Abstract

Convolutional neural nets (CNNs) have demonstrated remarkable performance in recent history. Such approaches tend to work in a “unidirectional” bottom-up feed-forward fashion. However, practical experience and biological evidence tells us that feedback plays a crucial role, particularly for detailed spatial understanding tasks. This work explores “bidirectional” architectures that also reason with top-down feedback: neural units are influenced by both lower and higher-level units.

We do so by treating units as rectified latent variables in a quadratic energy function, which can be seen as a hierarchical Rectified Gaussian model (RGs) [39]. We show that RGs can be optimized with a quadratic program (QP), that can in turn be optimized with a recurrent neural network (with rectified linear units). This allows RGs to be trained with GPU-optimized gradient descent. From a theoretical perspective, RGs help establish a connection between CNNs and hierarchical probabilistic models. From a practical perspective, RGs are well suited for detailed spatial tasks that can benefit from top-down reasoning. We illustrate them on the challenging task of keypoint localization under occlusions, where local bottom-up evidence may be misleading. We demonstrate state-of-the-art results on challenging benchmarks.

1. Introduction

Hierarchical models of visual processing date back to the iconic work of Marr [31]. Convolutional neural nets (CNN’s), pioneered by LeCun et al. [27], are hierarchical models that compute progressively more invariant representations of an image in a bottom-up, feedforward fashion. They have demonstrated remarkable progress in recent history for visual tasks such as classification [25,38,43], object detection [8], and image captioning [22], among others.

Feedback in biology: Biological evidence suggests that vision at a glance tasks, such as rapid scene categorization [48], can be effectively computed with feedforward hierarchical processing. However, vision with scrutiny tasks, such as fine-grained categorization [23] or detailed spatial manipulations [19], appear to require feedback along a “reverse hierarchy” [17]. Indeed, most neural connections in the visual cortex are believed to be feedback rather than feedforward [4,26].

Feedback in computer vision: Feedback has also played a central role in many classic computer vision models. Hierarchical probabilistic models [20,28,55], allow
random variables in one layer to be naturally influenced by those above and below. For example, lower layer variables may encode edges, middle layer variables may encode parts, while higher layers encode objects. Part models [5] allow a face object to influence the activation of an eye part through top-down feedback, which is particularly vital for occluded parts that receive misleading bottom-up signals. Interestingly, feed-forward inference on part models can be written as a CNN [9], but the proposed mapping does not hold for feedback inference.

Overview: To endow CNNs with feedback, we treat neural units as nonnegative latent variables in a quadratic energy function. When probabilistically normalized, our quadratic energy function corresponds to a Rectified Gaussian (RG) distribution, for which inference can be cast as a quadratic program (QP) [39]. We demonstrate that coordinate descent optimization steps of the QP can be “unrolled” into a recurrent neural net with rectified linear units. This observation allows us to discriminatively-tune RGs with neural network toolboxes: we tune Gaussian parameters such that, when latent variables are inferred from an image, the variables act as good features for discriminative tasks. From a theoretical perspective, RGs help establish a connection between CNNs and hierarchical probabilistic models. From a practical perspective, we introduce RG variants of state-of-the-art deep models (such as VGG16 [38]) that require no additional parameters, but consistently improve performance due to the integration of top-down knowledge.

2. Hierarchical Rectified Gaussians

In this section, we describe the Rectified Gaussian models of Socci and Seung [39] and their relationship with rectified neural nets. Because we will focus on convolutional nets, it will help to think of variables $z = [z_i]$ as organized into layers, spatial locations, and channels (much like the neural activations of a CNN). We begin by defining a quadratic energy over variables $z$:

$$S(z) = \frac{1}{2} z^T W z + b^T z$$

$$P(z) \propto e^{S(z)}$$

Boltzmann: $z_i \in \{0, 1\}, w_{ij} = 0$

Gaussian: $z_i \in \mathbb{R}, -W$ is PSD

Rect. Gaussian: $z_i \in \mathbb{R}^+, -W$ is copositive

where $W = [w_{ij}], b = [b_i]$. The symmetric matrix $W$ captures bidirectional interactions between low-level features (e.g., edges) and high-level features (e.g., objects). Probabilistic models such as Boltzmann machines, Gaussians, and Rectified Gaussians differ simply in restrictions on the latent variable - binary, continuous, or nonnegative. Hierarchical models, such as deep Boltzmann machines [36], can be written as a special case of a block-sparse matrix $W$ that ensures that only neighboring layers have direct interactions.

