AI VIETNAM
All-in-One Course

Convolutional Neural Network

Motivation and Introduction to CNN

Quang-Vinh Dinh
Ph.D. in Computer Science

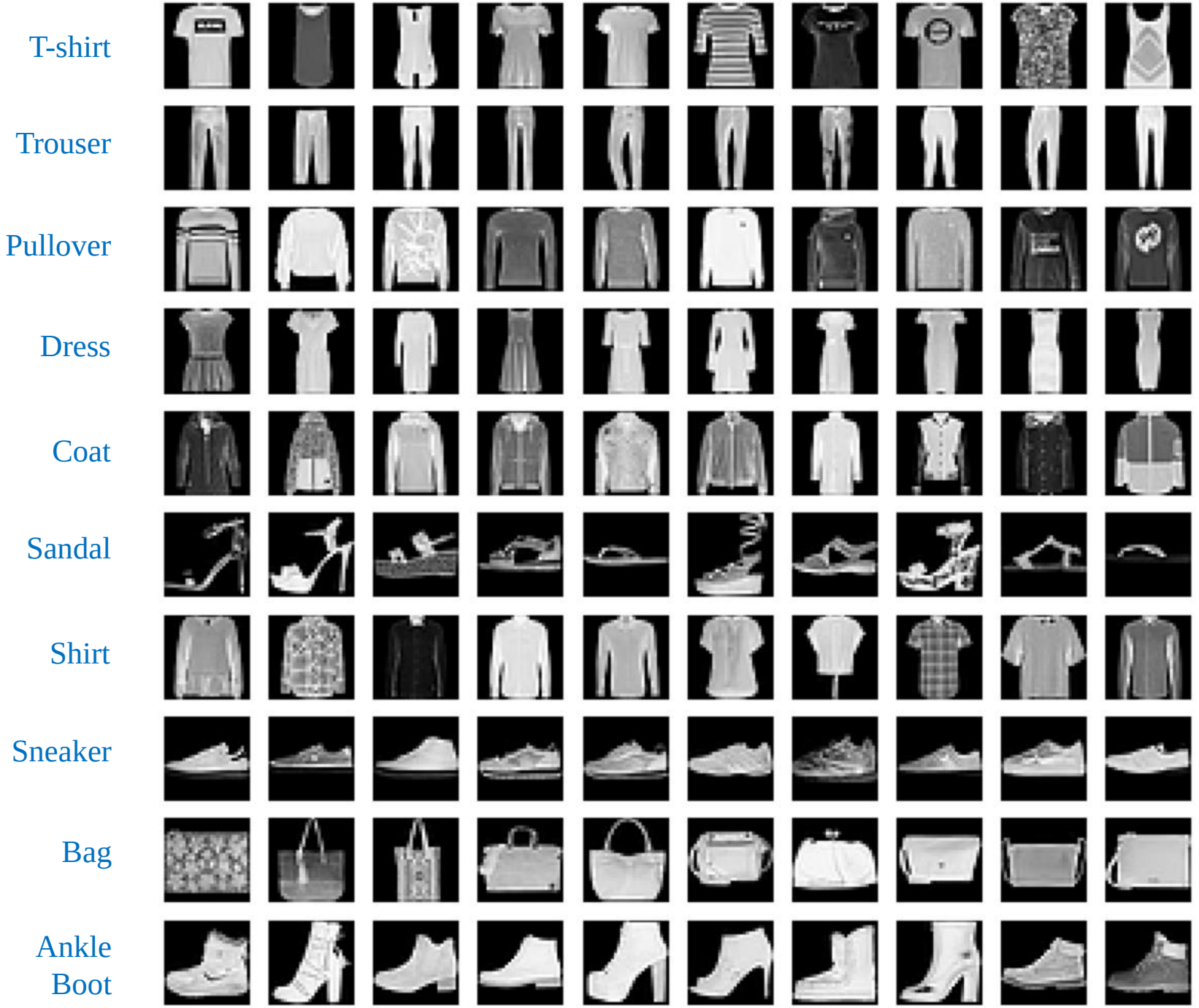
Year 2023

Outline

- MLP Limitation
- From MLP to CNN
- Feature Map Down-sampling
- Some Examples
- Application to Cifar10

Fashion-MNIST dataset

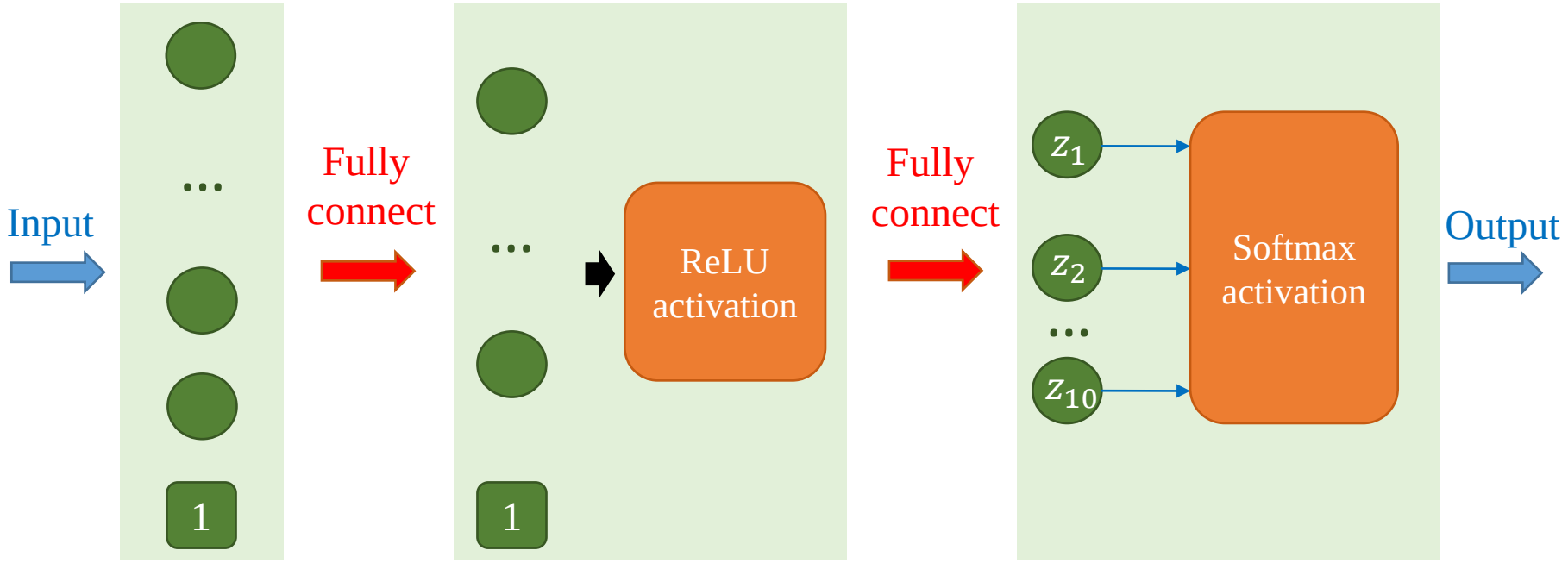
- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples



MLP for Fashion-MNIST

Case 1

❖ ReLU, He and Adam



MLP for Fashion-MNIST

Case 1

❖ ReLU, He and Adam

```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```

```
# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)
```

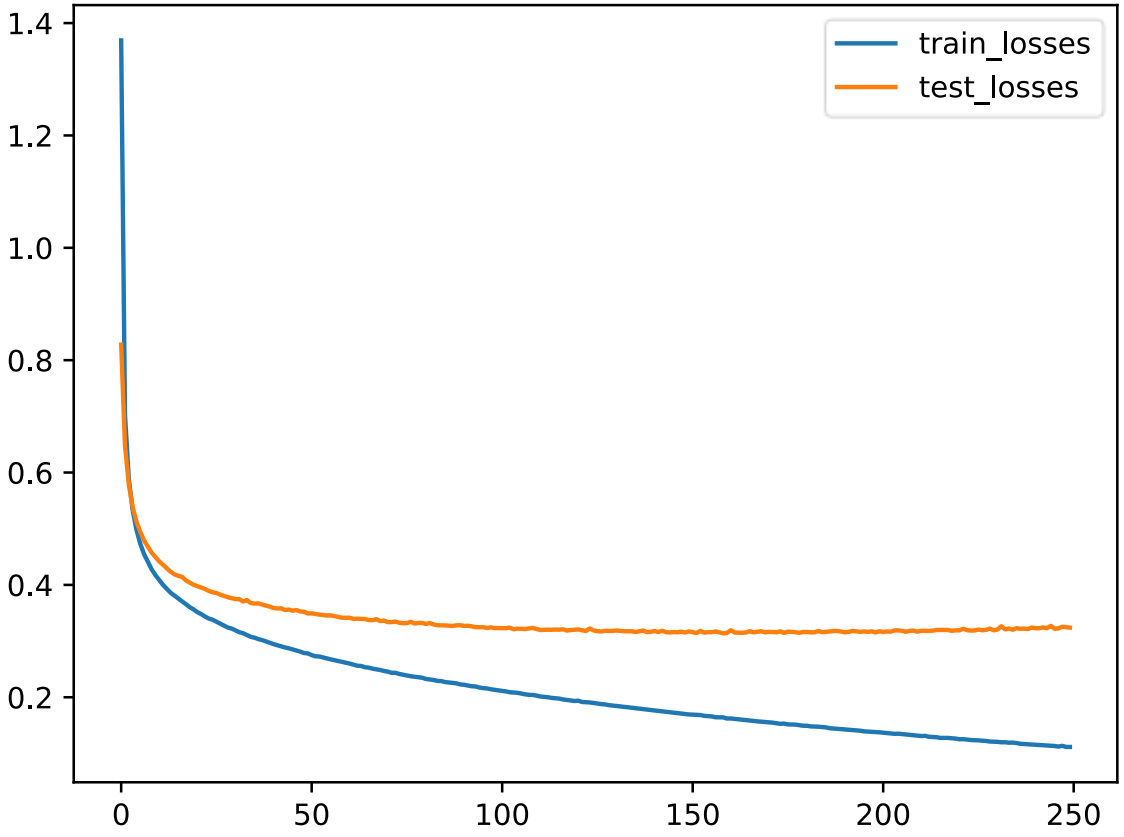
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

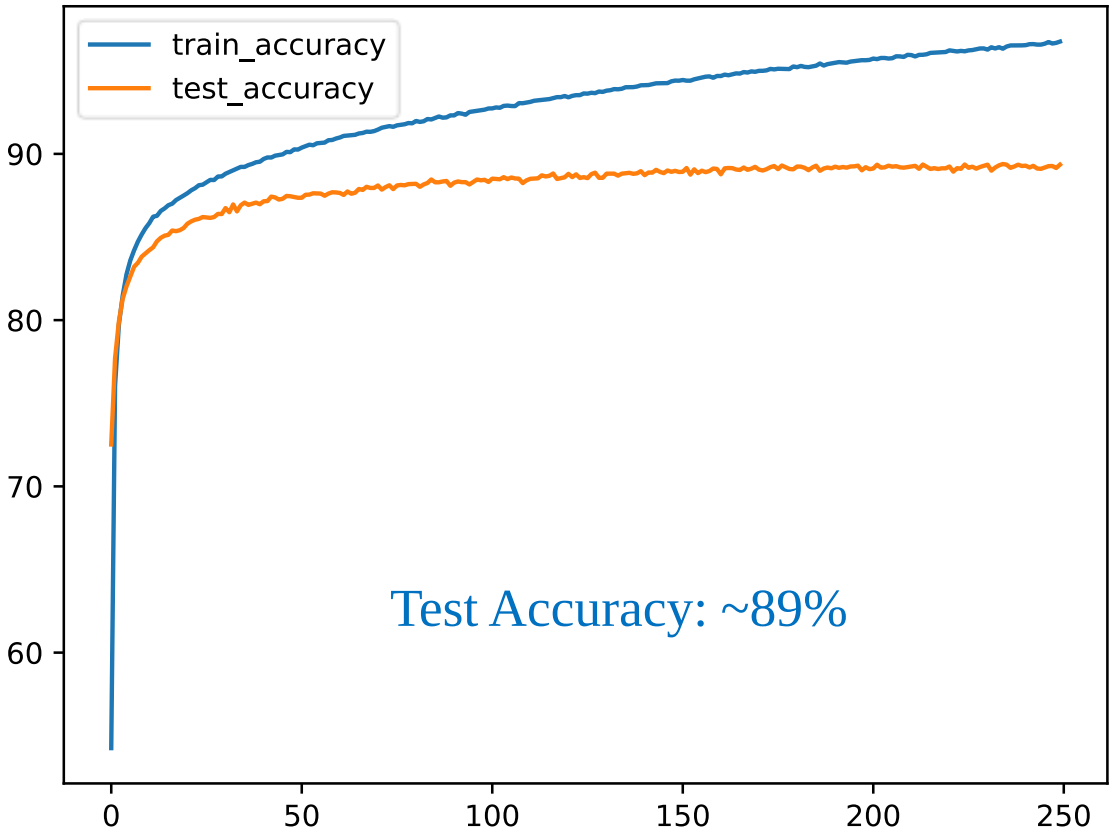
```
# Load CFashionMNIST dataset
transform = Compose([transforms.ToTensor(),
                     transforms.Normalize((0.5,),
                                          (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP for Fashion-MNIST

Case 1

❖ ReLU, He and Adam





Adam with learning rate of 1e-4

Perform reasonably

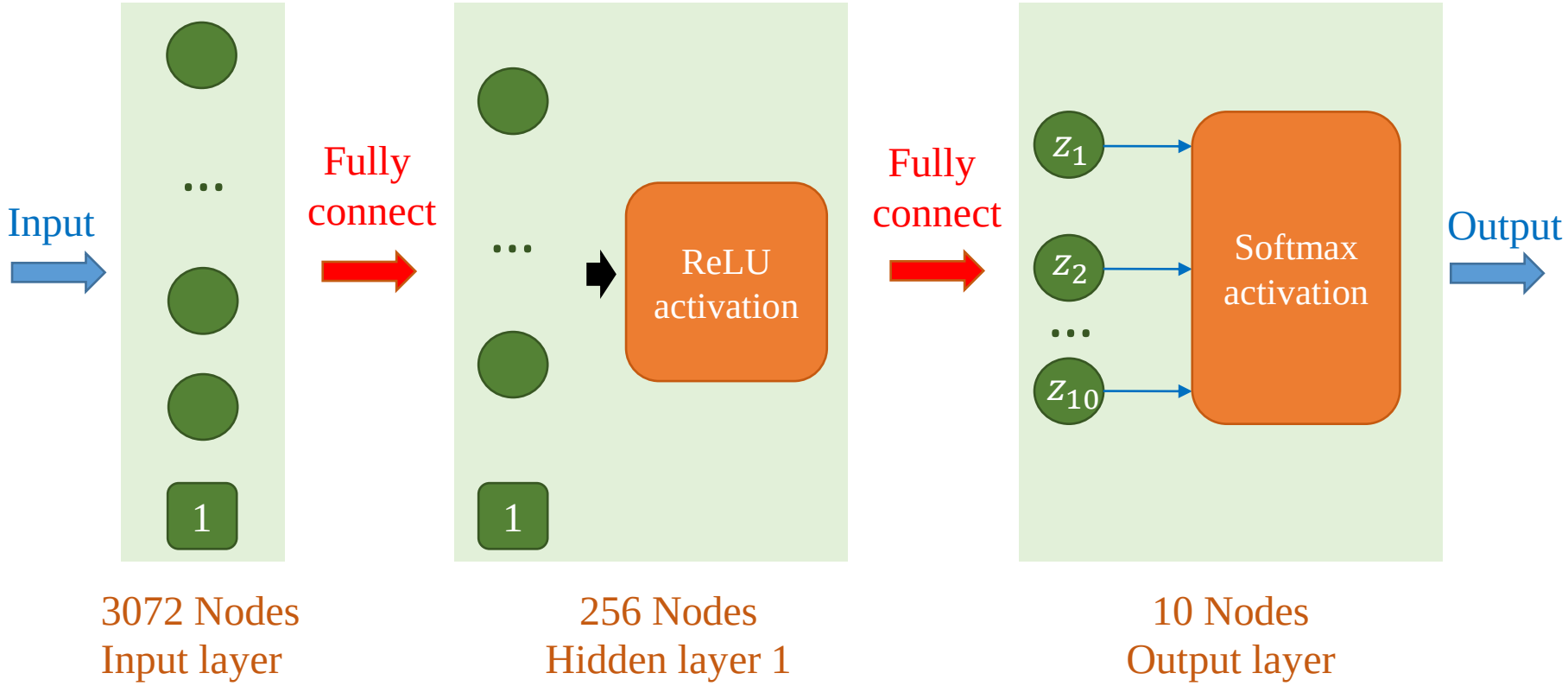
Cifar-10 dataset

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples

MLP for Cifar-10

Case 2

❖ ReLU, He and Adam



MLP for Cifar-10

Case 2

❖ ReLU, He and Adam

```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```

```
# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)
```

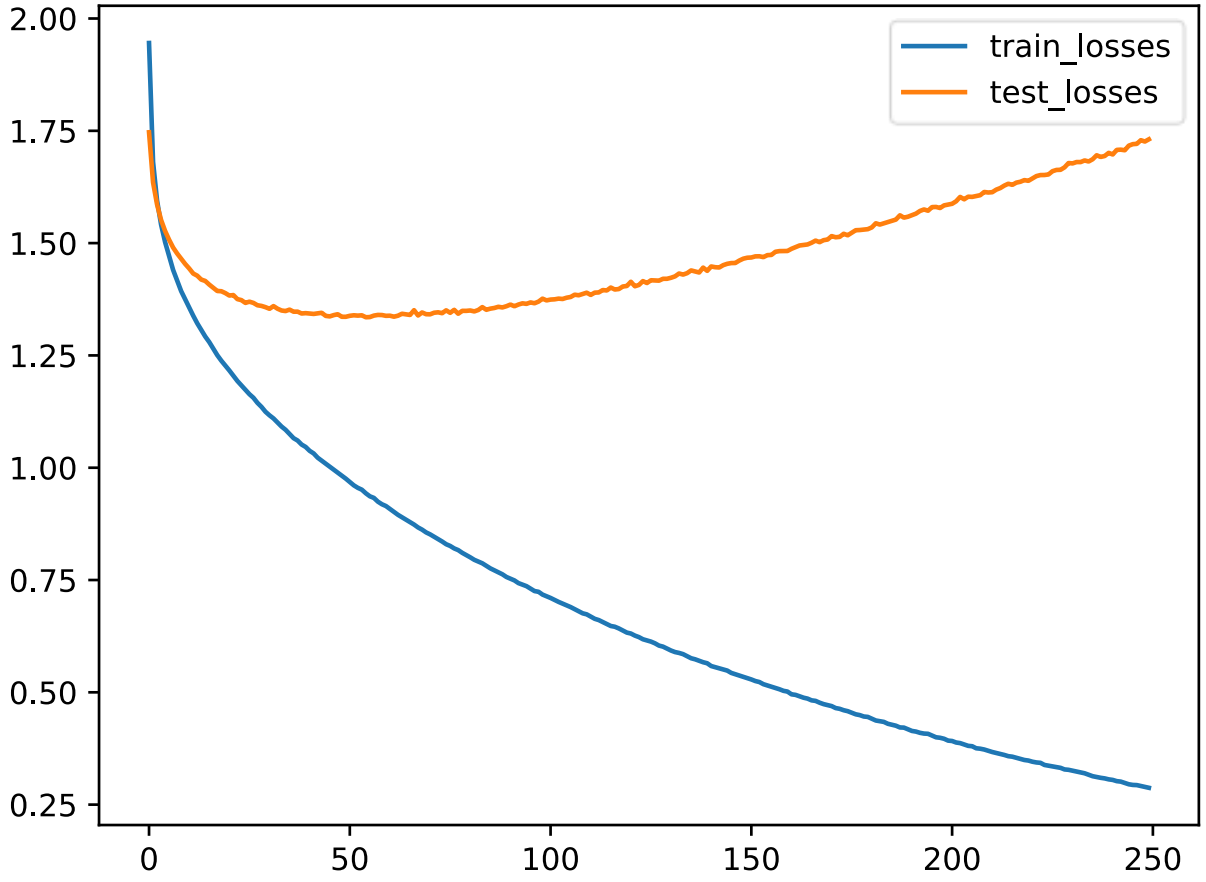
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

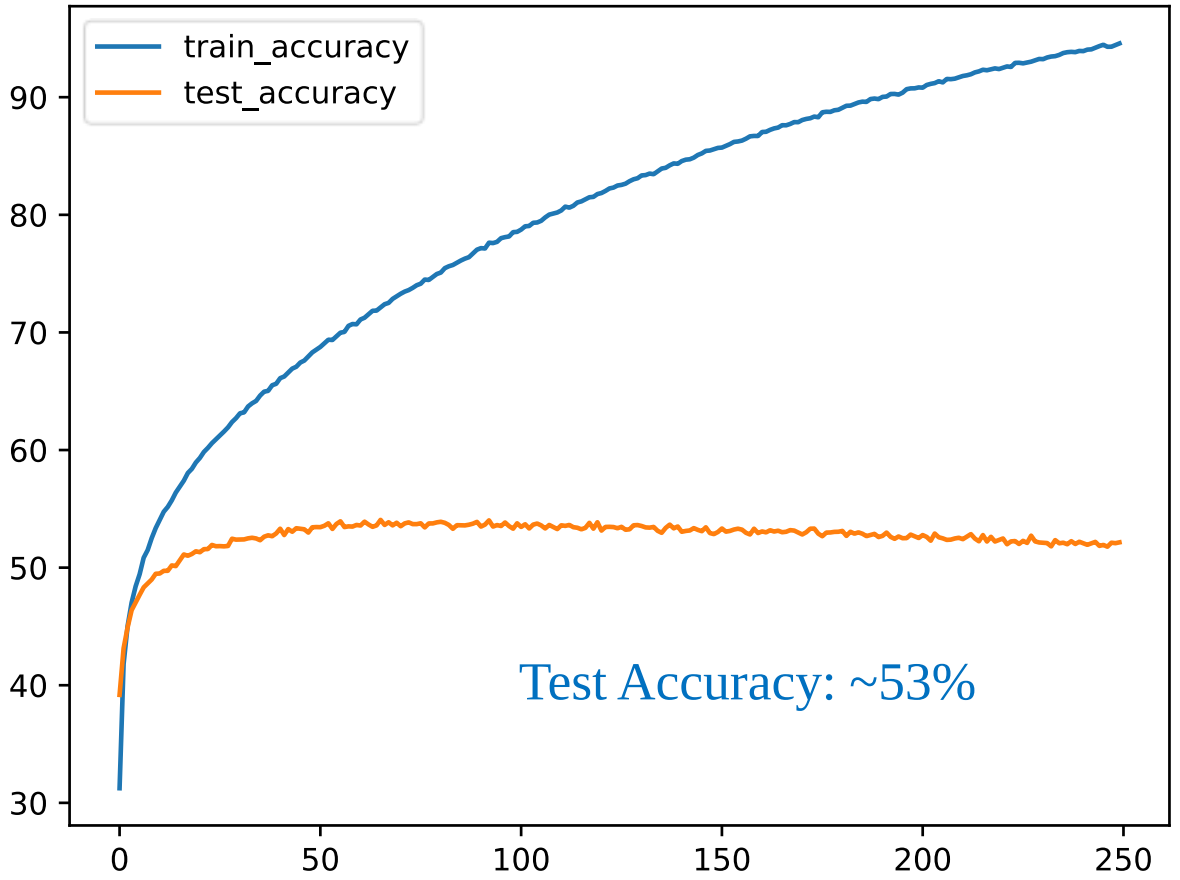
```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP for Cifar-10

Case 2

❖ ReLU, He and Adam



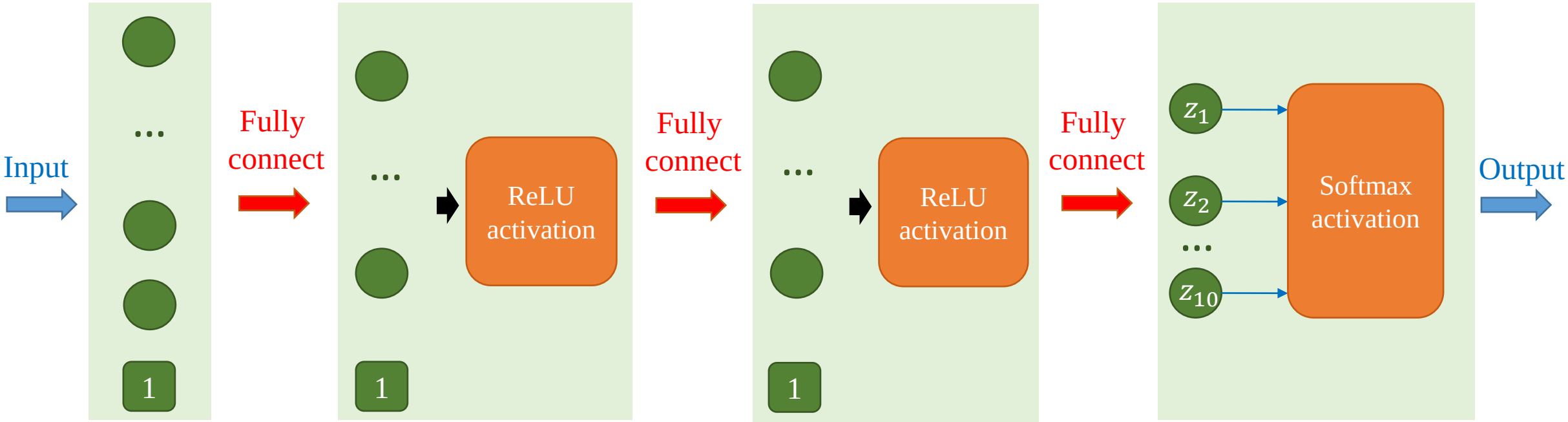


Perform disappointedly

MLP for Cifar-10

Case 3

❖ ReLU, He and Adam: add more layers



MLP for Cifar-10

Case 3

❖ ReLU, He and Adam

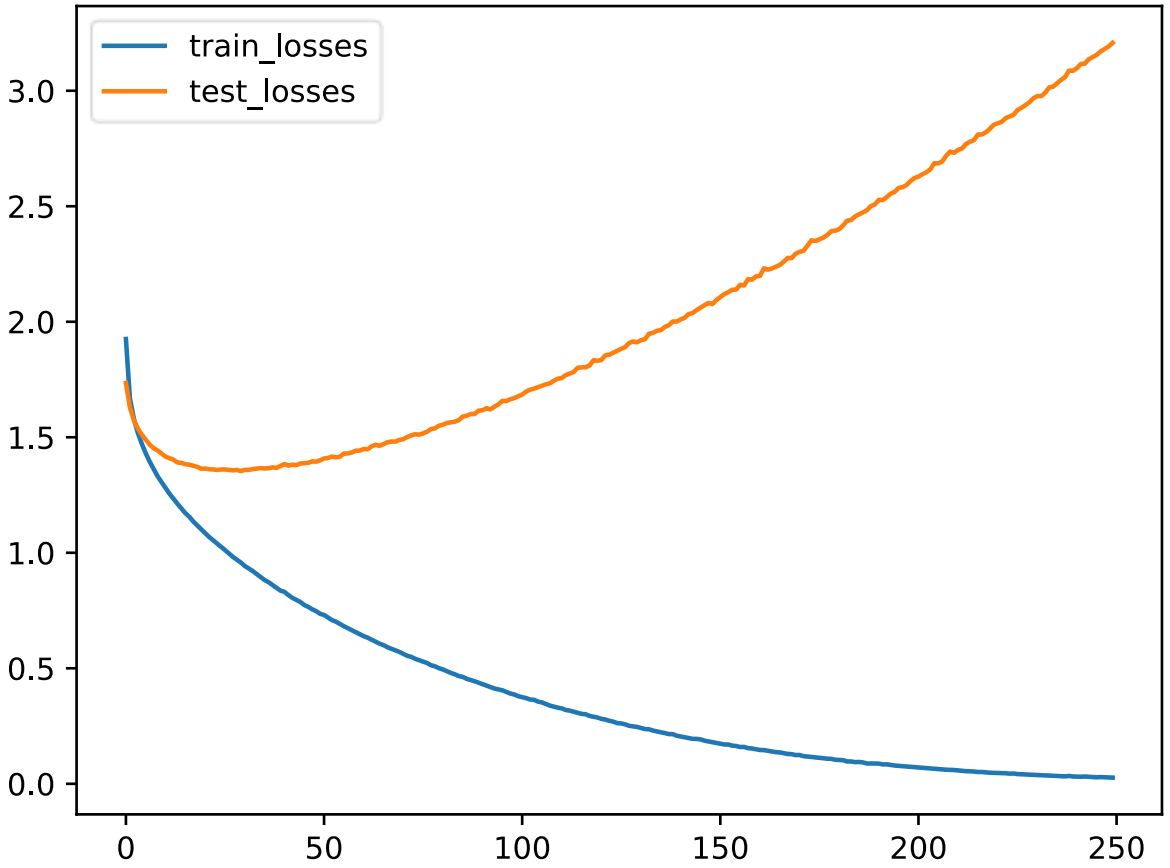
```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 256),
    nn.ReLU(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

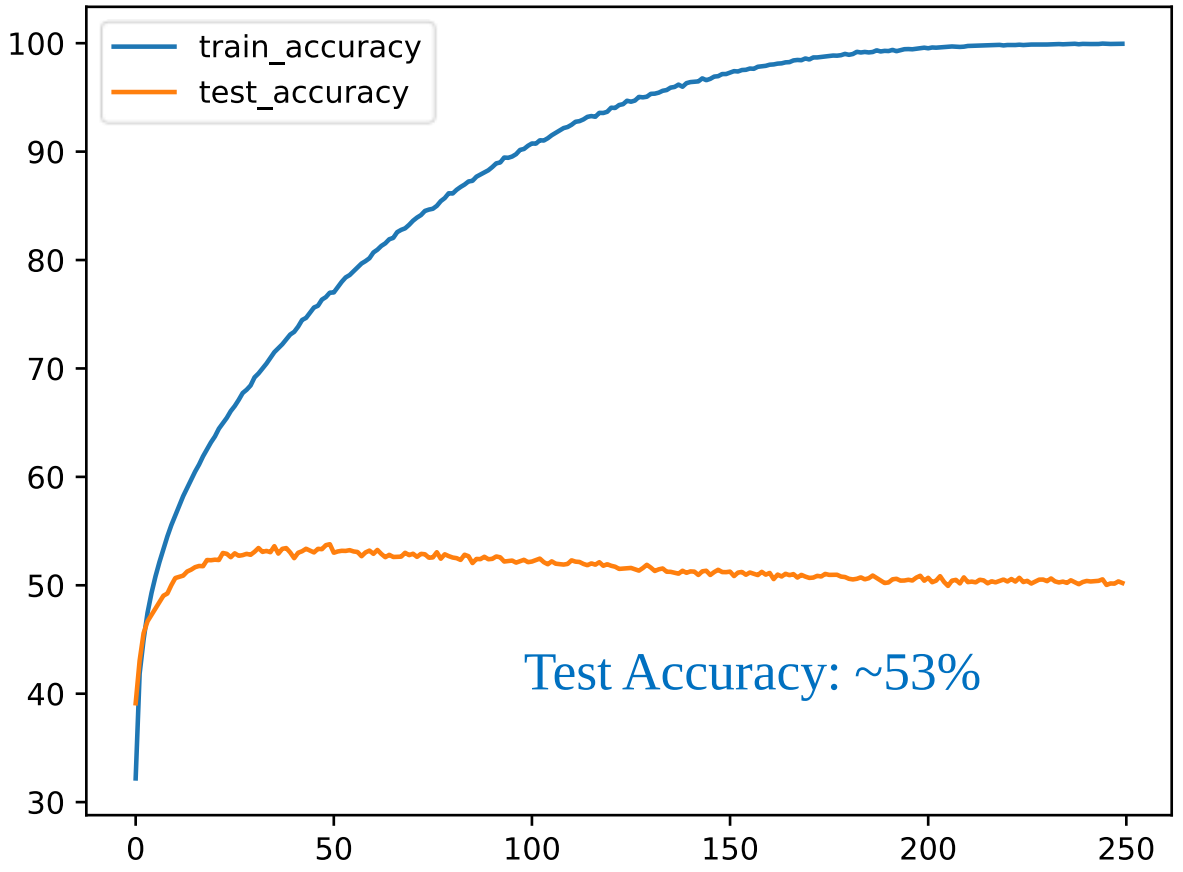
```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP for Cifar-10

Case 3

❖ ReLU, He and Adam



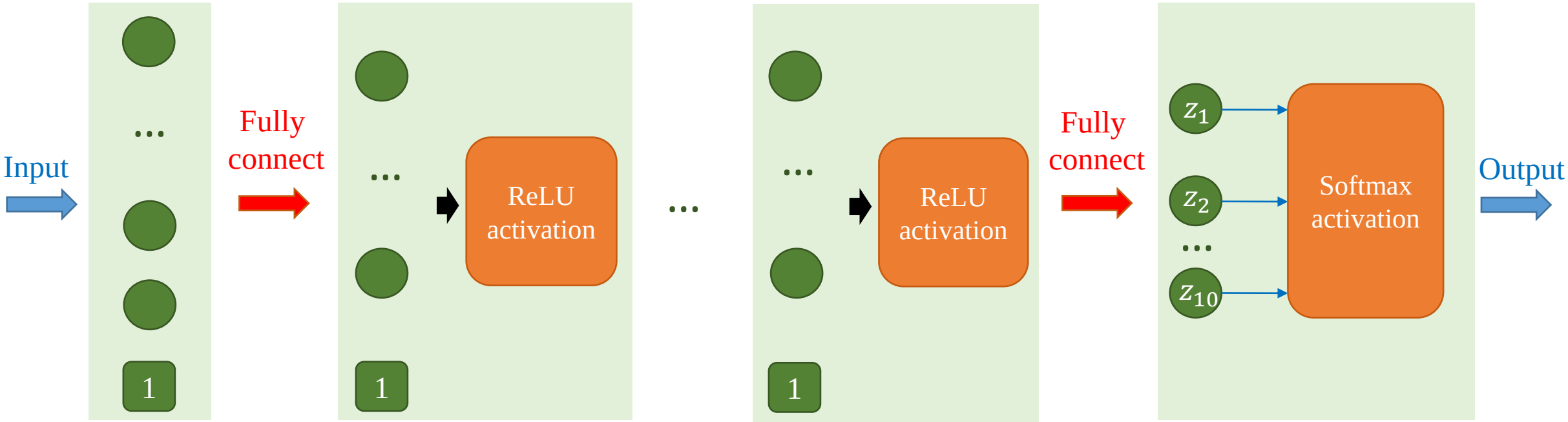


Still Perform poorly

MLP for Cifar-10

Case 4

❖ ReLU, He and Adam



MLP for Cifar-10

Case 4

❖ ReLU, He and Adam

```
# model
model = nn.Sequential(
    nn.Flatten(), nn.Linear(32*32*3, 256),
    nn.ReLU(), nn.Linear(256, 256),
    nn.ReLU(), nn.Linear(256, 256),
    nn.ReLU(), nn.Linear(256, 10)
)
```

```
# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)
```

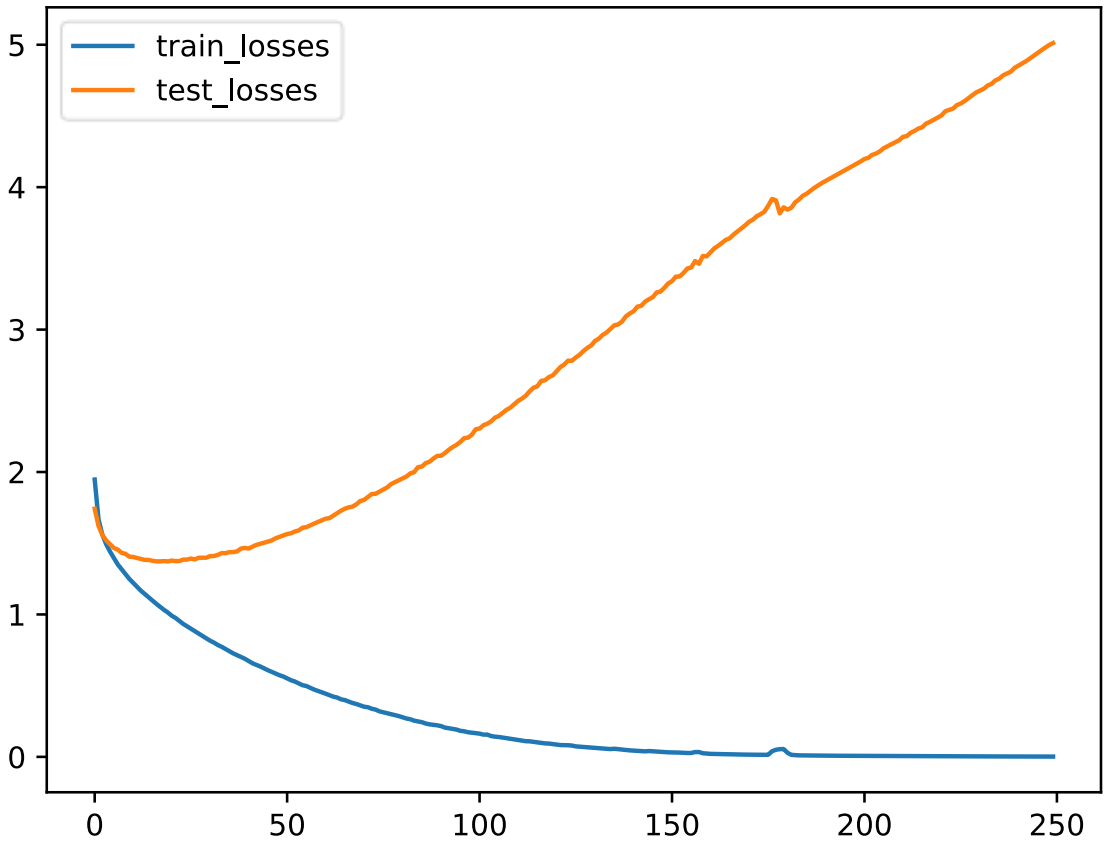
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

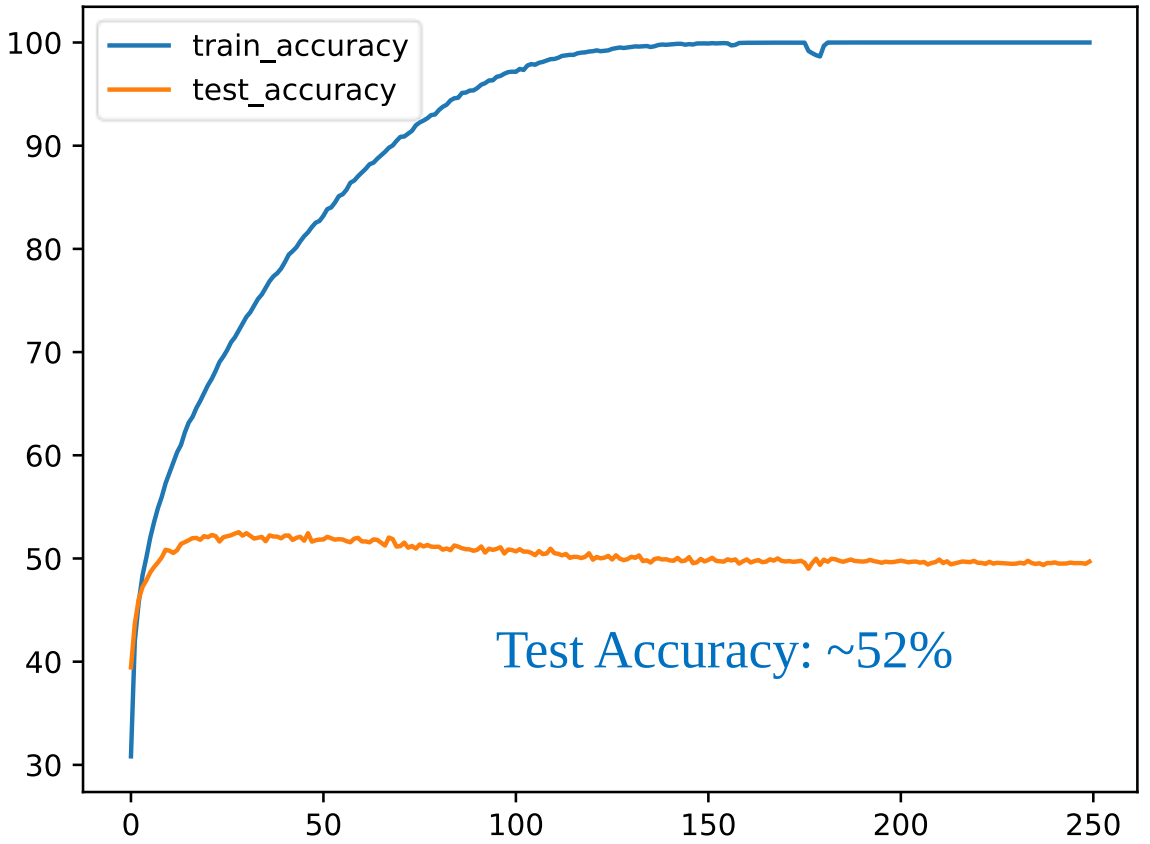
```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP for Cifar-10

Case 4

❖ ReLU, He and Adam: Using 3 hidden layers





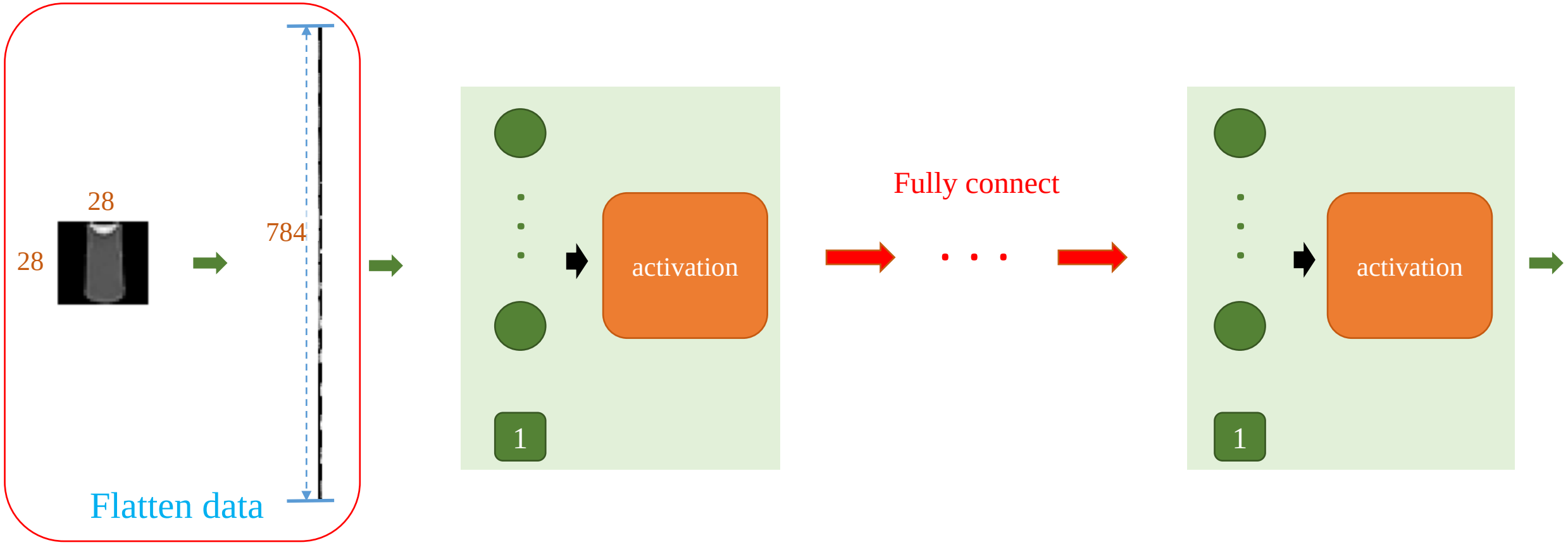
Perform even worse

Outline

- MLP Limitation
- From MLP to CNN
- Feature Map Down-sampling
- Some Examples
- Application to Cifar10

From MLP to CNN

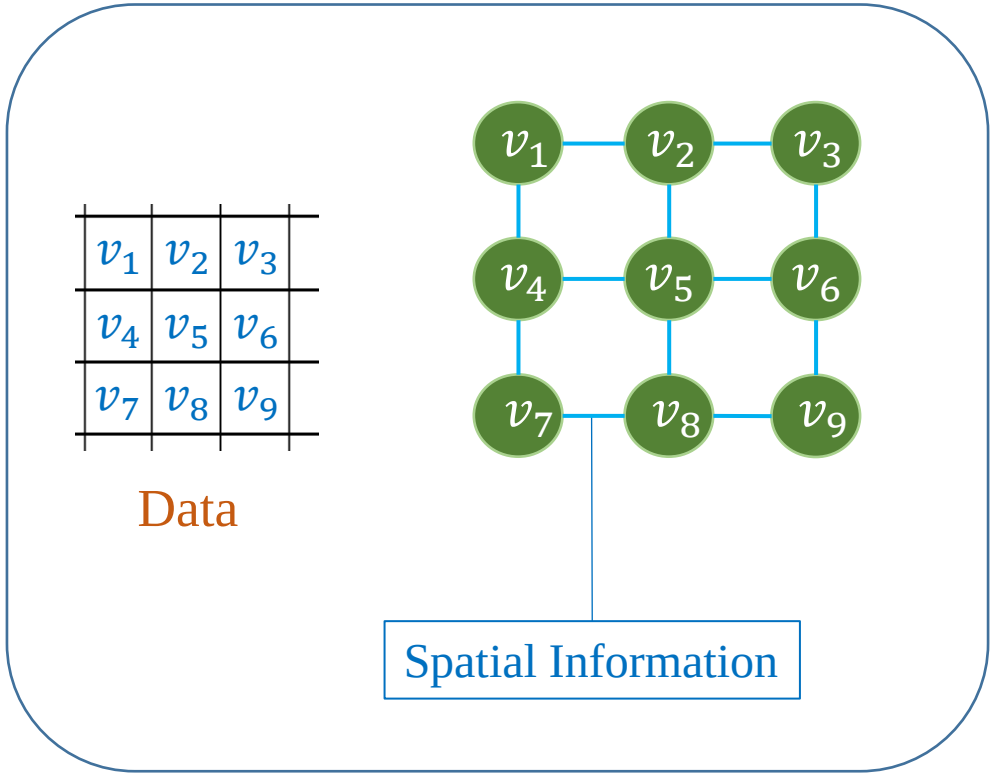
❖ Multi-layer Perceptron

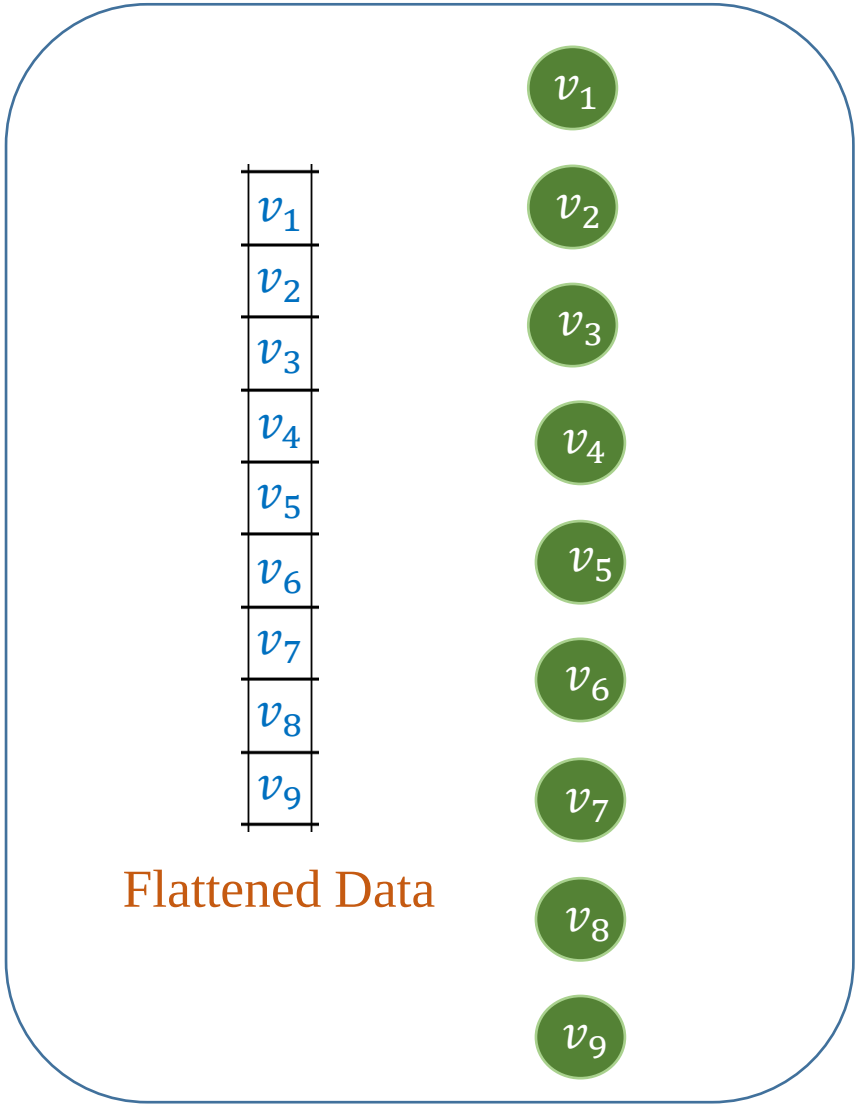


Problem: Remove spatial information of the data
Inefficiently have a large amount of parameters

From MLP to CNN

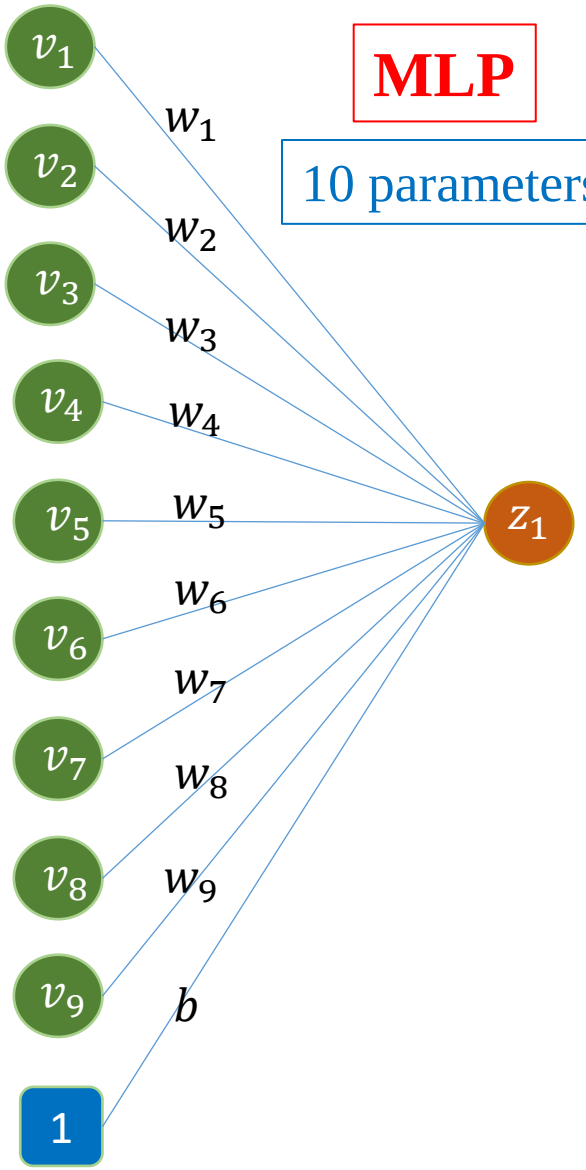
❖ Problem of flattening data





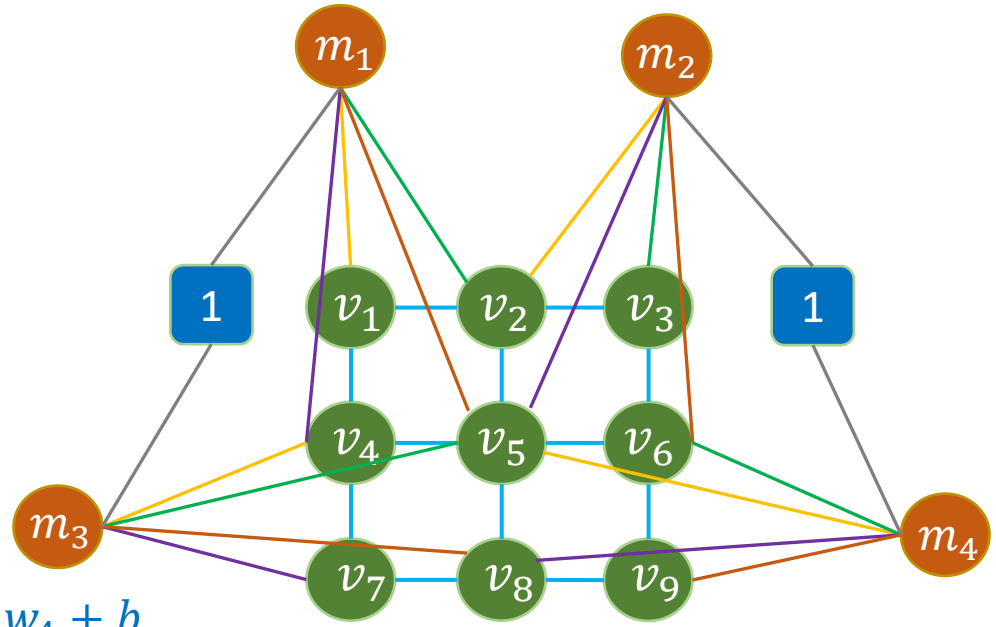
Remove spatial information of the data

From MLP to CNN

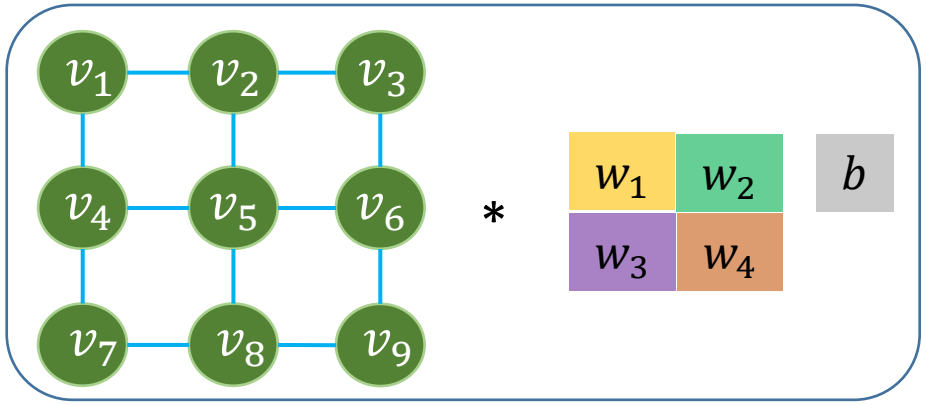


CNN 5 parameters

Kernel of parameters

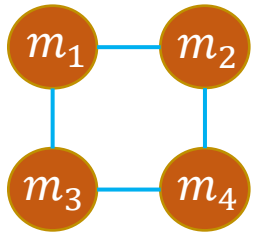


$$m_1 = v_1 w_1 + v_2 w_2 + v_4 w_3 + v_5 w_4 + b$$



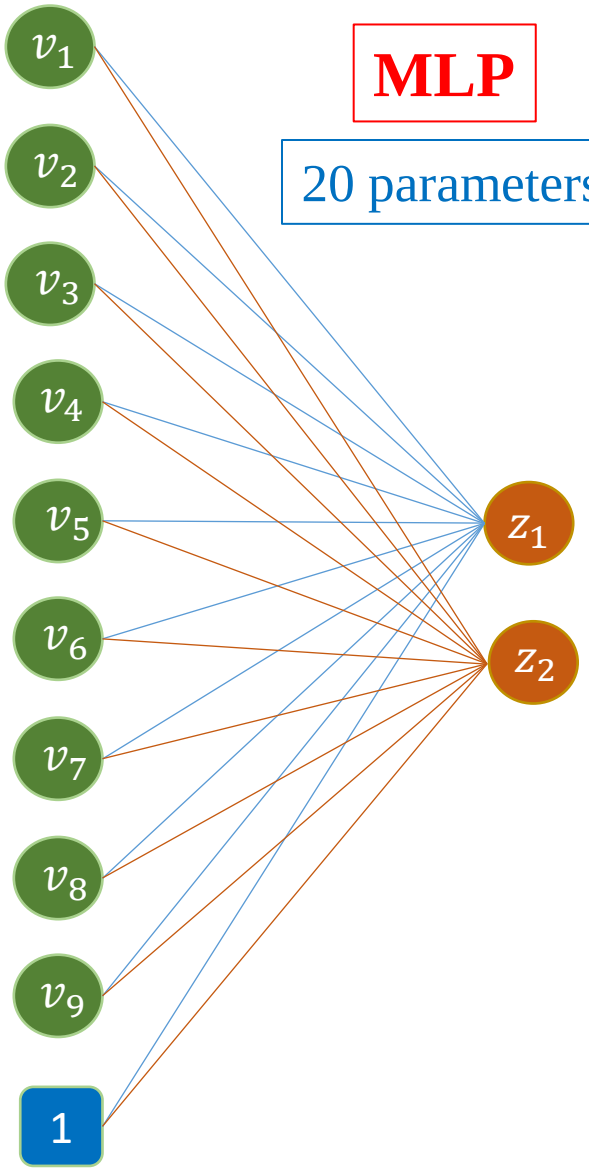
Convolution (=Correlation)

➡



Feature Map

From MLP to CNN

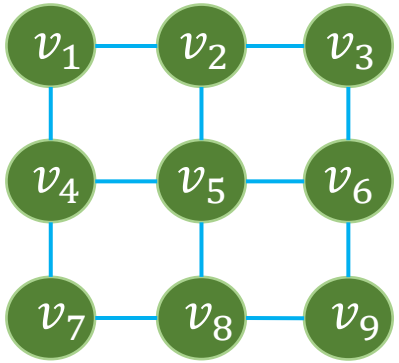


MLP

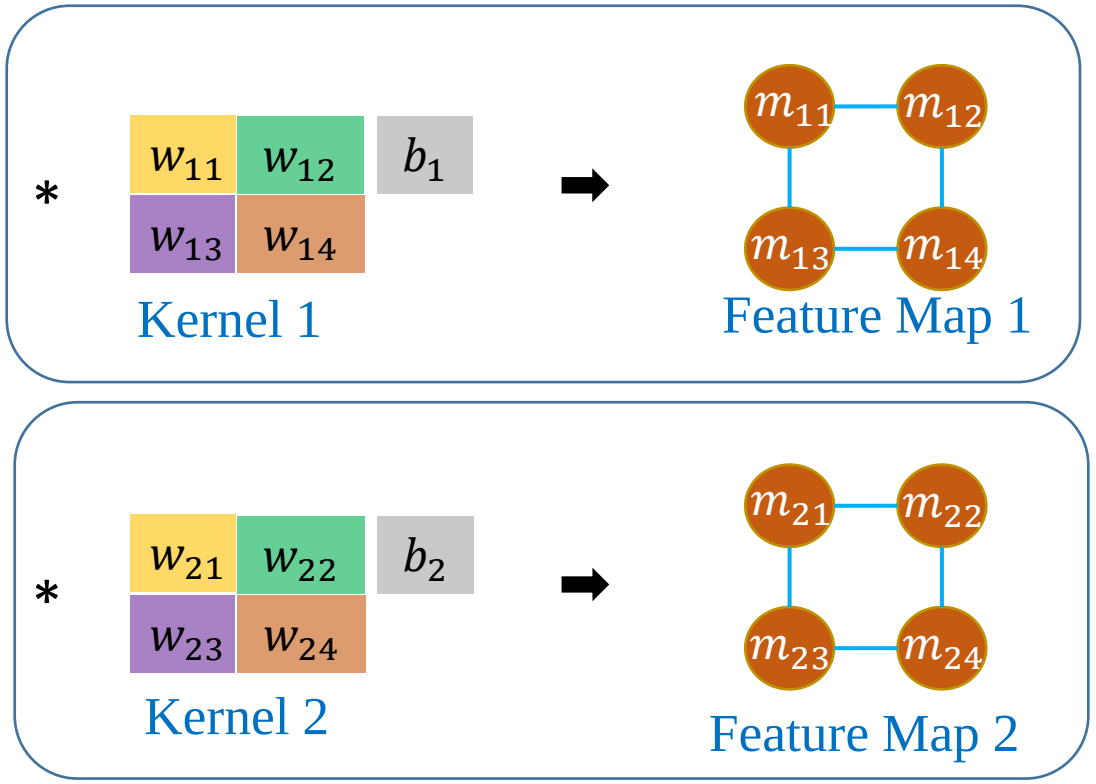
20 parameters

CNN

10 parameters



Kernel 1 ≠ Kernel 2

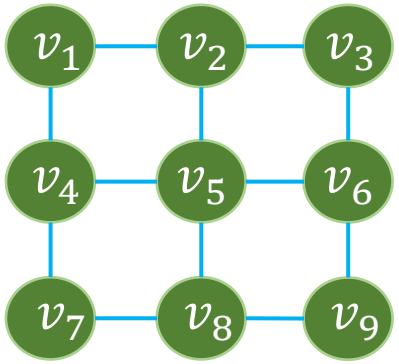


Global vs. Local?

Parameter size?

From MLP to CNN

❖ Understand convolution



Shape=(1,3,3)

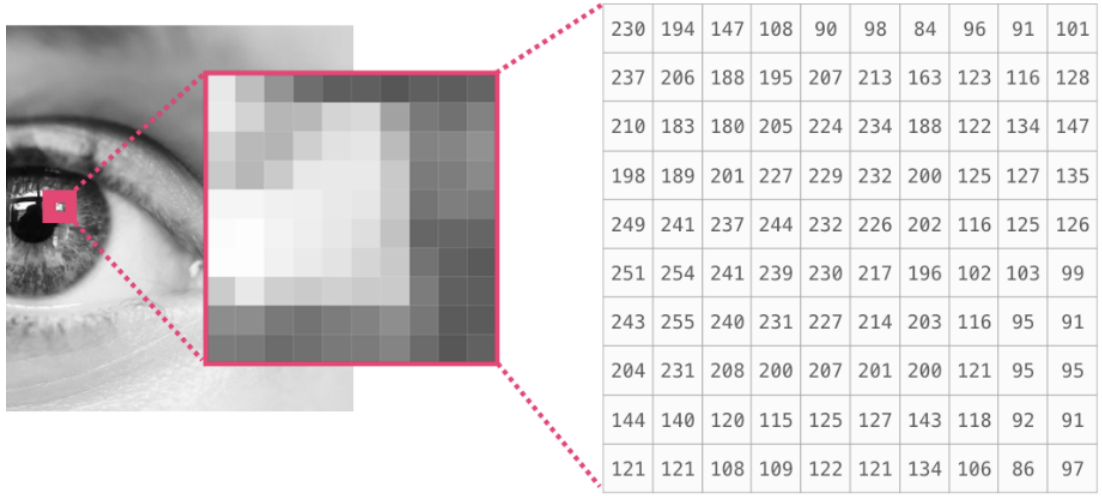
(Channel=1, Height=3, Width=3)

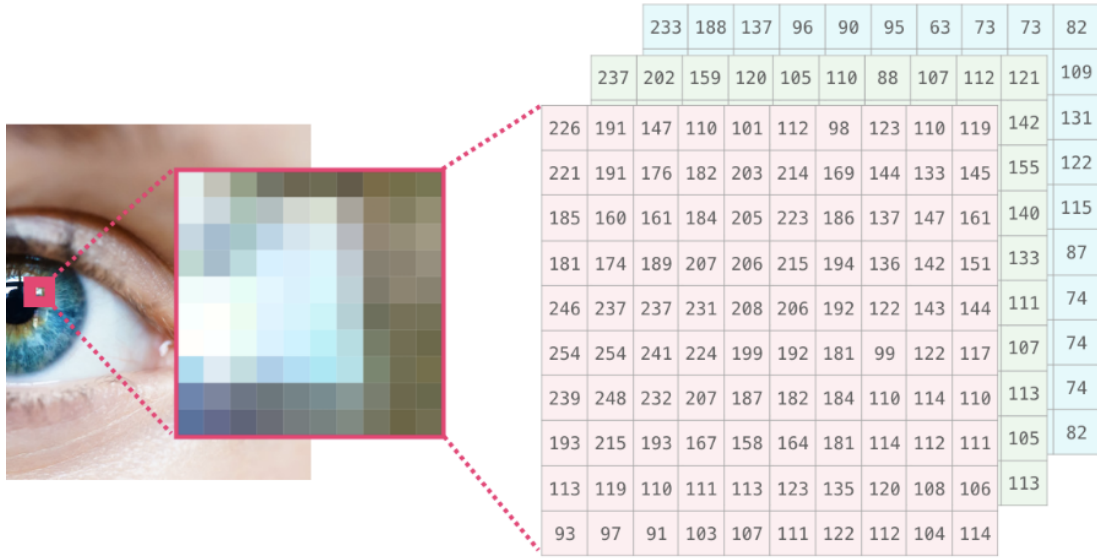
*

b

Shape=(1,2,2)

#parameters (+bias) = 5





#channels of data must = #channels of kernel

Convolution

❖ **How many cases?**

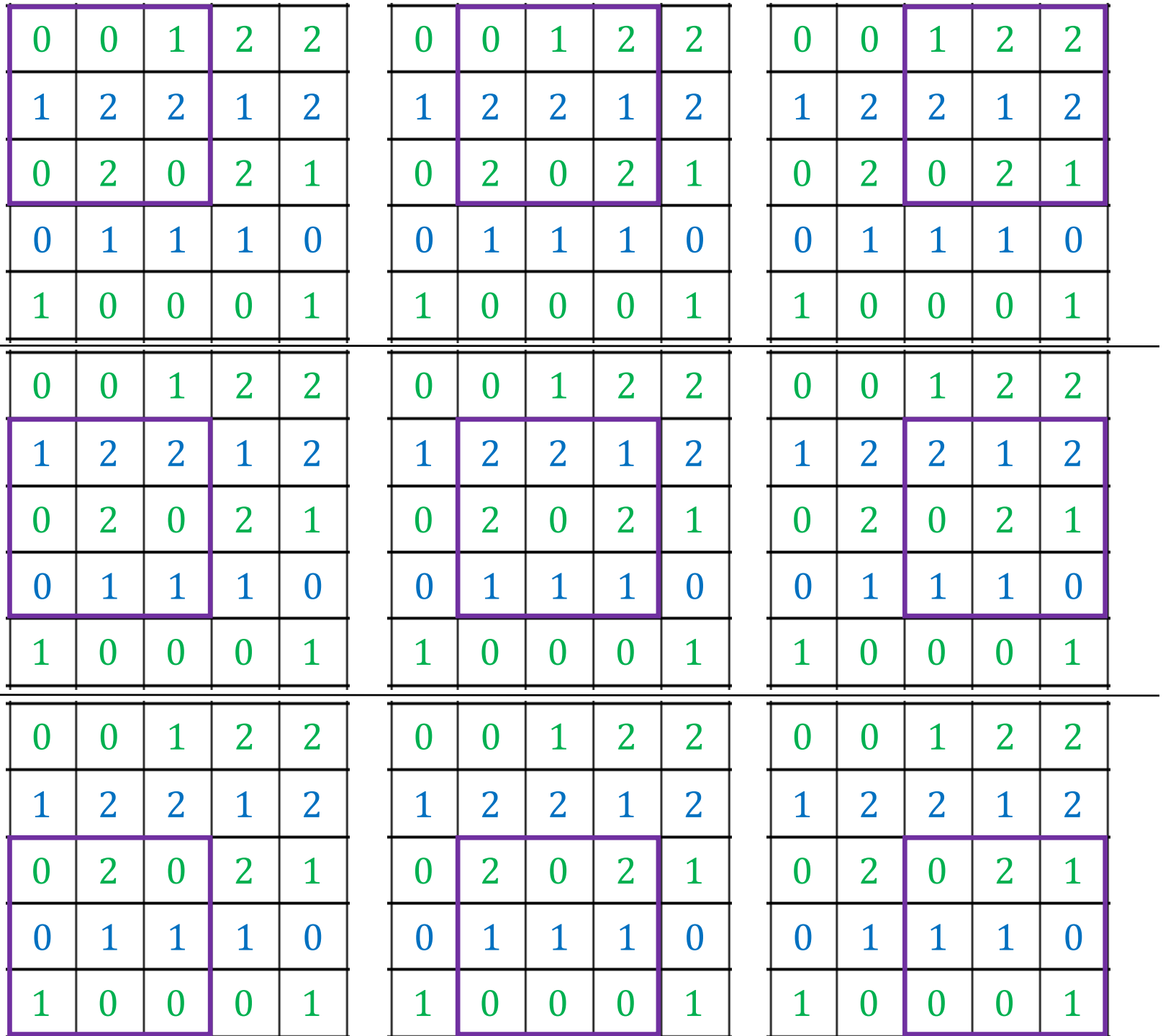
0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Data **D**

Kernel **K**

0.0	0.1	-0.1
-0.2	0.0	0.1
0.0	0.0	0.1

Bias b = 0.0



Convolution

❖ Example

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Data **D**

Bias b = 0.0

0.0	0.1	-0.1
-0.2	0.0	0.1
0.0	0.0	0.1

Kernel **K**

m_1		

Output

Data size = 5×5

Kernel size = 3×3

Stride = 1

$m_1 = 0\times0.0 + 0\times0.1 + 1\times-0.1 +$

$1\times-0.2 + 2\times0.0 + 2\times0.1 +$

$0\times0.0 + 2\times0.0 + 0\times0.1$

➡ $m_1 = -0.1$

Convolution

❖ Example

$$S_o = \left\lfloor \frac{S_D - K}{S} \right\rfloor + 1$$

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Data **D**

Bias b = 0.0

0.0	0.1	-0.1
-0.2	0.0	0.1
0.0	0.0	0.1

Kernel **K**

-0.1	-0.1	-0.2
0.3	-0.2	0.1
0.3	-0.3	0.1

Output

Data size = 5×5

Kernel size = 3×3

Stride = 1

Convolution

❖ Example

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Data **D**

Bias b = 0.0

0.0	0.1	-0.1
-0.2	0.0	0.1
0.0	0.0	0.1

Kernel **K**

m_1	

Output

Data size = 5×5

Kernel size = 3×3

Stride = 2

m_1 = 0×0.0 + 0×0.1 + 1×−0.1 +

1×−0.2 + 2×0.0 + 2×0.1 +

0×0.0 + 2×0.0 + 0×0.1

➡ $m_1 = -0.1$

Convolution

❖ Example

$$S_o = \left\lfloor \frac{S_D - K}{S} \right\rfloor + 1$$

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Data **D**

Bias b = 0.0

0.0	0.1	-0.1
-0.2	0.0	0.1
0.0	0.0	0.1

Kernel **K**

-0.1	-0.2
0.3	0.1

Output

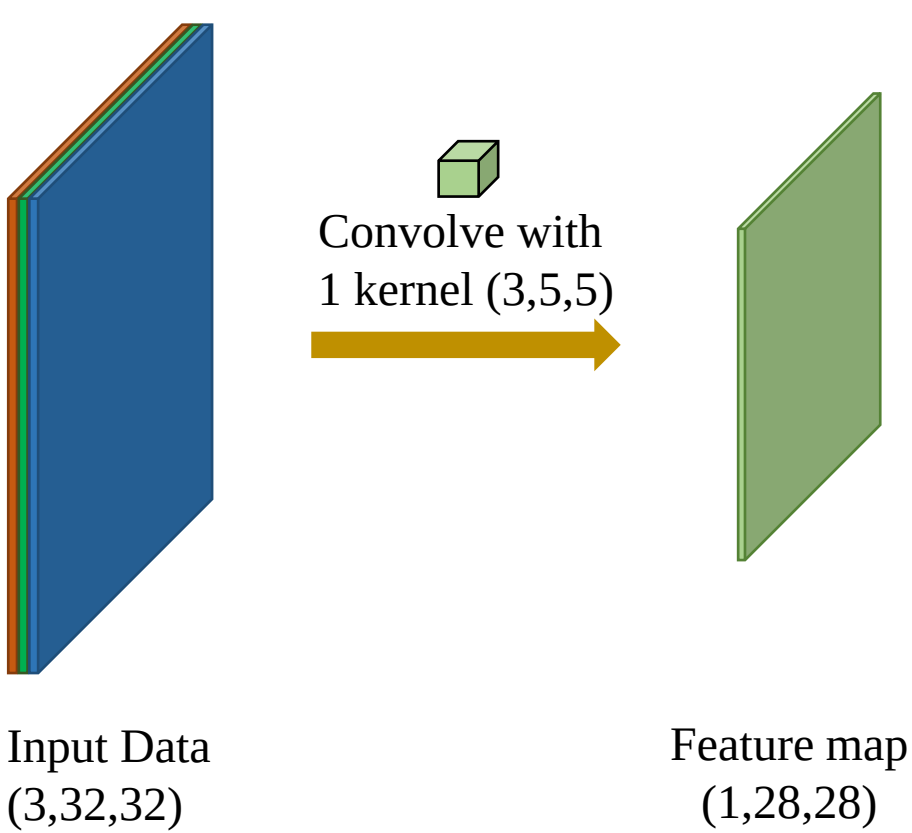
Data size = 5×5

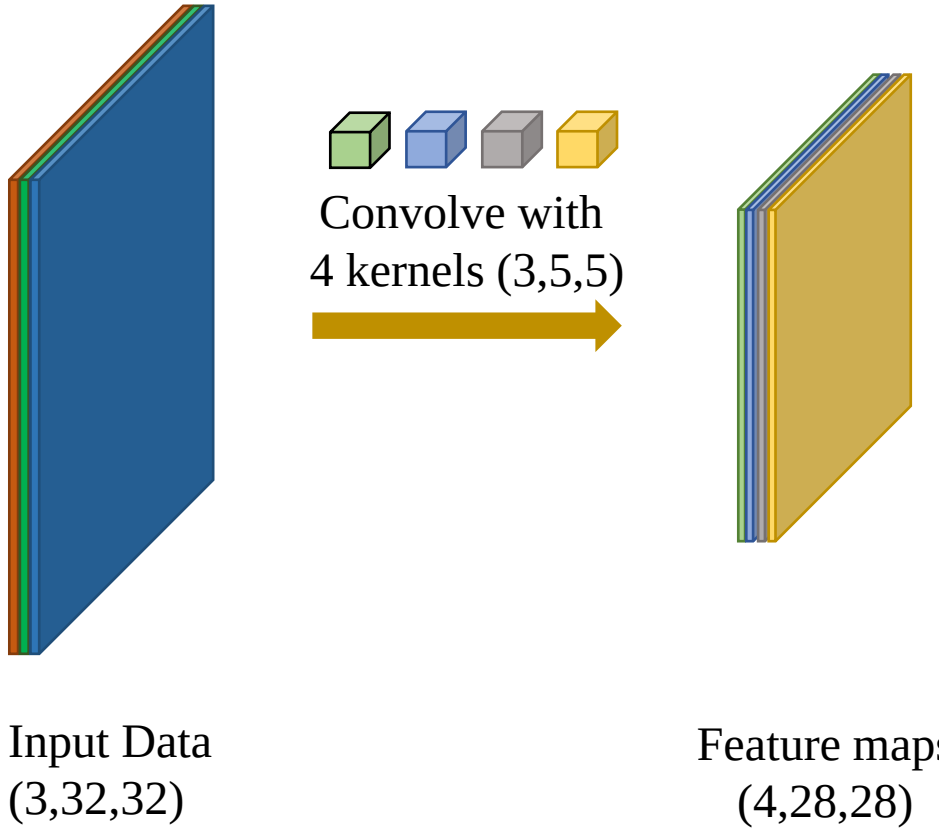
Kernel size = 3×3

Stride = 2

Convolutional Neural Network

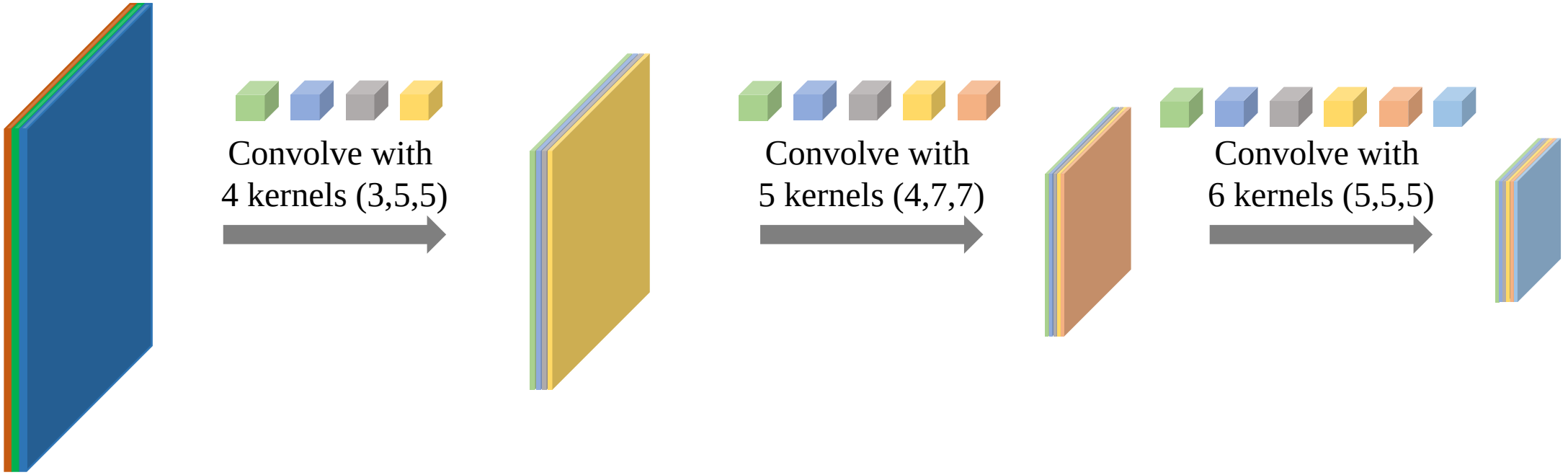
❖ Understand convolution





Convolutional Neural Network

❖ A stack of convolutions



Input Data
(3,32,32)

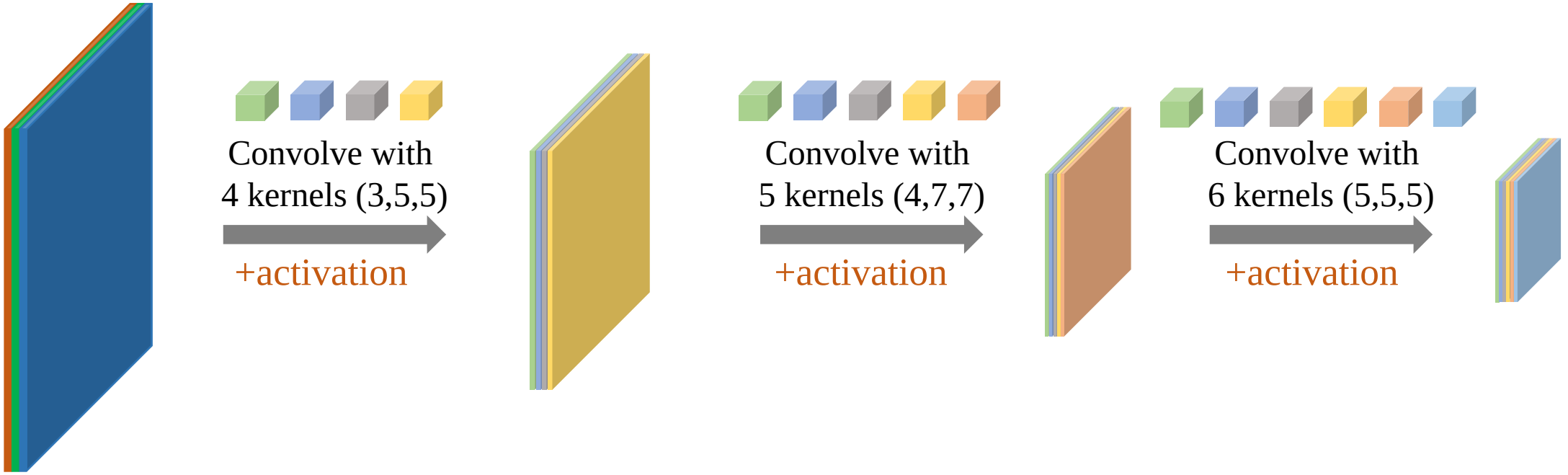
Feature maps
(4,28,28)

Feature maps
(5,22,22)

Feature maps
(6,18,18)

Convolutional Neural Network

❖ A stack of pairs of convolution+activation



Input Data
(3,32,32)

Feature maps
(4,28,28)

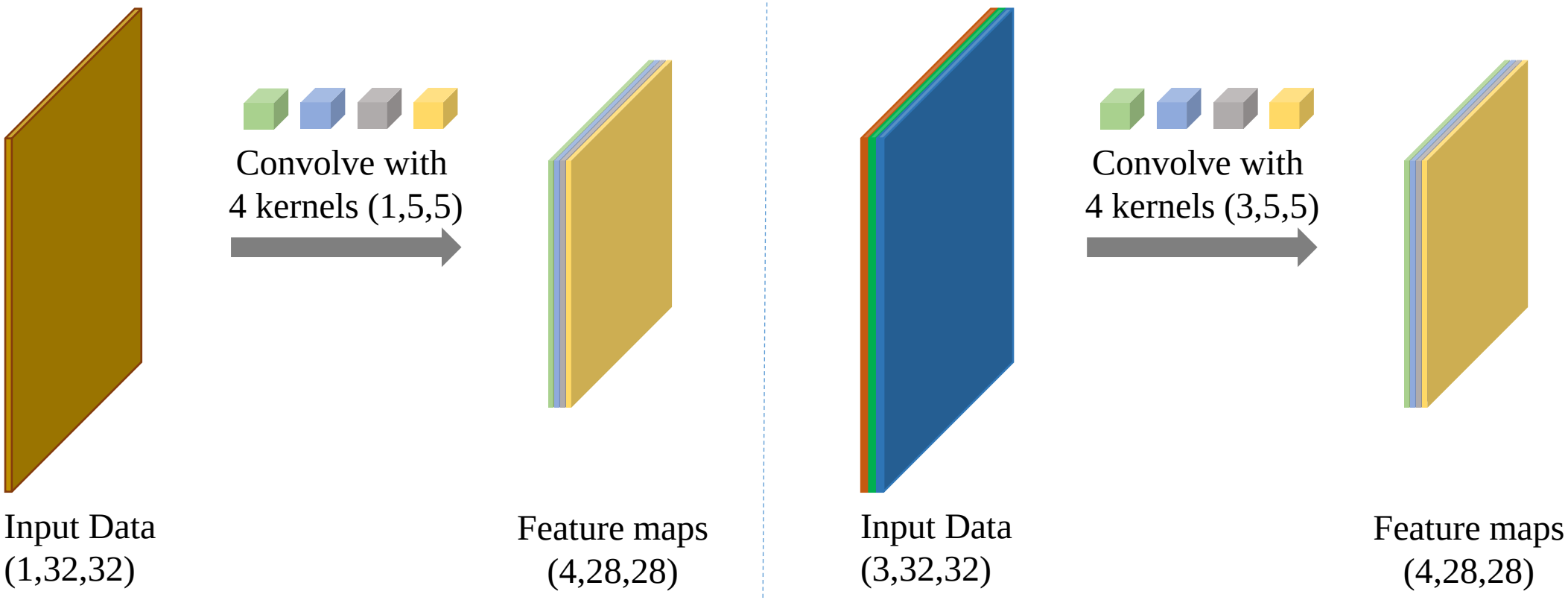
Feature maps
(5,22,22)

Feature maps
(6,18,18)

Convolutional Neural Network

❖ Convolution layer in PyTorch

nn.Conv2d(in_channels, out_channels, kernel_size)

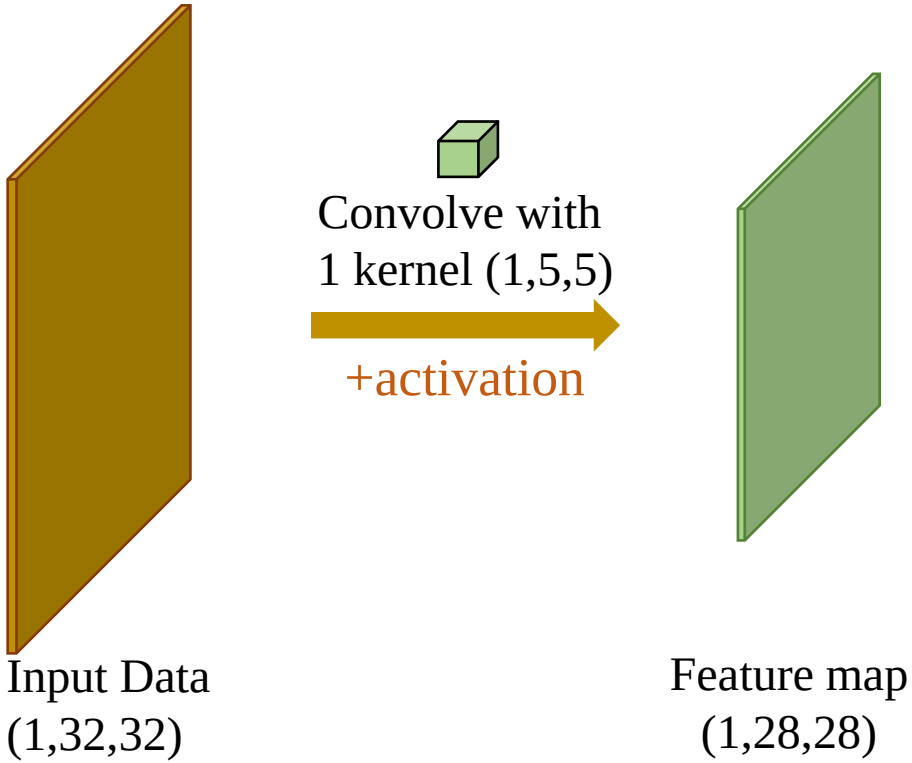


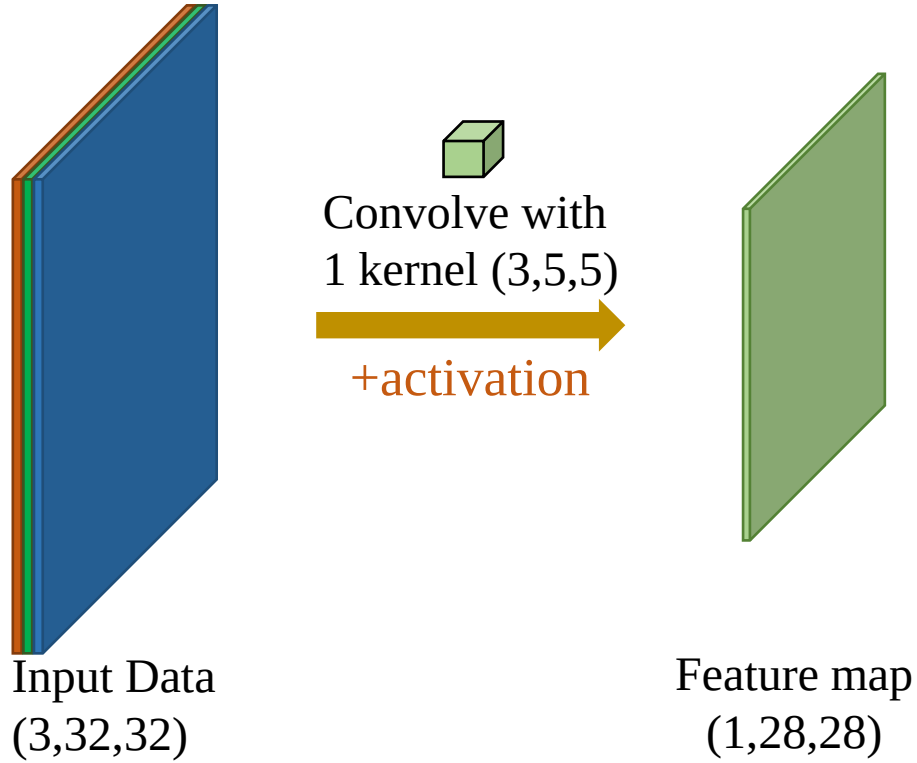
Convolutional Neural Network

❖ Convolution layer in PyTorch

demo

