

# 꼼꼼한 딥러닝 논문 리뷰와 코드 실습

Deep Learning Paper Review and Code Practice

나동빈([dongbinna@postech.ac.kr](mailto:dongbinna@postech.ac.kr))

Pohang University of Science and Technology

# Bag of Tricks for Image Classification

# 논문 소개: Bag of Tricks for Image Classification with Convolutional Neural Networks

- 기존에 다양한 논문에서 제안된 테크닉들을 조합해 이미지 분류 성능을 향상시킵니다.
- 사소한 트릭(trick)을 적절히 모아 사용하면 큰 성능 향상을 이끌어낼 수 있습니다.
  - 손실 함수(loss function), 데이터 전처리(data preprocessing), 최적화(optimization) 등

Model	FLOPs	top-1	top-5
ResNet-50 [9]	3.9 G	75.3	92.2
ResNeXt-50 [27]	4.2 G	77.8	-
SE-ResNet-50 [12]	3.9 G	76.71	93.38
SE-ResNeXt-50 [12]	4.3 G	78.90	94.51
DenseNet-201 [13]	4.3 G	77.42	93.66
ResNet-50 + tricks (ours)	4.3 G	<b>79.29</b>	<b>94.63</b>

성능 향상

[Table] Computational costs and validation accuracy of various models.

## 일반적인 이미지 분류 모델의 학습 절차 (Baseline Training Procedure)

### [ 데이터 전처리(data preprocessing) 과정 ]

1. 이미지를 불러와 float-32로 디코딩(decoding)
2. 이미지 내에서 랜덤하게 직사각형 영역을 잘라내 (cropping randomly) 224 X 224 크기로 변경
3. 50% 확률로 좌우 반전(flipping horizontally) ←
4. 색조(hue), 채도(saturation), 밝기(brightness)를 랜덤하게 변경
5. PCA 노이즈(noise) 추가
6. 입력 정규화(input normalization)


---

**Algorithm 1** Train a neural network with mini-batch stochastic gradient descent.

---

```
initialize(net)
for epoch = 1, ..., K do
  for batch = 1, ..., #images/b do
    images ← uniformly random sample b images
     $X, y \leftarrow \text{preprocess}(\text{images})$ 
     $z \leftarrow \text{forward}(\text{net}, X)$ 
     $\ell \leftarrow \text{loss}(z, y)$ 
    grad ← backward( $\ell$ )
    update(net, grad)
  end for
end for
```

---

 평가 시 256 X 256으로 크기 변경 후 중간의 224 X 224 영역을 잘라 정규화를 진행합니다.

## 베이스라인 모델의 성능 분석

- 8개의 Nvidia V100 GPU와 Xavier 초기화 알고리즘 사용하고, 배치 크기는 256으로 설정합니다.
- 총 120 epochs: 30, 60, 90번째 epoch에서 학습률(learning rate)을 10씩 나눕니다.
- ISLVR2012 데이터셋 (ImageNet)

원본 논문과  
유사한 구현 결과

Model	Baseline		Reference	
	Top-1	Top-5	Top-1	Top-5
ResNet-50 [9]	75.87	92.70	75.3	92.2
Inception-V3 [26]	77.32	93.43	78.8	94.4
MobileNet [11]	69.03	88.71	70.6	-

