AI VIETNAM
All-in-One Course

Convolutional Neural Network

Construction of Standard CNNs

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

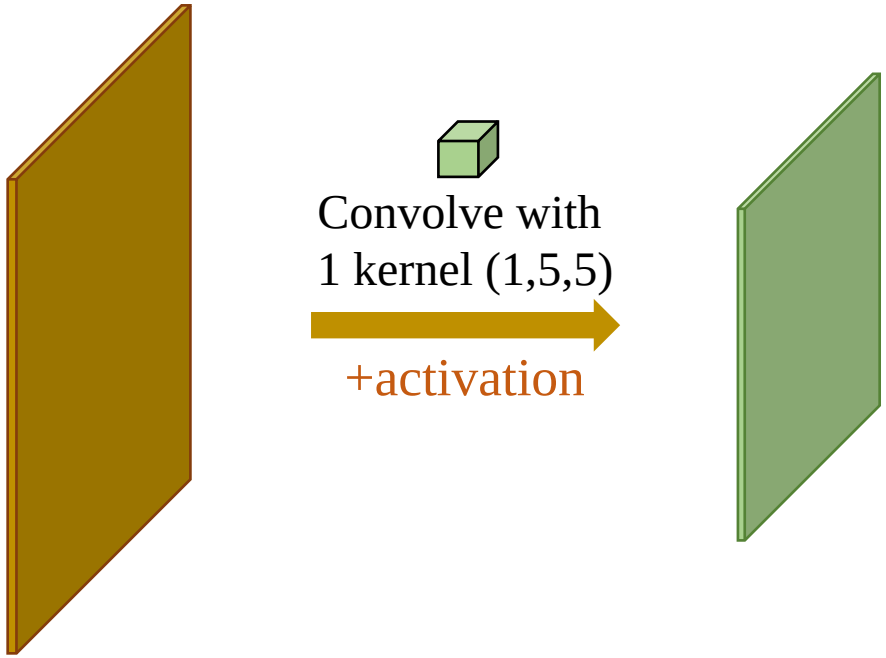
Outline

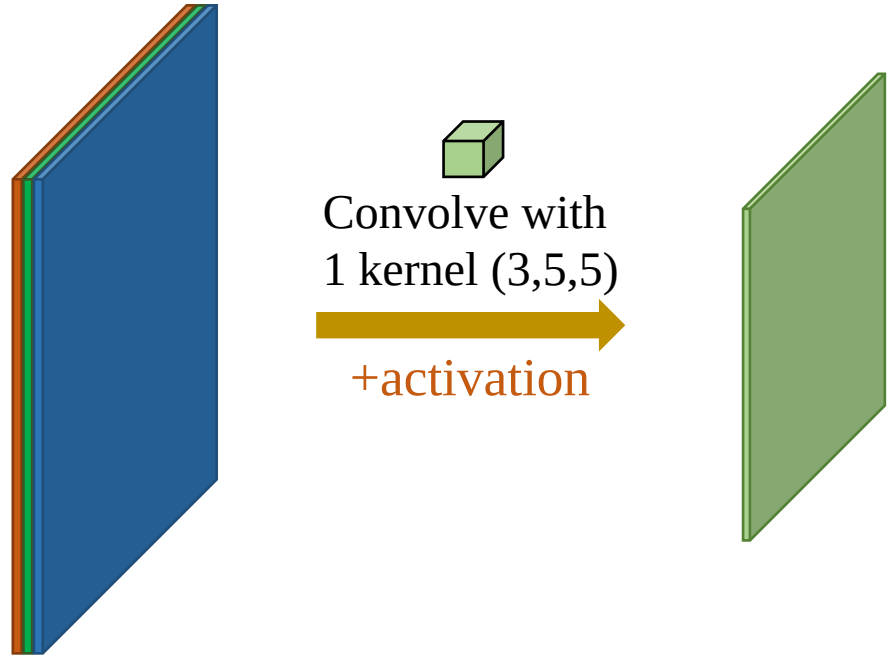
- Fashion-MNIST/Cifar10 Using only Conv2d
- Pooling
- Padding
- 1x1 Convolution
- LeNet and VGG Models

Convolutional Neural Network

❖ Convolution layer in PyTorch

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





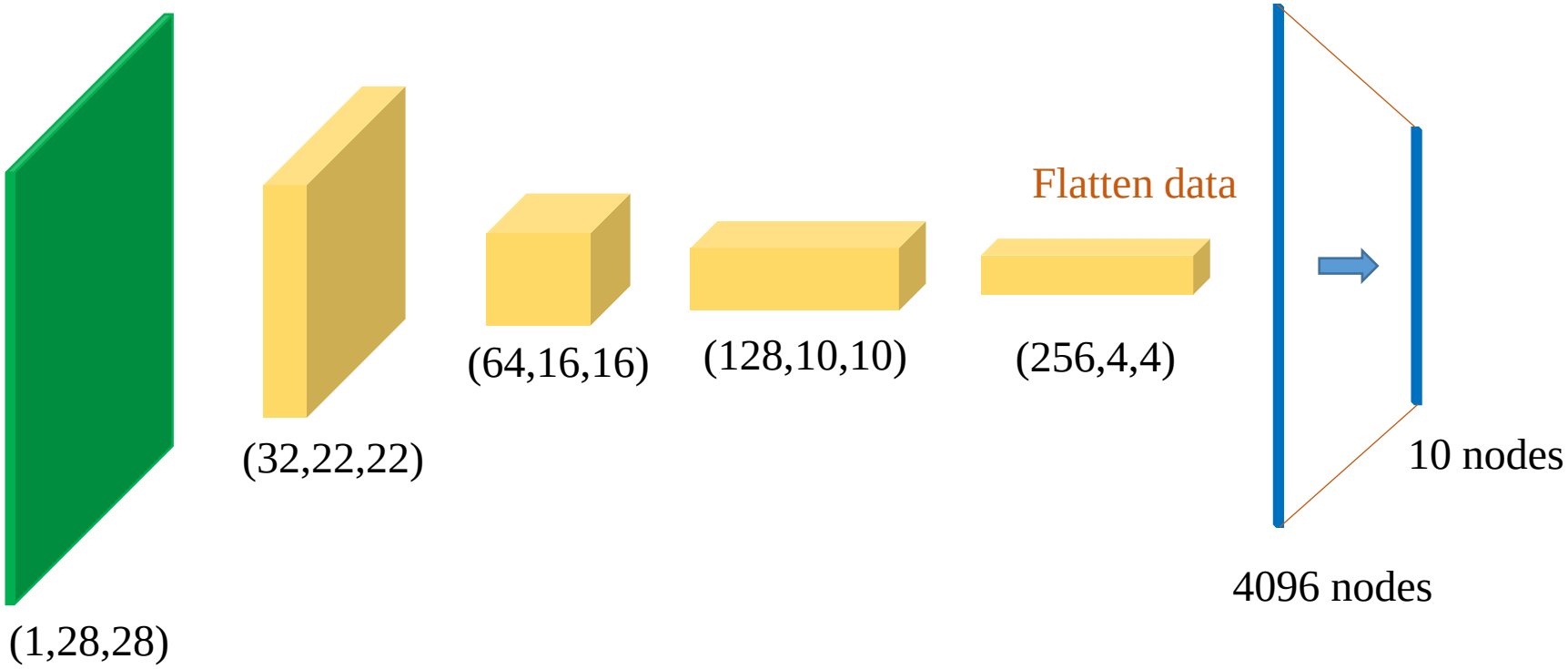
Convolutional Neural Network

Fashion-MNIST dataset

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

Convolutional Neural Network

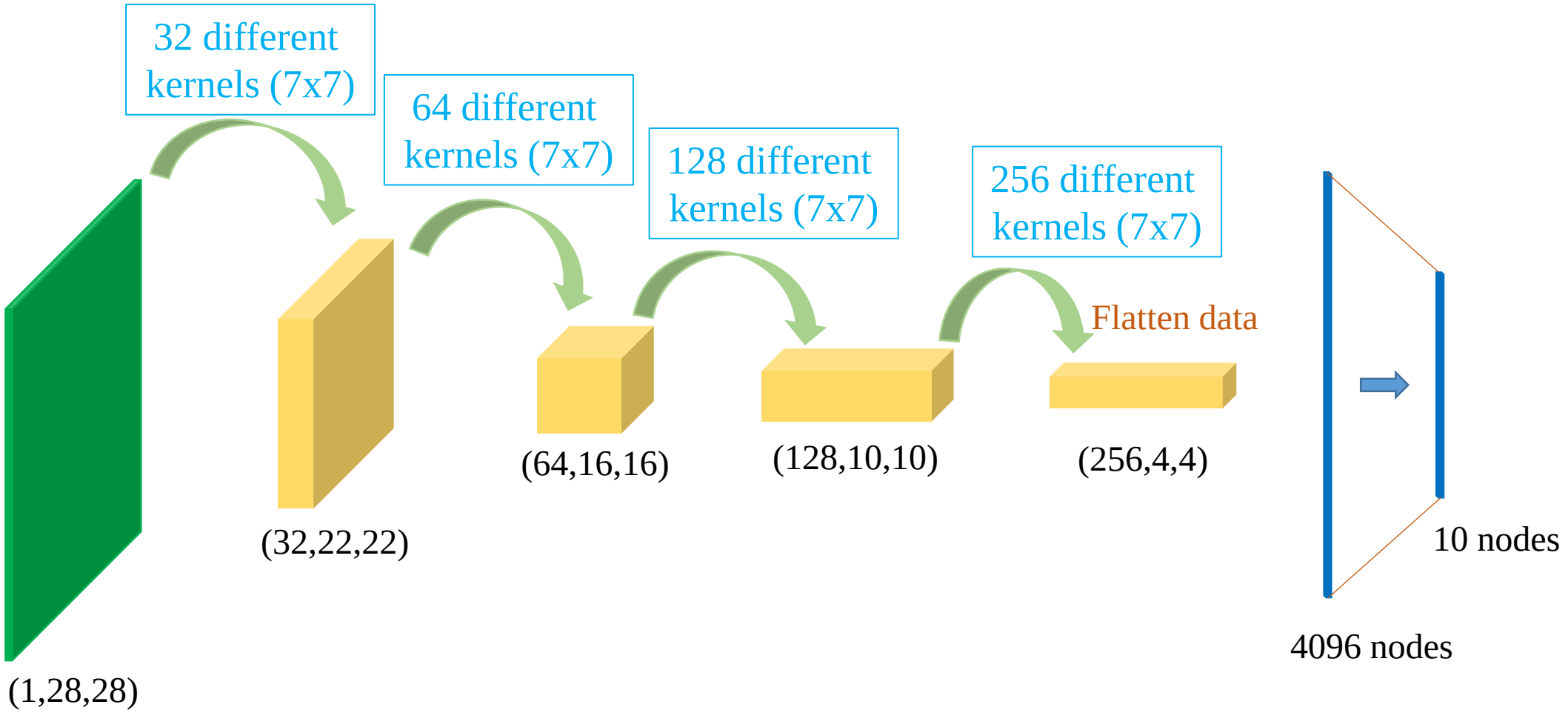
❖ Design a model for Fashion-MNIST dataset



Convolutional Neural Network

❖ Design a model for Fashion-MNIST dataset

demo



Simple Convolutional Neural Network

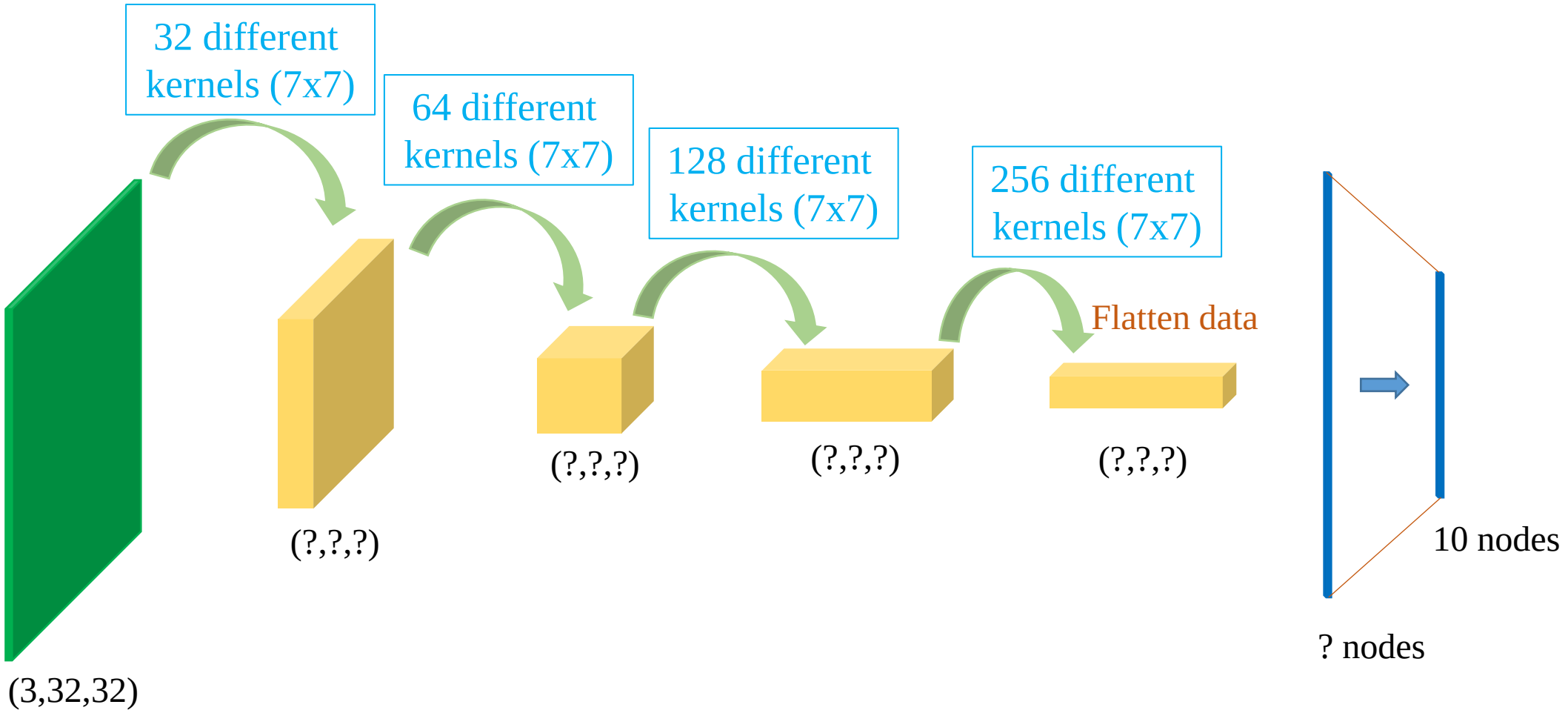
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128,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

