AI VIETNAM
All-in-One Course

CNN Training

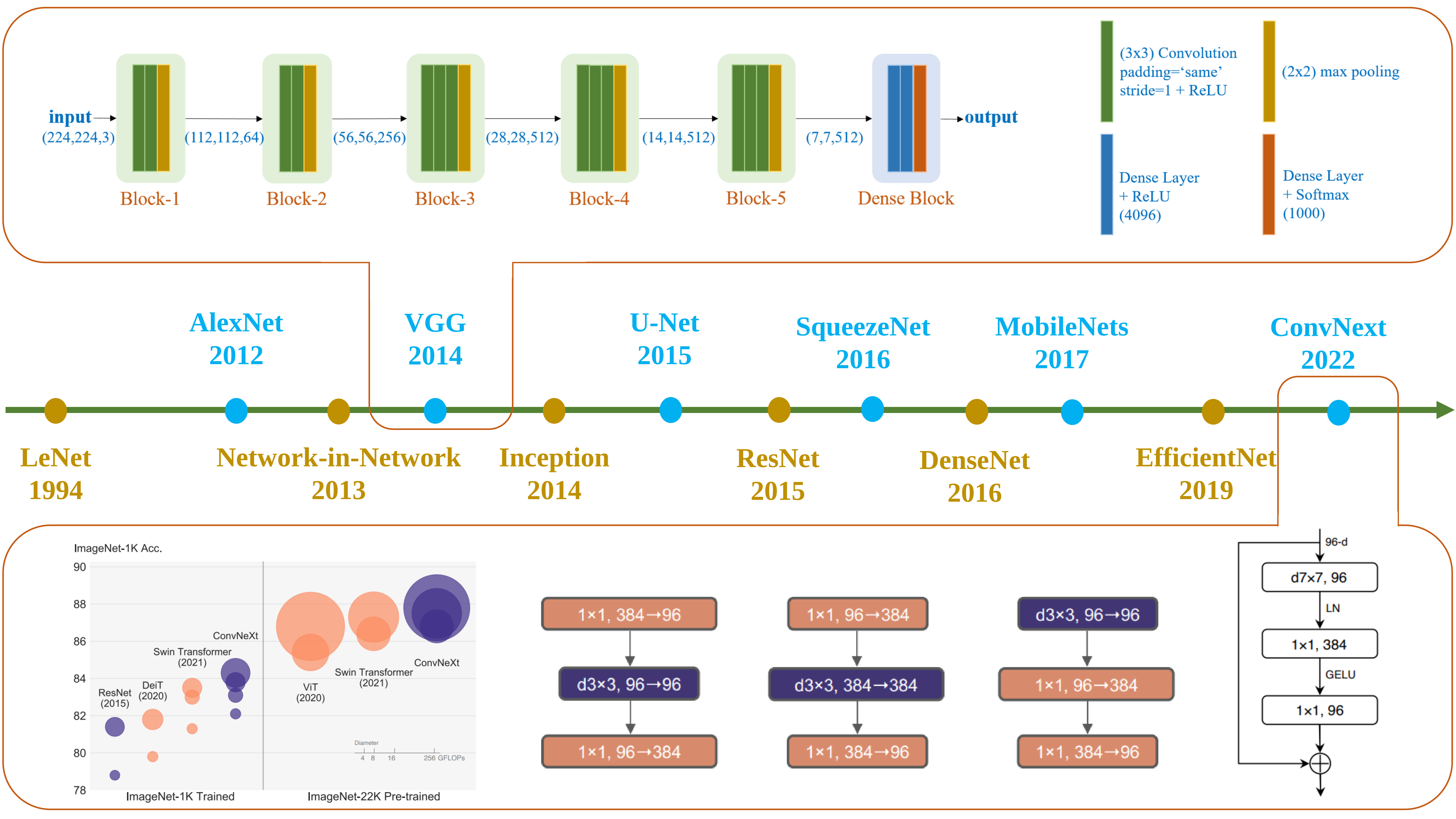
How to increase training accuracy?

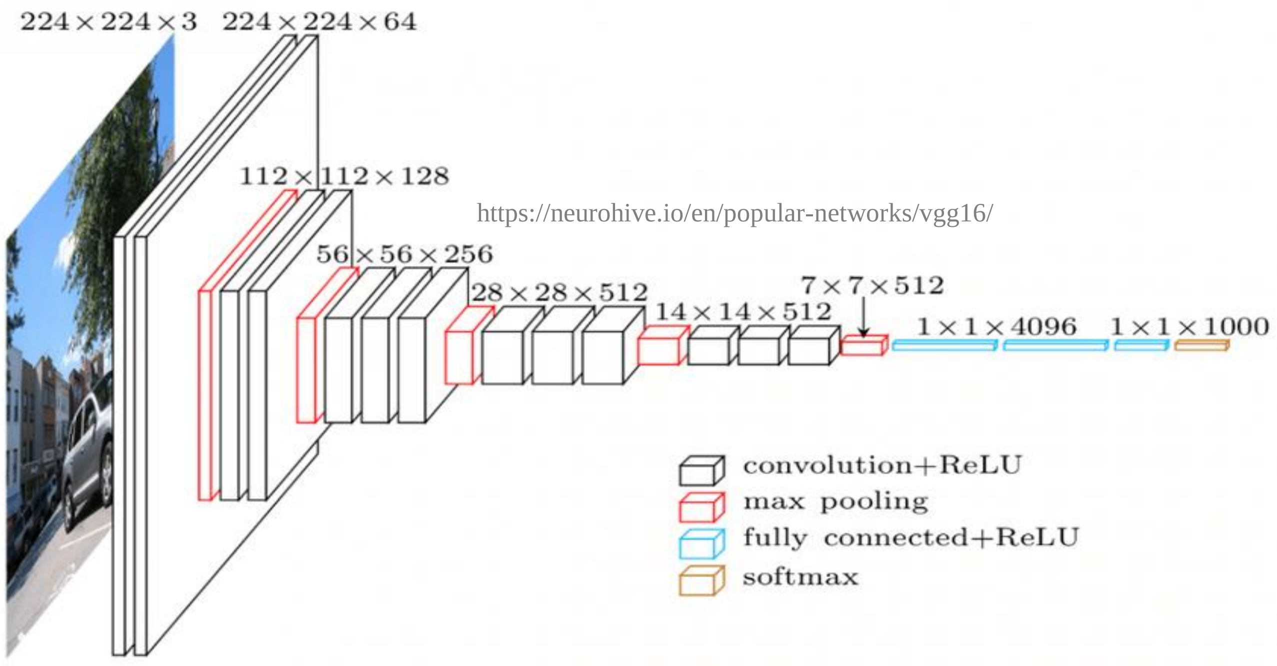
Quang-Vinh Dinh
Ph.D. in Computer Science

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)
AlexNet 2012
VGG 2014
U-Net 2015
SqueezeNet 2016
MobileNets 2017
ConvNext 2022
LeNet 1994
Network-in-Network 2013
Inception 2014
ResNet 2015
DenseNet 2016
EfficientNet 2019
ImageNet-1K Acc.
90
88
86
84
82
80
78
ResNet (2015)
DeiT (2020)
Swin Transformer (2021)
ConvNeXt
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



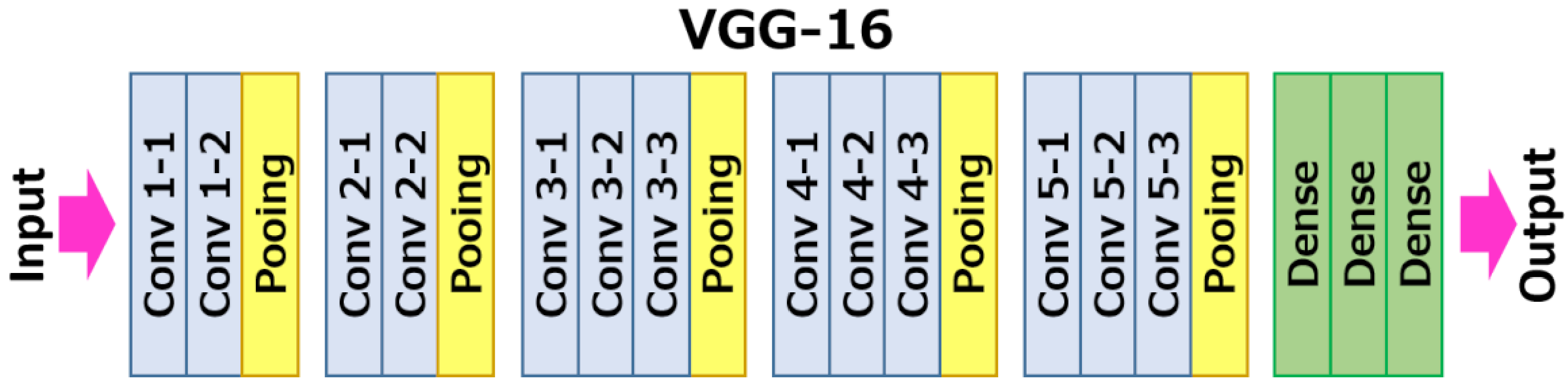
(3x3) Convolution
padding=‘same’
stride=1 + ReLU

(2x2) max pooling

Dense Layer
+ ReLU
(4096)

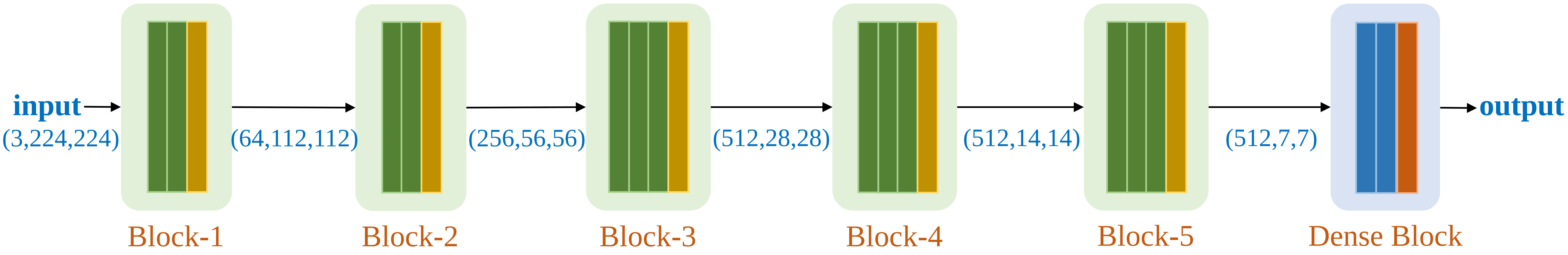
Dense Layer
+ Softmax
(1000)