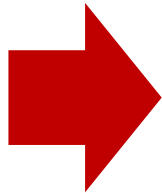
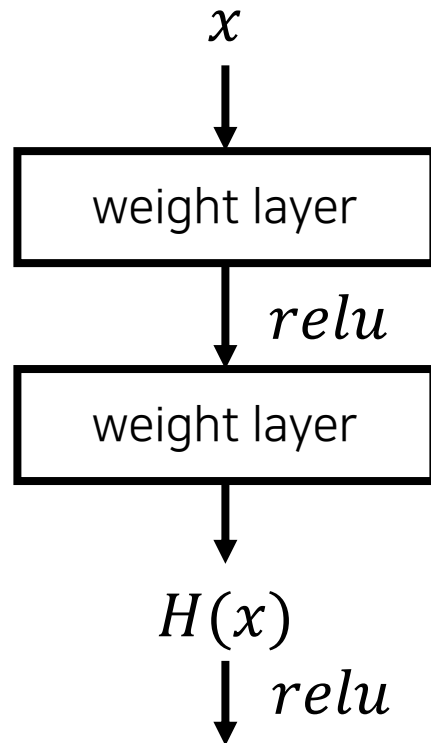
- ResNet과 MobileNet의 PyTorch 입력 전처리 코드 예시 (InceptionNet에서는 이미지 크기가 299 X 299)

```
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

## 배경지식: 잔여 블록 (Residual Block)

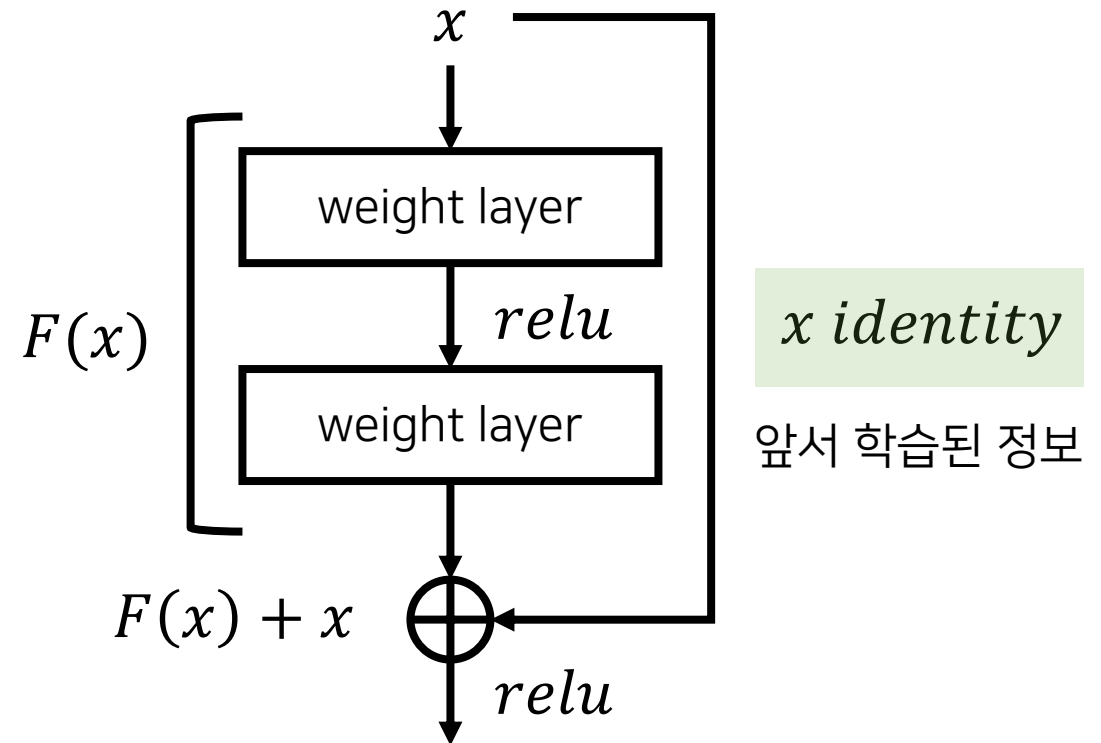
- ResNet에서는 잔여 블록(residual block)을 이용해 네트워크의 최적화(optimization) 난이도를 낮춥니다.
  - 실제로 내재한 mapping인  $H(x)$ 를 곧바로 학습하는 것은 어려우므로 대신  $F(x) = H(x) - x$ 를 학습합니다.

### < Plain layers >



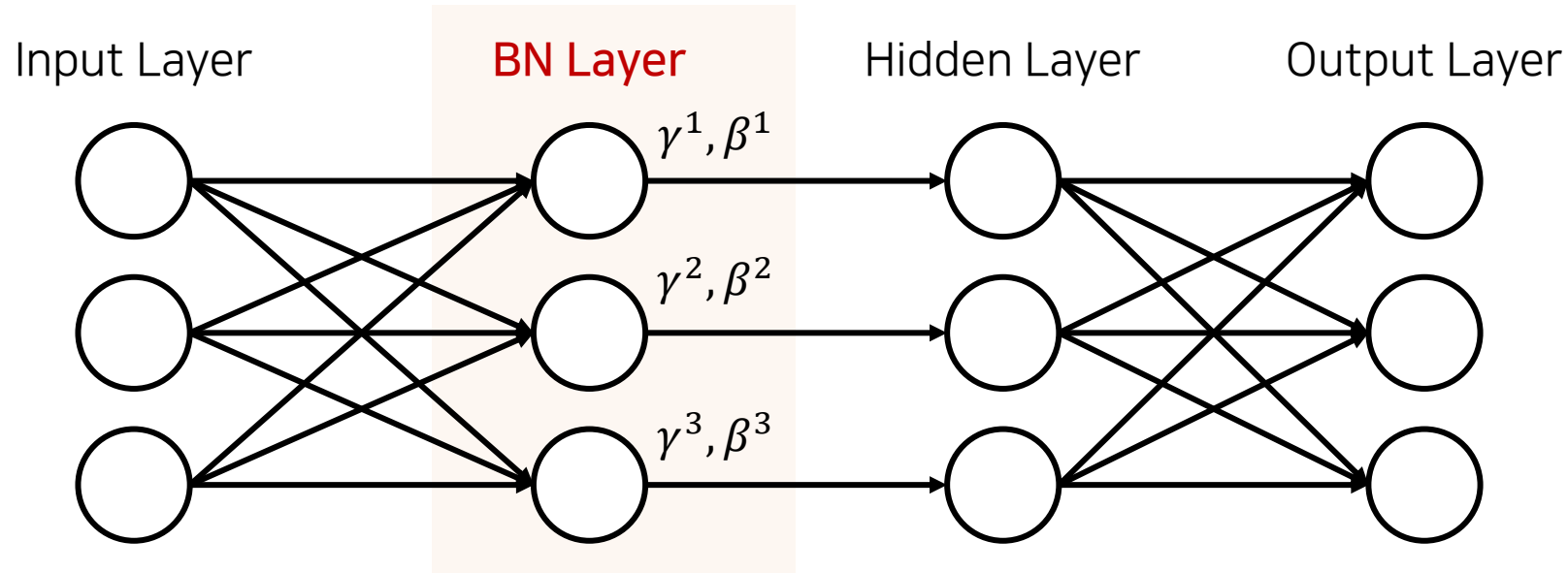
학습이 잘 되는 형태로 변경

### < Residual block >



## 배경지식: 배치 정규화(Batch Normalization)

- 배치 정규화의 잘 알려진 **장점**은 다음과 같습니다.
  - ① 학습 속도(training speed)를 빠르게 할 수 있습니다.
  - ② 가중치 초기화(weight initialization)에 대한 민감도를 감소시킵니다.
  - ③ 모델의 일반화(regularization) 효과가 있습니다.



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (PMLR 2015)

## 배경지식: 배치 정규화(Batch Normalization)

- 입력(Input)
  - A mini-batch:  $Batch = \{x_1, x_2, \dots, x_m\}$
  - Parameters to be learned:  $\gamma, \beta$
- 출력(Output)
  - $\{y_i = BN_{\gamma, \beta}(x_i)\}$