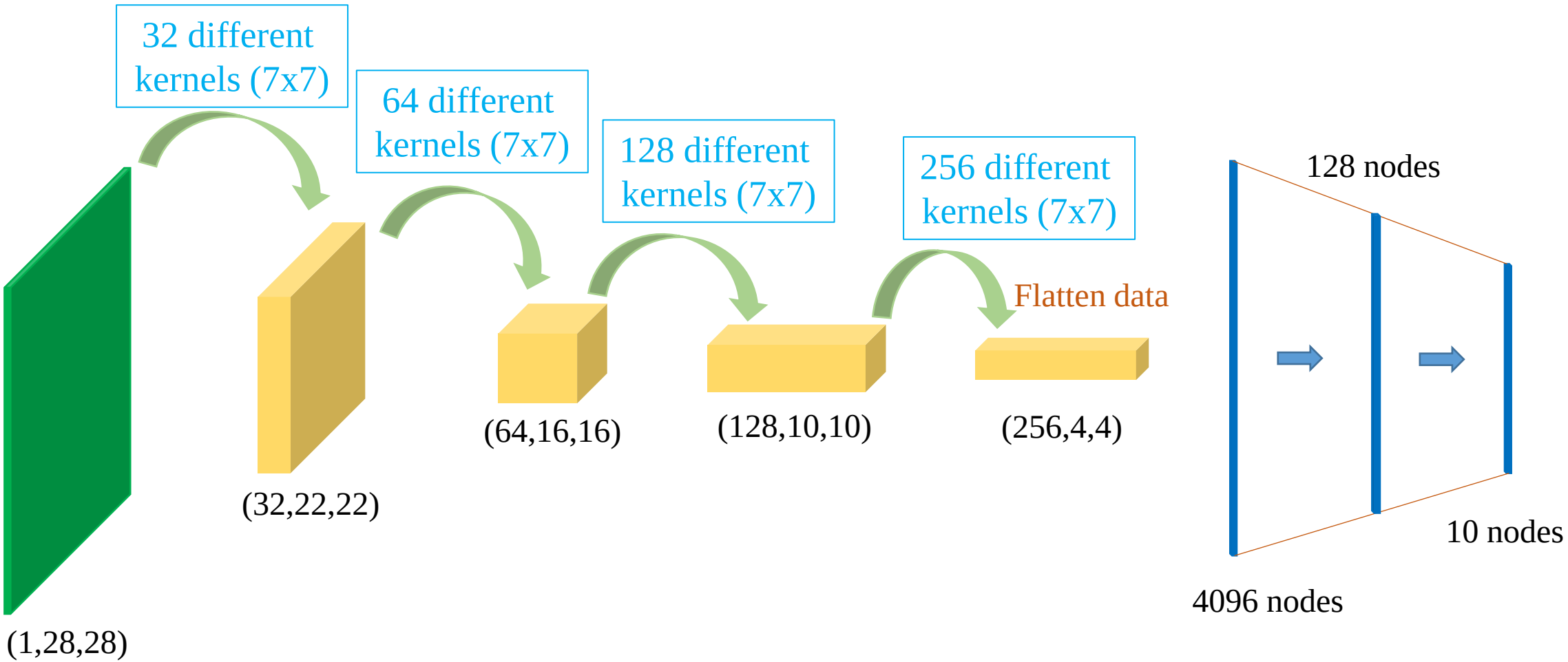
❖ Design a model for Cifar-10 dataset

demo



Convolutional Neural Network

❖ Fashion-MNIST dataset: case 1

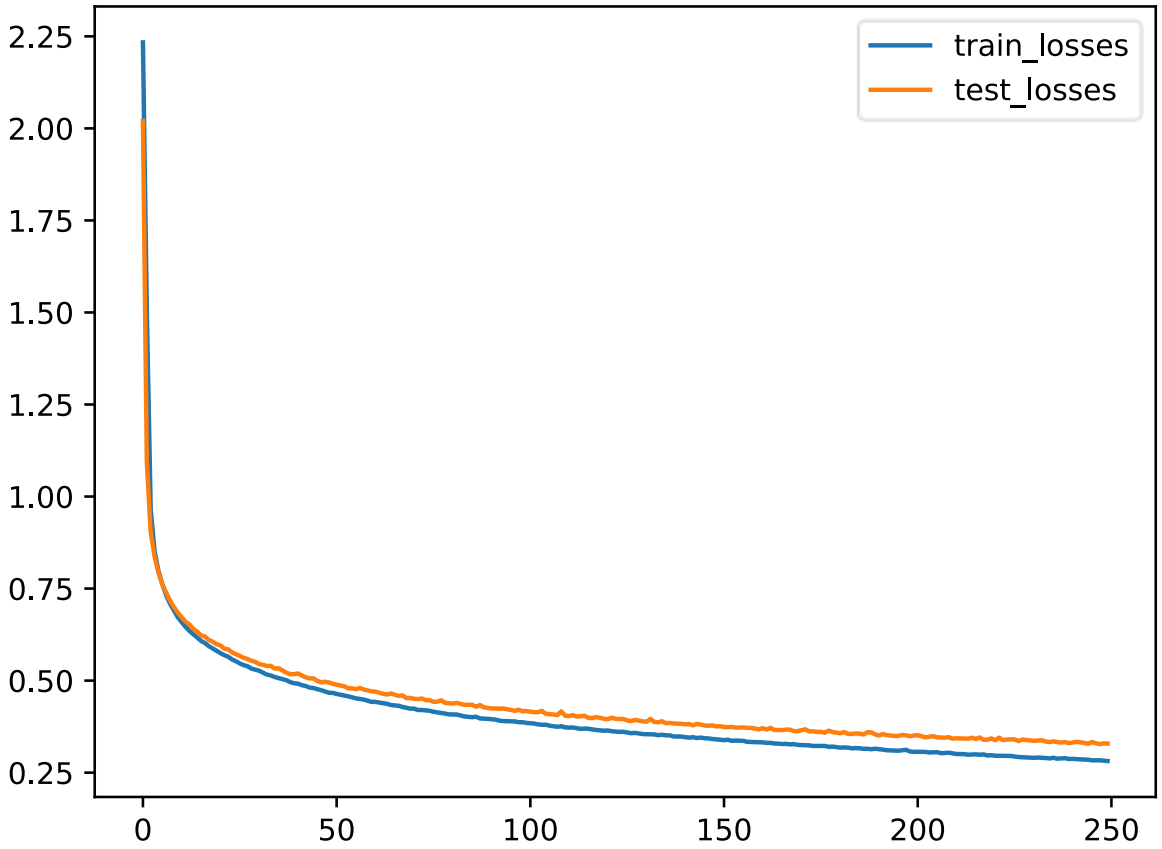


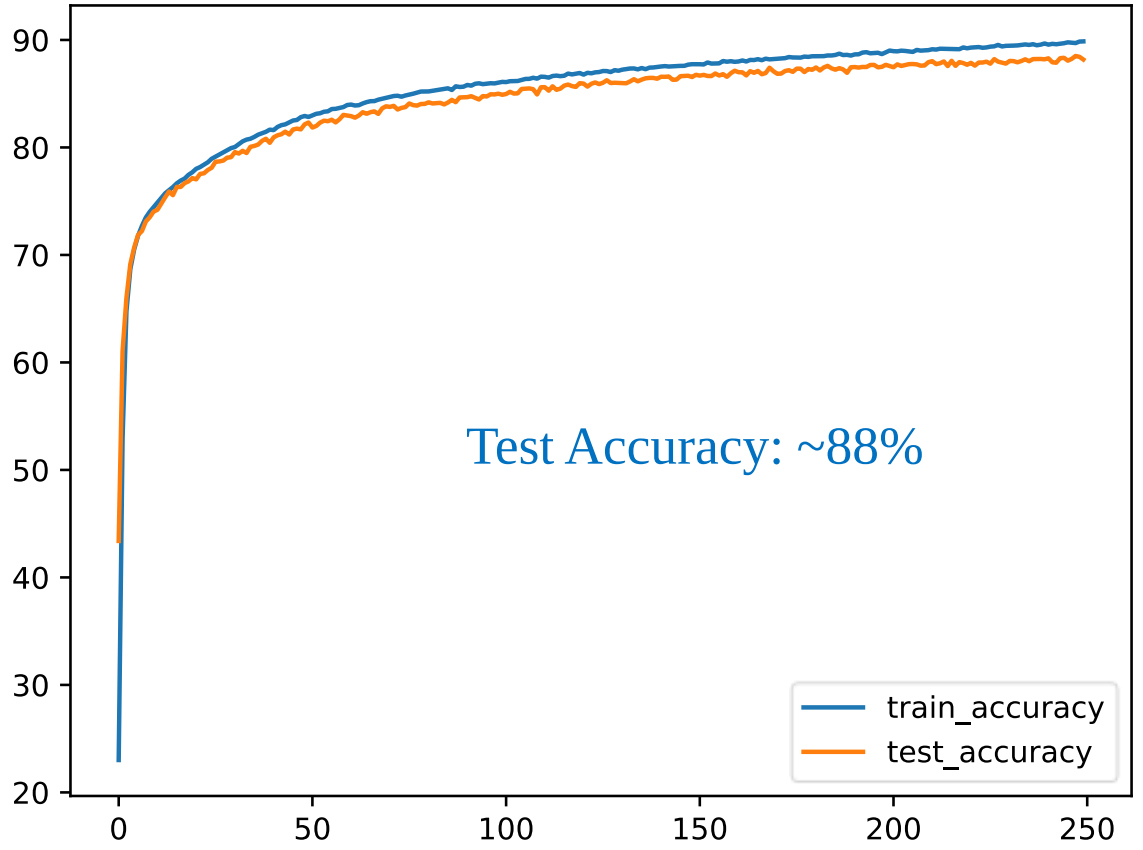
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128,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

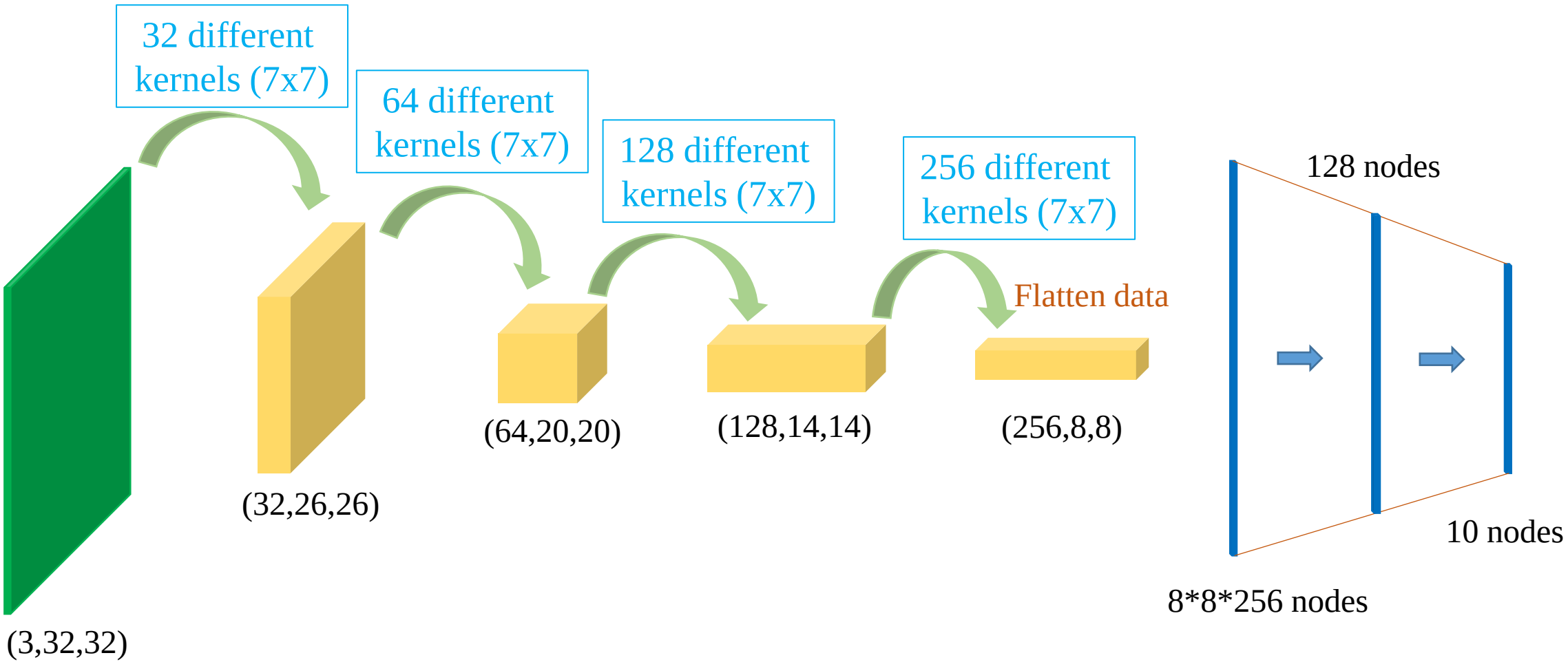
❖ Fashion-MNIST dataset: case 1





Convolutional Neural Network

❖ Cifar-10 dataset: case 2

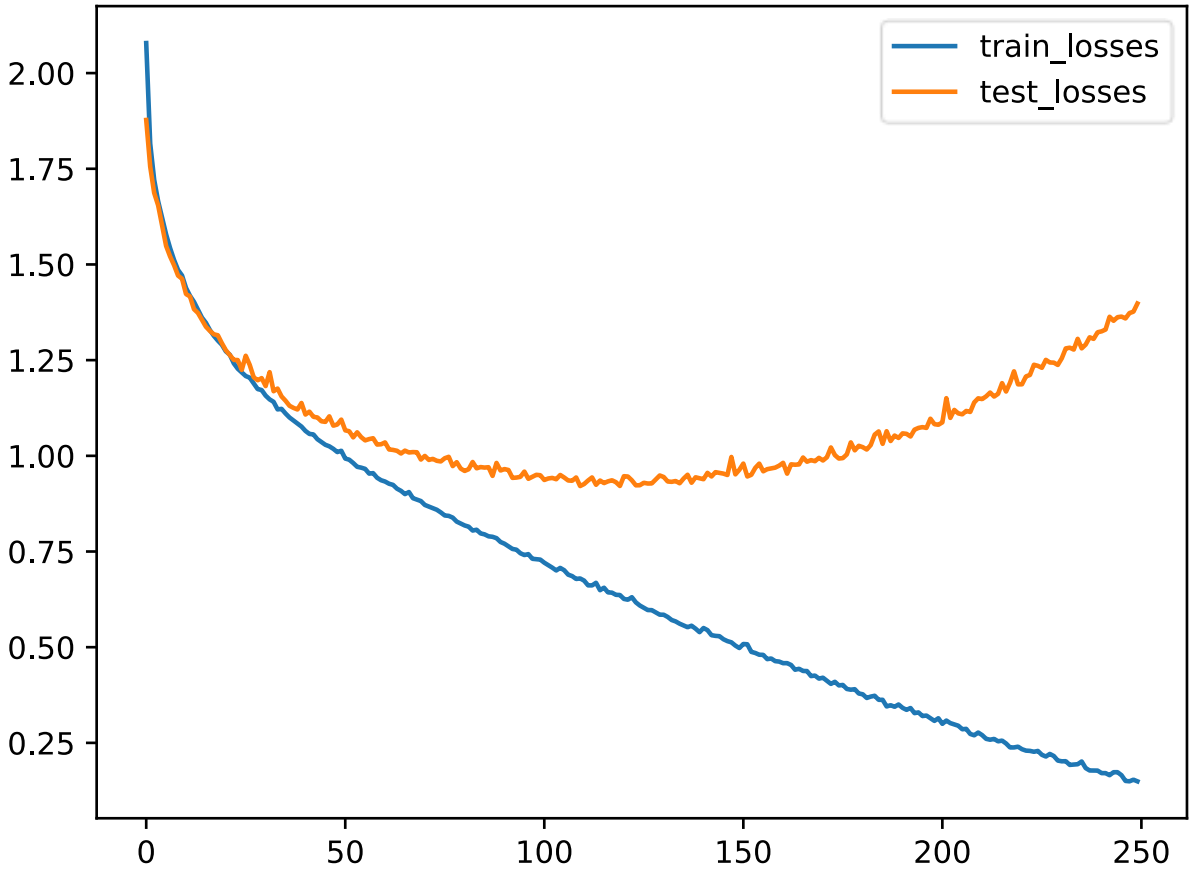


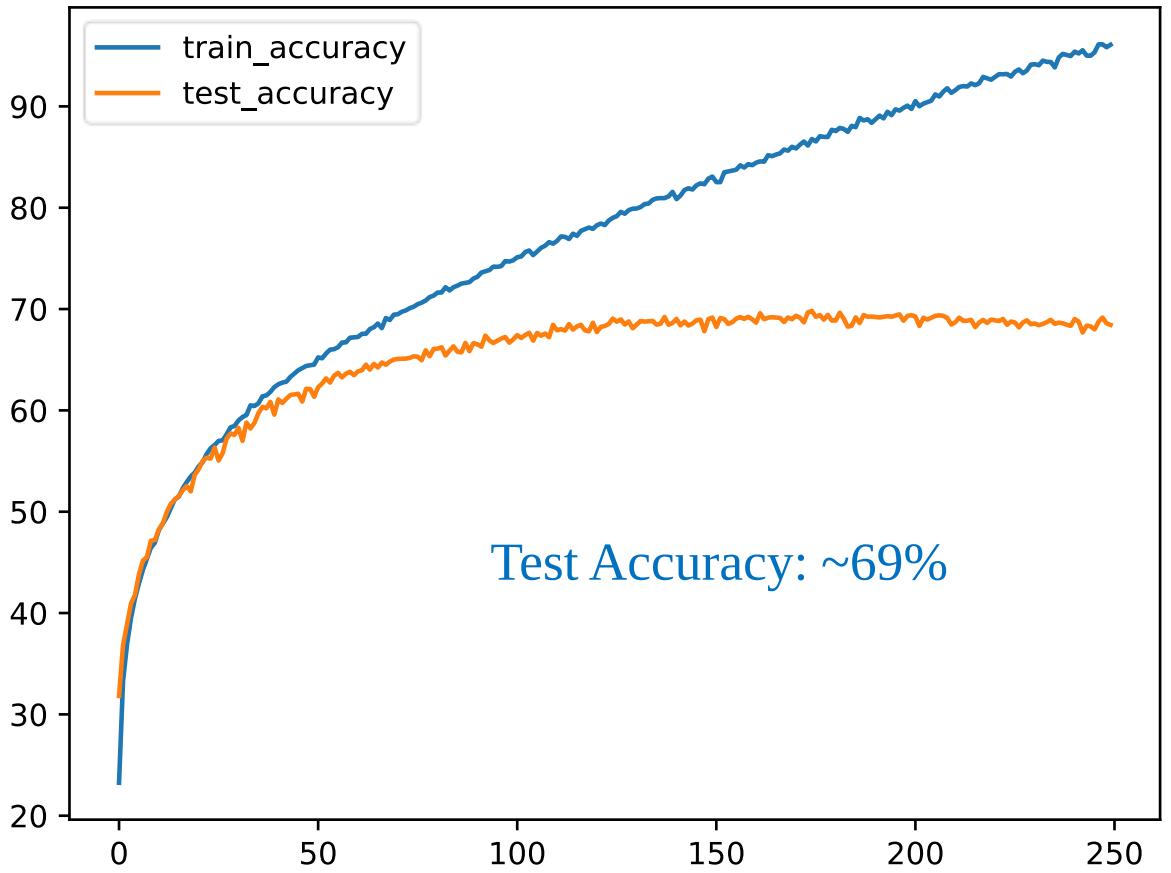
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128,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

