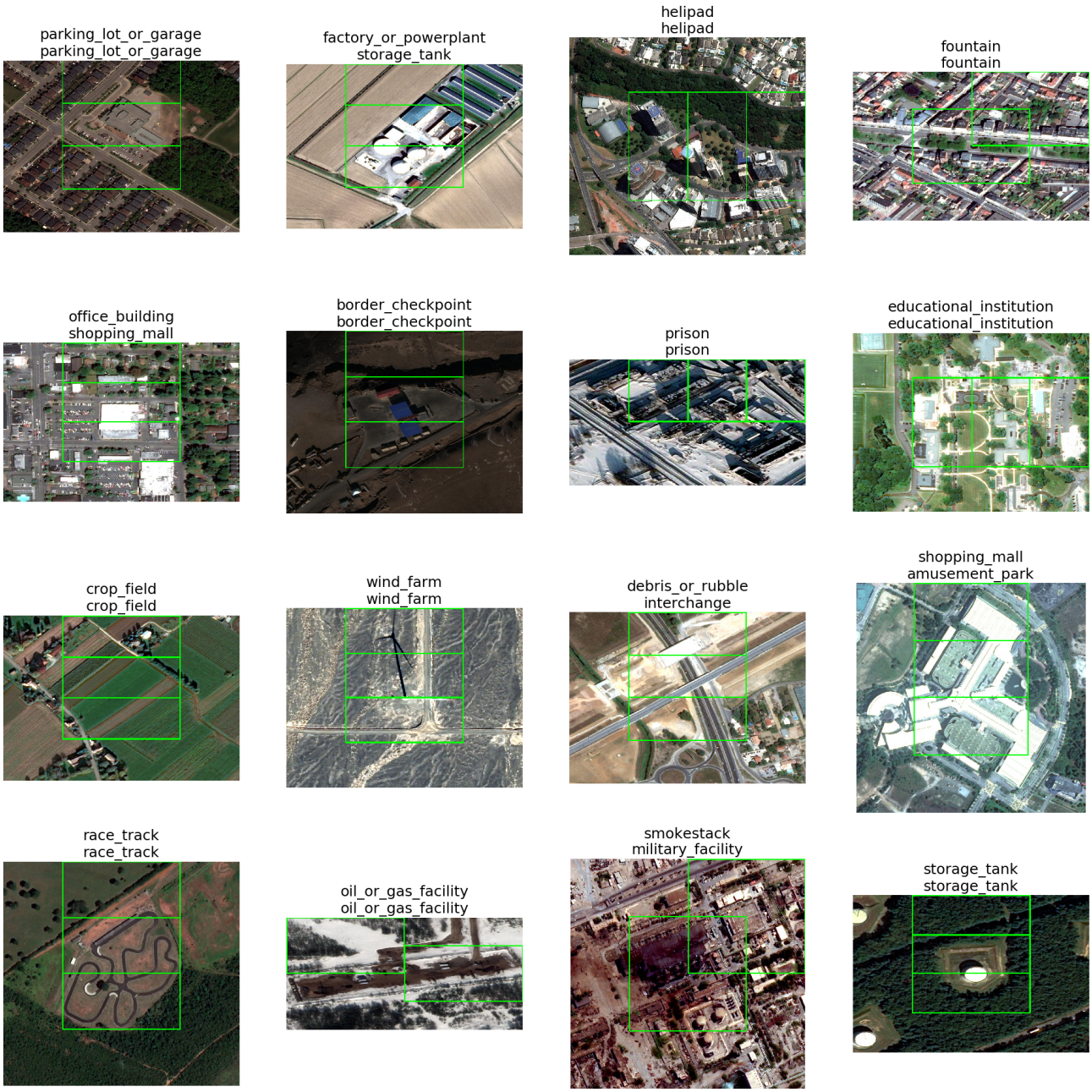

Figure 7. Attention policy examples with 2 locations (2nd processing level) on the fMoW test set. On top of each image we provide the correct and the predicted label.