Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks

Guangyong Chen, Pengfei Chen, Yujun Shi, Chang-Yu Hsieh, Benben Liao, Shengyu Zhang

presenter:Yaorong Xiao

Motivation

- - DNNs becomes deeper and larger
- - Whitening helps convergence -->batch norm(more efficient)
- - Independence among neurons can increase the representations of model.
- - Traditional ICA methods computational expensive.
- - location of batch norm

Uncorrelated Inputs

$$\min_A \sum_{i=1}^{n} \|y_i - Ax_i\|_2^2, \tag{1}$$

then based on the gradient method, the optimal solution of A must satisfy

$$A \sum_{i=1}^{n} x_i x_i^T = \sum_{i=1}^{n} y x_i^T. \tag{2}$$

$$A \cdot \mathrm{Cov}(x, x) = \mathrm{Cov}(y, x)$$

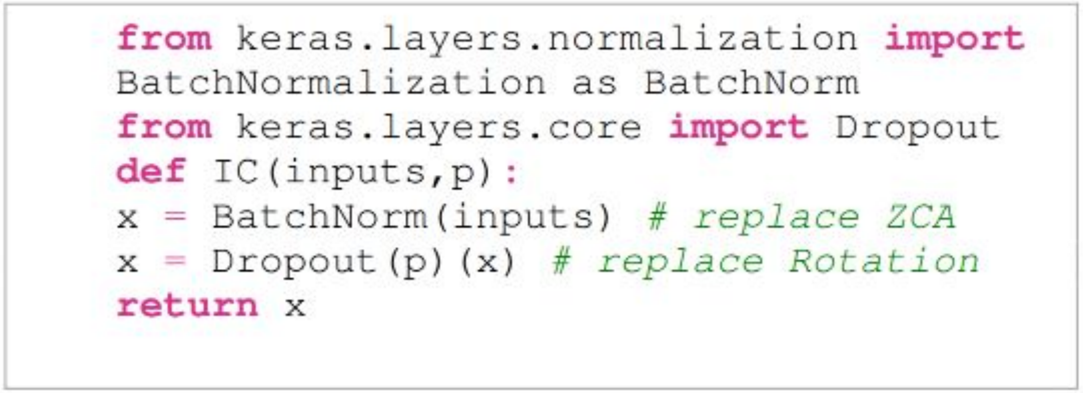
Goal & Contribution

Goal: Create a layer that encourages independent neuron activations

Contributions:

- - Propose IC Layer = BatchNorm + Dropout
- - Theoretical proof of reduced mutual info and correlation
- - Improved empirical results on CIFAR10/100 and ILSVRC2012

IC Layer Design



```
from keras.layers.normalization import
BatchNormalization as BatchNorm
from keras.layers.core import Dropout
def IC(inputs,p):
x = BatchNorm(inputs) # replace ZCA
x = Dropout(p)(x) # replace Rotation
return x
```

Figure 2. Python code of the IC layer based on Keras

ICA: PCA+rotation

Mutual Information

$$I(x;y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}.$$

Theorem:

Let g_i denote a family of independent random variables generated from the Bernoulli distribution with its mean being p. Let $\hat{x}_i = g_i x_i$. Then we have

$$I(\hat{x}_i; \hat{x}_j) = p^2 I(x_i, x_j), \quad \forall i \neq j,$$

$$H(\hat{x}_i) = pH(x_i) + \epsilon_p,$$

where H denotes the Shannon entropy and ϵ_p the entropy of the Bernoulli distribution.

$$I(\hat{x}_i;\hat{x}_j) = \sum_{a_1} + \sum_{a_2} + \sum_{a_1,a_2} .$$

The first part concerns with computations when $\hat{x}_j$ are restricted to zero

$$\sum_{a_1} = \sum_{a_1} P(\hat{x}_i = a_1, \hat{x}_j = 0) \log \frac{P(\hat{x}_i = a_1, \hat{x}_j = 0)}{P(\hat{x}_i = a_1)P(\hat{x}_j = 0)},$$

