Simpler Certified Radius Maximization by Propagating Covariances Supplementary

Xingjian Zhen[†], Rudrasis Chakraborty[‡], Vikas Singh[†]
[†]University of Wisconsin-Madison [‡]University of California, Berkeley
xzhen3@wisc.edu rudrasischa@gmail.com vsingh@biostat.wisc.edu

A. Difficulties Of Accounting For All Terms

In the main paper, we described at a high level, the problem of bookkeeping all covariances Σ and cross-correlations E because the number of terms grow very rapidly. Here, we provide the low-level details of where/why this problem arises and strategies to address it.

A.1. A 1-D Convolution Without Overlap: Exponential Explosion

As shown in Table. d, even if we only consider a simple case – a 1-D convolution network with a kernel size $k = 3$, we run into problems with a naive bookkeeping approach. Let us analyze this at the last layer and trace back the number of terms that will contribute to it.

In the last layer, we will need to calculate a single covariance matrix. Tracing back, in the second last layer, we will need to compute 3 covariance matrices Σ and 3 cross-correlation matrices E. In the third last layer, we will need to compute 3^2 covariance matrices Σ and $\frac{1}{2}(3^2+1)3^2$ cross-correlation matrices E. Then, in the $(q+1)^{th}$ last layer, we will require k^q covariance matrices Σ and $\frac{1}{2}(1+k^q)k^q = \Theta(k^q)$ cross-correlation matrices E. In this case, the computational cost increase exponentially, which is infeasible. in case of a deep neural network. This setup will rapidly increase the field of view of the network, and one would need special types of convolution, e.g., dilated convolutions, to address the problem [3].

A.2. A 1-D Convolution With Overlap: Polynomial But Higher Than Standard 2-D Convolution

The setup above did not consider the overlap between pixels as the shared kernel moves within the same layer. If we consider a setting with overlap as in a standard CNN, as shown in Table. e, similarly, in the last layer, we will need to compute a single covariance matrix. In the second last layer, we will need to compute 3 covariance matrices Σ and 3 cross-correlation matrices E. In the third last layer, we will need to compute $(3-1)\times 2+1$ covariance matrices Σ and $\frac{1}{2}((3-1)\times 2+1+1)((3-1)\times 2+1)$ cross-correlation matrices E. Then, in the $(q+1)^{th}$ last layer, it will require $(k-1)q+1$ covariance matrices Σ and $\frac{1}{2}((k-1)q+1+1)((k-1)q+1) = \Theta(k^2q^2)$ cross-correlation matrices E.

This may appear feasible since the computational cost is polynomial. But this setup indeed increases the computational cost of a 1-D convolution network $\Theta(k^2q^2)$ to be higher than a 2-D convolution network, $\Theta(k^2q)$ since $q < q^2$. When this strategy is applied to the 2-D image case, the computational cost ($\Theta(k^4q^2)$) will be more than the 4-D tensor convolution network ($\Theta(k^4q)$), which is not feasible. As shown in the literature, training CNNs efficiently on very high resolution 3-D images (or videos) is still an active topic of ongoing research.

A.3. Memory Cost

The GPU memory footprint also turns out to be high. For a direct impression of the numbers, we take the ImageNet with PreActResnet-18 as an example, shown in Table. a. The memory cost for a naive method is too high, even for large clusters, if we want to track all cross-correlation terms between any two pixels and the channels.

B. Tracking Distributions Through Layers

In the main paper, we briefly described the different layers used in our model. Here, we will provide more details about the layers.

We consider the i^{th} pixel, after perturbation, to be drawn from a Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, where $\boldsymbol{\mu}_i \in \mathbf{R}^N$ and Σ is the covariance matrix across the N channels (note that Σ is same for a layer across all pixels). In the following sections, we will remove the indices to simplify the formulation.

Several commonly-used basic blocks such as convolution and fully connected layers are used to setup our network architecture. In order to propagate the distribution through the entire network, we need a way to propagate the moments through these layers.

Table a: The memory cost of ImageNet with PreActResnet-18 for different layers. The brute force method would require to compute all the cross-correlation between different pixels and channels. The memory cost for the brute force method is too high to afford. Our method, in comparison, only increase a little from the tradition (non-robust) network.

Layer	Traditional network	Brute force method	Our method
Input ($224 \times 224 \times 3$)	$224 \times 224 \times 3 = 150528$	$\frac{1}{2}(224 \times 224 \times 3)^2 = 11329339392$	$(224 \times 224 \times 3) + (3 \times 3) = 150537$
1^{st} convolution ($112 \times 112 \times 64$)	$112 \times 112 \times 64 = 802816$	$\frac{1}{2}(112 \times 112 \times 64)^2 = 644513529856$	$(112 \times 112 \times 64) + (64 \times 64) = 806912$
1^{st} res-block ($56 \times 56 \times 64$)	$56 \times 56 \times 64 = 200704$	$\frac{1}{2}(56 \times 56 \times 64)^2 = 322256764928$	$(56 \times 56 \times 64) + (64 \times 64) = 204800$
2^{nd} res-block ($28 \times 28 \times 128$)	$28 \times 28 \times 128 = 100352$	$\frac{1}{2}(28 \times 28 \times 128)^2 = 5035261952$	$(28 \times 28 \times 128) + (128 \times 128) = 116736$
3^{rd} res-block ($14 \times 14 \times 256$)	$14 \times 14 \times 256 = 50176$	$\frac{1}{2}(14 \times 14 \times 256)^2 = 1258815488$	$(14 \times 14 \times 256) + (256 \times 256) = 115712$
4^{th} res-block ($7 \times 7 \times 512$)	$7 \times 7 \times 512 = 25088$	$\frac{1}{2}(7 \times 7 \times 512)^2 = 314703872$	$(7 \times 7 \times 512) + (512 \times 512) = 287232$

