Batch normalization : Accelerating deep network training by reducing internal covariate shift

Loffe, Sergey, and Christian Szegedy

경상국립대학교 석사과정 오서영

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change.
-> This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities
→ **Internal covariate shift**

We address the problem by normalizing layer inputs for each training mini-batch

Introduction

SGD optimizes the parameters Θ of the network, so as to minimize the loss :

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

where $x_{1\ldots N}$ is the training data set.
With SGD, the training proceeds in steps,
at each step considering a minibatch $x_{1\ldots m}$of size m

Introduction

1. The gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m}\frac{\partial \ell(\mathrm{x}_i,\Theta)}{\partial \Theta}$

is an estimate of the gradient over the training set, whose quality improves as the batch size increases.
2. Computation over a mini-batch can be more efficient than m computations for individual examples on modern computing platforms

→ it requires careful tuning of the model hyper-parameters, specifically the learning rate and the initial parameter values.

Introduction

Consider a layer with a sigmoid activation function z = g(Wu + b),
where **u** is the layer input, the weight matrix **W** and bias vector **b** are the layer parameters to be learned, and g(x) = $\frac{1}{1+\exp(-x)}$.

As |x| increases, g'(x) tends to zero.
-> For all dimensions of x = Wu + b except those with small absolute values, the gradient flowing down to u will vanish and the model will train slowly.

-> However, since x is affected by W, b and the parameters of all the layers below, changes to those parameters during training will likely move many dimensions of x into the saturated regime of the nonlinearity and slow down the convergence.

- We could ensure that the distribution of nonlinearity inputs remains more stable as the network trains, then the optimizer would be less likely to get stuck in the saturated regime, and the training would accelerate.

Internal Covariate Shift

Internal Covariate Shift
: the change in the distributions of internal nodes of a deep network, in the course of training

- Eliminating it offers a promise of faster training
- **Batch Normalization** takes a step towards reducing internal covariate shift

1. Accelerating the training of deep neural nets
2. Reducing the dependence of gradients on the scale of the parameters or of their initial values -> allowing us to use much higher learning rates
3. regularizing the model and reduces the need for Dropout
4. using saturating nonlinearities by preventing the network from getting stuck in the saturated modes.

Normalization via Mini-Batch Statistics

1. Normalize each scalar feature independently, by making it have zero mean and unit variance.
For a layer with d-dimensional input x = $(x_1,\ldots, x_d)$, we will normalize each dimension where the expectation and variance are computed over the training data set

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Simply normalizing each input of a layer may change what the layer can represent.
(EX) Normalizing the inputs of a sigmoid would constrain them
to the linear regime of the nonlinearity.
-> make sure that the transformation inserted in the network can represent the identity transform.
-> For each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}$, $\beta^{(k)}$,
which scale and shift the normalized value :

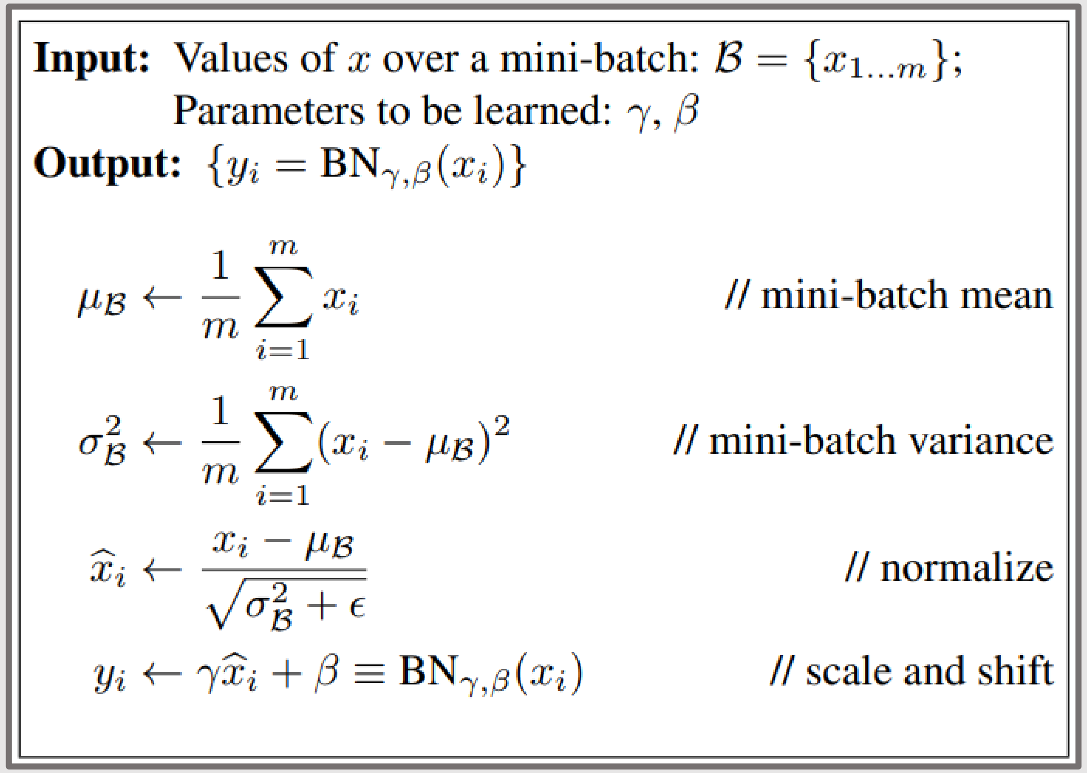
$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

