Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference
Supplementary material

1. Numeric results

Timings are measured on a Nvidia GTX 1050 Ti 4GB GPU with quad core Intel i7 4770K and 32 GB RAM. Algorithms are implemented in PyTorch 1.2.0 with cuDNN 7.6. PyTorch benchmark mode, which chooses the optimal back-end algorithm for tensor operations, is used for fair comparison. To measure inference speed, 2000 validation images are preloaded. The first 1000 images are used in the warmup phase and processing speed of the last 1000 images is reported as images processed per second (Im/Sec).Theoretical computational complexity is reported in Multiply-Accumulates (MACs). We use a publicly available script [1] to count the amount of operations in convolutional and fully connected layers.

1.1. Classification

1.1.1 ResNet-32 on CIFAR-10

Classification with standard ResNet-32 [4] on CIFAR-10 [6], standard train/test split with 50000 training and 10000 test images. Mask unit is a squeeze unit. See Table 1 for numeric results. Same hyperparameters as ConvNet-AIG [10]. Optimizer: SGD, momentum: 0.9, learning rate: 0.1, batch size: 256, epochs: 350, learning rate decayed at epochs 150 and 250 by factor 0.1. Gumbel temperature $\tau = 1$ and Gumbel noise set to zero at epoch 280. Data augmentation is the same as ResNet [4]: padding of 4 pixels and random crop of 32×32 pixels.

1.1.2 ResNet-101 on ImageNet

Standard ResNet-101 [4] on ILSVRC2012 [2] with 1000 ImageNet classes, using the standard train/validation split with 1.28 million train images and 50000 validation images. Mask unit is a squeeze unit. See Table 2 for results. Optimizer: SGD,

[1] https://github.com/sovrasov/flops-counter.pytorch

Table 1: Results on CIFAR-10

Model	budget θ	Parameters	# masks	Acc (top-1)	MACs	MACs (mask units)
	0.9			93.87	65.45×10^6	1.40×10^6
	0.8			93.66	61.46×10^6	1.40×10^6
	0.7			93.58	55.45×10^6	1.40×10^6
	0.6			93.53	48.11×10^6	1.40×10^6
ResNet-32	0.5	472056	15	93.05	41.08×10^6	1.40×10^6
(DynConv)	0.4			92.57	33.71×10^6	1.40×10^6
	0.3			92.27	26.31×10^6	1.40×10^6
	0.2			91.81	19.55×10^6	1.40×10^6
	0.1			90.44	14.73×10^6	1.40×10^6
ResNet-32		466906		93.57	70.06×10^6	N.A.
ResNet-26		369690		93.32	55.73×10^6	N.A.
ResNet-20	N.A.	272474	N.A	92.89	41.41×10^6	N.A.
ResNet-14		175258		91.47	27.08×10^6	N.A.
ResNet-8		78042		88.04	12.75×10^6	N.A.

Table 2: Results on ImageNet

Model	Budget θ	# Params	# Masks	Top 1 acc.	Top 5 acc.	MACs	MACs (mask units)
ResNet-101 (DynConv)	0.8	44854506	33	78.0	94.0	6418×10^6	80×10^6
	0.7	44854506	33	77.9	93.8	5640×10^6	80×10^6
	0.5	44854506	33	77.0	93.4	4235×10^6	80×10^6
	0.3	44854506	33	75.6	92.7	2974×10^6	80×10^6
ResNet-101 (baseline)	N.A	44549160	N.A.	78.1	94.0	7849×10^6	N.A.
ResNet-50 (baseline)	N.A	25557032	N.A.	76.2	93.0	4121×10^6	N.A.

