# Big Transfer (BiT): General Visual Representation Learning

Dmitry Medvedev

Moscow State University

# BiT: General Idea

[Kolesnikov et al., 2020]

**Big Model & Big Data**

**Fine-tuning for visual task of any size**

# Architectures

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | $112\times112$ | $7\times7$, 64, stride 2 | | | | |
| conv2_x | $56\times56$ | $3\times3$ max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | $28\times28$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | $14\times14$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | $7\times7$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | $1\times1$ | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

# Upstream Pre-Training: Data

| **Модель** | **Данные** | **Объем** | **Число классов** | **Label per Image** |
|---|---|---|---|---|
| BiT-L | JFT-300M | 300 M | 19 K | ~1.26 |
| BiT-M | ImageNet-21k | 14.2 M | 21 K | $\geq 1$ |
| BiT-S | ILSVRC-2012 | 1.28 M | 1 K | 1 |

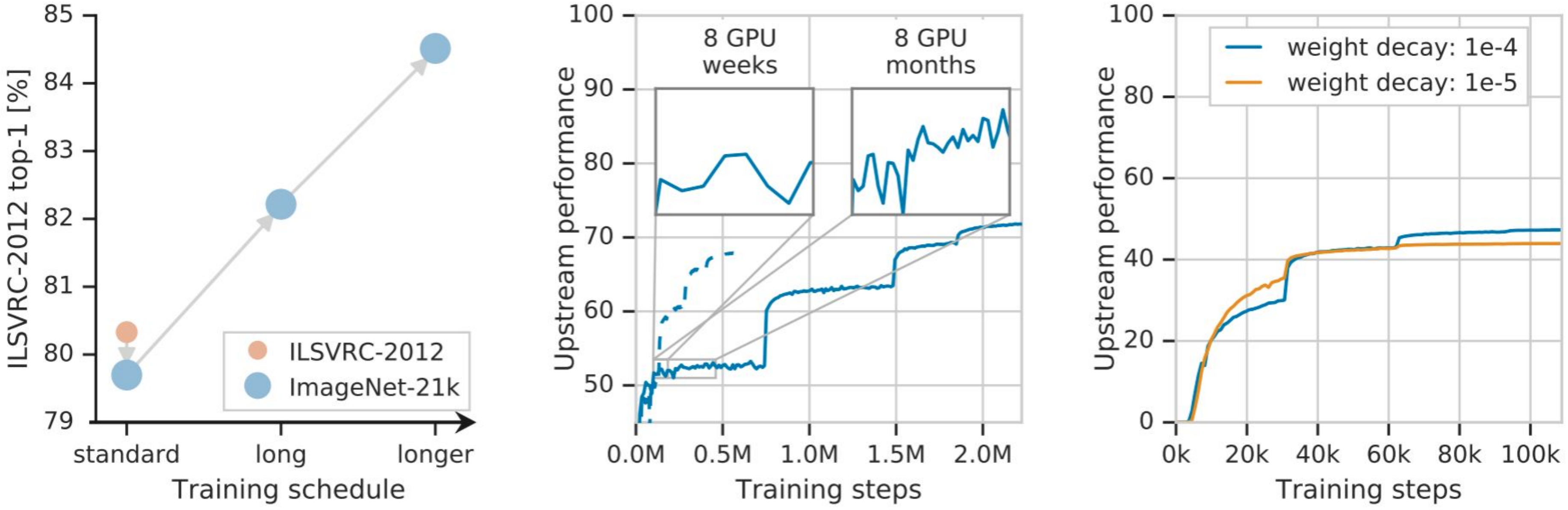
# Upstream Pre-Training

- Scale
- Batch Normalization -> Group Normalization + Weight Standardization
- SGD + momentum
- Аугментации: Crop + Horizontal Flip

| **Model** | **Num. Epochs** | **lr decay by 10 after epochs** |
|---|---|---|
| -S & -M | 90 | 30, 60, 80 |
| -L | 40 | 23, 30, 37 |

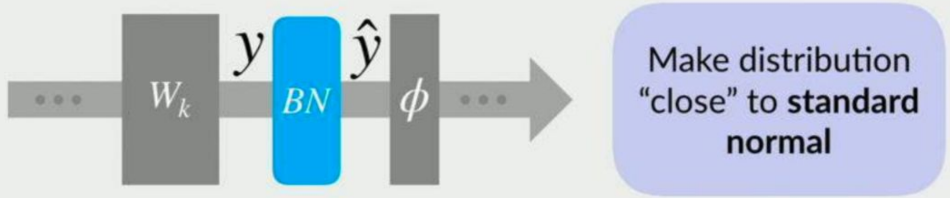
| **Image Size** | **Warm-up steps** | **Weight decay** | **Batch Size** | **Images per chip** |
|---|---|---|---|---|
| 224 x 224 | 5000 | 1e-4 | 4096 | 8 |

# Upstream Pre-Training



# Batch Normalization

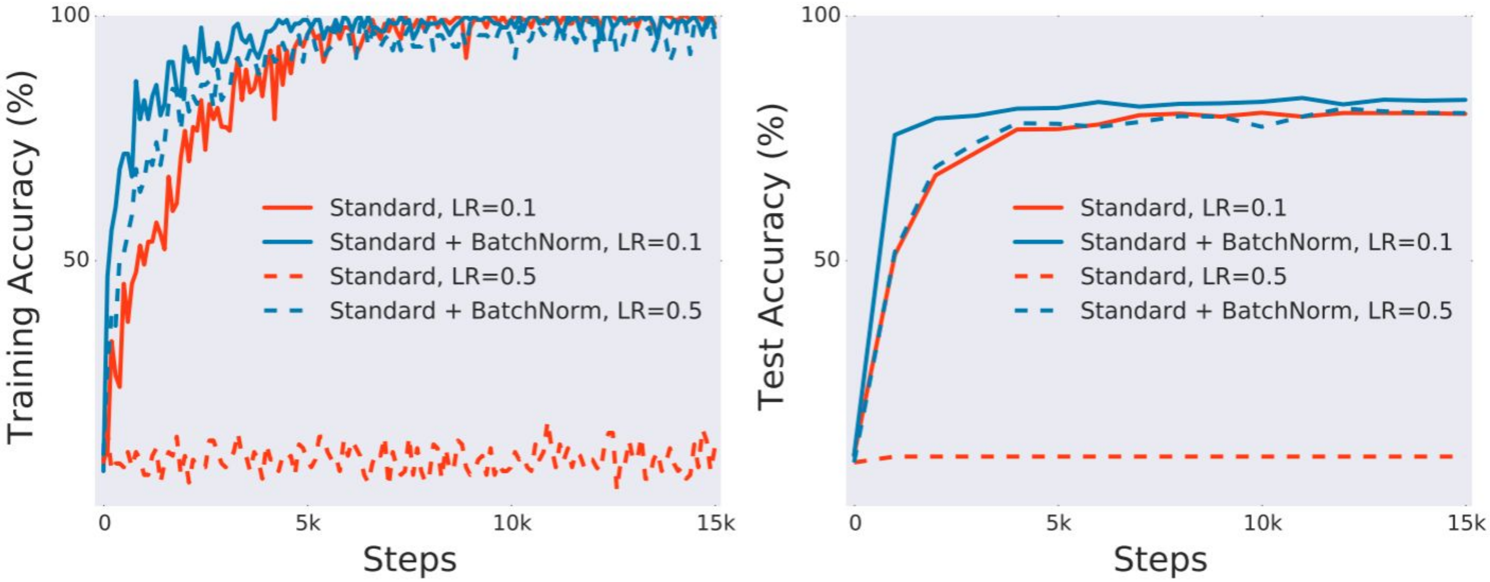
[Santurkar et al. 2020]



