Spatially Adaptive Computation Time for Residual Networks

Michael Figurnov[1*] Maxwell D. Collins[2] Yukun Zhu[2] Li Zhang[2] Jonathan Huang[2]
Dmitry Vetrov[1,3] Ruslan Salakhutdinov[4]
[1]National Research University Higher School of Economics [2]Google Inc.
[3]Yandex [4]Carnegie Mellon University
michael@figurnov.ru {maxwellcollins,yukun,zhl,jonathanhuang}@google.com
vetrovd@yandex.ru rsalakhu@cs.cmu.edu

Abstract

This paper proposes a deep learning architecture based on Residual Network that dynamically adjusts the number of executed layers for the regions of the image. This architecture is end-to-end trainable, deterministic and problem-agnostic. It is therefore applicable without any modifications to a wide range of computer vision problems such as image classification, object detection and image segmentation. We present experimental results showing that this model improves the computational efficiency of Residual Networks on the challenging ImageNet classification and COCO object detection datasets. Additionally, we evaluate the computation time maps on the visual saliency dataset cat2000 and find that they correlate surprisingly well with human eye fixation positions.

1. Introduction

Deep convolutional networks gained a wide adoption in the image classification problem [23, 38, 39] due to their exceptional accuracy. In recent years deep convolutional networks have become an integral part of state-of-the-art systems for a diverse set of computer vision problems such as object detection [34], image segmentation [32], image-to-text [22, 42], visual question answering [11] and image generation [9]. They have also been shown to be surprisingly effective in non-vision domains, *e.g.* natural language processing [44] and analyzing the board in the game of Go [37].

A major drawback of deep convolutional networks is their huge computational cost. A natural way to tackle this issue is by using attention to guide the computation, which is similar to how biological vision systems operate [35]. Glimpse-based attention models [26, 33, 2, 20] assume that the problem at hand can be solved by carefully processing a small number of typically rectangular sub-regions of the image. This makes such models unsuitable for multi-output problems (generating box proposals in object detection) and per-pixel prediction problems (image segmentation, image generation). Additionally, choosing the glimpse positions requires designing a separate prediction network or a heuristic procedure [1]. On the other hand, soft spatial attention models [42, 36] do not allow to save computation since they require evaluating the model at all spatial positions to choose per-position attention weights.

Figure 1: Left: object detections. Right: feature extractor SACT ponder cost (computation time) map for a COCO validation image. The proposed method learns to allocate more computation for the object-like regions of the image.

We build upon the Adaptive Computation Time (ACT) [12] mechanism which was recently proposed for Recurrent Neural Networks (RNNs). We show that ACT can be applied to dynamically choose the number of evaluated layers in Residual Network [15, 16] (the similarity between Residual Networks and RNNs was explored in [29, 13]). Next, we propose Spatially Adaptive Computation Time (SACT) which adapts the amount of computation between spatial positions. While we use SACT mechanism for Residual Networks, it can potentially be used for convolutional LSTM [41] models for video processing [28].

SACT is an end-to-end trainable architecture that incorporates attention into Residual Networks. It learns a deterministic policy that stops computation in a spatial position as soon as the features become "good enough". Since SACT maintains the alignment between the image and the feature maps, it is well-suited for a wide range of computer vision

*This work was done while M. Figurnov was an intern at Google.

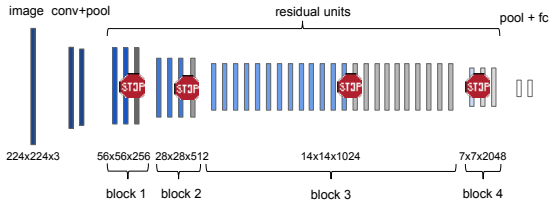


Figure 2: Residual Network (ResNet) with 101 convolutional layers. Each residual unit contains three convolutional layers. We apply Adaptive Computation Time to each block of ResNet to learn an image-dependent policy of stopping the computation.

problems, including multi-output and per-pixel prediction problems.

We evaluate the proposed models on the ImageNet classification problem [8] and find that SACT outperforms both ACT and non-adaptive baselines. Then, we use SACT as a feature extractor in the Faster R-CNN object detection pipeline [34] and demonstrate results on the challenging COCO dataset [31]. Example detections and a ponder cost (computation time) map are presented in fig. 1. SACT achieves significantly superior FLOPs-quality trade-off to the non-adaptive ResNet model. Finally, we demonstrate that the obtained computation time maps are well-correlated with human eye fixations positions, suggesting that a reasonable attention model arises in the model automatically without any explicit supervision.

2. Method

We begin by outlining the recently proposed deep convolutional model Residual Network (ResNet) [15, 16]. Then, we present Adaptive Computation Time, a model which adaptively chooses the number of residual units in ResNet. Finally, we show how this idea can be applied at the spatial position level to obtain Spatially Adaptive Computation Time model.