VGG16



CNN Architectures

❖ VGG16 for ImageNet

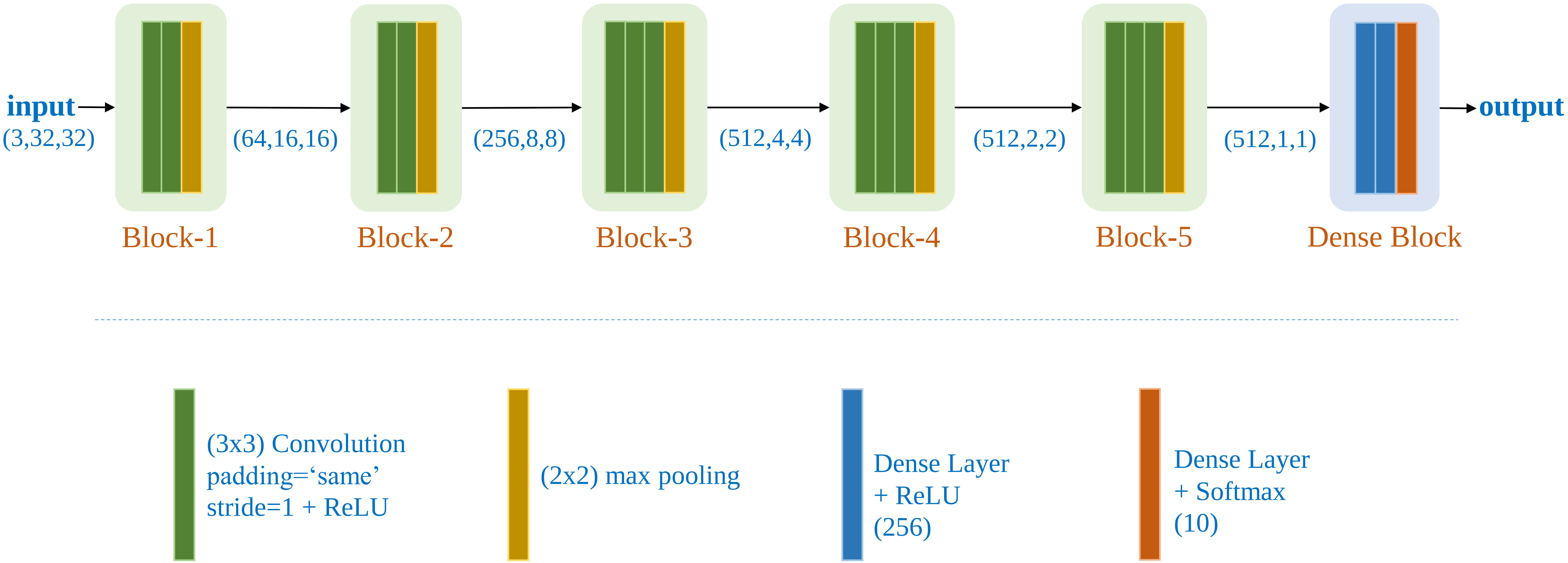


```python
# Define the blocks
block1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
block2 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=3,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

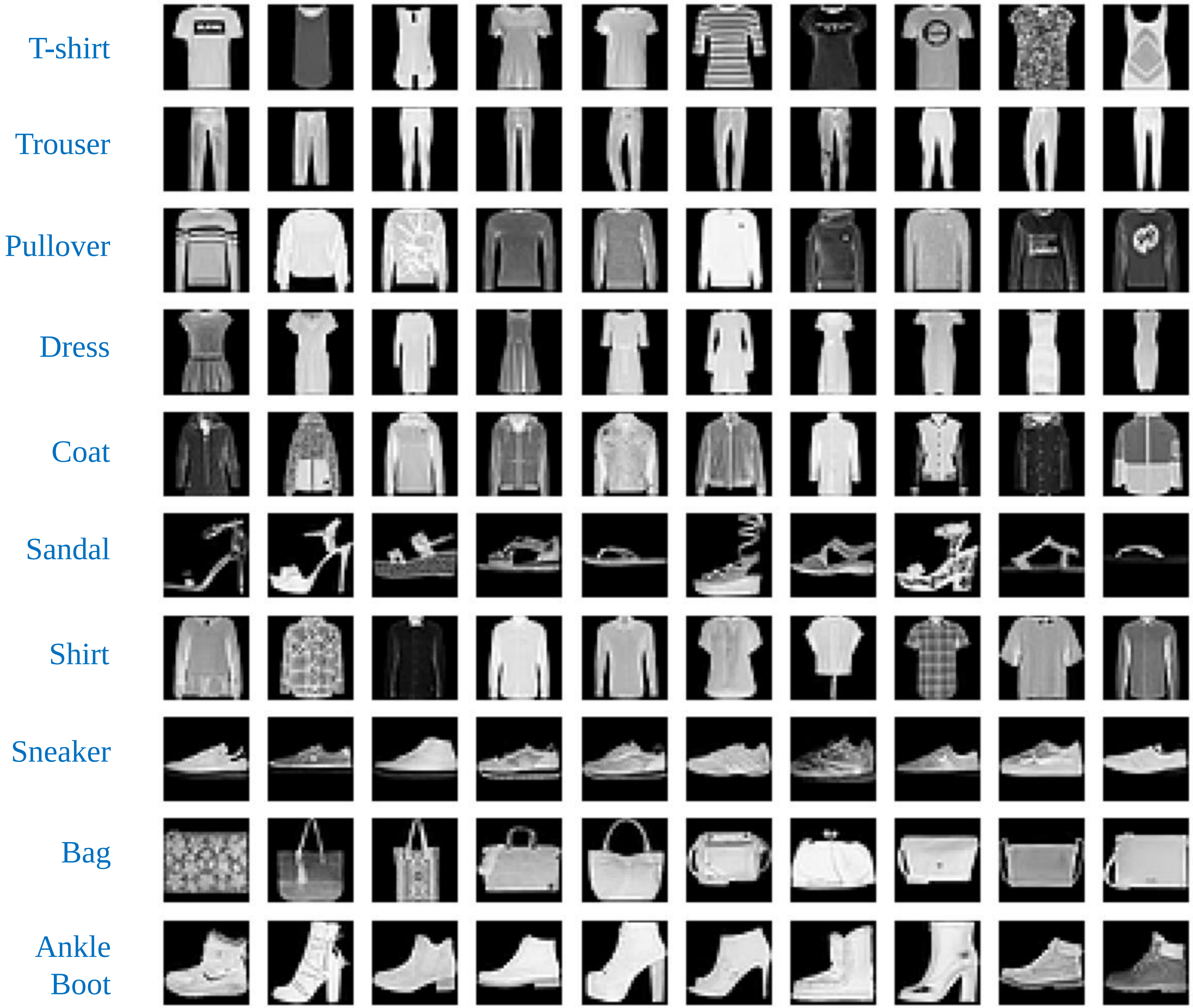
Outline

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

Image Data

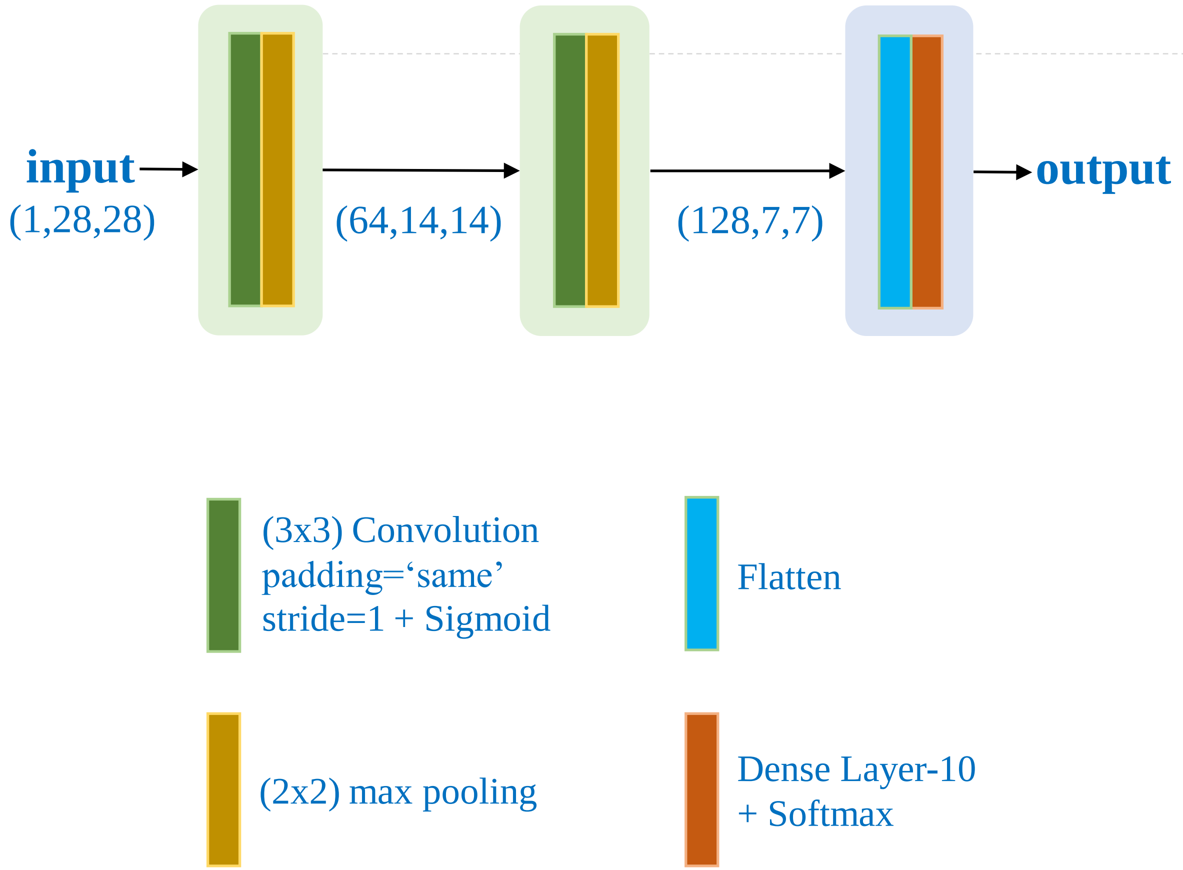
Fashion-MNIST dataset

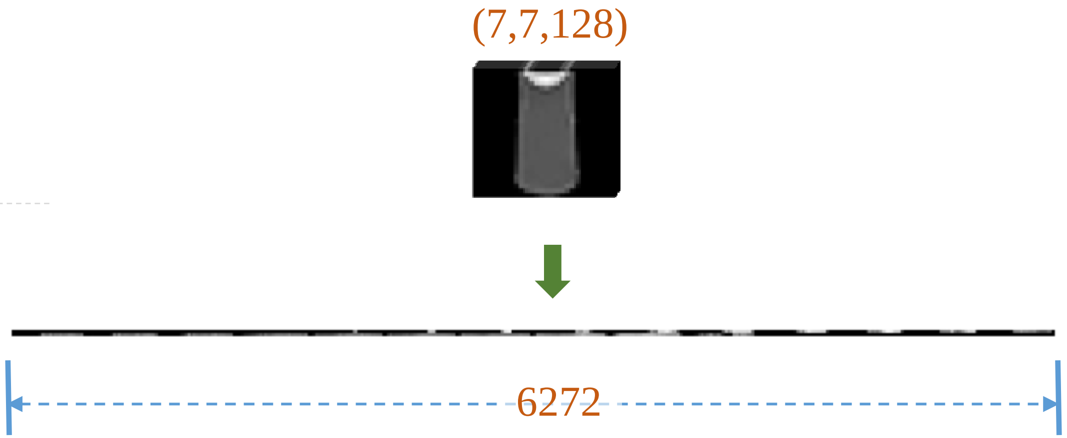
- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

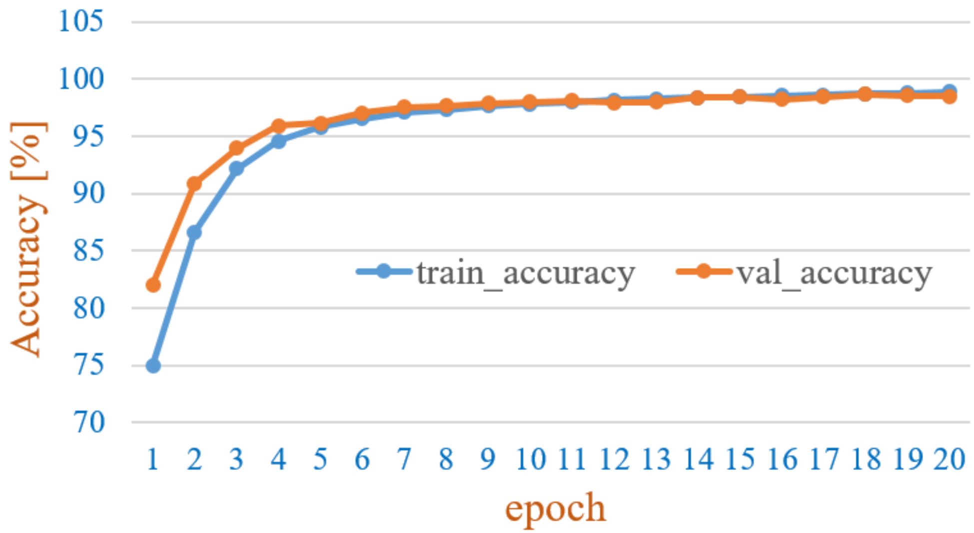


Network Training

❖ Fashion-MNIST dataset







Network Training

❖ Fashion-MNIST dataset

X-data format

(batch, channel, height, width)

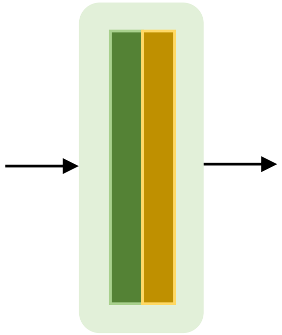
Data normalization [0,1]

(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

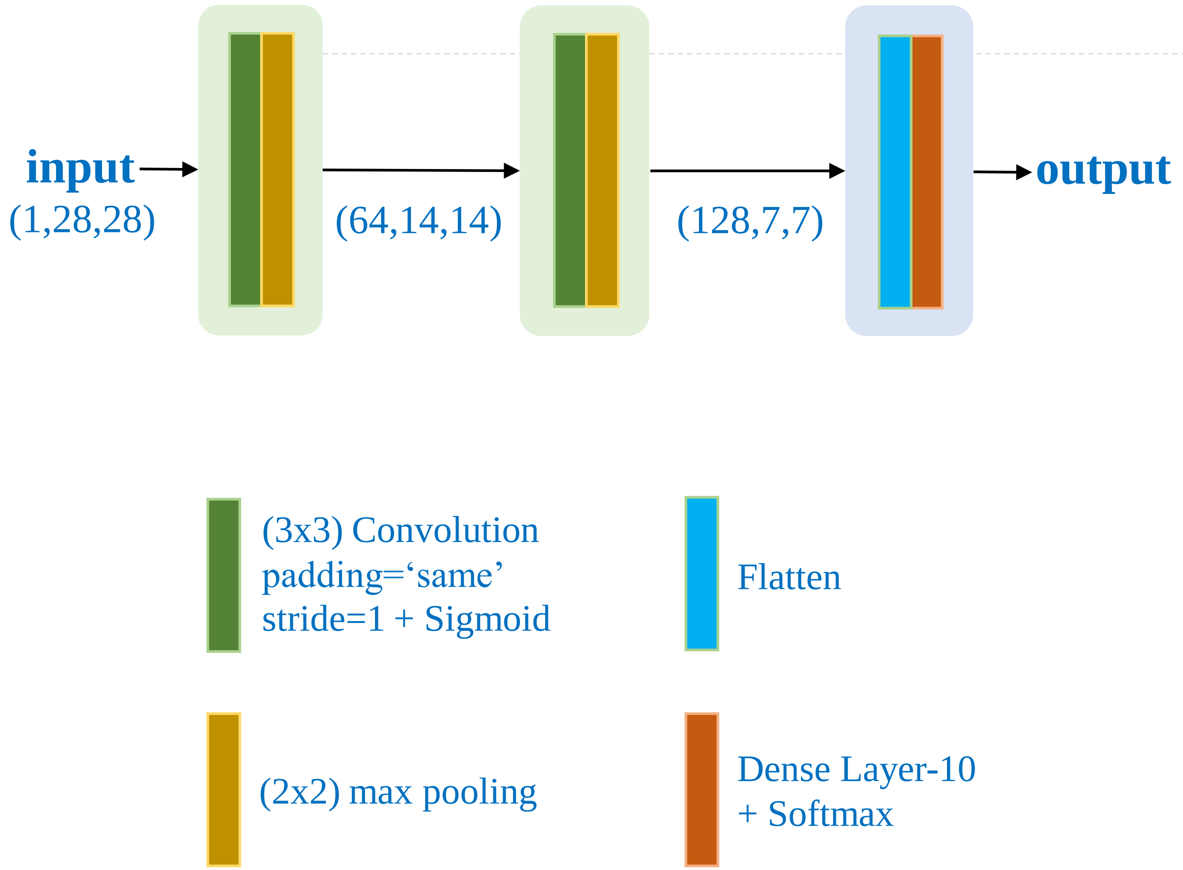


```
# Data
transform = Compose([transforms.ToTensor()])
train_set = FashionMNIST(root='data',
                         train=True,
                         download=True,
                         transform=transform)
trainloader = DataLoader(train_set,
                         batch_size=256,
      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

```
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




```
# 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)
```

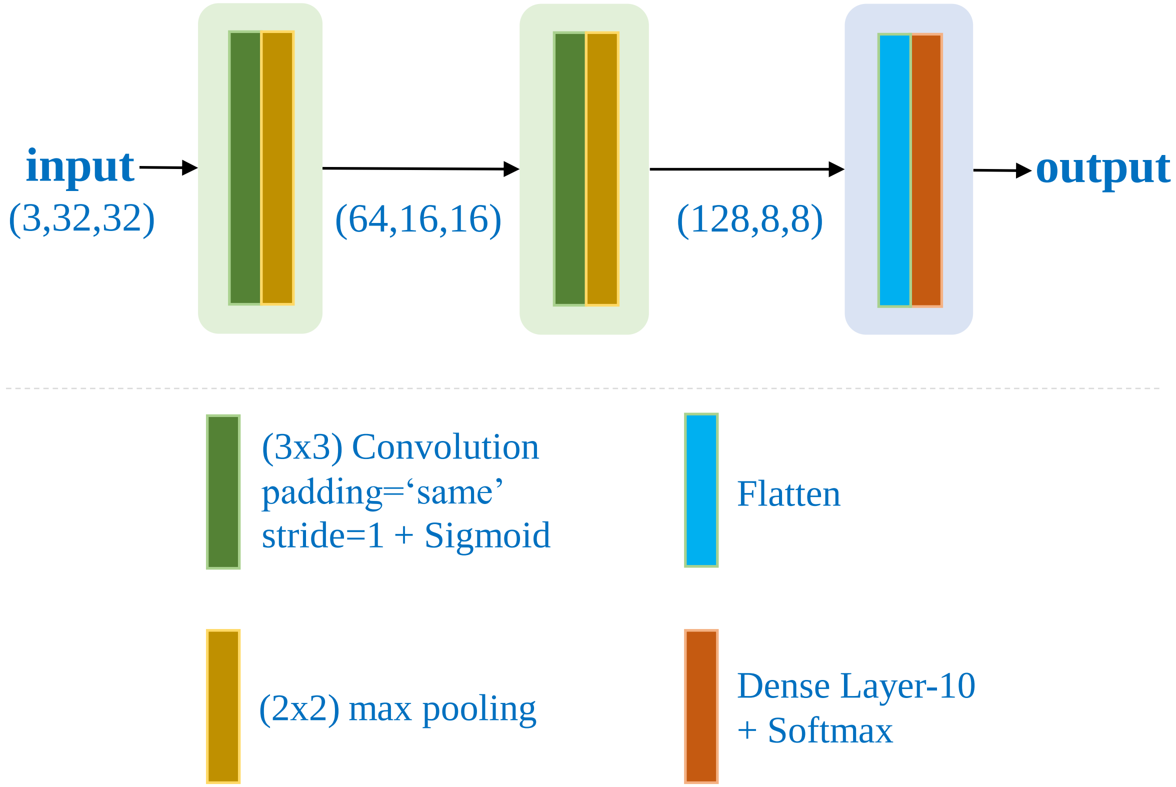
Network Training

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]

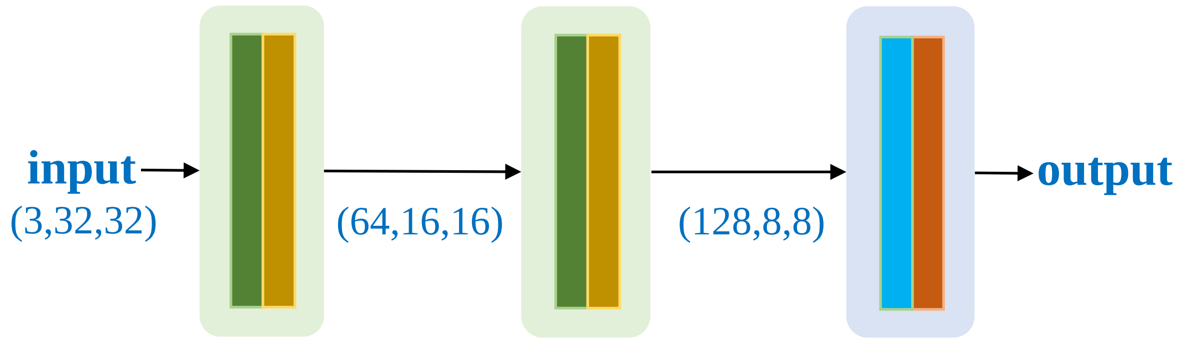
Glorot uniform initialization

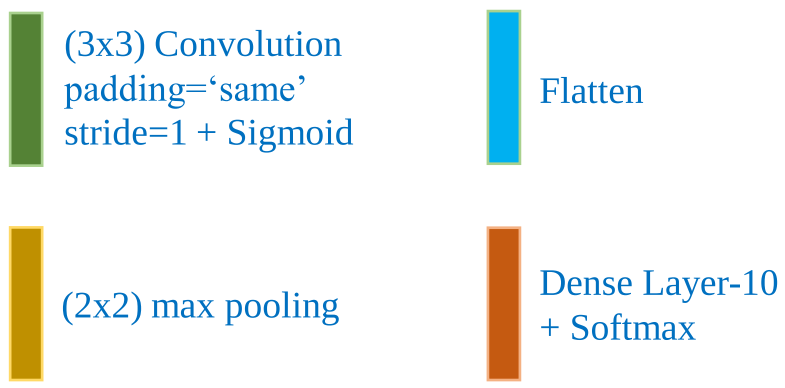
Adam optimizer with lr=1e-3

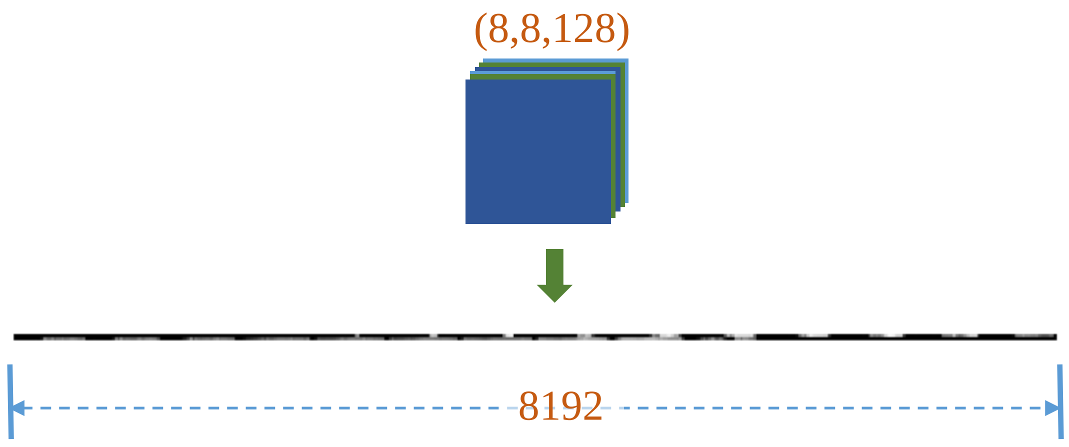
```
conv_layer1 = nn.Sequential(
    nn.Conv2d(1, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

