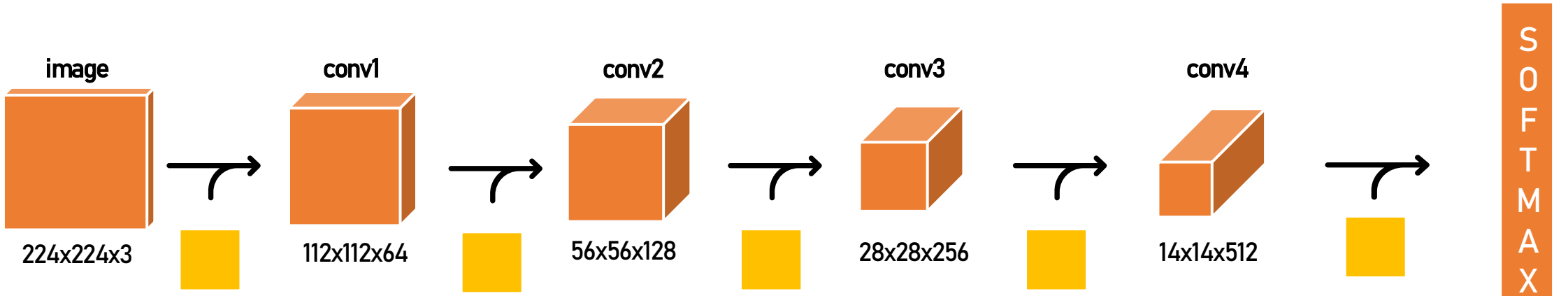
34-layer plain architecture



7x7x64 stride 2 + BN + ReLU
+ 3x3 max pooling stride 2

3x3x64 + BN + ReLU
3x3x64 + BN + ReLU
3x3x64 + BN + ReLU
3x3x64 + BN + ReLU
3x3x64 + BN + ReLU
3x3x64 + BN + ReLU
3x3x128 + conv stride 2

3x3x128 + BN + ReLU
3x3x128 + BN + ReLU
3x3x128 + BN + ReLU
3x3x128 + BN + ReLU
3x3x128 + BN + ReLU
3x3x256 + conv stride 2

3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x256 + BN + ReLU
3x3x512 + conv stride 2

3x3x512 + BN + ReLU
3x3x512 + BN + ReLU
3x3x512 + BN + ReLU
3x3x512 + BN + ReLU
3x3x512 + BN + ReLU
+ Global average pooling
+ Dense(activation = 'softmax')

Residual Net

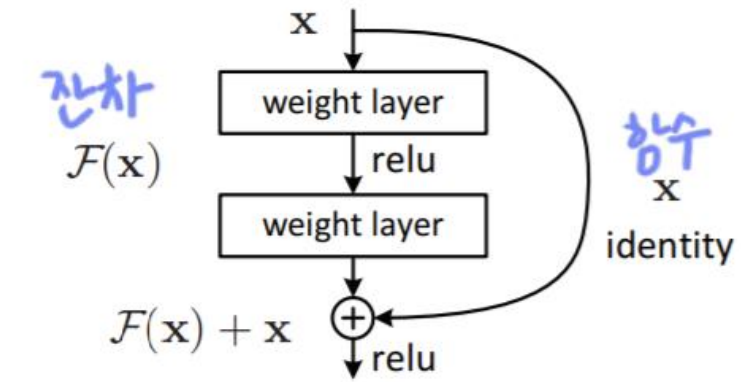
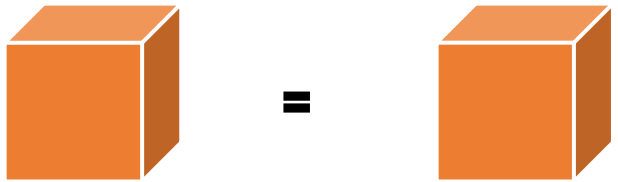


Figure 2. Residual learning: a building block.

