AI VIET NAM
@aivietnam.edu.vn

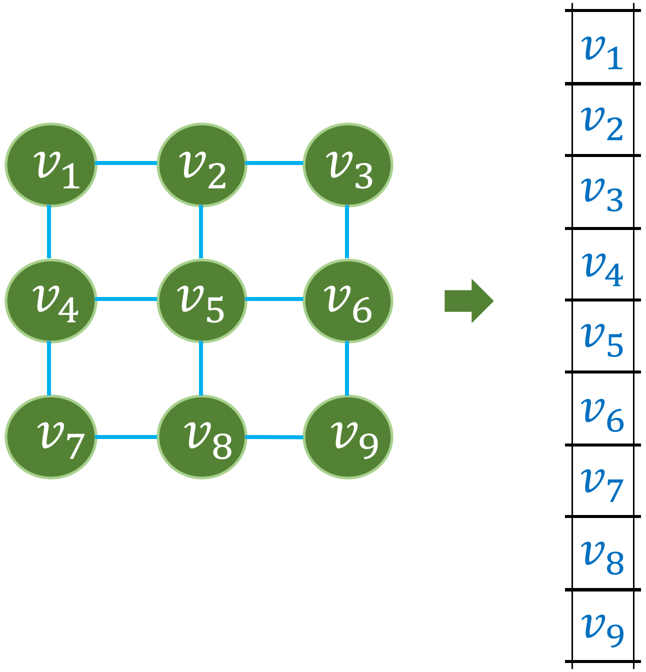
From MLP to CNN

Introduction to CNN

Quang-Vinh Dinh
Ph.D. in Computer Science

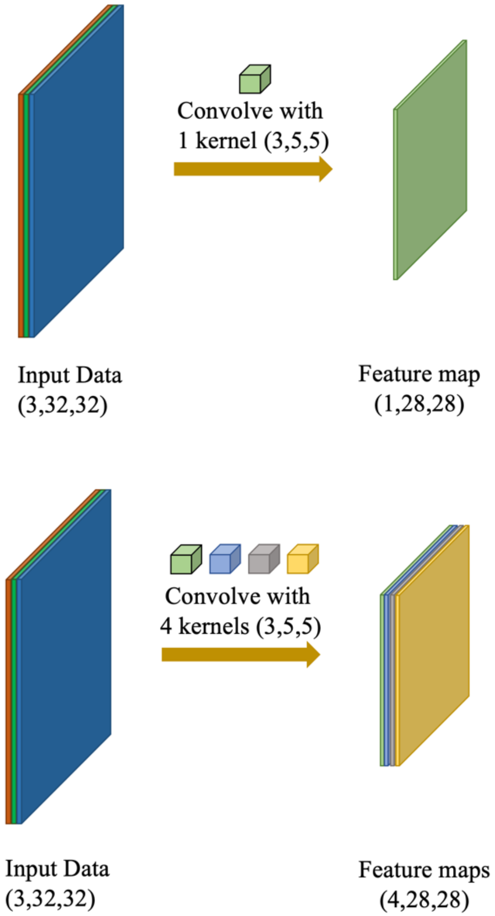
Objectives

MLP Limitations

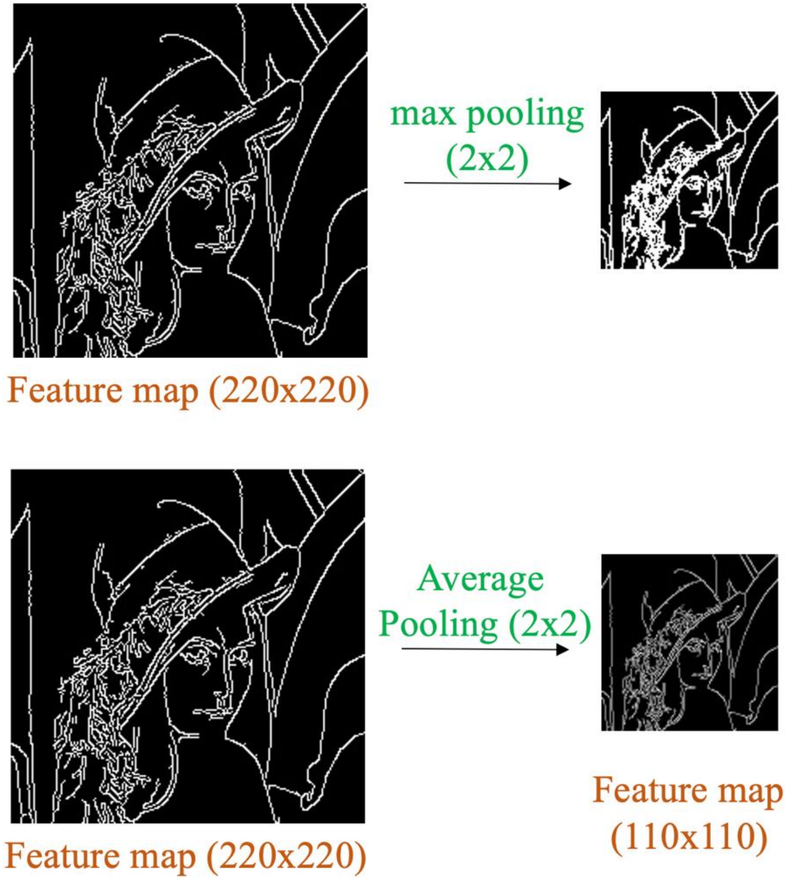


Flattened Data

Convolutional Layer



Standard CNNs



Outline

SECTION 1

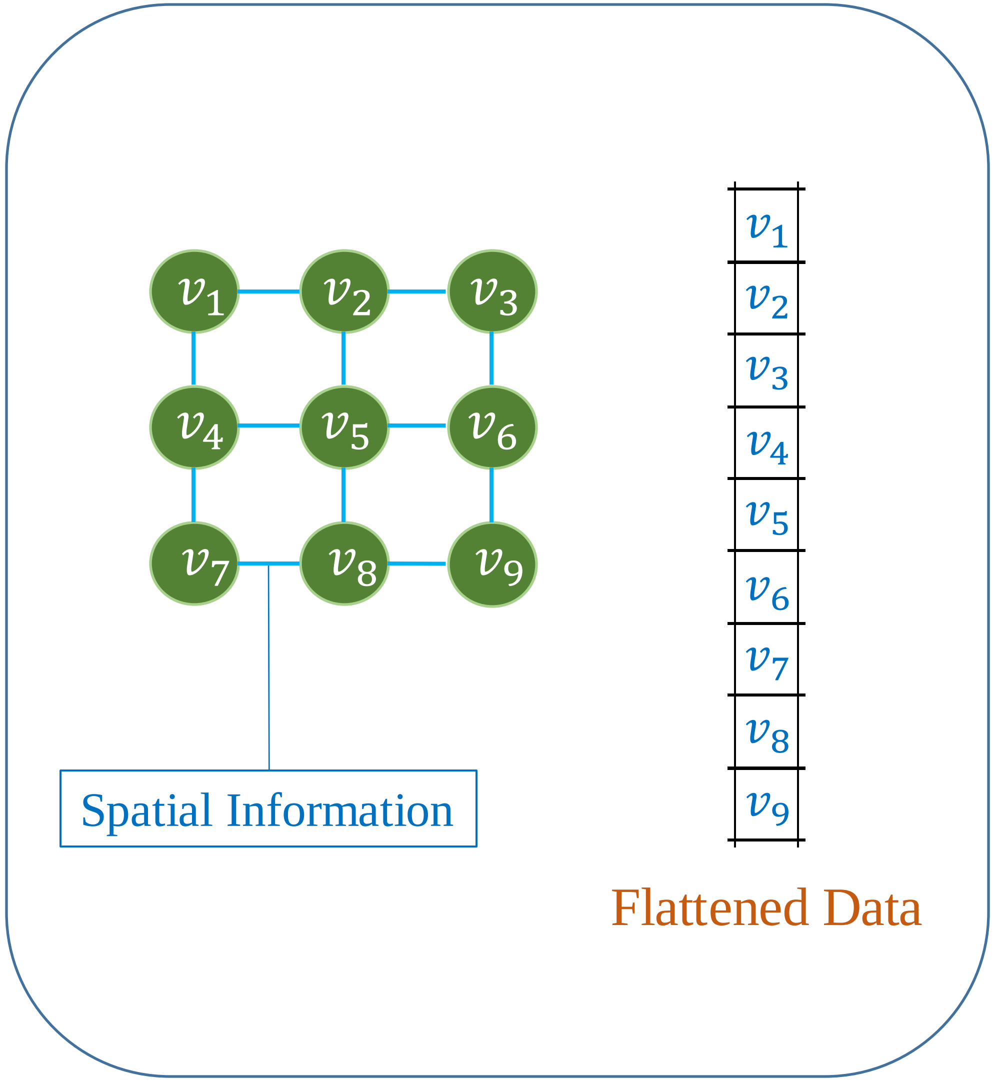
MLP Limitations

SECTION 2

Convolutional Layer

SECTION 3

Standard CNNs



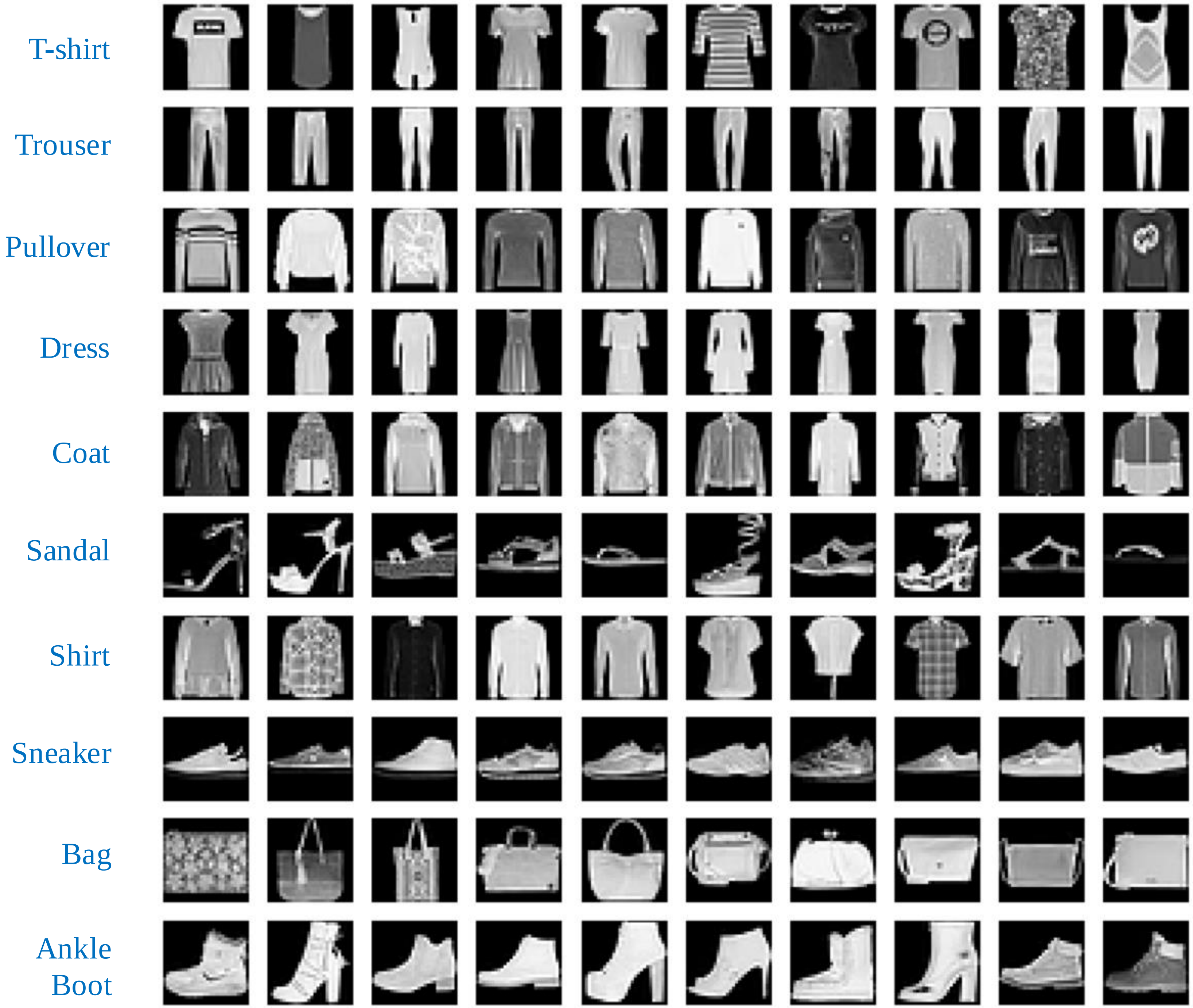
Fashion-MNIST dataset

Grayscale images

Resolution=28x28

Training set: 60000 samples

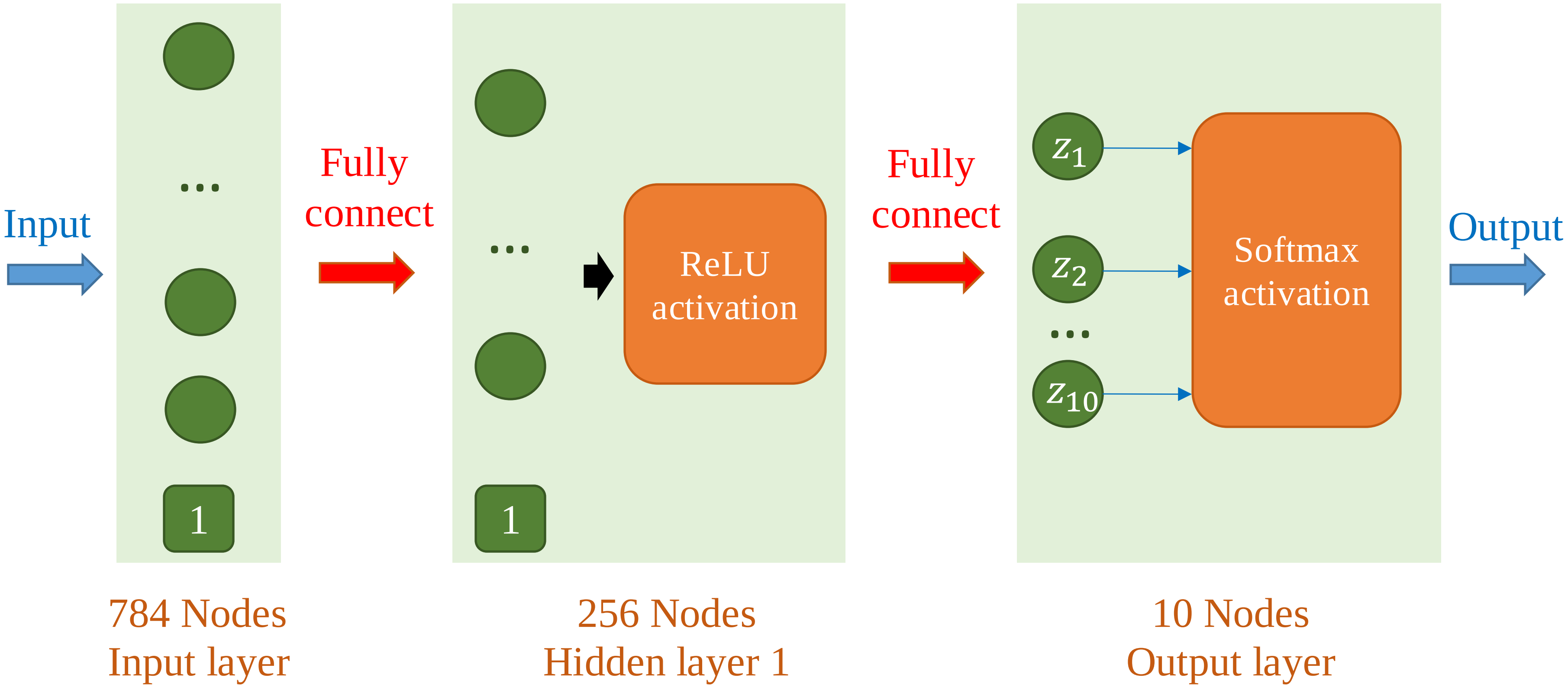
Testing set: 10000 samples



MLP for Fashion-MNIST

❖ ReLU, He and Adam

Case 1



MLP for Fashion-MNIST

Case 1

❖ ReLU, He and Adam

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

```python
# 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)
```

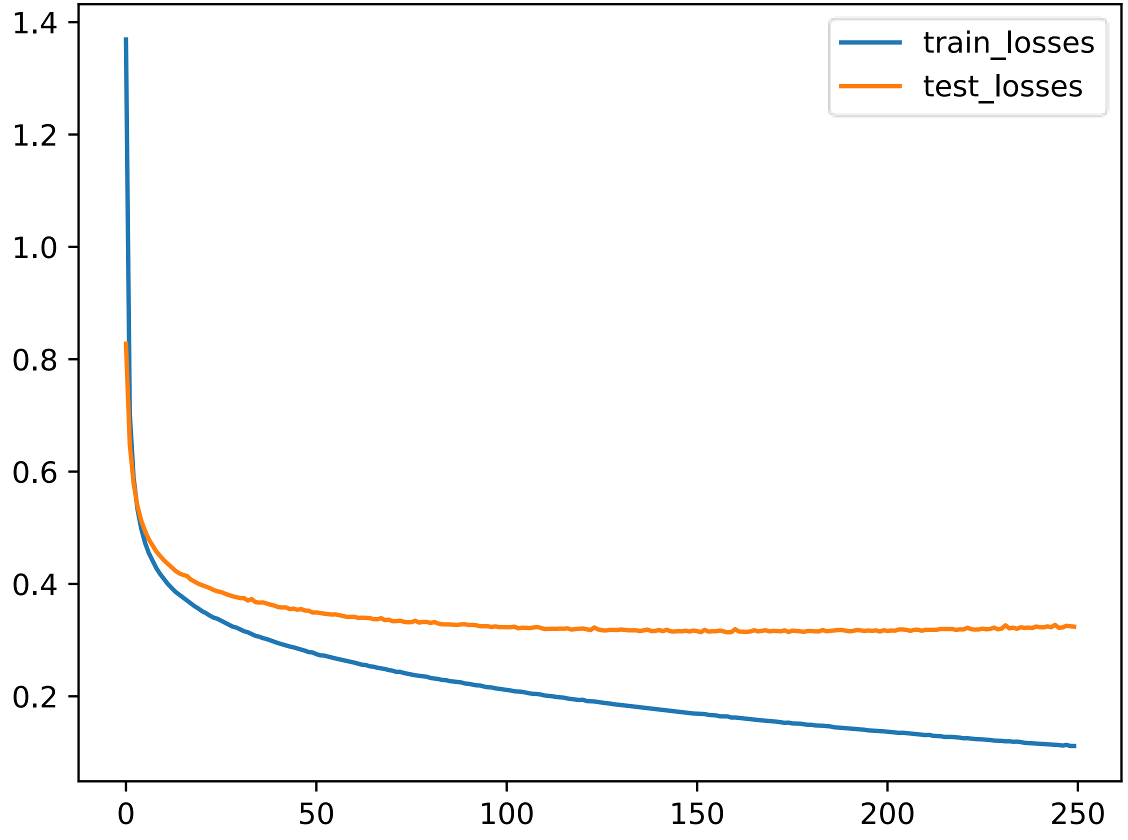
```python
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

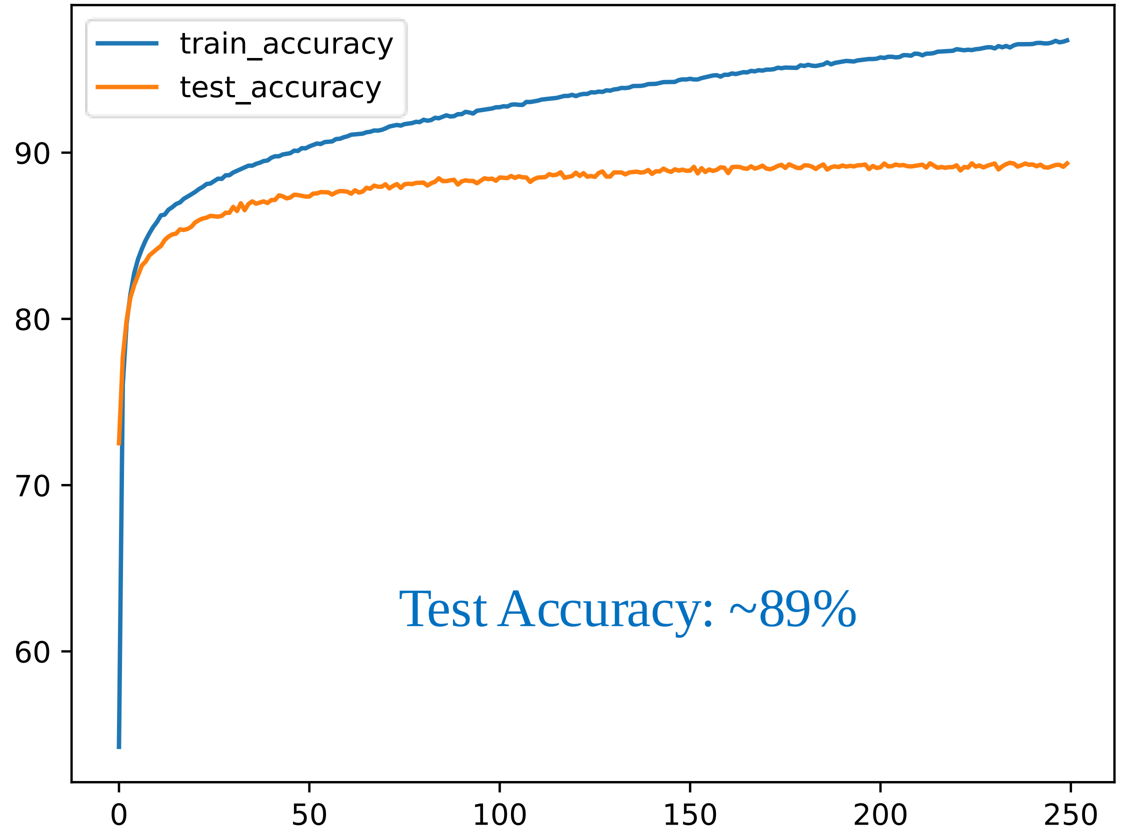
```python
# Load CFashionMNIST dataset
transform = Compose([transforms.ToTensor(),
                     transforms.Normalize((0.5,),
                                          (0.5,))])