B.1. Convolution Layer:

Let the pixels $\{\mathbf{x}_i\}$ inside a $k \times k$ convolution kernel window be independent. Let $\widetilde{\mathbf{x}} \in \mathbf{R}^{N_{in}k^2}$ be the vector which consists of $\{\mathbf{x}_i\}$, where N_{in} is the number of input channels. Let $\Sigma \in \mathbf{R}^{N_{in} \times N_{in}}$ be the covariance matrix of each pixel within the $k \times k$ window. Let W be the weight matrix of the convolution layer, which is of the shape $N_{in} \times k \times k \times N_{out}$. With a slight abuse of notation, let W be the reshaped weight matrix of the shape $N_{in}k^2 \times N_{out}$. Further, concatenate Σ from each $\{\mathbf{x}_i\}$ inside the $k \times k$ window to get a block diagonal $\widetilde{\Sigma} \in \mathbf{R}^{N_{in}k^2 \times N_{in}k^2}$. Then, we get the covariance of an output pixel to be $\Sigma_{\mathbf{h}} \in \mathbf{R}^{N_{out} \times N_{out}}$ defined as

$$\Sigma_{\mathbf{h}} = W^T \widetilde{\Sigma} W$$

We need to apply Theorem 2 (from the main paper) to compute the upper bound of $\Sigma_{\mathbf{h}}$. We get the upper bound covariance matrix (covariance matrix of independent random variable $\widehat{\mathbf{x}}$ used in Theorem 2) as

$$\widehat{\Sigma_{\mathbf{h}}} = (1 + r_{\max}) W^T \widetilde{\Sigma} W$$

Summary: Given each input pixel, $\mathbf{x}_i \in \mathbf{R}^{N_{in}}$ following $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$ and convolution kernel matrix W, the output distribution of each pixel is $\mathcal{N}(\boldsymbol{\mu}_{h_i}, (1 + r_{max}) W^T \widetilde{\Sigma} W)$, where, $\widetilde{\Sigma}$ is the block diagonal covariance matrix as mentioned before.

B.2. First Linear Layer:

For the first linear layer, we reshape the input in a vector by flattening both the channel and spatial dimensions. Similar to the convolution layer, we concatenate $\{\mathbf{x}_i\}$ to be $\widetilde{\mathbf{x}}$, whose covariance matrix has a block-diagonal structure. Thus, the covariance matrix of the output pixel is

$$\Sigma_{\mathbf{h}} = W^T \Sigma_{\widetilde{\mathbf{x}}} W$$

where W is the learnable parameter and $\Sigma_{\widetilde{\mathbf{x}}}$ is the block-diagonal covariance matrix of $\widetilde{\mathbf{x}}$ similar to the convolution layer.

Special case: From Obs. 1, we only need the largest two intensities to estimate the $\underline{p_{c_{\mathbf{x}}}}$. Thus if there is only one linear layer as the last layer in the entire network (as in most ResNet like models), this can be further simplified: it needs computing a 2×2 covariance matrix instead of the $C \times C$ covariance matrix.

B.3. Linear Layer:

If the network consists of multiple linear layers, calculating the moments of the subsequent linear layers is performed differently. Since it contains only 1-D inputs, we can either treat it spatially or channel-wise. In our setup, we consider it as along the channels.

Let $\mathbf{x} \in \mathbf{R}^{N_i}$ be the input of the i^{th} linear layer, where $i > 1$. Assume, $\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\mathbf{x}}^{(i)}, \Sigma_{\mathbf{x}}^{(i)}\right)$. Given the i^{th} linear layer with parameter $(W_i, \mathbf{b}_i)$, with the output given by

$$\mathbf{h} = W_i^T \mathbf{x} + \mathbf{b}_i$$

$$\mathbf{h} \sim \mathcal{N}\left(W_i^T \boldsymbol{\mu}_{\mathbf{x}}^{(i)} + \mathbf{b}_i, W_i^T \Sigma_{\mathbf{x}}^{(i)} W_i\right)$$

Here, $W_i \in \mathbf{R}^{N_i \times N_{i+1}}$ and $\mathbf{b}_i \in \mathbf{R}^{N_{i+1}}$ and $\mathbf{h} \in \mathbf{R}^{N_{i+1}}$.

