Pixel-wise Attentional Gating for Scene Parsing

Shu Kong, Charless Fowlkes
Department of Computer Science, University of California, Irvine, CA 92697, USA
{skong2, fowlkes}@ics.uci.edu

Abstract

To achieve dynamic inference in pixel labeling tasks, we propose Pixel-wise Attentional Gating (PAG), which learns to selectively process a subset of spatial locations at each layer of a deep convolutional network. PAG is a generic, architecture-independent, problem-agnostic mechanism that can be readily “plugged in” to an existing model with fine-tuning. We utilize PAG in two ways: 1) learning spatially varying pooling fields that improve model performance without the extra computation cost associated with multi-scale pooling, and 2) learning a dynamic computation policy for each pixel to decrease total computation (FLOPs) while maintaining accuracy.

We extensively evaluate PAG on a variety of per-pixel labeling tasks, including semantic segmentation, boundary detection, monocular depth and surface normal estimation. We demonstrate that PAG allows competitive or state-of-the-art performance on these tasks. Our experiments show that PAG learns dynamic spatial allocation of computation over the input image which provides better performance trade-offs compared to related approaches (e.g., truncating deep models or dynamically skipping whole layers). Generally, we observe PAG can reduce computation by 10% without noticeable loss in accuracy and performance degrades gracefully when imposing stronger computational constraints.

1. Introduction

The development of deep convolutional neural networks (CNN) has allowed remarkable progress in wide range of image pixel-labeling tasks such as boundary detection [41, 56, 29], semantic segmentation [31, 32, 6], monocular depth estimation [32, 36, 35, 38, 13], and surface normal estimation [53, 4, 12]. Architectures that enable training of increasingly deeper networks have resulted in corresponding improvements in prediction accuracy [49, 22]. However, with great depth comes great computational burden. This hinders deployment of such deep models in edge and mobile computing applications which have significant power/memory constraints.

To make deep models more practically applicable, a flurry of recent work has focused on reducing these storage and computational costs [21, 42, 26, 40, 5, 30]. Static offline techniques like network distillation [23], pruning [42], and model compression [5] take a trained network as input and synthesize a new network that approximates the same functionality with reduced memory footprint and test-time execution cost. Our approach is inspired by a complementary family of techniques that learn to vary the network computation depth adaptively, depending on the input data [51, 55, 54, 15].

In this paper, we study the problem of achieving dynamic inference for per-pixel labeling tasks with a deep CNN model under limited computational budget. For image classification, dynamic allocation of computational “attention” can be interpreted as expending more computation on ambiguous images (e.g., [51, 55, 54]) or limiting processing to informative image regions [15]. However, understanding the role of dynamic computation in pixel labeling tasks has not been explored. Pixel-level labeling requires analyzing fine-grained image details and making predictions at every spatial location, so it is not obvious that dynamically allocating computation to different image regions is useful. Unlike classification, labeling locally uninformative regions would seem to demand more computation rather than less (e.g., to incorporate long-range context).

To explore these questions, we introduce a Pixel-wise Attentional Gating (PAG) unit that selects a sparse subset of spatial locations to process based on the input feature map. We utilize the Gumbel sampling trick [19, 28, 39] to allow differentiable, end-to-end latent training of PAG units inserted across multiple computational blocks of a given task-specific architecture. We exploit this generic PAG unit in two ways: bypassing sequential (residual) processing layers and dynamically selecting between multiple parallel network branches.

Dynamic computation depth: Inserting PAG at multiple layers of a Residual Network enables learning a dynamic, feedforward computation path for each pixel that is conditional on the input image (see the second last row in Fig. 1). We in-
Figure 1: Pixel-wise Attentional Gating units (PAG) achieve dynamic inference by learning a dynamic computation path for each pixel under limited computation budget. The “ponder maps” shown in the last row provide a visualization of the amount of computation allocated to each location (generated by accumulating binary masks from PAG units across all layers); whereas the “MultiPool” adaptively chooses the proper pooling size for each pixel to aggregate information for inference. We apply PAG to a variety of per-pixel labeling tasks (boundary detection, semantic segmentation, monocular geometry) and evaluate over diverse image datasets (indoor/outdoor scenes, narrow/wide field-of-view).

We introduce a sparsity hyperparameter that provides control over the average total and per-layer computation. For a fixed computational budget, we show this dynamic, per-pixel gating outperforms architectures that meet the budget by either using a smaller number of fixed layers or learning to dynamically bypass whole layers (Section 3.3).