레이어의 입력 차원이  $k$ 일 때, 학습할 두 개의 파라미터  $\gamma$ 과  $\beta$  또한  $k$ 차원을 가집니다.

$$\mu_{Batch} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{평균}$$

$$\sigma_{Batch}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{Batch})^2 \quad // \text{분산}$$

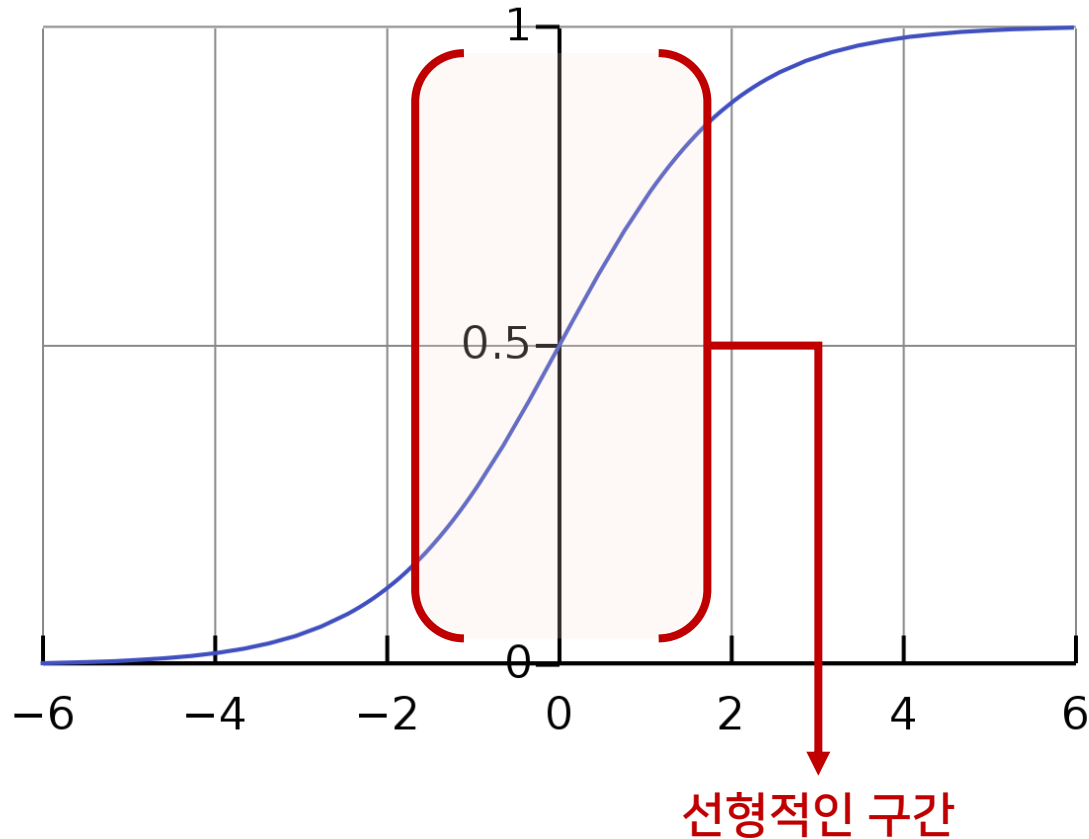
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{Batch}}{\sqrt{\sigma_{Batch}^2 + \epsilon}} \quad // \text{정규화}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$$



## 배경지식: 레이어 입력을 정규화할 때 유의할 점

- 각 레이어를 단순히  $N(0, 1)$ 로 정규화하면 비선형(non-linear) 활성화 함수의 영향력이 감소할 수 있습니다.
- Sigmoid 함수 예시



입력 데이터가  $N(0, 1)$ 로 정규화되므로 대부분의 입력에 대하여 매우 선형적으로 동작합니다.

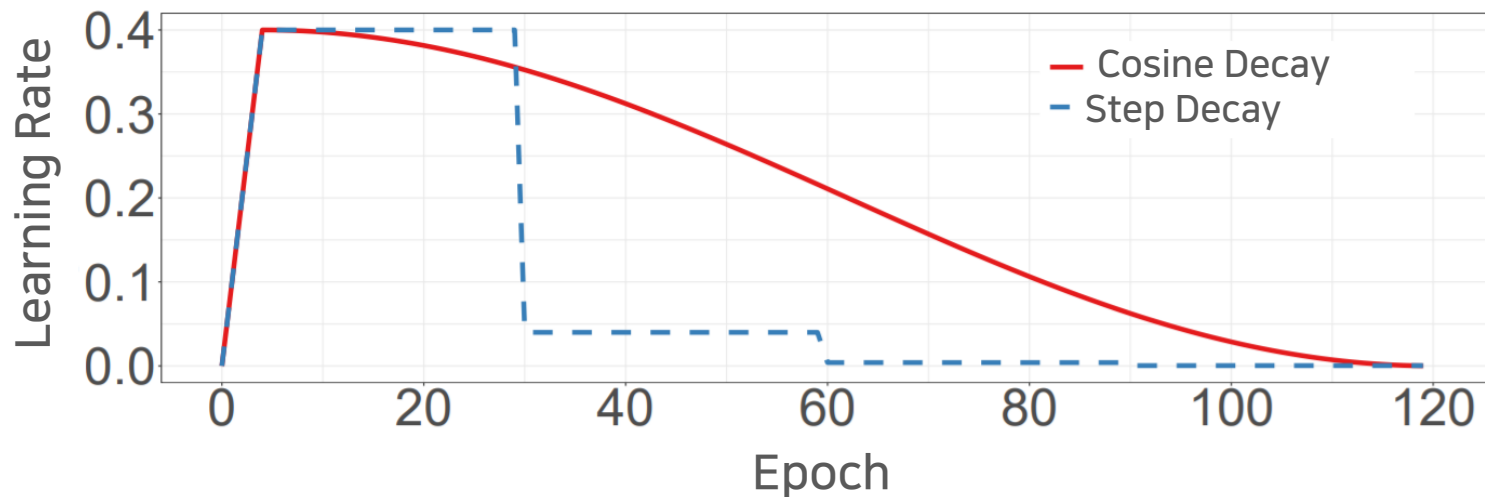


정규화 이후에 사용하는 감마( $\gamma$ )와 베타( $\beta$ )는 non-linearity를 유지할 수 있도록 해줍니다.

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$$

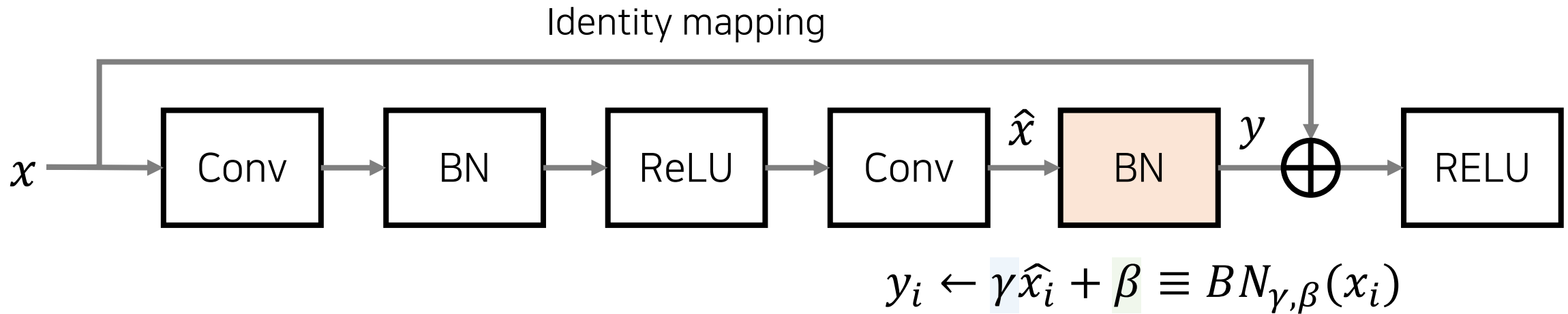
## Large Batch Training: 학습률(Learning Rate)

- 배치 크기가 증가하면 수렴률(convergence rate)이 감소하는 문제가 발생할 수 있습니다.
  - 다시 말해 동일한 횟수로 학습한다면 배치 크기가 큰 모델의 평가 성능이 낮을 수 있습니다.
- 학습률(learning rate)을 선형적으로 증가시킵니다.