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

❖ ReLU, He and Adam

Case 1





Adam with learning rate of 1e-4

Perform reasonably

Cifar-10 dataset

Color images

Resolution=32x32

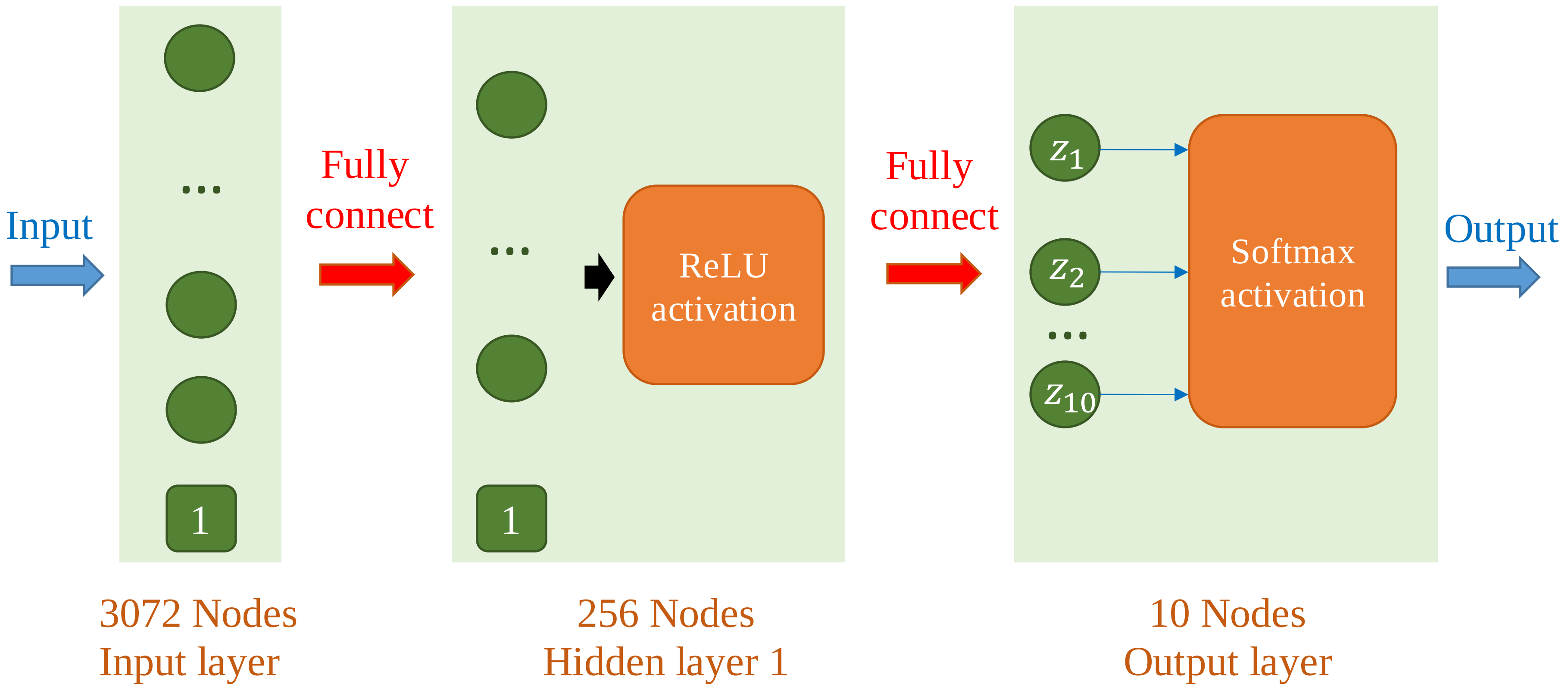
Training set: 50000 samples

Testing set: 10000 samples

MLP for Cifar-10

❖ ReLU, He and Adam

Case 2



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)
```

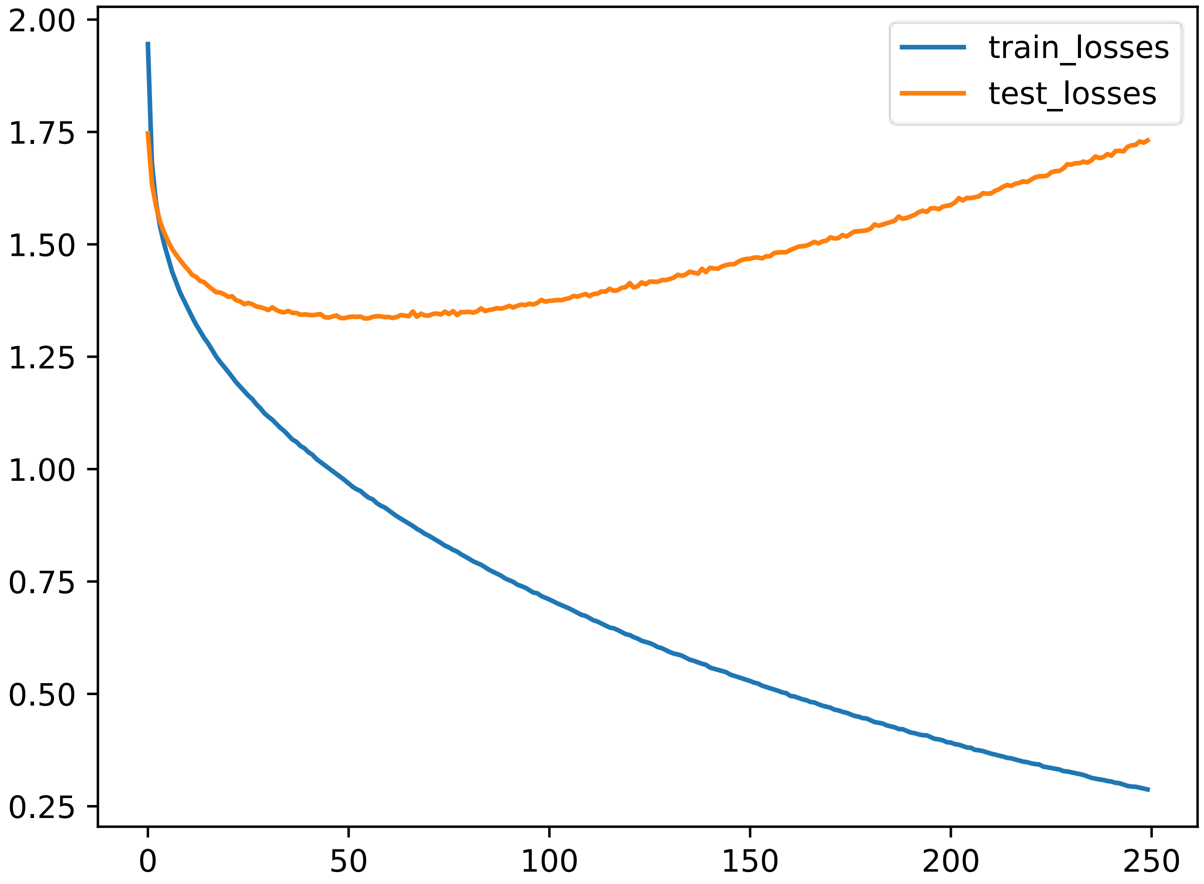
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

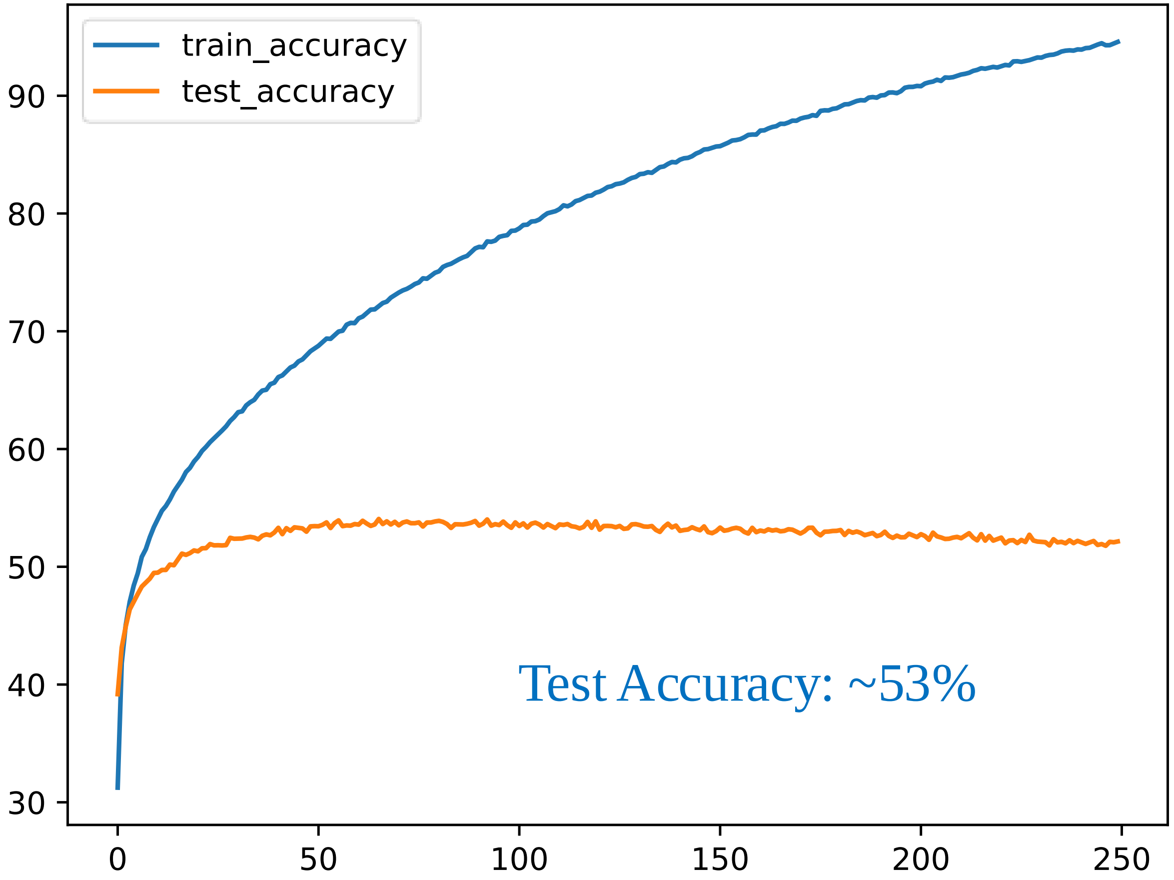
```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

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

❖ ReLU, He and Adam

Case 2



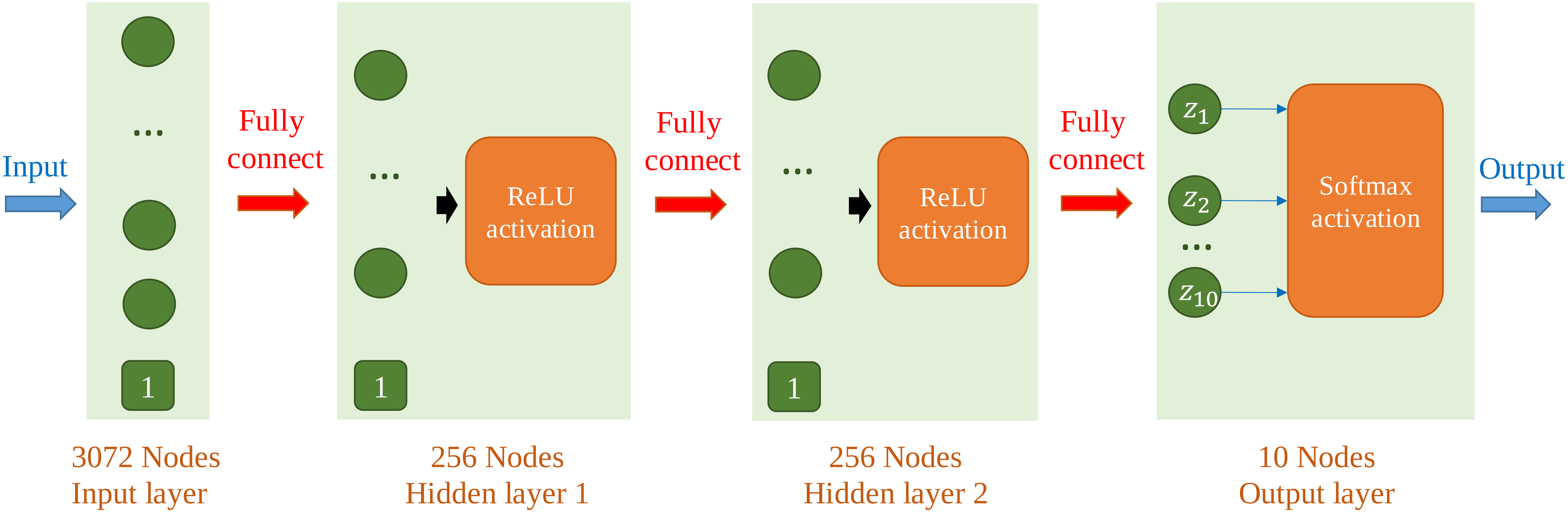


Perform disappointedly

MLP for Cifar-10

❖ ReLU, He and Adam: add more layers

Case 3



MLP for Cifar-10

Case 3

❖ ReLU, He and Adam

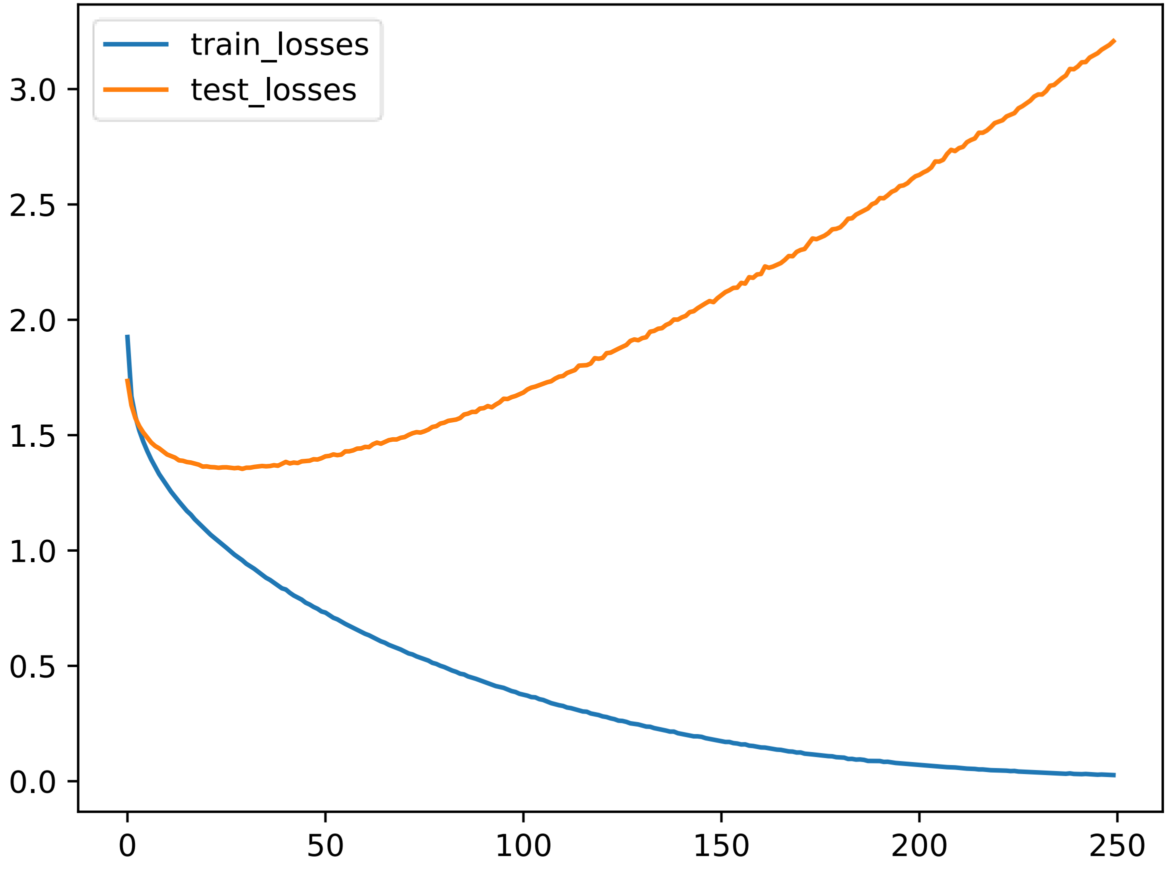
```python
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 256),
    nn.ReLU(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

# 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)
```

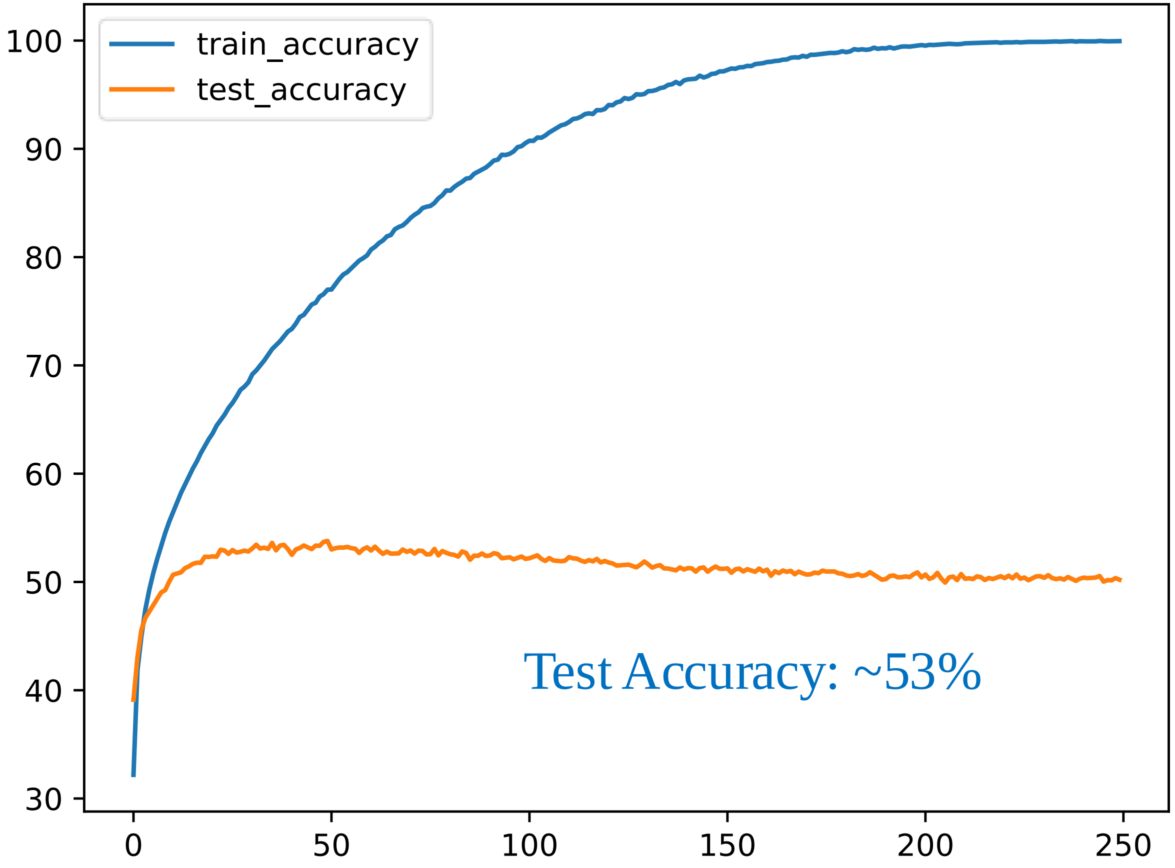
```python
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

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)
```

MLP for Cifar-10

❖ ReLU, He and Adam

Case 3



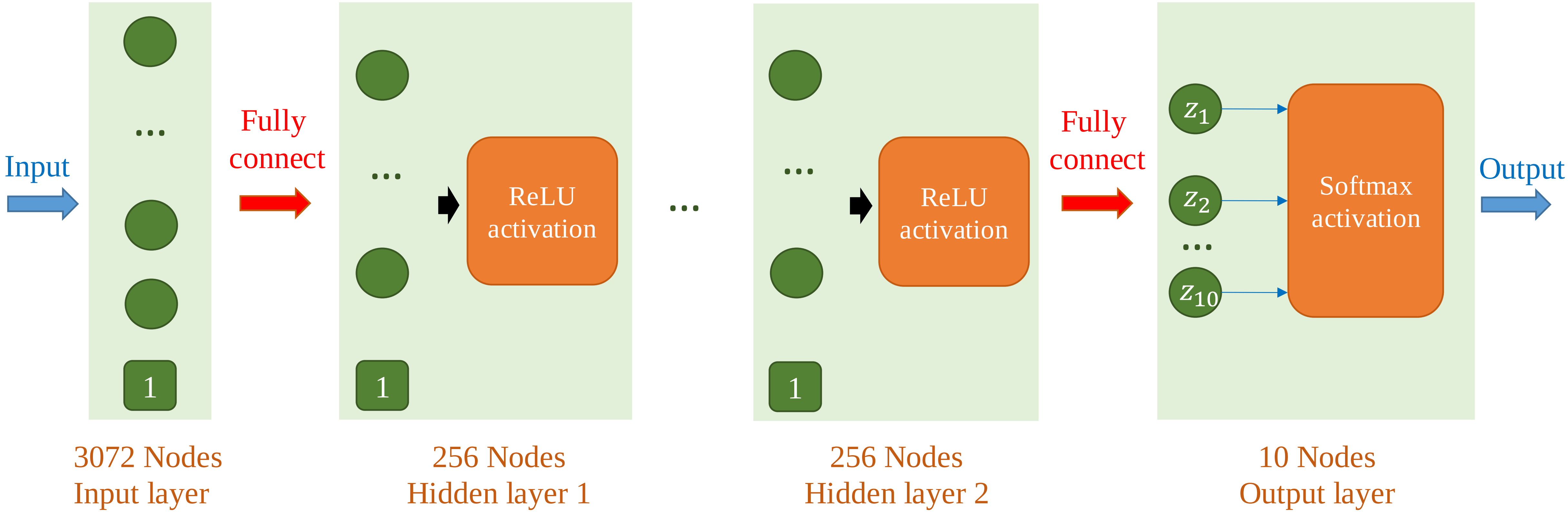


Still Perform poorly

MLP for Cifar-10

❖ ReLU, He and Adam

Case 4



MLP for Cifar-10

Case 4

❖ ReLU, He and Adam

```python
# 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)
)
```

```python
# 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)
```