Dynamic spatial pooling: We exploit PAG to dynamically select the extent of pooling regions at each spatial image location (see the last row in Fig. 1). Previous work has demonstrated the benefits of averaging features from multiple pooling scales using either learned weights [6], or spatially varying weights based on attention [7] or scene depth [32]. However, such multi-scale pooling requires substantially more computation. We show the proposed PAG unit can learn to select appropriate spatially-varying pooling, outperforming the recent work of [32] without the computational burden of multiple parallel branches (Section 3.4).

We carry out an extensive evaluation of pixel-wise attentional gating over diverse datasets for a variety of per-pixel labeling tasks including boundary detection, semantic segmentation, monocular depth estimation and surface normal estimation (see Fig. 1). We demonstrate that PAG helps deliver state-of-the-art performance on these tasks by dynamically allocating computation. In general, we observe that the introduction of PAG units can reduce total computation by 10% without noticeable drop in accuracy and shows graceful degradation in performance even with substantial budget constraints (e.g., a 30% budget cut).

To summarize our primary contribution: (1) we introduce a pixel-wise attentional gating unit which is problem-agnostic, architecture-independent and provides a simple method to allow user-specified control computational parsimony with standard training techniques; (2) we investigate the role of dynamic computation in pixel-labeling tasks and demonstrate improved prediction performance while maintaining or reducing overall compute cost.

2. Related Work

Deep CNN models with residual or “skip” connections have yielded substantial performance improvements with increased depth [22, 24], but also introduced redundant parameters and computation [21, 42]. In interpreting the success of residual networks (ResNet) [22], it has been suggested that ResNet can be seen as an ensemble of many small networks [52], each defined by a path through the network topology. This is supported by the observation that ResNet still performs well even when some layers are removed after training [25, 15]. This indicates it may be possible to reduce test-time computation by dynamically choosing only a subset of these paths to evaluate [51, 55, 54, 15].

This can be achieved by learning a halting policy that stops computation after evaluation of a particular layer [15], or a more flexible routing policy trained through reinforcement learning [55, 54]. Our method is most closely related to [51], which utilizes the “Gumbel sampling trick” [19, 28, 39] to learn binary gating that determines whether each layer is computed. The Gumbel sampling technique allows one to perform gradient descent on models that include a discrete argmax operation without resorting to approximation by softmax or reinforcement learning techniques.

The PerforatedCNN [16] demonstrated that convolution operations could be accelerated by learning static masks that skip computation at a subset of spatial positions. This was used in [15] to achieve spatially varying dynamic depth.
Our approach is simpler (it uses a simple sparsity regularization to directly control amount per-pixel or per-layer computation rather than ponder cost) and more flexible (allowing more flexible routing policies than early halting\(^1\)).

Finally, our use of dynamic computation to choose between branches is related to [32], which improves semantic segmentation by fusing features from multiple branches with various pooling sizes using a spatially varying weighted average. Unlike [6, 7, 32] which require computing the outputs of parallel pooling branches, our PAG-based learners to select a pooling size for each spatial location and only computes the necessary pooled features. This is similar in spirit to the work of [46], which demonstrated that sparsely-gated mixture-of-experts can dramatically increase model capacity using multi-branch configuration with only minor losses in computational efficiency.

3. Pixel-wise Attentional Gating

We first describe our design of Pixel-wise Attentional Gating (PAG) unit and its relation to the ResNet architecture [22]. Then, we elaborate how we exploit the Gumbel-sampling technique to learning PAG differentiable even when generating binary masks. Finally, we describe how the PAG unit can be used to perform dynamic inference by (1) selecting the subset of layers in the computational path for each spatial location, and (2) selecting the correct pooling size at each spatial location.