2.1. Residual Network

We first describe the ResNet-101 ImageNet classification architecture (fig. 2). It has been extended for object detection [15, 7] and image segmentation [6] problems. The models we propose are general and can be applied to any ResNet architecture. The first two layers of ResNet-101 are a convolution and a max-pooling layer which together have a total stride of four. Then, a sequence of four blocks is stacked together, each block consisting of multiple stacked *residual units*. ResNet-101 contains four blocks with 3, 4, 23 and 3 units, respectively. A residual unit has a form $F(\mathbf{x}) = \mathbf{x} + f(\mathbf{x})$, where the first term is called a shortcut connection and the second term is a residual function. A residual function consists of three convolutional layers: 1×1 layer that reduces the number of channels, 3×3 layer that has equal number of input and output channels and 1×1 layer

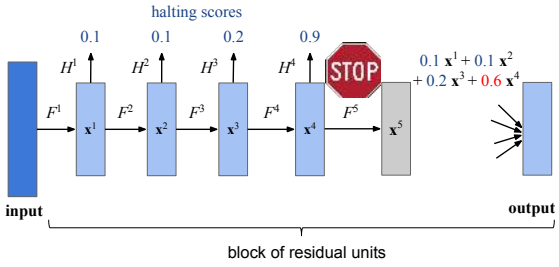


Figure 3: Adaptive Computation Time (ACT) for one block of residual units. The computation halts as soon as the cumulative sum of the halting score reaches 1. The remainder is $R = 1 - h^1 - h^2 - h^3 = 0.6$, the number of evaluated units $N = 4$, and the ponder cost is $\rho = N + R = 4.6$. See alg. 1. ACT provides a deterministic and end-to-end learnable policy of choosing the amount of computation.

that restores the number of channels. We use *pre-activation* ResNet [16] in which each convolutional layer is preceded by batch normalization [19] and ReLU non-linearity. The first units in blocks 2-4 have a stride of 2 and increases the number of output channels by a factor of 2. All other units have equal input and output dimensions. This design choice follows Very Deep Networks [38] and ensures that all units in the network have an equal computational cost (except for the first units of blocks 2-4 having a slightly higher cost).

Finally, the obtained feature map is passed through a global average pooling layer [30] and a fully-connected layer that outputs the logits of class probabilities. The global average pooling ensures that the network is *fully convolutional* meaning that it can be applied to images of varying resolutions without changing the network's parameters.

2.2. Adaptive Computation Time

Let us first informally explain Adaptive Computation Time (ACT) before describing it in more detail and providing an algorithm. We add a branch to the outputs of each residual unit which predicts a *halting score*, a scalar value in the range $[0, 1]$. The residual units and the halting scores are evaluated sequentially, as shown in fig. 3. As soon as the cumulative sum of the halting score reaches one, all following residual units in this block will be skipped. We set the halting distribution to be the evaluated halting scores with the last value replaced by a *remainder*. This ensures that the distribution over the values of the halting scores sums to one. The output of the block is then re-defined as a weighted sum of the outputs of residual units, where the weight of each unit is given by the corresponding probability value. Finally, a *ponder cost* is introduced that is the number of evaluated residual units plus the remainder value. Minimizing the ponder cost increases the halting scores of the non-last residual units making it more likely that the computation would stop earlier. The ponder cost is then multiplied by a constant τ

and added to the original loss function. ACT is applied to each block of ResNet independently with the ponder costs summed.

Formally, we consider a block of L residual units (boldface denotes tensors of shape Height $\times$ Width $\times$ Channels):

$$\mathbf{x}^0 = \textbf{input}, \tag{1}$$

$$\mathbf{x}^l = F^l(\mathbf{x}^{l-1}) = \mathbf{x}^{l-1} + f^l(\mathbf{x}^{l-1}),\ l = 1 \dots L, \tag{2}$$

$$\textbf{output} = \mathbf{x}^L. \tag{3}$$

We introduce a halting score $h^l \in [0, 1]$ for each residual unit. We define $h^L = 1$ to enforce stopping after the last unit.

$$h^l = H^l(\mathbf{x}^l),\ l = 1 \dots (L-1), \tag{4}$$

$$h^L = 1. \tag{5}$$

We choose the halting score function to be a simple linear model on top of the pooled features:

$$h^l = H^l(\mathbf{x}^l) = \sigma(W^l \operatorname{pool}(\mathbf{x}^l) + b^l), \tag{6}$$

where pool is a global average pooling and $\sigma(t) = \frac{1}{1+\exp(-t)}$.

Next, we determine N, the number of residual units to evaluate, as the index of the first unit where the cumulative halting score exceeds $1 - \varepsilon$:

$$N = \min\left\{n \in \{1 \dots L\} : \sum_{l=1}^{n} h^l \geq 1 - \varepsilon\right\}, \tag{7}$$

where ε is a small constant (*e.g.*, 0.01) that ensures that N can be equal to 1 (the computation stops after the first unit) even though h^1 is an output of a sigmoid function meaning that $h^1 < 1$.

Additionally, we define the remainder R:

$$R = 1 - \sum_{l=1}^{N-1} h^l. \tag{8}$$

Due to the definition of N in eqn. (7), we have $0 \leq R \leq 1$.