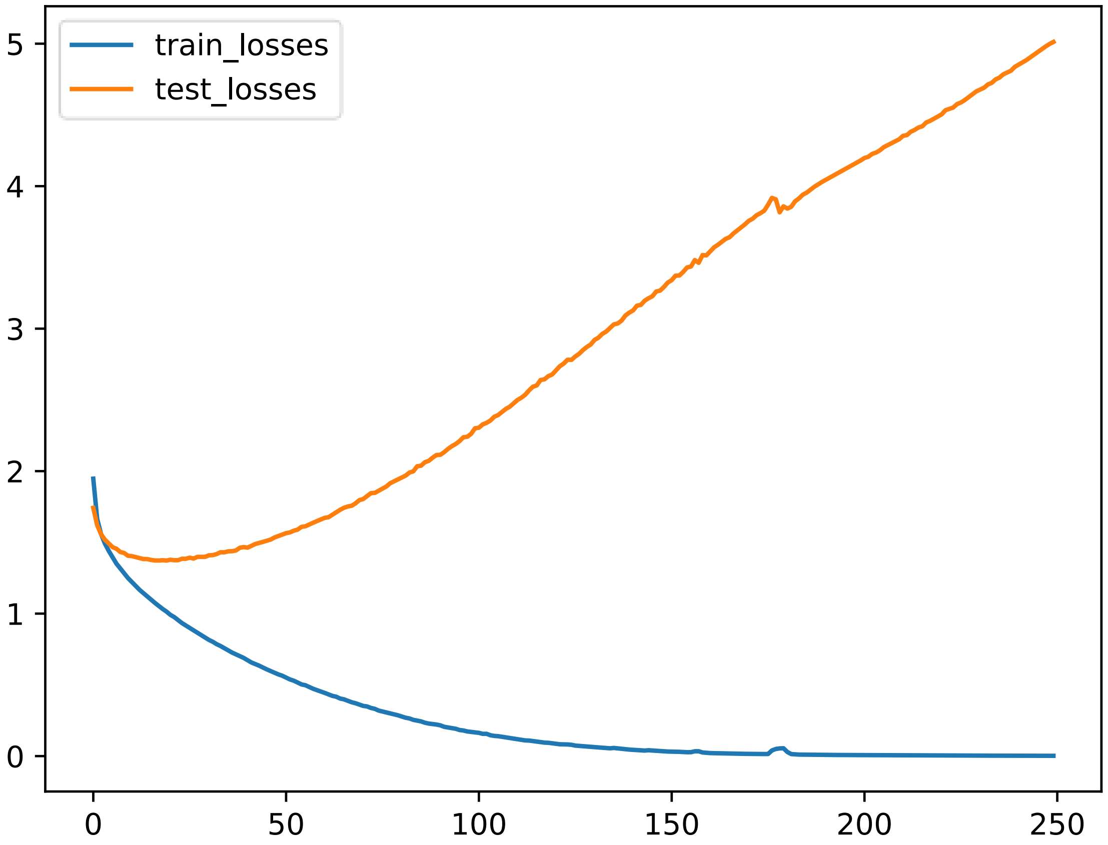
```python
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                       lr=0.001)
```

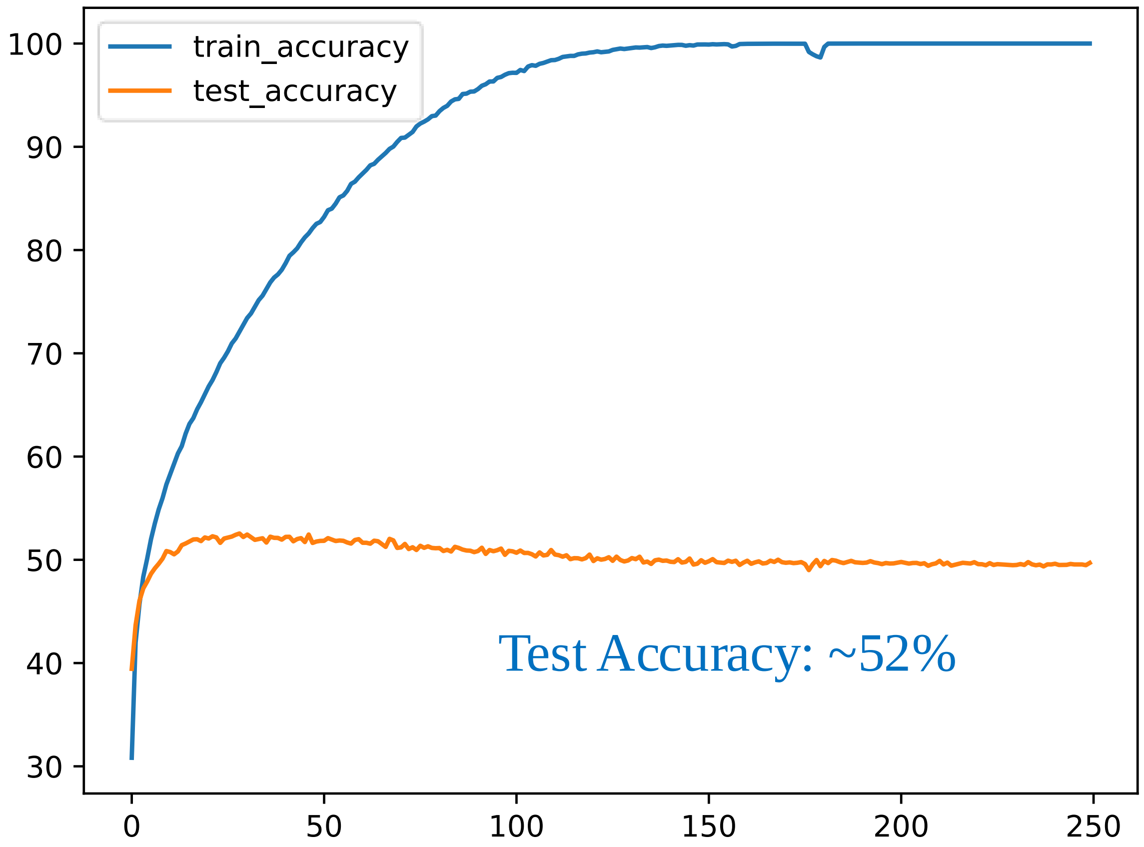
```python
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

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)
```

MLP for Cifar-10

❖ ReLU, He and Adam: Using 3 hidden layers

Case 4

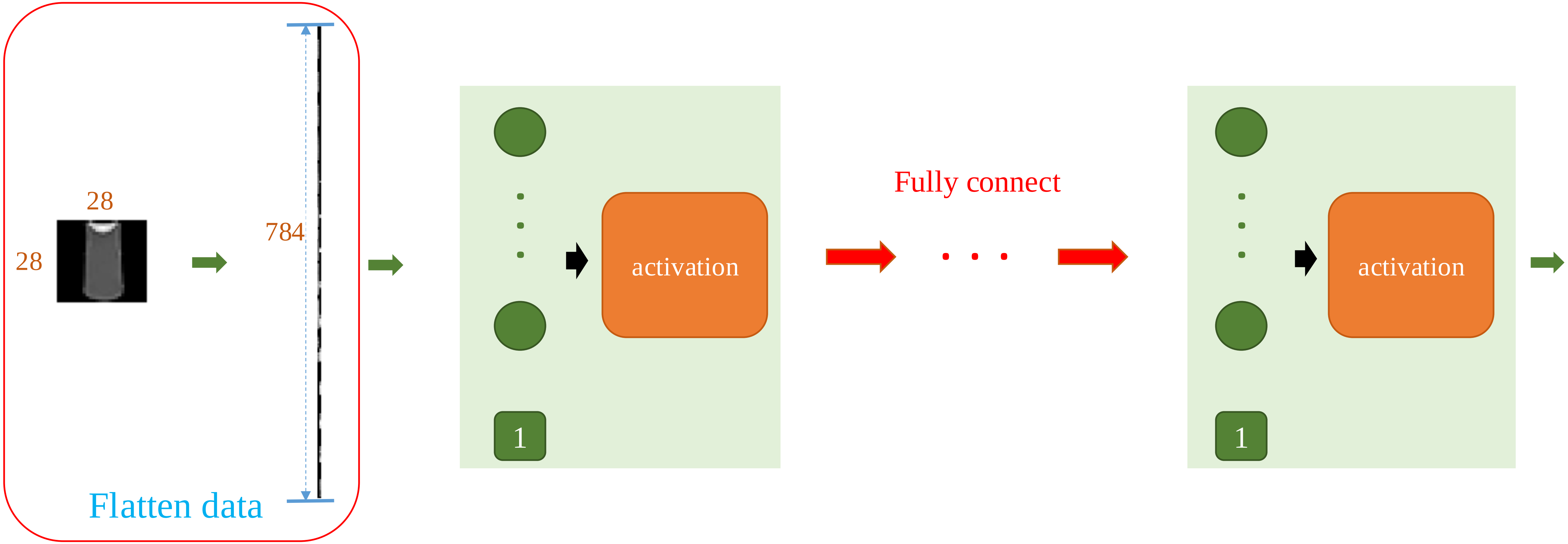




Perform even worse

MLP Limitations

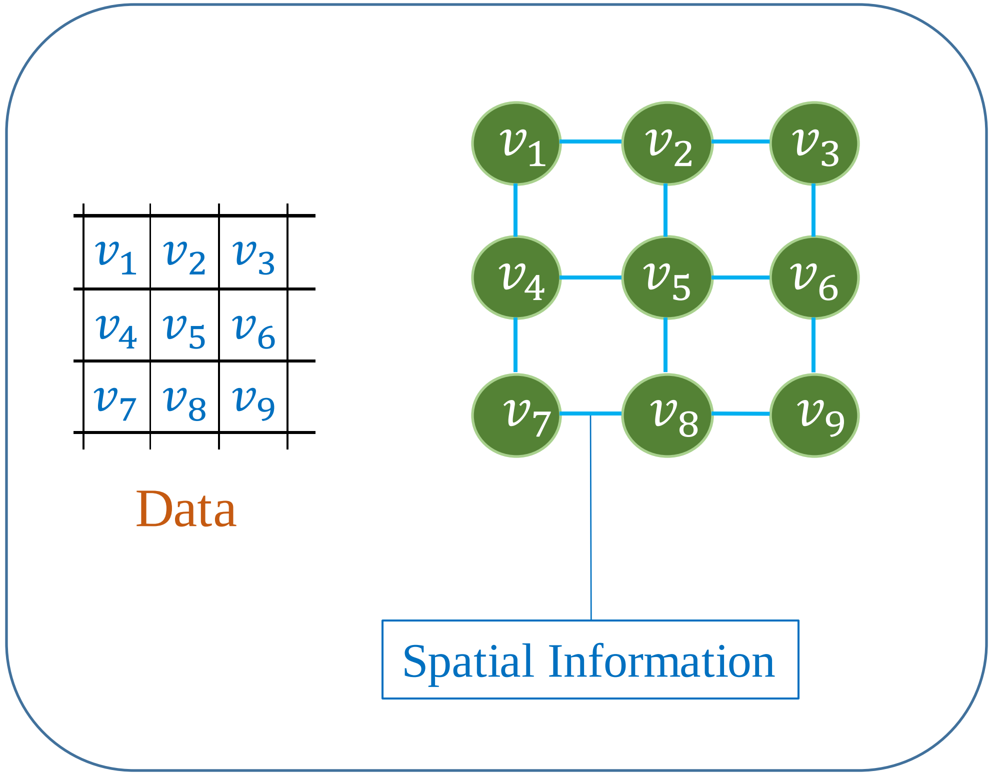
❖ Multi-layer Perceptron

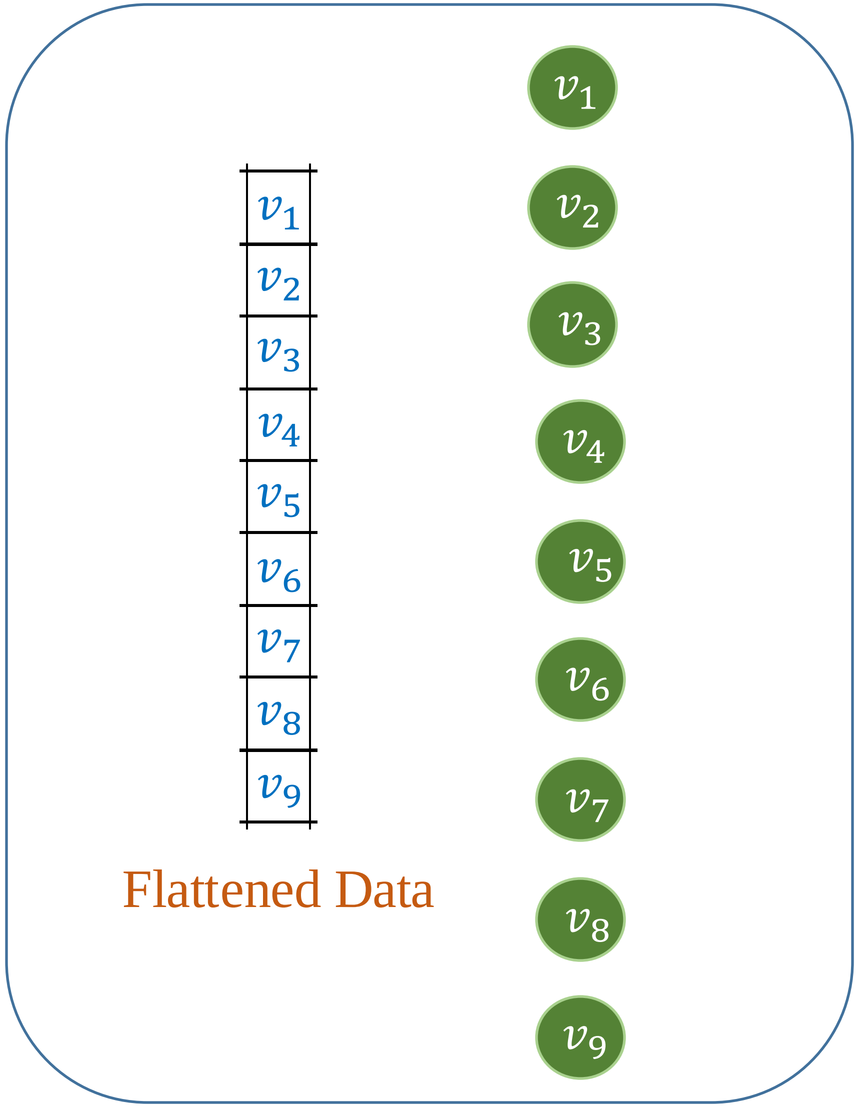


Problem: Remove spatial information of the data
Inefficiently have a large amount of parameters

MLP Limitations

❖ Problem of flattening data





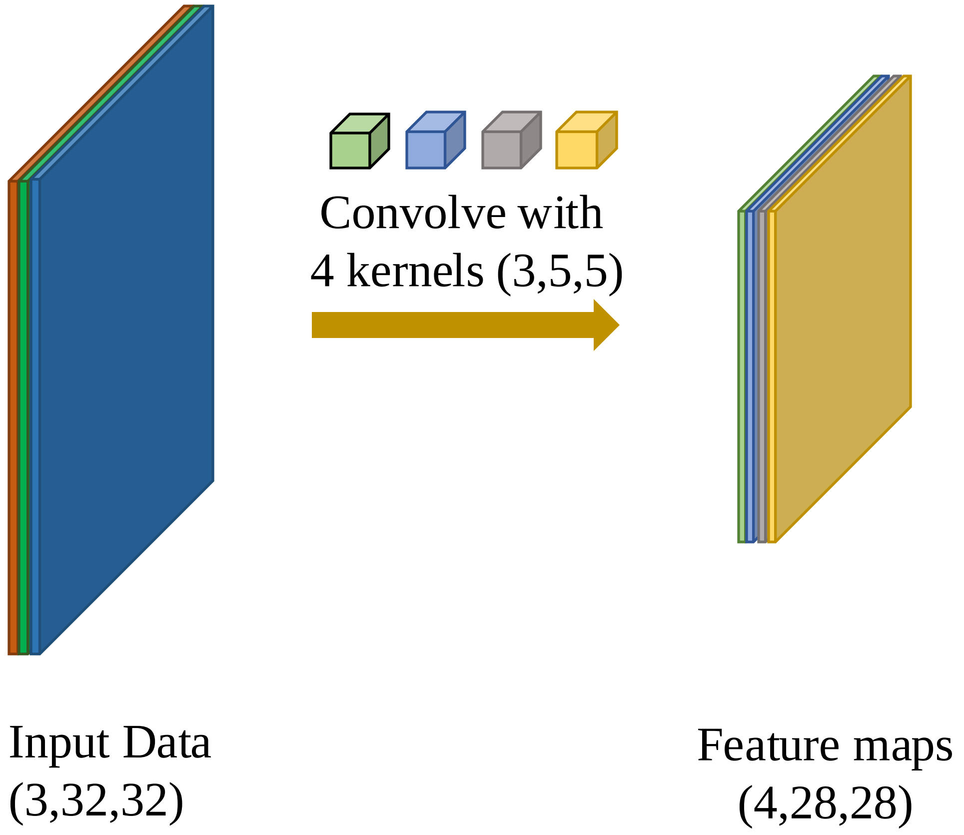
Remove spatial information of the data

Outline

SECTION 1
MLP Limitations

SECTION 2
Convolutional Layer

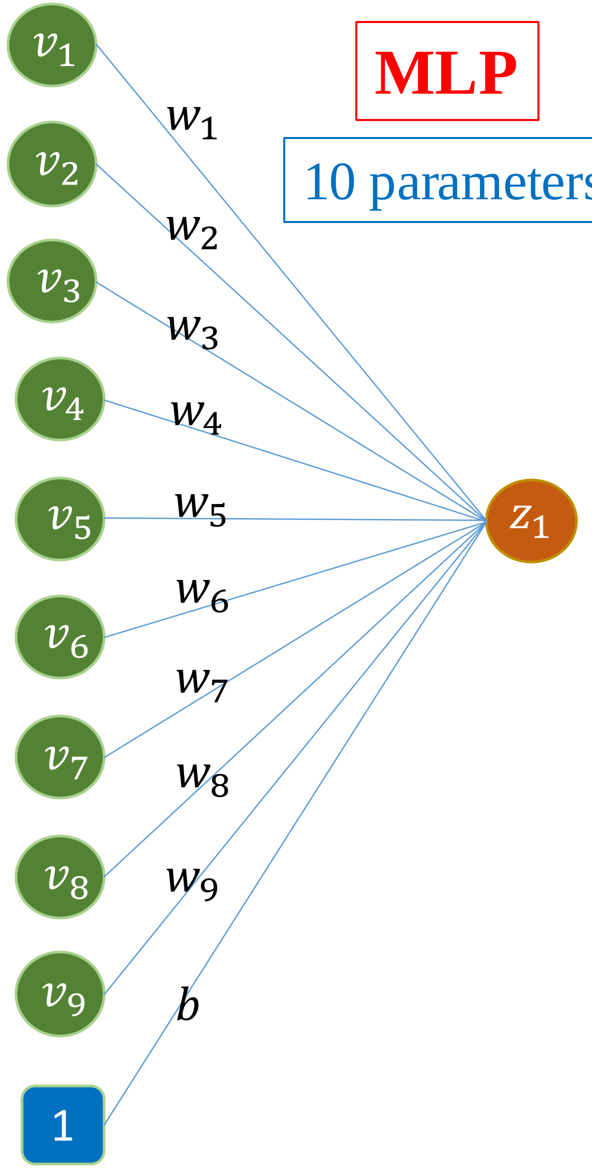
SECTION 3
Standard CNNs



From MLP to CNN

MLP

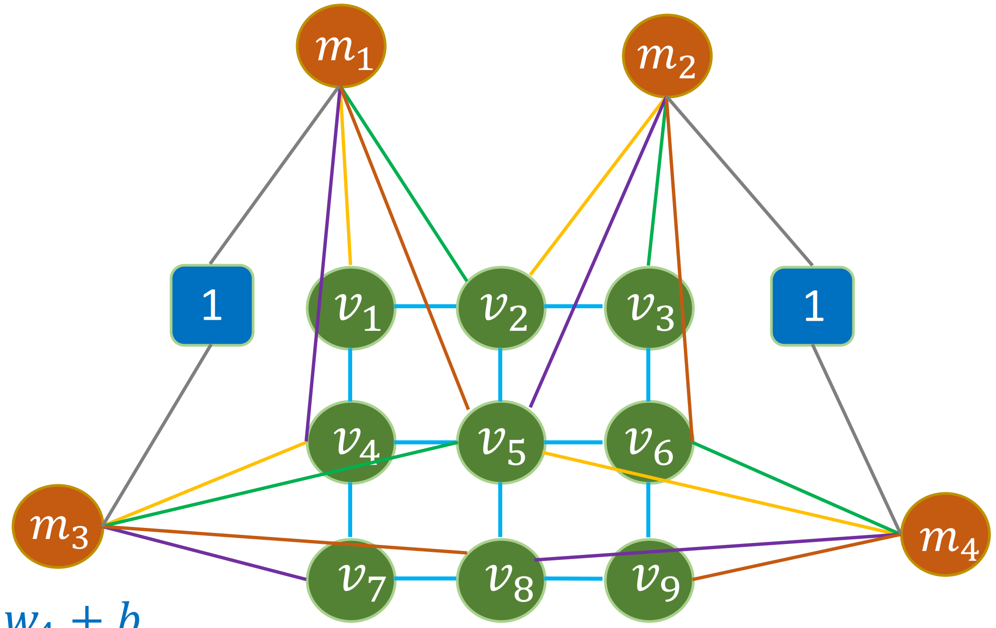
10 parameters



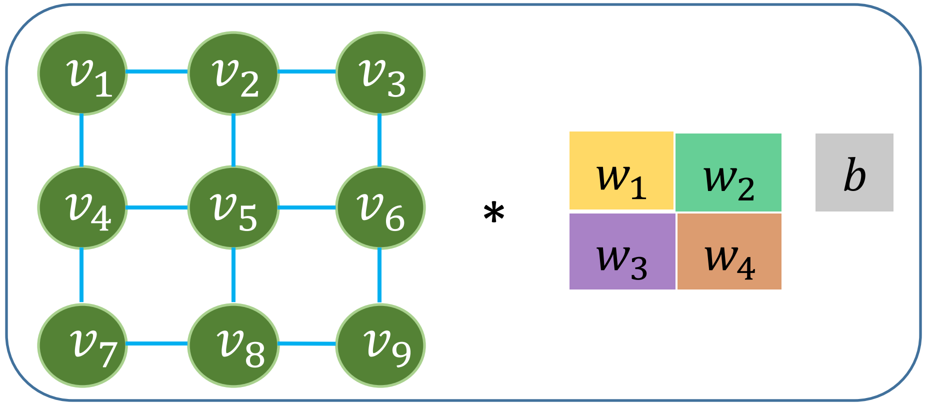
CNN

5 parameters

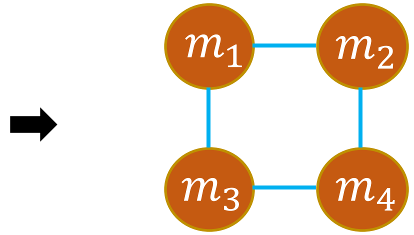
Kernel of parameters



$$m_1 = v_1w_1 + v_2w_2 + v_4w_3 + v_5w_4 + b$$



Convolution (=Correlation)