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)
```

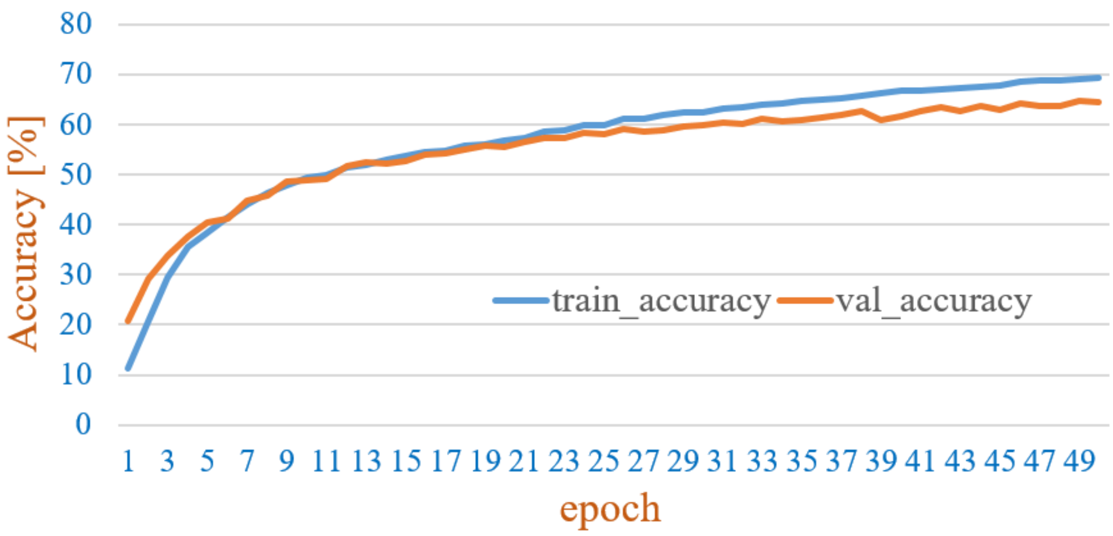
Network Training

❖ Cifar-10 dataset



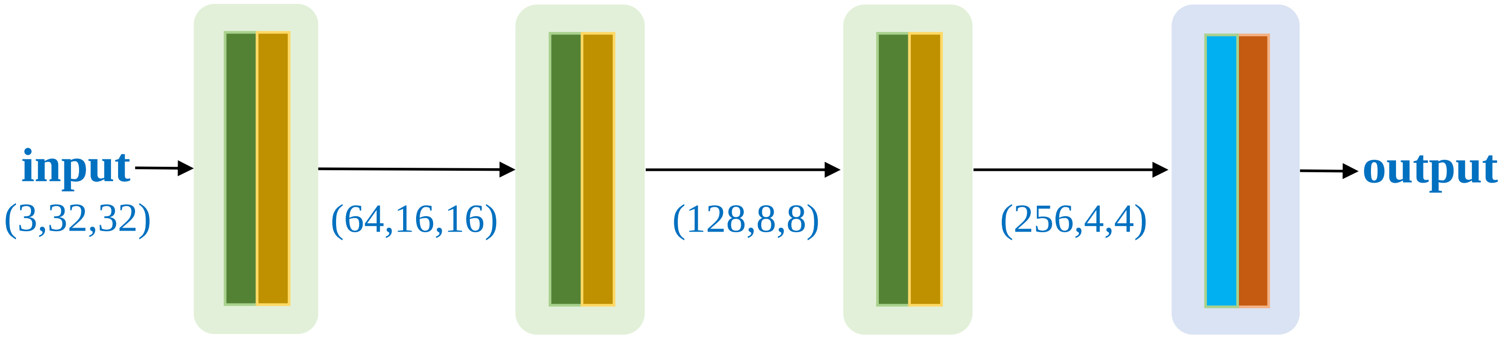






Accuracy: 69.3% - Val_accuracy: 64.5%

Network Training



❖ Cifar-10 dataset:

❖ Adding more layers

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

Flatten

(2x2) max pooling

Dense Layer-10 + Softmax

Data normalization [0,1]
Glorot uniform initialization
Adam optimizer with lr=1e-3

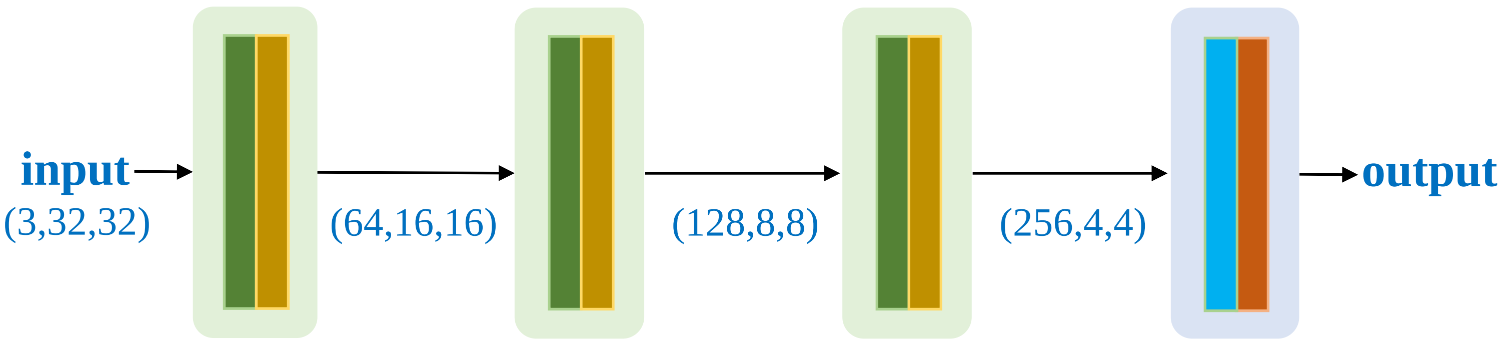
```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer3 = nn.Sequential(
    nn.Conv2d(128, 256,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

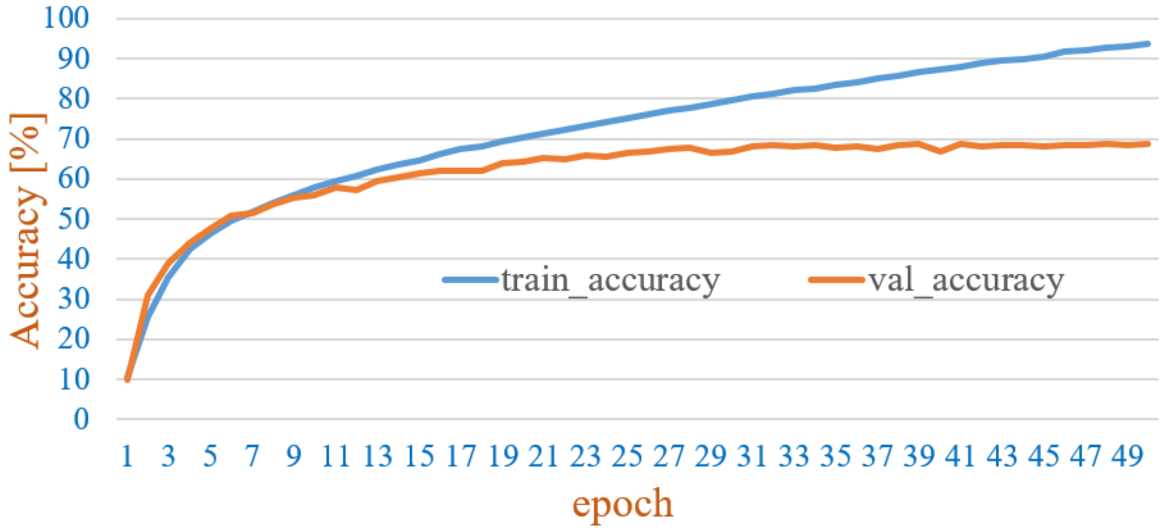
Network Training

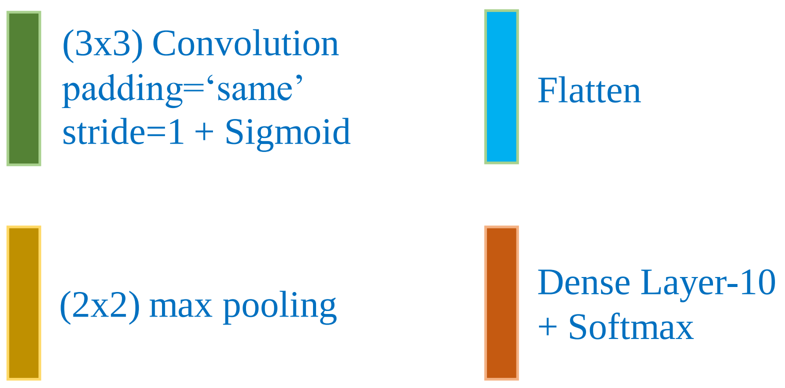
❖ Cifar-10 dataset:

❖ Adding more layers

Good news: Network accuracy increases about 25%





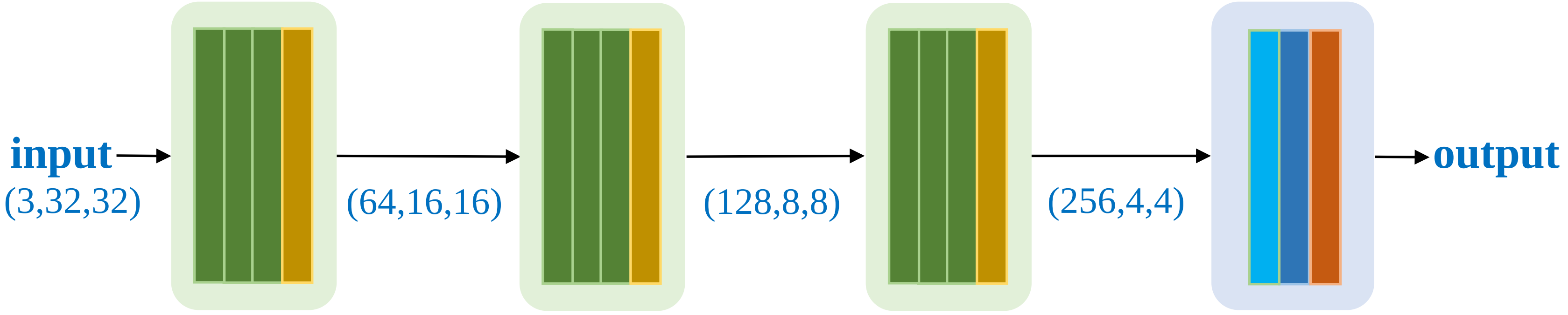


Accuracy: 93.8% - Val_accuracy: 68.7%

Network Training

Cifar-10 dataset:

❖ Keep adding more layers



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

Flatten

(2x2) max pooling

Dense Layer-10 + Softmax

Data normalization [0,1]
Glorot uniform initialization
Adam optimizer with lr=1e-3

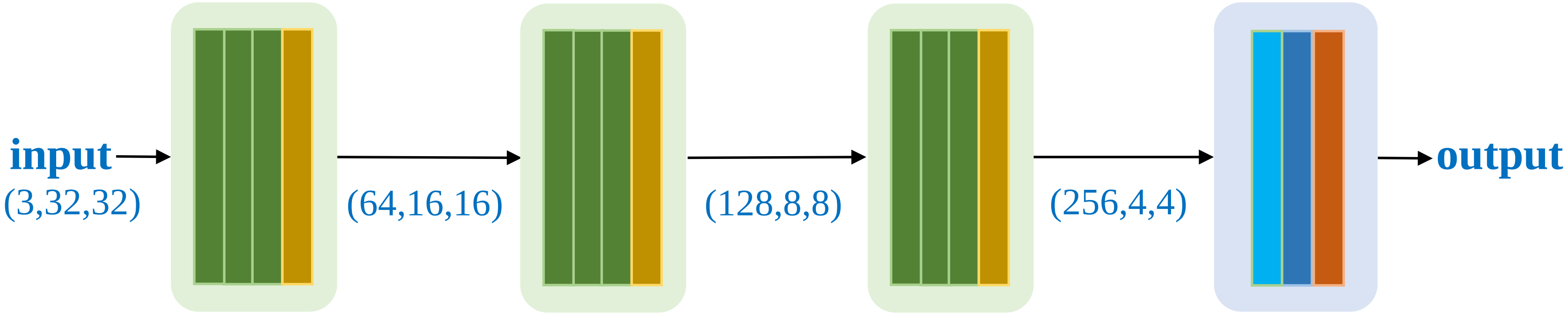
```
conv_layer1 = nn.Sequentialn(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid(),
                          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

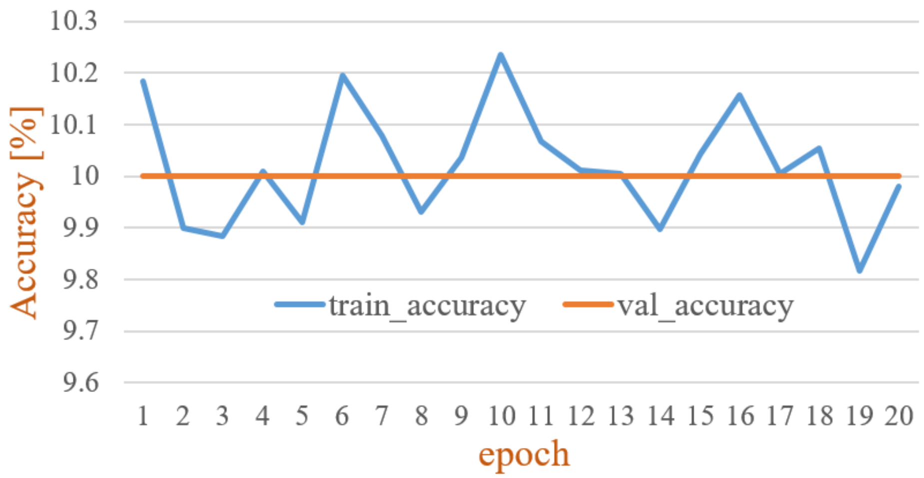
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

The network does not learn





- (3x3) Convolution padding='same' stride=1 + Sigmoid
- Flatten
- Dense Layer-512 + Sigmoid
- (2x2) max pooling
- Dense Layer-10 + Softmax

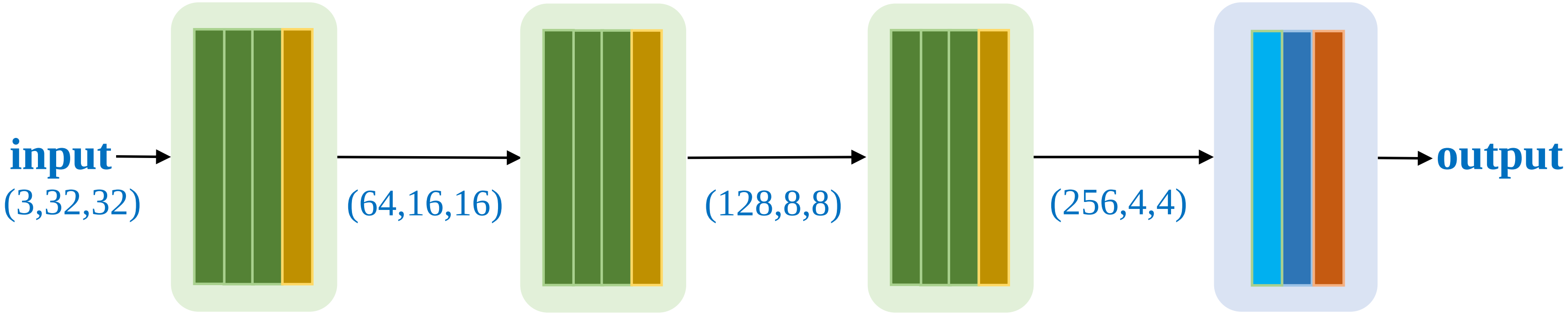
Network Training

❖ Cifar-10 dataset:

❖ Keep adding more layers

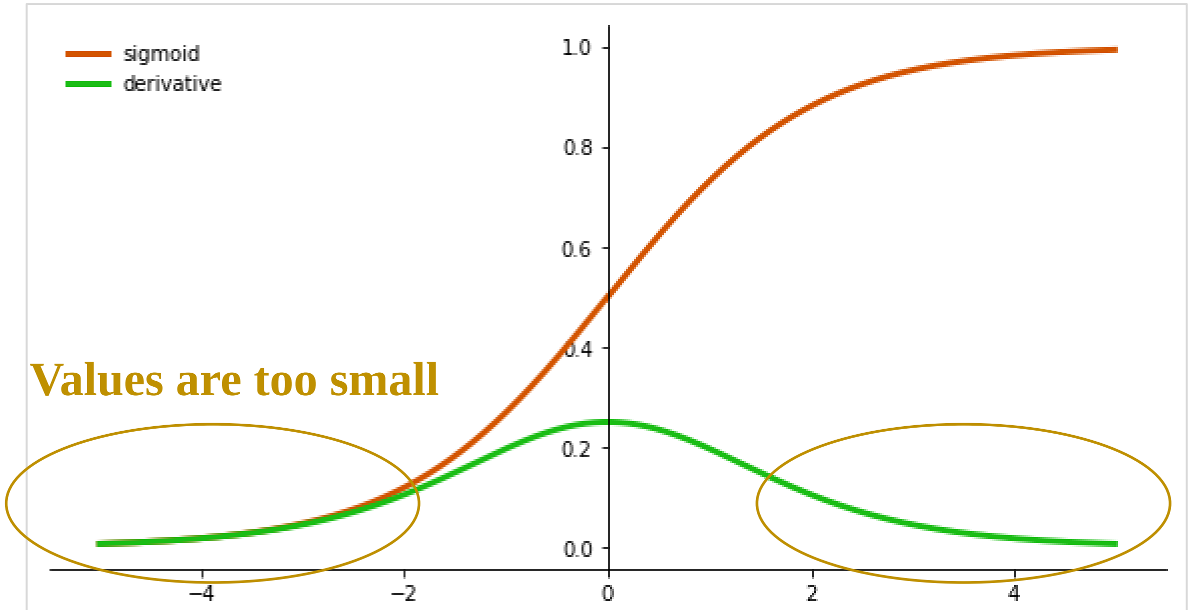
(3x3) Convolution
padding=‘same’
stride=1 + Sigmoid

Dense Layer-512
+ Sigmoid



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

Vanishing Problem



AI VIETNAM
All-in-One Course

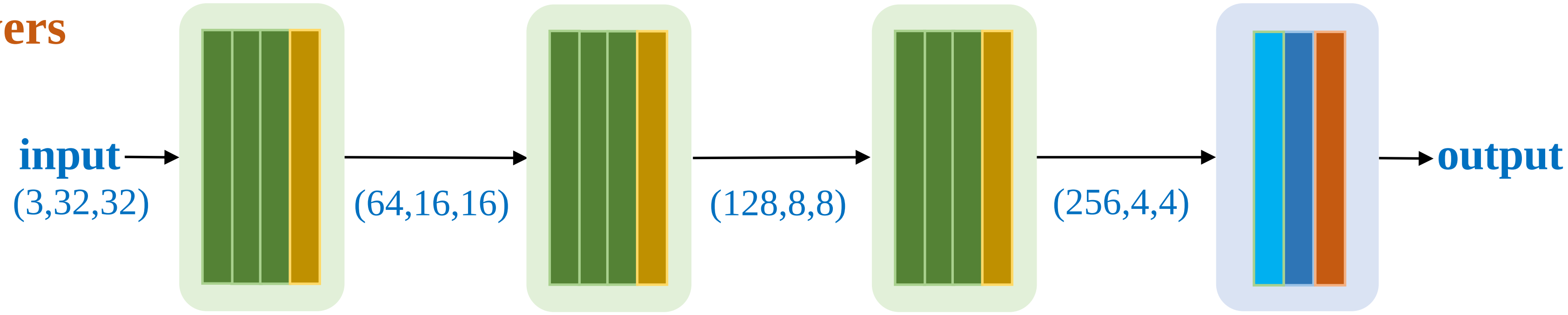
Network Training

❖ Cifar-10 dataset:

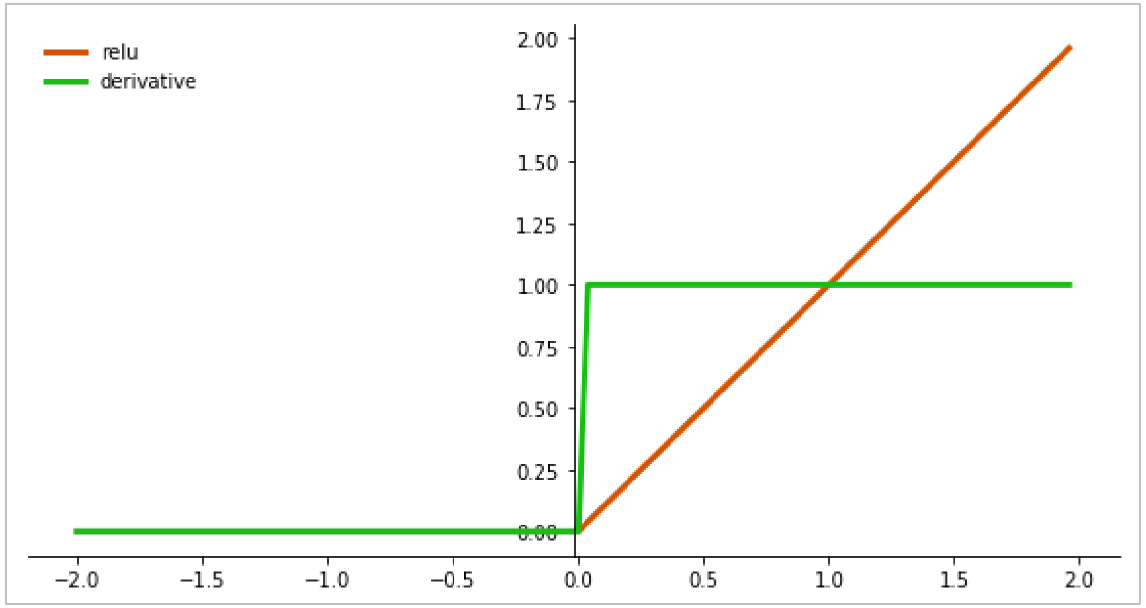
❖ Keep adding more layers

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

Dense Layer-512
+ ReLU



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



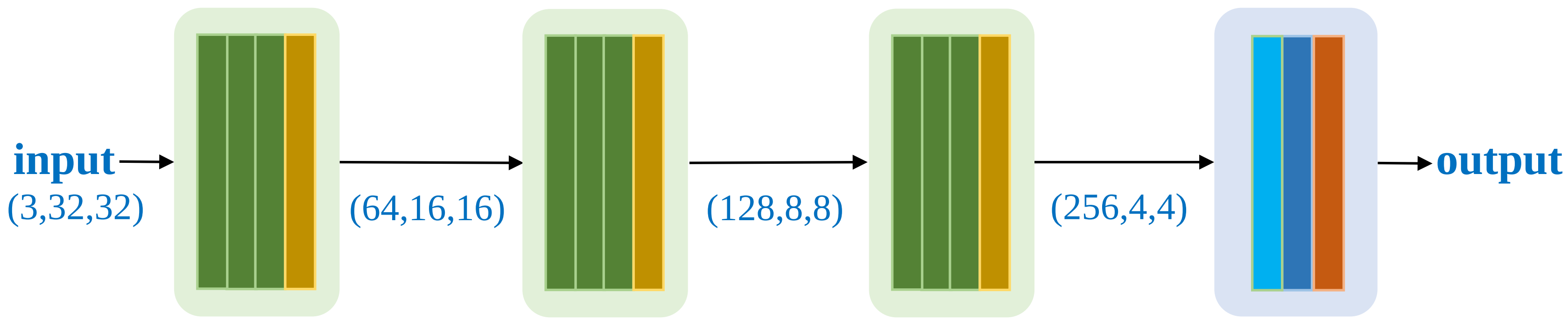