$$P(\hat{x}_i = a_1, \hat{x}_j = 0) = P(\hat{x}_i = a_1, g_j = 0)$$

$$= P(\hat{x}_i = a_1)P(g_j = 0) = P(\hat{x}_i = a_1)P(\hat{x}_j = 0).$$

=0

$$I(\hat{x}_i;\hat{x}_j) = \sum_{a_1} + \sum_{a_2} + \sum_{a_1,a_2}.$$

$$\sum_{a_1,a_2} = \sum_{a_1,a_2} P(\hat{x}_i = a_1, \hat{x}_j = a_2) \log \frac{P(\hat{x}_i = a_1, \hat{x}_j = a_2)}{P(\hat{x}_i = a_1)P(\hat{x}_j = a_2)}$$

$$P(\hat{x}_i = a_1, \hat{x}_j = a_2) = P(x_i = a_1, x_j = a_2, g_i = g_j = 1)$$

$$= p^2 P(x_i = a_1, x_j = a_2).$$

$$P(\hat{x}_i = a_1) = pP(x_i = a_1),$$

$$\sum_{a_1,a_2} = p^2 I(x_i;x_j).$$

Entropy of Neuro

$$H(\hat{x}_i) = -\sum_a P(\hat{x}_i = a) \log P(\hat{x}_i = a)$$

$$H(\hat{x}_i) = -P(\hat{x}_i = 0) \log P(\hat{x}_i = 0) + \sum_{a \neq 0} .$$

◆ **For $a = 0$:**

Since $\hat{x}_i = 0$ happens when $g_i = 0$:

$$P(\hat{x}_i = 0) = P(g_i = 0) = 1 - p$$

So:

$$-P(\hat{x}_i = 0) \log P(\hat{x}_i = 0) = -(1 - p) \log(1 - p)$$

$$H(\hat{x}_i) = -\sum P(\hat{x}_i = a) \log P(\hat{x}_i = a)$$

$$H(\hat{x}_i) = -P(\hat{x}_i = 0) \log P(\hat{x}_i = 0) + \sum_{a \neq 0} .$$

- **For** $a \neq 0$:

$$P(\hat{x}_i = a) = P(x_i = a, g_i = 1) = p \cdot P(x_i = a)$$

$$= -\sum_{a \neq 0} pP(x_i = a) \log pP(x_i = a)$$

$$\overset{\cdot}{=} -\sum_{a \neq 0} pP(x_i = a) \log p - p \sum_{a \neq 0} P(x_i = a) \log P(x_i = a)$$

$$= -p \log p + pH(x_i).$$

$$H(\hat{x}_i) = -\sum P(\hat{x}_i = a) \log P(\hat{x}_i = a)$$

$$H(\hat{x}_i) = -P(\hat{x}_i = 0) \log P(\hat{x}_i = 0) + \sum_{a \neq 0} .$$

$$H(\hat{x}_i) = pH(x_i) + \epsilon_p,$$

where ϵ_p is the entropy of the Bernoulli g_i.

Correlation Coefficient

$$\hat{c}_{ij} = \frac{1}{\sigma_i \sigma_j} \mathbb{E}(g_i x_i g_j x_j), \qquad \sigma_i^2 = \mathbb{E}[g_i^2 x_i^2] = \mathbb{E}[g_i^2]\mathbb{E}[x_i^2] = p$$

$$\mathbb{E}[g_i g_j x_i x_j] = p^2 \mathbb{E}[x_i x_j]$$

$$\hat{c}_{ij} = \frac{1}{\sqrt{p}\sqrt{p}} \cdot p^2 \cdot \mathbb{E}[x_i x_j] = p \cdot c_{ij}$$

Then the correlation also reduced

Experiment Setup

- **Benchmarks**:

  CIFAR10, CIFAR100, ILSVRC2012

- **Architectures**:

  Based on **ResNet** designs from [He et al., 2016a/b] and [Li et al., 2018]

- **Parameters**:

  - **IC Layer** includes learnable **scale & shift** parameters (similar to BatchNorm)
  - Total number of learnable parameters is matched to the baseline ResNet for fair comparison