3.1. Plug-in PAG inside a Residual Block

Consider a block that computes output \(O\) using a residual update \(Z = \mathcal{F}(I)\) to some input \(I\). To reduce computation, one can learn a gating function \(\mathcal{G}(I)\) that selects a subset of spatial locations (pixels) to process conditional on the input. We represent the output of \(\mathcal{G}\) as a binary spatial mask \(\mathcal{G}\) which is replicated along feature channel dimension as needed to match dimension of \(O\) and \(I\). The spatially gated residual update can be written as:

\[
\begin{align*}
\mathcal{G} &= \mathcal{G}(I) \\
O &= \mathcal{G} \odot I + \mathcal{G} \odot (\mathcal{F}_\mathcal{G}(I) + I) \\
&= I + \mathcal{G} \odot \mathcal{F}_\mathcal{G}(I) 
\end{align*}
\]

where \(\odot\) is element-wise product, \(\mathcal{G} = 1 - \mathcal{G}\), and the notation \(\mathcal{F}_\mathcal{G}\) indicates that we only evaluate \(\mathcal{F}\) at the locations specified by \(\mathcal{G}\). An alternative to spatially varying computation is for the gating function to predict a single binary value that determines whether or not the residual is calculated at this layer [51] in which case \(\mathcal{F}_\mathcal{G}\) is only computed if \(\mathcal{G} = 1\).

Both pixel-wise and layer-wise gating have the intrinsic limitation that the gating function \(\mathcal{G}\) must be evaluated prior to \(\mathcal{F}\). To overcome this limitation we integrate the gating function more carefully within the ResNet block. We demonstrate ours in the equations below comparing a standard residual block (left) and the one with PAG (right), respectively with corresponding illustrations in Fig. 2:

\[
\begin{align*}
X &= \mathcal{F}^1(I) \\
Y &= \mathcal{F}^2(X) \\
Z &= \mathcal{F}^3(Y) \\
O &= I + Z
\end{align*}

\[
\begin{align*}
X &= \mathcal{F}^1(I), \quad G = \mathcal{G}(I) \\
Y &= \mathcal{F}^2(X), \quad Y = \mathcal{F}^2_G(X) \\
Z &= \mathcal{F}^3(Y), \quad Z = \mathcal{F}^3_G(G \odot X + G \odot Y) \\
O &= I + Z
\end{align*}
\]

The transformation functions \(\mathcal{F}\)'s consist of convolution, batch normalization [27] and ReLU [43] layers. As seen from the right set of equations, our design advocates computing the gating mask on the input \(I\) to the current building block in parallel with \(X = \mathcal{F}_X(I)\). ResNet adopts bottleneck structure so the first transformation \(\mathcal{F}^1\) performs dimensionality reduction with a set of \(1 \times 1\) kernels, \(\mathcal{F}^2\) utilizes \(3 \times 3\) kernels, and \(\mathcal{F}^3\) is another transform with \(1 \times 1\) kernels that restores dimensionality. As a result, the most costly computation is in the second transformation \(\mathcal{F}^2\) which is mitigated by gating the computation. We show in our ablation study (Section 5.2) that for per-pixel labeling tasks, this design outperforms layer-wise gating.

3.2. Learning Discrete Attention Maps

The key to the proposed PAG is the gating function \(\mathcal{G}\) that produces a discrete (binary) mask which allows for reduced computation. However, producing the binary mask using hard thresholding is non-differentiable, and thus cannot be simply incorporated in CNN where gradient descent is used for training. To bridge the gap, we exploit the Gumbel-Max trick [19] and its recent continuous relaxation [39, 28].

A random variable \(m\) follows a Gumbel distribution if \(m = -\log(-\log(u))\), where \(u\) is a sample from the uniform distribution \(u \sim U[0, 1]\). Let \(g\) be a discrete random variable with probabilities \(P(g = k) \propto \alpha_k\), and let \(\{m_k\}_{k=1,...,K}\) be a sequence of i.i.d. Gumbel random variables. Then we can sample from the discrete variable with:

\[
g = \text{argmax}_{k=1,...,K} \left( \log \alpha_k + m_k \right)
\]

The drawback of this approach is that the argmax operation is not continuous when mapping the Gumbel samples

\[\text{Figure 2: (a) A standard residual block. (b) Pixel-wise Attentional Gating unit (PAG) integrated into a residual block. Boxes/arrows denote activations/computations. G is a sparse, binary map that modulates what processing applied to each spatial location. “⊙” means the perforated convolution [16], which assembles only active pixels for computation.}\]
to the realizations of discrete distribution. To address this issue, a continuous relaxation the Gumbel Sampling Trick, proposed in [39, 28], replaces the argmax operation with a softmax. Using a one-hot vector $g = [g_1, \ldots, g_K]$ to encode $g$, a sample from the Gumbel softmax relaxation can be expressed by the vector:

$$g = \text{softmax}((\log(\alpha) + m)/\tau)$$

where $\alpha = [\alpha_1, \ldots, \alpha_K]$, $m = [m_1, \ldots, m_K]$, and $\tau$ is the “temperature” parameter. In the limit as $\tau \to 0$, the softmax function approaches the argmax function and Eq. (4) becomes equivalent to the discrete sampler. Since the softmax function is differentiable and $m$ contains i.i.d Gumbel random variables which are independent to input activation $\alpha$, we can easily propagate gradients to the probability vector $\alpha$, which is treated as the gating mask for a single pixel in the per-pixel labeling tasks.