```
nn.Conv2D(...), nn.Sigmoid()
```

```
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

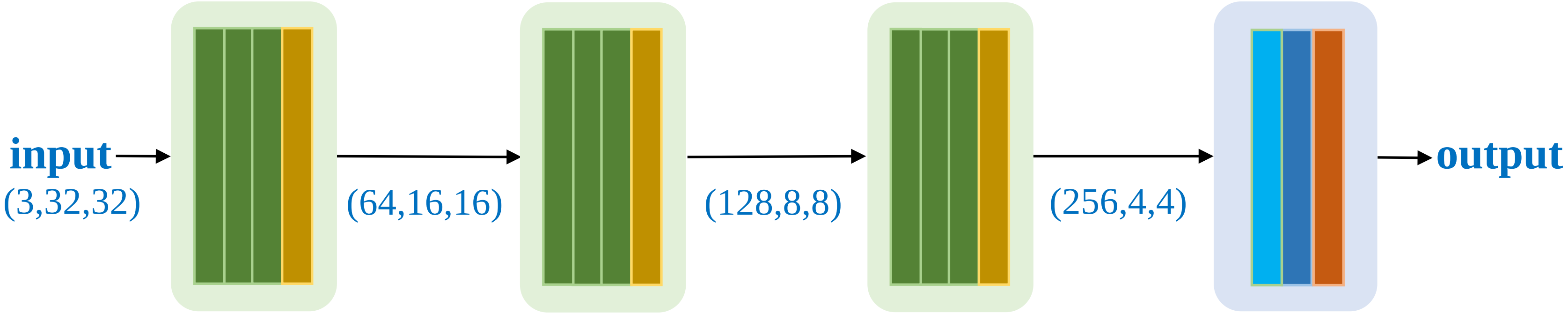
Data normalization [0,1]

Glorot uniform initialization

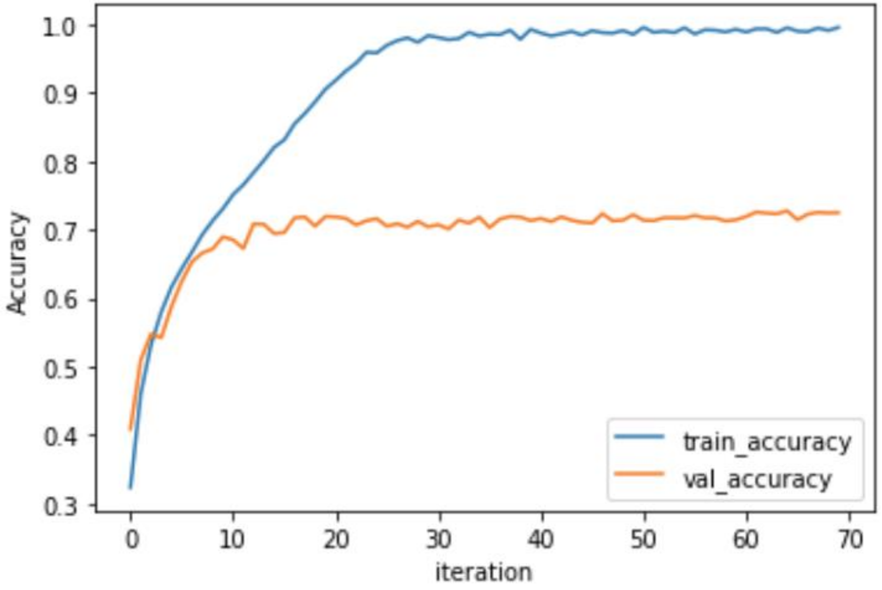
Adam optimizer with lr=1e-3

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

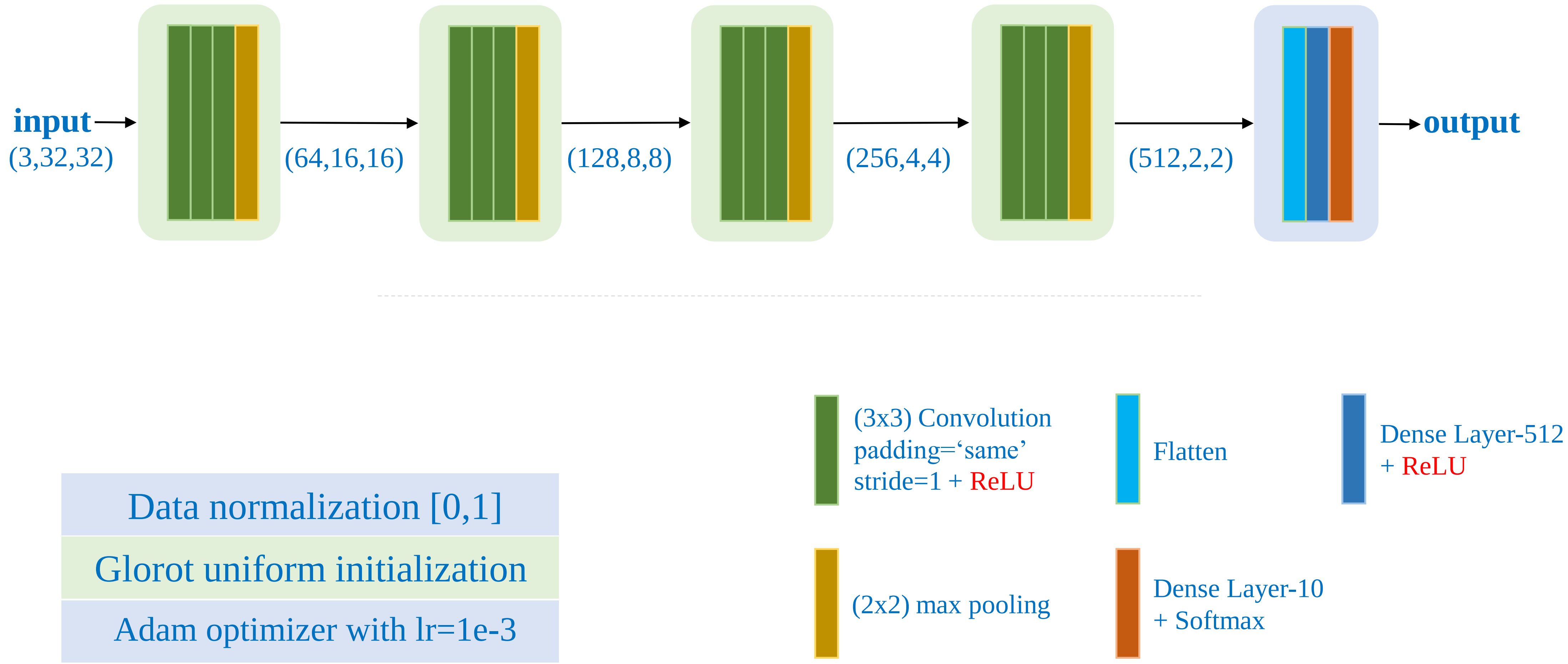
(2x2) max pooling

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

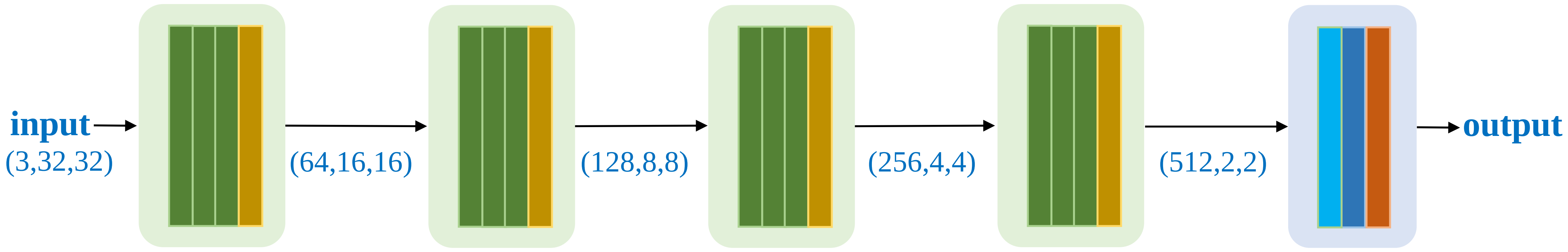
Implementation

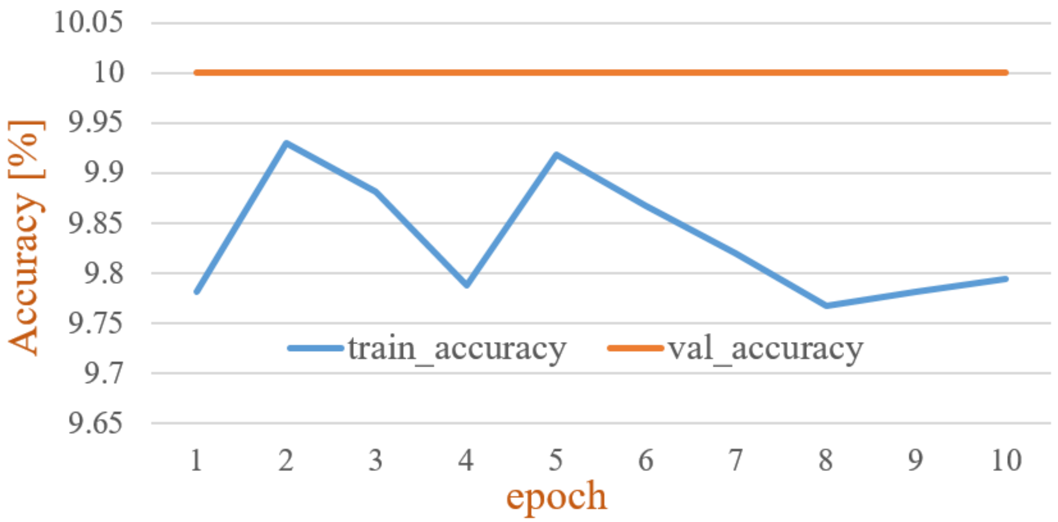
```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2,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

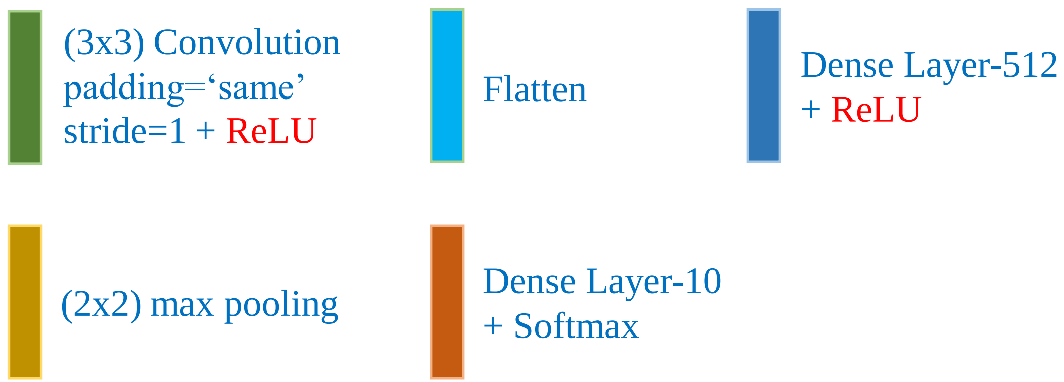
```
def initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            init.xavier_uniform_(m.weight)
            if m.bias is not None:
                init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
    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







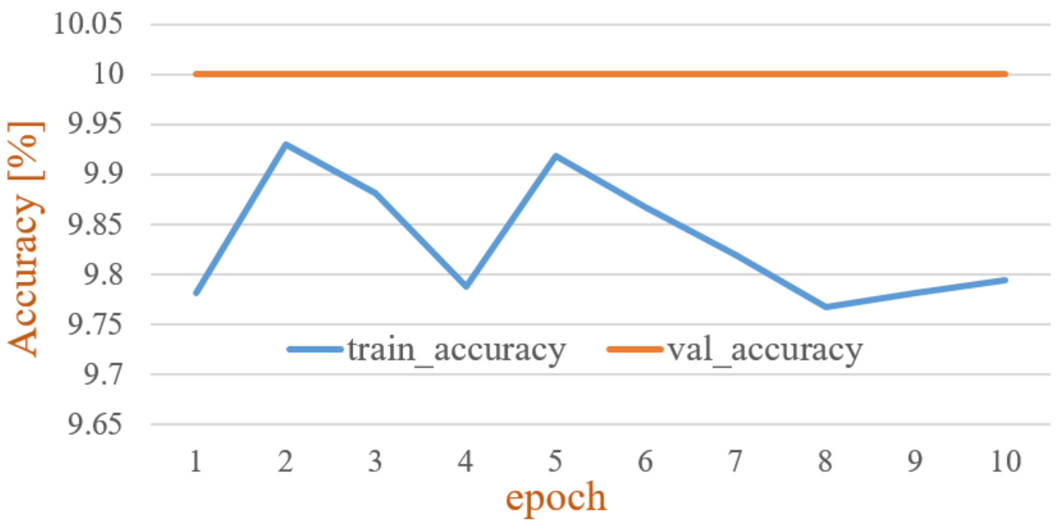
Network does not learn again

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)

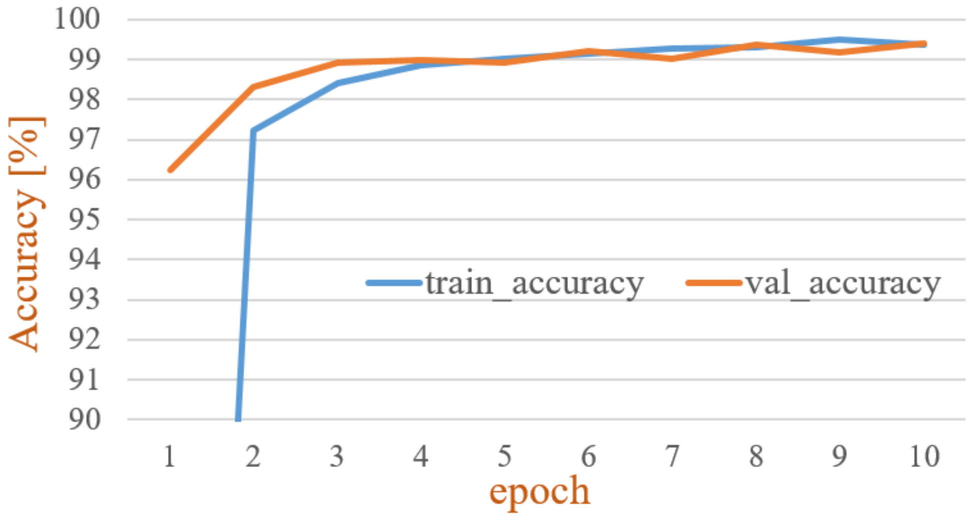
Network does not learn



Network Training

❖ Solution 1: Observation

MNIST 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)



The current network performs excellently for MNIST dataset

Network Training

❖ Solution 1: Idea

Current Network → Train (failed) → Cifar-10 Dataset

Current Network → Train (ok) → MNIST Dataset

Cifar-10 Dataset > MNIST Dataset (complex)

How to reduce the complexity of the Cifar-10 dataset

Data Normalization (scale to [0,1])

0 … 255 → 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

$$Y = aX + b$$

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

$$Y = aX + b$$

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

$$= \frac{aX - \frac{1}{n}\Sigma_i \, aX_i}{\sqrt{\frac{1}{n}\Sigma_i \left(aX_i - \frac{1}{n}\Sigma_j \, aX_j\right)^2}}$$

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

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)