B.4. Pooling Layer:

Recall that the input of a max pooling layer is $\{\mathbf{x}_i\}$ where each $\mathbf{x}_i \in \mathbf{R}^{N_{in}}$ and the index i varies over the spatial dimension. Observe that as we identify each $\mathbf{x}_i$ by the respective distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, applying max pooling over $\mathbf{x}_i$ essentially requires computing the maximum over $\{\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)\}$. Thus, we restrict ourselves to average pooling. To be precise, with a kernel window $\mathbb{W}$ of size $k \times k$ with stride k used for average pooling, the output of average pooling, denoted by

$$\mathbf{h} \sim \mathcal{N}\left(\frac{1}{k^2} \sum_{\mathbf{x}_i \in \mathbb{W}} \boldsymbol{\mu}_i, \frac{\Sigma}{k^2}\right)$$

B.5. Normalization Layer:

For the normalization layer, given by $\mathbf{h} = (\mathbf{x} - \mu)/\sigma$, where μ, σ can be computed in different ways [9, 2, 18], we have

$$\mathbf{h} \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_{\mathbf{x}}^{(i)} - \mu}{\sigma}, \frac{\Sigma_{\mathbf{x}}^{(i)}}{\sigma^2}\right)$$

However, as the normalization layers often have large Lipschitz constant [1], we remove those layers in this work.

Batch normalization: For the batch normalization, the mean and variance are computed within each mini-batch [9].

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

where m is the size of the mini-batch. One thing to notice that $\mu, \sigma \in \mathbf{R}^N$ when $x_i \in \mathbf{R}^N$, N is the channel size. In this setting, the way to compute the bounded distribution of output will need to be modified as following:

$$\mathbf{h} \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_{\mathbf{x}}^{(i)} - \mu}{\sigma}, \frac{\Sigma_{\mathbf{x}}^{(i)}}{\sigma\sigma^T}\right)$$

where "/" is the element-wise divide. $\sigma\sigma^T \in \mathbf{R}^{N\times N}$. And μ, σ will be computed dynamically as the new data being fed into the network.

Act normalization: As the performance for batch normalization being different between training and testing period, there are several other types of normalization layers. One of the recent one [11] proposed an act normalization layer where μ, σ are trainable parameters. These two parameters are initialed during warm-up period to compute the mean and variance of the training dataset. After initialization, these parameters are trained freely.

In this case, the update rule is the same as above. The only difference is that μ, σ do not depended on the data being fed into the network.

B.6. Activation Layer:

ReLU: In [4, 13], the authors introduced a way to compute the mean and variance after ReLU. Since ReLU is an element-wise operation, assume $x \sim \mathcal{N}(\mu, \sigma^2)$. After ReLU activation, the first and second moments of the output are given by:

$$\mathbb{E}(\text{ReLU}(x)) = \frac{1}{2}\mu - \frac{1}{2}\mu\,\mathsf{erf}(\frac{-\mu}{\sqrt{2}\sigma}) + \frac{1}{\sqrt{2\pi}}\sigma\exp(-\frac{\mu^2}{2\sigma^2})$$

$$\text{var}(\text{ReLU}(x)) < \text{var}(x)$$

Here, erf is the Error function. We need to track an upper bound of the covariance matrix, so we use $\text{ReLU}(x) \sim \mathcal{N}(\mu_a, \Sigma_a)$, where,

$$\mu_a = \frac{1}{2}\mu - \frac{1}{2}\mu\,\mathsf{erf}(\frac{-\mu}{\sqrt{2}\sigma}) + \frac{1}{\sqrt{2\pi}}\sigma\exp(-\frac{\mu^2}{2\sigma^2}),$$
$$\Sigma_a \preceq \Sigma$$

Last layer/prediction: Here, the last layer is the layer before softmax layer, which represents the "strength" of the model for the label l. By Obs. 1, we have the estimation of

$$p_{c_{\mathbf{x}}} = \underline{p_{c_{\mathbf{x}}}} = \Phi(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}})$$

and

$$p_{\widetilde{c}} = \overline{p_{\widetilde{c}}} = 1 - p_{c_{\mathbf{x}}}$$

By Theorem 1, the certified radius is

$$C_R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_{\mathbf{x}}}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}}))$$
$$= \sigma\frac{\boldsymbol{\mu}[c] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}}$$

Other activation functions: Other than ReLU, there is no closed form to compute the mean after the activation function. One simplification can be: use the local linear function to approximate the activation function, as Gowal, et al. did [7]. The good piece of this approximation is that the output covariance matrix Σ_a can be bounded by $\alpha^2\Sigma$, where α is the largest slope of the linear function.

Another direction is to apply the Hermite expansion [14] which will add more parameters but is theoretically sound.

As ReLU is the most well-used activation function as well as the elegant closed form mean after the activation function, in this paper, we only focus on the ReLU layer and hold other activation functions into future work.

C. Training Loss

Now that we have defined the basic modules and the ways to propagate the distribution through these modules, we now need to make the entire model trainable. In order to do that, we need to define an appropriate training loss which is the purpose of this section. In the spirit of [19], the training loss contains two parts: the classification loss and the robustness loss, i.e.,

$$l(g_\theta; \mathbf{x}, y) = l_C(g_\theta; \mathbf{x}, y) + \lambda l_{C_R}(g_\theta; \mathbf{x}, y)$$

Let the distribution of the output of the last layer be $u_\theta(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. We need to output the expectation as the prediction, i.e., $\boldsymbol{\mu} = \mathbb{E}[u_\theta(\mathbf{x})]$. Similar to the literature, we use the softmax layer on the expectation to compute the cross-entropy of the prediction and the true label.

$$l_C(g_\theta; \mathbf{x}, y) = y\log(\text{softmax}(\mathbb{E}[u_\theta(\mathbf{x})]))$$

The robustness loss measures the certified radius of each sample. In our case, we have the radius

$$C_R = \sigma\frac{\boldsymbol{\mu}[c] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c, c] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c, \widetilde{c}]}}$$

Thus, we want to maximize the certified radius which is equivalent to minimize the robustness loss.

$$l_{C_R}(g_\theta; \mathbf{x}, y = c_{\mathbf{x}})$$
$$= \max(0, \Gamma - \sigma\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}})$$

where $c_{\mathbf{x}}$ is the true label, $\widetilde{c}$ is the second highest possible label. Γ is the offset to control the certified radius to consider.

