Advanced Convolutional Neural Network

TA Dang-Nha Nguyen
TA Khoa Tho Anh Nguyen

Outline

Review

1. Convolution Layer
2. Pooling Layer
3. Multiple Input – Output
4. Read CNN Architecture

Section 1

Batch Normalization

1. Standard and Normalization
2. Batch Normalization
3. Training CNN
4. Training CNN with Batch Norm

Section 2

Drop Out

1. Why using Dropout
2. Inverted Dropout
3. Apply Dropout

Section 3

Skip Connection

1. Degradation Problem
2. Skip Connection
3. Implement Skip Connection
4. Briefly about Resnet, Densnet

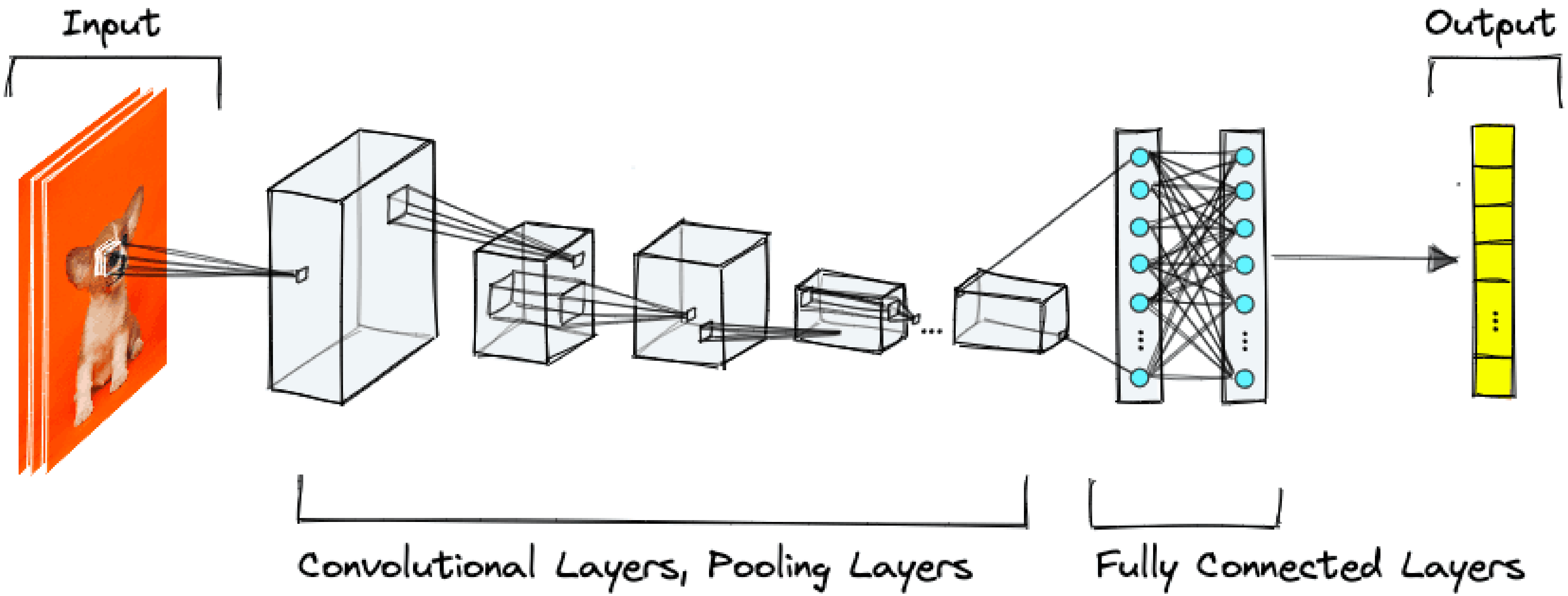
Section 4

Quiz&Feedbacks

Review: CNN

The Building Blocks of a CNN

❖ The building blocks of a CNN



Review

❖ Convolutional Layer

Stride: (S, T)

0	3	1	1
3	1	2	0
3	4	2	3
3	0	0	2

Input: M x N

Padding: (P, Q)

0	0	0	0	0	0
0	0	3	1	1	0
0	3	1	2	0	0
0	3	4	2	3	0
0	3	0	0	2	0
0	0	0	0	0	0

*

1	1	1
1	1	1
0	1	0

Kernel: K x O

1

Bias

=

7	8
15	13

$$\left\lfloor \frac{M + 2P - K}{S} + 1 \right\rfloor \text{x} \left\lfloor \frac{N + 2Q - K}{T} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2P - K}{S} + 1 \right\rfloor$$

Review

❖ Max Pooling

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Input: 6 x 6

Kernel Size: 2
Stride: 2

3	3	3
4	4	4
4	4	4

Output: 3 x 3

❖ Average Pooling

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Input: 6 x 6

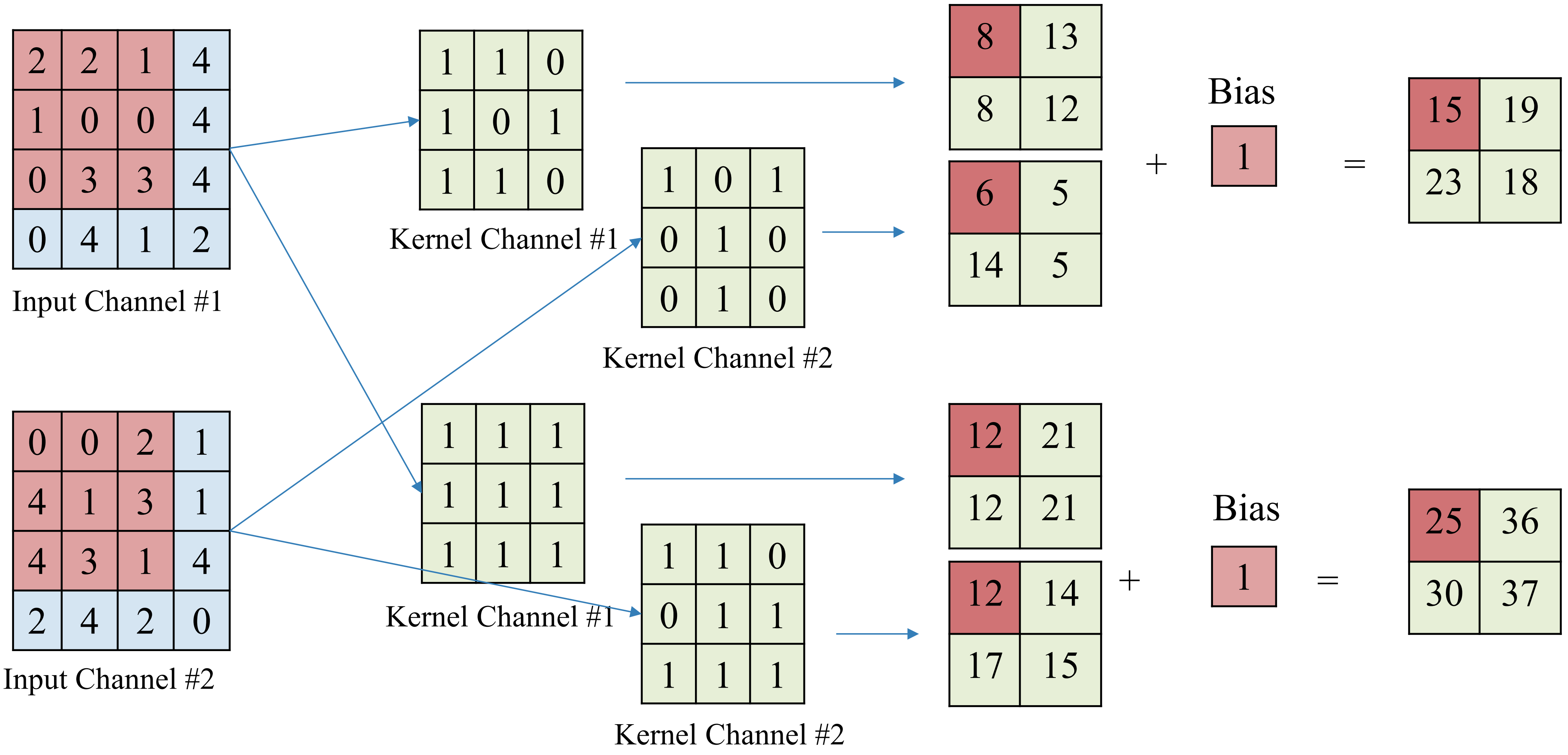
Kernel Size: (3, 2)
Stride: 2

2	1.7	0.8
1.8	1.6	1.3

Output: 2 x 3

Neural Network

❖ Multiple Input - Output



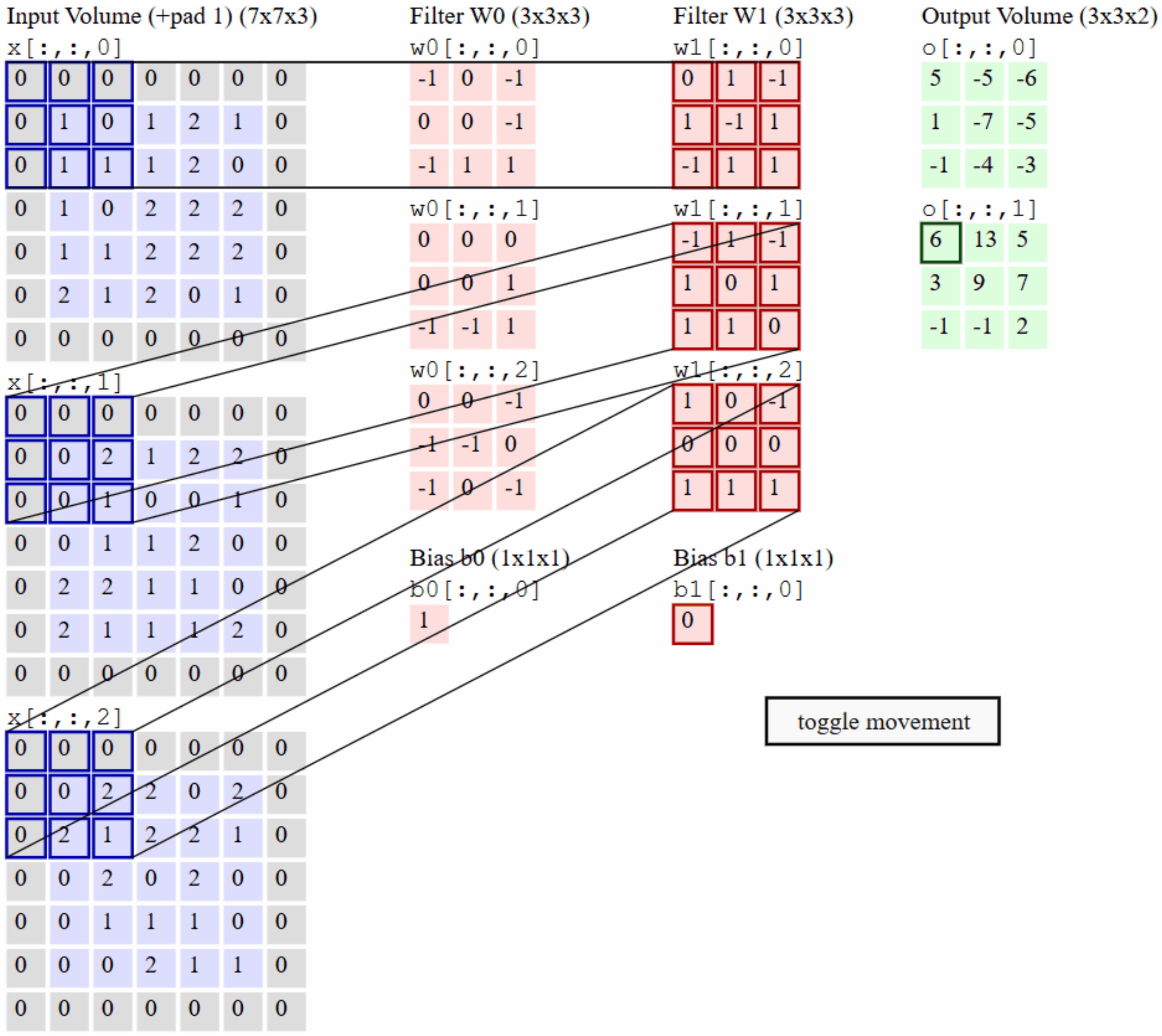
Neural Network

❖ Multiple Input - Output

!