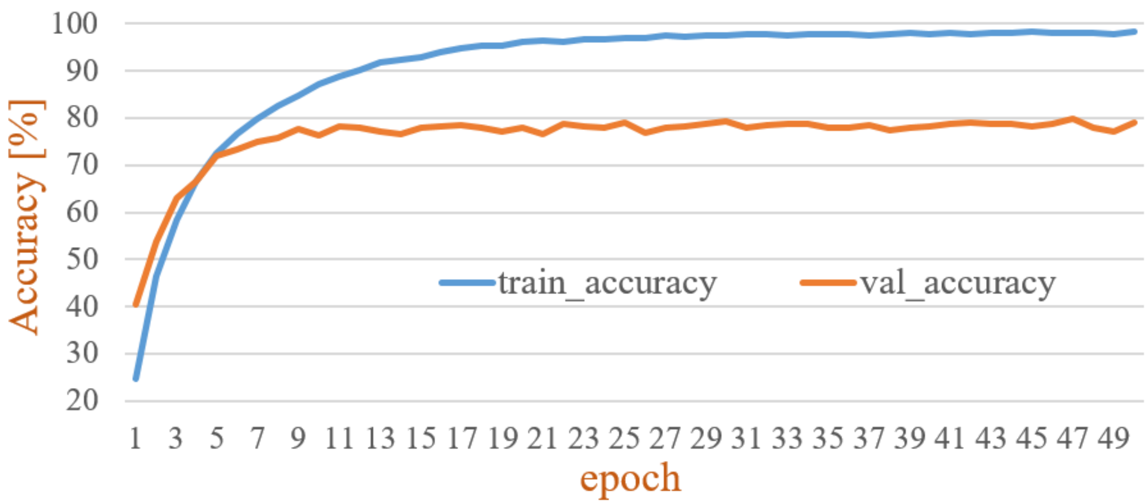
$$X = \frac{X - \mu_d}{\sigma_d}$$

μ_d is the mean of dataset

σ_d is the deviation for the whole dataset

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)
```



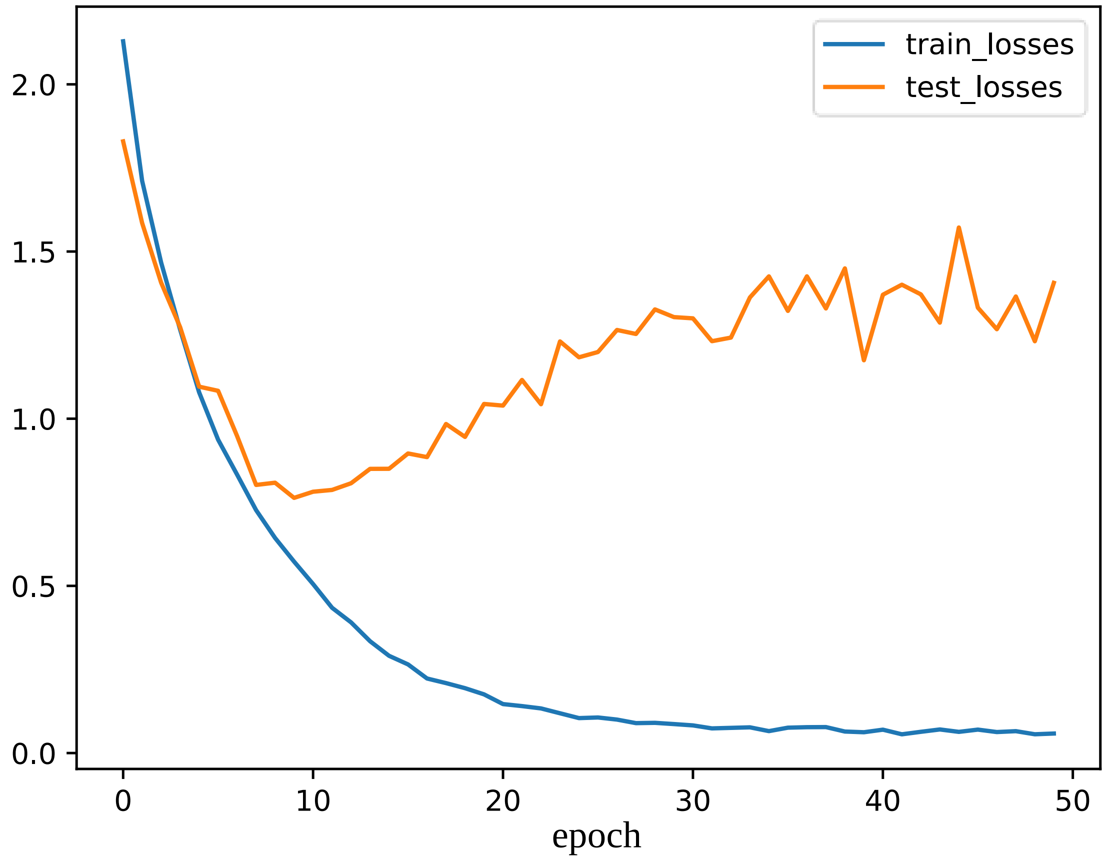
Network Training

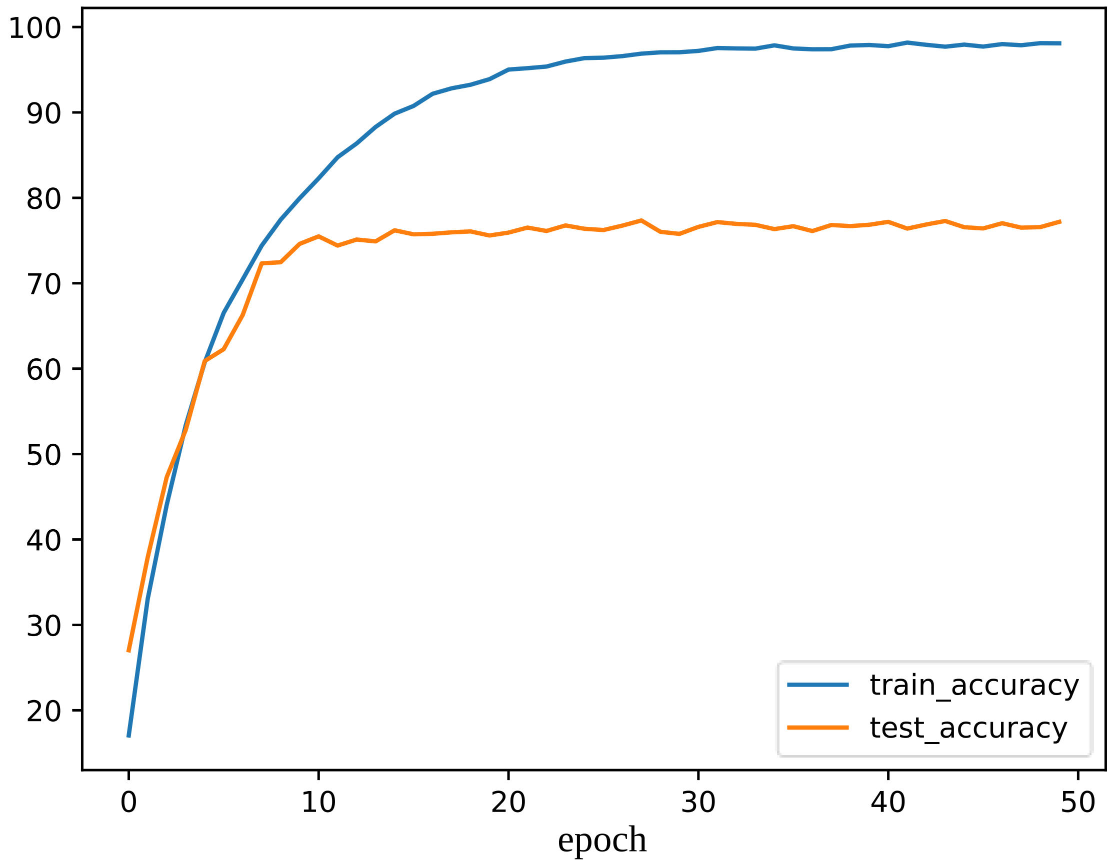
❖ **Solution 1 (extension):**

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

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

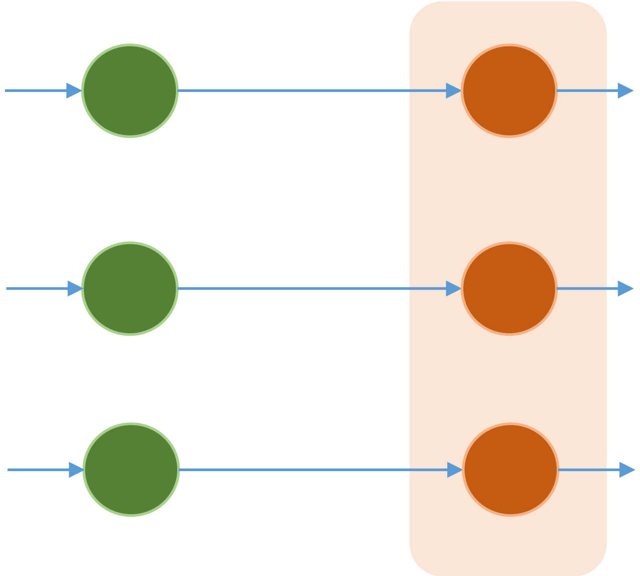
MNIST Dataset → **Data Normalization** (convert to 0-mean and 1-deviation) → **Network Construction** (Convs , ReLU, max pooling, Dense layers) → **Parameter Initialization** (Glorot uniform) → **Training** (Adam and cross-entropy loss)

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

Batch Normalization

Network Training

❖ Solution 2: 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

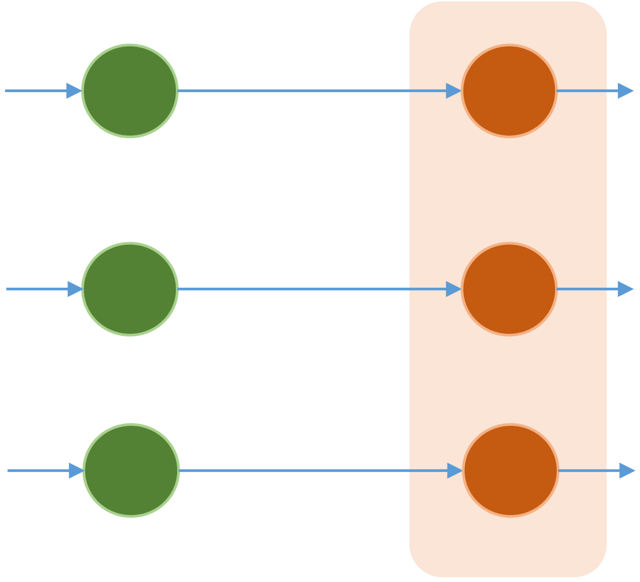
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

Network Training

❖ Solution 2: Batch normalization



What if

$\gamma = \sqrt{\sigma^2 + \epsilon}$ 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

$\epsilon = 10^{-5}$

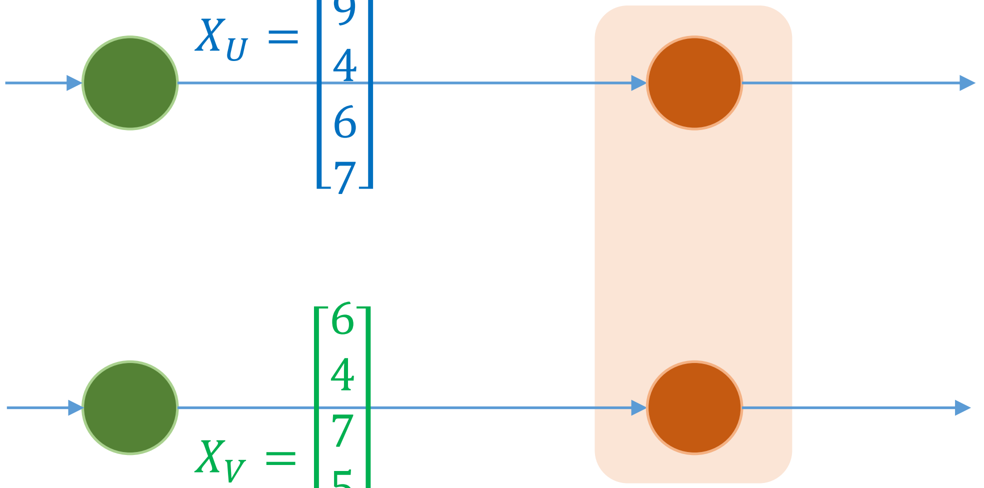
$\mu_U = 5.0$

$\sigma_U = 2.64$

$\gamma_U = 1.0$

$\beta_U = 0.0$

$$\hat{U} = \begin{bmatrix} -1.51 \\ -0.75 \\ 1.51 \\ -0.37 \\ 0.37 \\ 0.75 \end{bmatrix}$$



$$X_U = \begin{bmatrix} 1 \\ 3 \\ 9 \\ 4 \\ 6 \\ 7 \end{bmatrix} \qquad Y_U = \begin{bmatrix} -1.51 \\ -0.75 \\ 1.51 \\ -0.37 \\ 0.37 \\ 0.75 \end{bmatrix}$$

$$X_V = \begin{bmatrix} 6 \\ 4 \\ 7 \\ 5 \\ 6 \\ 2 \end{bmatrix} \qquad Y_V = \begin{bmatrix} 0.61 \\ -0.61 \\ 1.22 \\ 0.0 \\ 0.61 \\ -1.83 \end{bmatrix}$$

$\mu_V = 5.0$

$\sigma_v = 1.63$

$\gamma_V = 1.0$

$\beta_V = 0.0$

$$\hat{V} = \begin{bmatrix} 0.61 \\ -0.61 \\ 1.22 \\ 0.0 \\ 0.61 \\ -1.83 \end{bmatrix}$$

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

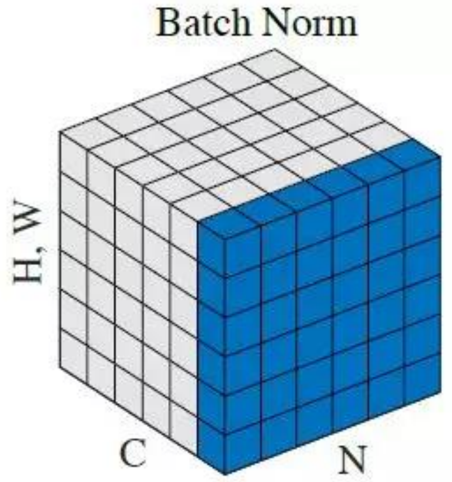
Scale and shift $\hat{X}_i$

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

γ and β are two learning parameters

γ and β are updated in training process

Batch Normalization



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

$\epsilon = 10^{-5}$

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

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

$\mu = 2.5$

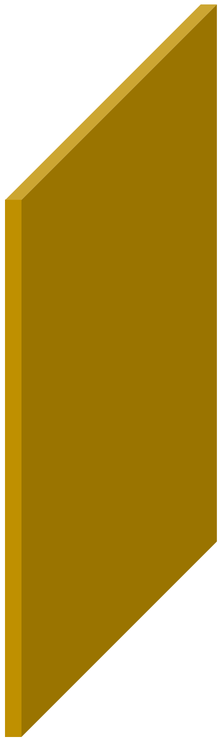
$\sigma^2 = 6.58$

$\gamma = 1.0$

$\beta = 0.0$

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

$\epsilon = 10^{-5}$

$\mu = [2.0, 3.0]$

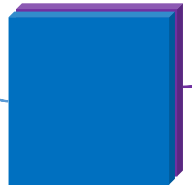
$\sigma^2 = [6.0, 8.67]$

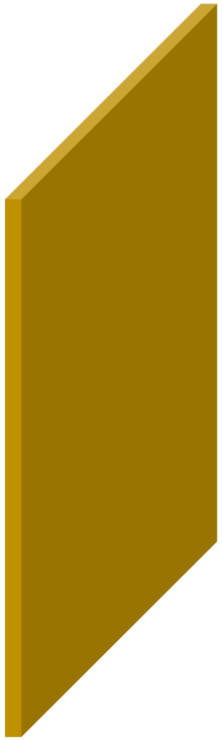
$\gamma = 1.0$

$\beta = 0.0$

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

sample 1

$X = \{$  $\}$



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

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

batch-size = 1

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

Network Training

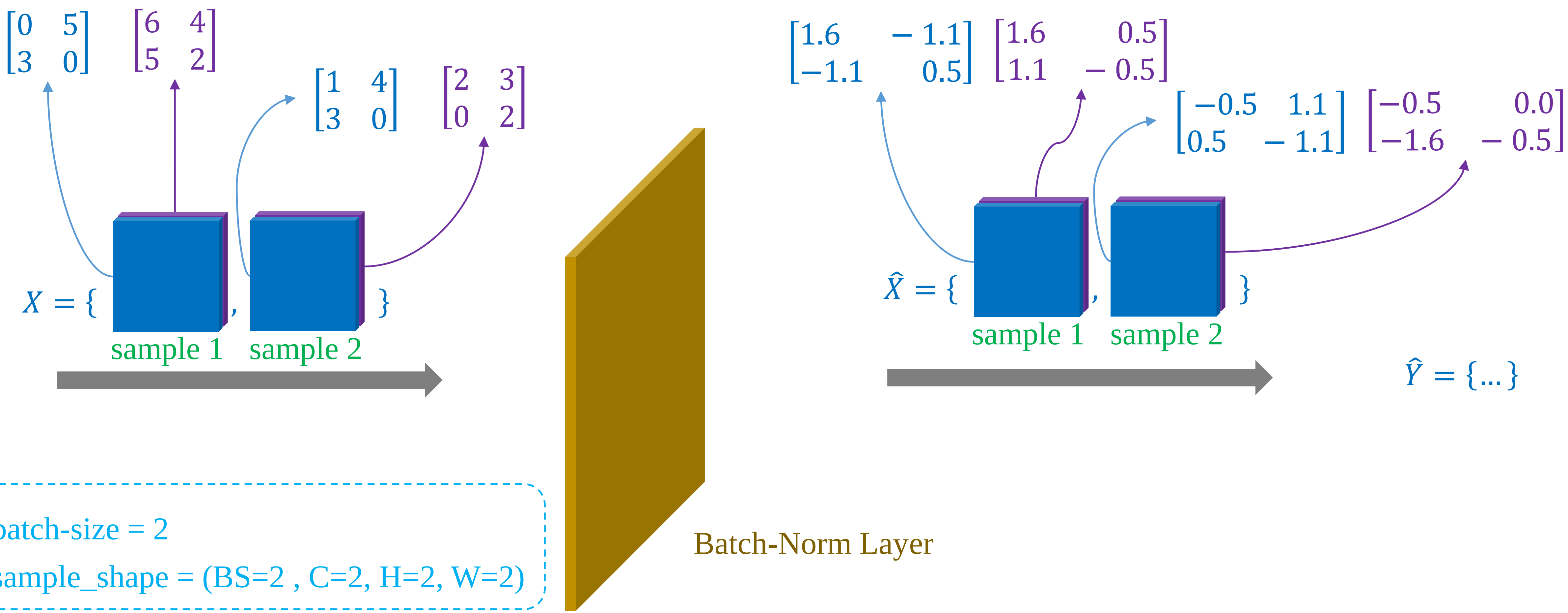
$\epsilon = 10^{-5}$

$\mu = [2.0, 3.0]$

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

$\gamma = 1.0$

$\beta = 0.0$

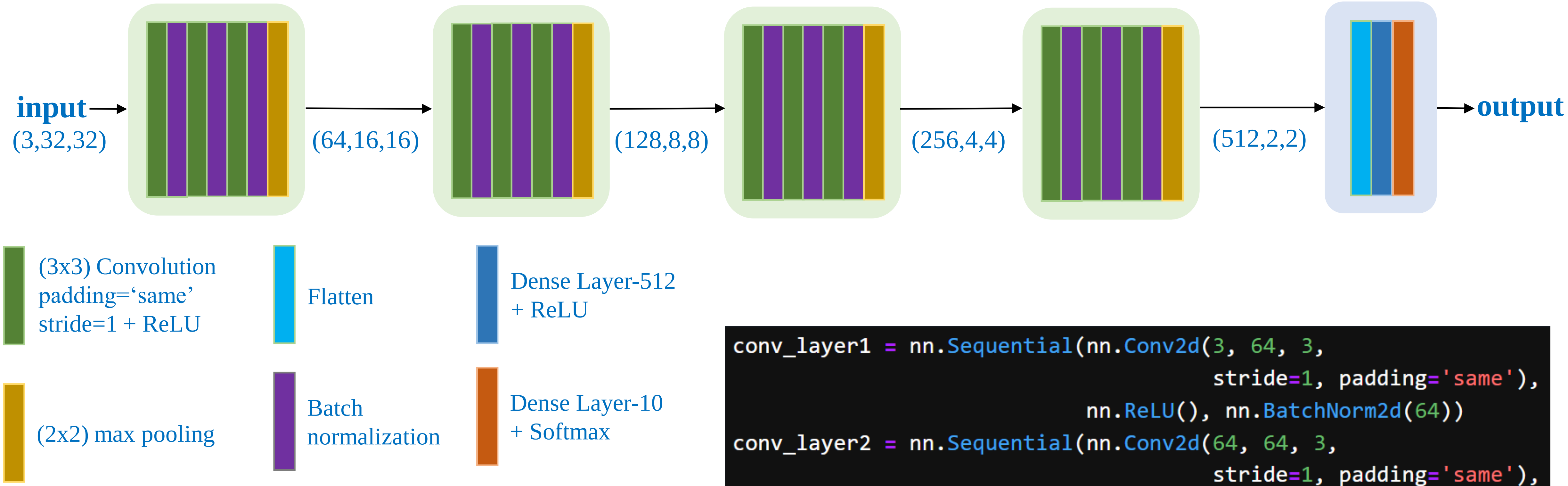


batch-size = 2

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

Network Training

❖ Solution 2: Batch normalization

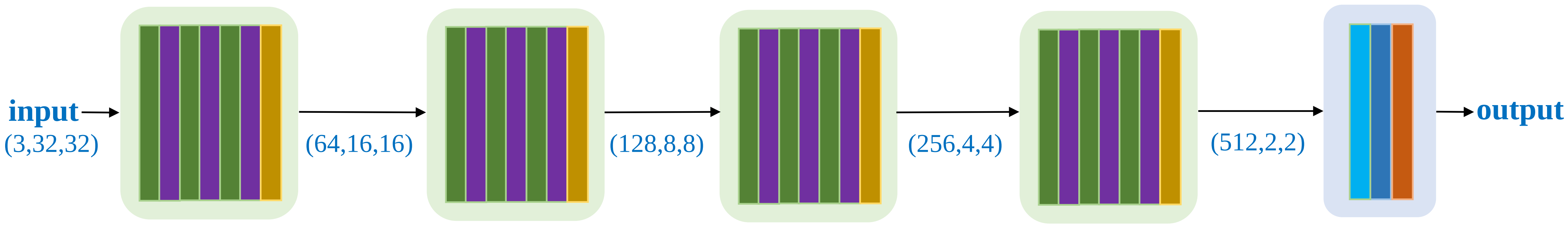


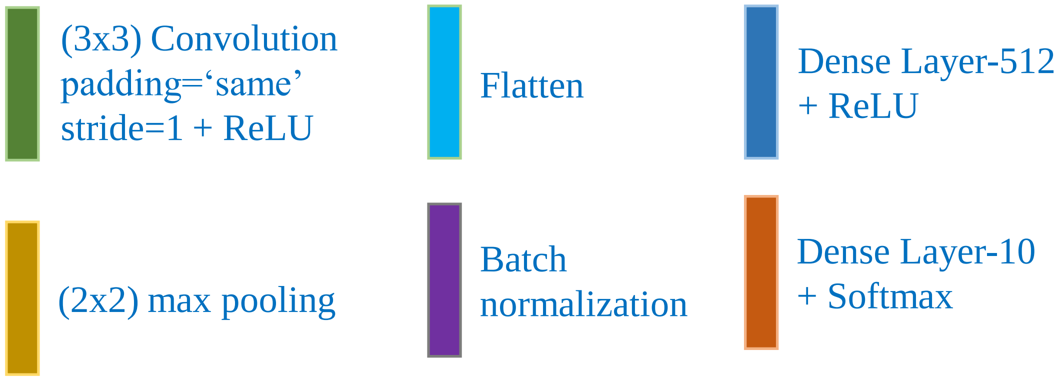
torch.nn.BatchNorm2d(num_features)
num_features (int): C from an expected input of size (N, C, H, W)

