MIST: Multiple Instance Spatial Transformer

Supplementary Material

A. Heatmap network architectures

Standard architecture. Our standard architecture is the multiscale heatmap network is inspired by LF-Net [38]. We employ a fully convolutional network to produce a single heatmap for each scale indexed by $s = 1 \dots S$, on the input image $\mathcal{I}$. Specifically, for each scale we first scale the image proportionally to the inverse of the scale producing $\mathcal{I}_s$, execute the network $\mathcal{H}_\eta$ on it, and finally scale the heatmap back to the original resolution. This strategy ensures that the network cannot implicitly favor a particular scale and produces scale-independent responses.

This process generates a multiscale heatmap tensor $\mathbf{h} = \{\mathbf{h}_s\}$ of size $H \times W \times S$ where $\mathbf{h}_s = \mathcal{H}_\eta(\mathcal{I}_s)$, and H is the height of the image and W is the width. For the convolutional network we use 4 ResNet blocks [21], where each block is composed of two 3×3 convolutions with 32 channels and relu activations without any downsampling. We then perform a *local spatial softmax* operator [38] with spatial extent of 15×15 to sharpen the responses. Then we further relate the scores across different scales by performing a "softmax pooling" operation over scale. Specifically, if we denote the heatmap tensor after local spatial softmax as $\tilde{\mathbf{h}} = \{\tilde{\mathbf{h}}_s\}$, since after the local spatial softmax $\mathcal{H}_\eta(\mathcal{I}_s)$ is already an "exponentiated" signal, we do a weighted normalization without an exponential, *i.e.* $\mathbf{h}' = \sum_s \tilde{\mathbf{h}}_s(\tilde{\mathbf{h}}_s / \sum_{s'}(\tilde{\mathbf{h}}_{s'} + \epsilon))$, where $\epsilon = 10^{-6}$ is added to prevent division by zero.

Note that differently from LF-Net [38], we do not perform a softmax along the scale dimension. The scale-wise softmax in LF-Net is problematic as the computation for a softmax function relies on the input to the softmax being *unbounded*. For example, in order for the softmax function to behave as a max function, due to exponentiation, it is necessary that one of the input value reaches infinity (i.e. the value that will correspond to the max), or that all other values to reach negative infinity. However, at the network stage where softmax is applied in [38], the score range from zero to one, effectively making the softmax behave similarly to averaging. Our formulation does not suffer from this drawback.

Backbone-based heatmap network. For experiments on natural images, we restrict the detector to only localize the objects without estimating object scales to simplify the task. Therefore, we only use a single-scale heatmap for this setting. Also, because the number of images in our dataset is limited, we leverage a pretrained ResNet34 [21] as the backbone feature extractor. Specifically, we resize the input images to have a shorter edge of 224 pixels and use the output of fourth convolution block – a tensor of $H/16 \times W/16 \times 256$ where H and W are the height and the width of the input image, respectively, and 256 is the number of channels. We further append a 1×1 conv layers to reduce the heatmap to have 3 channels: response, x-offset, and y-offset. We make use of offsets since our heatmap size is only $1/16$ of the input image size, and integer pixel coordinates on the heatmap cannot provide accurate localization on the input image. We also do not use a *local spatial softmax* operator for this setting due to small heatmap size. While not using spatial softmax makes our heatmaps similar to the ones in [30], we note that we still rely on top-K, rather than the sampling-without-replacement approach of [30], and is therefore easy to expand to various K.

B. Generative model for the heatmap

Standard architecture. To convert optimized locations into ideal heatmaps (see Section 4.1), as our standard architecture we apply a simple model where the heatmap is zero everywhere else except on the corresponding keypoint locations (patch center). However, as the optimized patch parameters can be floats corresponding to subpixel locations, we need to quantize them with care to turn them back into a heatmap . For the spatial locations we simply round to the nearest pixel, which at most creates a quantization error of half a pixel, which does not cause problems in practice. For scale however, simple nearest-neighbor assignment causes too much quantization error as our scale-space is sparsely sampled. We therefore assign values to the two nearest neighboring scales in a way that the center of mass would be the optimized scale value, making sure $\{\mathbf{x}_k\} = \mathcal{E}_K(\mathcal{G}(\{\mathbf{x}_k\}))$.

Backbone-based heatmap network. For the backbone-based network, as the feature map is very coarse, we found using a single pixel insufficient. Hence, we use a Gaussian kernel to reconstruct the first channel (response channel) of the ideal heatmap from the optimized keypoints:

$$H^r_{\mathbf{p}} = \sum_{i=1}^{k} \exp(-\frac{1}{2}(\lfloor \mathbf{x}_i \rfloor - \mathbf{p})^T \Sigma^{-1} (\lfloor \mathbf{x}_i \rfloor - \mathbf{p})) \ ,$$

where $\lfloor \mathbf{x}_i \rfloor$ are the rounded optimized keypoints, $\mathbf{p}$ is a pixel location on heatmap. We use one eighth of the patch size as the values of the diagonal covariance matrix Σ.