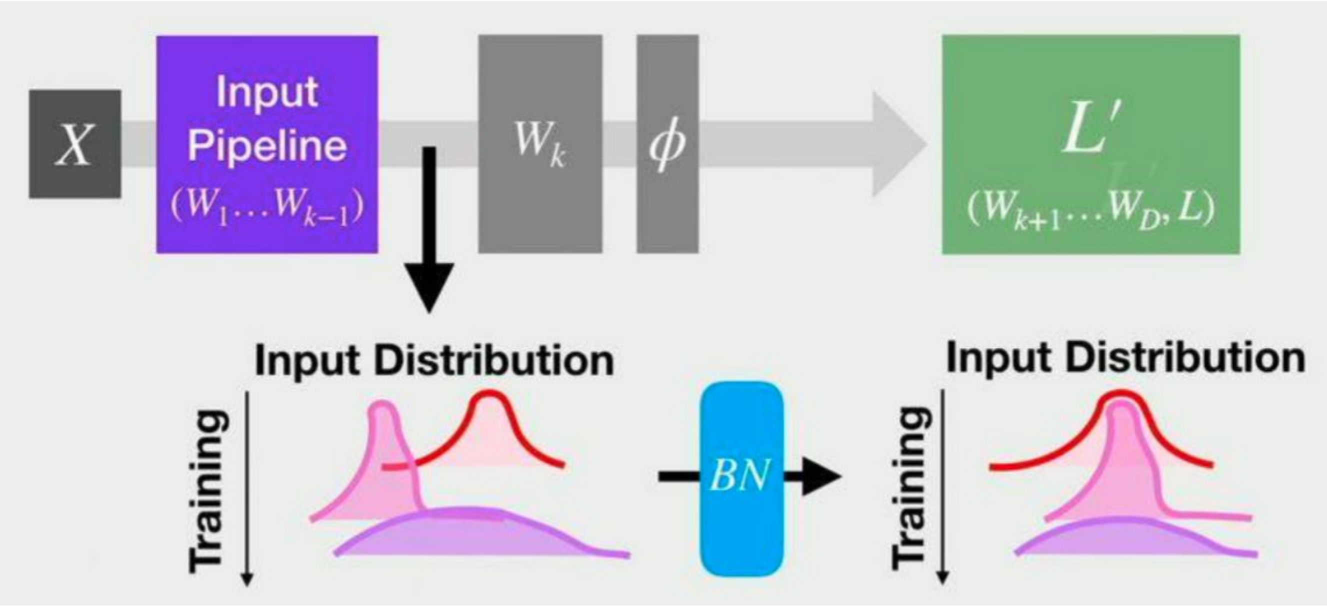

$$\hat{y} = \frac{y - \mu}{\sigma + \epsilon} \quad ,$$

where $\quad \mu = \mathbb{E}[y] \qquad \sigma^2 = Var(y)$

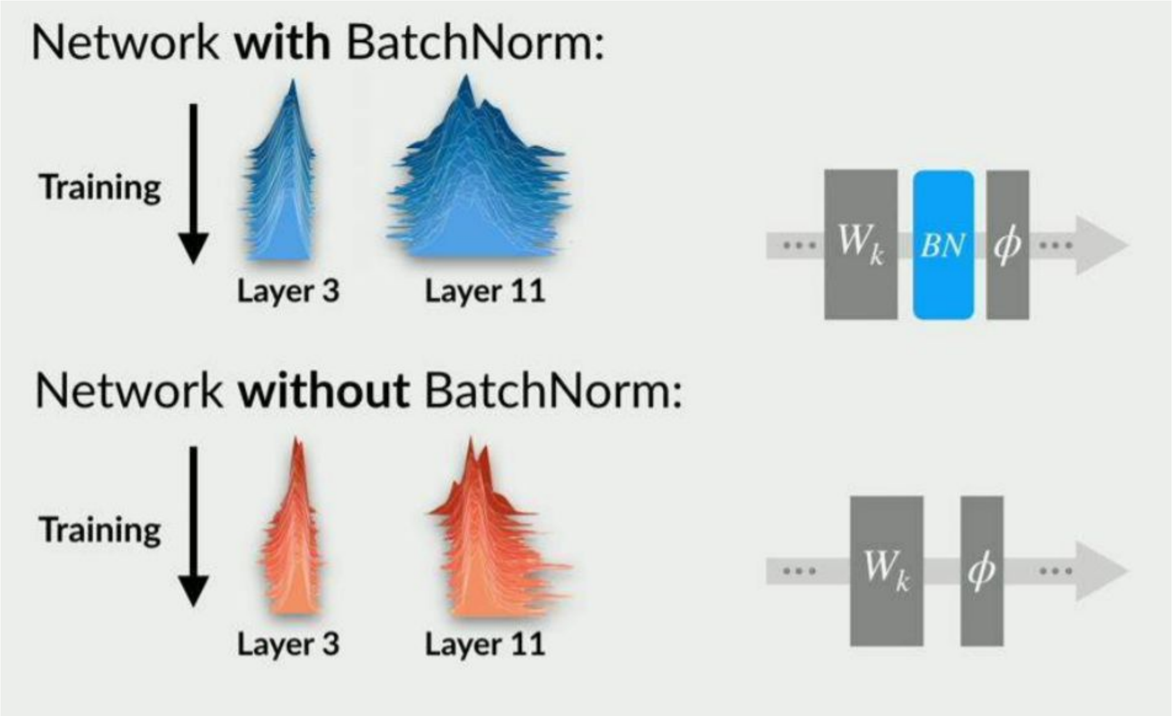
# Batch Normalization



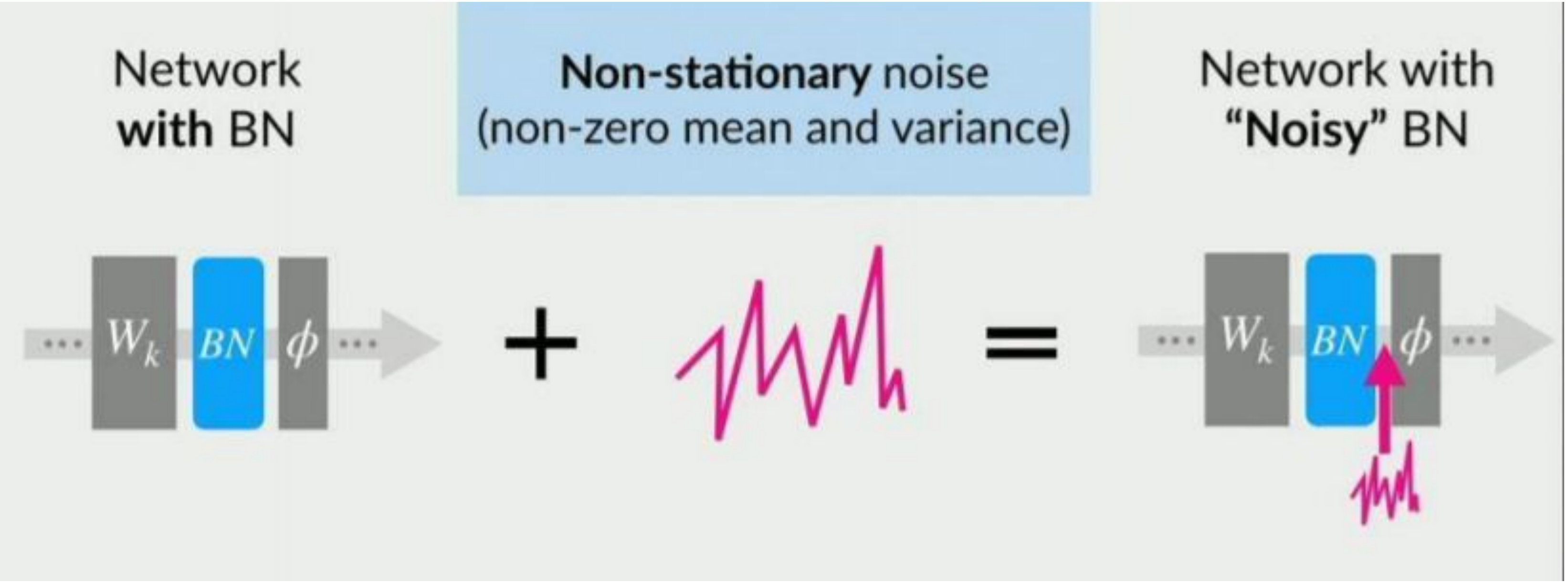
# The internal Covariate Shift Hypothesis



# Experiment: Input Distribution



# Experiment: Noisy Batch Normalization



# Experiment: Noisy Batch Normalization

**Algorithm 1** "Noisy" BatchNorm

% **For constants** $n_m, n_v, r_m, r_v$.

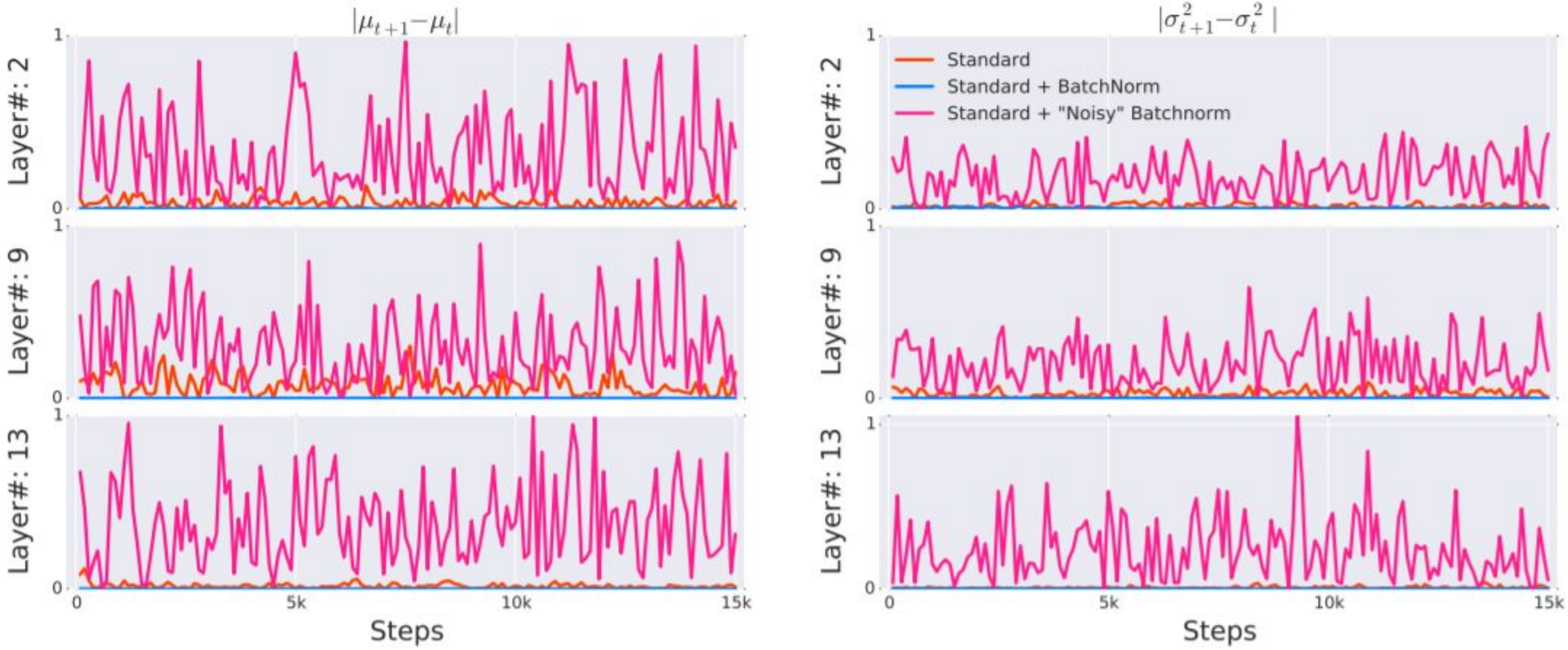
**for** each layer at time $t$ **do**
  $a^t_{i,j} \leftarrow$ *Batch-normalized activation for unit j and sample i*