Normalization: To ensure that the scoring function can be probabilistically normalized, Gaussian models require that $(-W)$ be positive semidefinite (PSD) $(-z^T W z \geq 0, \forall z)$ Socci and Seung [39] show that Rectified Gaussians require the matrix $(-W)$ to only be copositive $(-z^T W z \geq 0, \forall z \geq 0)$, which is a strictly weaker condition. Intuitively, copositivity ensures that the maximum of $S(z)$ is still finite, allowing one to compute the partition function. This relaxation significantly increases the expressive power of a Rectified Gaussian, allowing for multimodal distributions. We refer the reader to the excellent discussion in [39] for further details.

Comparison: Given observations (the image) in the lowest layer, we will infer the latent states (the features) from the above layers. Gaussian models are limited in that features will always be linear functions of the image. Boltzmann machines produce nonlinear features, but may be limited in that they pass only binary information across layers [33]. Rectified Gaussians are nonlinear, but pass continuous information across layers: $z_i$ encodes the presence or absence of a feature, and if present, the strength of this activation (possibly emulating the firing rate of a neuron [21]).

Inference: Socci and Seung point out that MAP estimation of Rectified Gaussians can be formulated as a quadratic program (QP) with nonnegativity constraints [39]:

$$\max_{z \geq 0} \frac{1}{2} z^T W z + b^T z$$

However, rather than using projected gradient descent (as proposed by [39]), we show that coordinate descent is particularly effective in exploiting the sparsity of $W$. Specifically, let us optimize a single $z_i$ holding all others fixed. Maximizing a 1-d quadratic function subject to nonnegative constraints is easily done by solving for the optimum and clipping:
max \( f(z_i) \) where \( f(z_i) = \frac{1}{2} w_{ii} z_i^2 + (b_i + \sum_{j \neq i} w_{ij} z_j) z_i \)

\[
\frac{\partial f}{\partial z_i} = w_{ii} z_i + b_i + \sum_{j \neq i} w_{ij} z_j = 0
\]

\[
z_i = \frac{1}{w_{ii}} \max(0, b_i + \sum_{j \neq i} w_{ij} z_j) \tag{3}
\]

\[
= \max(0, b_i + \sum_{j \neq i} w_{ij} z_j) \quad \text{for} \quad w_{ii} = -1
\]

By fixing \( w_{ii} = -1 \) (which we do for all our experiments), the above maximization can solved with a rectified dot-product operation.

**Layerwise updates:** The above updates can be performed for all latent variables in a layer in parallel. With a slight abuse of notation, let us define the input image to be the (observed) bottom-most layer \( x = z_0 \), and the variable at layer \( i \) and spatial position \( u \) is written as \( z_i[u] \). The weight connecting \( z_{i-1}[v] \) to \( z_i[u] \) is given by \( w_{i}[\tau] \), where \( \tau = u - v \) depends only on the relative offset between \( u \) and \( v \) (visualized in Fig. 2):

\[
z_i[u] = \max(0, b_i + \text{top}_{i}[u] + \text{bot}_{i}[u]) \quad \text{where} \quad \text{(4)}
\]

\[\text{top}_{i}[u] = \sum_{\tau} w_{i+1}[\tau] z_{i+1}[u - \tau]\]

\[\text{bot}_{i}[u] = \sum_{\tau} w_{i}[\tau] z_{i-1}[u + \tau]\]

where we assume that layers have a single one-dimensional channel of a fixed length to simplify notation. By tying together weights such that they only depend on relative locations, bottom-up signals can be computed with cross-correlational filtering, while top-down signals can be computed with convolution. In the existing literature, these are sometimes referred to as deconvolutional and convolutional filters (related through a \( 180^\circ \) rotation) [53]. It is natural to start coordinate updates from the bottom layer \( z_1 \), initializing all variables to 0. During the initial bottom-up coordinate pass, \( \text{top}_{i} \) will always be 0. This means that the bottom-up coordinate updates can be computed with simple filtering and thresholding. **Hence a single bottom-up pass of layer-wise coordinate optimization of a Rectified Gaussian model can be implemented with a CNN.**

