Extra Class

Advanced CNN

Nguyen Quoc Thai

CONTENT

1 – Review: CNN

Convolutional Layer

Input: M x N

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

Padding: (P, Q)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 1 & 0 \\ 0 & 3 & 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 2 & 3 & 0 \\ 0 & 3 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 7 & 8 \\ 15 & 13 \end{bmatrix}$$

Shape: (M+2P) x (N+2Q)

Stride: (S, T)

Kernel: K x O

$$\boxed{1}$$

Bias

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

AI VIET NAM
@aivietnam.edu.vn

1 – Review: CNN

Pooling Layer

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

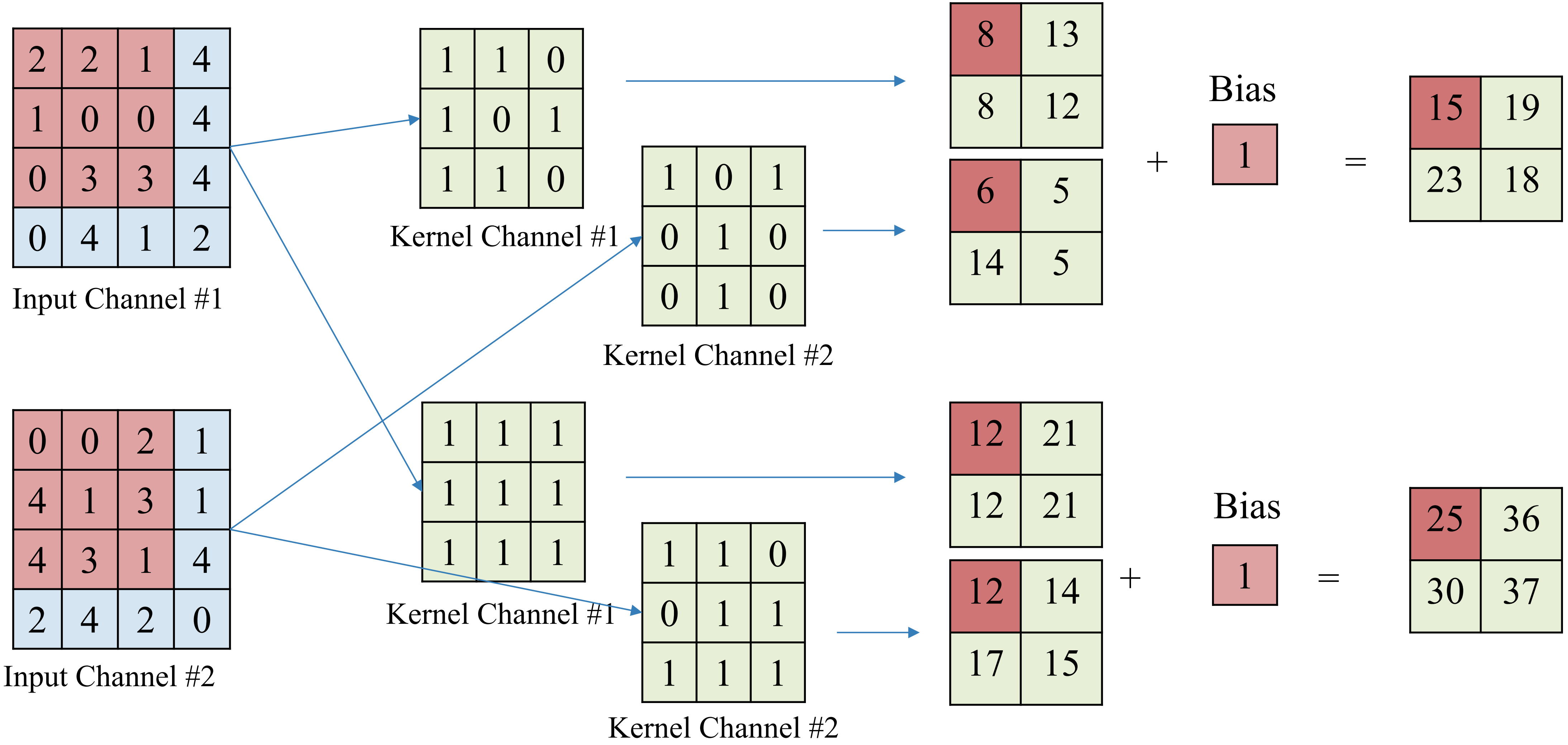
Kernel Size: (3, 2)
Stride: 2

2	1.7	0.8
1.8	1.6	1.3

Output: 2 x 3

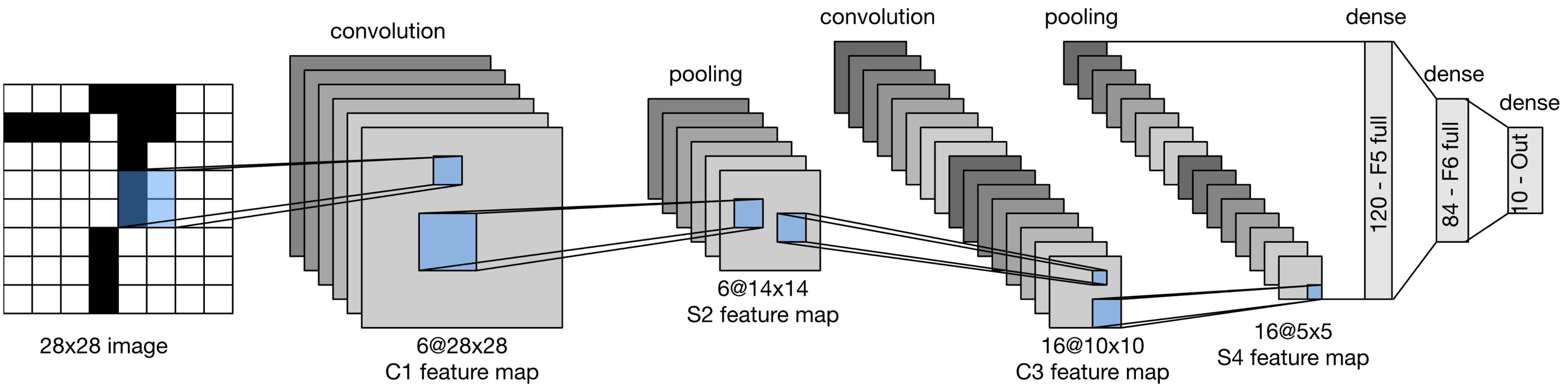
1 – Review: CNN

Multiple Input – Output Channels



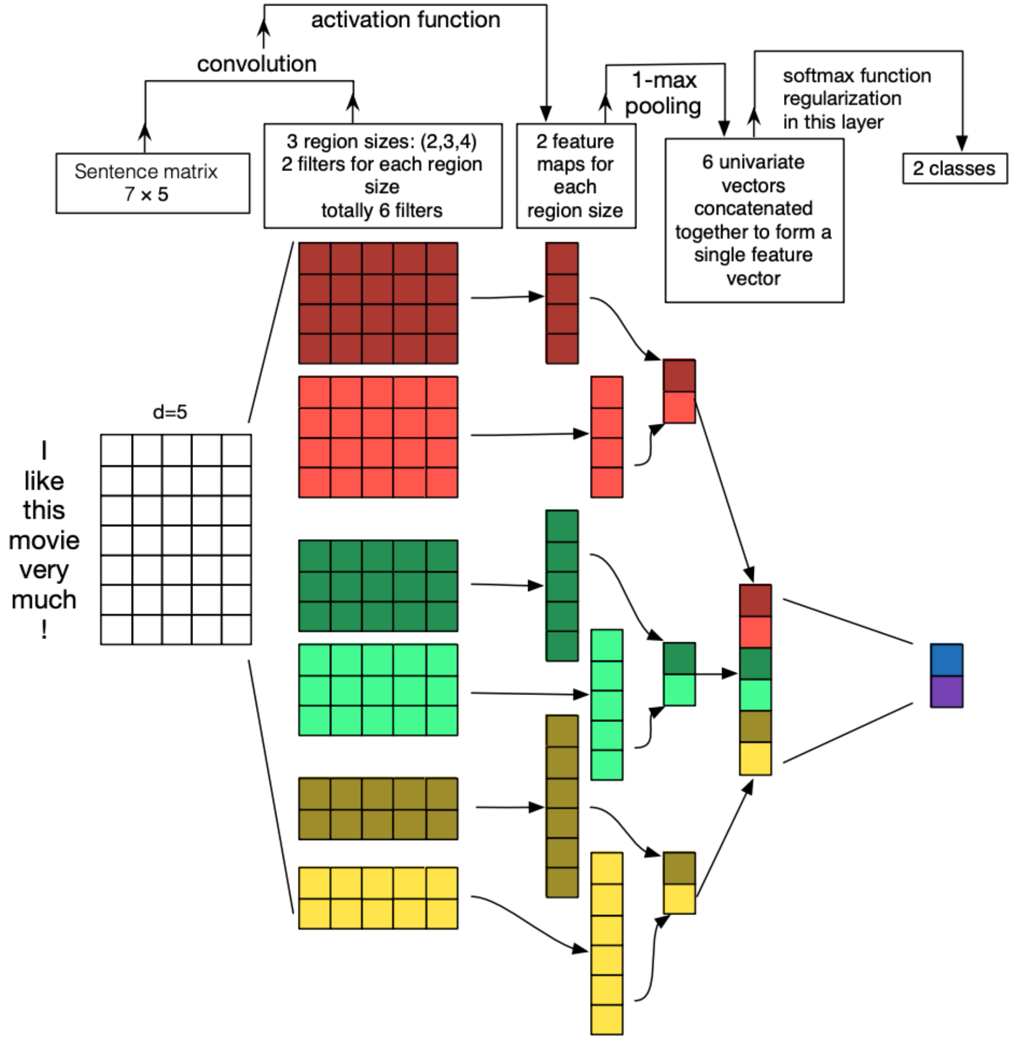
1 – Review: CNN

LeNet Model



1 – Review: CNN

TextCNN Model



AI VIET NAM
@aivietnam.edu.vn

2 – Batch Normalization

Data Normalization

❖ **Compute mean and variance**

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

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

❖ **Normalize X_i**

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

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

1	→	- 0.817
1		
2		
4		

2 – Batch Normalization

Data Normalization

- **Compute mean and variance**

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

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

- **Normalize X_i**

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

1
1
2
4

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

- 0.817
- 0.817
0.000

2 – Batch Normalization

Data Normalization

- **Compute mean and variance**

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

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

- **Normalize X_i**

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

1	- 0.817
1	- 0.817
2	0.000
4	1.634

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

2 – Batch Normalization

Batch Normalization

- **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-05)

- **Scale and shift $\hat{X}_i$**

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

γ and β are two learning parameters

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

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

$\gamma = 1$

$\beta = 0$

2 – Batch Normalization

Batch Normalization

- **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-05)

- **Scale and shift $\hat{X}_i$**

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

γ and β are two learning parameters

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