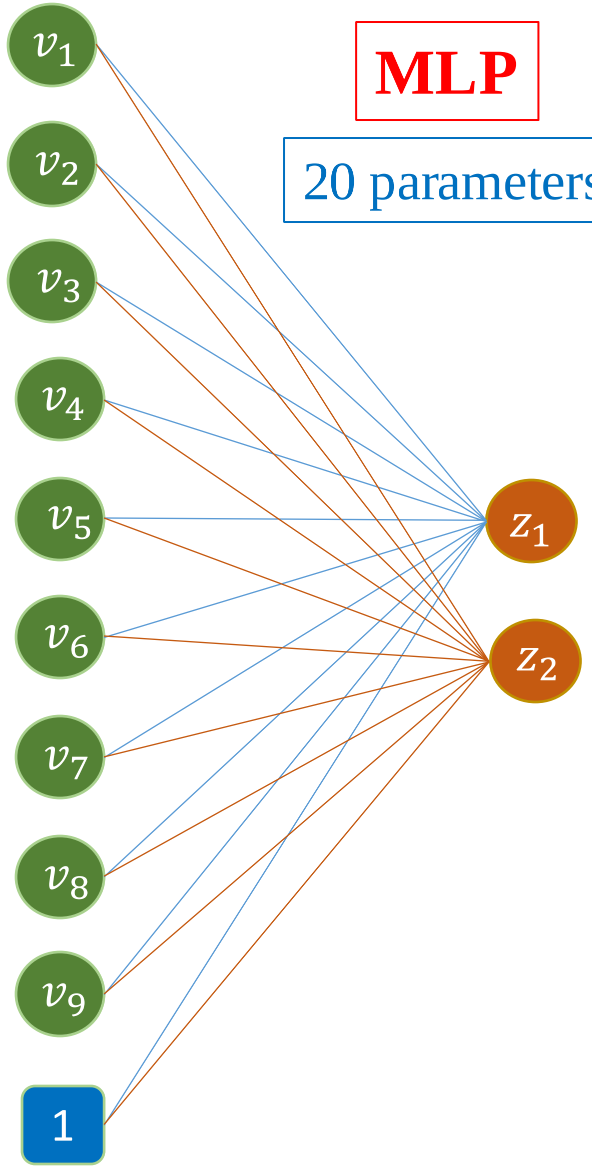


Feature Map

From MLP to CNN

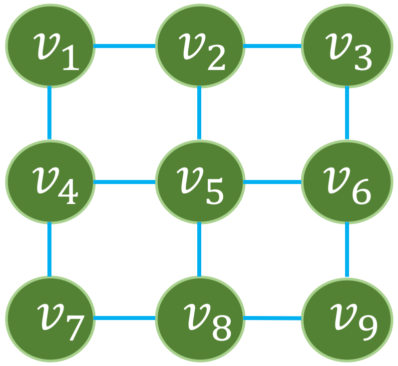
MLP

20 parameters

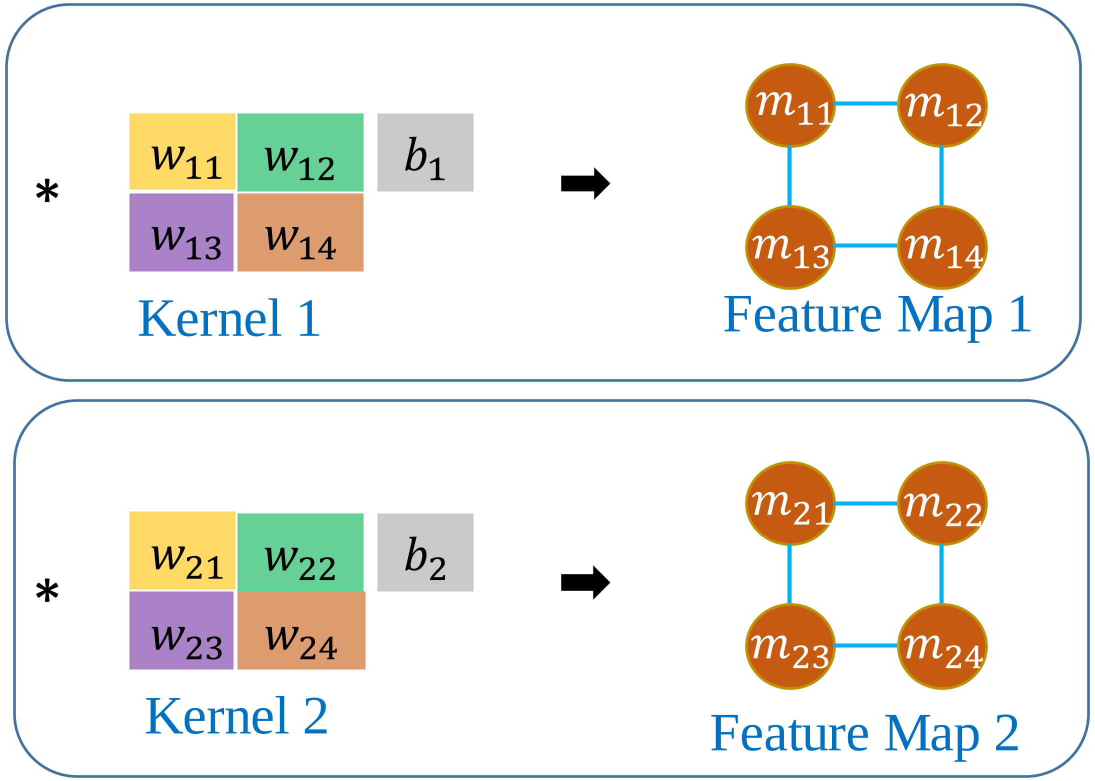


CNN

10 parameters



Kernel 1 $\neq$ Kernel 2

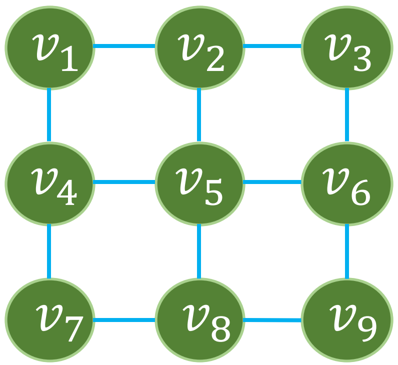


Global vs. Local?

Parameter size?

From MLP to CNN

❖ Understand convolution



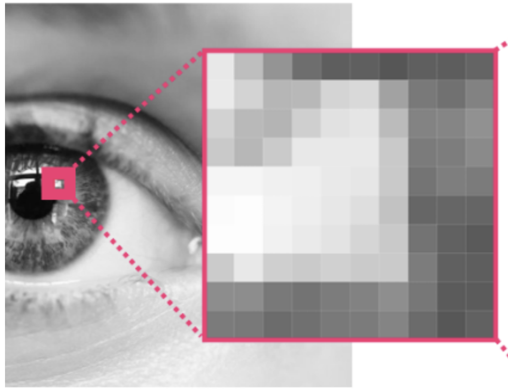
Shape=(1,3,3)

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

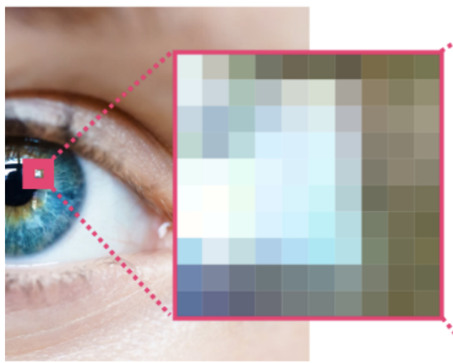
*

Shape=(1,2,2)

#parameters (+bias) = 5



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97



226	191	147	110	101	112	98	123	110	119
221	191	176	182	203	214	169	144	133	145
185	160	161	184	205	223	186	137	147	161
181	174	189	207	206	215	194	136	142	151
246	237	237	231	208	206	192	122	143	144
254	254	241	224	199	192	181	99	122	117
239	248	232	207	187	182	184	110	114	110
193	215	193	167	158	164	181	114	112	111
113	119	110	111	113	123	135	120	108	106
93	97	91	103	107	111	122	112	104	114

#channels of data $\overset{\text{must}}{=}$ #channels of kernel

Convolutional Layer

❖ **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

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

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

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

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

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

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

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

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

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

Convolutional Layer

❖ 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

$$m_1 = 0\times0.0 + 0\times0.1 + 1\times-0.1 + 1\times-0.2 + 2\times0.0 + 2\times0.1 + 0\times0.0 + 2\times0.0 + 0\times0.1$$

➡ $m_1 = -0.1$

Convolutional Layer

❖ 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

Convolutional Layer

❖ 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

$$m_1 = 0\times0.0 + 0\times0.1 + 1\times-0.1 + 1\times-0.2 + 2\times0.0 + 2\times0.1 + 0\times0.0 + 2\times0.0 + 0\times0.1$$

➡ $m_1 = -0.1$

Convolutional Layer

❖ 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

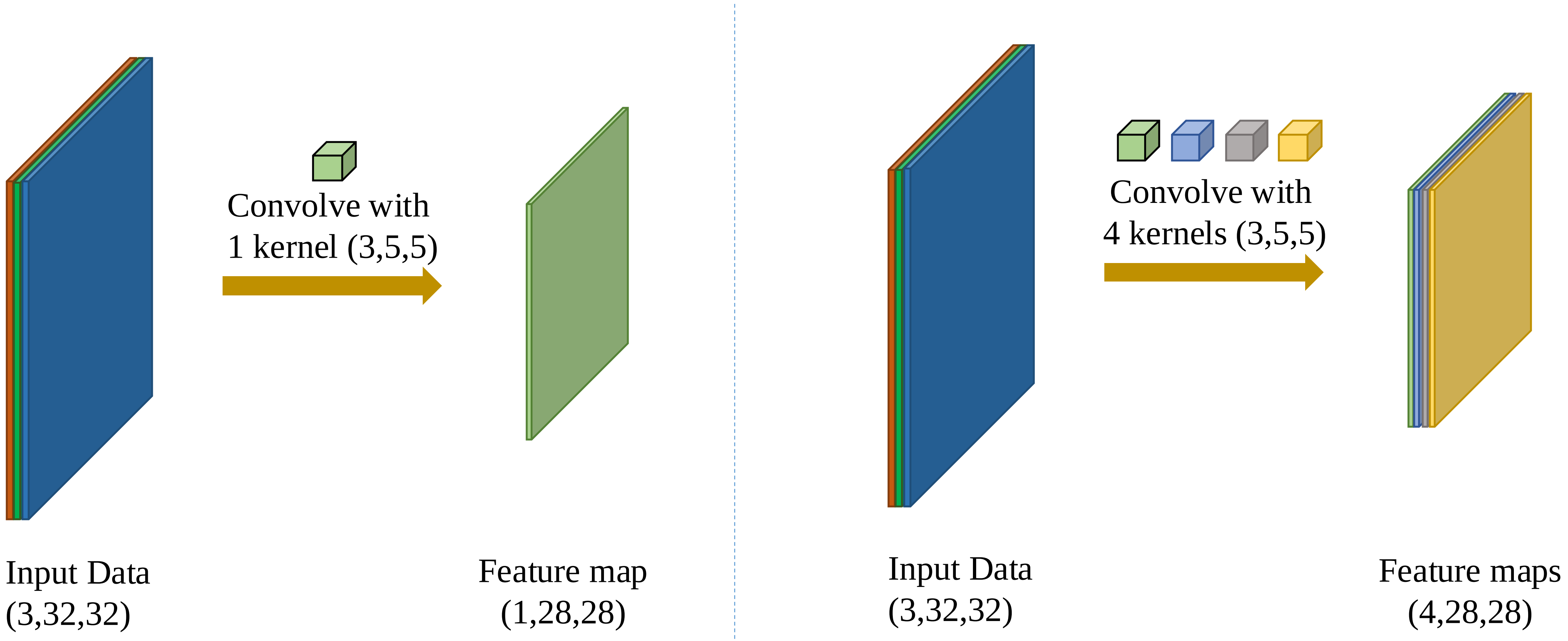
Data size = 5×5

Kernel size = 3×3

Stride = 2

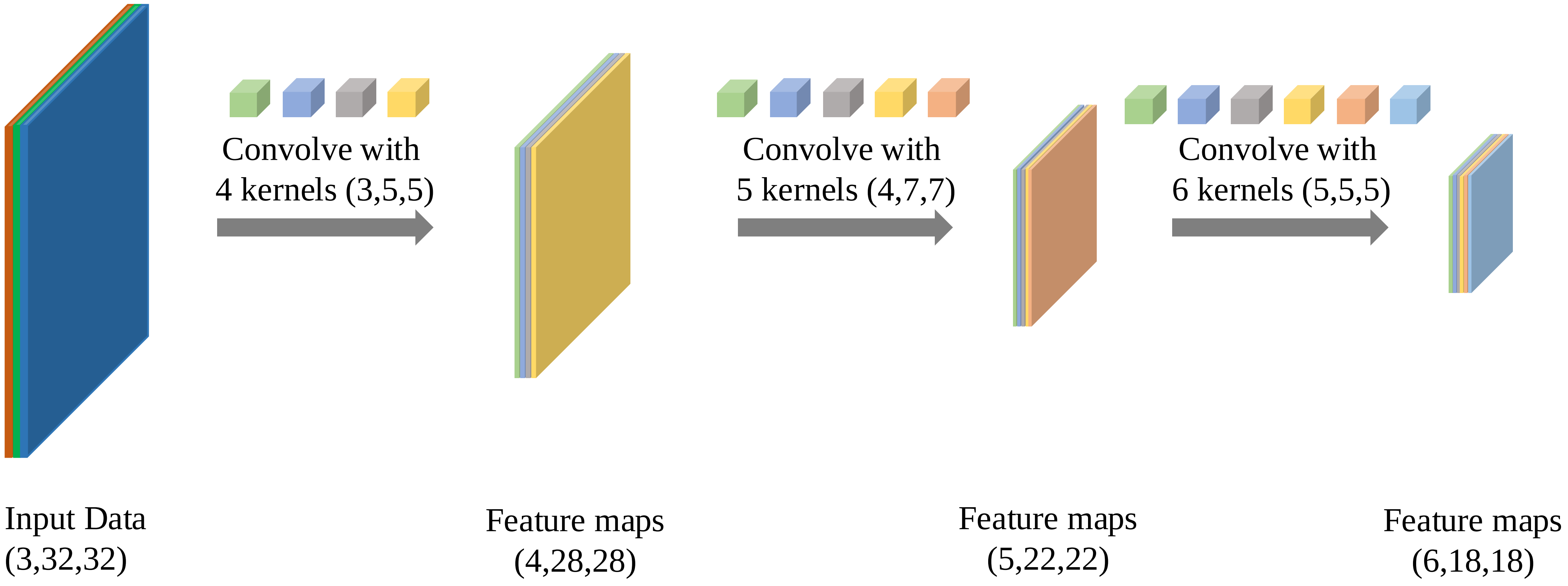
Convolutional Neural Network

❖ Understand convolutional layers



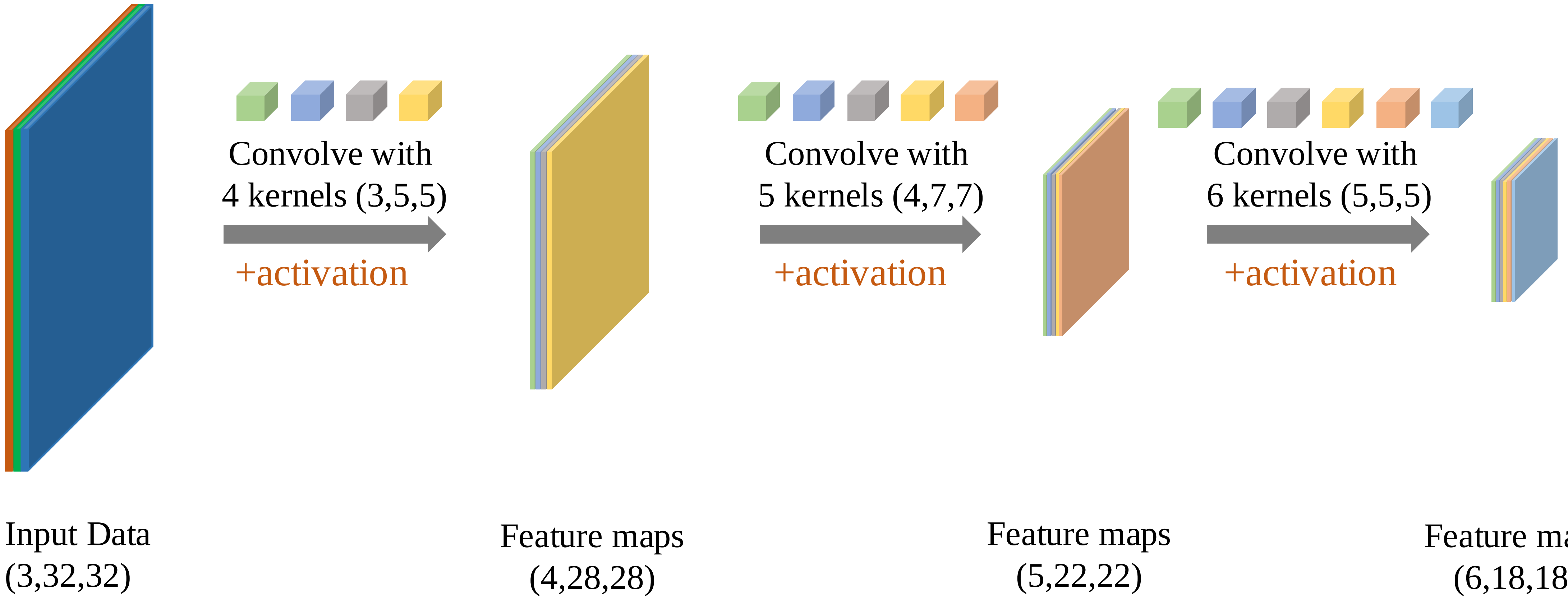
Convolutional Neural Network

❖ A stack of convolutional layers



Convolutional Neural Network

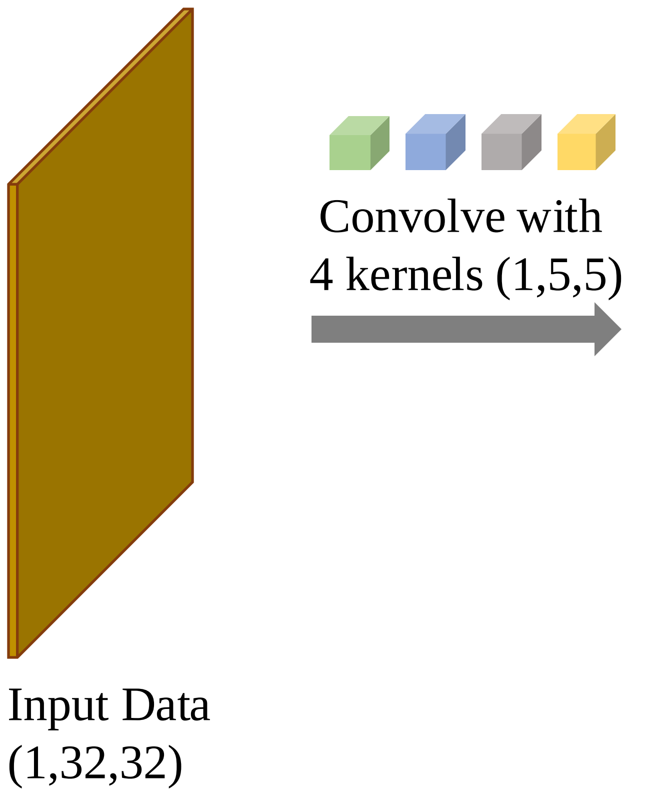
❖ A stack of pairs of convolutional layer + activation