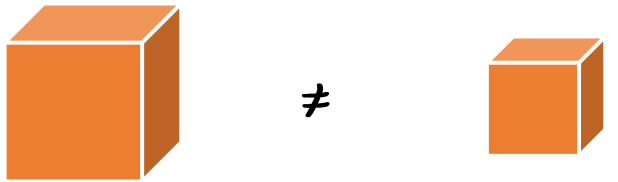
잔차
$\mathcal{F}(\mathbf{x})$

항수
x

차원이 같을때

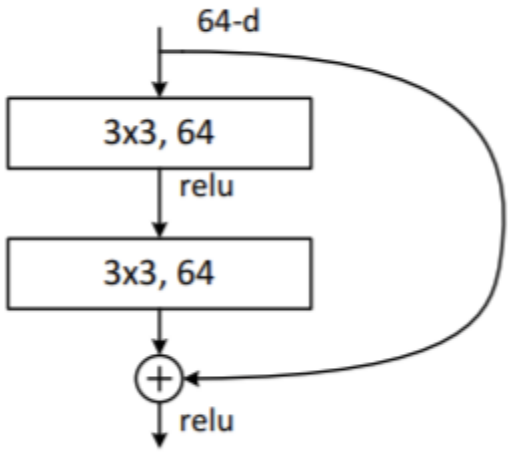


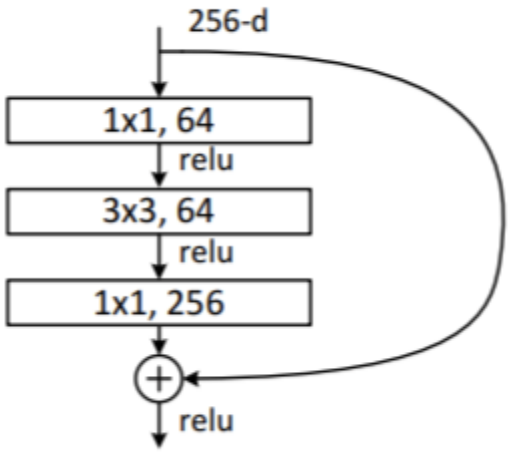
차원이 다를때



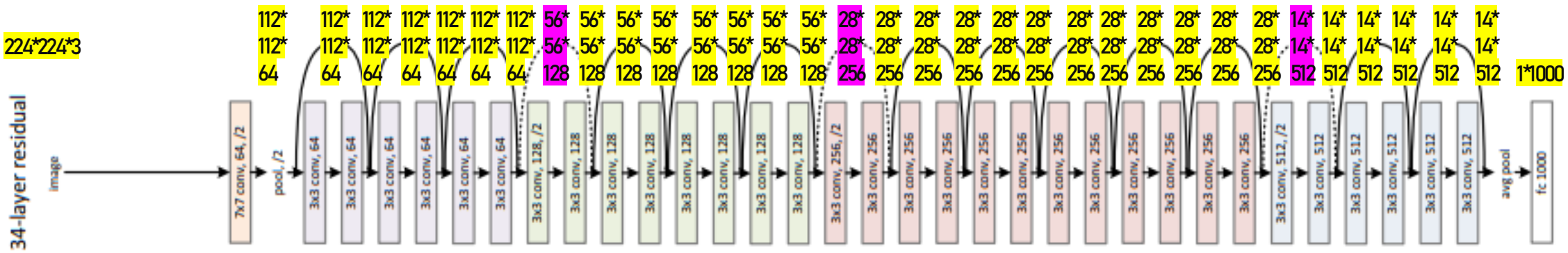
(1) zero padding
(2) convolution
(3) projection

ResBlock_Bottleneck





ResNet 34 - layer



Different dimension Add

verbose = True

56*56*128 + Conv2D(filter_num*2, (3, 3), strides=(2, 2), padding='valid') 112*112*64

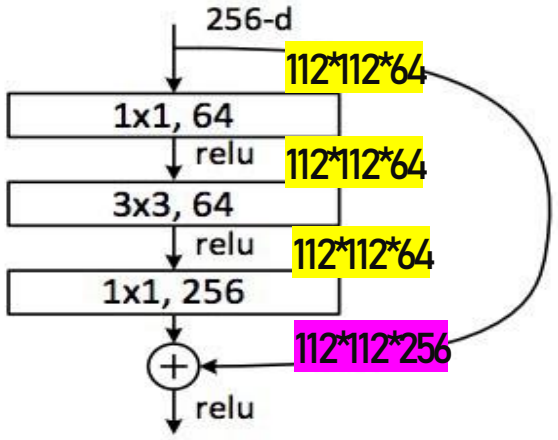
28*28*256 + Conv2D(filter_num*2, (3, 3), strides=(2, 2), padding='valid') 56*56*128

14*14*512 + Conv2D(filter_num*2, (3, 3), strides=(2, 2), padding='valid') 28*28*256