```
nn.Conv2d(in_channels, out_channels, kernel_size)
nn.ReLU()
```





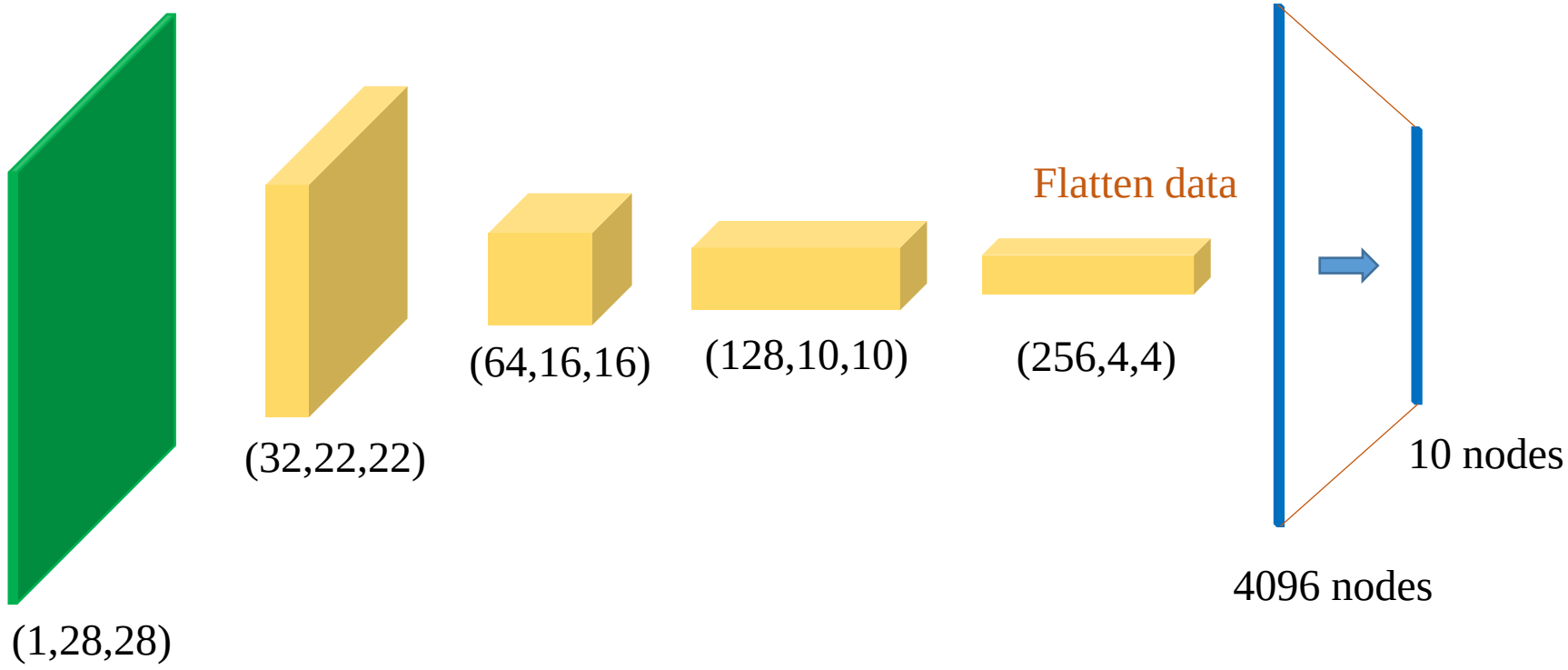
Convolutional Neural Network

Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

Convolutional Neural Network

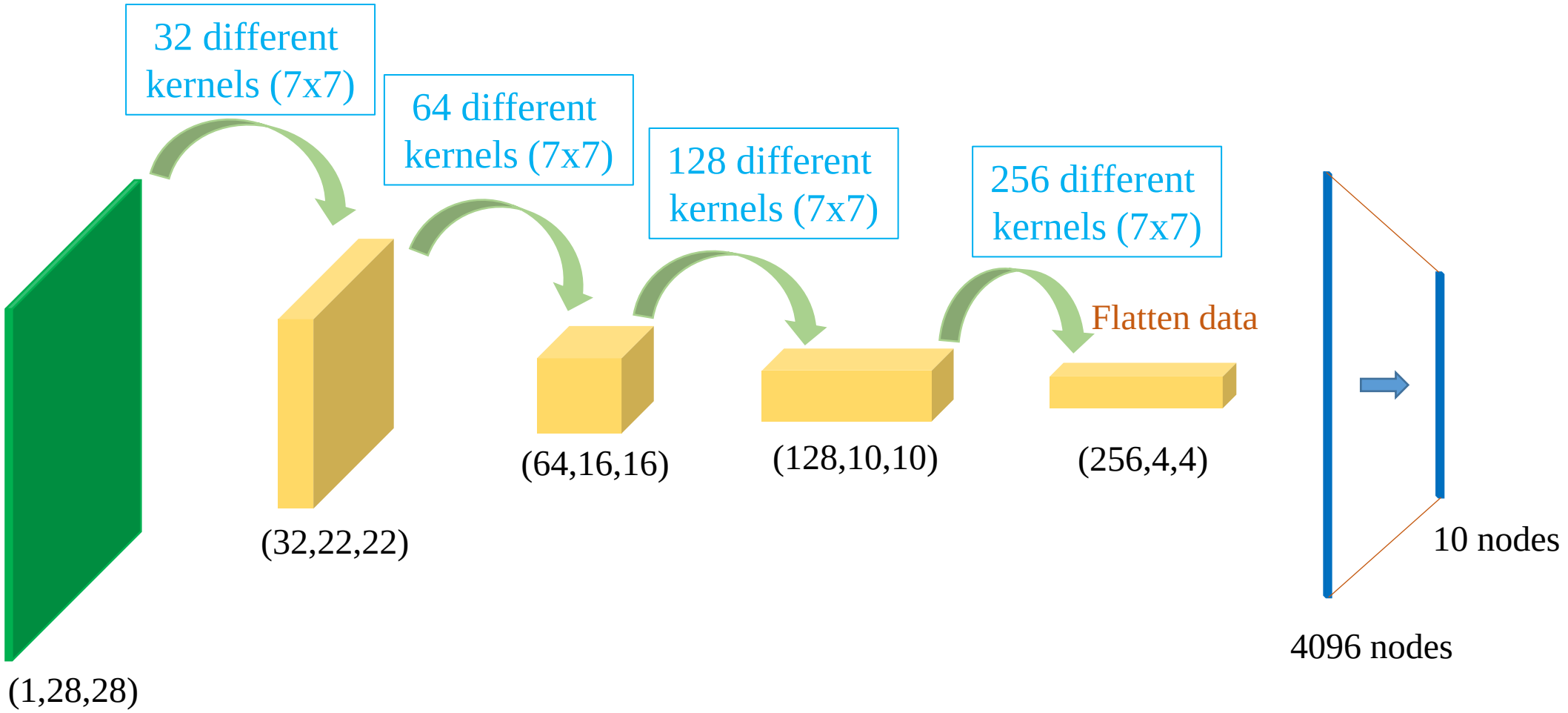
❖ Apply for Fashion-MNIST dataset



Convolutional Neural Network

❖ Apply for Fashion-MNIST dataset

demo



Simple Convolutional Neural Network

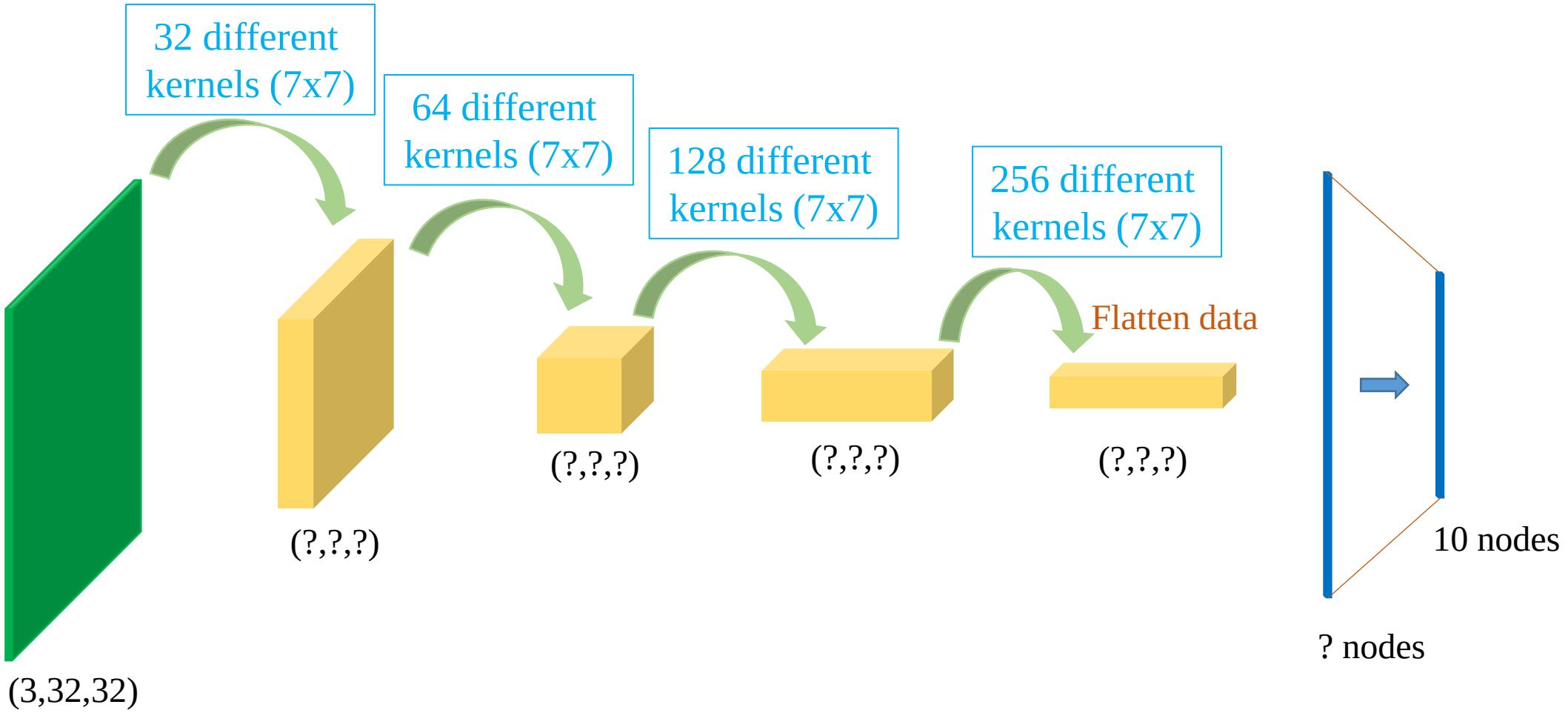
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(4*4*256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.dense(x)
        return x

model = CustomModel()
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 22, 22]           1,600
              ReLU-2          [-1, 32, 22, 22]               0
            Conv2d-3          [-1, 64, 16, 16]         100,416
              ReLU-4          [-1, 64, 16, 16]               0
            Conv2d-5         [-1, 128, 10, 10]         401,536
              ReLU-6         [-1, 128, 10, 10]               0
            Conv2d-7           [-1, 256, 4, 4]       1,605,888
              ReLU-8           [-1, 256, 4, 4]               0
           Flatten-9                [-1, 4096]               0
           Linear-10                 [-1, 128]         524,416
             ReLU-11                 [-1, 128]               0
           Linear-12                  [-1, 10]           1,290
================================================================
Total params: 2,635,146
Trainable params: 2,635,146
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.78
Params size (MB): 10.05
Estimated Total Size (MB): 10.83
----------------------------------------------------------------
```

Convolutional Neural Network

❖ Apply for Cifar-10 dataset

demo

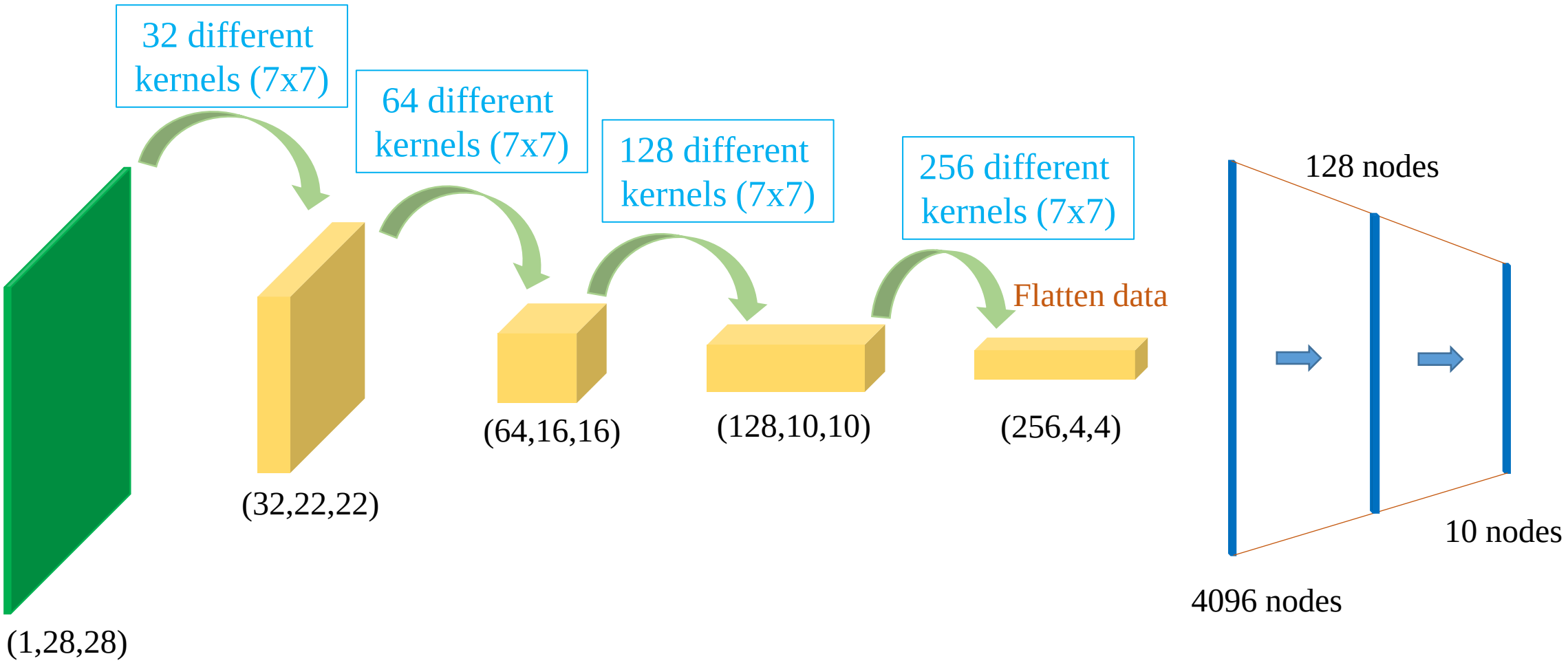


Outline

- MLP Limitation
- From MLP to CNN
- Feature Map Down-sampling
- Some Examples
- Application to Cifar10

Convolutional Neural Network

❖ Apply for Fashion-MNIST dataset: case 1

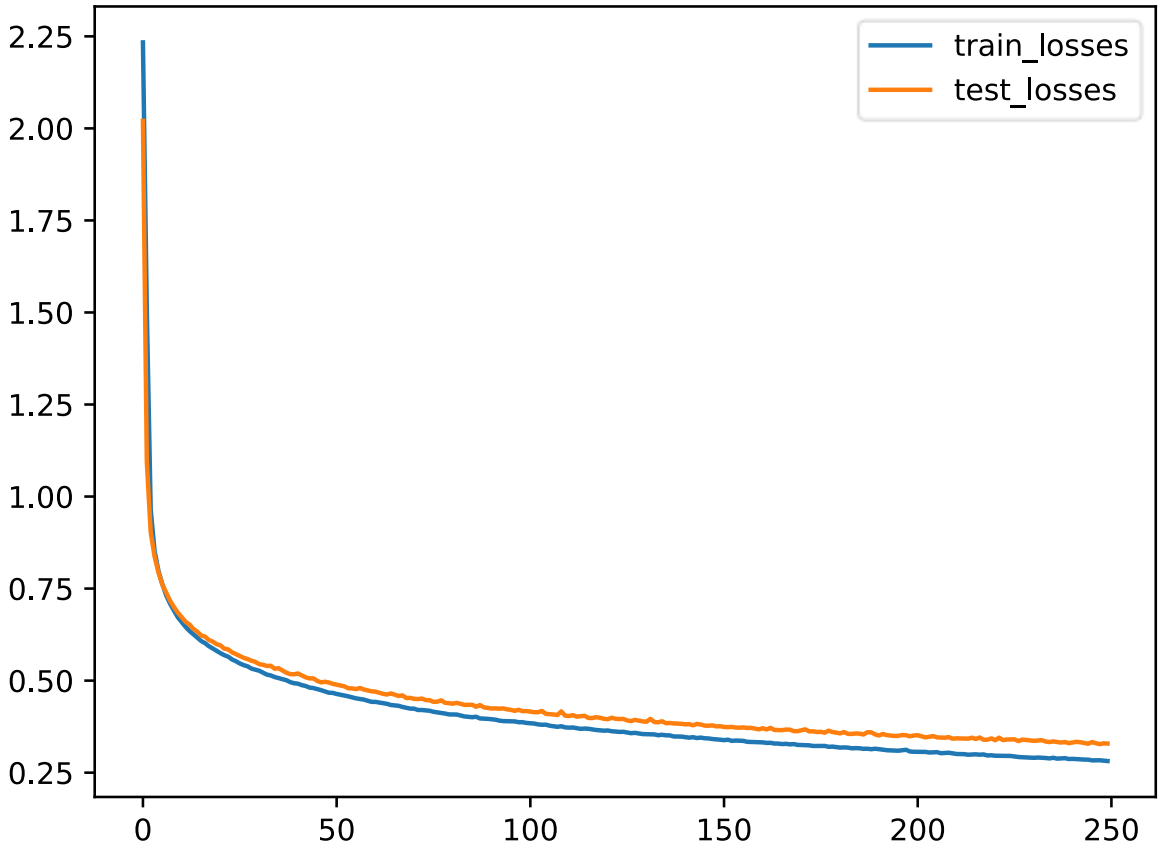


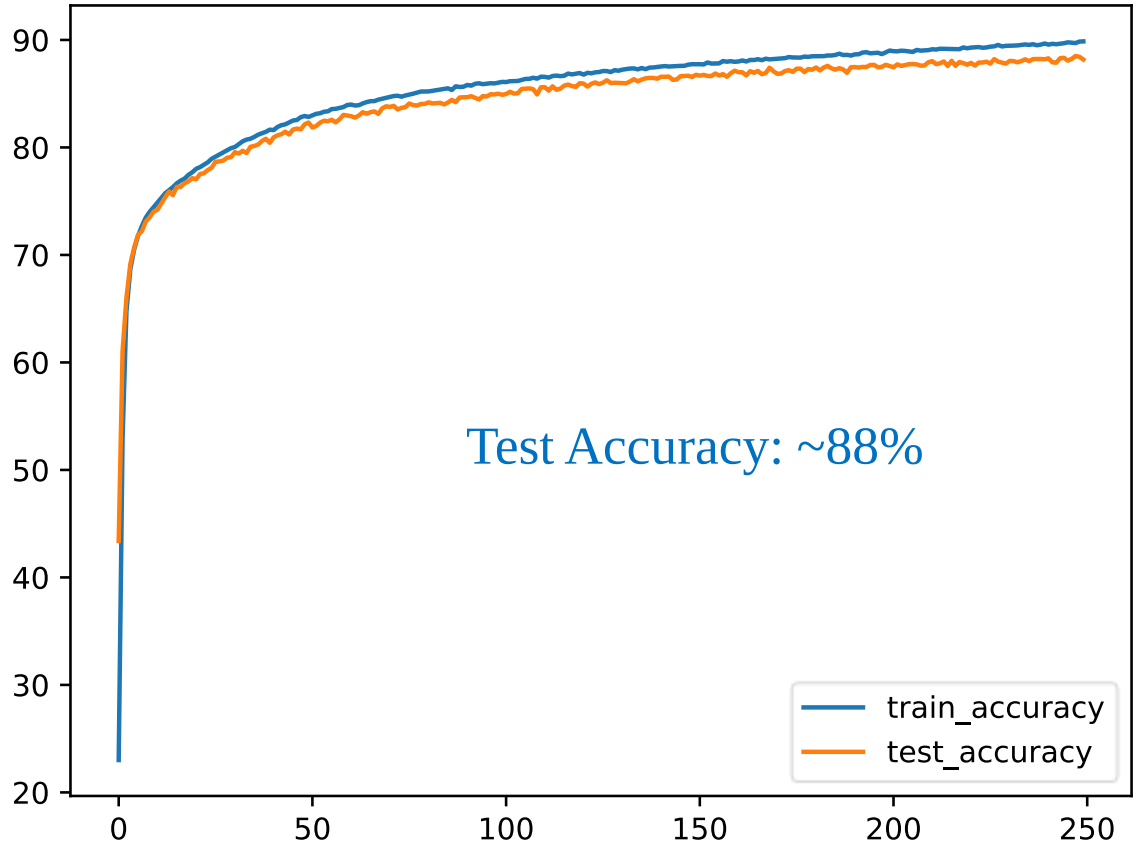
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(4*4*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)
```

```python
# Load FashionMNIST dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,),
                               (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

Convolutional Neural Network

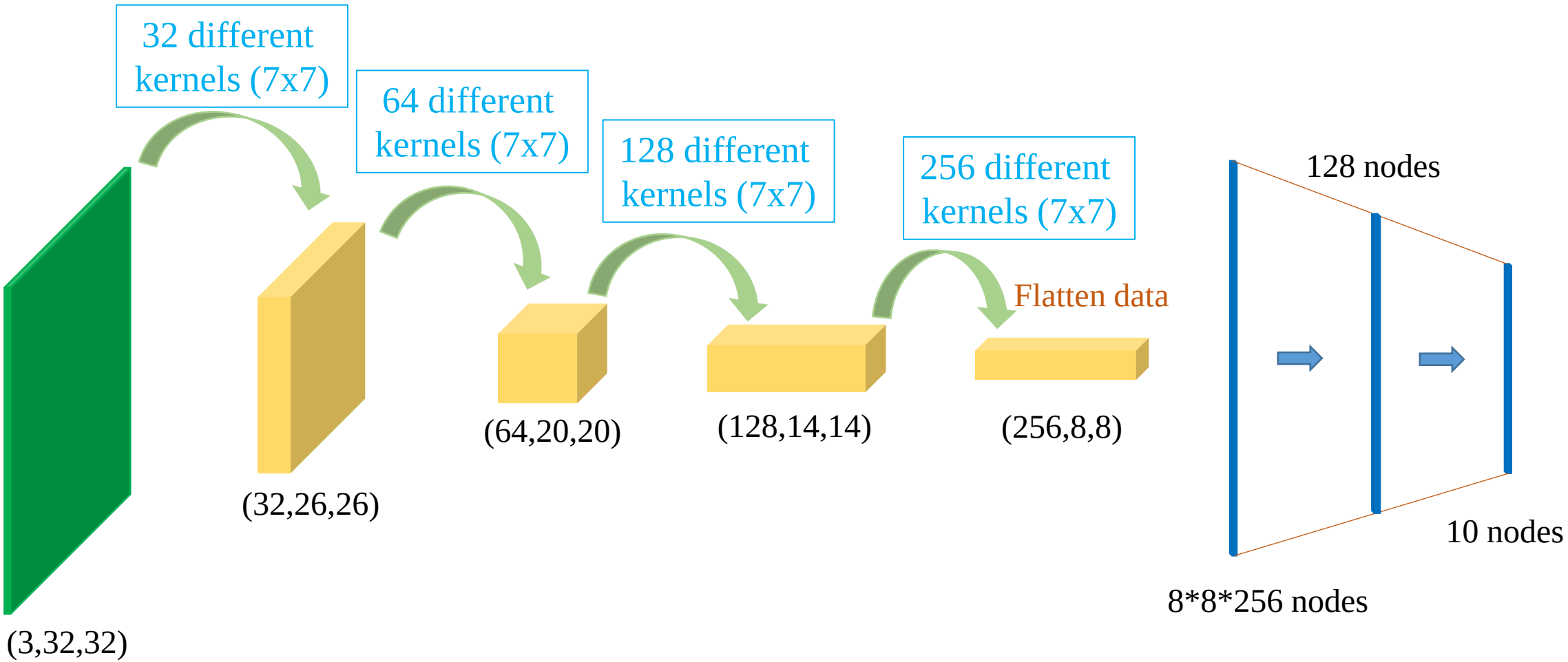
❖ Apply for Fashion-MNIST dataset: case 1





Convolutional Neural Network

❖ Apply for Cifar-10 dataset: case 2

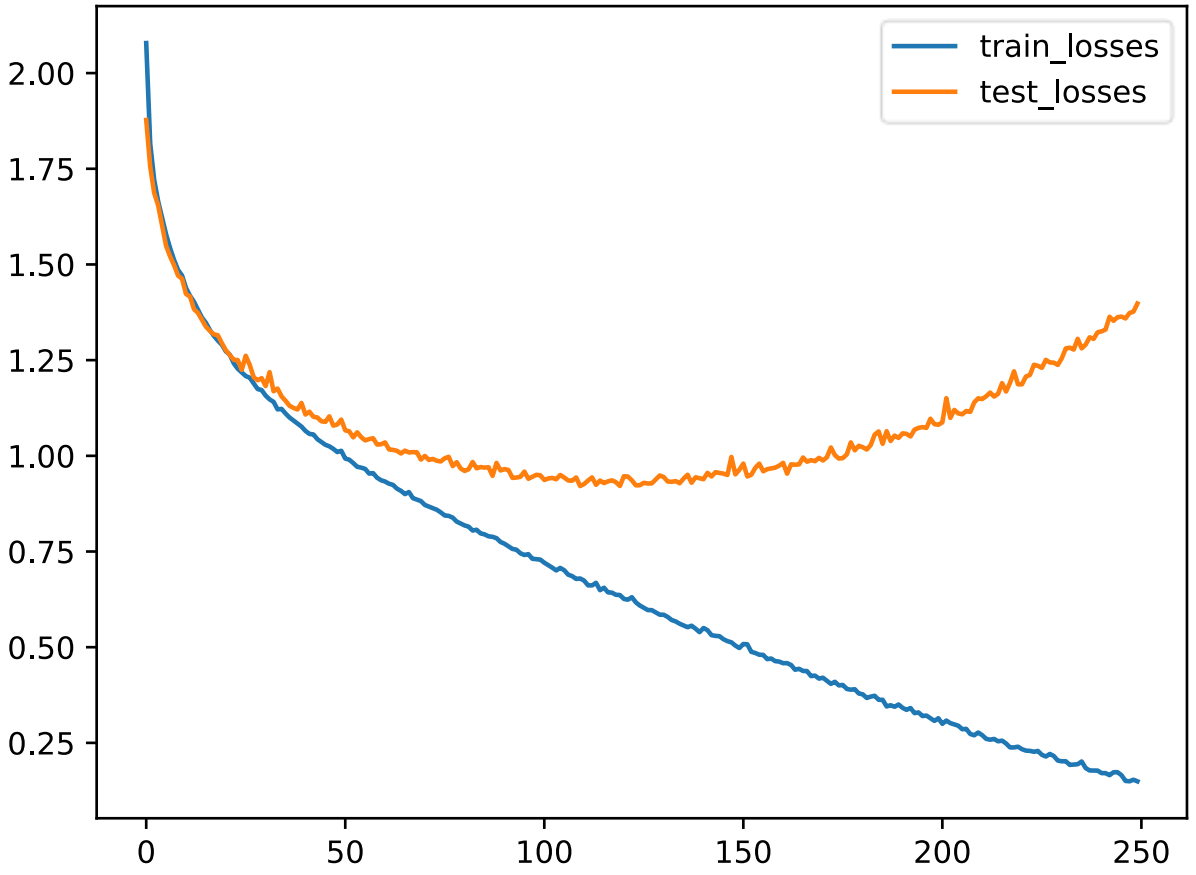


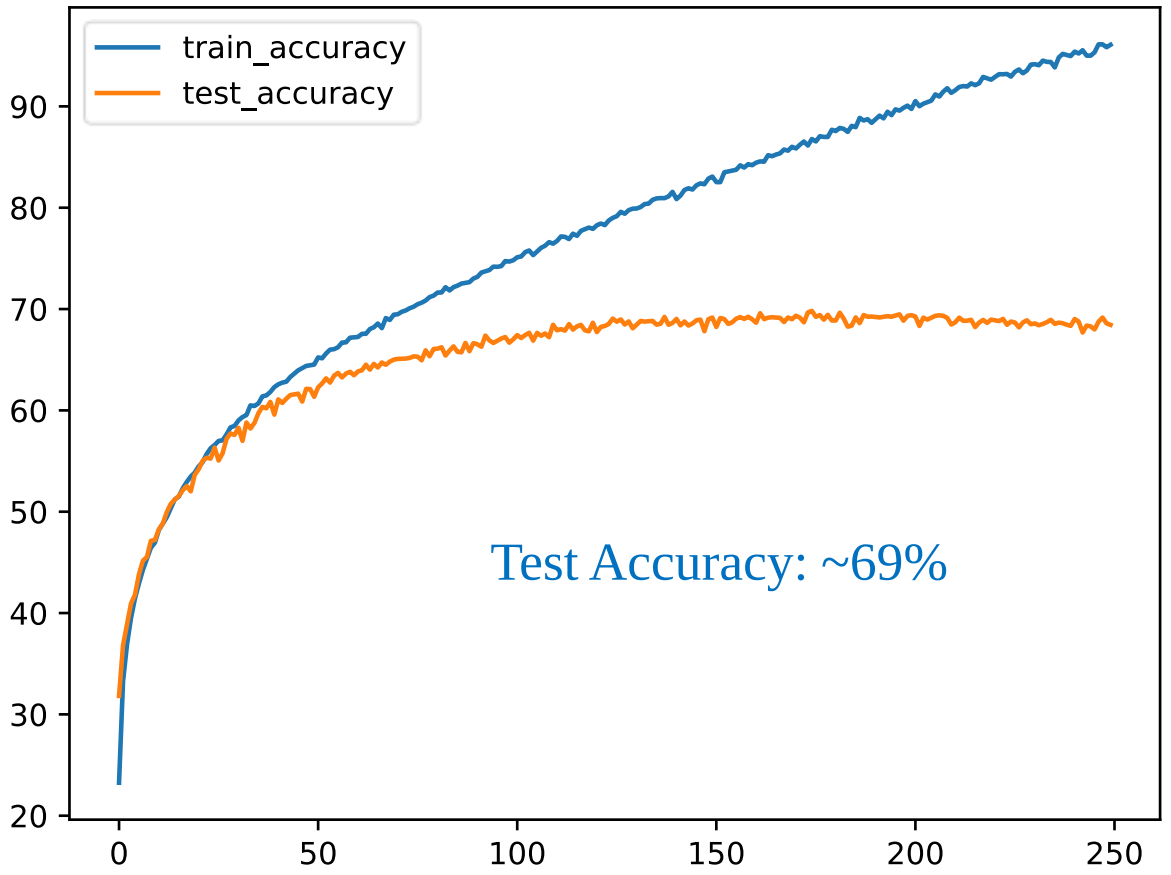
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(8*8*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)
```

```python
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

Convolutional Neural Network

❖ Apply for Cifar-10 dataset: case 2





Test Accuracy from MLP: ~53%

Further Reading

❖ Reading

https://cs231n.github.io/convolutional-networks/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

AI VIETNAM
All-in-One Course

Convolutional Neural Network

Construction of Standard CNNs

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

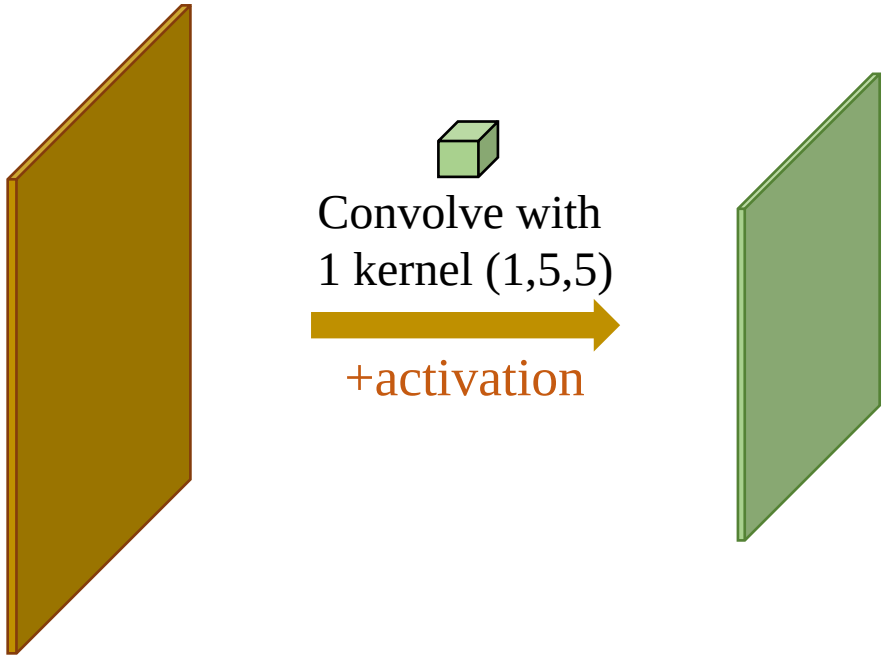
Outline

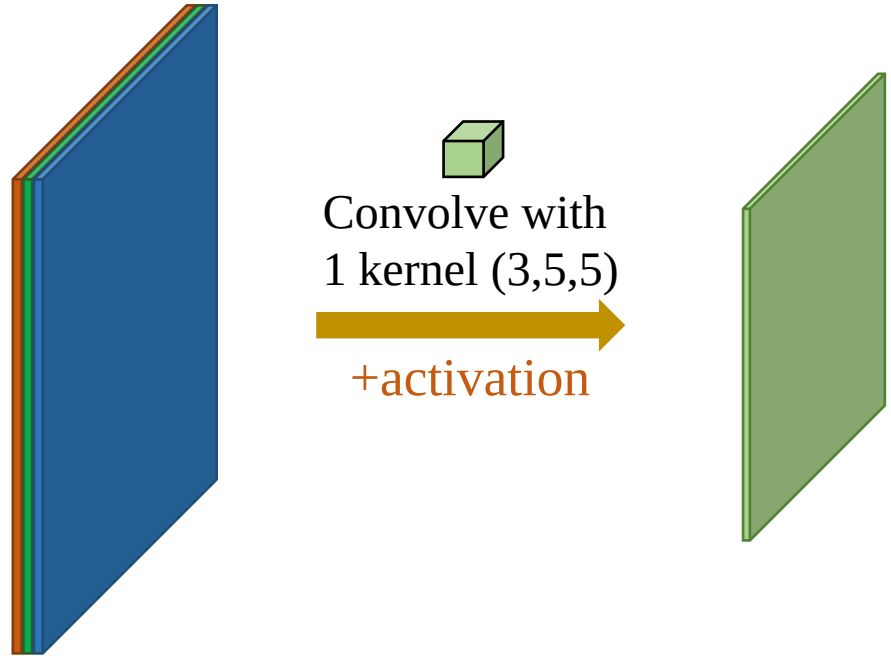
- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

Convolutional Neural Network

❖ Convolution layer in PyTorch

```
nn.Conv2d(in_channels, out_channels, kernel_size)
nn.ReLU()
```





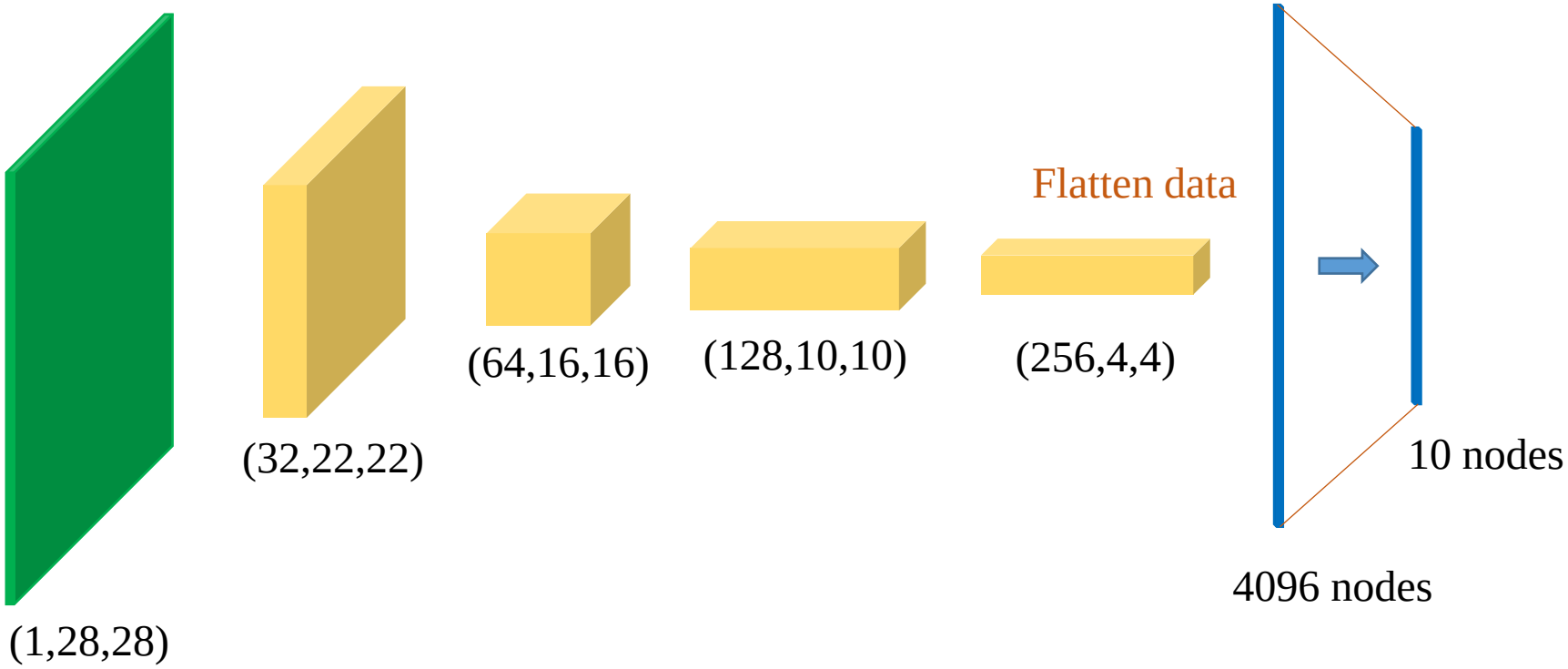
Convolutional Neural Network

Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

Convolutional Neural Network

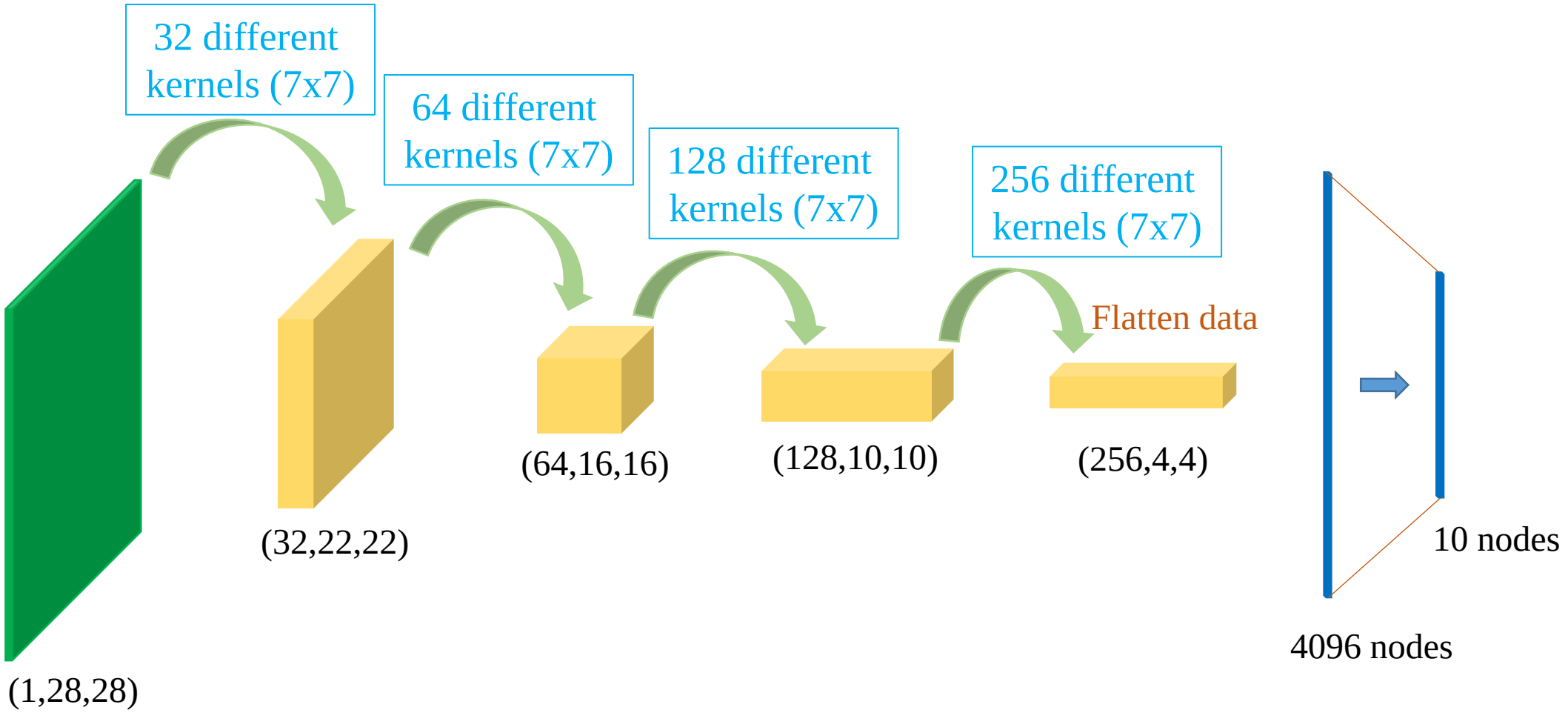
❖ Design a model for Fashion-MNIST dataset