We next transform the halting scores into a *halting distribution*, which is a discrete distribution over the residual units. Its property is that all the units starting from $(N+1)$-st have zero probability:

$$p^l = \begin{cases} h^l & \text{if } l < N, \\ R & \text{if } l = N, \\ 0 & \text{if } l > N. \end{cases} \tag{9}$$

The output of the block is now defined as the outputs of residual units weighted by the halting distribution. Since representations of residual units are compatible with each other [18, 13], the weighted average also produces a feature representation of the same type. The values of $\mathbf{x}^{N+1}, \dots, \mathbf{x}^L$ have zero weight and therefore their evaluation can be skipped:

$$\text{output} = \sum_{l=1}^{L} p^l \mathbf{x}^l = \sum_{l=1}^{N} p^l \mathbf{x}^l. \tag{10}$$

Ideally, we would like to directly minimize the number of evaluated units N. However, N is a piecewise constant function of the halting scores that cannot be optimized with gradient descent. Instead, we introduce the ponder cost ρ, an almost everywhere differentiable upper bound on the number of evaluated units N (recall that $R \geq 0$):

$$\rho = N + R. \tag{11}$$

When differentiating ρ, we ignore the gradient of N. Also, note that R is not a continuous function of the halting scores [25]. The discontinuities happen in the configurations of halting scores where N changes value. Following [12], we ignore these discontinuities and find that they do not impede training. Algorithm 1 shows the description of ACT.

The partial derivative of the ponder cost w.r.t. a halting

Algorithm 1 Adaptive Computation Time for one block of residual units. ACT does not require storing the intermediate residual units outputs.

Input: 3D tensor **input**
Input: number of residual units in the block L
Input: $0 < \varepsilon < 1$
Output: 3D tensor **output**
Output: ponder cost ρ

$\mathbf{x} = \textbf{input}$
$c = 0$ ▷ Cumulative halting score
$R = 1$ ▷ Remainder value
output $= 0$ ▷ Output of the block
$\rho = 0$
for $l = 1 \dots L$ **do**
  $\mathbf{x} = F^l(\mathbf{x})$
  if $l < L$ **then** $h = H^l(\mathbf{x})$
  else $h = 1$
  end if
  $c \mathrel{+}= h$
  $\rho \mathrel{+}= 1$
  if $c < 1 - \varepsilon$ **then**
    output $\mathrel{+}= h \cdot \mathbf{x}$
    $R \mathrel{-}= h$
  else
    output $\mathrel{+}= R \cdot \mathbf{x}$
    $\rho \mathrel{+}= R$
    break
  end if
end for
return output, ρ

score h^l is

$$\frac{\partial \rho}{\partial h^l} = \begin{cases} -1 & \text{if } l < N, \\ 0 & \text{if } l \geq N. \end{cases} \tag{12}$$

Therefore, minimizing the ponder cost increases $h^1, \ldots, h^{N-1}$, making the computation stop earlier. This effect is balanced by the original loss function $\mathcal{L}$ which also depends on the halting scores via the block output, eqn. (10). Intuitively, the more residual units are used, the better the output, so minimizing $\mathcal{L}$ usually increases the weight R of the last used unit's output $\mathbf{x}^N$, which in turn decreases $h^1, \ldots, h^{N-1}$.

ACT has several important advantages. First, it adds very few parameters and computation to the base model. Second, it allows to calculate the output of the block "on the fly" without storing all the intermediate residual unit outputs and halting scores in memory. For example, this would not be possible if the halting distribution were a softmax of halting scores, as done in soft attention [42]. Third, we can recover a block with any constant number of units $l \leq L$ by setting $h^1 = \cdots = h^{l-1} = 0, h^l = 1$. Therefore, ACT is a strict generalization of standard ResNet.

We apply ACT to each block independently and then stack the obtained blocks as in the original ResNet. The input of the next block becomes the weighted average of the residual units from the previous block, eqn. (10). A similar connectivity pattern has been explored in [17]. We add the sum of the ponder costs $\rho_k, k = 1 \ldots K$ from the K blocks to the original loss function $\mathcal{L}$:

$$\mathcal{L}' = \mathcal{L} + \tau \sum_{k=1}^{K} \rho_k. \tag{13}$$

The resulting loss function $\mathcal{L}'$ is differentiable and can be optimized using conventional backpropagation. $\tau \geq 0$ is a regularization coefficient which controls the trade-off between optimizing the original loss function and the ponder cost.

2.3. Spatially Adaptive Computation Time

In this section, we present Spatially Adaptive Computation Time (SACT). We adjust the per-position amount of computation by applying ACT to each spatial position of the block, as shown in fig. 4. As we show in the experiments, SACT can learn to focus the computation on the regions of interest.

We define the *active positions* as the spatial locations where the cumulative halting score is less than one. Because an active position might have *inactive* neighbors, the values for the the inactive positions need to be imputed to evaluate the residual unit in the active positions. We simply copy the previous value for the inactive spatial positions, which is equivalent to setting the residual function $f(\mathbf{x})$ value to zero, as displayed in fig. 5. The evaluation of a block can be stopped completely as soon as all the positions become inactive. Also, the ponder cost is averaged across the spatial positions to make it comparable with the ACT ponder cost. The full algorithm is described in alg. 2.