As suggested in [51], we employ the straight-through version [39] of Eq. (4) during training. In particular, for the forward pass, we use discrete samples from Eq. (3), but during the backwards pass, we compute the gradient of the softmax relaxation in Eq. (4). Based on our empirical observation as well as that reported in [39], such greedy straight-through estimator performs slightly better than strictly following Eq. (4), even though there is a mismatch between forward and backward pass. In our work, we initialize $\tau = 1$ and decrease it to 0.1 gradually during training. We find this works even better than training with a constant small $\tau$.

### 3.3. Dynamic Per-Pixel Computation Routing

By stacking multiple PAG residual blocks, we can construct a model in which the subset of layers used to compute an output varies for each spatial location based on the collection of binary masks. We allow the user to specify the computational budget in terms of a target sparsity $\rho$. For a binary mask $G \in \{0, 1\}^{H \times W}$, we compute the empirical sparsity $g = \frac{1}{H \times W} \sum_{h,w} G_{h,w}$ (smaller values indicate sparser computation) and measure how well it matches the target $\rho$ using the KL divergence:

$$KL(\rho || g) \equiv \rho \log\left(\frac{\rho}{g}\right) + (1 - \rho) \log\left(\frac{1 - \rho}{1 - g}\right)$$

To train the model, we jointly minimize the sum of a task-specific loss $\ell_{task}$ and the per-layer sparsity loss summed over all layers of interest:

$$\ell = \ell_{task} + \lambda \sum_{l=1}^{L} KL(\rho || g_l)$$

where $l$ indexes one of $L$ layers which have PAG inserted for dynamic computation and $\lambda$ controls the weight for the constraints. In our experiments we set $\lambda = 10^{-4}$ but found performance is stable over a wide range of penalties ($\lambda \in [10^{-5}, 10^{-2}]$). To visualize the spatial distribution of computation, we accumulate the binary gating masks from all to produce a “ponder map”. This reveals that trained models do not allocate computation uniformly, but instead respond to image content (e.g., focusing computation on boundaries between objects where semantic labels, depths or surface normals undergo sharp changes).

An alternative to per-layer sparsity is to compute the total sparsity $g = \frac{1}{L} \sum_{l=1}^{L} g_l$ and penalize $g$ with $KL(\rho || g)$. However, training in this way does not effectively learn dynamic computational paths and results in trivial, non-dynamic solutions, e.g., completely skipping a subset of layers and always using the remaining ones. Similar phenomenon is reported in [51]. In training models we typically start from a pre-trained model and insert sparsity constraints progressively over residual blocks. We found this incremental construction produces better diversity in the PAG computation paths. We also observe that when targeting reduced computation budget, fine-tuning a model which has already been trained with larger $\rho$ consistently brings better performance than fine-tuning a pre-trained model directly with a small $\rho$.

### 3.4. Dynamic Spatial Pooling

In pixel-labeling tasks, the ideal spatial support for analyzing a pixel can vary over the visual field in order to simultaneously maintain fine-grained details and capture context. This suggests an adaptive pooling mechanism at pixel level, or multi-scale pooling module (MultiPool) that chooses the appropriate pooling size for each pixel (see e.g., [32]). Given a collection of $P$ pooled feature maps $\{M_i\}_{i=1,\ldots,P}$ computed with different pooling sizes, we can generate a MultiPool feature map $O = \sum_i W_i \circ M_i$, where $\{W_i\}_{i=1,\ldots,P}$ are spatial selection masks, and $\circ$ indicates element-wise product between $W_i$ and each channel of $M_i$. We utilize the PAG to select the “correct” pooling region at each spatial location by applying Eq. (4). This MultiPool module, illustrated in Fig. 3, can be inserted in place of regular pooling with little computational overhead and learned in a latent manner using the task-specific loss (no additional sparsity loss).
layers, replacing them with atrous convolution with dilation in our approach using the toolbox MatConvNet \cite{57}, and train processing, and no external training data. We implement whistles`, e.g., labeling tasks, we implement our models without “bells and our goal is to explore computational parsimony in per-pixel

\[ \rho \]

crushing

\[ \times \]

downsampling the ground-truth) \(8 \times 8\) pooling layer and the last two \(2 \times 2\) pooling layers, replacing them with atrous convolution with dilation rate 2 and 4, respectively, to maintain a spatial sampling rate. Such a modification thus outputs predictions at \(1/8\) the input resolution. Rather than up-sampling the output (or downsampling the ground-truth) \(8 \times 8\) for benchmarking \cite{6,32}, we find it better to apply a deconvolution layer followed by two or more convolutional layers before the final output.