D. Proof Of The Theorem

Theorem 1. *[5] Let $f_\theta : \mathbf{R}^d \to \mathcal{Y}$ be any deterministic or random function, and let $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$. Let g_θ be the random smooth classifier defined as $g_\theta(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}(f_\theta(\mathbf{x}+\boldsymbol{\varepsilon}) = c_{\mathbf{x}})$. Suppose $c_{\mathbf{x}}, \widetilde{c} \in \mathcal{Y}$ and $\underline{p_{c_{\mathbf{x}}}}, \overline{p_{\widetilde{c}}} \in [0,1]$ satisfy $\mathbb{P}(f_\theta(\mathbf{x}+\boldsymbol{\varepsilon}) = c_{\mathbf{x}}) \geq \underline{p_{c_{\mathbf{x}}}} \geq \overline{p_{\widetilde{c}}} \geq \max_{\widetilde{c} \neq c_{\mathbf{x}}} \mathbb{P}(f_\theta(\mathbf{x}+\boldsymbol{\varepsilon}) = \widetilde{c})$. Then $g_\theta(\mathbf{x}+\boldsymbol{\delta}) = c_{\mathbf{x}}$ for all $\|\boldsymbol{\delta}\|_2 < C_R$, where $C_R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_{\mathbf{x}}}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}}))$.*

The symbol Φ denotes the CDF of the standard Normal distribution, and Φ and Φ^{-1} are involved because of smoothing the Gaussian perturbation $\boldsymbol{\varepsilon}$.

Proof. To show that $g_\theta(\mathbf{x}+\boldsymbol{\delta}) = c_{\mathbf{x}}$, it follows from the definition of g_θ that we need to show that

$$\mathbb{P}(f(\mathbf{x}+\boldsymbol{\delta}+\boldsymbol{\varepsilon}) = c_{\mathbf{x}}) > \max_{c' \neq c_{\mathbf{x}}} \mathbb{P}(f(\mathbf{x}+\boldsymbol{\delta}+\boldsymbol{\varepsilon}) = c')$$

We will show $\mathbb{P}(f(\mathbf{x}+\boldsymbol{\delta}+\boldsymbol{\varepsilon}) = c_{\mathbf{x}}) > \max_{c' \neq c_{\mathbf{x}}} \mathbb{P}(f(\mathbf{x}+\boldsymbol{\delta}+\boldsymbol{\varepsilon}) = c')$ for every class $c' \neq c_{\mathbf{x}}$. Fix one c' without loss of generality.

Define the random variables

$$X := \mathbf{x} + \boldsymbol{\varepsilon} = \mathcal{N}(\mathbf{x}, \sigma^2 I)$$
$$Y := \mathbf{x} + \boldsymbol{\delta} + \boldsymbol{\varepsilon} = \mathcal{N}(\mathbf{x}+\boldsymbol{\delta}, \sigma^2 I)$$

We know that

$$\mathbb{P}(f_\theta(X) = c_{\mathbf{x}}) \geq \underline{p_{c_{\mathbf{x}}}}, \text{ and } \overline{p_{\widetilde{c}}} \geq \mathbb{P}(f_\theta(X) = c')$$

We need to show

$$\mathbb{P}(f(Y) = c_{\mathbf{x}}) > \mathbb{P}(f(Y) = c')$$

Define the half-spaces:

$$A := \{z : \delta^T(z-x) \leq \sigma ||\delta|| \Phi^{-1}(\underline{p_{c_{\mathbf{x}}}})\}$$
$$B := \{z : \delta^T(z-x) \geq \sigma ||\delta|| \Phi^{-1}(1-\overline{p_{\widetilde{c}}})\}$$

We have $\mathbb{P}(X \in A) = \underline{p_{c_{\mathbf{x}}}}$. Therefore, we know $\mathbb{P}(f(X) = c_{\mathbf{x}}) \leq \mathbb{P}(X \in A)$. Apple the Neyman-Pearson for Gaussians with different means Lemma [16] with $h(z) := \mathbf{1}[f(z) = c_{\mathbf{x}}]$ to conclude:

$$\mathbb{P}(f(Y) = c_{\mathbf{x}} \geq \mathbb{P}(Y \in A)$$

Similarly, we have $\mathbb{P}(X \in B) = \overline{p_{\widetilde{c}}}$, then we know $\mathbb{P}(f(X) = c') \leq \mathbb{P}(X \in B)$. Use the Lemma [16] again with $h(z) := \mathbf{1}[f(z) = c']$:

$$\mathbb{P}(f(Y) = c') \leq \mathbb{P}(Y \in B)$$

Then to guarantee $\mathbb{P}(f(Y) = c_{\mathbf{x}}) > \mathbb{P}(f(Y) = c')$, it suffices to show that $\mathbb{P}(Y \in A) > \mathbb{P}(Y \in B)$. Thus, if and only if

$$||\boldsymbol{\delta}|| < \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_{\mathbf{x}}}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}}))$$