**Top-down feedback:** We add top-down feedback simply by applying additional coordinate updates (4) in a top-down fashion, from the top-most layer to the bottom. Fig. 3 shows that such a sequence of bottom-up and top-down updates can be “unrolled” into a feed-forward CNN with “skip” connections between layers and tied weights. One can interpret such a model as a recurrent CNN that is capable of feedback, since lower-layer variables (capturing say, edges) can now be influenced by the activations of high-layer variables (capturing say, objects). Note that we make use of recurrence along the depth of the hierarchy, rather than along time or spacial dimensions as is typically done [14]. When the associated weight matrix \( W \) is copositive, an infinitely-deep recurrent CNN must converge to the solution of the QP from (2).

**Non-maximal suppression (NMS):** To encourage sparse activations, we add lateral inhibitory connections between variables from same groups in a layer. Specifically, we write the weight connecting \( z_{i-1}[u] \) and \( z_{i}[v] \) for \( (u,v) \in \text{group} \) as \( w_i[u,v] = -\infty \). Such connections are shown as red edges in Fig. 2. For disjoint groups (say, non-overlapping 2x2 windows), layerwise updates correspond to filtering, rectification (4), and non-maximal suppression (NMS) within each group.

Unlike max-pooling, NMS encodes the spatial location of the max by returning 0 values for non-maximal locations. Standard max-pooling can be obtained as a special case by replicating filter weights \( w_{i+1} \) across variables \( z_i \) within the same group (as shown in Fig. 2). This makes NMS independent of the top-down signal \( \text{top}_{i} \). However, our approach is more general in that NMS can be guided by top-down feedback: high-level variables (e.g., car detections) influence the spatial location of low-level variables (e.g., wheels), which is particularly helpful when parsing occluded wheels. Interestingly, top-down feedback seems to encode spatial information without requiring additional “capsule” variables [15].

**Approximate inference:** Given the above global scoring function and an image \( x \), inference corresponds to \( \arg\max_z S(x, z) \). As argued above, this can be implemented with an infinitely-deep unrolled recurrent CNN. However, rather than optimizing the latent variables to completion, we perform a fixed number \( k \) of layer-wise coordinate descent updates. This is guaranteed to report back finite variables \( z^* \) for any weight matrix \( W \) (even when not

![Figure 3](image-url)
\[ z^* = \mathbf{QP}_k(x, W, b), \quad z^* \in \mathbb{R}^N \]  

We write \( \mathbf{QP}_k \) in bold to emphasize that it is a *vector-valued function* implementing \( k \) passes of layer-wise coordinate descent on the \( \mathbf{QP} \) from (2), returning a vector of all \( N \) latent variables. We set \( k = 1 \) for a single bottom-up pass (corresponding to a standard feed-forward CNN) and \( k = 2 \) for an additional top-down pass. We visualize examples of recurrent CNNs that implement \( \mathbf{QP}_1 \) and \( \mathbf{QP}_2 \) in Fig. 4.

**Output prediction:** We will use these \( N \) variables as features for \( M \) recognition tasks. In our experiments, we consider the task of predicting heatmaps for \( M \) keypoints. Because our latent variables serve as rich, multi-scale description of image features, we assume that simple linear predictors built on them will suffice:

\[ y = V^T z^*, \quad y \in \mathbb{R}^M, V \in \mathbb{R}^{N \times M} \]