```
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))
```

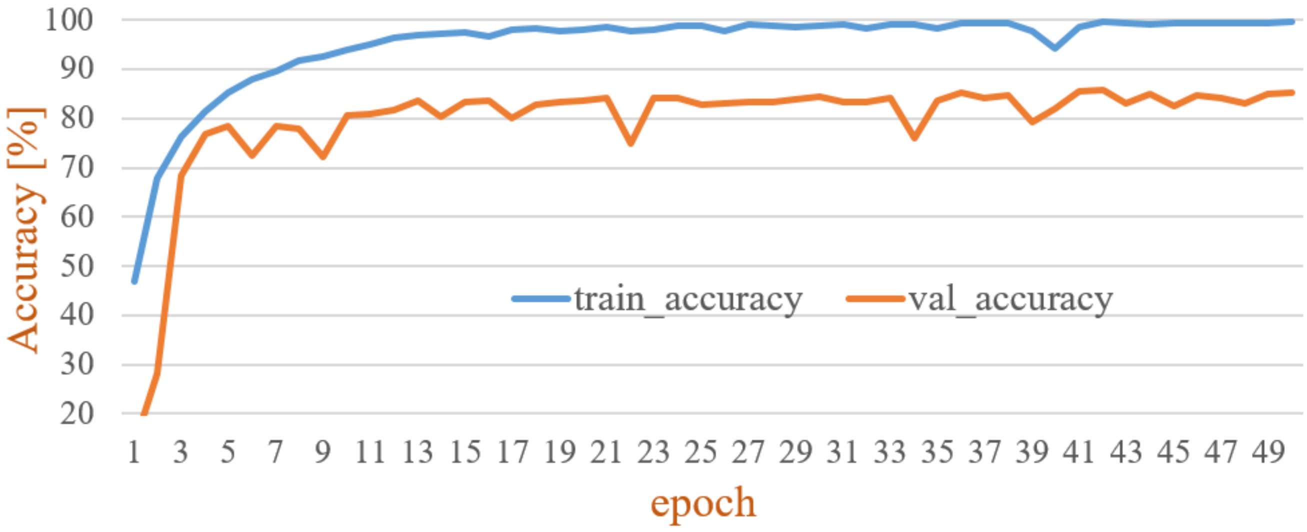
Network Training

❖ Solution 2: Batch normalization






```
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

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)