□

E. Proof Of The Observations

Observation 1. *Let u_θ denote the network without the last softmax layer, i.e., the full neural network can be written as*

$$f_\theta(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \ softmax(u_\theta(\mathbf{x}))$$

Let $C = |\mathcal{Y}|$ and assume

$$u_\theta(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

where $\boldsymbol{\mu} \in \mathbf{R}^C$ and $\Sigma \in \mathbf{R}^{C \times C}$. Then the estimation of $\underline{p_{c_{\mathbf{x}}}}$ is

$$\underline{p_{c_{\mathbf{x}}}} = \Phi\left(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}}\right),$$

where

$$c_{\mathbf{x}} = \arg\max_{c \in \mathcal{Y}} \mathbb{P}\left(u_\theta(\mathbf{x}) = c\right)$$
$$\widetilde{c} = \arg\max_{c \in \mathcal{Y}, c \neq c_{\mathbf{x}}} \mathbb{P}\left(u_\theta(\mathbf{x}) = c\right)$$

Proof. Let us call the strength of two labels to be $u^{c_{\mathbf{x}}}, u^{\widetilde{c}}$. We have

$$u^{c_{\mathbf{x}}} \sim \mathcal{N}(\boldsymbol{\mu}[c_{\mathbf{x}}], \Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}]), u^{\widetilde{c}} \sim \mathcal{N}(\boldsymbol{\mu}[\widetilde{c}], \Sigma[\widetilde{c}, \widetilde{c}]).$$

The cross-correlations $E_{c_{\mathbf{x}}, \widetilde{c}} = \Sigma[c_{\mathbf{x}}, \widetilde{c}]$. Thus,

$$\underline{p_{c_{\mathbf{x}}}} = \mathbb{P}(u^{c_{\mathbf{x}}} > u^{\widetilde{c}}) = \mathbb{P}(u^{c_{\mathbf{x}}} - u^{\widetilde{c}} > 0).$$

Let $\eta = u^{c_{\mathbf{x}}} - u^{\widetilde{c}} \in \mathbf{R}$, then

$$\eta \sim \mathcal{N}(\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}], \Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}})$$

Then, to compute $\underline{p_{c_{\mathbf{x}}}}$,

$$\begin{aligned}
\underline{p_{c_{\mathbf{x}}}} &= \mathbb{P}(u^{c_{\mathbf{x}}} > u^{\widetilde{c}}) \\
&= \mathbb{P}(\eta > 0) \\
&= \mathbb{P}\Big(\frac{\eta - \boldsymbol{\mu}[c_{\mathbf{x}}] + \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}} > \\
&\frac{-\boldsymbol{\mu}[c_{\mathbf{x}}] + \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}}\Big) \\
&= \mathbb{P}(\eta' > \frac{-\boldsymbol{\mu}[c_{\mathbf{x}}] + \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}}) \\
&\text{where } \eta' = \frac{\eta - \boldsymbol{\mu}[c_{\mathbf{x}}] + \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}} \sim \mathcal{N}(0,1) \\
&= 1 - \Phi(\frac{-\boldsymbol{\mu}[c_{\mathbf{x}}] + \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}}) \\
&= \Phi(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2E_{c_{\mathbf{x}}, \widetilde{c}}}}), \Phi : c.d.f. \text{ of } \mathcal{N}(0,1)
\end{aligned}$$

□

Observation 2. *With the input perturbation ε to be identical along the spatial dimension and without nonlinear activation function, for q^{th} hidden convolution layer with $\{\mathbf{h}_i\}_{i=1}^{M_q}$ output pixels, we have $\Sigma^{(q)}_{\mathbf{h}_i} = \Sigma^{(q)}_{\mathbf{h}_j}, \forall i, j \in \{1, \cdots, M_q\}$.*

Proof. Let us prove by induction. In the first layer, there is no cross-correlation between pixels, while the input perturbation is given to be identical. Thus, identical assumption is true for the first layer.

Assume in the layer 1 to q, the second moments are all identical within the layer. We need to prove that for layer $q+1$, this property also holds. Consider two pixels in layer $q+1$, $\mathbf{h}_i^{(q+1)}$ and $\mathbf{h}_j^{(q+1)}$. By definition of convolution, we know $\mathbf{h}_i^{(q+1)} = \sum W_k \mathbf{h}_k^{(q)}$ where $\mathbf{h}_k^{(q)}$ is the view field of $\mathbf{h}_i^{(q+1)}$ and W is the shared kernel. So the covariance matrix of $\mathbf{h}_i^{(q+1)}$ is

$$\Sigma^{(q+1)}_{\mathbf{h}_i} = \sum_k W_k^T \Sigma^{(q)}_{\mathbf{h}_k} W_k + \sum_{m,n} W_m^T E^{(q)}_{m,n} W_n$$