  **for** each $j$ **do** ▷ Sample the parameters $(m^t_j, v^t_j)$ of $D^t_j$ from $D_j$
    $\mu^t \sim U(-n_\mu, n_\mu)$
    $\sigma^t \sim U(1, n_\sigma)$

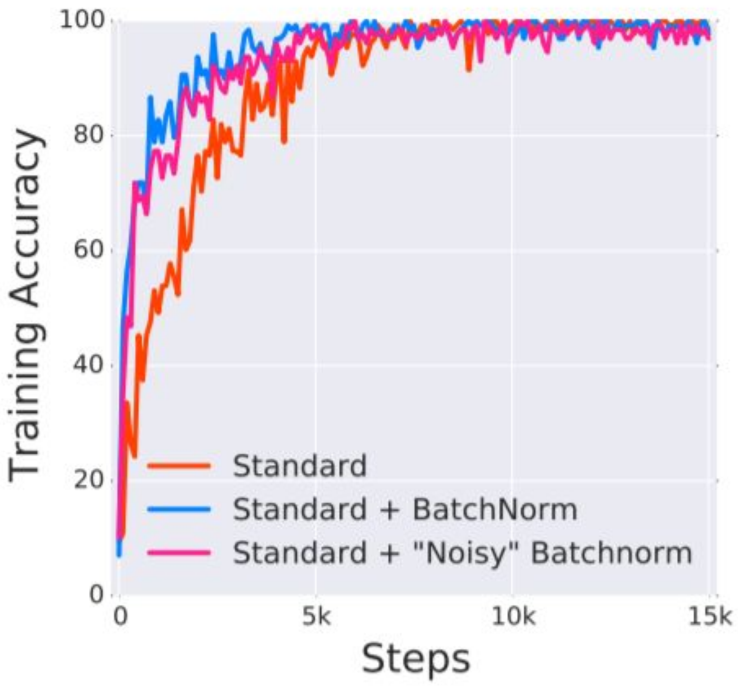
  **for** each $i$ **do** ▷ Sample noise from $D^t_j$
    **for** each $j$ **do**
      $m^t_{i,j} \sim U(\mu - r_\mu, \mu + r_\mu)$
      $s^t_{i,j} \sim \mathcal{N}(\sigma, r_\sigma)$
      $a^t_{i,j} \leftarrow s^t_{i,j} \cdot a_{i,j} + m^t_{i,j}$

In experiments, $n_\mu = 0.5,\ n_\sigma = 1.25$ and $r_\mu = r_\sigma = 0.1$

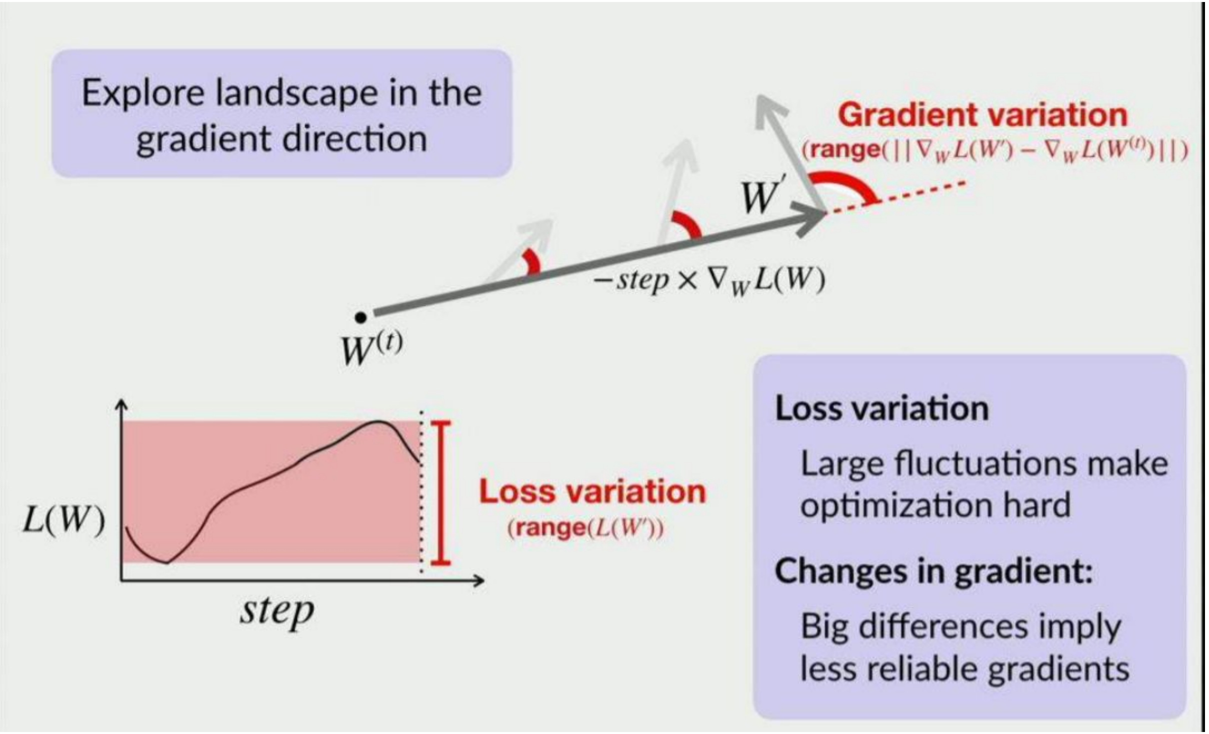
# Experiment: Noisy Batch Normalization



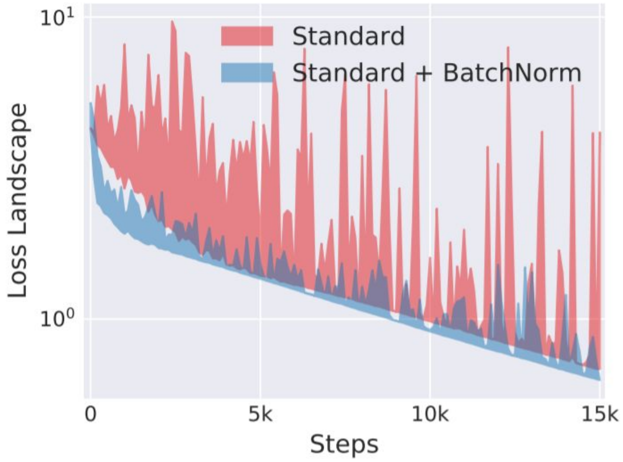
# Experiment: Noisy Batch Normalization



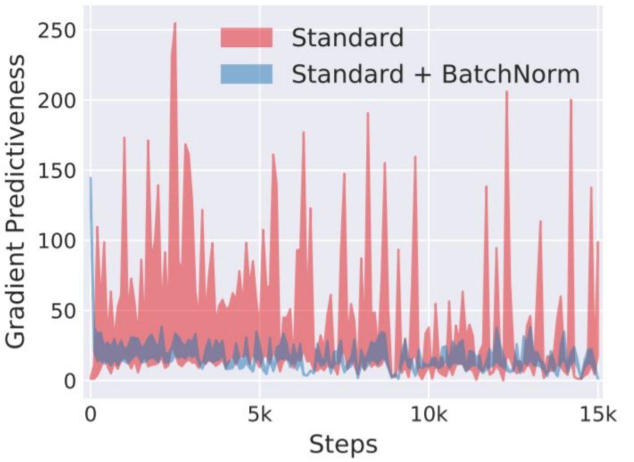
# Experiment: Gradient and Loss Variation



# Experiment: Gradient and Loss Variation



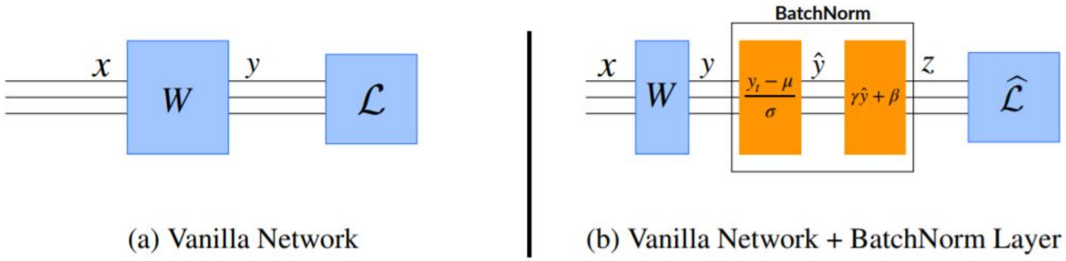

(a) loss landscape




(b) gradient predictiveness

At a particular training step, we measure the variation (shaded region) in loss (a) and L2 changes in the gradient (b) as we move in the gradient direction.