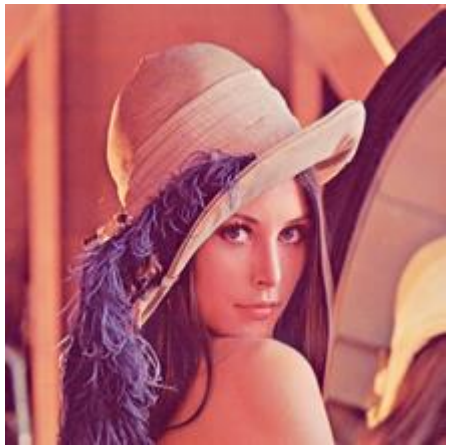
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

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}

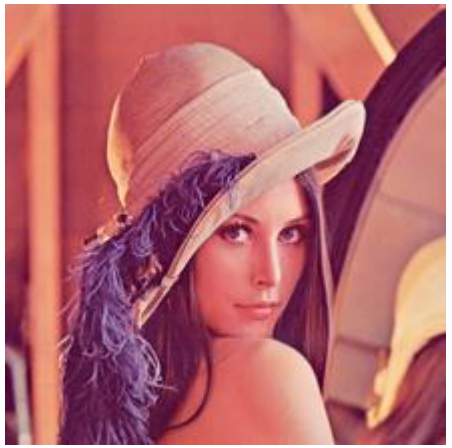
Data

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)

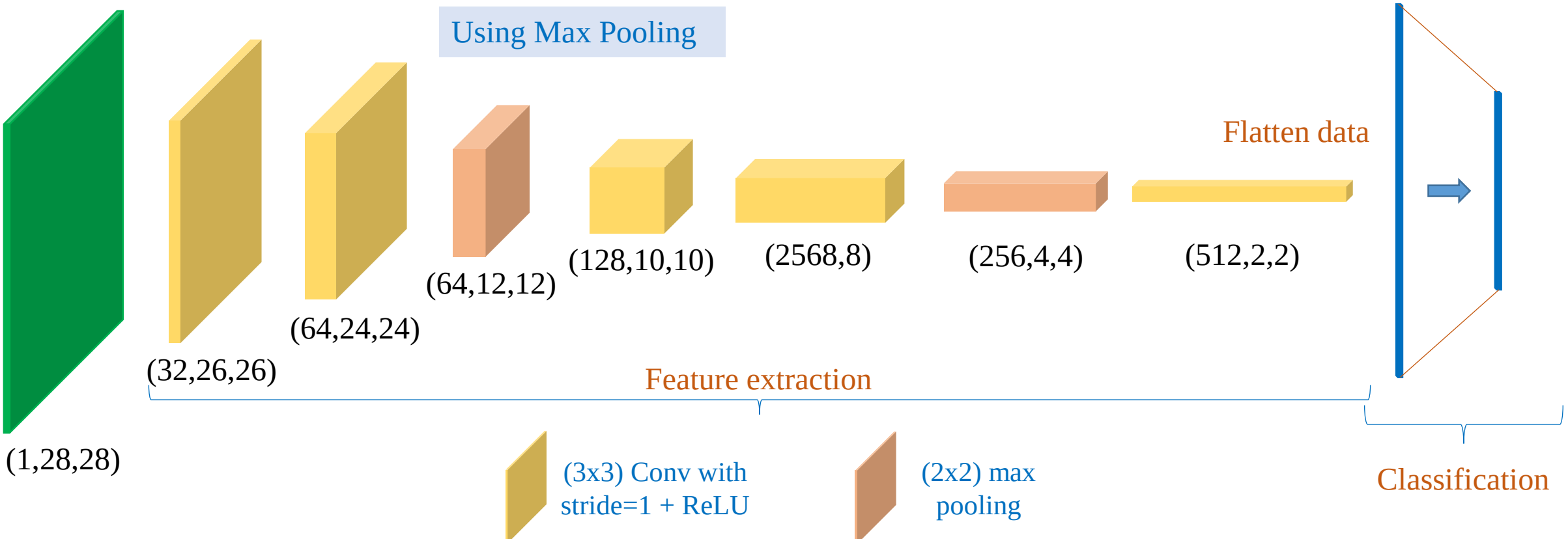
Feature map (110x110)

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**

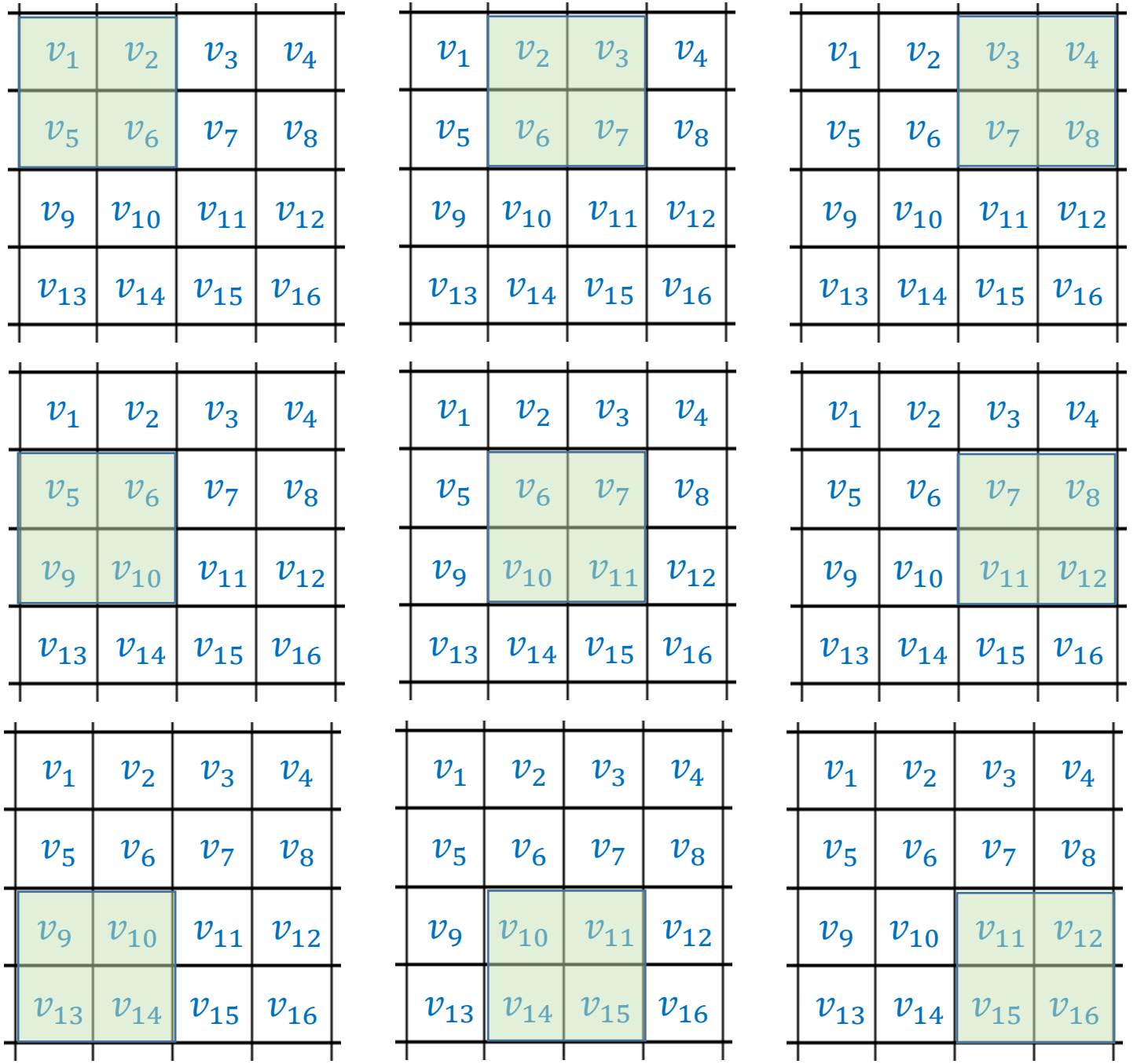
w_1	w_2	b
w_3	w_4	

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**

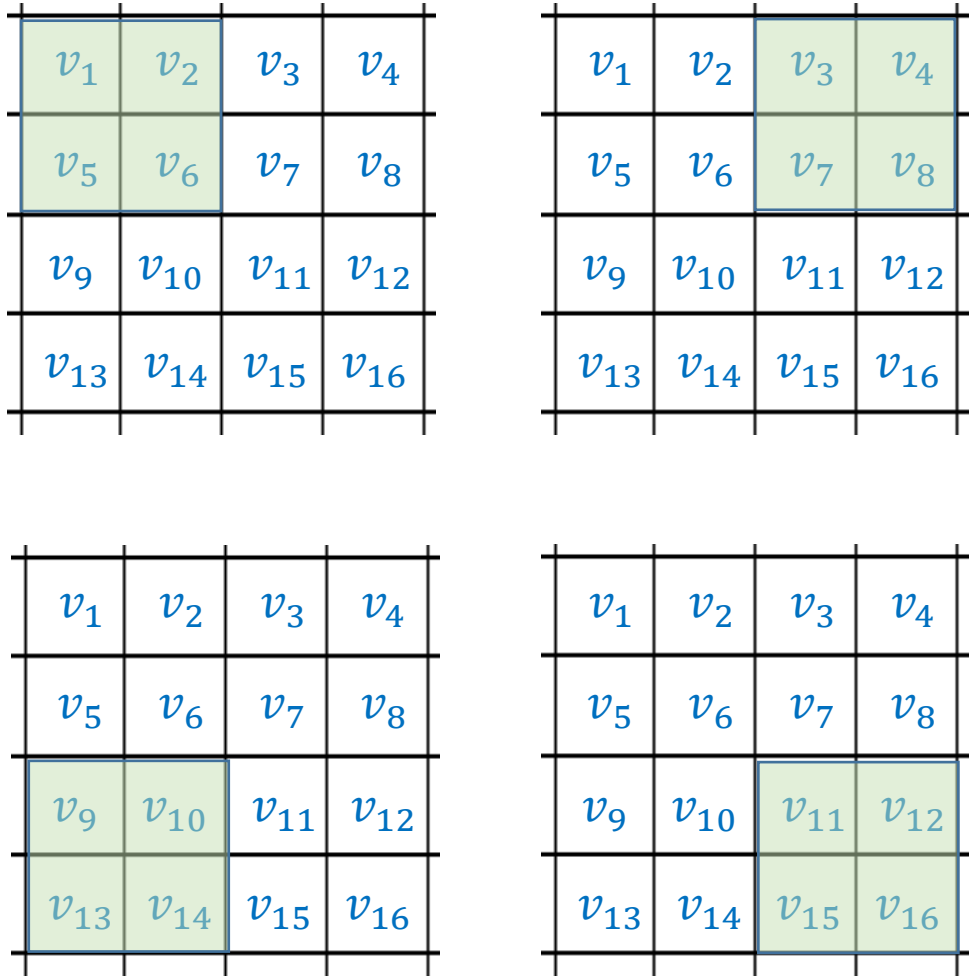
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=2

m_1	m_2
m_3	m_4

Output

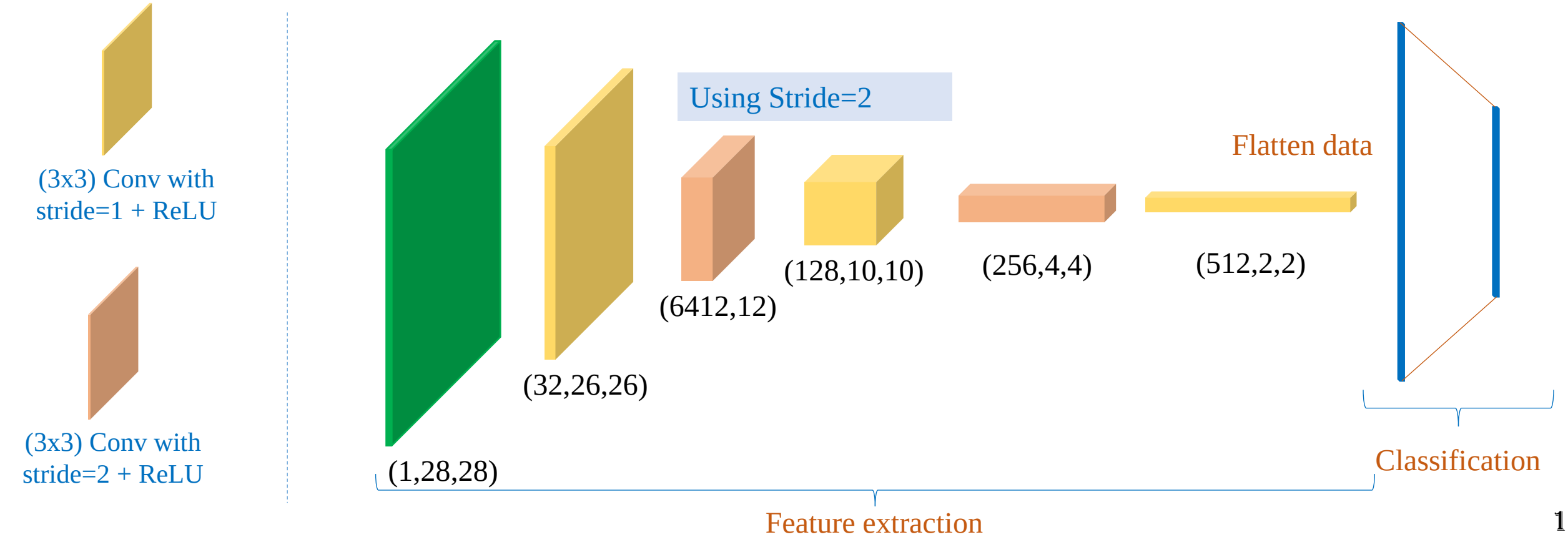


nn.Conv2d(in_channels, out_channels, kernel_size, stride=1)