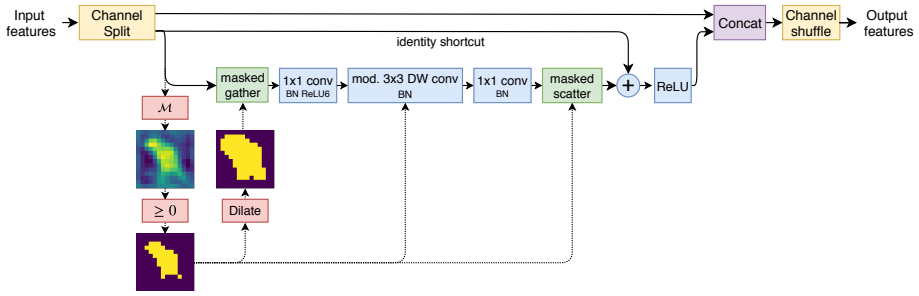


Figure 1: Integration of dynamic convolutions in the residual variant of ShuffleNetV2. The residual block is similar to those of MobileNetV2, but has additional channel split and shuffle operations.

momentum: 0.9, learning rate: 0.025, batch size: 64, total epochs: 50, learning rate decayed at epochs 15, 30 and 45 by factor 0.1. Gumbel temperature $\tau = 5$, decayed to 2 and 1 at epoch 15 and 30 respectively. Gumbel noise set to zero at epoch 40. Model initialized with pretrained ResNet-101 weights. Standard InceptionV3 augmentation [9]. During training, a random crop is resized to size 224×224 with random horizontal flip. During testing, the image is rescaled to 256×256 and a 224×224 center crop is taken.

1.1.3 MobileNetV2 and ShuffleNetV2 on Food-101

Classification with MobileNetV2 [8] and ShuffleNetV2 [12] on Food-101 [6], standard train/test split with 75750 training and 25250 test images equally distributed over 101 classes. The MobileNetV2 model uses width multiplier 0.75, The ShuffleNetV2 model is the residual variant (Fig. 1) with width multiplier 1.0. Since each residual block of ShuffleNetV2 has fewer computations than those of MobileNetV2, we experimented with a 1×1 convolution as mask unit, in addition to the squeeze unit described previously. Optimizer: SGD, momentum: 0.9, learning rate: 0.05, batch size: 64, total epochs: 100, cosine learning rate annealing. Gumbel temperature $\tau = 1$ and Gumbel noise set to zero at epoch 80. Standard InceptionV3 augmentation [9]. During training, a random crop is resized to size 224×224 with random horizontal flip. During testing, the image is rescaled to 256×256 and a 224×224 center crop is taken.Gumbel-Softmax uses the standard formulation, while Gumbel-Binary is our computationally cheaper reformulation. Results in Table 3.

1.2. Pose Estimation

1.2.1 Stacked hourglass on MPII

Pose estimation using a stacked hourglass network [7], where the residual blocks are replaced by those of MobileNetV2 [8] (Fig. 2). The mask unit is a 1×1 convolution.

Note that existing methods for spatial conditional execution, *e.g.* SACT [3], are not directly applicable due to the large amount of branches in the network. Our method operates on each residual block individually.The expand ratio of the residual blocks is 6, *e.g.* 96 features get expanded to 576 features for the depthwise convolution. We use the MPII [1] dataset with the same train/validation split as [7], having respectively 22k and 3k images. Models with different layer width are obtained by

Table 3: Results on Food-101

Model	Parameters	# Masks	Acc (top-1)	MACs	MACs (mask units)	Im/Sec
MobileNetV2 x0.75 (baseline)	1484805	N.A.	82.0	225×10^6	N.A.	508
+ DynConv ($\theta = 0.75$)	1490825	10	81.2	200×10^6	1.82×10^6	541
+ DynConv ($\theta = 0.50$)	1490825	10	80.6	174×10^6	1.82×10^6	629
+ DynConv ($\theta = 0.25$) (Gumbel-Binary)	1490825	10	79.8	148×10^6	1.82×10^6	724
+ DynConv ($\theta = 0.25$) (Gumbel-Softmax)	1490825	10	79.8	148×10^6	1.82×10^6	522
ShuffleNetV2 (baseline)	1357129	N.A.	78.7	149×10^6	N.A.	710
+ DynConv($\theta = 0.25$) (squeeze mask unit)	1373975	13	76.5	100×10^6	3.30×10^6	781
+ DynConv($\theta = 0.25$) (1×1) conv mask unit)	1358824	13	76.3	97×10^6	0.33×10^6	889