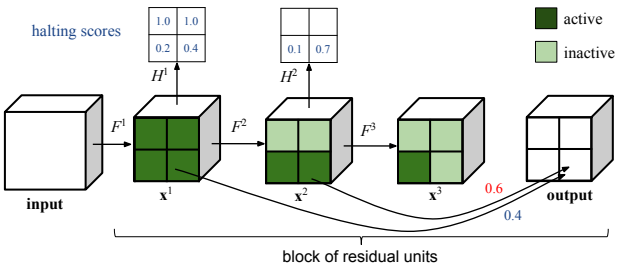


Figure 4: Spatially Adaptive Computation Time (SACT) for one block of residual units. We apply ACT to each spatial position of the block. As soon as position's cumulative halting score reaches 1, we mark it as inactive. See alg. 2. SACT learns to choose the appropriate amount of computation for each spatial position in the block.

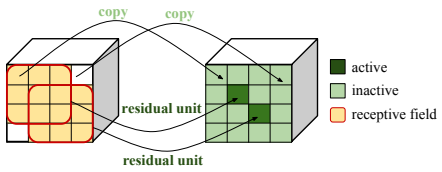


Figure 5: Residual unit with active and inactive positions in SACT. This transformation can be implemented efficiently using the perforated convolutional layer [10].

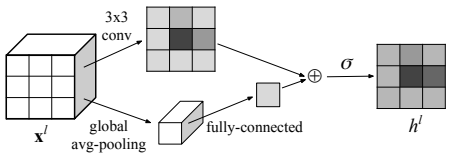


Figure 6: SACT halting scores. Halting scores are evaluated fully convolutionally making SACT applicable to images of arbitrary resolution. SACT becomes ACT if the 3×3 conv weights are set to zero.

We define the halting scores for SACT as

$$H^l(\mathbf{x}) = \sigma(\widetilde{W}^l * \mathbf{x} + W^l \operatorname{pool}(\mathbf{x}) + b^l), \tag{14}$$

where $*$ denotes a 3×3 convolution with a single output channel and pool is a global average-pooling (see fig. 6). SACT is fully convolutional and can be applied to images of any size.

Note that SACT is a more general model than ACT, and, consequently, than standard ResNet. If we choose $\widetilde{W}^l = 0$, then the halting scores for all spatial positions coincide. In this case the computation for all the positions halts simultaneously and we recover the ACT model.

SACT requires evaluation of the residual function $f(\mathbf{x})$ in just the active spatial positions. This can be performed

Algorithm 2 Spatially Adaptive Computation Time for one block of residual units

Input: 3D tensor **input**
Input: number of residual units in the block L
Input: $0 < \varepsilon < 1$
▷ input and output have different shapes
Output: 3D tensor **output** of shape $H \times W \times C$
Output: ponder cost ρ

$\hat{\mathbf{x}} =$ **input**
$\mathcal{X} = \{1 \dots H\} \times \{1 \dots W\}$
for all $(i, j) \in \mathcal{X}$ **do**
  $a_{ij} =$ **true** ▷ Active flag
  $c_{ij} = 0$ ▷ Cumulative halting score
  $R_{ij} = 1$ ▷ Remainder value
  $\mathbf{output}_{ij} = 0$ ▷ Output of the block
  $\rho_{ij} = 0$ ▷ Per-position ponder cost
end for
for $l = 1 \dots L$ **do**
  if not $a_{ij}\ \forall (i, j) \in \mathcal{X}$ **then break**
  end if
  for all $(i, j) \in \mathcal{X}$ **do**
    if a_{ij} **then** $\mathbf{x}_{ij} = F^l(\hat{\mathbf{x}})_{ij}$
    else $\mathbf{x}_{ij} = \hat{\mathbf{x}}_{ij}$
    end if
  end for
  for all $(i, j) \in \mathcal{X}$ **do**
    if not a_{ij} **then continue**
    end if
    if $l < L$ **then** $h_{ij} = H^l(\mathbf{x})_{ij}$
    else $h_{ij} = 1$
    end if
    c_{ij} += h_{ij}
    ρ_{ij} += 1
    if $c_{ij} < 1 - \varepsilon$ **then**
      $\mathbf{output}_{ij}$ += $h_{ij} \cdot \mathbf{x}_{ij}$
      R_{ij} −= h_{ij}
    else
      $\mathbf{output}_{ij}$ += $R_{ij} \cdot \mathbf{x}_{ij}$
      ρ_{ij} += R_{ij}
      $a_{ij} =$ **false**
    end if
  end for
  $\hat{\mathbf{x}} = \mathbf{x}$
end for
$\rho = \sum_{(i,j)\in\mathcal{X}} \rho_{ij} / (HW)$
return output, ρ

efficiently using the *perforated convolutional layer* proposed in [10] (with skipped values replaced by zeros instead of the nearest neighbor's values). Recall that the residual function consists of a stack of 1×1, 3×3 and 1×1 convolutional layers. The first convolutional layer has to be evaluated in the positions obtained by *dilating* the active positions set with a 3×3 kernel. The second and third layers need to be evaluated just in the active positions.