By the identical assumption, the first components are the same for $\mathbf{h}_i^{(q+1)}$ and $\mathbf{h}_j^{(q+1)}$. The only problematic component is the second piece which requires computing $E^{(q)}_{m,n}$. For $E^{(q)}_{m,n}$, we can compute recursively that

$$\begin{aligned} E^{(q)}_{m,n} &= \mathbb{E}[(\mathbf{h}_m^{(q)} - \mathbb{E}[\mathbf{h}_m^{(q)}])^T (\mathbf{h}_n^{(q)} - \mathbb{E}[\mathbf{h}_n^{(q)}])] \\ &= g^{(q)}(\Sigma^{(q-1)}, E^{(q-1)}_{m',n'}, W^{(q-1)}) \end{aligned}$$

where $g^{(q)}(\cdot)$ is a determined function. Thus, the only problematic piece is also $E^{(q-1)}_{m',n'}$. If we keep doing until we reach the first layer, we will have

$$\Sigma^{(q)} = f^{(q)} \circ f^{(q-1)} \circ \ldots f^{(1)}(\Sigma^{(1)})$$

with $f(\cdot)$ is the function depending on $g(\cdot)$, which is a determined value across all the pixels. This assumption depends on the linearity of the network, which will in turn separate the first and second order moments when passing through the network.

As a simplification case, we can assume that the pixels in the same layer are independent, which implies $E^{(q)}_{m,n}$ to be 0 everywhere. This case is suitable when we apply Theorem 2 in the main paper to each convolution layer. □

One thing to notice is the activation function. In our method, in order to keep the identical assumption, we need to make the second moments after activation function to be identical when given the identical input. Also, we will need to keep an upper bound so that the whole method is theoretically sound. Due to the fact that the ReLU will reduce the variance of the variable, we assign the output covariance to be the same as the input covariance. Thus, the identical assumption is kept through all the layers.

F. Other Results Of Our Method

We also performed the proposed method on Cifar-100 [12]. This dataset is more challenging than Cifar-10 as the number of classes increases from 10 to 100. The results are shown in Table. b.

Table b: Average test accuracy on pair-flipping with noisy rate 45% over the last ten epochs of Cifar-100. We show the results of Bootstrap[17], S-model[6], Decoupling[15], MentorNet[10], Co-teaching[8], Trunc $\mathcal{L}_q$[20], and Ours.

Method	Bootstrap	S-model	Decoupling	MentorNet	Co-teaching	Trunc $\mathcal{L}_q$	Ours
mean	0.321	0.218	0.261	0.316	0.348	**0.477**	0.346
std	$3.0e-3$	$8.6e-3$	$0.3e-3$	$5.1e-3$	$0.7e-3$	$6.9e-3$	$0.6e-3$

G. Strengths And Limitations Of Our Method

The biggest benefit of our method is the training time, which is also the main focus of the paper. As in the main paper, we have shown that our methods can be $5\times$ faster on Cifar-10, etc. dataset, with a comparable ACR as MACER. In real-world applications, training speed is an important consideration. Thus, our method is a cheaper substitute of MACER with a marginal performance compromise.

On the other hand, we also need to discuss that under which circumstances our method does not perform well. The first case is when the network is extremely deep, e.g., Resnet-101. Due to the nature of upper bound, the estimation of the second moments tends to become looser as the network grows deeper. Thus, this will lead to a looser estimation of the distribution of the last layer and the robustness estimation would be less meaningful. Another minor weakness is when the input perturbation is large, for example $\sigma = 1.0$. As shown in the main paper, the ACR drops from 0.56 to 0.52 on ImageNet when the noise perturbation increases from $\sigma = 0.5$ to $\sigma = 1.0$. The main reason relates to the assumption of a Gaussian distribution. As the perturbation grows larger, the number of channels, by the central limit theorem, should also be larger to satisfy the Gaussian distribution. Thus, when the network structure is fixed, there is an inherent limit on the input perturbation.

We note that to perform a fair comparison, we run Cohen's [5], MACER [19], and our method based on the PreActResnet-18 for the Table. 3 in the main paper. Since the network is shallower than the original MACER paper, the performance numbers reported here are lower. As described above, a large perturbation $\sigma = 1.0$ leads to small drop in performance. Thus, almost for all three methods, the ACR for $\sigma = 1.0$ is worse than the one for $\sigma = 0.5$ on ImageNet. For Places365 dataset, the results are slightly better on $\sigma = 1.0$ than $\sigma = 0.5$.

Also, similar as the main paper, we statistically test the variance based on MC sampling and our upper bound tracking method. The results are shown in Table. c. As one can see that when the network gets deeper, the upper bound

tends to be looser. But in most case, the upper bound is around 3 times higher than the sample-based variance, which is affordable in the real-world application.

Table c: Statistics for different layers of MC sampling and our upper bound tracking method for deeper network.

Layer number	1	9	17	25	33
MC (1000 samples)	0.595	0.782	2.751	7.692	0.712
Upper bound	0.685	3.231	5.583	22.546	3.960

References

[1] Muhammad Awais, Fahad Shamshad, and Sung-Ho Bae. Towards an adversarially robust normalization approach. *arXiv preprint arXiv:2006.11007*, 2020.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Convolutional sequence modeling revisited. 2018.

[4] Adel Bibi, Modar Alfadly, and Bernard Ghanem. Analytic expressions for probabilistic moments of pl-dnn with gaussian input. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9099–9107, 2018.

[5] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[6] Jacob Goldberger and Ehud Ben-Reuven. Training deep neural-networks using a noise adaptation layer. 2016.

[7] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[8] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*, pages 8527–8537, 2018.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[10] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313. PMLR, 2018.