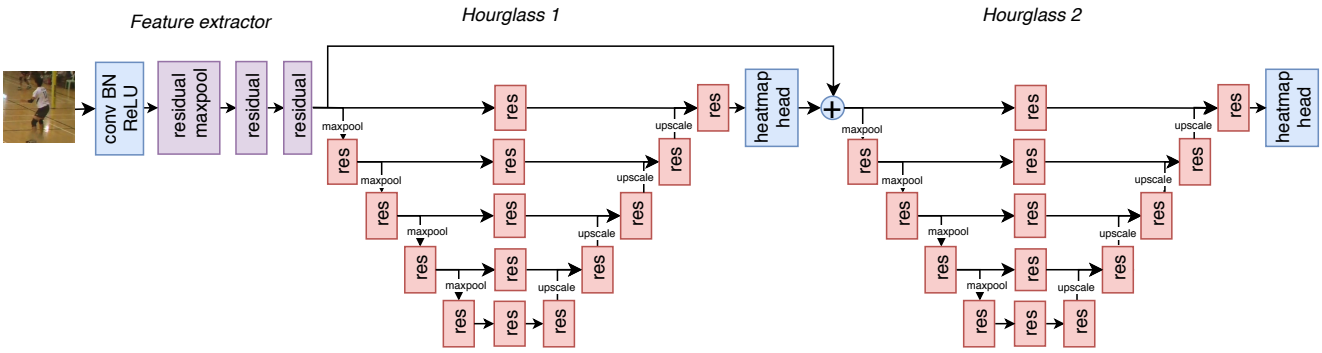


Figure 2: Overview of an hourglass network with 2 stacks. All red residual blocks are executed with dynamic convolutions.

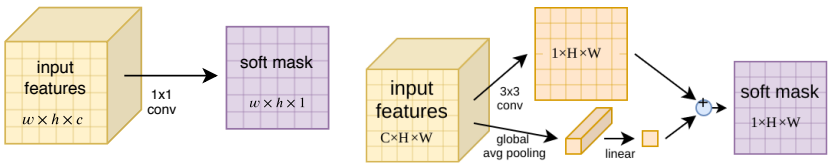
scaling the number of channels with a *width multiplier*. We use the implementation of Fast Pose Distillation [11] [2].

Optimizer: Adam, momentum: 0.9, learning rate: 2e-4, batch size 6. Output heatmaps of size 64×64 with mean-square error loss. Total epochs: 100, learning rate decayed at epochs 60 and 90 by factor 0.1. Gumbel temperature τ fixed to 1 and Gumbel noise set to 0 at epoch 80. Data augmentation: Images are resized to 256×256 and augmented with ±30 degrees rotation, ±25 percent scaling and random horizontal flip. no flip augmentation during evaluation. Results in Table 4. Accuracy is given in PCKh@0.5, defined as the percentage of correct keypoints where the prediction and ground truth are closer than 50% of the head bone link length.

[2] https://github.com/ilovepose/fast-human-pose-estimation.pytorch

Table 4: Results on MPII validation set (pose estimation)

Model	Width mult	# Params	Budget θ	# Masks	PCKh@0.5	MACs	MACs (mask units)	Im/Sec
4-stack (DynConv)	1	6885433	0.75	TODO	88.1	5.39×10^9	0.02×10^9	32
			0.50		88.2	3.78×10^9	0.02×10^9	46
			0.25		87.5	2.30×10^9	0.02×10^9	69
			0.125		86.7	1.71×10^9	0.02×10^9	80
4-stack (baseline)	1	6879904	N.A.	N.A.	88.1	6.88×10^9	N.A.	29
	0.875	5317726			87.6	5.32×10^9		35
	0.75	3956176			87.0	3.96×10^9		39
	0.50	183496			85.2	1.83×10^9		68
2-stack (DynConv)	1	3527101	0.75	TODO	86.7	3.08×10^9	0.01×10^9	37
			0.50		86.5	2.19×10^9	0.01×10^9	77
			0.25		86.1	1.39×10^9	0.01×10^9	104
			0.125		85.3	1.06×10^9	0.01×10^9	130
2-stack (baseline)	1	3524288	N.A.	N.A.	86.6	3.88×10^9	N.A.	51
	0.875	2724278			86.1	3.00×10^9		58
	0.75	2026976			85.5	2.23×10^9		68
	0.50	940496			83.0	1.03×10^9		110
1-stack (DynConv)	1	1.847935	0.75	TODO	83.6	1.89×10^9	0.01×10^9	
			0.50		83.5	1.43×10^9	0.01×10^9	
			0.25		82.7	0.95×10^9	0.01×10^9	
			0.125		81.1	0.74×10^9	0.01×10^9	
1-stack (baseline)	1	1846480	N.A.	N.A.	83.6	2.37×10^9	N.A	
	0.875	1427554			82.6	1.83×10^9		
	0.75	1062376			82.0	1.37×10^9		
	0.50	493264			78.9	0.63×10^9		