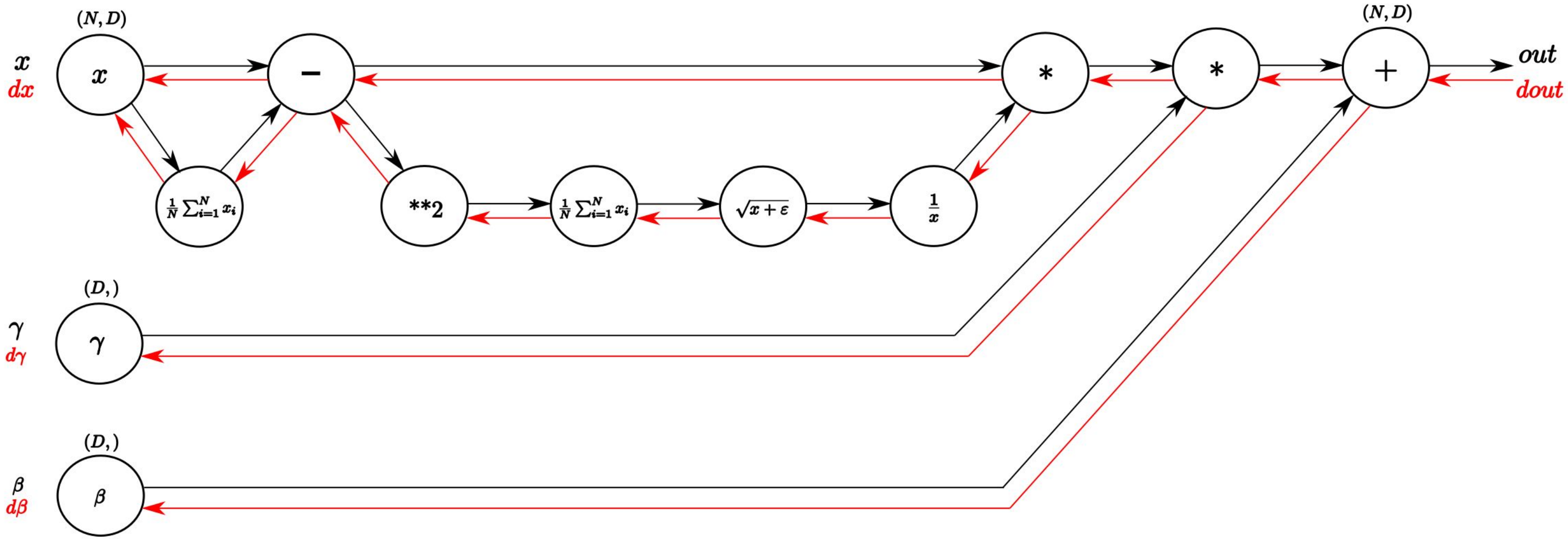
# Theoretical Results



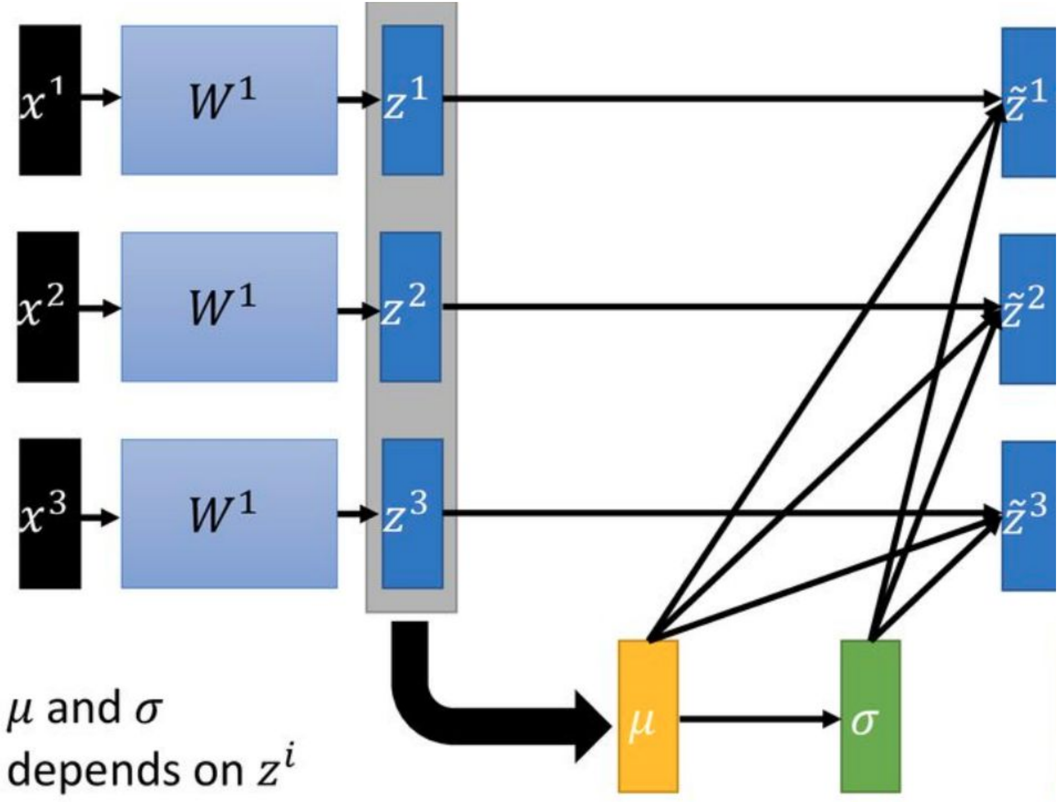

$$|f(x_1 - f(x_2))| \leq L||x_1 \; - x_2||$$

$$\left|\left|\nabla_{y_j}\hat{L}\right|\right|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\left|\left|\nabla_{y_j}L\right|\right|^2 - \frac{1}{m}\langle 1, \, \nabla_{y_j}L\rangle^2 - \frac{1}{m}\langle \nabla_{y_j}L, \, \hat{y_j}\rangle^2\right)$$

# What is the difference with division by a constant?

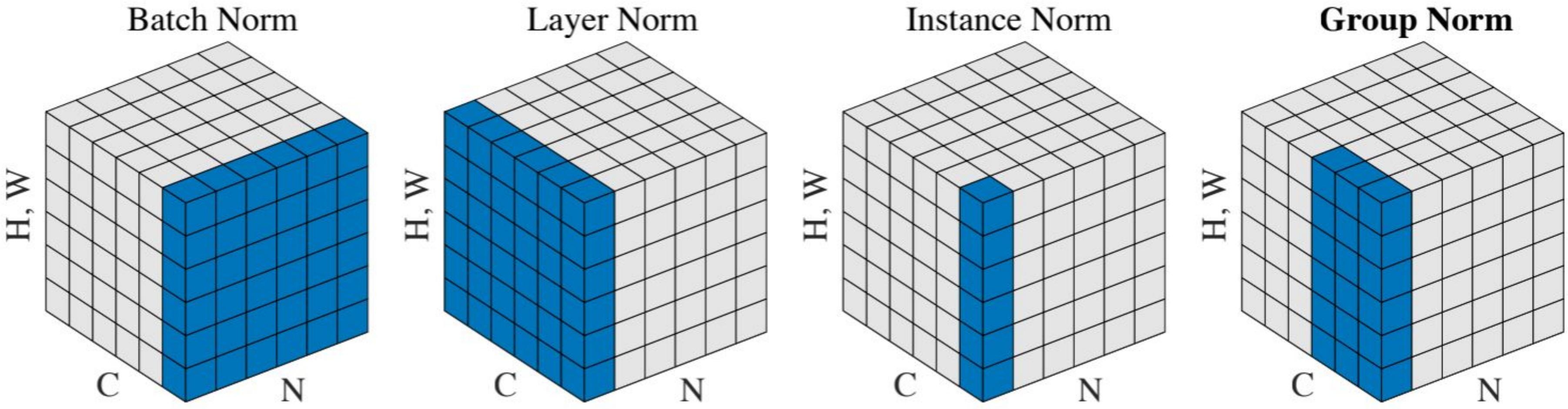


# Batch Norm Problems



# Group Normalization

[Wu & He, 2018]