Convolutional Neural Network

❖ Design a model for Fashion-MNIST dataset

demo



Simple Convolutional Neural Network

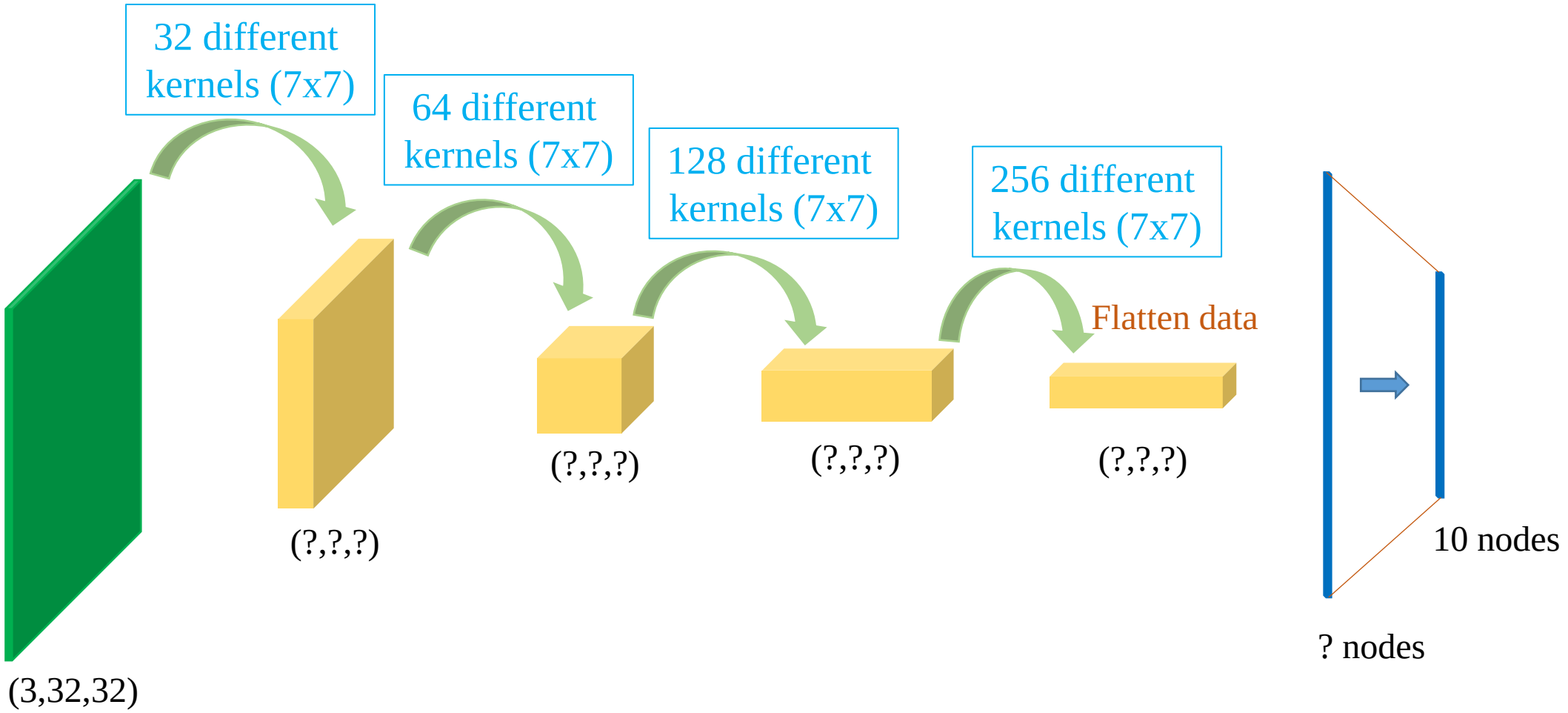
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(4*4*256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.dense(x)
        return x

model = CustomModel()
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 22, 22]           1,600
              ReLU-2          [-1, 32, 22, 22]               0
            Conv2d-3          [-1, 64, 16, 16]         100,416
              ReLU-4          [-1, 64, 16, 16]               0
            Conv2d-5         [-1, 128, 10, 10]         401,536
              ReLU-6         [-1, 128, 10, 10]               0
            Conv2d-7           [-1, 256, 4, 4]       1,605,888
              ReLU-8           [-1, 256, 4, 4]               0
           Flatten-9                [-1, 4096]               0
           Linear-10                 [-1, 128]         524,416
             ReLU-11                 [-1, 128]               0
           Linear-12                  [-1, 10]           1,290
================================================================
Total params: 2,635,146
Trainable params: 2,635,146
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.78
Params size (MB): 10.05
Estimated Total Size (MB): 10.83
----------------------------------------------------------------
```

Convolutional Neural Network

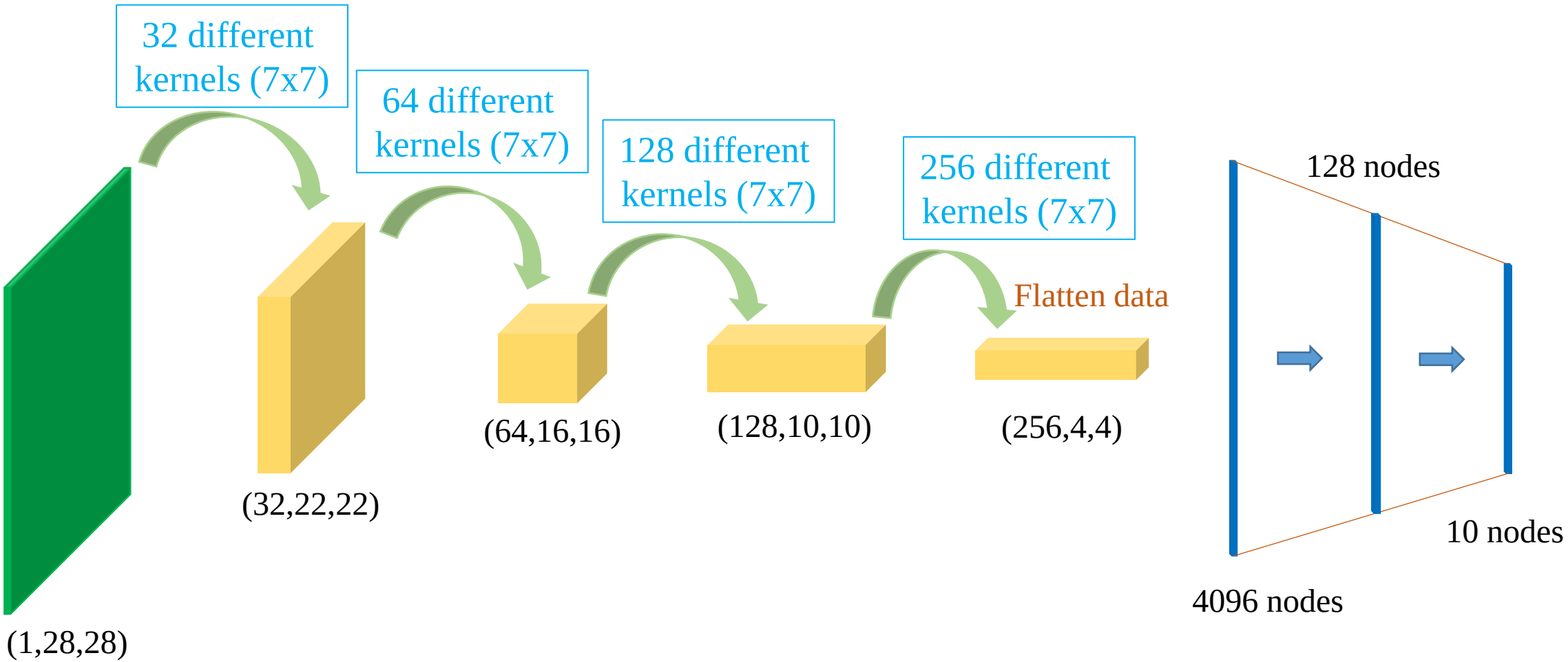
❖ Design a model for Cifar-10 dataset

demo



Convolutional Neural Network

❖ Fashion-MNIST dataset: case 1

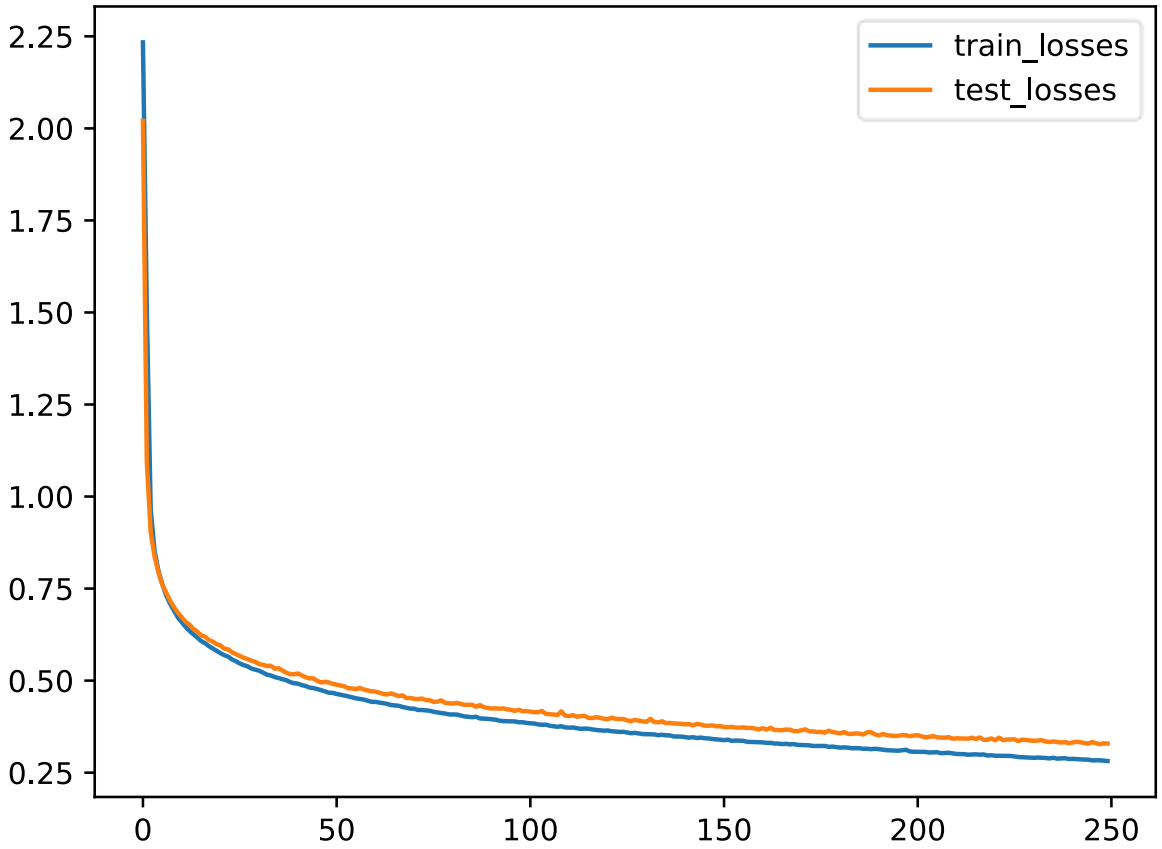


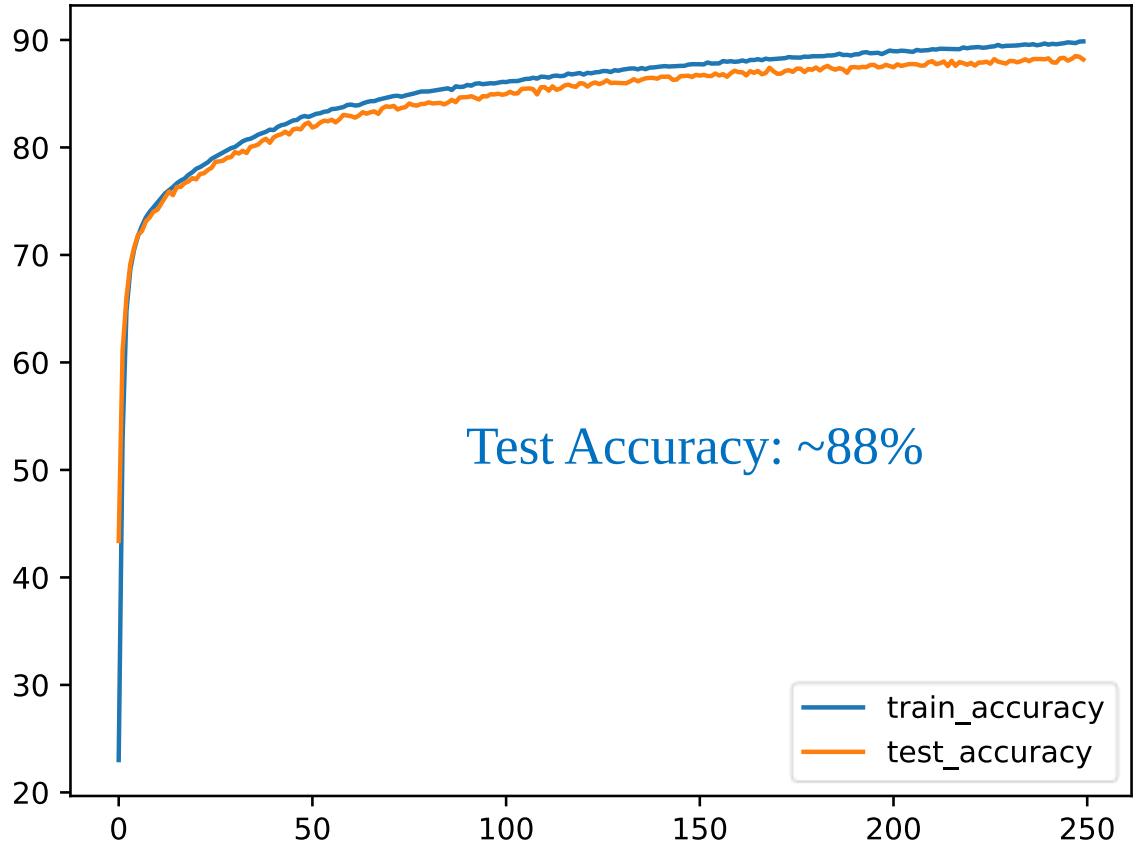
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(4*4*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)
```

```python
# Load FashionMNIST dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,),
                               (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

Convolutional Neural Network

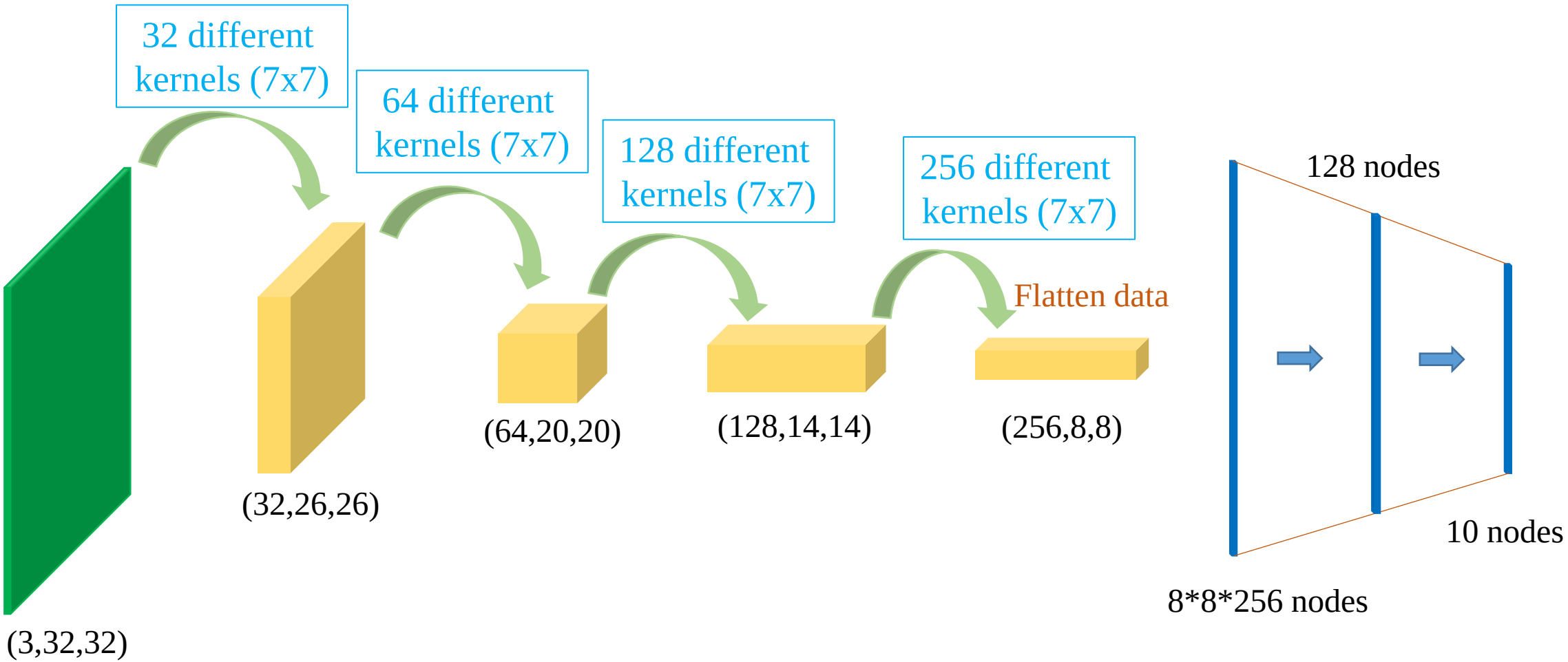
❖ Fashion-MNIST dataset: case 1





Convolutional Neural Network

❖ Cifar-10 dataset: case 2

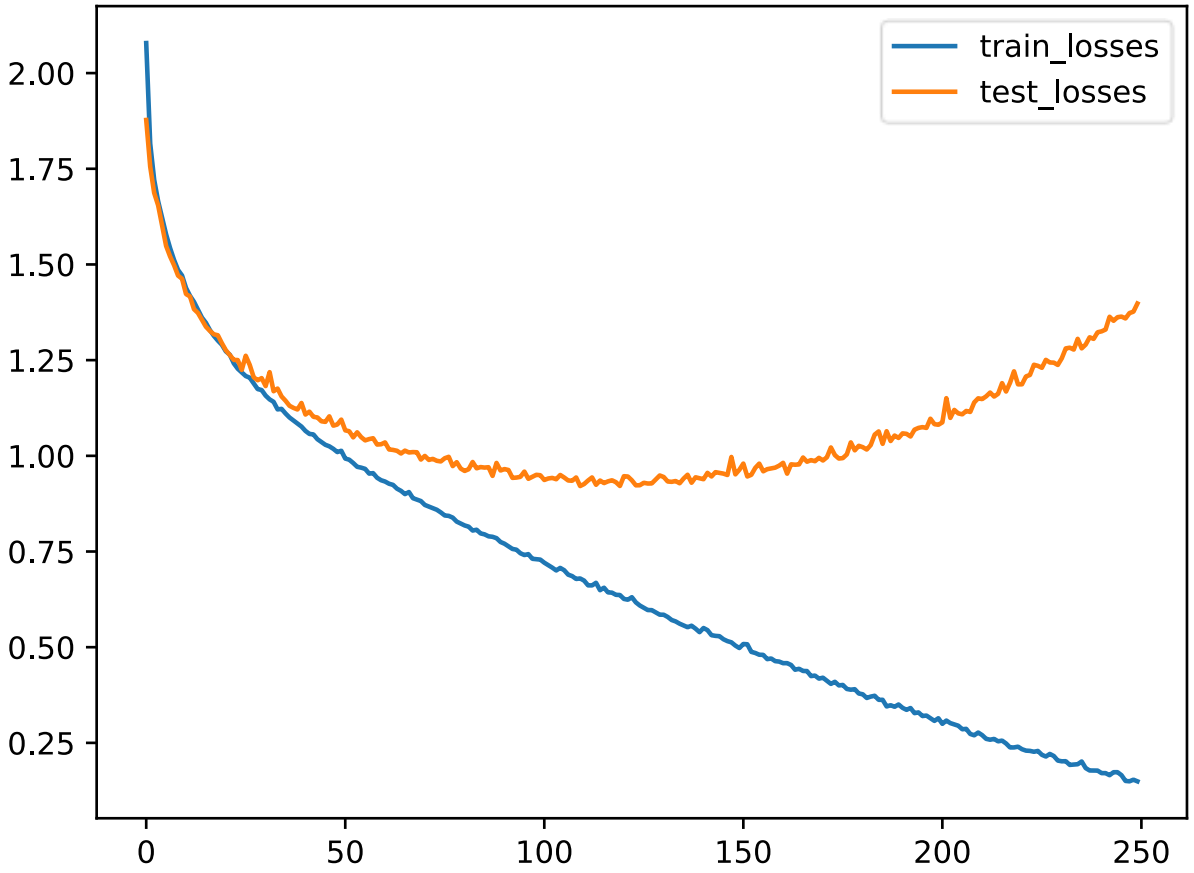


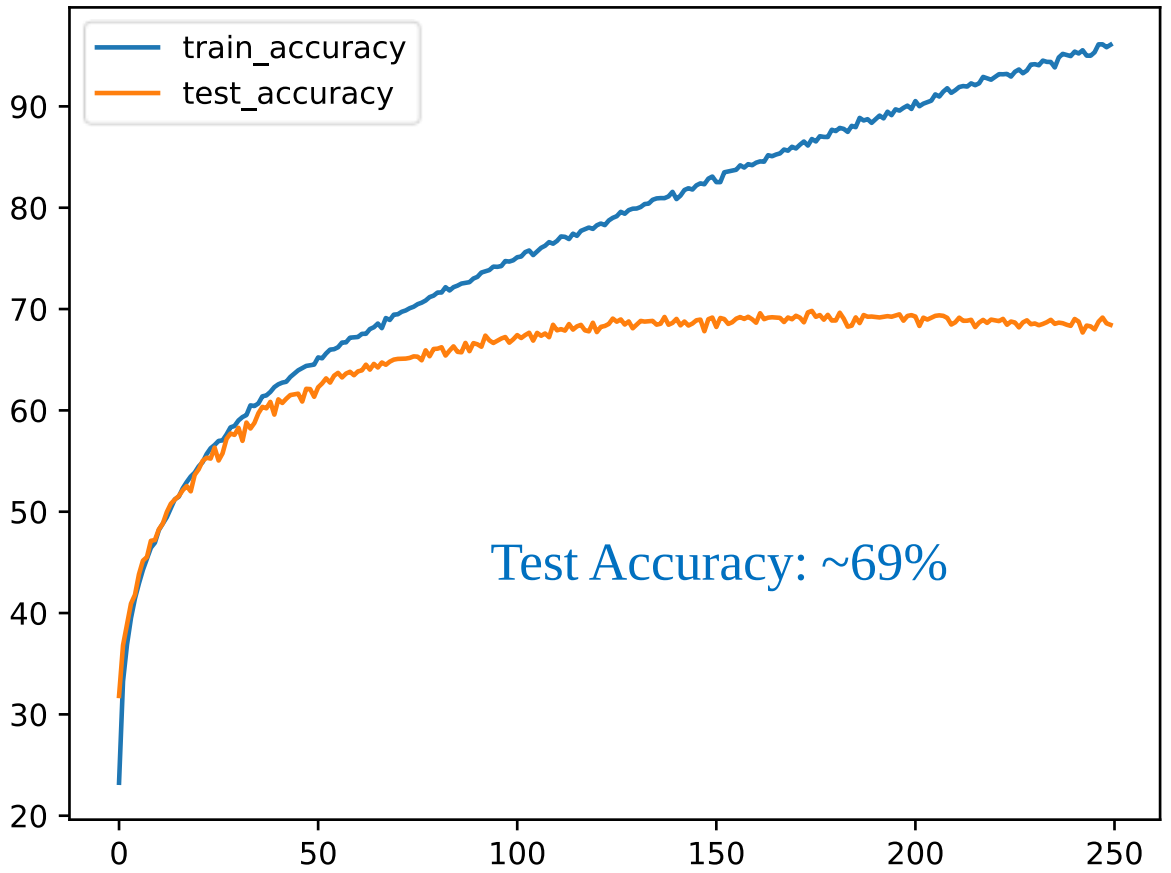
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(8*8*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)
```

```python
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=10,
                         shuffle=True,
                         drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

Convolutional Neural Network

❖ Cifar-10 dataset: case 2





Test Accuracy from MLP: ~53%

Outline

- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

Down-sample Feature Map

Max pooling: Features are preserved

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

2x2 max pooling (

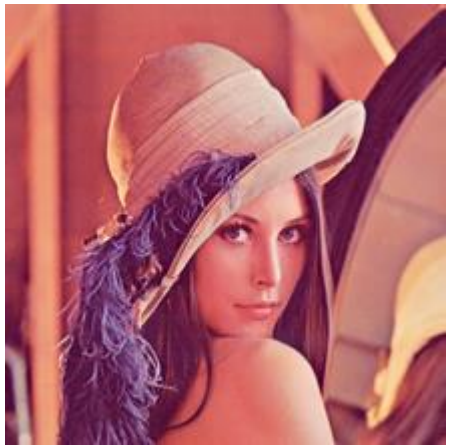
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \max(v_1, v_2, v_5, v_6)$$
$$m_2 = \max(v_3, v_4, v_7, v_8)$$
$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$
$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

nn.MaxPool2d(2, 2)



Feature map (220x220)

max pooling (2x2) →

Feature map (110x110)

max pooling (2x2) →

Feature map (55x55)

Down-sample Feature Map

Average pooling: Features are preserved

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

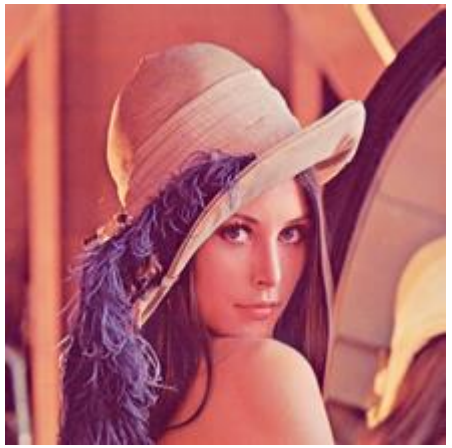
Data

average pooling (| v_1 | v_2 | v_3 | v_4 |
|---|---|---|---|
| v_5 | v_6 | v_7 | v_8 |
| v_9 | v_{10} | v_{11} | v_{12} |
| v_{13} | v_{14} | v_{15} | v_{16} |) =

m_1	m_2
m_3	m_4

$$m_1 = \text{mean}(v_1, v_2, v_5, v_6)$$
$$m_2 = \text{mean}(v_3, v_4, v_7, v_8)$$
$$m_3 = \text{mean}(v_9, v_{10}, v_{13}, v_{14})$$
$$m_4 = \text{mean}(v_{11}, v_{12}, v_{15}, v_{16})$$

nn.AvgPool2d(2, 2)



Feature map (220x220)

Average Pooling (2x2) →

Feature map (110x110)

Average Pooling (2x2) →

Feature map (55x55)

Down-sample Feature Map

Max pooling vs. Average pooling

nn.MaxPool2d(2, 2)

Feature map (220x220)

max pooling (2x2)

max pooling (2x2)

Feature map (55x55)

nn.AvgPool2d(2, 2)

Feature map (220x220)

Average Pooling (2x2)

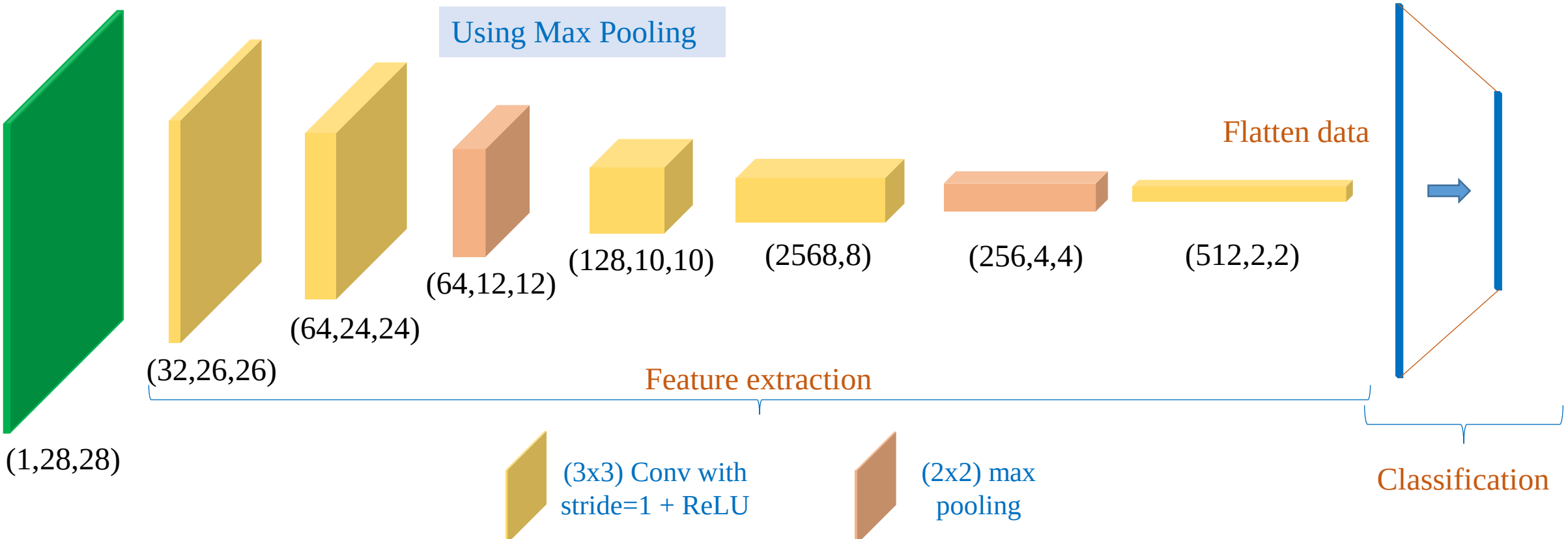
Feature map (110x110)

Average Pooling (2x2)

Feature map (55x55)

```python
    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.pool(x)

        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.pool(x)

        x = self.relu(self.conv5(x))
        x = self.flatten(x)
        x = self.dense(x)
        return x
```

```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=3)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)  # 2x2 Max pooling
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(2*2*512, 10)
```



Down-sample Feature Map

❖ **Convolve with stride**

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data **D**

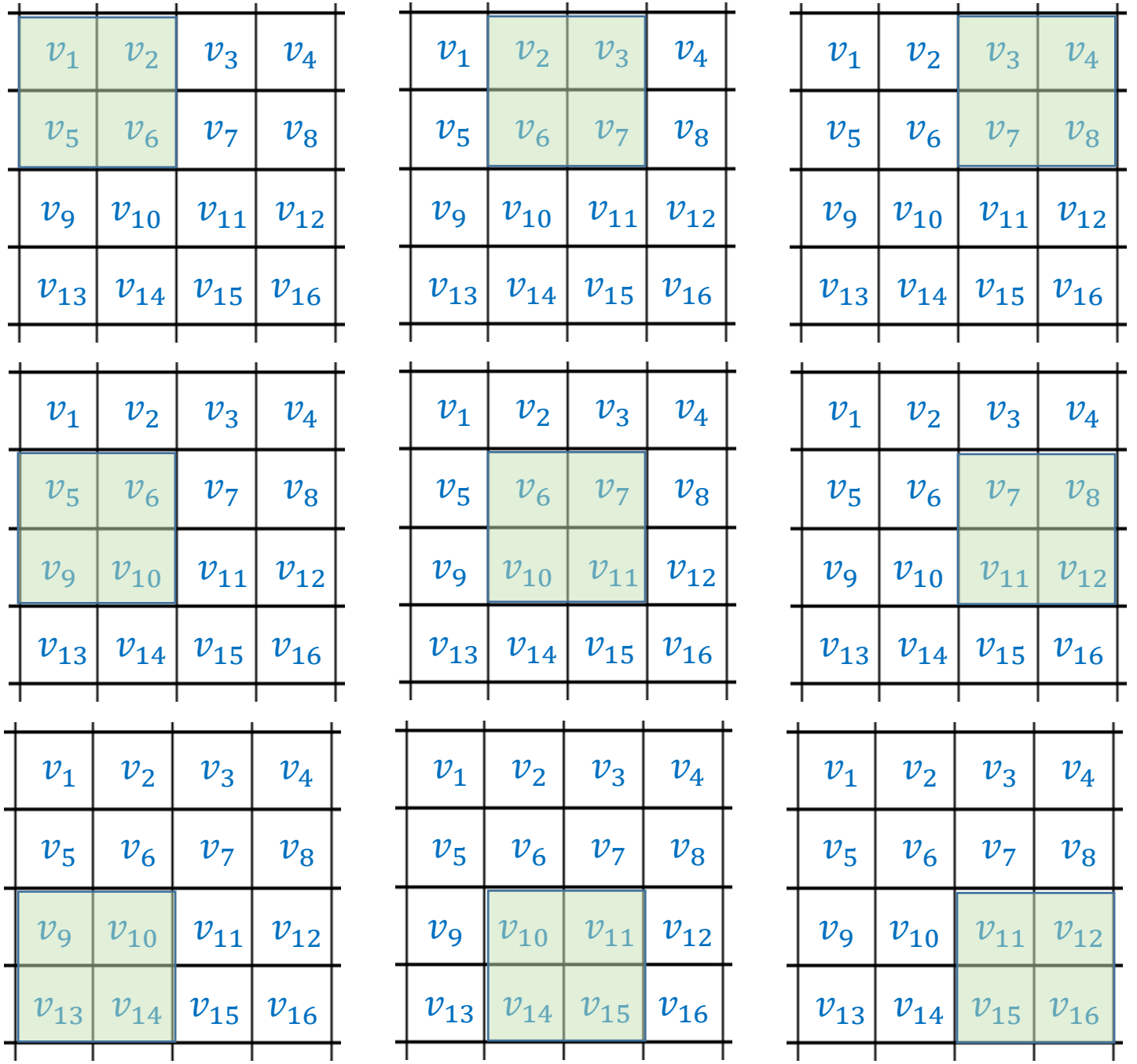
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=1

m_1	m_2	m_3
m_4	m_5	m_6
m_7	m_8	m_9

Output



Down-sample Feature Map

❖ **Convolve with stride**

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data **D**

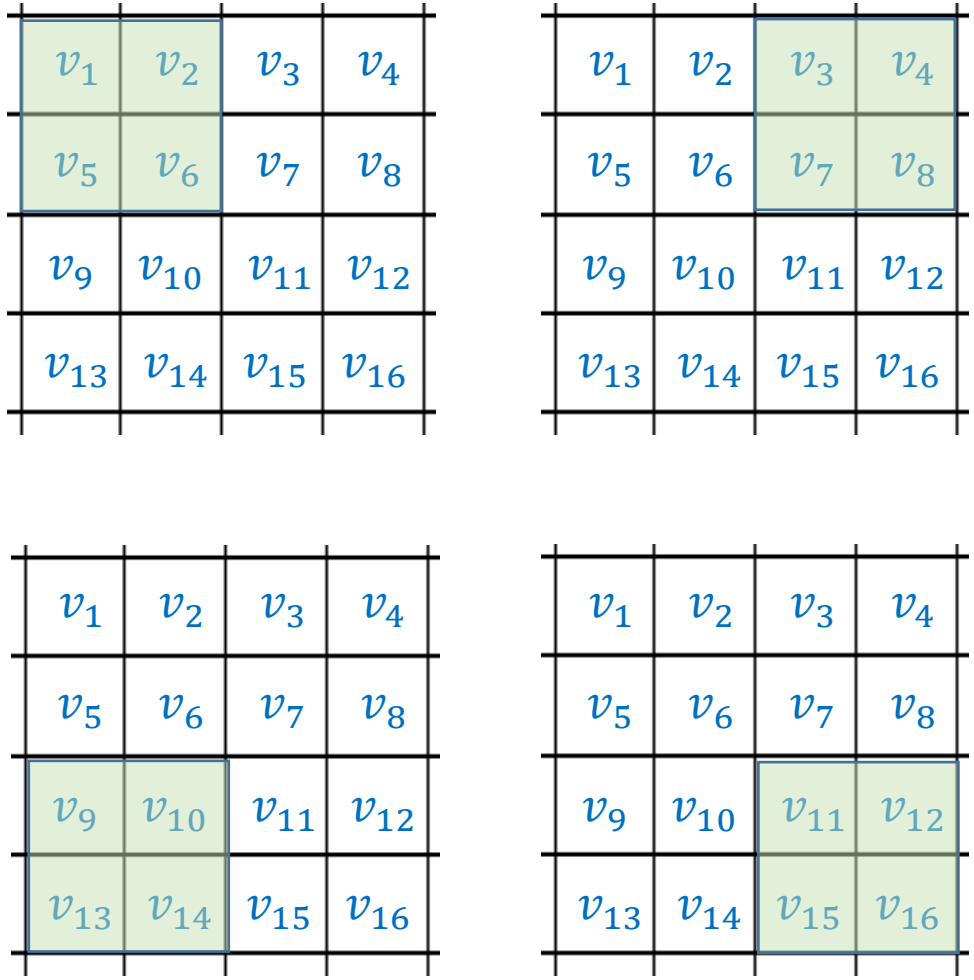
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=2

m_1	m_2
m_3	m_4

Output

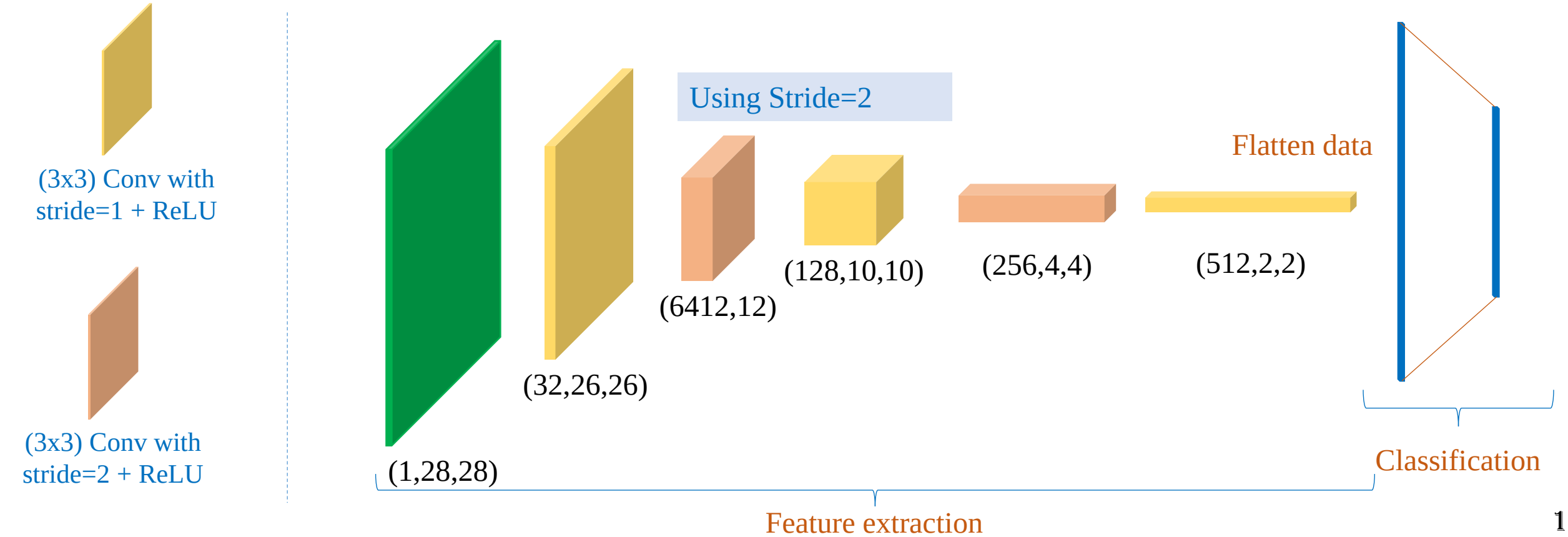


nn.Conv2d(in_channels, out_channels, kernel_size, stride=1)

nn.Conv2d(in_channels, out_channels, kernel_size, stride=2)

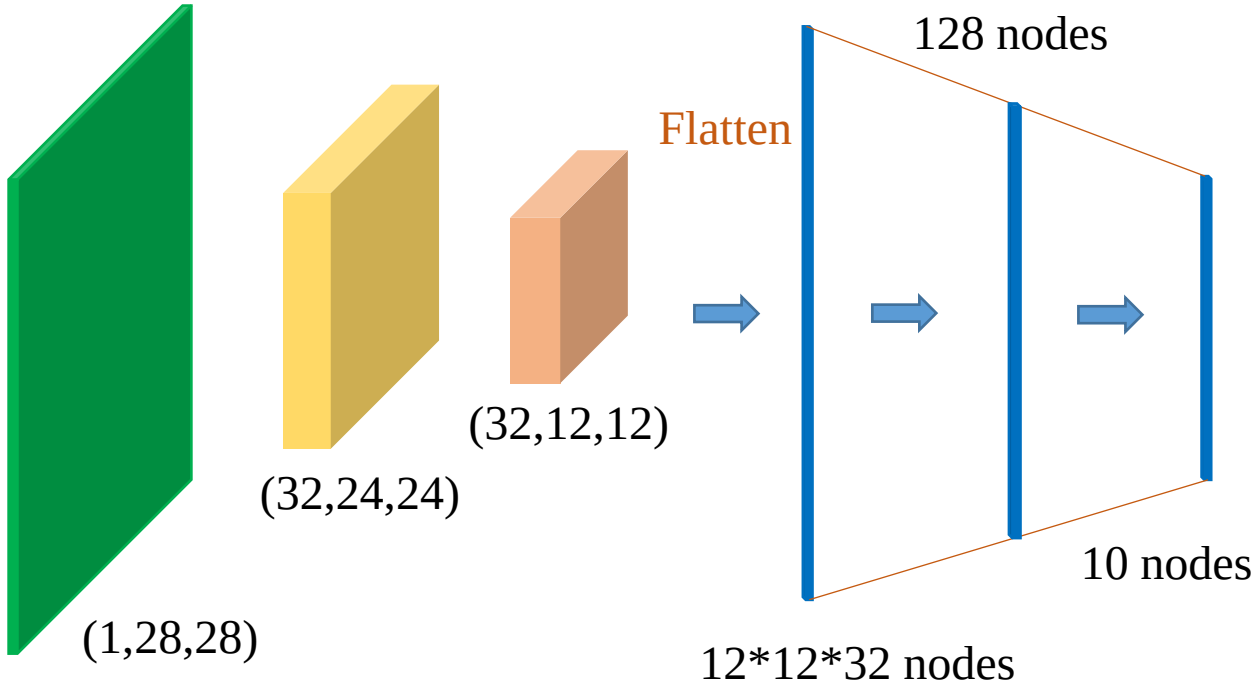
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=2)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=3)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(2*2*512, 10)
```

```python
def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.relu(self.conv2(x))

    x = self.relu(self.conv3(x))
    x = self.relu(self.conv4(x))

    x = self.relu(self.conv5(x))
    x = self.flatten(x)
    x = self.dense(x)
    return x
```



Comparison of down-samplings

Model Design



(5x5) Conv with stride=1 + ReLU

(2x2) down-sampling

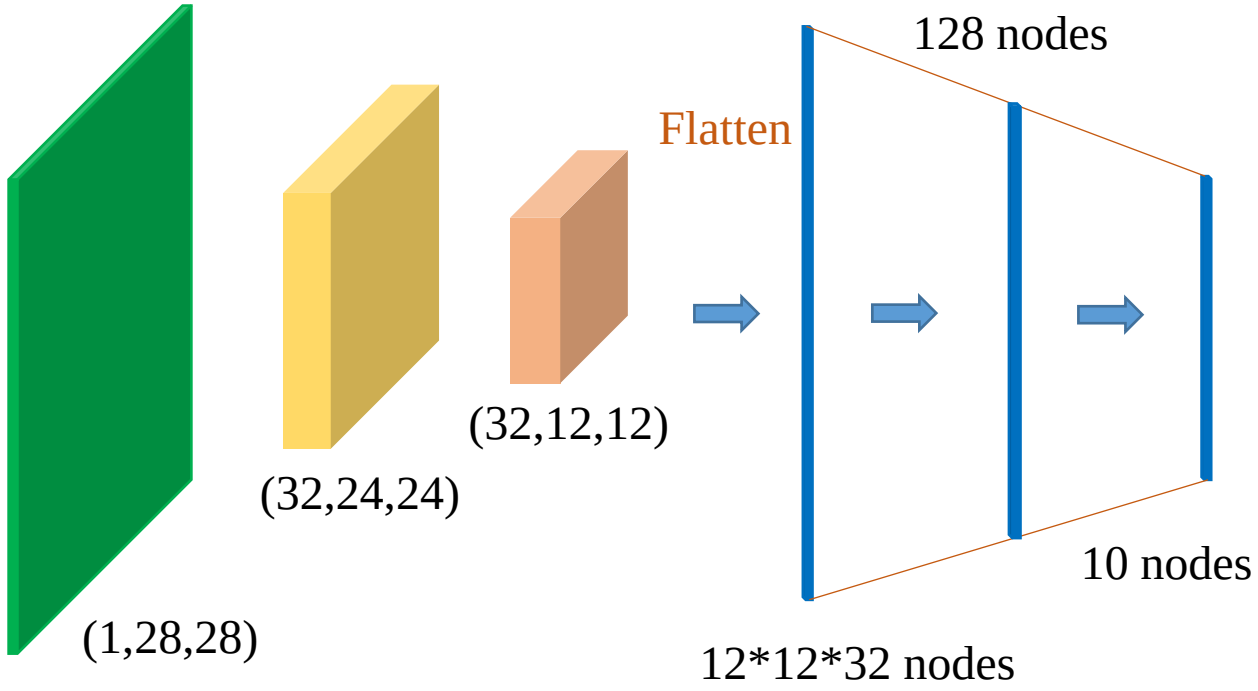
Implementation 1

```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.pool = nn.MaxPool2d(2, 2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

# create model
model = CustomModel()
model = model.to(device)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

max pooling

Comparison of down-samplings

Model Design



(5x5) Conv with stride=1 + ReLU

(2x2) down-sampling

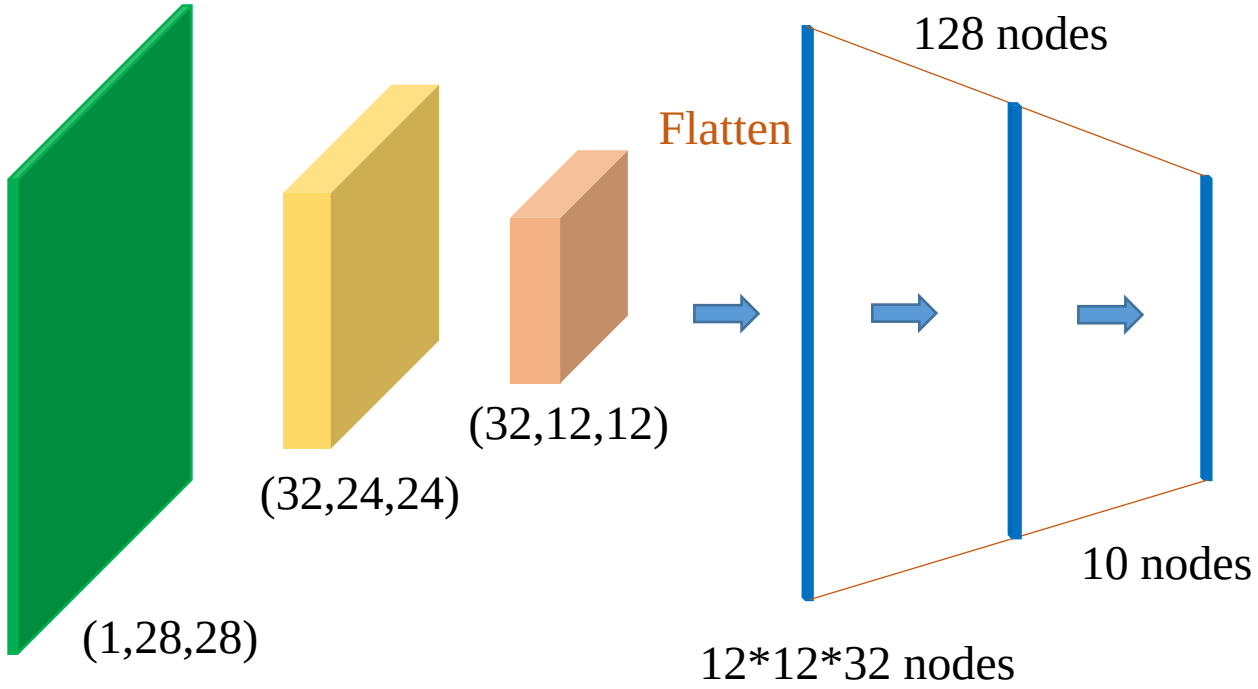
Implementation 2

```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.pool = nn.AvgPool2d(2, 2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

# create model
model = CustomModel()
model = model.to(device)

# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

average pooling

Comparison of down-samplings

Model Design



(5x5) Conv with stride=1 + ReLU

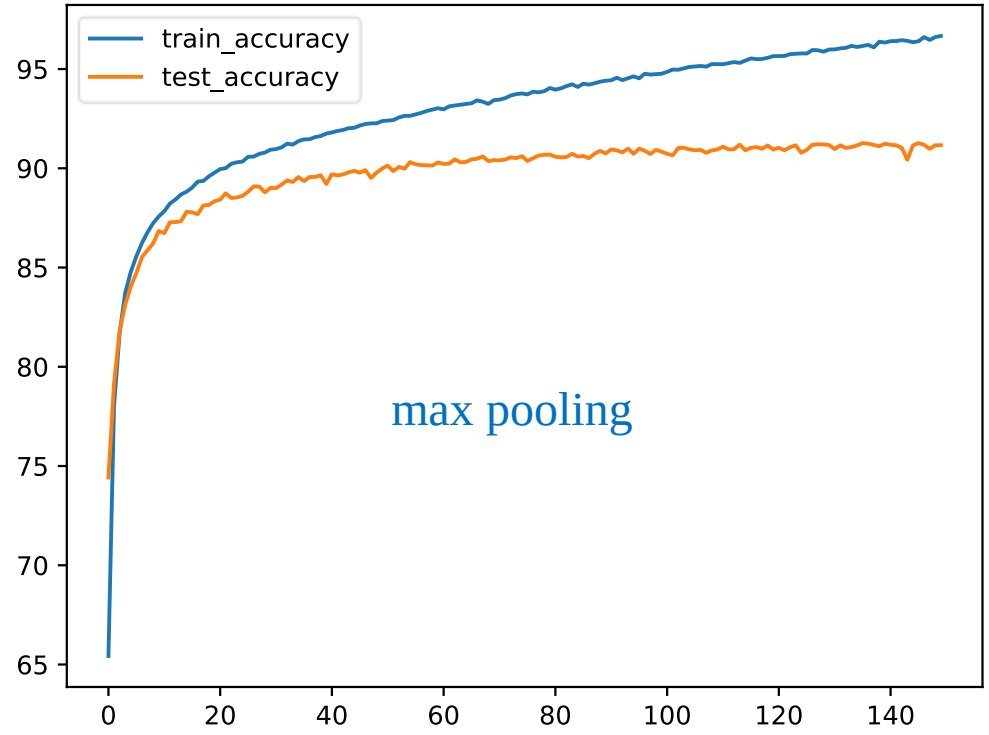
(2x2) down-sampling

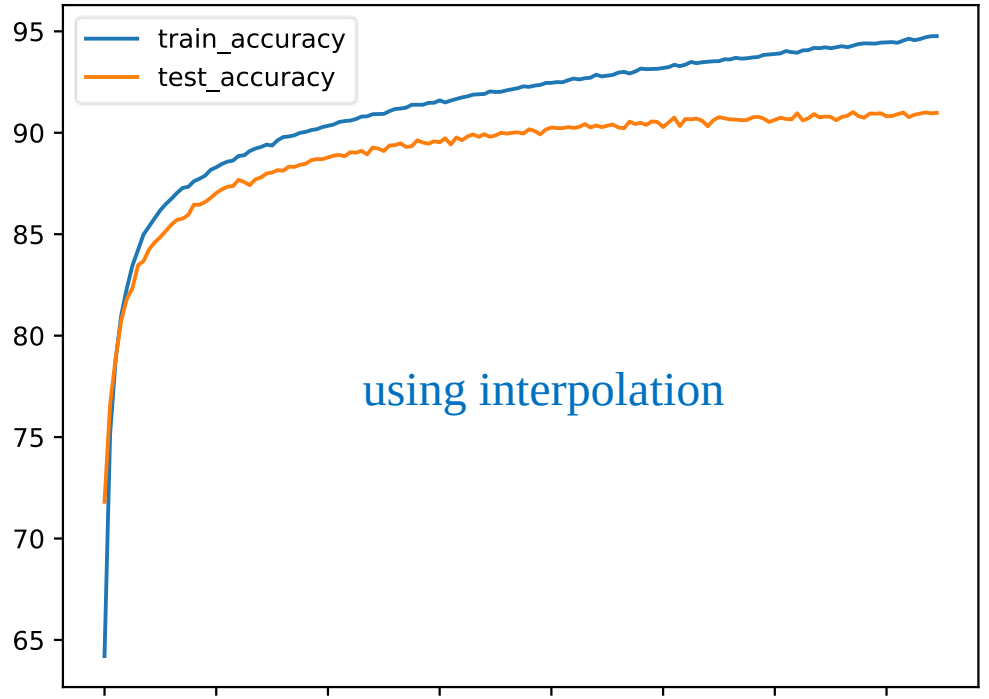
```python
class ResizeLayer(nn.Module):
    def __init__(self, scale_factor, mode='bilinear',
                 align_corners=False):
        super(ResizeLayer, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode
        self.align_corners = align_corners

    def forward(self, x):
        return F.interpolate(x, scale_factor=self.scale_factor,
                             mode=self.mode,
                             align_corners=self.align_corners)
```

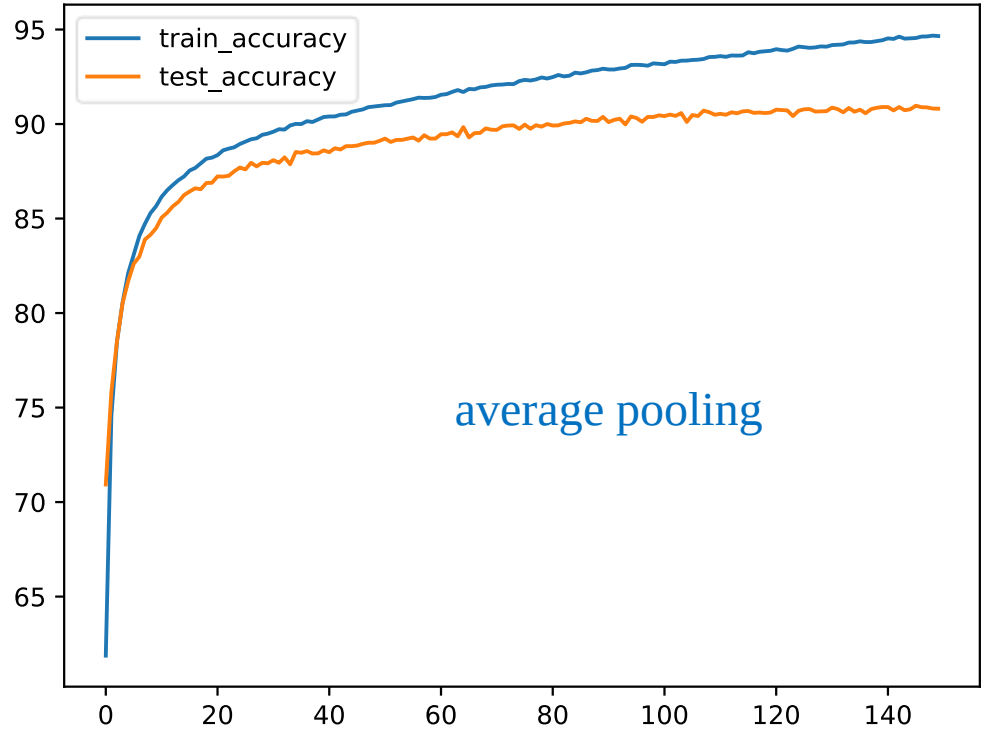
interpolation

```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.resize = ResizeLayer(0.5)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv(x))
        x = self.resize(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x
```

Comparison of down-samplings







Padding

$$S_o = \left\lfloor \frac{S_D - K + 2P}{S} \right\rfloor + 1$$

Keep resolution of feature map

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data $\boldsymbol{D}$
(4x4)

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

b

Kernel of parameters

Without using padding or padding=0

Convolve with stride=1 $(\boldsymbol{D}) =$

m_1	m_2
m_4	m_5

Output
(2x2)

Padding = 1

v	v	v	v	v	v
v	v_1	v_2	v_3	v_4	v
v	v_5	v_6	v_7	v_8	v
v	v_9	v_{10}	v_{11}	v_{12}	v
v	v_{13}	v_{14}	v_{15}	v_{16}	v
v	v	v	v	v	v

Data $\boldsymbol{D}_p$

Convolve with stride=1 $(\boldsymbol{D}_p) =$

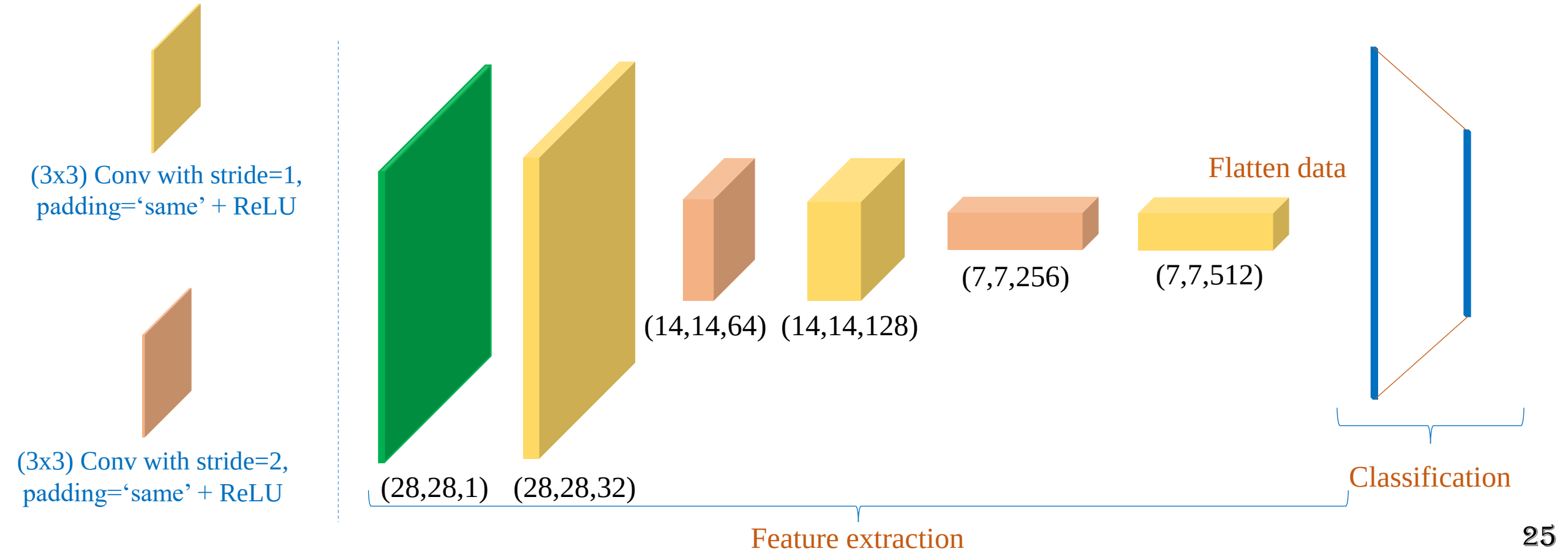
m_1	m_2	m_3	m_4
m_5	m_6	m_7	m_8
m_9	m_{10}	m_{11}	m_{12}
m_{13}	m_{14}	m_{15}	m_{16}

Output
(4x4)

Padding

Example

```
model = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding='same', stride=1), nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2), nn.ReLU(),
    nn.Conv2d(64, 128, kernel_size=3, padding=1, stride=1), nn.ReLU(),
    nn.Conv2d(128, 256, kernel_size=3, padding=1, stride=2), nn.ReLU(),
    nn.Conv2d(256, 512, kernel_size=3, padding=1, stride=1), nn.ReLU(),
    nn.Flatten(), nn.Linear(7*7*512, 10)
)
```

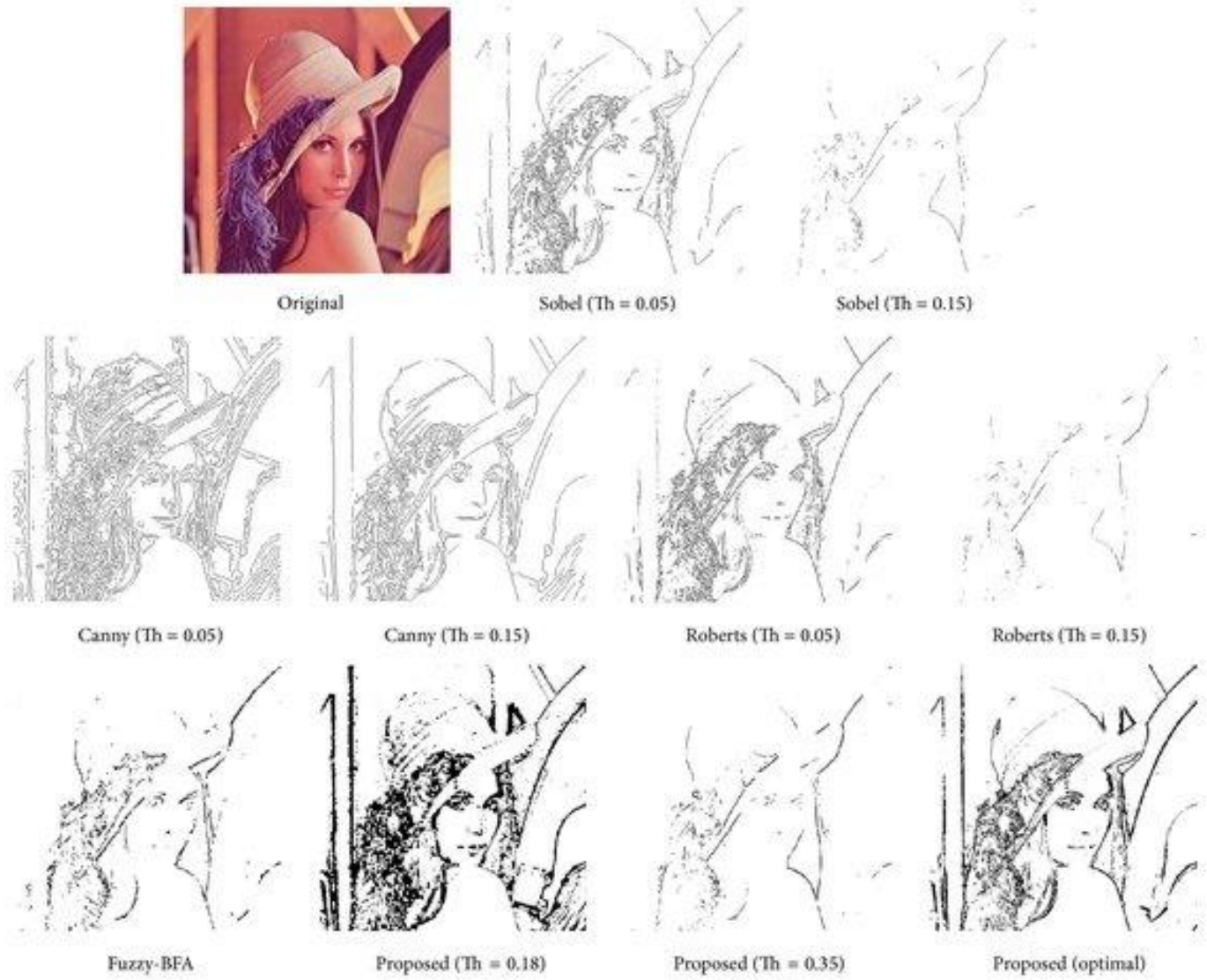


Outline

- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

Engineering Feature Extractors

- **Edge detection**
- **Gradient**



Edge Detection via Edge-Strength Estimation Using Fuzzy Reasoning and Optimal Threshold Selection Using Particle Swarm Optimization

Derivative and Applications

Tính đạo hàm trung bình theo hướng x

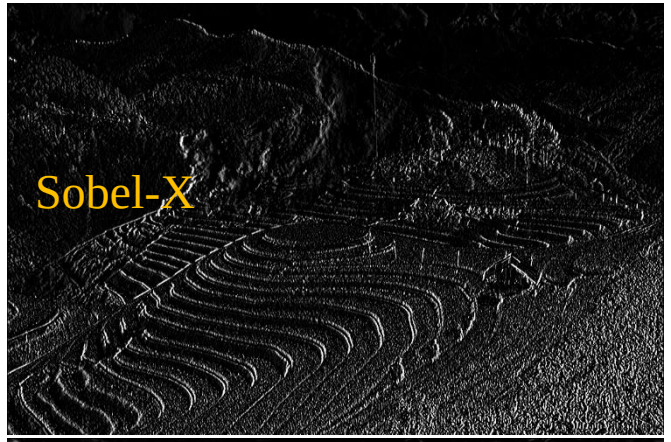
1				-1	0	1		-1	0	1
2	*	-1	0	1	=	-2	0	2
1						-1	0	1

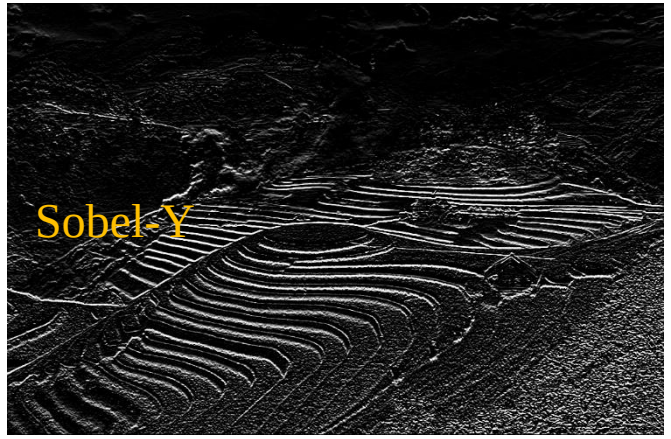
weighted average | x-derivative | Sobel for x direction

Tính đạo hàm trung bình theo hướng y

1				1	2	1		1	2	1
0	*	1	2	1	=	0	0	0
-1						-1	-2	-1

y-derivative | weighted average | Sobel for y direction



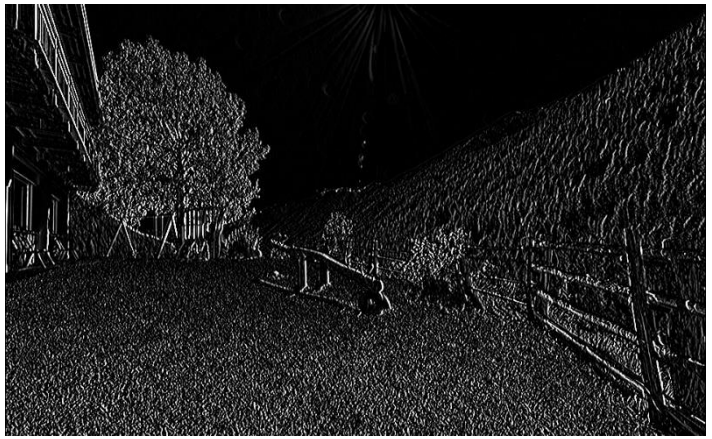


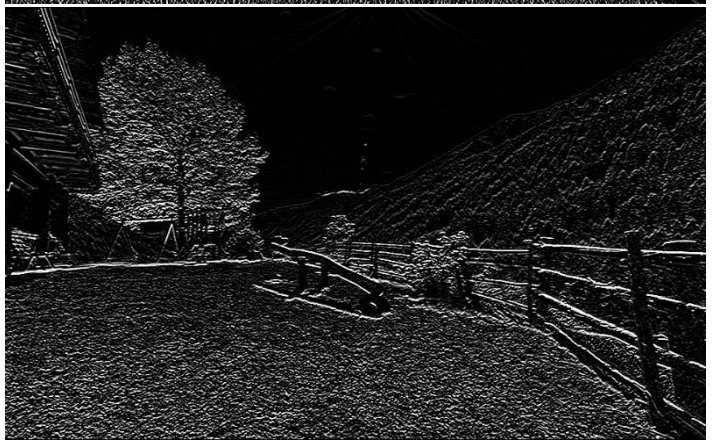
Derivative and Applications

Edge detection

Edge detection

Sobel-X





Sobel-Y

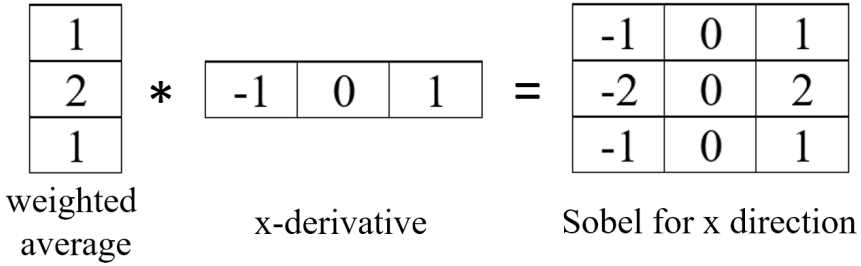
$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