[11] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] Joonho Lee, Kumar Shridhar, Hideaki Hayashi, Brian Kenji Iwana, Seokjun Kang, and Seiichi Uchida. Probact: A probabilistic activation function for deep neural networks. *arXiv preprint arXiv:1905.10761*, 2019.

[14] Vishnu Suresh Lokhande, Songwong Tasneeyapant, Abhay Venkatesh, Sathya N Ravi, and Vikas Singh. Generating accurate pseudo-labels in semi-supervised learning and avoiding overconfident predictions via hermite polynomial activations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11435–11443, 2020.

[15] Eran Malach and Shai Shalev-Shwartz. Decoupling" when to update" from" how to update". In *Advances in Neural Information Processing Systems*, pages 960–970, 2017.

[16] Jerzy Neyman and Egon Sharpe Pearson. Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933.

[17] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

[18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[19] Runtian Zhai, Chen Dan, Di He, Huan Zhang, Boqing Gong, Pradeep Ravikumar, Cho-Jui Hsieh, and Liwei Wang. Macer: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*, 2019.

[20] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in neural information processing systems*, pages 8778–8788, 2018.

Table d: Consider a 1-D convolution network with kernel size $k = 3$. The blue dots are the nodes as well as the covariance matrices Σ to be computed. The red arrows are the cross-correlation E between two nodes.

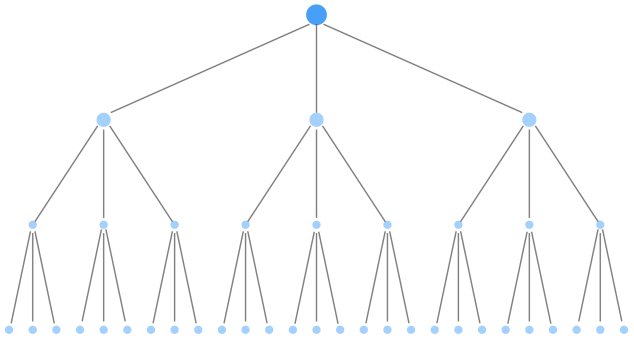
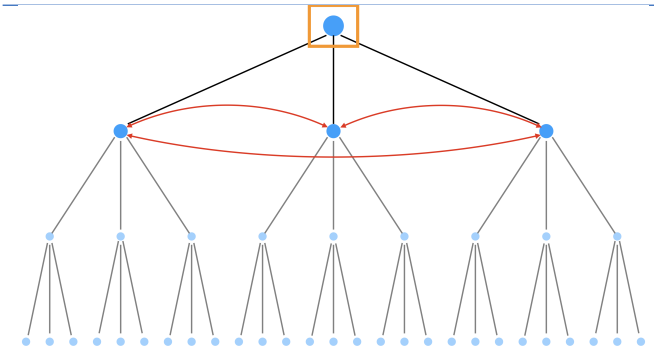
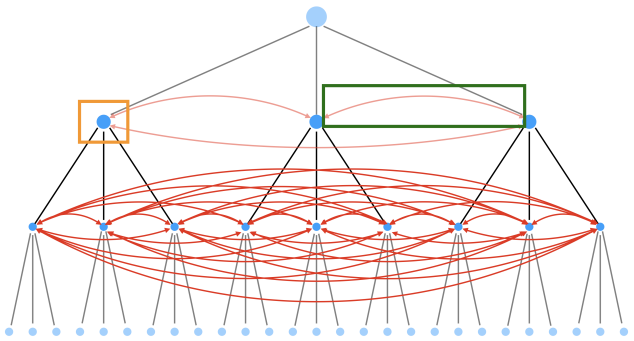
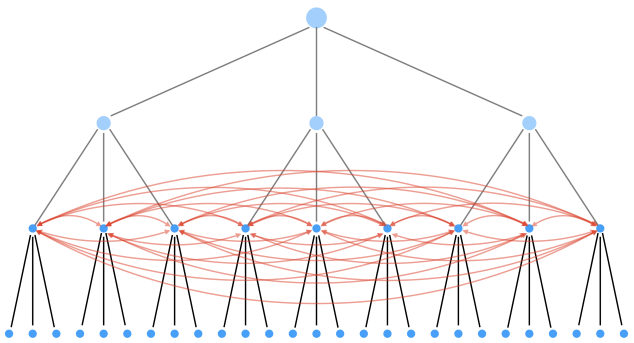
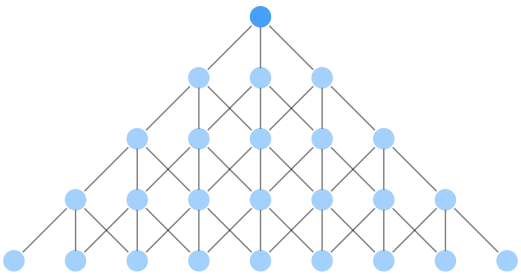
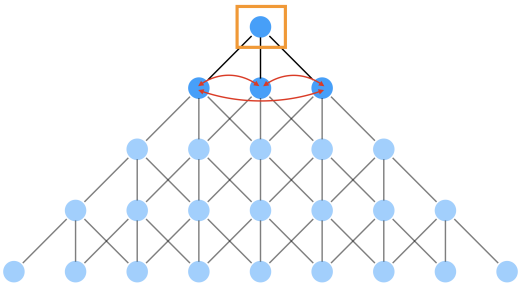
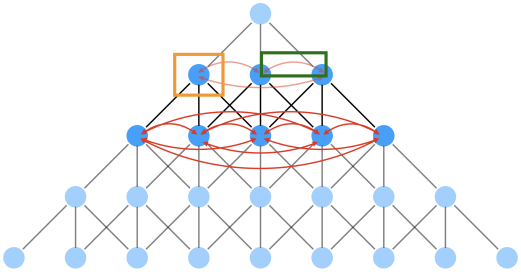
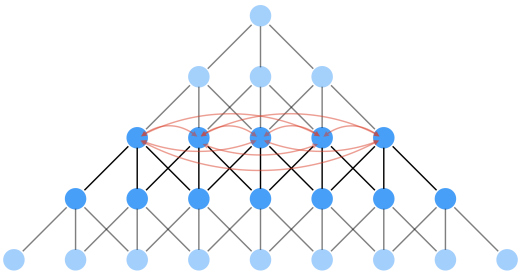
	In the last layer, $p_{c_{\mathbf{x}}} = \underline{p_{c_{\mathbf{x}}}} = \Phi\left(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}}\right)$, We need to compute 1 $\Sigma^{(i)}$
	To compute the $\Sigma^{(i)}$ above (highlighted in orange box), in the second last layer $\Sigma^{(i)} = \mathbb{E}(W^T {X^{(i-1)}}^T X^{(i-1)} W)$ $= \sum_{m=1}^{3} W_m^T \Sigma_m^{(i-1)} W_m + \sum_{m,n=1,m\neq n}^{3} W_m^T E_{mn}^{(i-1)} W_n$ We need to compute 3 $\Sigma^{(i-1)}$ and 3 $E^{(i-1)}$.
	To compute the $\Sigma^{(i-1)}$ above (highlighted in orange box), in the third last layer, $\Sigma^{(i-1)} = \sum_{m=1}^{3} W_m^T \Sigma_m^{(i-2)} W_m + \sum_{m,n=1,m\neq n}^{3} W_m^T E_{mn}^{(i-2)} W_n$ To compute the $E_{12}^{(i-1)}$ above (highlighted in green box) $E_{12}^{(i-1)} = \mathbb{E}({x_1^{(i-1)}}^T x_2^{(i-1)}) = \mathbb{E}(W^T {X_1^{(i-2)}}^T X_2^{(i-2)} W)$ $= \sum_{m,n=3}^{8} \tilde{W}^T E_{mn}^{(i-2)} \tilde{W}'$, where $\tilde{W}, \tilde{W}'$ are the weights depending on the position of m, n. We need to compute 3^2 $\Sigma^{(i-2)}$ and $\frac{1}{2}(3^2+1)3^2$ $E^{(i-2)}$.
	In the $(q+1)^{th}$ last layer, We need to compute k^q $\Sigma^{(i-q)}$ and $\frac{1}{2}(1+k^q)k^q$ $E^{(i-q)}$.