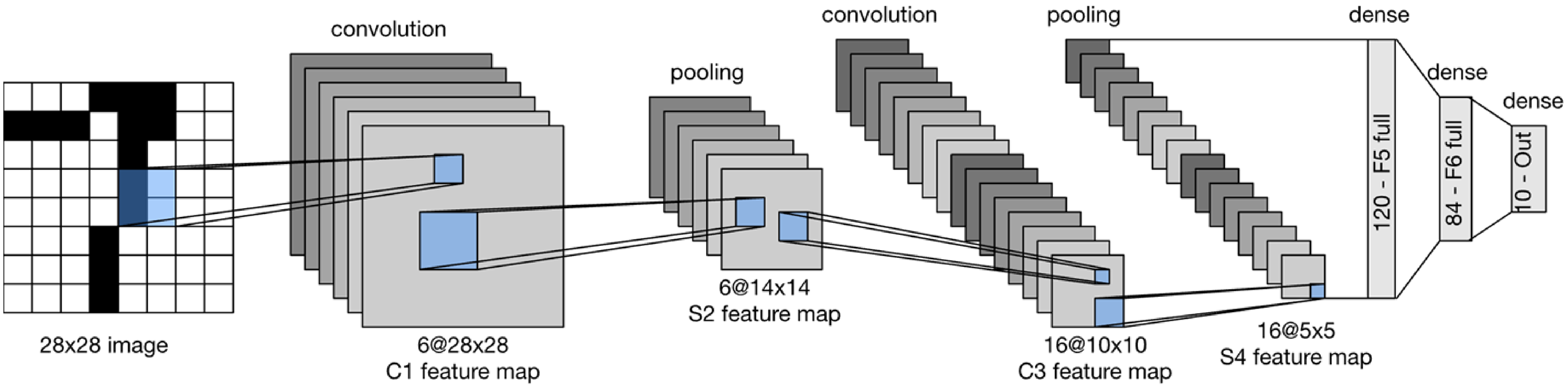
Shape input channel = Depth of Kernel

Shape output channel = Number of Kernel



Neural Network

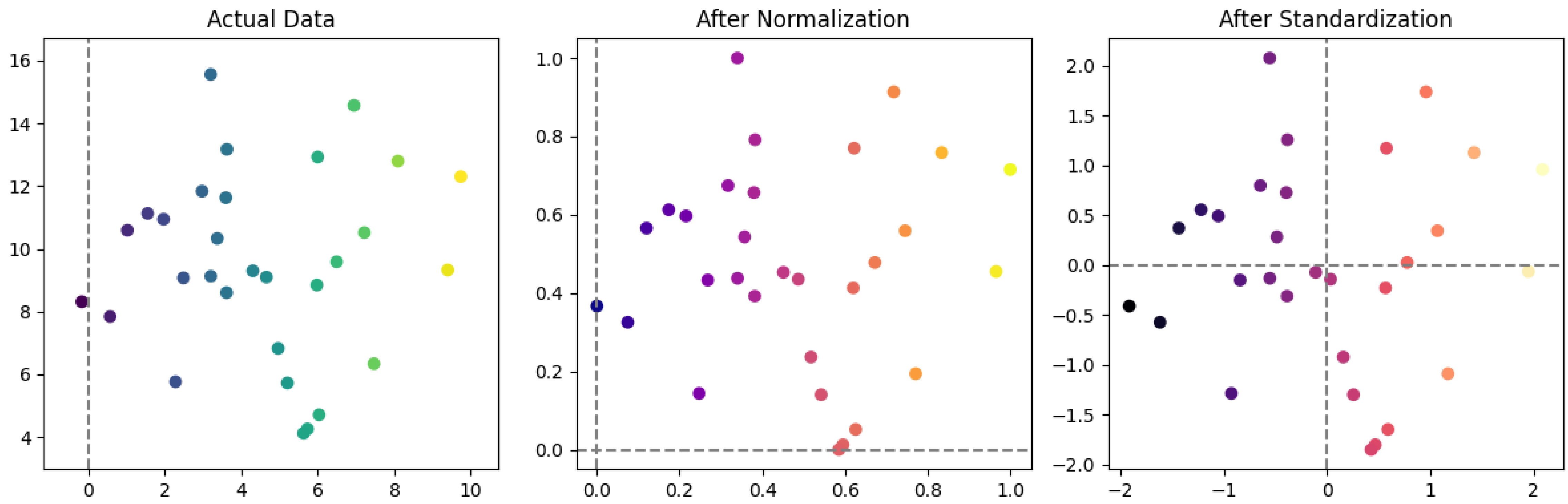
❖ Lenet Model – Simple Implement



Batch Normalization

Batch Normalization

❖ What is Data Normalization?



Batch Normalization

❖ What is Data Normalization?

❖ **Compute mean and variance**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X_i - \mu)^2$$

❖ **Normalize X_i**

$$\widehat{X_i} = \frac{X_i - \mu}{\sqrt{\sigma^2}}$$

$\mu = 2 \qquad \sigma^2 = 1.5 \qquad \sigma = 1.224$

$$\widehat{X_0} = \frac{1 - 2}{1.224}$$

Input	Output
1	- 0.817
1	
2	
4	

Batch Normalization

❖ What is Data Normalization?

❖ **Compute mean and variance**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ **Normalize X_i**

$$\widehat{X_i} = \frac{X_i - \mu}{\sqrt{\sigma^2}}$$

$\mu = 2 \qquad \sigma^2 = 1.5 \qquad \sigma = 1.224$

1
1
2
4

$$\widehat{X_0} = \frac{2 - 2}{1.224}$$

- 0.817
- 0.817
0.000

Batch Normalization

❖ What is Data Normalization?

❖ **Compute mean and variance**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ **Normalize X_i**

$$\widehat{X_i} = \frac{X_i - \mu}{\sqrt{\sigma^2}}$$

$\mu = 2 \qquad \sigma^2 = 1.5 \qquad \sigma = 1.224$

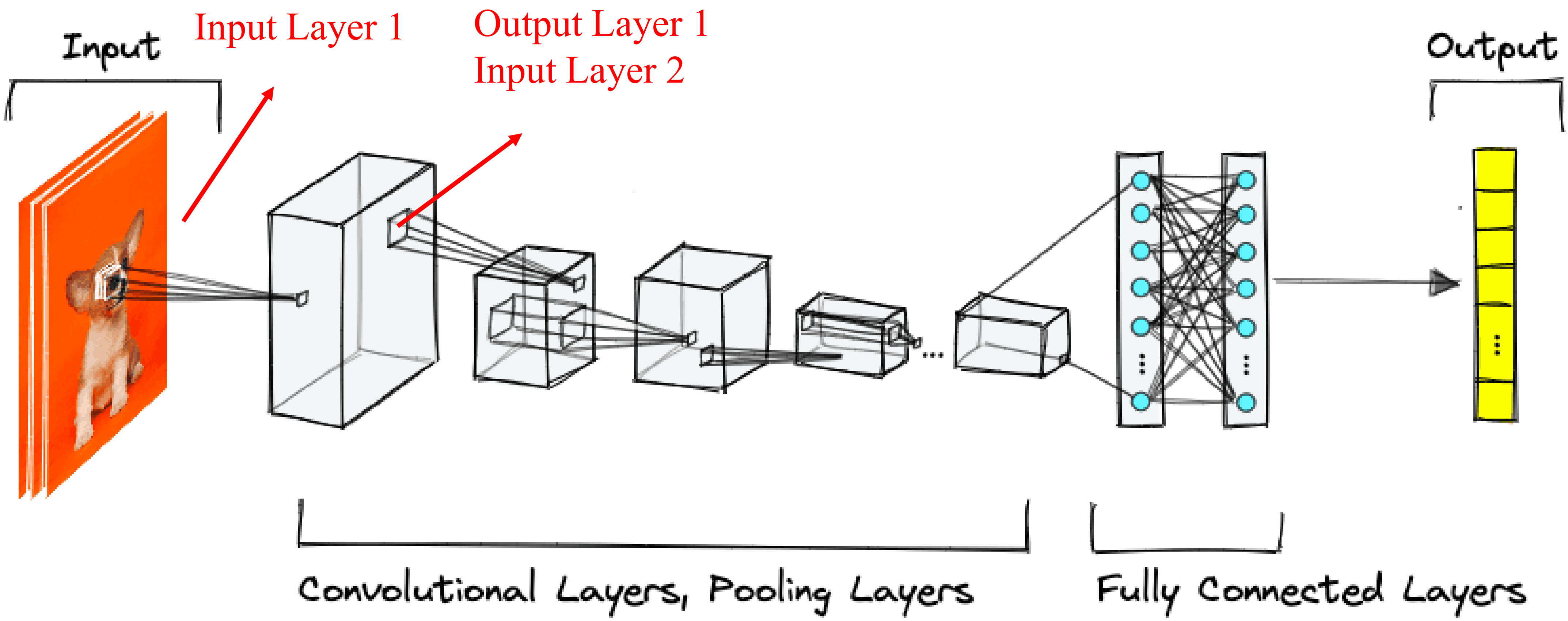
X	$\widehat{X}$
1	- 0.817
1	- 0.817
2	0.000
4	1.634

$$\widehat{X_0} = \frac{4 - 2}{1.224}$$

Batch Normalization

❖ Internal Covariate Shift problem

When layer 1 update params, what is the problem?



Batch Normalization

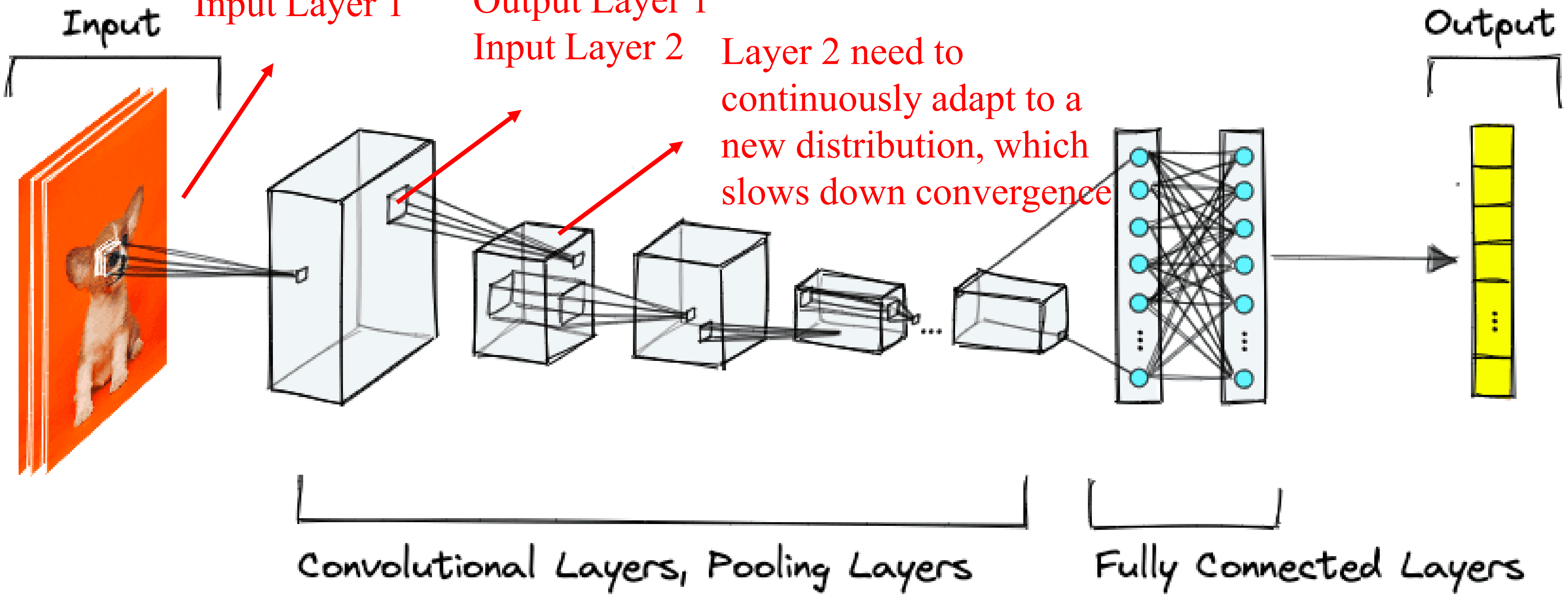
❖ Internal Covariate Shift problem

Deep Learning is a method for learning **pattern from data**

Input Layer 1

Output Layer 1
Input Layer 2

Layer 2 need to continuously adapt to a new distribution, which slows down convergence