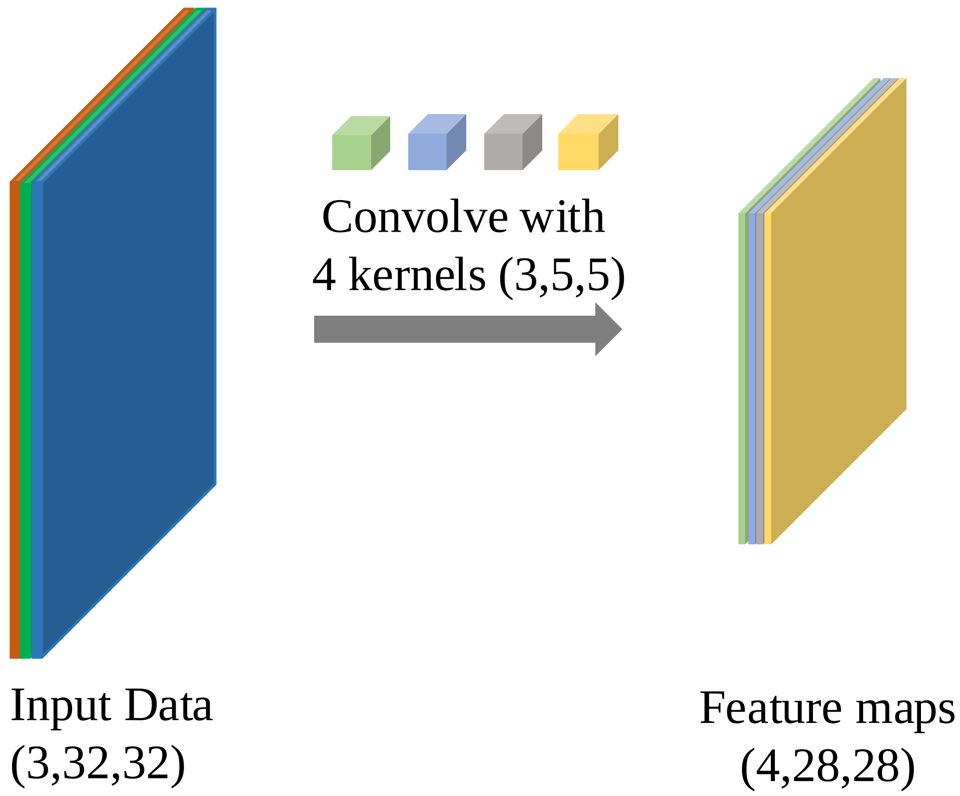


Convolutional Neural Network

❖ Convolutional layer in PyTorch

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



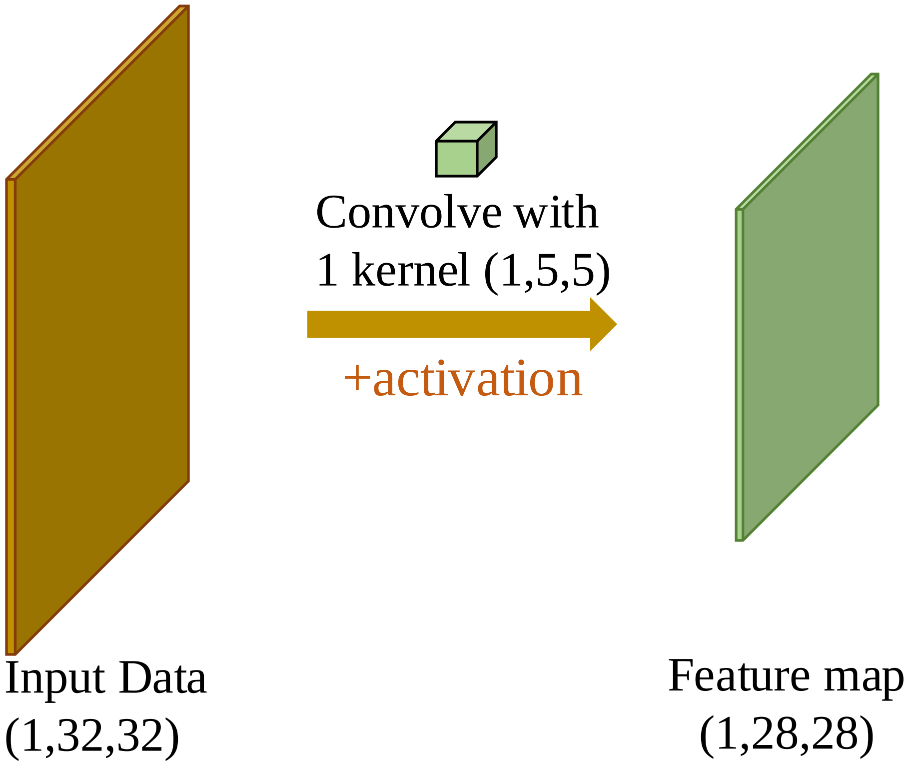


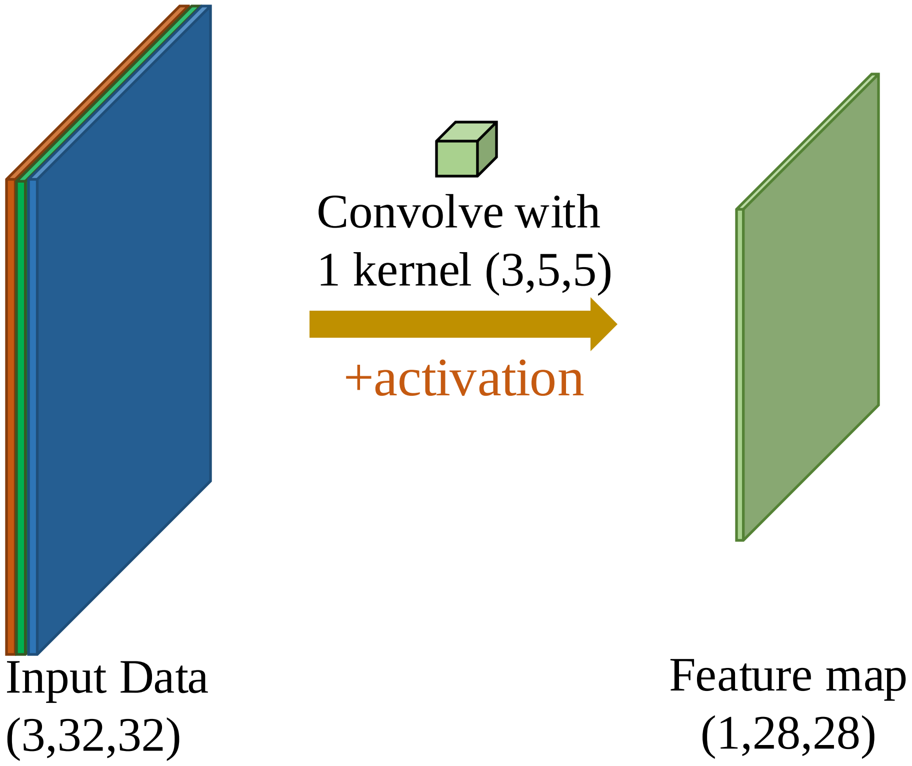
Convolutional Neural Network

❖ Convolutional 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

T-shirt

Trouser

Pullover

Dress

Coat

Sandal

Shirt

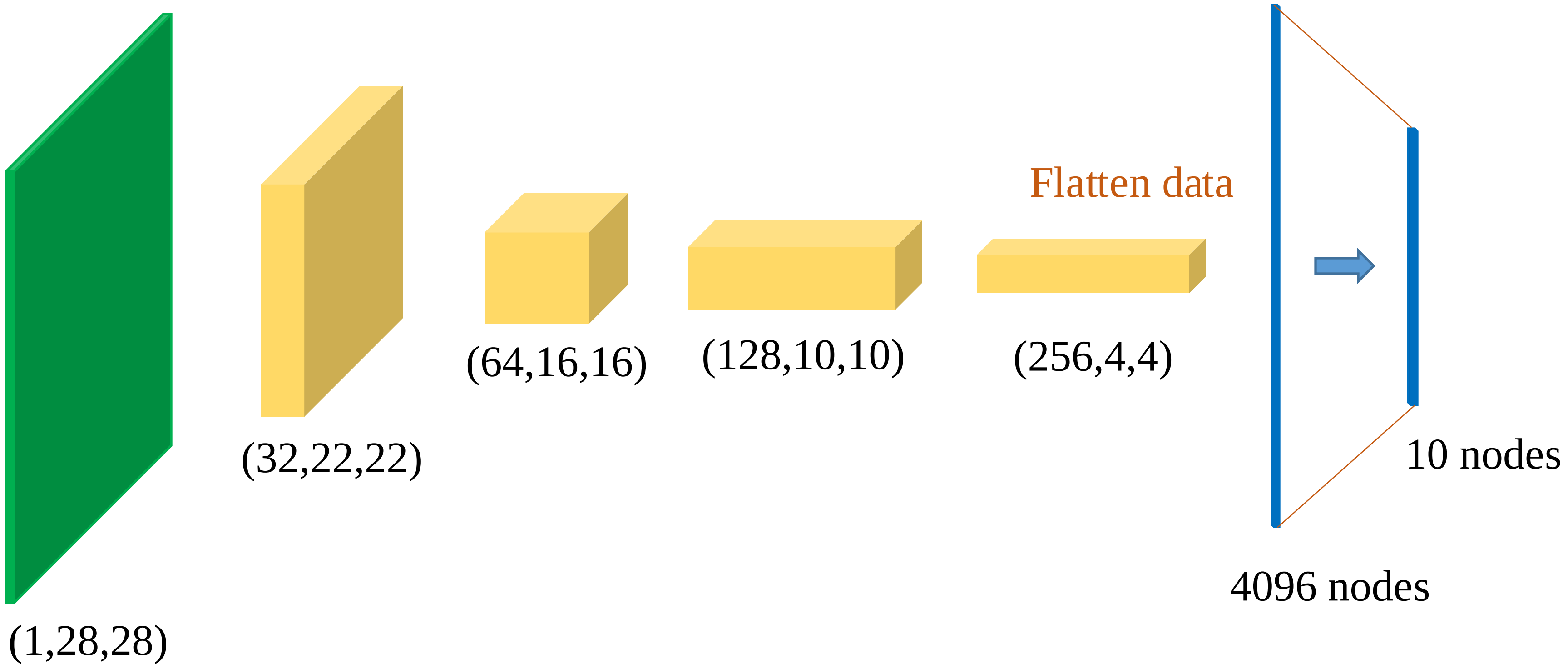
Sneaker

Bag

Ankle Boot

Convolutional Neural Network

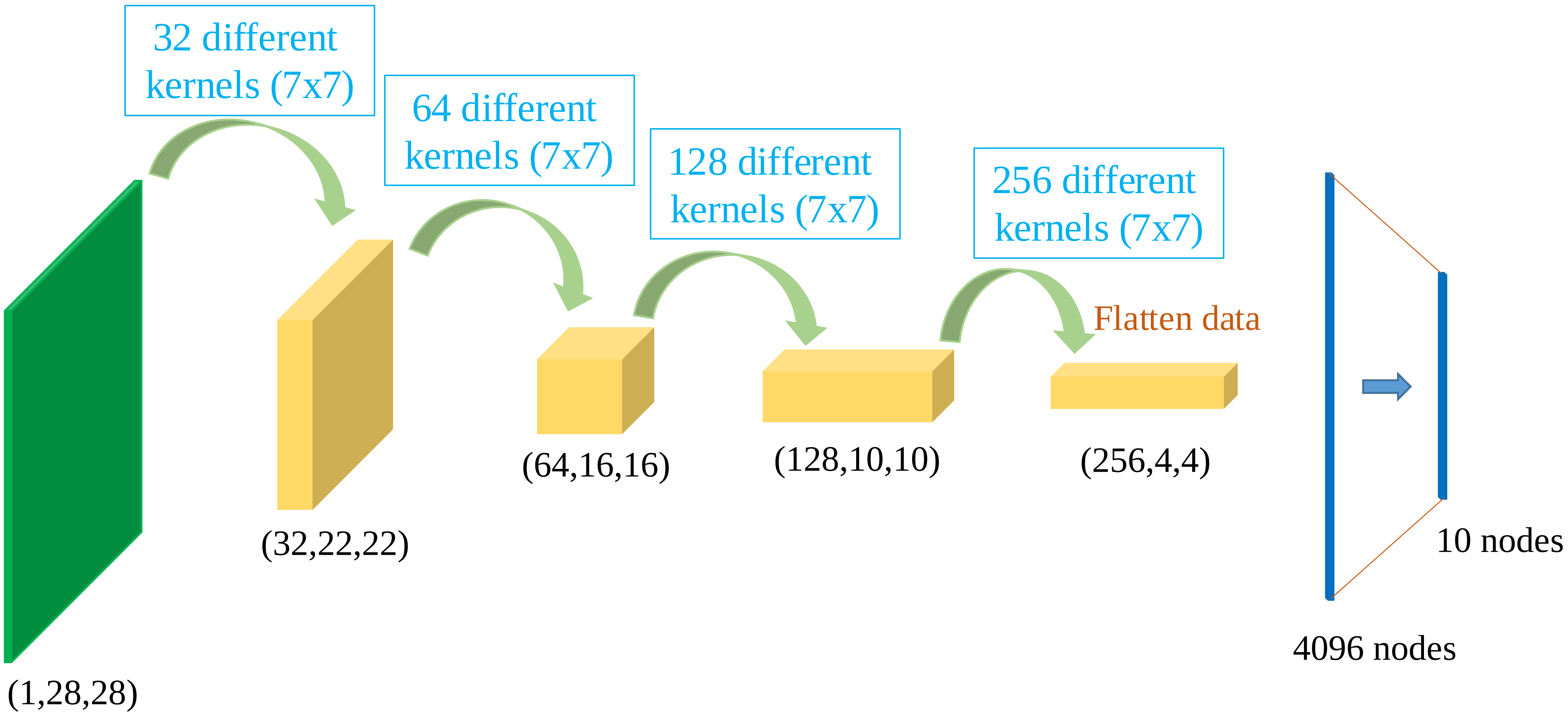
❖ Apply for Fashion-MNIST dataset



Convolutional Neural Network

❖ Apply for Fashion-MNIST dataset

demo



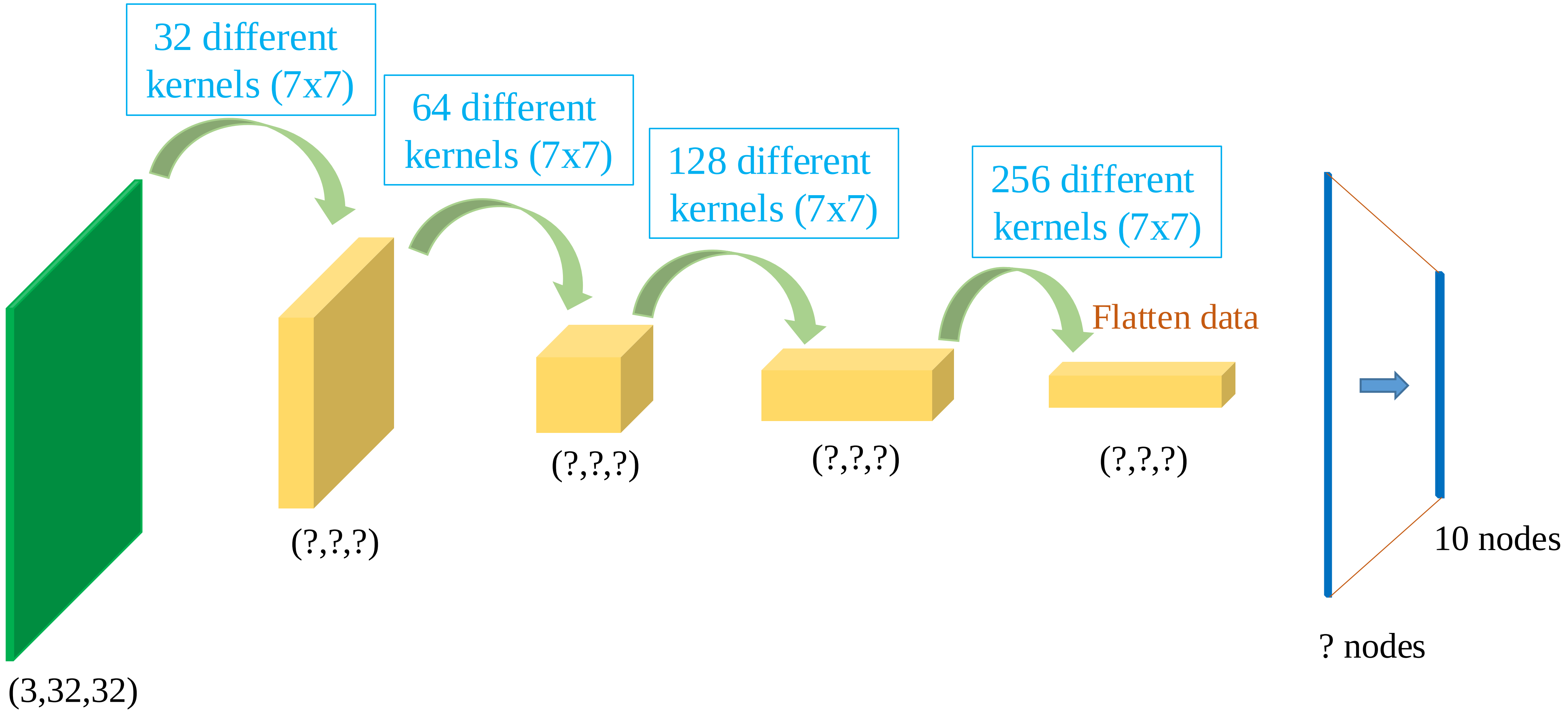
Simple Convolutional Neural Network

```
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()
```