ResNet 50 - layer

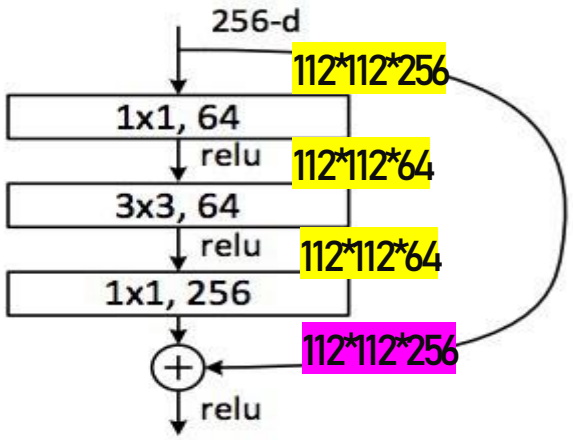
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

verbose = True



112*112*256 + Conv2D(filter_num*4, (3, 3), strides=(2, 2), padding='same') 112*112*64

verbose =False

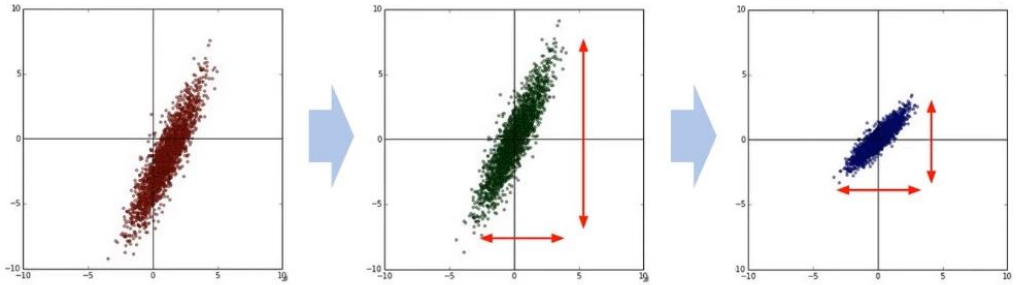


112*112*256 + 112*112*256

Batch Normalization

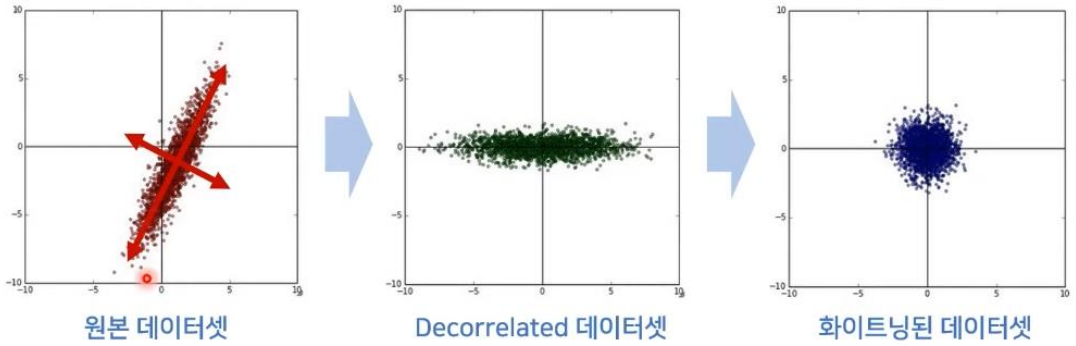
Normalization: 각 차원의 데이터가 동일한 범위 내의 값을 가지도록 만드는 것

[2개의 특성(feature)으로 구성된 데이터셋의 정규화 예시]



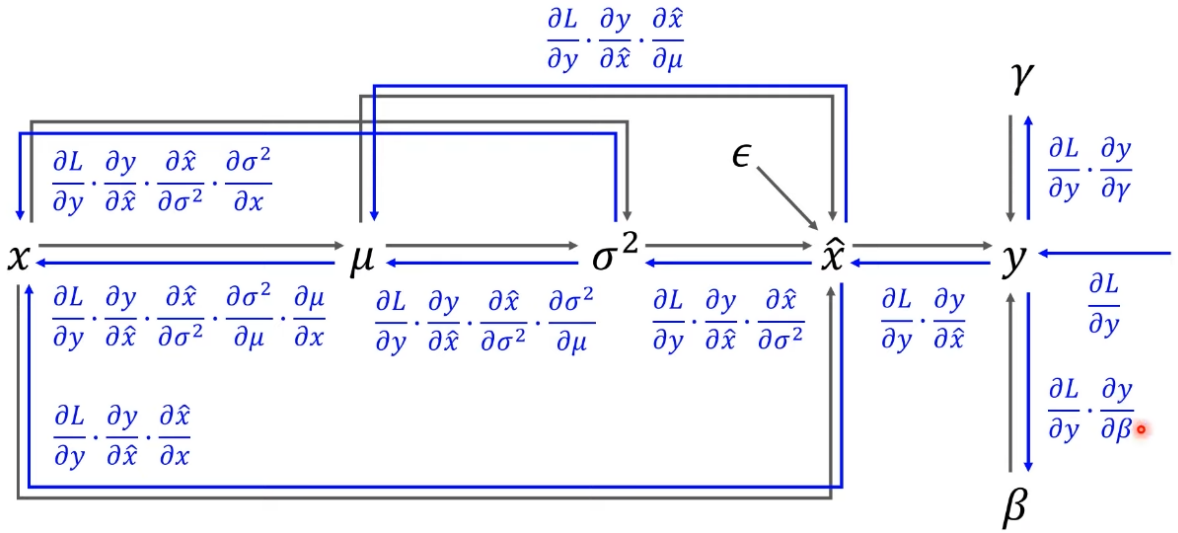
Whitening: 평균이 0이며 공분산이 단위행렬인 정규분포 형태의 데이터로 변환하는 기법
Problem: 이는 covariance matrix의 계산과 inverse의 계산이 필요하기 때문에 계산량이 많을 뿐더러, Whitening은 일부 파라미터들의 영향이 무시된다.