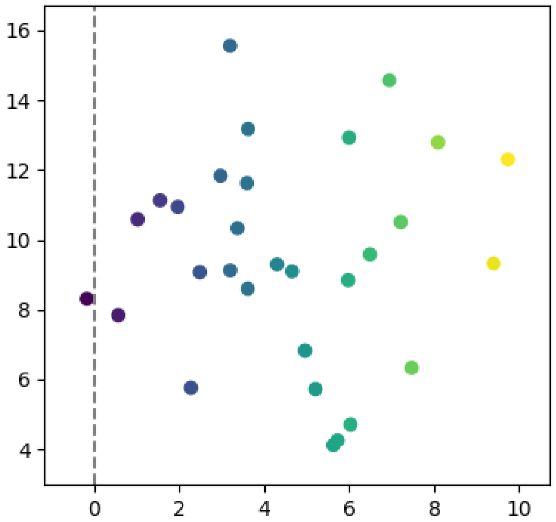


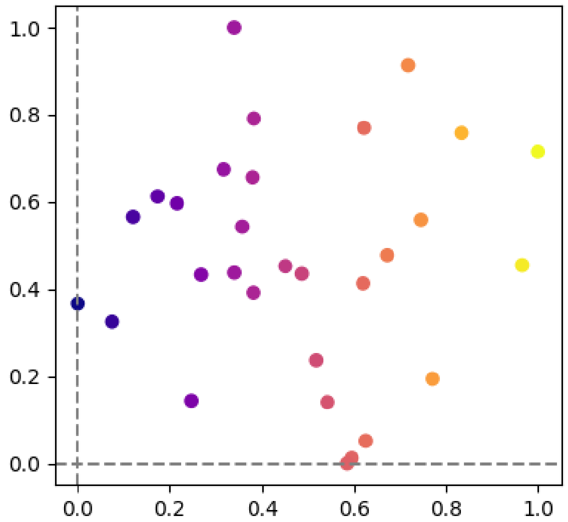
https://arxiv.org/pdf/1502.03167

Batch Normalization

❖ Solving Problem

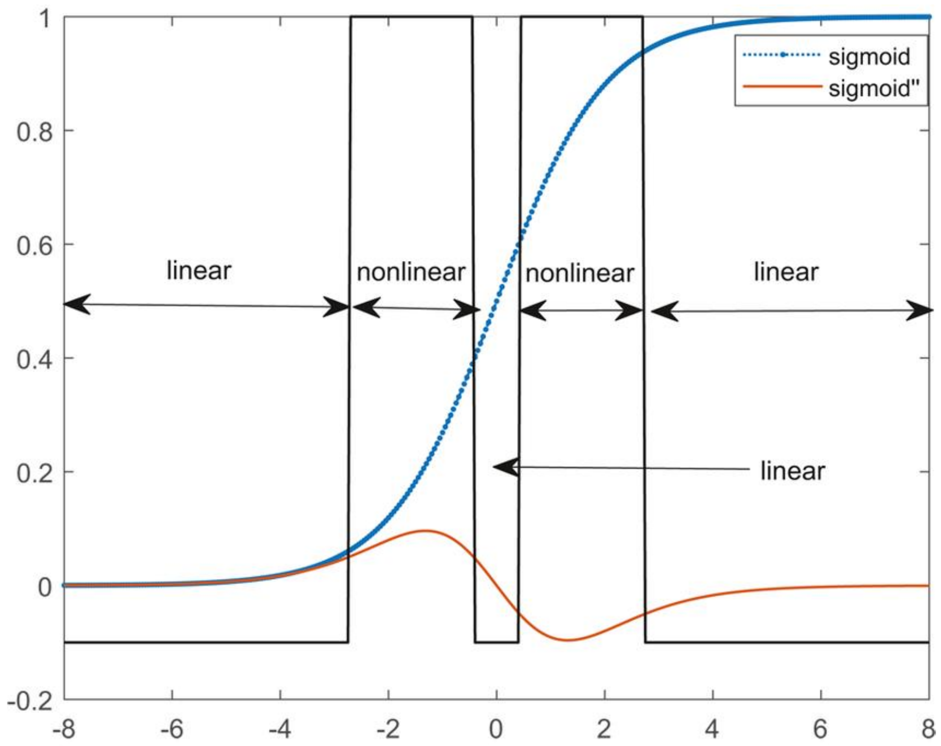
Params Change -> Output Distribution not Change





Mini-Batch Normalization

The Problem of mean = 0 and Variance = 1



Scale and Shift

Batch Normalization

❖ Hand-Ons Calculation

❖ **Get batch data (m: batch size)**

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

γ *and* β are two learning parameters

	C1	C2	C3
X1 -	2	2	1
X2 -	4	1	0
X3 -	0	4	0
X4 -	3	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ **Compute mean and variance**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

	C1	C2	C3
X1 -	2	2	1
X2 -	4	1	0
X3 -	0	4	0
X4 -	3	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

↓

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ **Normalize X_i**

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

	C1	C2	C3
X1 -	2	2	1
X2 -	4	1	0
X3 -	0	4	0
X4 -	3	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

↓

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

$$\hat{X}_0 = \frac{2 - 2.25}{\sqrt{1.479^2 + 1e^{-5}}}$$

2	- 0.169
4	1.183
0	-1.521
3	0.507

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ **Scale and Shift $\hat{X}_i$**

$$Y_i = \gamma \hat{X}_i + \beta$$

γ *and* β are two learning parameters

	C1	C2	C3
X1 -	2	2	1
X2 -	4	1	0
X3 -	0	4	0
X4 -	3	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

↓

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

$$\hat{X}_0 = \frac{2 - 2.25}{\sqrt{1.479^2 + 1e^{-5}}} \qquad Y_0 = 1 * (-0.169) + 0$$

C1	$\hat{X}$	Y
2	- 0.169	- 0.169
4	1.183	1.183
0	-1.521	-1.521
3	0.507	0.507

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ **Scale and Shift $\hat{X}_i$**

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

	C1	C2	C3
X1 -	2	2	1
X2 -	4	1	0
X3 -	0	4	0
X4 -	3	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

↓

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

- 0.169	- 0.447	- 0.153
1.183	- 1.342	- 0.763
-1.521	- 1.342	- 0.763
0.507	0.447	1.677

$\hat{X}_i$

- 0.169	- 0.447	- 0.153
1.183	- 1.342	- 0.763
-1.521	- 1.342	- 0.763
0.507	0.447	1.677

Y_i

Batch Normalization

❖ Hand-Ons Implement - Pytorch

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

γ *and* β are two learning parameters

```
input = torch.randint(5, (4, 3), dtype=torch.float32)
input
```

```
tensor([[2., 2., 1.],
        [4., 1., 0.],
        [0., 4., 0.],
        [3., 3., 4.]])
```

```
batch_norm_layer = nn.BatchNorm1d(num_features=3)
```

```
batch_norm_layer.weight
```

```
Parameter containing:
tensor([1., 1., 1.], requires_grad=True)
```

```
batch_norm_layer.bias
```

```
Parameter containing:
tensor([0., 0., 0.], requires_grad=True)
```

```
output = batch_norm_layer(input)
output
```

```
tensor([[-0.1690, -0.4472, -0.1525],
        [ 1.1832, -1.3416, -0.7625],
        [-1.5213,  1.3416, -0.7625],
        [ 0.5071,  0.4472,  1.6775]], grad_fn=<NativeBatchNormBackward0>)
```

Batch Normalization

❖ Hand-Ons Implement - Pytorch

```
inputs = torch.randint(5, (3, 32, 32), dtype=torch.float32)
```

```
labels = torch.tensor([0, 1, 1])
```

```
model = nn.Sequential(
      nn.Flatten(),
      nn.Linear(32 * 32, 16),
      nn.BatchNorm1d(16),
      nn.ReLU(),
      nn.Linear(16, 2)
    )
```

```
predictions = model(inputs)
```

```
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

```
loss = loss_function(predictions, labels)
```

```
loss.backward()
```

```
optimizer.step()
```

```
Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True)
Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       requires_grad=True)
```

```
Parameter containing:
tensor([-0.0074,  0.0227, -0.0200, -0.0058,  0.0272, -0.0055,  0.0181,  0.0027,
        -0.0221, -0.0286,  0.0310, -0.0004, -0.0141,  0.0302,  0.0015, -0.0182],
       requires_grad=True)
Parameter containing:
tensor([1.0001, 1.0001, 0.9999, 0.9999, 0.9999, 0.9999, 1.0001, 1.0001, 1.0001,
        0.9999, 0.9999, 1.0001, 1.0001, 0.9999, 0.9999, 1.0001],
       requires_grad=True)
```

Batch Normalization

❖ Hand-Ons Implement - Pytorch

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

γ *and* β are two learning parameters

How to compute mean and variance with a sample?

2	2	1

μ_{pop} estimated mean of the studied population

${\sigma^2}_{pop}$ estimated standard-deviation of the studied population

computed using all the (μ_batch , σ_batch) determined during training

Batch Normalization

❖ Inference

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu_{pop}}{\sqrt{{\sigma^2}_{pop} + \epsilon}}$$

ϵ is a very small value (1e-05)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

How to compute mean and variance with a sample?

2	2	1

μ_{pop}	2.25	2.5	1.25
σ_{pop}	1.479	1.118	1.639

$\hat{X}_i$	- 0.169	- 0.447	- 0.153
Y_i	- 0.035	- 0.129	- 0.027

$\gamma = 0.5$

$\beta = 0.05$

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

X1 -

2	2	1	0	0	3
1	4	0	4	3	4

X2 -

0	4	0	0	4	1
1	2	2	1	3	1

X3 -

4	3	2	4	0	4
1	4	2	0	3	4

X4 -

4	1	1	2	2	3
2	0	2	4	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

X1 -

2	2	1	0	0	3
1	4	0	4	3	4

X2 -

0	4	0	0	4	1
1	2	2	1	3	1

X3 -

4	3	2	4	0	4
1	4	2	0	3	4

X4 -

4	1	1	2	2	3
2	0	2	4	3	4

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

$$\mu_c = \frac{1}{N \times H \times W}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{k=1}^{W} F_{ijk}$$

$$\sigma_c = \sqrt{\frac{1}{N \times H \times W}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{k=1}^{W}\left(F_{ijk} - \mu_c\right)^2}$$

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\widehat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\widehat{X}_i$

$$Y_i = \gamma \widehat{X}_i + \beta$$

γ *and* β are two learning parameters

X1 -	2	2	1	0	0	3
	1	4	0	4	3	4
X2 -	0	4	0	0	4	1
	1	2	2	1	3	1
X3 -	4	3	2	4	0	4
	1	4	2	0	3	4
X4 -	4	1	1	2	2	3
	2	0	2	4	3	4

↓

mean	2.1875	1.5625	2.6250
variance	2.0273	1.9961	1.8594

$\epsilon = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \ldots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

mean	2.1875	1.5625	2.6250
variance	2.0273	1.9961	1.8594

$\in = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

X1 -

$$\begin{bmatrix} 2 & 2 \\ 1 & 4 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 \\ 3 & 4 \end{bmatrix}$$

$$\frac{2 - 2.1875}{\sqrt{2.0273 + 1e^{-5}}}$$

$$\frac{1 - 2.1875}{\sqrt{2.0273 + 1e^{-5}}}$$

$$\frac{4 - 2.1875}{\sqrt{2.0273 + 1e^{-5}}}$$

$\hat{X}_1$ -

$$\begin{bmatrix} -0.13 & -0.13 \\ -0.40 & 1.27 \end{bmatrix} \quad \begin{bmatrix} & \\ & \end{bmatrix} \quad \begin{bmatrix} & \\ & \end{bmatrix}$$

Batch Normalization

❖ Hand-Ons Calculation

- Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

- Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

- Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

- Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

mean	2.1875	1.5625	2.6250
variance	2.0273	1.9961	1.8594

$\in = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

X1 -

2	2	1	0	0	3
1	4	0	4	3	4

-0.40	-1.11
-1.11	1.72

$\hat{X}_1$ -

-0.13	-0.13	-1.93	0.28
-0.83	1.27	0.28	1.00

Batch Normalization

❖ Hand-Ons Calculation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma \hat{X}_i + \beta$$

***γ and β* are two learning parameters**

mean	2.1875	1.5625	2.6250
variance	2.0273	1.9961	1.8594

$\in = 1e^{-5}$

$\gamma = 1$

$\beta = 0$

X Shape (4, 3, 2, 2)

X1 -

2	2
1	4

1	0
0	4

0	3
3	4

$\hat{X}_1$ -

-0.13	-0.13
-0.83	1.27

-0.40	-1.11
-1.11	1.72

-1.93	0.28
0.28	1.00

Y_1 -

-0.13	-0.13
-0.83	1.27

-0.40	-1.11
-1.11	1.72

-1.93	0.28
0.28	1.00

Batch Normalization

❖ Hand-Ons - Implementation

❖ Get batch data (m: batch size)

$$X = \{X_1, X_2, \dots, X_m\}$$

❖ Compute mean and variance

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)^2$$

❖ Normalize X_i

$$\hat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a very small value (1e-5)

❖ Scale and Shift $\hat{X}_i$

$$Y_i = \gamma\hat{X}_i + \beta$$

γ *and* β are two learning parameters