Convolutional Neural Network

❖ Apply for Cifar-10 dataset

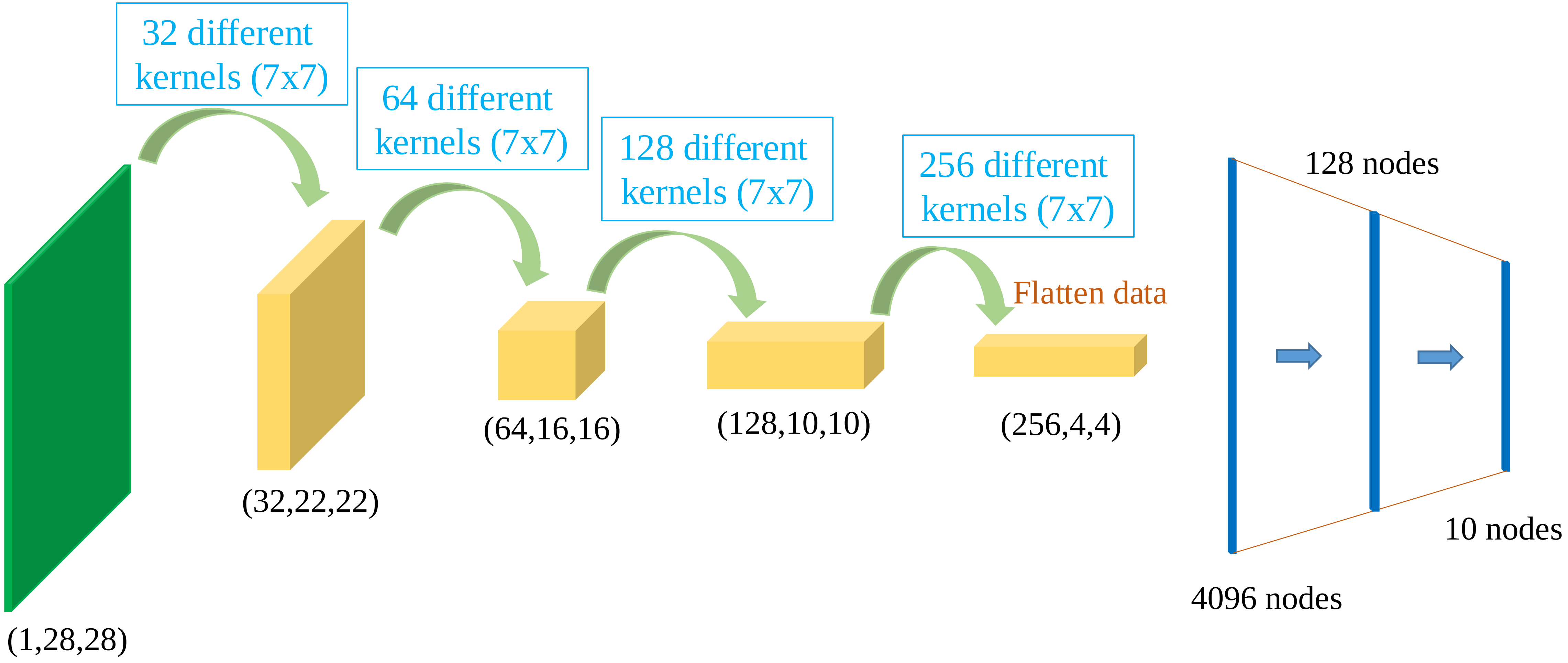
demo



More Examples

Convolutional Neural Network

❖ Apply for Fashion-MNIST dataset: case 1

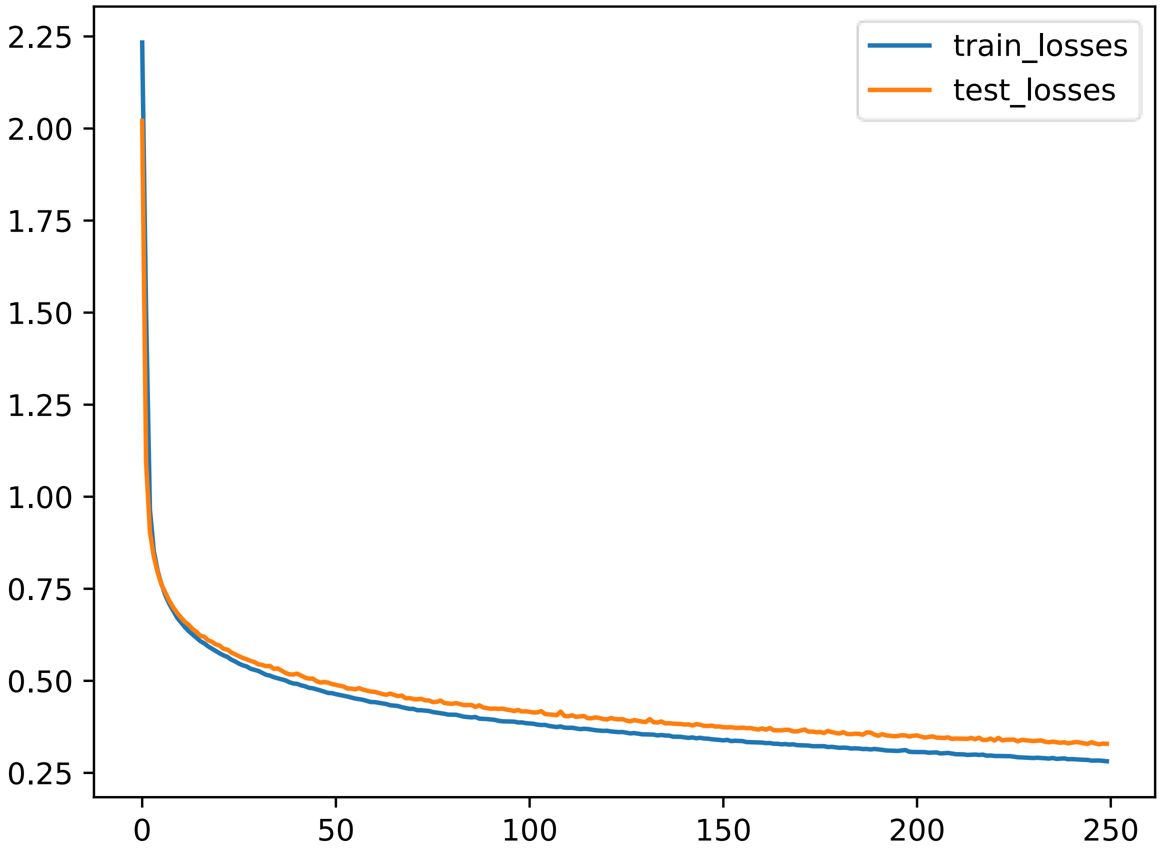


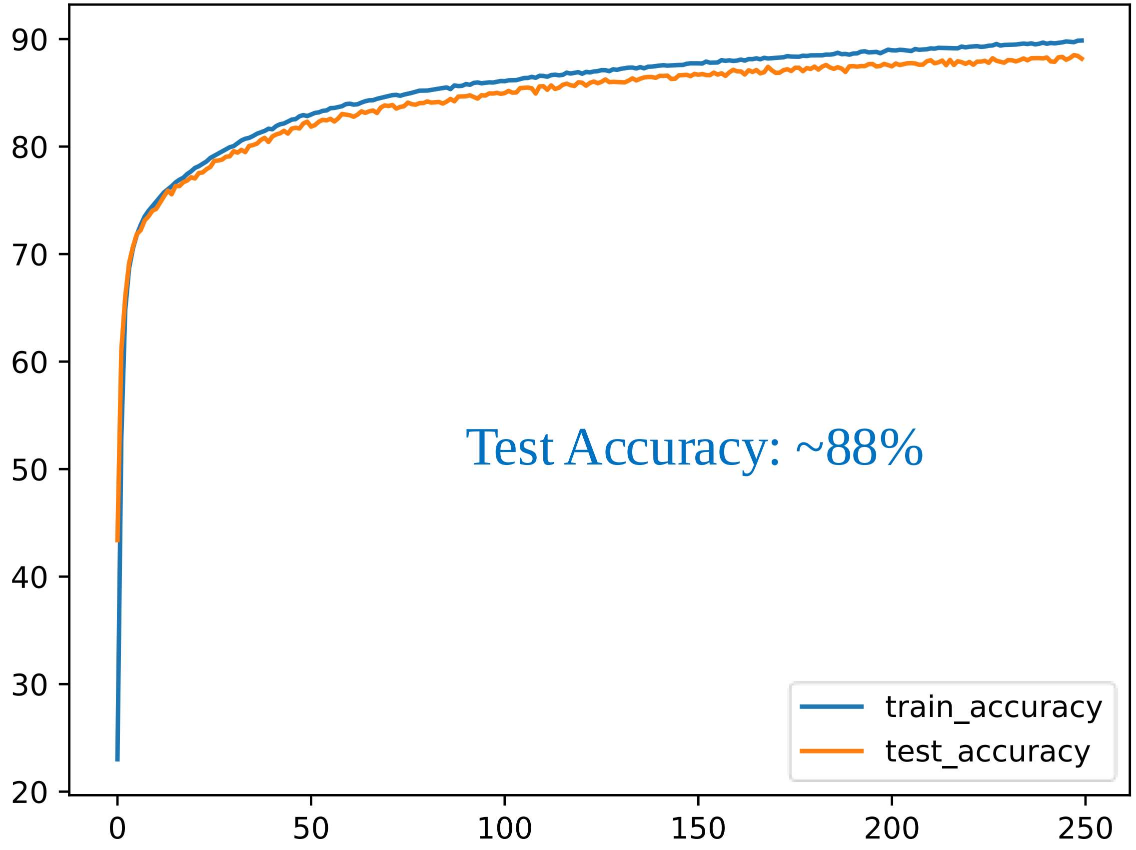
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 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

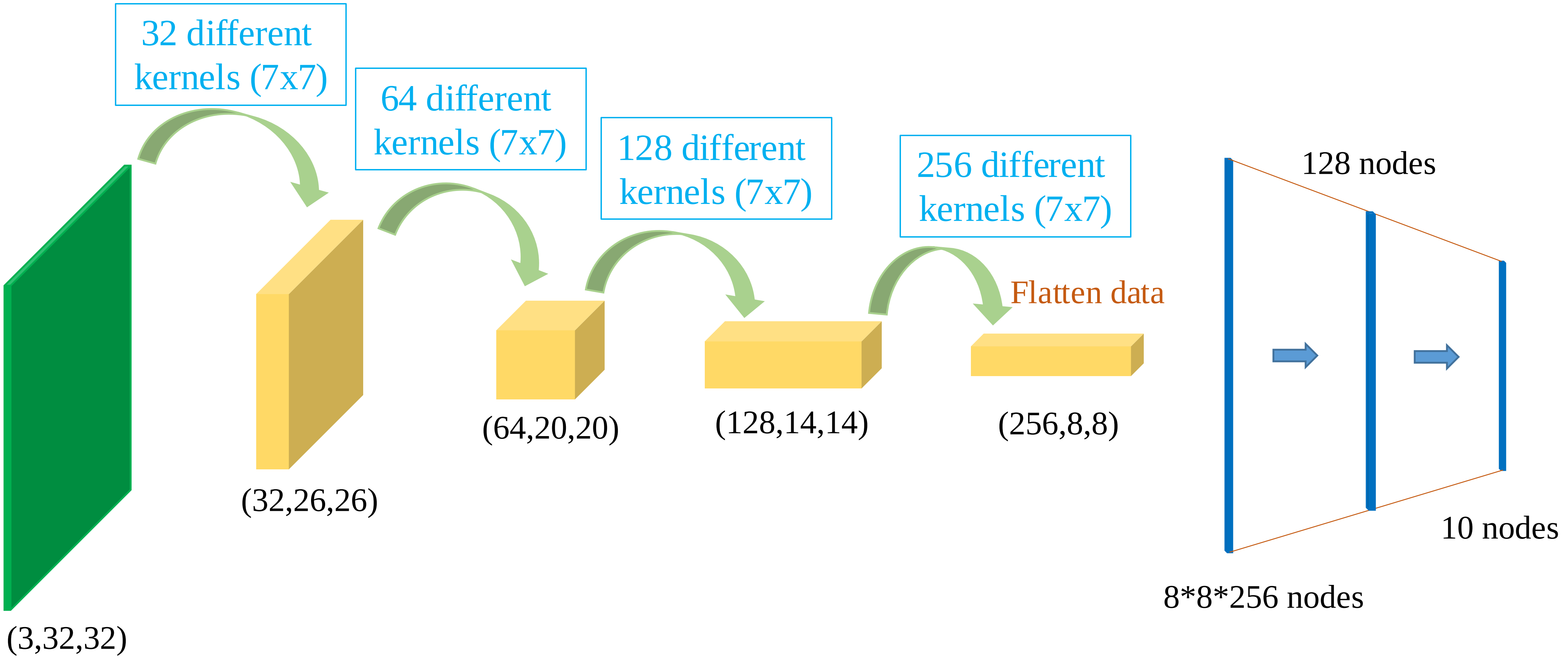
❖ Apply for Fashion-MNIST dataset: case 1





Convolutional Neural Network

❖ Apply for Cifar-10 dataset: case 2

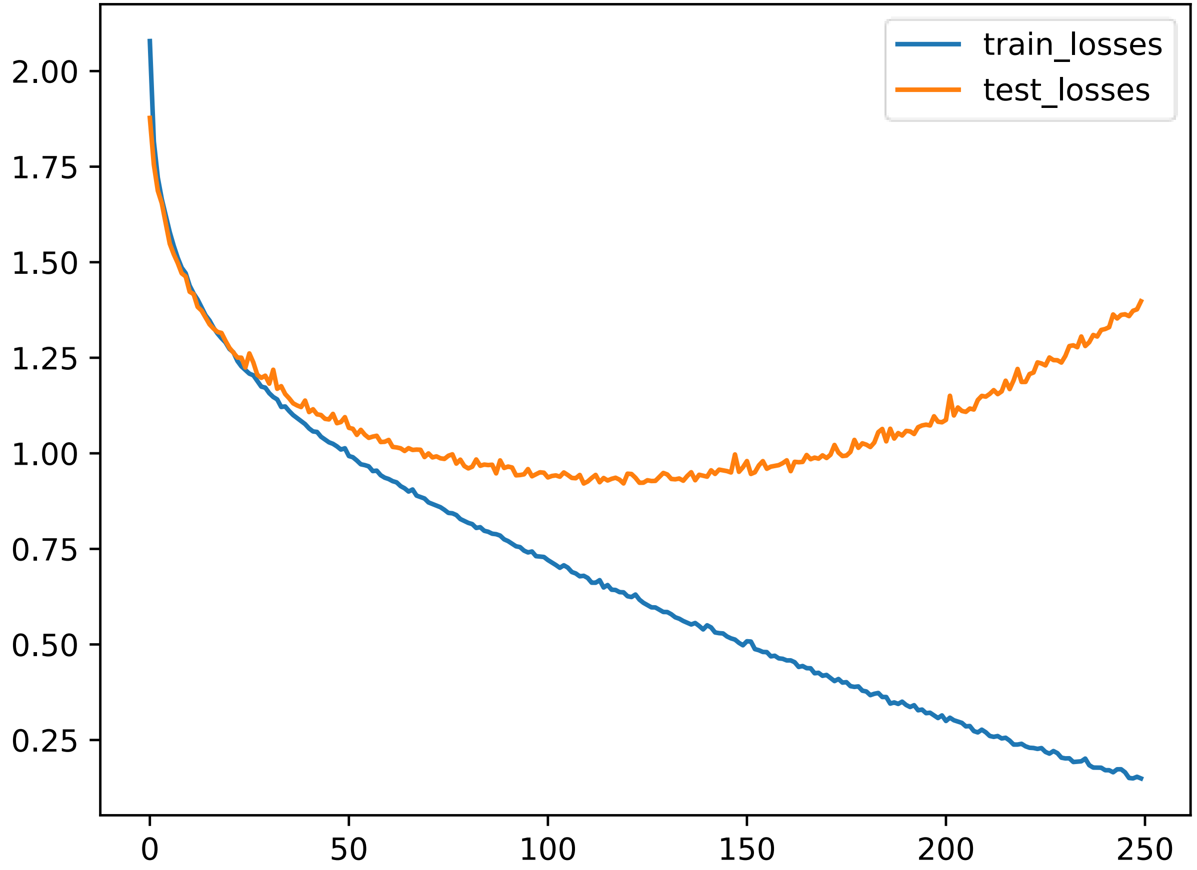


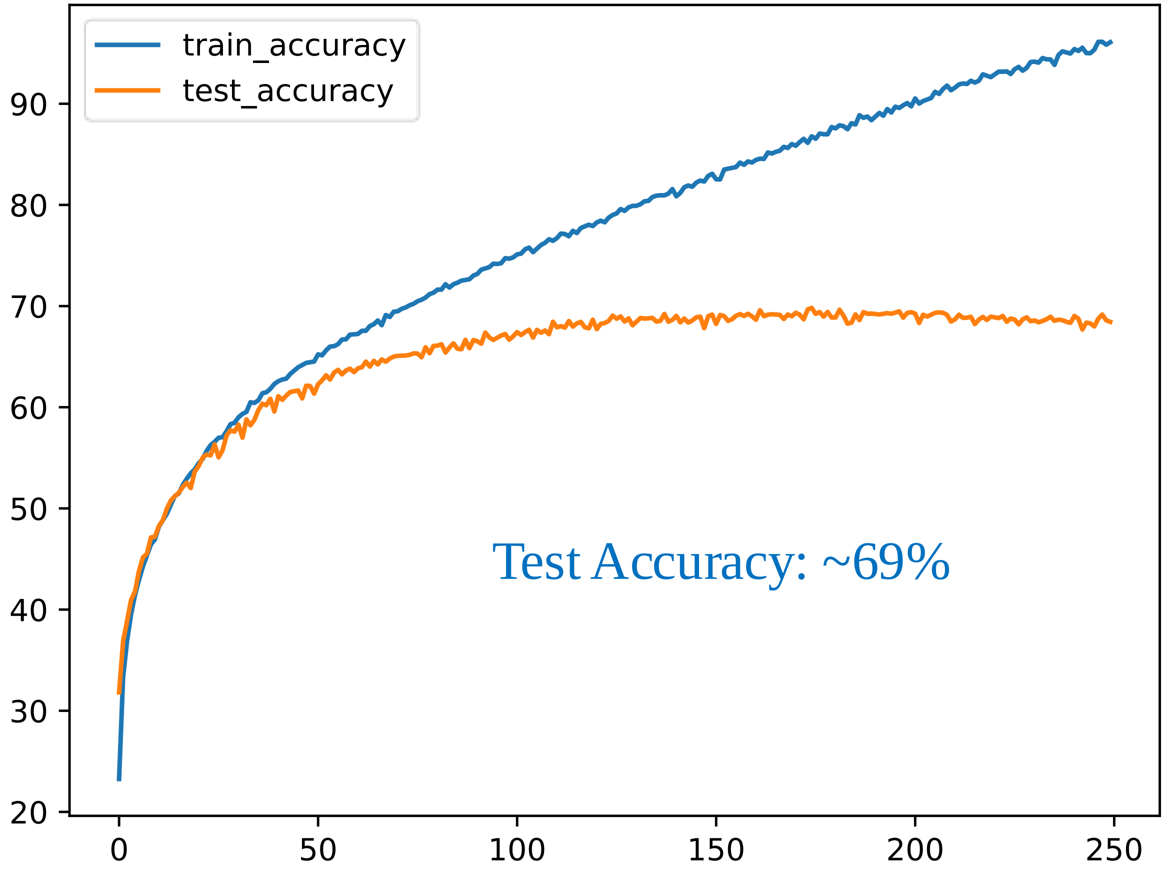
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 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%

Outline

SECTION 1

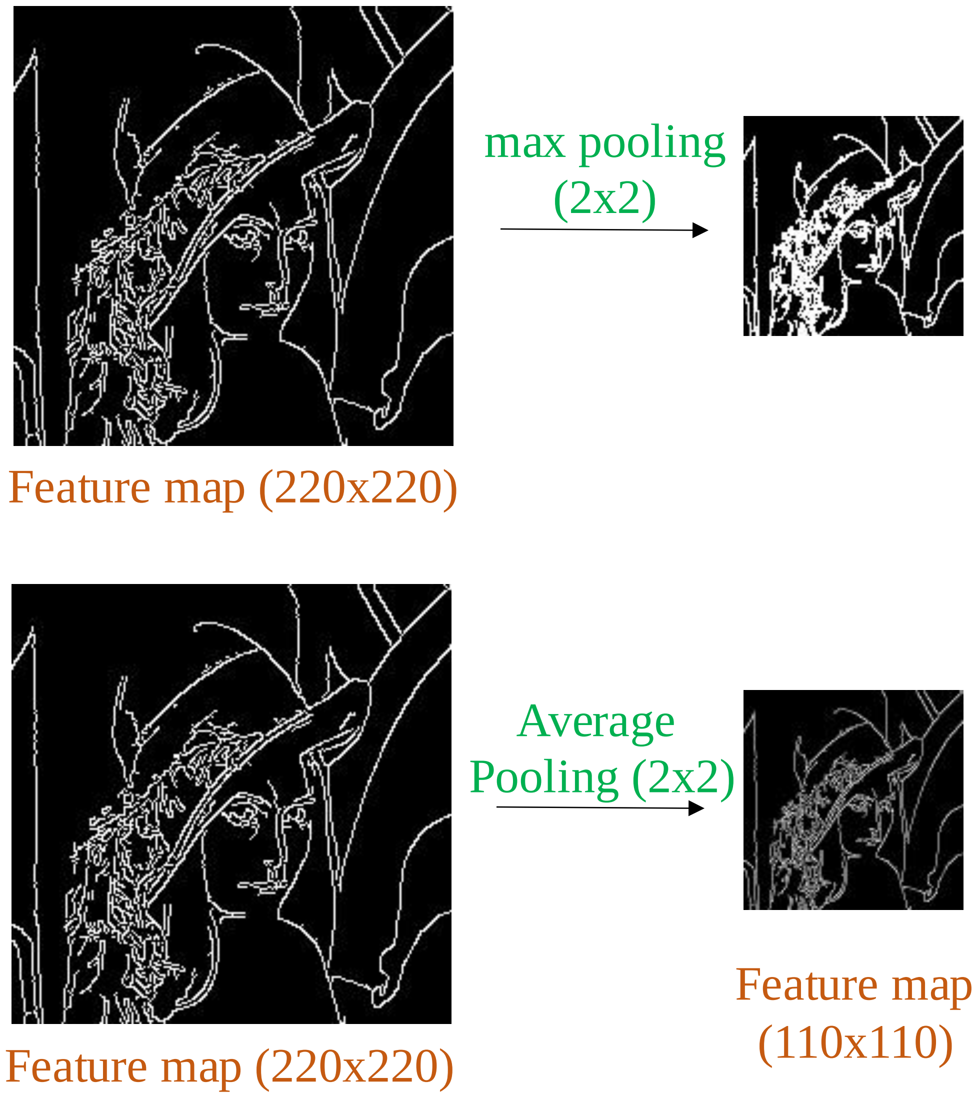
MLP Limitations

SECTION 2

Convolutional Layer

SECTION 3

Standard CNNs



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 $\left(\begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_5 & v_6 & v_7 & v_8 \\ \hline v_9 & v_{10} & v_{11} & v_{12} \\ \hline v_{13} & v_{14} & v_{15} & v_{16} \\ \hline \end{array}\right) = \begin{array}{|c|c|} \hline m_1 & m_2 \\ \hline m_3 & m_4 \\ \hline \end{array}$

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

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

$$\textbf{average pooling}\left(\begin{array}{|c|c|c|c|}\hline v_1 & v_2 & v_3 & v_4 \\ \hline v_5 & v_6 & v_7 & v_8 \\ \hline v_9 & v_{10} & v_{11} & v_{12} \\ \hline v_{13} & v_{14} & v_{15} & v_{16} \\ \hline \end{array}\right) = \begin{array}{|c|c|}\hline m_1 & m_2 \\ \hline m_3 & m_4 \\ \hline \end{array}$$

$$m_1 = \text{mean}(v_1, v_2, v_5, v_6)$$

$$m_2 = \text{mean}(v_3, v_4, v_7, v_8)$$

$$m_3 = \text{mean}(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \text{mean}(v_{11}, v_{12}, v_{15}, v_{16})$$

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)

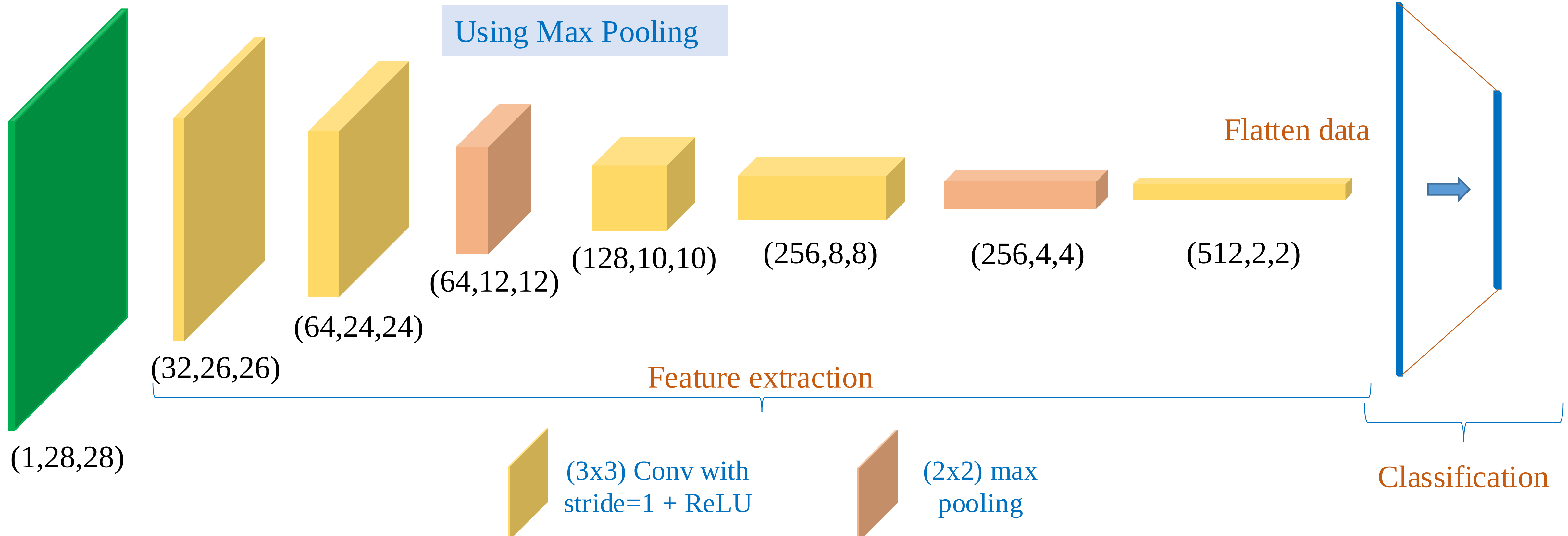
Feature map (110x110)