For the x and y offset channels we only supervise pixels which contain optimized keypoints:

$$H^{xy}_{\mathbf{p}} = \mathbf{x}_i - \lfloor \mathbf{x}_i \rfloor \quad \text{if } \mathbf{p} = \lfloor \mathbf{x}_i \rfloor_i \in \mathbf{x} \ . \tag{10}$$

To train the detector, we use the ℓ_2 loss for response channel as in Equation 7, and for the offset channels we use a Huber

loss [23] as in [17]:

$$\text{loss} = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}. \tag{11}$$

C. Additional implementation details

C.1. Task-specific networks

MIST auto-encoder network. The input layer of the autoencoder is $32 \times 32 \times C$ where C is the number of color channels. We use five up/down-sampling levels. Each level is made of three standard non-bottleneck ResNet v1 blocks [21] and each ResNet block uses a number of channels that doubles after each downsampling step. ResNet blocks use 3×3 convolutions of stride 1 with ReLU activation. For downsampling we use 2D max pooling with 2×2 stride and kernel. For upsampling we use 2D transposed convolutions with 2×2 stride and kernel. The output layer uses a sigmoid function, and we use layer normalization before each convolution layer.

MIST classification network for MNIST dataset. We use the same architecture as the encoder part of the auto-encoder and append a dense layer to it to map the latent space to the score vector of our 10 digit classes.

MIST classification network for PASCAL+COCO dataset. We crop patches at keypoint locations on *the feature map* from fourth convolution block and feed the patches into a single ResNet block – same as the one used for the auto-encoder network – followed by a global average pooling layer, and a dense layer that converts the output into 21 logit values for classification.

C.2. Implementations of compared methods

Baseline unsupervised reconstruction methods. To implement the *Esl16* [12] baseline, we use a publicly available third-party implementation[1]. We note that their method was originally applied to a dataset consisting of images of 0, 1, or 2 digits with equal probability. We found that the model failed to converge unless it was trained with examples where various number of total digits exist, so for fair comparison, we populate the training set with images consisting of all numbers of digits between 0 and 9. For the *Zha18* [62] baseline, we use the authors' implementation and hyperparameters.

Differentiable top-K (*Xie20*) [57]. We implemented their method as a PyTorch module according to the pseudo-code provided in [57]. In addition to the provided pseudo-code, we add a small epsilon clipping for the division operations within the equations for stability. The differentiable top-K operation in [57] outputs a top-k selection mask $\mathbf{m} \in (0,1)^N$, where N is number of elements to select from – top-K elements have a mask score close to 1, and non selected elements have a mask score close to 0. To apply this method to our task, one needs to then apply this mask to the classification results of patches at all heatmap location, as the operation is not indexing, but rather masking – this is how differentiability is obtained in this method. It is therefore necessary that all results that gets masks to stay in memory, and requires a smaller heatmap to be trained on reasonable system – we use a *GeForce RTX 2080 Ti* with 11GB memory. In addition, we modify the classification loss in Eq.(9) to incorporate the selection mask:

$$\mathcal{L}_{\text{task}} = \left\| \frac{1}{L} \sum_{l=1}^{L} \mathbf{y}_l - \frac{1}{K} \hat{\mathbf{p}} \times \mathbf{m} \right\|_2^2, \tag{12}$$

where L is the number of instances in the image, $\hat{\mathbf{p}}$ is the $C \times N$ classification score matrix, $\mathbf{m}$ is the mask vector of size N, with C being the number of classes. Note that the only difference here is that the top-K selection via indexing in the main paper has know been transformed into a mask-based selection.

D. Non-uniform distributions

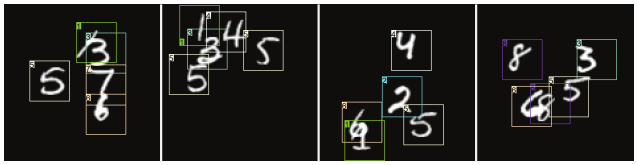

Figure 8. Examples with uneven distributions of digits.

Although the images we show in Figure 2 involve small displacements from a uniformly spaced grid, our method does not require the keypoints to be evenly spread. As shown in Figure 8, our method is able to successfully learn even when the digits are placed unevenly. Note that, as our detector is fully convolutional and local, it does not learn the absolute location of keypoints. In fact, we weakened the randomness of the locations for fairness against [62], which is not designed to deal with severe displacements.

E. Performance on the full PASCAL VOC 2007 dataset with multiple anchors

The curious reader may be intrigued how the method performs when the method is tested on a more unrestricted setup. While this is out of scope, since introducing new dimensions such as scale require more than just the classification itself – one needs to understand the extent of an object – we report results when four anchors with different scale and aspect ratio are considered Specifically, we resize the images to

[1] https://github.com/aakhundov/tf-attend-infer-repeat

have a shorter edge of 224 pixels to be compatible with the pre-trained part of the network, and use four anchors each having the size of 56×56, 56×168, 168×56, 168×168, and predict confidence scores for each anchor. The final bounding box is a weighted average of the four anchors based on their confidence scores after a softmax operation.

Even when using the original dataset and introducing scale estimation our method still outperforms Xie20. In this case, the localization performance, in terms of the F1 score on the full dataset was 33.8% for our method, and 29.8% for Xie20, both of which are significantly worse than the simplified task. This is expected, as there is nothing in the loss functions asking for correct localization – the task itself is simply image-level classification. We expect further regularization on how the detection should behave is necessary for a more practical detection setup, but this is out of scope of the current paper, and is left as future work.