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

$\gamma = 1$

$\beta = 0$

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

2 – Batch Normalization

Batch Normalization

- **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-05)

- **Scale and shift $\hat{X}_i$**

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

γ and β are two learning parameters

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

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

$\gamma = 1$

$\beta = 0$

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

2 – Batch Normalization

Batch Normalization

- **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-05)

- **Scale and shift $\hat{X}_i$**

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

γ and β are two learning parameters

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

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

$\gamma = 1$

$\beta = 0$

mean	2.25	2.5	1.25
variance	1.479	1.118	1.639

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

$$Y_0 = 1 * (-0.169) + 0$$

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

AI VIET NAM
@aivietnam.edu.vn

2 – Batch Normalization

Batch Normalization

- **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-05)

- **Scale and shift $\hat{X}_i$**

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

γ and β are two learning parameters

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

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

$\gamma = 1$

$\beta = 0$

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

2 – Batch Normalization

Batch Normalization - Demo

- **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-05)

- **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>)
```

2 – Batch Normalization

Batch Normalization - Demo

```
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)
    )
```

```
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)
```

```
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([-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)
```

2 – Batch Normalization

Normalization during Inference

- **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-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}: 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

2 – Batch Normalization

Normalization during 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

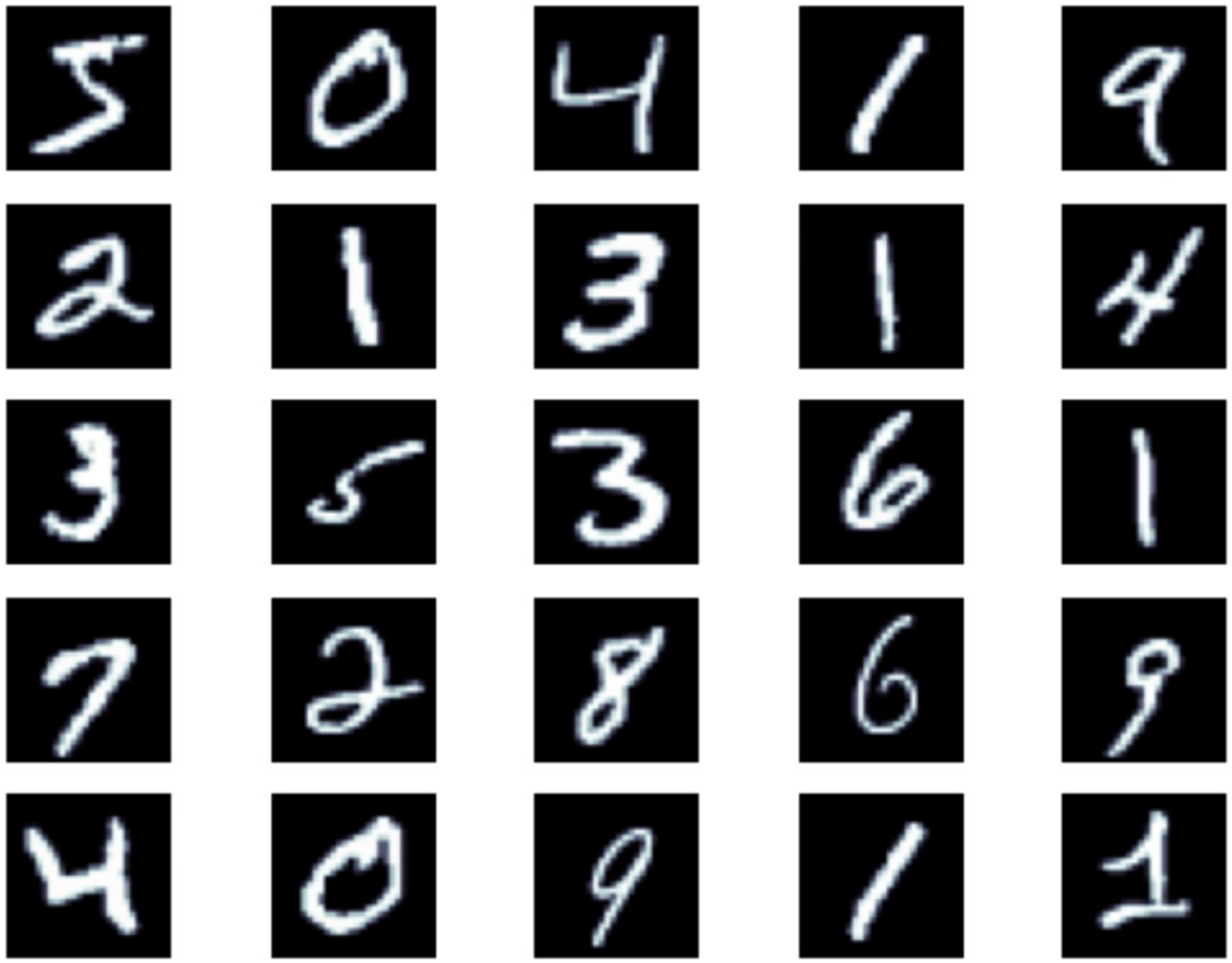
$\gamma = 0.5$

$\beta = 0.05$

2 – Batch Normalization

MNIST using Batch Normalization

- **MNIST dataset**
  - Images: 70.000
  - Class: 10
  - Image Size: 28 x 28



2 – Batch Normalization

MNIST using Batch Normalization - Demo

❖ MNIST dataset

➢ Preprocessing