We augment the training sets with random left-right flips and random crops with 20-pixel margin and of size divisible by 8. When training the model, we fix the batch normalization, using the same constant global moments in both training and testing. This modification does not impact the performance and allows a batch size of one during training (a single input image per batch). We use the “poly” learning rate policy \cite{6} with a base learning rate of 0.0002 scaled as a function of iteration by \(1 - \frac{\text{iter}}{\text{maxiter}} \)^{0.9}. We adopt a stage-wise training strategy over all tasks, i.e. training a base model, adding PAG-based MultiPool, inserting PAG for dynamic computation progressively, and finally decreasing \(\rho\) to achieve target computational budget. Since our goal is to explore computational parsimony in per-pixel labeling tasks, we implement our models without “bells and whistles”, e.g. no utilization of ensembles, no CRF as post-processing, and no external training data. We implement our approach using the toolbox MatConvNet \cite{50}, and train using SGD on a single Titan X GPU\(^2\).

4. Implementation and Training

While our PAG unit is agnostic to network architectures, in all our experiments we utilize ResNet \cite{22} pre-trained on ImageNet \cite{11} as the base model. We following \cite{6,32} and increase the output resolution of ResNet by removing the top global \(7 \times 7\) pooling layer and the last two \(2 \times 2\) pooling layers, replacing them with atrous convolution with dilation rate 2 and 4, respectively, to maintain a spatial sampling rate. Such a modification thus outputs predictions at \(1/8\) the input resolution. Rather than up-sampling the output (or downsampling the ground-truth) \(8 \times 8\) for benchmarking \cite{6,32}, we find it better to apply a deconvolution layer followed by two or more convolutional layers before the final output.

We augment the training sets with random left-right flips and random crops with 20-pixel margin and of size divisible by 8. When training the model, we fix the batch normalization, using the same constant global moments in both training and testing. This modification does not impact the performance and allows a batch size of one during training (a single input image per batch). We use the “poly” learning rate policy \cite{6} with a base learning rate of 0.0002 scaled as a function of iteration by \(1 - \frac{\text{iter}}{\text{maxiter}} \)^{0.9}. We adopt a stage-wise training strategy over all tasks, i.e. training a base model, adding PAG-based MultiPool, inserting PAG for dynamic computation progressively, and finally decreasing \(\rho\) to achieve target computational budget. Since our goal is to explore computational parsimony in per-pixel labeling tasks, we implement our models without “bells and whistles”, e.g. no utilization of ensembles, no CRF as post-processing, and no external training data. We implement our approach using the toolbox MatConvNet \cite{50}, and train using SGD on a single Titan X GPU\(^2\).

4.1. Per-Pixel Labeling Vision Tasks

Boundary Detection We train a base model using (binary) logistic loss. Following \cite{56,41,31}, we include four prediction branches at macro residual blocks (denoted by Res 2, 3, 4, 5) and a fusion branch for training. To handle class imbalance, we utilize a weighted loss accumulated over the prediction losses given by:

\[ \ell_{\text{boundary}} = - \sum_{b \in B} \sum_{j \in Y} \beta_y \log(P(y_j | \theta_b)) \] (7)

where \(b\) indexes the branches, \(\beta_y = |Y_−|/|Y_− \cup Y_+|\), \(\beta_+ = 1 - \beta_−; Y_+ \text{ and } Y_−\) denote the set of boundary and non-boundary annotations, respectively. \(Y = Y_− \cup Y_+\) contains the indices of all pixels. This base model is modeled after HED \cite{56} and performs similarly.

Semantic Segmentation For semantic segmentation, we train a model using \(K\)-way cross-entropy loss as in \cite{6,32}:

\[ \ell_{\text{semantic}} = - \sum_{i} \sum_{c=1}^{K} \mathbb{1}_{[y_i = c]} \cdot \log(C_i) \] (8)

where \(C_i\) is the class prediction (from a softmax transform) at pixel \(i\), and \(y_i\) is the ground-truth class label.