An alternative approach to using the perforated convolutional layer is to *tile* the halting scores map. Suppose that we share the values of the halting scores h^l within $k \times k$ tiles. For example, we can perform pooling of h^l with a kernel size $k \times k$ and stride k and then upscale the results by a factor of k. Then, all positions in a tile have the same active flag, and we can apply the residual unit densely to just the active tiles, reusing the commonly available convolution routines. k should be sufficiently high to mitigate the overhead of the additional kernel calls and the overlapping computations of the first 1×1 convolution. Therefore, tiling is advisable when the SACT is applied to high-resolution images.

3. Related work

The majority of the work on increasing the computational efficiency of deep convolutional networks focuses on *static* techniques. These include decompositions of convolutional kernels [21] and pruning of connections [14]. Many of these techniques made their way into the design of the standard deep architectures. For example, Inception [39] and ResNet [15, 16] use factorized convolutional kernels.

Recently, several works have considered the problem of varying the amount of computation in computer vision. Cascaded classifiers [27, 43] are used in object detection to quickly reject "easy" negative proposals. Dynamic Capacity Networks [1] use the same amount of computation for all images and use image classification-specific heuristic. PerforatedCNNs [10] vary the amount of computation spatially but not between images. [3] proposes to tune the amount of computation in a fully-connected network using a REINFORCE-trained policy which makes the optimization problem significantly more challenging.

BranchyNet [40] is the most similar approach to ours although only applicable to classification problems. It adds classification branches to the intermediate layers of the network. As soon as the entropy of the intermediate classifications is below some threshold, the network's evaluation halts. Our preliminary experiments with a similar procedure based on ACT (using ACT to choose the number of blocks to evaluate) show that it is inferior to using less units per block.

4. Experiments

We first apply ACT and SACT models to the image classification task for the ImageNet dataset [8]. We show that SACT achieves a better FLOPs-accuracy trade-off than ACT by directing computation to the regions of interest. Additionally, SACT improves the accuracy on high-resolution images compared to the ResNet model. Next, we use the obtained SACT model as a feature extractor in the Faster R-CNN object detection pipeline [34] on the COCO dataset [31]. Again we show that we obtain significantly improved FLOPs-mAP trade-off compared to basic ResNet models. Finally, we demonstrate that SACT ponder cost maps correlate well with

the position of human eye fixations by evaluating them as a visual saliency model on the cat2000 dataset [4] without any training on this dataset.

4.1. Image classification (ImageNet dataset)

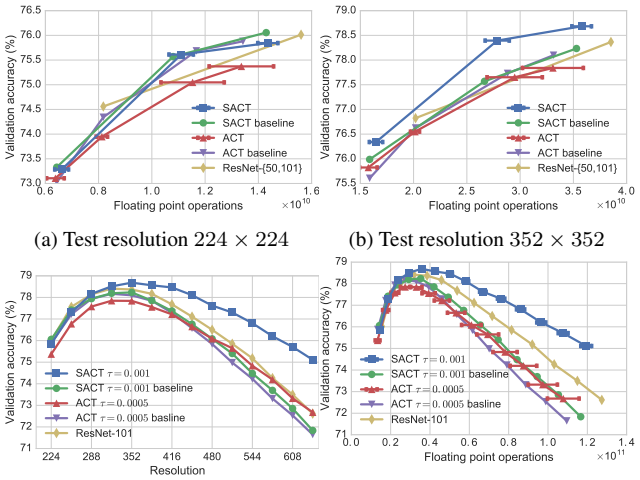
First, we train the basic ResNet-50 and ResNet-101 models from scratch using asynchronous SGD with momentum (see the supplementary text for the hyperparameters). Our models achieve similar performance to the reference implementation[1]. For a single center 224×224 resolution crop, the reference ResNet-101 model achieves 76.4% accuracy, 92.9% recall@5, while our implementation achieves 76% and 93.1%, respectively. Note that our model is the newer pre-activation ResNet [16] and the reference implementation is the post-activation ResNet [15].

We use ResNet-101 as the basic architecture for ACT and SACT models. Thanks to the end-to-end differentiability and deterministic behaviour, we find the same optimization hyperparameters are applicable for training of ACT and SACT as for the ResNet models. However, special care needs to be taken to address the *dead residual unit* problem in ACT and SACT models. Since ACT and SACT are deterministic, the last units in the blocks do not get enough training signal and their parameters become obsolete. As a result, the ponder cost saved by not using these units overwhelms the possible initial gains in the original loss function and the units are never used. We observe that while the dead residual units can be recovered during training, this process is very slow. Note that ACT-RNN [12] is not affected by this problem since the parameters for all timesteps are shared.

We find two techniques helpful for alleviating the dead residual unit problem. First, we initialize the bias of the halting scores units to a negative value to force the model to use the last units during the initial stages of learning. We use $b^l = -3$ in the experiments which corresponds to initially using $1/\sigma(b^l) \approx 21$ units. Second, we use a *two-stage training* procedure by initializing the ACT/SACT network's weights from the pretrained ResNet-101 model. The halting score weights are still initialized randomly. This greatly simplifies learning of a reasonable halting policy in the beginning of training.

As a baseline for ACT and SACT, we consider a non-adaptive ResNet model with a similar number of floating point operations. We take the average numbers of units used in each block in the ACT or SACT model (for SACT we also average over the spatial dimensions) and round them to the nearest integers. Then, we train a ResNet model with such number of units per block. We follow the two-stage training procedure by initializing the network's parameters with the the first residual units of the full ResNet-101 in each block. This slightly improves the performance compared to using the random initialization.