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

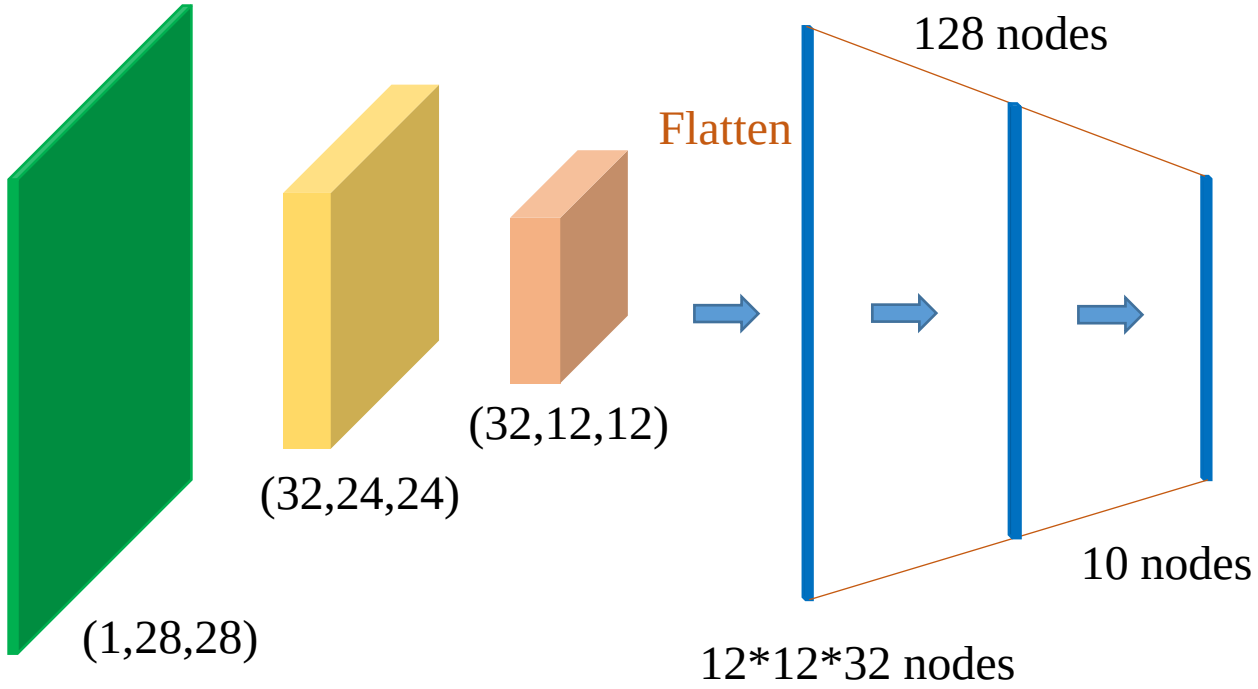
```python
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = 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

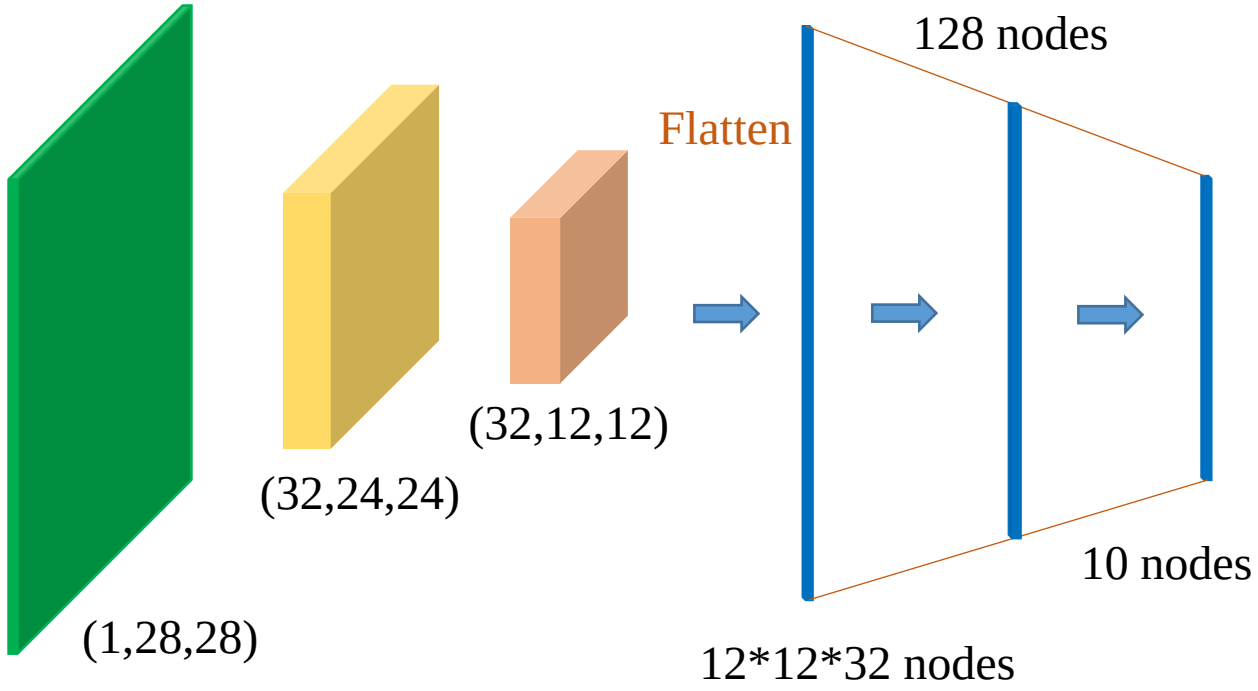
Implementation 1

```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

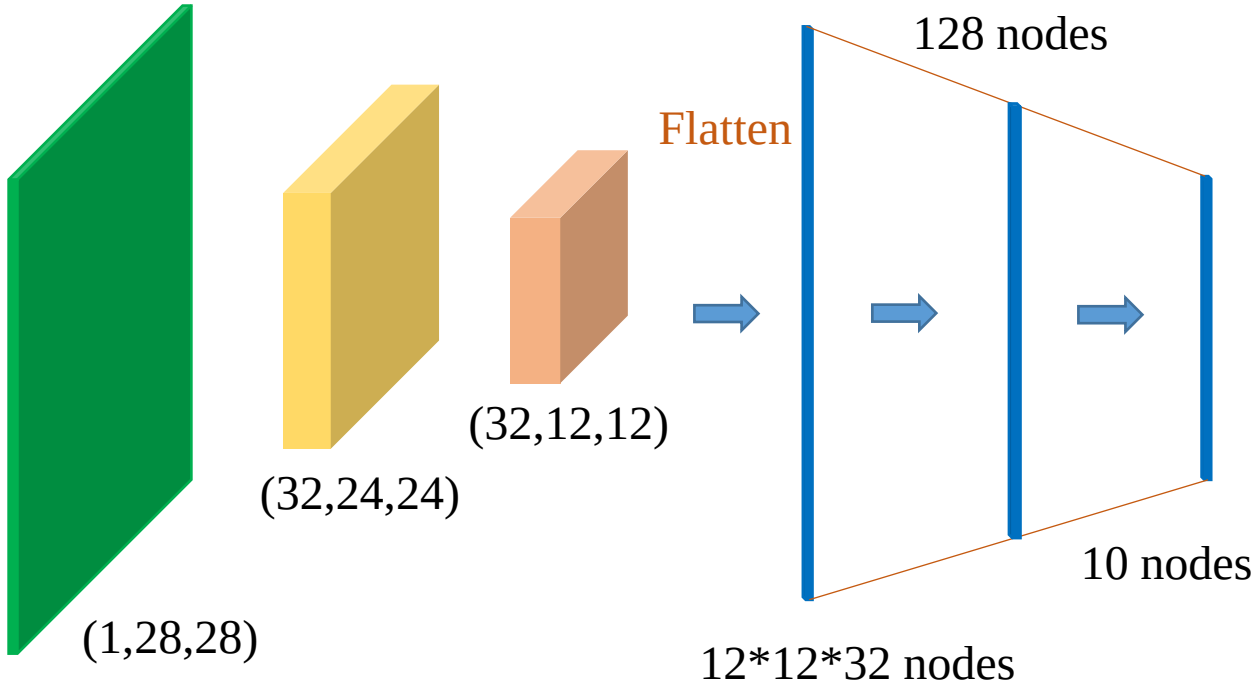
Implementation 2

```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

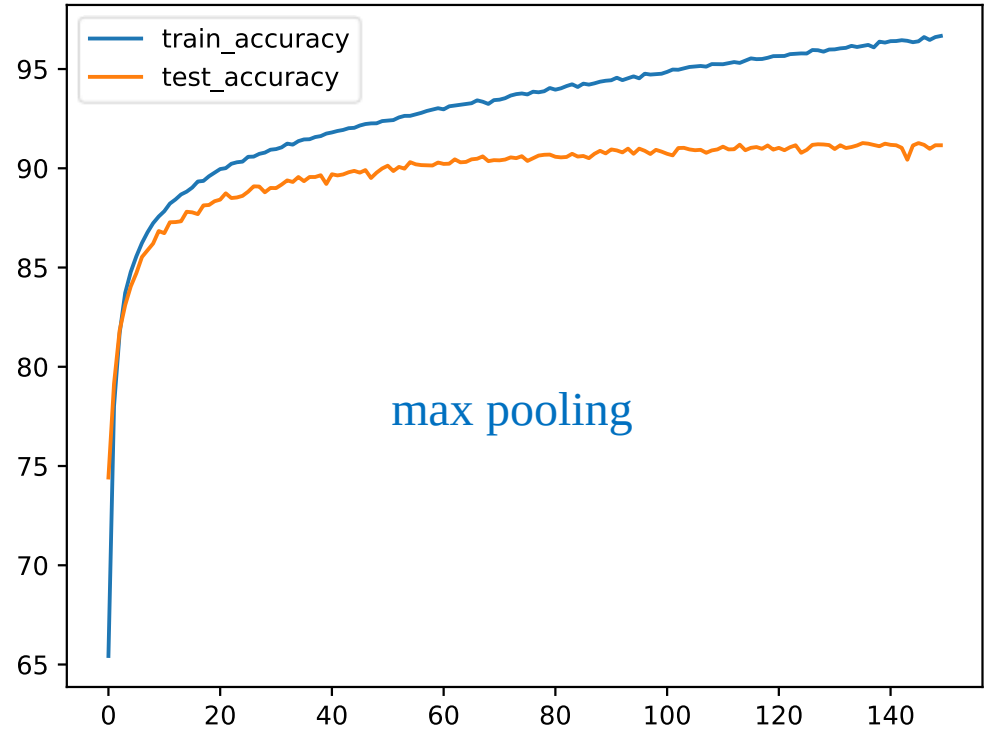
(2x2) down-sampling

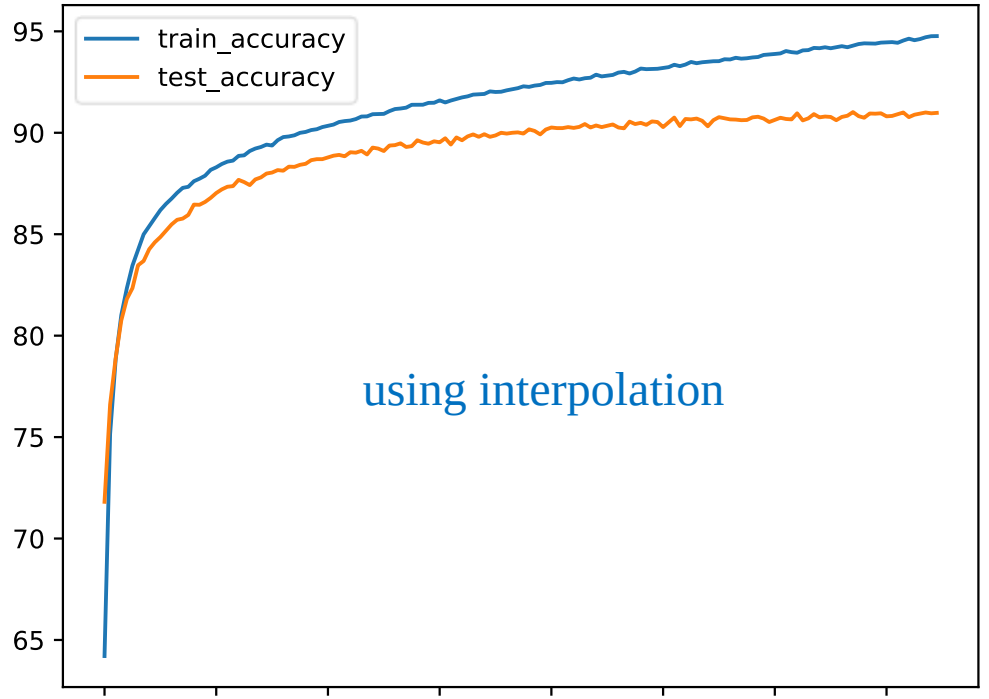
```python
class ResizeLayer(nn.Module):
    def __init__(self, scale_factor, mode='bilinear',
                 align_corners=False):
        super(ResizeLayer, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode
        self.align_corners = align_corners

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

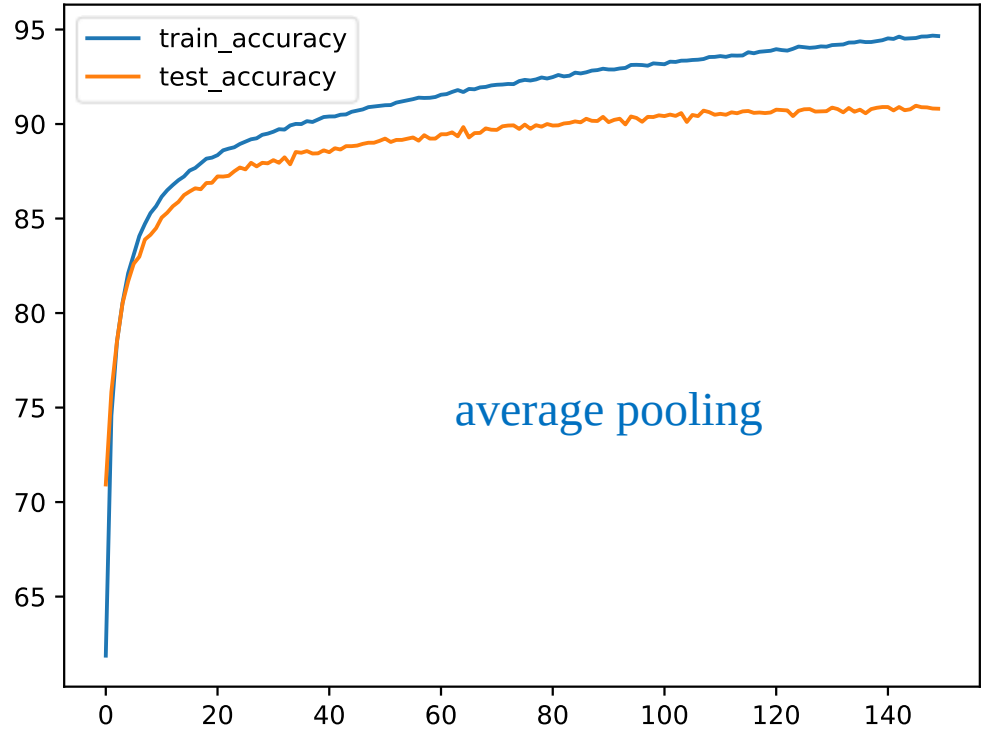
interpolation

```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 $(\boldsymbol{D}_p) =$

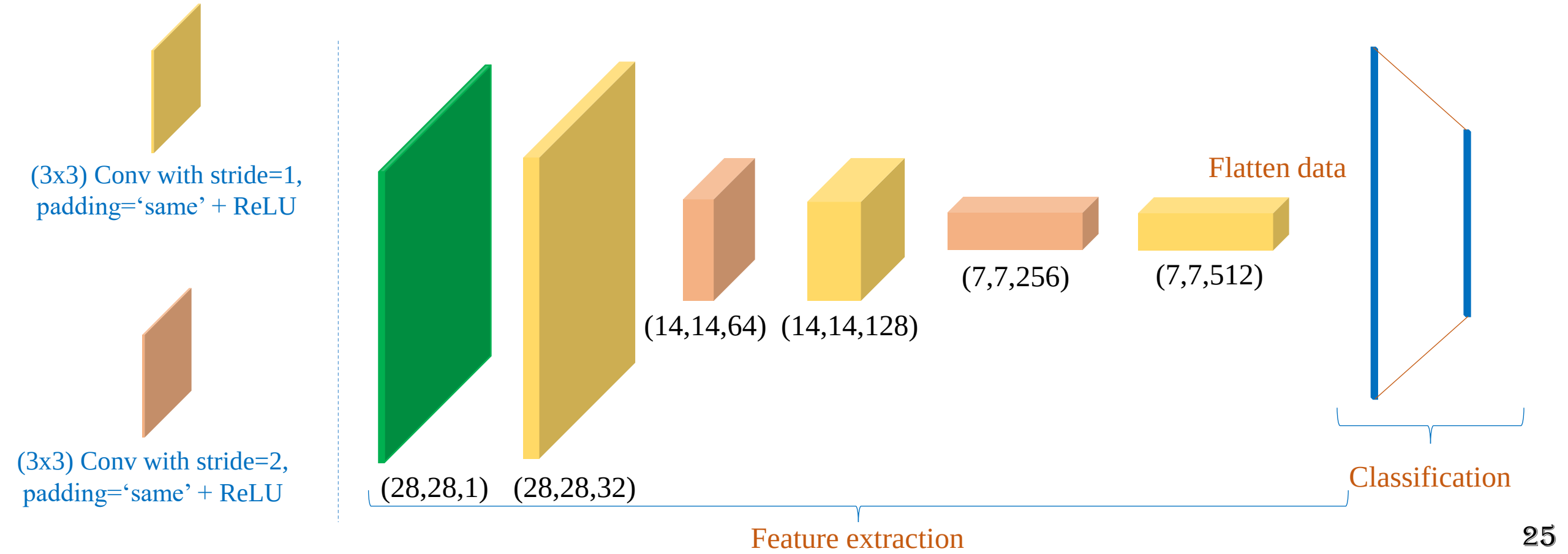
m_1	m_2	m_3	m_4
m_5	m_6	m_7	m_8
m_9	m_{10}	m_{11}	m_{12}
m_{13}	m_{14}	m_{15}	m_{16}

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)
)
```