  - 초기 학습률을  $0.1 \times b/256$ 으로 설정합니다. ( $b$ 는 배치 크기)
- 워밍업(warmup): 초반에는 작은 학습률을 이용합니다.
  - 초기  $m$ 번의 배치에 대하여 학습률을  $\eta \div m$ 으로 설정합니다. ( $\eta$ 는 초기 학습률)



## Large Batch Training: 학습 파라미터 설정

- 일반적으로 잔여 블록(residual block)의 뒷부분에는 배치 정규화가 사용됩니다.
  - 배치 정규화에서는 일반적으로  $\gamma$ 와  $\beta$ 를 1과 0으로 초기화합니다.
  - 학습을 시작할 때  $\gamma$ 를 0으로 초기화하여 초반 학습 난이도를 쉽게 만들 수 있습니다.



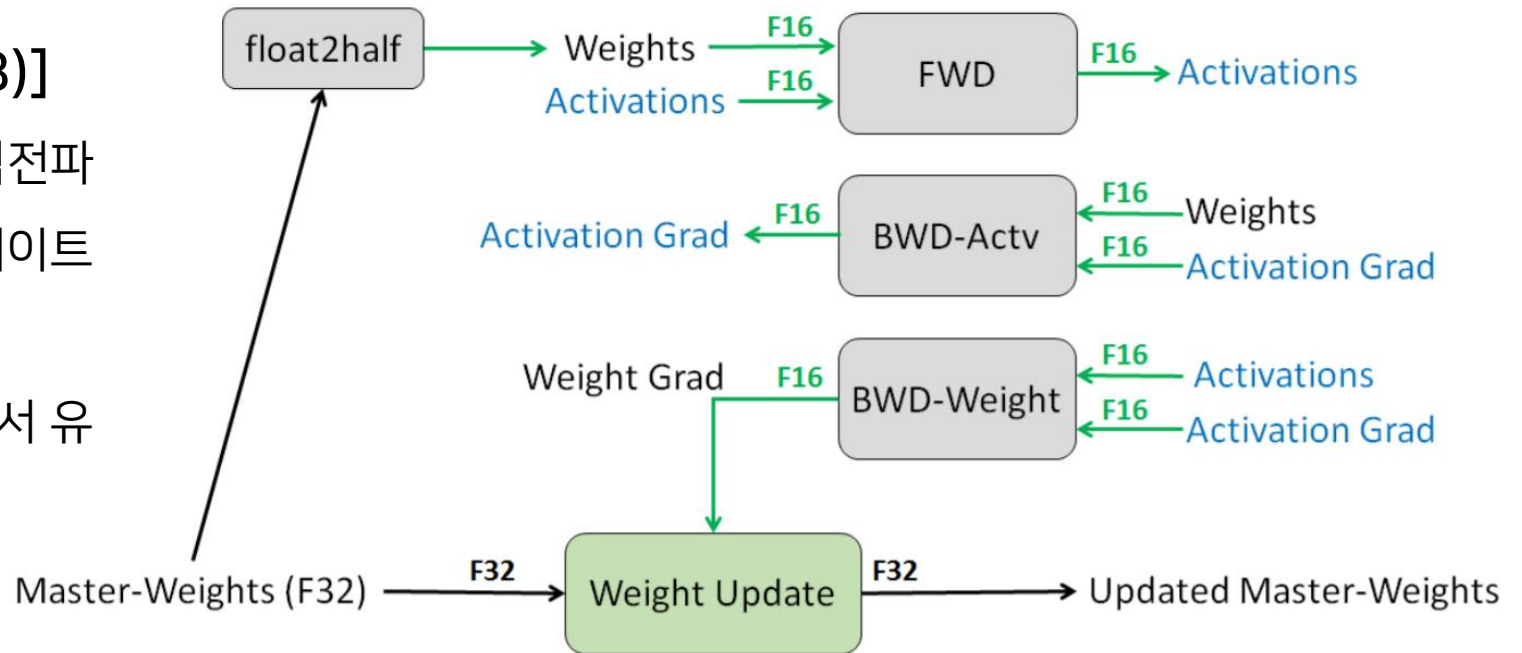
- 정규화 기법으로 사용되는 weight decay는 종종 모든 학습 가능한 파라미터에 적용됩니다.
  - 몇몇 논문에서는 bias에 대하여 weight decay를 적용하지 않는 것을 추천합니다.

## 배경지식: Low-Precision Training

- 일반적으로 뉴럴 네트워크(neural network)는 32-bit floating point 정밀도(precision)를 사용합니다.
- 다양한 하드웨어는 더 낮은 정밀도의 자료형을 효율적으로 지원합니다.
- Nvidia V100은 FP32에 대하여 14 TFLOPS, FP16에 대하여 100 TFLOPS 이상의 성능을 제공합니다.
  - 실제로 V100을 쓰는 경우 FP32를 FP16으로 바꾸는 것으로 학습 속도가 2~3배 빨라질 수 있습니다.

### [Mixed precision training (ICLR 2018)]

- FP32를 FP16으로 변환한 뒤에 순전파 및 역전파를 FP16 자료형으로 수행합니다. 가중치 업데이트를 위해 다시 FP32로 변환합니다.
- FP32만 사용했을 때보다 이미지 분류 작업에서 유사하거나 더 좋은 성능을 보일 수 있습니다.



## Efficient Training (Low-Precision + Large Batch) 성능 분석

- **Baseline:** ResNet-50 with FP32 (BS = 256)