→ we could recover the original activations, if that were the optimal thing to do

Normalization via Mini-Batch Statistics

2. We use mini-batches in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation

Batch Normalizing Transform, applied to activation x over a mini-batch.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Training and Inference with Batch-Normalized Networks

- To Batch-Normalize a network, we specify a subset of activations and insert the BN transform for each of them

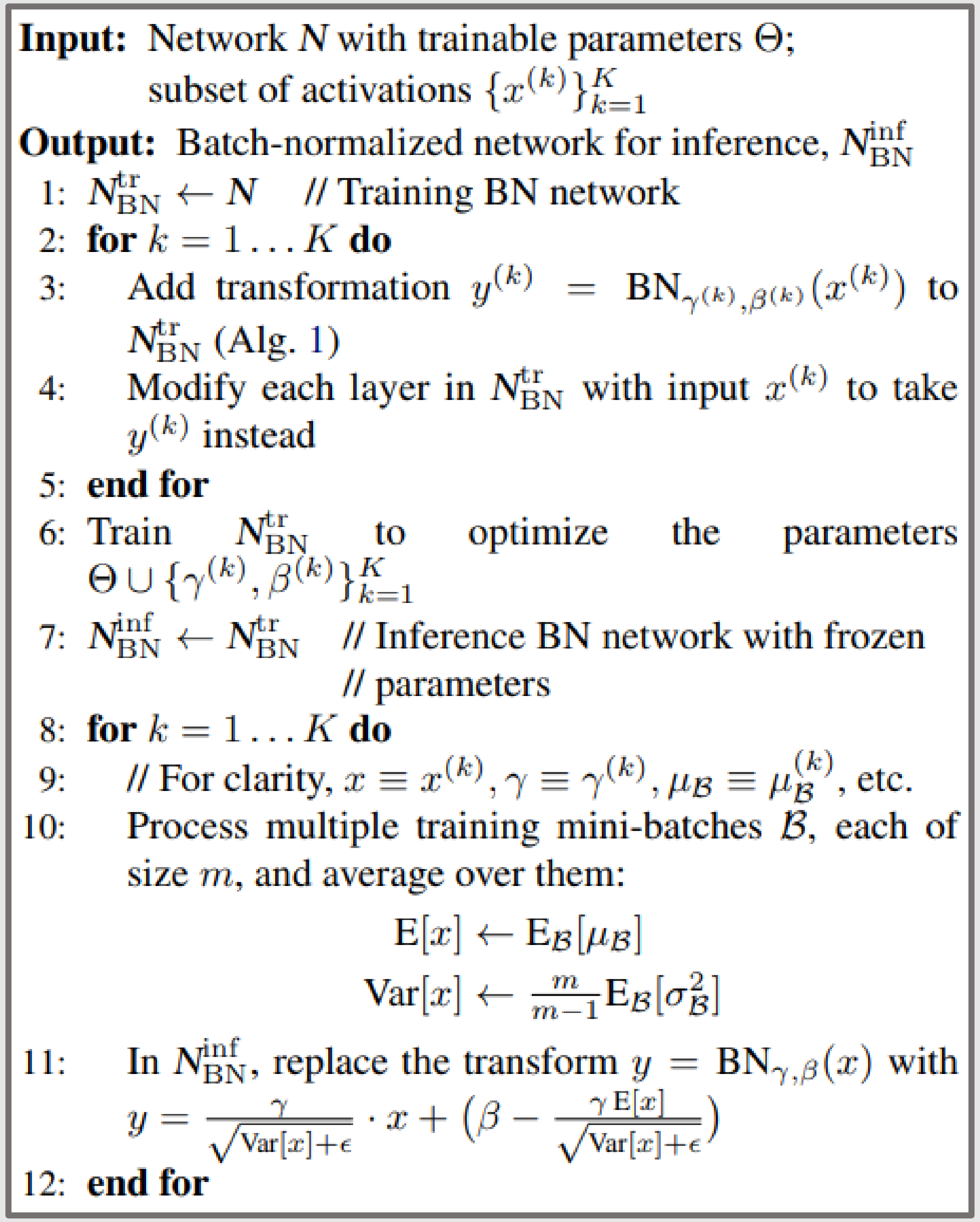
- Any layer that previously received x as the input, now receives BN(x).
A model employing Batch Normalization can be trained using batch gradient descent, or Stochastic Gradient Descent with a mini-batch size $m > 1$.