```
input = torch.randint(5, (4, 3, 2, 2), dtype=torch.float32)
input
```

```
mean_per_channel = torch.mean(input, dim=(0, 2, 3))
```

```
var_per_channel= torch.var(input, dim=(0, 2, 3), correction=0)
```

```
mean_reshaped = mean_per_channel.view(1, -1, 1, 1)
var_reshaped = var_per_channel.view(1, -1, 1, 1)

x_normalized = (input - mean_reshaped) / torch.sqrt(var_reshaped + eps)
x_normalized
```

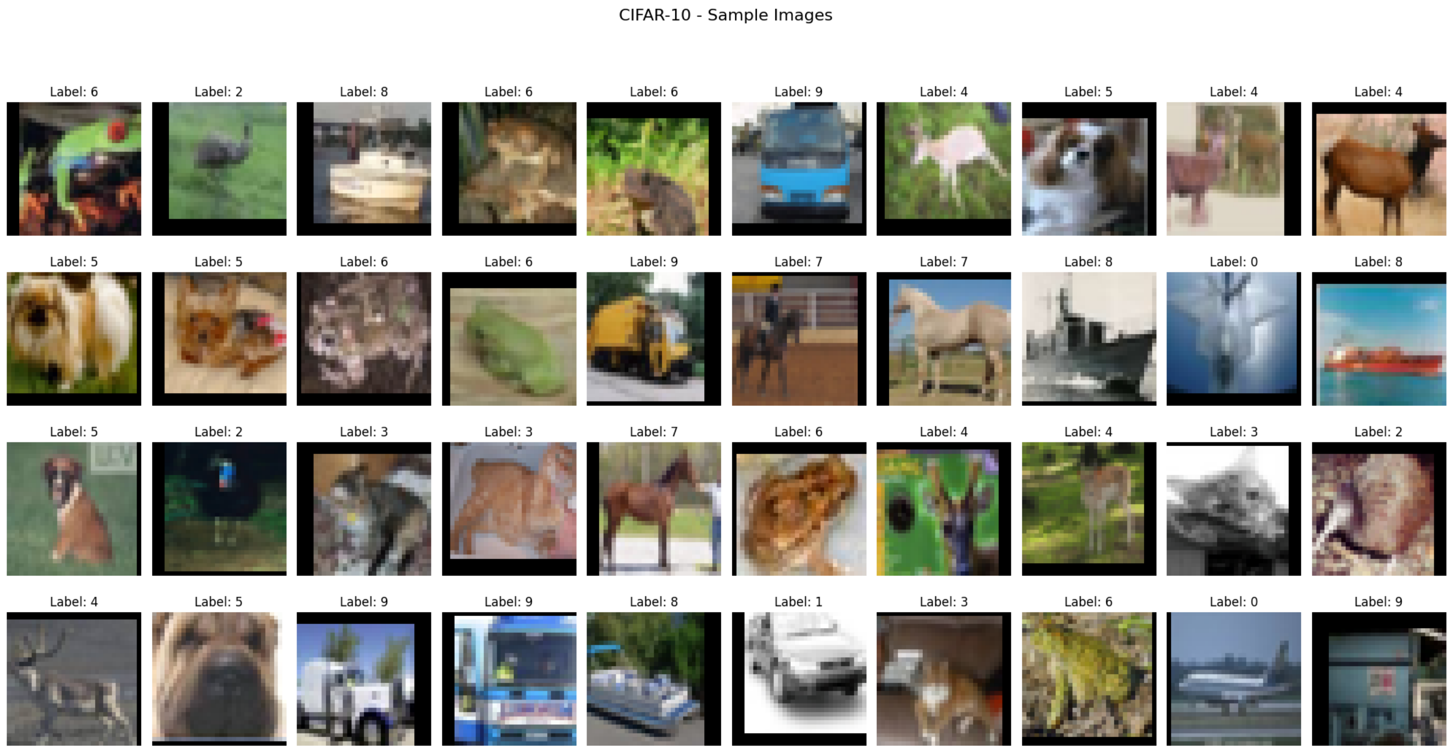
```
batch_norm_layer = nn.BatchNorm2d(num_features=3)
```

Batch Normalization

❖ Cifar10 Training

❖ Cifar10 dataset

- ➢ Images Train: 40.000
- ➢ Images Test: 10.000
- ➢ Class: 10
- ➢ Image Size: 32 x 32



Batch Normalization

❖ Cifar10 Training

Simple Processing

```python
# CIFAR-10 Dataset
cifar_train = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=cifar_transform
)
cifar_test = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=cifar_test_transform
)

# Split CIFAR-10 training into train/val
cifar_train_size = int(0.8 * len(cifar_train))
cifar_val_size = len(cifar_train) - cifar_train_size
cifar_train, cifar_val = torch.utils.data.random_split(
    cifar_train, [cifar_train_size, cifar_val_size]
)
```

```python
cifar_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                         (0.2023, 0.1994, 0.2010))
])

cifar_test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                         (0.2023, 0.1994, 0.2010))
])
```

```python
# Create data loaders
batch_size = 64

cifar_train_loader = DataLoader(cifar_train, batch_size=batch_size,
                                shuffle=True, num_workers=2)
cifar_val_loader = DataLoader(cifar_val, batch_size=batch_size,
                              shuffle=False, num_workers=2)
cifar_test_loader = DataLoader(cifar_test, batch_size=batch_size,
                               shuffle=False, num_workers=2)
```

Batch Normalization

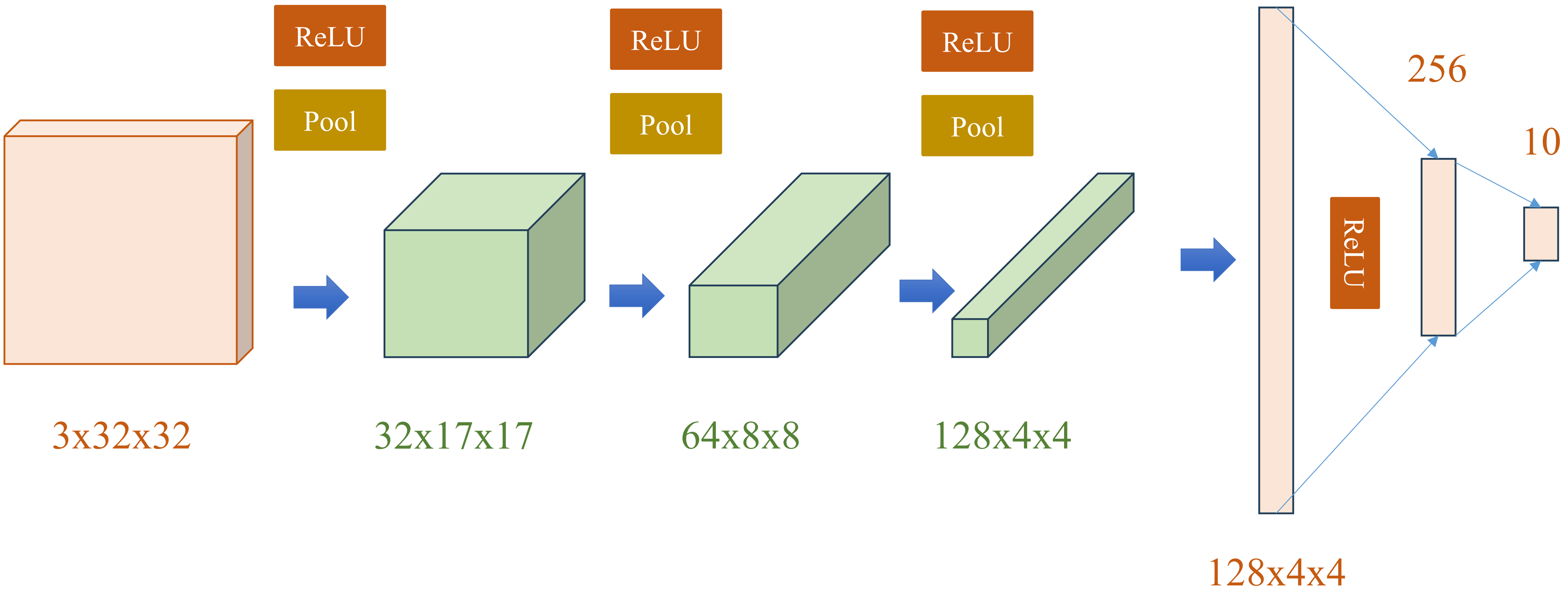
❖ Cifar10 Training