[1] https://github.com/KaimingHe/deep-residual-networks



(a) Test resolution 224×224 (b) Test resolution 352×352

(c) Resolution *vs*. accuracy (d) FLOPs *vs*. accuracy for varying resolution

Figure 7: ImageNet validation set. Comparison of ResNet, ACT, SACT and the respective baselines. Error bars denote one standard deviation across images. All models are trained with 224×224 resolution images. SACT outperforms ACT and baselines when applied to images whose resolutions are higher than the training images. The advantage margin grows as resolution difference increases.

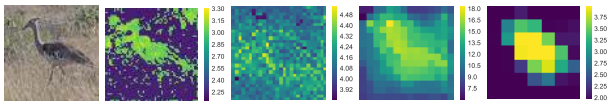


Figure 8: Ponder cost maps for each block (SACT $\tau = 0.005$, ImageNet validation image). Note that the first block reacts to the low-level features while the last two blocks attempt to localize the object.

We compare ACT and SACT to ResNet-50, ResNet-101 and the baselines in fig. 7. We measure the average per-image number of floating point operations (FLOPs) required for evaluation of the validation set. We treat multiply-add as two floating point operations. The FLOPs are calculated just for the convolution operations (perforated convolution for SACT) since all other operations (nonlinearities, pooling and output averaging in ACT/SACT) have minimal impact on this metric. The ACT models use $\tau \in \{0.0005, 0.001, 0.005, 0.01\}$ and SACT models use $\tau \in \{0.001, 0.005, 0.01\}$. If we increase the image resolution at the test time, as suggested in [16], we observe that SACT outperforms ACT and the baselines. Surprisingly, in this setting SACT has higher accuracy than the ResNet-101 model while being computationally cheaper. Such accuracy improvement does not happen for the baseline models or ACT models. We attribute this to the improved scale tolerance provided by the SACT mechanism. The extended results of fig. 7(a,b), including the average number of residual units per block, are presented in the supplementary.

We visualize the ponder cost for each block of SACT as

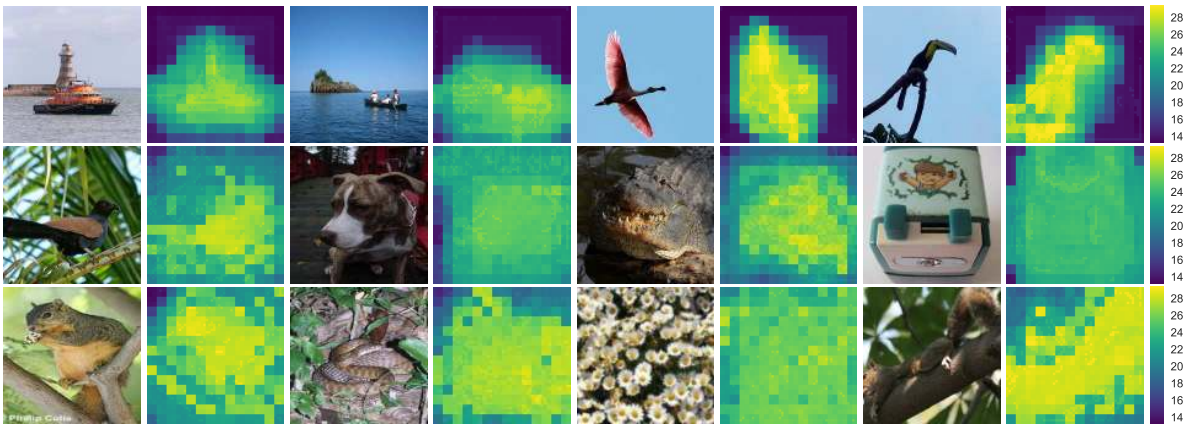


Figure 9: ImageNet validation set. SACT ($\tau = 0.005$) ponder cost maps. Top: low ponder cost (19.8-20.55), middle: average ponder cost (23.4-23.6), bottom: high ponder cost (24.9-26.0). SACT typically focuses the computation on the region of interest.

heat maps (which we call *ponder cost maps* henceforth) in fig. 8. More examples of the total SACT ponder cost maps are shown in fig. 9.

4.2. Object detection (COCO dataset)

Motivated by the success of SACT in classification of high-resolution images and ignoring uninformative background, we now turn to a harder problem of object detection. Object detection is typically performed for high-resolution images (such as 1000×600, compared to 224×224 for ImageNet classification) to allow detection of small objects. Computational redundancy becomes a big issue in this setting since a large image area is often occupied by the background.

We use the Faster R-CNN object detection pipeline [34] which consists of three stages. First, the image is processed with a feature extractor. This is the most computationally expensive part. Second, a Region Proposal Network predicts a number of class-agnostic rectangular proposals (typically 300). Third, each proposal box's features are cropped from the feature map and passed through a box classifier which predicts whether the proposal corresponds to an object, the class of this object and refines the boundaries. We train the model end-to-end using asynchronous SGD with momentum, employing Tensorflow's `crop_and_resize` operation, which is similar to the Spatial Transformer Network [20], to perform cropping of the region proposals. The training hyperparameters are provided in the supplementary.