- The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference
we want the output to depend only on the input, deterministically.
For this, once the network has been trained, we use the normalization

→ Since the means and variances are fixed during inference, the normalization is simply a linear transform applied to each activation

Training and Inference with Batch-Normalized Networks

Training a Batch-Normalized Network



```
Input: Network $N$ with trainable parameters $\Theta$;
       subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$
 1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$    // Training BN network
 2: for $k = 1 \dots K$ do
 3:     Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to
        $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
 4:     Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take
        $y^{(k)}$ instead
 5: end for
 6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters
    $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
 7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$    // Inference BN network with frozen
                     // parameters
 8: for $k = 1 \dots K$ do
 9:     // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10:     Process multiple training mini-batches $\mathcal{B}$, each of
        size $m$, and average over them:
            $\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$
            $\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$
11:     In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with
        $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: end for
```

Batch-Normalized Convolutional Networks

$$z = g(Wu + b)$$

where W and b are learned parameters of the model, and
g(·) is the nonlinearity such as sigmoid or ReLU.

- This formulation covers both fully-connected and convolutional layers.
- We add the BN transform immediately before the nonlinearity, by normalizing x = Wu + b.

Batch Normalization enables higher learning rates

In traditional deep networks, too high a learning rate may result in the gradients that explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issue
- This enables the sigmoid nonlinearities to more easily stay in their non-saturated regime

- Batch Normalization also makes training more resilient to the parameter scale.
- Normally, large learning rates may increase the scale of layer parameters, which then amplify the gradient during backpropagation and lead to the model explosion.
- However, with Batch Normalization, backpropagation through a layer is unaffected by the scale of its parameter