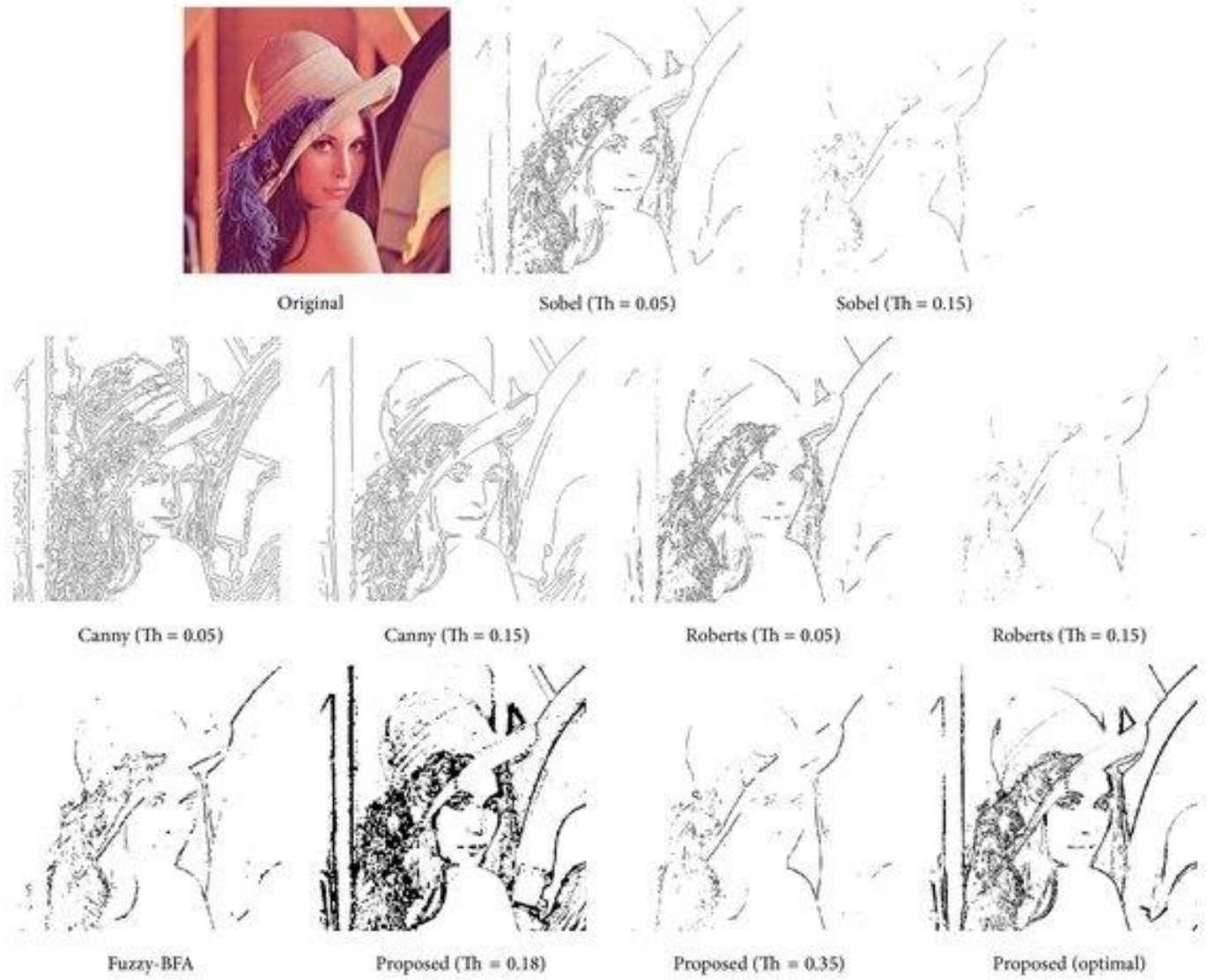


Outline

- 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

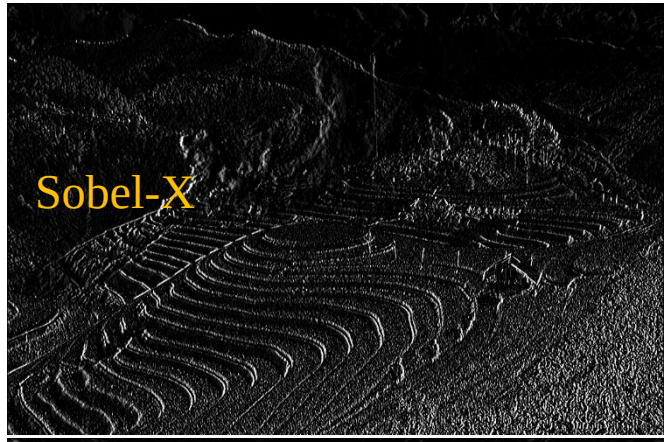
1				-1	0	1		-1	0	1
2	*	-1	0	1	=	-2	0	2
1						-1	0	1

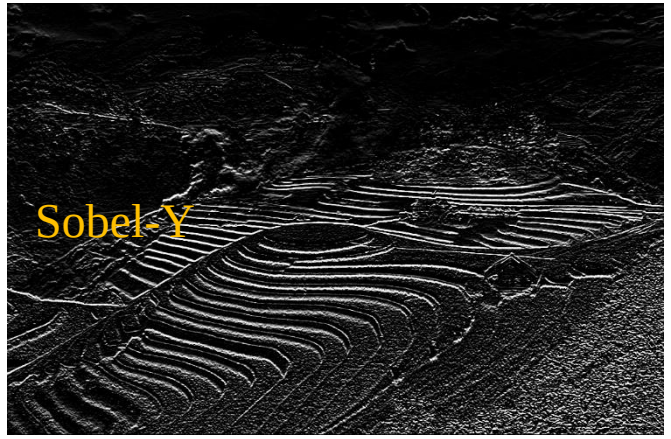
weighted average | x-derivative | Sobel for x direction

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



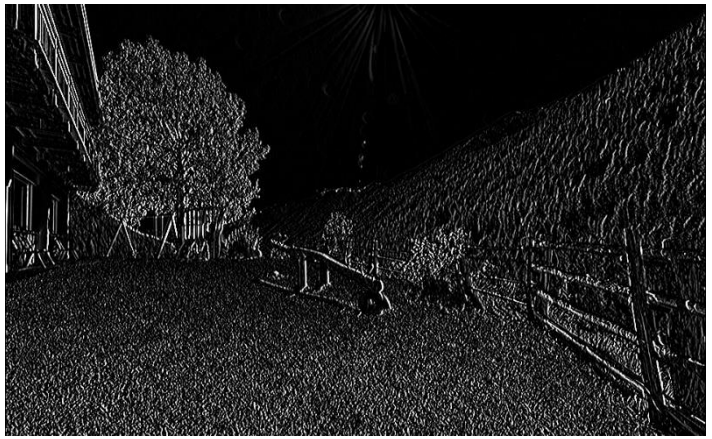


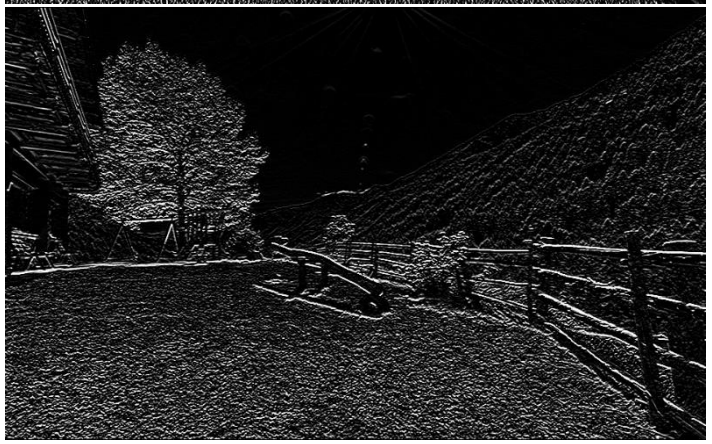
Derivative and Applications

Edge detection

Edge detection

Sobel-X





Sobel-Y

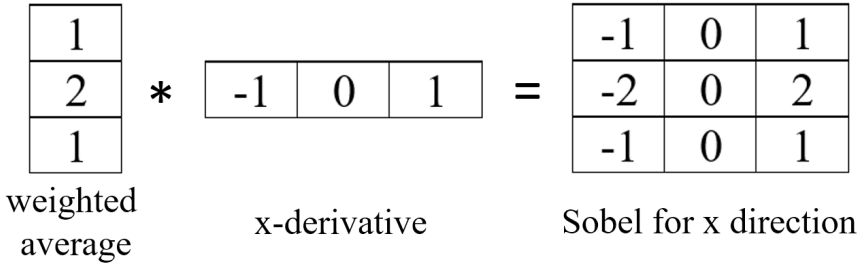
$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