**Training:** Our overall model is parameterized by \((W, V, b)\). Assume we are given training data pairs of images and output label vectors \( \{x_i, y_i\} \). We define a training objective as follows

\[
\min_{W,V,b} R(W) + R(V) + \sum_i \text{loss}(y_i, V^T \mathbf{QP}_k(x_i, W, b))
\]

where \( R \) are regularizer functions (we use the Frobenius matrix norm) and “loss” sums the loss of our \( M \) prediction tasks (where each is scored with log or softmax loss). We optimize the above by stochastic gradient descent. Because \( \mathbf{QP}_k \) is a deterministic function, its gradient with respect to \((W, b)\) can be computed by backprop on the \( k \)-times unrolled recurrent CNN (Fig. 3). We choose to separate \( V \) from \( W \) to ensure that feature extraction does not scale with the number of output tasks (\( \mathbf{QP}_k \) is independent of \( M \)). During learning, we fix diagonal weights (\( w_i[u, u] = -1 \)) and lateral inhibition weights (\( w_i[u, v] = -\infty \) for \((u, v) \in \text{group}\).

**Related work (learning):** The use of gradient-based backpropagation to learn an unrolled model dates back to ‘backprop-through-structure’ algorithms [11,40] and graph transducer networks [27]. More recently, such approaches were explored general graphical models [41] and Boltzmann machines [12]. Our work uses such ideas to learn CNNs with top-down feedback using an unrolled latent-variable model.

**Related work (top-down):** Prior work has explored networks that reconstruct images given top-down cues. This is often cast as unsupervised learning with autoencoders [16,32,49] or deconvolutional networks [53], though supervised variants also exist [29,34]. Our network differs in that all nonlinear operations (rectification and max-pooling) are influenced by both bottom-up and top-down knowledge (4), which is justified from a latent-variable perspective.

### 3. Implementation

In this section, we provide details for implementing \( \mathbf{QP}_1 \) and \( \mathbf{QP}_2 \) with existing CNN toolboxes. We visualize our specific architecture in Fig. 4, which closely follows the state-of-the-art VGG-16 network [38]. We use \( 3 \times 3 \) filters and \( 2 \times 2 \) non-overlapping pooling windows (for NMS). Note that, when processing NMS-layers, we conceptually use \( 6 \times 6 \) filters with replication after NMS, which in practice can be implemented with standard max-pooling and \( 3 \times 3 \) filters (as argued in the previous section). Hence \( \mathbf{QP}_1 \) is essentially a re-implementation of VGG-16.

**\( \mathbf{QP}_2 \):** Fig. 5 illustrates top-down coordinate updates, which require additional feedforward layers, skip connections, and tied weights. Even though \( \mathbf{QP}_2 \) is twice as deep as \( \mathbf{QP}_1 \) (and [38]), *it requires no additional parameters*. Hence top-down reasoning “comes for free”. There is a small notational inconvenience at layers that decrease in size. In typical CNNs, this decrease arises from a previous pooling operation. Our model requires an explicit \( 2 \times \) subsampling step (sometimes known as strided filtering) because it employs NMS instead of max-pooling. When this subsampled layer is later used to produce a top-down signal for a future coordinate update, variables must be zero-interlaced before applying the \( 180^\circ \) rotated convolutional filters (as shown by hollow circles in Fig. 5). Note that is **not** an approximation, but the mathematically-correct application of coordinate descent given subsampled weight connections.

**Supervision y:** The target label for a single keypoint is a sparse 2D heatmap with a ‘1’ at the keypoint location (or all ‘0’s if that keypoint is not visible on a particular training image). We score this heatmap with a per-pixel log-loss. In practice, we assign ‘1’s to a circular neighborhood that implicitly adds jittered keypoints to the set of positive examples.