Experiments

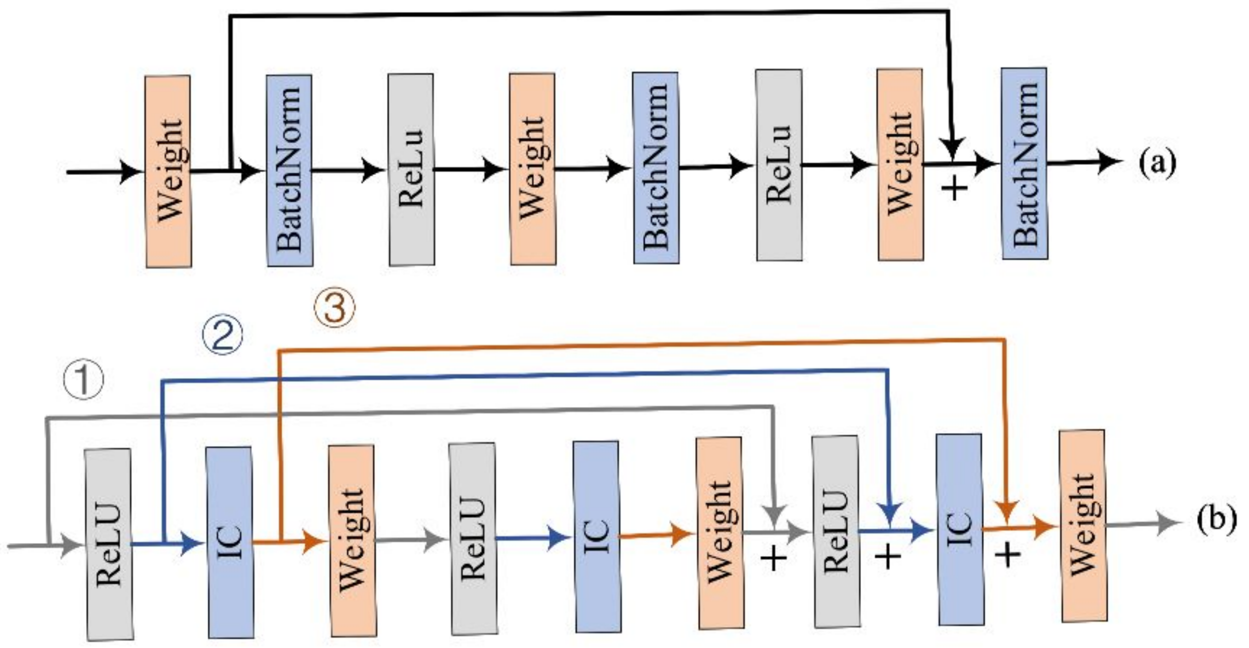
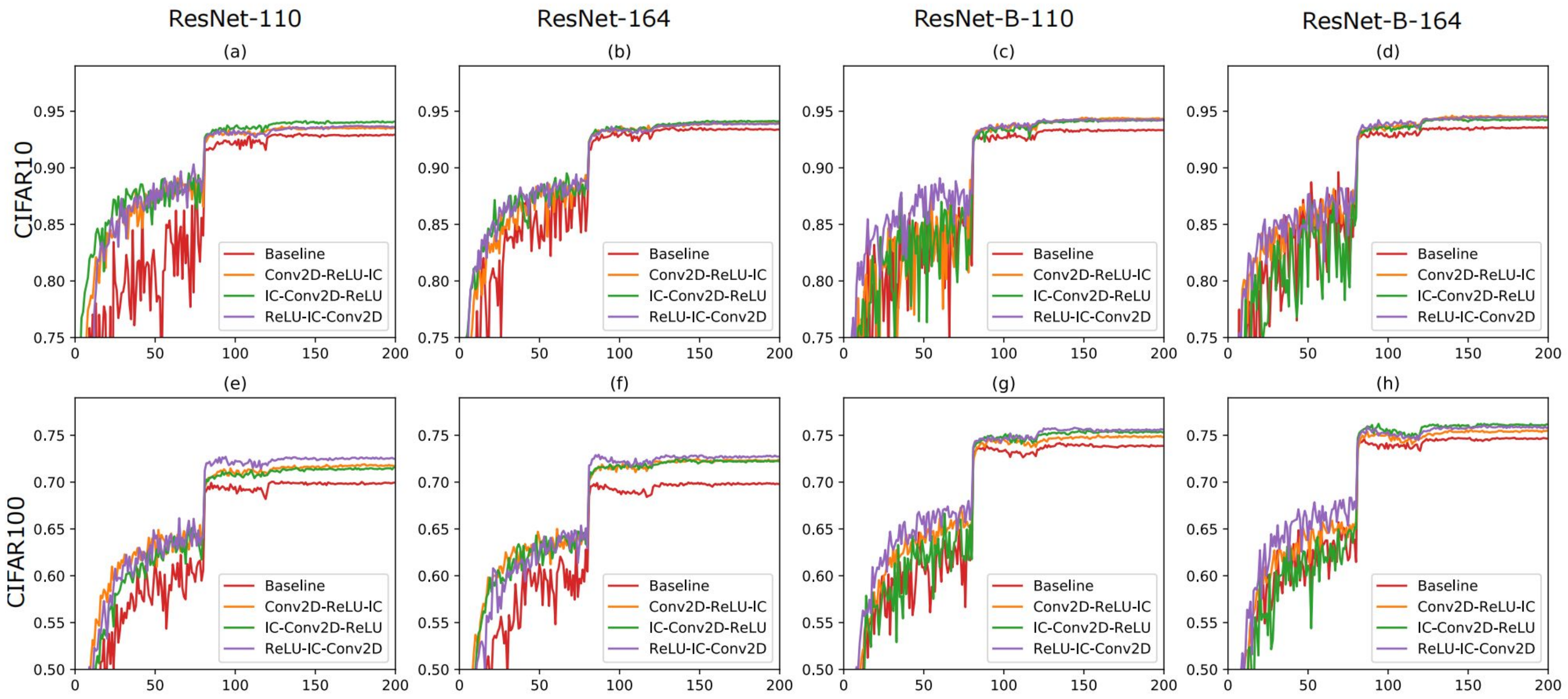