```python
class CIFAR10Net(nn.Module):
    def __init__(self):
        super(CIFAR10Net, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x
```

Based Model

Batch Normalization

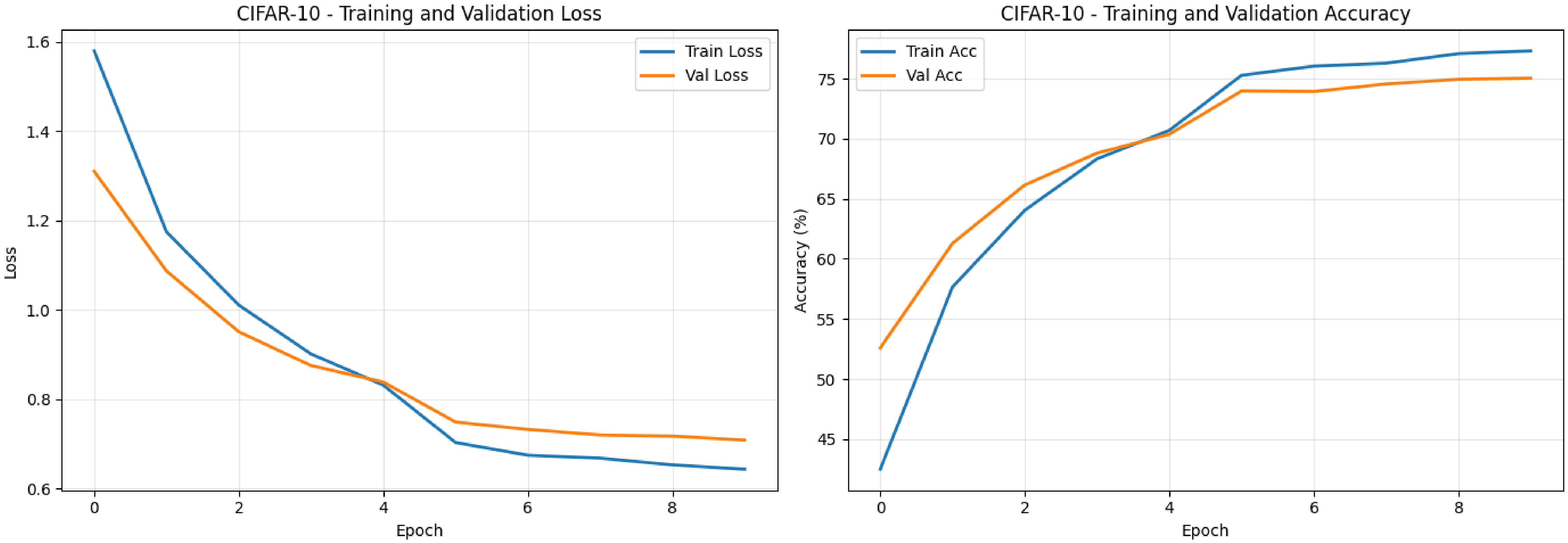
❖ Cifar10 Training



Based Model

Batch Normalization

❖ Cifar10 Training



Batch Normalization

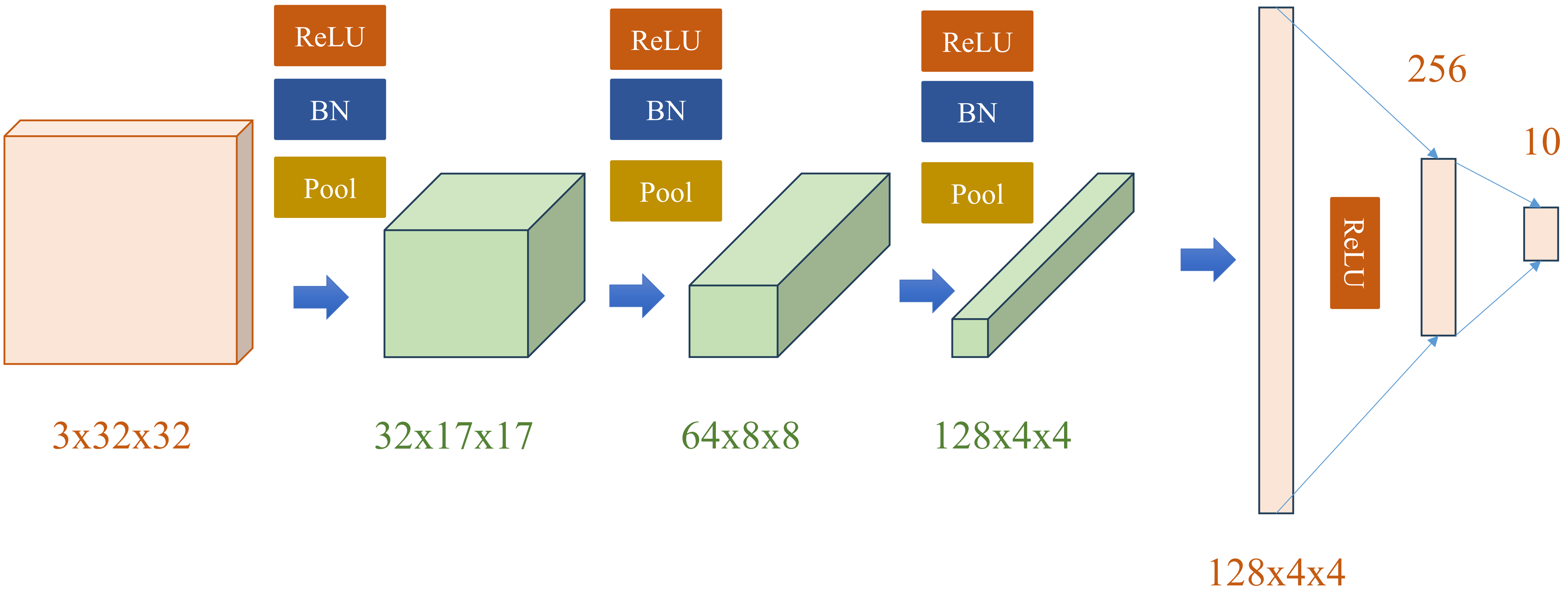
❖ Cifar10 Training

```python
class CIFAR10Net_BN(nn.Module):
    def __init__(self):
        super(CIFAR10Net_BN, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2)
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x
```

Based Model + Batch Normalization

Batch Normalization

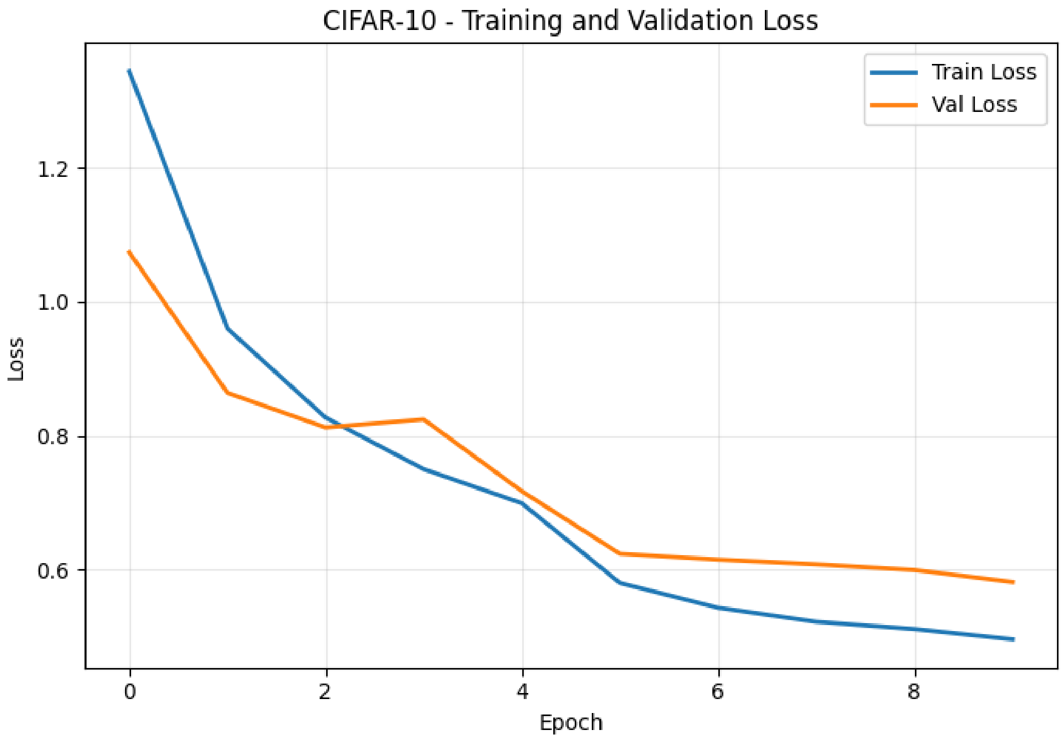
❖ Cifar10 Training

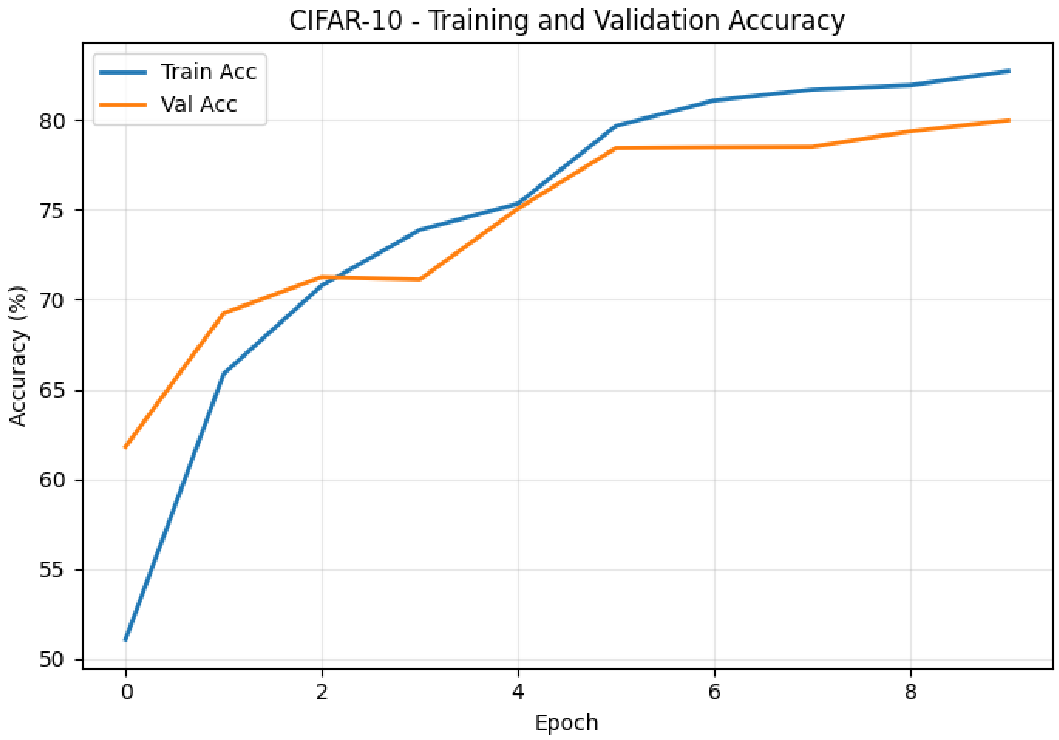


Based Model + Batch Normalization

Batch Normalization

❖ Cifar10 Training



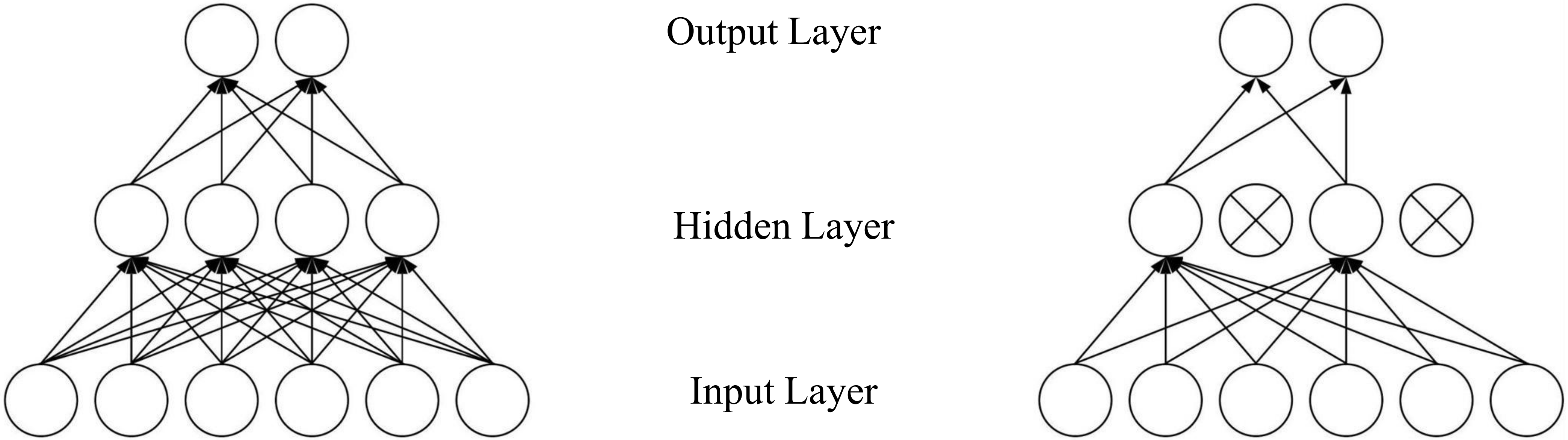


Dropout

Drop Out

❖ Dropout

- ❖ Removing units at random during the forward pass and putting them all back during test
- ❖ Probability of an element to be zeroed (P)



Without Dropout

With Dropout

Drop Out

❖ Inverted Dropout

Drop Rate: P

$$y_{output} = \begin{cases} 0 \; while \; p \\ \frac{x}{1-p} \; while \; 1-p \end{cases}$$

Mask Drop: $m \sim Bernoulli(1-p)$

Output value: $y = x \, . m$

Expected value: $\mathbb{E}[y] = x\mathbb{E}.[m]$

$$\mathbb{E}[\boldsymbol{X}] = \sum_{\boldsymbol{x}} \boldsymbol{x}.\boldsymbol{P}(\boldsymbol{X} = \boldsymbol{x})$$

The **expectation** $\mathbb{E}$ (or expected value, mean) is the long-run average outcome of a random event, calculated as a weighted average of all possible outcomes, where each outcome is multiplied by its probability.

Drop Out

❖ Dropout Implement