```
VALID_RATIO = 0.9

n_train_examples = int(len(train_data) * VALID_RATIO)
n_valid_examples = len(train_data) - n_train_examples
```

```
train_data, valid_data = data.random_split(
    train_data,
    [n_train_examples, n_valid_examples]
)
```

```
# compute mean and std
mean = train_data.dataset.data.float().mean() / 255
std = train_data.dataset.data.float().std() / 255
mean, std
```

```
(tensor(0.1307), tensor(0.3081))
```

```
train_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])

test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])
```

```
train_data.dataset.transform = train_transforms
valid_data.dataset.transform = test_transforms
```

2 – Batch Normalization

MNIST using Batch Normalization - Demo

- **MNIST dataset**
  - Base Model

```
base_model = nn.Sequential(
    nn.Conv2d(1, 6, 5, stride=2),
    nn.Flatten(),
    nn.Linear(6 * 12 * 12, 64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Linear(32, 10)
)
```

```
summary(base_model, (1, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 12, 12]             156
           Flatten-2                  [-1, 864]               0
            Linear-3                   [-1, 64]          55,360
              ReLU-4                   [-1, 64]               0
            Linear-5                   [-1, 32]           2,080
              ReLU-6                   [-1, 32]               0
            Linear-7                   [-1, 10]             330
================================================================
Total params: 57,926
Trainable params: 57,926
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.22
Estimated Total Size (MB): 0.24
----------------------------------------------------------------
```

2 – Batch Normalization

MNIST using Batch Normalization - Demo

- **MNIST dataset**
  - Model with BatchNorm Layer

```
batchnorm_model = nn.Sequential(
    nn.Conv2d(1, 6, 5, stride=2),
    nn.Flatten(),
    nn.Linear(6 * 12 * 12, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.BatchNorm1d(32),
    nn.ReLU(),
    nn.Linear(32, 10)
)
```

```
summary(batchnorm_model, (1, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 12, 12]             156
           Flatten-2                  [-1, 864]               0
            Linear-3                   [-1, 64]          55,360
       BatchNorm1d-4                   [-1, 64]             128
              ReLU-5                   [-1, 64]               0
            Linear-6                   [-1, 32]           2,080
       BatchNorm1d-7                   [-1, 32]              64
              ReLU-8                   [-1, 32]               0
            Linear-9                   [-1, 10]             330
================================================================
Total params: 58,118
Trainable params: 58,118
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.22
Estimated Total Size (MB): 0.24
----------------------------------------------------------------
```

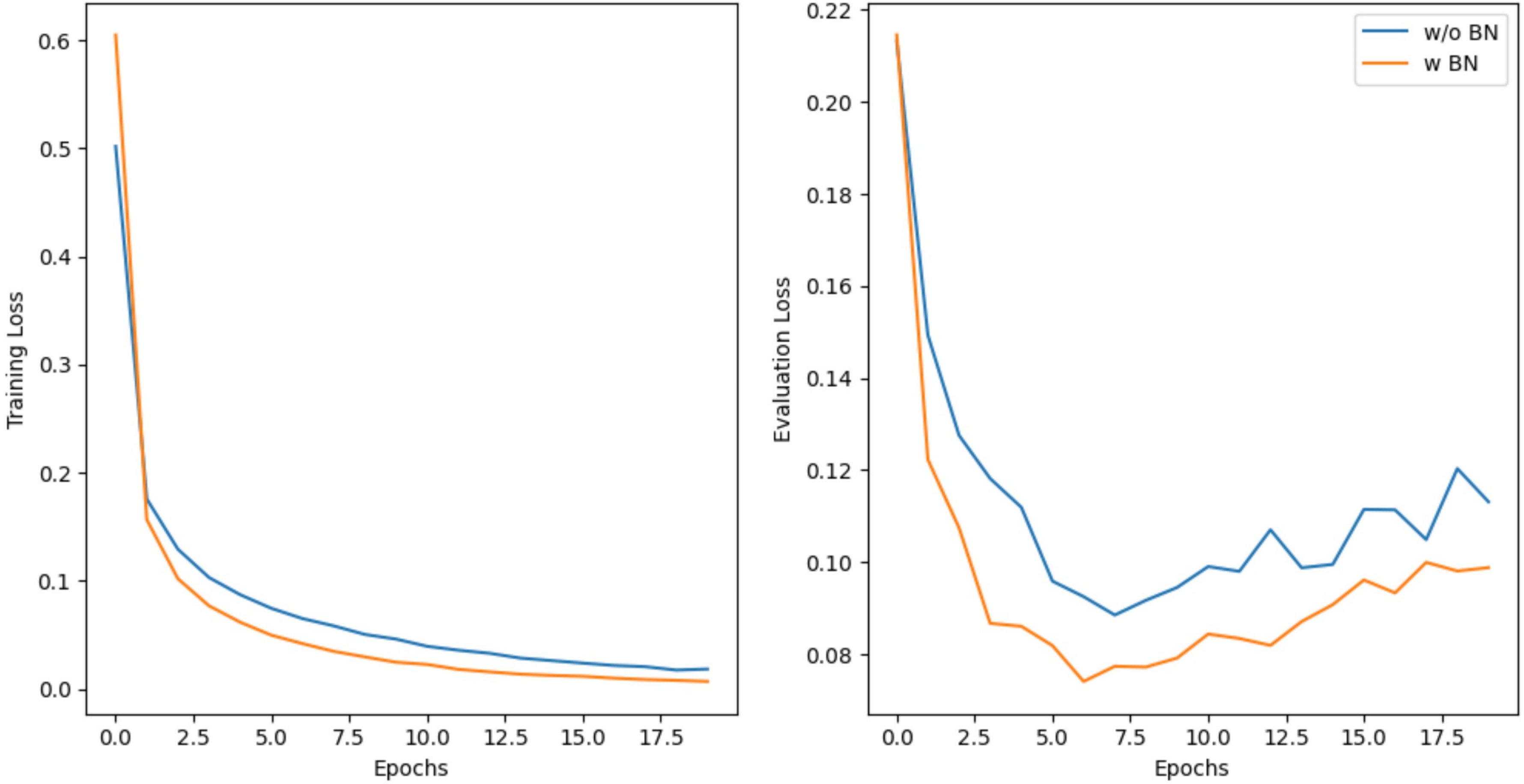
2 – Batch Normalization

MNIST using Batch Normalization

MNIST dataset

- Training
- Evaluation

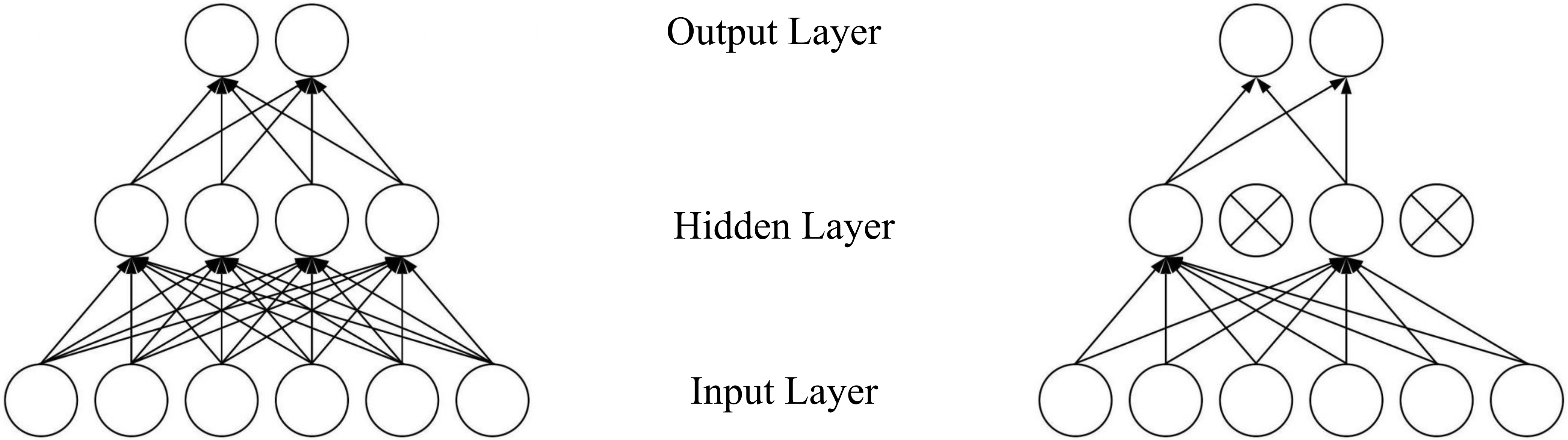
w/o BN: 97.53
w BN: 97.85



3 – Dropout

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

3 – Dropout

Dropout - Demo

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

Initial