Engineering Feature Extractors

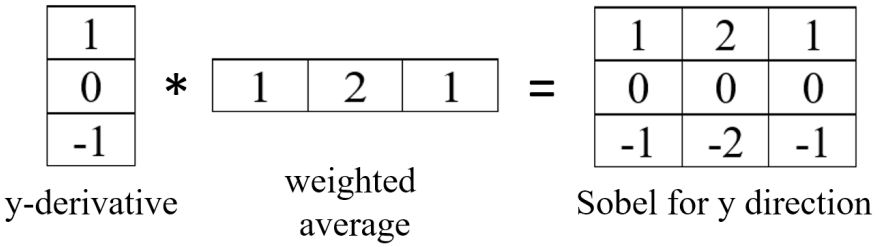
❖ Edge detection

Tính đạo hàm trung bình theo hướng x

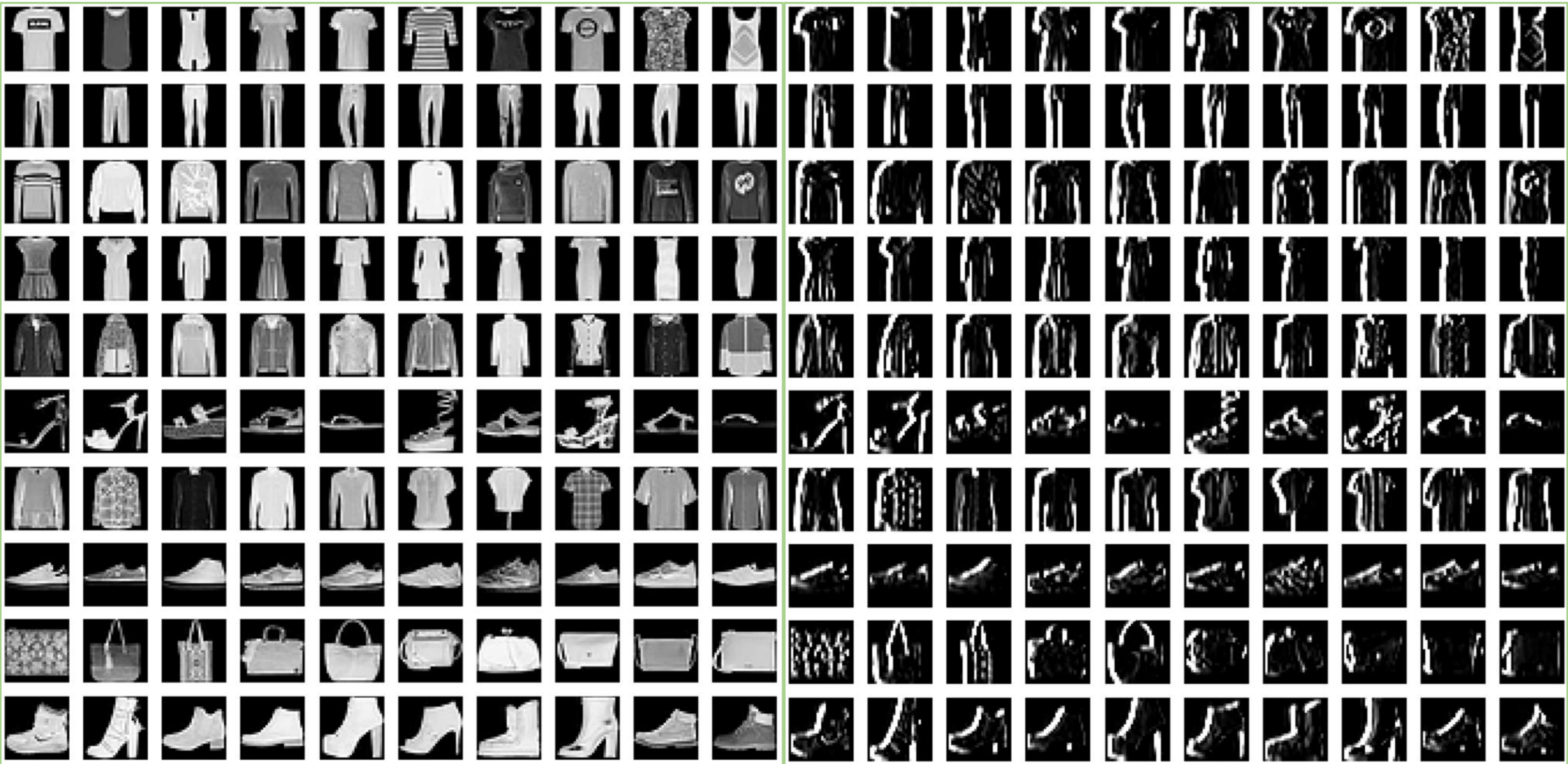



```python
@staticmethod
def _sobel(image, ksize=3):
    sobel_x = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=ksize)
    sobel_y = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=ksize)
    sobel_x = torch.from_numpy(sobel_x)
    sobel_y = torch.from_numpy(sobel_y)
    sobel_magnitude = torch.hypot(sobel_x, sobel_y)
    return sobel_magnitude
```

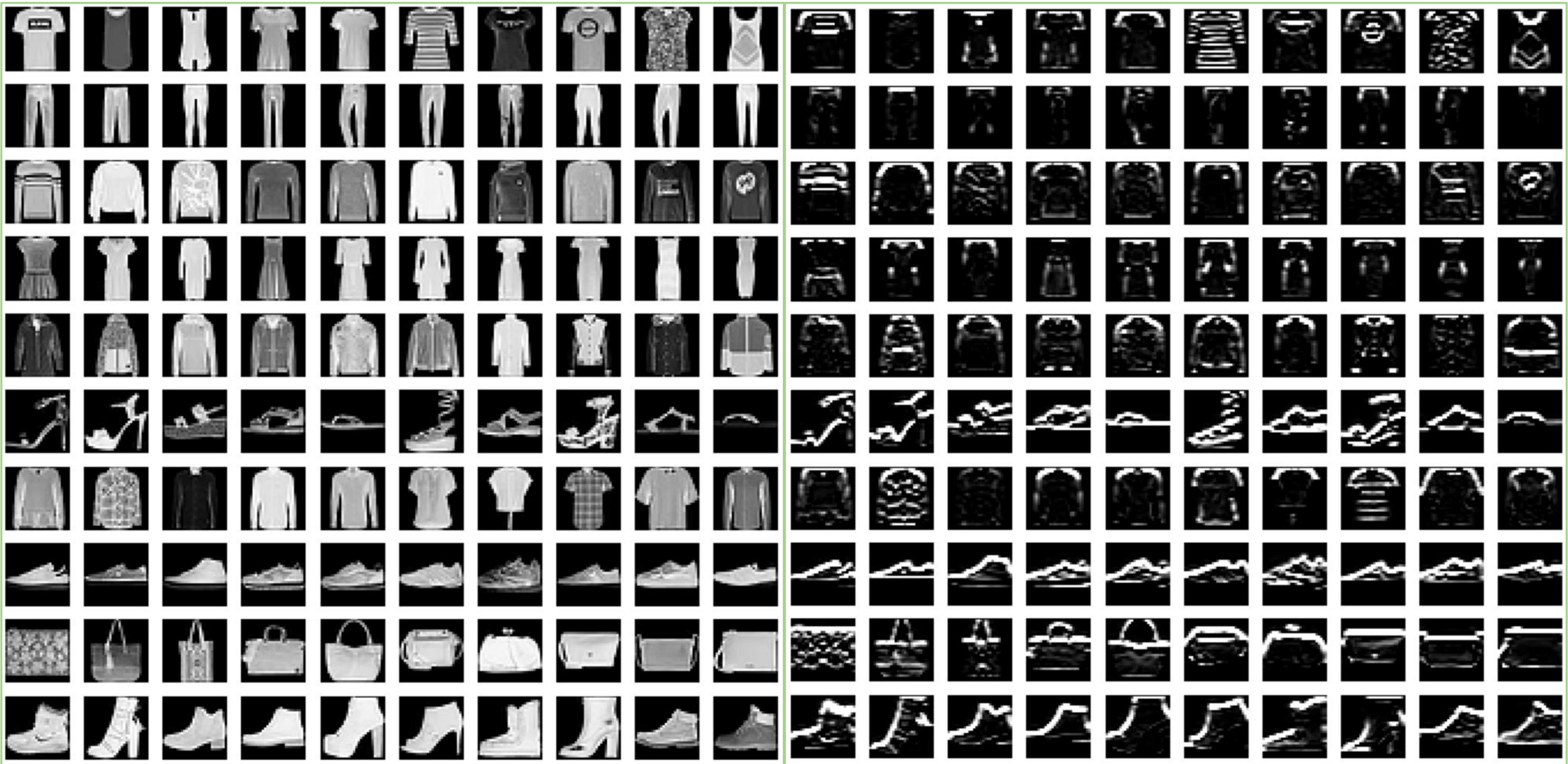
Tính đạo hàm trung bình theo hướng y



Sobel_X

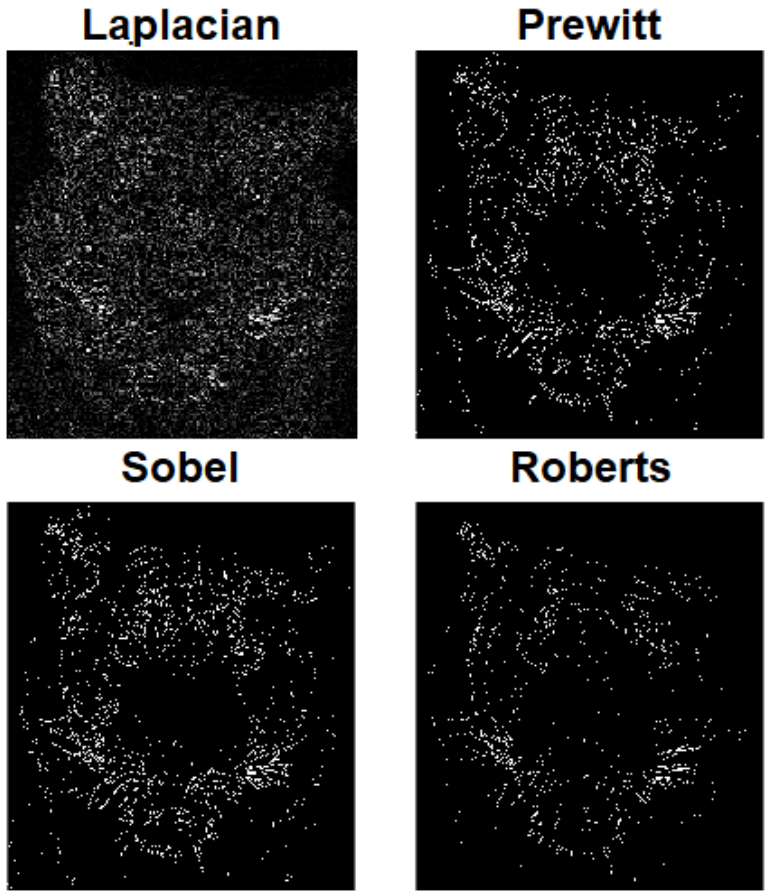


Sobel_Y



Engineering Feature Extractors

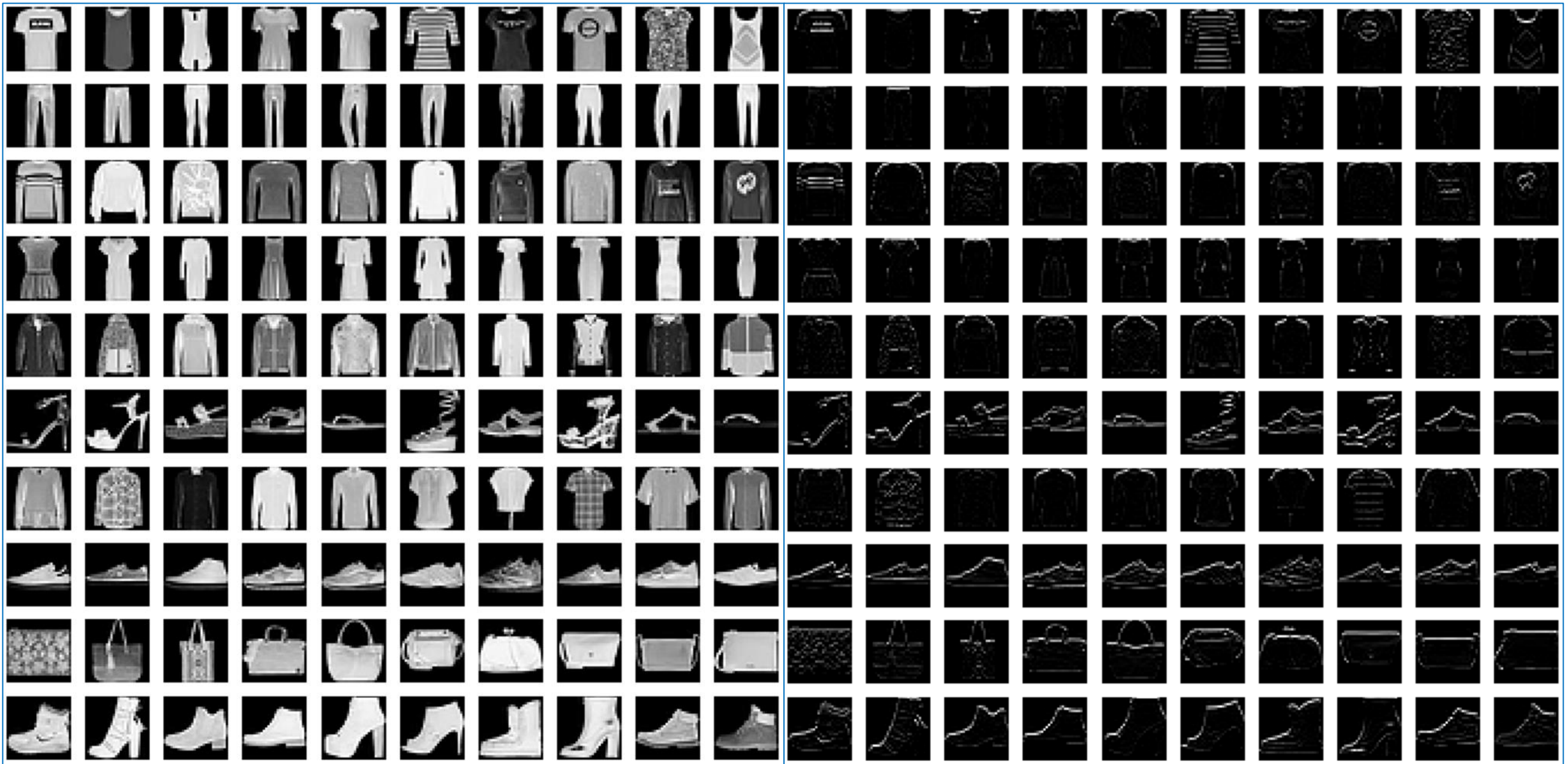
❖ Traditional methods



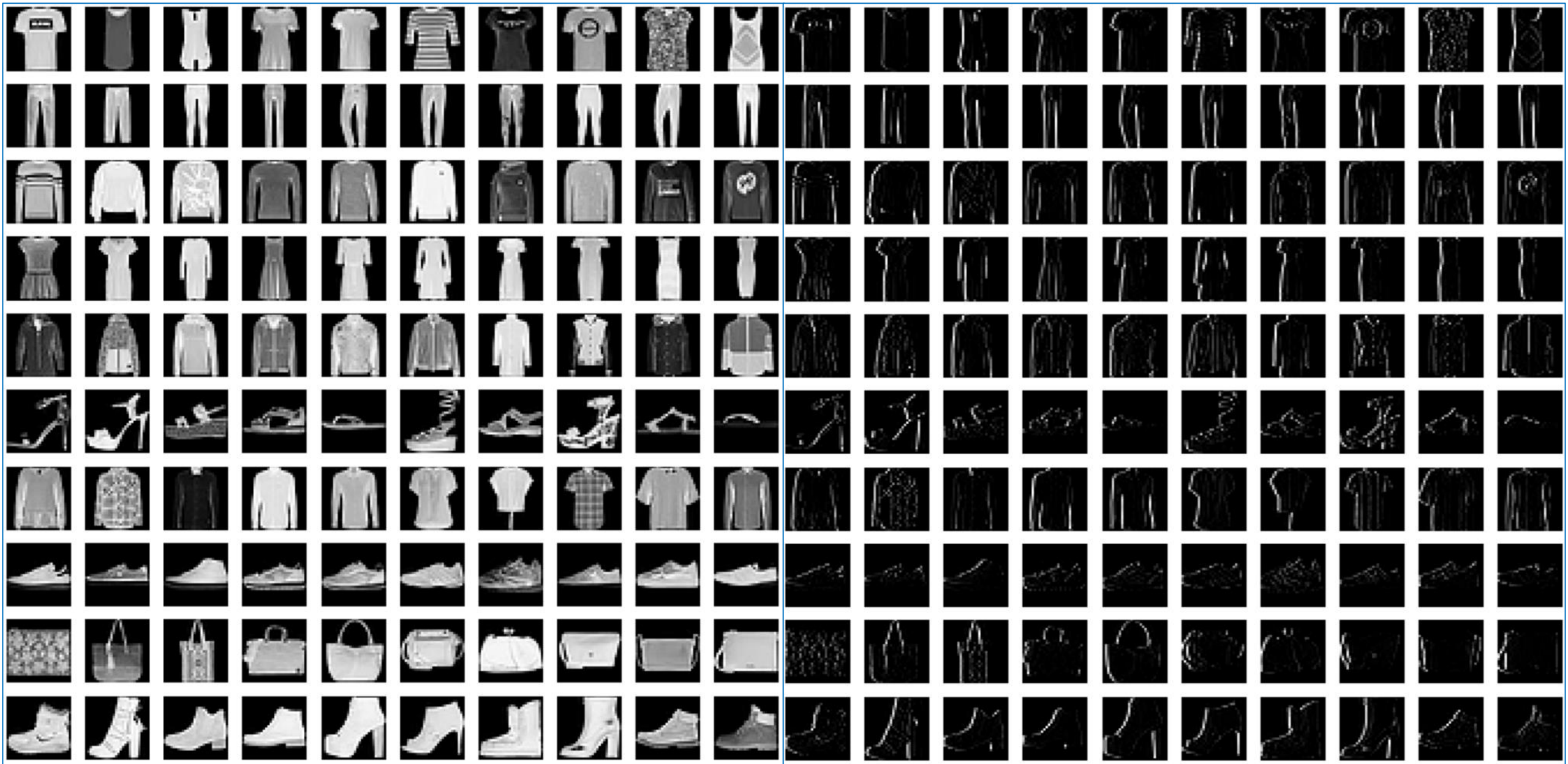

```python
@staticmethod
def _scharr(image):
    scharr_x = cv2.Scharr(image, cv2.CV_32F, 1, 0)
    scharr_y = cv2.Scharr(image, cv2.CV_32F, 0, 1)
    scharr_x = torch.from_numpy(scharr_x)
    scharr_y = torch.from_numpy(scharr_y)
    scharr_magnitude = torch.hypot(scharr_x, scharr_y)
    return scharr_magnitude

@staticmethod
def _laplacian(image):
    laplacian_img = cv2.Laplacian(image, cv2.CV_32F)
    laplacian_img = torch.from_numpy(laplacian_img)
    return laplacian_img
```

https://www.researchgate.net/figure/a-Laplacian-filter-b-Prewitt-filter-c-Sobel-filter-d-Roberts-filter_fig4_357351763

Gradient_X

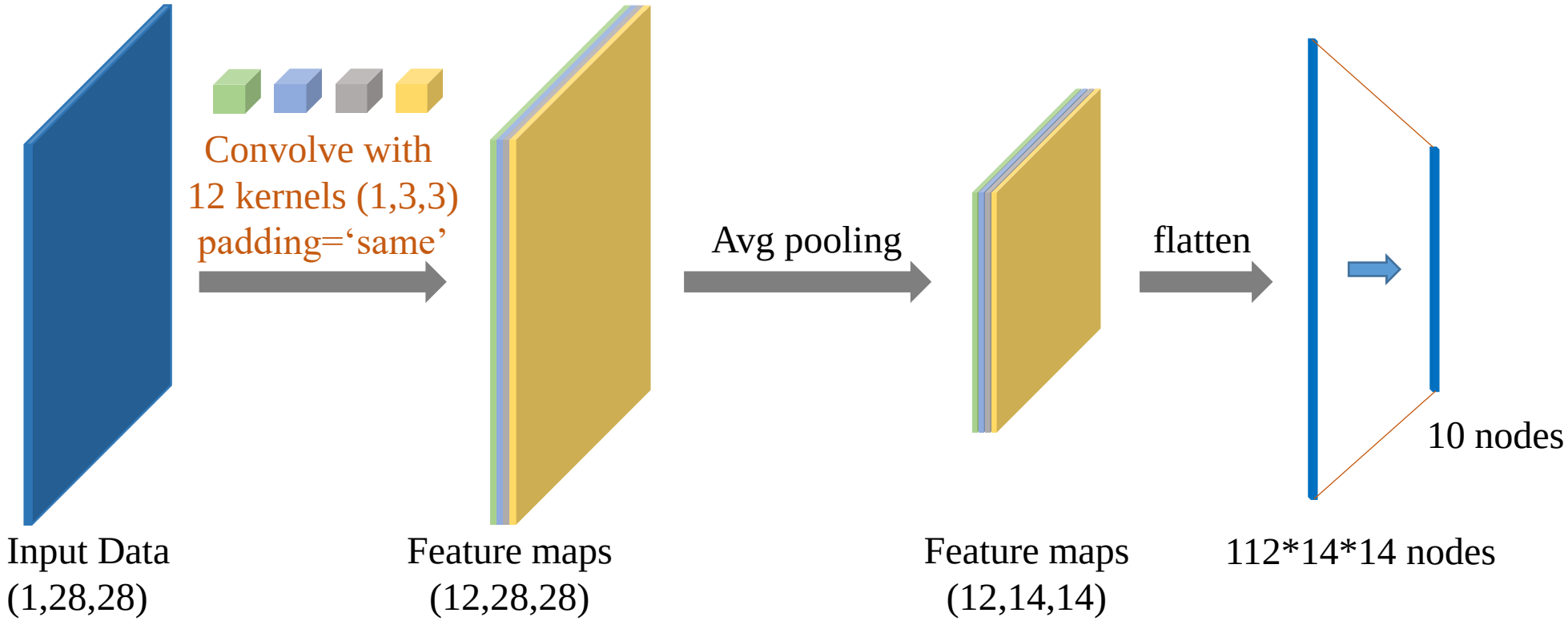


Gradient_Y



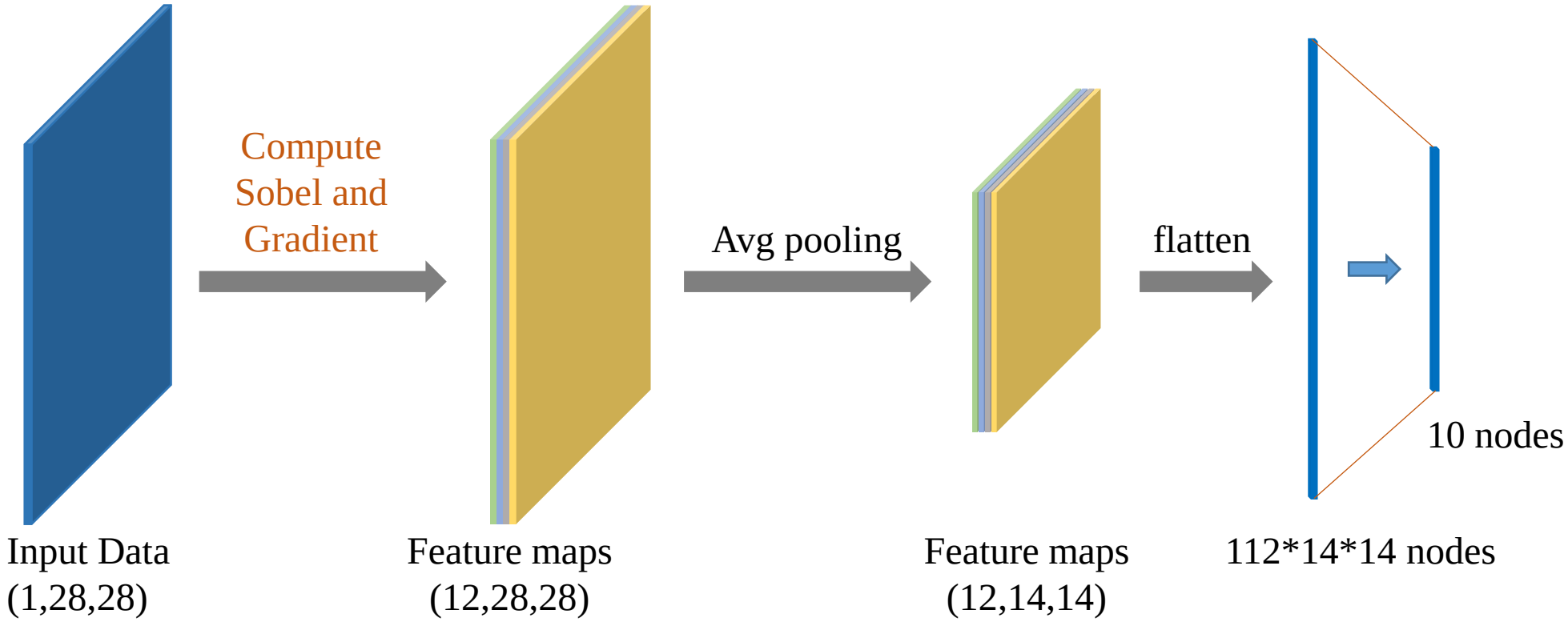
Comparison

❖ Engineering features vs. CNN features

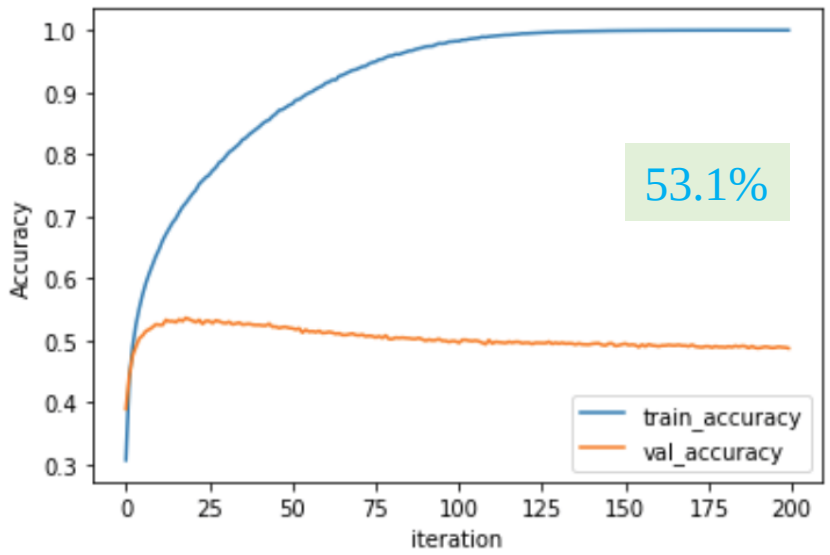


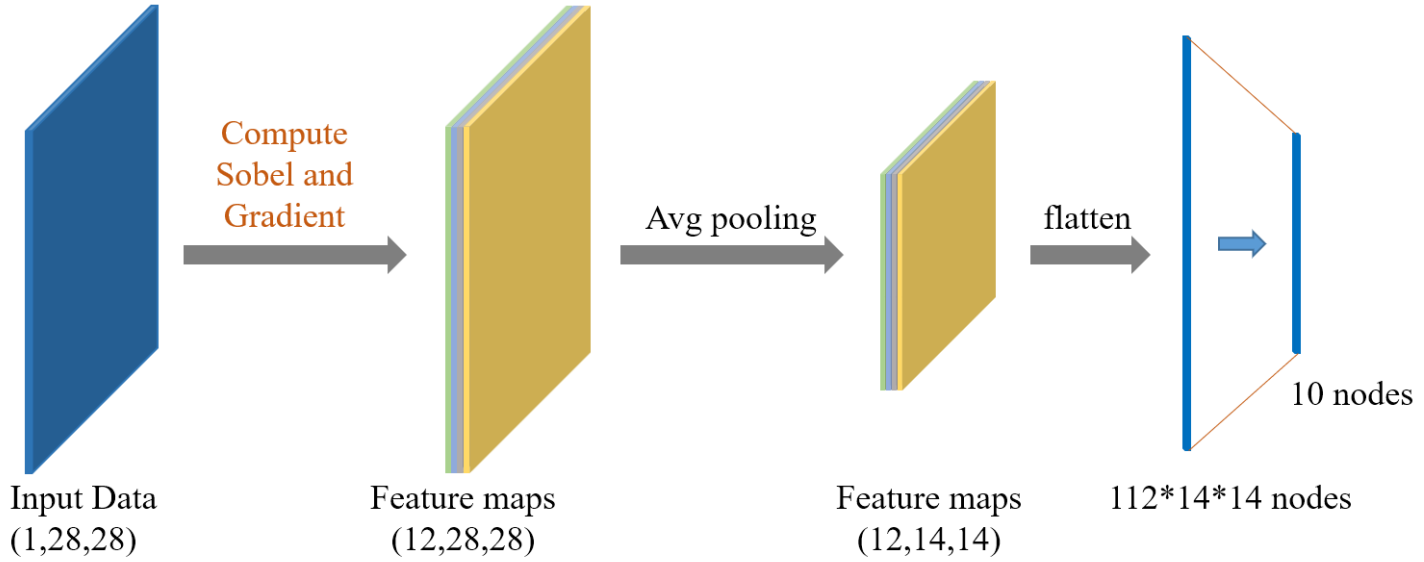
Comparison

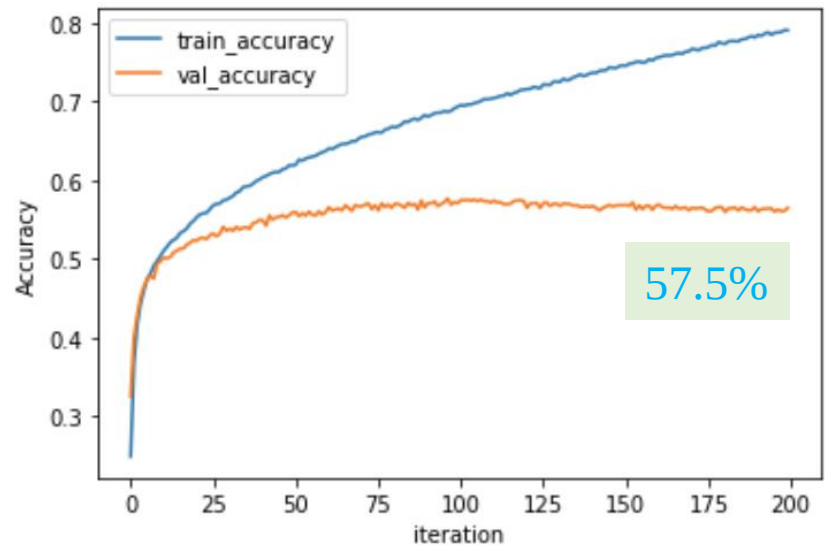
❖ Engineering features vs. CNN features

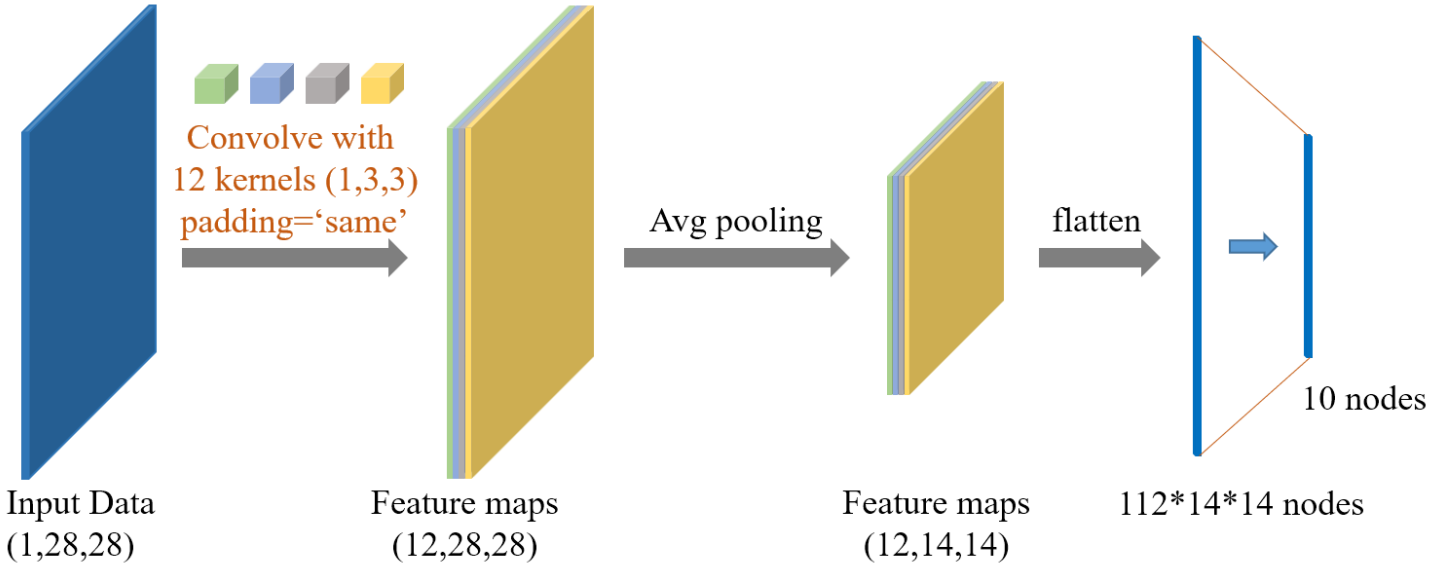


Engineering Features vs. CNN Features using Cifar10









Outline

- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

1x1 Convolution

❖ Why 1x1 Convolution

❖ Flexible input size

Yann LeCun
April 7, 2015 ·

In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.

It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.

Yann LeCun

Yann LeCun in 2018

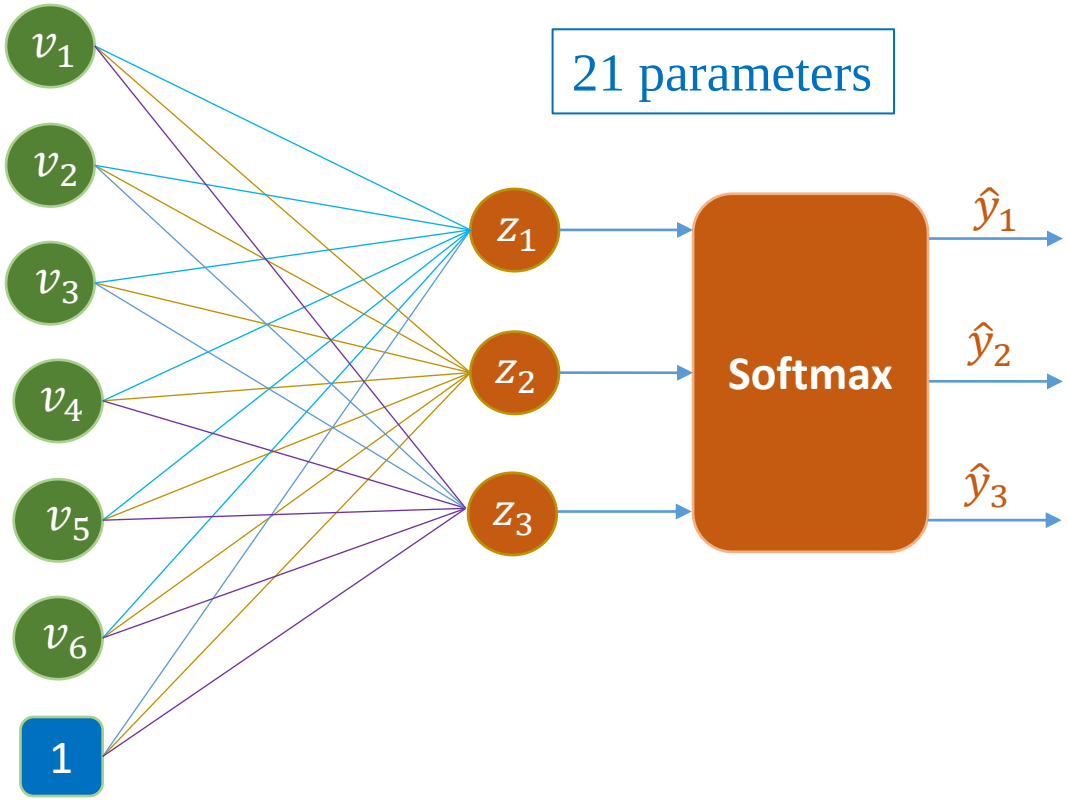
Born	July 8, 1960 (age 60) Soisy-sous-Montmorency, France
Alma mater	ESIEE Paris (MSc) Pierre and Marie Curie University (PhD)
Known for	Deep learning
Awards	Turing Award (2018) AAAI Fellow (2019) Legion of Honour (2020)
	Scientific career
Institutions	Bell Labs (1988-1996) New York University Facebook
Thesis	*Modèles connexionnistes de l'apprentissage (connectionist learning models)* (1987a)
Doctoral advisor	Maurice Milgram
Website	yann.lecun.com

1x1 Convolution

❖ Comparison

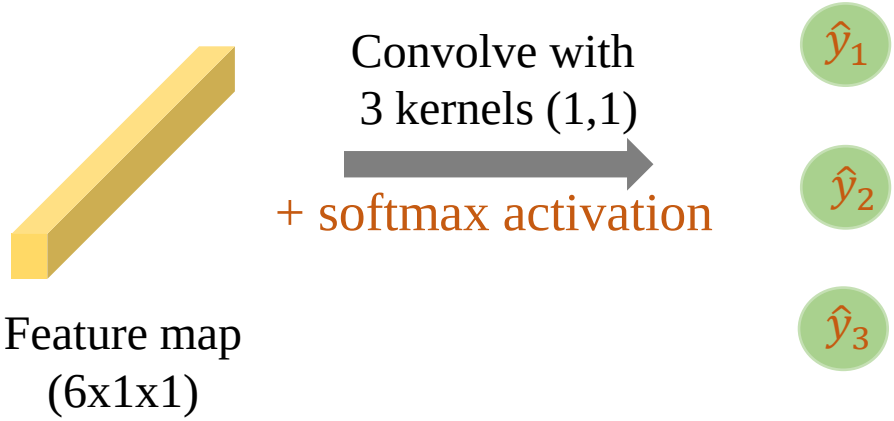
Fully connected layer

21 parameters



CNN

21 parameters



1x1 Convolution

Replace FC by 1x1 Conv

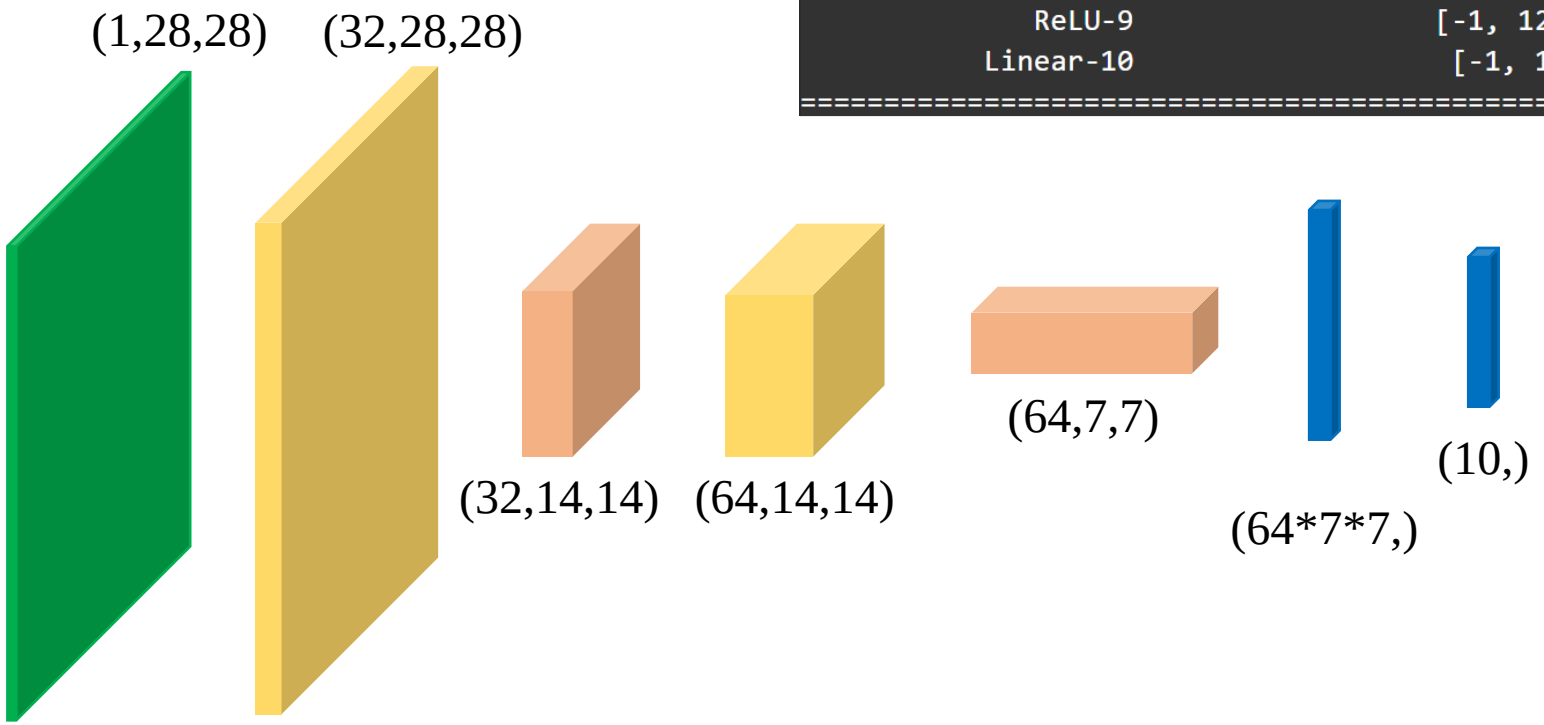
(3x3) Conv with stride=1, padding='same' + ReLU

(3x3) Conv with stride=2, padding='same' + ReLU

(7x7) Conv + ReLU

```
LeNet = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(7*7*64, 128),
    nn.ReLU(),
    nn.Linear(128, 10)
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	320
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	18,496
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Flatten-7	[-1, 3136]	0
Linear-8	[-1, 128]	401,536
ReLU-9	[-1, 128]	0
Linear-10	[-1, 10]	1,290

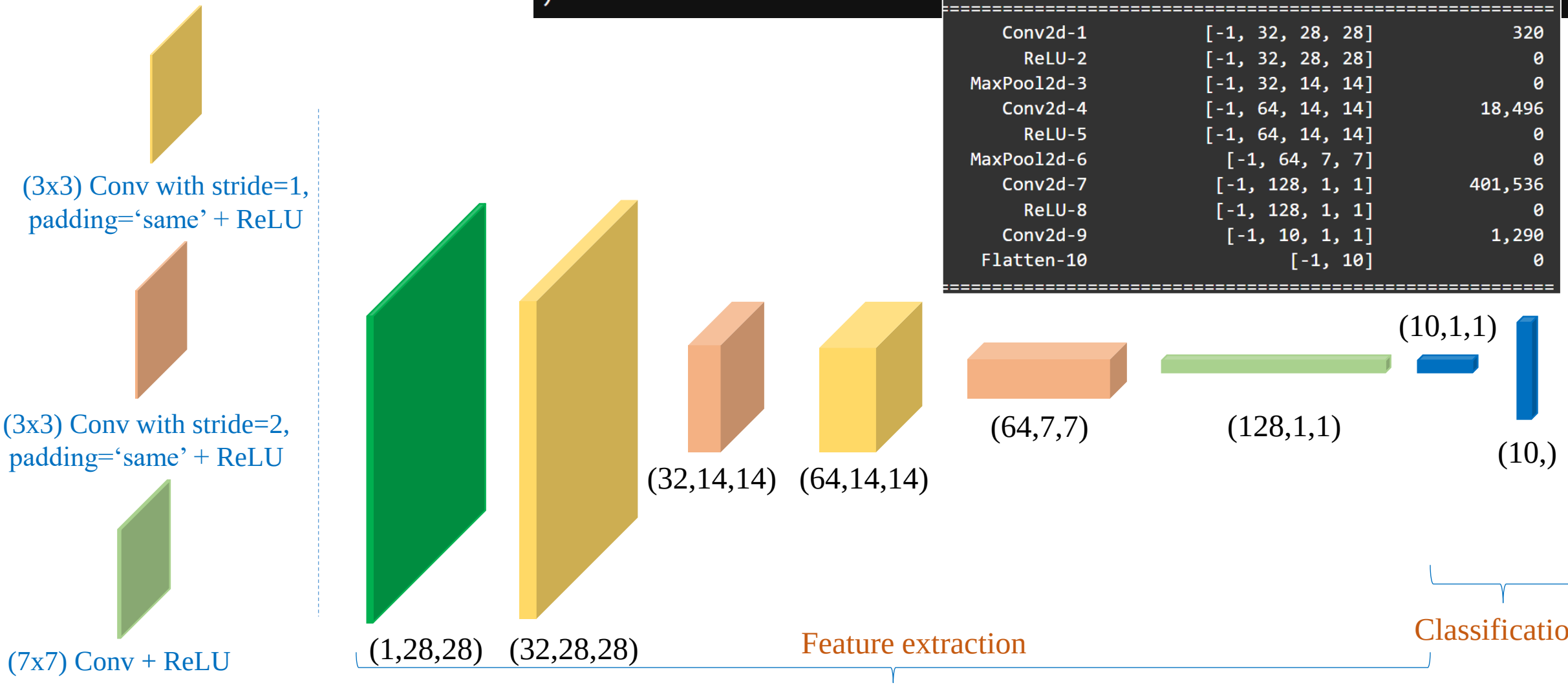


1x1 Convolution

Replace FC by 1x1 Conv

```
LeNet = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(64, 128, kernel_size=7), nn.ReLU(),
    nn.Conv2d(128, 10, kernel_size=1),
    nn.Flatten()
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	320
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	18,496
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Conv2d-7	[-1, 128, 1, 1]	401,536
ReLU-8	[-1, 128, 1, 1]	0
Conv2d-9	[-1, 10, 1, 1]	1,290
Flatten-10	[-1, 10]	0

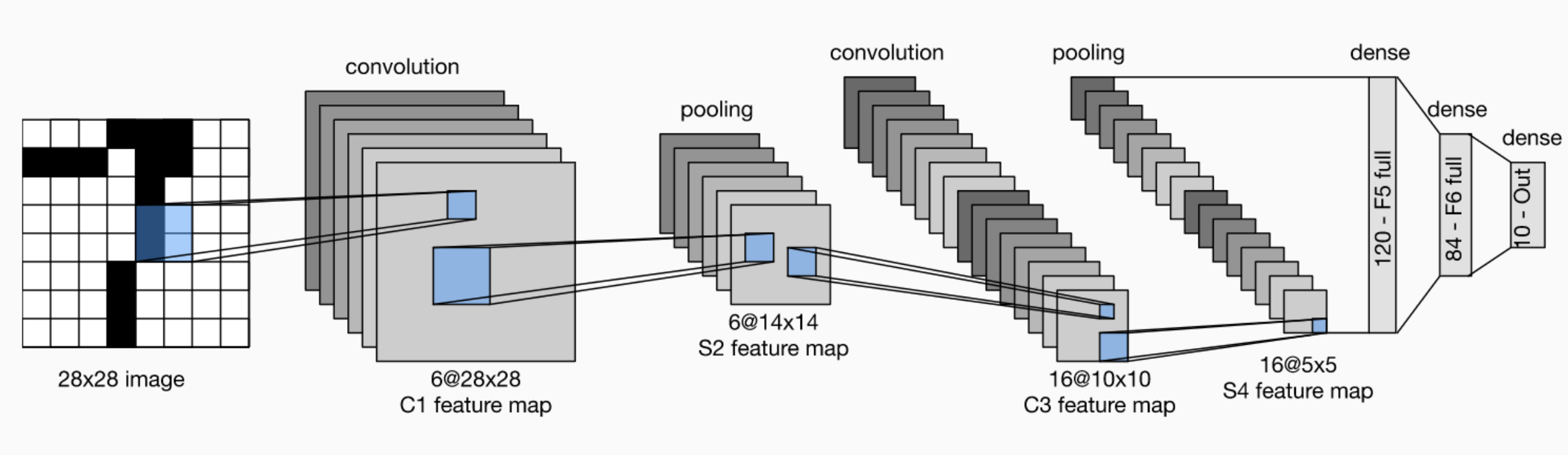


Outline

- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

LeNet Architecture

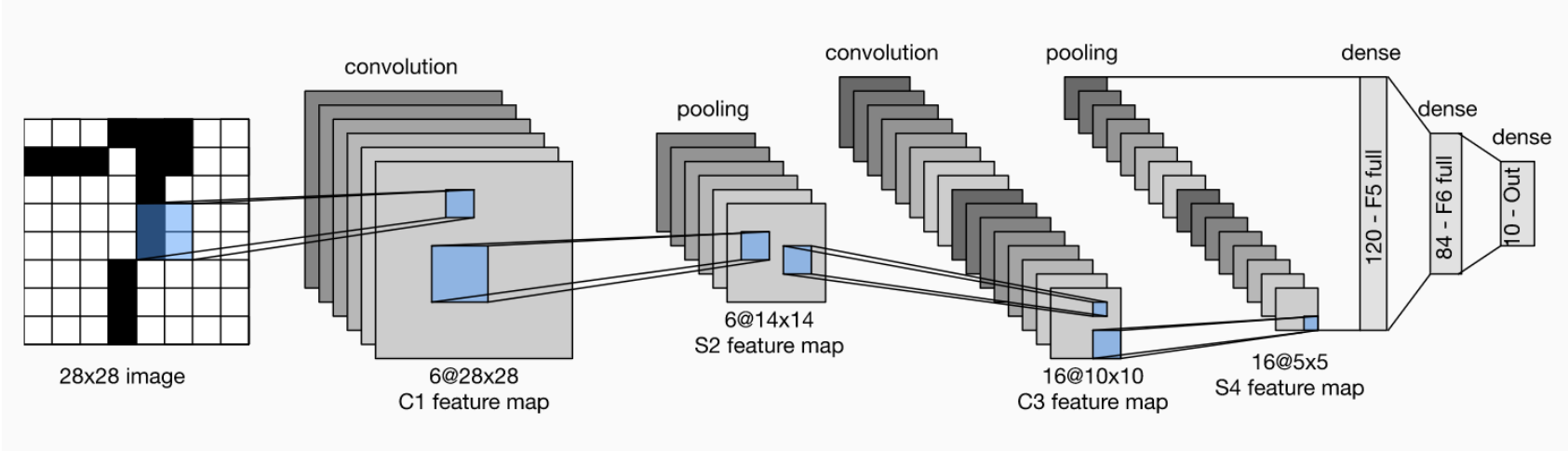
❖ Model Construction



https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

LeNet Architecture

❖ Model Construction

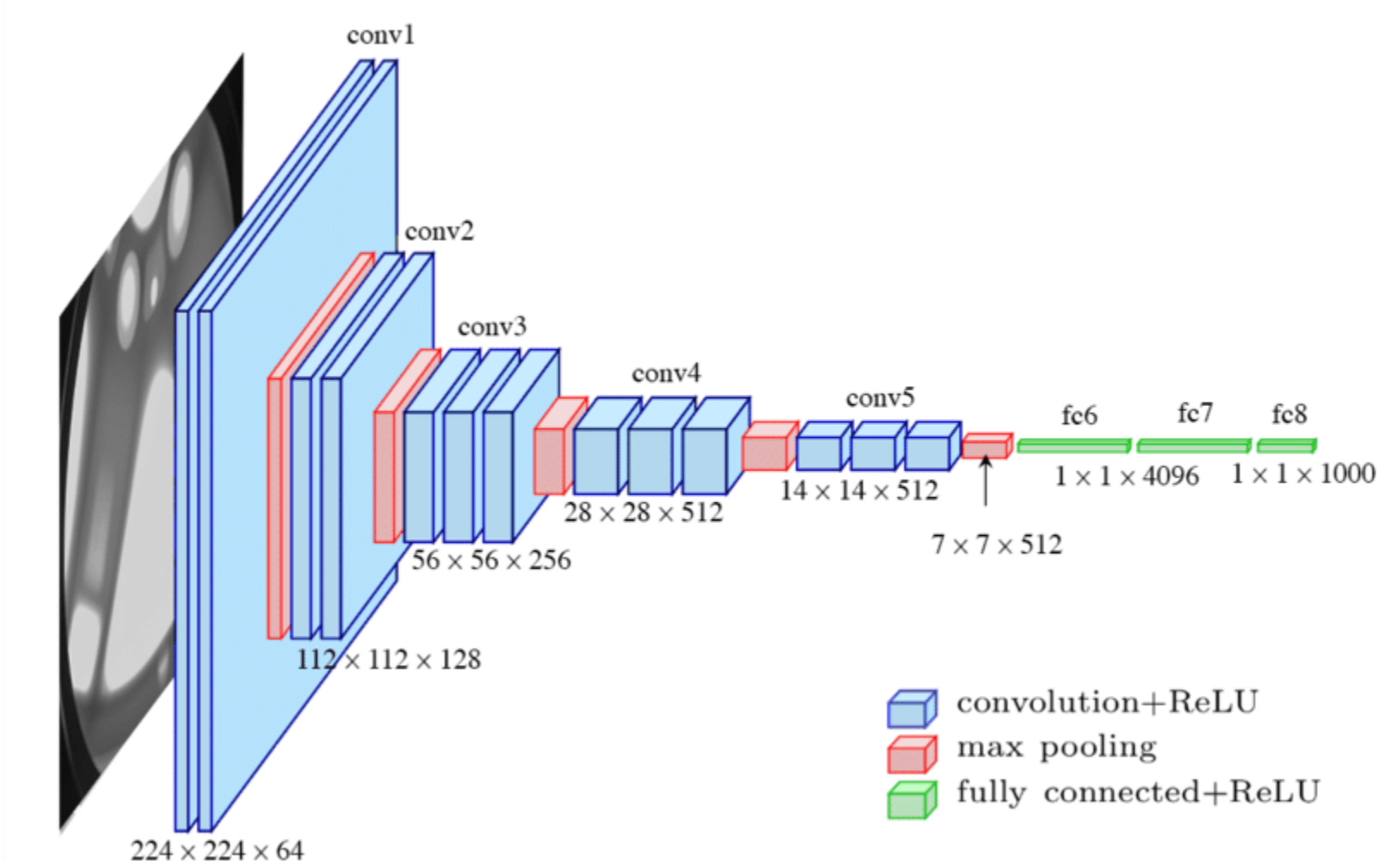



```
LeNet = nn.Sequential(
    nn.Conv2d(1, 6, 5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(6, 16, 5),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Flatten(),
    nn.Linear(16*5*5, 120),
    nn.ReLU(),

    nn.Linear(120, 84),
    nn.ReLU(),
    nn.Linear(84, 10)
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 6, 28, 28]             156
              ReLU-2          [-1, 6, 28, 28]               0
         MaxPool2d-3          [-1, 6, 14, 14]               0
            Conv2d-4         [-1, 16, 10, 10]           2,416
              ReLU-5         [-1, 16, 10, 10]               0
         MaxPool2d-6           [-1, 16, 5, 5]               0
           Flatten-7                [-1, 400]               0
            Linear-8                [-1, 120]          48,120
              ReLU-9                [-1, 120]               0
           Linear-10                 [-1, 84]          10,164
             ReLU-11                 [-1, 84]               0
           Linear-12                 [-1, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
```

VGG16 Architecture

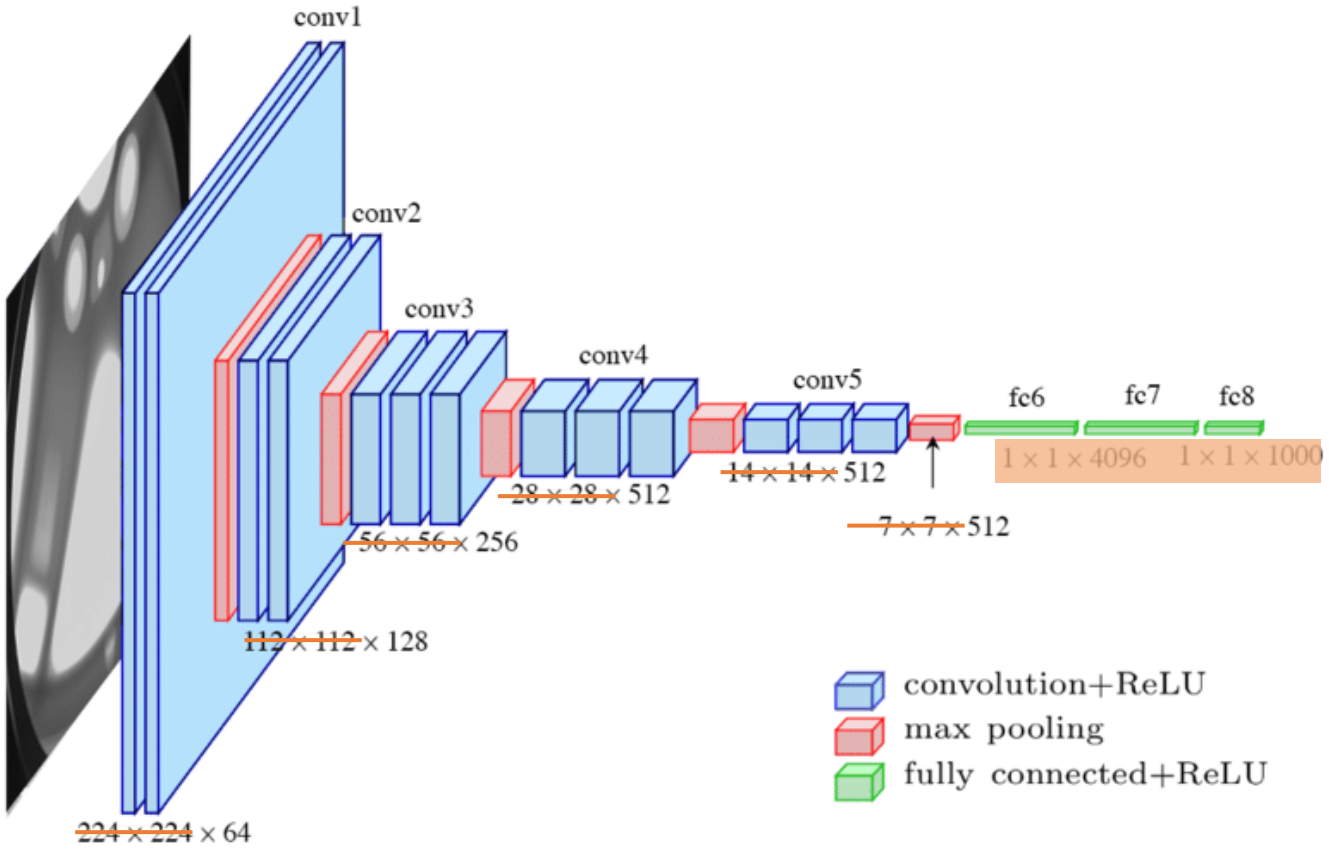
❖ Model Construction



```
self.relu = nn.ReLU()
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(64, 64, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(64, 128, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(128, 128, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(128, 256, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(256, 256, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(256, 256, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(256, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2)
)

self.classifier = nn.Sequential(
    nn.Flatten(), nn.Linear(512 * 1 * 1, 128), self.relu,
    nn.Linear(128, 128), self.relu, nn.Linear(128, 10)
)
```

VGG16 Architecture

❖ Model Construction



Further Reading

❖ Reading

https://cs231n.github.io/convolutional-networks/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

AI VIETNAM
All-in-One Course

CNN Training

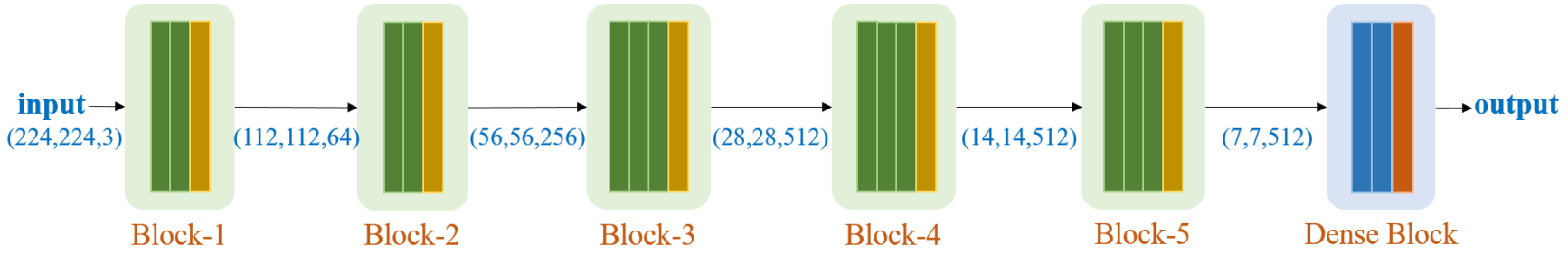
How to increase training accuracy?