Monocular Depth Estimation For monocular depth estimation, we use combined \(L_2\) and \(L_1\) losses to compare the predicted and ground-truth depth maps \(\mathbf{D}\) and \(\mathbf{D}_\text{gt}\) which are on a log scale:

\[ \ell_{\text{depth}} = \sum_{i=1}^{N} ||\mathbf{D}_i - \mathbf{D}_{\text{gt}}||_2^2 + \gamma ||\mathbf{D}_i - \mathbf{D}_\text{gt}||_1 \] (9)

where \(\gamma = 2\) controls the relative importance of the two losses. This mixed loss penalizes large errors quadratically (the \(L_2\) term) while still assuring a non-vanishing gradient that continues to drive down small errors (the \(L_1\) term). The idea behind our loss is similar to the reverse Huber loss as used in \cite{35}, which can be understood as concatenation of truncated \(L_2\) and \(L_1\) loss. However, the reverse Huber loss requires specifying a hyper-parameter for the boundary between \(L_2\) and \(L_1\); we find our mixed loss is robust and performs well with \(\gamma \in [1,5]\).

Surface Normal Estimation To predict surface normals, we insert a final \(L_2\) normalization layer so that predicted normals have unit Euclidean length. In the literature, cosine distance is often used in the loss function to train the model, while performance metrics for normal estimation measure the angular difference between prediction \(\mathbf{n}\) and the target normal \(\mathbf{\hat{n}}\) \cite{17,13}. We address this discrepancy by incorporating inverse cosine distance along with cosine distance as our objective function:

\[ \ell_{\text{normal}} = \sum_i (\mathbf{n}_i \cdot \mathbf{\hat{n}}_i + \lambda \cos^{-1}(\mathbf{n}_i \cdot \mathbf{\hat{n}}_i)) \] (10)

where \(\lambda\) controls the importance of the two part and we set \(\lambda = 4\) throughout our experiments. Fig. 4 compares the

\(^2\)As MatConvNet itself does not provide perforated convolution, we release the code and models implemented with multiplicative gating at http://XXXXX.
curves of the two losses, and we can clearly see that the inverse cosine loss always produce meaningful gradients, whereas the popular cosine loss has “vanishing gradient” issue when prediction errors become small (analogous to the mixed $L_1/L_2$ loss for depth).

5. Experiments

To evaluate our method based on PAG, we choose datasets that span a variety of per-pixel labeling tasks, including boundary detection, semantic segmentation, depth and surface normal estimation. We first describe the datasets, and then carry out experiments to determine the best architectural configurations to exploit PAG and measure compute-performance trade-offs. We then evaluate our best models on standard benchmarks and show our approach achieves state-of-the-art or competitive performance for pixel-labeling. Finally, we visualize the attentional maps from MultiPool and ponder maps, and demonstrate qualitatively that our models pay more “attention” to specific regions/pixels, especially on boundaries between regions, e.g. semantic segments, and regions with sharp change of depth and normal.

5.1. Datasets

We utilize the following benchmark datasets.

**BSDS500** [1] is the most popular dataset for boundary detection. It provides a standard split [1, 56] of 300 train-val images and 200 test images.

**NYUv2** [48] consists of 1,449 RGB-D indoor scene images of the resolution 640 × 480 which include color and pixel-wise depth obtained by a Kinect sensor. We use the ground-truth segmentation into 40 classes provided in [20] and a standard train/test split into 795 and 654 images, respectively. For surface normal estimation, we compute the normal as target from depth using the method in [48] by fitting least-squares planes to neighboring sets of points in the point cloud.

**Stanford-2D-3D** [3] contains 1,559 RGB panoramic images with depths, surface normal and semantic annotations covering six large-scale indoor areas from three different buildings. We use area 3 and 4 as a validation set (489 panoramas) and the remaining four areas for training (1,070 panoramas). The panoramas are very large (2048×4096) and contain black void regions at top and bottom due to the spherical panoramic topology. We rescale them by 0.5 and crop out the central two-thirds ($y \in [160, 863]$) resulting in final images of size 704×2048-pixels. We randomly crop out sub-images of 704×704 resolution for training. Note that the surface normals in panoramic images are relative to the global coordinate system which cannot be determined from the image alone. Thus we transformed this global normal into local normal specified relative to the camera viewing direction (details in supplementary material). Note that such relative normals are also useful in scene understanding and reconstruction.