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

Engineering Feature Extractors

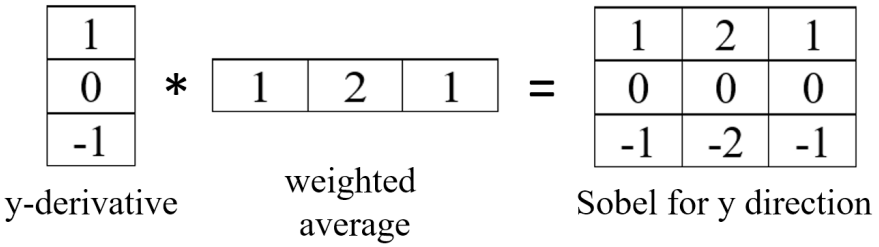
❖ Edge detection

Tính đạo hàm trung bình theo hướng x

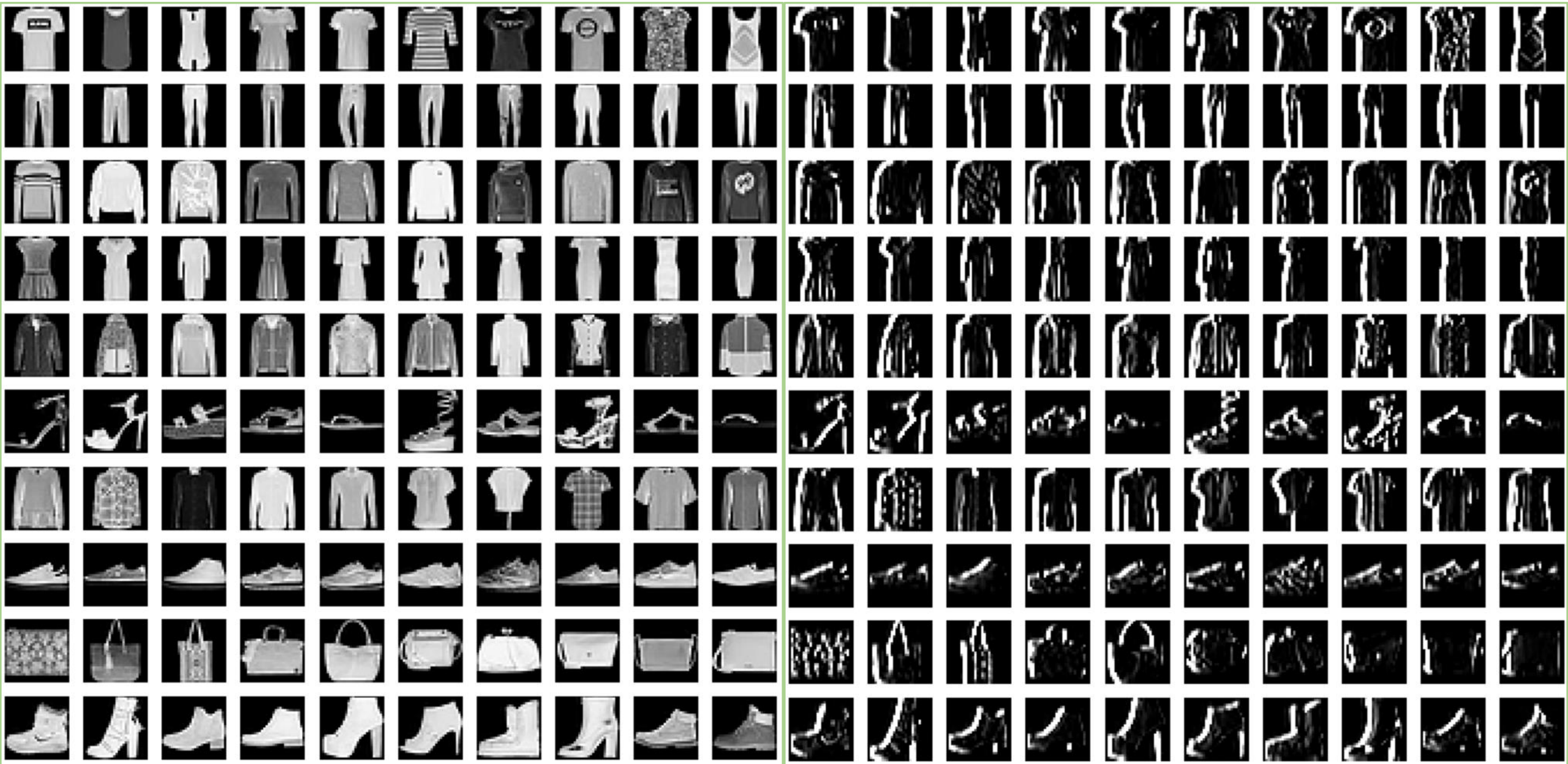



```python
@staticmethod
def _sobel(image, ksize=3):
    sobel_x = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=ksize)
    sobel_y = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=ksize)
    sobel_x = torch.from_numpy(sobel_x)
    sobel_y = torch.from_numpy(sobel_y)
    sobel_magnitude = torch.hypot(sobel_x, sobel_y)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

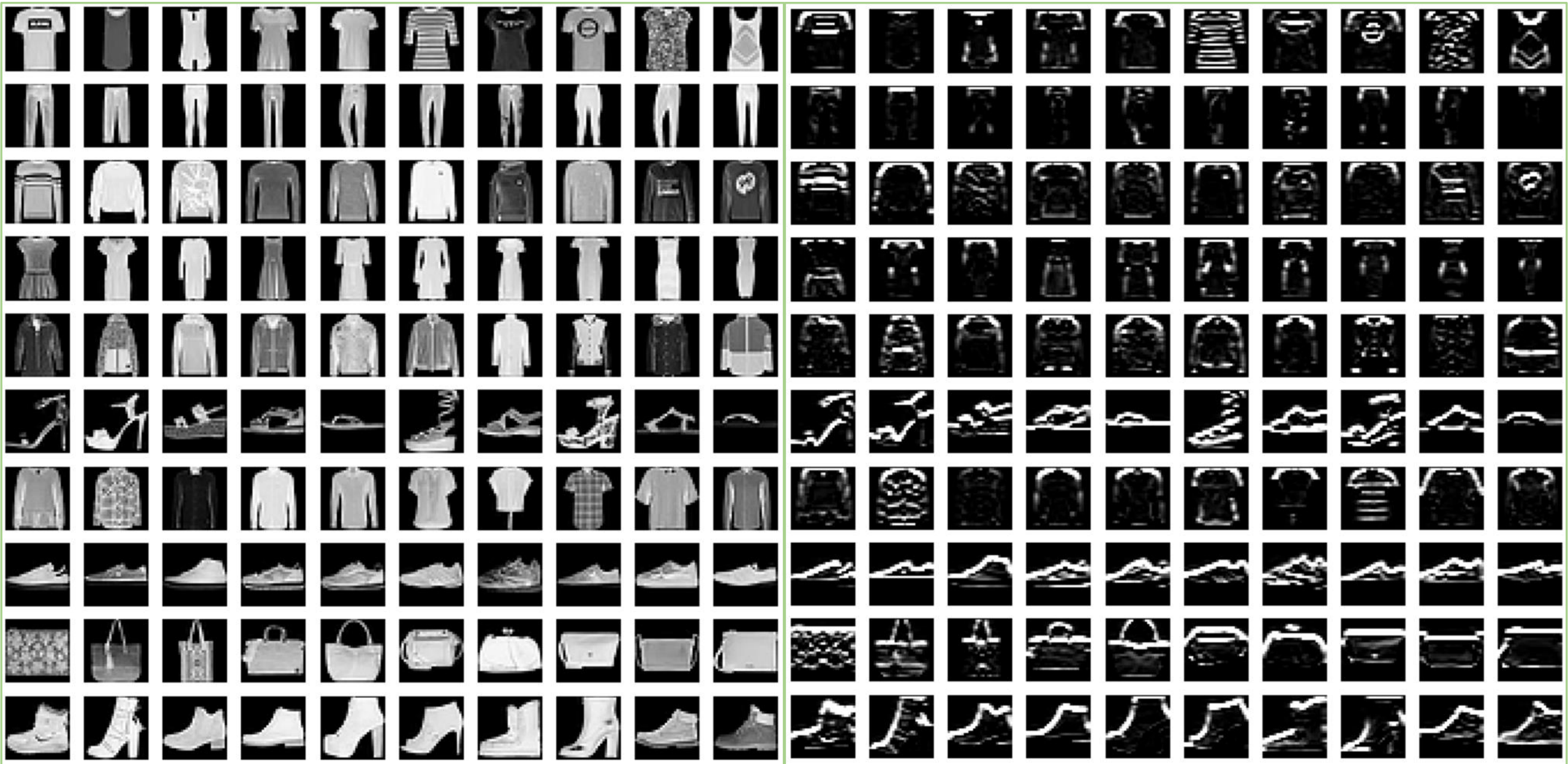
Tính đạo hàm trung bình theo hướng y



Sobel_X

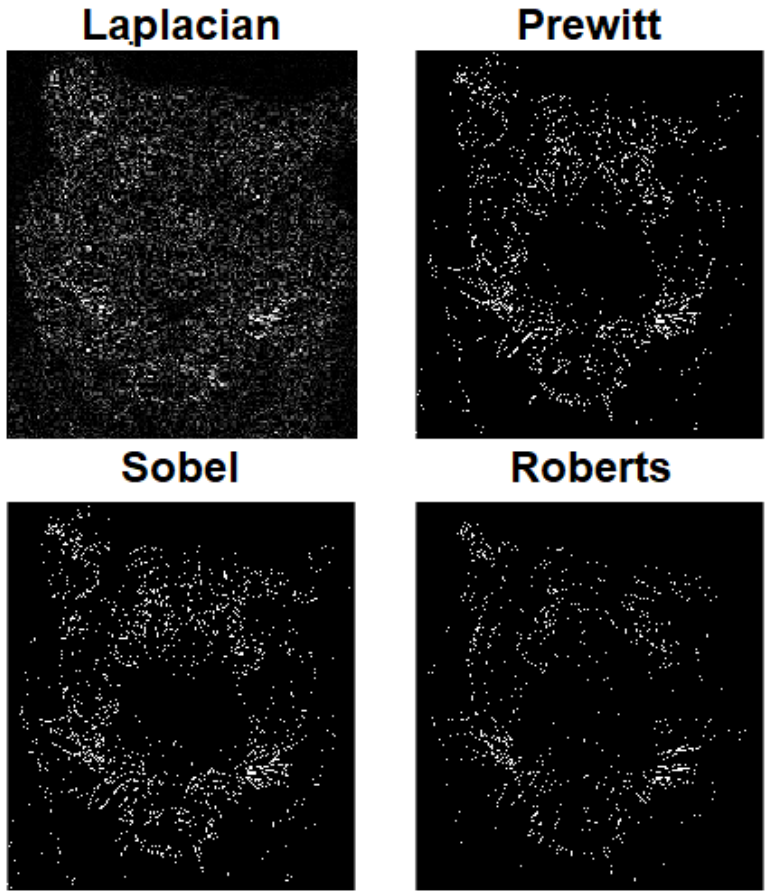


Sobel_Y



Engineering Feature Extractors

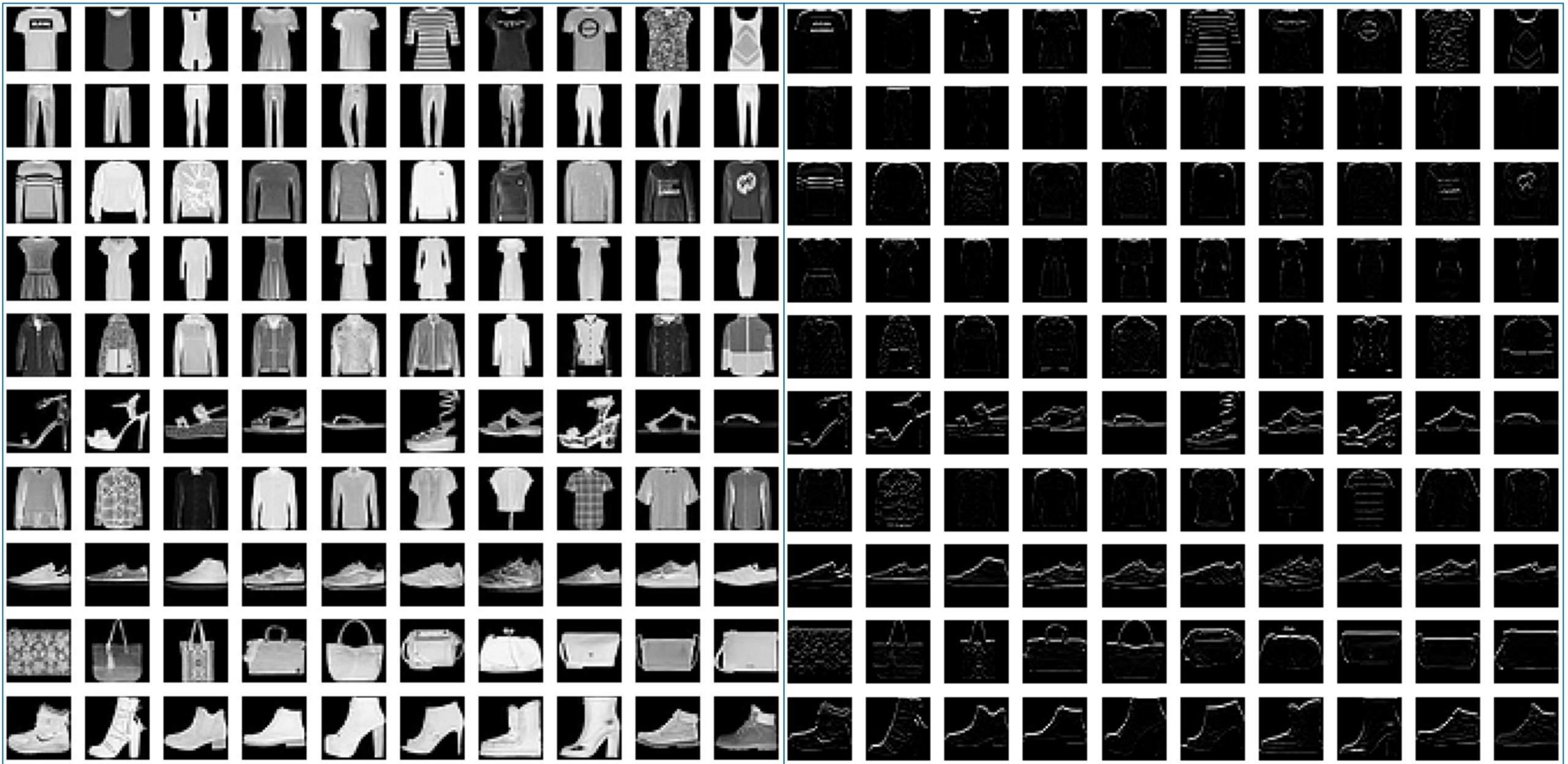
❖ Traditional methods



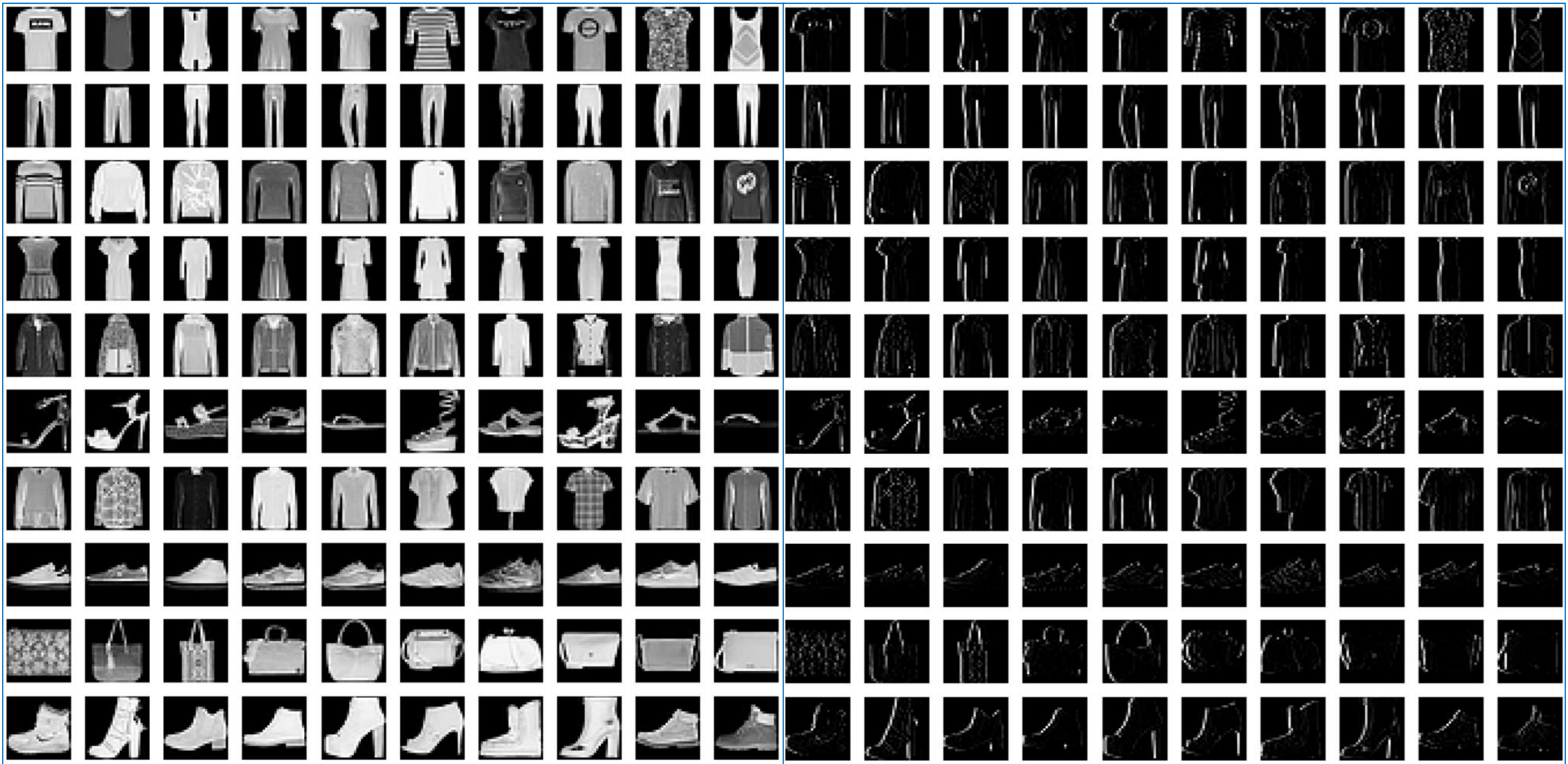

```python
@staticmethod
def _scharr(image):
    scharr_x = cv2.Scharr(image, cv2.CV_32F, 1, 0)
    scharr_y = cv2.Scharr(image, cv2.CV_32F, 0, 1)
    scharr_x = torch.from_numpy(scharr_x)
    scharr_y = torch.from_numpy(scharr_y)
    scharr_magnitude = torch.hypot(scharr_x, scharr_y)
    return scharr_magnitude