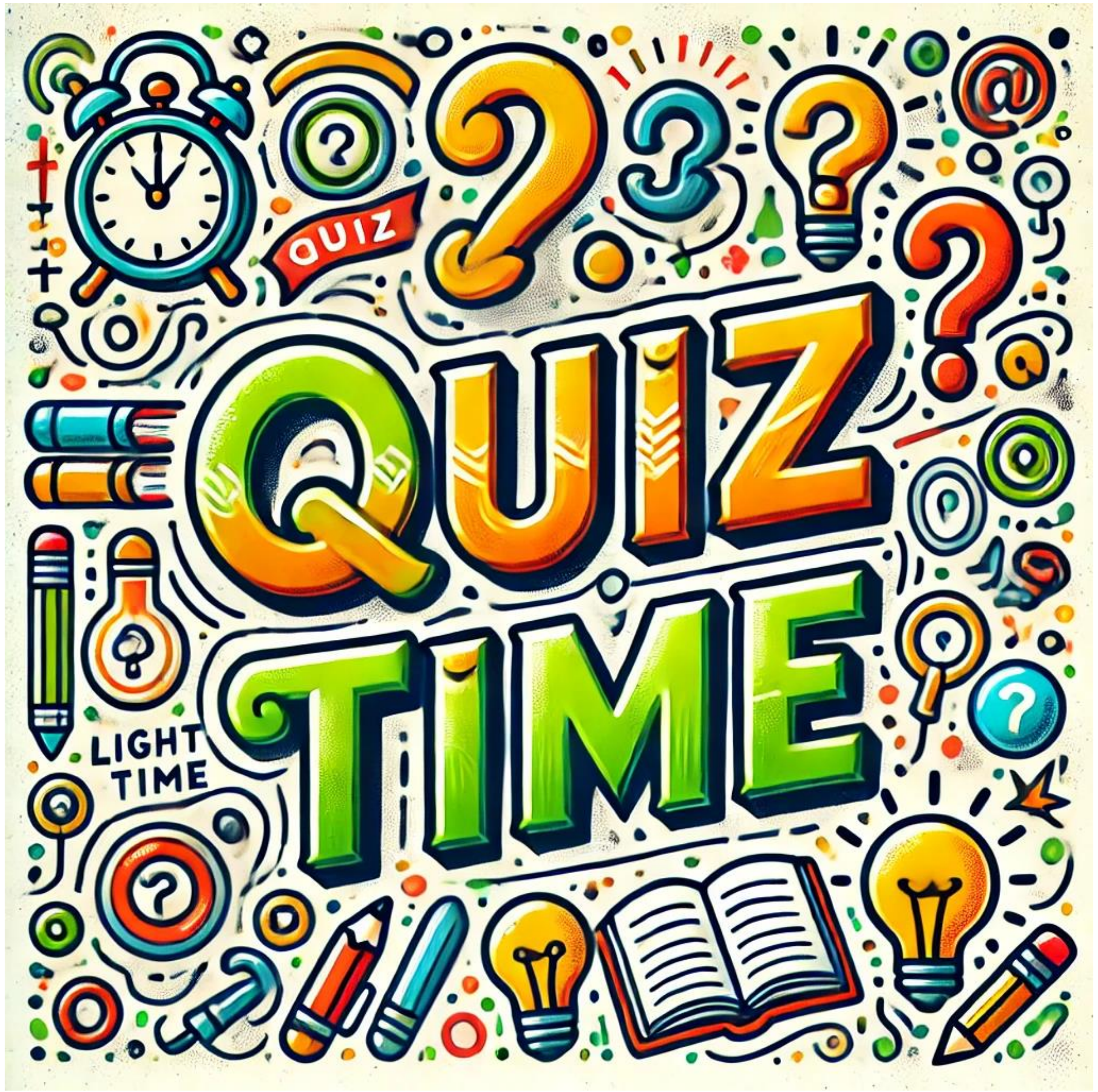
Average Pooling (2x2)

Feature map (55x55)

```python
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)
```

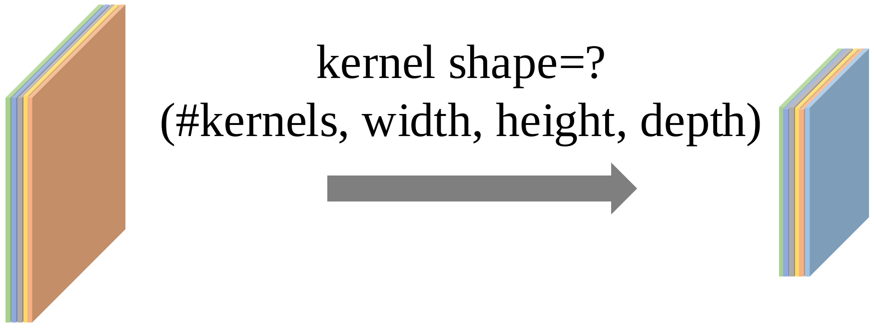
```python
    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.pool(x)

        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
```




QUIZ
QUIZ
TIME
LIGHT
TIME

Question 1

❖ Giả sử chúng ta dùng kernel có width=3 và height=3, shape của kernel là gì (#kernels là số lượng kernel)?



Feature maps
(10,22,22)

Feature maps
(6,20,20)

a) (10, 3, 3, 10)

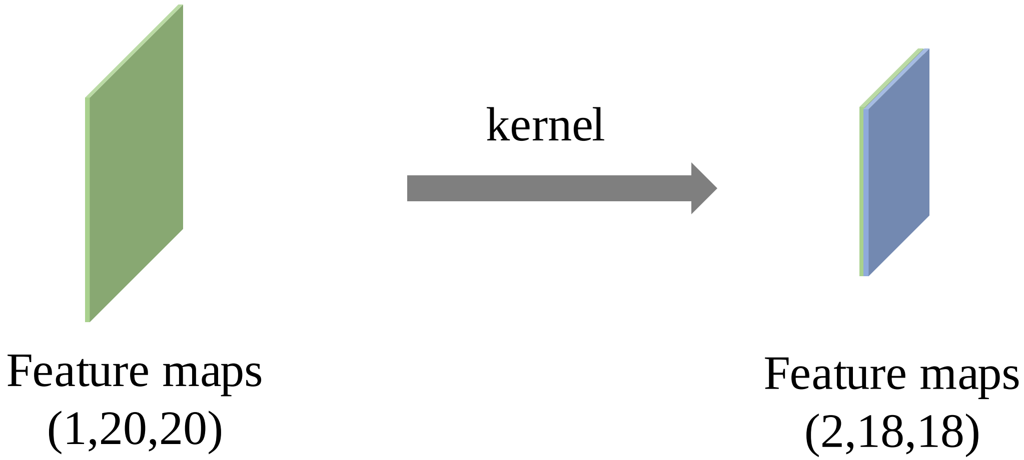
b) (10, 3, 3, 6)

c) (6, 3, 3, 10)

d) Không xác định

Question 2

❖ Giả sử chúng ta dùng kernel có width=3 và height=3, tổng số lượng tham số của kernel là bao nhiêu?



a) Có 9 tham số

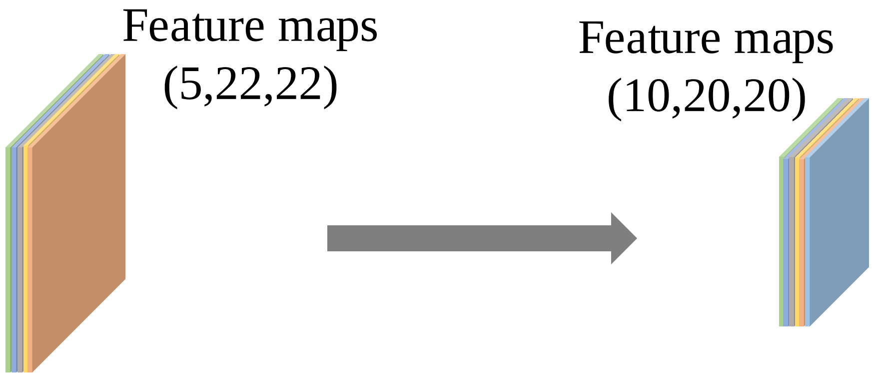
b) Có 10 tham số

c) Có 18 tham số

d) Có 20 tham số

Question 3

❖ Code Pytorch nào áp dụng đúng cho hình sau?



C1
```
nn.Conv2d(5, 10, kernel_size=3)
```

C2
```
nn.Conv2d(10, 10, kernel_size=3)
```

C3
```
nn.Conv2d(5, 5, kernel_size=3)
```

C4
```
nn.Conv2d(10, 5, kernel_size=3)
```

Question 4

❖ Input (cho model) có shape là (1, 50, 50) thì output có shape là gì (bỏ qua phần batch size)?

```
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.relu = nn.ReLU()

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

model = CustomModel()
summary(model, (1, 50, 50))
```

a) (1, 48, 48)

b) (1, 24, 24)

c) (32, 48, 48)

d) (32, 24, 24)

Question 5

❖ Input (cho model) có shape là (1, 50, 50) thì output có shape là gì (bỏ qua phần batch size)?

```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool1(x)
        x = self.relu(self.conv2(x))
        x = self.pool2(x)
        return x

model = CustomModel()
summary(model, (1, 50, 50))
```

a) (32, 24, 24)

b) (32, 10, 10)

c) (32, 8, 8)

d) (32, 6, 6)

Question 6

❖ Input (cho model) có shape là (1, 50, 50) thì output có shape là gì (bỏ qua phần batch size)?

```
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(1, 32, kernel_size=3)
        self.relu = nn.ReLU()

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

model = CustomModel()
summary(model, (1, 50, 50))
```

a) shape=(32, 50, 50)

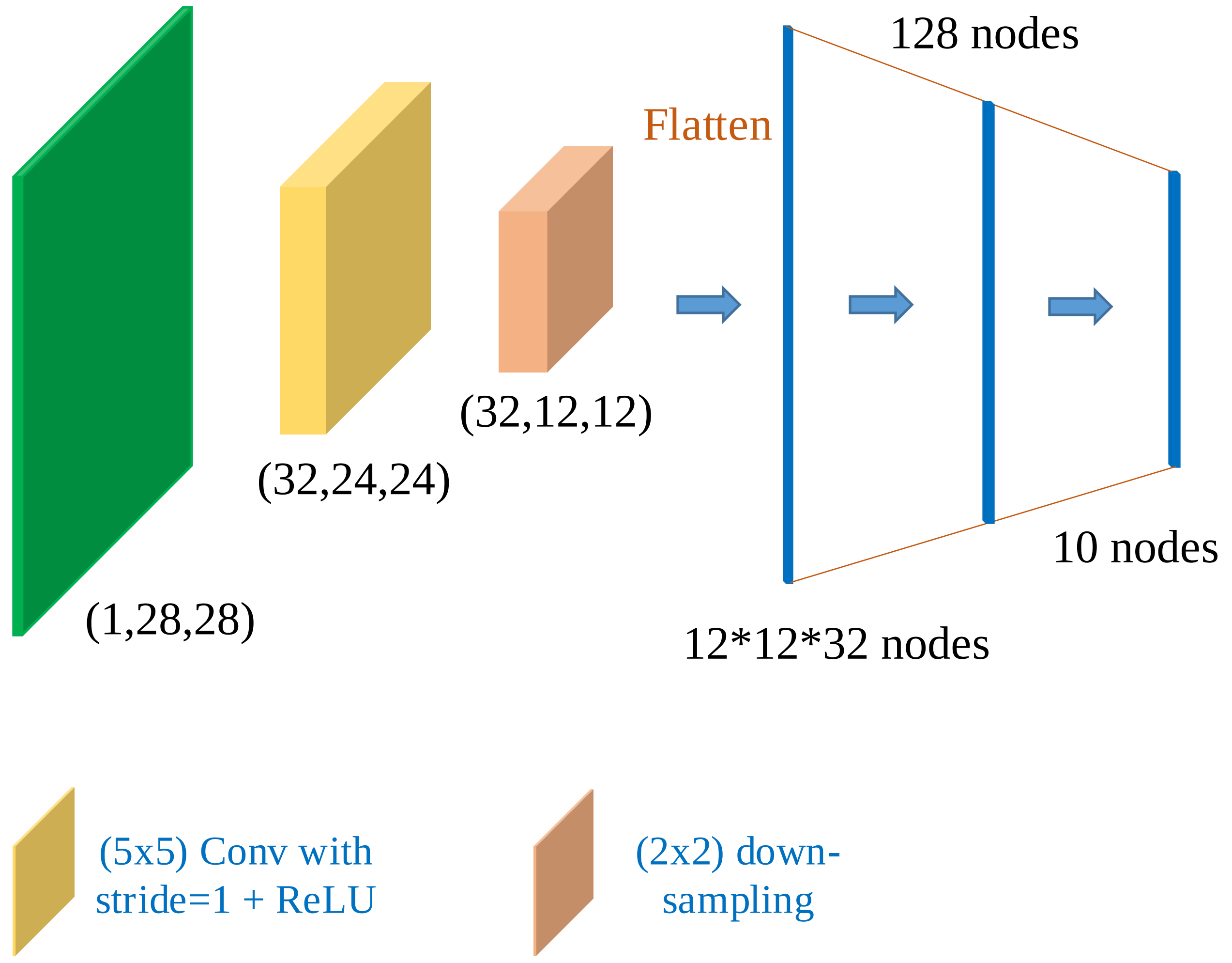
b) shape=(32, 48, 48)

c) shape=(32, 46, 46)

d) Có lỗi

Comparison of down-samplings

Model Design



Implementation 1

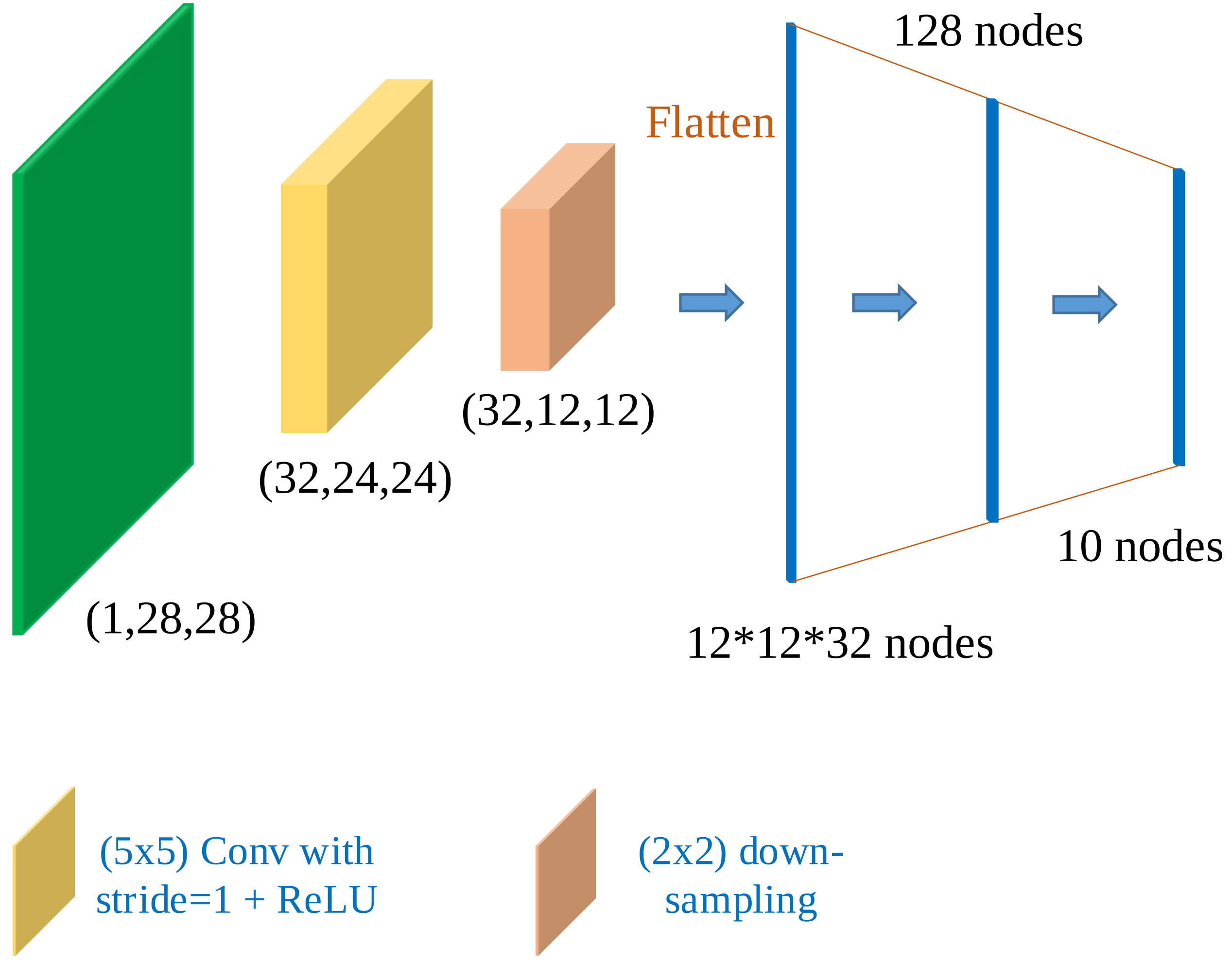
max pooling

```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)
```