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

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

```python
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

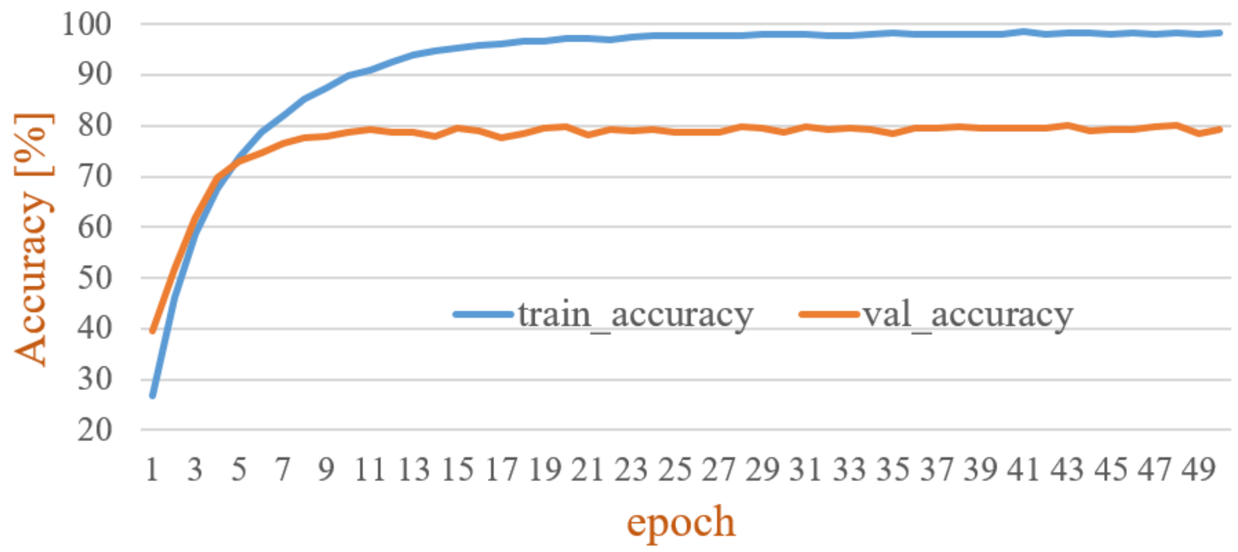
He initialization (2015)

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

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)

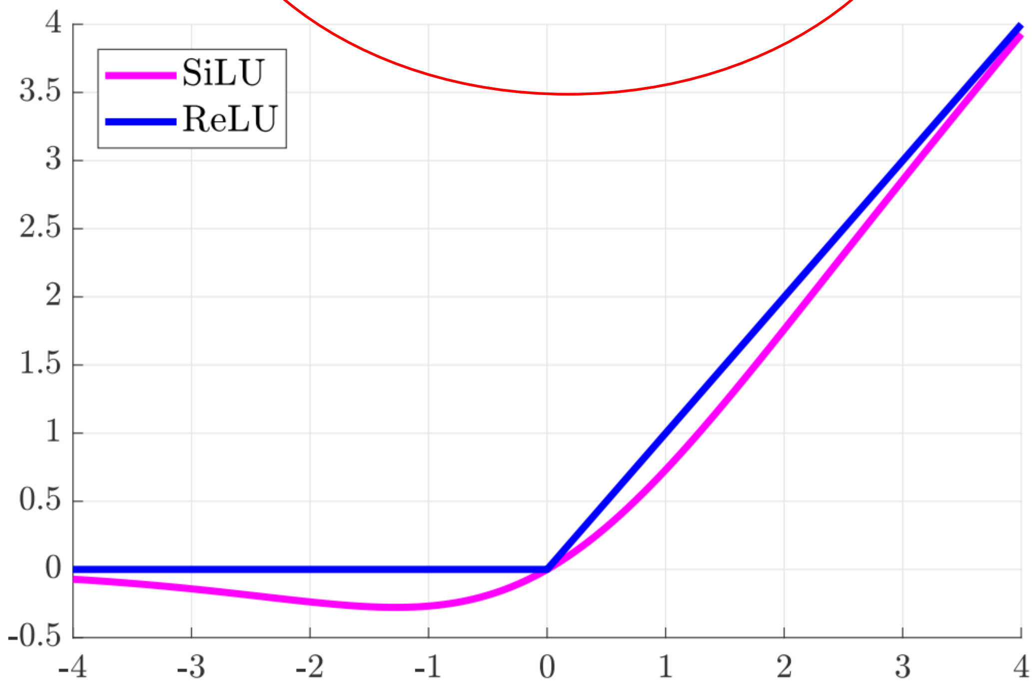
2017 Sigmoid Linear Unit (SiLU)

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

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)

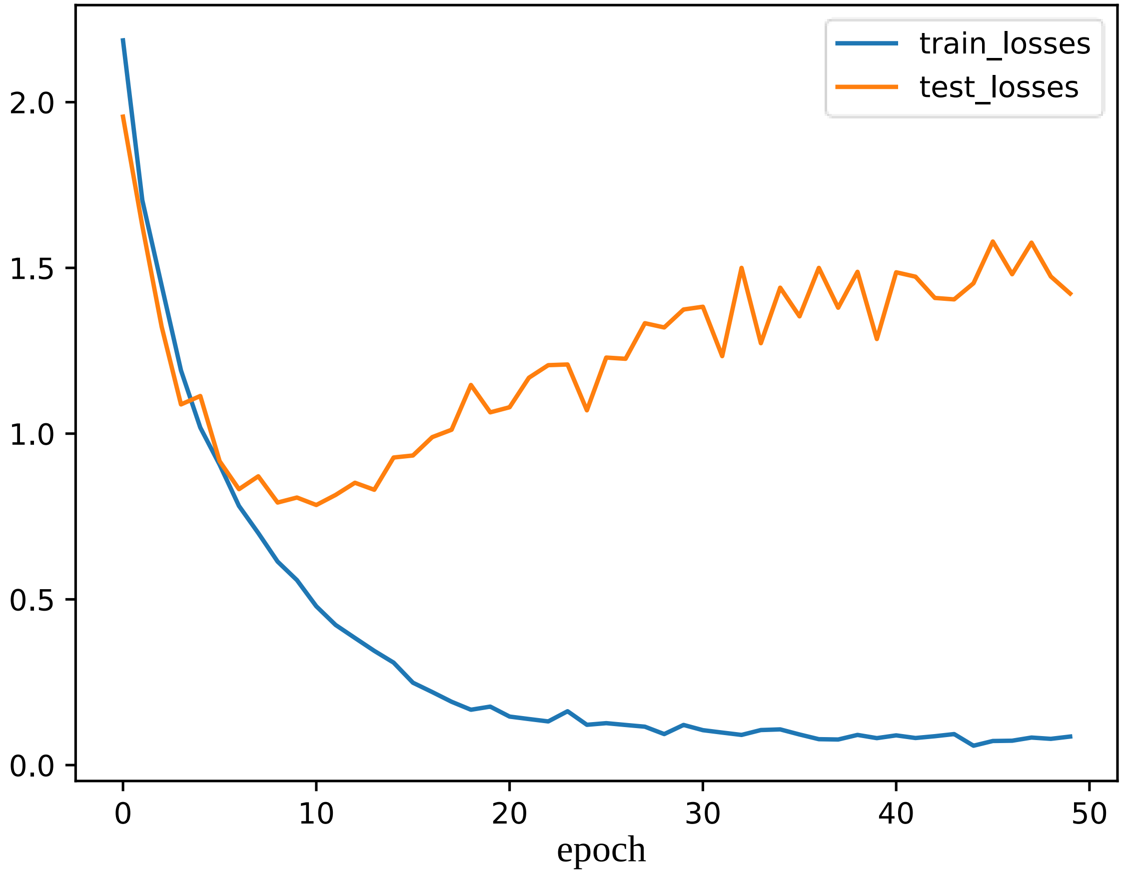
$$\text{swish}(x) = x * \frac{1}{1 + e^{-x}}$$

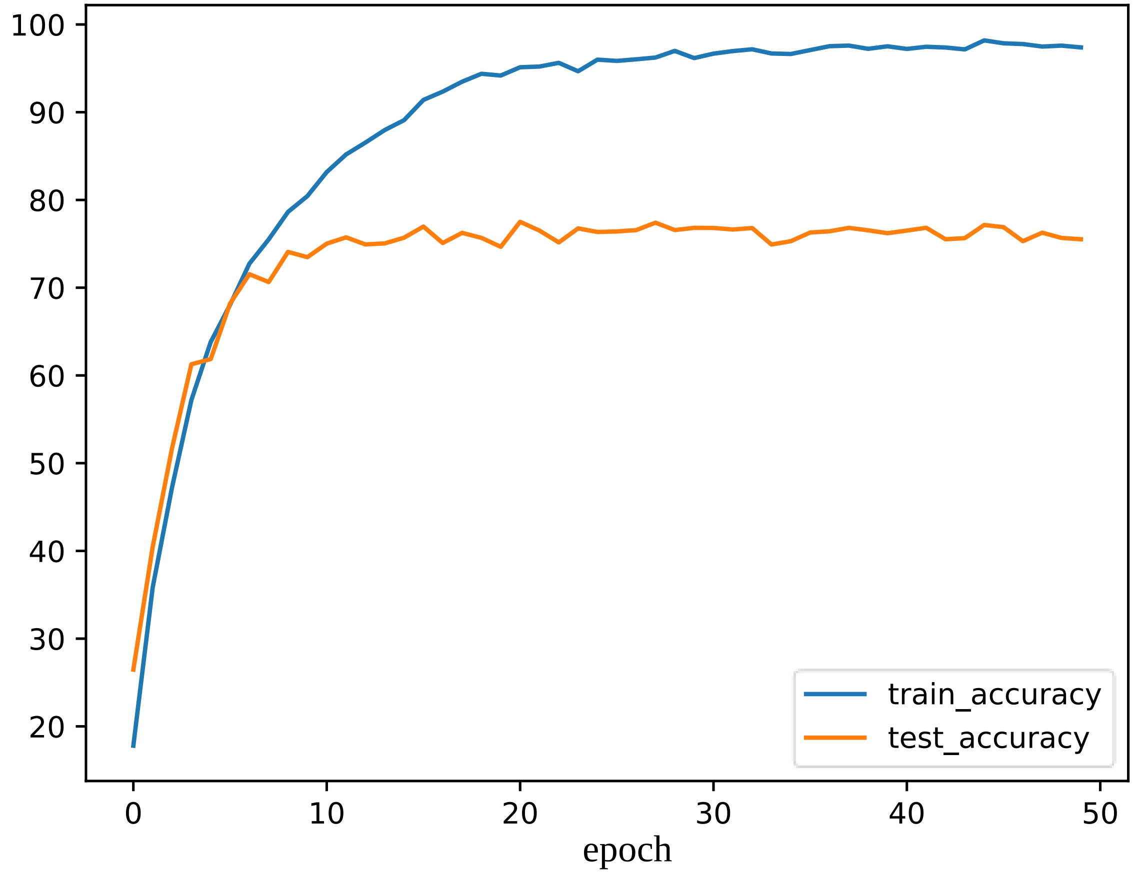
```python
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding=1),
    nn.SiLU()
)
```

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

❖ **Solution 4:**

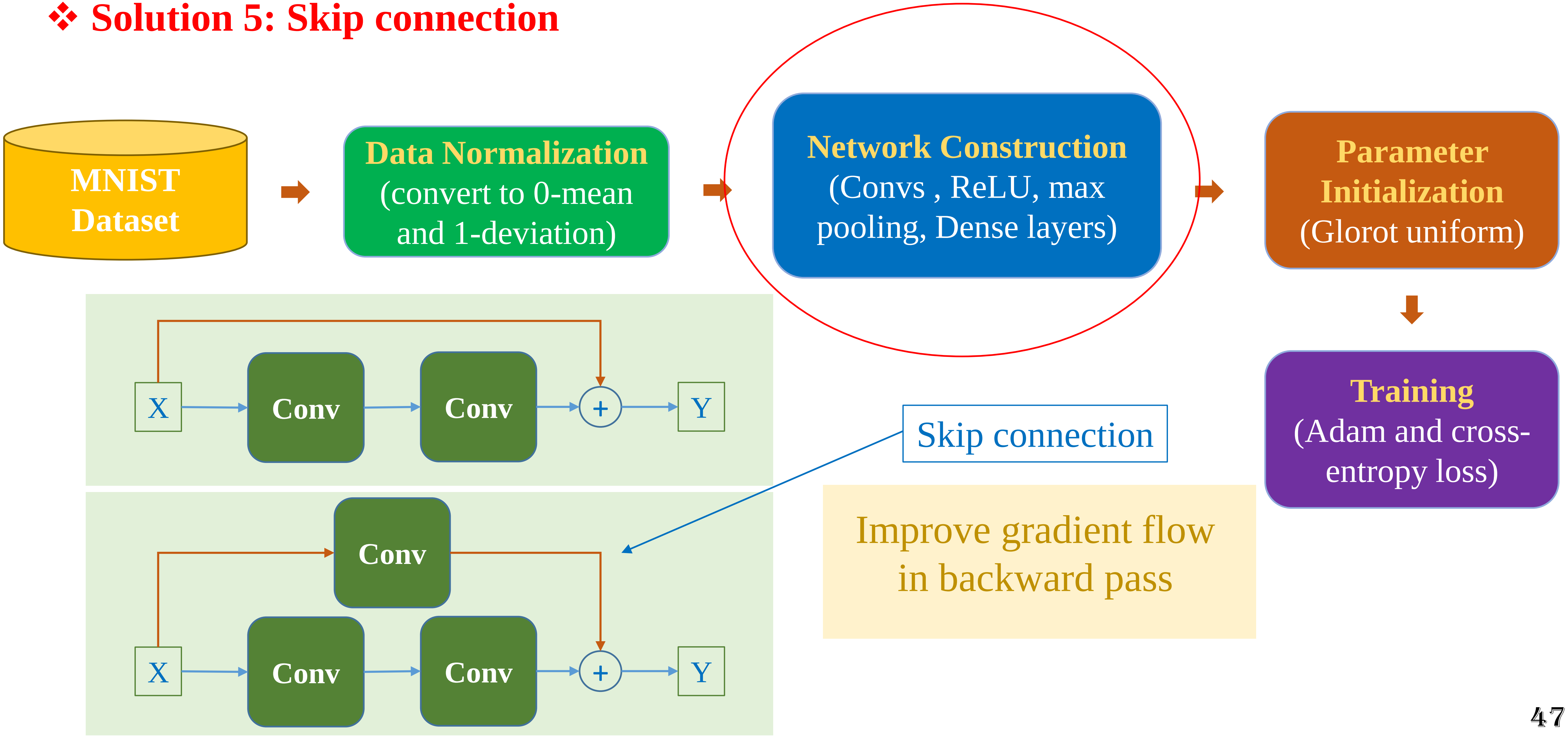
Using advanced activation





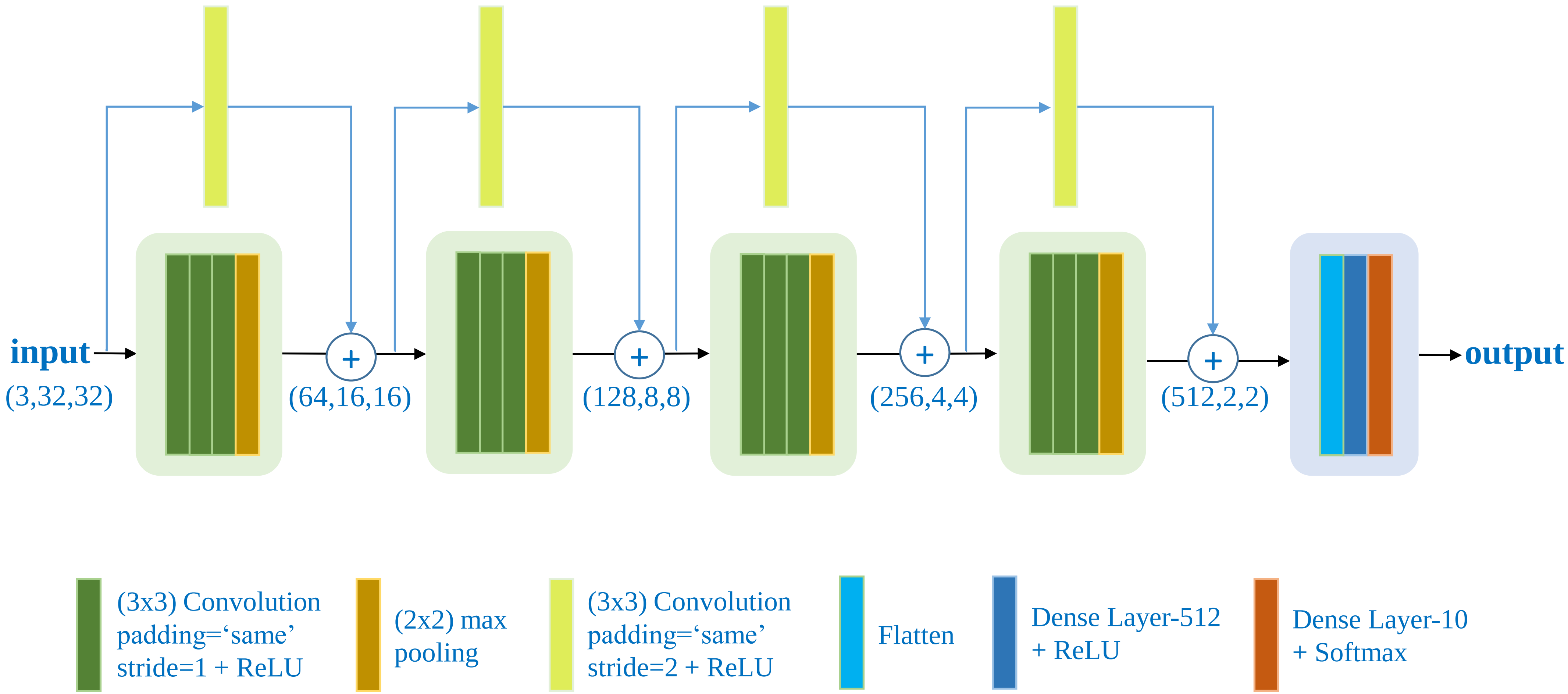
Network Training

❖ Solution 5: Skip connection



Network Training

❖ Solution 5: Skip connection

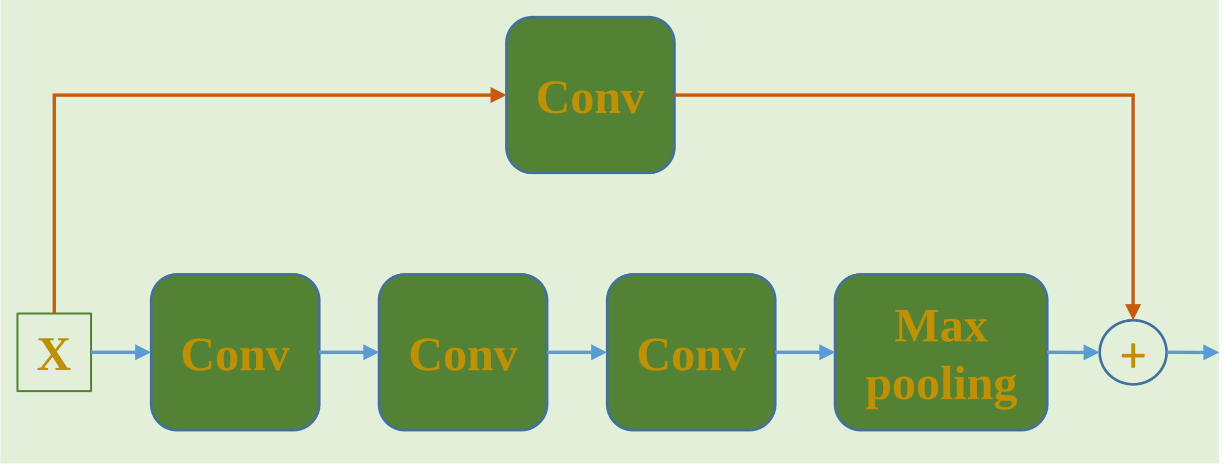


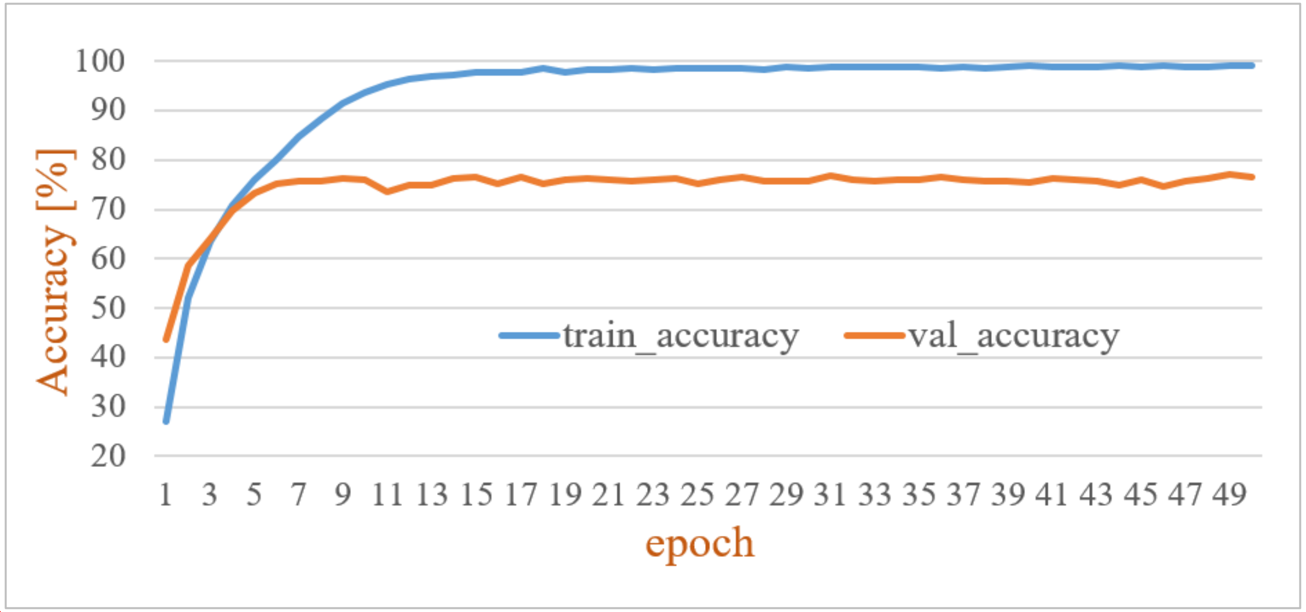
Network Training

❖ **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
```

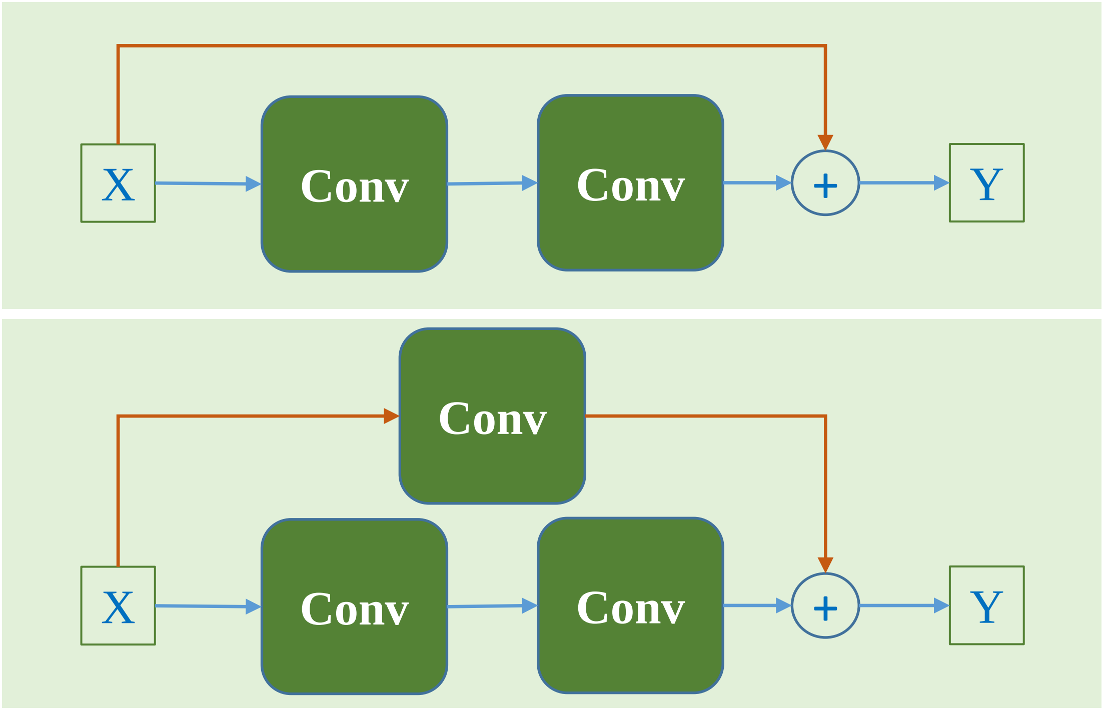


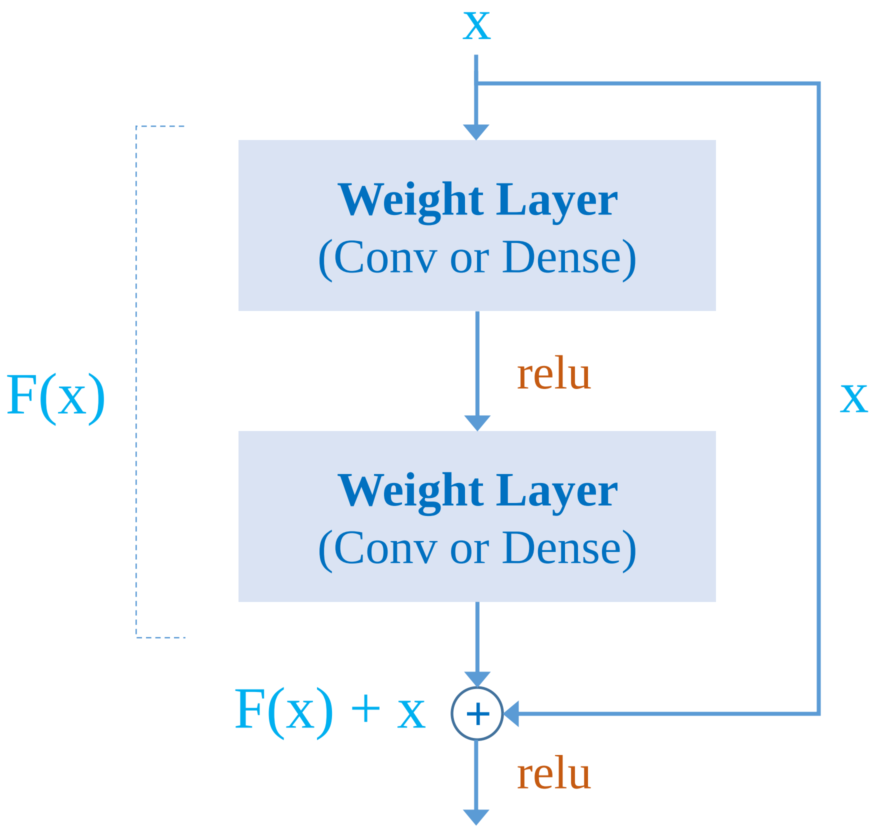


There are several variants that use fully skip connection, concatenation, long skip connection