Normalization methods. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

# Group Normalization: realization

$$y_i = \gamma \hat{x_i} + \beta$$

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i)$$

$$\mu_i = \frac{1}{m} \sum_{k \in S_i} x_k$$

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{k \in S_i} (x_k - \mu_i)^2 + \epsilon}$$

$$S_i = \left\{ k \mid k_n = i_n, \left\lfloor \frac{k_C}{C/G} \right\rfloor = \left\lfloor \frac{i_C}{C/G} \right\rfloor \right\}$$

```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x − mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x ∗ gamma + beta
```

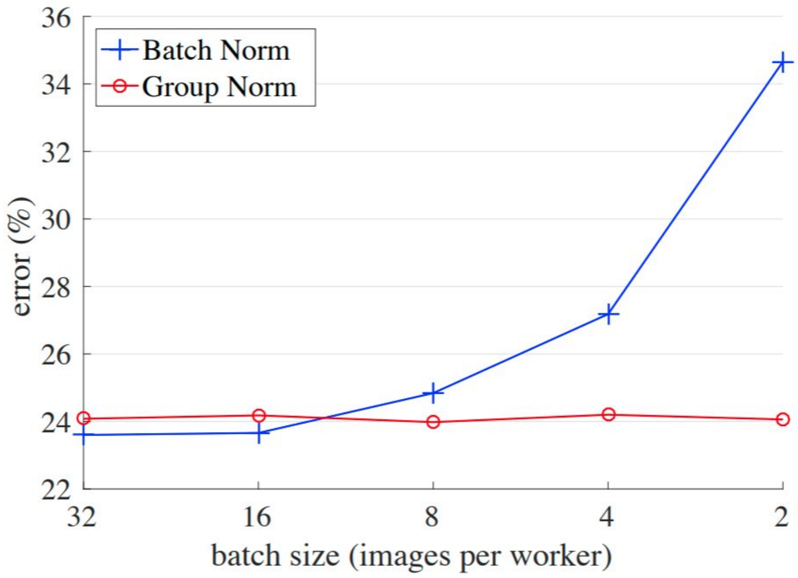
# Group Normalization: cases

| # groups ($G$) | | | | | | |
|---|---|---|---|---|---|---|
| 64 | 32 | 16 | 8 | 4 | 2 | 1 (=LN) |
| 24.6 | **24.1** | 24.6 | 24.4 | 24.6 | 24.7 | 25.3 |
| *0.5* | - | *0.5* | *0.3* | *0.5* | *0.6* | *1.2* |

| # channels per group | | | | | | |
|---|---|---|---|---|---|---|
| 64 | 32 | 16 | 8 | 4 | 2 | 1 (=IN) |
| 24.4 | 24.5 | **24.2** | 24.3 | 24.8 | 25.6 | 28.4 |
| *0.2* | *0.3* | - | *0.1* | *0.6* | *1.4* | *4.2* |

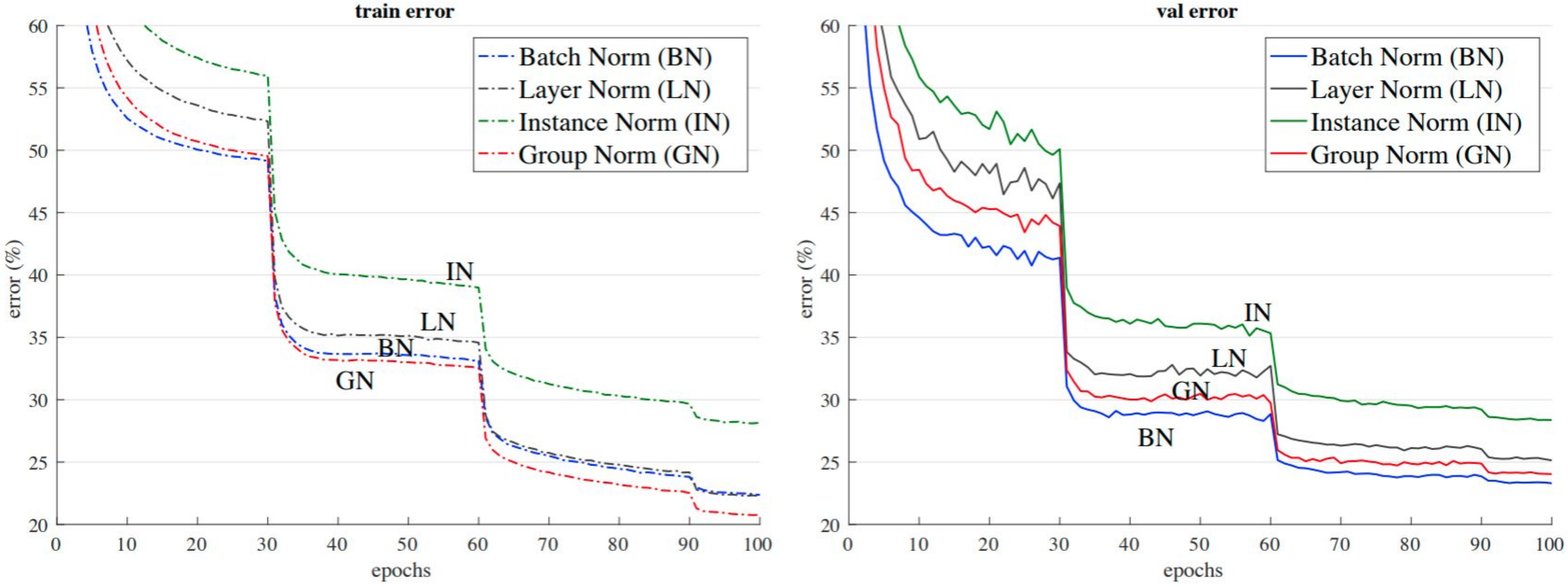
Group division. We show ResNet-50's validation error (%) in ImageNet, trained with 32 images/GPU. (Top): a given number of groups. (Bottom): a given number of channels per group. The last rows show the differences with the best number.

# Group Normalization: better than BN?




ImageNet classification error vs. batch sizes. This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set

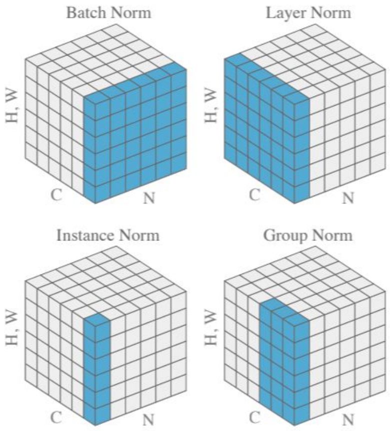
# Group Normalization: better than BN?

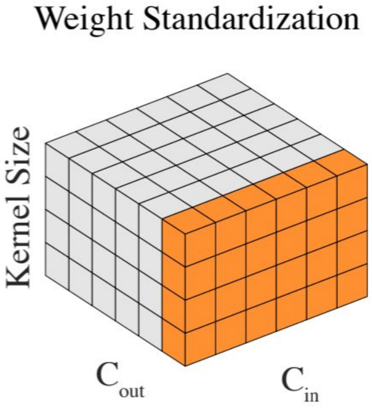


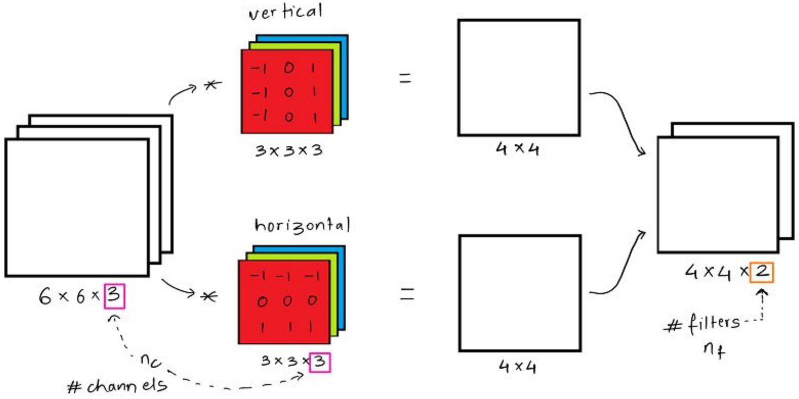

Comparison of error curves with a batch size of 32 images/GPU. We show the ImageNet training error (left) and validation error (right) vs. numbers of training epochs. The model is ResNet-50.