@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

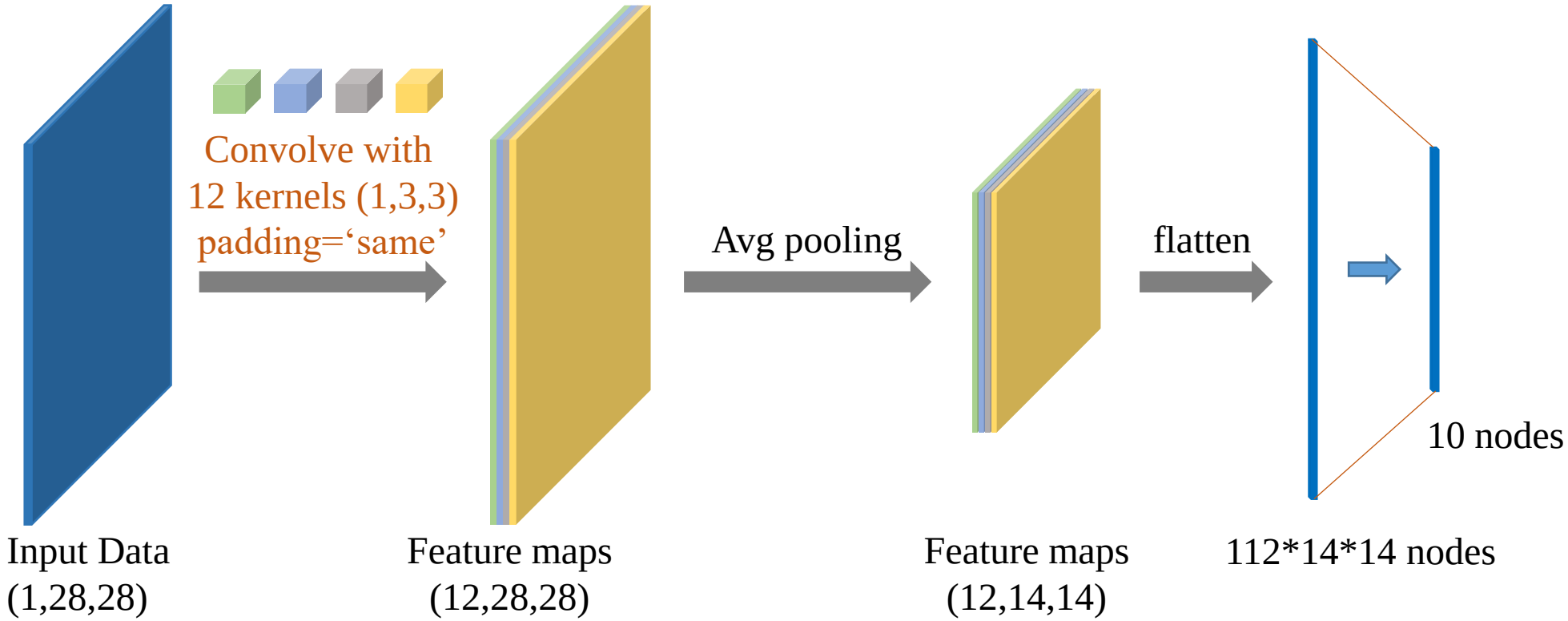


Gradient_Y



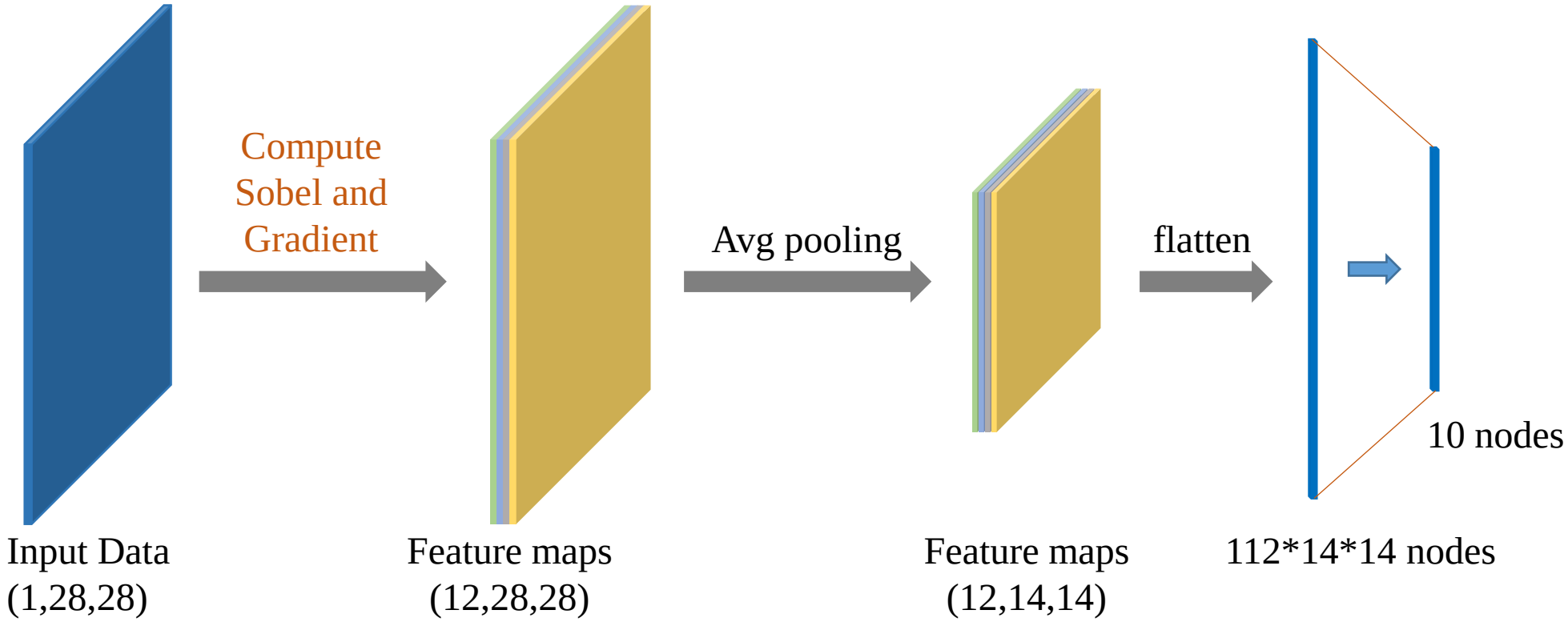
Comparison

❖ Engineering features vs. CNN features

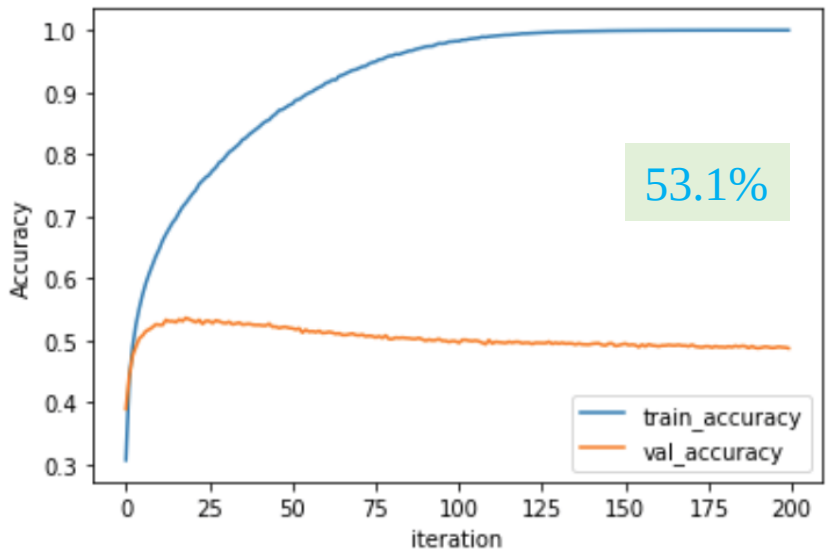


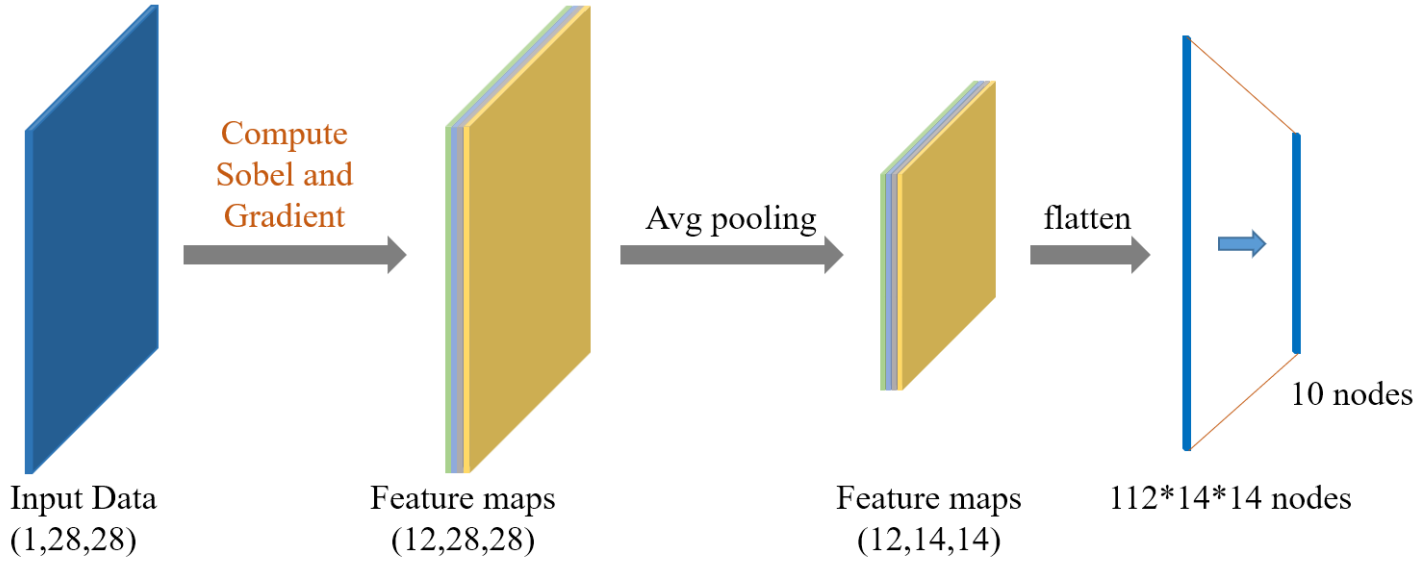
Comparison

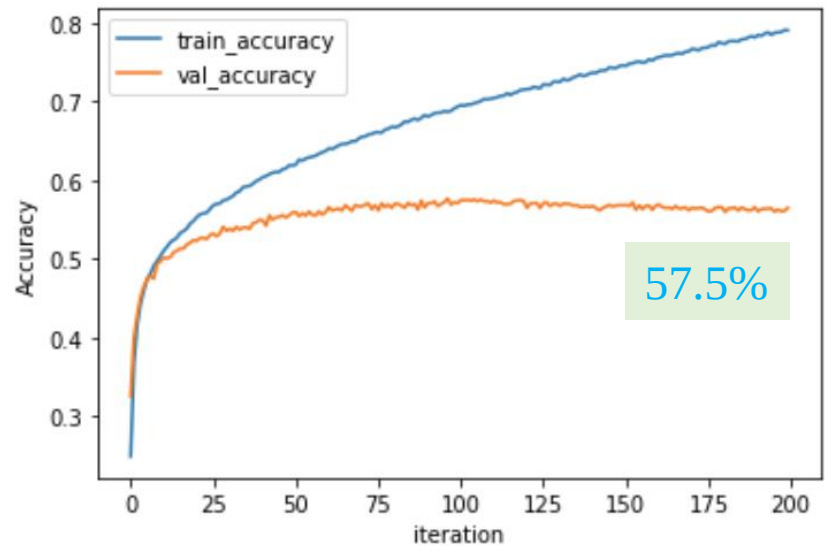
❖ Engineering features vs. CNN features

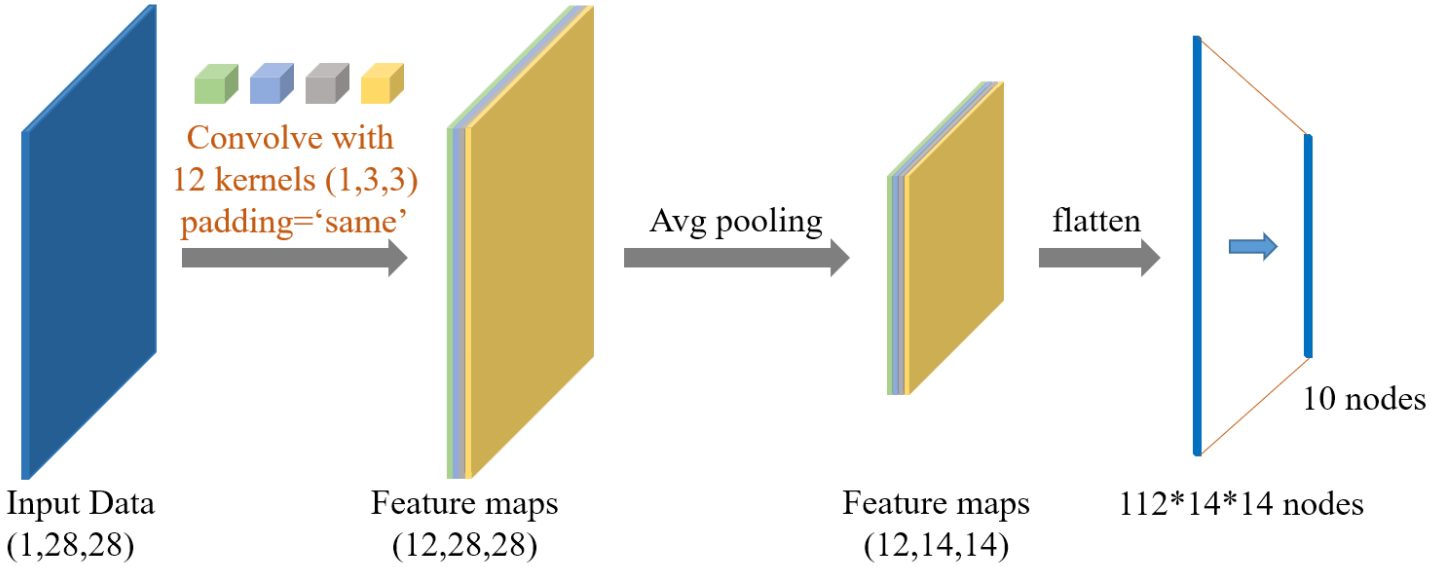


Engineering Features vs. CNN Features using Cifar10









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

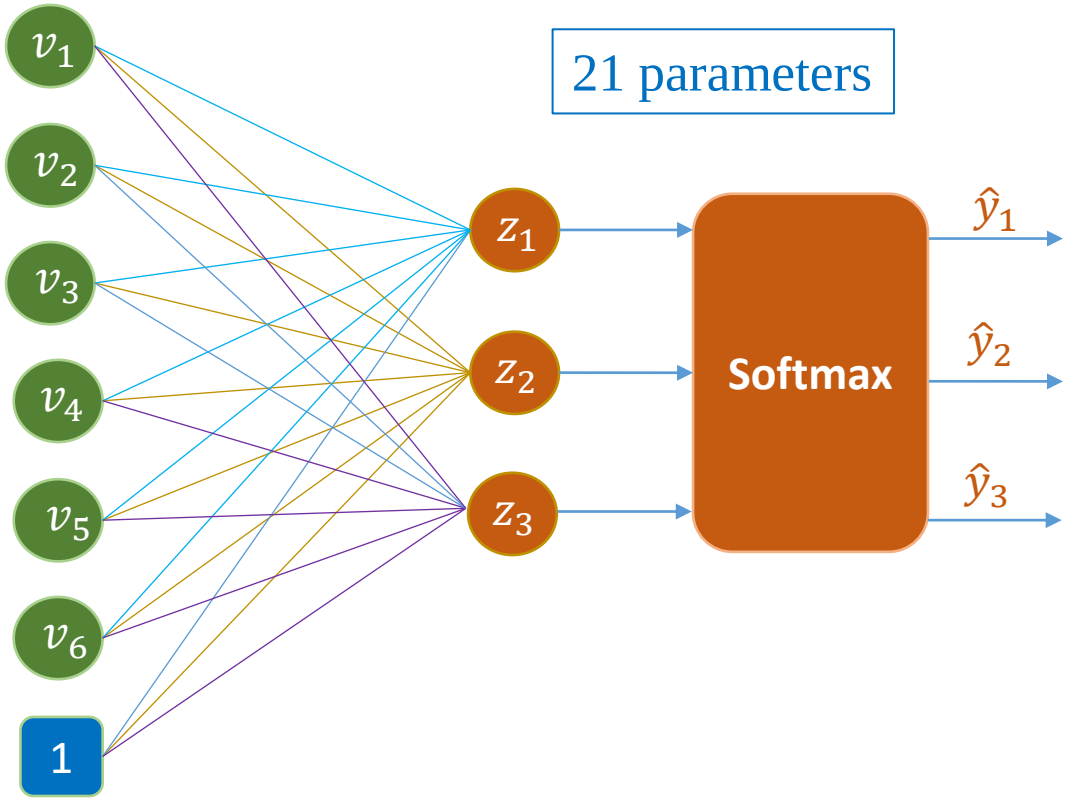
Born	July 8, 1960 (age 60) Soisy-sous-Montmorency, France
Alma mater	ESIEE Paris (MSc) Pierre and Marie Curie University (PhD)
Known for	Deep learning
Awards	Turing Award (2018) AAAI Fellow (2019) Legion of Honour (2020)
	Scientific career
Institutions	Bell Labs (1988-1996) New York University Facebook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

1x1 Convolution

❖ Comparison

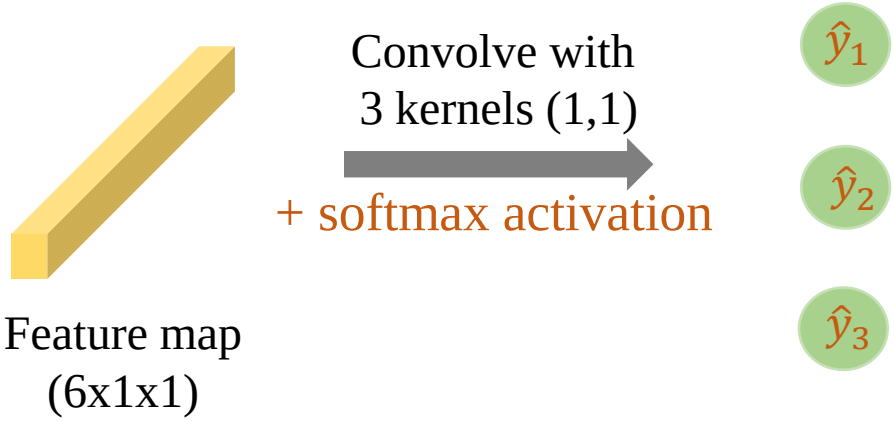
Fully connected layer

21 parameters



CNN

21 parameters



1x1 Convolution

Replace FC by 1x1 Conv

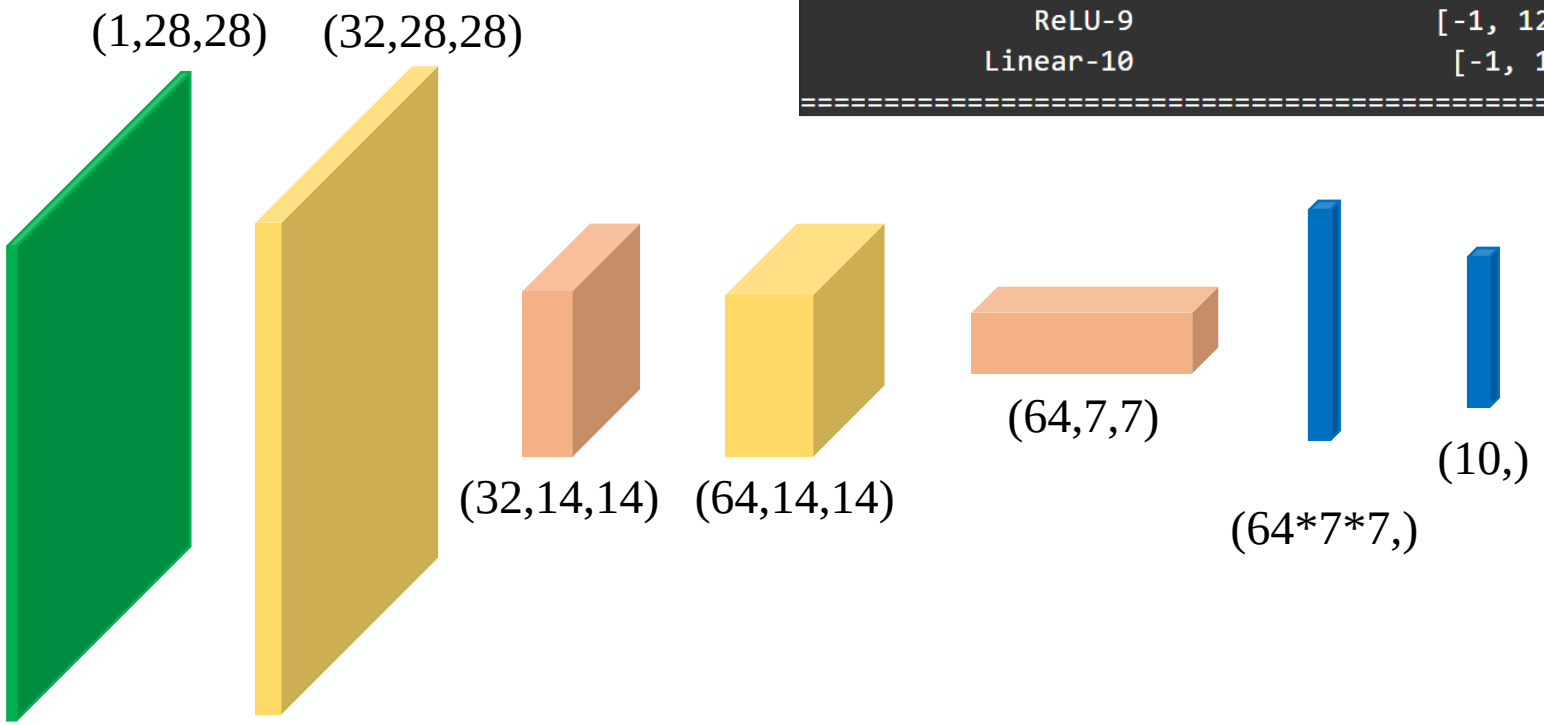
(3x3) Conv with stride=1, padding='same' + ReLU