```
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)
```

Updated

```
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)
```

3 – Dropout

MNIST using Dropout - Demo

MNIST dataset

- Model with Dropout Layer

```
dropout_model = nn.Sequential(
    nn.Conv2d(1, 6, 5, stride=2),
    nn.Flatten(),
    nn.Linear(6 * 12 * 12, 64),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(32, 10)
)
```

```
summary(dropout_model, (1, 28, 28))
```

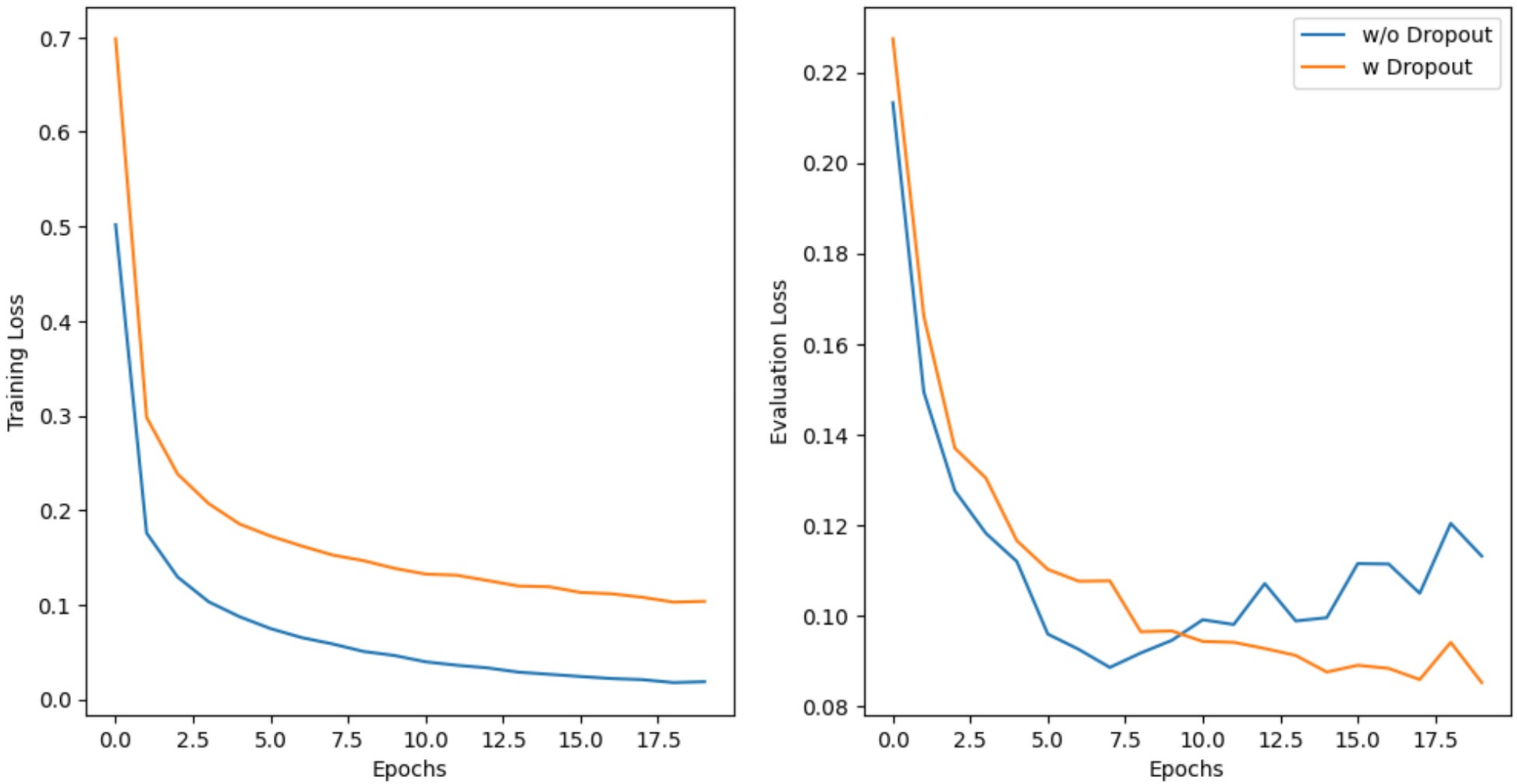
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 12, 12]             156
           Flatten-2                  [-1, 864]               0
            Linear-3                   [-1, 64]          55,360
              ReLU-4                   [-1, 64]               0
           Dropout-5                   [-1, 64]               0
            Linear-6                   [-1, 32]           2,080
              ReLU-7                   [-1, 32]               0
           Dropout-8                   [-1, 32]               0
            Linear-9                   [-1, 10]             330
================================================================
Total params: 57,926
Trainable params: 57,926
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.22
Estimated Total Size (MB): 0.24
----------------------------------------------------------------
```