Comparison of down-samplings

Implementation 2

Model Design

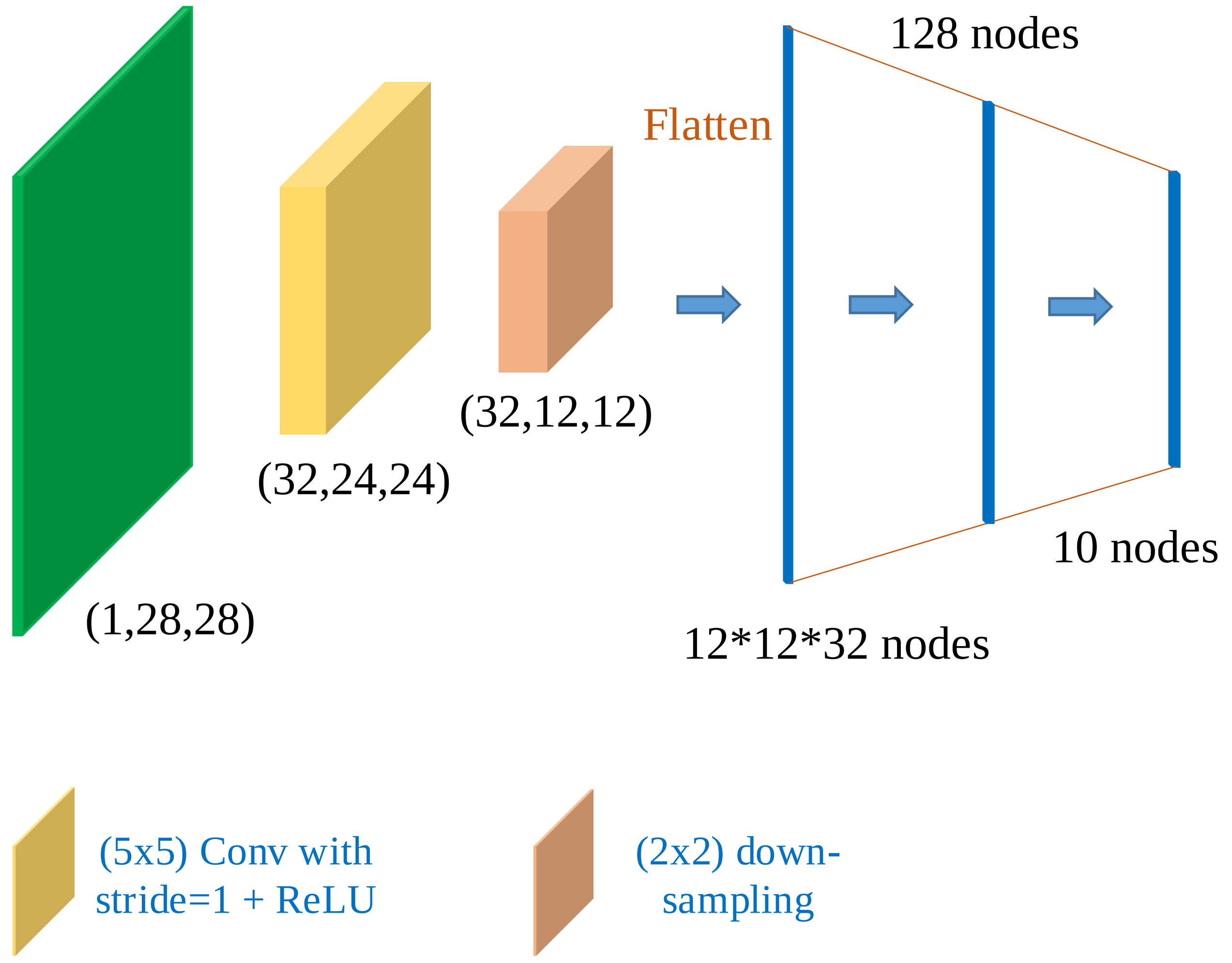


average pooling

```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)
```

Comparison of down-samplings

Model Design

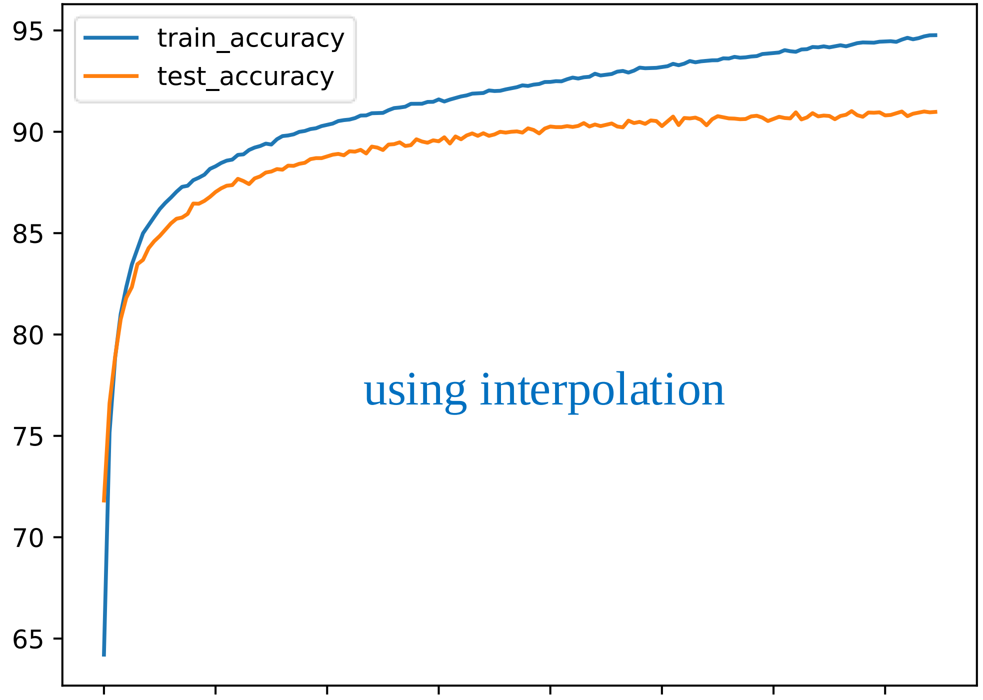


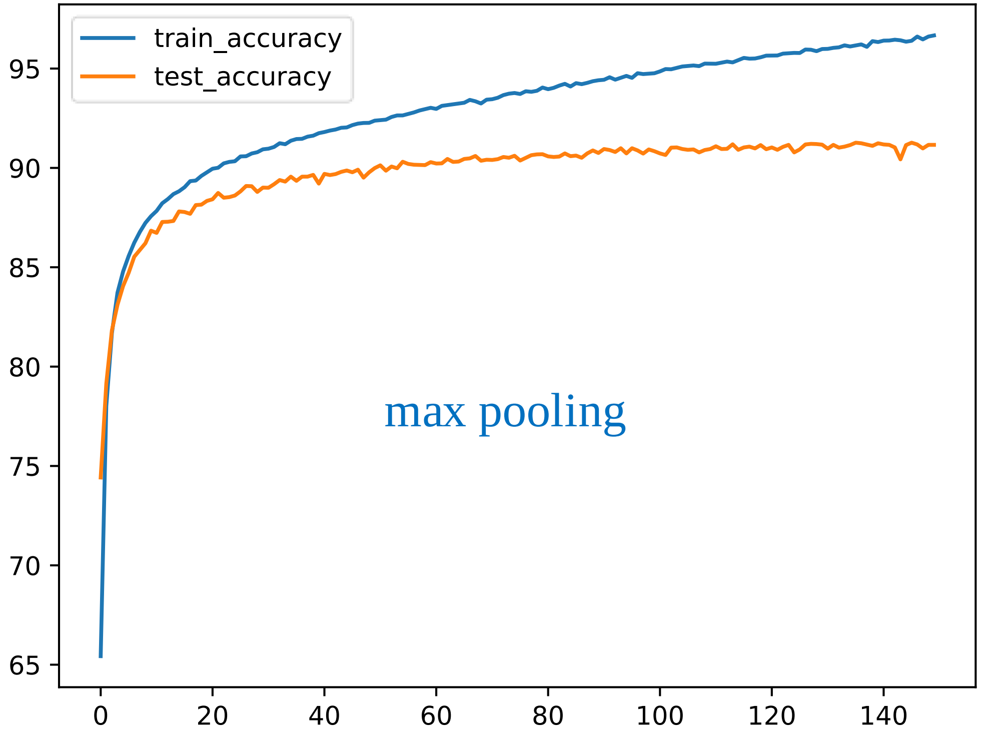

```python
class ResizeLayer(nn.Module):
    def __init__(self, scale_factor, mode='bilinear',
                 align_corners=False):
        super(ResizeLayer, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode
        self.align_corners = align_corners

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

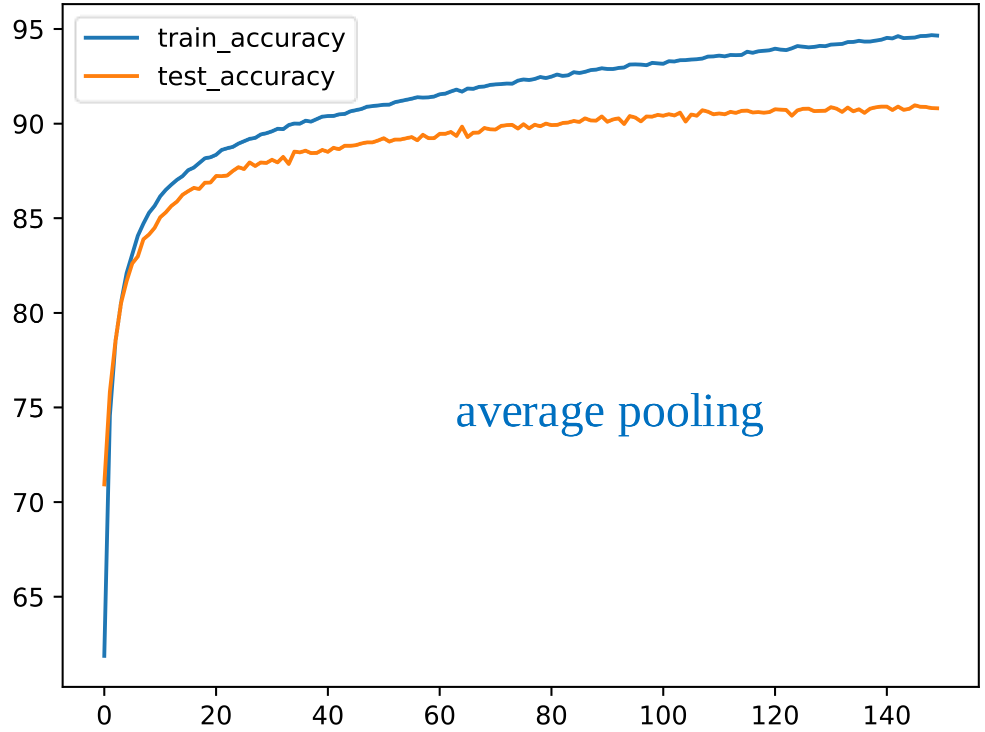
interpolation

```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







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
w_3	w_4

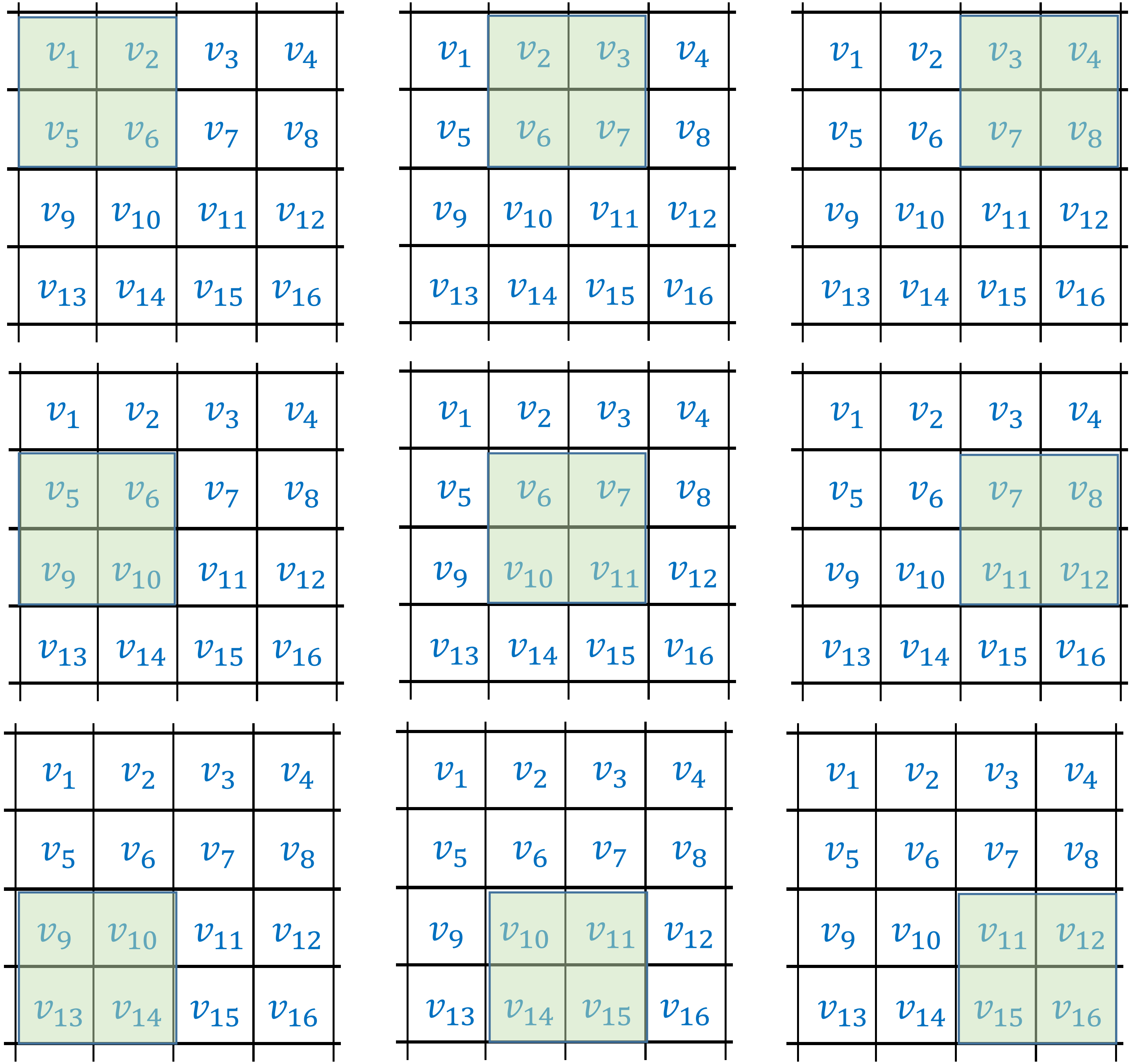
b

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**

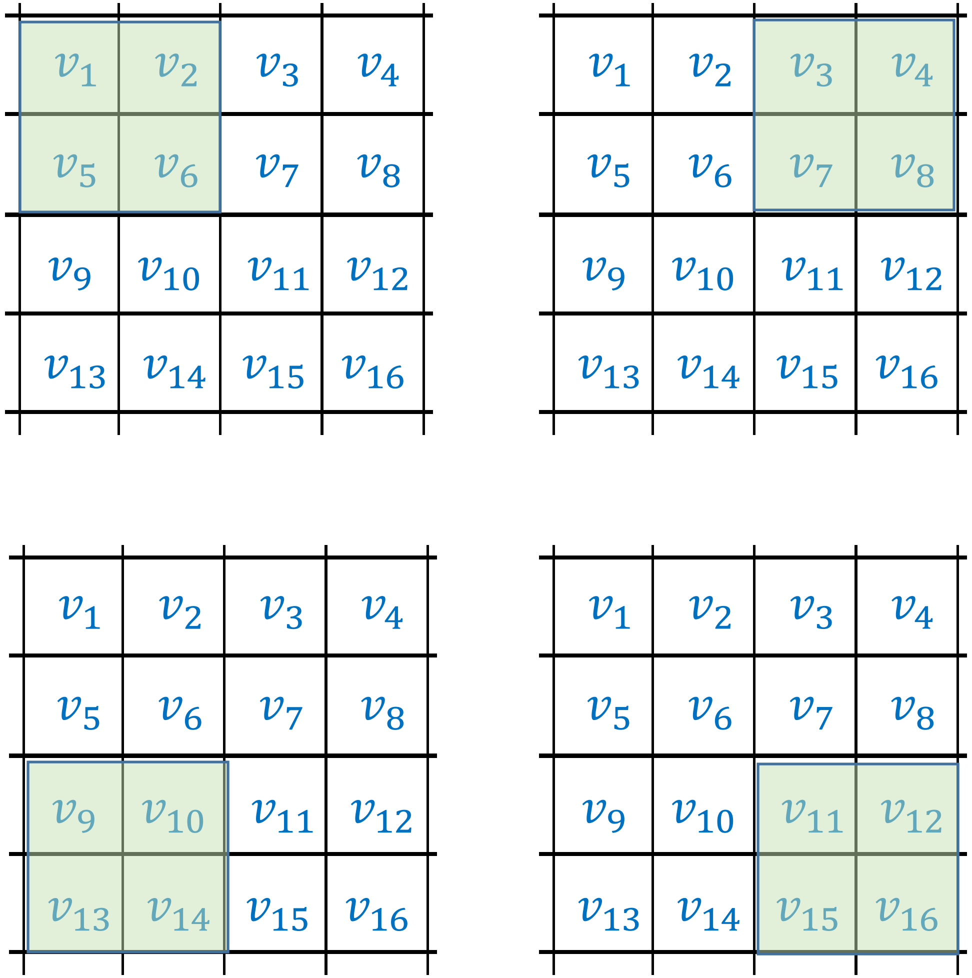
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=2

m_1	m_2
m_3	m_4

Output

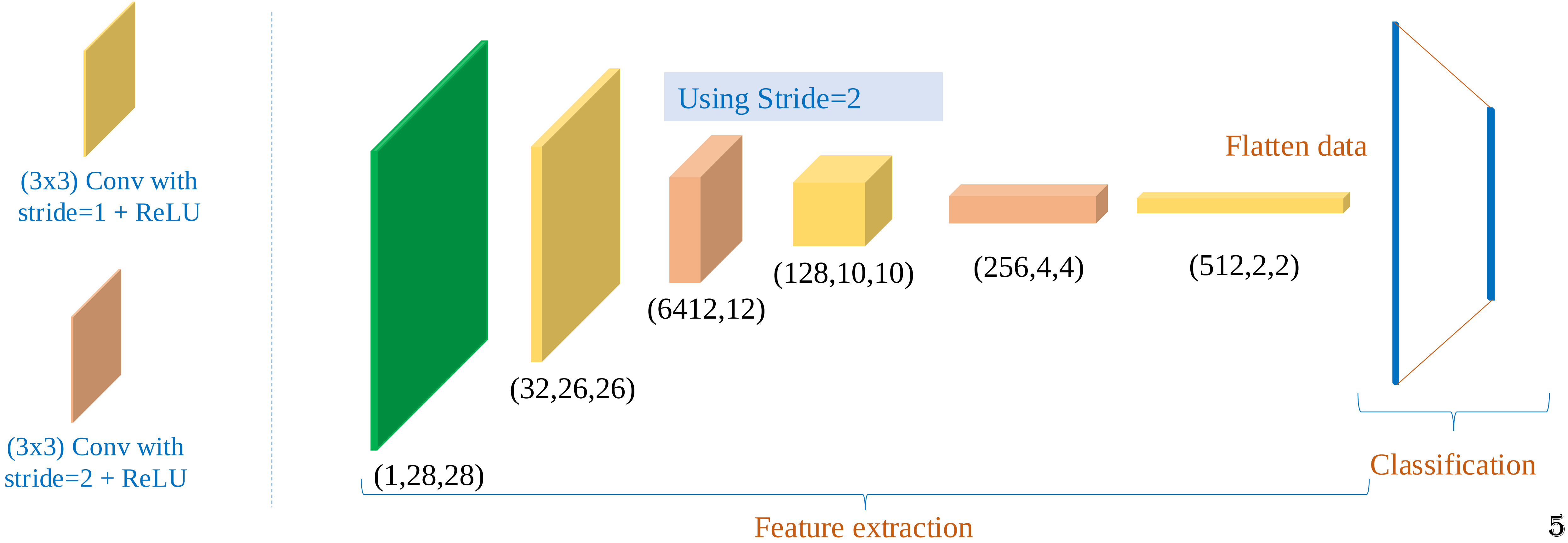


nn.Conv2d(in_channels, out_channels, kernel_size, stride=1)

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
```



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 $(\boldsymbol{D}_p) =$

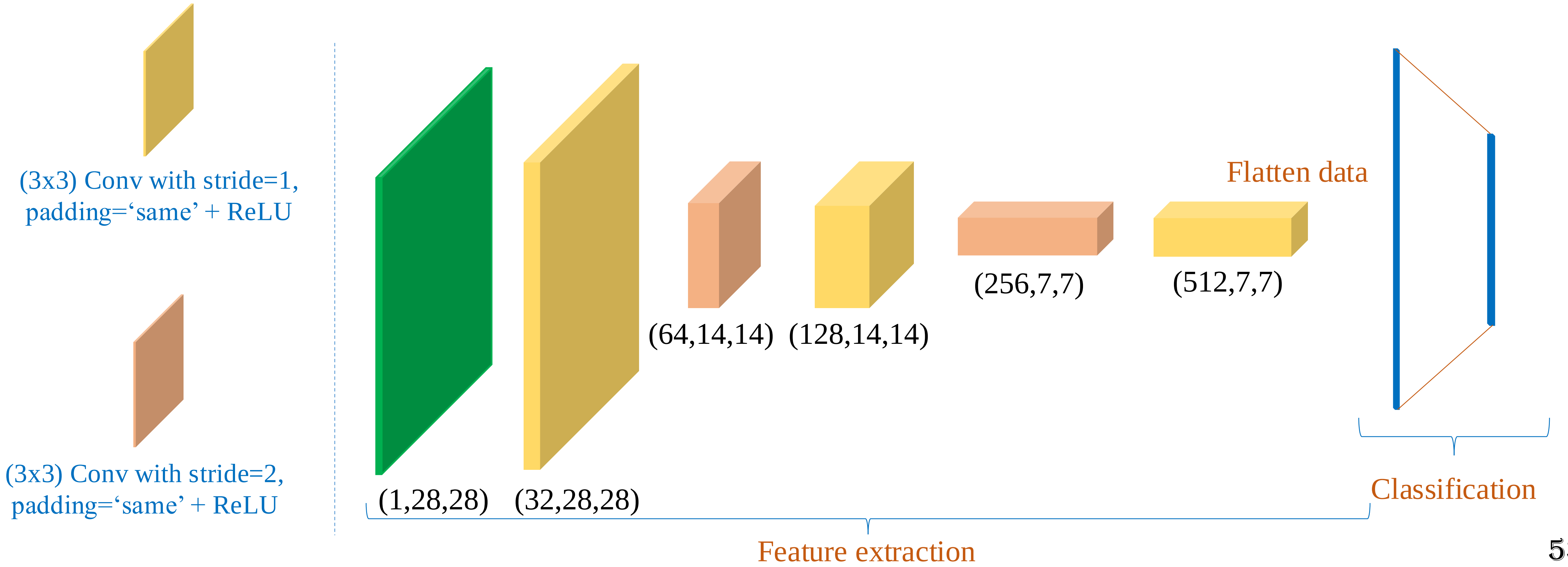
m_1	m_2	m_3	m_4
m_5	m_6	m_7	m_8
m_9	m_{10}	m_{11}	m_{12}
m_{13}	m_{14}	m_{15}	m_{16}

Output
(4x4)

Padding

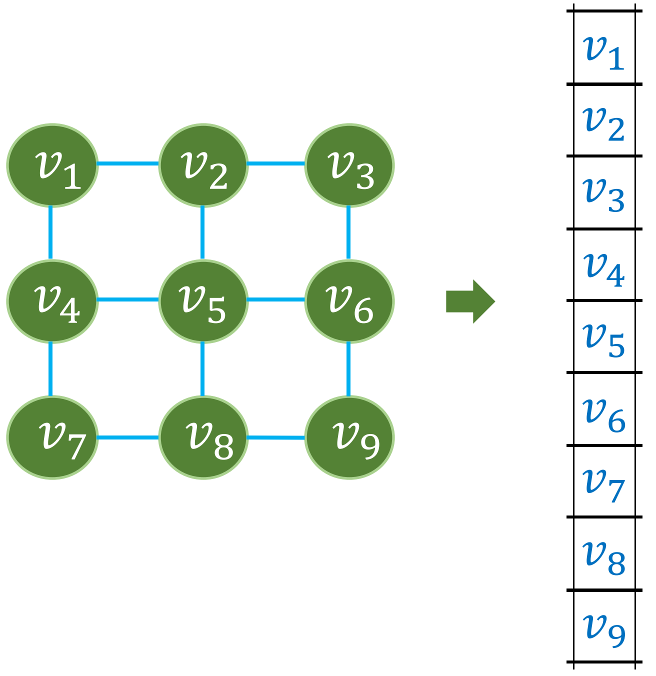
❖ Example

```
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)
)
```



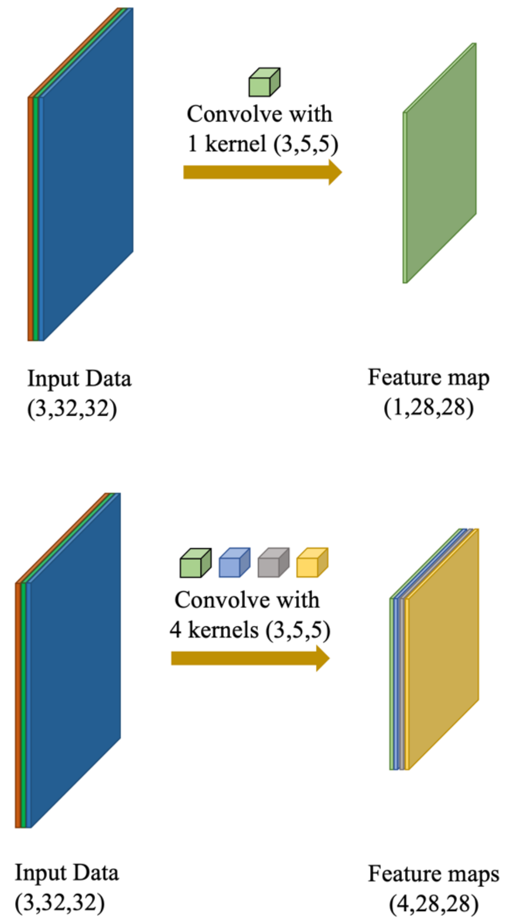
Summary

MLP Limitations

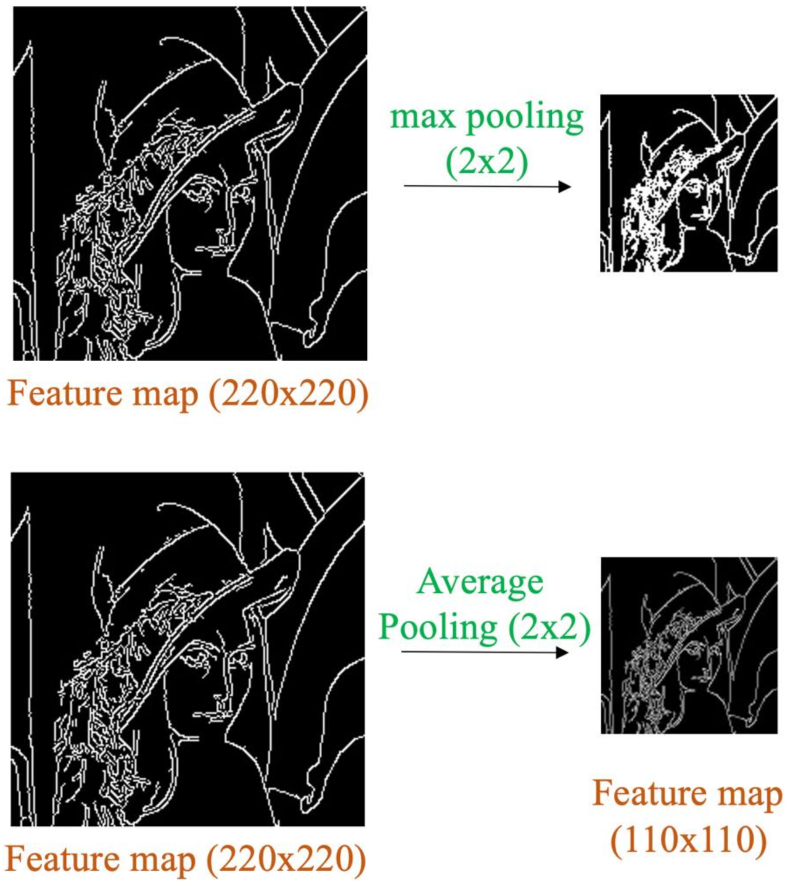


Flattened Data

Convolutional Layer



Standard CNNs



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