```
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(3 * 2, 5),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(5, 2)
)
```

```
for p in model.parameters():
    print(p)
```

```
Parameter containing:
tensor([[ 0.0339,  0.2694, -0.0702, -0.1022, -0.0501, -0.1658],
        [-0.3597, -0.1011,  0.3167,  0.2188, -0.1090,  0.2430],
        [ 0.3705,  0.2145, -0.3251,  0.0081, -0.0650,  0.0284],
        [ 0.3639, -0.1841,  0.0548,  0.3922,  0.1981,  0.0232],
        [ 0.3353,  0.0434, -0.1930, -0.0349, -0.3071, -0.3159]],
       requires_grad=True)
Parameter containing:
tensor([-0.1539, -0.1738, -0.4064,  0.3608,  0.2249], requires_grad=True)
Parameter containing:
tensor([[-0.1894, -0.3698, -0.1071,  0.3176, -0.0480],
        [ 0.0826,  0.3068,  0.1251,  0.4449, -0.3024]], requires_grad=True)
Parameter containing:
tensor([ 0.4070, -0.1317], requires_grad=True)
```

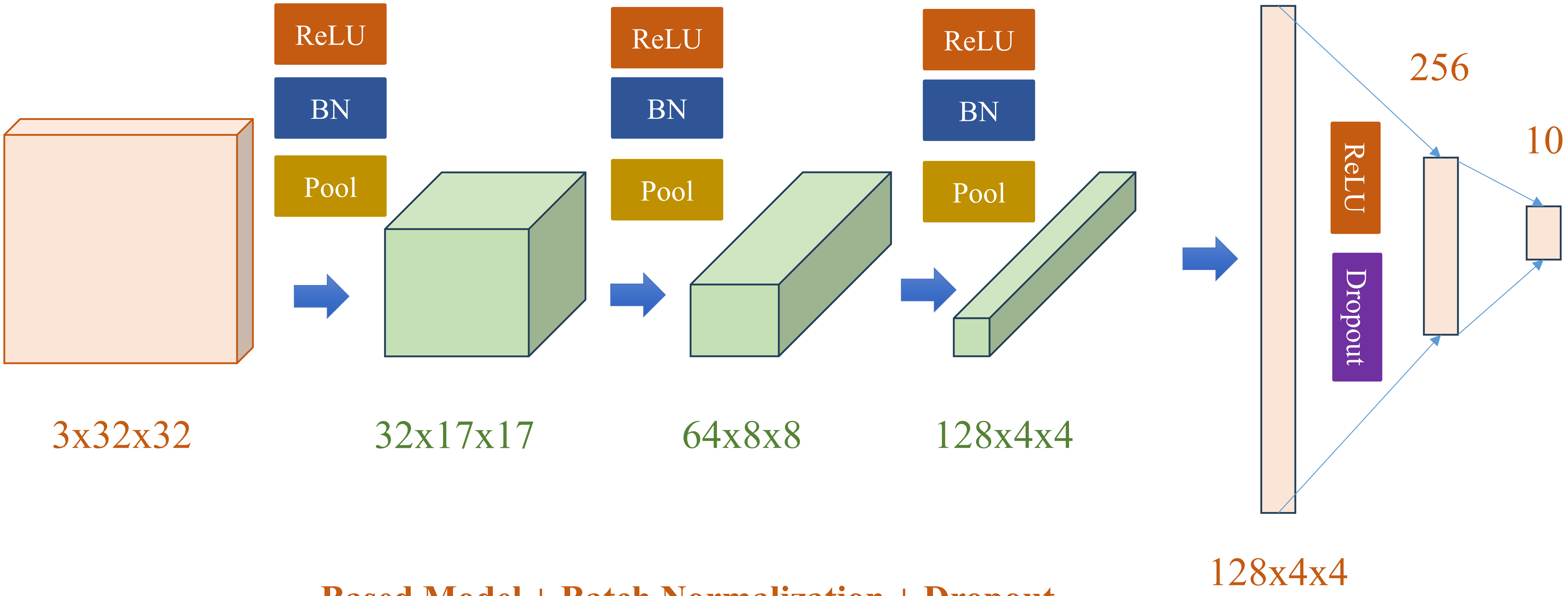
```
for p in model.parameters():
    print(p)
```