(a) 1×1 convolution mask unit (b) squeeze unit

Figure 3: Mask unit architectures

2. Mask unit architecture

The mask unit is a small trainable network that indicates the positions where the spatial 3×3 convolution of the residual block should be evaluated. Figure 3 shows the architecture of the mask units used in our experiments. The mask units are computationally cheap compared to the convolutions in a residual block, since the output of this operation is only a single channel. We observe that the squeeze unit (proposed in [3]) achieves slightly higher accuracy than the 1×1 convolution, but is more complex.

3. Binary Gumbel-Softmax

This section includes the full derivation of Section 3.1.2 in the paper. As shown in Section 4.1.1 in the paper, the simplified formulation of the Gumbel-Softmax for binary cases is needed for faster inference. We start from the formulation by Jang *et al*. [5]. Take a categorical distribution with class probabilities $\boldsymbol{\pi} = \pi_1, \pi_2, ...\pi_n$, then discrete samples z can be drawn using

$$z = \texttt{one\_hot}\big(\arg\max_i \left[log(\pi_i) + g_i\right]\big) \tag{1}$$

with g_i being noise samples drawn from a Gumbel distribution. The Gumbel-Softmax trick defines a continuous, differentiable approximation by replacing the argmax operation with a softmax:

$$y_i = \frac{\exp((log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((log(\pi_j) + g_j)/\tau)}. \tag{2}$$

Gating decisions are binary, which makes it possible to strongly simplify the Gumbel-Softmax formulation. A soft-decision $m \in (-\infty, \infty)$, outputted by a neural network, is converted to a probability π_1 indicating the probability that a pixel should be executed, using a sigmoid σ:

$$\pi_1 = \sigma(m). \tag{3}$$

Then, the probability that a pixel is not executed is

$$\pi_2 = 1 - \sigma(m). \tag{4}$$

Writing out Equation 2 for the binary case ($k = 2$) and $i = 1$ gives

$$y_1 = \frac{\exp\left((\log \pi_1 + g_1)/\tau\right)}{\exp\left((\log \pi_1 + g_1)/\tau\right) + \exp\left((\log \pi_2 + g_2)/\tau\right)} \tag{5}$$

Substituting π_1 and π_2 in Equation 5, for the binary case of $k = 2$ and $i = 1$, makes it possible to reduce this:

$$y_1 = \frac{\exp\left(\frac{\log \sigma(m)+g_1}{\tau}\right)}{\exp\left(\frac{\log \sigma(m)+g_1}{\tau}\right) + \exp\left(\frac{\log (1-\sigma(m))+g_2}{\tau}\right)} \tag{6}$$

$$= \frac{\left(\exp\left(\log \sigma(m) + g_1\right)\right)^{\frac{1}{\tau}}}{\exp\left(\log \sigma(m) + g_1\right)^{\frac{1}{\tau}} + \exp\left(\log\left(1 - \sigma(m)\right) + g_2\right)^{\frac{1}{\tau}}} \tag{7}$$