Figure 3. (a) The classical ResNet architecture, where $'+'$ denotes summation. (b) Three proposed ResNet architectures reformulated with the IC layer.

Model	Depth	Layers in Residual Unit	CIFAR10	CIFAR100
ResNet	110	①. 2×{ReLU-IC-Conv2D}	0.9361	0.725
		②. 2×{IC-Conv2D-ReLU}	0.9352	0.7165
		③. 2×{Conv2D-ReLU-IC}	0.9408	0.7174
		Baseline	0.9292	0.6893
	164	①. 2×{ReLU-IC-Conv2D}	0.9395	0.7224
		②. 2×{IC-Conv2D-ReLU}	0.9366	0.7273
		③. 2×{Conv2D-ReLU-IC}	0.9411	0.7237
		Baseline	0.9339	0.6981
ResNet-B	110	①. 3×{ReLU-IC-Conv2D}	0.9448	0.7563
		②. 3×{IC-Conv2D-ReLU}	0.9425	0.7532
		③. 3×{Conv2D-ReLU-IC}	0.9433	0.7482
		Baseline	0.9333	0.7387
	164	①. 3×{ReLU-IC-Conv2D}	0.9445	0.758
		②. 3× {IC-Conv2D-ReLU}	0.9424	0.7616
		③. 3×{Conv2D-ReLU-IC}	0.9453	0.7548
		Baseline	0.9355	0.7465



Key Findings

- `ReLU−IC−Conv2D` **residual unit** consistently:
    - Achieves **more stable training**
    - Yields **faster convergence**
    - Reaches a **better convergence limit**
- IC layer improves ResNet/ResNet-B especially on **CIFAR100**, indicating stronger generalization.