# Weight Standardization

[Qiao et al., 2019]



# Weight Standardization

$$y = \hat{W} * x; \hat{W} \in \mathbb{R}^{O \times I} \text{ denotes the weights in the layer}$$

$*$ denotes the convolution operation

$$\hat{W} = \left[ \hat{W}_{i,j} \middle| \hat{W}_{i,j} = \frac{W_{i,j} - \mu_{W_{i,\cdot}}}{\sigma_{W_{i,\cdot} + \epsilon}} \right]$$

$$\mu_{W_{i,\cdot}} = \frac{1}{I} \sum_{j=1}^{I} W_{i,j}$$

$$\sigma_{W_{i,\cdot}} = \sqrt{\frac{1}{I} \sum_{i=1}^{I} \left( W_{i,j} - \mu_{W_{i,\cdot}} \right)^2}$$

```
class StdConv2d(nn.Conv2d):

  def forward(self, x):
    w = self.weight
    v, m = torch.var_mean(w, dim=[1, 2, 3], keepdim=True, unbiased=False)
    w = (w - m) / torch.sqrt(v + 1e-10)
    return F.conv2d(x, w, self.bias, self.stride, self.padding,
                    self.dilation, self.groups)
```

# Weight Standardization: Lipschitzness

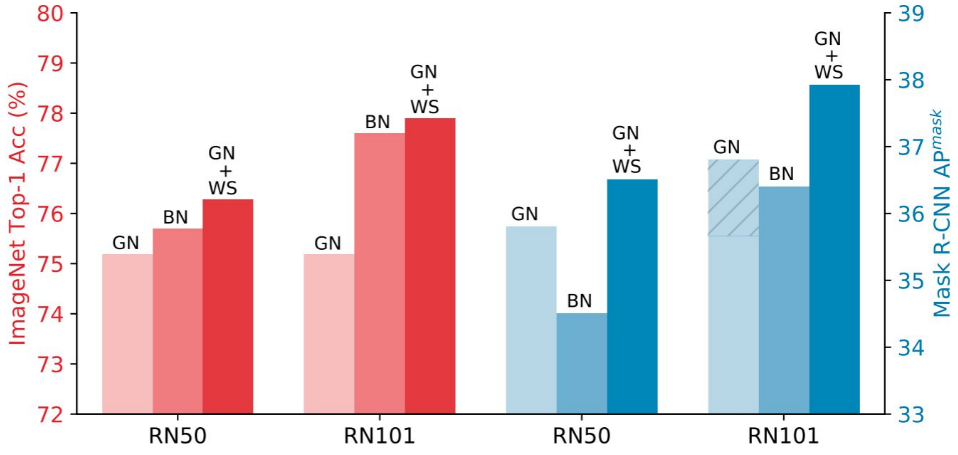
$$\dot{W}_{c,\cdot} = W_{c,\cdot} - \frac{1}{I}1\langle 1,\, W_{c,\cdot}\rangle$$

$$\hat{W}_{c,\cdot} = \dot{W}_{c,\cdot} / \left(\sqrt{\frac{1}{I}\langle 1,\, \dot{W}^{o2}_{c,\,.}\rangle}\right)$$

$$\left|\left|\nabla_{\dot{W}_{c,\cdot}} L\right|\right|^2 = \frac{1}{\sigma_{W_{c,\cdot}}}\left(\left|\left|\nabla_{\hat{W}_{c,\cdot}} L\right|\right|^2 - \frac{1}{I}\langle \hat{W}_{c,\cdot},\, \nabla_{\hat{W}_{c,\cdot}} L\rangle^2\right)$$

$$\left|\left|\nabla_{W_{c,\cdot}} L\right|\right|^2 = \left|\left|\nabla_{\dot{W}_{c,\cdot}} L\right|\right|^2 - \frac{1}{I\cdot\sigma_{W^2_{c,\cdot}}}\langle 1, \nabla_{\hat{W}_{c,\cdot}} L\rangle^2$$

# Weight Standardization: better than BN?




Comparing BN, GN, and WS used with GN on ImageNet and COCO. On ImageNet, BN is trained with large batch sizes while GN and GN+WS are trained with 1 image/GPU. On COCO, BN is frozen for micro-batch training. GN+WS still outperforms both BN and GN comfortably.

| | Plain Conv | Weight Std. |
|---|---|---|
| Batch Norm. | 75.6 | 75.8 |
| Group Norm. | 70.2 | **76.0** |

Top-1 accuracy of ResNet-50 trained from scratch on ILSVRC-2012 (BiT-S) with a batch-size of 4096.

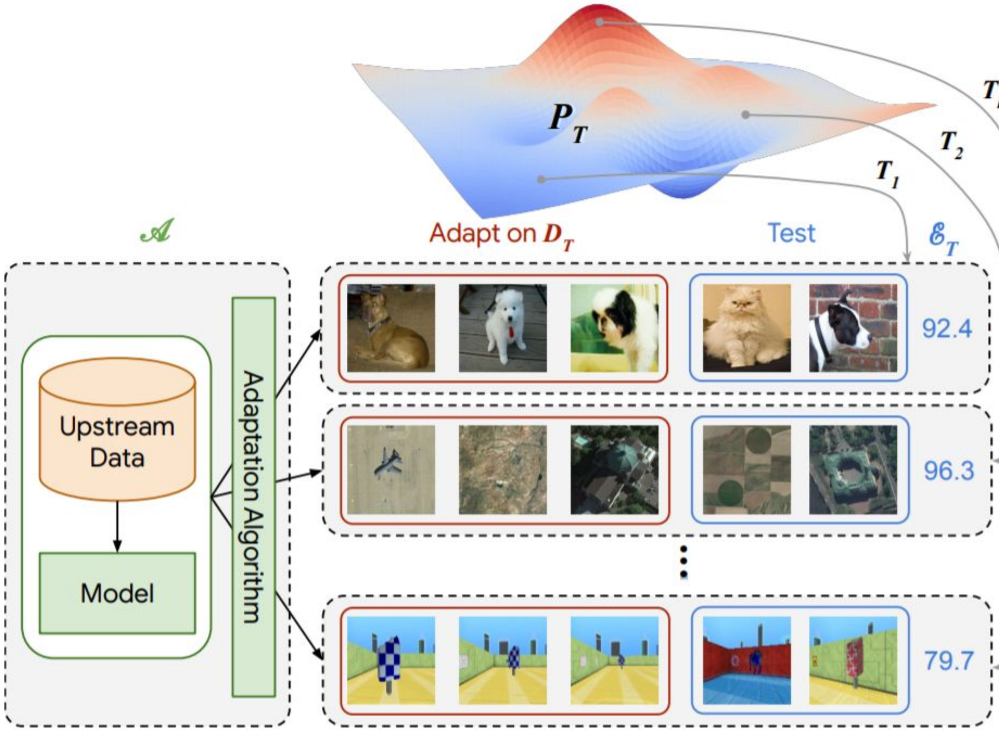
# Downstream Tasks

| | BiT-L | Generalist SOTA | Specialist SOTA |
|---|---|---|---|
| ILSVRC-2012 | **87.54 ± 0.02** | 86.4 [57] | 88.4 [61]* |
| CIFAR-10 | **99.37 ± 0.06** | 99.0 [19] | - |
| CIFAR-100 | **93.51 ± 0.08** | 91.7 [55] | - |
| Pets | **96.62 ± 0.23** | 95.9 [19] | 97.1 [38] |
| Flowers | **99.63 ± 0.03** | 98.8 [55] | 97.7 [38] |
| VTAB (19 tasks) | **76.29 ± 1.70** | 70.5 [58] | - |

| | ILSVRC-2012 | CIFAR-10 | CIFAR-100 | Pets | Flowers | VTAB-1k (19 tasks) |
|---|---|---|---|---|---|---|
| BiT-S (ILSVRC-2012) | 81.30 | 97.51 | 86.21 | 93.97 | 89.89 | 66.87 |
| BiT-M (ImageNet-21k) | 85.39 | 98.91 | 92.17 | 94.46 | 99.30 | 70.64 |
| Improvement | +4.09 | +1.40 | +5.96 | +0.49 | +9.41 | +3.77 |

# VTAB: The Visual Task Adaptation Benchmark

[Zhai et al., 2020]



# VTAB: The Visual Task Adaptation Benchmark

object identification, scene classification, pathology detection, counting, localization, and 3D geometry -> classification

| Category | Dataset | Train size | Classes | Reference |
|---|---|---|---|---|
| • Natural | Caltech101 | 3,060 | 102 | (Li et al., 2006) |
| • Natural | CIFAR-100 | 50,000 | 100 | (Krizhevsky, 2009) |
| • Natural | DTD | 3,760 | 47 | (Cimpoi et al., 2014) |
| • Natural | Flowers102 | 2,040 | 102 | (Nilsback & Zisserman, 2008) |
| • Natural | Pets | 3,680 | 37 | (Parkhi et al., 2012) |
| • Natural | Sun397 | 87,003 | 397 | (Xiao et al., 2010) |
| • Natural | SVHN | 73,257 | 10 | (Netzer et al., 2011) |
| • Specialized | EuroSAT | 21,600 | 10 | (Helber et al., 2019) |
| • Specialized | Resisc45 | 25,200 | 45 | (Cheng et al., 2017) |
| • Specialized | Patch Camelyon | 294,912 | 2 | (Veeling et al., 2018) |
| • Specialized | Retinopathy | 46,032 | 5 | (Kaggle & EyePacs, 2015) |
| • Structured | Clevr/count | 70,000 | 8 | (Johnson et al., 2017) |
| • Structured | Clevr/distance | 70,000 | 6 | (Johnson et al., 2017) |
| • Structured | dSprites/location | 663,552 | 16 | (Matthey et al., 2017) |
| • Structured | dSprites/orientation | 663,552 | 16 | (Matthey et al., 2017) |
| • Structured | SmallNORB/azimuth | 36,450 | 18 | (LeCun et al., 2004) |
| • Structured | SmallNORB/elevation | 36,450 | 9 | (LeCun et al., 2004) |
| • Structured | DMLab | 88,178 | 6 | (Beattie et al., 2016) |
| • Structured | KITTI/distance | 5,711 | 4 | (Geiger et al., 2013) |