- **Efficient:** ResNet-50 with FP16 (BS = 1,024)
- FP16를 이용해도 다양한 테크닉을 적용하여 높은 성능을 얻을 수 있습니다. 더불어 학습 속도를 매우 빠르게 만들 수 있다는 장점이 있습니다.

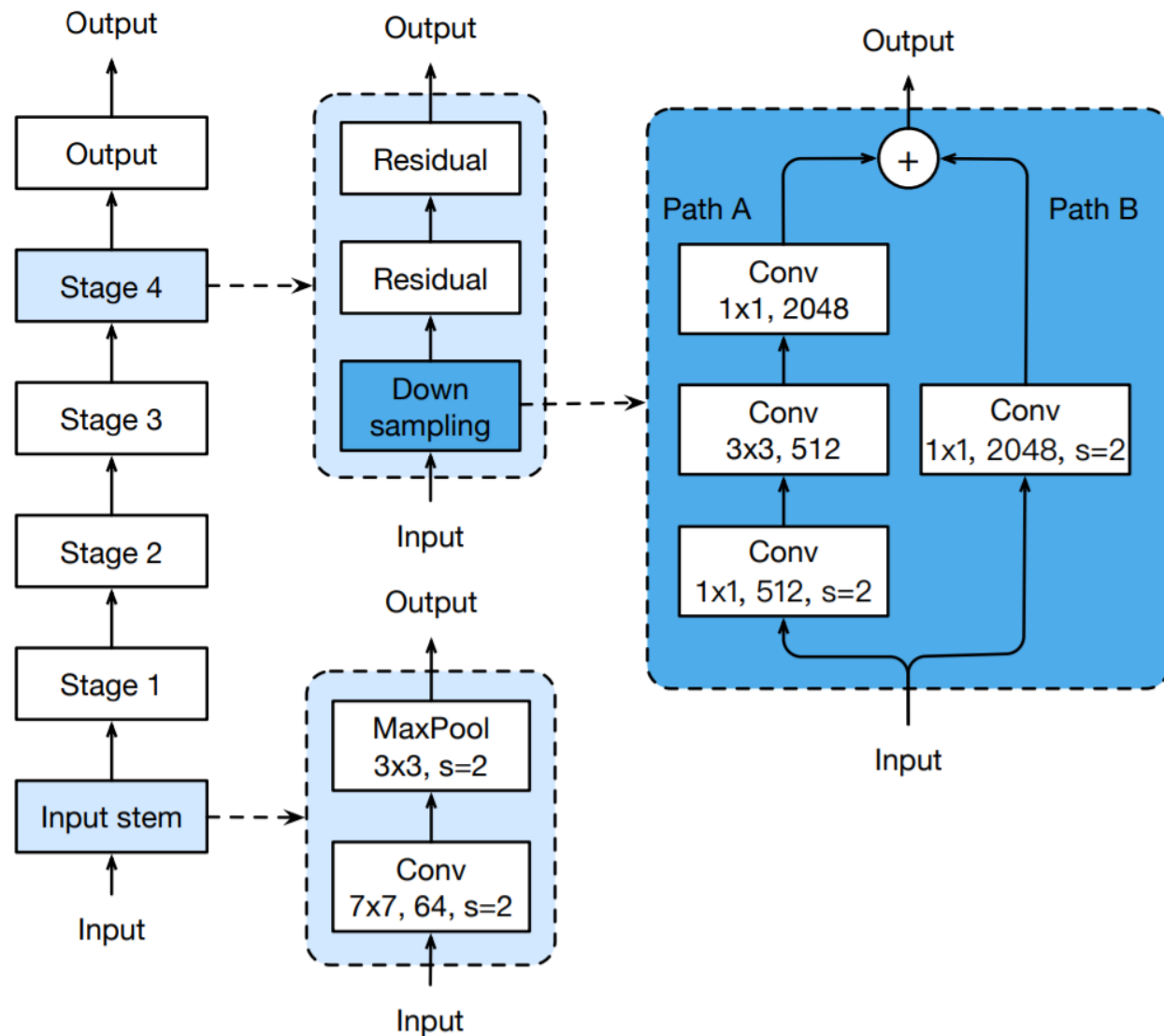
Heuristic	BS=256		BS=1024	
	Top-1	Top-5	Top-1	Top-5
Linear scaling	75.87	92.70	75.17	92.54
+ LR warmup	76.03	92.81	75.93	92.84
+ Zero $\gamma$	76.19	93.03	76.37	92.96
+ No bias decay	76.16	92.97	76.03	92.86
+ FP16	76.15	93.09	76.21	92.97

Model	Efficient			Baseline		
	Time/epoch	Top-1	Top-5	Time/epoch	Top-1	Top-5
ResNet-50	4.4 min	76.21	92.97	13.3 min	75.87	92.70
Inception-V3	8 min	77.50	93.60	19.8 min	77.32	93.43
MobileNet	3.7 min	71.90	90.47	6.2 min	69.03	88.71

[Table] Comparison of the training time and validation accuracy for ResNet-50 between the baseline (BS=256 with FP32) and a more hardware efficient setting (BS=1024 with FP16).

# Model Tweaks: ResNet Architecture

- ResNet-50 아키텍처는 다음과 같습니다.
  - 총 4번의 stage가 포함됩니다.
  - 각 stage에서 다운샘플링을 진행합니다.
- 기본(default) stride size는 1입니다.
- ResNet에는 다양한 variant가 있습니다.