We use ResNet blocks 1-3 as a feature extractor and block 4 as a box classifier, as suggested in [15]. We reuse the models pretrained on the ImageNet classification task and fine-tune them for COCO detection. For SACT, the ponder cost penalty τ is only applied to the feature extractor (we use the same value as for ImageNet classification). We use COCO train for training and COCO val for evaluation (instead of the combined train+val set which is sometimes used in the literature). We do not employ multiscale inference, iterative box refinement or global context.

Feature extractor	FLOPs (%)	mAP @ [.5, .95] (%)
ResNet-101 [15]	100	27.2
ResNet-50 (our impl.)	46.6	25.56
SACT $\tau = 0.005$	$\mathbf{56.0} \pm 8.5$	**27.61**
SACT $\tau = 0.001$	72.4 ± 8.4	29.04
ResNet-101 (our impl.)	100	29.24

Table 1: COCO val set. Faster R-CNN with SACT results. FLOPs are average ($\pm$ one standard deviation) feature extractor floating point operations relative to ResNet-101 (that does 1.42E+11 operations). SACT improves the FLOPs-mAP trade-off compared to using ResNet without adaptive computation.

We find that SACT achieves superior speed-mAP trade-off compared to the baseline of using non-adaptive ResNet as a feature extractor (see table 1). SACT $\tau = 0.005$ model has slightly higher FLOPs count than ResNet-50 and 2.1 points better mAP. Note that this SACT model outperforms the originally reported result for ResNet-101, 27.2 mAP [15]. Several examples are presented in fig. 10.

4.3. Visual saliency (cat2000 dataset)

We now show that SACT ponder cost maps correlate well with human attention. To do that, we use a large dataset of visual saliency: the cat2000 dataset [4]. The dataset is obtained by showing 4,000 images of 20 scene categories to 24 human subjects and recording their eye fixation positions. The ground-truth saliency map is a heat map of the eye fixation positions. We do not train the SACT models on this dataset and simply reuse the ImageNet- and COCO-trained models. Cat2000 saliency maps exhibit a strong center bias. Most images contain a blob of saliency in the center even when there is no object of interest located there. Since our

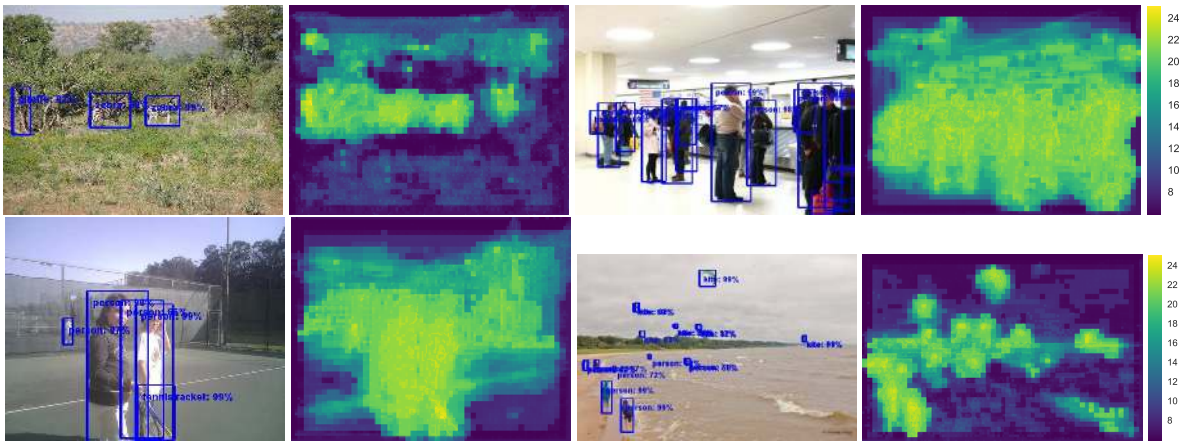


Figure 10: COCO testdev set. Detections and feature extractor ponder cost maps ($\tau = 0.005$). SACT allocates much more computation to the object-like regions of the image.

Model	AUC-Judd (%)
Center baseline [4]	83.4
DeepFix [24]	87†
"Infinite humans" [4]	90†
ImageNet SACT $\tau = 0.005$	84.6
COCO SACT $\tau = 0.005$	84.7

Table 2: cat2000 validation set. † - results for the test set. SACT ponder cost maps work as a visual saliency model even without explicit supervision.

model is fully convolutional, we cannot learn such bias even if we trained on the saliency data. Therefore, we combine our ponder cost maps with a constant center-biased map.

We resize the 1920×1080 cat2000 images to 320×180 for ImageNet model and to 640×360 for COCO model and pass them through the SACT model. Following [4], we consider a linear combination of the Gaussian blurred ponder cost map normalized to $[0, 1]$ range and a "center baseline," a Gaussian centered at the middle of the image. Full description of the combination scheme is provided in the supplementary. The first half of the training set images for every scene category is used for determining the optimal values of the Gaussian blur kernel size and the center baseline multiplier, while the second half is used for validation.