$$= \frac{\left(\exp\left(\log \sigma(m)\right) \exp\left(g_1\right)\right)^{\frac{1}{\tau}}}{\left(e^{\log \sigma(m)} e^{g_1}\right)^{\frac{1}{\tau}} + \left(e^{\log\left(1-\sigma(m)\right)} e^{g_2}\right)^{\frac{1}{\tau}}} \tag{8}$$

$$= \frac{\left(\sigma(m) e^{g_1}\right)^{\frac{1}{\tau}}}{\left(\sigma(m) e^{g_1}\right)^{\frac{1}{\tau}} + \left((1 - \sigma(m)) e^{g_2}\right)^{\frac{1}{\tau}}} \tag{9}$$

$$= \frac{\left(\sigma(m) e^{g_1}\right)^{\frac{1}{\tau}}}{\left(\sigma(m) e^{g_1}\right)^{\frac{1}{\tau}} + \left((1 - \sigma(m)) e^{g_2}\right)^{\frac{1}{\tau}}} \tag{10}$$

$$= \frac{1}{1 + \frac{\left((1-\sigma(m)) e^{g_2}\right)^{\frac{1}{\tau}}}{\left(\sigma(m) e^{g_1}\right)^{\frac{1}{\tau}}}} \tag{11}$$

$$= \frac{1}{1 + \left(\frac{(1-\sigma(m))}{\sigma(m)} \exp\left(g_2 - g_1\right)\right)^{\frac{1}{\tau}}} \tag{12}$$

$$= \frac{1}{1 + \left(\frac{\left(1 - \frac{1}{1+\exp(-m)}\right)}{\frac{1}{1+\exp(-m)}} \exp\left(g_2 - g_1\right)\right)^{\frac{1}{\tau}}} \tag{13}$$

$$= \frac{1}{1 + \left((1 - 1 + \exp(-m)) \exp\left(g_2 - g_1\right)\right)^{\frac{1}{\tau}}} \tag{14}$$

$$= \frac{1}{1 + \left(\exp(-m) \exp\left(g_2 - g_1\right)\right)^{\frac{1}{\tau}}} \tag{15}$$

$$= \frac{1}{1 + \exp\left(\frac{-m+g_2-g_1}{\tau}\right)} \tag{16}$$

$$= \sigma\left(\frac{m + g_1 - g_2}{\tau}\right). \tag{17}$$

4. Implementation Details

This section elaborates on the implementation of our components. Our PyTorch implementation is available at https://github.com/thomasverelst/dynconv. We use residual blocks of MobileNetV2, but our method is generally applicable on different architectures of residual blocks. For simplicity of notation, we only consider residual blocks where the dimensions of the block's input features are the same as the dimensions of the block's output. The architecture of a MobileNetV2 block with dynamic convolutions is given in Algorithm 4. Pseudocode is given in Algorithm 1 and the next sections elaborate on the individual functions.

Algorithm 1 Residual block with dynamic convolutions

procedure RESIDUALBLOCK(I)
- $M \leftarrow \mathcal{M}(I)$
- $T \leftarrow$ `Gather`(I, M)
- $T \leftarrow 1{\times}1$ *convolution*(T)
- $T \leftarrow ReLU(batchnorm(T))$
- $S \leftarrow$ `ModifiedDepthwiseConvolution`(T, M)
- $S \leftarrow ReLU(batchnorm(S))$
- $S \leftarrow 1{\times}1$ *convolution*(S)
- $S \leftarrow batchnorm(S)$
- $R \leftarrow$ `Scatter`(S, M)
- **return** $R + I$