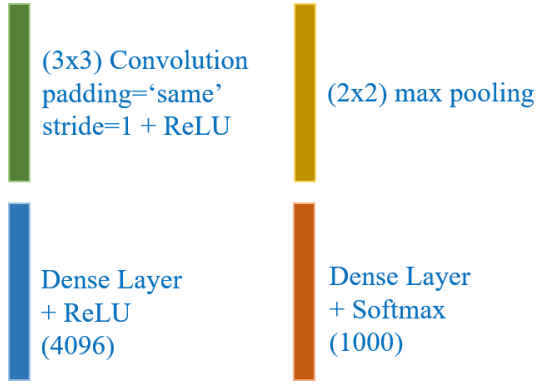
Quang-Vinh Dinh
Ph.D. in Computer Science

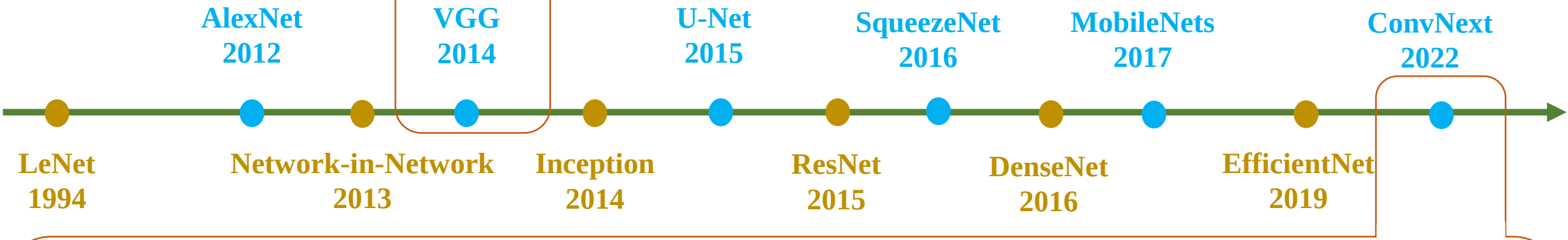
Year 2023

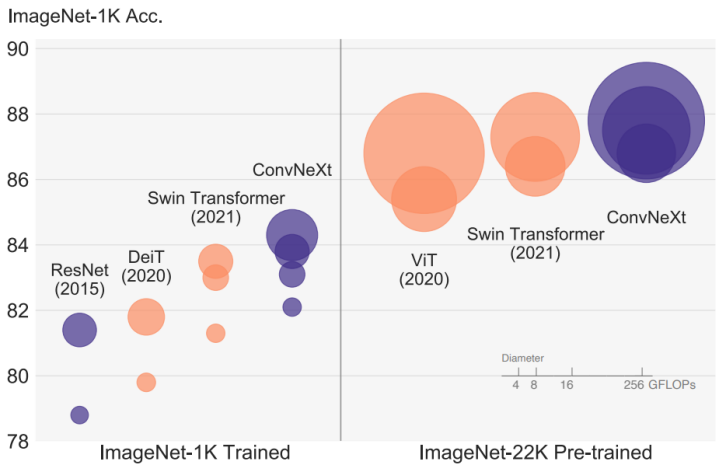
Outline

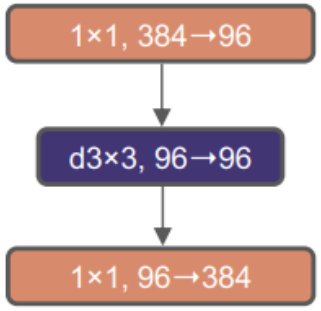
- Network Architectures
- Network Training
- Case Study
- Problem-Solving Approach

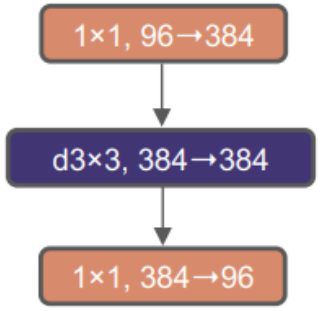

input
(224,224,3)
(112,112,64)
(56,56,256)
(28,28,512)
(14,14,512)
(7,7,512)
output
Block-1
Block-2
Block-3
Block-4
Block-5
Dense Block

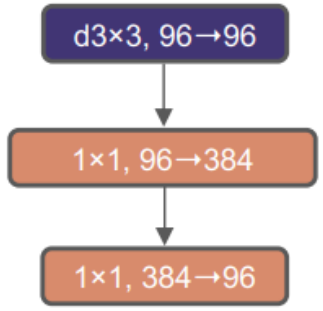

(3x3) Convolution
padding='same'
stride=1 + ReLU
(2x2) max pooling
Dense Layer
+ ReLU
(4096)
Dense Layer
+ Softmax
(1000)

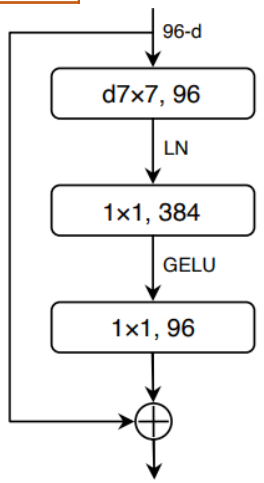

LeNet
1994
AlexNet
2012
Network-in-Network
2013
VGG
2014
Inception
2014
U-Net
2015
ResNet
2015
SqueezeNet
2016
DenseNet
2016
MobileNets
2017
EfficientNet
2019
ConvNext
2022

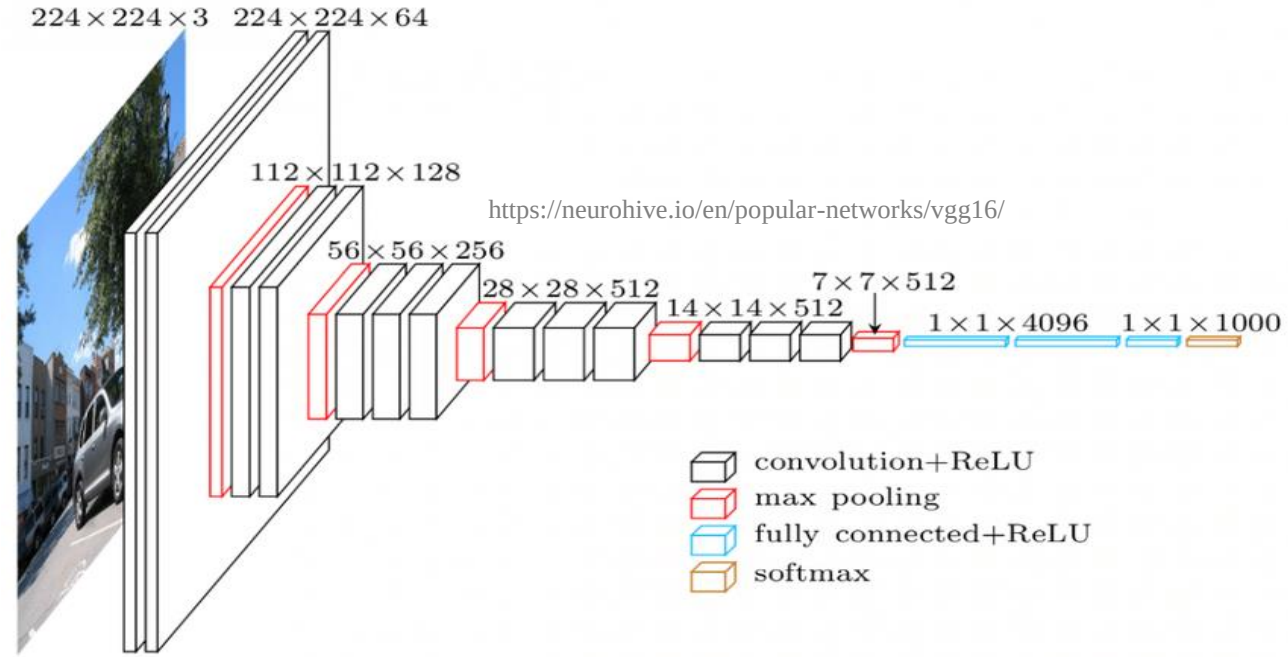

ImageNet-1K Acc.
90
88
86
84
82
80
78
ConvNeXt
Swin Transformer
(2021)
DeiT
(2020)
ResNet
(2015)
ViT
(2020)
Swin Transformer
(2021)
ConvNeXt
Diameter
4 8 16 256 GFLOPs
ImageNet-1K Trained
ImageNet-22K Pre-trained


1×1, 384→96
d3×3, 96→96
1×1, 96→384


1×1, 96→384
d3×3, 384→384
1×1, 384→96


d3×3, 96→96
1×1, 96→384
1×1, 384→96


96-d
d7×7, 96
LN
1×1, 384
GELU
1×1, 96



https://neurohive.io/en/popular-networks/vgg16/

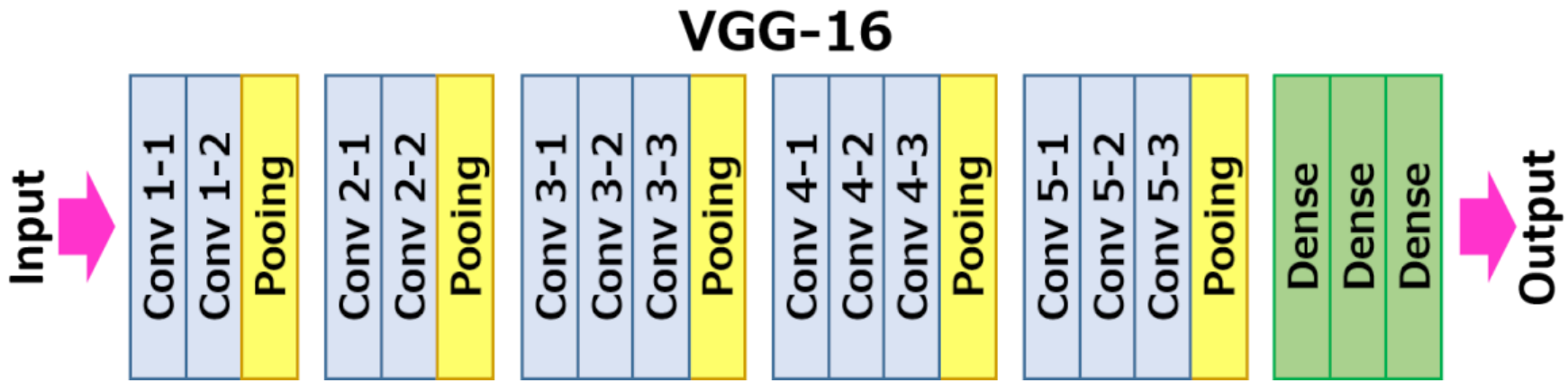
(3x3) Convolution padding='same' stride=1 + ReLU

(2x2) max pooling

Dense Layer + ReLU (4096)

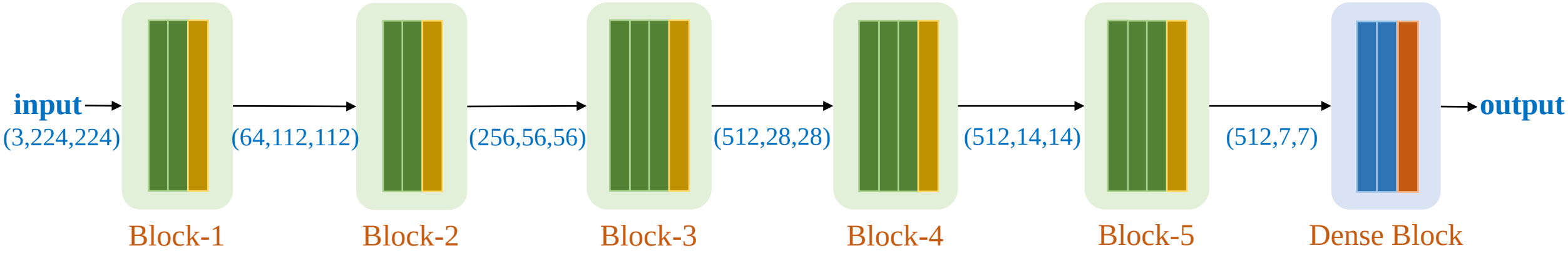
Dense Layer + Softmax (1000)

VGG16



CNN Architectures

❖ VGG16 for ImageNet



(3x3) Convolution
padding='same'
stride=1 + ReLU

(2x2) max pooling

Dense Layer
+ ReLU
(4096)

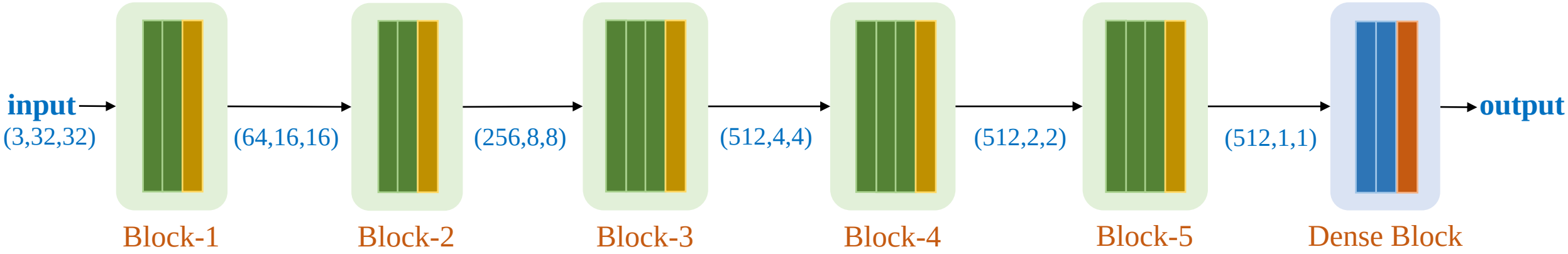
Dense Layer
+ Softmax
(1000)

```python
# Define the blocks
block1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block2 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block3 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block4 = nn.Sequential(
    nn.Conv2d(256, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block5 = nn.Sequential(
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
```

```python
# Classifier
classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(512*7*7, 4096), nn.ReLU(inplace=True),
    nn.Linear(4096, 4096), nn.ReLU(inplace=True),
    nn.Linear(4096, 1000),
)

# Combine all blocks into one model
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.block1 = block1
        self.block2 = block2
        self.block3 = block3
        self.block4 = block4
        self.block5 = block5
        self.classifier = classifier

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.classifier(x)
        return x

# Instantiate the model
model = VGG16()
```

CNN Architectures

❖ VGG16-like for Cifar-10



```python
# Define the blocks
block1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block2 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block3 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block4 = nn.Sequential(
    nn.Conv2d(256, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block5 = nn.Sequential(
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
```

```python
# Classifier
classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(512*1*1, 256), nn.ReLU(inplace=True),
    nn.Linear(256, 256), nn.ReLU(inplace=True),
    nn.Linear(256, 10),
)

# Combine all blocks into one model
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.block1 = block1
        self.block2 = block2
        self.block3 = block3
        self.block4 = block4
        self.block5 = block5
        self.classifier = classifier

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.classifier(x)
        return x

# Instantiate the model
model = VGG16()
```

Outline

- Network Architectures
- Network Training
- Case Study
- Problem-Solving Approach

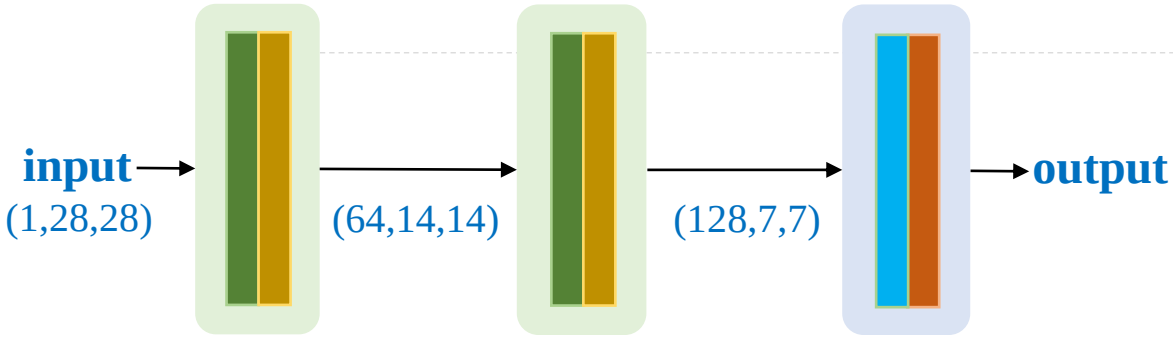
Image Data

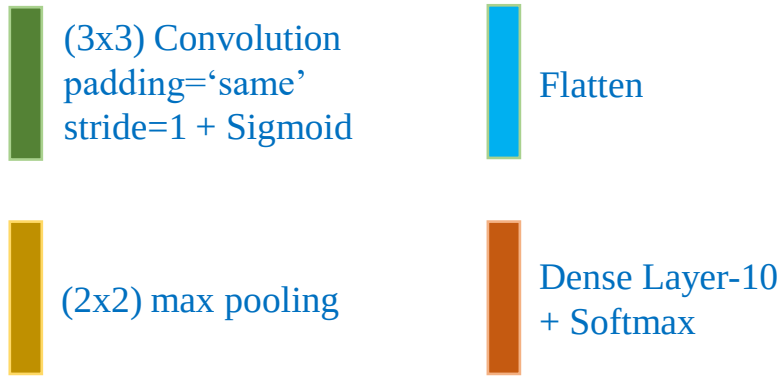
Fashion-MNIST dataset

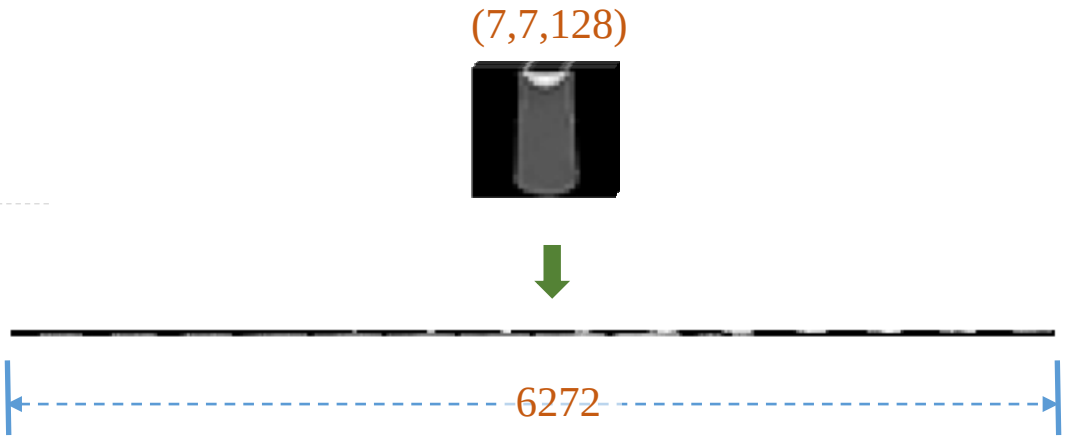
- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

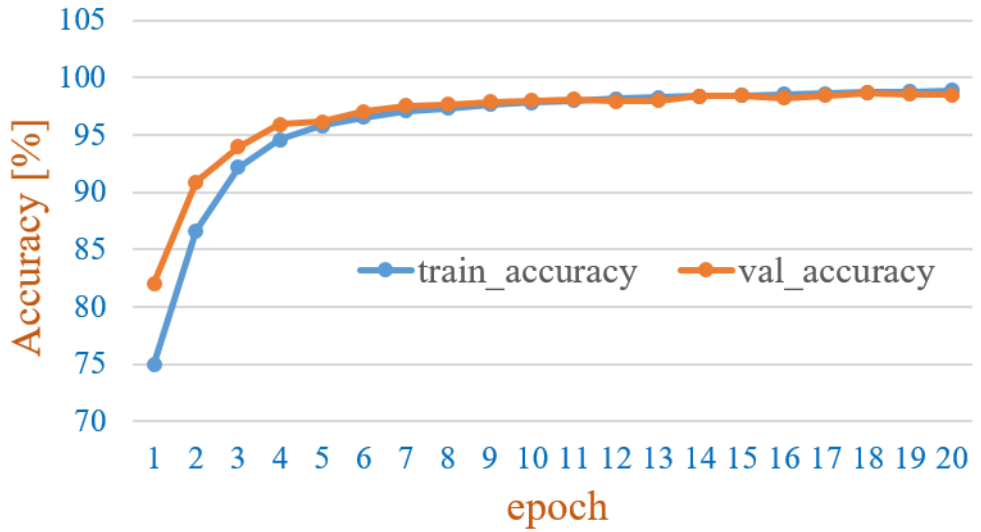
Network Training

❖ Fashion-MNIST dataset









AI VIETNAM
All-in-One Course

Network Training

❖ Fashion-MNIST dataset

X-data format

(batch, channel, height, width)

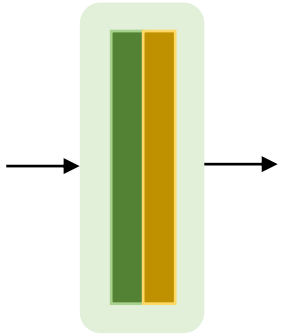
Data normalization [0,1]

(3x3) Convolution with 64 filters,
stride=1, padding='same'
+ Sigmoid activation
+ glorot_uniform initialization

Adam optimizer and Cross-entropy loss

(3x3) Convolution
padding='same'
stride=1 + Sigmoid

(2x2) max pooling

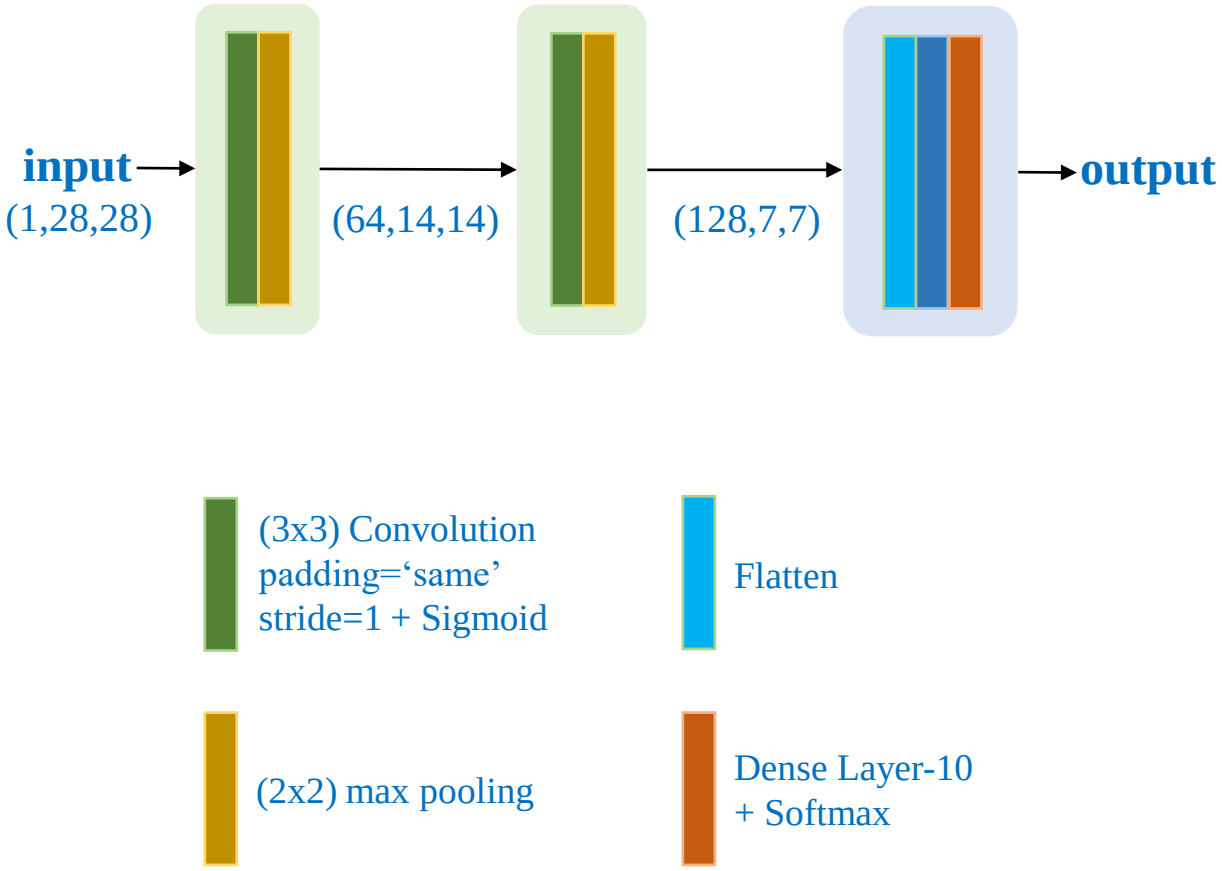


```python
# Data
transform = Compose([transforms.ToTensor()])
train_set = FashionMNIST(root='data',
                         train=True,
                         download=True,
                         transform=transform)
trainloader = DataLoader(train_set,
                         batch_size=256,
                         shuffle=True,
                         num_workers=4)
```

```python
import torch.nn as nn
import torch.nn.init as init

block = nn.Sequential(nn.Conv2d(1, 64, 3,
                                stride=1,
                                padding='same'),
                      nn.Sigmoid(),
                      nn.MaxPool2d(2, 2))

for m in block:
    if isinstance(m, nn.Conv2d):
        init.xavier_uniform_(m.weight)
        if m.bias is not None:
            init.zeros_(m.bias)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-3)
```

Network Training

❖ Fashion-MNIST dataset




```python
# Declare layers
conv_layer1 = nn.Sequential(
    nn.Conv2d(1, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(128*7*7, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, 10)

# Given data x
x = conv_layer1(x)
x = conv_layer2(x)
x = flatten(x)
x = fc_layer1(x)
x = fc_layer2(x)
```

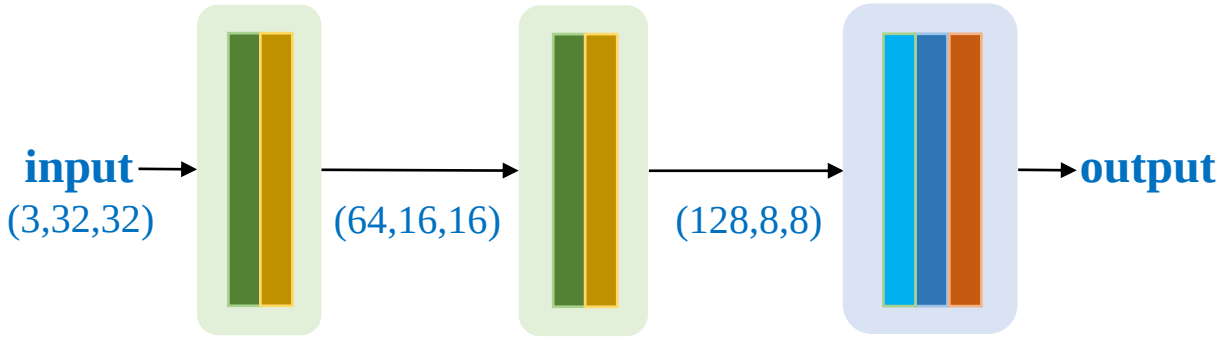
Network Training

Cifar-10 dataset
(more complex dataset)

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples

Network Training

❖ Cifar-10 dataset



Data normalization [0,1]

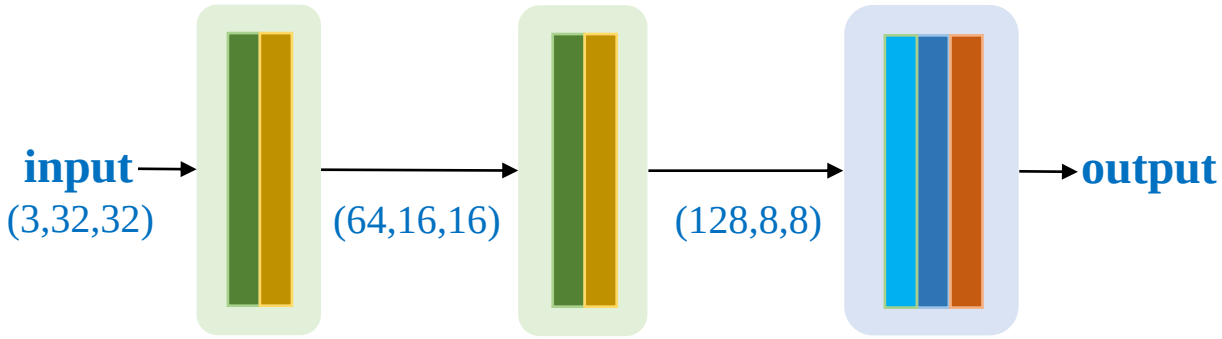
Glorot uniform initialization

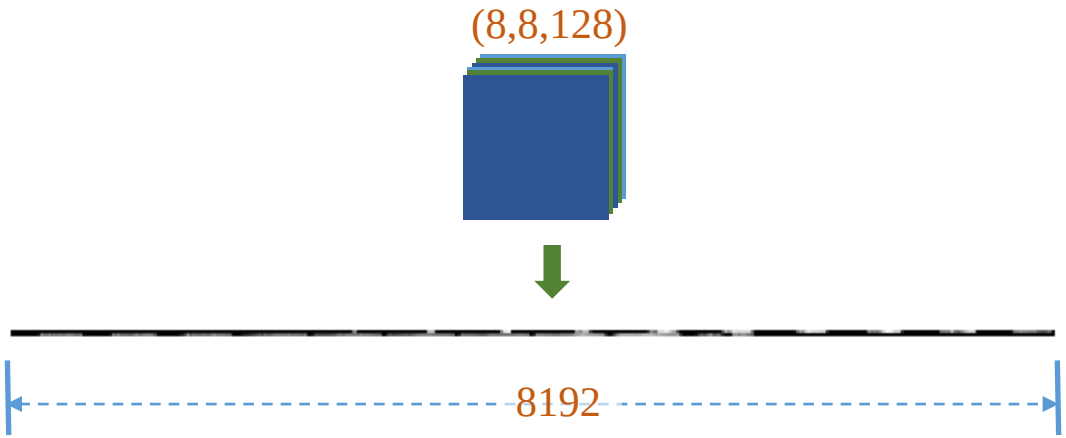
Adam optimizer with lr=1e-3

```
conv_layer1 = nn.Sequential(
    nn.Conv2d(1, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(128*8*8, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, 10)

# Given data x
x = conv_layer1(x)
x = conv_layer2(x)
x = flatten(x)
x = fc_layer1(x)
x = fc_layer2(x)
```

Network Training

❖ Cifar-10 dataset



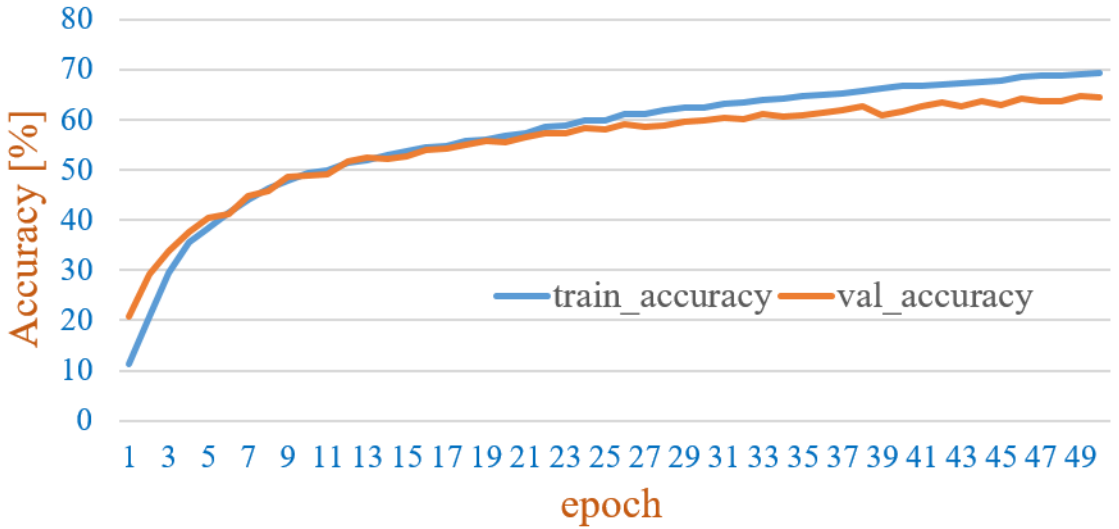


(3x3) Convolution
padding='same'
stride=1 + Sigmoid

Flatten

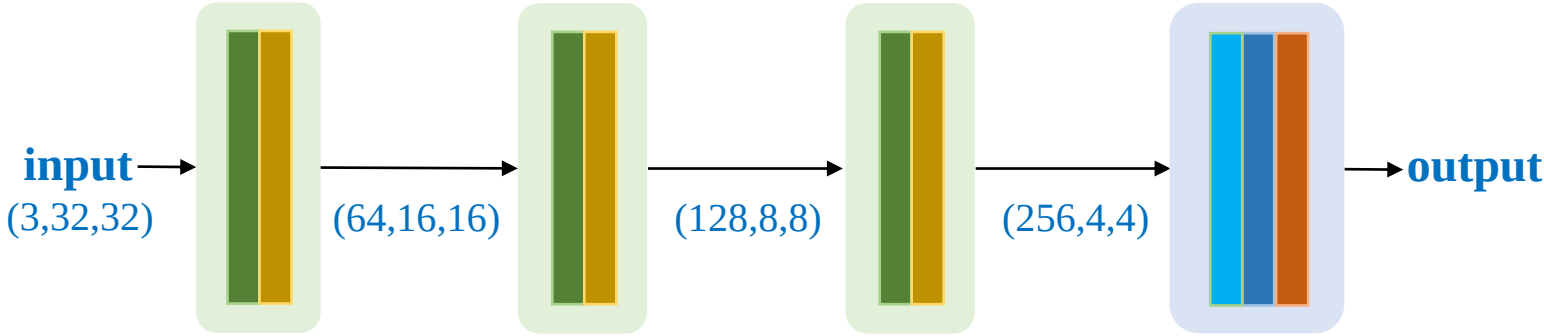
(2x2) max pooling

Dense Layer-10
+ Softmax



Accuracy: 69.3% - Val_accuracy: 64.5%

Network Training



❖ Cifar-10 dataset:

❖Adding more layers

(3x3) Convolution
padding='same'
stride=1 + Sigmoid

Flatten

(2x2) max pooling

Dense Layer-10
+ Softmax

```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer3 = nn.Sequential(
    nn.Conv2d(128, 256, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(256*4*4, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, n_classes)
```

Data normalization [0,1]

Glorot uniform initialization

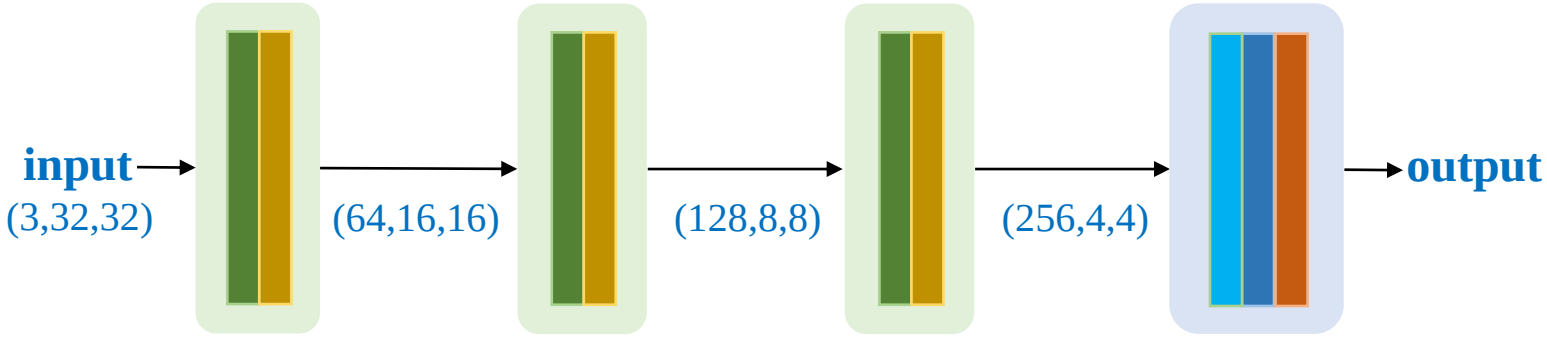
Adam optimizer with lr=1e-3

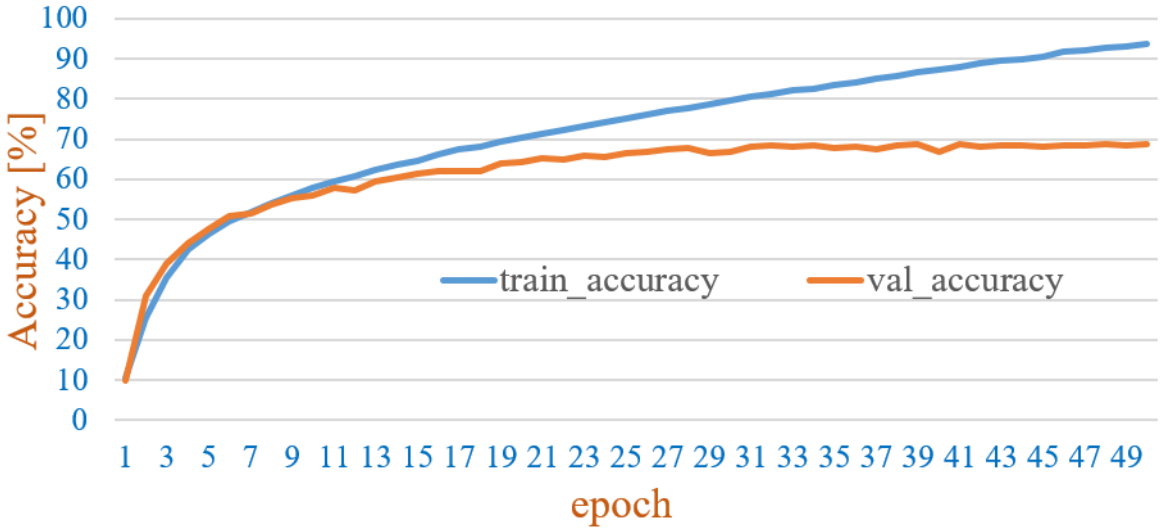
Network Training

❖ Cifar-10 dataset:

❖Adding more layers

Good news: Network accuracy increases about 25%





(3x3) Convolution padding='same' stride=1 + Sigmoid

Flatten

(2x2) max pooling

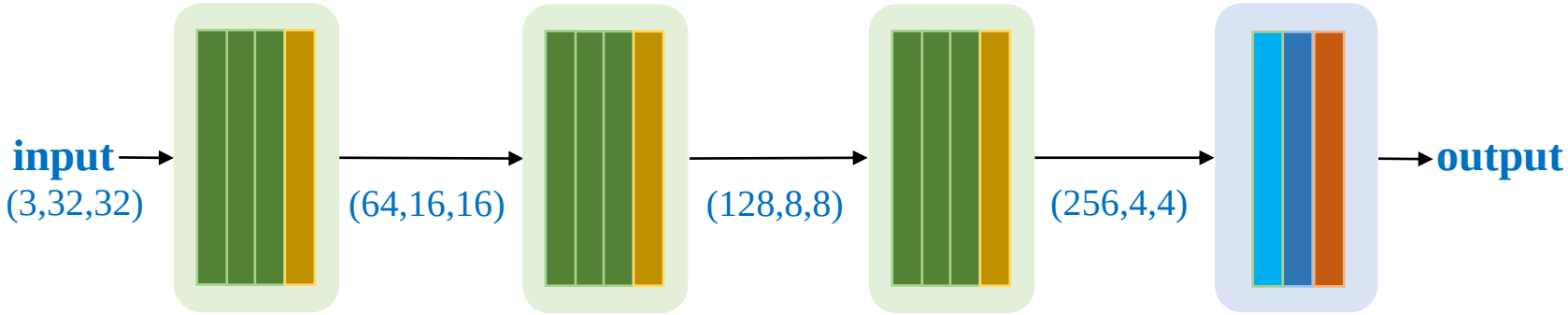
Dense Layer-10 + Softmax

Accuracy: 93.8% - Val_accuracy: 68.7%

Network Training

Cifar-10 dataset:

- ❖ **Keep adding more layers**



(3x3) Convolution padding='same' stride=1 + Sigmoid

Flatten

(2x2) max pooling

Dense Layer-10 + Softmax

```
conv_layer1 = nn.Sequentialn(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid(),
                            nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.Sigmoid(),
                            nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.Sigmoid(),
                            nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(256*4*4, 512), nn.Sigmoid())
fc_layer2 = nn.Linear(512, 10)
```

Data normalization [0,1]

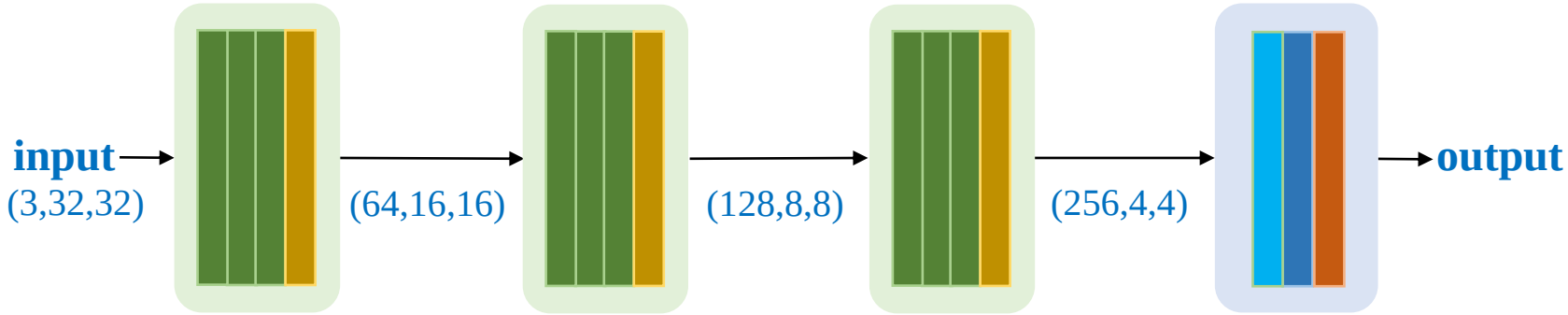
Glorot uniform initialization

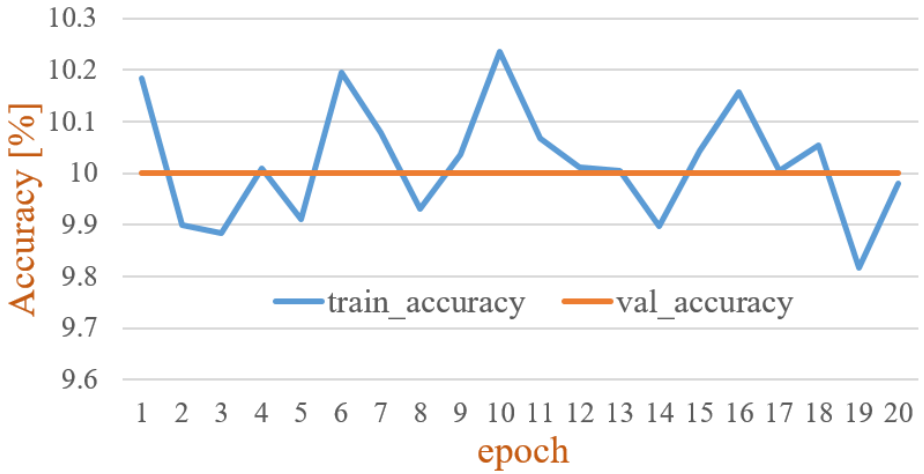
Adam optimizer with lr=1e-3

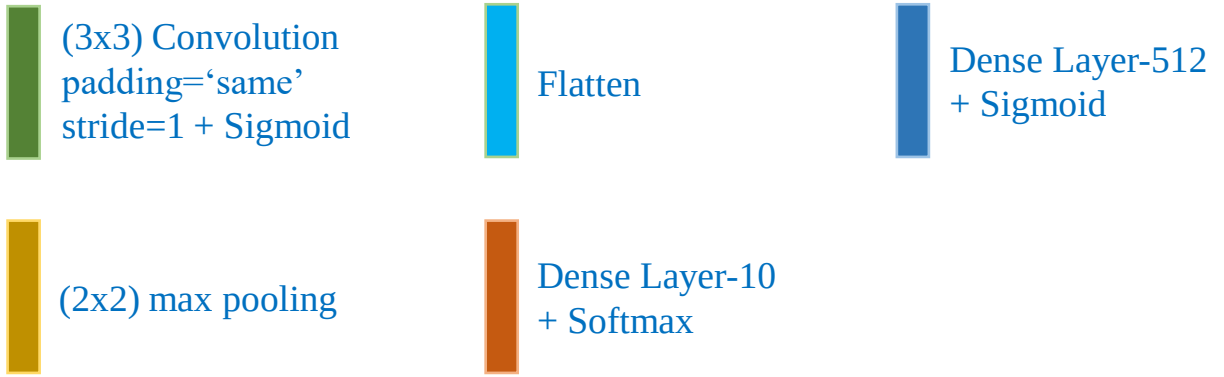
Network Training

- **Cifar-10 dataset:**
  - **Keep adding more layers**

The network does not learn







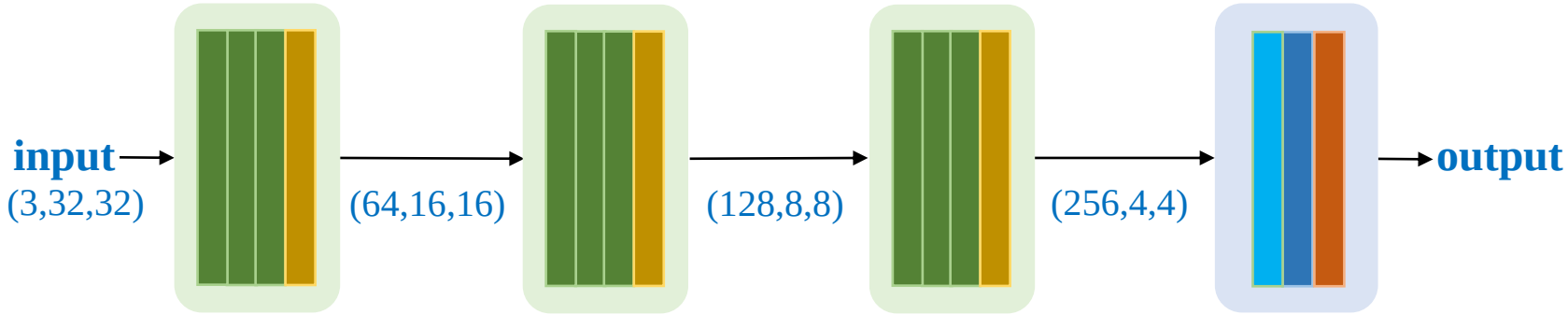
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

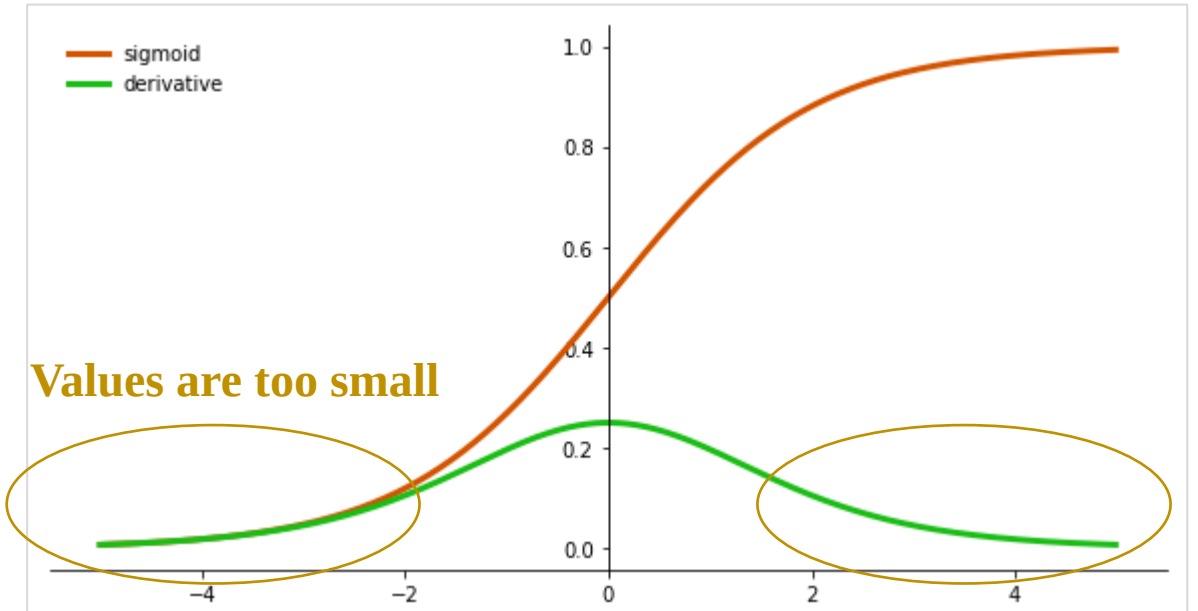
(3x3) Convolution
padding='same'
stride=1 + Sigmoid

Dense Layer-512
+ Sigmoid



$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Vanishing Problem



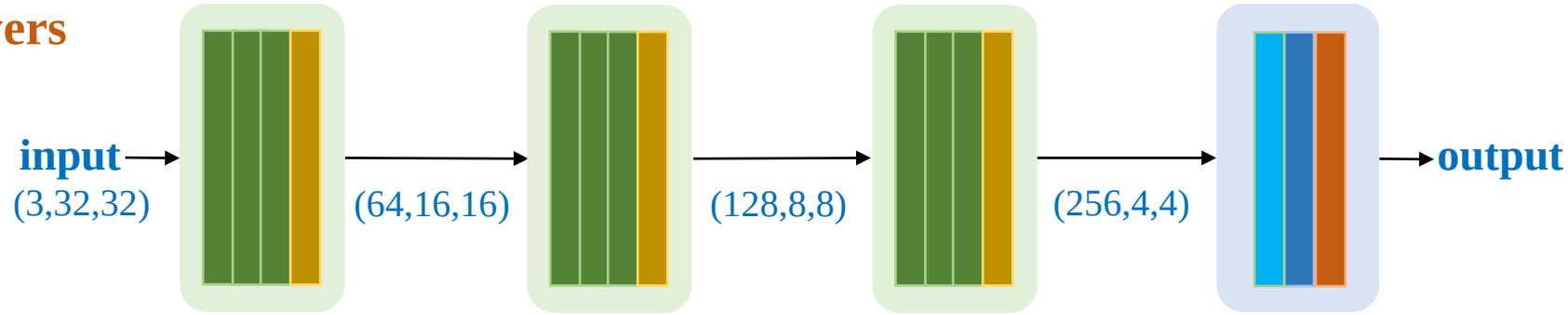
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

(3x3) Convolution
padding='same'
stride=1 + ReLU

Dense Layer-512
+ ReLU

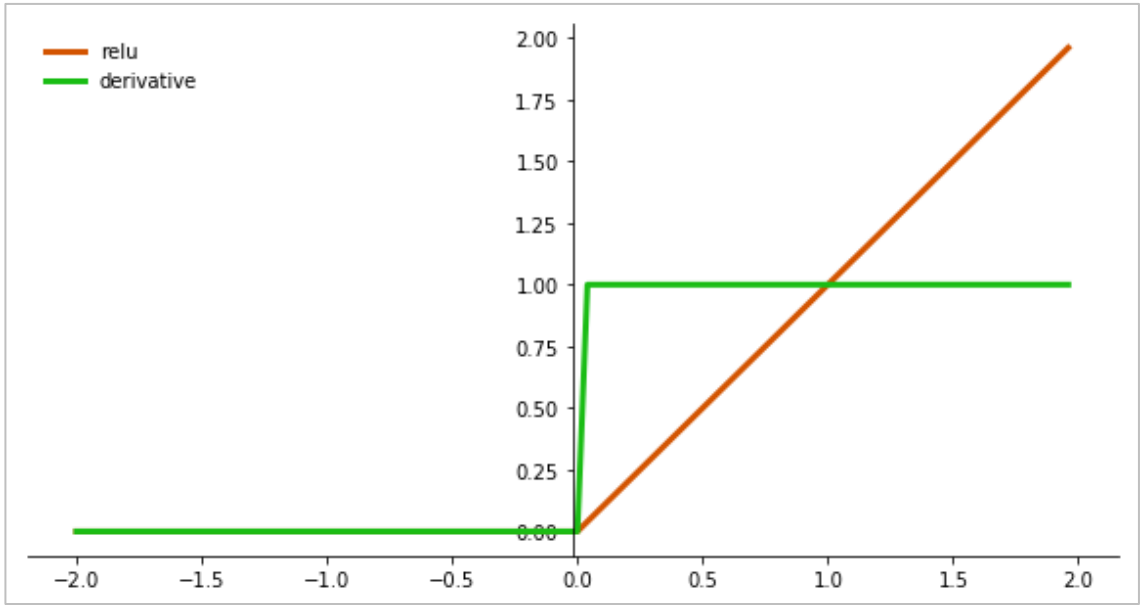


$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

```
nn.Conv2D(...), nn.Sigmoid()
```

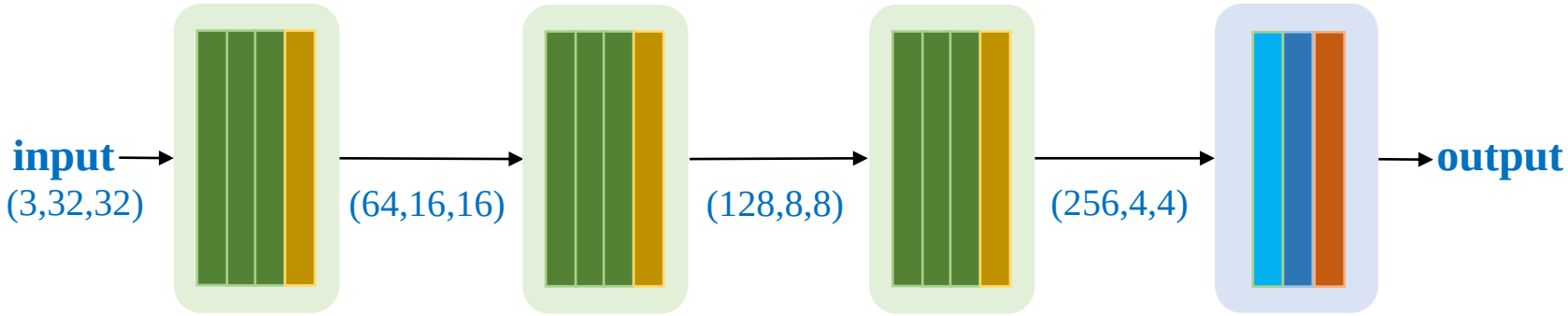


```
nn.Conv2D(...), nn.ReLU()
```



Network Training

- **Cifar-10 dataset:**
  - **Use ReLU**




```
conv_layer1 = nn.Sequentialn(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(256*4*4, 512), nn.ReLU())
fc_layer2 = nn.Linear(512, 10)
```

(3x3) Convolution padding='same' stride=1 + ReLU

Flatten

(2x2) max pooling

Dense Layer-10 + Softmax

Dense Layer-512 + ReLU

Data normalization [0,1]

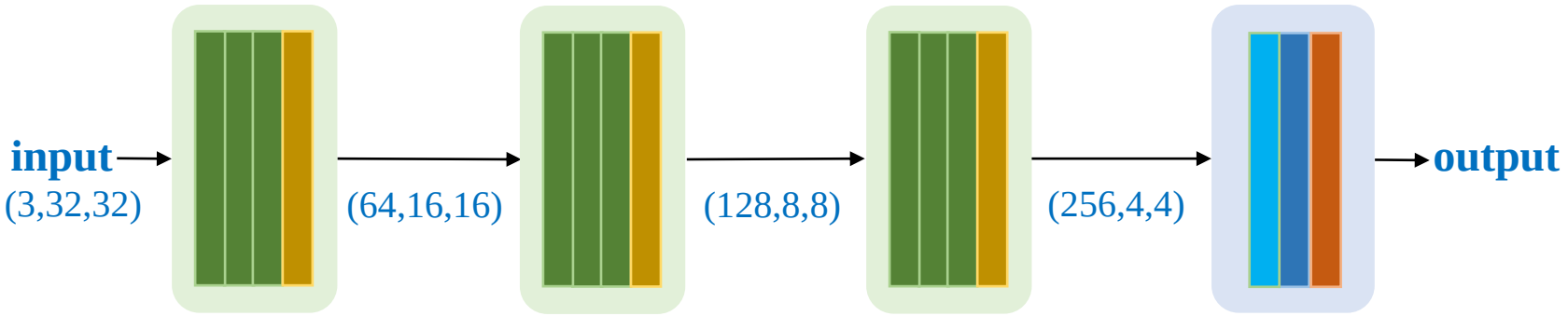
Glorot uniform initialization

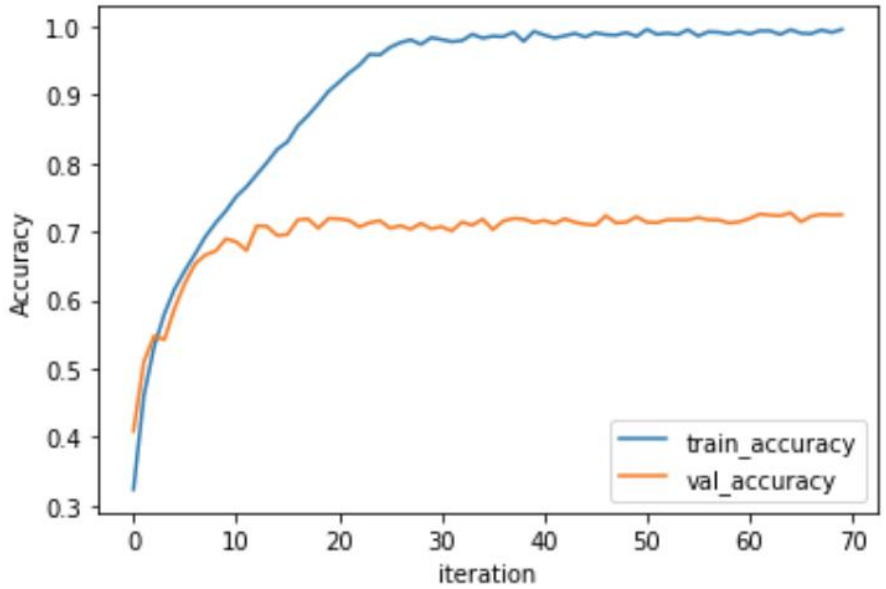
Adam optimizer with lr=1e-3

Network Training

- **Cifar-10 dataset:**
  - **Use ReLU**

Training Accuracy reaches up to 99%





(3x3) Convolution padding='same' stride=1 + ReLU

Flatten

Dense Layer-512 + ReLU

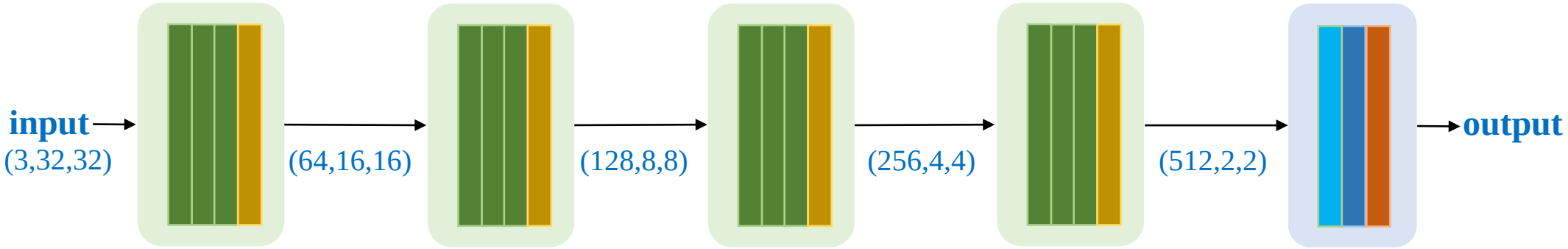
(2x2) max pooling

Dense Layer-10 + Softmax

Adding more layers; Hope reach to 100%

Network Training

Use ReLU and add more layers



Data normalization [0,1]

Glorot uniform initialization

Adam optimizer with lr=1e-3

(3x3) Convolution padding='same' stride=1 + ReLU

(2x2) max pooling

Flatten

Dense Layer-512 + ReLU

Dense Layer-10 + Softmax

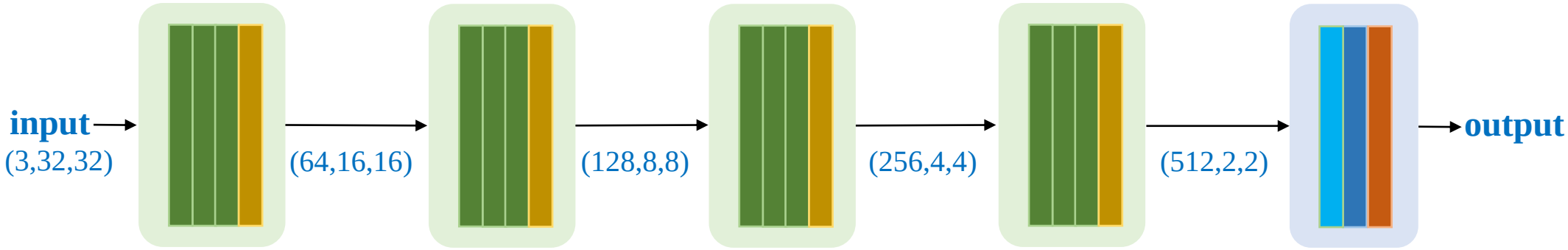
Implementation

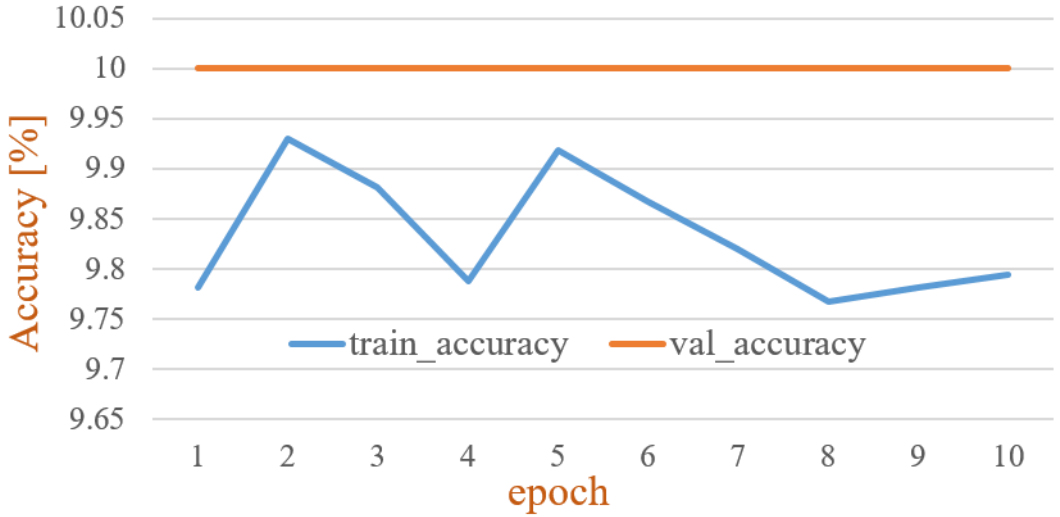
```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),)
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))

conv_layer10 = nn.Sequential(nn.Conv2d(256, 512, 3, stride=1, padding='same'), nn.ReLU())
conv_layer11 = nn.Sequential(nn.Conv2d(512, 512, 3, stride=1, padding='same'), nn.ReLU())
conv_layer12 = nn.Sequential(nn.Conv2d(512, 512, 3, stride=1, padding='same'), nn.ReLU(),
                             nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(512 * 2 * 2, 512), nn.ReLU())
fc_layer2 = nn.Linear(512, 10)
```

```
def initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            init.xavier_uniform_(m.weight)
            if m.bias is not None:
                init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
            if m.bias is not None:
                init.zeros_(m.bias)

def forward(self, x):
    x = self.conv_layer1(x)
    x = self.conv_layer2(x)
    x = self.conv_layer3(x)
    x = self.conv_layer4(x)
    x = self.conv_layer5(x)
    x = self.conv_layer6(x)
    x = self.conv_layer7(x)
    x = self.conv_layer8(x)
    x = self.conv_layer9(x)
    x = self.conv_layer10(x)
    x = self.conv_layer11(x)
    x = self.conv_layer12(x)
    x = self.flatten(x)
    x = self.fc_layer1(x)
    out = self.fc_layer2(x)
    return out
```

Network Training

Use ReLU and add more layers





(3x3) Convolution padding='same' stride=1 + ReLU

Flatten

Dense Layer-512 + ReLU

(2x2) max pooling

Dense Layer-10 + Softmax

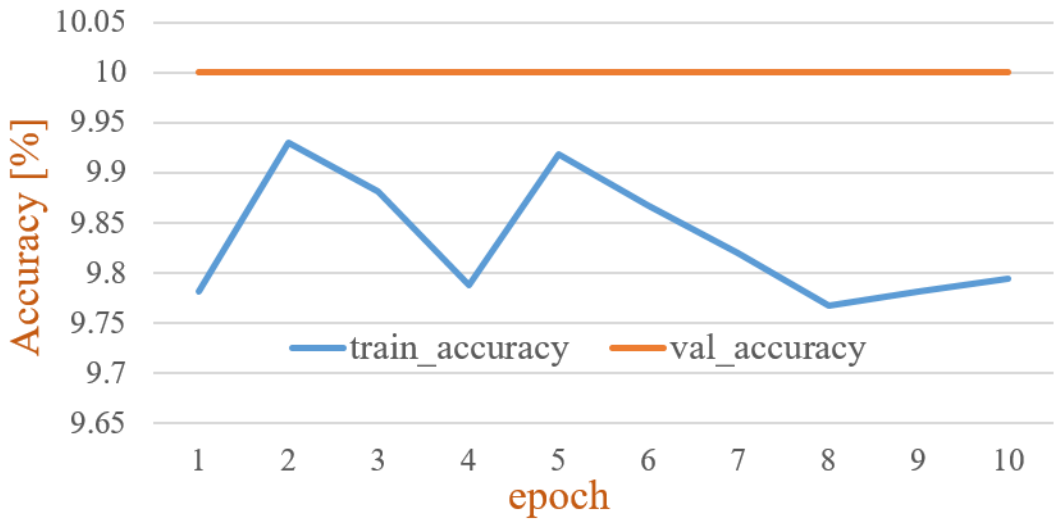
Network does not learn again

Network Training

❖ Summary of the current network

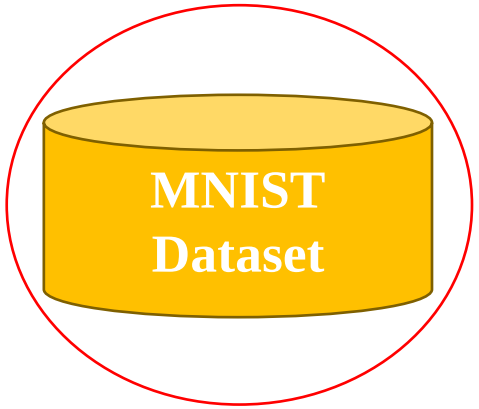
Cifar-10 Dataset → **Data Normalization** (scale to [0,1]) → **Network Construction** (Convs , ReLU, max pooling, Dense layers) → **Parameter Initialization** (Glorot uniform) → **Training** (Adam and cross-entropy loss)

Network does not learn



Network Training

❖ Solution 1: Observation



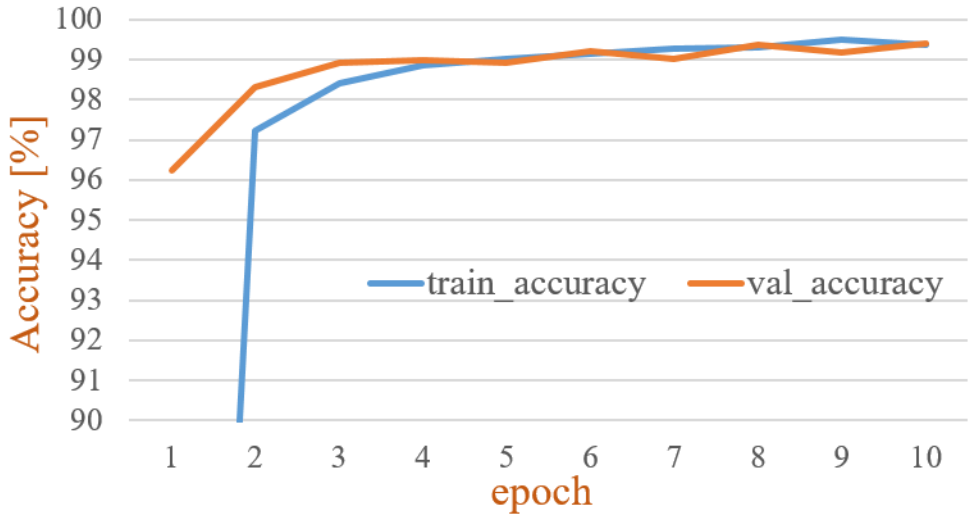
Data Normalization
(scale to [0,1])

Network Construction
(Convs , ReLU, max pooling, Dense layers)

Parameter Initialization
(Glorot uniform)

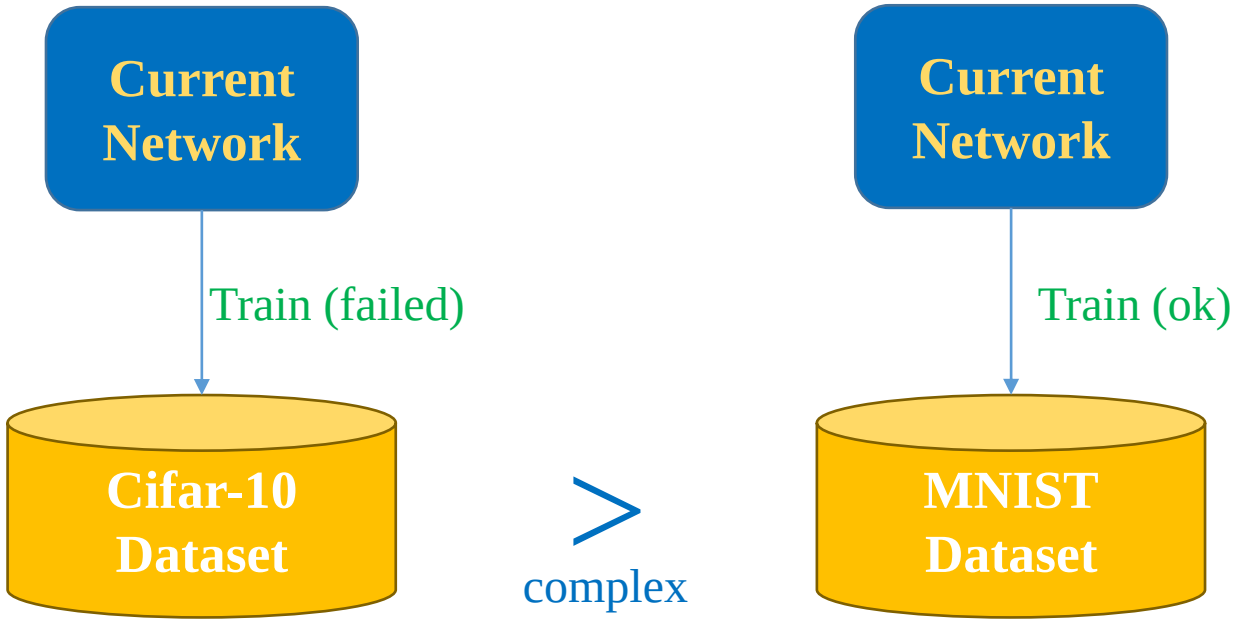
Training
(Adam and cross-entropy loss)

The current network performs excellently for MNIST dataset



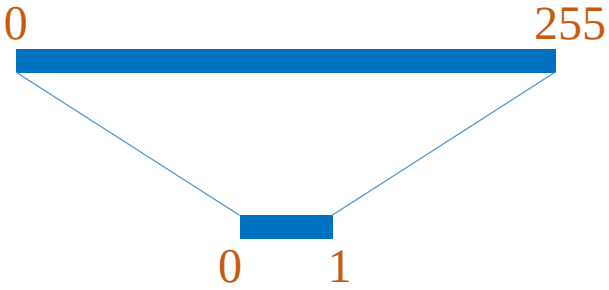
Network Training

❖ Solution 1: Idea



How to reduce the complexity of the Cifar-10 dataset

Data Normalization (scale to [0,1])



Data Normalization (convert to 0-mean and 1-deviation)

$$X = \frac{X - \mu}{\sigma}$$

$$\mu = \frac{1}{n}\sum_i X_i$$

$$\sigma = \sqrt{\frac{1}{n}\sum_i (X_i - \mu)^2}$$

$X =$ 

Network Training

❖ Solution 1: Idea

$$\bar{X} = \frac{X - \mu}{\sigma}$$

$$\mu = \frac{1}{n}\sum_i X_i$$

$$\sigma = \sqrt{\frac{1}{n}\sum_i (X_i - \mu)^2}$$

This normalization helps network to be invariant to linear transformation

$$Y = aX + b$$

$$\bar{Y} = \frac{Y - \mu_Y}{\sigma_Y} = \bar{X}$$

$$Y = aX + b$$

$$\bar{Y} = \frac{Y - \mu_Y}{\sigma_Y} = \frac{(aX + b) - \frac{1}{n}\sum_i (aX_i + b)}{\sqrt{\frac{1}{n}\sum_i \left((aX_i + b) - \frac{1}{n}\sum_i (aX_i + b)\right)^2}}$$

$$= \frac{aX - \frac{1}{n}\sum_i aX_i}{\sqrt{\frac{1}{n}\sum_i \left(aX_i - \frac{1}{n}\sum_j aX_j\right)^2}}$$

$$= \frac{X - \frac{1}{n}\sum_i X_i}{\sqrt{\frac{1}{n}\sum_i \left(X_i - \frac{1}{n}\sum_j X_j\right)^2}} = \frac{X - \mu_X}{\sqrt{\frac{1}{n}\sum_i (X_i - \mu_X)^2}} = \bar{X}$$

Network Training

Solution 1: 0-mean and unit-deviation normalization

Data Normalization (convert to 0-mean and 1-deviation)

$$X = \frac{X - \mu_d}{\sigma_d}$$

μ_d is the mean of dataset

σ_d is the deviation for the whole dataset

```python
# Load dataset with only the ToTensor transform
compute_transform = transforms.Compose([transforms.ToTensor()])
dataset = torchvision.datasets.CIFAR10(root='data', train=True,
                                       transform=compute_transform,
                                       download=True)
loader = torch.utils.data.DataLoader(dataset, batch_size=1024,
                                     shuffle=False, num_workers=4)

mean = 0.0
for images, _ in loader:
    batch_samples = images.size(0)  # Batch size
    images = images.view(batch_samples, images.size(1), -1)
    mean += images.mean(2).sum(0)
mean = mean / len(loader.dataset)

variance = 0.0
for images, _ in loader:
    batch_samples = images.size(0)
    images = images.view(batch_samples, images.size(1), -1)
    variance += ((images - mean.unsqueeze(1))**2).sum([0,2])
std = torch.sqrt(variance / (len(loader.dataset)*32*32))
```

```python
# Data
transform = Compose([ToTensor(),
                     Normalize(mean, std)])
train_set = CIFAR10(root='data', train=True,
                    download=True, transform=transform)
trainloader = DataLoader(train_set, batch_size=256,
                         shuffle=True, num_workers=4)
```