Network Training

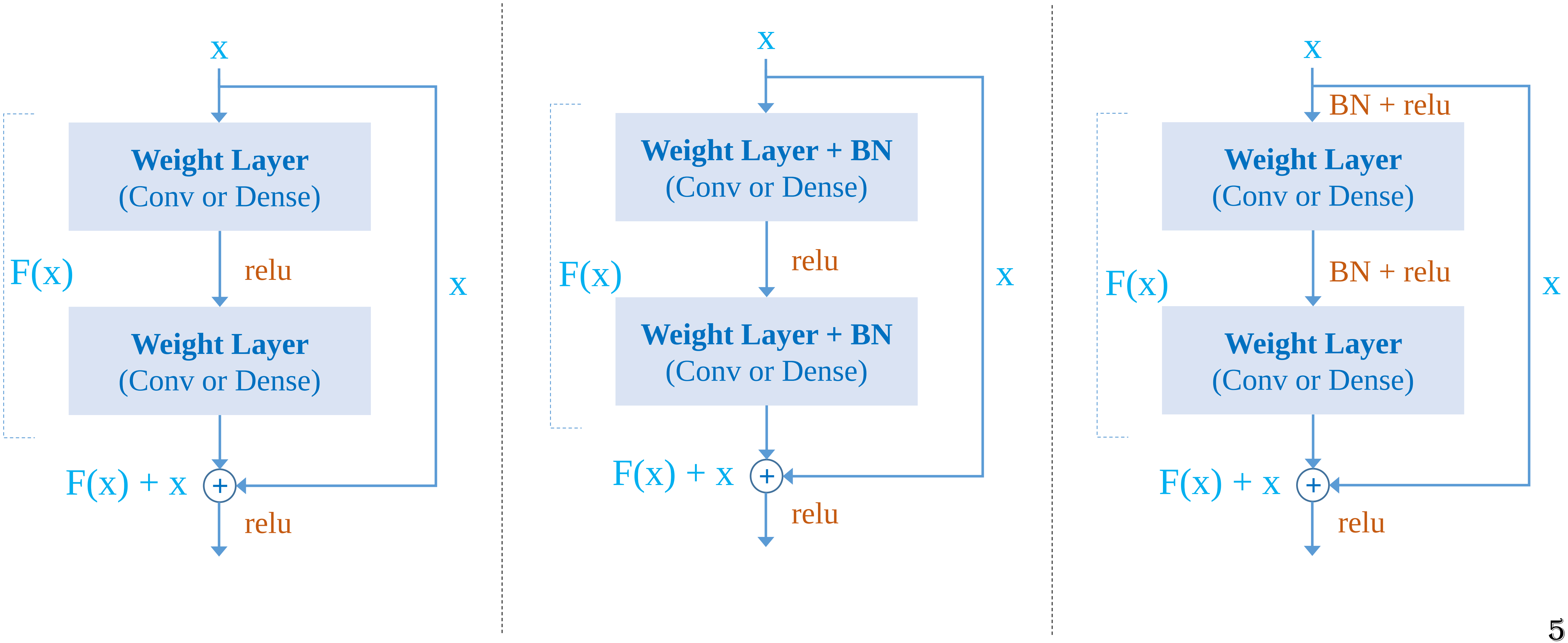
❖ Solution 5: Skip connection





Network Training

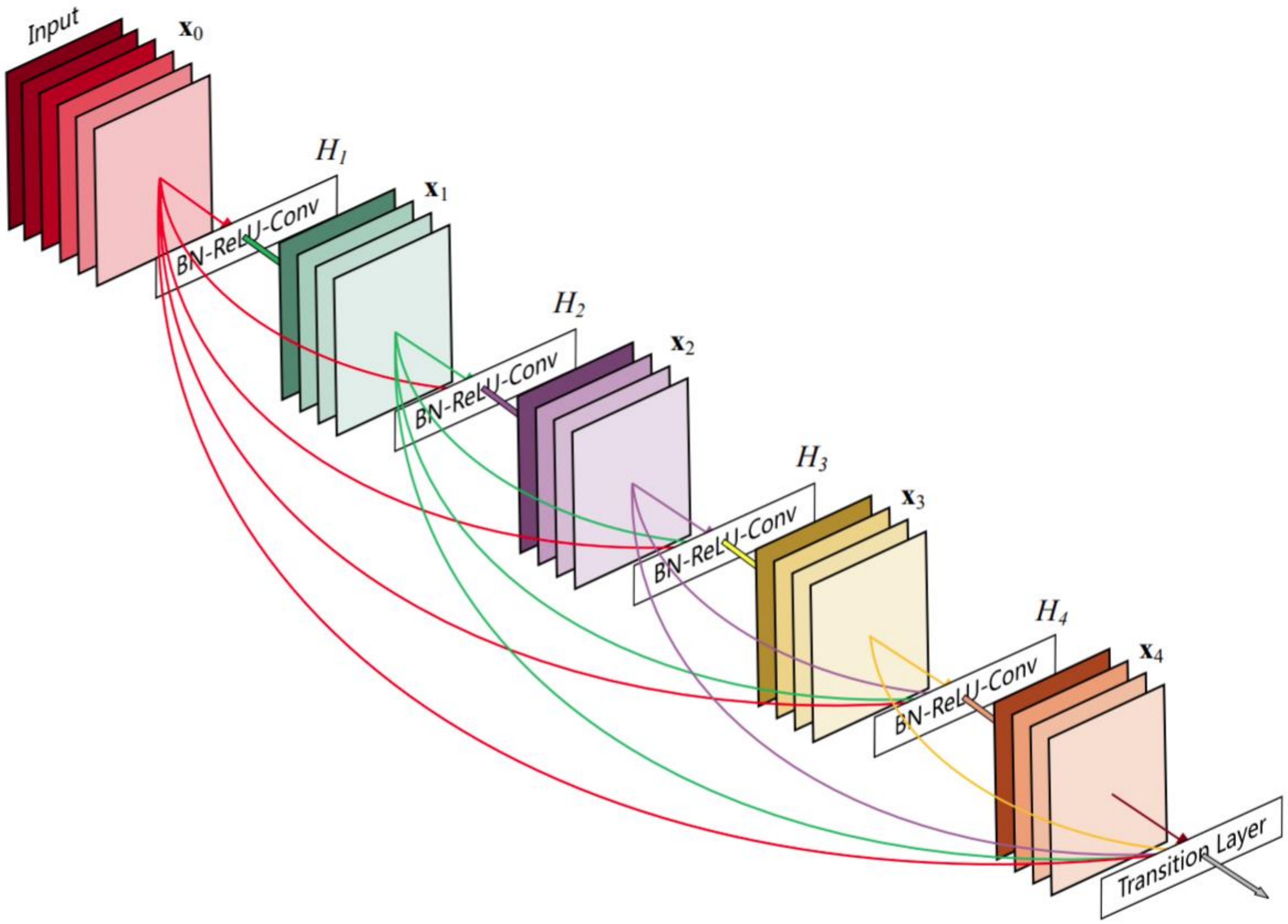
❖ Solution 5: Skip connection

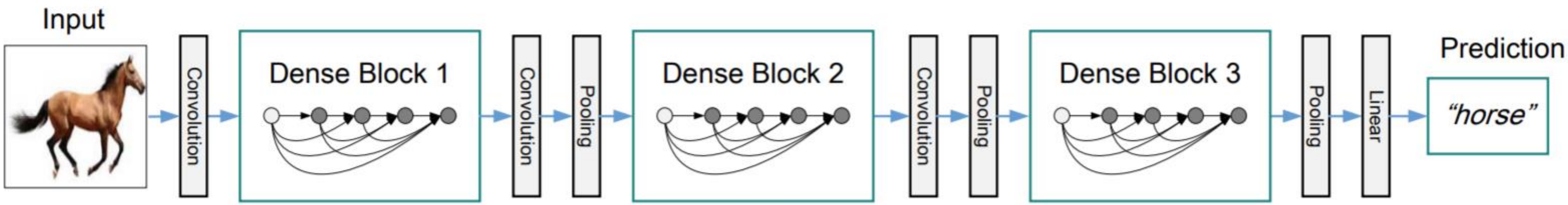


Network Training

❖ **Solution 5: Skip connection**

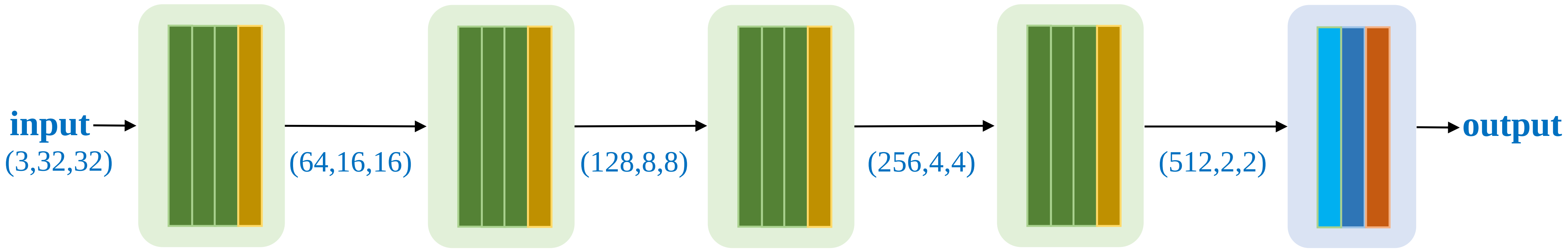
https://arxiv.org/pdf/1608.06993v5.pdf





Network Training

❖ Solution 6: Reduce learning rate



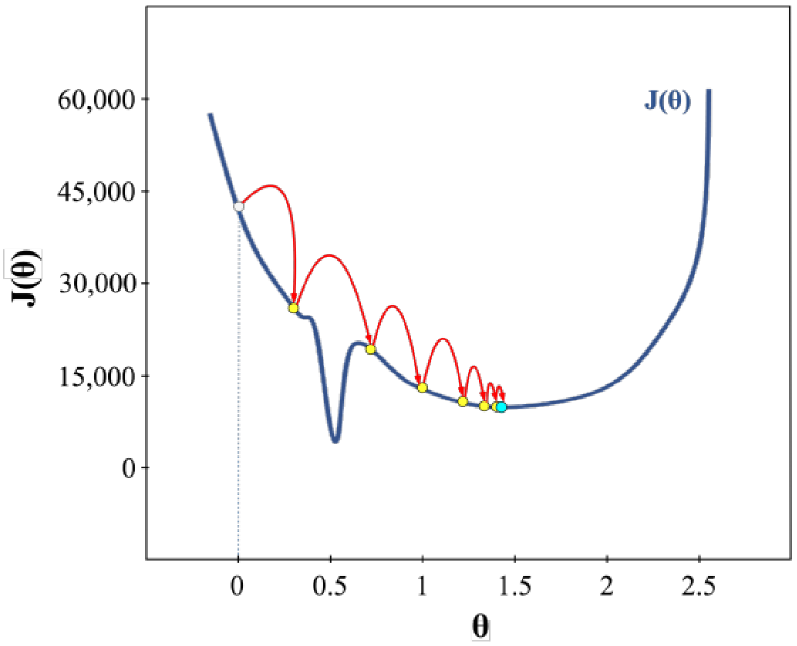
(3x3) Convolution
padding='same'
stride=1 + ReLU

Flatten

Dense Layer-512
+ ReLU

(2x2) max pooling

Dense Layer-10
+ Softmax

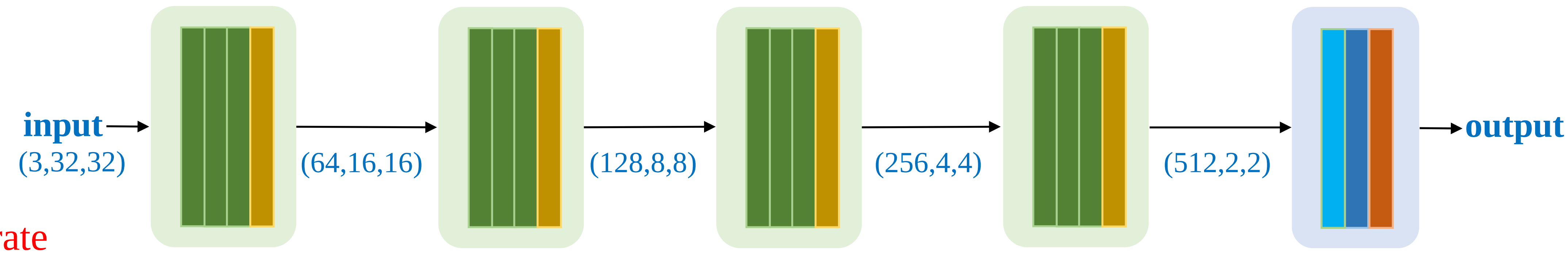


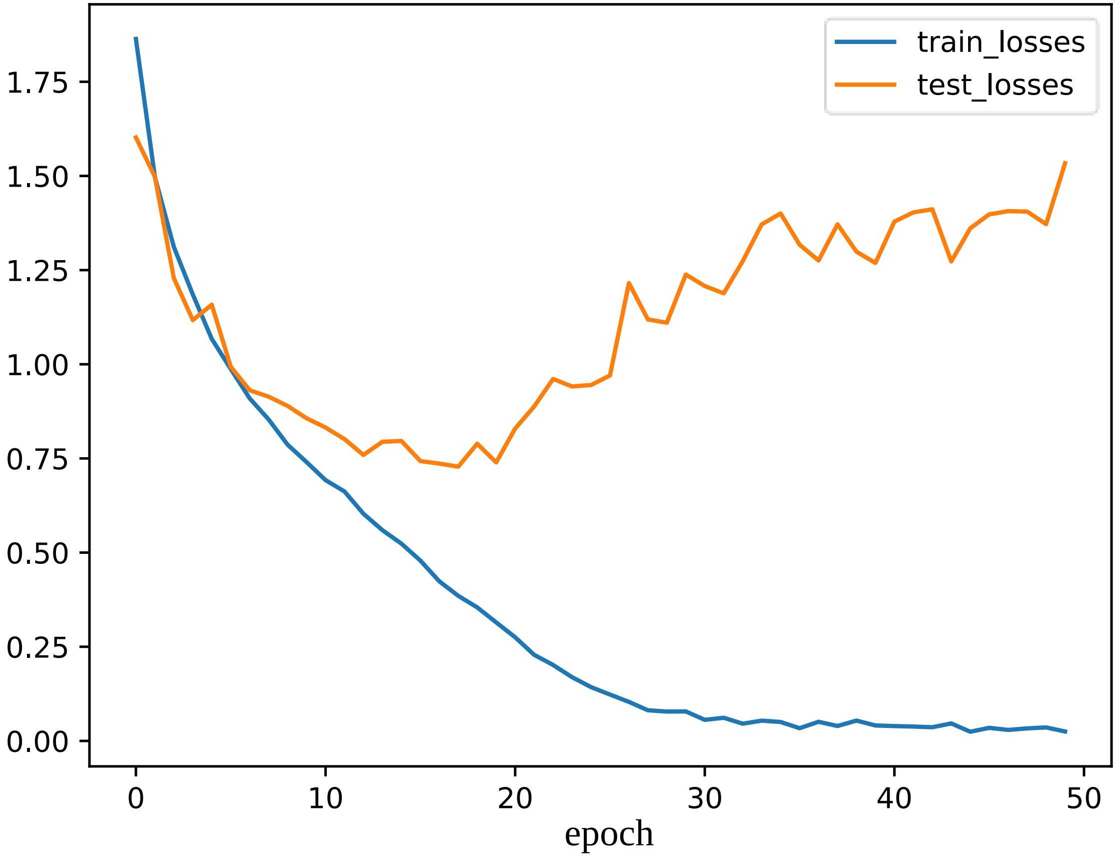
From "Machine Learning Simplified"

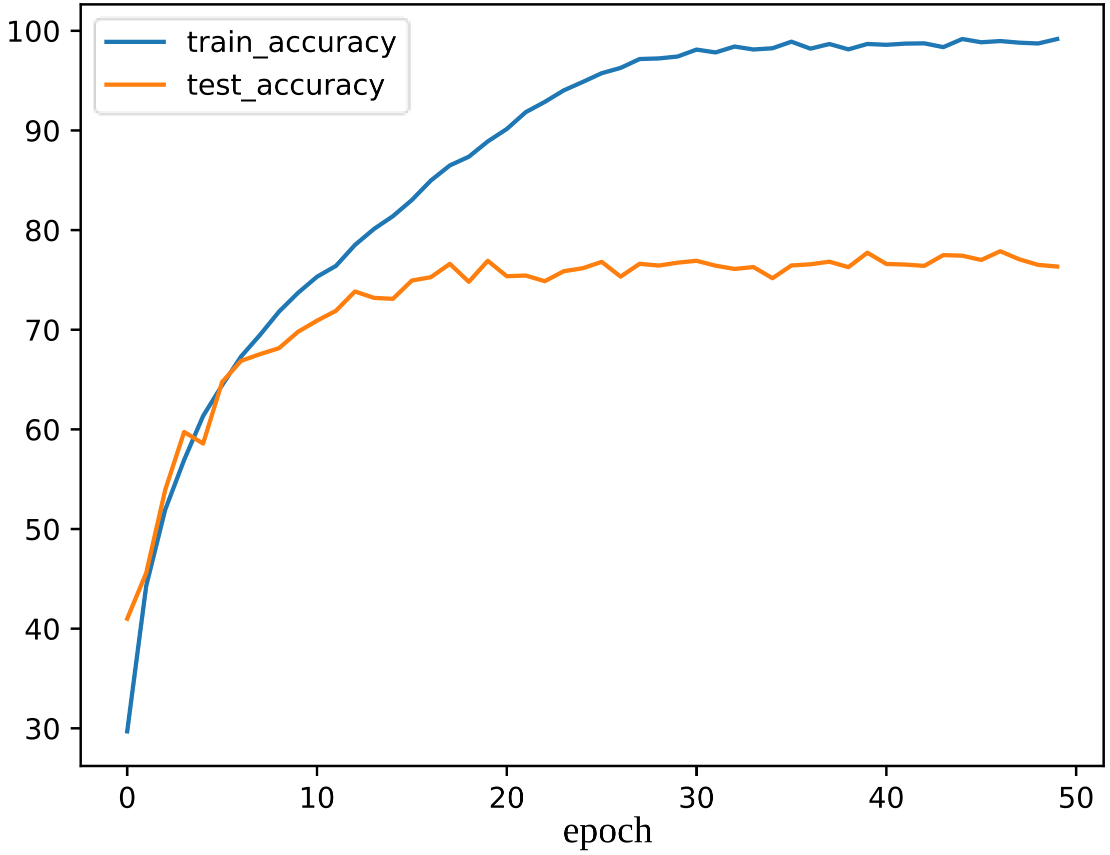
```
# loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-4)
```

Network Training

Reduce learning rate







Further Reading

Skip connection

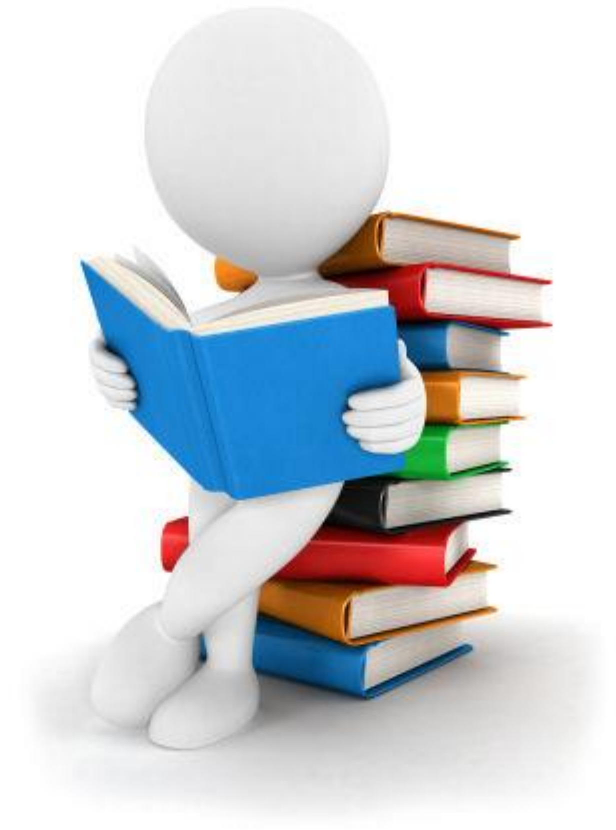
https://theaisummer.com/skip-connections/

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