3 – Dropout

MNIST using Dropout

- **MNIST dataset**
  - Training
  - Evaluation

w/o Dropout: 97.53
w Dropout: 97.58

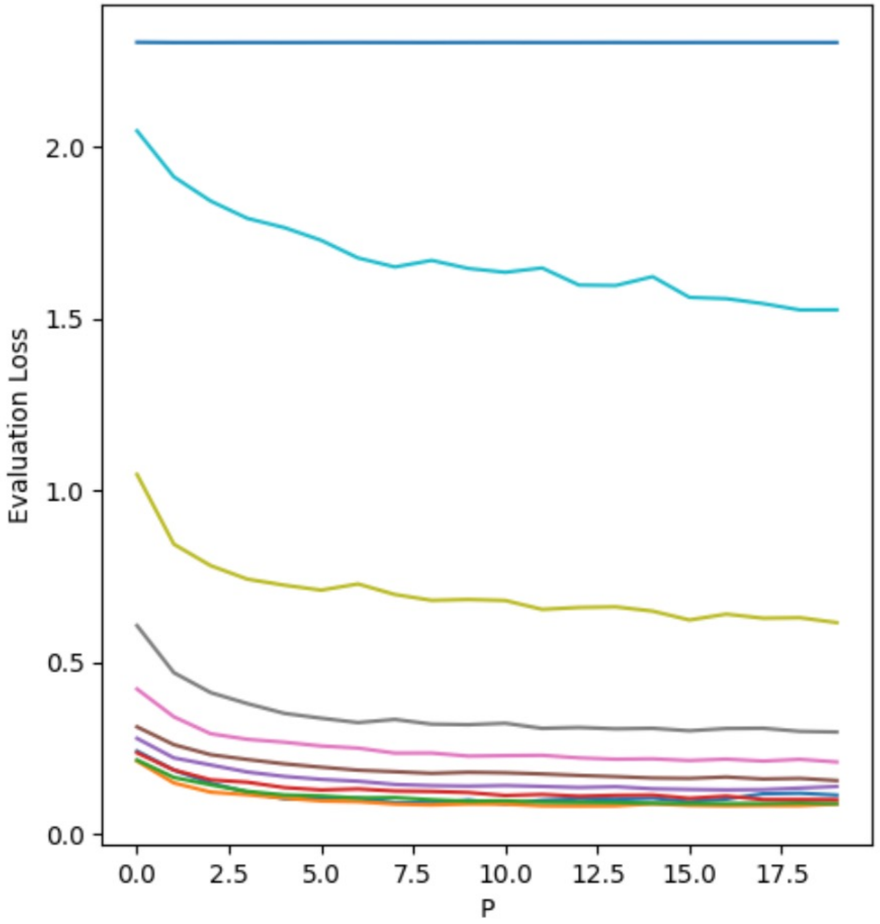


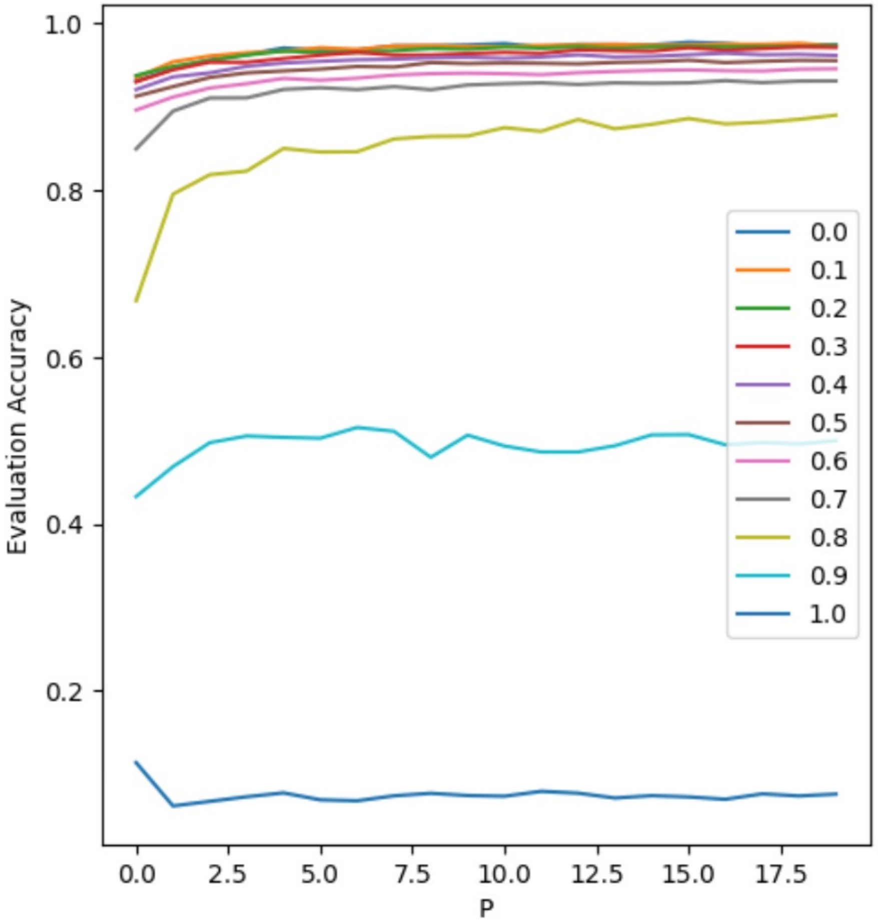
AI VIET NAM
@aivietnam.edu.vn

3 – Dropout

MNIST using Dropout - Demo

❖ **Probability of an element to be zeroed (P)**