# Model Tweaks: ResNet Tweaks

## [단계적으로 ResNet 아키텍처 변경]

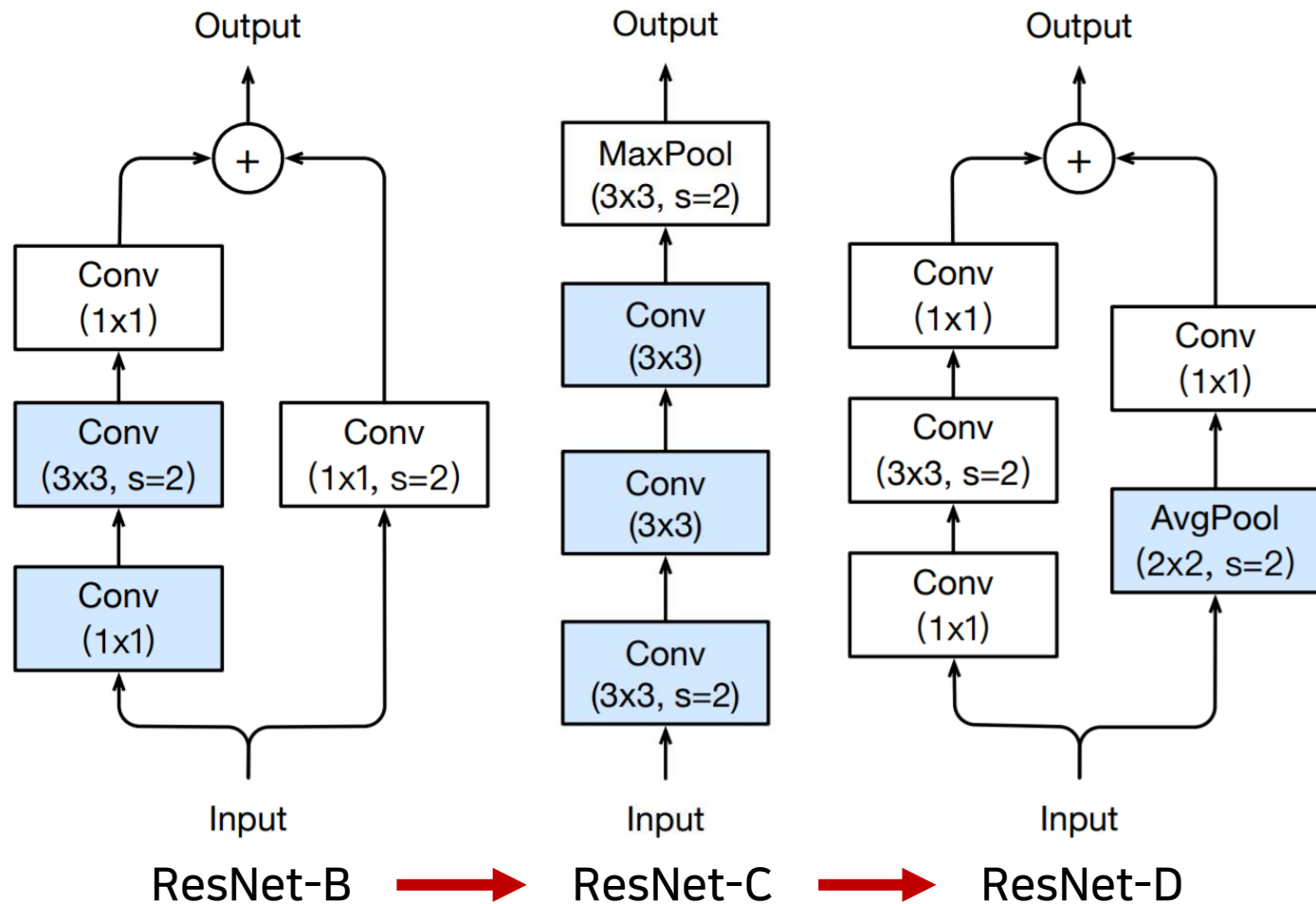
- ResNet-B: downsampling block 변경
- ResNet-C: input stem 변경
- ResNet-D: downsampling block 변경

Model	#params	FLOPs	Top-1	Top-5
ResNet-50	25 M	<b>3.8 G</b>	76.21	92.97
ResNet-50-B	25 M	4.1 G	76.66	93.28
ResNet-50-C	25 M	4.3 G	76.87	93.48
ResNet-50-D	25 M	4.3 G	<b>77.16</b>	<b>93.52</b>

[Table] ResNet 아키텍처를 조금씩 변경하여 점진적으로 성능을 향상시킬 수 있습니다.

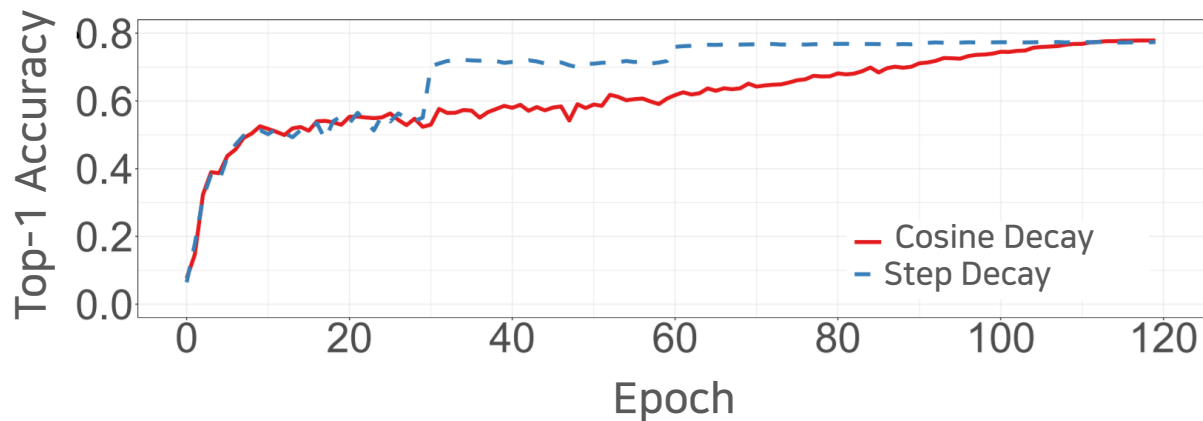
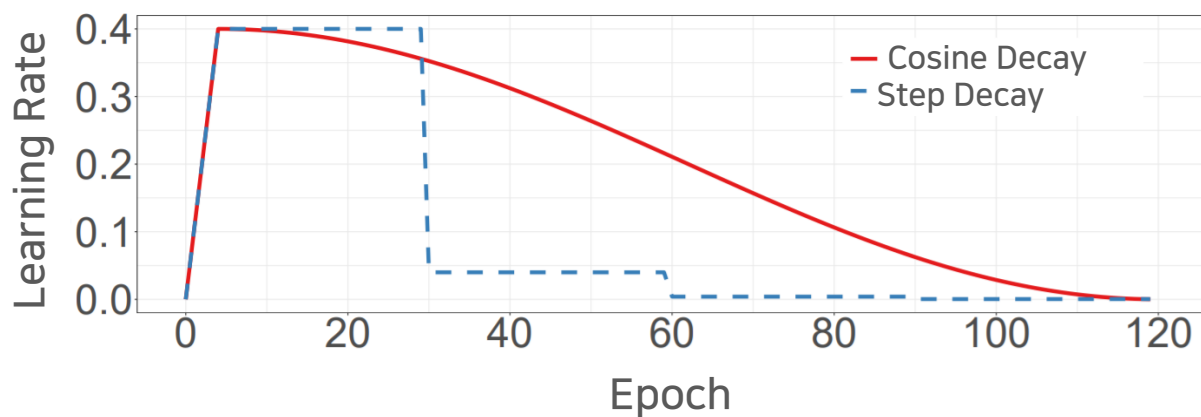
■ : 변경된 부분

(Efficient training 사용)



## Training Refinements ① Cosine Learning Rate Decay

- 학습률 조정(learning rate adjustment) 기법으로 cosine decay를 사용할 수 있습니다.
  - 학습 중반부에서 linear하게 learning rate이 감소하는 특징이 있습니다.



$\eta$ : Learning rate

$T$ : Total number of batches

$t$ : Each batch

$$\eta_t = \frac{1}{2} \left( 1 + \cos \left( \frac{t\pi}{T} \right) \right) \eta$$



## Training Refinements ② Label Smoothing

- 일반화(generalization) 성능을 높이기 위해 레이블(label)을 smoothing합니다.
  - 정답 레이블에 대해서만 100%의 확률을 부여하지 않습니다.

$$q_i = \begin{cases} 1 - \varepsilon & \text{if } i = y, \\ \varepsilon / (K - 1) & \text{otherwise,} \end{cases}$$