$$\mathrm{BN}(W\mathrm{u}) = \mathrm{BN}((aW)\mathrm{u})$$

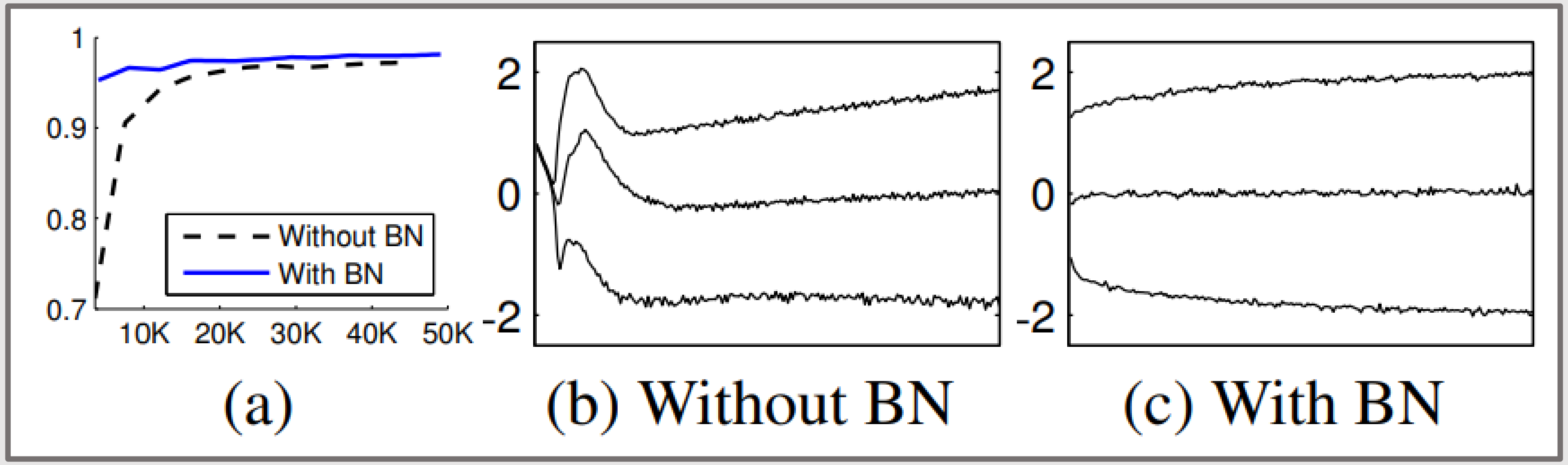
$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial \mathrm{u}} = \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial \mathrm{u}}$$

$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial W}$$

→ so larger weights lead to smaller gradients, and Batch Normalization will stabilize the parameter growth