# Fine-tuning: BiTHyperRule

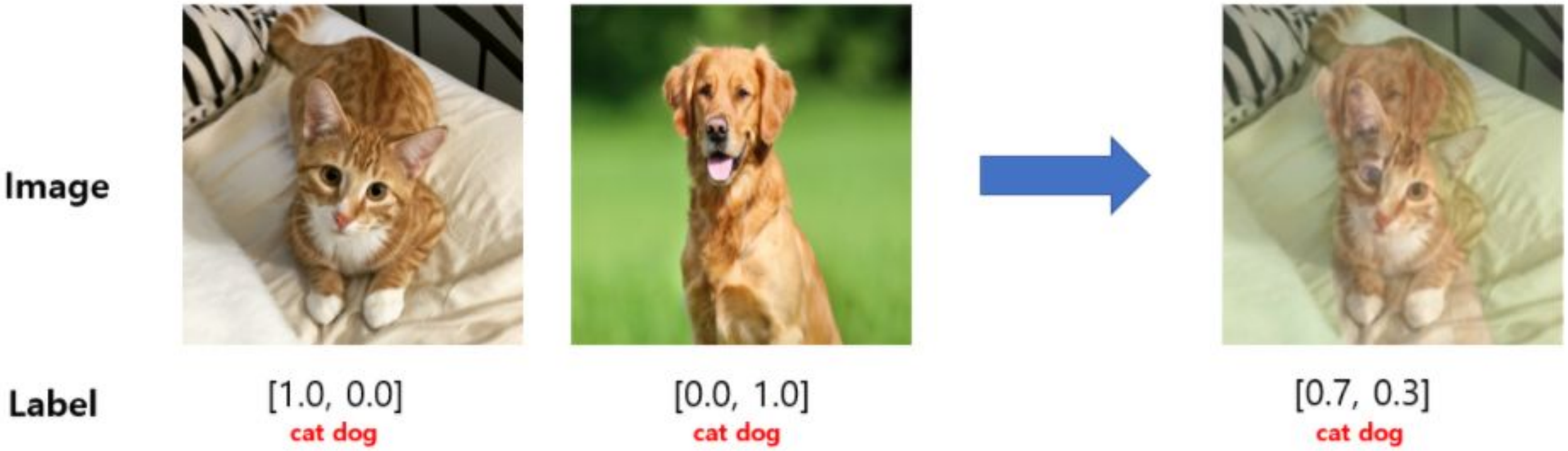
- SGD + momentum 0.9;
- initial lr = 0.003; decay the learning rate by 10 at 30%, 60% and 90% of the training steps
- Batch Size = 512
- Augmentation: random crops + horizontal flips
- MixUp, with α = 0.1, for medium and large tasks

| **Dataset size (number of examples)** | **Schedule length (fine-tuning)** | **Use MixUp? (mixing parameter $\alpha=0.1$ )** |
|---|---|---|
| less than 20K | 500 steps | No |
| 20k-500k | 10K steps | Yes |
| more than 500k | 20K steps | Yes |

| **Input Image area** | **1 step: resize to** | **2 step: random crop to** |
|---|---|---|
| < 96 x 96 | 160 x 160 | 128 x 128 |
| > 96 x 96 | 448 x 448 | 384 x 384 |
| R152x4 & > 96 x 96 | 512 x 512 | 480 x 480 |

# mixup: beyond empirical risk minimization

[Zhang et al., 2018]



# mixup: empirical risk minimization (ERM)

$f \in \mathcal{F};\ (x,\ y)\ \sim\ P(X,\ Y)$

$P_\delta(x,\ y)\ =\ \frac{1}{n}\sum_{i=1}^{n}\delta(x = x_i, y = y_i)$

$R(f)\ =\ \int l(f(x),\ y)dP(x,\ y)$

$R_\delta(f) = \int l(f(x),\ y)dP_\delta(x, y) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i),\ y_i)$

# mixup: empirical vicinal risk minimization

$$P_\nu(\tilde{x},\, \tilde{y}) = \; \frac{1}{n}\sum_{i=1}^{n} \nu(\tilde{x},\, \tilde{y}|x_i,\, y_i)$$

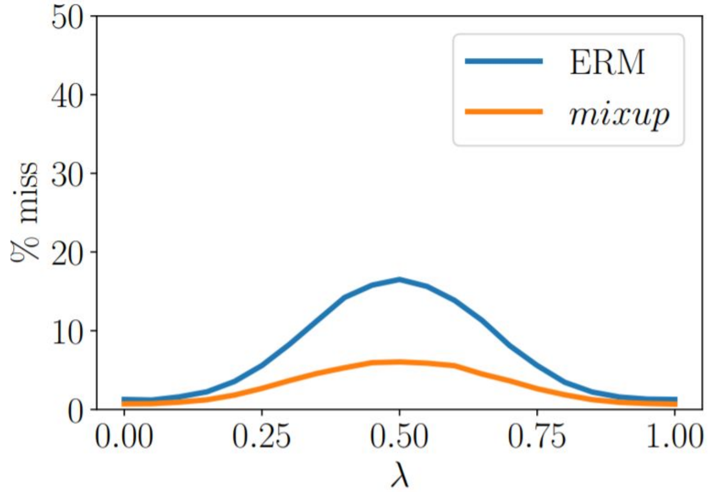
$$\nu(\tilde{x},\, \tilde{y}|x_{i,}y_i) \;=\; N\left(\tilde{x} - x_i,\, \sigma^2\right)\delta(\tilde{y} = y_i)$$

$$R_\nu(f) = \frac{1}{m}\sum_{i=1}^{m} l(f(\tilde{x}_i),\, \tilde{y}_i)$$

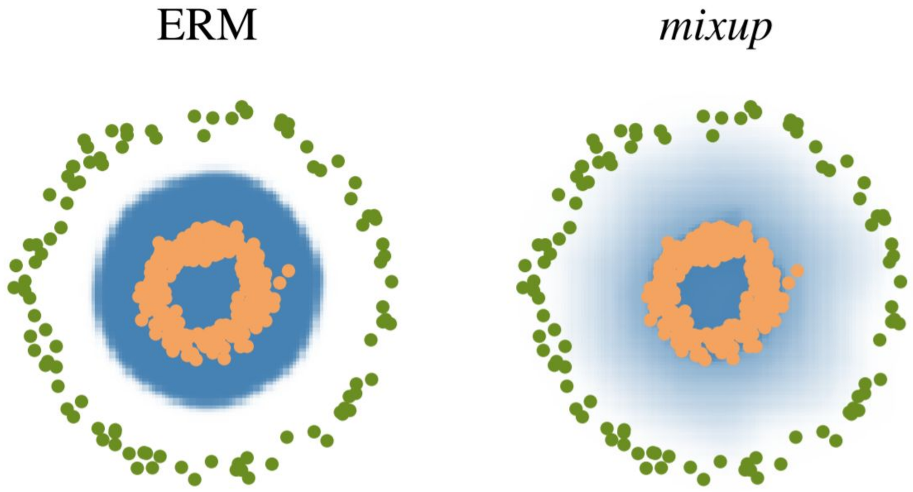
$$\lambda \sim \mathrm{Beta}(\alpha, \alpha), \mathrm{for}\, \alpha \in (0, \infty)$$

$$\mu(\tilde{x},\, \tilde{y}|\, x_i,\, y_i) = \frac{1}{n}\sum_{j}^{n} E_\lambda[\delta(\tilde{x} = \lambda \cdot x_i + (1-\lambda)\cdot x_j,\, \tilde{y} = \lambda \cdot y_i + (1-\lambda)\cdot y_j)]$$

# What is mixup doing?




Prediction errors in-between training data. Evaluated at x = λ+xi(1−λ)xj , a prediction is counted as a “miss” if it does not belong to {yi, yj}. The model trained with mixup has fewer misses.