Table 2 presents the AUC-Judd [5] metric, the area under the ROC-curve for the saliency map as a predictor for eye fixation positions. SACT outperforms the naïve center baseline. Compared to the state-of-the-art deep model DeepFix [24] method, SACT does competitively. Examples are shown in fig. 11.

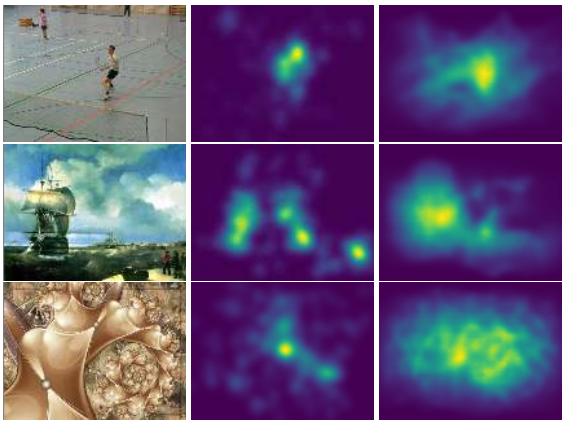


Figure 11: cat2000 saliency dataset. Left to right: image, human saliency, SACT ponder cost map (COCO model, $\tau = 0.005$) with postprocessing (see text) and softmax with temperature $1/5$. Note the center bias of the dataset. SACT model performs surprisingly well on out-of-domain images such as art and fractals.

5. Conclusion

We present a Residual Network based model with a spatially varying computation time. This model is end-to-end trainable, deterministic and can be viewed as a black-box feature extractor. We show its effectiveness in image classification and object detection problems. The amount of per-position computation in this model correlates well with the human eye fixation positions, suggesting that this model captures the important parts of the image. We hope that this paper will lead to a wider adoption of attention and adaptive computation time in large-scale computer vision systems. The source code is available at https://github.com/mfigurnov/sact.

Acknowledgments. D. Vetrov is supported by Russian Academic Excellence Project '5-100'. R. Salakhutdinov is supported in part by ONR grants N00014-13-1-0721, N00014-14-1-0232, and the ADeLAIDE grant FA8750-16C-0130-001.

References

[1] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic capacity networks. *ICML*, 2016. 1, 5

[2] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *ICLR*, 2015. 1

[3] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. *ICLR Workshop*, 2016. 5

[4] Z. Bylinskii, T. Judd, A. Borji, L. Itti, F. Durand, A. Oliva, and A. Torralba. Mit saliency benchmark. http://saliency.mit.edu/. 6, 7, 8

[5] Z. Bylinskii, T. Judd, A. Oliva, A. Torralba, and F. Durand. What do different evaluation metrics tell us about saliency models? *arXiv preprint arXiv:1604.03605*, 2016. 8

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 2

[7] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. *NIPS*, 2016. 2

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009. 2, 5

[9] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CVPR*, 2015. 1

[10] M. Figurnov, A. Ibraimova, D. Vetrov, and P. Kohli. PerforatedCNNs: Acceleration through elimination of redundant convolutions. *NIPS*, 2016. 4, 5

[11] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847*, 2016. 1

[12] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. 1, 3, 6

[13] K. Greff, R. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *ICLR 2017*. 1, 3

[14] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *ICLR*, 2016. 5

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. 1, 2, 5, 6, 7

[16] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ECCV*, 2016. 1, 2, 5, 6

[17] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 4

[18] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *ECCV*, 2016. 3

[19] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 2

[20] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *NIPS*, 2015. 1, 7

[21] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC*, 2014. 5

[22] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *CVPR*, 2015. 1

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012. 1

[24] S. S. Kruthiventi, K. Ayush, and R. V. Babu. Deepfix: A fully convolutional neural network for predicting human eye fixations. *arXiv preprint arXiv:1510.02927*, 2015. 8

[25] H. Larochelle. My notes on adaptive computation time for recurrent neural networks. *Blog post https://goo.gl/QxBucH*, 2016. 3

[26] H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. *NIPS*, 2010. 1

[27] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. *CVPR*, 2015. 5

[28] Z. Li, E. Gavves, M. Jain, and C. G. Snoek. Videolstm convolves, attends and flows for action recognition. *arXiv preprint arXiv:1607.01794*, 2016. 1

[29] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016. 1

[30] M. Lin, Q. Chen, and S. Yan. Network in network. *ICLR*, 2014. 2

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. *ECCV*, 2014. 2, 5

[32] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015. 1

[33] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. *NIPS*, 2014. 1

[34] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *NIPS*, 2015. 1, 2, 5, 7

[35] R. A. Rensink. The dynamic representation of scenes. *Visual cognition*, 7(1-3), 2000. 1

[36] S. Sharma, R. Kiros, and R. Salakhutdinov. Action recognition using visual attention. *ICLR Workshop*, 2016. 1

[37] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 2016. 1

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 1, 2

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CVPR*, 2015. 1, 5

[40] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *ICPR*, 2016. 5

[41] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015. 1

[42] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *ICML*, 2015. 1, 4

[43] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. *CVPR*, 2016. 5

[44] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *NIPS*, 2015. 1