Network Training

❖ Solution 1: 0-mean and unit-deviation normalization

Data Normalization (convert to 0-mean and 1-deviation)

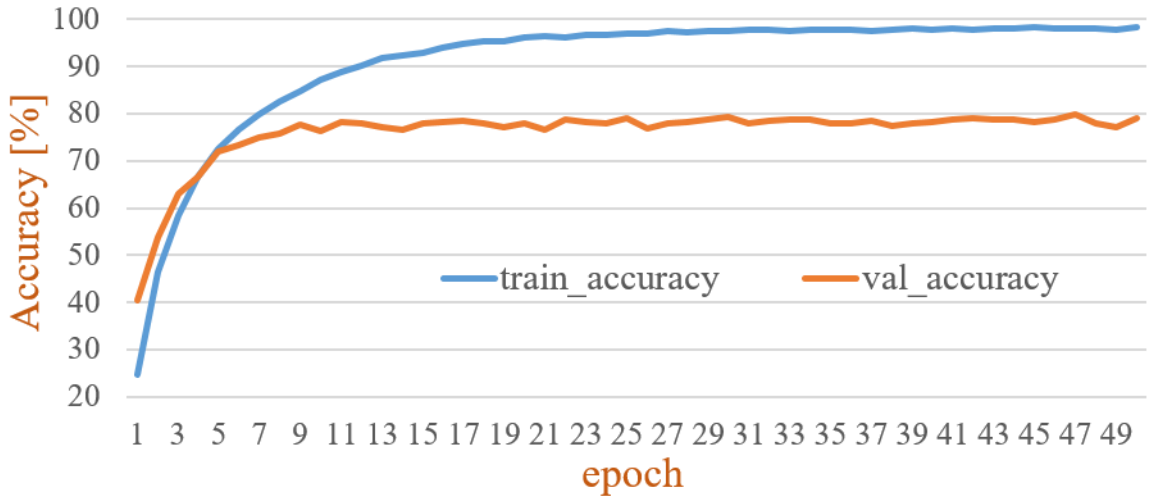
Normalize each channel separately

```
transform = Compose([ToTensor(),
                     Normalize([0.4914, 0.4822, 0.4465],
                               [0.2470, 0.2435, 0.2616])])
train_set = CIFAR10(root='data', train=True,
                    download=True, transform=transform)
```

$$X = \frac{X - \mu_d}{\sigma_d}$$

μ_d is the mean of dataset

σ_d is the deviation for the whole dataset



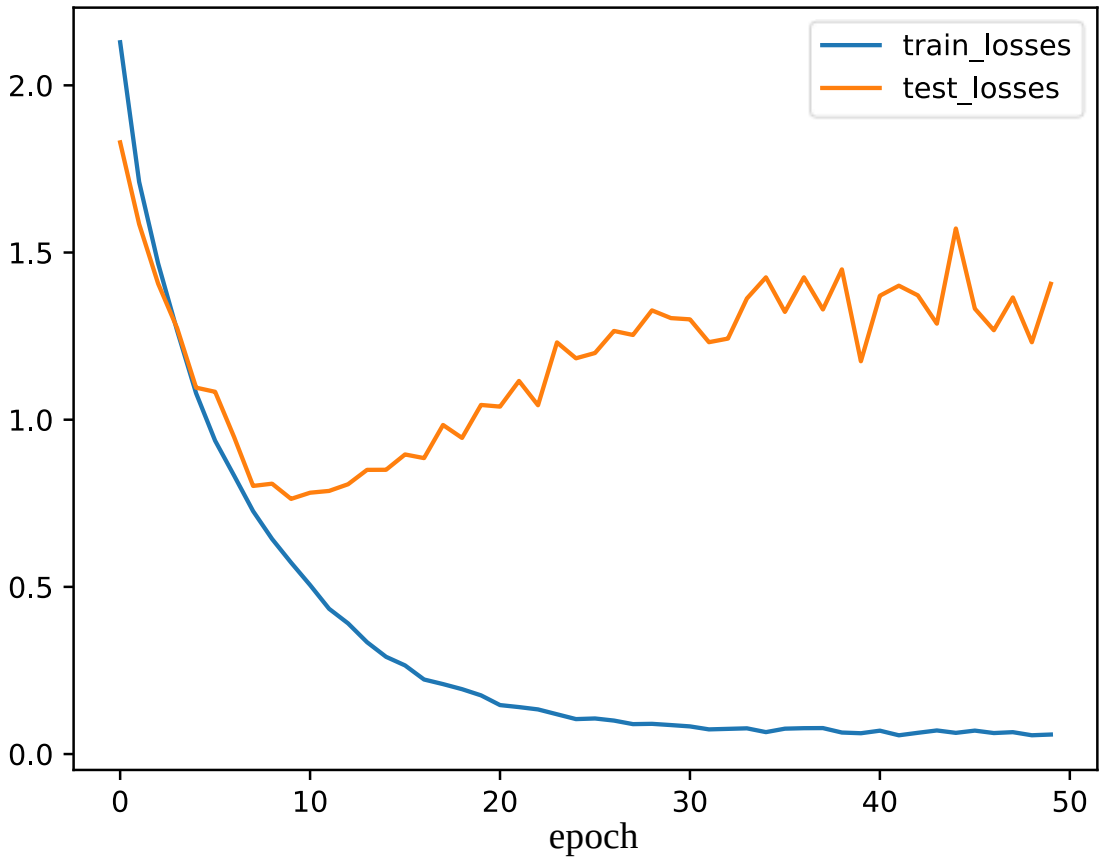
Network Training

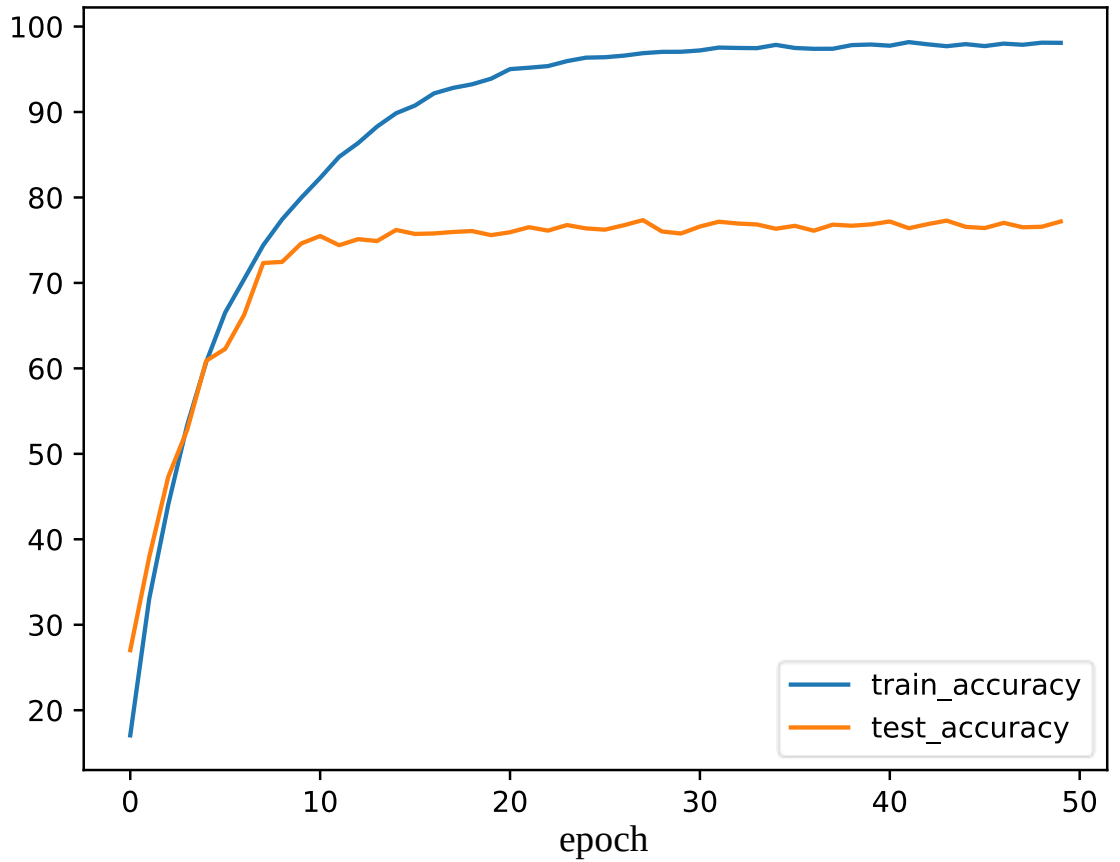
❖ **Solution 1 (extension):**

Normalize to [-1, 1]

Normalize each channel separately

```
transform = Compose([ToTensor(),
                     Normalize((0.5, 0.5, 0.5),
                               (0.5, 0.5, 0.5))])
train_set = CIFAR10(root='data', train=True,
                    download=True, transform=transform)
```

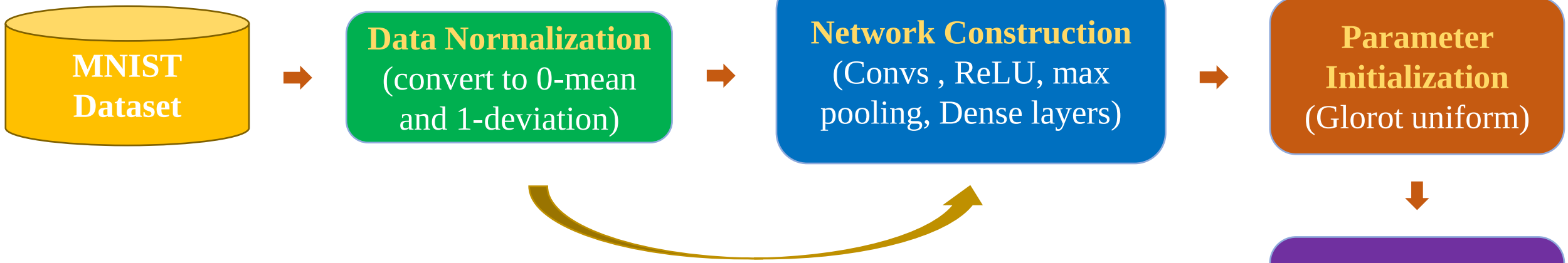




Network Training

❖ Solution 2

$$X = \frac{X - \mu_d}{\sigma_d}$$

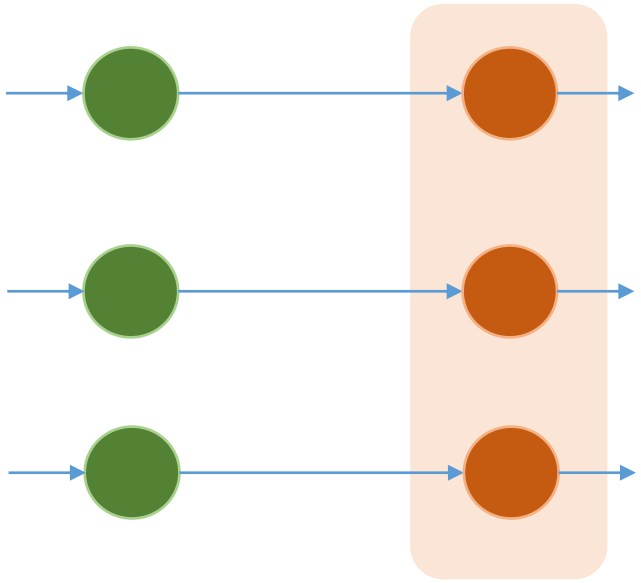


How to use the idea (from solution 1) to integrate to network

Batch Normalization

Network Training

❖ Solution 2: Batch normalization



Batch Normalization

Do not need bias when using BN*

μ and σ are updated in forward pass
γ and β are updated in backward pass

Input data for a node in batch normalization layer

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

m is mini-batch size

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

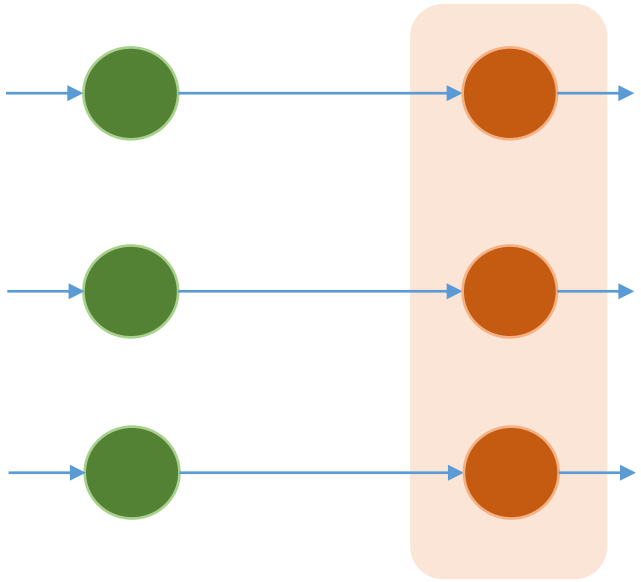
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Solution 2: Batch normalization



Batch Normalization

What if

$$\gamma = \sqrt{\sigma^2 + \epsilon} \text{ and } \beta = \mu$$

Input data for a node in batch normalization layer

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

m is mini-batch size

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

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

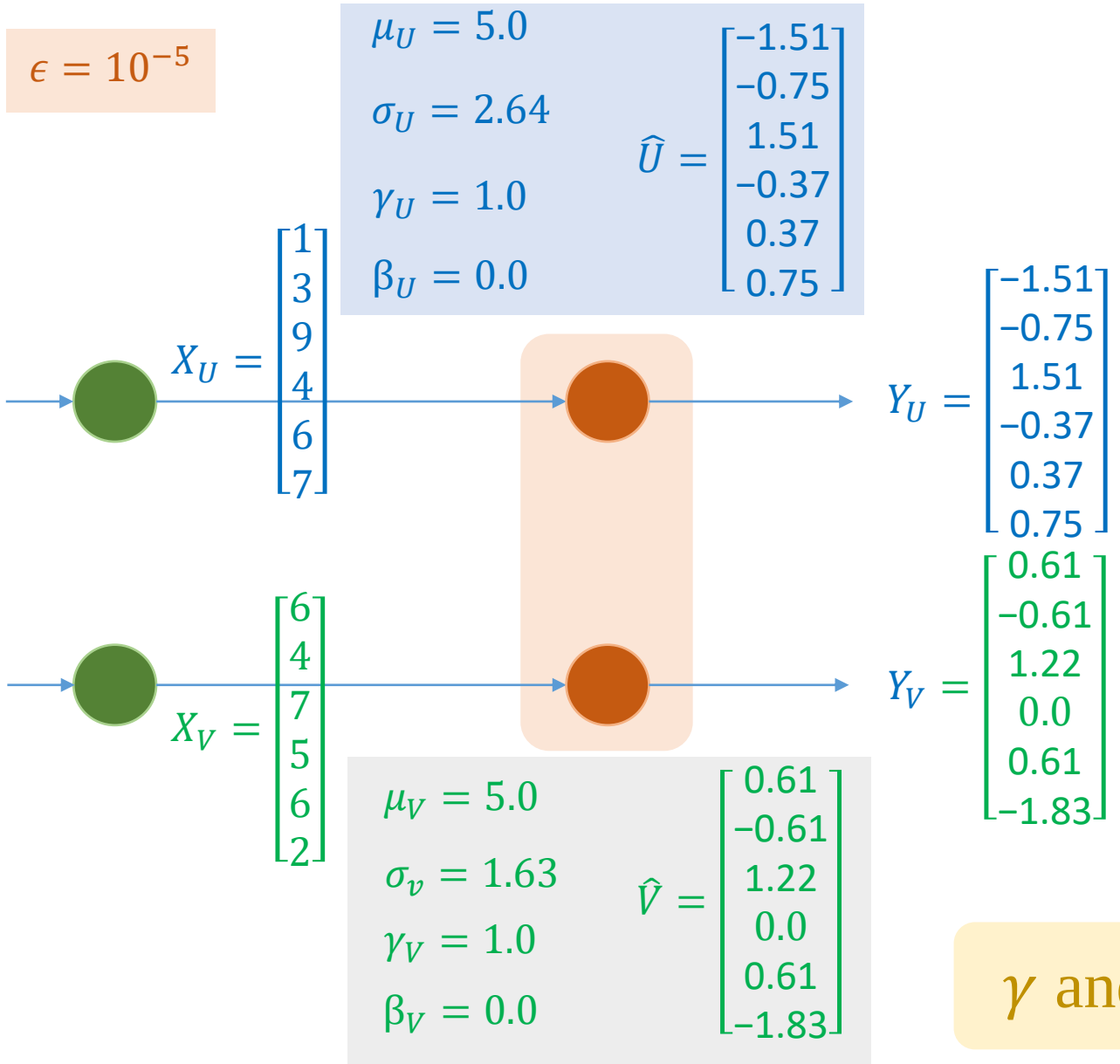
ϵ is a very small value

Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training



Solution 2: Batch normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

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

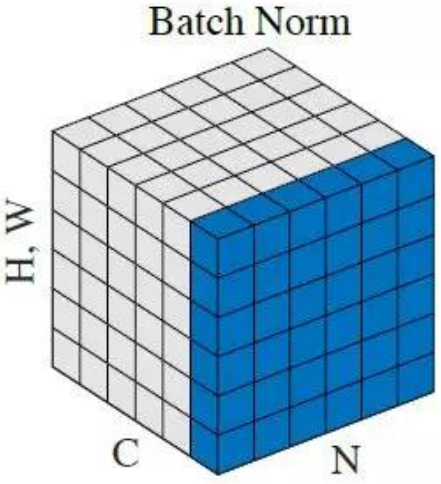
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

γ and β are updated in training process

Batch Normalization



https://arxiv.org/pdf/1803.08494.pdf

$\epsilon = 10^{-5}$

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

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

$\mu = 2.5$

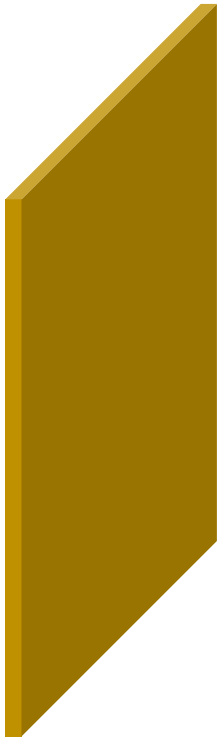
$\sigma^2 = 6.58$

$\gamma = 1.0$

$\beta = 0.0$

sample 1 sample 2 sample 3

$$X = \left\{ \begin{bmatrix} 7 & 5 \\ 0 & 4 \end{bmatrix}, \begin{bmatrix} 0 & 7 \\ 3 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \right\}$$



$$\hat{X} = \left\{ \begin{bmatrix} 1.75 & 0.97 \\ -0.97 & 0.58 \end{bmatrix}, \begin{bmatrix} -0.97 & 1.75 \\ 0.19 & -0.58 \end{bmatrix}, \begin{bmatrix} -0.19 & -0.97 \\ -0.97 & -0.58 \end{bmatrix} \right\}$$

$$\hat{Y} = \left\{ \begin{bmatrix} 1.75 & 0.97 \\ -0.97 & 0.58 \end{bmatrix}, \begin{bmatrix} -0.97 & 1.75 \\ 0.19 & -0.58 \end{bmatrix}, \begin{bmatrix} -0.19 & -0.97 \\ -0.97 & -0.58 \end{bmatrix} \right\}$$

batch-size = 3

input_shape = (BS=3, C=1, H=2, W=2)

Network Training

$\epsilon = 10^{-5}$

$\mu = [2.0, 3.0]$

$\sigma^2 = [6.0, 8.67]$

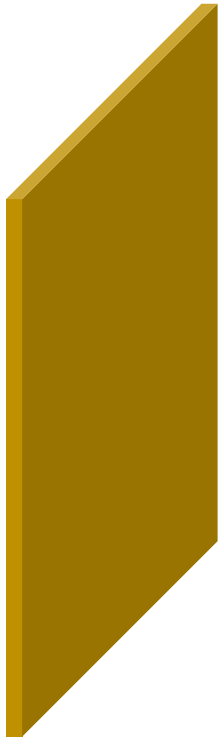
$\gamma = 1.0$

$\beta = 0.0$

$$\begin{bmatrix} 0 & 5 \\ 3 & 0 \end{bmatrix} \qquad \begin{bmatrix} 7 & 2 \\ 0 & 3 \end{bmatrix}$$

sample 1

$X = \{$ $\}$



$$\hat{X} = \left\{ \begin{bmatrix} -0.94 & 1.41 \\ 0.47 & -0.94 \end{bmatrix} \begin{bmatrix} 1.56 & -0.39 \\ -1.17 & 0 \end{bmatrix} \right\}$$

$$\hat{Y} = \left\{ \begin{bmatrix} -0.94 & 1.41 \\ 0.47 & -0.94 \end{bmatrix} \begin{bmatrix} 1.56 & -0.39 \\ -1.17 & 0 \end{bmatrix} \right\}$$

batch-size = 1

sample_shape = (BS=1, C=2, H=2, W=2)

Network Training

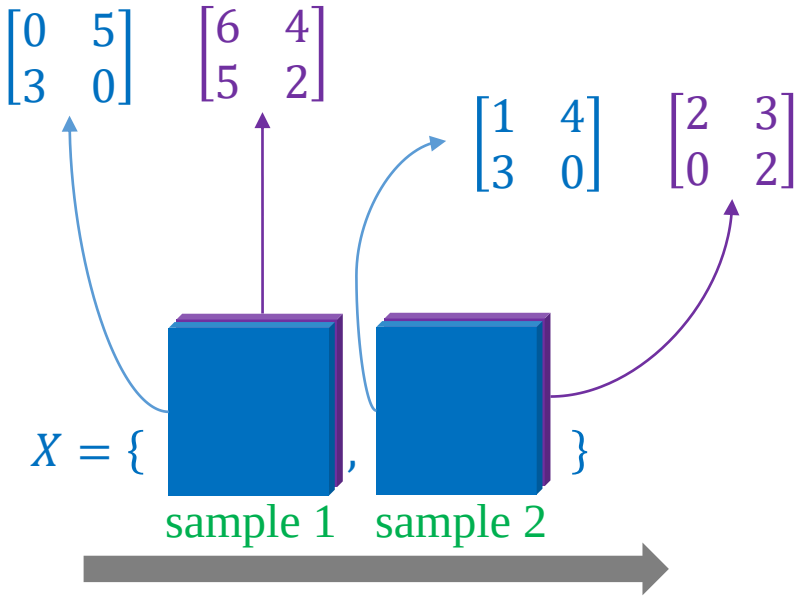
$\epsilon = 10^{-5}$

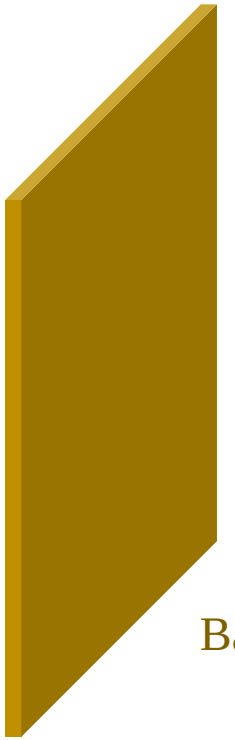
$\mu = [2.0, 3.0]$

$\sigma^2 = [4.0, 3.7]$

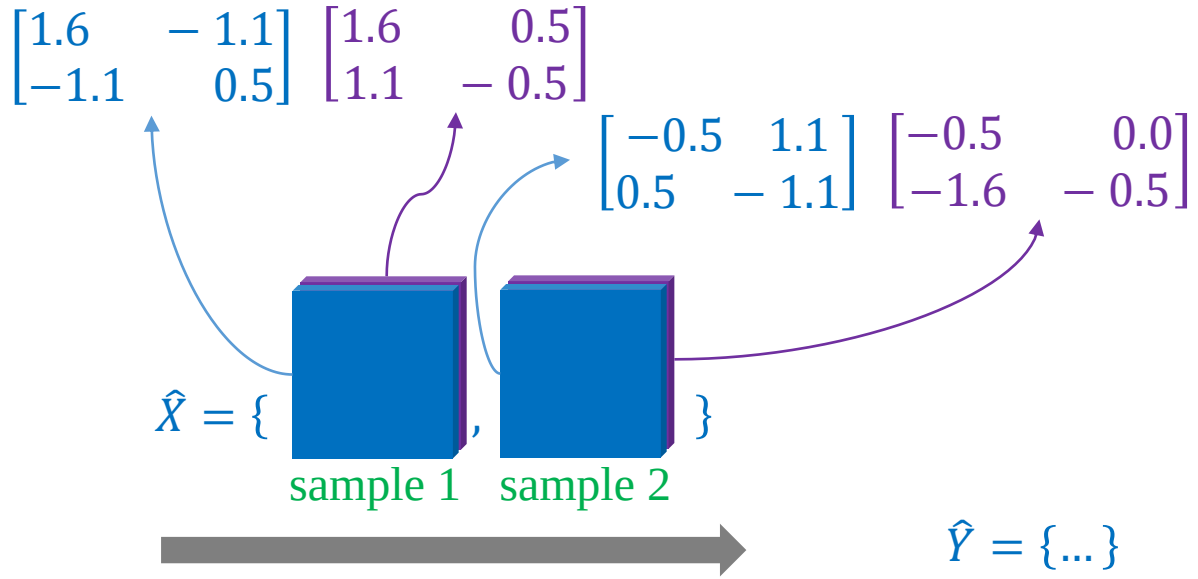
$\gamma = 1.0$

$\beta = 0.0$





Batch-Norm Layer

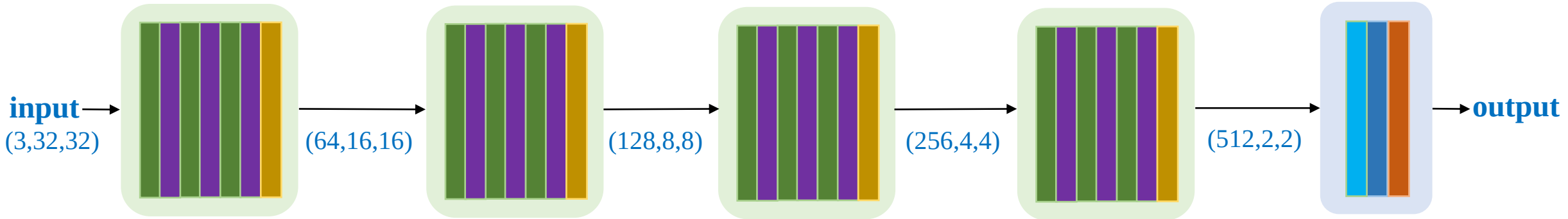


batch-size = 2

sample_shape = (BS=2 , C=2, H=2, W=2)

Network Training

❖ Solution 2: Batch normalization



(3x3) Convolution padding='same' stride=1 + ReLU

Flatten

Dense Layer-512 + ReLU

(2x2) max pooling

Batch normalization

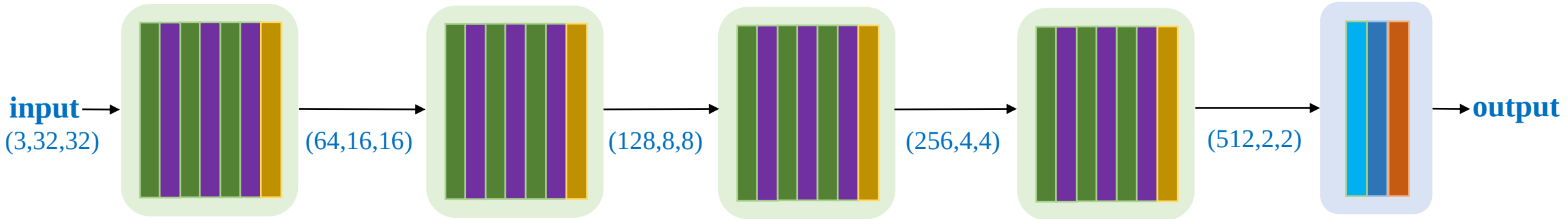
Dense Layer-10 + Softmax

```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3,
                                      stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64))
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3,
                                      stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64))
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3,
                                      stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64),
                            nn.MaxPool2d(2, 2))
```

torch.nn.BatchNorm2d(num_features)
num_features (int): C from an expected input of size (N, C, H, W)

Network Training

❖ Solution 2: Batch normalization



(3x3) Convolution padding='same' stride=1 + ReLU

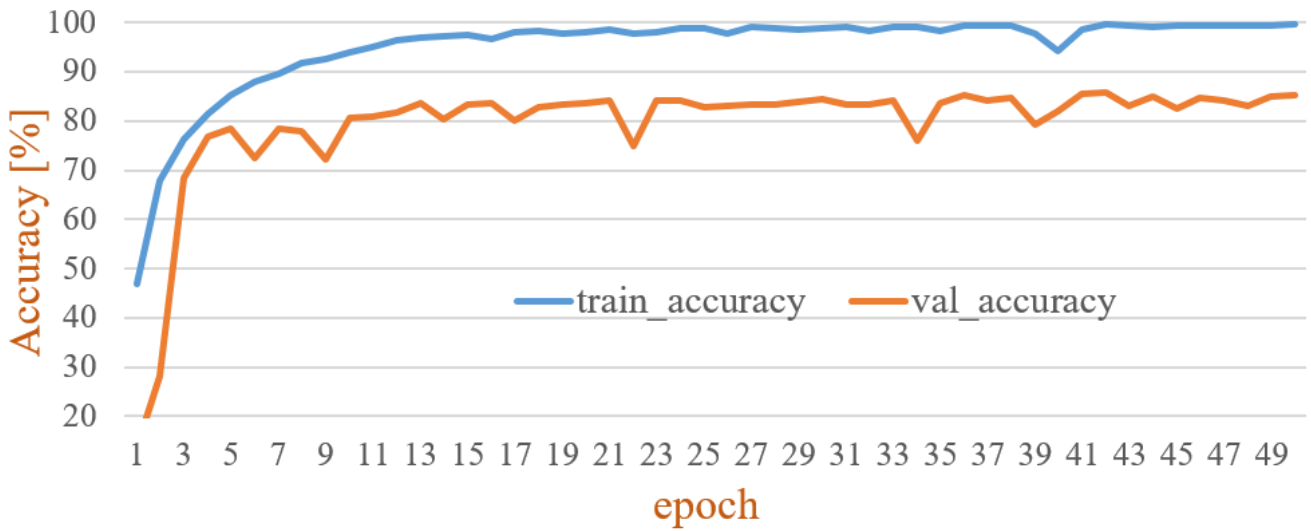
Flatten

Dense Layer-512 + ReLU

(2x2) max pooling

Batch normalization

Dense Layer-10 + Softmax




```
conv = nn.Sequential(nn.Conv2d(3, 64, 3),
                     nn.ReLU(),
                     nn.BatchNorm2d(64))
```

Network Training

❖ Solution 2: Batch normalization

Speed up training

Reduce the dependence on initial weights

Model Generalization

Input data for a node in batch normalization layer

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

m is mini-batch size

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

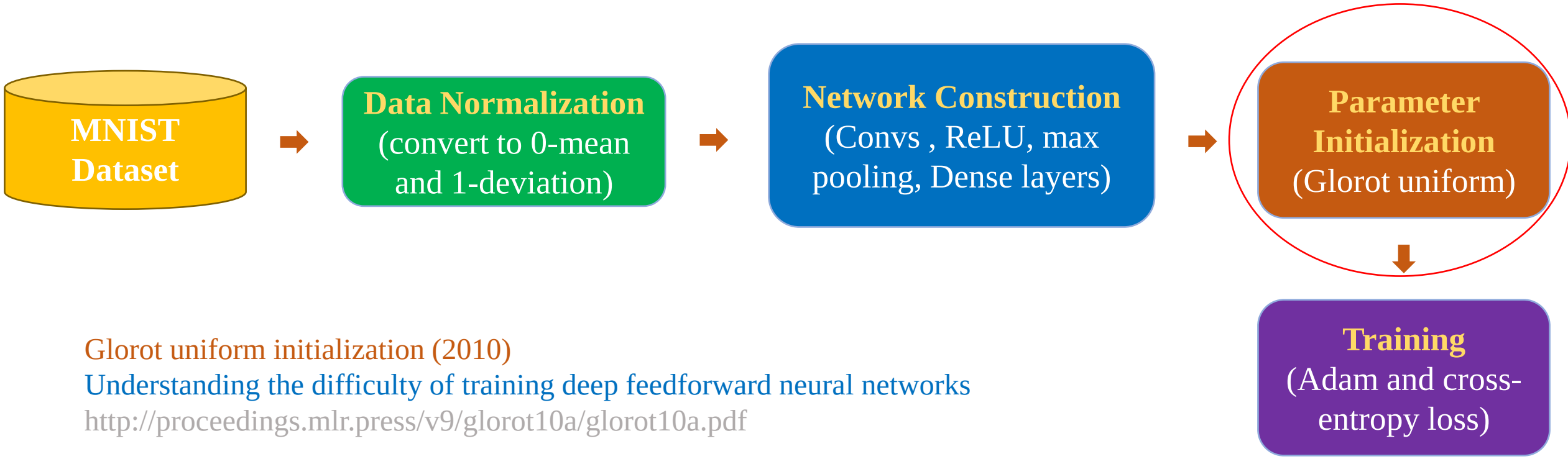
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Solution 3: Use more robust initialization



Glorot uniform initialization (2010)
Understanding the difficulty of training deep feedforward neural networks
http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

He initialization (2015)
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
https://arxiv.org/pdf/1502.01852.pdf

Network Training

❖ Solution 3: He Initialization

Glorot initialization (2010)

$$W \sim \mathcal{N}\left(0, \frac{1}{n_j}\right)$$

n_j is #inputs in layer j

Assuming activation functions are linear

He initialization (2015)

Taking activation function into account

Adapt to ReLU activation

$$W \sim \mathcal{N}\left(0, \frac{2}{n_j}\right)$$

```
def initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            init.kaiming_normal_(m.weight,
                                 nonlinearity='relu')
            if m.bias is not None:
                init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            init.kaiming_normal_(m.weight,
                                 nonlinearity='relu')
            if m.bias is not None:
                init.zeros_(m.bias)
```

Data normalization [0,1]

He normal initialization

Adam optimizer with lr=1e-3

Network Training

❖ Solution 3: He Initialization

Glorot initialization (2010)

$$W \sim \mathcal{N}\left(0, \frac{1}{n_j}\right)$$

n_j is #inputs in layer j

Assuming activation functions are linear

He initialization (2015)

Taking activation function into account

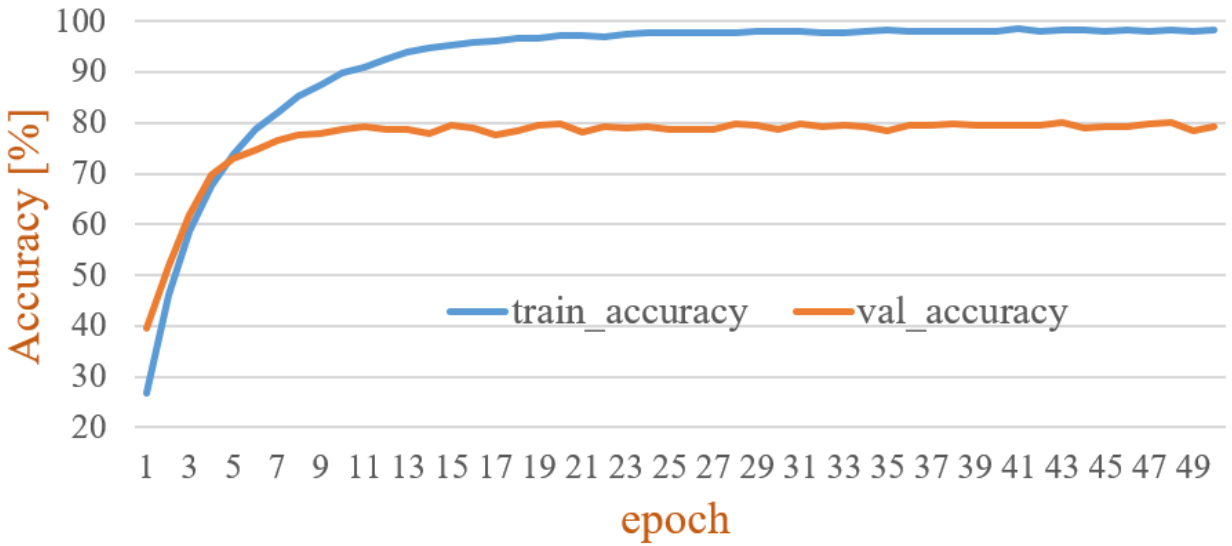
Adapt to ReLU activation

$$W \sim \mathcal{N}\left(0, \frac{2}{n_j}\right)$$

Data normalization [0,1]

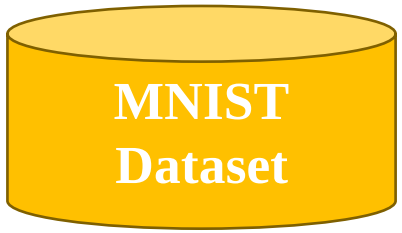
He normal initialization

Adam optimizer with lr=1e-3



Network Training

❖ Solution 4: Using advanced activation



Data Normalization (convert to 0-mean and 1-deviation)

Network Construction (Convs , ~~ReLU~~, max pooling, Dense layers)

Parameter Initialization (Glorot uniform)

Training (Adam and cross-entropy loss)

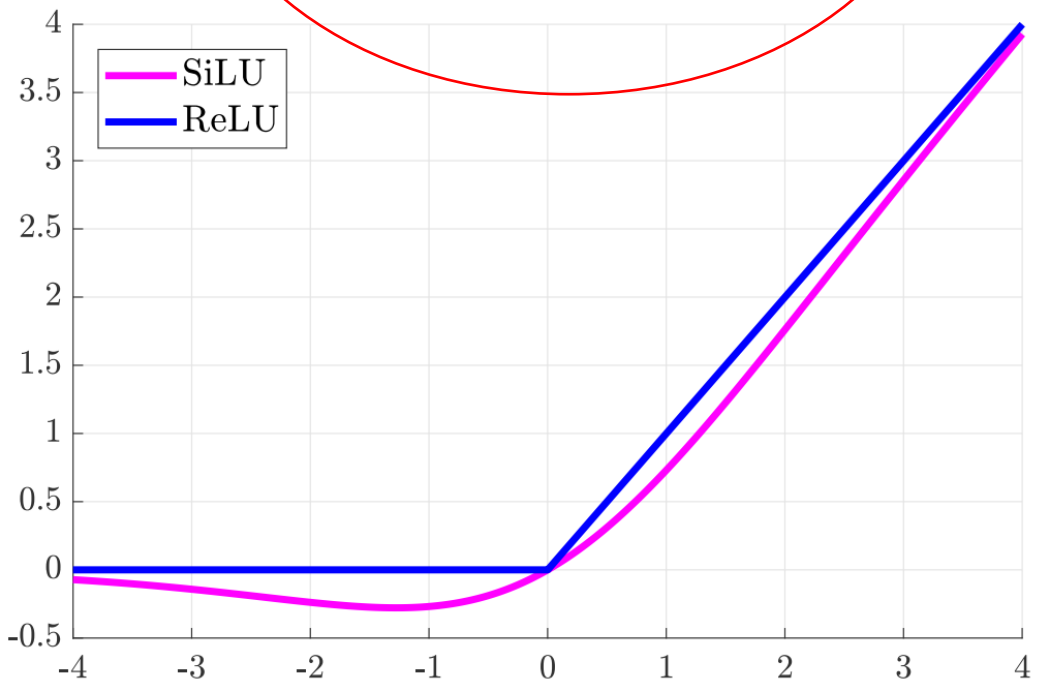
2017 Sigmoid Linear Unit (SiLU)

$$\text{swish}(x) = x * \frac{1}{1 + e^{-x}}$$

2010

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding=1),
    nn.SiLU()
)
```



https://arxiv.org/pdf/1702.03118.pdf

Network Training

2017 Sigmoid Linear Unit (SiLU)

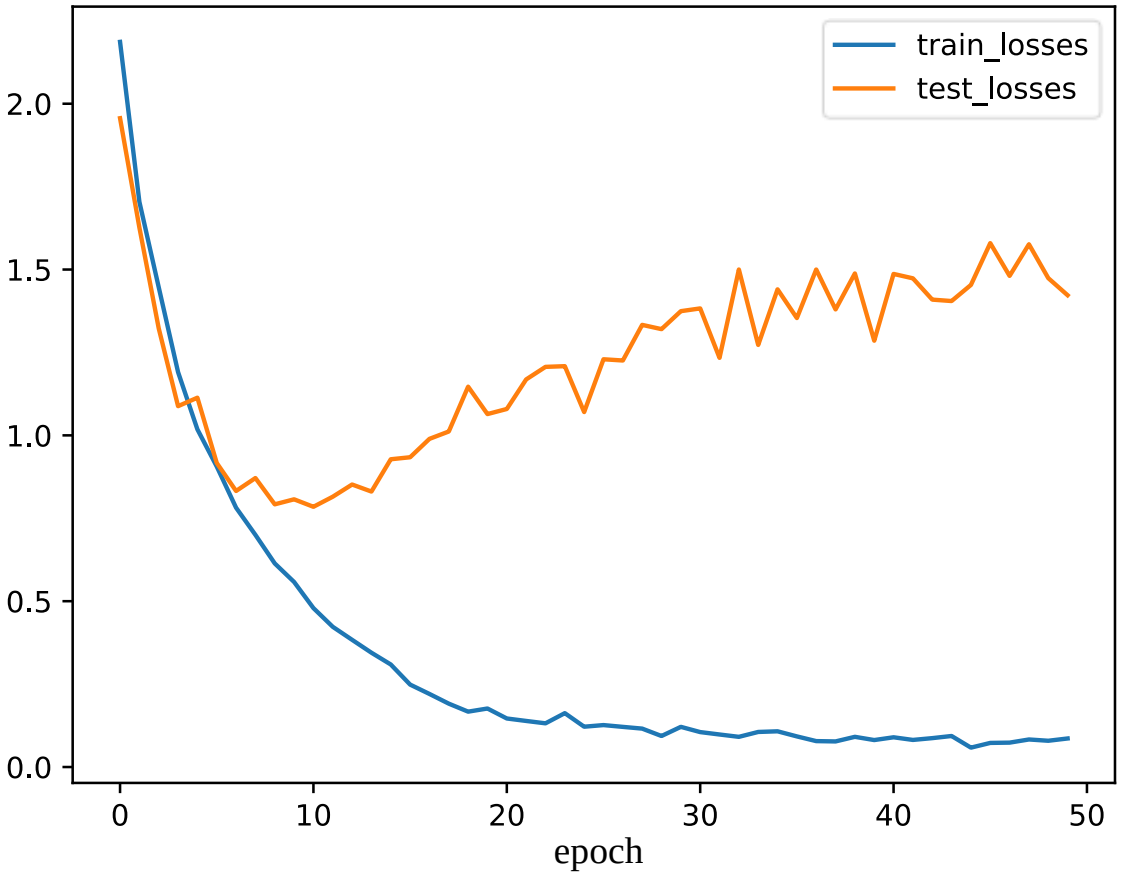
$$\text{swish}(x) = x * \frac{1}{1 + e^{-x}}$$

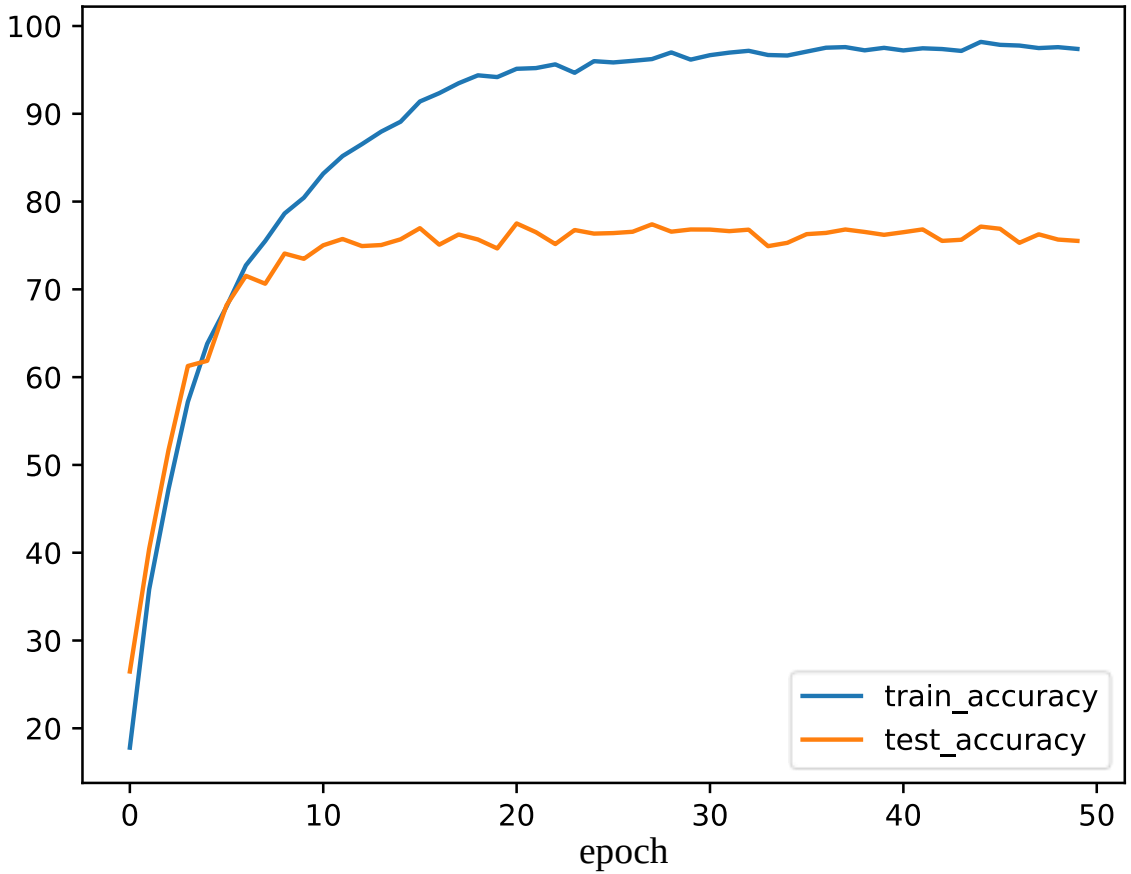
❖ **Solution 4:**

Using advanced activation

```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding=1),
    nn.SiLU()
)
```

https://arxiv.org/pdf/1702.03118.pdf





Network Training

❖ Solution 5: Skip connection

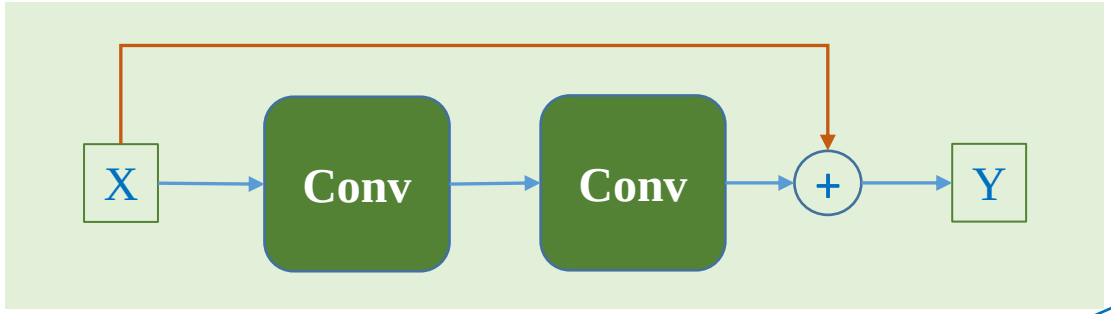
MNIST Dataset

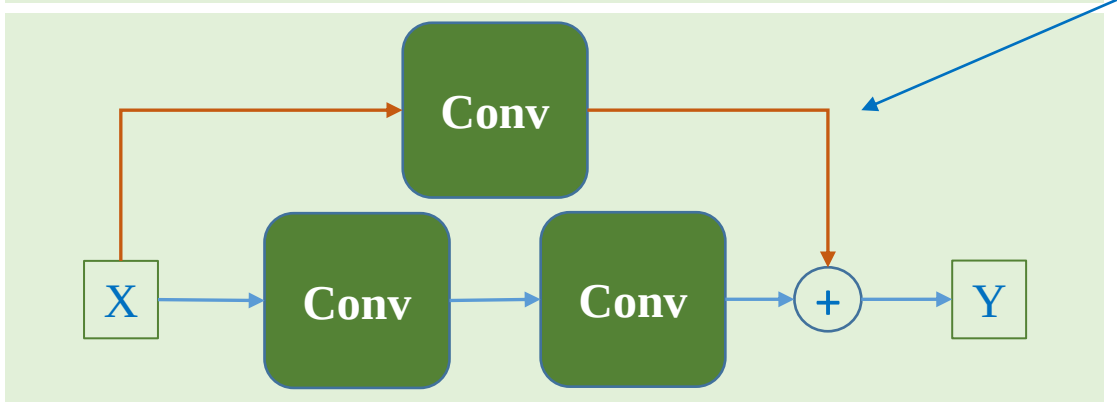
Data Normalization (convert to 0-mean and 1-deviation)

Network Construction (Convs , ReLU, max pooling, Dense layers)

Parameter Initialization (Glorot uniform)

Training (Adam and cross-entropy loss)



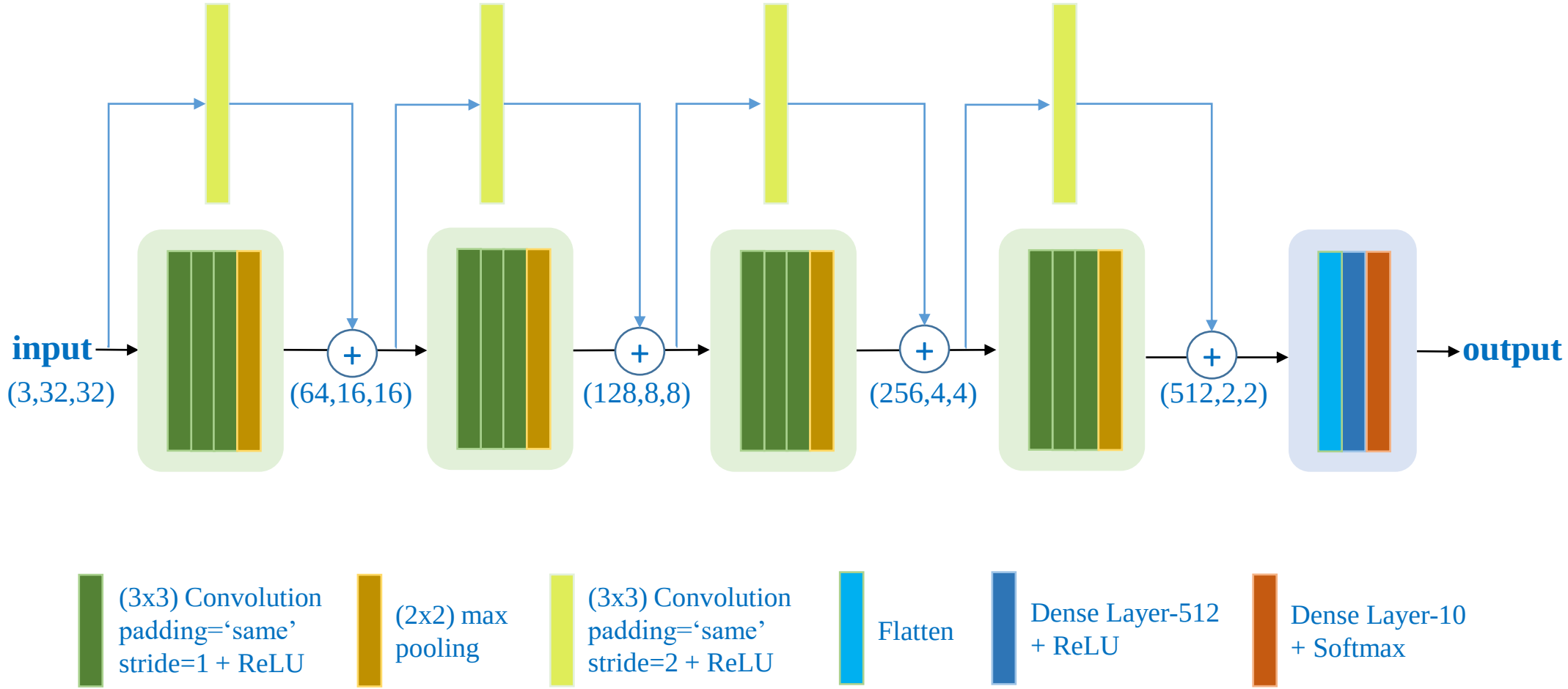


Skip connection

Improve gradient flow in backward pass

Network Training

❖ Solution 5: Skip connection

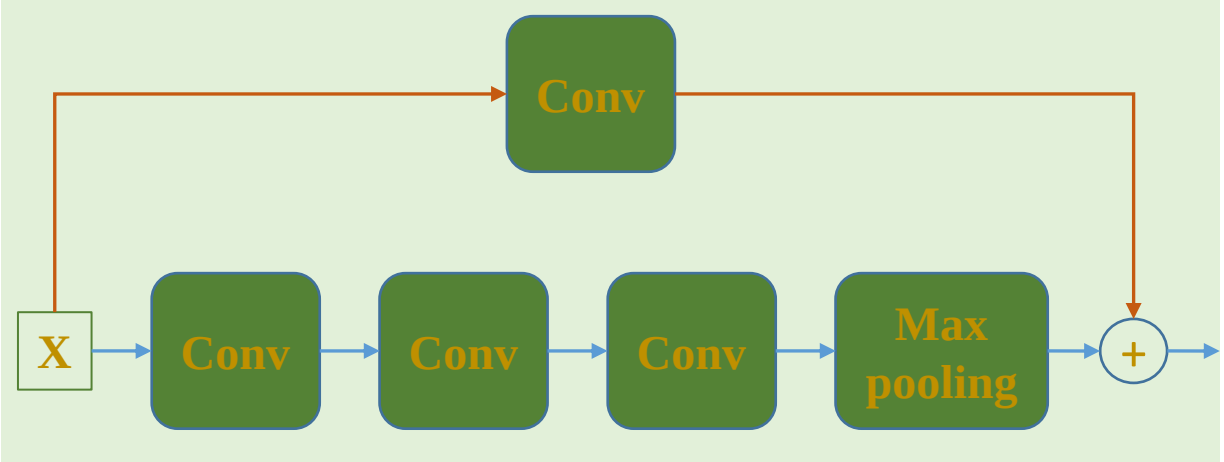


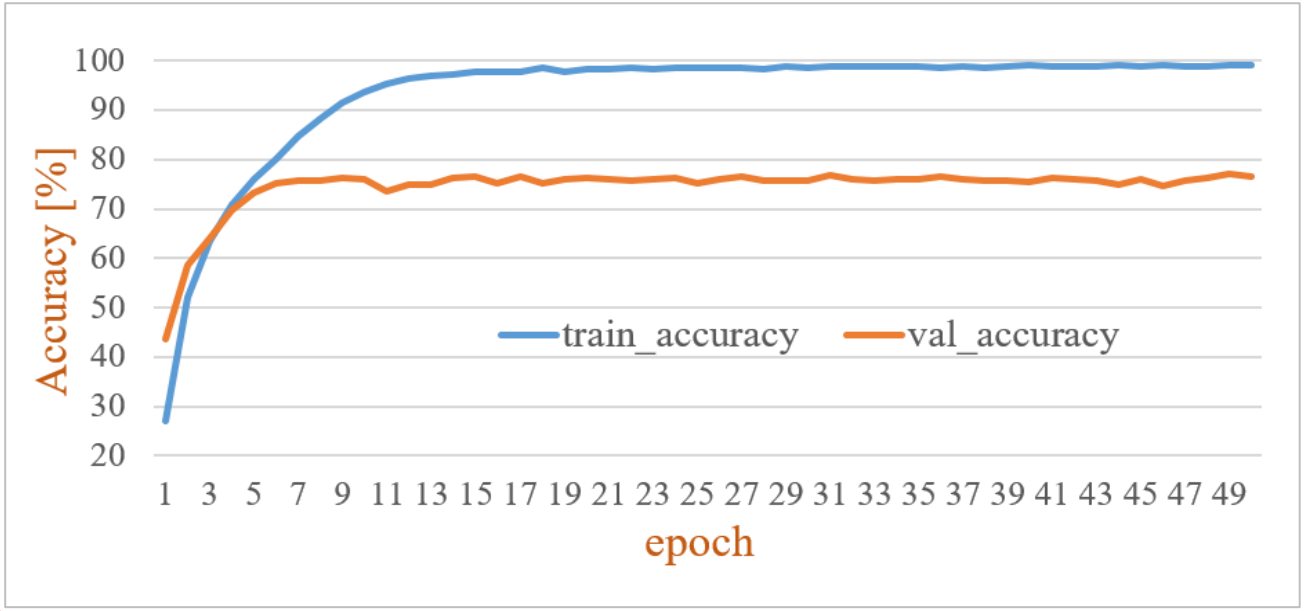
Network Training

❖ **Solution 5:**

Skip connection

```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))
res_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=2, padding=1), nn.ReLU())