**Multi-scale classifiers V:** We implement our output classifiers (7) as multi-scale convolutional filters defined over different layers of our model. We use upsampling to enable efficient coarse-to-fine computations, as described for fully-convolutional networks (FCNs) [29] (and shown in Fig. 4). Specifically, our multi-scale filters are implemented as \( 1 \times 1 \) filters over 4 layers (referred to as fc7, pool4, pool3, and pool2 in [38]). Because our top (fc7) layer is limited in spatial resolution (\( 1 \times 1 \times 4096 \)), we define our coarse-scale filter to be “spatially-varying”, which can alternatively be thought of as a linear “fully-connected” layer that is reshaped to predict a coarse (\( 7 \times 7 \)) heatmap of...
Figure 4: We show the architecture of QP_2, implemented in our experiments. QP_1 corresponds to the left half of QP_2, which essentially resembles the state-of-the-art VGG-16 CNN [38]. QP_2 is implemented with an 2X “unrolled” recurrent CNN with transposed weights, skip connections, and zero-interlaced upsampling (as shown in Fig. 5). Importantly, QP_2 does not require any additional parameters. Red layers include lateral inhibitory connections enforced with NMS. Purple layers denote multi-scale convolutional filters that (linearly) predict keypoint heatmaps given activations from different layers. Multi-scale filters are efficiently implemented with coarse-to-fine upsampling [29], visualized in the purple dotted rectangle (to reduce clutter, we visualize only 3 of the 4 multiscale layers). Dotted layers are not implemented to reduce memory.

Figure 5: Two-pass layer-wise coordinate descent for a two-layer Rectified Gaussian model can be implemented with modified CNN operations. White circles denote 0’s used for interlacing and border padding. We omit rectification operations to reduce clutter. We follow the color scheme from Fig. 2.

keypoint predictions given fc7 features. Our intuition is that spatially-coarse global features can still encode global constraints (such as viewpoints) that can produce coarse keypoint predictions. This coarse predictions are upsamples and added to the prediction from pool4, and so on (as in [29]).

Multi-scale training: We initialize parameters of both QP_1 and QP_2 to the pre-trained VGG-16 model [38], and follow the coarse-to-fine training scheme for learning FCNs [29]. Specifically, we first train coarse-scale filters, defined on high-level (fc7) variables. Note that QP_1 and QP_2 are equivalent in this setting. This coarse-scale model is later used to initialize a two-scale predictor, where now QP_1 and QP_2 differ. The process is repeated up until the full multi-scale model is learned. To save memory during various stages of learning, we only instantiate QP_2 up to the last layer used by the multi-scale predictor (not suitable for QP_k when k > 2). We use a batch size of 40 images, a fixed learning rate of 10^{-6}, momentum of 0.9 and weight decay of 0.0005. We also decrease learning rates of parameters built on lower scales [29] by a factor of 10. Batch normalization [18] is used before each non-linearity. Both our models and code are available online.

Prior work: We briefly compare our approach to recent work on keypoint prediction that make use of deep architectures. Many approaches incorporate multi-scale cues by evaluating a deep network over an image pyramid [44, 46, 47]. Our model processes only a single image scale, extracting multi-scale features from multiple layers of a single network, where importantly, fine-scale features are refined through top-down feedback. Other approaches cast the problem as one of regression, where (x,y) keypoint locations are predicted [54] and often iteratively refined [3, 42]. Our models predict heatmaps, which can be thought of as marginal distributions over the (x,y) location of a keypoint, capturing uncertainty. We show that by thresholding the heatmap value (certainty), one can also produce keypoint visibility estimates “for free”. Our comments hold for our bottom-up model QP_1, which can be thought of as a FCN tuned for keypoint heatmap prediction, rather than seman-

1https://github.com/peiyunh/rg-mpii
tic pixel labeling. Indeed, we find such an approach to be a surprisingly simple but effective baseline that outperforms much prior work.

4. Experiment Results

We evaluated fine-scale keypoint localization on several benchmark datasets of human faces and bodies. To better illustrate the benefit of top-down feedback, we focus on datasets with significant occlusions, where bottom-up cues will be less reliable. All datasets provide a rough detection window for the face/body of interest. We crop and resize detection windows to 224x224 before feeding into our model. Recall that QP1 is essentially a re-implementation of a FCN [29] defined on a VGG-16 network [38], and so represents quite a strong baseline. Also recall that QP2 adds top-down reasoning without any increase in the number of parameters. We will show this consistently improves performance, sometimes considerably. Unless otherwise stated, results are presented for a 4-scale multi-scale model.