```
Parameter containing:
tensor([[ 0.0339,  0.2694, -0.0702, -0.1022, -0.0501, -0.1658],
        [-0.3587, -0.1001,  0.3177,  0.2198, -0.1080,  0.2440],
        [ 0.3695,  0.2155, -0.3241,  0.0071, -0.0640,  0.0294],
        [ 0.3629, -0.1831,  0.0558,  0.3912,  0.1991,  0.0242],
        [ 0.3363,  0.0444, -0.1930, -0.0339, -0.3071, -0.3149]],
       requires_grad=True)
Parameter containing:
tensor([-0.1539, -0.1728, -0.4074,  0.3598,  0.2259], requires_grad=True)
Parameter containing:
tensor([[-0.1894, -0.3688, -0.1061,  0.3186, -0.0470],
        [ 0.0826,  0.3058,  0.1241,  0.4439, -0.3034]], requires_grad=True)
Parameter containing:
tensor([ 0.4080, -0.1327], requires_grad=True)
```

Drop Out

❖ Dropout Implement

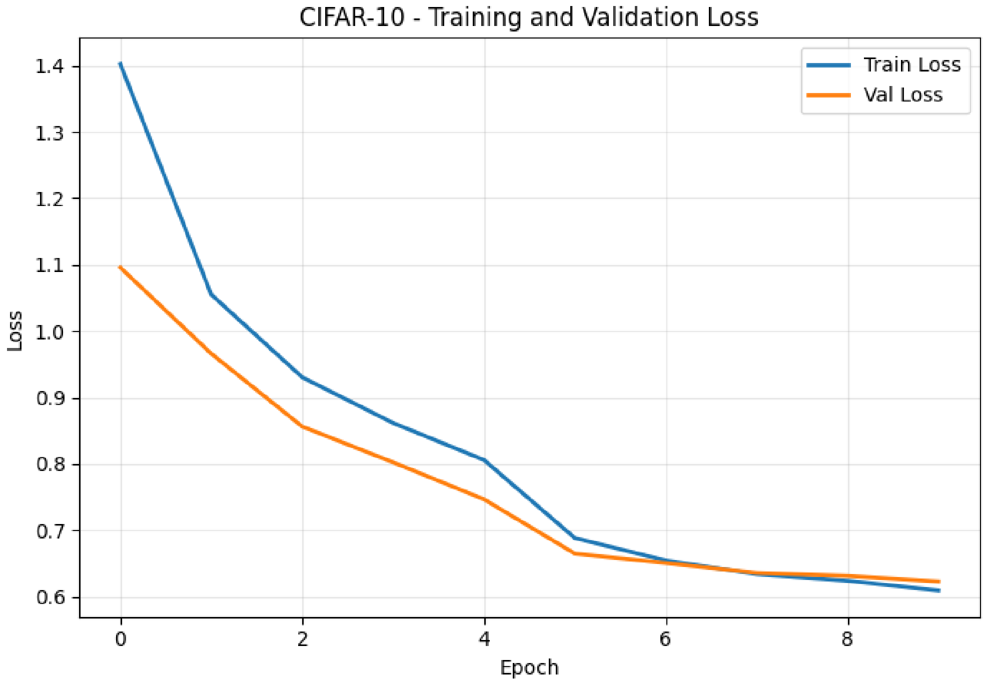
```
self.fc_layers = nn.Sequential(
    nn.Linear(128 * 4 * 4, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, 10)
)
```

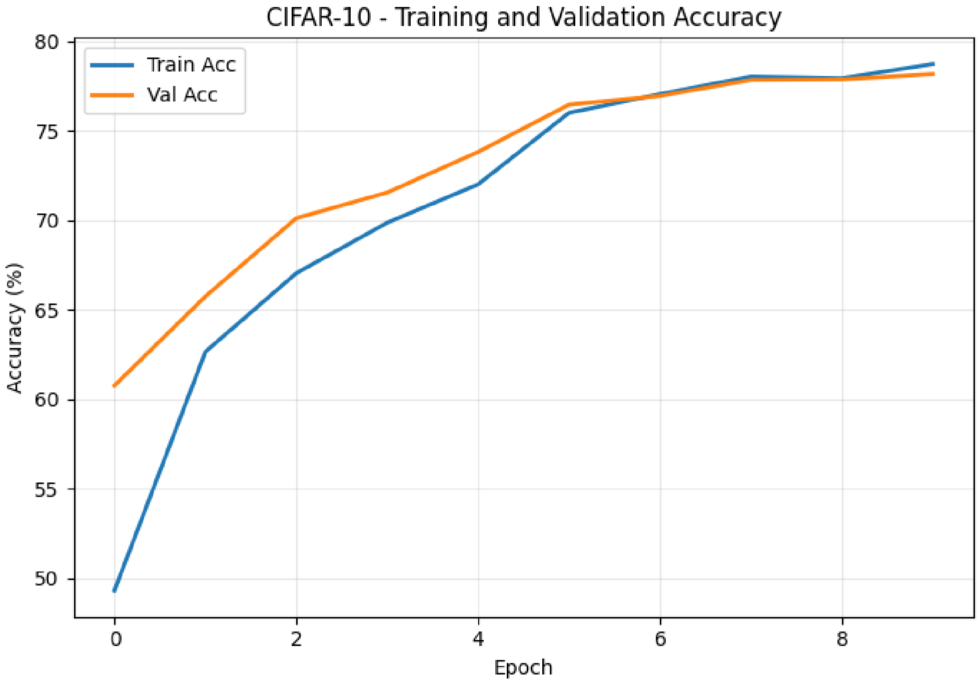


Based Model + Batch Normalization + Dropout

Batch Normalization

❖ Cifar10 Training





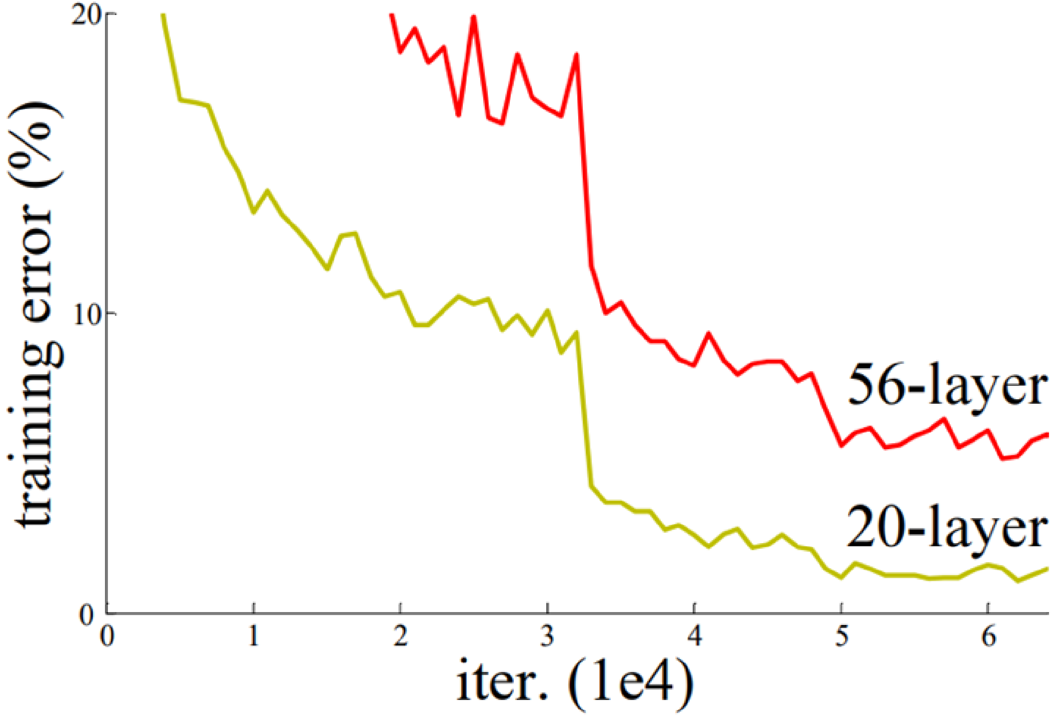

QUIZ
TIME

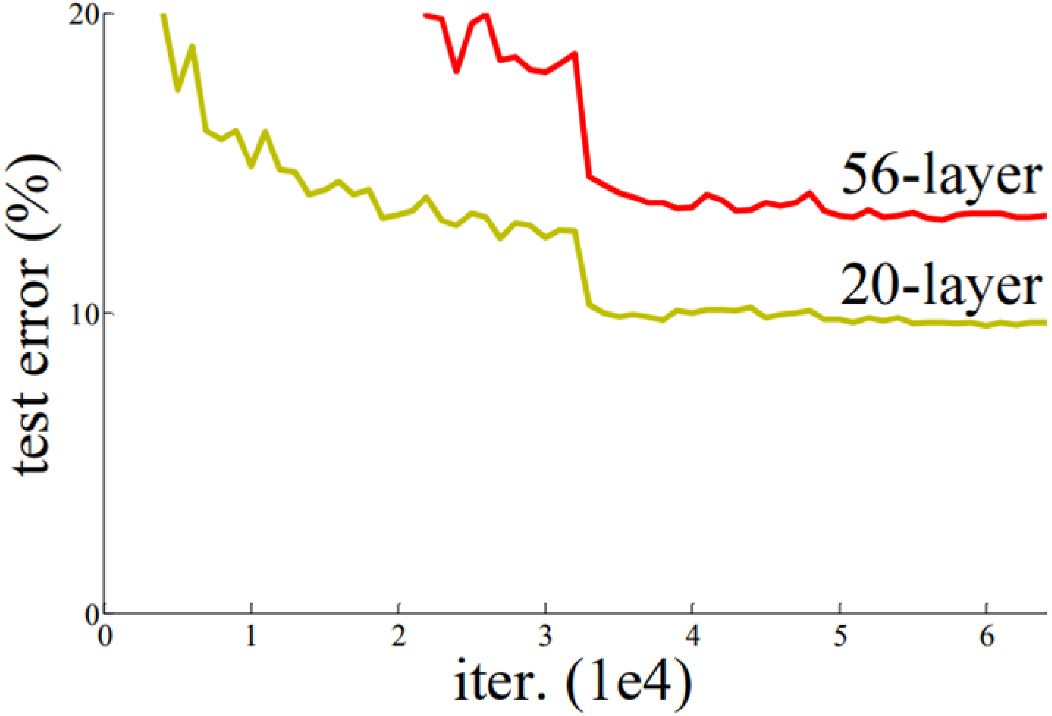
Skip Connection

Skip Connection

❖ Degradation Problem

The deeper model doesn't perform as well as the shallow one

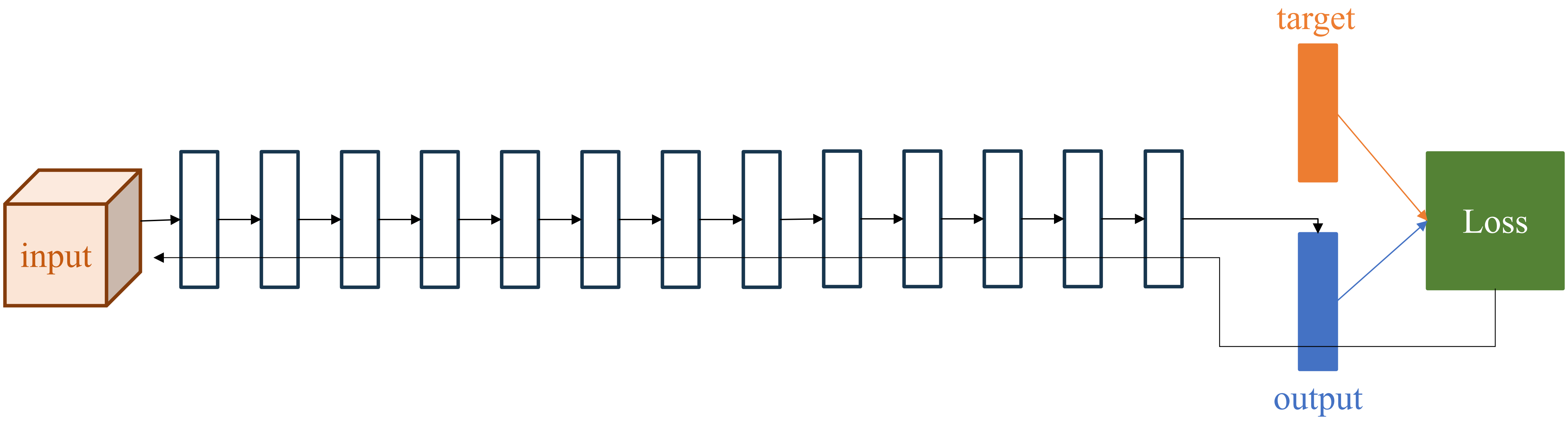




https://arxiv.org/pdf/1512.03385.pdf

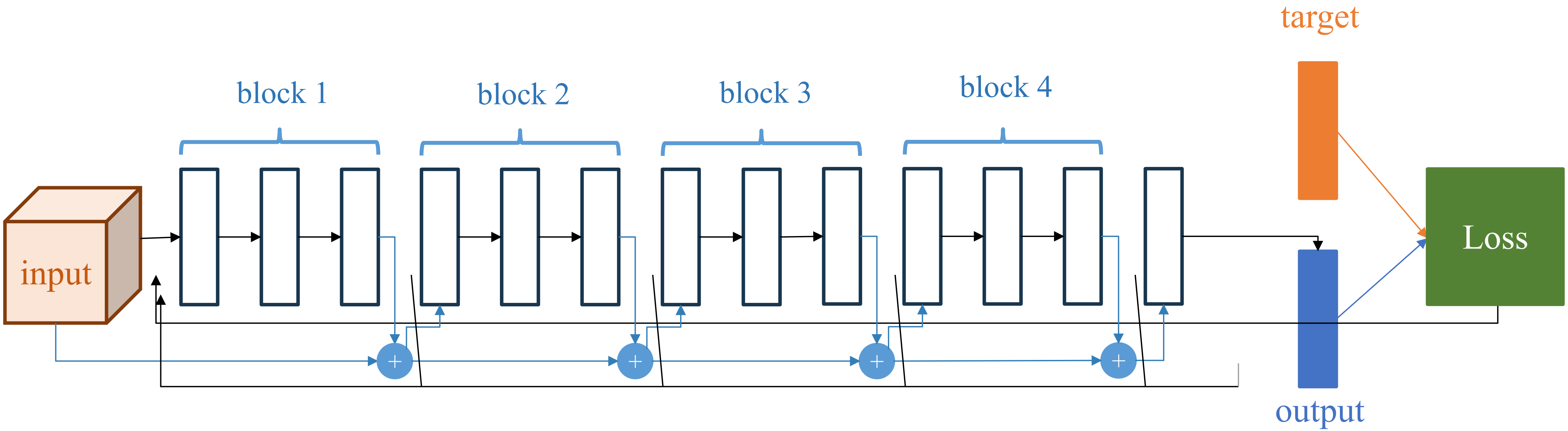
Skip Connection

❖ Degradation Problem



Skip Connection

❖ Skipp Connection



Skip Connection

❖ Example

Input		
140	0	5
8	47	175
133	203	217

Conv2d		
-12.1794	-2.7855	6.7117
-14.4943	-105.878	-17.4401
-67.1033	-84.1202	29.6049

ReLU		
0	0	6.7117
0	0	0
0	0	29.6049

Conv2d		
0.2087	1.9920	-1.4108
0.2087	2.4680	4.0437
0.2087	8.0747	-6.9349

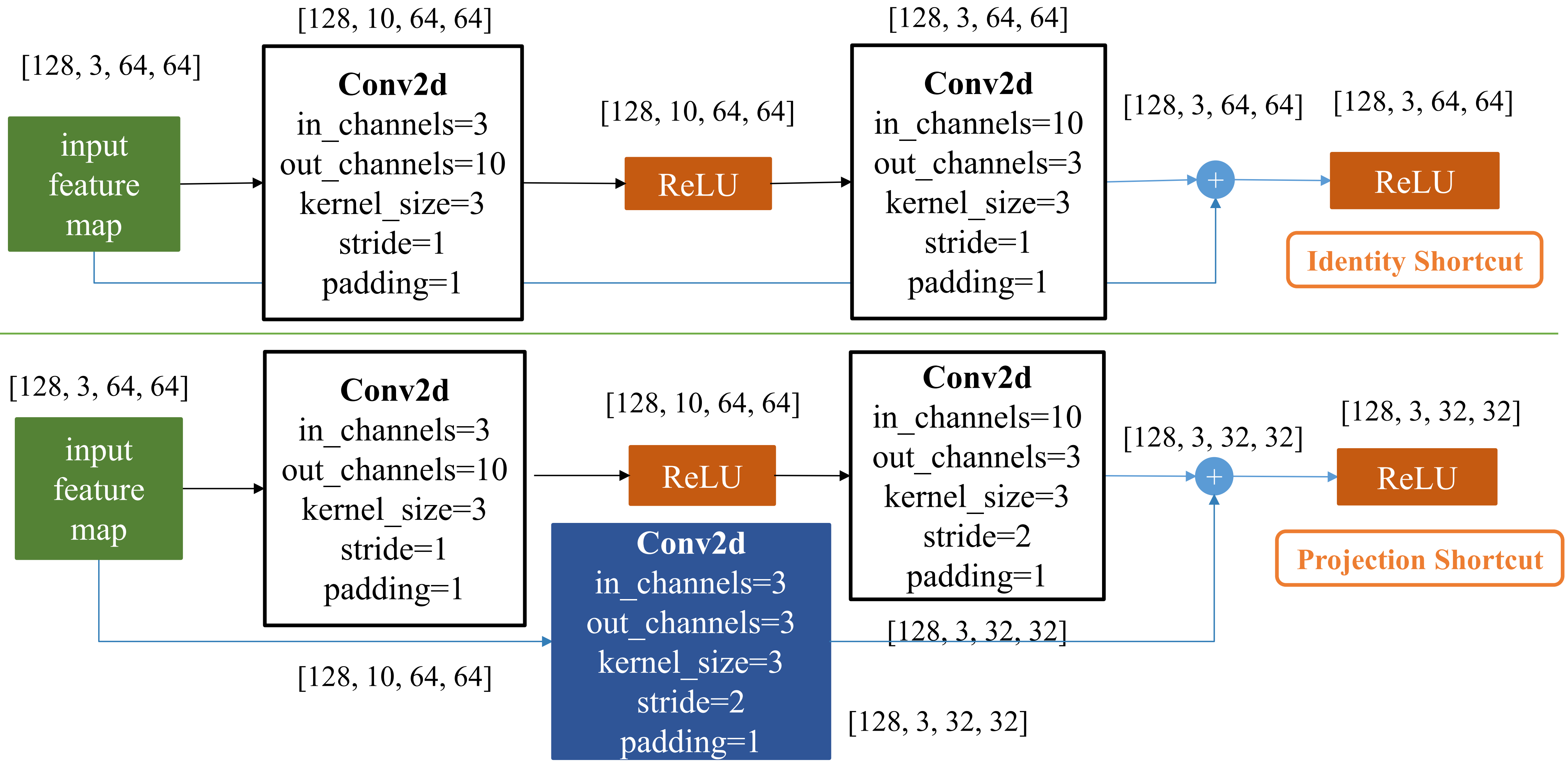
+

140.2087	1.9920	3.5892
8.2087	49.4680	179.0437
133.2087	211.0747	210.0651

ReLU		
140.2087	1.9920	3.5892
8.2087	49.4680	179.0437
133.2087	211.0747	210.0651

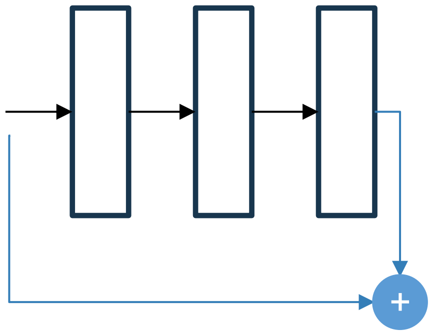
Skip Connection

❖ Skip Connection Variant

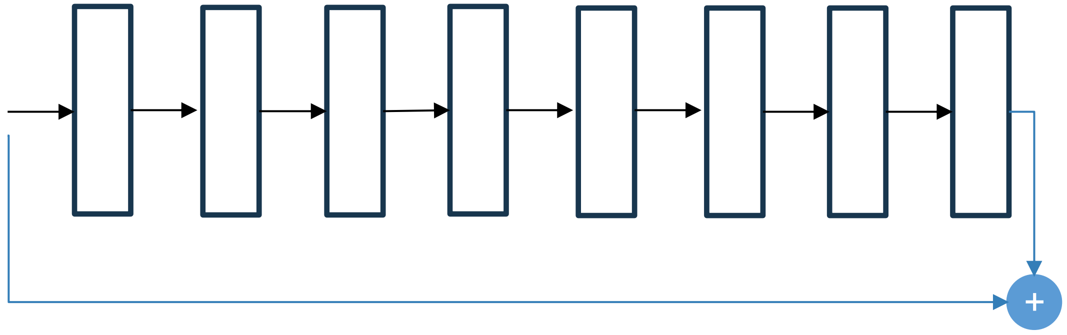


Skip Connection

❖ Skip Connection Variant



Short Skip Connection
(ResNet, …)



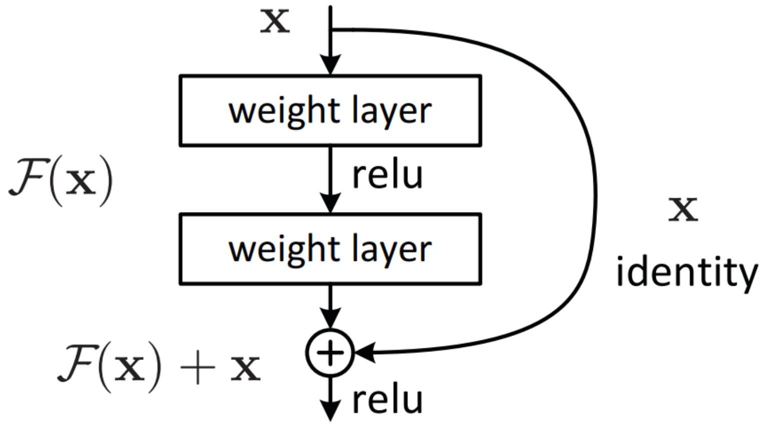
Long Skip Connection
(UNet, …)

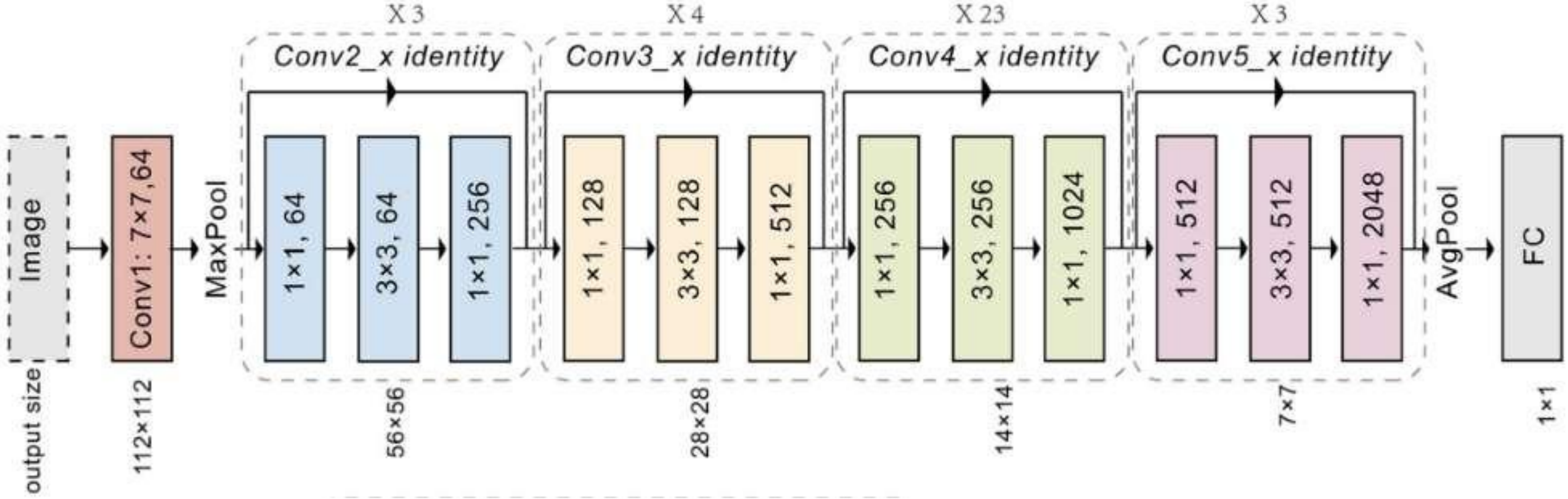
❖ Skip Connection Implementation