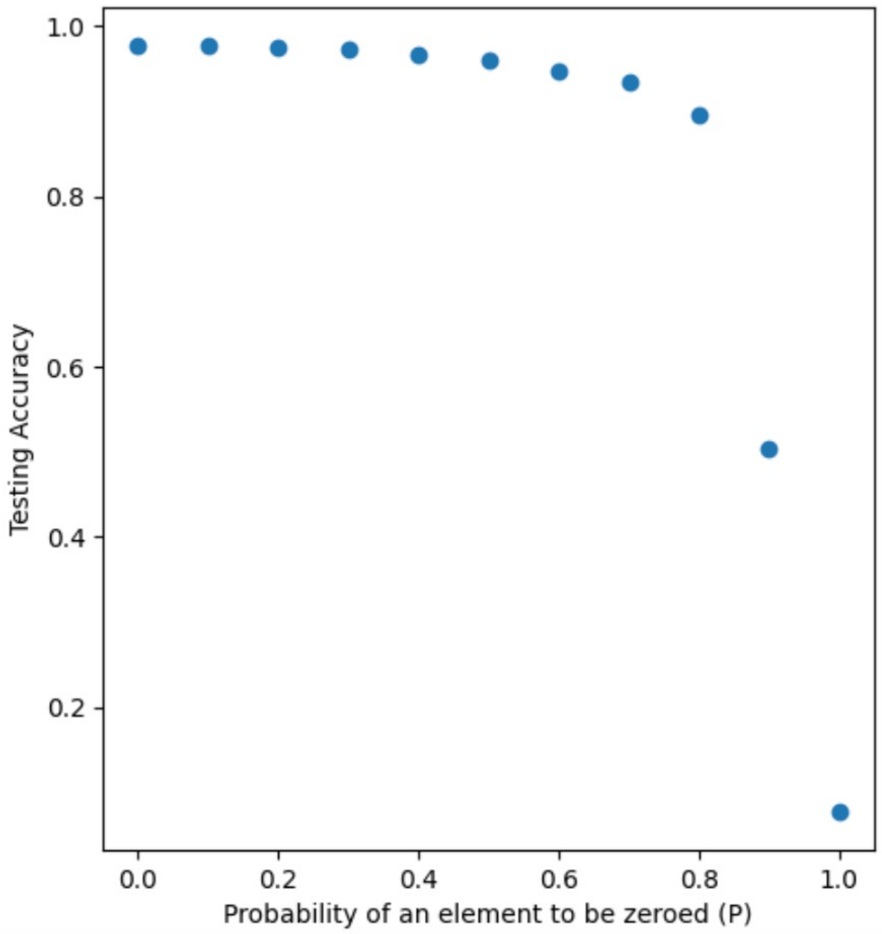


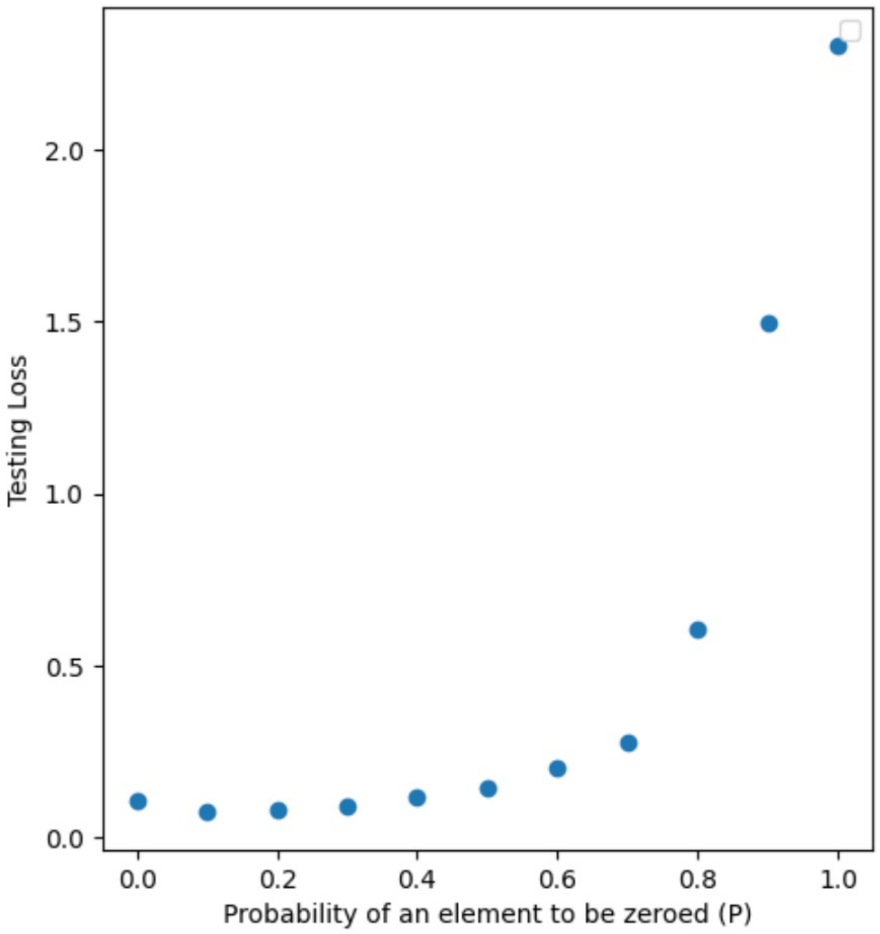


3 – Dropout

MNIST using Dropout

❖ **Probability of an element to be zeroed (P)**

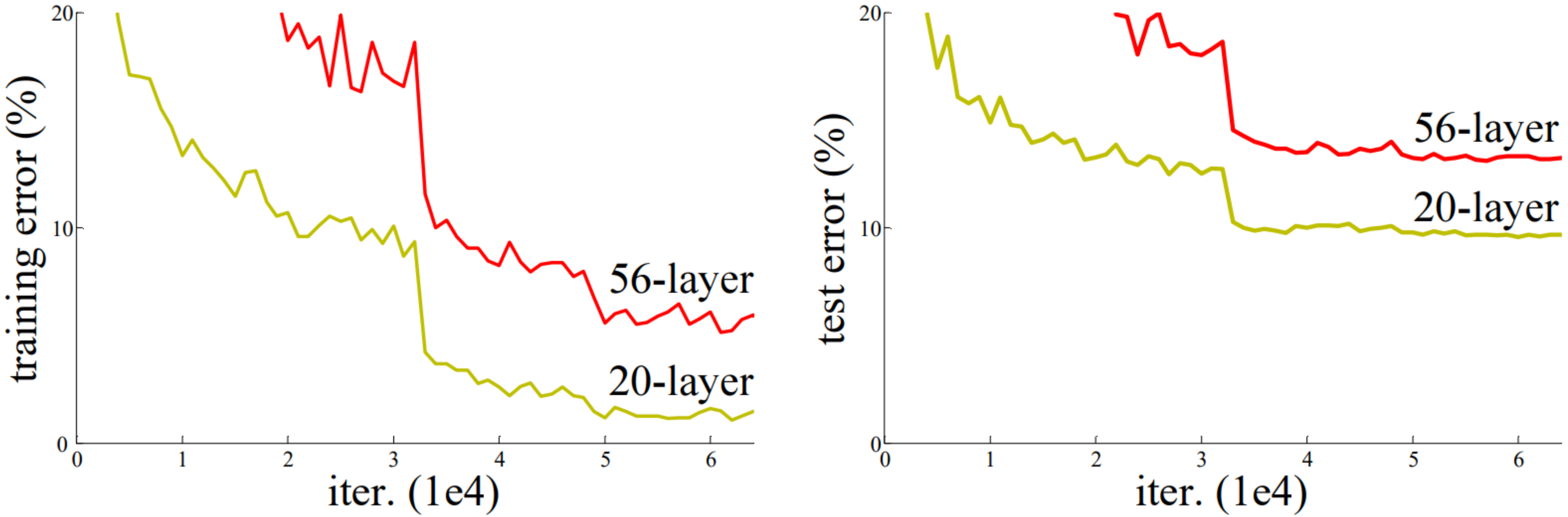




4 – Skip Connection

Degradation Problem

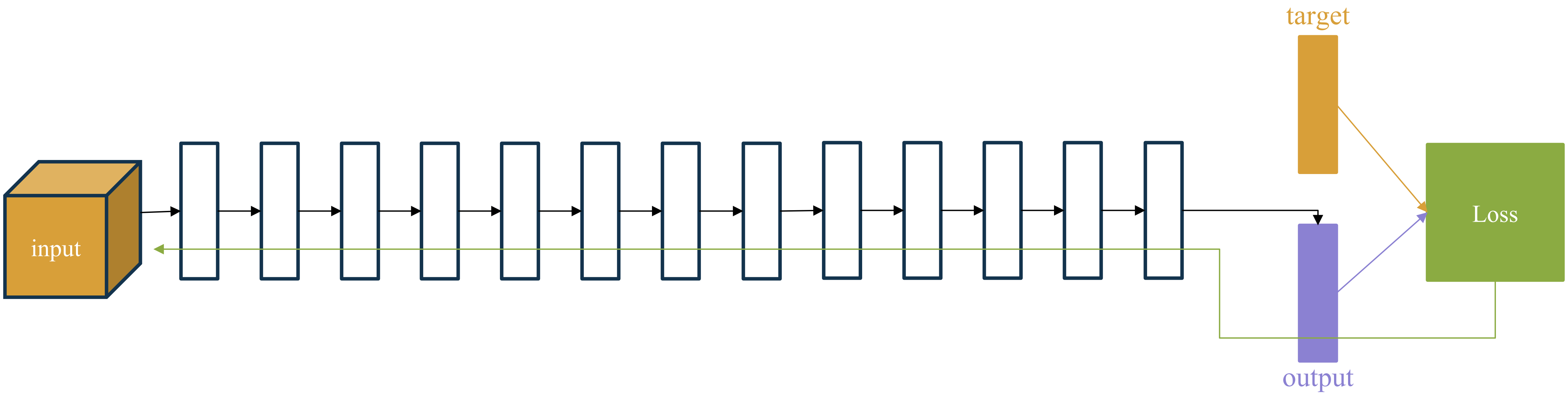
The deeper model doesn't perform as well as the shallow one