**Cityscapes** [10] contains high-quality pixel-level annotations of images collected in street scenes from 50 different cities. We use the standard split of training set (2,975 images) and validation set (500 images) for testing, respectively, labeled for 19 semantic classes as well as depth obtained by disparity. The images are of high resolution (1024 × 2048), and we randomly crop out sub-images of 800×800 resolution during training.

5.2. Analysis of Pixel-wise Attentional Gating

We evaluate different configurations of PAG on the BSDS500 and NYUv2 datasets for boundary detection and semantic segmentation (similar observations hold on other tasks, see supplementary materials). The goal of these experiments is to establish:

1. whether our base model is comparable to state-of-the-art methods;
2. where to insert the PAG-based MultiPool module for the best performance;
3. how our PAG-based method for computational parsimony impacts performance, and how it performs compared with other related methods, e.g. truncated ResNet and methods learning to skip/drop layers.

<table>
<thead>
<tr>
<th>parameter &amp; FLOPs truncated</th>
<th>layer-skipping perforatedCNN</th>
<th>MP@Res5 (PAG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho \times 10^{12}$</td>
<td>IoU acc.</td>
<td>IoU acc.</td>
</tr>
<tr>
<td>0.5</td>
<td>6.29</td>
<td>36.30</td>
</tr>
<tr>
<td>0.7</td>
<td>8.27</td>
<td>37.09</td>
</tr>
<tr>
<td>0.9</td>
<td>8.95</td>
<td>40.29</td>
</tr>
<tr>
<td>1.0</td>
<td>9.63</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 1: Ablation study for where to insert the PAG-based MultiPool module. Experiments are from boundary detection and semantic segmentation on BSDS500 and NYUv2 dataset, measured by F-score ($F_{\text{bnd.}}$) and IoU (IoU$_{\text{seg.}}$), respectively. Numbers are in % (higher is better).

Table 2: Performance comparison w.r.t computational parsimony controlled by hyper-parameter $\rho$ on NYUv2 dataset for semantic segmentation.

Base models: We train our base models as described in...
Section 4 without PAG units. The performance of our base model is on-par with state-of-the-art systems, achieving IoU=42.05% on NYUv2 for semantic segmentation (RefineNet [37] achieves IoU=44.5 with multi-resolution input), and $F = 0.79$ on BSDS500 for boundary detection (HED [56] achieves $F = 0.78$). More comprehensive comparisons with other related methods are shown later in Section 5.3.

MultiPool: Table 1 explores the effect of inserting the MultiPool operation at different layers in the base model. In Table 1, Res6 means that we insert MultiPool module in the additional convolutional layers above the ResNet50 base. For boundary detection, we do not initialize more convolutional layers above the backbone, so there is no Res6. For both tasks, we observe that including a PAG-based MultiPool module improves performance, but including more than one MultiPool module does not offer further improvements. We find inserting MultiPool module at second last macro residual block (Res4 or Res5 depending on task) yields the largest gain.

For semantic segmentation, our MultiPool also outperforms the weighted pooling in [32], which uses the same ResNet50 base. We conjecture this is due to three reasons. First, we apply the deconvolutional layer way before the last convolutional layer for softmax input as explained in Section 4.1. This increases resolution that enables the model to see better the fine details. Additionally, our set of pooling regions includes finer scales (rather than using powers of 2). Finally, the results in Table 4 show that PAG with binary masks performs slightly better (IoU=46.5 vs. IoU=46.3) than the (softmax) weighted average operation used in [32].

Computation-Performance Tradeoffs: Lastly, we evaluate how our dynamic parsimonious computation setup impacts performance and performs compared with other baselines. We show results of semantic segmentation on NYUv2 dataset in Table 2, comparing different baselines and our models with MultiPool at macro block Res5, MP@Res5 (PAG) for short, which are trained with different target computational budgets (specified by $\rho$). The "truncated" baseline means we simple remove top layers of ResNet to save computation, while "layer-skipping" is an implementation of [51] that learns to dynamically skip a subset of layer. "PerforatedCNN" is our implementation of [16] that matches the computational budget using a learned constant gating function (not dependent on input image). For fair comparison, we insert MultiPool module at the top of all the compared methods. These results clearly suggest that the PAG approach outperforms all these methods, demonstrating that learning dynamic computation path at the pixel level is helpful for per-pixel labeling tasks. It is also worth noting that PerforatedCNN does not support fully convolutional computation requiring that the input image have a fixed size in order to learn fixed computation paths over the image. In contrast, our method is fully convolutional that is able to take as input images of arbitrary size and perform computing with input-dependent dynamic paths.