# Given x
previous_input_x = x
x = self.conv_layer1(x)
x = self.conv_layer2(x)
x = self.conv_layer3(x)
res = self.res_layer1(previous_input_x)
x = x + res
```

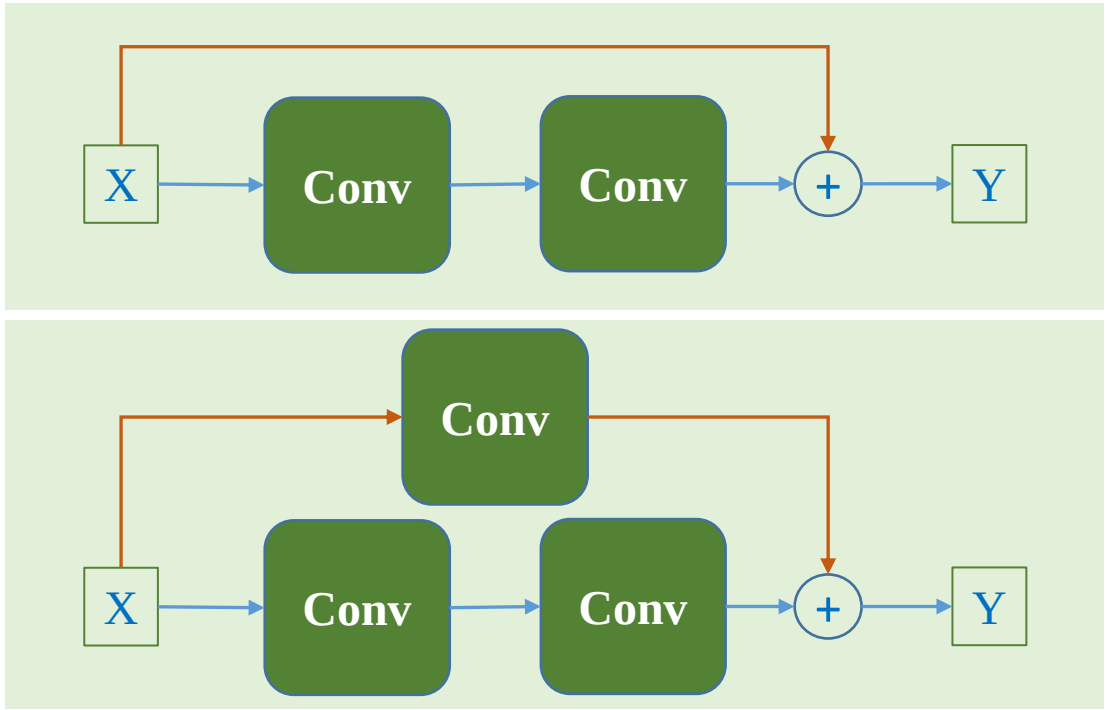


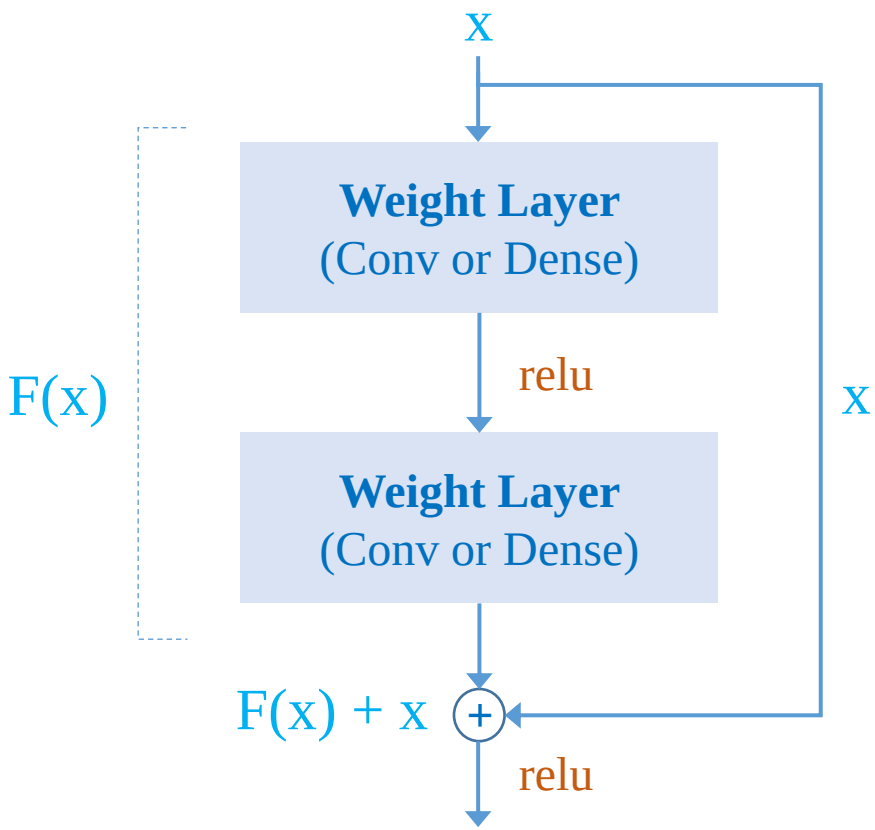


There are several variants that use fully skip connection, concatenation, long skip connection

Network Training

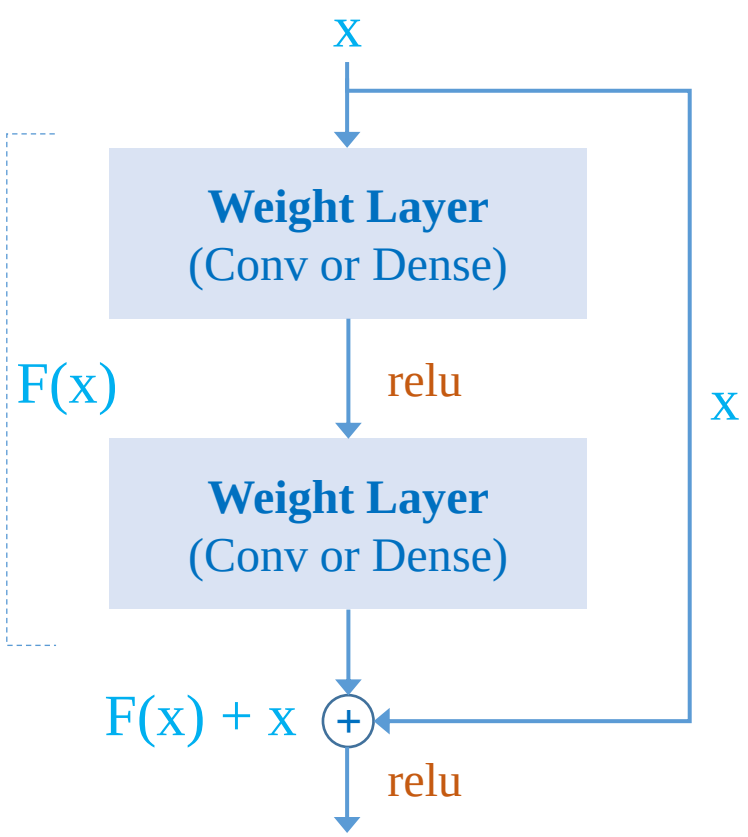
❖ Solution 5: Skip connection

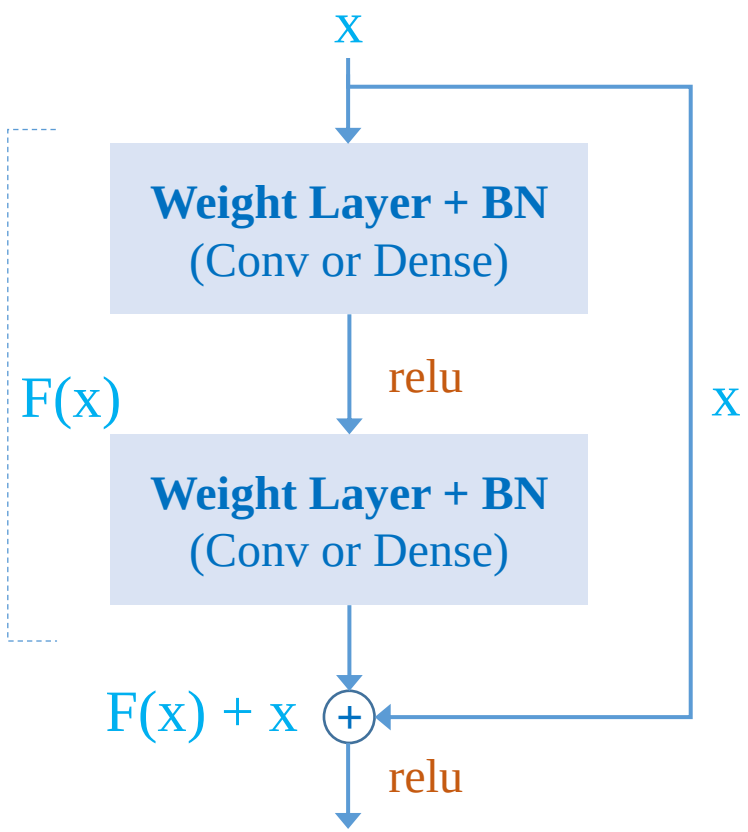


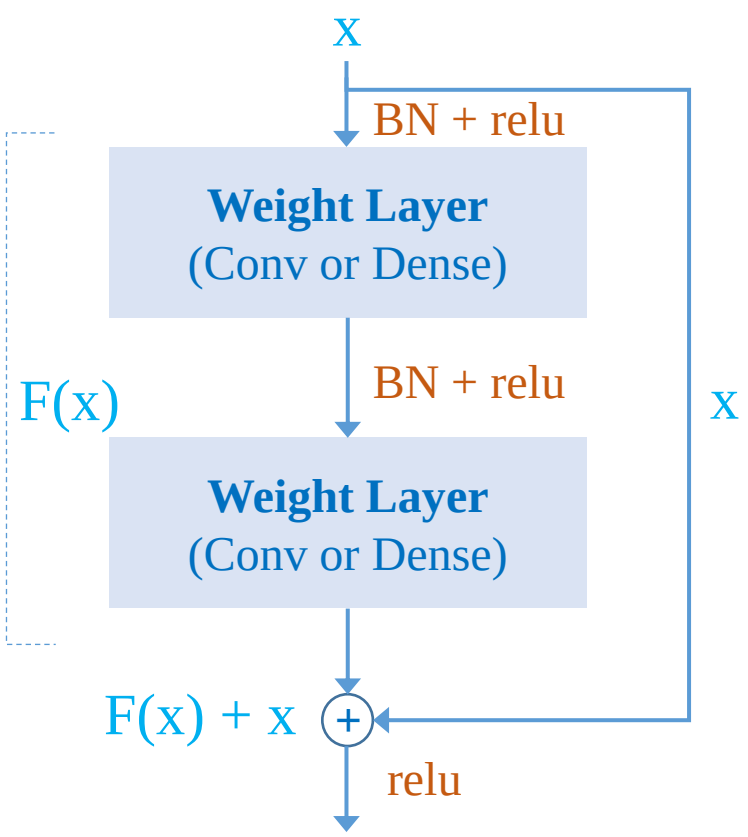


Network Training

❖ Solution 5: Skip connection



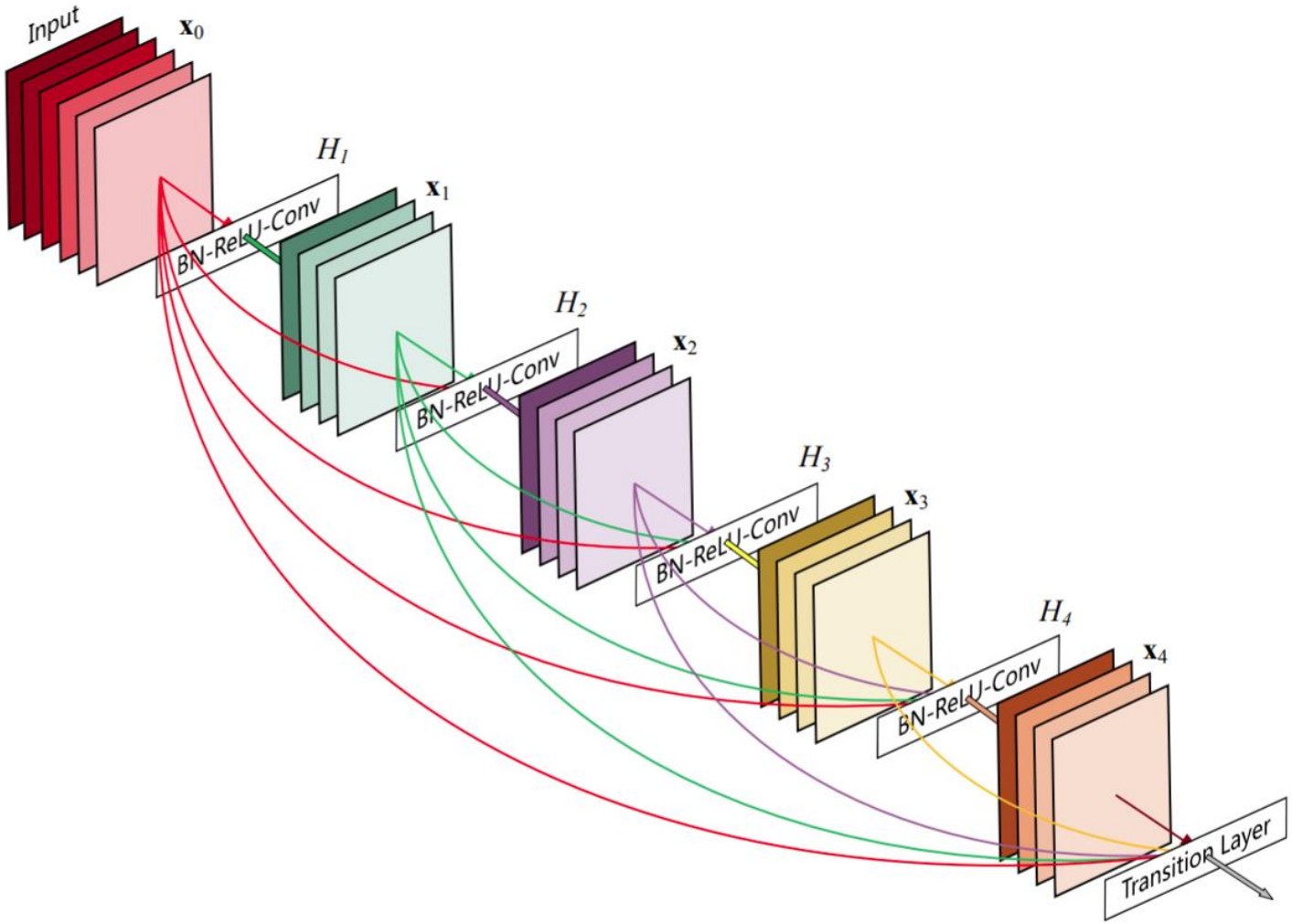


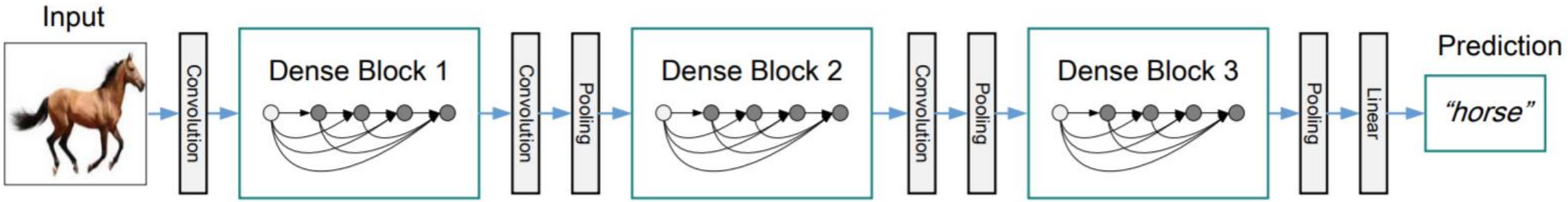


Network Training

❖ Solution 5: Skip connection

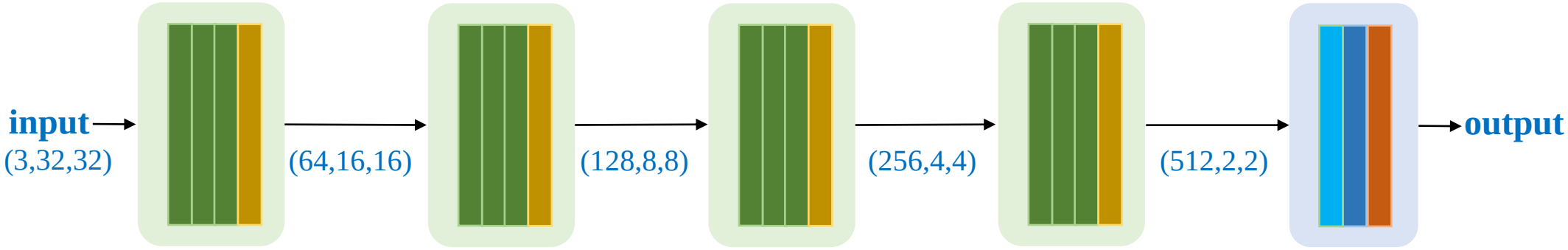
https://arxiv.org/pdf/1608.06993v5.pdf

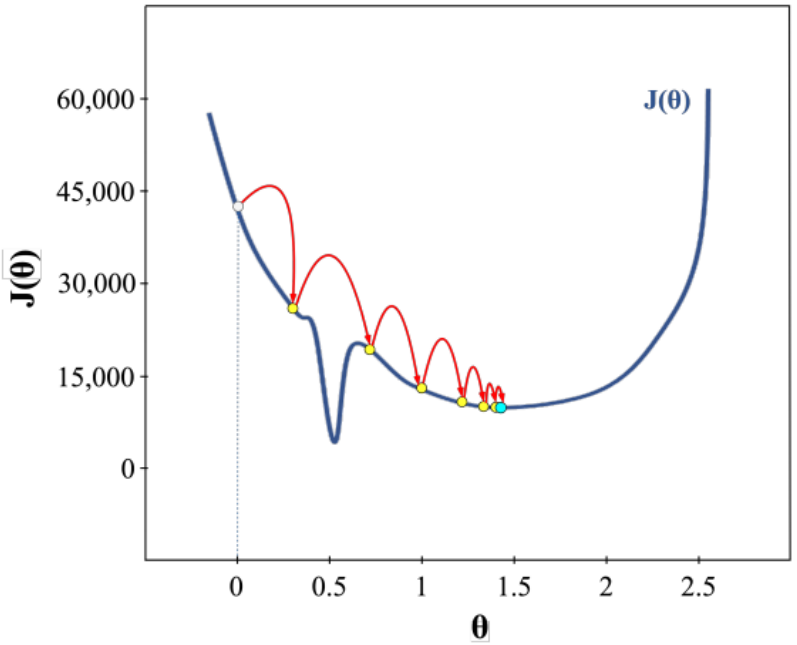




Network Training

❖ Solution 6: Reduce learning rate





(3x3) Convolution padding='same' stride=1 + ReLU

Flatten

Dense Layer-512 + ReLU

(2x2) max pooling

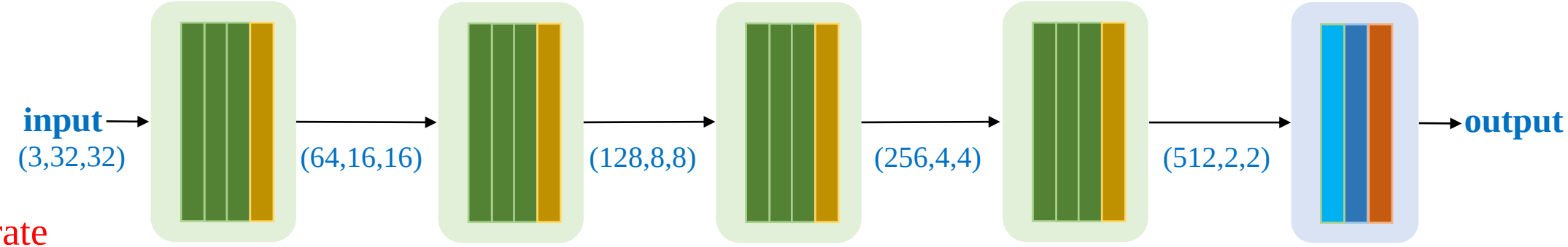
Dense Layer-10 + Softmax

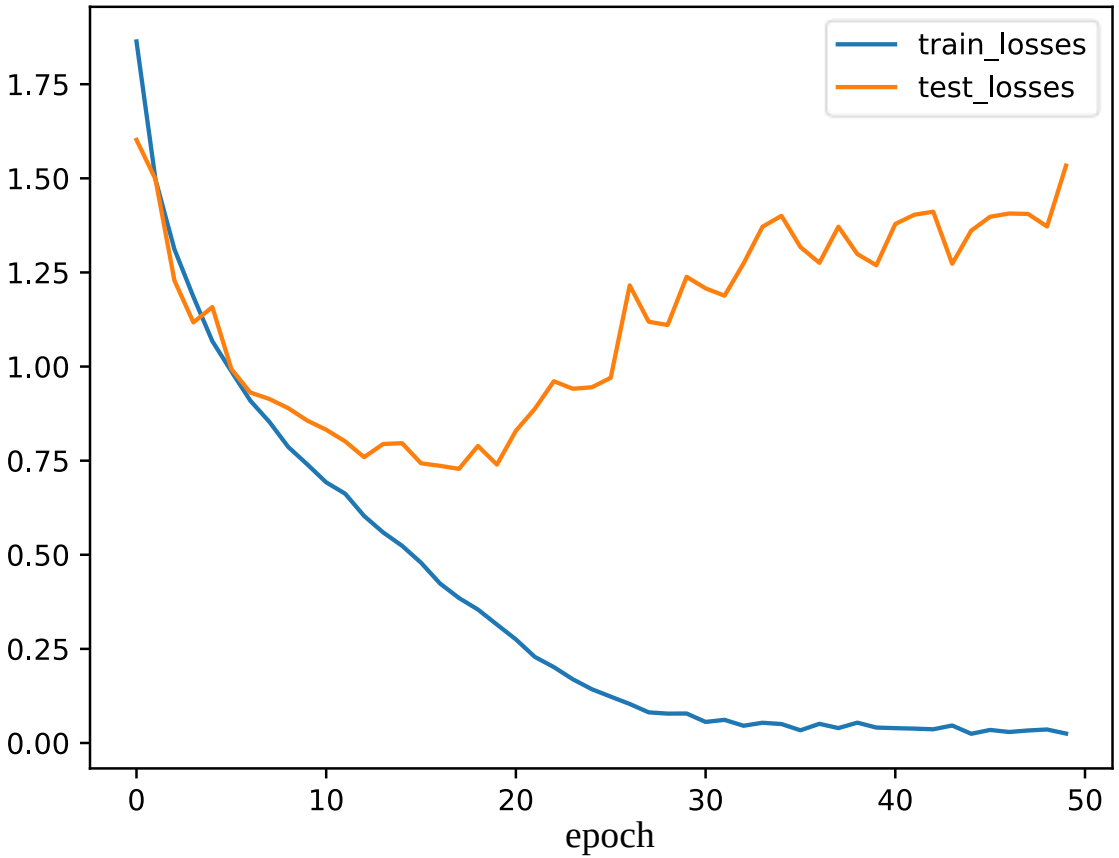
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-4)
```

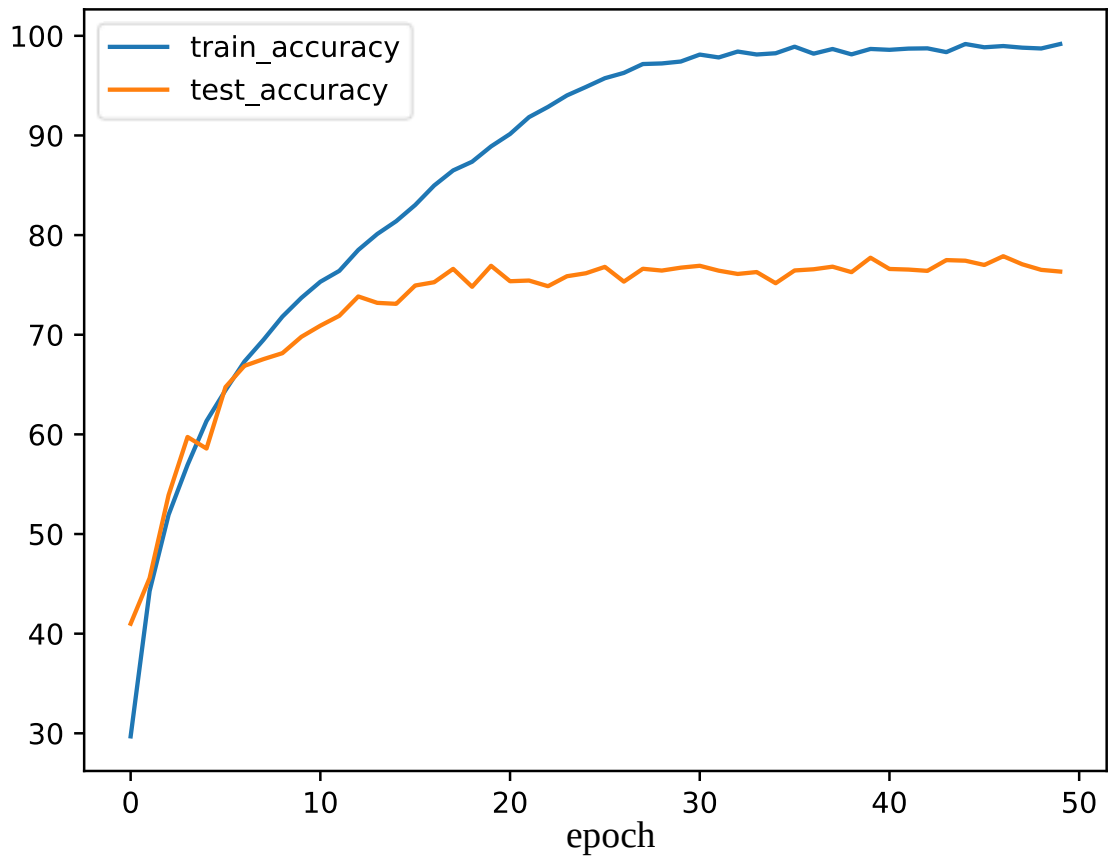
From "Machine Learning Simplified"

Network Training

Reduce learning rate







Further Reading

Skip connection

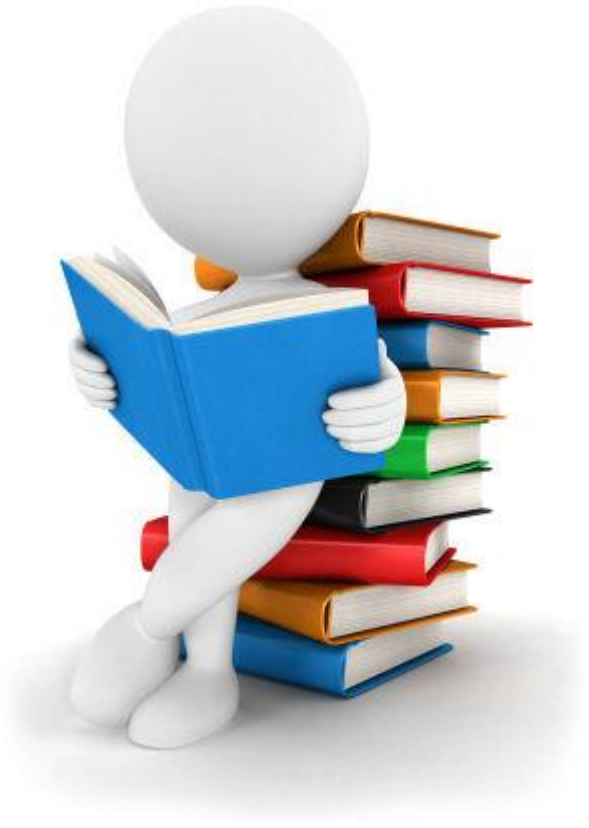
https://theaisummer.com/skip-connections/

Trying to overfit Data

http://karpathy.github.io/2019/04/25/recipe/

DenseNet

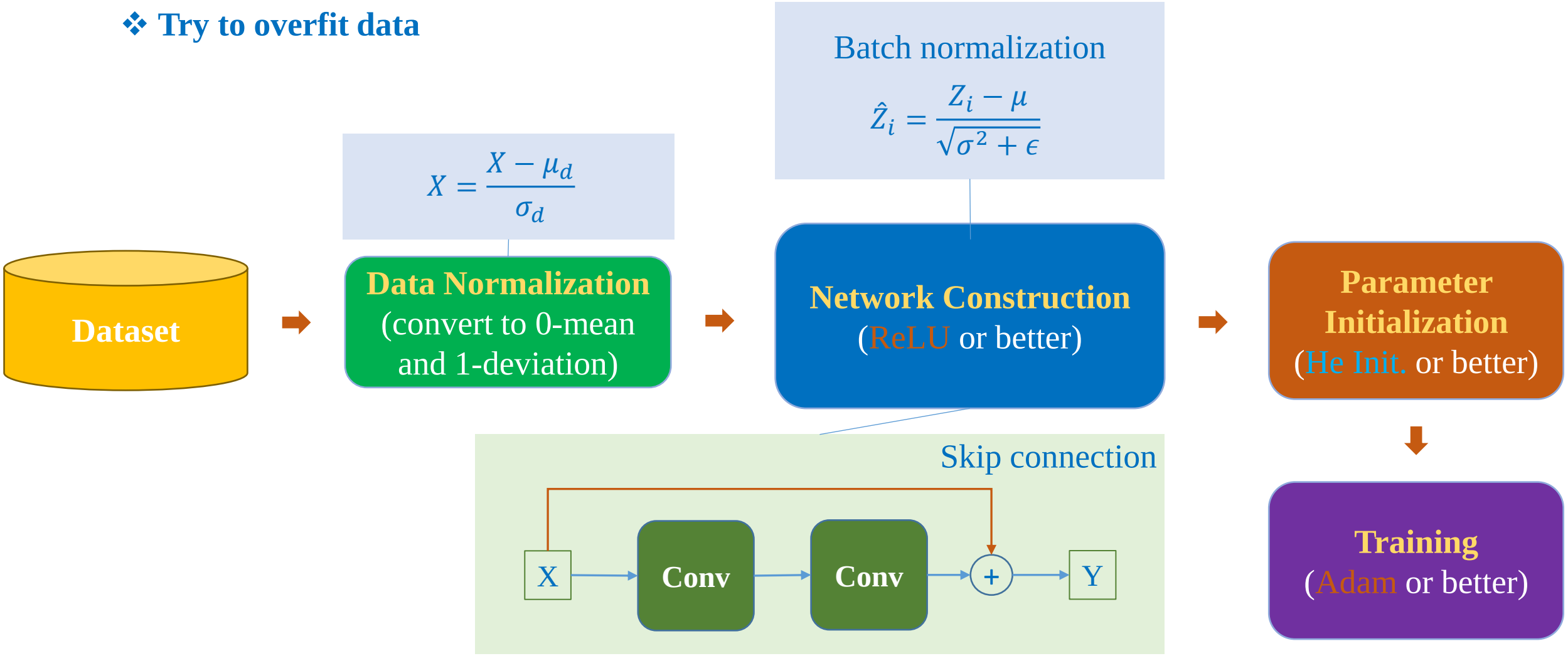
https://arxiv.org/pdf/1608.06993v5.pdf



Summary

❖ Train a CNN model

❖ Try to overfit data



AI VIETNAM
All-in-One Course

Model Generalization

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

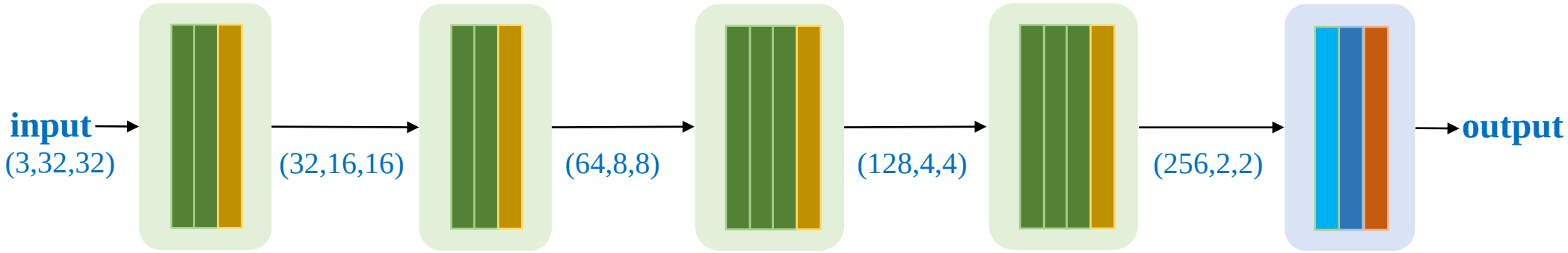
Network Training

Cifar-10 dataset (complex dataset)

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples

Model Generalization

Cifar-10 Dataset



Data Normalization (convert to 0-mean and 1-deviation)

$$\bar{X} = \frac{X - \mu}{\sigma}$$

$$\mu = \frac{1}{n}\sum_i X_i$$

$$\sigma = \sqrt{\frac{1}{n}\sum_i (X_i - \mu)^2}$$

(3x3) Convolution padding='same' stride=1 + ReLU

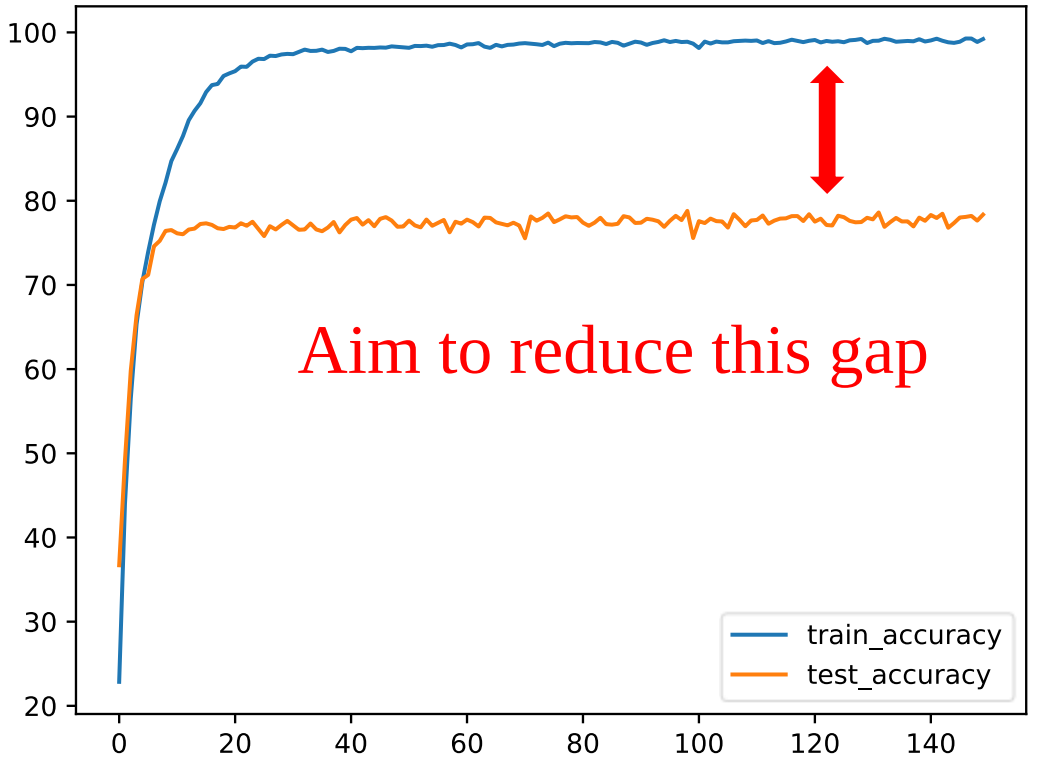
(2x2) max pooling

Flatten

Dense Layer-10 + Softmax

Dense Layer-512 + ReLU

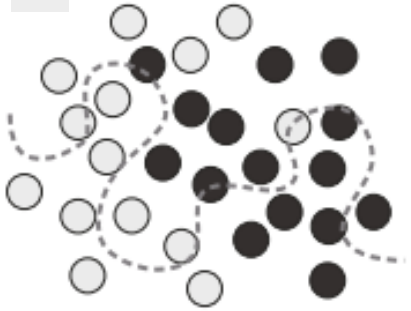
Adam lr=1e-3 ; He Init



Training

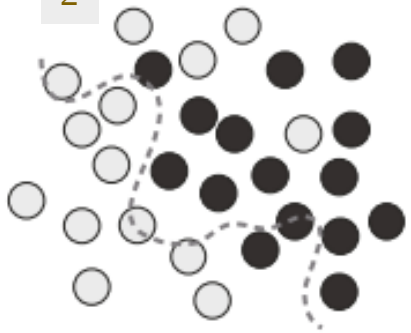
Before training:
the model starts
with a random initial state.

1



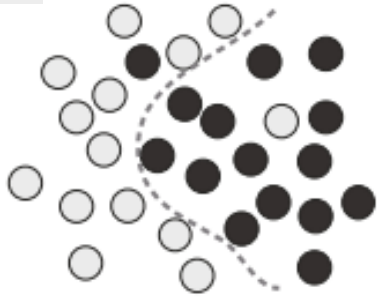
Beginning of training:
the model gradually
moves toward a better fit.

2



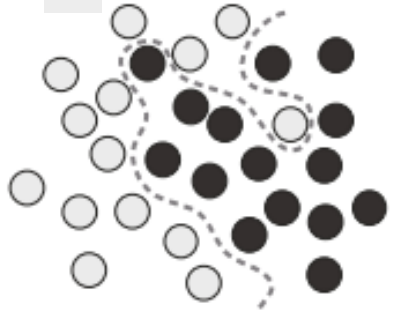
Further training: a robust
fit is achieved, transitively,
in the process of morphing
the model from its initial
state to its final state.

3



Final state: the model
overfits the training data,
reaching perfect training loss.

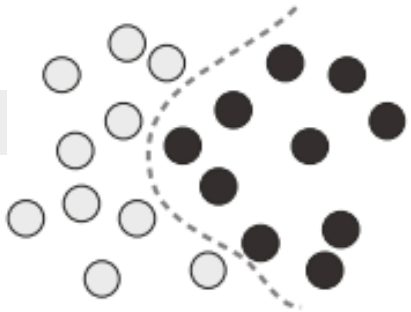
4



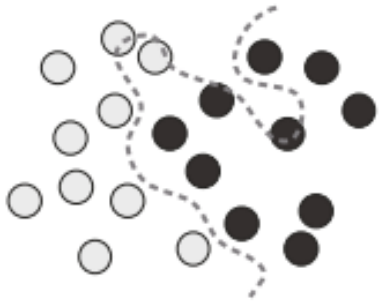
Testing

Test time: performance
of robustly fit model
on new data points

Robustly fit



Test time: performance
of overfit model
on new data points



Overfit

Model Generalization

❖ **Trick 1: 'Learn hard ' – randomly add noise to training data**

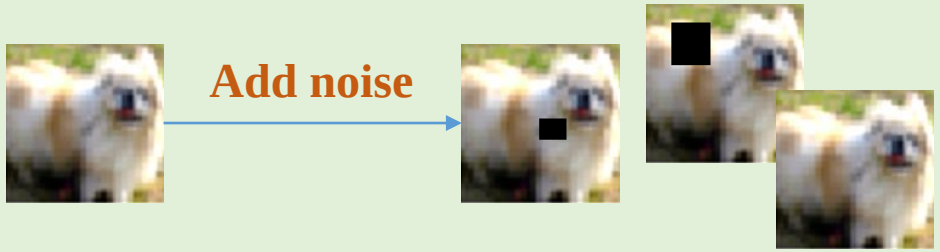
Model Generalization

❖ **Trick 1: 'Learn hard ' – randomly add noise to training data**

Speed-limit sign detection

Model Generalization

❖ Trick 1: 'Learn hard ' – randomly add noise to training data



In PyTorch

```
RandomErasing(p=0.75, scale=(0.01, 0.3),
              ratio=(1.0, 1.0), value=0,
              inplace=True)
```

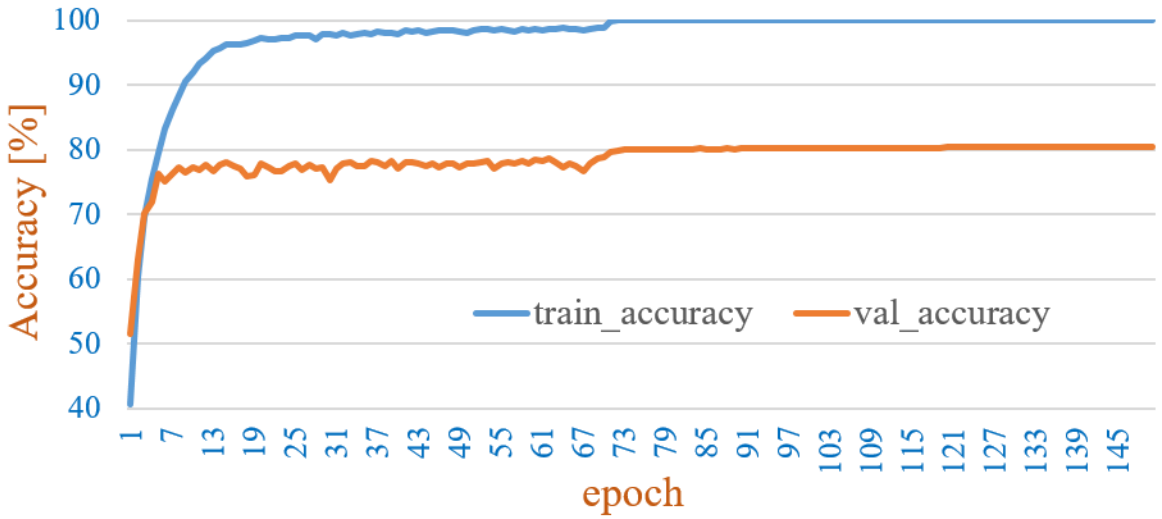
```
train_transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465],
                             [0.2470, 0.2435, 0.2616]),
        transforms.RandomErasing(p=0.75,
                                 scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0,
                                 inplace=True)
    ])
train_set = CIFAR10(root='./data', train=True,
                    download=True,
                    transform=train_transform )
trainloader = DataLoader(train_set,
                         batch_size=256,
                         shuffle=True,
                         num_workers=10 )
```

Model Generalization

❖ Trick 1: 'Learn hard ' – randomly add noise to training data

In PyTorch

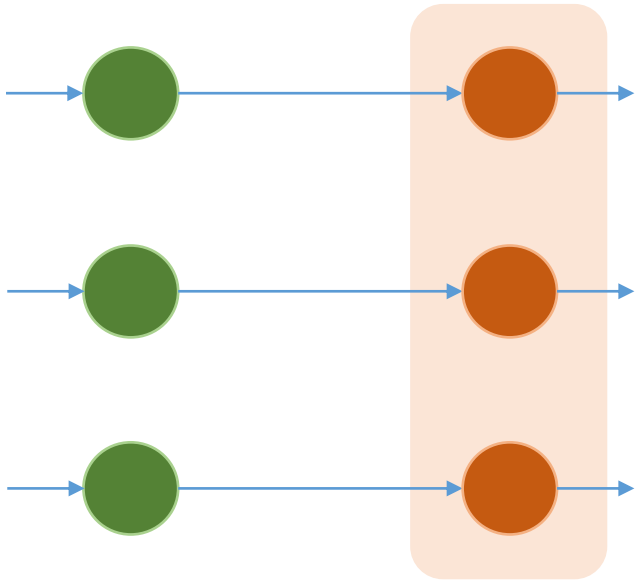
```
RandomErasing(p=0.75, scale=(0.01, 0.3),
              ratio=(1.0, 1.0), value=0,
              inplace=True)
```



val_accuracy increases from ~78% to ~80%

Network Training

❖ Solution 2: Batch normalization



Batch Normalization

Do not need bias when using BN

μ and σ are updated in forward pass
γ and β are updated in backward pass

Input data for a node in batch normalization layer

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

m is mini-batch size

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

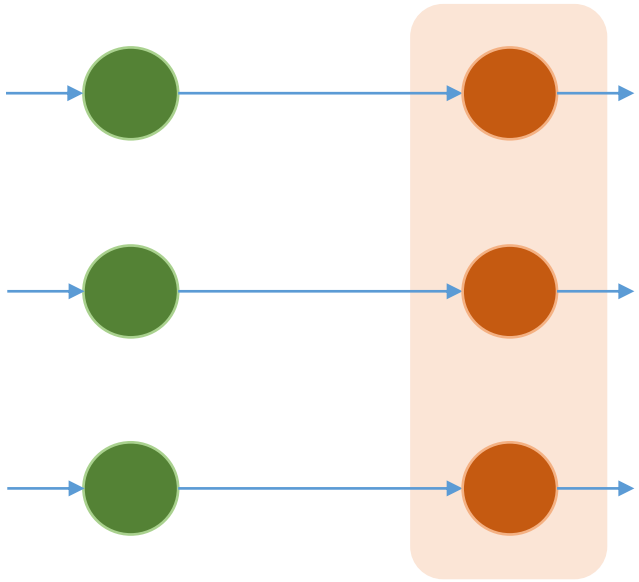
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Trick 2: Batch normalization



Batch Normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

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

Scale and shift $\hat{X}_i$

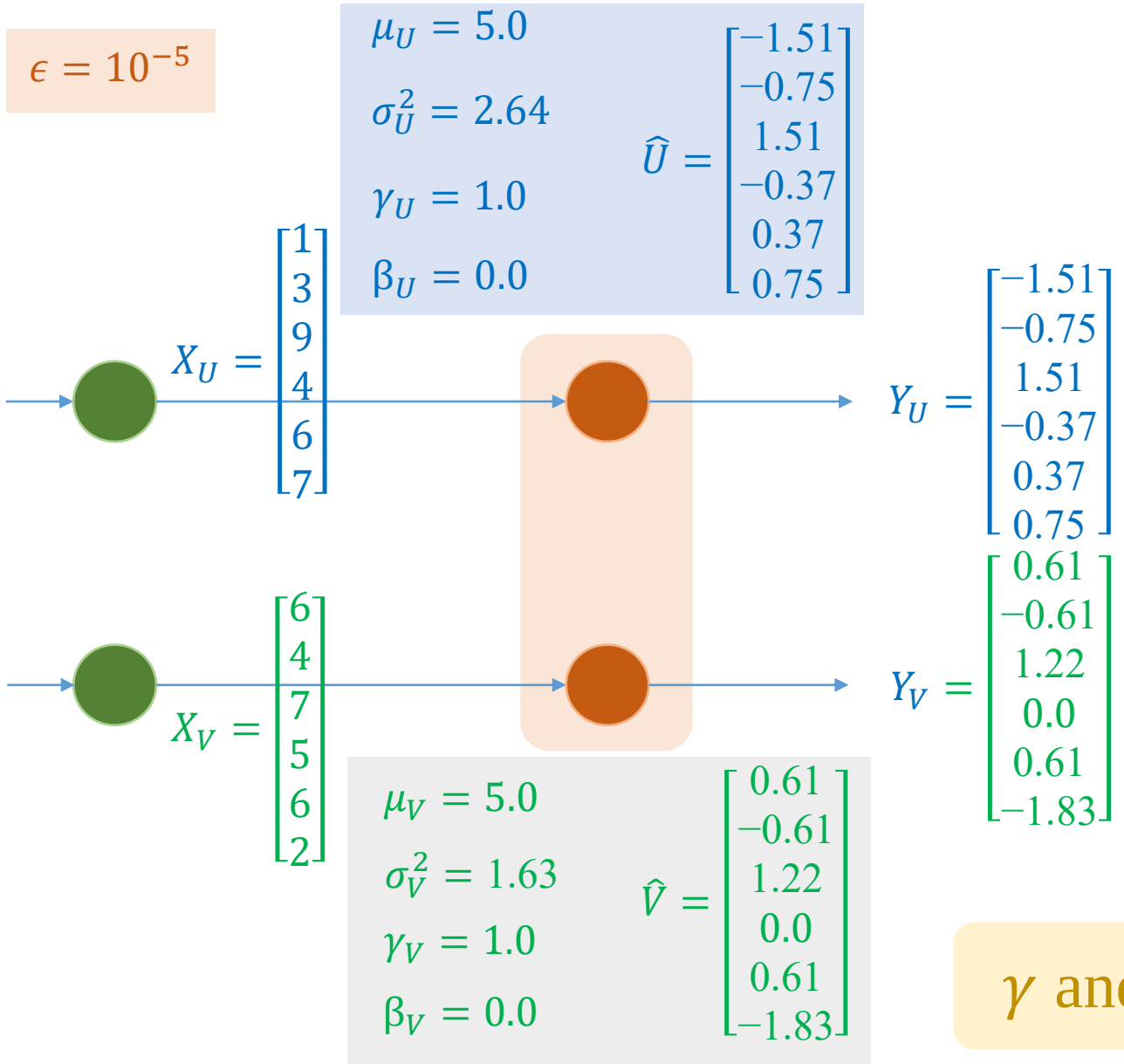
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

What if

$$\gamma = \sqrt{\sigma^2 + \epsilon} \text{ and } \beta = \mu$$

Network Training



Trick 2: Batch normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

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

Scale and shift $\hat{X}_i$

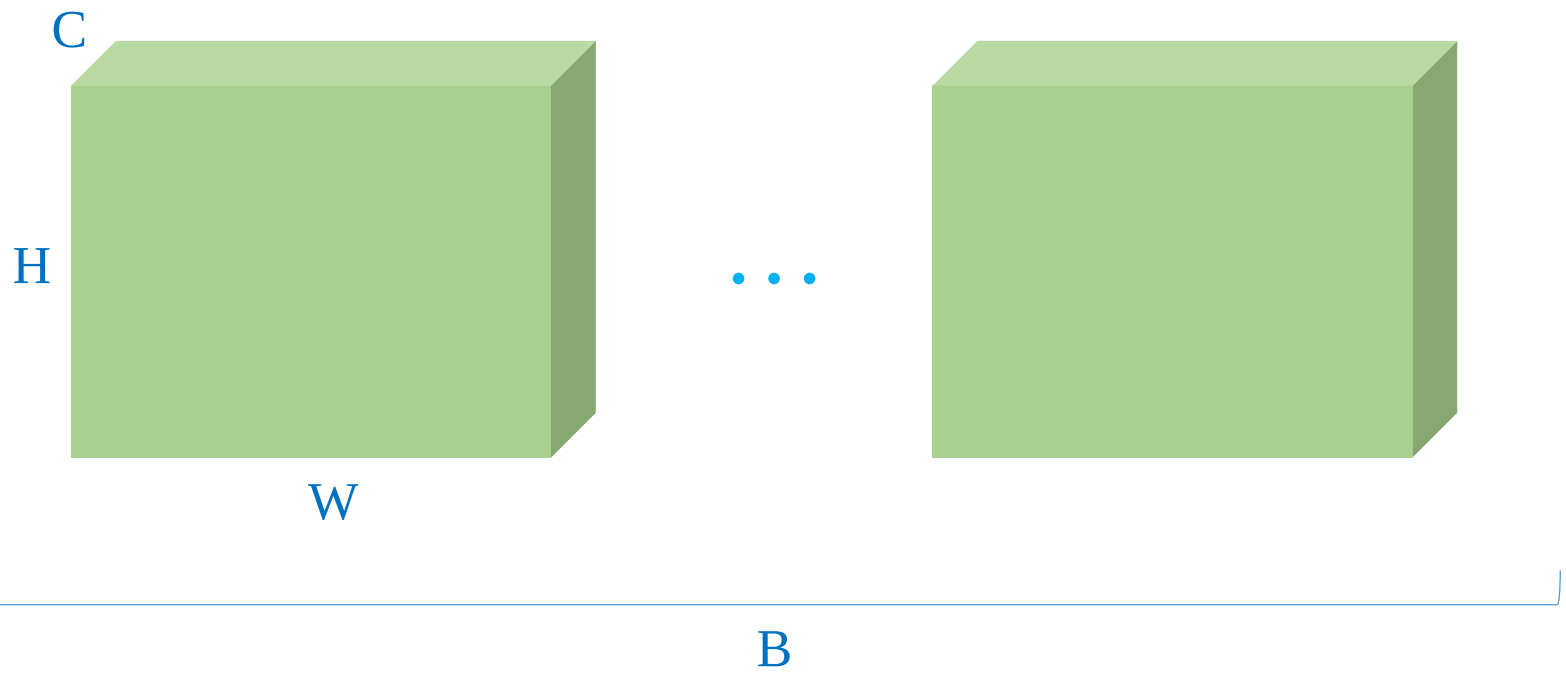
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

γ and β are updated in training process

Network Training

❖ Trick 2: Batch normalization for 2D data



Compute C means of H*W*B values

Compute C variances of H*W*B values

Network Training

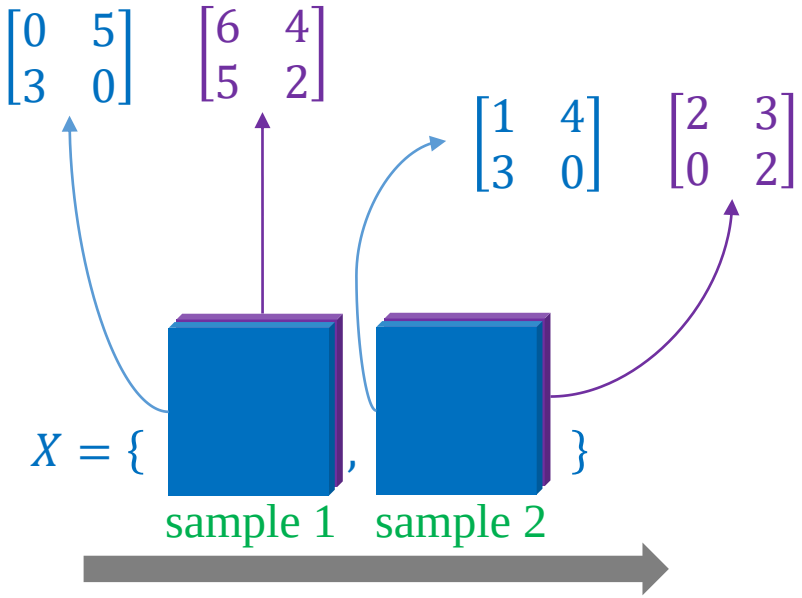
$\epsilon = 10^{-5}$

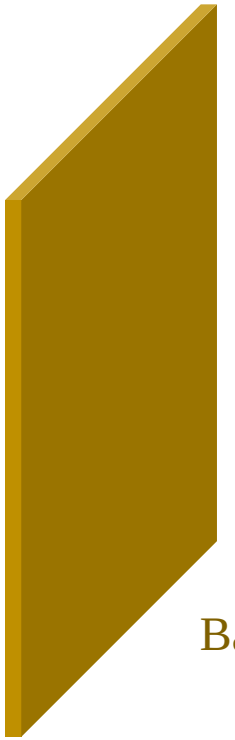
$\mu = [2.0, 3.0]$

$\sigma^2 = [4.0, 3.7]$

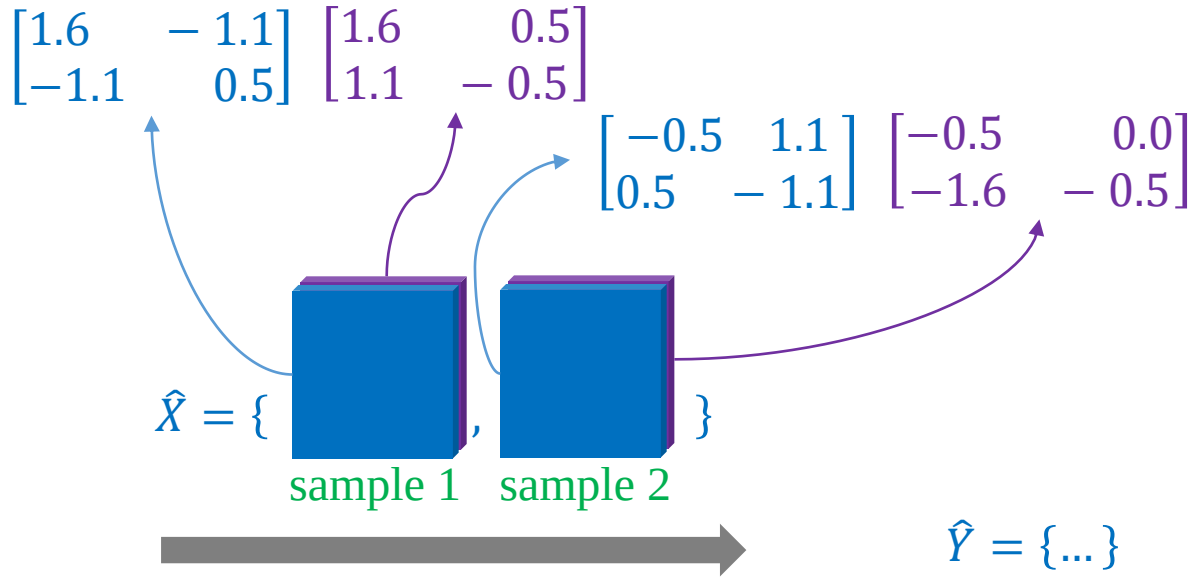
$\gamma = 1.0$

$\beta = 0.0$





Batch-Norm Layer



batch-size = 2

sample_shape = (2, 2, 2)

Network Training

❖ Trick 2: Batch normalization

Input data for a node in batch normalization layer

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

m is mini-batch size

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

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

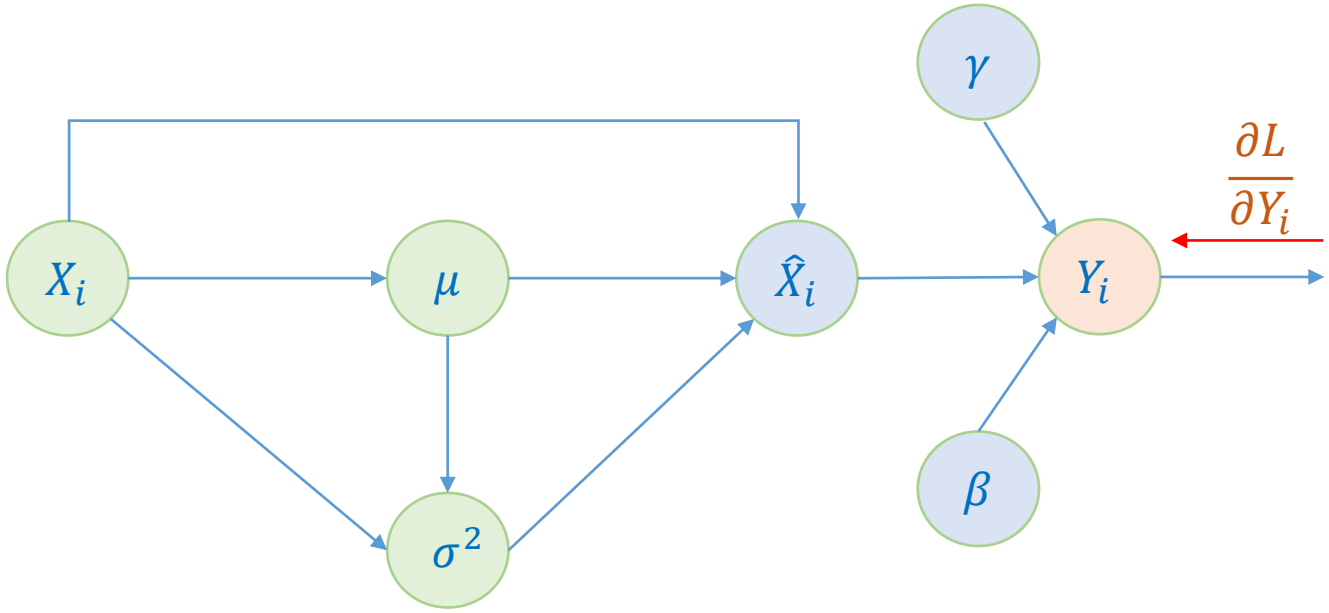
ϵ is a very small value

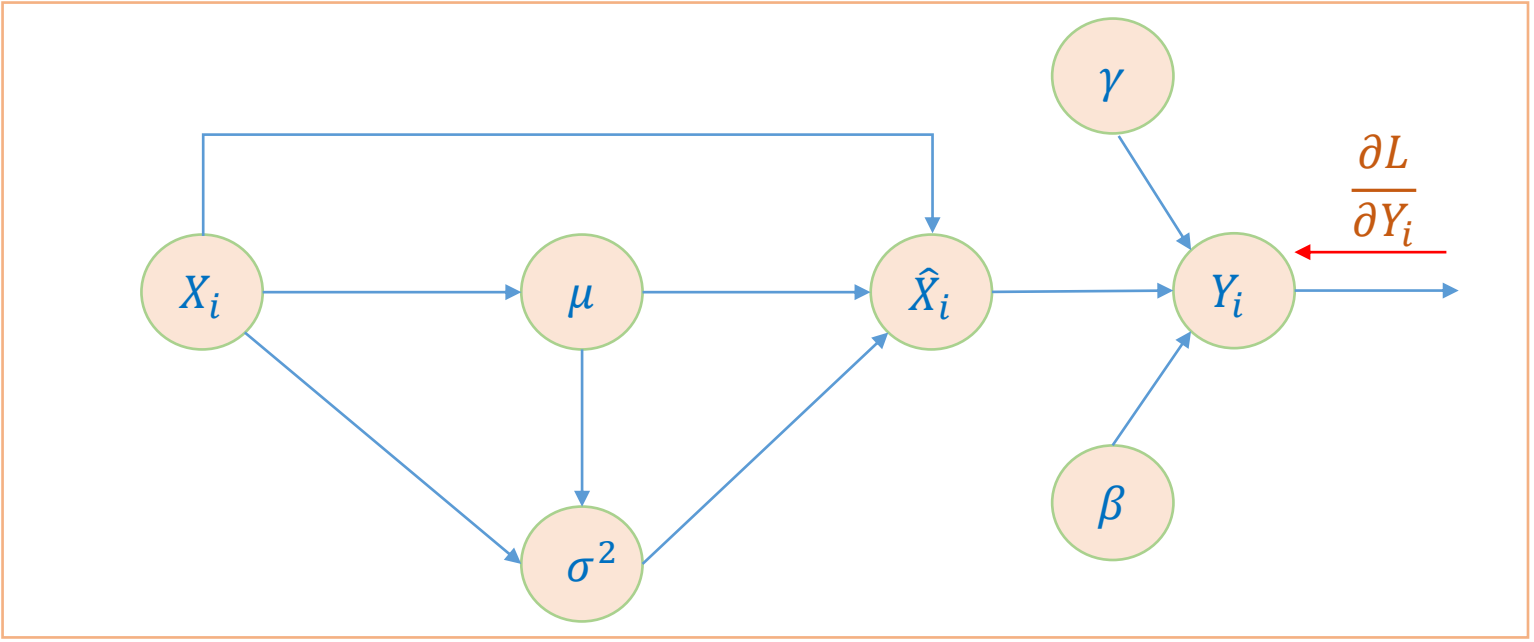
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Backward





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

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

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial Y_i} \hat{X}_i$$

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial Y_i}$$

$$\frac{\partial L}{\partial \hat{X}_i} = \frac{\partial L}{\partial Y_i} \gamma$$

$$\frac{\partial L}{\partial \sigma^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{X}_i} \frac{\partial \hat{X}_i}{\partial \sigma^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{X}_i} (X_i - \mu) \frac{-1}{2} (\sigma^2 + \epsilon)^{\frac{-3}{2}}$$

$$\frac{\partial L}{\partial \mu} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{X}_i} \frac{-1}{\sqrt{\sigma^2 + \epsilon}} - \frac{\partial L}{\partial \sigma^2} \frac{1}{m} \sum_{i=1}^{m} 2(X_i - \mu)$$

$$\frac{\partial L}{\partial X_i} = \frac{\partial L}{\partial \hat{X}_i} \frac{\partial \hat{X}_i}{\partial X_i} + \frac{\partial L}{\partial \mu} \frac{\partial \mu}{\partial X_i} + \frac{\partial L}{\partial \sigma^2} \frac{\partial \sigma^2}{\partial X_i}$$

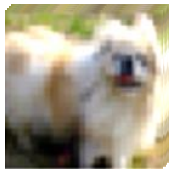
$$\frac{\partial \hat{X}_i}{\partial X_i} = \frac{1}{\sqrt{\sigma^2 + \epsilon}}$$

$$\frac{\partial \sigma^2}{\partial X_i} = \frac{2(X_i - \mu)}{m}$$

$$\frac{\partial \mu}{\partial X_i} = \frac{1}{m}$$

Model Generalization

❖ Trick 2: Batch normalization



mini-batch 1

mini-batch 2

$$(\mu_1, \sigma_1) \neq (\mu_2, \sigma_2)$$

very likely

Add noise to the output of BN layers

Input data for a node in batch normalization layer

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

m is mini-batch size

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

Scale and shift $\hat{X}_i$

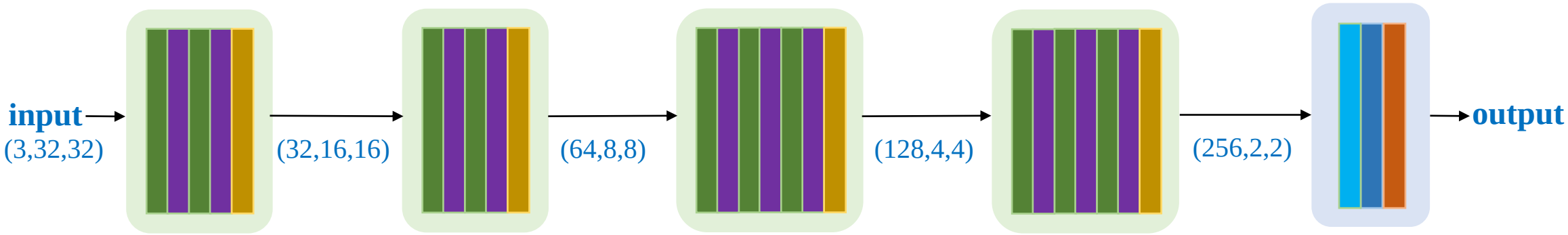
$$Y_i = \gamma \hat{X}_i + \beta$$

γ and β are two learning parameters

Model Generalization

```
conv_layer = nn.Sequential(nn.Conv2d(3, 32, 3,
                                     padding=1),
                           nn.ReLU(),
                           nn.BatchNorm2d(32))
```

❖ Trick 2: Batch normalization



(3x3) Convolution padding='same' stride=1 + ReLU

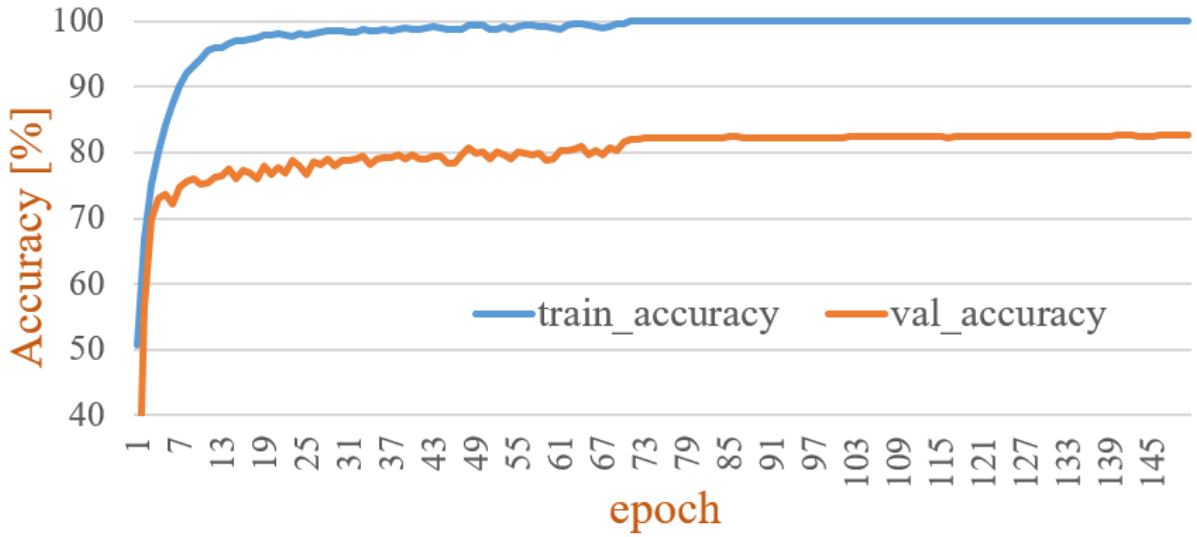
(2x2) max pooling

Batch normalization

Flatten

Dense Layer-512 + ReLU

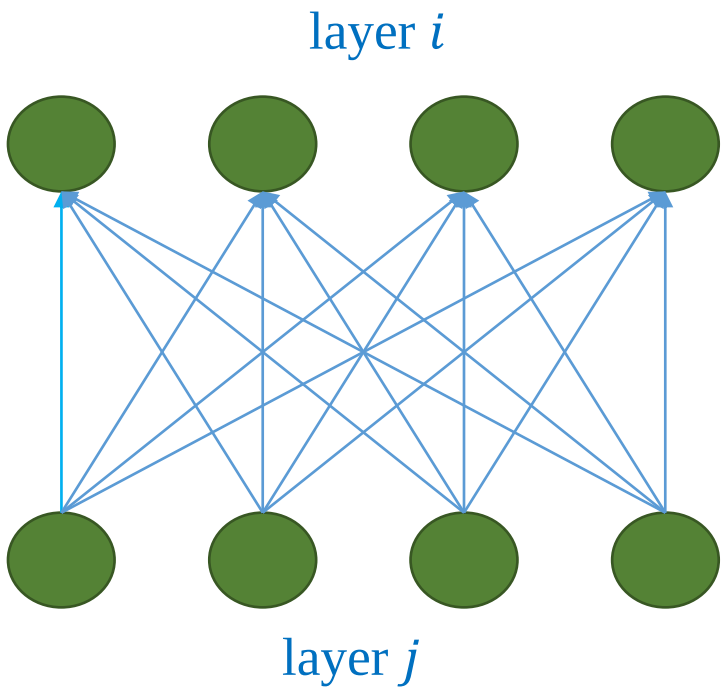
Dense Layer-10 + Softmax



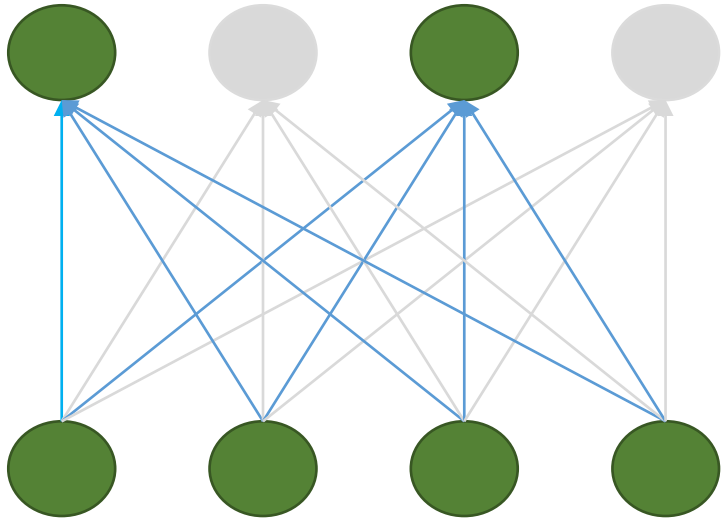
val_accuracy increases from ~80.9% to ~84%

Model Generalization

❖ Trick 3: Dropout



Apply dropout 50% to layer i



~50% nodes randomly selected in the i^{th} layer are set to zeros (kind of noise adding)

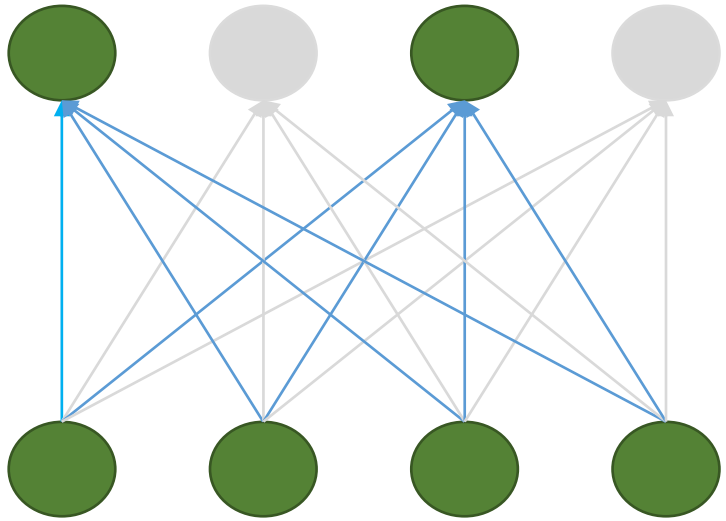
Model Generalization

❖ Trick 3: Dropout

$$a = D \odot \sigma(Z)$$

$$\frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial \sigma} = \frac{\partial L}{\partial a} \times D$$