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

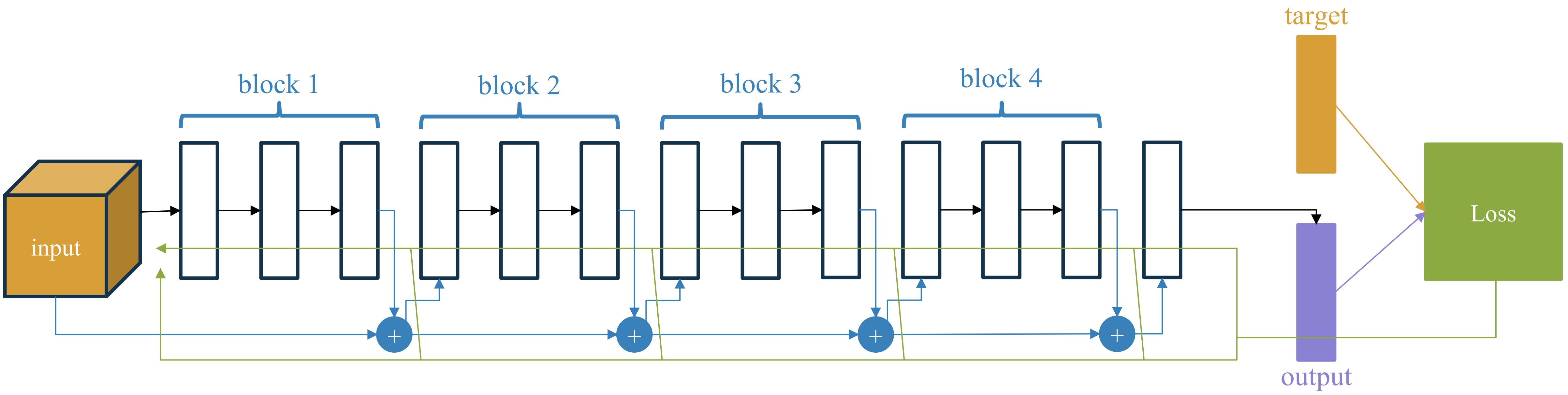
4 – Skip Connection

Degradation Problem



4 – Skip Connection

Skip Connection



4 – Skip 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

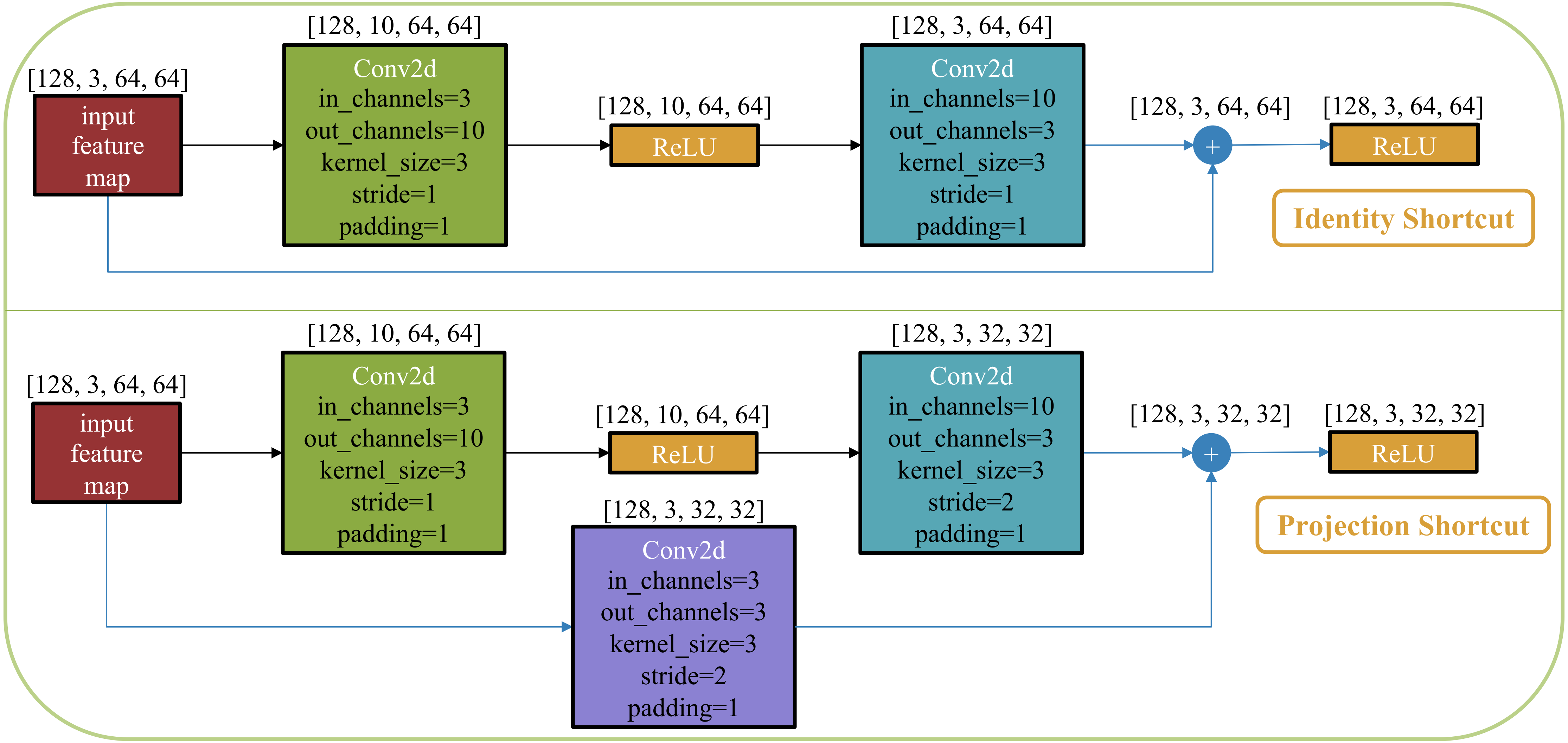
ReLU

140.2087	1.9920	3.5892
8.2087	49.4680	179.0437
133.2087	211.0747	210.0651

AI VIET NAM
@aivietnam.edu.vn

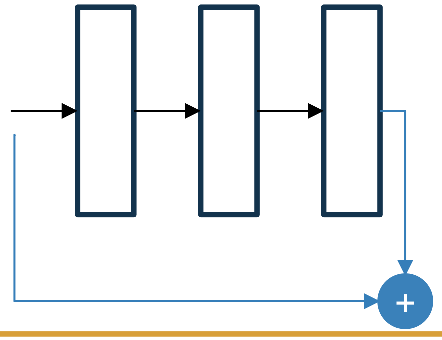
4 – Skip Connection

Skip Connection Variants

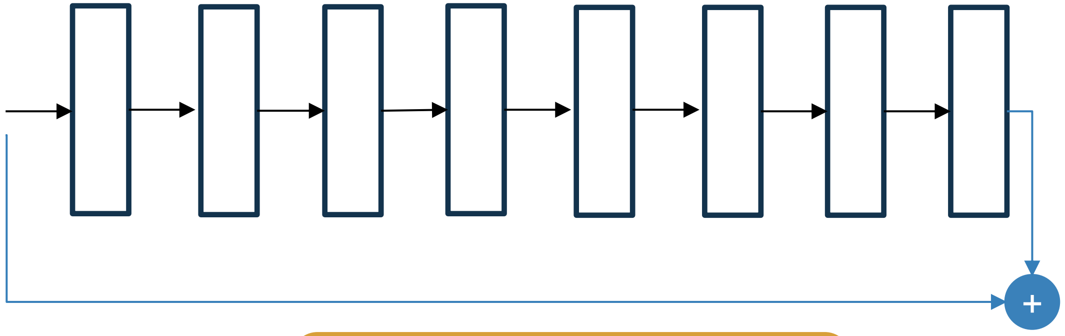


4 – Skip Connection

Skip Connection Variants



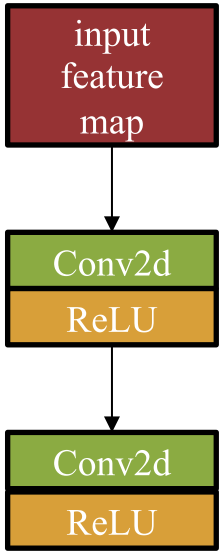
Short Skip Connection
(ResNet, …)



Long Skip Connection
(UNet, …)

4 – Skip Connection

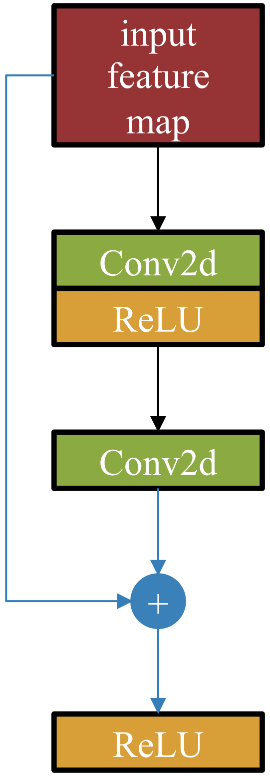
Implementation – No Skip Connection




```python
class NoSkipConnection(nn.Module):
    def __init__(self):
        super(NoSkipConnection, self).__init__()
        self.conv1 = nn.Conv2d(1, 1, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(1, 1, kernel_size=3, padding=1)

    def forward(self, x):
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)

        return out
```