Experiment

Activations over time



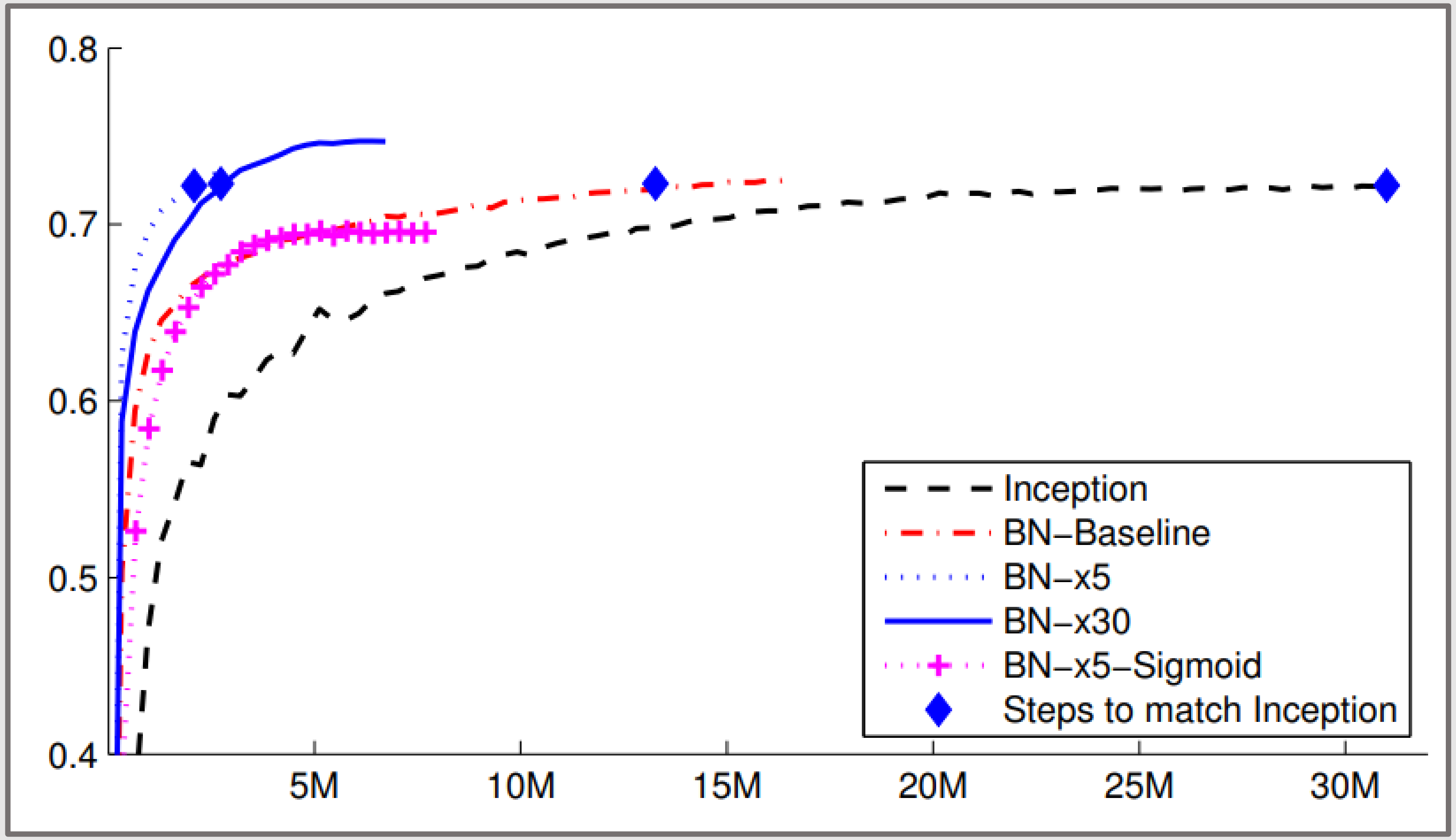
(a) (b) Without BN (c) With BN

Test acc of the MNIST **The evolution of input distribution**

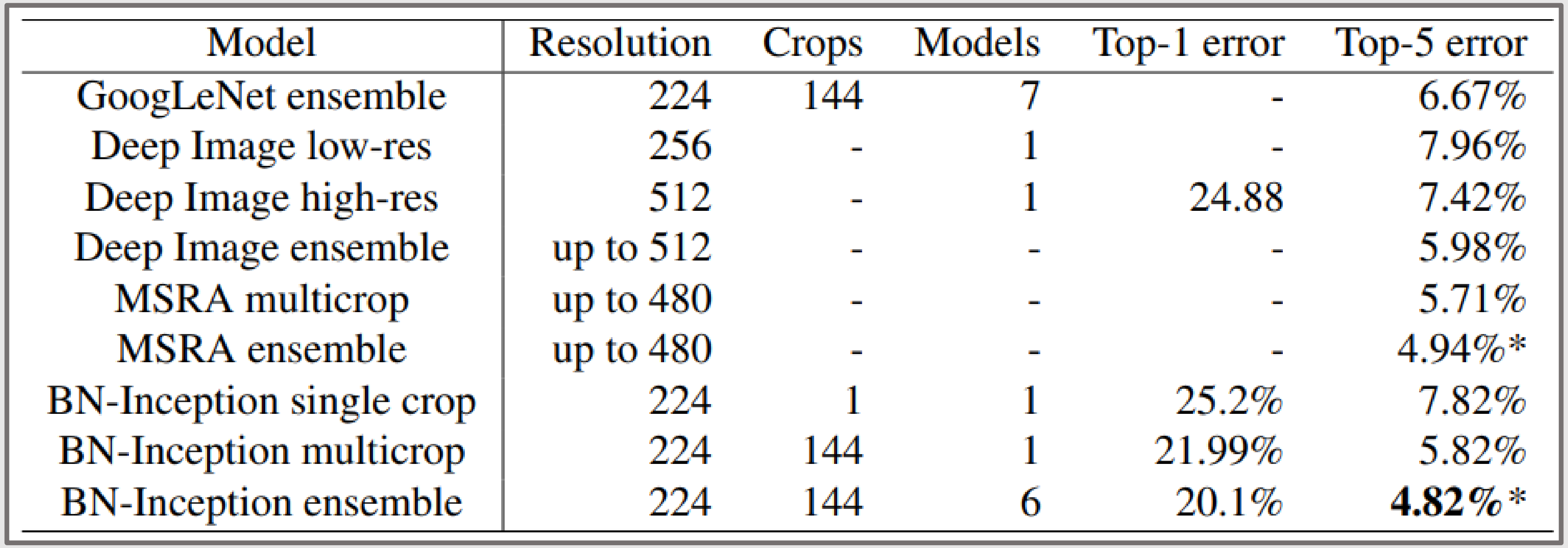
Batch Normalization makes the distribution more stable
and reduces the internal covariate shift

Experiment

ImageNet classification



Experiment



Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	up to 512	-	-	-	5.98%
MSRA multicrop	up to 480	-	-	-	5.71%
MSRA ensemble	up to 480	-	-	-	4.94%*
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	**4.82%***