Table e: Consider a 1-D convolution network with kernel size $k = 3$ with overlapping. The blue dots are the nodes as well as the covariance matrices Σ to be computed. The red arrows are the cross-correlation E between two nodes.

	In the last layer, $p_{c_{\mathbf{x}}} = \underline{p_{c_{\mathbf{x}}}} = \Phi\left(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}}\right)$, We need to compute 1 $\Sigma^{(i)}$
	To compute the $\Sigma^{(i)}$ above (highlighted in orange box), in the second last layer $\Sigma^{(i)} = \mathbb{E}(W^T {X^{(i-1)}}^T X^{(i-1)} W)$ $= \sum_{m=1}^{3} W_m^T \Sigma_m^{(i-1)} W_m + \sum_{m,n=1,m\neq n}^{3} W_m^T E_{mn}^{(i-1)} W_n$ We need to compute 3 $\Sigma^{(i-1)}$ and 3 $E^{(i-1)}$.
	To compute the $\Sigma^{(i-1)}$ above (highlighted in orange box), in the third last layer, $\Sigma^{(i-1)} = \sum_{m=1}^{3} W_m^T \Sigma_m^{(i-2)} W_m + \sum_{m,n=1,m\neq n}^{3} W_m^T E_{mn}^{(i-2)} W_n$ To compute the $E_{12}^{(i-1)}$ above (highlighted in green box) $E_{12}^{(i-1)} = \mathbb{E}({x_1^{(i-1)}}^T x_2^{(i-1)}) = \mathbb{E}(W^T {X_1^{(i-2)}}^T X_2^{(i-2)} W)$ $= \sum_{m,n=1}^{4} \tilde{W}^T E_{mn}^{(i-2)} \tilde{W}'$, where $\tilde{W}, \tilde{W}'$ are the weights depending on the position of m, n. We need to compute $(3-1)\times 2+1$ $\Sigma^{(i-2)}$ and $\frac{1}{2}((3-1)\times 2+1+1)((3-1)\times 2+1)$ $E^{(i-2)}$.
	In the $(q+1)^{th}$ last layer, We need to compute $(k-1)q+1$ $\Sigma^{(i-q)}$ and $\frac{1}{2}((k-1)q+1+1)((k-1)q+1)$ $E^{(i-q)}$.