AFLW: The AFLW dataset [24] is a large-scale real-world collection of 25,993 faces in 21,997 real-world images, annotated with facial keypoints. Notably, these faces are not limited to responses from an existing face detector, and so this dataset contains more pose variation than other landmark datasets. We hypothesize that such pose variation might illustrate the benefit of bidirectional reasoning. Due to a lack of standard splits, we randomly split the dataset into training (60%), validation (20%) and test (20%). As this is not a standard benchmark dataset, we compare ourselves for exploring the best practices to build multi-scale predictors for keypoint localization (Fig. 6). We include qualitative visualizations in our supplementary material.

COFW: Caltech Occluded Faces-in-the-Wild (COFW) [2] is dataset of 1007 face images with severe occlusions. We present qualitative results in Fig. 7 and Fig. 8, and quantitative results in Table 1 and Fig. 9. Our bottom-up QP1 already performs near the state-of-the-art, while the QP2 significantly improves in accuracy of visible landmark localization and occlusion prediction. In terms of the latter, our model even approaches upper bounds that make use of ground-truth segmentation labels [7]. Our models are not quite state-of-the-art in localizing occluded points. We believe this may point to a limitation in the underlying benchmark. Consider an image of a face mostly occluded by the hand (Fig. 7). In such cases, humans may not even agree on keypoint locations, indicating that a keypoint distribution may be a more reasonable target output. Our models provide such uncertainty estimates, while most keypoint architectures based on regression cannot.

Pascal Person: The Pascal 2011 Person dataset [13] consists of 11,599 person instances, each annotated with a bounding box around the visible region and up to 23 human keypoints per person. This dataset contains significant occlusions. We follow the evaluation protocol of [30].
Figure 8: Facial landmark localization and occlusion prediction results of $Q P_2$ on COFW, where red means occluded. Our bidirectional model is robust to occlusions caused by objects, hair, and skin. We also show cases where the model correctly predicts visibility but fails to accurately localize occluded landmarks (b).

<table>
<thead>
<tr>
<th></th>
<th>Visible Points</th>
<th>All Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPP [51]</td>
<td>-</td>
<td>7.52</td>
</tr>
<tr>
<td>HPM [6]</td>
<td>-</td>
<td>7.46</td>
</tr>
<tr>
<td>SAPM [7]</td>
<td>5.77</td>
<td>6.89</td>
</tr>
<tr>
<td>FLD-Full [50]</td>
<td>5.18</td>
<td>5.93</td>
</tr>
<tr>
<td>$Q P_1$</td>
<td>5.26</td>
<td>10.06</td>
</tr>
<tr>
<td>$Q P_2$</td>
<td><strong>4.67</strong></td>
<td>7.87</td>
</tr>
</tbody>
</table>

Table 1: Average keypoint localization error (as a fraction of inter-ocular distance) on COFW. When adding top-down feedback ($Q P_2$), our accuracy on visible keypoints significantly improves upon prior work. In the text, we argue that such localization results are more meaningful than those for occluded keypoints. In Fig. 9, we show that our models significantly outperform all prior work in terms of keypoint visibility prediction.

and present results for localization of visible keypoints on a standard testset in Table 2. Our bottom-up $Q P_1$ model already significantly improves upon the state-of-the-art (including prior work making use of deep features), while our top-down models $Q P_2$ further improve accuracy by 2% without any increase in model complexity (as measured by the number of parameters). Note that the standard evaluation protocols evaluate only visible keypoints. In supplementary material, we demonstrate that our model can also accurately predict keypoint visibility “for free”.

**MPII:** MPII is (to our knowledge) the largest available articulated human pose dataset [1], consisting of 40,000 people instances annotated with keypoint, visibility flags, and activity labels. We present qualitative results in Fig. 11 and quantitative results in Table 3. Our top-down model $Q P_2$ appears to outperform all prior work on full-body keypoints. Note that this dataset also includes visibility labels for keypoints, even though these are not part of the standard evaluation protocol. In supplementary material, we demonstrate that visibility prediction on MPII also benefits from top-down feedback.