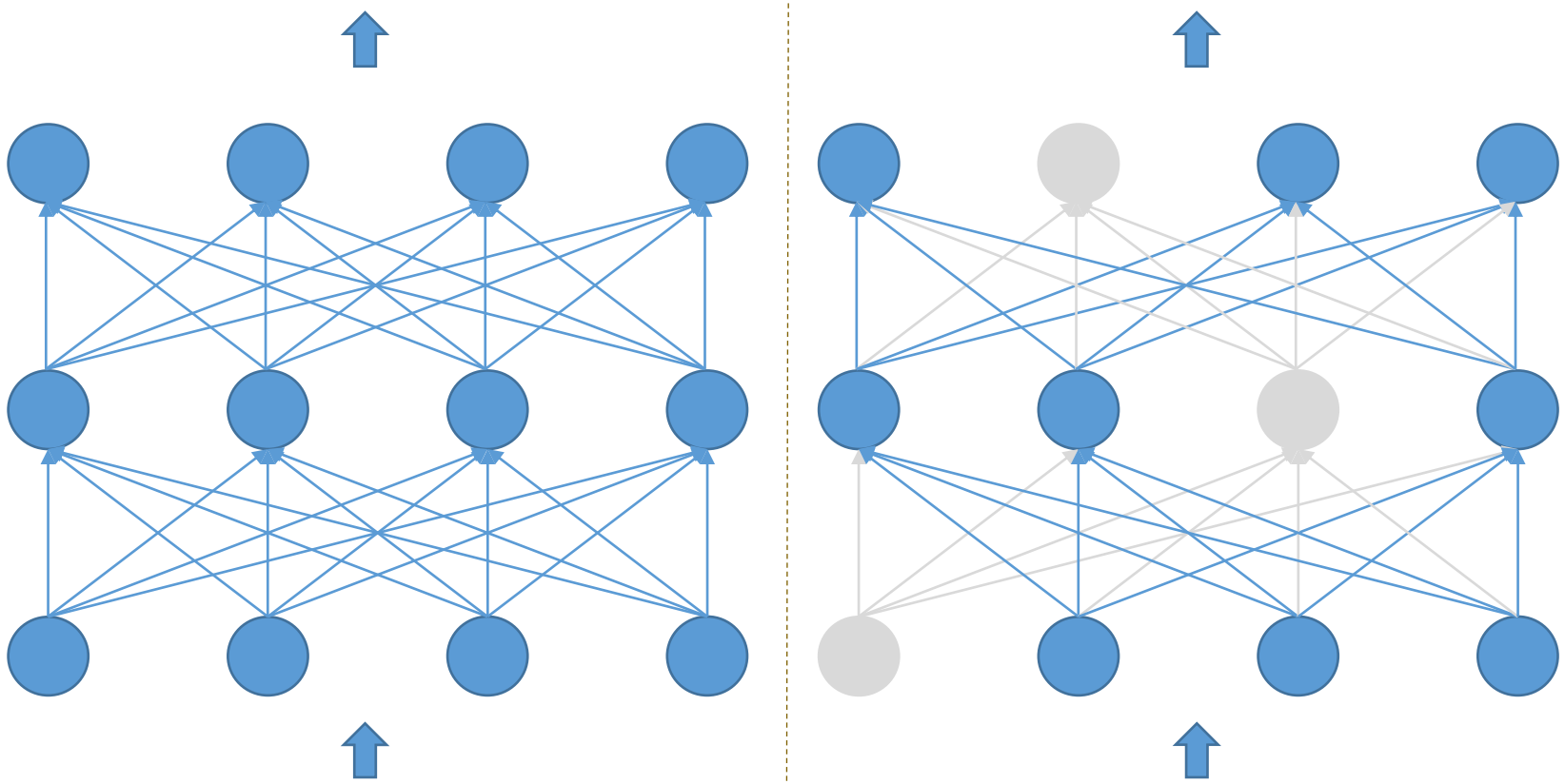
Apply dropout 50% to layer i



~50% nodes randomly selected in the i^{th} layer are set to zeros

Overfitting

Dropout



Given a dropping rate r

Randomly sets input units to 0 with a frequency of r

Only applying in training mode

```
nn.Sequential(nn.Conv2d(32, 32, 3,
                        padding=1),
              nn.ReLU(),
              nn.BatchNorm2d(32),
              nn.MaxPool2d(2, 2),
              nn.Dropout(dropout_rate)
```

$$scale = \frac{1}{1-r}$$

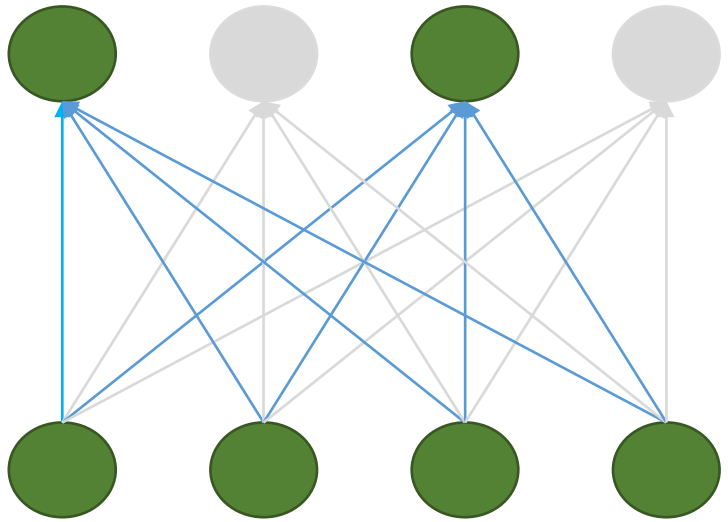
Model Generalization

❖ Trick 3: Dropout

```python
class Dropout():

    def __init__(self,prob=0.5):
        self.prob = prob
        self.params = []

    def forward(self,X):
        self.mask = np.random.binomial(1,self.prob,size=X.shape) / self.prob
        out = X * self.mask
        return out.reshape(X.shape)

    def backward(self,dout):
        dX = dout * self.mask
        return dX,[]
```

https://deepnotes.io/dropout

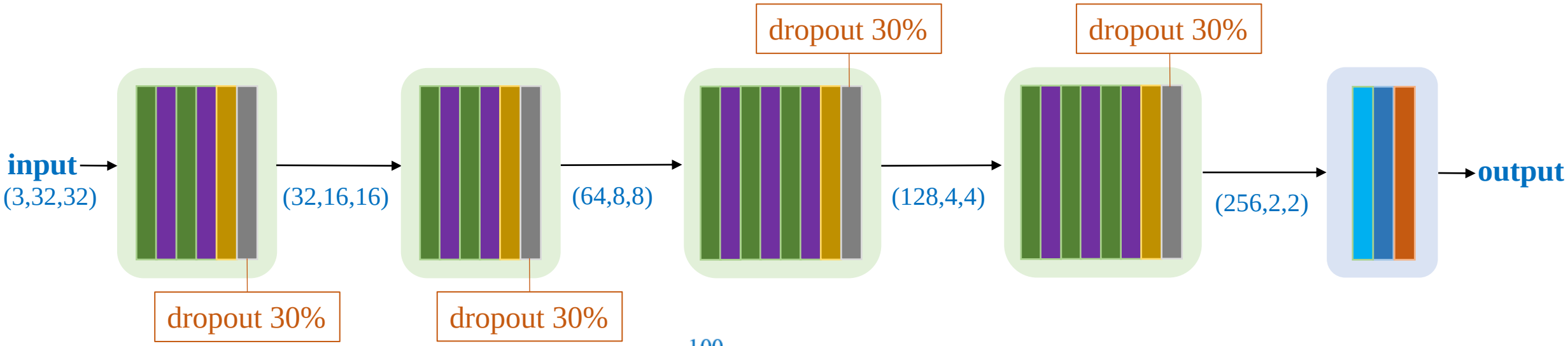
Apply dropout 50% to layer i



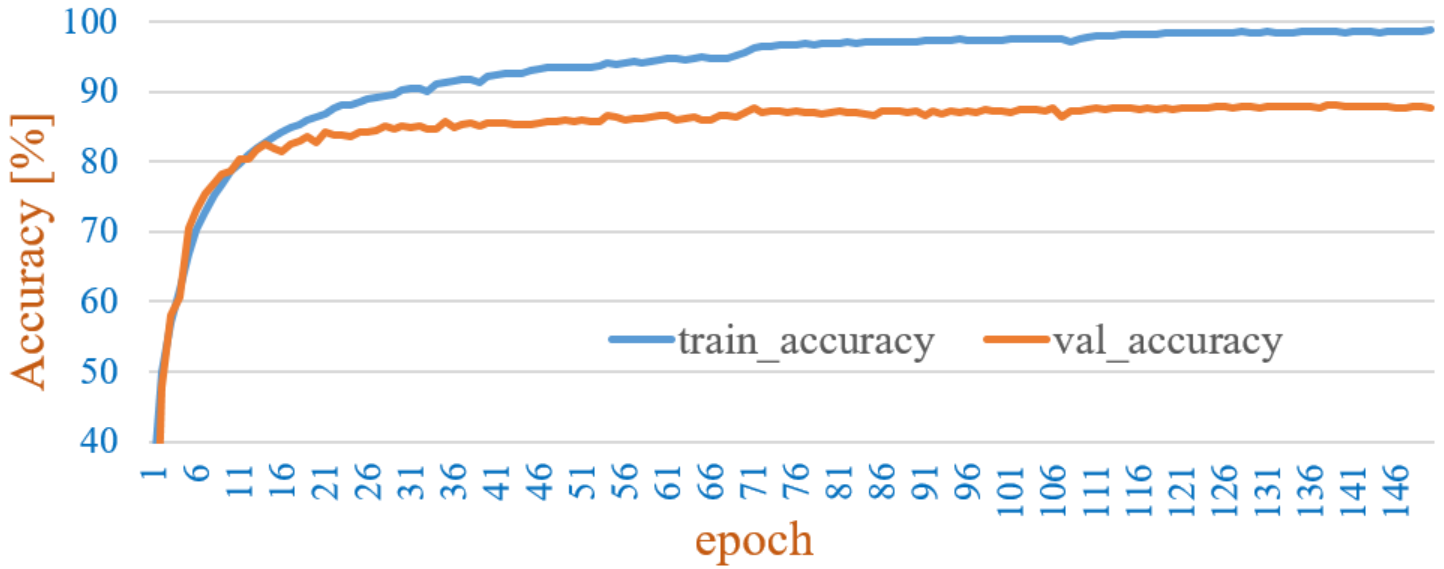
~50% nodes randomly selected in the i^{th} layer are set to zeros

Model Generalization

❖ Trick 3: Dropout



val_accuracy increases from ~84% to ~86.6%



Model Generalization

❖ Trick 4: Kernel regularization

$$L = crossentropy + \underbrace{\lambda \|W\|^2}_{L_2 \text{regularization}}$$

Prevent network from focusing on specific features

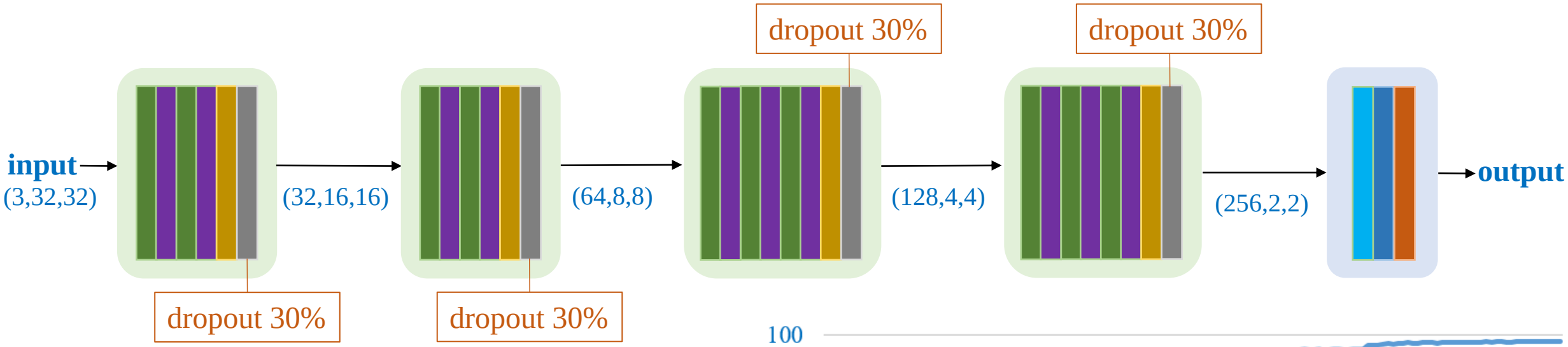
Smaller weights
→ simpler models

In PyTorch

```
optimizer = Adam(model.parameters(),
                 lr=1e-3,
                 weight_decay=5e-4)
```

Model Generalization

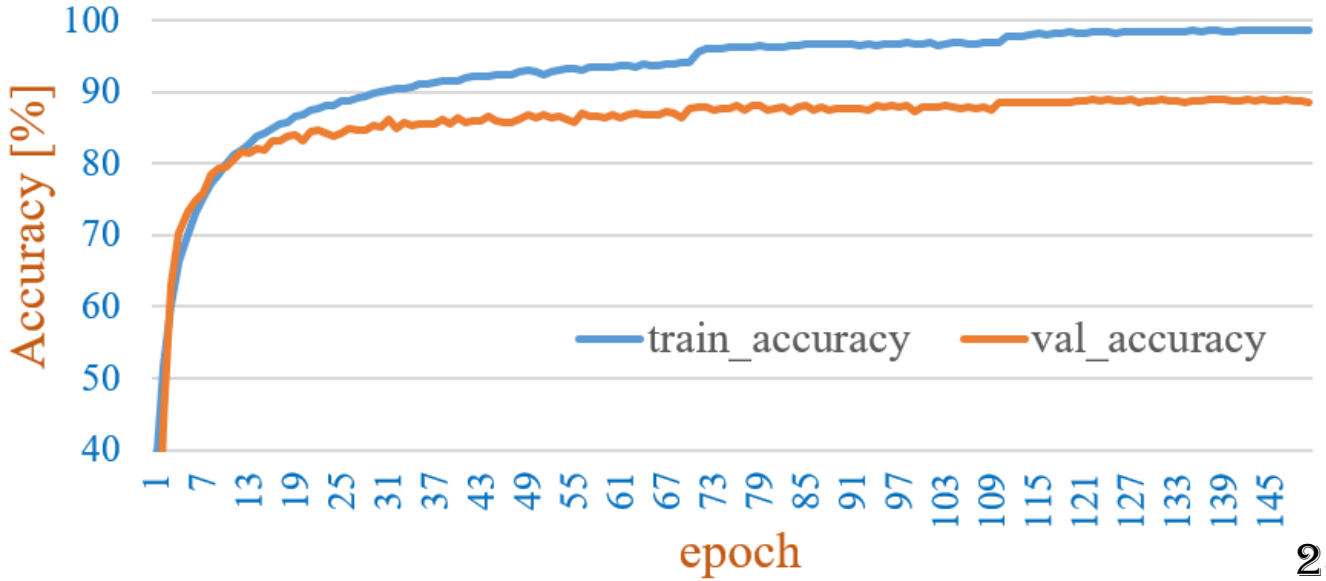
❖ Trick 4: Kernel regularizer



Dropout

(3x3) Convolution padding='same' stride=1 + ReLU + kernel regularization

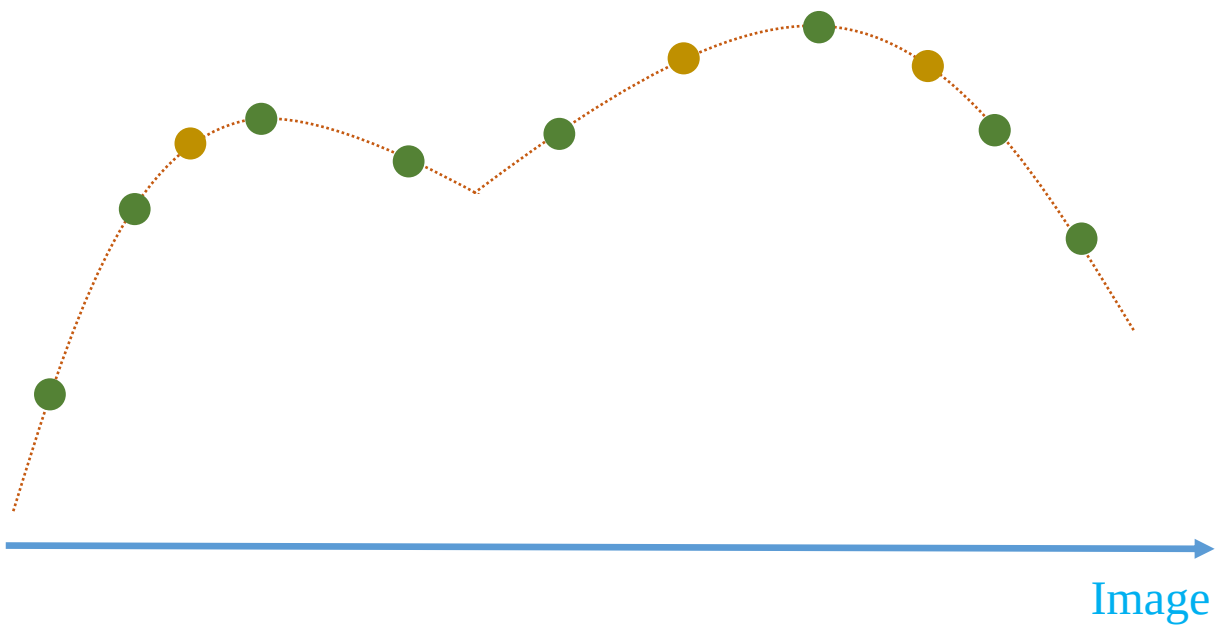
val_accuracy increases from ~86.6% to ~87.5%



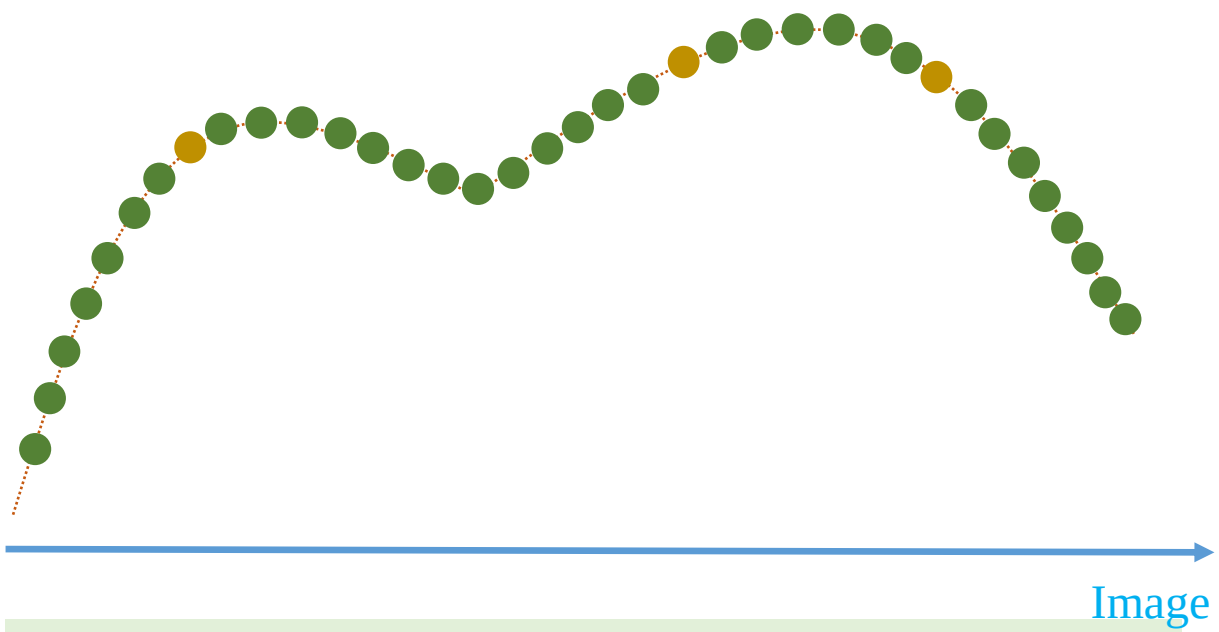
Model Generalization

❖ Trick 5: Data augmentation

A normal case



A perfect case: Have unlimited training

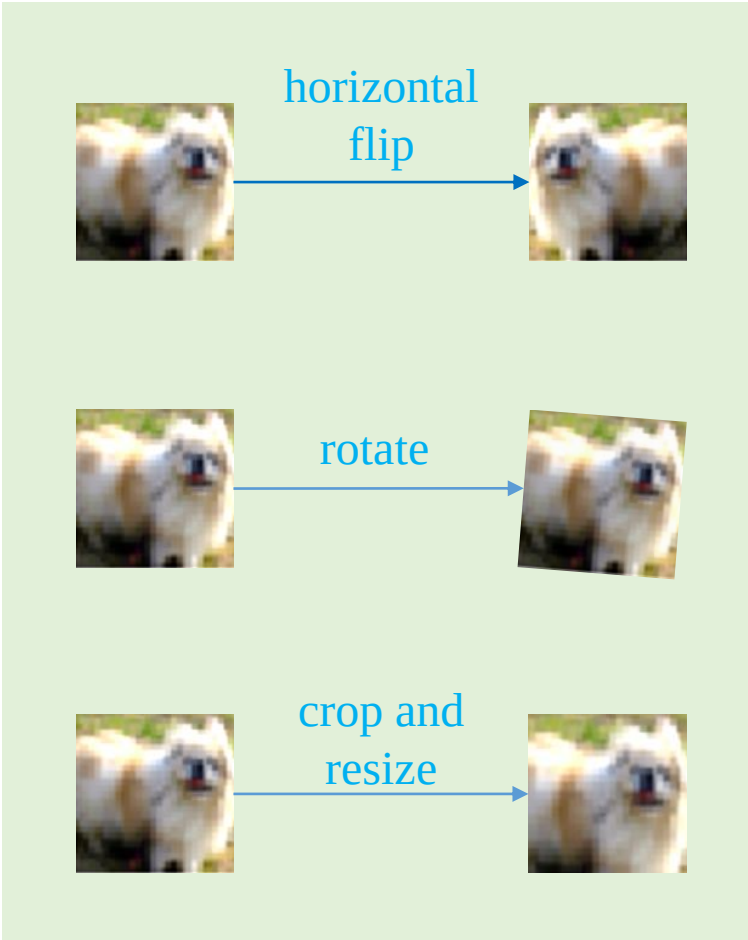


Training data cover the whole distribution

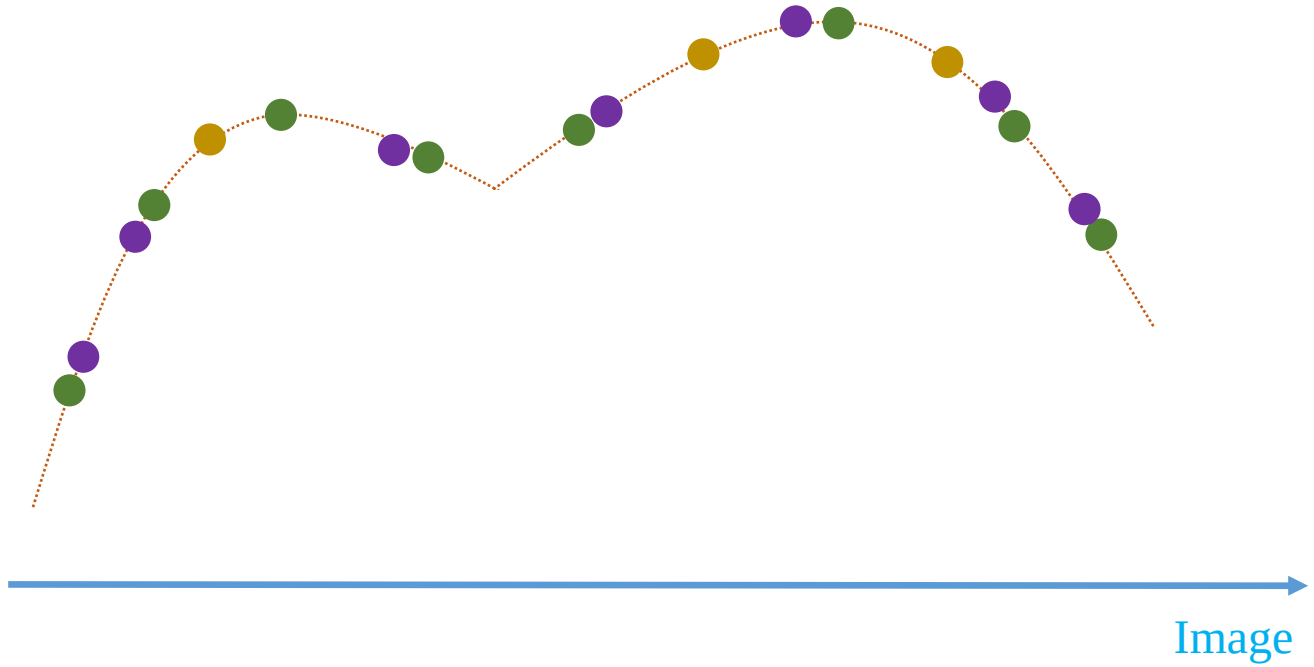
But, impractical!!!

Model Generalization

❖ Trick 5: Data augmentation



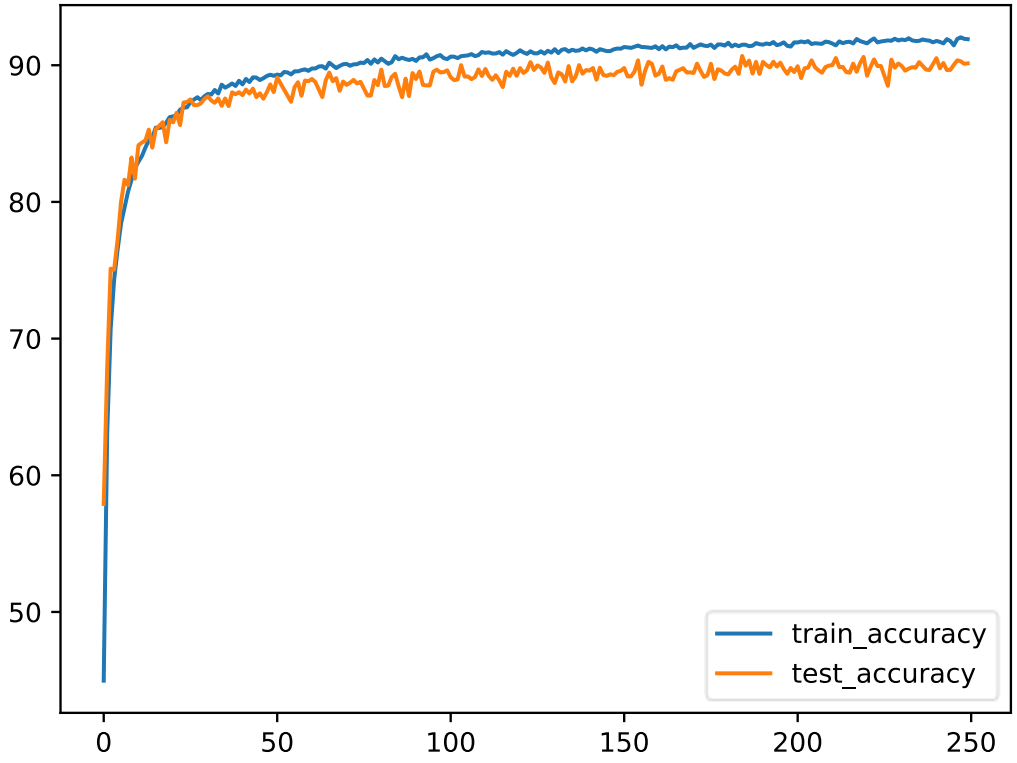
Increase data by altering the training data



Model Generalization

❖ Trick 5: Data augmentation

Horizontal flip + crop-and-resize



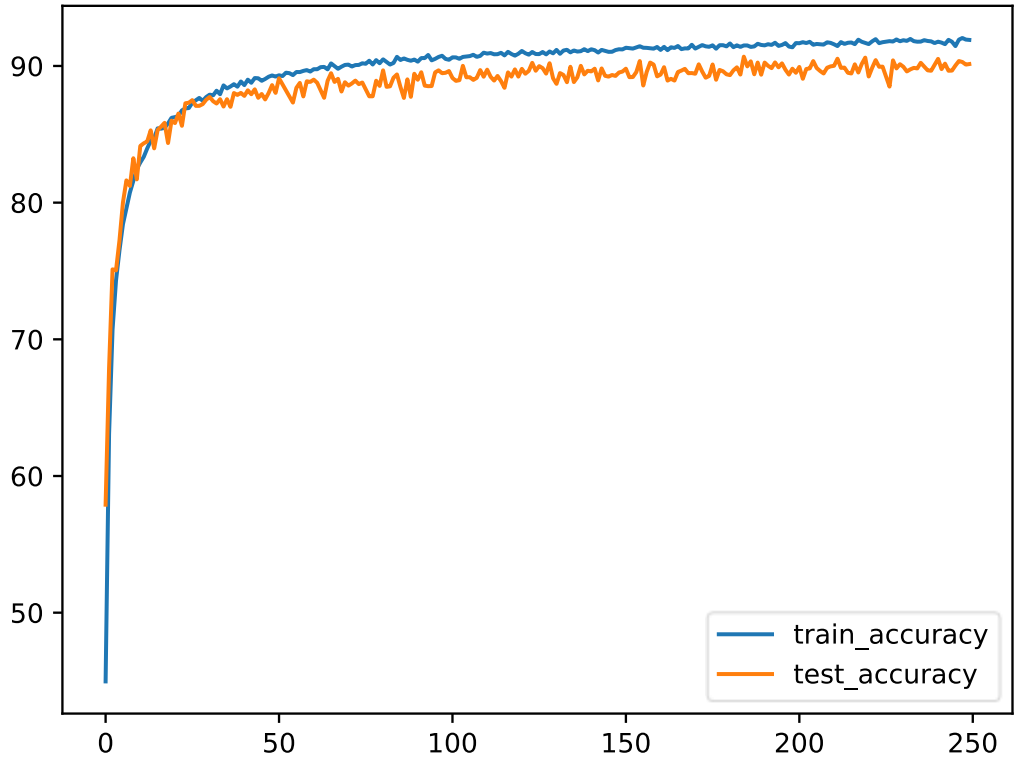

```python
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                         std=[0.2471, 0.2435, 0.2616])
])
```

val_accuracy reaches to ~90.6%

Model Generalization

❖ What we have

Horizontal flip + crop-and-resize



val_accuracy reaches to ~90.6%

train_accuracy reaches to ~92%

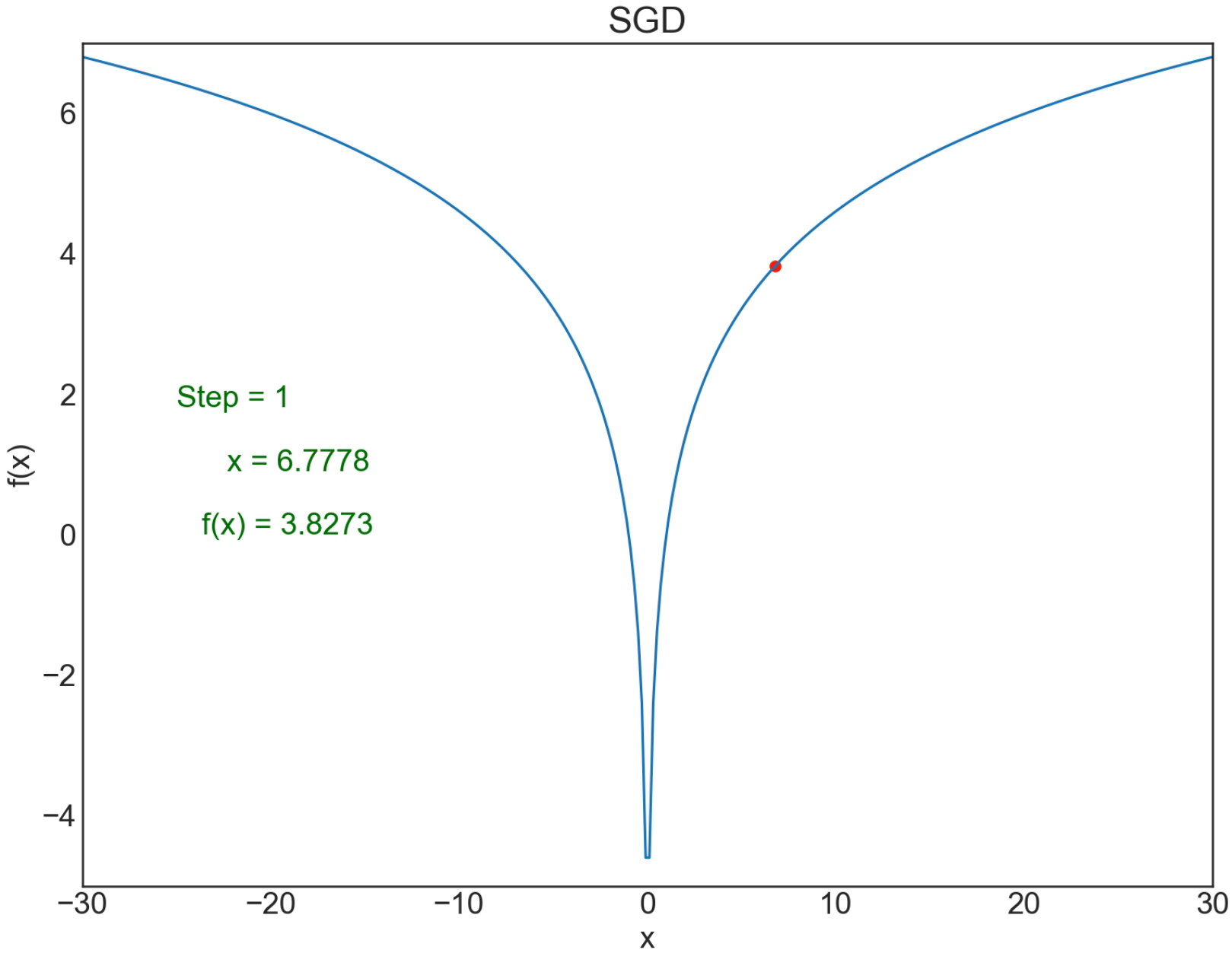
- Batch normalization
- Dropout
- Kernel regularization
- Data augmentation

Idea: try to increase train_accuracy, expect val_accuracy increases too

➡ **Increase model capacity**

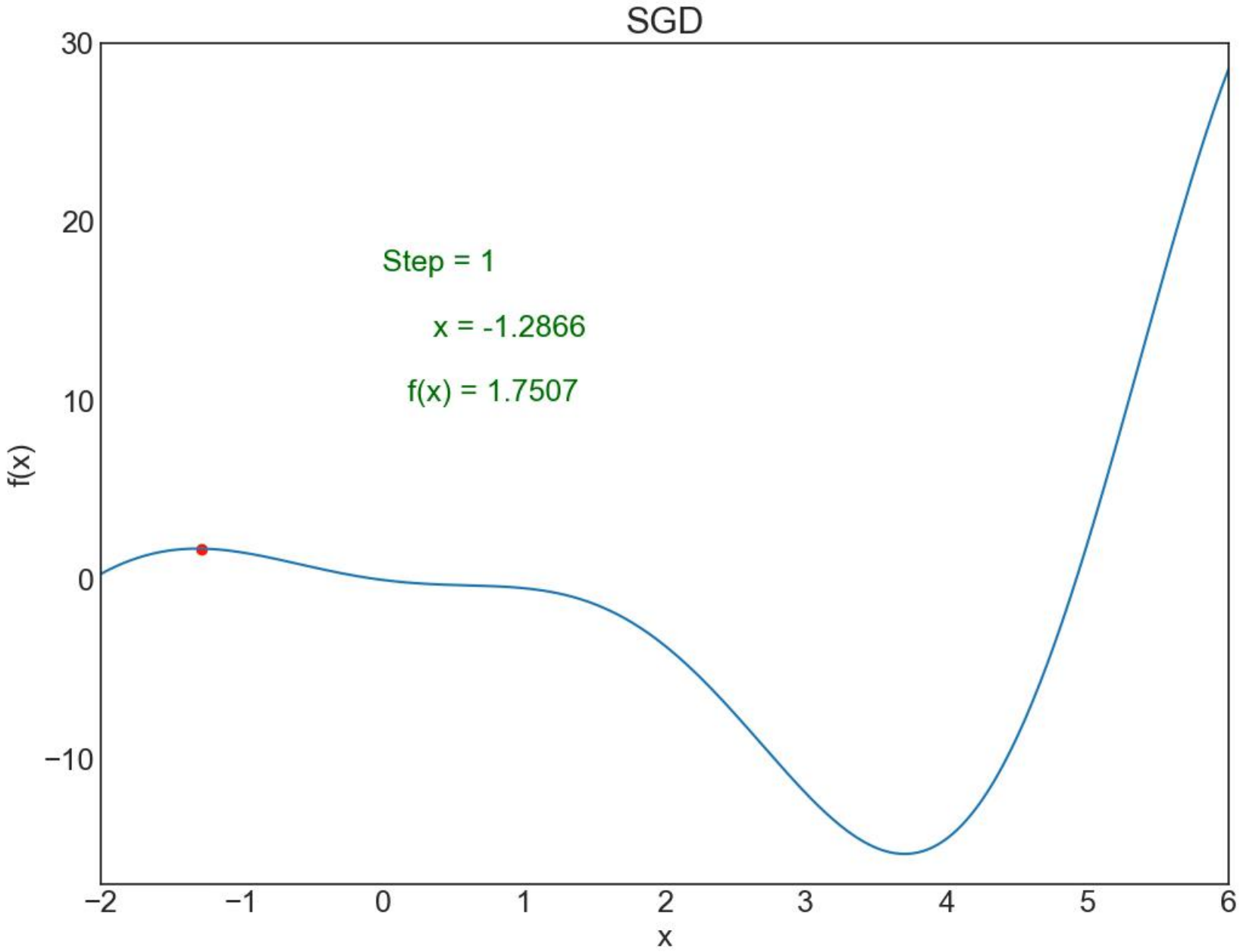
Optimization

❖ **Learning rate**



Optimization

❖ **Learning rate**

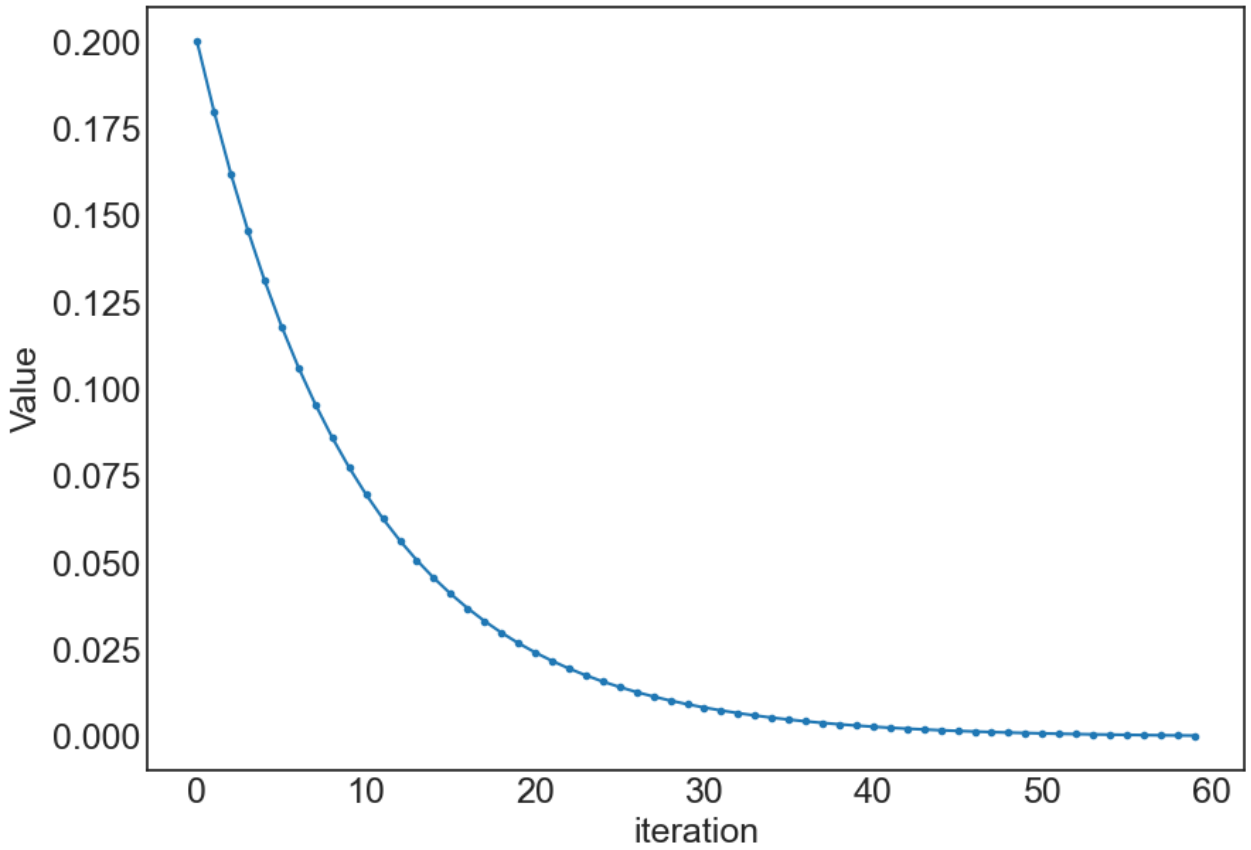


Model Generalization

❖ Trick 6: Reduce learning rate

$$\eta = \eta_0 \times \gamma^{epoch}$$

```python
lr_scheduler.ExponentialLR(optimizer=optimizer,
                           gamma=0.96)

# train
for epoch in range(max_epoch):
    # ...
    for i, (inputs, labels) in enumerate(trainloader):
        # ...

    #...

    # update learning rate
    lr_scheduler.step()
```

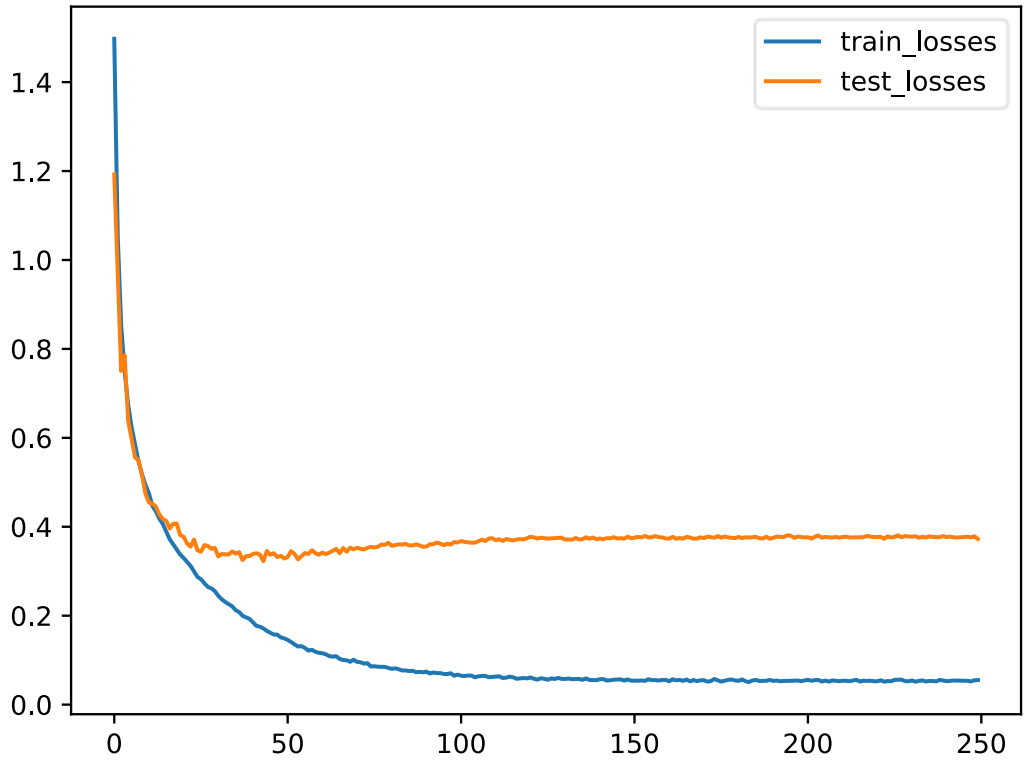


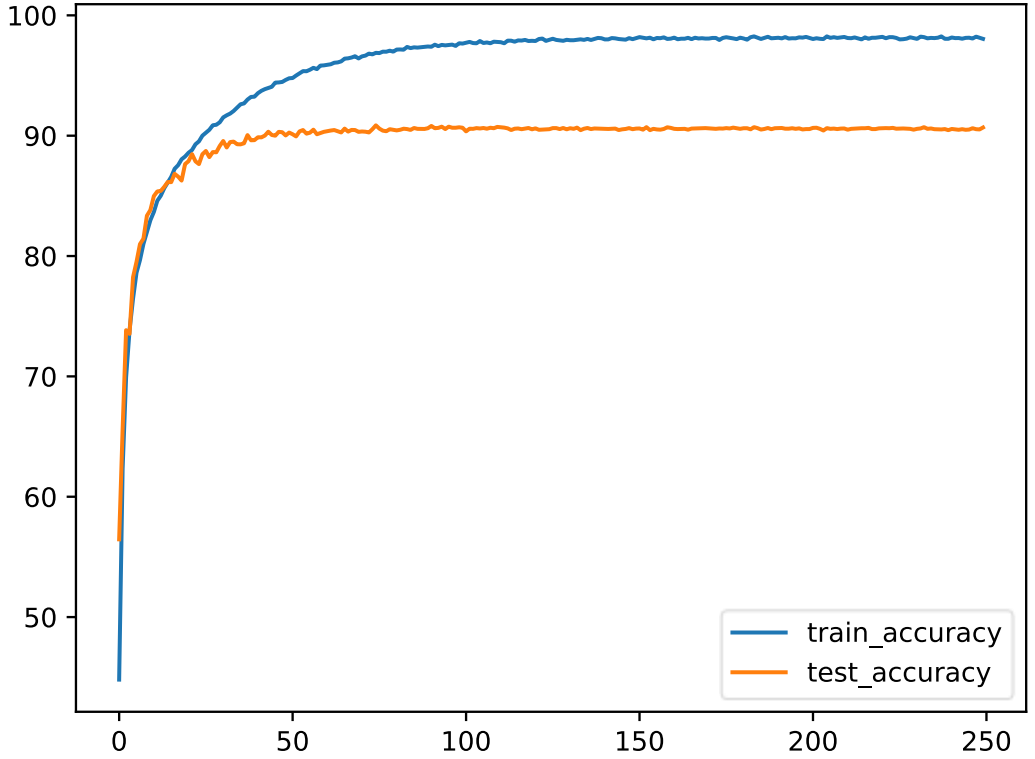
Model Generalization

❖ Trick 6: Reduce learning rate

val_accuracy reaches to ~90.6%

train_accuracy reaches to ~98%

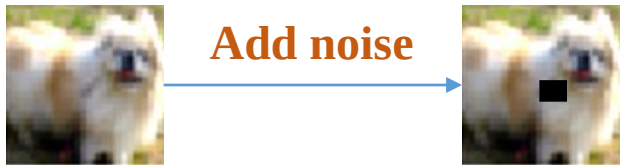




Model Generalization

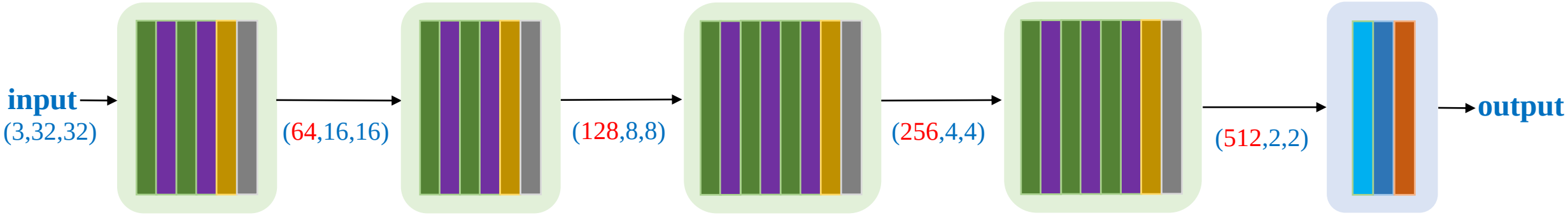
❖ **Discussion: Predict training and test accuracy when using more data augmentation**

```
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                         std=[0.2471, 0.2435, 0.2616]),
    transforms.RandomErasing(p=0.75,
                             scale=(0.01, 0.3),
                             ratio=(1.0, 1.0),
                             value=0,
                             inplace =True)
])
```



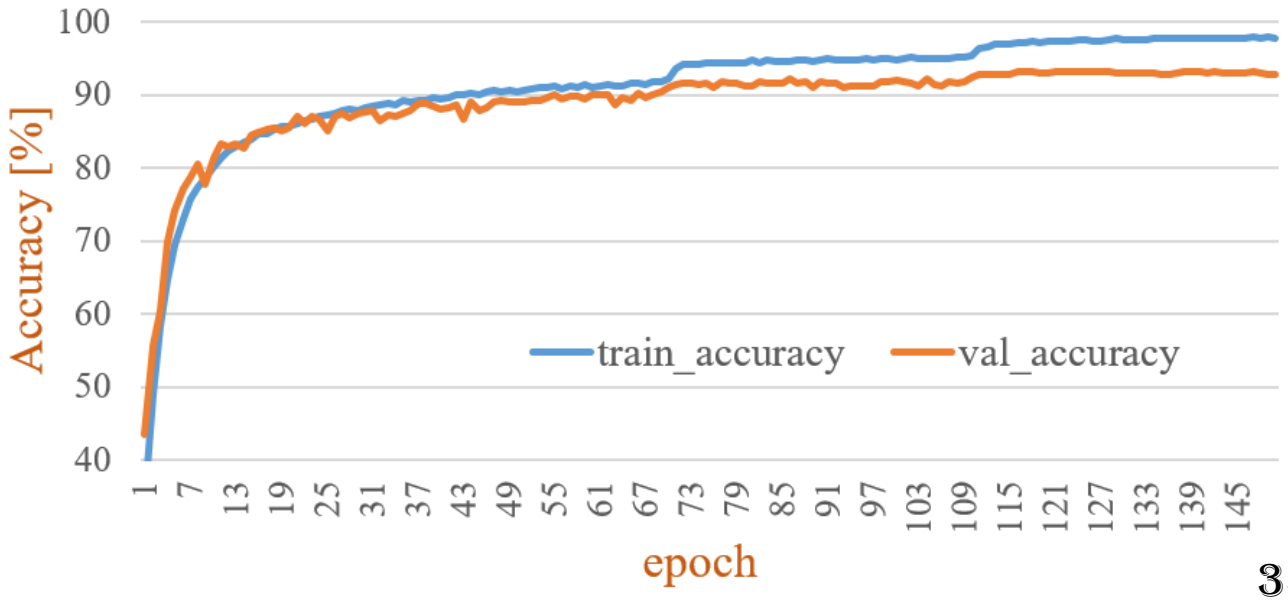
Model Generalization

❖ Trick 7: Increase model capacity (and use more data augmentation)



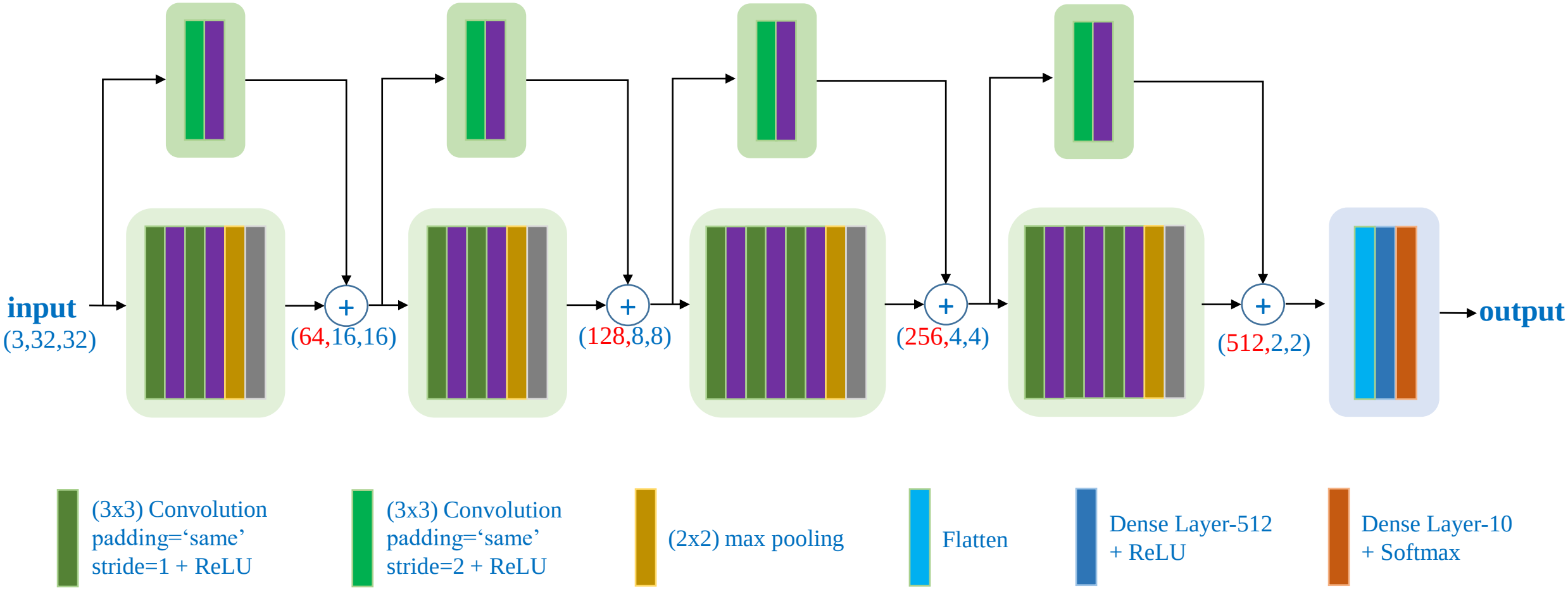
val_accuracy reaches to ~93%

train_accuracy reaches to ~96%



Model Generalization

❖ Trick 8: Using skip-connection

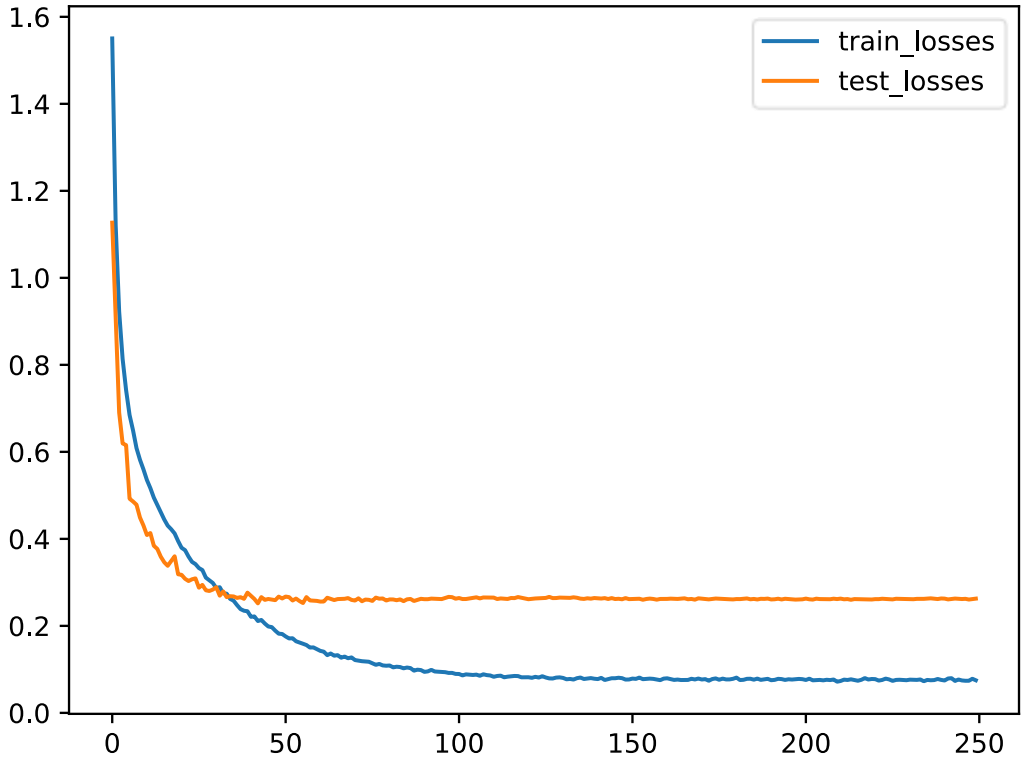


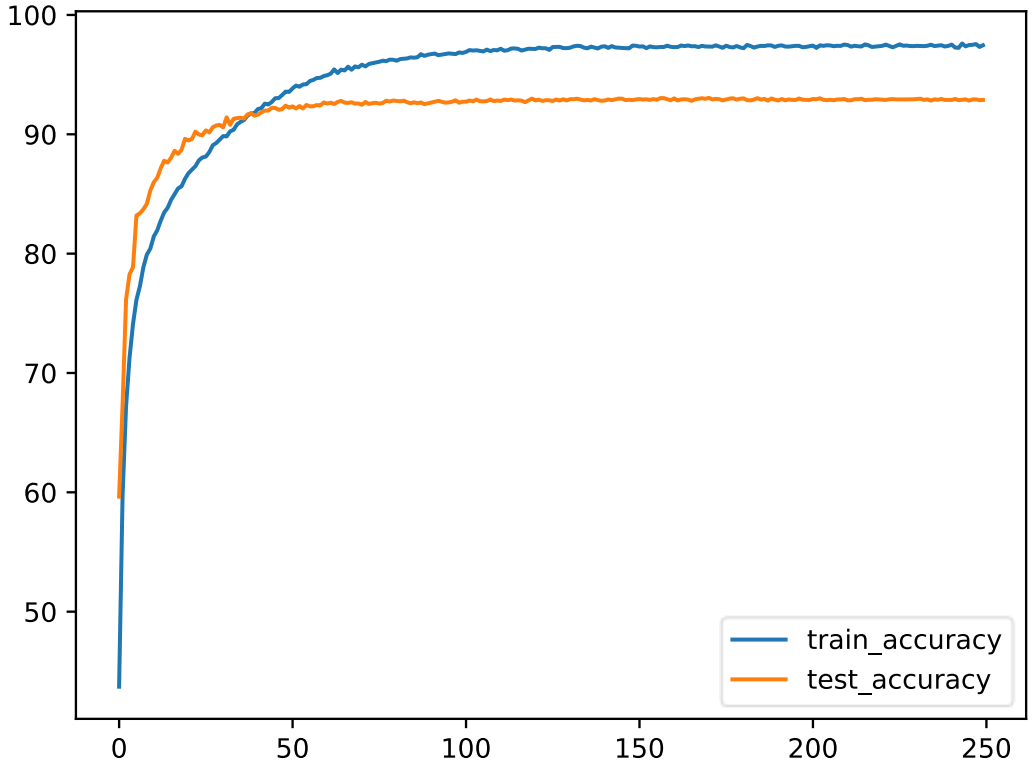
Model Generalization

❖ Trick 8: Using skip-connection

val_accuracy reaches to ~93%

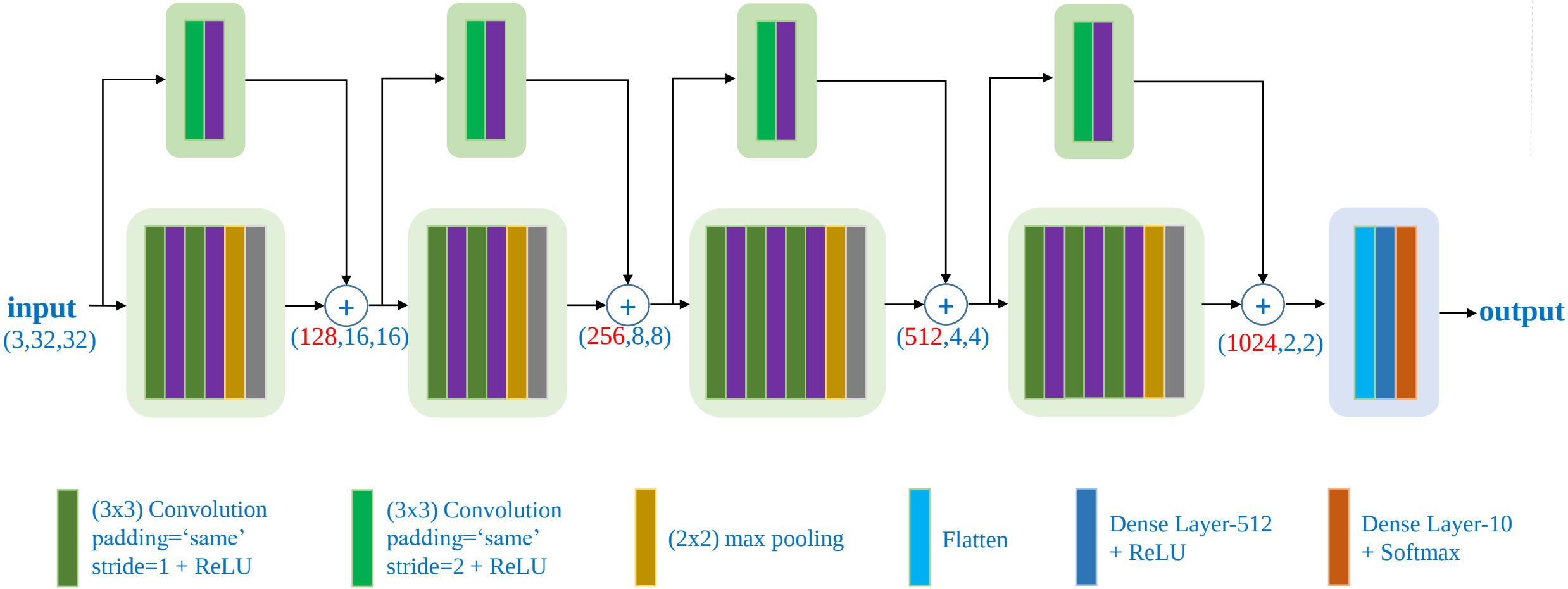
train_accuracy reaches to ~97%





Model Generalization

❖ Increase model capacity once more

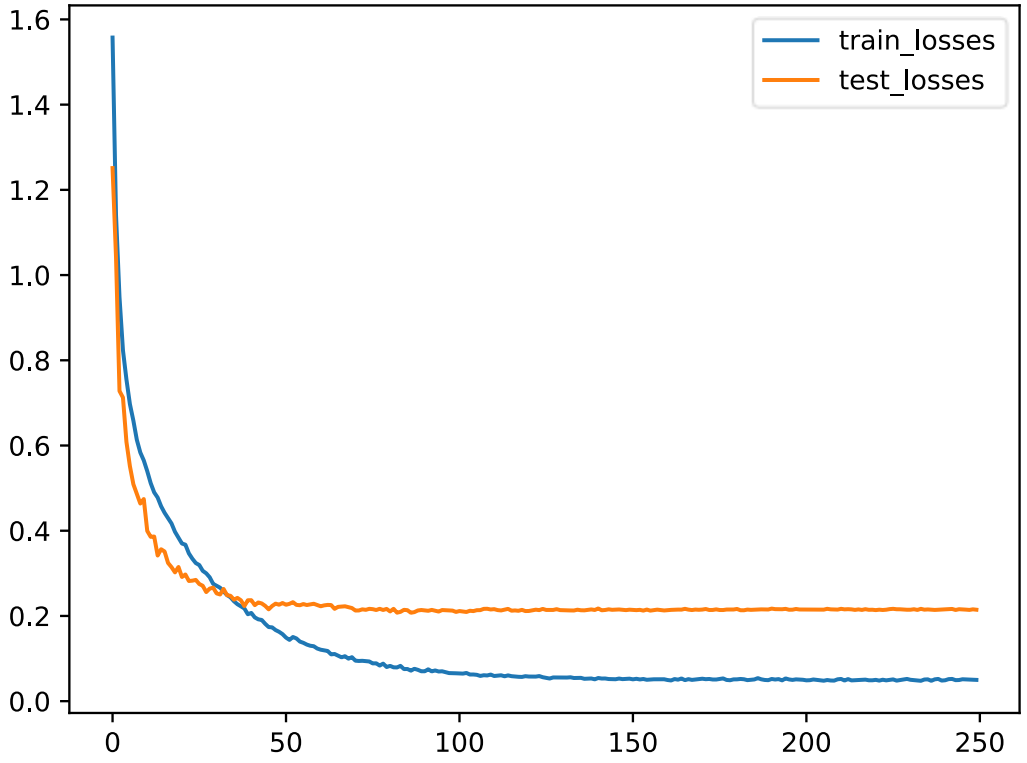


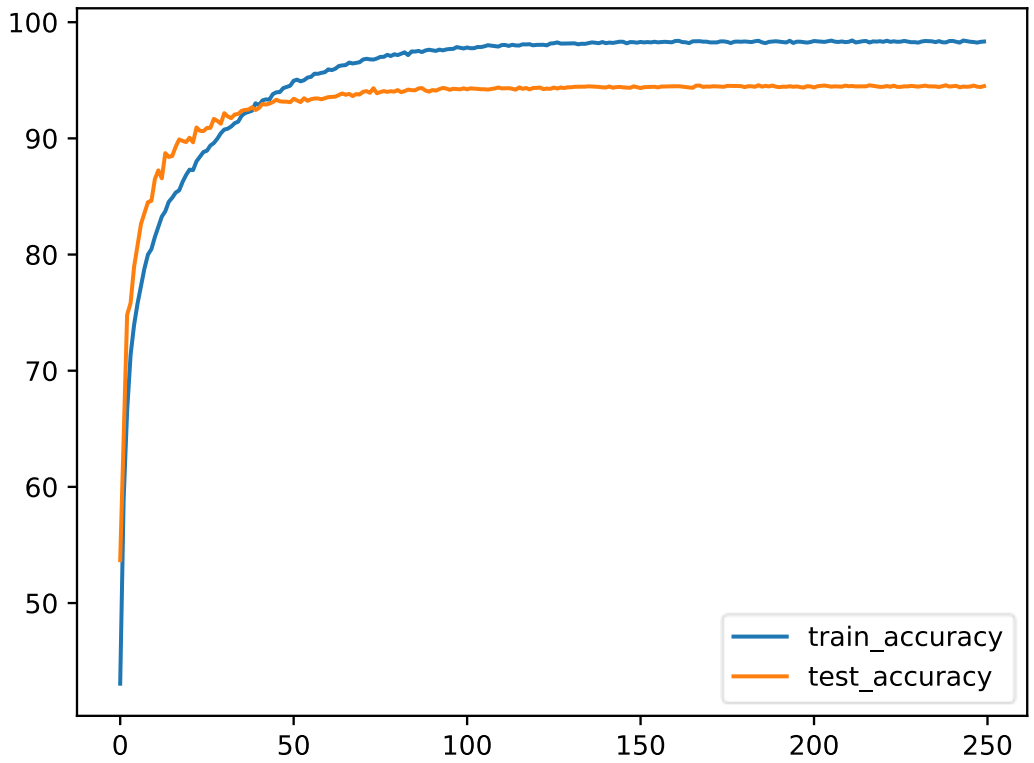
Model Generalization

❖ **Increase model capacity once more**

val_accuracy reaches to ~94.5%

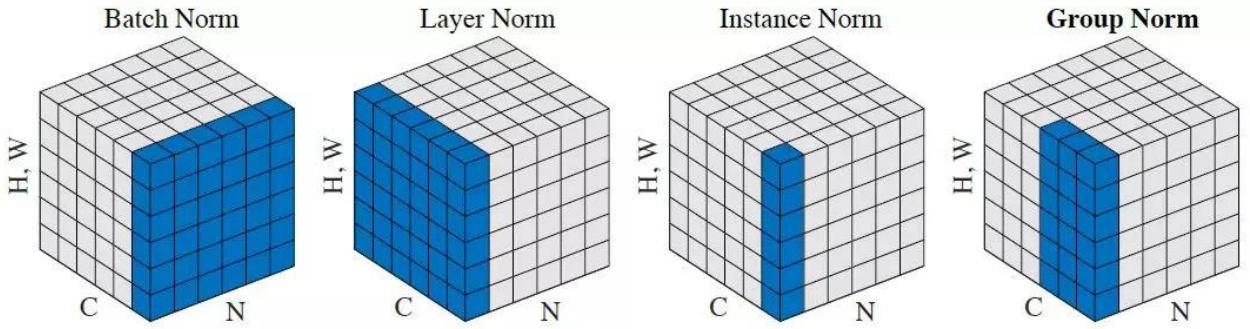
train_accuracy reaches to ~98.3%



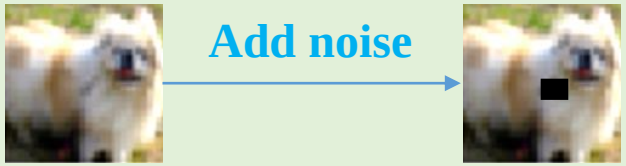


AI VIETNAM
All-in-One Course

Summary

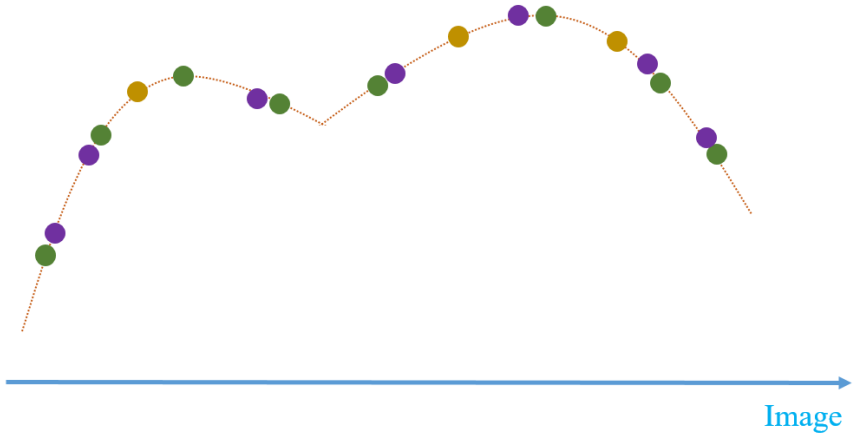


❖ How to increase validation accuracy



Trick 1: 'Learn hard ' – randomly add noise to training data

Increase data by altering the training data

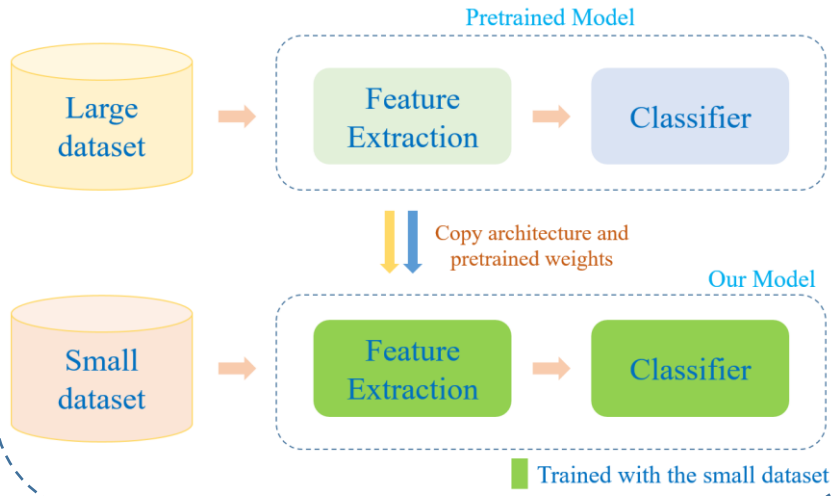


Trick 5: Data augmentation

Trick 2:
Using Batch Normalization

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

(*) Use Pretrained Models



$$L = CE + \underbrace{\lambda \|W\|^2}_{L_2 \text{regularization}}$$

Trick 4: Kernel regularization

Trick 3: Using Dropout