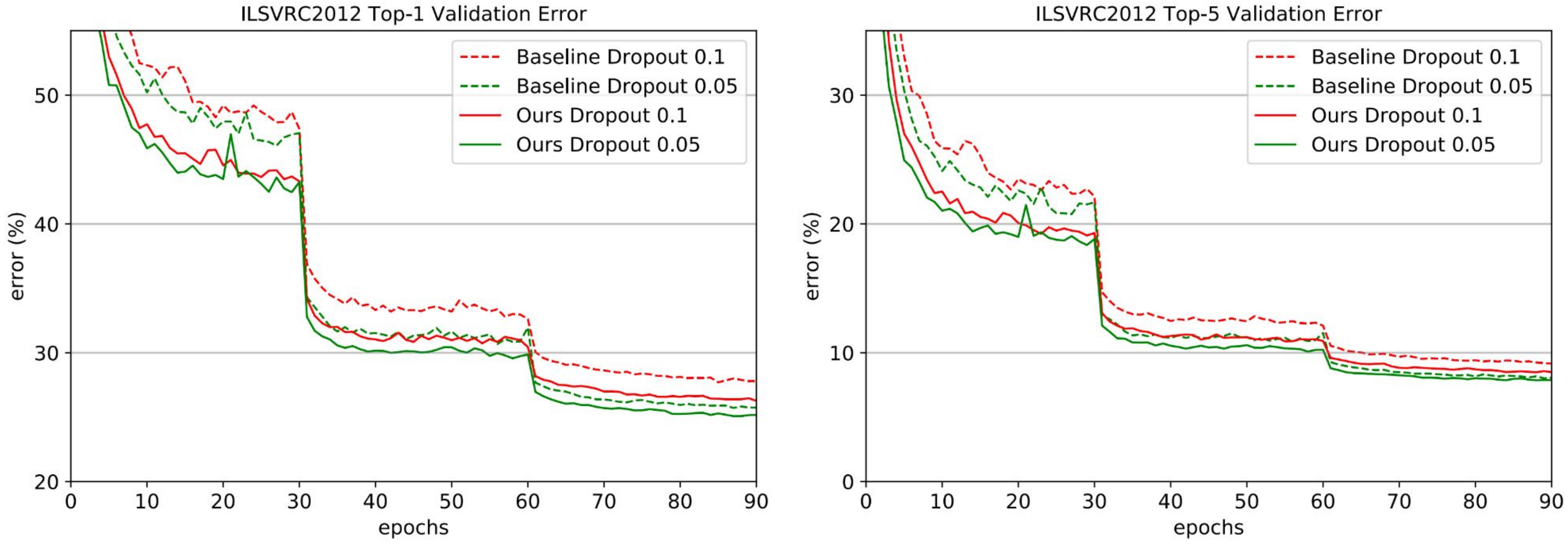


Figure 5. Top-1 and Top-5 (1-crop testing) error on ImageNet validation.

- **Benchmark**:

  ILSVRC2012 (ImageNet)

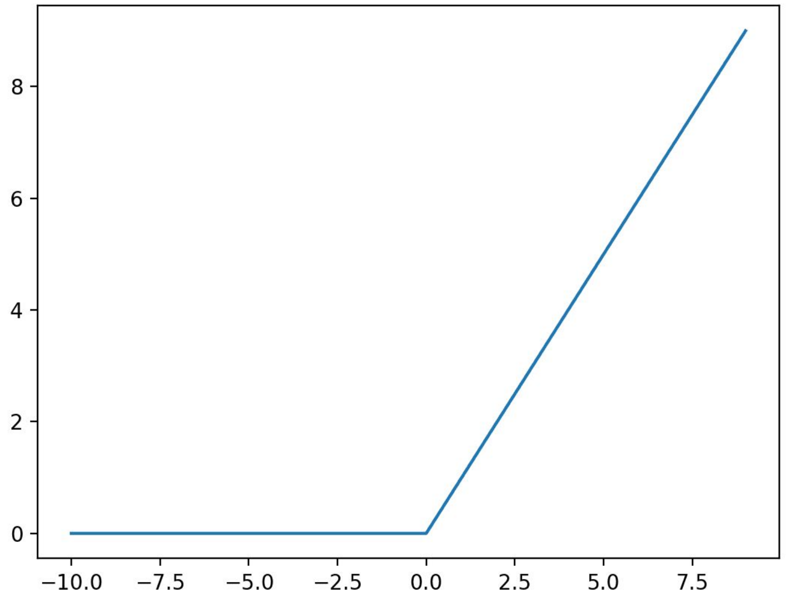
  - 1,000 classes
  - 1.28M training images
  - 50k validation images

BatchNorm Placement Rethink

Set ReLU output as x,

$$y = Wx$$

$$\partial_{w_j} l = \partial_{y_j} l \cdot x^T$$



- ReLU outputs non-negative values → all x components ≥ 0
- Weight updates $\partial_{w_j} l$ are **constrained**:
  - Updates must be **all positive or all negative**, depending on $\partial_{y_j} l$
- Thus, **weights can't move in true gradient direction** → optimization zigzags

- $L(Wx)$ expands as:

$$\sum a_{jk} W_{jk}^2 \quad (\text{with } a_{jk} > 0)$$

- Gradient $\nabla_{W_{jk}} L \propto W_{jk}$

  ⇒ direction aligns only if all w_j lie in **same orthant** (e.g., all positive)

- That's only $1/2^{n-1}$ of the whole space!

  ⇒ SGD follows **zigzag paths** instead of true gradient

Trade-off

- Pros:
- - Better independence
- - Faster convergence

- Cons:
- - Lower expected activations (px)
- - May reduce information throughput

Q&A

- Thank you!
- Happy to answer questions or discuss ideas.