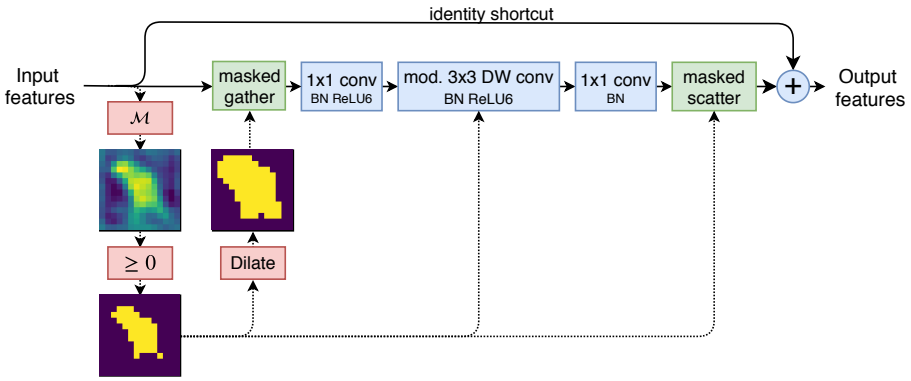


Figure 4: Architecture of a residual block for efficient sparse inference. The mask unit $\mathcal{M}$ generates a mask based on the block's input. The gather operation uses the mask to copy selected spatial positions (yellow) to a new intermediate tensor. Non-spatial operations use standard implementations, while the 3×3 convolution is modified to operate on the intermediate tensor.

4.1. Gather-Scatter Operation

The input I of a residual block is a 4D tensor with dimensions $N \times C \times H \times W$. A single input element has indices n, c, h, w. A spatial position is evaluated when a value of the mask $M = \mathcal{M}(I)$ is greater than or equal to 0. To avoid gaps in the input of the 3×3 convolution, the mask M is morphologically dilated, resulting in a new mask D. The *gather* operation selects all channels of I at spatial positions indicated by D puts them sequentially in a new contiguous tensor T with dimensions $P \times C \times 1 \times 1$. This tensor can be seen as a large batch of 1×1 images with C channels. Non-spatial operations such 1×1 convolutions, ReLU and batchnorm can be evaluated with standard implementations. Algorithm 2 gives psuedocode for the gather operation and Algorithm 4 is used in the gather operation to map between I and T. The scatter operation (Alg. 3) applies the inverse operation. Our CUDA implementation executes most operations in parallel.

Algorithm 2 Gather operation

procedure GATHER(I, M)
  # input: input features $I \in \mathbb{R}^{N \times C \times H \times W}$
  # input: mask $M \in \mathbb{R}^{N \times H \times W}$
  # output: intermediate tensor $T \in \mathbb{R}^{P \times C \times 1 \times 1}$
  $D \leftarrow \texttt{dilate}(M)$
  $L \leftarrow \texttt{TensorToList}(D)$
  $P \leftarrow \texttt{length}(L)$
  $T \leftarrow$ empty tensor $\in \mathbb{R}^{P \times C \times 1 \times 1}$

  for $p = 0..P$ **do**
    $flattened\_position \leftarrow L[p]$

    # Convert flattened position to batch, width and height indices
    $n \leftarrow flattened\_position\ /(H \times W)$
    $h \leftarrow (flattened\_position/W)\ \%\ H$
    $w \leftarrow flattened\_position\ \%\ W$

    # Copy elements to intermediate tensor
    for channel index $c = 0..C$ **do**
      $T_{p,c,1,1} \leftarrow I_{n,c,h,w}$
  return T

Algorithm 3 Scatter operation

procedure SCATTER(S, M)
  # input: intermediate tensor $S \in \mathbb{R}^{P_S \times C \times 1 \times 1}$
  # input: mask $M \in \mathbb{R}^{N \times H \times W}$
  # output: output features $O \in \mathbb{R}^{N \times C \times H \times W}$
  $L \leftarrow \texttt{TensorToList}(M)$
  $P_S \leftarrow \texttt{length}(L)$
  $O \leftarrow$ tensor initialized with $0 \in \mathbb{R}^{N \times C \times H \times W}$

  for $p = 0..P_S$ **do**
    $flattened\_position \leftarrow L[p]$

    # Convert flattened position to batch, width and height indices
    $n \leftarrow flattened\_position\ /(H \times W)$
    $h \leftarrow (flattened\_position/W)\ \%\ H$
    $w \leftarrow flattened\_position\ \%\ W$

    # Copy elements to output tensor
    for channel index $c = 1..C$ **do**
      $I_{n,c,h,w} \leftarrow S_{p,c,1,1}$
  return S