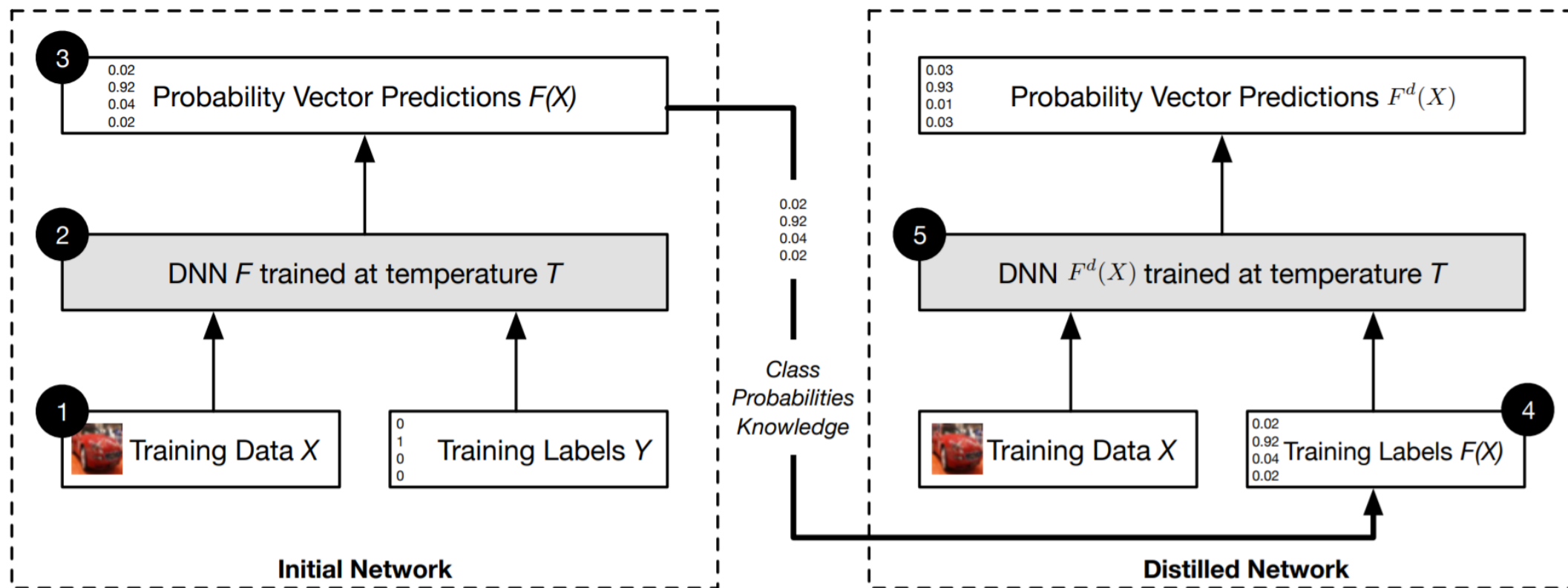
	Cat	Dog	Frog
image 1	0.90	0.05	0.05
image 2	0.90	0.05	0.05
image 3	0.05	0.90	0.05
image 4	0.05	0.05	0.90

[Table] label smoothing 예시 ( $\varepsilon = 0.1$ )

## Training Refinements ③ Knowledge Distillation

- 지식 증류(knowledge distillation) 방법을 사용합니다. ( $z$ 는 *student*,  $r$ 은 *teacher*,  $p$ 는 *real distribution*)

$$\ell(p, \text{softmax}(z)) + T^2 \ell(\text{softmax}(r/T), \text{softmax}(z/T))$$



Distilling the Knowledge in a Neural Network (NIPS 2014)

Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks (S&P 2016)

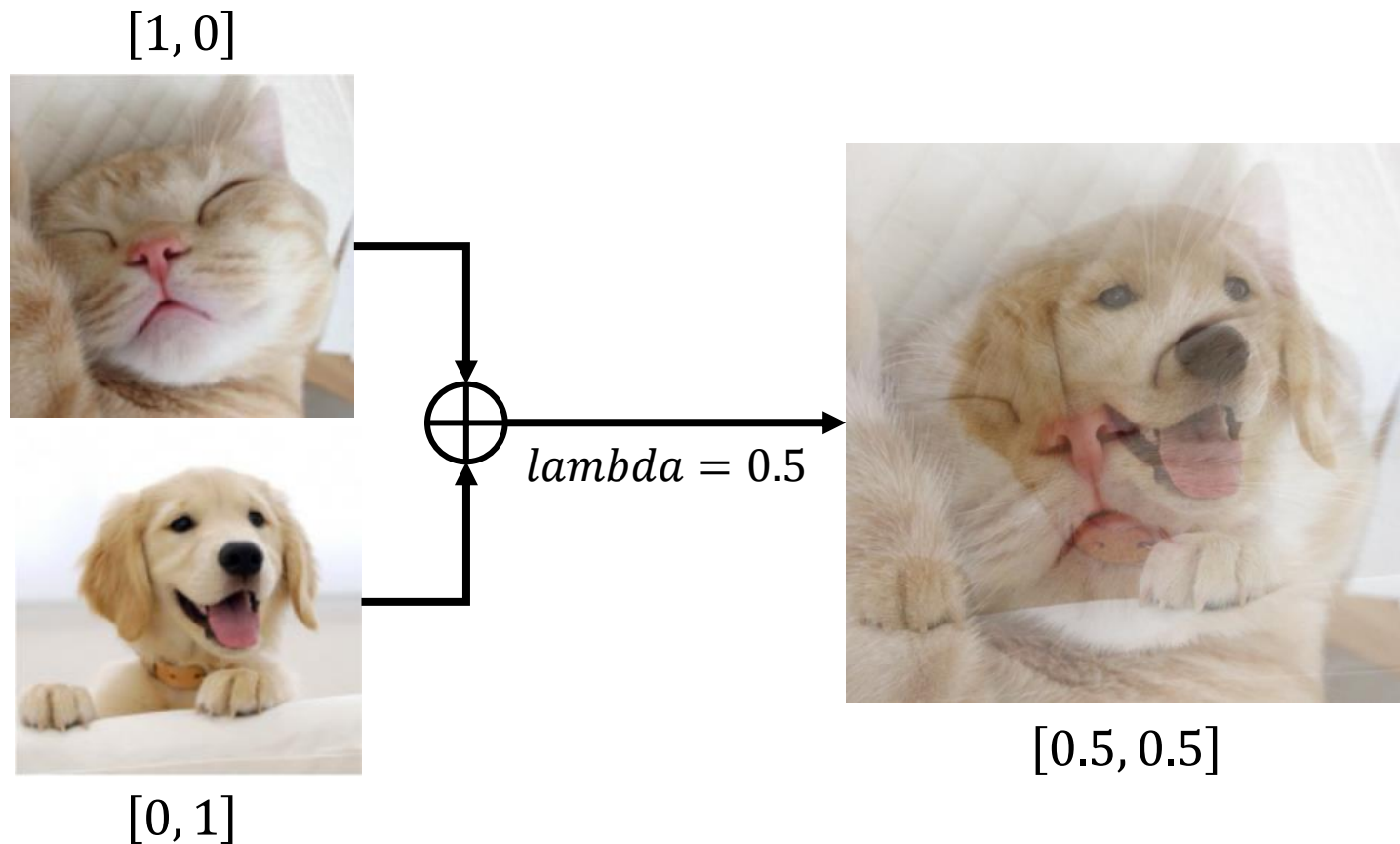
## Training Refinements ④ Mixup Training

- 학습을 진행할 때 랜덤하게 두 개의 샘플  $(x_i, y_i)$ 와  $(x_j, y_j)$ 를 뽑아서  $(\hat{x}, \hat{y})$ 를 만들어 학습에 사용합니다.

$$\hat{x} = \lambda x_i + (1 - \lambda)x_j$$

$$\hat{y} = \lambda y_i + (1 - \lambda)y_j$$

$\lambda \in [0, 1]$ 는  $Beta(\alpha, \alpha)$ 에서 추출합니다.



## Training Refinements 결과 분석

- 지금까지 언급된 테크닉을 모두 활용하여 state-of-the-art 성능을 얻을 수 있습니다.