(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

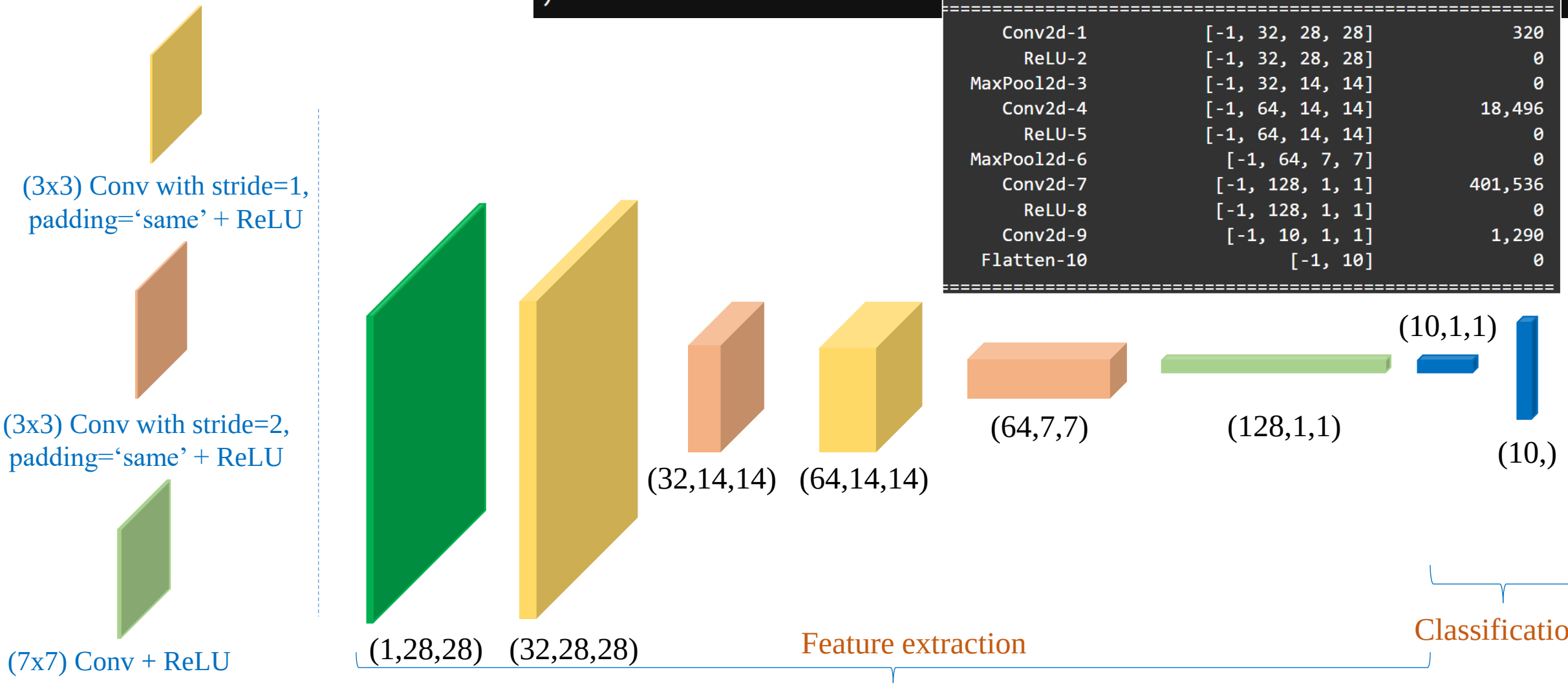


1x1 Convolution

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

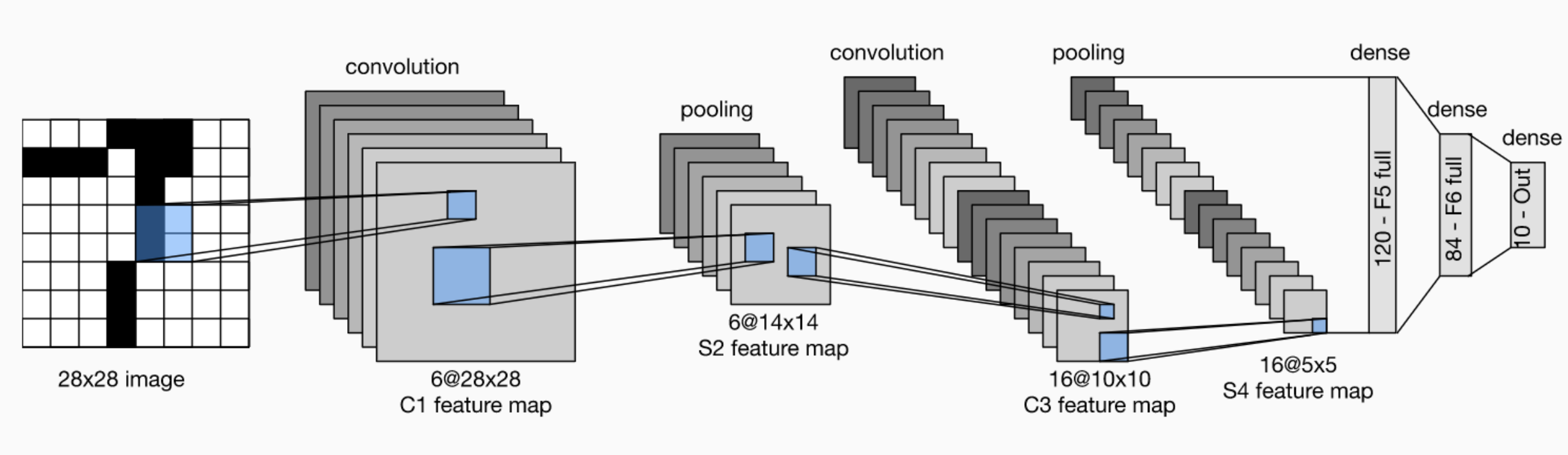


Outline

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

LeNet Architecture

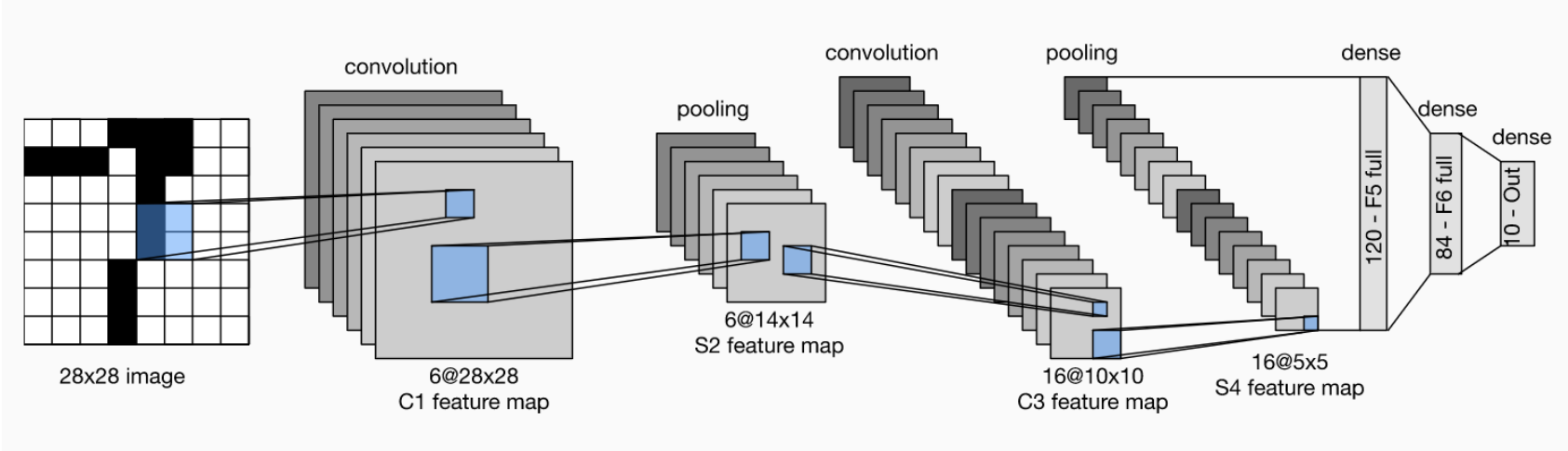
❖ Model Construction



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

LeNet Architecture

❖ Model Construction

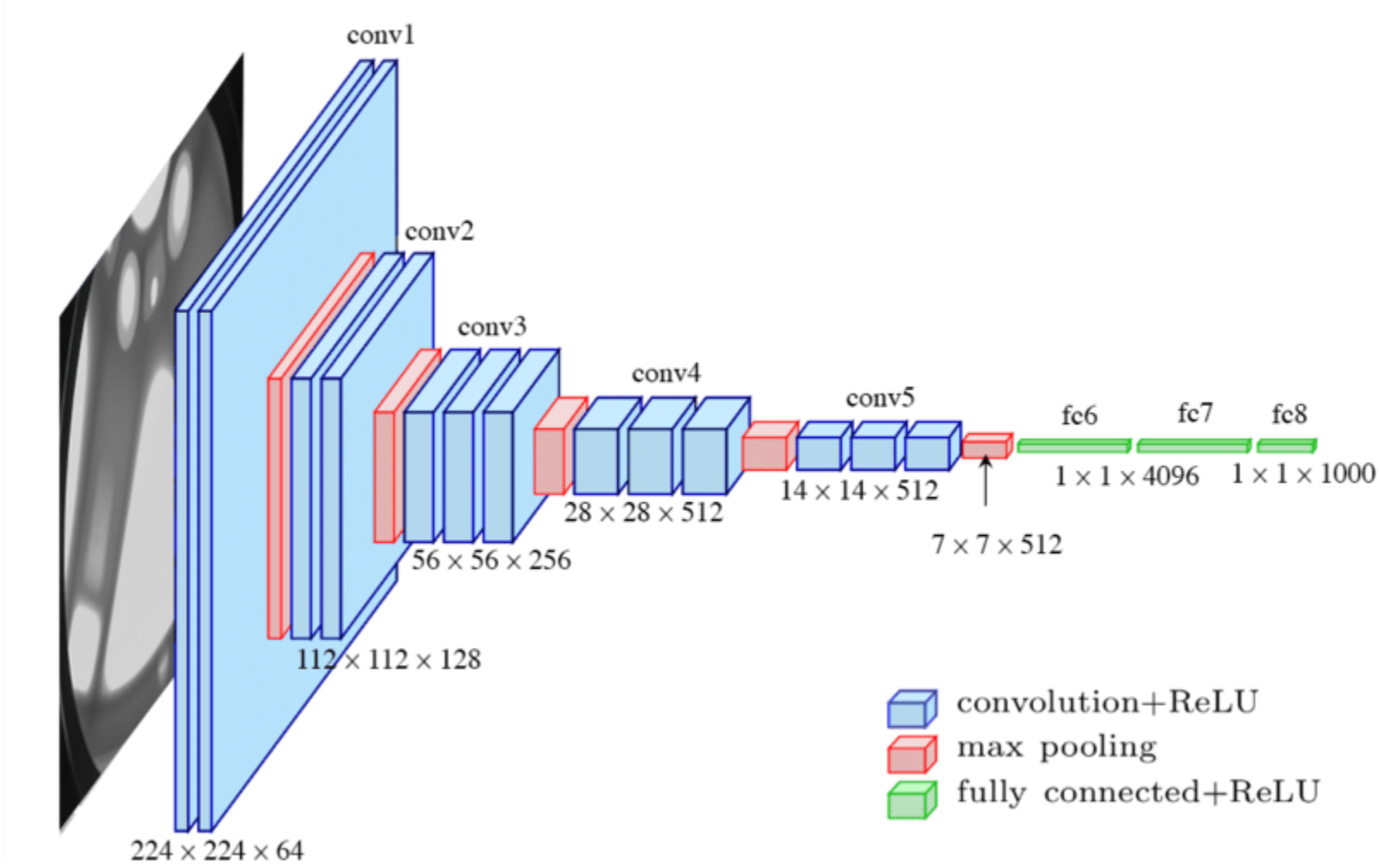



```
LeNet = nn.Sequential(
    nn.Conv2d(1, 6, 5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),

    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

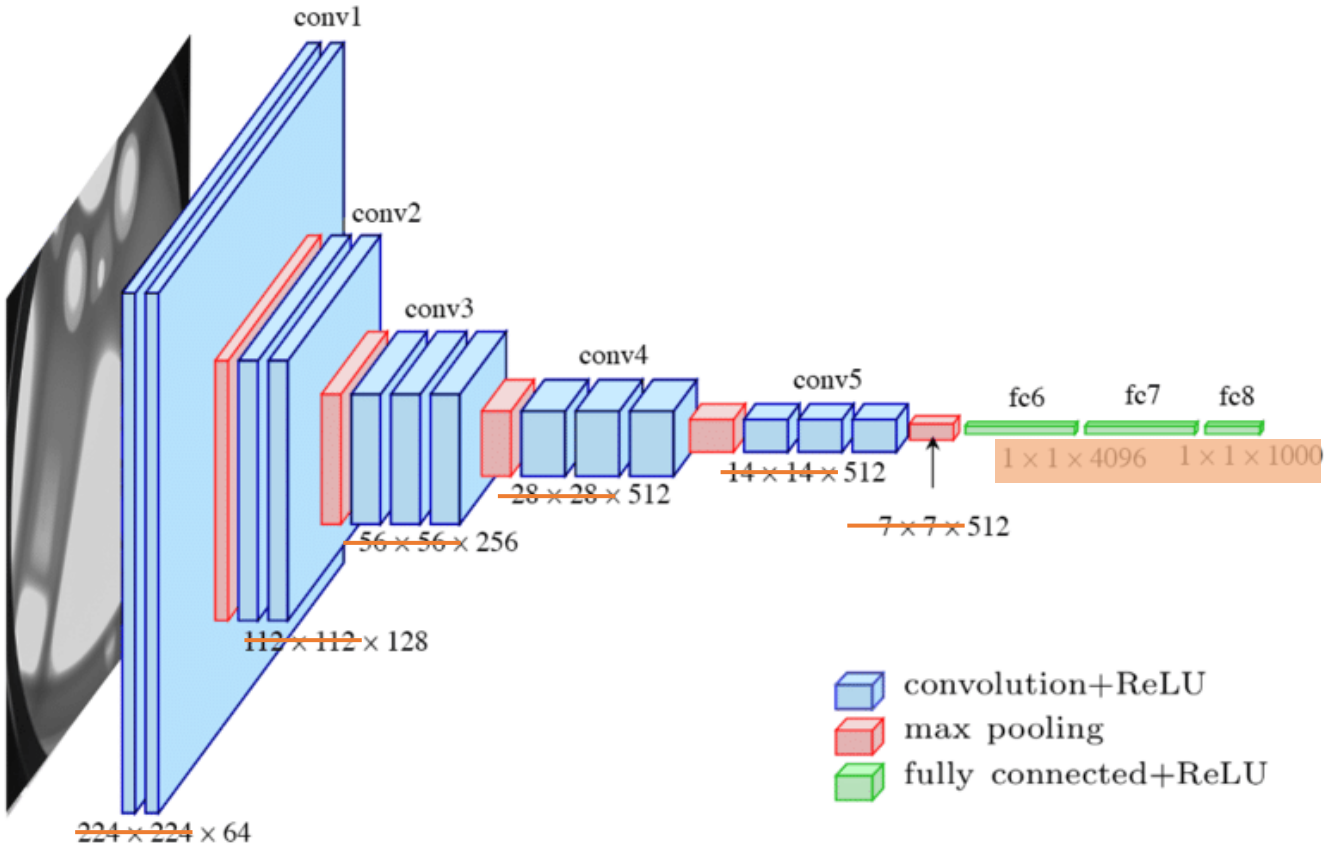
❖ Model Construction



```
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