Algorithm 4 Mask-to-list function

procedure TENSORTOLIST(A)
  # input: tensor $A \in \mathbb{R}^{N \times H \times W}$
  # output: list $L \in \mathbb{Z}^P$ of positions where $T \geq 0$
  $L \leftarrow$ empty list
  for batch index $n = 0..N$ **do**
    for height index $h = 0..H$ **do**
      for width index $w = 0..W$ **do**
        if $A_{n,h,w} > 0$ **then**
          $flattened\_position \leftarrow n * H * W + h * W + w$
          $L.add(flattened\_position)$
  return L

4.2. Modified 3x3 Depthwise Convolution Operation

The depthwise convolution applies a 3×3 convolutional kernel to each channel separately. We implement a custom CUDA kernel that applies the channelwise filtering efficiently on T. The spatial relation between elements of T is lost due to its dimensions being $P \times C \times 1 \times 1$. When processing an element t in T, our implementation retrieves the memory locations of spatial neighbors using an index mapping from I to T. The output is a new intermediate tensor S where the number of elements in S is smaller than or equal to the number of elements in T.

Algorithm 5 Modified depthwise convolution

procedure MODIFIEDDEPTHWISECONVOLUTION(T, M)
  # input: intermediate tensor $T \in \mathbb{R}^{P \times C \times 1 \times 1}$
  # input: mask $M \in \mathbb{R}^{N \times H \times W}$
  # output: intermediate tensor $S \in \mathbb{R}^{P_S \times C \times 1 \times 1}$
  # the convolution weight matrix is $W \in \mathbb{R}^{C \times 1 \times 3 \times 3}$

  # Make index mapping from I to T
  $D \leftarrow \texttt{dilate}(M)$
  $Q \leftarrow$ empty tensor $\in \mathbb{Z}^{N \times H \times W}$
  $i \leftarrow 0$
  for batch index $n = 0..N$ **do**
    for height index $h = 0..H$ **do**
      for width index $w = 0..W$ **do**
        if $D_{n,h,w} > 0$ **then**
          $i \leftarrow i + 1$
          $Q_{n,h,w} \leftarrow i$

  $L \leftarrow \texttt{TensorToList}(M)$
  $P_S \leftarrow \texttt{length}(L)$
  $S \leftarrow$ empty tensor $\in \mathbb{R}^{P_S \times C \times 1 \times 1}$
  for $p = 1..P_S$ **do**
    $flattened\_position \leftarrow L[p]$

    # Convert flattened position to batch, height and width indices
    $n \leftarrow flattened\_position\ /(H \times W)$
    $h \leftarrow (flattened\_position/W)\ \%\ H$
    $w \leftarrow flattened\_position\ \%\ W$

    # Apply convolution on all channels
    for channel index $c = 0..C$ **do**
      $val \leftarrow 0$
      for kernel offset $kh = 0..3$ **do**
        for kernel offset $kw = 0..3$ **do**
          if $h + kh - 1 \geq 0$ and $h + kh - 1 < H$ **then**
            if $w + kw - 1 \geq 0$ and $w + kw - 1 < W$ **then**
              $weight \leftarrow W_{c,1,kh,kw}$
              $t \leftarrow Q_{n,h+kh-1,w+kw-1}$
              $val \leftarrow val + weight \times T_{t,c,1,1}$
      $t \leftarrow Q_{n,h,w}$
      $S_{t,c,1,1} \leftarrow val$
  return S

4.3. Other tweaks

PyTorch with benchmark mode enabled is faster when the input matrices of convolutions have regular dimensions. Therefore the batch dimension of matrices T and S is rounded up to the nearest multiple of 64. The padded elements are not initialized and ignored by the scatter operation.

References

[1] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 2

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009. 1

[3] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1039–1048, 2017. 2, 5

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. 5

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 1, 2

[7] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *Proceedings of the European conference on Computer Vision (ECCV)*, pages 483–499. Springer, 2016. 2

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 2

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 2

[10] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018. 1

[11] Feng Zhang, Xiatian Zhu, and Mao Ye. Fast human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3517–3526, 2019. 3

[12] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 2