```
class CIFAR10Net_SK(nn.Module):
    def __init__(self):
        super(CIFAR10Net_SK, self).__init__()

        # Layer 1
        self.block1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2)   # 32x32 -> 16x16
        )

        # Layer 2
        self.block2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2)   # 16x16 -> 8x8
        )

        # Layer 3
        self.block3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2)   # 8x8 -> 4x4
        )

        # Skip projection: Layer 1 → Layer 4
        self.skip1_to_4 = nn.Sequential(
            nn.Conv2d(32, 128, kernel_size=1),  # channel match
            nn.MaxPool2d(4)                     # 16x16 -> 4x4
        )

        # FC
        self.fc_layers = nn.Sequential(
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x1 = self.block1(x)       # Layer 1 output
        x2 = self.block2(x1)
        x3 = self.block3(x2)      # Layer 4 main path

        skip = self.skip1_to_4(x1) # Skip connection
        x = x3 + skip              # Residual add

        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x
```

Skip Connection

❖ Resnet





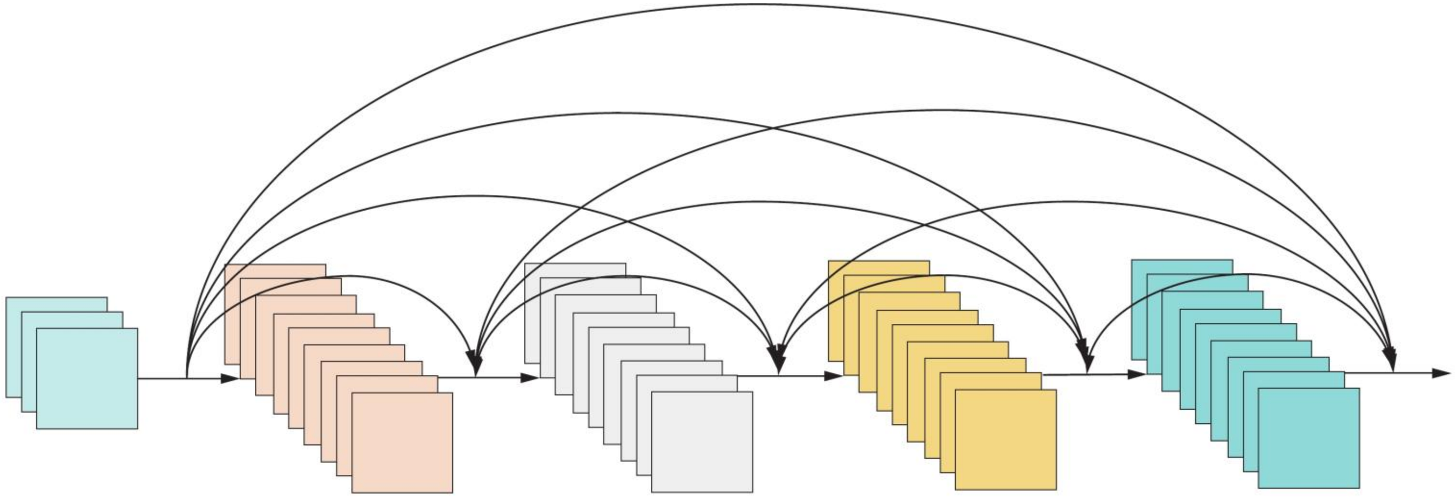
Skip Connection

❖ Resnet

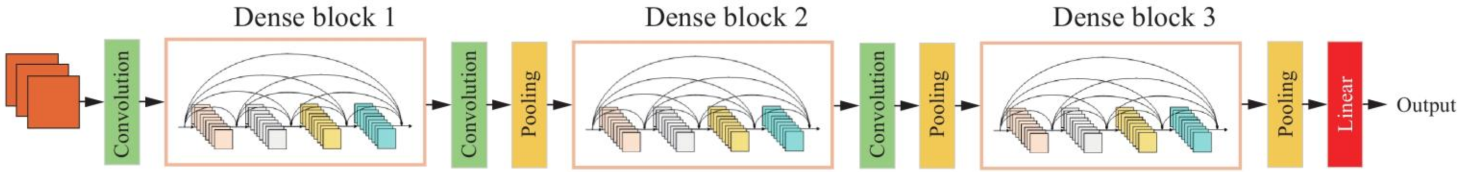
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	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

Skip Connection

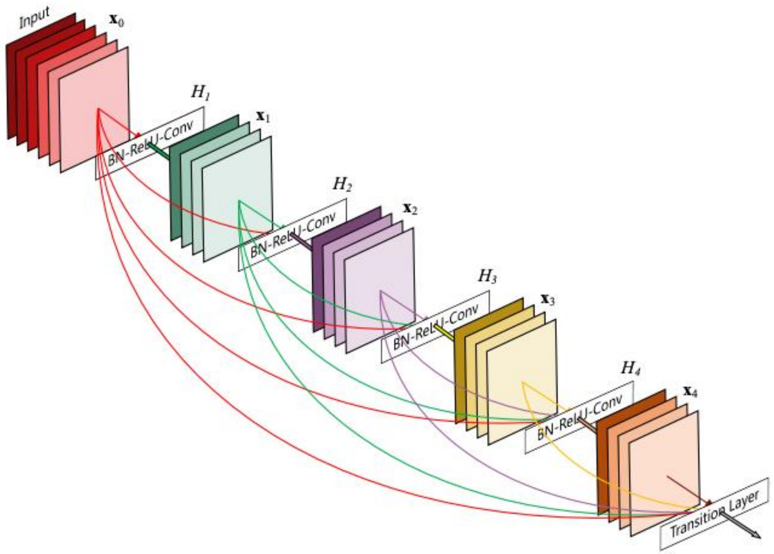
❖ Densenet



(a) Dense block structure



(b) DenseNet overall structure



Practice

Practice

Coding
Time

Review

Review

1. Convolution Layer
2. Pooling Layer
3. Multiple Input – Output
4. Read CNN Architecture

Section 1

Batch Normalization

1. Standard and Normalization
2. Batch Normalization
3. Training CNN
4. Training CNN with Batch Norm

Section 2

Drop Out

1. Why using Dropout
2. Apply Dropout

Section 3

Skip Connection

1. Degradation Problem
2. Skip Connection
3. Implement Skip Connection
4. Briefly about Resnet, Densenet

Section 4

Quiz&Feedbacks


Thank you!