**TB:** It is worth contrasting our results with TB [45], which implicitly models feedback by (1) using a MRF to post-process CNN outputs to ensure kinematic consistency between keypoints and (2) using high-level predictions from a coarse CNN to adaptively crop high-res features for a fine CNN. Our single CNN endowed with top-down feedback is slightly more accurate without requiring any additional parameters, while being 2X faster (86.5 ms vs TB’s 157.2 ms). These results suggest that top-down reasoning may elegantly capture structured outputs and attention, two active areas of research in deep learning.
Figure 10: We visualize bottom-up and top-down models trained for human pose estimation, using the naming convention of Fig. 6. Top-down feedback (QP₂) more accurately guides finer-scale predictions, resolving left-right ambiguities in the ankle (red) and poor localization of the knee (green) in the bottom-up model (QP₁).

Figure 11: Keypoint localization results of QP₂ on the MPII Human Pose testset. We quantitatively evaluate results on the validation set in Table 2. Our models are able to localize keypoints even under significant occlusions. Recall that our models can also predict visibility labels “for free”, which are evaluated in supplementary material.

Table 3: We show PCKh-0.5 keypoint localization results on MPII using the recommended benchmark protocol [1].

<table>
<thead>
<tr>
<th>Head</th>
<th>Shou</th>
<th>Elb</th>
<th>Wri</th>
<th>Hip</th>
<th>Kne</th>
<th>Ank</th>
<th>Upp</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM [10]</td>
<td>-</td>
<td>36.3</td>
<td>26.1</td>
<td>15.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>25.9</td>
</tr>
<tr>
<td>ST [37]</td>
<td>-</td>
<td>38.0</td>
<td>26.3</td>
<td>19.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>27.9</td>
</tr>
<tr>
<td>YR [52]</td>
<td>73.2</td>
<td>56.2</td>
<td>41.3</td>
<td>32.1</td>
<td>36.2</td>
<td>33.2</td>
<td>34.5</td>
<td>43.2</td>
</tr>
<tr>
<td>PS [35]</td>
<td>74.2</td>
<td>49.0</td>
<td>40.8</td>
<td>34.1</td>
<td>36.5</td>
<td>34.4</td>
<td>35.1</td>
<td>41.3</td>
</tr>
<tr>
<td>TB [45]</td>
<td>96.1</td>
<td>91.9</td>
<td>83.9</td>
<td>77.8</td>
<td>80.9</td>
<td>72.3</td>
<td>64.8</td>
<td>84.5</td>
</tr>
<tr>
<td>QP₁</td>
<td>94.3</td>
<td>90.4</td>
<td>81.6</td>
<td>75.2</td>
<td>80.1</td>
<td>73.0</td>
<td>68.3</td>
<td>82.4</td>
</tr>
<tr>
<td>QP₂</td>
<td>95.0</td>
<td>91.6</td>
<td>83.0</td>
<td>76.6</td>
<td>81.9</td>
<td>74.5</td>
<td>69.5</td>
<td>83.8</td>
</tr>
</tbody>
</table>

Table 4: PCKh(0.5) on MPII-Val for a smaller network sized inputs for 100 epochs. As shown in Table 4, we conclude: (1) all recurrent models outperform the bottom-up baseline QP₁; (2) additional iterations generally helps, but performance maxes out at QP₄. A two-pass model (QP₂) is surprisingly effective at capturing top-down info while being fast and easy to train.

**Conclusion:** We show that hierarchical Rectified Gaussian models can be optimized with rectified neural networks. From a modeling perspective, this observation allows one to discriminatively-train such probabilistic models with neural toolboxes. From a neural net perspective, this observation provides a theoretically-elegant approach for endowing CNNs with top-down feedback – *without any increase in the number of parameters*. To thoroughly evaluate our models, we focus on “vision-with-scrutiny” tasks such as keypoint localization, making use of well-known benchmark datasets. We introduce (near) state-of-the-art bottom-up baselines based on multi-scale prediction, and consistently improve upon those results with top-down feedback (particularly during occlusions when bottom-up evidence may be ambiguous).

**Acknowledgments:** This research is supported by NSF Grant 0954083 and by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R & D Contract No. 2014-14071600012. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

**More recurrence iterations:** To explore QPₖ’s performance as a function of K without exceeding memory limits, we trained a smaller network from scratch on 56X56
References