Fig. 5 shows that, as we decrease the computation budget, the performance of the PAG-based method degrades gracefully even as the amount of computation is scaled back to 70%, merely inducing 2.4% and 5.6% performance degradation on boundary detection and semantic segmentation compared to their full model, respectively, i.e., $F=0.773$ vs. $F=0.792$ and $IoU=0.409$ vs. $IoU=0.465$. Table 2 highlights the comparison to truncation and layer-skipping models adjusted to match the same computational budget as PAG. For these approaches, performance decays much more sharply with decreasing budget. These results also highlight that the target sparsity parameter $\rho$ provides tight control over the actual average computation of the model.

5.3. Comprehensive Benchmark Comparison

We now compare our models under different degrees of computational parsimony ($\rho\in\{0.5, 0.7, 0.9, 1.0\}$) with other state-of-the-art systems for pixel labeling.

Taking boundary detection as the first task, we quantitatively compare our model to COB [41], HED [56], gPb-UCM [1], LEP [44], UCM [1], NCuts [47], EGB [14], MCG [2] and the mean shift (MShift) algorithm [9]. Table 3 shows comparison to all the methods (PR curves in supplementary material), demonstrating our model achieves state-of-the-art performance. Note that our model has the same backbone architecture of HED [56], but outperforms it with
our MultiPool module which increases receptive fields at higher levels. Our model performs on par with COB [41], which uses auxiliary losses for oriented boundary detection. Note that it is possible to surpass human performance with sophisticated techniques [29], but we don’t pursue this as it is out the scope of this paper.

Table 4, 5 and 6 show the comprehensive comparisons on the tasks of semantic segmentation, monocular depth and surface normal estimation, respectively. In addition to comparing with state-of-the-art methods, we also show the result of MultiPool module with softmax weighted average operation, termed by MP@Res5 (w-Avg.). Interestingly, MultiPool performs slightly better when equipped with PAG than the weighted average fusion. We attribute this to the combination of the proposed PAG MultiPool and carefully designed losses for depth and surface normal estimation.

5.4. Qualitative Visualization

We visualize the prediction and attention maps in Fig. 1 for the four datasets, respectively. We find that the binary attention maps are qualitatively similar across layers and hence summarize them with a “ponder map” by summing maps across layers (per-layer maps can be found in the supplementary material). We can see our models allocate more computation on the regions/pixels which are likely sharp transitions, e.g. boundaries between semantic segments, depth discontinuities and normal discontinuities (e.g. between wall and ceiling).

6. Conclusion and Future Work

In this paper, we have studied the problem of dynamic inference for pixel labeling tasks under limited computation budget with a deep CNN network. To achieve this, we propose a Pixel-wise Attentional Gating unit (PAG) that learns to generate sparse binary masks that control computation at each layer on a per-pixel basis. Our approach differs from previous methods in demonstrating improved performance on pixel labeling tasks using spatially varying computation trained with simple task-specific loss. This makes our approach a good candidate for general use as it is agnostic to tasks and architectures, and avoids more complicated reinforcement learning-style approaches, instead relying on a simple, easy-to-set sparsity target that correlates closely with empirical computational cost. As our PAG is based on a generic attention mechanism, we anticipate future work might explore task-driven constraints for further improvements and savings.

Table 4: Semantic segmentation is measured by Intersection over Union (IoU), pixel accuracy (acc), and iLOU that leverages the size of segments w.r.t categories. Results marked by † are from our trained models with the released code.

Table 5: Depth estimation is measured by standard threshold accuracy, i.e. the percentage of predicted pixel depths $d_i$ s.t. $\delta = \max(\frac{d_i}{5}, \frac{5}{d_i}) < \tau$, where $\tau = \{1.25, 1.25^2, 1.25^3\}$. Methods with * use ~100k extra images to train.

Table 6: Surface normal estimation is measured by mean angular error and the percentage of prediction error within $t^\circ$ degree where $t = \{11.25, 22.50, 30.00\}$. Smaller ang. err. means better performance as marked by ↓.
References