[2개의 특성(feature)으로 구성된 데이터셋의 화이트닝 예시]



Batch Normalization

Batch Normalization : 평균과 분산을 조정하는 과정이 별도의 과정으로 떼어진 것이 아니라, 신경망 안에 포함되어 학습 시 평균과 분산을 조정하는 과정



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

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

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

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

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

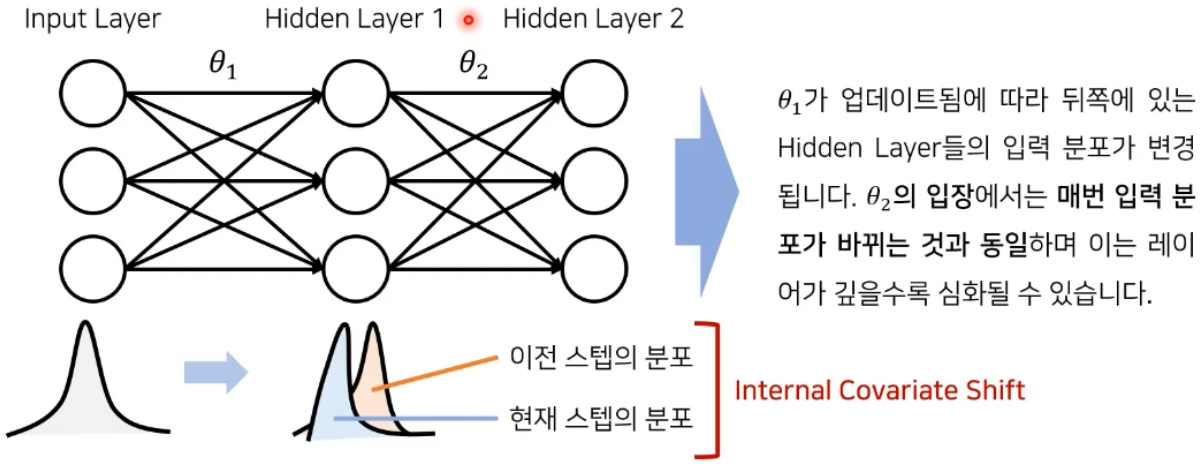
```
def batchnorm_backward(dout, cache):
    dx, dgamma, dbeta = None, None, None
    m = dout.shape[0]
    x, normalized_x, sample_mean, sample_var, gamma, beta, bn_param = cache
    eps = bn_param.get('eps', 1e-5)
    dgamma = np.sum(dout * normalized_x, axis=0)
    dbeta = np.sum(dout, axis=0)
    dxhat = dout * gamma
    dx1 = dxhat / np.sqrt(sample_var + eps)
    dvar = np.sum(dxhat * (-0.5) * (x-sample_mean) * (sample_var + eps) ** (-1.5), axis=0)
    dx2 = dvar * 2 * (x-sample_mean) / m
    dmu = np.sum(dxhat / (-np.sqrt(sample_var + eps)), axis=0)
    dx3 = dmu / m
    dx = dx1 + dx2 + dx3
    return dx, dgamma, dbeta
```

Batch Normalization

Internal Covariance Shift

Covariate Shift(공변량 변화) : 학습 시기와는 다르게 테스트 시기에 입력 데이터의 분포가 변경되는 현상

Internal Covariate Shift : 네트워크 내부에서 **Covariate Shift** 가 일어나면서 입력의 분포가 변하는 현상



θ_1가 업데이트됨에 따라 뒤쪽에 있는 Hidden Layer들의 입력 분포가 변경됩니다. θ_2**의 입장**에서는 **매번 입력 분포가 바뀌는 것과 동일**하며 이는 레이어가 깊을수록 심화될 수 있습니다.