4 – Skip Connection

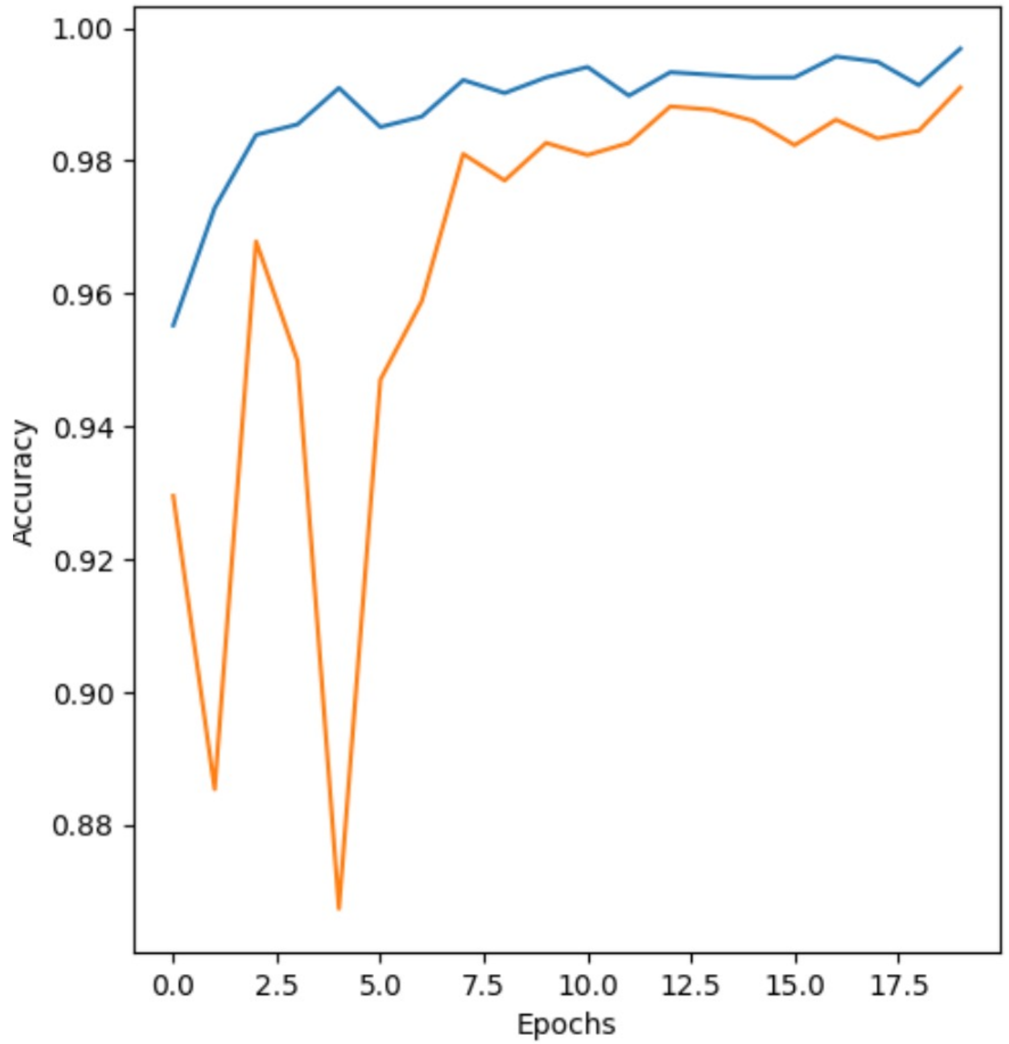
Implementation – Skip Connection

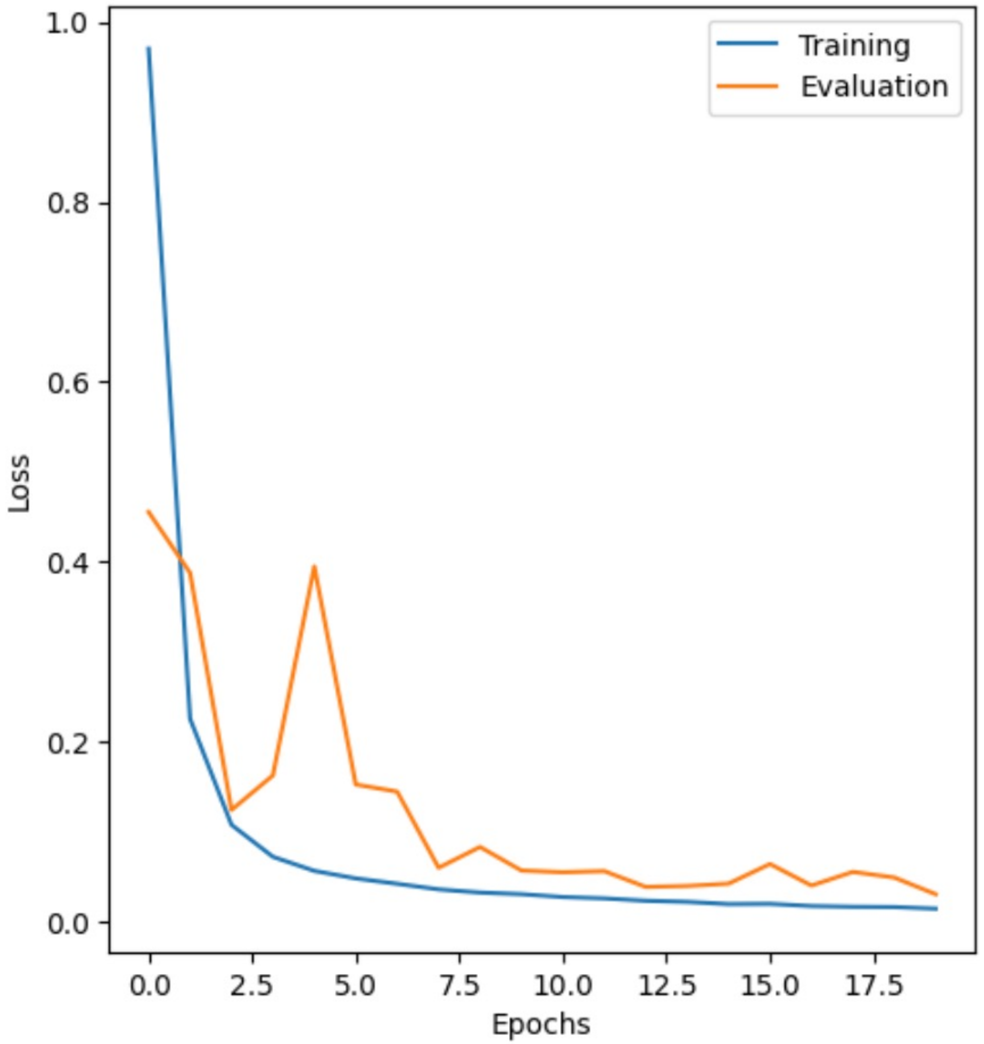



```
class SkipConnection(nn.Module):
    def __init__(self):
        super(SkipConnection, self).__init__()
        self.conv1 = nn.Conv2d(1, 1, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(1, 1, kernel_size=3, padding=1)

    def forward(self, x):
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out += x
        out = F.relu(out)

        return out
```

4 – Skip Connection

MNIST using Skip Connection - Demo





Thanks!

Any questions?