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.16	93.52	77.50	93.60	71.90	90.53
+ cosine decay	77.91	93.81	78.19	94.06	72.83	91.00
+ label smoothing	78.31	94.09	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.67	94.36	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.15	94.58	<b>78.77</b>	<b>94.39</b>	<b>73.28</b>	<b>91.30</b>
+ distill w/ mixup	<b>79.29</b>	<b>94.63</b>	78.34	94.16	72.51	91.02

Model	Val Top-1 Acc	Val Top-5 Acc	Test Top-1 Acc	Test Top-5 Acc
ResNet-50-D Efficient	56.34	86.87	57.18	87.28
ResNet-50-D Best	<b>56.70</b>	<b>87.33</b>	<b>57.63</b>	<b>87.82</b>

[Table] Results on both the validation set and the test set of MIT Places 365 dataset.

## Transfer Learning: Object Detection

- 일반적인 이미지 클래스 분류 작업 말고도 object detection에서도 성능 향상을 보입니다.

Refinement	Top-1	mAP
B-standard	76.14	77.54
D-efficient	77.16	78.30
+ cosine	77.91	79.23
+ smooth	78.34	80.71
+ distill w/o mixup	78.67	80.96
+ mixup w/o distill	79.16	81.10
+ distill w/ mixup	79.29	<b>81.33</b>

[Table] Faster-RCNN performance with various pretrained base networks evaluated on Pascal VOC.