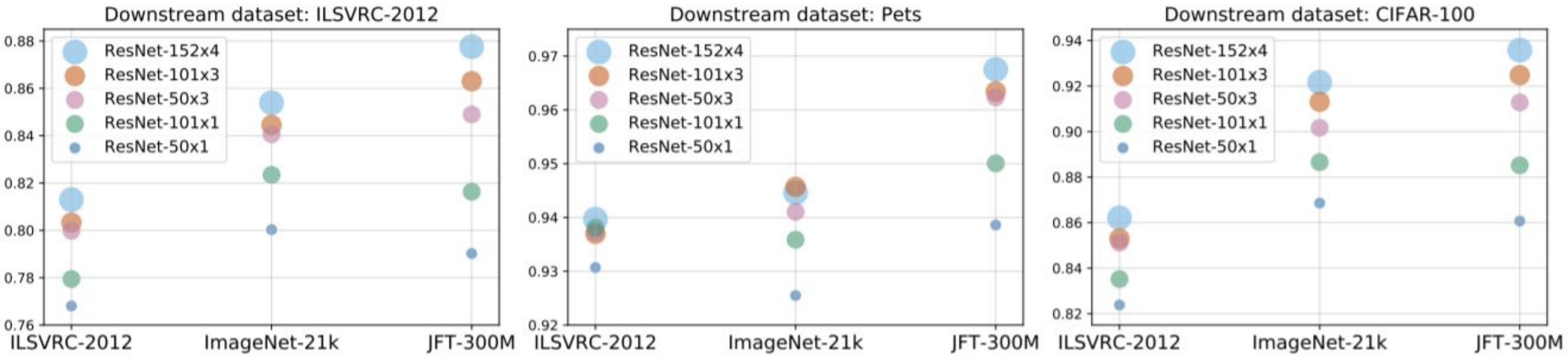



Effect of mixup (α = 1) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates p(y = 1|x).

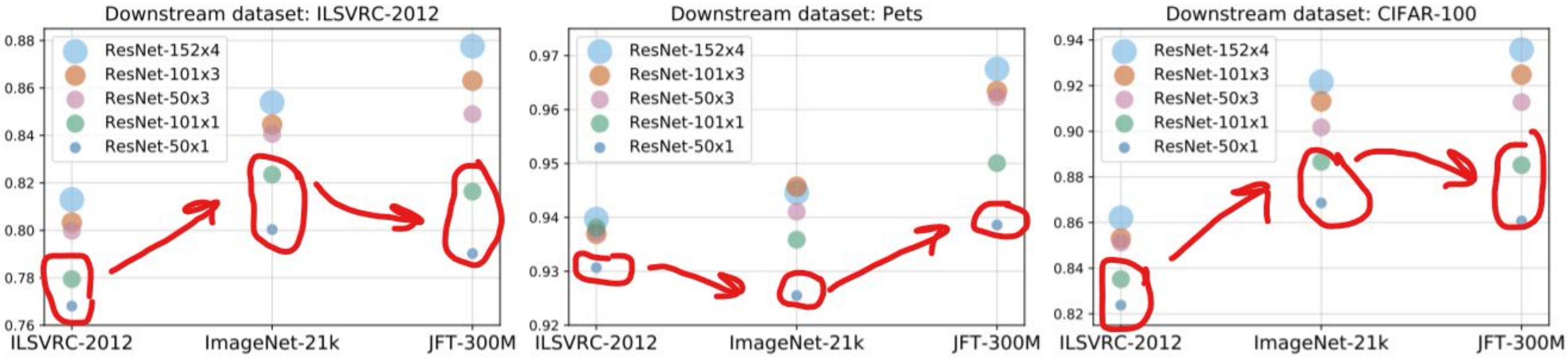
# What is mixup doing?

| Model | Method | Epochs | Top-1 Error | Top-5 Error |
|---|---|---|---|---|
| ResNet-50 | ERM (Goyal et al., 2017) | 90 | 23.5 | - |
| | *mixup* $\alpha = 0.2$ | 90 | **23.3** | **6.6** |
| ResNet-101 | ERM (Goyal et al., 2017) | 90 | 22.1 | - |
| | *mixup* $\alpha = 0.2$ | 90 | **21.5** | **5.6** |
| ResNeXt-101 32*4d | ERM (Xie et al., 2016) | 100 | 21.2 | - |
| | ERM | 90 | 21.2 | 5.6 |
| | *mixup* $\alpha = 0.4$ | 90 | **20.7** | **5.3** |
| ResNeXt-101 64*4d | ERM (Xie et al., 2016) | 100 | 20.4 | 5.3 |
| | *mixup* $\alpha = 0.4$ | 90 | **19.8** | **4.9** |
| ResNet-50 | ERM | 200 | 23.6 | 7.0 |
| | *mixup* $\alpha = 0.2$ | 200 | **22.1** | **6.1** |
| ResNet-101 | ERM | 200 | 22.0 | 6.1 |
| | *mixup* $\alpha = 0.2$ | 200 | **20.8** | **5.4** |
| ResNeXt-101 32*4d | ERM | 200 | 21.3 | 5.9 |
| | *mixup* $\alpha = 0.4$ | 200 | **20.1** | **5.0** |

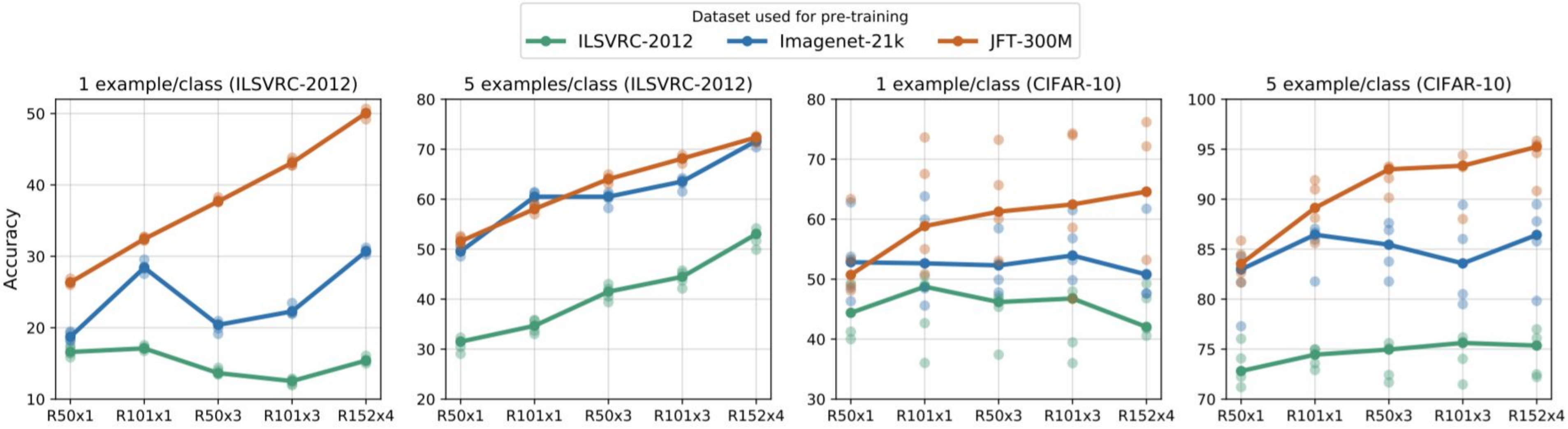
# BiT Interesting results: Data & Model Sizes



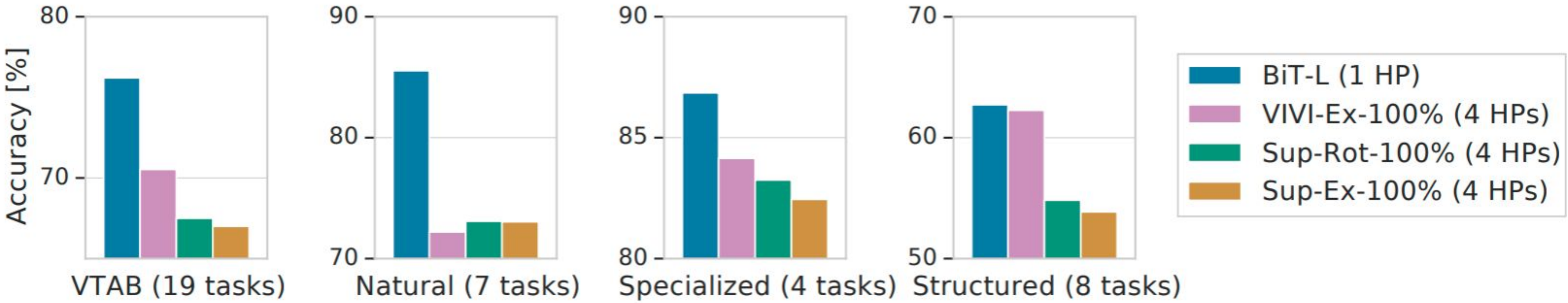
# BiT Interesting results: Data & Model Sizes



# BiT Interesting results: Few-Shot training



# BiT Interesting results: VTAB



# Literature


https://arxiv.org/abs/1912.11370 - Big Transfer

https://arxiv.org/pdf/1803.08494.pdf - Group Normalization

https://arxiv.org/pdf/1903.10520v1.pdf - Weight Standardization

https://arxiv.org/pdf/1710.09412.pdf - mixup

https://arxiv.org/pdf/1805.11604.pdf - How Does Batch Normalization Help Optimization?

https://www.youtube.com/watch?v=EvAVCxZJN2U&feature=emb_logo - How Does Batch Normalization Help Optimization? (video)